Hardware Predictor Implementation and Evaluation for Distributed Shared Memory Systems in Gem5

Nicole Gathman
Purdue University
ngathman@purdue.edu

Ganesha V Sivagurunathan
Purdue University
gvedaraj@purdue.edu

I. Project Introduction

This project aims to implement and analyze two hardware predictors for multiprocessor distributed shared memory (DSM) systems. The first hardware predictor is Memory Sharing Predictor (MSP) introduced by Lai and Falsafi. MSP is a pattern-based predictor that realizes high accuracy for predicting remote memory accesses [1]. Predicting remote memory accesses with high accuracy allows the system to hide remote memory access latencies, which resulted in a 12% execution time reduction. The second hardware predictor we will replicate and evaluate is Lai and Falsafi's Last Touch Predictor [2]. Last Touch Predictor (LTP) predicts when a processor last accesses a cache block before the block is invalidated by a coherence request. Predicting a last touch allows self-invalidation of the specific block in advance of the future invalidation. Doing so hides the invalidation time and can lead to approximately 11% average speedup in execution time [2].

MSP was one of the first designs with speculative requests for DSM to hide remote access latencies. The paper showed how important it is to have a predictor with high accuracy to minimize the occurrence of wrong predictions, which may add latency to future requests. Therefore, we want to replicate the design to understand more behind one of the instrumental schemes in DSM design.

Like all predictors, LTP was developed to reduce the remote access latency, particularly the ones due to invalidations. LTP attempts to be better than Dynamic Self-Invalidation mechanism by providing timely and selective invalidations by achieving a higher prediction accuracy [4]. This predictor's mechanism is intriguing and challenging; therefore, we want to replicate it to understand its true impact on DSM design.

MSP focuses on speeding up remote memory accesses through predicting future memory accesses and executing the predicted request. Whereas LTP aims to self-invalidate a block before the block is invalidated by a coherence request. At a high level, both schemes optimize different aspects of a DSM system. However, both schemes can speed up remote memory access latency through their respective methodologies. Thus, there is a meaningful comparison to be made between MSP and LSP in the context of remote memory accesses and overall program execution time. For other cases, the schemes are not comparable and will be analyzed individually.

To limit the scope of the project, we have selected certain aspects of each scheme to replicate. For MSP, there are three necessary implementation components to fully replicate the original research and see similar speed up measurements. First is the actual memory sharing predictor, which decides what the speculative request(s) is. Second is the hardware that triggers the execution of the speculative requests, called Speculative Write Invalidation (SWI). Third is the hardware that executes the speculative request when triggered [1].

To effectively evaluate LTP, we need to implement 2 components. First, is the LTP predictor scheme. The paper discusses two different organizations of the last touch table: PA-p (per-block) and PA-g (global) [2]. In this project, we will be implementing the PA-p organization as it gives a higher prediction accuracy. The second component is a mechanism to implement speculative self-invalidation. Self-invalidations don't require any change to the coherence protocol, but a mechanism to verify the predictions must be implemented. If any prediction appears to be of low accuracy, a self-invalidation for that block is prevented.

Team Member Name	**Scheme Responsibility**
Nicole Gathman	Memory Sharing Predictor
Ganesha V Sivagurunathan	Last Touch Predictor

Table 1: Team member scheme responsibility

II. Methodology

Evaluation methodologies for MSP and LTP are similar, but use different parameter values, which are described below in Sections A and B. Section C describes the methods to compare MSP and LTP using execution speedup under comparable conditions as described in the section.

Note, both original publications pre-dated Gem5 system simulator. Therefore, some of the parameters mentioned in the papers do not directly translate to controllable parameters in Gem5. However, there are parameters in Gem5 that are similar to the parameters in the paper, which are listed in Table 2 below.

A. MSP Evaluation Methodology

To match the parameters used in the original paper, we will use the following parameters in our design.

Number of Nodes	16
Processor Cache	1 MB
Local Memory Access Time	104 cycles

Table 2: System configuration parameters for MSP evaluation

Additionally, we will record the overall benchmark speedup to compare our results and paper results. Execution time or number of cycles measured by Gem5 will be used to determine system speedup for a given benchmark. Speedup also gives us a universal metric to compare MSP and LTP optimizations for DSM.

