CSE 240A Project: Prefetching Competition

Chiwei Tseng, Chun-Tse Shao

Description of Prefetcher:

We design our prefetcher based on Markov predictors described in the paper "Prefetcher using Markov Predictors.[1]" A Markov predictor maintains a table where possible transitions between states are saved with their respective probabilities. In our prefetcher, the states are the memory addresses requested by the CPU and missed in the L1D cache. On L1D misses, the prefetcher prefetch the data at the address whose state has the highest probability that the current state can transit into. Our prefetcher differs from the standard Markov prefetcher in the following aspects, to achieve better performances.

1. We use an array to save the state transitions. Every entry in the array contains two addresses, a source and a destination, and the number of times the transition occurs.

2. Since the storage of prefetcher is only 4KB, we take advantage of the fact that L1D miss patterns tend to repeat, and save only the transitions between two missed addresses in our table instead of transitions between all consecutive memory operations.

3. We use LRU policy to replace the entries in the array.

4. Adjustments to the number of occurrences for each transition are tuned to obtain the best AMAT on the five memory traces provided. For instance, whenever a transition that can be found in our table happens, we increase the occurrence of the transition by two. For all transitions that have the same source but different destination, we decrease their occurrences by one.

5. We verify the last prediction the prefetcher has made by tracking L1D hits after the prediction and before the next miss occurs. If the actual memory requests do not match the prediction, we decrease the occurrence of the erroneous transition by four.

6. We prefetch several addresses in a row, instead of one, by tracing through the transition table. We stop tracing only when more than 20 unissued prefetching requests are queued, or the transition probability has dropped below a certain threshold we set. For example, suppose the predicted upcoming memory misses are a —> b —> c —> d with transition probabilities of 0.9 in (a —> b), (b —> c) and (c —> d). Hence, the transition probability of (a —> c) is 0.81, and (a —> d) is 0.729. Since the probability threshold we set is 0.8, we will prefetch data at memory addresses b and c in this case and will not prefetch d.

State Accounting:

We use an array to save the transition states. There are 300 entries in the array. Each entry stores two 32-bit addresses and one integer.

```
struct Entry {
   Entry();              // Constructor
   u_int32_t _src;       // The previous state, which is 4 bytes
   u_int32_t _dest;      // The destination state, which is 4 bytes
   unsigned int _occ;  // The occurrence times of the transition, which is 4 bytes
};
```

[1] Doug Joseph, Dirk Grunwald. "Prefetching using Markov Predictors." ISCA '97, Proceedings of the 24th annual international symposium on Computer architecture. Pages 252-263.

Therefore, the total size of our transition table is: 12B * 300 = 3.6 KB.

Furthermore, we use a queue to save unissued prefetching requests. The queue has 20 elements. The total size is 4B * 20 = 80B.

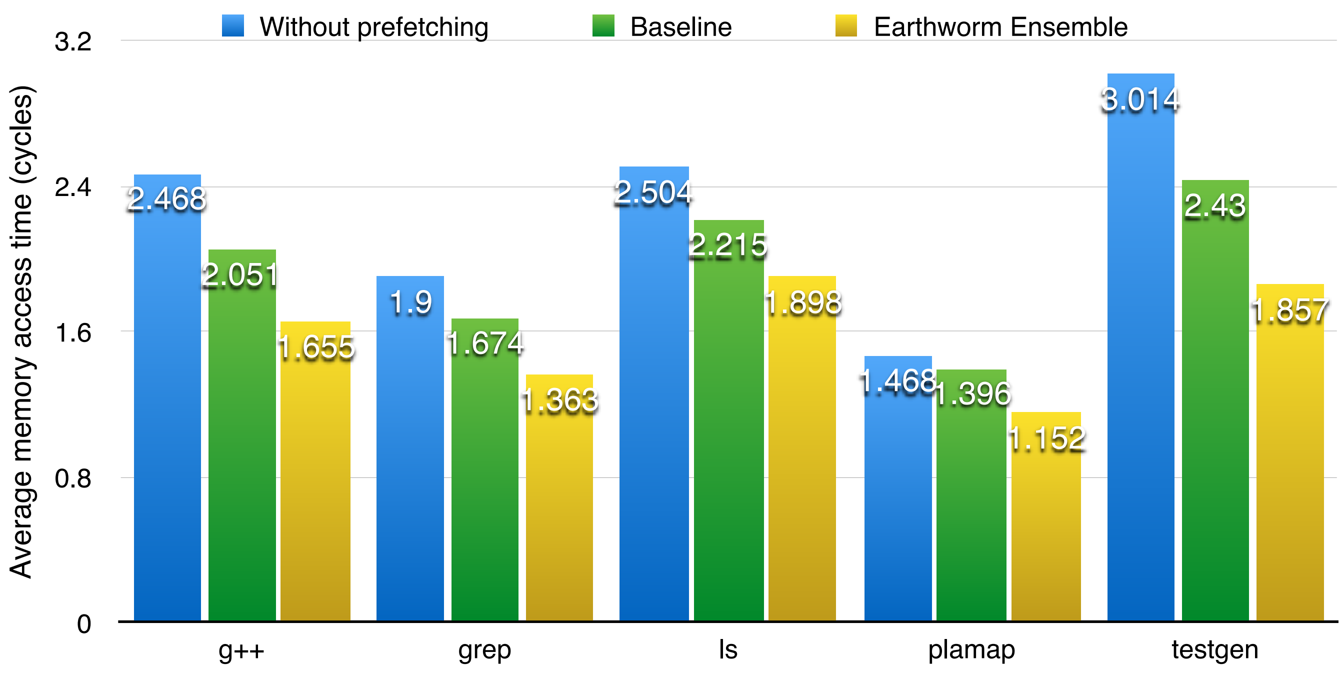
Since the request queue and the transition table are implemented as circular buffers, three integers — _oldReqCounter, _nextReqCounter and _oldEntryCounter — are used to store the producer and consumer indices. This occupies 4B * 3 = 12 B of space.

To implement the verification mechanism for the previous prediction, we store the 32-bit address of the previous prefetch target in _checkAddr, as well as its respective transition probability in _checkProb. This takes 4B * 2 = 8 B of space.

Finally, we store the requested address of the previous L1D miss in _prev that takes up 4 bytes, and two bits indicating that whether the request queue is full, and that whether the verification mechanism is turned on. This takes less than 5B of space.

So the total size of our prefetcher state is: 3.6KB + 80B + 12B + 8B + 5B ~ 3.7 KB, which does not exceed the constraint 4KB.

AMAT Graph



In the above AMAT bar chart, baseline prefetcher prefetches data in the next cache line on every memory miss. Earthworm ensemble is the prefetcher we designed.