LINGI2241 CACHE SIMULATOR REPORT

Students :
Syntyche Shimbi Amunaso 00621300

Mbungu Safi Alain 67261600

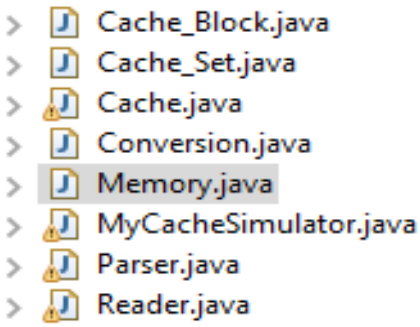
Lecturer: Ramin Sadre

1. Introduction

We will discuss in this project on what would be the best cache configuration for our trace based on the number of rows, the size of a cache line, the set-associativity, the replacement policy and the Write policy, in order to reduce the miss rate. The choice of a level in cache hierarchy being judicious, we only simulated a L1 cache and ran our pin trace file containing des addresses and several memory access in read or write.

2. Architecture of the program

In this part, we explain different elements and module used for making this application. It's important the report that we used 32-bit addresses. Our application is divides in the following 8 classes:



a. Parsing Module

We have collected 1048576 memory-access trace generated by the Pin tool . The resulting file is named printrace.txt and it is included in the program directory/src.

b. Address Partitioning

For partitioning a given 32-bit address in set-associativity, we observed the basic rules by computing:

*cache_size = number_rows * cache_blocksize * set_associativity * 1024 / 4*
*tag = address / (sets.length * blocksize)*
index = (address / blocksize) % sets.length
offset = address % blocksize

c. Memory class

The main memory has been implemented as an integer array of size 4,096 MB. Broken down in words. Words addresses are simply indices of this array. Words data are initialized by default to indices values (address locations) and will be changed dynamically during the execution of the program.

d. Cache

Structure

The set has been implemented by a class Cache_Set which reads, writes, search for and apply LRU on blocks. In this class, the cache is implemented as an array of array: it is an array of blocks and blocks are arrays of words. This class also applies LRU replacement policy.

e. Block

Cache blocks have also been implemented with a class named Cache_Block. This class has various fields namely: block tag, validity of the block, the dirtiness of the block, and a field for defining how recent the block is. If a single word is written to the cache, the validity and the dirtiness field are set to true. If all the words belonging to a block are written to the cache, then the dirtiness field is set to false. If a word of a block is dirty, the entire block is rendered inaccessible by other processes.

At the program start-up, the cache is empty and it is updates when the processor reads or write data to the cache. The processor looks for data at a certain location or address in the cache. If a datum is found in the cache, the read counter or write counter is incremented as well as the hit counter. Otherwise, the miss counter is incremented. If the data is not in the cache, then it looks for it in the memory and copy it back to the cache.

3. Output of the simulation

A Hit rate

The following figure shows the hit rate on the vertical axis we have the hit rate and on the horizontal axis we have the block size.

