ECE 585 Data Structure Explanation

Chris Kane-Pardy
Caleb Monti
Evan Brown
Ameer Melli

November 19, 2024

Here is the mathematical breakdown of everything we used to determine our caches' parameters:

Cache Size: 64MB

Line Size: 64 Bytes

Address Size: 32 Bits

$$\frac{2^{26}}{2^{6}} = 2^{20} \rightarrow \text{1Mi Cache Lines}$$

$$\frac{2^{20}\text{ lines}}{\text{16 lines (ways) per set}} = \frac{2^{20}}{2^{4}} = 2^{16} \rightarrow \text{64Ki sets of 16 lines}$$

$$\text{Byte Select} = \text{64 Bytes} = \lceil log_2(64) \rceil = \text{6 bits}$$

$$\text{Index} = \text{64Ki} = \lceil log_2(2^{16}) \rceil = \text{16 bits}$$

$$\text{Tag} = \text{32 - (6 + 16)} = \text{10 bits}$$

$$\text{PLRU} = \text{(n - 1)} = \text{(16 - 1)} = \text{15 bits per set}$$

$$\text{MESI} = \text{4 states} \rightarrow \lceil log_2(4) \rceil = \text{2 bits per line}$$

$$\text{Total Tag Array} = 2^{16}((16 \times (10 + 2)) + 15) = \text{13,565,952 bits} = \text{1,695,744 Bytes}$$

Address Breakdown:
The six least significant bits will be used for the byte select (offset from the nearest aligned block of 64 bytes).

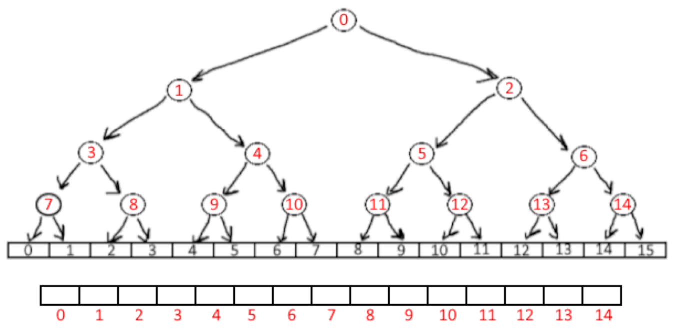
The next 16 will be used for the index (which will designate which set we're in, w/ options 0 through $(2^{16} - 1)$), which corresponds to our 64Ki possible sets.

The tag will be the remaining 10 upper bits of the address.

Tag	Index	Byte Select
|A31 : A22|	|A21 : A6|	|A5 : A0|

Pseudo LRU Explanation:
The PLRU replacement policy will consist of 15 bits for each of the 64Ki sets, which will be used to track the "Pseudo" Least Recently Used. To model the tree seen below, we can use an array, which will track the most recently used way, then, when the time comes for an eviction, the traversal method for accesses (0 = Left, 1 = Right), will be flipped for replacement (0 = Right, 1 = Left).

Pseudo LRU for 16 Way Set Associativity



When traversing the tree, the bit for each node (whose position is written in red will be stored in an array, where the node number in the tree diagram corresponds to its position in the array), will be updated to reflect the most recently used way in the set. Then for eviction, those same bits will be followed, but taking the opposite direction. After an eviction, the path taken to determine the LRU should have its bits flipped (i.e. $1 \rightarrow 0$, and $0 \rightarrow 1$). The 15-bit PLRU array for each set will be stored under the same index.

Note- The method for storing each node as an array address follows the following algorithm:
Left Child $= 2i + 1$
Right Child $= 2i + 2$
Where i is the currently selected node in the tree

Sets (Index) Breakdown:

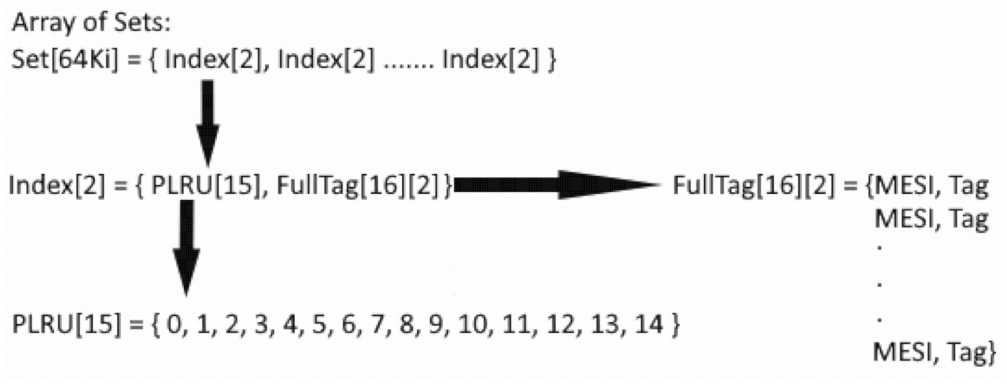
For our sets, we plan to use an array, *Set*$[2^{16}]$, where each element of the array will correspond to the hashed **index** value from the address of the data input via a trace file.

At each element of *Set* will be another array, *Index*, which will contain two components, each of which will also be an array. The first is a one-dimensional array, which will have 15 elements (node 0 through node 14) ($PLRU[15]$). The second will be a 2-dimensional array which will have 2 columns and 16 rows and will store the **MESI bits** and the **Tag** which have been extracted from the address ($FullTag[16][2]$).

MESI Breakdown:
The MESI portion will be comprised of 2 bits, which will signify whether the tag which they accompany is either **Invalid** (00), **Shared** (01), **Exclusive** (10), or **Modified** (11). They will be stored in the *FullTag* array in its first (0th) column and will be updated via the snooped results from the other processors.

Tag Breakdown:
The tag will be the highest 10 bits of the address and will be stored after the MESI bits in the *FullTag* array. They will be checked after it's verified that MESI for that way of the set is set to one of the three valid states, otherwise, it will be ignored.

Overall Data Structure:
The overall data structure is pictured below, demonstrating the access path once an index has been entered.


Array of Sets:
Set[64Ki] = { Index[2], Index[2] .... Index[2] }
Index[2] = { PLRU[15], FullTag[16][2] }
FullTag[16][2] = {MESI, Tag
MESI, Tag
.
.
.
MESI, Tag}
PLRU[15] = { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14 }