Hussam El-Araby (900132161) and Bishoy Boshra Labib (900132990)

Prof. Mohamed Shalan CSCE 2303-02 Dec. 9th, 2014

Project Report (cache memory simulation)

Introduction

This project implements a C++ program simulating the performance of direct-mapped, fully-associative, and set-associative caches. The main purpose of the program is to test and study how the cache miss ratio varies, keeping the cache size and DRAM size constant, with respect to different parameters such as type of cache, line size, associativity, and the address generating function used.

The details of the variables used in this simulation are the following:

- Constant number of instructions simulated: 1 Million.

- Constant DRAM size: 64 MiB
  Constant Cache size: 32 KiB[1]

- Direct mapped cache:
  - Variable Line size: 8, 16, 32, 64, 128 B

- Fully associative cache:
  - Variable Line size: 8, 16, 32, 64, 128 B

- Set associative cache:
  - Constant Line size: 16 B
  - Variable Set associativity: 2, 4, 8 ways

- Variable Address generation functions[2], namely:
  - memGen(1)
  - memGen(2)
  - memGen(3)
  - memGen(4)

[1] Binary prefixed bytes, 1 KiB is 2 to the power of 10 bytes, 1 MiB is 2 to the power of 20 bytes.

[2] The implementation of the address generation functions is documented later.

Simulation Method

For each run of the simulation:

1. Obtain (or calculate):
    i. Cache line size.
    ii. Number of cache lines
    iii. Number of ways (for set associativity).
2. Create a cache (as an abstract data type) with the corresponding parameters.
3. Loop the simulation function for a number of times equal to the number of instructions.
4. For each iteration of the loop:
    i. Generate a random address with a certain generation function.
    ii. Check if this address is valid in the cache.
    iii. If found, return HIT, else, update the cache line and return MISS.

Repeat the method with the different parameters stated in the introduction.

Notes:

- The cache update policy:
    - Direct mapped: replace the cache line which the address maps to.
    - Set associative:
        1. Determine the set which the address belongs to.
        2. Replace a cache line in a random “way” from this set.
    - Fully associative: replace a random cache line.
- The cache names in this report are be abbreviated to:
    - DM = direct mapped
    - FA = fully associative
    - SA[N] = set associative (where [N] is the number of ways)

Hussam El-Araby (900132161) and Bishoy Boshra Labib (900132990)

