CSE240C: Simulation Assignment 1

Instruction Prefetcher

# Summary of top 3 prefetchers

As the speed of processes and memory grows further apart, we have an ever-increasing need for find high precision prefetchers that can prefetch data into the cache. This report we try and recreate all the results of the Entangled instruction prefetcher and D-JOLT prefetcher that were the winners of the 1st Instruction Prefetching Championship competition.

## Entangled Instruction Prefetcher

The main idea of this paper is to identify cache misses and (link them) entangle them to a basic block that comes before it. Block is sequence of instructions that are (almost) sequential. Every time a cache miss is encountered the prefetcher will note down the amount of time it takes to service this cache miss. Then the prefetcher entangles this cache miss do the previous basic block that was accessed latency amount of time (time taken to service the miss) before this cache miss actually occurred. Next time the basic block is accessed from the cache, it’s entangled basic blocks are also prefetched into the L1 cache, this provides timeliness to the prefetcher scheme.

## The FNL+MMA Instruction Cache Prefetcher

This prefetcher is a combination of two prefetcher, the FNL prefetcher which is a short-range prefetcher and the MMA prefetcher which is a long-range prefetcher. The next line prefetcher suffers from cache pollution because it is too aggressive. The FNL prefetcher is a modification of the next line prefetcher which only prefetch the next 5 lines if they will be used in the near future. The MMA prefetcher is long range prefetcher that relies on the fact that a miss will be replicated after ‘n’ cache misses. This (FNL+MMA) prefetcher uses a small tag only cache (called I-Shadow cache) for both FNL and MMA prefetcher.

## D-JOLT: Distant Jolt Prefetcher

This prefetcher is a return address based prefetcher. It uses a fifo (“FIFORETCNT”) to record return addresses and the number of successive returns. The three main parts of this prefetcher are *Siggen*, *Histlen*, and *Distance*. The Siggen generates a signature from the *histlen* addresses obtained from the queue head and the return counter (which counts number of successive returns). *Histlen* tells how many of the return addresses to use to generate the signature. And the *Distance* tells how much further in the future the miss actually is. The *Distance* is used during training. The “FIFORETCNT” and “RASWHOLE“ are different because after a return the (“FIFORETCNT”) does not pop the return address. This provides higher variation in signature compares to “RASWHOLE“. The D-Jolt prefetcher is made of a Long-range prefetcher, a Short-range prefetcher, and a Fallback prefetcher. The Long-range prefetcher is the main prefetcher that tries to prefetch instructions that will be required further into the future. Not all misses are corelated to return addresses that are so far behind. So, to service these misses a Short-range prefetcher is used. And Fallback prefetcher is a stream prefetcher so service the misses that are not handled by the former two prefetchers.

# Key Design Details

## Entangled Instruction Prefetcher

### Main Data structures:

*History buffer* records the history of basic blocks heads (i.e., the first non-consecutive cache line), together with the timestamp of the triggering instruction (i.e., the instruction whose execution led to installing the block in the cache). It has a total of 1072 entries and each entry uses 58 bits for address of basic block heads and 20 bits for time stamp.

*Basic block* represents the set of consecutive cache lines (where consecutive refers to the program order of instructions, grouped in cache lines). The head of a basic block is therefore the first non-consecutive cache line. The size of the basic blocks is the number of consecutive lines. This table has 4 entries (fully associative) and uses 7 bits per entry for storing the size of the basic block.

*Timing table* records I-cache misses and prefetches. An entry consists of the tag of the cache line (58 bits), the timestamp when the miss/prefetch is triggered (12 bits), an access bit (indicating if it has been a demand access for this line), a valid bit (indicates whether the entry is used or the table is underutilized), and the corresponding source-entangled cache line (42 bits LSB). This table has a total of 42 entries.

*Entangled table* is the core structure of this proposal and encodes the entangled basic block heads. An entry contains the source-entangled cache line (34 bits LSB), its basic block size (7 bits), a compressed array of destination-entangled cache lines (up to 6 destinations, 10 bits each, total of 60 bits), and their associated confidence (is a part if the previously mentioned 60 bits). This table has 256 sets and is 34 way set associative (total of 34 x 256 entries).

*Confidence* field is initialized to the maximum value (since it was just computed prior to inserting it in the table and therefore expected to be accurate). The confidence is increased by timely prefetchers and decreased by late and wrong prefetchers. When confidence reaches 0, the destination-entangled becomes invalid (no prefetch is triggered). *Confidence* field uses 2-bit saturating counters with a threshold value of 1.