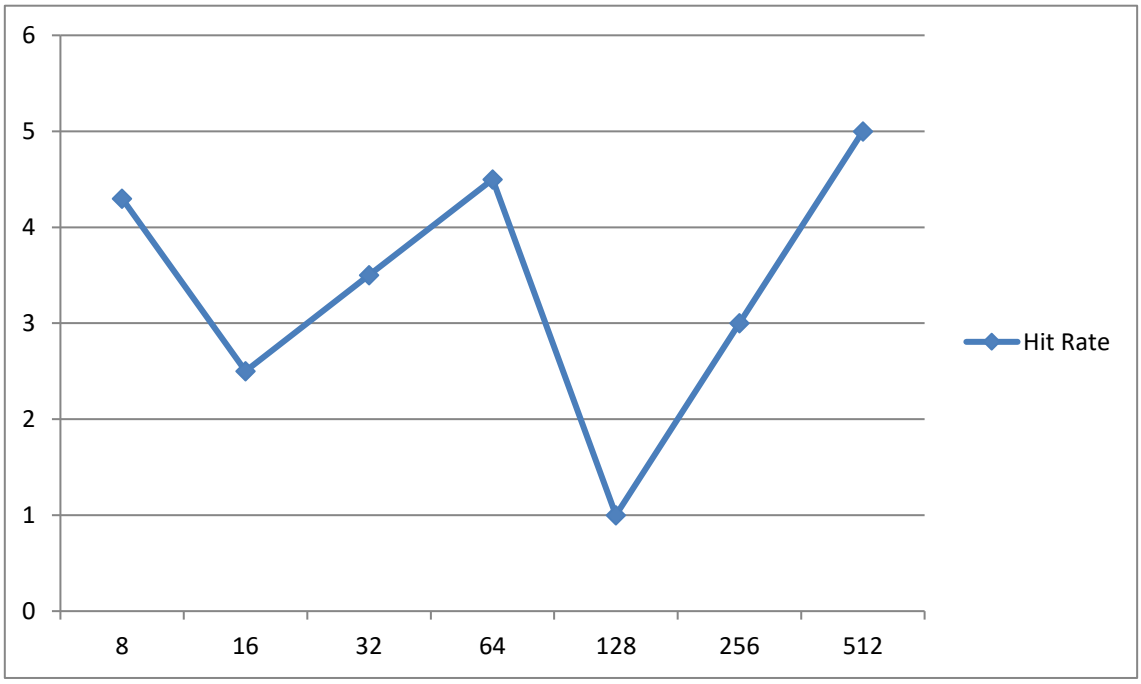
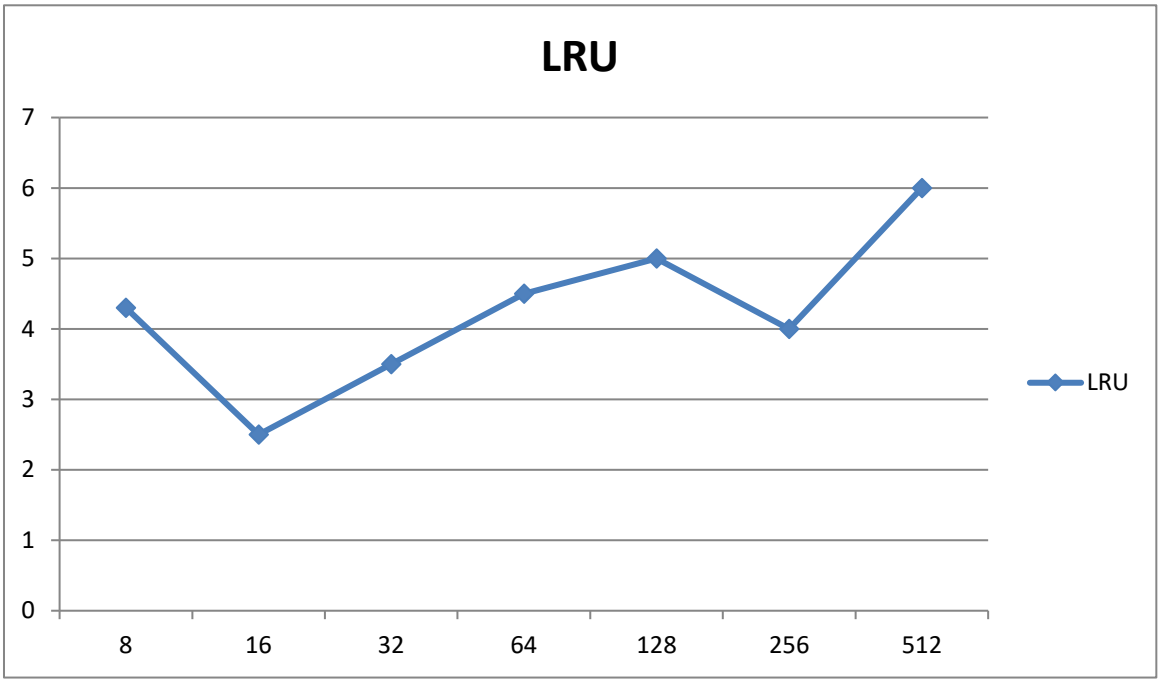


