ECE 66600 Final Project Proposal

Vaibhav Ramachandran
Nikkitha Subbiah

Objective:

This project will implement and compare the performance of two self-invalidation schemes, namely Dynamic Self-Invalidation and Last Touch Prediction against each other as well as against a baseline model, the MESI Two-Level Cache model in gem5. These two schemes aim to implement different forms of predictive invalidation to help alleviate remote memory accesses in a distributed shared memory (DSM) system. Although their overarching ideas are similar, their implementations are quite different and hence will act as perfect schemes for performance comparison.

Motivation:

Read-Write sharing in shared memory protocols lead to large number of invalidations which incur a latency of few hundred cycles. Apart from long latencies, they also add to the communication overhead and lead to decrease in performance. The idea of Self-Invalidation protocols is to predict upcoming invalidations and commit them beforehand in order to eliminate these overheads. To this end, several schemes have been proposed, a lot of which involve altering communication patterns at the network level. The above mentioned schemes, Dynamic Self-Invalidation and Last-Touch Prediction, offer a much cleaner approach to speed up performance by proposing changes to the coherence protocols of the processors cache. The implementation simplicity and performance benefits provided by these ideas is what makes this topic intriguing.

Description:

The two schemes work as follows:

1. The Dynamic self-invalidation scheme eliminates invalidation messages by having a processor automatically invalidate it's local copy of cache block before a conflicting access by another processor. The type of DSI scheme being implemented in this project will make use of version numbers in the directory to keep track of when the blocks are modified by different processors to decide whether to obtain a self-invalidate block.
2. The Last Touch Prediction learns and predicts the 'last touch' to a memory block by keeping track of the sequence of instructions to determine the point at which a block is invalidated due to a coherence miss. Once a prediction is made the block is invalidated by the cache controller before the next access and the block is moved to another processor. This works because programs are repetitive which makes it possible to associate self-invalidation with program instructions.

Evaluation Methodology:

Evaluation of the two schemes will be done using the Ocean and Barnes benchmarks from the SPLASH-2 benchmark suite, Tomcatv from SPEC and EM3D from Split-C since these were cited in both the papers. They will provide a fair comparison and aid in directly comparing our results to those obtained in the papers.

The global last-touch signature table for the shared blocks in the Last-Touch Prediction model and "versioning" for the shared blocks in the Dynamic Self-Invalidation model will be implemented in this project. The reason for implementing the global signature table in place of a per-block structure is the reduced storage overhead. As previously mentioned, the baseline model is the MESI Two-Level Cache model in gem5 since it most closely resembles the architectures described in both papers.

References:

1. A. C. Lai and B. Falsafi. Selective, accurate and timely self invalidation using last-touch prediction. In *Proceedings of the 27th International Symposium on Computer Architecture (ISCA),* 2000.
2. A. R. Lebeck and D. A. Wood. Dynamic self-invalidation: Reducing coherence overhead in shared-memory multiprocessors. In *Proceedings of the 22nd Annual International Symposium on Computer Architecture (ISCA),* 1996.
3. A. Kägi, N. Aboulenein, D. C. Burger, and J. R. Goodman. Techniques for reducing overheads of shared-memory multiprocessing. In *Proceedings of the 1995 International Conference on Supercomputing, pages 11–20,* July 1995
4. S. S. Mukherjee and M. D. Hill. Using prediction to accelerate coherence protocols. In *Proceedings of the 25th Annual International Symposium on Computer Architecture,* June 1998