COMP350: Algorithms & Optimisation

3: Optimising for CPU & Memory

Learning outcomes

By the end of today's session, you will be able to:

- **Understand** the memory hierarchy in modern PC/Consoles/Mobile
- **Implement** optimisations of Data Structures for memory access
- **Describe** CPU Optimisation

FALMOUTH
UNIVERSITY

Introduction

- When we start the optimisation process in games, we want to check if we are CPU or GPU bound

- When we start the optimisation process in games, we want to check if we are CPU or GPU bound
- Once this is identified then we can start looking at different ways of optimising our code

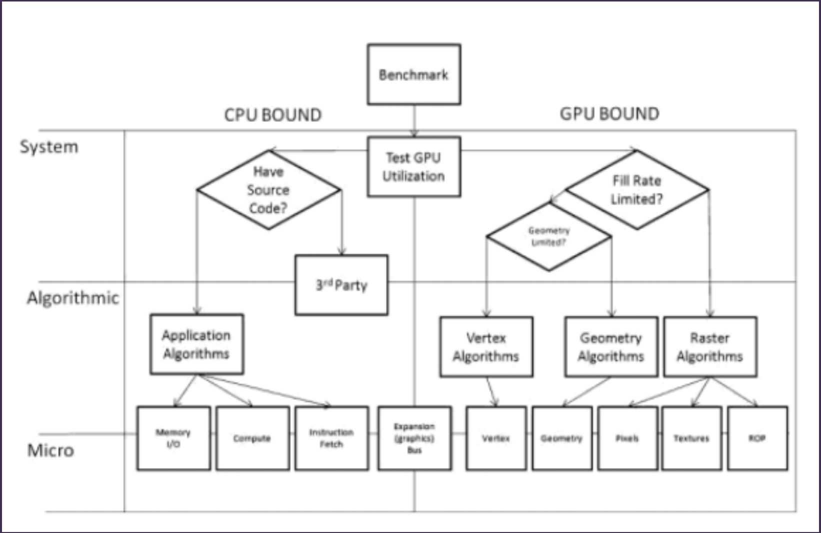
- When we start the optimisation process in games, we want to check if we are CPU or GPU bound
- Once this is identified then we can start looking at different ways of optimising our code
    - System Level

- When we start the optimisation process in games, we want to check if we are CPU or GPU bound
- Once this is identified then we can start looking at different ways of optimising our code
  - System Level
  - Algorithm

- When we start the optimisation process in games, we want to check if we are CPU or GPU bound
- Once this is identified then we can start looking at different ways of optimising our code
  - System Level
  - Algorithm
  - Micro

- When we start the optimisation process in games, we want to check if we are CPU or GPU bound
- Once this is identified then we can start looking at different ways of optimising our code
  - System Level
  - Algorithm
  - Micro
- In this session we are going to assume our application is CPU bound

Optimisation Flow



FALMOUTH
UNIVERSITY

Algorithms

Standard Template Library

Standard Template Library

- STL is a collection of containers and algorithms to support the creation of more reusable systems

Standard Template Library

- STL is a collection of containers and algorithms to support the creation of more reusable systems
- Containers:

Standard Template Library

- STL is a collection of containers and algorithms to support the creation of more reusable systems
- Containers:
  - Vector: Drop in replacement for a C/C++ array, elements are stored **contiguously**

Standard Template Library

- STL is a collection of containers and algorithms to support the creation of more reusable systems
- Containers:
  - Vector: Drop in replacement for a C/C++ array, elements are stored **contiguously**
  - List: Supports constant time insertion and removal of elements

Standard Template Library

- STL is a collection of containers and algorithms to support the creation of more reusable systems
- Containers:
  - Vector: Drop in replacement for a C/C++ array, elements are stored **contiguously**
  - List: Supports constant time insertion and removal of elements
  - Map: Search, removal and insertion have logarithmic complexity