The original paper measures predictor accuracy and quantity of speculations and mis-speculations. To determine the correctness of our design and validate our performance speedup, we will run SPLASH benchmarks and measure the number of speculations and mis-speculations along with predictor accuracy by counting the number of correct predictions and mis-predictions. We will run the benchmarks on our MSP system and the unaltered Gem5 multiprocessor system. Two SPLASH benchmarks, Barnes and Ocean, were used in the paper and are available for our evaluation and design verification. In addition to Barnes and Ocean, we will also use the SPLASH benchmarks listed in Table 3 to evaluate our design considering the read/write sharing composition (#read sharing instructions / total instructions) and use of patterns in the benchmark [3]. We chose the benchmarks below because of the diversity in read/write sharing. Cache block sharing with MSP can increase or decrease performance resulting in different speedups, which will help evaluate the overall design.

Benchmark	**%Shared Reads**	**%Shared Writes**
Barnes	11%	5%
Ocean	20%	4%
LU	19%	9%
Raytrace	19%	3%
Radiosity	9%	1%
FMM	17%	2%

Table 3: Sharing composition of select SPLASH benchmarks

B. LTP Evaluation Methodology

To match the parameters used in the original LTP paper, we will use the following system parameters in the LTP design evaluation.

Number of Nodes	32
Processor Cache	1 MB
Local Memory Access Time	104 cycles

Table 4: System configuration parameters for LTP evaluation

The evaluation will be performed using the Ocean, Raytrace and Barnes applications from SPLASH, tomcatv from SPEC and Em3d from SPLIT-C (subject to access), as these are some of the benchmarks the paper evaluates [2]. In addition to these, the benchmarks listed in Table 3 will also be used to evaluate LTP design and verify design correctness.

The original paper published the prediction accuracy by determining the fraction of invalidations correctly predicted, not predicted and mis-predicted. We intend to do the same in our evaluation by incrementing a counter in the design for every correct prediction and calculating accuracy. In addition, the overall speedup for these benchmarks will also be evaluated and reported.
The original paper evaluates the sensitivity of the predictor to the signature size (signature is the encoded trace information). We will not be performing this evaluation. We will be using a 13-bit signature, the minimum that the paper recommends, to maintain high prediction accuracy.

C. MSP and LTP Comparison Methodology

To compare the two schemes, both designs will undergo evaluation on a 16-node system with parameters identical to those outlined in Table 2.

We will assess MSP and LTP based on their invalidation prediction accuracies by executing the benchmarks detailed in Table 3. It's important to note that a fair comparison of execution times (overall speedup) is contingent upon MSP solely implementing speculative self-invalidation, as opposed to speculative data accesses which LTP cannot support. Thus, our comparison will exclusively involve Self-Invalidation with MSP versus Self-Invalidation with LTP.

REFERENCES

(1) An-Chow Lai and B. Falsafi, "Memory sharing predictor: the key to a speculative coherent DSM," *Proceedings of the 26th International Symposium on Computer Architecture (Cat. No.99CB36367)*, Atlanta, GA, USA, 1999, pp. 172-183, doi: 10.1109/ISCA.1999.765949.

(2) An-Chow Lai and B. Falsafi, "Selective, accurate, and timely self-invalidation using last-touch prediction," *Proceedings of 27th International Symposium on Computer Architecture (IEEE Cat. No.RS00201)*, Vancouver, BC, Canada, 2000, pp. 139-148, doi: 10.1109/ISCA.2000.854385.

(3) The splash-2 programs. (n.d.-b). https://pages.cs.wisc.edu/~markhill/restricted/isca95_splash2.pdf

(4) Alvin R. Lebeck and David A. Wood. 1995. Dynamic self-invalidation: reducing coherence overhead in shared-memory multiprocessors. In Proceedings of the 22nd annual international symposium on Computer architecture (ISCA '95). Association for Computing Machinery, New York, NY, USA, 48–59. https://doi.org/10.1145/223982.223995

(5) F. Dahlgren, M. Dubois and P. Stenstrom, "Sequential hardware prefetching in shared-memory multiprocessors," in *IEEE Transactions on Parallel and Distributed Systems*, vol. 6, no. 7, pp. 733-746, July 1995, doi: 10.1109/71.39540