Memory Dependence Prediction Techniques

Elton Rodrigues, Jeneel Kachhadiya, Victor Abad

1 Topic Description

The goal of the project is to evaluate the accuracy of different memory dependence predictors in a x86 out of order architecture. Memory dependence prediction allows us to improve the latency of memory accesses by speculating memory instructions. This has motivated the development of several memory prediction techniques, which can vary in effectiveness according to the target architecture and workload. We plan to compare a naive implementation that is no dependence prediction vs Store-set prediction vs Store-load pair prediction and analyse the performance metrics like IPC and latency between them.

2 Motivation

As talked about in the lectures, the CPU performance is increasing at a much faster rate than the memory performance, which is creating a huge bottleneck with the gap widening each year. Recent super-scalar processors allow scheduling of processors out of order for exploiting instruction level parallelism, but with this data dependencies also creep in. Register dependencies are determined in the decode stage, but the memory dependencies need to wait until the address is evaluated, which creates a problem in an out-of-order processor. If the scheduler executes a load instruction before a preceding store instruction has finished writing the data to memory, the load will have an incorrect value leading to this load and proceeding instructions to be flushed and re-executed, incurring a heavy penalty. To avoid this and not implement a load before a store, we incorrectly classify independent loads, unnecessarily waiting for the execution. To address this, a need for memory dependence prediction was created.

3 Implementation

We plan to simulate workloads on Gem5 using O3CPU (out of order implementation of the x86 architecture and loosely based on the Alpha 21264) to evaluate three memory dependence prediction approaches [1]. The first is a naive implementation, this model allows loads to execute out of order without considering the possible dependencies. The second implementation is store-sets prediction, which is based on the concept of the store sets of a memory load instruction. Initially, the store set of a load instruction is empty and out of order memory operations are allowed. Whenever a dependence violation is detected, the write instruction that caused that violation is added to the store set of the load instruction. If the store set of a load instruction is not empty, the load instruction is allowed to proceed only after all the stores of the store set are executed [2]. The third implementation is a store-load pair prediction, which has the goal of predicting whether a dynamic iteration of a load-store pair will result in a dependence violation based on a set condition. This method uses two fully associative tables to predict dependencies in the load-store pairs. The memory dependence prediction table (MDPT), is used to identify

synchronization pairs. A new entry is allocated in this table when a violation occurs and when a MDPT matching entry is found this instruction must be synchronized and an entry in a second table memory dependence synchronization table (MDST) is allocated; otherwise should be immediately executed [3].

4 Timeline

Phase 1 : Study and analyze the current load store dependency implementation in the out of order O3 CPU in Gem5.
Phase 2 : Make changes to the memory dependency module identified in the first phase to modify it as an naive implementation so which does not make any predictions and allows the load instructions to commit violations.
Phase 3: Implement a store store-load pair predictor.
Phase 4: Evaluate the performance of the above-mentioned memory dependency predictors by simulating different benchmark programs like gcc, bzip2 and perlbench. We aim to compare three metrics : Latency, IPC and number of memory references.
Fallback Plan : For some reason, we are unable to correctly simulate the full system with the memory dependence predictors, we plan to simulate only the predictor mechanism without any interaction with the CPU load scheduler. The accuracy of the different predictors can be analyzed and compared.

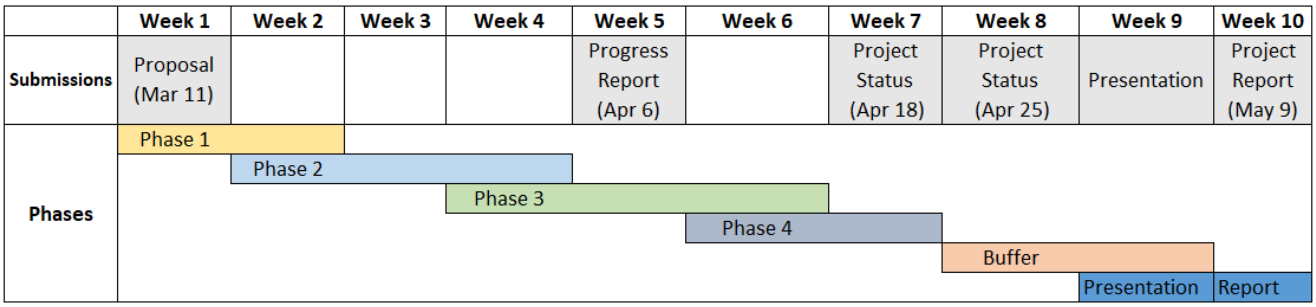

Figure 1: Project Timeline

.

References

[1] Nathan Binkert, Bradford Beckmann, Gabriel Black, Steven K. Reinhardt, Ali Saidi, Arkaprava Basu, Joel Hestness, Derek R. Hower, Tushar Krishna, Somayeh Sardashti, Rathijit Sen, Korey Sewell, Muhammad Shoaib, Nilay Vaish, Mark D. Hill, and David A. Wood. The gem5 simulator. *SIGARCH Comput. Archit. News*, 39(2):1–7, aug 2011.

[2] G.Z. Chrysos and J.S. Emer. Memory dependence prediction using store sets. In *Proceedings. 25th Annual International Symposium on Computer Architecture (Cat. No.98CB36235)*, pages 142–153, 1998.

[3] A. Moshovos, S.E. Breach, T.N. Vijaykumar, and G.S. Sohi. Dynamic speculation and synchronization of data dependence. In *Conference Proceedings. The 24th Annual International Symposium on Computer Architecture*, pages 181–193, 1997.