A Comparison of Line Sizes with CPI

Abdallah Ibrahim, Aly Youssef, Mahmoud Aly

1. Introduction

In this project, we explore the functionality of caches and examine how different will affect its performance.We developed a two-level, direct-mapped data cache simulator , it incorporates miss penalties ,write back alogirthm and can quantify overall performance using average CPI.The simulator uses 5 different memory access pattern functions MemGen1 to MemGen5.

2. MemGen Functions

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

memGen1 increments addresses by 1 generating the addresses sequentially until reaching the DRAM size then repeating as if scanning through an array
This generator behaviour:
First address is always 0 resulting in a miss (cold start).Then the cache fetches 16 bytes from memory (0-15).
The next address, address 1, is already in cache so it is a hit. This continues on for addresses 2 to 15.
The pattern then continues with 1 miss and 15 hits

```
unsigned int memGen2()
{
     static unsigned int addr = 0;
     return rand_() % (24 * 1024);
}
```

With small ranged memory access, memGen2 generates a random address within a small range from 0 to 24KB. This application doesn't rely on spatial locality as much as memGen1 as its access isn't sequential.

```
unsigned int memGen3() {return rand_() % DRAM_SIZE;}
```

Generates random address across the whole 64GB memory,this generally has a very high miss rate and very low chance receive constant hits, ignores heavily both temporal and spatial locality

```
unsigned int memGen4()
{
    static unsigned int addr = 0;
    return (addr++) % (4 * 1024);
}
```

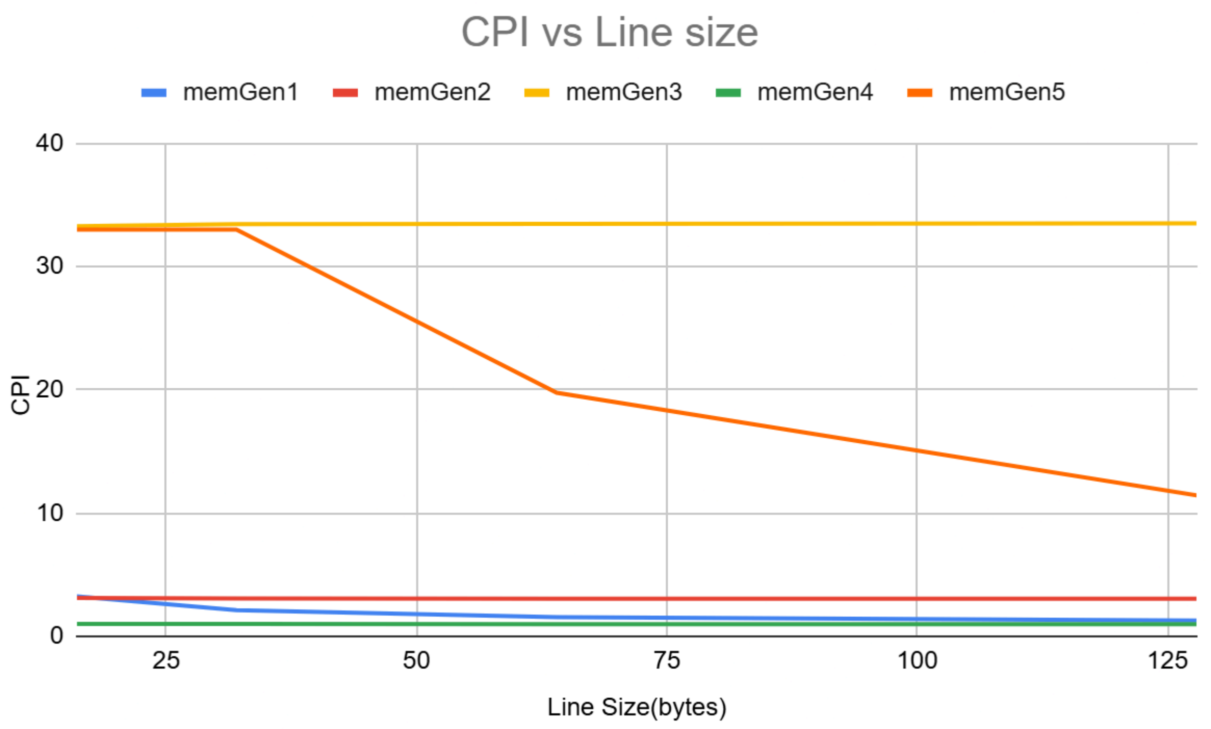
Sequential search incrementing every 4KBs works well for small memories, has a high hate rate and applies high spatial locality hit rates increases with larger line sizes decreasing miss rate.

```
unsigned int memGen5()
{
    static unsigned int addr = 0;
    return (addr += 32) % (64 * 16 * 1024);
}
```

This function is similar to memGen4 but differs as it iterates over a much larger subset of the DRAM (1MB instead of 4KB) and increments each cache access by 32 bytes which negates spatial locality when the line size is smaller than 32 bytes as all accesses would be a miss.

3. Results and findings

We found that the cycles needed for Load/Store instructions rapidly decreased once we started to increase the line size, falling by almost 50%. We also experimented with adding an additional L3 cache with a size of 1MB between the DRAM and L2 significantly increased the hit ratio especially in the memGen5 function.



4. Conclusion

Through our cache simulator we realised how important line size is to the system performance, particularly in reducing the CPI for memory intensive applications. From our results we learn that increasing the line size significantly reduces miss rates and write backs for both L1 and L2.

We also found that adding a third level of cache, L3, slightly improves performance especially in memGen5 where there are large memory ranges.