In the rest of this document, we summarize the following papers on thread-level data speculation in multiprocessors:


Machines which can simultaneously execute multiple parallel threads are becoming increasingly commonplace on a wide variety of scales. Perhaps the most stumbling block to exploiting all of this raw performance potential is our ability to automatically covert single-threaded programs into parallel programs. Currently, the burden is typically on the programmer to identify independent regions that can be safely executed in parallel and synchronize explicitly. Complicated control flow and memory access patterns are the challenges to efficiently parallelizing programs. The idea is to find the “oblivious” parallelization by dividing the program up arbitrarily, executing sub-regions in parallel and speculating on the contents of memory. But hazards like RAW, which implies “earlier” regions write memory read by “later” regions, make this difficult. Thread-level speculation is one of the solutions that enable parallel execution of sequential applications on a multiprocessor by violation detection, violation handling and memory consistency ensuring.

The first paper (Stanford) describes a complete, detailed implementation of the support for thread-level speculation on the Hydra chip multiprocessor. The violation is detected and handled by write-through coherence, in which the writes are globally visible through a “write bus” and invalidation (violation detection) is made on each write. The consistency is ensured by write buffer, which records speculatively-modified cache lines. Buffers drain to L2 cache when safe, and drain a byte at a time when L2 cache is free.

The second paper (CMU) propose and evaluate a design for supporting
thread-level speculation that seamlessly scales to any machine size because it is a straightforward extension of writeback invalidation-based cache coherence, which itself scales both up and down. This scheme depends on ensuring that epochs commit their speculative modification to memory in logical order. Each epoch is waiting for and passing the homefree token at the end of the epoch. When the token arrives, we know that all logically-earlier epochs have completely performed all speculative memory operations, and that any pending coherence messages have been processed. Hence memory is consistent.

Ownership required buffer is introduced to counteract the delay in passing the homefree token. Whenever a line becomes both speculatively modified and shared, the address will be added to the ORB. The buffer drains as soon as speculation ends.

However, this increased speculative parallelism also increases the likelihood of hazards and decreases locality due to frequent invalidation and distributed data across multiple threads. Software overhead in thread creation and finalization increase memory traffic. And failed threads increase superfluous memory access. In loop speculation, the variables that are shared across loop iterations cannot be register allocated, because the data speculation mechanism cannot protect against true dependency violations in registers, which may also increase the amount of memory traffic.

Serious consideration must be given to the size of the threads selected. Long threads increase the likelihood of hazards and write more to the buffer, which is potentially for costly overflows. And in the case of short threads, overhead becomes prohibitive.

Whether to speculate or not? There are two heuristics that are used by the first paper to find and prevent speculation on nonparallel threads: violation counters, to eliminate threads with many dependencies, thread timers, to eliminate threads that are too short or long.