Hybrid Data Logging Scheme for Hardware Transactional Memory

L. Emurian[a], C. Ferri[b], R. I. Bahar[b], T. Moreshet[c], M. Herlihy[b]

[a]Hamilton College [b]Brown University [c]Swarthmore College

Introduction

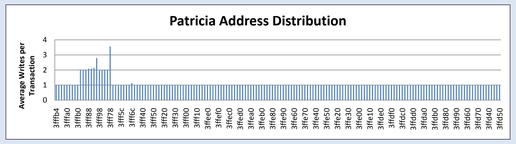
Transactional memory is a speculative scheme used for managing memory contention in multiprocessing systems. In hardware transactional memory, we can use a transactional cache to store all original copies of data modified during a transaction and the modified data. In case of data contention among processors, we must abort, or revert each thread back to its original state using the original data we stored in the transactional cache. Unfortunately, the transactional cache consumes approximately 30% of the total energy used during execution, partially due to storing two copies of data. The transactional cache also frequently overflows, causing serialization and slower execution.

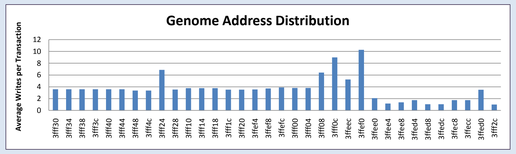
We are proposing the use of a transactional logging scheme to move private data out of the transactional cache. In the transactional logging scheme, we only need to save the original copy of private data in case of contention. By reducing the amount of data stored on the transactional cache, we aim to reduce overflows and energy consumed. The transactional logging scheme is more efficient than the transactional cache because it is a simple stack. We can improve the efficiency of the log by keeping track of the types of accesses we make to private data and only storing the addresses and data that we need. To do so, we use a filter cache. The filter cache understands accesses to memory, so it can estimate if we will need to restore the data. Preliminary results show that the logging scheme is promising and may reduce power while retaining performance.

Logging Scheme

- The log is stored on the scratchpad memory.
- We only store private data and addresses on the log – shared data remains in the transactional cache.
- Each time we are in a transaction and we store (write) a private address, we save the private data to the log at a given index and increase the index.
- On a commit, reset the log index to 0, quickly discarding all saved data.
- On an abort, write the contents of the log back to memory, starting at the index, or the top of the log.
- Address distributions show that logging can be more efficient by only storing each address once

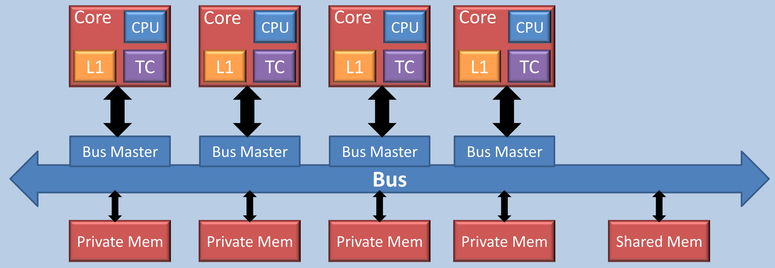
Fig. Basic Transactional Log





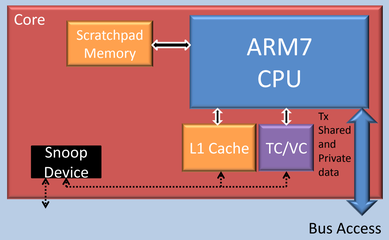
Architecture Overview

- Diagram shows a 4 core architecture
- Each core has an ARM7 CPU
- Each core has a 4KB 1-way L1 instruction cache and a 1 KB 1-way data cache
- The transactional cache (TC) is 512 B



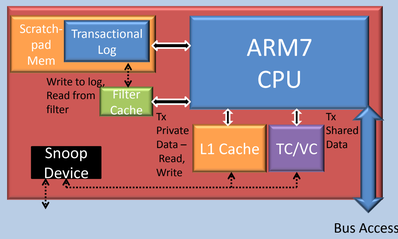
Core Architecture without Logging Scheme

- All private data is read and written to and from the L1 cache and the transactional cache or victim cache.
- The scratchpad memory is only used to store registers saves registers before a transaction



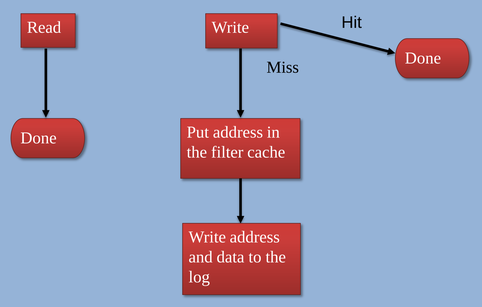
Core Architecture with Logging Scheme and Filter Cache