*Extended prefetch queue* stores prefetches that cannot be sent due to a full prefetch queue, in a compressed manner. It stores the first address to prefetch (58 bits), its source-entangled (58 bits), and the total number of consecutive lines to prefetch (7 bits). This table has a total of 32 entries.

### Prefetching strategy:

Whenever there is a cache access the *Entangled table* is checked.

1. The basic block that starts with this cache line (as the head) is fetched.
2. All the destination basic blocks that this cache line (head of this basic block) is entangled with is also fetched if its confidence value is greater than the threshold.

The above-mentioned fetches are added to the extended prefetch queue and serviced one by one by the processor.

The key point of this strategy is the use of basic blocks. The use of basic blocks (a collection of cache lines) removes the necessity for next line prefetcher. All other prefetchers that we saw uses some form of next line prefetching which reduces the storage available for the non-sequential part of code that needs prefetching. Ideally, we don’t want any miss, so the prefetch should be triggered early enough that the cache access is a hit. This timely prefetching is achieved using time stamps to entangle the basic blocks early enough.

### Update strategy

1. An access to non-L1 cache is registered in the timing table. The address of the requested cache line and timestamp of the instruction the triggered it is recorded.
2. When data is filled in the L1 cache, the time is recoded again and the latency of the access is calculated.
3. If the access to non-L1 cache was a late pre-fetch or a L1 cache miss, the entangled table is updated so the prefetch will be triggered in a timely manner next time.
4. Whenever there is a cache hit the access bit in the timing table is set.
5. Upon a cache line evection check if it was a prefetch. If the cache line was a prefetch and the access bit is not set then decrement the confidence of the source entangled prefetch entry in the entangled table. If the access bit was set then increment the confidence counter.

## D-JOLT: Distant Jolt Prefetcher

### Main Data structures:

*Siggen* represents an algorithm used to generate a signature based on the return addresses. The generated signatures are pushed in the signature queue to the prefetchers to use.

*Histlen* is the number of addresses used to generate a signature. The Short-range prefetcher uses a *histlen* of 4 and the Long-range prefetcher uses a h*istlen* of 7.

*Distance* indicates how much time (in terms of calls) has elapsed since the signature associated with the miss address was generated. When the distance is increased, prefetches can be issued earlier than demand accesses, but the prediction accuracy will decrease because the prefetcher needs to predict a distant future. Also, as distance increases more *histlen* is required to make an accurate prediction. The Short-range prefetcher uses a *distance* of 4 and the Long-range prefetcher uses a *distance* of 15.

FIFORETCNT is a FIFO that stores the return addresses whenever call instructions is executed (similar to RASWHOLE). The key difference between FIFORETCNT and RASWHOLE is that the former will not pop address out of the FIFO when a return instruction is executed, and the second difference is that FIFORETCNT also stores a count value that counts the number of successive returns. Both Short-range and Long-range prefetchers have their own FIFORETCNT.

*Miss table* records the association between a signature and the line address of a cache miss. The Short-range prefetcher uses 4-way set associative *Miss table* with 1024 sets (4096 entries) and the long-range prefetcher uses 4-way set associative *Miss table* with 2048 sets (8192 entries).

*Upper bit table* records only the upper bits of the miss line address to be recorded in miss table. This table is used by both Short-range and long-range prefetchers. It is fully-associative with 31 entries and 1 valid bit and 40 bits of address (MSB). It is accesses using the same signatures as the miss table.

*Extra-miss table* record the address when an address of a cache miss is outside the range of a single bit vector corresponding to a signature. This table is used by both Short-range and long-range prefetchers. It is accesses using the same signatures as the miss table. It is 4-way set-associative with 256 sets (2048 entries).

### Prefetching strategy:

This prefetcher uses 3 different prefetchers: Long-range prefetcher, Short-range prefetcher and the Fallback prefetcher. The Short-range and the Long-range prefetchers are the same FIFORETCNT based prefetchers that use different *Distance* and *histlen*.

Short-range and long-range prefetcher:

1. Every time the FIFORETCNT is updated, the *siggen* generates a new signature using *histlen* number of return addresses.
2. This signature is used to access the miss table (also upper bit table and extra-miss table) and generate the prefetch address.

The Fallback prefetcher is a simple window-based stream prefetcher.

