Hardware Transactional Memory

Sean Lie†, Krste Asanovic†, Bradley C. Kuszmaul‡, Charles E. Leiserson‡

†MIT CSAIL Computer Architecture Group
‡MIT CSAIL Supercomputing Technologies Group

***Abstract*— This work shows how hardware transactional memory (HTM) can be implemented to support transactions of arbitrarily large size, while ensuring that small transactions run efficiently. Our implementation handles small transactions similar to Herlihy and Moss's scheme in that it holds tentative updates in a cache. Unlike their scheme, which uses a special fully associative cache, ours augments the ordinary processor cache and provides a mechanism to handle cache spills of uncommitted transactional data. Consequently, our scheme runs faster for small transactions while correctly handling transactions of arbitrarily large size.**

Although transactions are small in the common case, we argue that HTM should not restrict the size of transactions, because it complicates the programmer/compiler model and precludes some important programs from exploiting transactional memory. We show that the Linux 2.4.19 kernel can be automatically and efficiently "transactified" if boundless transactions can be supported. Our experimental results show that the largest transaction touches over 7000 64-byte cache lines, whereas 99.94\% of the transactions touch fewer than 64 cache lines. We further show that synchronized methods in Java can be easily compiled to our HTM scheme, thereby providing the advantages of nonblocking atomicity (including absence of deadlock) in a straightforward fashion.

Our HTM scheme for boundless transactions uses an efficiently implementable hardware snapshot and the ordinary set-associative L2 cache extended with less than two bits per cache line. One of the bits tells whether the cached item is part of a transaction (as in the Herlihy-Moss scheme), and all the lines in an associative set share another bit telling whether a line has overflowed from the cache and is now stored in a special overflow area of main memory. We provide empirical results to show that our scheme does not adversely affect the processor pipeline or hinder speculative execution.

[Full Text Not Available]