Standard Template Library

- STL is a collection of containers and algorithms to support the creation of more reusable systems
- Containers:
    - Vector: Drop in replacement for a C/C++ array, elements are stored **contiguously**
    - List: Supports constant time insertion and removal of elements
    - Map: Search, removal and insertion have logarithmic complexity
- Algorithm:

Standard Template Library

- STL is a collection of containers and algorithms to support the creation of more reusable systems
- Containers:
  - Vector: Drop in replacement for a C/C++ array, elements are stored **contiguously**
  - List: Supports constant time insertion and removal of elements
  - Map: Search, removal and insertion have logarithmic complexity
- Algorithm:
  - Find: Find an element in a container

Standard Template Library

- STL is a collection of containers and algorithms to support the creation of more reusable systems
- Containers:
  - Vector: Drop in replacement for a C/C++ array, elements are stored **contiguously**
  - List: Supports constant time insertion and removal of elements
  - Map: Search, removal and insertion have logarithmic complexity
- Algorithm:
  - Find: Find an element in a container
  - Transform: Apply a function to an element

Standard Template Library

- STL is a collection of containers and algorithms to support the creation of more reusable systems
- Containers:
  - Vector: Drop in replacement for a C/C++ array, elements are stored **contiguously**
  - List: Supports constant time insertion and removal of elements
  - Map: Search, removal and insertion have logarithmic complexity
- Algorithm:
  - Find: Find an element in a container
  - Transform: Apply a function to an element
  - Swap: Swaps the values of two objects

Standard Template Library

- STL is a collection of containers and algorithms to support the creation of more reusable systems
- Containers:
  - Vector: Drop in replacement for a C/C++ array, elements are stored **contiguously**
  - List: Supports constant time insertion and removal of elements
  - Map: Search, removal and insertion have logarithmic complexity
- Algorithm:
  - Find: Find an element in a container
  - Transform: Apply a function to an element
  - Swap: Swaps the values of two objects
  - Reverse: Reverse the elements in a container

Common Container Errors

FALMOUTH UNIVERSITY

Common Container Errors

- Picking the wrong container!

Common Container Errors

- Picking the wrong container!
  - If you are adding/removing elements from the middle of collection then use a list

Common Container Errors

- Picking the wrong container!
  - If you are adding/removing elements from the middle of collection then use a list
  - Require random access, use a vector over a list

FALMOUTH UNIVERSITY

Common Container Errors

- Picking the wrong container!
    - If you are adding/removing elements from the middle of collection then use a list
    - Require random access, use a vector over a list
    - If you want to retrieve an element quickly, consider using a **map** or a **set**

Common Container Errors

- Picking the wrong container!
  - If you are adding/removing elements from the middle of collection then use a list
  - Require random access, use a vector over a list
  - If you want to retrieve an element quickly, consider using a **map** or a **set**
  - Sorted vectors can give same performance as associative containers for retrieval

Common Container Errors

- Picking the wrong container!
  - If you are adding/removing elements from the middle of collection then use a list
  - Require random access, use a vector over a list
  - If you want to retrieve an element quickly, consider using a **map** or a **set**
  - Sorted vectors can give same performance as associative containers for retrieval
  - `http://john-ahlgren.blogspot.co.uk/2013/10/stl-container-performance.html`

Common Container Errors

Common Container Errors

- Resizing of vector

Common Container Errors

- Resizing of vector
    - Resizing of vector requires a memory allocation

FALMOUTH UNIVERSITY

Common Container Errors

- Resizing of vector
  - Resizing of vector requires a memory allocation
  - If you know the amount of data you are managing, you should use the **reserve** function to allocate the memory in advance

Algorithms

Algorithms

- Before you write something you should consider checking out the algorithms contained in the **algorithms** library

Algorithms

- Before you write something you should consider checking out the algorithms contained in the **algorithms** library
- These contain all sorts of useful functions for searching, sorting, counting, manipulating

Algorithms

- Before you write something you should consider checking out the algorithms contained in the **algorithms** library
- These contain all sorts of useful functions for searching, sorting, counting, manipulating
- These are usually optimised for the compiler, so use these rather than write your own

Algorithms

- Before you write something you should consider checking out the algorithms contained in the **algorithms** library
- These contain all sorts of useful functions for searching, sorting, counting, manipulating
- These are usually optimised for the compiler, so use these rather than write your own
- You can also customise how these algorithms function by providing predicate functions

What about Unity or Unreal?