Figure 1. Hit rate per cache sizes

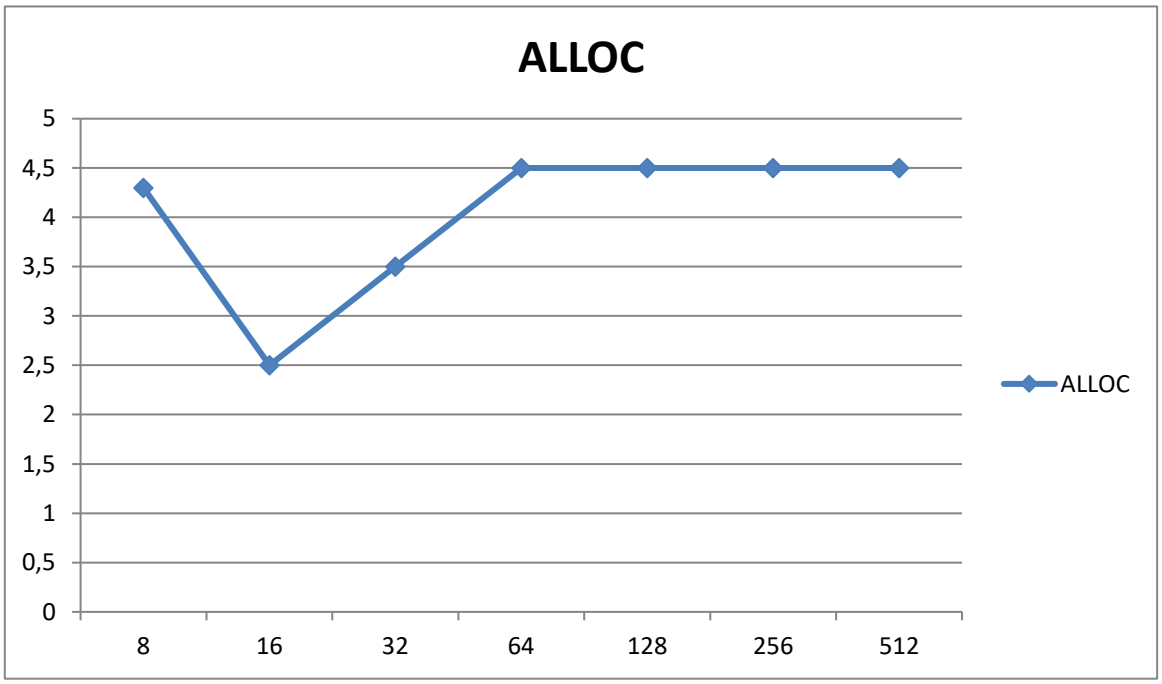
B Replacement Policies

The following figure shows the hit rate on the vertical axis LRU we have the hit rate and on the horizontal axis the cache size.



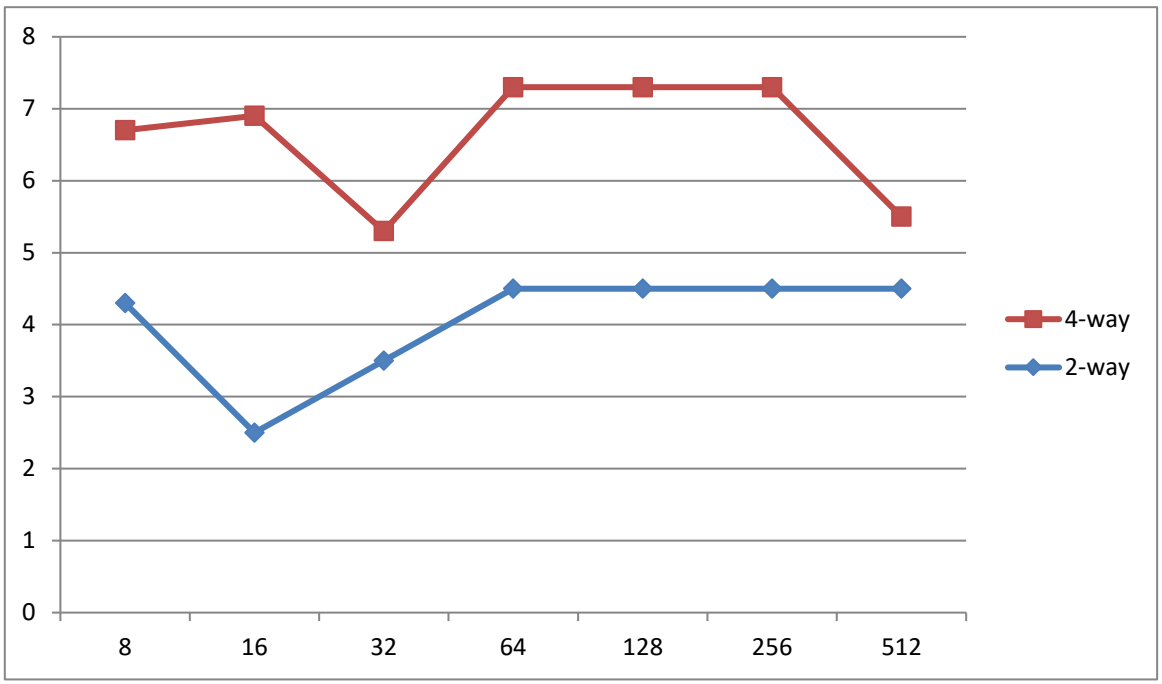
C Writing Policy

The following figure shows the hit rate on the vertical axis ALLOC we have the hit rate and on the horizontal axis the cache size.



D way et 4-way associativity

The following figure shows the hit rate on the vertical axis 4-way and 2-way we have the hit rate and on the horizontal axis the cache size.



4. Program running Challenges

We used different Java EDI Eclipse and Java. In this two, there ae different rules for variable declaration and initialization. For example for our split method used by the parser: it works with "\t" in Netbeans, but in java we have to pass it "\\s" for it to work. It took us a lot of time to find out.
The program is working. However, it takes a while to show the results for over 1000 000 instructions.

5. Conclusion

In this report, we have discussed in this project on what would be the best cache configuration for our trace based on the number of rows, the size of a cache line, the set-associativity, the replacement policy and the Write policy, in order to reduce the miss rate.

It was interesting to find that associativity wasn't as important as block size or capacity in optimizing our cache, although without other considerations the maximum of everything capacity, block size, andassociativity gives the best results