Memory Prefetching

Lecture #8: Monday, 24 April 2000
Lecturer: Shuaib Arshad
Scribe: Ayodele Embry
Reviewer: Mattan Erez

1 Chen and Baer - Effective HW Based Data Prefetching for High Performance Processors

Chen and Baer's hardware based approach seeks to hide memory latency by providing data in the cache just in time for usage. They felt hardware was more advantageous than software since hardware approaches required no additional instructions and operate dynamically at runtime. Further, hardware can directly see the address stream and knows if something is already in the cache.

They describe four different types of access patterns - scalar, zero stride, constant stride, and irregular. They also described two different types of prefetching, temporal which is used with lookahead hardware and spatial which prefetches using the location of an address, but only examined the temporal.

Their hardware implementation uses a Reference Prediction Table (RPT) which contains four entries and records information about the tag, previous address, stride and a 2-bit counter for state.

They used three different techniques for hardware prefetching which differ mostly on the timing of prefetching: 1) Basic stride - the next prefetch is one stride ahead and is performed when the load is encountered. It may occur too late to be effective for something like a small loop. 2) Lookahead - this is similar to basic but triggered by LA-PC, which is several iterations ahead of the PC, improving timeliness. 3) Correlated - deal with nested loops by correlating on the direction of the end-of-loop branch. The correlated predictor used 2 bits, one bit for the inner loop and one bit for the outer loop.

Rather than making any changes to the PC itself, they utilize a second PC that is several instructions ahead. This lookahead PC (LA-PC) uses the PC to reference the RPT. The counter must be able to travel ahead of the PC in order to predict where the code will go next. The LA-PC should be enough iterations (branches) ahead of the PC so as to correspond to a latency that is between L1 miss and L2 hit latencies.

There are several options for prefetching including operating n iterations ahead and using the LA-PC. The LA-PC uses a self-measure of latency. It is unnecessary if good information is given to the compiler. The lookahead limit should be greater than the

maximum number of cycles to access the memory, but not cross too many basic blocks because decreased branch prediction accuracy will negatively impact performance. They also described several software approaches. Software approaches usually can go further ahead than a HW approach because they can take into account other factors such as data coherency, task scheduling, and task migration. The software approach has a slower start because of priming the pump, but can catch up over time assuming the loops are long enough. A hardware n iterations ahead approach could come close to the prefetch distance of the software approach with better branch prediction.

Several memory models are discussed in this paper: 1) Non-overlapped which is a completely sequential memory access. 2) Overlapped where the access is divided into its three stages of execution and each stage can execute in parallel. 3) Pipelined is similar to overlapped except the pipeline stages are determined by the amount of computation that can fit in a single clock cycle (i.e. a new access every cycle).

We discussed that the 30 cycle memory access time that they gave seemed small for main memory since the L2 access time is around 8-12 cycles.

Performance: Matrix had good performance since it is easy to predict dense matrices and the cache is able to hold most of the data. Xlist's relatively good performance was surprising since it contains many linked lists. However, it is probably that the linked list was formed contiguously in memory so spatial predicting was effective. MCPI (miss cycles per instruction) increases for matrix with a block size past 32 due to wasted bandwidth and cache pollution that is bad for performance.

In general, mechanisms with a lot of state don't work well with a multiprocessor environment since they depend on retained state and paging and context switching could have a negative effect. However, training times are short (2 iterations per loop), so much performance may not be lost due to paging.

You get better performance when you try and cover the memory - L2 latency rather than L1 - L2 latency. This is because more can be gained with the longer latency. Other methods such as out-of-order execution are effective for short latencies.

For the RPT size graphs, it seemed that more data points should have been included since the graphs did not flattened out. However, we also noted that the table size would be prohibitive after 1K entries.

The lookahead limit seemed to correspond to a number of cycles slightly above the latency of the main memory. A little extra room is provided to avoid contention.

2 Joseph and Grunwald - Prefetching with Markov Predictors

Joseph and Grunwald present another hardware based approach to prefetching using Markov predictors. One version replaces entries based on probability and the other utilizes a LRU (least recently used) replacement policy.

They discuss the metrics along which prefetching should be evaluated including: a)

Coverage - percentage of loads which are predicted b) Accuracy - was the prefetch guessed correctly or not c) Timeliness - is the information fetched soon enough to be used but not so early that it is discarded before it is used.

In comparison to the Chen and Baer paper, there is a difference in the hardware due to the location of the prefetch mechanism. Chen used the address stream to obtain prefetch information. In this paper, the miss stream is used because it allows for less frequent prediction and its hardware can be located off chip. Since the miss stream is outside of the device it is feeding, it has much less knowledge and can't access hits or the PC.

Joseph and Grunwald assumed that an observed Markov model could approximate the miss reference stream. On a miss, the value is looked up in the table and all 4 addresses recorded for this entry are fetched. Then, on the next miss, the table entry is updated for the previous miss and a Markov model for misses is built. They assume that a miss will usually follow a miss to the same few addresses. However, this method only kicks in after the 1st miss and is only triggered by an L2 miss even after it has been trained. In contrast, the Chen approach uses the PC as a trigger so it can have zero misses after training. Further, always fetching 4 addresses is wasteful.

The Markov Model is actually based on correlation. A second model was presented which uses the same prefetch and prediction, but uses the LRU replacement policy. The LRU model ended up being better than the original Markov model. Basically, this means that LRU rank is better than probability in determining a prefetch. This is probably due to changing program behavior and few L2 misses.

The Hidden Markov Model (HMM) is a different model than the one discussed in this paper. The Hidden Markov Model doesn't know states in advance but it instead discovers states. It may improve accuracy, but not the hit rate. However, provided that enough training data is available, it can possibly give better results.

Although they listed timeliness as an important metric for evaluating prefetching, no support is given in this paper.

3 Mowry - Design and Evaluation of Compiler Algorithm for Prefetching

The Mowry paper was the only software approach discussed. In actuality, even software approaches utilize specialized hardware in that the ISA (instruction set architecture) must include a prefetch instruction and the cache needs to be non-blocking. The prefetch is usually not done in the normal instruction pipeline because in case of an exception, the prefetch should not cause the system to stall.

The main problems with software approaches are increased overhead and stress on memory. To limit the number of prefetches and concentrate only on the useful ones, Mowry looked at a Locality Analysis. Reuse Analysis discovers intrinsic data reuse within a loop nest using a reuse vector. The reuse can be spatial, temporal, or group (a

combination of spatial and temporal). The localized iteration space focuses on accuracy and distance between reuse to determine if reuse translates to locality. Reuse only translates to locality if subsequent uses of data occur before the data is displaced from the cache. They use an algorithm that tries to determine whether or not information will already be in the cache since the access is a prefetch and the cache can not be directly accessed.

A prefetch predicate tries to determine whether or not a request has already been made if the instruction is not present in the cache. If a request has been made, bandwidth will be saved by not making a duplicate request. In scheduling the instructions, use the shortest path to determine and hope that it is not degenerate.

4 Our Approach

A brief discussion ensued about which approach we would use to build a system. Some would use an address generation approach and others would use some combination of both hardware and software. It was mentioned that a software approach could be used orthogonal to the Markov or non-stride versions.