What about Unity or Unreal?

- Both Unity/C# & Unreal have containers

What about Unity or Unreal?

- Both Unity/C# & Unreal have containers
- List in C# & TArray in Unreal is equivalent to std::vector

What about Unity or Unreal?

- Both Unity/C# & Unreal have containers
- List in C# & TArray in Unreal is equivalent to std::vector
- Dictionary in C# & TMap in Unreal is equivalent to std::map

What about Unity or Unreal?

- Both Unity/C# & Unreal have containers
- List in C# & TArray in Unreal is equivalent to std::vector
- Dictionary in C# & TMap in Unreal is equivalent to std::map
- The containers in those engines should perform exactly like the C++ equivalents

What about Unity or Unreal?

- Both Unity/C# & Unreal have containers
- List in C# & TArray in Unreal is equivalent to std::vector
- Dictionary in C# & TMap in Unreal is equivalent to std::map
- The containers in those engines should perform exactly like the C++ equivalents
- **BUT!** check the docs as these engines/languages have additional container types

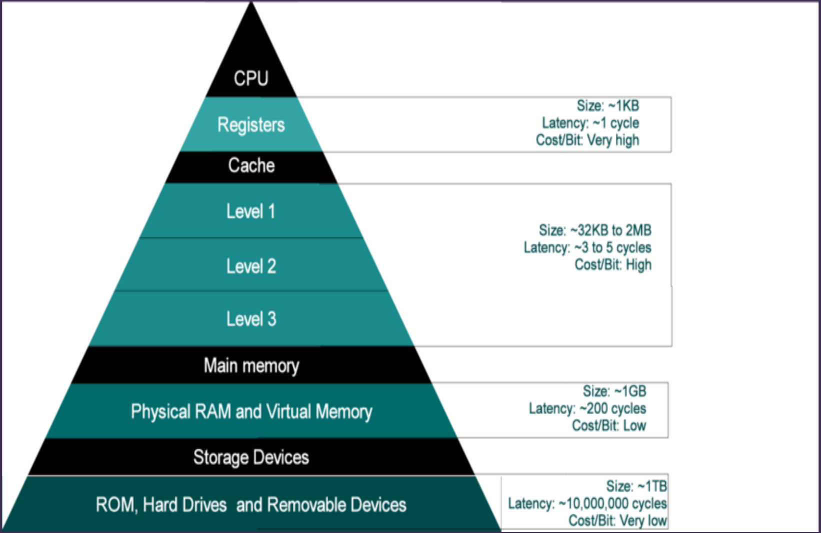
FALMOUTH
UNIVERSITY

Coffee Break

Memory & CPU

FALMOUTH UNIVERSITY

Memory Hierarchy



Registers

Registers

- Registers are the only memory directly on the CPU

Registers

- Registers are the only memory directly on the CPU
- These tend to act as quick scratch pad for current calculations

Registers

- Registers are the only memory directly on the CPU
- These tend to act as quick scratch pad for current calculations
- Everything that the CPU operate on will get shifted into registers

Registers

- Registers are the only memory directly on the CPU
- These tend to act as quick scratch pad for current calculations
- Everything that the CPU operate on will get shifted into registers
- Only issue it is slow to transfer data from RAM to registers

FALMOUTH UNIVERSITY

Registers

- Registers are the only memory directly on the CPU
- These tend to act as quick scratch pad for current calculations
- Everything that the CPU operate on will get shifted into registers
- Only issue it is slow to transfer data from RAM to registers
- This is where the Cache comes in!

