

Comparison of Programming and Synchronization Techniques

Current Work

I am currently working on the design and implementation of a simple transactional memory system (see Transactional Memory Overview below). The implementation will be in a software simulator of a distributed shared memory machine (based on the SGI Origin 3000). The memory system should be completely integrated into the simulator in the next few weeks.

6.338 Final Project Proposal

I propose to compare the various programming and synchronization techniques currently available on distributed shared memory machines. I am currently thinking of MPI and OpenMP however I may look at some others if time permits. In addition, I will compare conventional locking (in both MPI and OpenMP) with the transactional memory synchronization technique mentioned above. I will evaluate factors such as programming ease and performance from the user/programmer point of view.

Methodology

I will choose one algorithm for the comparison. The algorithm must require locking (in the conventional sense) and be well suited to shared memory architectures. In addition, the algorithm should be representative of a “real world” problem so that the comparison has some practical value. I will implement the algorithm in both MPI and OpenMP using locking. Then, I will change the implementations so that they use transactional memory instead of locks. All testing and performance benchmarking will be performed using the software simulator or an actual machine (I have access to a 32 processor SGI Origin 2000) when possible.

After some preliminary research, I believe that a graph algorithm such as maximum flow will be well suited for this project. However, if time permits, I will explore other algorithms as well.

Transactional Memory Overview

Synchronization between processors running on a parallel machine is one of the most difficult aspects of parallel programming. When data is shared between processors, the use of locks is often required for correct program execution. However, locking is not a very natural concept for programmers and thus can often lead to undesirable situations such as deadlock if used incorrectly. Therefore, programs often use very conservative locking in the interest of correctness or ease of implementation. In addition, the use of locks requires very high overhead in terms of machine resources as well as programming ease.

Ideally, the programmer should have the ability to perform atomic actions on shared data without having to worry about the potential problems associated with locks. Hardware support for transactional memory achieves this goal. With transactional memory, the software running on an individual processor is ensured, by the hardware, that a set of memory accesses is either performed atomically or not at all. This notion is very similar to that of a transaction in database terms; hence the name. Transactional memory can replace locks altogether resulting in a more intuitive programming environment as well as a significant increase in performance.