Parallel Computing with GPUs

Optimisation
Part 1 – Overview

The University Of Sheffield.

Dr Paul Richmond

http://paulrichmond.shef.ac.uk/teaching/COM4521/

This Lecture (learning objectives)

- Optimisation Overview
  - Recognise when it is appropriate to optimise a program
  - Identify the key differences between benchmarking and profiling
  - Explain the use of visual studio profiling
  - Classify code as compute or memory bound

When to Optimise

- Is your program complete?
  - If not then don’t start optimising
  - If you haven't started coding then don’t try to perform advanced optimisations until its complete
    - This might be counter intuitive
- Is it worth it?
  - Is your code already fast enough?
  - Are you going to optimise the right bit?
  - What are the likely benefits? Is it cost effective?
    - (number of runs × number of users × time savings × user's salary) - (time spent optimizing × programmer's salary)

*“Programmers waste enormous amounts of time thinking about, or worrying about, th parts of their programs, and these attempts at efficiency actually have a strong negat debugging and maintenance are considered. We should forget about small efficiencie the time: **premature optimization** is the root of all evil. Yet we should not pass up ou critical 3%.”* Donald Knuth, Computer Programming as an Art (1974)

First step: Profiling

- Which part of the program is the bottleneck
  - This may be obvious if you have a large loop
  - May be less obvious in a complicated program or procedure
- Manually benchmark/profile using `time()` function
  - We can time critical aspects of the program using the time command
  - This gives us insight into how long it takes to execute.
- Profiling using a profiler
  - Unix: gprof
  - Visual Studio: Built in profiler

Benchmarking with clock() – Windows only

- `#include <time.h>`
- The `clock()` function returns a `clock_t` value the number of clock ticks elapsed since the program was launched
- To calculate the time in seconds divide by `CLOCK_PER_SEC`

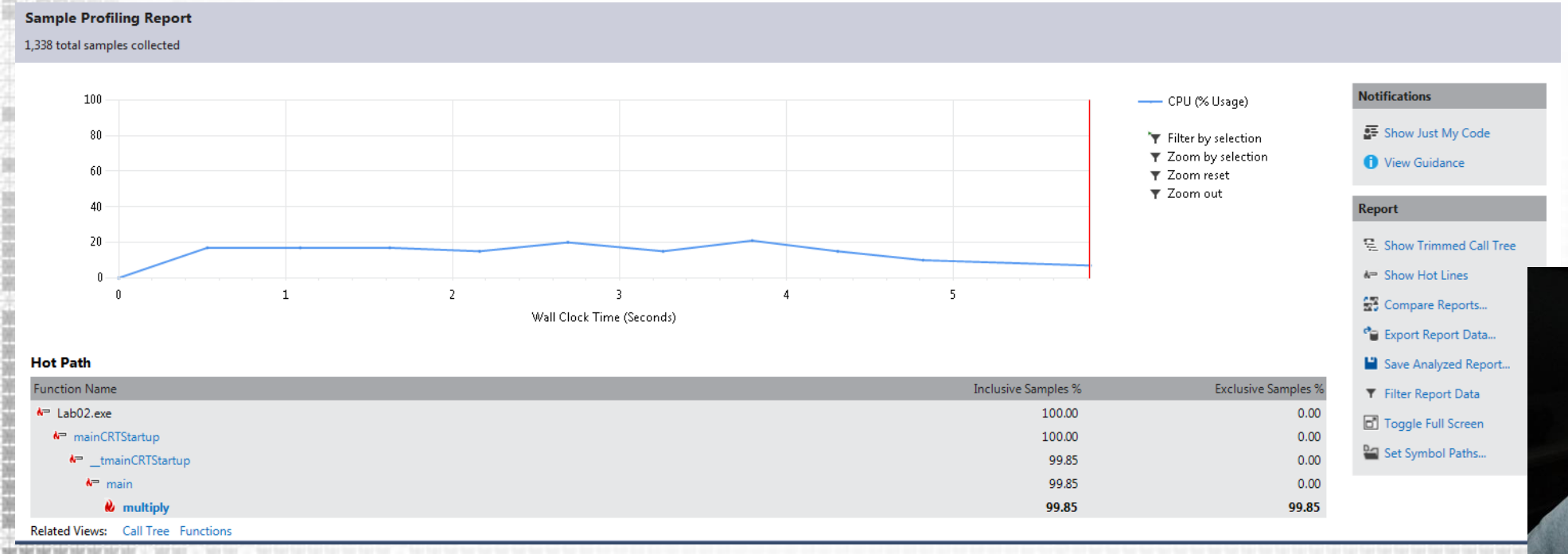
```
clock_t begin, end;
float seconds;

begin = clock();
func();
end = clock();

seconds = (end - begin) / (float)CLOCKS_PER_SEC;
```

