UNIVERSITY OF TEXAS AT ARLINGTON

ADVANCED MICROPROCESSOR SYSTEMS

FALL 2020

CACHE CONTROLLER

Guided by,

Prof. Dr. Jason Losh

Project by,

Mihir Marathe (1001738780)

Nikhil Kumbhar (1001775805)

PROJECT OVERVIEW:

The goal of this project is to determine the best architecture for a cache controller that interfaces to a 16- bit microprocessor that is used for signal processing. While the microprocessor is general purpose in design and performs a number of functions, it is desired to speed up certain signal processing functions, such as those calculating the eigenvalues and eigenvectors of a system.

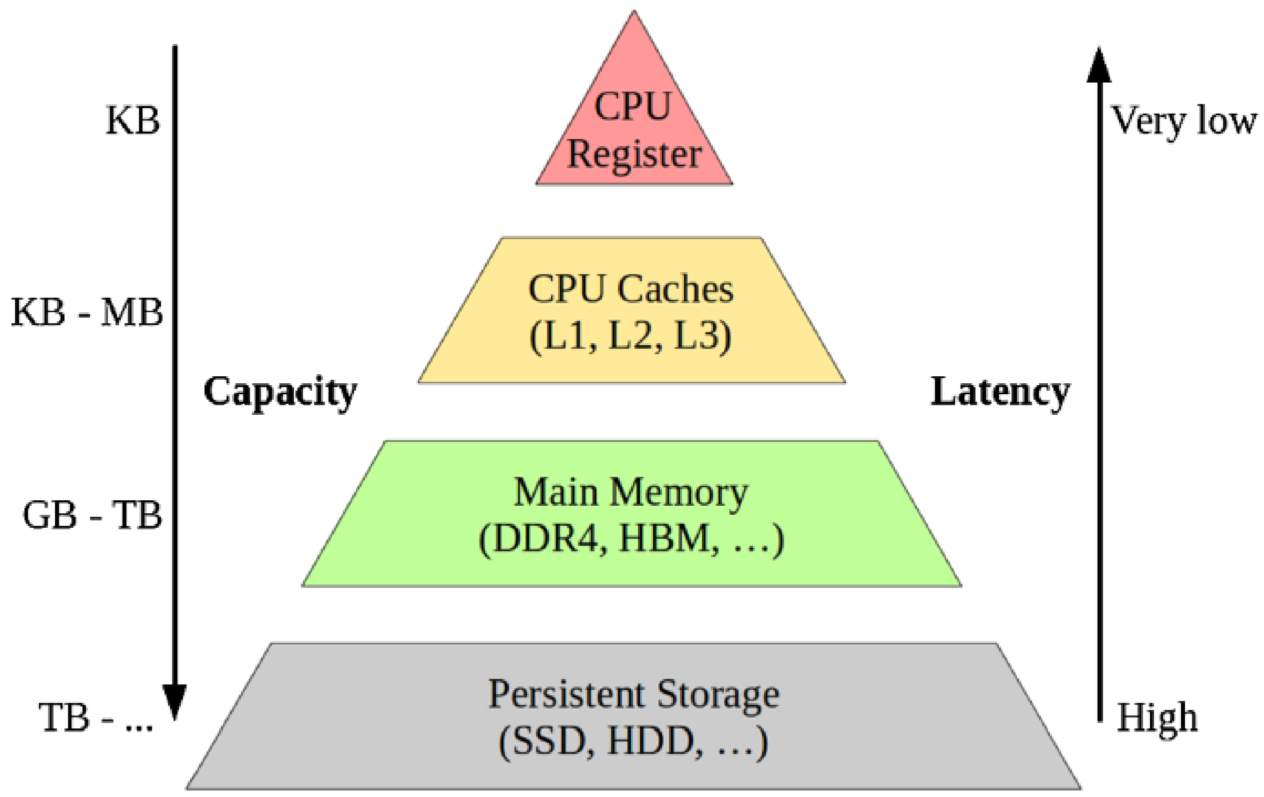
KEY REQUIREMENTS:

- Your project should seek to allow 4GiB of memory (1 Gi x 32 bit) to be interfaced and the cache should be limited to 128 KiB in size
- After the program being executed was profiled, the largest hit in performance was seen in functions called choldc() and cholsl() which operate on a 256x256 symmetric matrix of single-precision real floating numbers
- Please use the assumption that the miss penalty is 60 ns (for first memory access) and 17ns for subsequent accesses in the same SDRAM word line and the cache hit time is 1 ns.
- Please calculate the performance as a function of set size (n-way), number of cache lines, block size, and write strategy (write-back allocate, write-through allocate, and write-through non-allocate). Determine which 3 configurations result in the best performance. At a minimum, n-way associativity of 1, 2, 4, 8, and 16 and burst lengths of 1, 2, 4, and 8 should be evaluated for all 3 write strategies (60 permutations).

➢ WHAT IS CACHE?

A cache is a small block of data that is designed to helps the processor, retrieve data that is frequently used. It stores the data that is used by the processor to execute the instructions. The cache is placed between main memory and processor thus providing faster access time to the processor than traditional time was there between main memory and processor.

