1 Objective

Create a tool which will count the number of bit transitions on the memory data bus, which can instrument realistic programs.

2 Short Introduction

We have used Intel Pin to implement the tool as discussed.

Intel Pin provides a very simple mechanism to instrument all memory accesses of the program it attaches to, so it is straightforward to detect every read/write operation.

The problem here was, **there is no way to detect which memory accesses are actually done to the external memory using Intel Pin**, as it can only detect the calls the program makes. In a realistic scenario with a standard cache implementation, most of the accesses would be to the internal caches, and only in the case of the last level cache miss the memory would have been accessed. Our solution to this was **to simulate a simple cache hierarchy and count the number of transitions on the data bus only when the unified L3 cache has a miss.**

3 Pin Tool Implementation

We have examined the tools supplied by Intel, and found out that there is a tool named *allcache*[1] which simulates the following:

- Separate L1 caches for data and instruction
- Unified L2 and L3 caches
- Separate TLB for data and instruction

The cache parameters are easily configured, however **the tool can only simulate the architecture of a single core cache hierarchy.** We assumed that this tool would be a good enough starting point, so we decided **to modify the tool slightly to allow us to count the transitions in case of L3 misses.**

The supplied tool was already able to detect L3 misses, however the accesses were done in bulk and it was reported as a miss in case of any of the accesses being a miss. **We have replaced the bulk access function with a single line access function (in a loop) to detect the misses**

[1] location: pin/source/tools/memory/allcache.cpp

at line access granularity. We have also added the necessary code to copy the data (using *PIN_SafeCopy*) residing in memory to a buffer and then do the actual transition counting.

3.1 Transition Counting

We have assumed that the memory is accessed with the granularity of the L3 cache line size (typically 64B). After doing some research into how DDR SDRAM read/write operations work, we have understood that the typical bus width is 8B (+1B in case of error correction), so the data is read in bursts of up to 8 (or lower and more concatenated read operations) to accomodate for the L3 cache line size.

The data bus is in a high impedance state when no read/write operations are being carried out, so in our implementation we ignored the transition count from the high impedance state to the first word (8B in size) being read/written. We also do not count the transitions from the last word to the high impedance state.

To count the number of transitions on the data bus, we have implemented a **look up table (LUT)** with size 256*256 (64kB). The row is chosen by the first byte to be compared, and the column is chosen by the second one. The read result is a single byte value between 0 to 8. To fill the look up table the tool calls an initialization function which calculates **the hamming distance** between the two bytes using a xor to set the differing bytes and then applying Brian Kernighan's algorithm[2] to count the number of bits inside a byte.

The number of transitions are counted and accumulated for every byte of neighboring 8B words (i.e. bus width) being transmitted as shown in Figure 1.

4 Output

The default *allcache* tool outputs the cache and TLB access/hit/miss statistics and miss ratios. As we have built on top of the existing tool, we have kept this output format and added the number of bit transitions to the bottom as shown in Figure 2.

[2]https://graphics.stanford.edu/ seander/bithacks.html#CountBitsSetKernighan

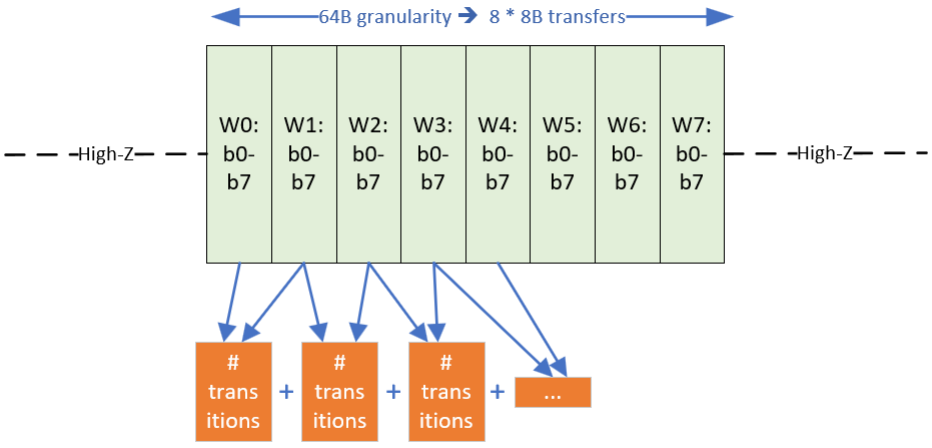

Figure 1: Calculating the transition count on the memory data bus

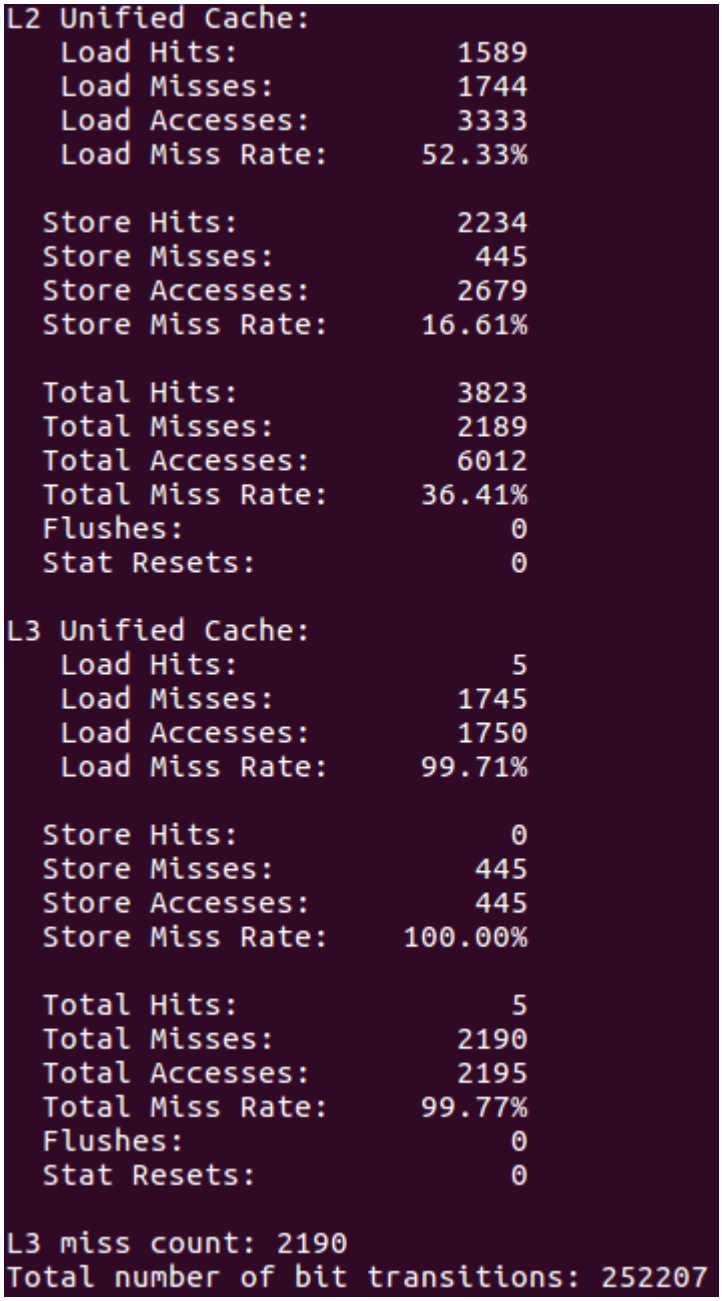

Figure 2: Tool output for linux ls