Prof. Mohamed Shalan CSCE 2303-02 Dec. 9th, 2014

Results

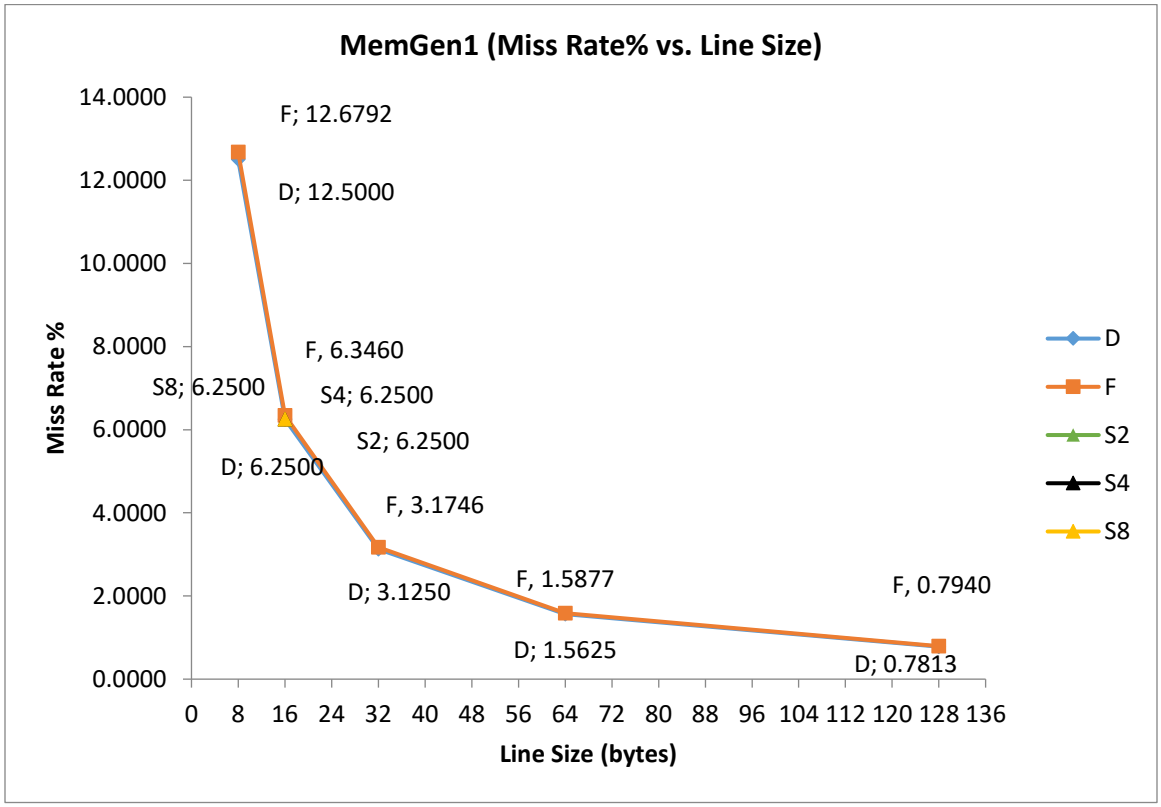
Address generation type 1

```
unsigned int memGen1()
{
        static unsigned int addr = 0;
        return (addr++) % (DRAM_SIZE);
}
```

Algorithm Analysis:

This function generates sequential addresses, 0, 1, 2, 3, 4... (DRAM size – 1), and repeats the loop once there have been more than DRAM size number of executions (or DRAM size multiples).

Obtained Results:



Observations:

- DM and FA miss rates are almost equal at any line size.
- The miss rates of DM and FA are inversely proportional to the line size (as the line size increases the miss rate decreases).
- SA is almost equal to DM and FA at line size 16.
- There is no considerable difference between SA2, SA4, or SA8.

Explanation:

A sequence of addresses will be stored one into a line, will return MISS the first time, and will always return HIT for the next (n-1) times, and where "n" is the line size.

Effect of increasing line size: Increasing line size increases the data stored into the cache after one miss, hence, a decrease in occurrence of a miss because of **Spatial Locality**. This explains lower miss ratio.

Also note that the number of executions (1 million) is less than the DRAM size (64 MiB). So, no address would repeat itself. Therefore, there is always possibility of a miss during the whole simulation.

Hussam El-Araby (900132161) and Bishoy Boshra Labib (900132990)