➢ MEMORY HIERARCHY



https://www.google.com/url?sa=i&url=https%3A%2F%2Flink.springer.com%2Farticle%2F10.1007%2Fs00778-019-00546-z&psig=AOvVaw21-hgMXd8pOAijY0SE3IzG&ust=1606994301512000&source=images&cd=vfe&ved=0CAIQjRxqFwoTCKCxsraWr-0CFQAAAAAdAAAAABAO

TYPES OF CACHE

There are three types of cache:

- DIRECT MAPPED CACHE: n a direct-mapped cache, the cache is organized into multiple sets with a single cache line per set. Based on the address of the memory block, it can only occupy a single cache line.

- FULLY ASSOCIATIVE CACHE: A fully associative cache permits data to be stored in any cache block, instead of forcing each memory address into one block. When data is fetched from memory, it can be placed in any unused block of the cache.

- N – WAY SET ASSOCIATIVE CACHE: An N-way set associative cache reduces these conflicts by providing N blocks in each set where data mapping to that set might be found. Each memory address still maps to a specific set, but it can map to any one of the N blocks in the set. We have used this in our project.

GET LINE LOGIC

```
uint32_t get_line(uint32_t add){

uint8_t a;

a =(tag_bits + (log2(B)));

Line = (add<<tag_bits)>>a ;
return Line;
}
```

Here the address is decoded to get the line.The line is calculated by shifting the address bits left by no tag bits and then shifted right by Block size ie 2 in our case to get the line.

GET TAG LOGIC

```
uint32_t get_tag(uint32_t add){

   uint32_t Tag;

   uint8_t a;

   tag_bits = (DW - (log2(L)) - (log2(B)) );

   a =((log2(L)) + (log2(B)));

   Tag = add >> a;

   return Tag;

}
```

Here the address is decoded to get the tag. The tag is calculated by shifting the address bits right Log2(L) + Log2(B) bits .

➢ HIT AND MISS LOGIC

- When the processor needs some data, rather than accessing the main memory every time (as it increases the memory access time by a huge amount), it will first search the CACHE.
- If the CACHE contains the data which is being requested, it counts as a HIT.
- Each line in CACHE will have its own tag and the tag from the data(that is being requested), if they match its HIT.
- If the data is not in the CACHE, it's a MISS.
- The number HITs should always be more than the number of MISS, and that is the goal of a optimizing a CACHE controller.

➢ LEAST RECENTLY USED LOGIC

- For the number of HIT's to be greater than number of MISS's, we use a replacement policy in our project called LEAST RECENTLY USED(LRU).
- The LRU eviction scheme is to remove the least recently used data when the cache is full and a new data that is most recently used stored in it.
- This helps the CACHE to keep the most frequently used data within itself, and hence the number of HITs would be more than the number of MISS.

➢ INTERACTION POLICIES WITH THE MEMORY(WRITE_STRATEGY)

There are three types of interaction policies**:**

- WRITEBACK (WB):
  In WRITEBACK, if we get a hit, it goes to cache, gets the data from cache.
  If it's a MISS, it finds the block to replace in the cache, and reads data to cache from memory.
  It never writes to memory(d=1), until fushed.

- WRITE THROUGH ALLOCATE:
  In WRITE THROUGH ALLOCATE, regardless being a HIT or MISS, it always writes to memory and writes to cache.
  Therefore it never has dirty bit set to 1.

- WRITE THROUGH NON-ALLOCATE:
  In WRITE THROUGH NON- ALLOCATE, regardless being a HIT or MISS, it always writes to memory.
  When it is a HIT, it writes to cache otherwise it doesn't write to cache.
  This also has the dirty bit set to 0 always, as the data will always be written to memory.

We have shown the implementation of all the three interaction policies.

Write strategy	Cache	MEMORY
WB	ALWAYS	NEVER
WTNA	IF HIT = 1	ALWAYS
WTA	ALWAYS	ALWAYS

CALCULATION FOR PERFORMANCE COUNTERS :

Formulae

1 Total Read Time = (RLH_count * 1nsec) + (RLM_count*(60nsec+(BL-1)*17nsec)) + (RLMD_count*(60nsec+(BL-1)*17nsec))
2 Write Time for WB = (WLH_count*1nsec) + (WLM_count * (60nsec+(BL-1)*17nsec)) + (WLMD_count * (60nsec+(BL-1)*17nsec))
3 Write Time for WTNA = (WT_count*60nsec) + (WLH_count*1nsec)
4 Write time for WTA = (WT_count*60nsec) + (WLH_count*1nsec) + (WLM_count*(60nsec+(BL-1)*17nsec))
5 Write Time for (WB,WTNA,WTA)
6 Total Time for WB = Total Read Time + Write Time for WB + (FCD_count*60nsec)
7 Total Time for WTNA = Total Read Time + Write Time for WTNA + (FCD_count*60nsec)
8 Total Time for WTA = Total Read Time + Write Time for WTA + (FCD_count*60nsec)
9 Total Time for (WB,WTNA,WTA)
10 Write line HIT ratio = (WLH_count/(WLH_count+WLM_count))
11 Read line HIT ratio = (RLH_count/(RLH_count+RLM_count))
12 Total HIT Ratio = Read line HIT Ratio + Write Line HIT Ratio
13 Total MISS Ratio = (1 - Read line HIT Ratio) + (1 - Write line HIT Ratio)
14 Total Memory access time = (Total HIT Ratio * 1nsec) + (Total MISS Ratio *(60nsec+(BL-1)*17nsec))

Conclusion:

The minimum time required to access the memory(read time + write time)
Was found minimum in the case of WRITEBACK(BL=2, N=1).