Visual Studio Profiling Example

- Debug->Performance and Diagnostics
  - Start
  - Select CPU Sampling, Finish (or next and select project)
  - No Data? Your program might not run for long enough to sample



Visual Studio Profiling Example

- Samples
  - The profiler interrupts at given time intervals to collect information on the stack
  - Default sampling is 10,000,000 clock cycles
- Inclusive Samples
  - Time samples including any sub call
- Exclusive Samples
  - Time samples excluding any sub calls
- Hot Path
  - Slowest path of execution through the program
    - **Best candidate for optimisation**
  - Select the function for a line-by-line breakdown of sampling percentage

Function Code View

E:\Google Drive\com4521_6521 - parallel computing with gpus\TEACHING\LABS\Lab02\memory.c

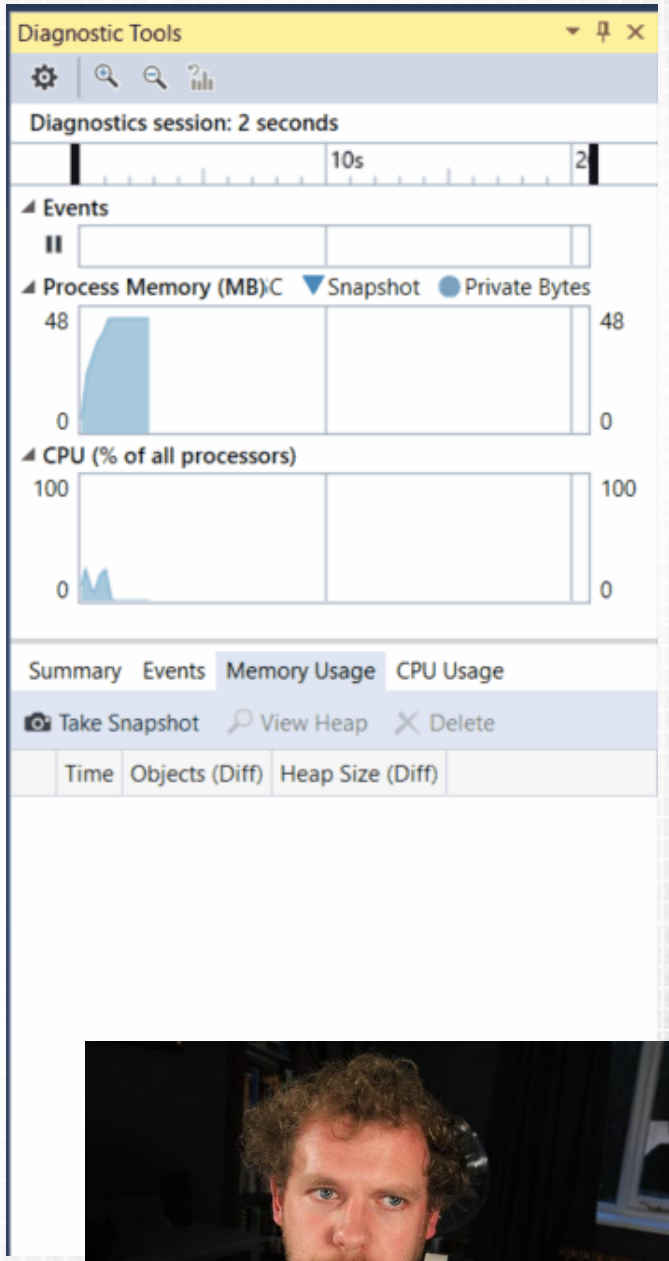
```
                void multiply(matrixNN r, matrixNN a, matrixNN b){


                    int i, j, k;
                    for (i = 0; i < N; i++){
                        for (j = 0; j < N; j++){
  0.2 %                     r[i][j] = 0;
  7.5 %                     for (k = 0; k < N; k++){
 88.0 %                         r[i][j] += a[i][k] * b[k][j];
  3.9 %                     }
                        }
                    }


                }
```

Compute vs Memory Bound

- Compute bound
  - Performance is limited by the speed of the CPU
  - CPU usage is high: typically 100% for extended periods of time
- Memory Bound
  - Performance is limited by the memory access speed
  - CPU usage might be lower
  - Typically the cache usage will be poor
    - poor hit rate if fragmented or random accesses



Summary

- Optimisation Overview
  - Recognise when it is appropriate to optimise a program
  - Identify the key differences between benchmarking and profiling
  - Explain the use of visual studio profiling
  - Classify code as compute or memory bound