Prof. Mohamed Shalan CSCE 2303-02 Dec. 9th, 2014

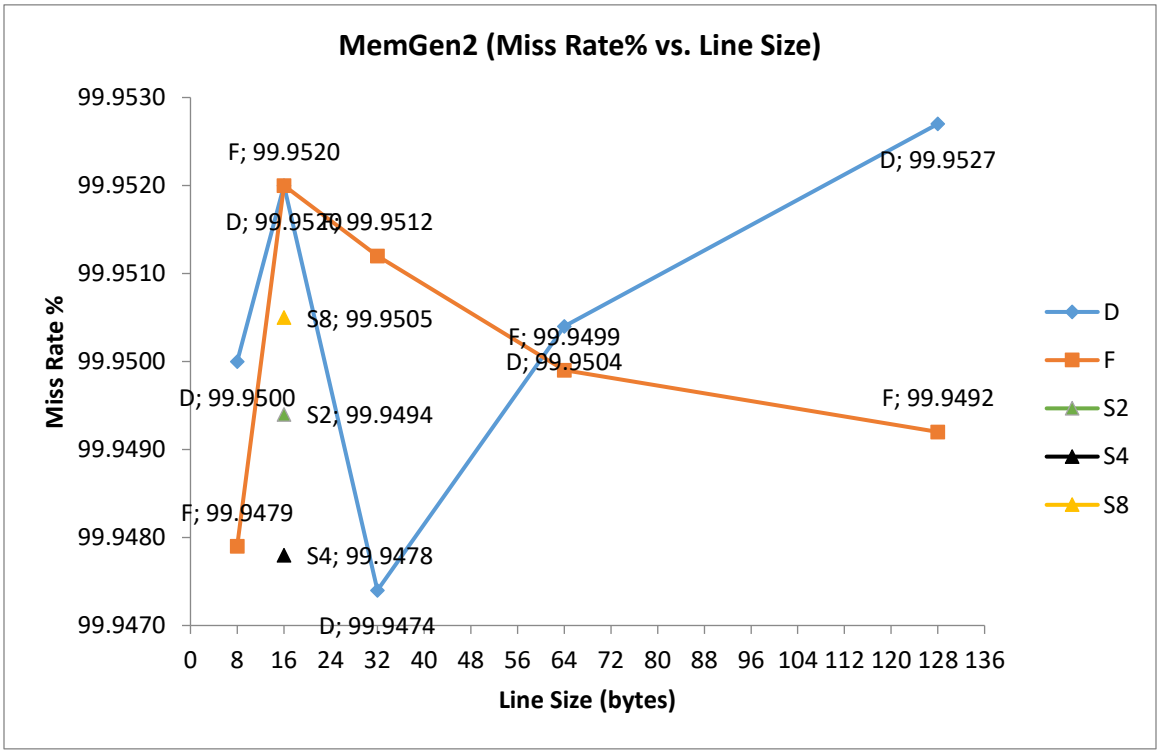
Address generation type 2

```
unsigned int memGen2()
{
        return rand_() % (DRAM_SIZE);
}
```

Algorithm Analysis:

This function generates random addresses between 0 and (DRAM size – 1). It does not loop and will depend on a random function generator to have equal probability covering the numbers between 0 and (DRAM size -1).

Obtained Results:



Observations:

- There is not considerable difference in the miss rate of any cache at any line size with any associativity. All are almost equal (difference is negligible in the 3rd decimal place).
- The miss rate is very high, almost 100%.

Explanation:

The behavior of the miss ratio is mainly due to the behavior of the address-generating function. It generates random addresses (in a very huge range) that can so rarely have the same tag within the capacity of the cache. Therefore, the miss ratio is nearly 100 %, due to the address-generating function.

Effect of increasing line size: There is no notable change in miss ratio with respect to varying line size.