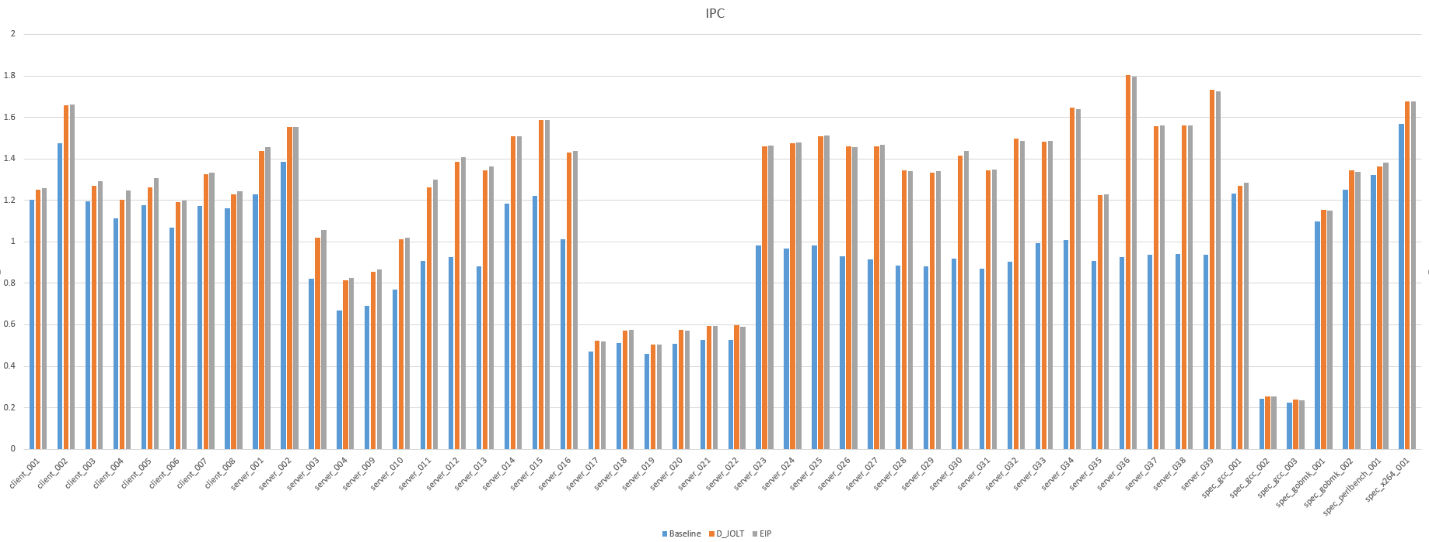
### Update strategy

1. When a cache miss is encountered the signature for the signature queue is used. This is the signature that was pushed to the queue *distance* times ago.
2. The signature is used to access the upper bit table and the miss table. If the signature is a hit and the missed line address is not within the range of the bit vector in the entry, the missed line address is recorded in the extra-miss table.

# Reproducing the results

The given Source code was used and the results given in the paper were reproduced.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Sources | Baseline (No Prefetcher) | D-JOLT | | EIP | |
| IPC | IPC | Improvement | IPC | Improvement |
| Original results | - | - | 28.9% |  | 29.5% |
| Reproduced results | 0.883685 | 1.139425 | 28.9402% | 1.145456 | 29.6227 |
| Error | - | - | 0.0402% |  | 0.1227% |



# Design space exploration

## D-JOLT

### History Length of Long range prefetcher

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| History Length | 13 | 14 | 15 | 16 | 17 |
| IPC | 1.138899 | 1.13798 | 1.139425 | 1.13838 | 1.136417 |

History length number of return addresses are used to calculate the signature. Using higher History length gives better accuracy but also takes longer to learn, on the other hand using smaller history lengths would give lower accuracy. So, we expect there to be a value for which we get the best performance and the performance should on average decrease as we pick a value farther away from the ideal value.

We can see in the graph, History length of 15 for the long-range prefetcher (which was the value chosen by the authors) gives the best IPC and the IPC of the values for the rest of the History lengths is lower.

### Distance of Long range prefetcher

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Distance | 5 | 6 | 7 | 8 | 9 |
| IPC | 1.138865 | 1.138605 | 1.139425 | 1.139079 | 1.138826 |

Distance tells how much farther in the future do we need to predict the prefetch address. If the Distance is to high the prediction accuracy lowers as the correlation between the return addresses to the address to be prefetched will be lower. If the distance is too low then, then the prefetch is not happening is a timely fashion and stalls may still occur. So, we expect there to be a value for which we get the best performance and the performance should on average decrease as we pick a value farther away from the ideal value.