Caches

Caches

- The cache acts as intermediate between the CPU and all off-board memory

Caches

- The cache acts as intermediate between the CPU and all off-board memory
- There is usually different levels of cache, with different capabilities dependent on CPU e.g

FALMOUTH UNIVERSITY

Caches

- The cache acts as intermediate between the CPU and all off-board memory
- There is usually different levels of cache, with different capabilities dependent on CPU e.g
- Intel i-7 4770(Haswell)

FALMOUTH UNIVERSITY

Caches

- The cache acts as intermediate between the CPU and all off-board memory
- There is usually different levels of cache, with different capabilities dependent on CPU e.g
- Intel i-7 4770(Haswell)
  - L1 Data cache = 32KB with 4 cycles for simple access

Caches

- The cache acts as intermediate between the CPU and all off-board memory
- There is usually different levels of cache, with different capabilities dependent on CPU e.g
- Intel i-7 4770(Haswell)
  - L1 Data cache = 32KB with 4 cycles for simple access
  - L1 instruction cache = 32KB

Caches

- The cache acts as intermediate between the CPU and all off-board memory
- There is usually different levels of cache, with different capabilities dependent on CPU e.g
- Intel i-7 4770(Haswell)
  - L1 Data cache = 32KB with 4 cycles for simple access
  - L1 instruction cache = 32KB
  - L2 cache = 256KB with 12 cycles

Caches

- The cache acts as intermediate between the CPU and all off-board memory
- There is usually different levels of cache, with different capabilities dependent on CPU e.g
- Intel i-7 4770(Haswell)
  - L1 Data cache = 32KB with 4 cycles for simple access
  - L1 instruction cache = 32KB
  - L2 cache = 256KB with 12 cycles
  - L3 cache = 8MB with 26 cycles

Caches

- The cache acts as intermediate between the CPU and all off-board memory
- There is usually different levels of cache, with different capabilities dependent on CPU e.g
- Intel i-7 4770(Haswell)
  - L1 Data cache = 32KB with 4 cycles for simple access
  - L1 instruction cache = 32KB
  - L2 cache = 256KB with 12 cycles
  - L3 cache = 8MB with 26 cycles
- NB Ram typically takes 26 cycles + 57 nanoseconds to access

Memory Optimisation

FALMOUTH
UNIVERSITY

Memory Optimisation

- Reduce Memory footprint

Memory Optimisation

- Reduce Memory footprint
- Write algorithms which reduce memory traversal

Memory Optimisation

- Reduce Memory footprint
- Write algorithms which reduce memory traversal
- Increase cache hits

Memory Optimisation

- Reduce Memory footprint
- Write algorithms which reduce memory traversal
- Increase cache hits
- Increase temporal coherence

Memory Optimisation

- Reduce Memory footprint
- Write algorithms which reduce memory traversal
- Increase cache hits
- Increase temporal coherence
- Utilise Pre-fetching

Memory Optimisation

- Reduce Memory footprint
- Write algorithms which reduce memory traversal
- Increase cache hits
- Increase temporal coherence
- Utilise Pre-fetching
- Avoid patterns which break caching

Reduce Memory Footprint

Reduce Memory Footprint

- Consider use smaller data types

Reduce Memory Footprint

- Consider use smaller data types
- This will mean that your data structures will be more cache friendly

Reduce Memory Footprint

- Consider use smaller data types
- This will mean that your data structures will be more cache friendly
- You can perhaps collapse several booleans into one integer and use bit flags

Memory Alignment

Memory Alignment

- The CPU will read memory aligned data much faster than non-aligned

Memory Alignment

- The CPU will read memory aligned data much faster than non-aligned
- Any *N*-byte data is memory aligned if its starting address is evenly divisible by *N*

