Data-oriented Design in C++

Melbourne C++ Meetup - February

What is Data-oriented Design?

- Design paradigm to organise data in a way that requires minimal work to process
- Close attention to hardware architecture   i.e. CPU cache, memory allocation, multithreading, etc..
- Not exclusive from other paradigms such as OOP

Why Data-oriented Design?

- Fixing performance issues is increasingly difficult as codebase grows
- Better to make the “right” decisions from the beginning
- Different to premature optimisation

Intelligently balance performance with other design factors

Agenda

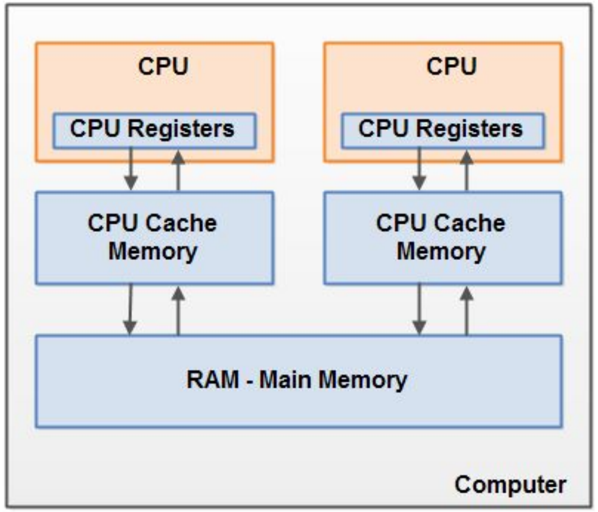
→ Utilising CPU cache

→ Memory Alignment

→ Cache in multithreading

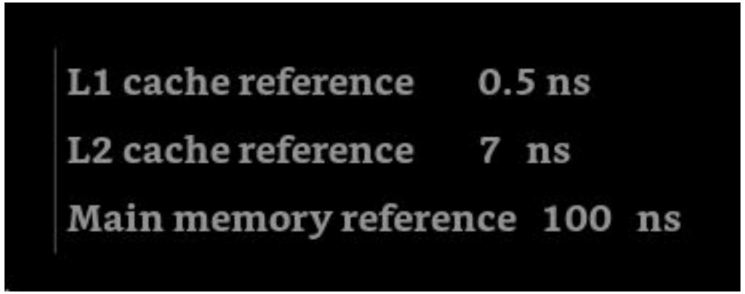
CPU Cache Architecture

- The CPU stores data from the main memory in “cache memory”
- CPU cache memory is divided into L1, L2 and L3
- CPU fetches from the main memory in units known as “cache line”
- Cache line is 64 bytes in modern desktop CPUs



Utilising CPU Cache

- Fetching from main memory to CPU is very expensive
- Minimise the round trip between CPU cache and main memory



L1 cache reference	0.5 ns
L2 cache reference	7 ns
Main memory reference	100 ns

Extract from an article by Peter Norvig, Teach yourself programming in ten years, 2001

Demo

What happened?

```
int i = 0; i += 1;
```

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Cache Miss																*Cache Miss*

```
int i = 0; i += 16;
```

0	16	32	48	64	80	96	112	128	144	160	176	192	208	224	240	256
Cache Miss	*Cache Miss*	*Cache Miss*	*Cache Miss*	*Cache Miss*	*Cache Miss*	*Cache Miss*	*Cache Miss*	*Cache Miss*	*Cache Miss*	*Cache Miss*	*Cache Miss*	*Cache Miss*	*Cache Miss*	*Cache Miss*	*Cache Miss*	*Cache Miss*

Demo

Cache - Summary

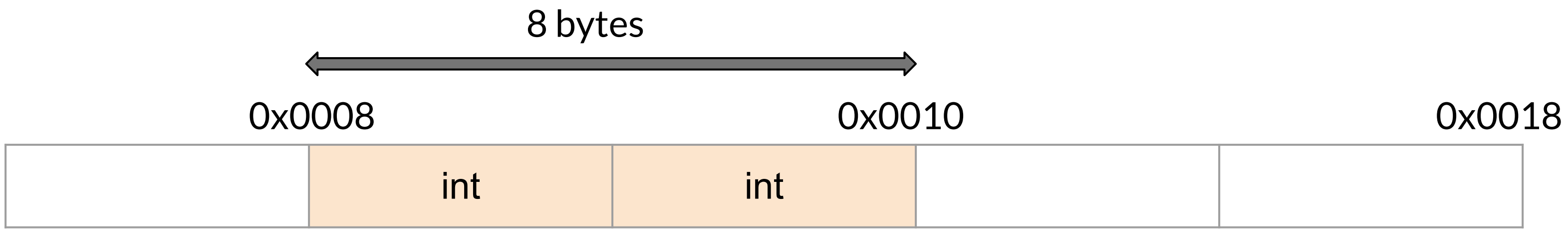
- Access contiguous memory wherever possible
    - Make data fit in cache line as much as possible.
- Chasing random pointers is expensive
    - Improve spatial locality by allocating memory in chunks