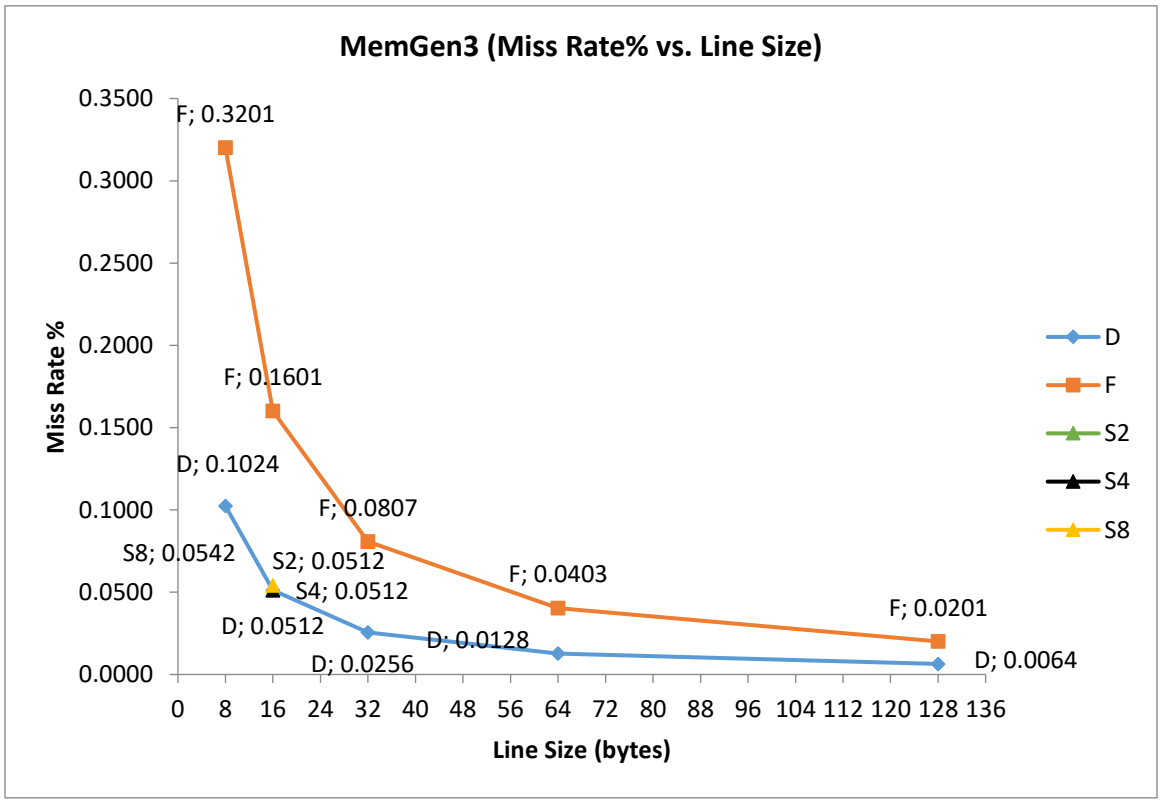
Address generation type 3

```
unsigned int memGen3()
{
        static unsigned int addr = 0;
        return (addr++) % (1024 * 8);
}
```

Algorithm analysis:

This function generates sequential addresses, 0, 1, 2, 3, 4... (8K – 1), and repeats the loop once there have been more than 8K executions (or 8K multiples).

Obtained Results:



Observations:

- FA > DM
- FA miss rates are between 3-4 times the miss rates of the DM at any line size.
- The miss rate of DM and FA are inversely proportional to the line size (as the line size increases the miss rate decreases).
- SA is almost equal to DM at line size 16.
- There is no considerable difference between SA2, SA4, or SA8.

Explanation:

Since the addresses generated by this function are within the capacity of the cache, miss ratio takes place due to compulsory (cold start). The data will be stored into the array once, won't be replaced, and will always return HIT after that cold start.

Effect of increasing line size: Increasing line size increases the data stored into the cache after one miss, hence, a decrease in occurrence of a miss because of **Spatial Locality**. This explains lower miss ratio.