- Transactional log is located on the scratchpad memory.
- The Filter cache is a direct-mapped cache because we need fast lookup time
- Read and write signals for private data go to the scratchpad memory and the filter cache to determine data to be written to the log
- Filter cache is a direct-mapped cache due to need for fast look up
- Only read and write signals for shared data are on the transactional cache



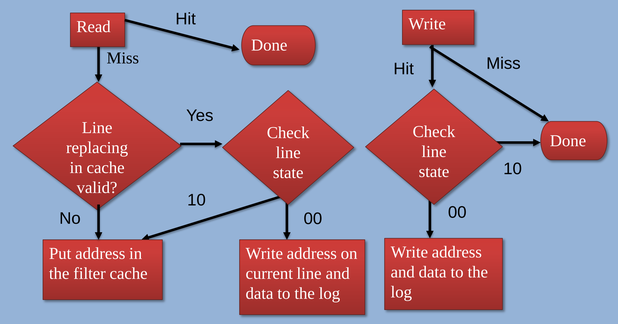
Filter Cache

Static



- Only stores addresses written to the log.
- We only need to store an address on the log once, since we only need to restore the original copy of data
- We check the filter cache before we write any data to the log and do one of two things:
  - The address is in the filter cache: Don't write the address to the log
  - The address is not in the filter cache: Store the address in the filter cache and write the address to the log

Dynamic



- Only stores addresses that are read from memory.
- If we only write data to an address during a transaction, we do not need to save that address and data because the data is not used by other data during a transaction. We only need to store addresses that have been read and are then written later on in the transaction.
- In the dynamic filter cache scheme, we use two bits to identify the data stored in a cache line.
  - 01 indicates that a line is invalid.
  - 00 indicates that the line has been read, but not written.
  - 10 indicates that the line has been written to the log.

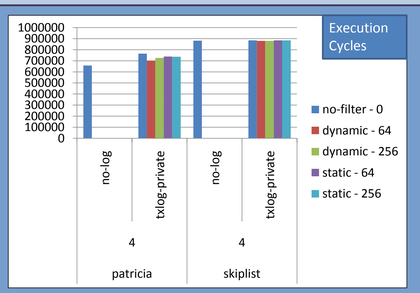
Results

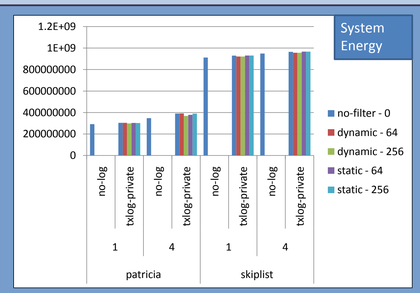
Benchmarks

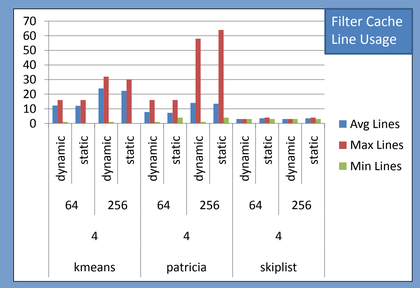
- Patricia (MiBench): Patricia Trie structure used to match IP address prefixes
- K-means (STAMP): Partition-based program commonly used in image filtering
- Skiplist (STAMP): Uses a skip-list data structure, commonly used in memory management applications
- Genome (STAMP): Gene sequencing program
- Vacation(STAMP): Travel reservation system that is non-distributed

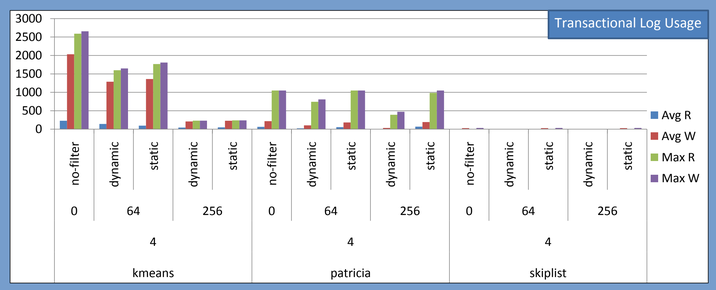
Analysis

- Preliminary results show a decrease in overflows, and a slight decrease in execution cycles
- With more extensive simulation with a broader range of benchmarks, we expect to see better results. In particular, genome and vacation may benefit more from this logging scheme since they contain longer transactions, and therefore store more data in the transactional cache. This should lead to more opportunities to move data to the log.
- Kmeans may show strange results - it has short transactions and spends little execution time in transaction.
- Skiplist does not abort and also rarely uses the log.









References

Cesare Ferri, SamanthaWood, Tali Moreshet, R. Iris Bahar, and Maurice Herlihy. Embedded-tm: Energy and complexity-effective hardware transactional memory for embedded multicore systems. *Journal of Parallel and Distributed Computing*, In Press, 2010.