

Preview

CPU cache is essential part of any modern processor design. Its purpose follows from speed limits of DRAM – there are several levels of memory with different ratios of the capacity to the speed between CPU and DRAM. This design allows to neutralize the difference in the speed of the CPU and DRAM.

Application Design

This application simulates operations in CPU cache of one of the possible configurations. Current cache parameters of the app as follows:

Cache level	Cache type	Capacity, Kbyte	Block size, Byte	Associativity, ways	Write policy
L1	Instructions	32	64	8	
L1	Data	32	64	8	Write-back, write allocate
L2	Unified	32	64	8	Write-through, write allocate

```

v cachesim
  v src
    v cachesim
      > Main.java
    v cachesim.BinaryTree
      > BinaryTree.java
      > BinaryTreeNode.java
    v cachesim.Cache
      > Cache.java
      > CacheAccessParameters.java
      > CacheBlock.java
      > InstructionTypes.java
      > Stats.java
    v cachesim.PLRU
      > AbstractPLRU.java
      > BitBasedPLRU.java
      > TreeBasedPLRU.java
  > JRE System Library [jdk1.8.0_65]
  
```

The design is not complicated, but it allows flexible change of the cache configuration if necessary.

All cache logic completed with class *Cache* within package *Cache*. This package also contains other service classes that are needed for *Cache*.

The *Cache* class could be instantiated with different parameters as follows: size of instructions (bytes), cache capacity (Kbytes), cache block size (bytes), ways of associativity and PLRU type. Thus, creating instances of *Cache* with different parameters, we could simulate different configurations of CPU cache so.

Also, there is *PLRU* package. It contains classes implementing Pseudo LRU. The app allows create two type of PLRU – bit based and tree based. Reference to instances of both PLRU types could be passed as *Cache* class constructor parameters, thereby any *Cache* instance could be combined with any PLRU (or it could be missed and cache will be simulated without PLRU).

BinaryTree package contains implementation of binary tree used by *TreeBasedPLRU* class.

CPU cache bus provided transport used by different levels of cache. Via this bus CPU organizes interactions with all caches. Implementation of bus behavior is modeled in *Main* class. So if bus logic needs to reorganize we could just change the *Main* logic.

In *Main* class instances of *Cache* and *PLRU* created with characteristics from table above. Also, victim cache instance created for L1 cache level . Further, *Main* class contains algorithm which instruction by instruction trace the source trace file.

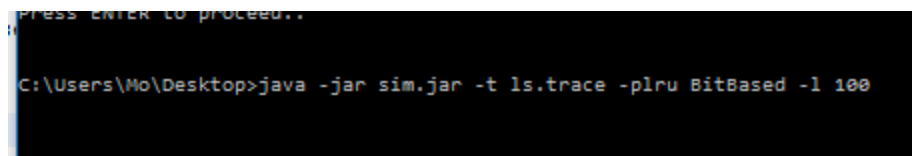
Usage

Application requires mandatory command line option *-t trace_path* where *trace_path* is the full path to source trace file with instructions.

Also, there is two optional arguments:

-l count_limit, where *count_limit* is the number of reads limit.

-plru plru_type, where *plru_type* is could be *BitBased* or *TreeBased*. It changes the PLRU type.



```
Press ENTER to proceed..  
C:\Users\Mo\Desktop>java -jar sim.jar -t ls.trace -plru BitBased -l 100
```

Let's try to start the app with the following command line parameters: *-t ls.trace -plru TreeBased*
The trace result output below:

```
Trace statistics:

Read 5242852 rows.

L1 data cache:
  Total hits: 134193
  Total misses: 4421149
  Total accesses: 4555342
  Hit rate: 2,95%
  Miss-rate: 97,05%

L1 instruction cache:
  Total hits: 569840
  Total misses: 117670
  Total accesses: 687510
  Hit rate: 82,88%
  Miss-rate: 17,12%

L2 cache:
  Total hits: 50330
  Total misses: 4488489
  Total accesses: 4538819
  Hit rate: 1,11%
  Miss-rate: 98,89%

Press ENTER to proceed..
```

As we could see the result counts total hits and misses and accesses, hit and miss ratios for each used cache level.

Let's run the app with other configuration – we will use *gz.trace* trace file (with 10 000 000 reads limit) and bit based PLRU: *-t gz.trace -plru BitBased -l 10000000*

The trace result output below:

```
Trace statistics:

Read 10000000 rows.

L1 data cache:
  Total hits: 1316553
  Total misses: 7466889
  Total accesses: 8783442
  Hit rate: 14,99%
  Miss-rate: 85,01%

L1 instruction cache:
  Total hits: 912607
  Total misses: 303951
  Total accesses: 1216558
  Hit rate: 75,02%
  Miss-rate: 24,98%

L2 cache:
  Total hits: 262093
  Total misses: 7508747
  Total accesses: 7770840
  Hit rate: 3,37%
  Miss-rate: 96,63%

Press ENTER to proceed..
```