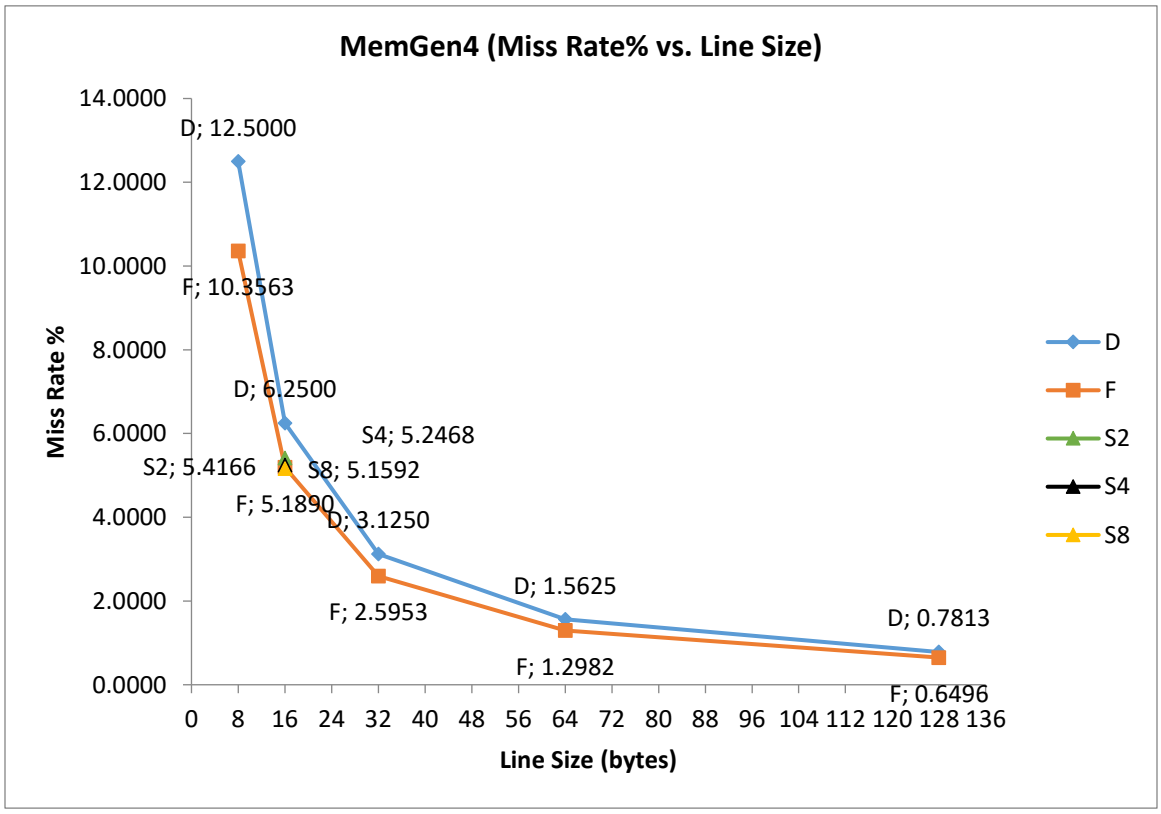
Address generation type 4

```
unsigned int memGen4()
{
        static unsigned int addr = 0;
        return (addr++) % (1024 * 64);
}
```

Algorithm analysis:

This function generates sequential addresses, 0, 1, 2, 3, 4... (64K – 1), and repeats the loop once there have been more than 64K executions (or 64K multiples).

Obtained Results:



Observations:

- DM > FA
- DM miss rates are between 1.2-1.3 times the miss rates of the FA at any line size.
- The miss rate of DM and FA are inversely proportional to the line size (as the line size increases the miss rate decreases).
- SA is almost equal to FA at line size 16.
- There is no considerable difference between SA2, SA4, or SA8.

Explanation:

Since the address generating function generates sequential addresses, the hit ratio will vary directly with line size. A sequence of addresses will be stored one into a line, will return MISS the first time, and will always return HIT for the next (n-1) times, and where "n" is the line size.

This explains that DM and FA have same miss rates because DM has higher miss rate due to competition for an index that is mapped by more than one address.

Effect of increasing line size: Increasing line size increases the data stored into the cache after one miss, hence, a decrease in occurrence of a miss because of **Spatial Locality**. This explains lower miss ratio.

Concluding Remarks

Increasing line size reduces miss ratio due to the advantage of **Spatial Locality**. However, larger blocks imply fewer of them, and hence increase competition between addresses, what might increase miss ratio. This behavior is not observable in the data collected because the cache size used (32KB) is relatively large, so there is a **less chance of conflict.**

Increasing associativity decreases conflict; it reduces misses that result from memory addresses competing for the same location; therefore it reduces the miss ratio. Since the cache size is relatively large, there is not so notable improvement in increasing associativity, beyond 2 “ways”.

While this project may focus on the miss rate the miss rate is not the sole decider on the cache performance. Other factors that affect the performance are the miss penalty, the replacement policy, hit time, and bandwidth of memory and cache.

For example, with larger lines sizes the miss penalty is larger because you need to transfer a larger block of data with each miss occurring. This factor was not considered in this project.