CS3210: Tutorial session 3

Elephant in memory

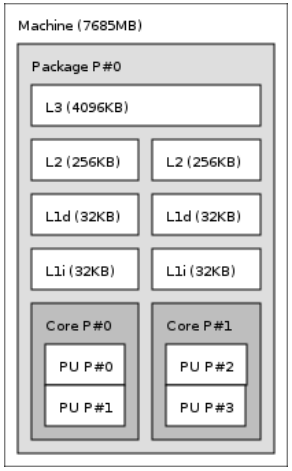
Overview

- Memory size and its latency
- Structure alignment
- Discussion on lab2 exercise 1

Memory size and its Latency

- Memory analogy (commonly found):
  - Desk and storage racks!
  - RAM -- storage bin
  - Caches -- files lying on your desk

Caches



- L1 : 3.0 ns
- L2 : 4.8 ns
- L3 : 9.5 ns
- RAM : 33.1 ns

Cache Associativity

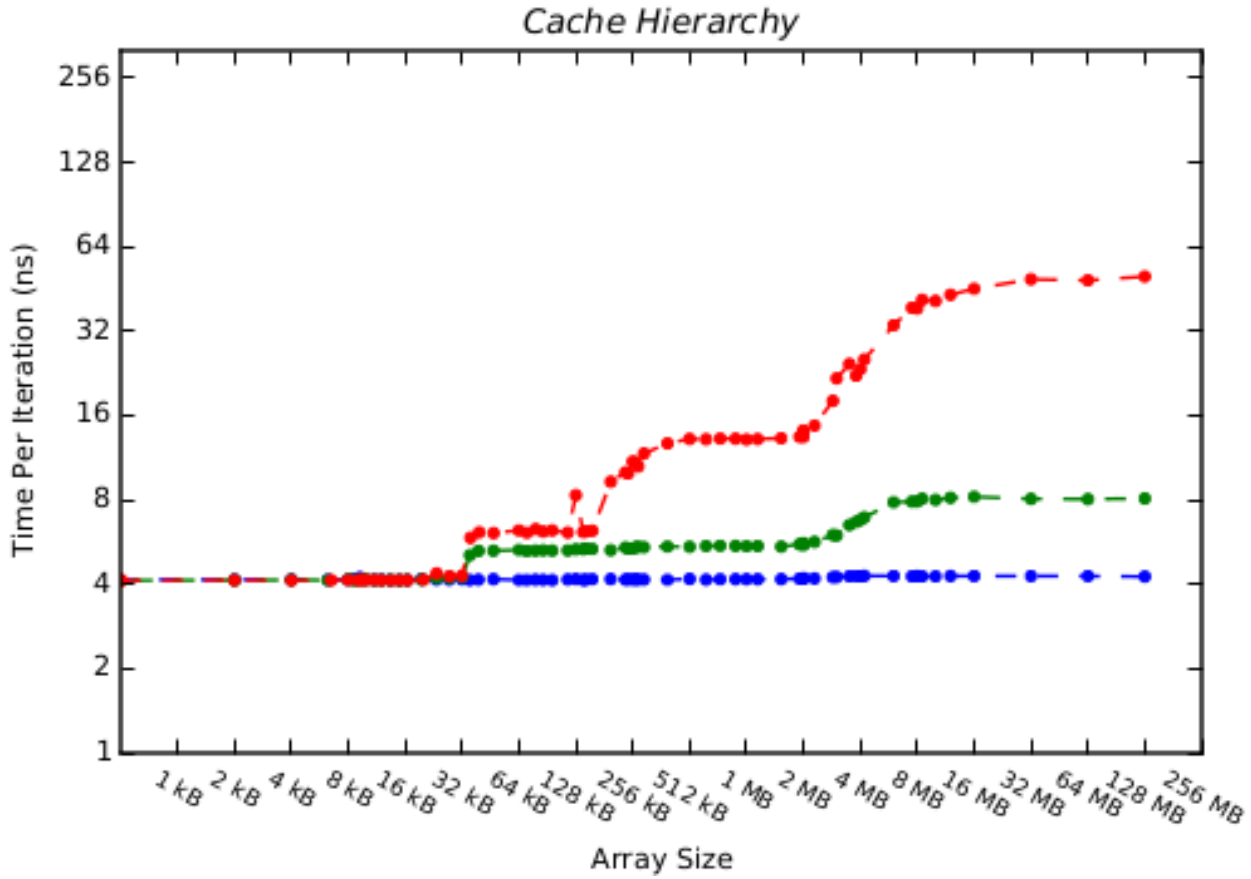
- Direct mapped
  - Only one block for each line
  - Easy to find, but difficult to put
- Fully associative
  - Can use whole cache
  - Easy to allocate, but difficult to find
- m-way set associative
  - m blocks in each set of the cache
  - Easier to allocate and find

Approximating Cache and acccess time

- Strided approach
  - Sequential acccess of large chunk of array
- Pointer chasing approach
  - Randomly accessing the array elements

Demo

My machine's cache and access time



Structures alignment

```
struct {
    char *p;
    char c;
};
```

- Expected size: 9 bytes
- Acutal size: 16 bytes??

Demo : struct alignment

Structures alignment

```
struct {
    char *p;
    char c;
};
```

- Expected size: 9 bytes
- Acutal size: 16 bytes??

- Makes memory access faster
- Fetching/storing the data via single instruction

Points to consider for structure alignment

- Generally, struct will have alignment of widest member
- Reorder members in decreasing alignment:
  - pointers / long (8 bytes)
  - int (4 bytes)
  - short (2 bytes)
  - char (1 byte)

Demo : struct alignment

Cacheline alignment

- Aligning structs on the cacheline boundary

```
#define L1D_CACHELINE_SIZE (64)
struct foo {
    /* elements */
} __attribute__ ((aligned (L1D_CACHELINE_SIZE)));
```

Discussion on lab2 exercise 1

- Let's play with the code!