We can see in the graph, distance of 7 for the long-range prefetcher (which was the value chosen by the authors) gives the best IPC and the IPC of the values for the rest of the History lengths is lower.

## EIP

### Basic Block Merge size

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| BB Size | 32 | 64 | 128 | 256 | 512 |
| IPC | 1.143443 | 1.144273 | 1.145456 | 1.145456 | 1.145456 |

The Basic Block merge size tells the granularity with which Basic Blocks should be merged. We expect the performance to saturate when this size is equal to the cache line size and to decrease as the Basic Block merge size is lowered. This lowering of the performance as Basic Block merge size is lowered is because the prefetcher now has to keep track of much more basic blocks as they were not merged together, this will result in aliasing in the prefetcher data structures.

We can see in the graph, granularity of 128 (which cache line size of the processor) gives the best IPC and the IPC of the values decrease as Basic Block merge size is lowered.

### Threshold for confidence field

|  |  |  |  |
| --- | --- | --- | --- |
| Threshold | 1 | 2 | 3 |
| IPC | 1.145456 | 1.144946 | 1.145617 |
| IPC (Top 5 traces) | 1.210792 | 1.208506 | 1.210154 |

Threshold for the confidence field tells after how many wrong prefetches should that prefetch be invalidated. The confidence field is initialized to maximum (3 – 2-bit counter). So, a lower Threshold means that wrong prefetches should be given more chances. A high threshold would mean a wrong prefetches should be given less chances. Predicting the trend in for this variable is difficult and it mainly depends on the workload.

We can see from the graph that a threshold value of 2 gives lower performance. This is because prefetches that are wrong (in the long run) are getting a chance to stay valid and prefetch, while the prefetches that are going to alternate between (correct and not correct) do not get prefetched. This behavior is similar to a 1-bit branch predictor trying to predict odd and even branches in a for loop.

We can see that for the top 5 traces (top 5 w.r.t. MPKI on baseline model) the threshold value of 1 gives a better result than 3 and for all traces the results are inverted. I believe that the difference between IPC for all traces was not too big between threshold value of 1 and 3, so the authors decided to choose threshold value of 1 which gives a better performance on top 5 traces.

### Destination blocks

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 2 | 3 | 4 | 5 | 6 |
| IPC |  |  |  |  |  |

# Hardware budget exploration

|  |  |  |  |
| --- | --- | --- | --- |
| Hardware size | 63 KiB | 128 KiB (Original) | 250 KiB |
| D-JOLT (IPC) | 1.130787  (2-way) (513,623 bits) | 1.139425  (4-way)  (1,023,062 bits) | 1.141329  (8-way) (2,041,943 bits) |
| EIP (IPC) | 1.138372  (14-way) (515,945 bits) | 1.145456  (34-way) (1,048,425 bits) | 1.147377  (71-way)  (2,033,513 bits) |

We evaluate the prefetchers for different hardware budgets by changing the associativity of their primary data structures.

For D-JOLT, the associativity of miss table (of both short-range and long-range predictors) and extra miss table was changed. Associativity of 2, 4(original) and 8 was used. Then the sizes for these predictors were calculated.

For EIP, the associativity of the entangled table was varied to match the size of the D-JOLT prefetcher as closely as possible.

The sizes of original prefetchers were left unchanged.

We can see from the graph that as hardware size increases, the IPC also increases. But we can see that the returns are diminishing as the size increases. Comparing the two prefetchers we see that EIP comes out on top for all sizes.

EIP performs better than D—JOLT on lower budgets because:

1. D-JOLT is a combination of 3 prefetchers the stitched together to improve the design instead of solving the underlying problem with each prefetcher. Because D-JOLT lacks the concept of basic blocks it has to employ a Fallback prefetcher (stream prefetcher) to handle sequential code execution (without jumps). This leaves lower hardware budget for the main prefetchers.
2. EIP uses address compression more efficiently to lower resources usage. This allows more budget to spent of number of table entries which in turn reduces aliasing.
3. D-JOLT losses because it has to rely on 3 different predictors and a lot of the hardware budget is spent of bookkeeping bits like LRU etc.

# Main computations in the prefetcher