Memory Alignment

- The CPU will read memory aligned data much faster than non-aligned
- Any *N*-byte data is memory aligned if its starting address is evenly divisible by *N*
- e.g. A 32 bit integer is memory aligned if the starting address is 0x04000000, not 0x04000002

FALMOUTH UNIVERSITY

Memory Alignment

- The CPU will read memory aligned data much faster than non-aligned
- Any *N*-byte data is memory aligned if its starting address is evenly divisible by *N*
- e.g. A 32 bit integer is memory aligned if the starting address is 0x04000000, not 0x04000002
- Make sure your structs or class has data types order highest to lowest

Avoiding Cache Misses

Avoiding Cache Misses

- The compiler and linker dictate how your code is laid out in memory

Avoiding Cache Misses

- The compiler and linker dictate how your code is laid out in memory
- However the linker/compiler follows some simple rules

Avoiding Cache Misses

- The compiler and linker dictate how your code is laid out in memory
- However the linker/compiler follows some simple rules
- If you know these, then you can leverage them

Compiler & Linker rules

FALMOUTH
UNIVERSITY

Compiler & Linker rules

- The machine code for a single function is contiguous in memory

Compiler & Linker rules

- The machine code for a single function is contiguous in memory
- Functions are laid out in memory in the order they appear in the cpp file

Compiler & Linker rules

- The machine code for a single function is contiguous in memory
- Functions are laid out in memory in the order they appear in the cpp file
- Function in the cpp are always contigous

Applying Compiler & Linker rules

Applying Compiler & Linker rules

- Keep high performance code as small as possible

Applying Compiler & Linker rules

- Keep high performance code as small as possible
- Avoid calling functions from a performance critical section of code

Applying Compiler & Linker rules

- Keep high performance code as small as possible
- Avoid calling functions from a performance critical section of code
- If you do have to call a function, place it as close as possible (never in another translation unit)

Applying Compiler & Linker rules

- Keep high performance code as small as possible
- Avoid calling functions from a performance critical section of code
- If you do have to call a function, place it as close as possible (never in another translation unit)
- Use inline functions. Inlining a small function can lead to a performance boost. But this can lead to bloated code if over used (and lead to cache misses)

Branch Prediction

Branch Prediction

- CPUs will try to predict what branch to take (if loop)

Branch Prediction

- CPUs will try to predict what branch to take (if loop)
- If the guess is wrong then we executed instructions that shouldn't have been called

Branch Prediction

- CPUs will try to predict what branch to take (if loop)
- If the guess is wrong then we executed instructions that shouldn't have been called
- This causes a stall, as the pipeline is flushed and the first instruction of the branch is then called

Branch Prediction

- CPUs will try to predict what branch to take (if loop)
- If the guess is wrong then we executed instructions that shouldn't have been called
- This causes a stall, as the pipeline is flushed and the first instruction of the branch is then called
- To solve this issue, you should attempt to reduce or remove all branches (see loop unrolling &)

FALMOUTH UNIVERSITY

Exercise

Research two of following areas

- Please start of your research journal!
    1. Loop unrolling
    2. Avoiding loops
    3. Memory alignment
    4. Array of Structures vs Structure of Arrays
    5. Memory Pools
    6. Statically allocated memory
    7. STL Algorithms & Containers

Further Reading

- Intel Optimisation Manual - `https://software.intel.com/sites/default/files/managed/9e/bc/64-ia-32-architectures-optimization-manual.pdf`
- Effective STL - `http://voyager.falmouth.ac.uk/vwebv/holdingsInfo?bibId=666539`
- Game Coding Complete 4th Ed Chapter 3 - `http://voyager.falmouth.ac.uk/vwebv/holdingsInfo?bibId=755157`
- Game Engine Architecture 2nd Ed Chapter 3 - `http://voyager.falmouth.ac.uk/vwebv/holdingsInfo?bibId=1084476`