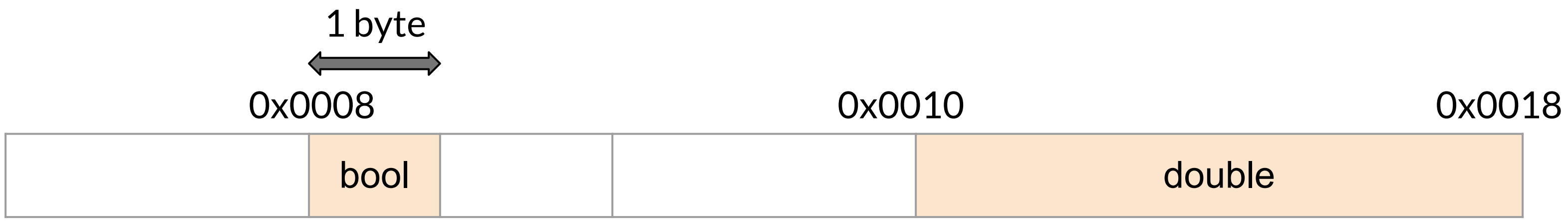
Memory Alignment

- At the lowest level, the CPU uses “registers” to store data for processing
- When processing data, the CPU reads into registers one word at a time (64 bits for x64 architecture = 8 bytes)
- CPU has a requirement on memory addresses of objects to be a multiple of a number in order to map the data to its registers.

Memory Alignment

Examples:

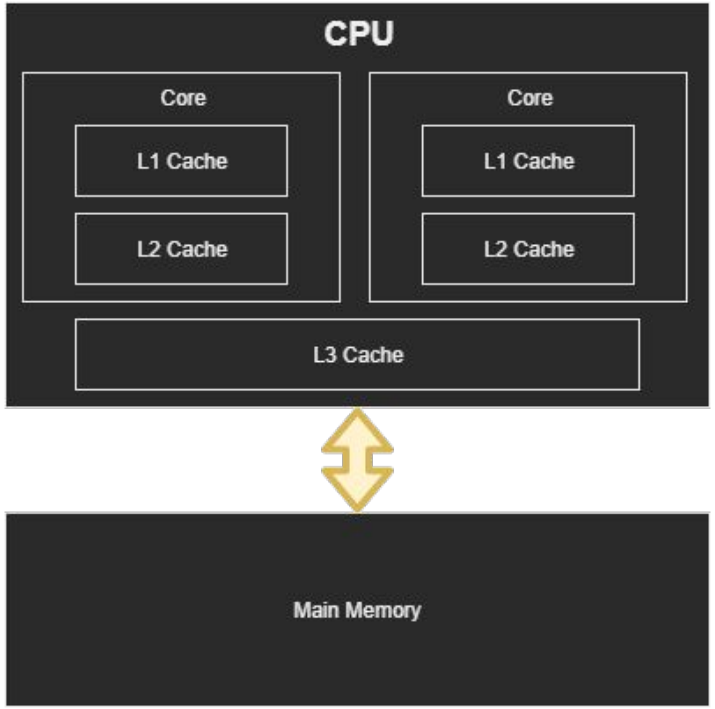




Demo

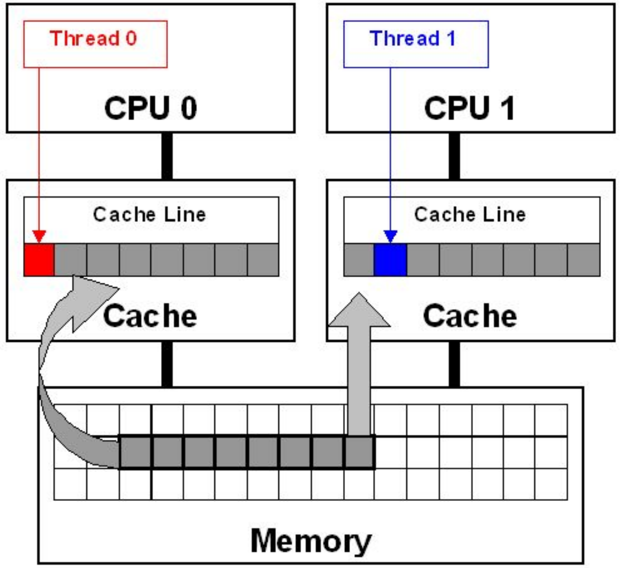
Cache in multithreading

- In multithreading, each physical core has its own L1 cache and cache line.
- Multiple threads may access some data in memory which happen to be on the same cache line



Cache in multithreading

- If one thread updates the data, this invalidates the cache line in the other thread and forces a memory update to maintain cache coherency.
- This occurrence is called ‘false sharing’



Demo

Summary

- Organising data for cache is crucial to good performance
- Does not necessarily require elaborate hacking / design
- Not restricted to particular design patterns

Further readings

- Mike Action’s code review on Ogre3D - revealing OOP inefficiencies https://www.bounceapp.com/116414
- A Data Oriented Approach to Using Component Systems - https://www.youtube.com/watch?v=p65Yt20pw0g
- Object pool pattern - allocate memory in chunks instead of per object - http://gameprogrammingpatterns.com/object-pool.html

Questions?