ENGSCI 233 - Performance
Lecture 2

Colin Simpson

Semester One, 2022

Department of Engineering Science
University of Auckland
(Beamer Theme: Metropolis)

Introduction

In this lecture we will consider:

- Profiling of large programs.
- Parallel implementation of algorithms.

Profiling for Optimisation

What is a Profiler?

In the previous lecture, we discussed concepts around the performance of algorithms.

In a large or complicated computer program, there may be many different algorithms or functions at work. It is not always practical to manually determine the performance of each one.

Instead, we can use a *profiler* to automate the process of measuring the performance of, and the number of calls to, each function. This data can be used to identify any bottlenecks and subsequently optimise our code.

Profiler

There are two profilers included in the Python standard library:

- `cProfile` - officially recommended.
- `profile`

In the lab you will be using `cProfile` to profile some code. I also demonstrated its basic usage in the lab preparation video.

Example of `cProfile` Output

Example output of `cProfile` from the Jupyter notebook:

```
Mon Feb 18 19:00:58 2019    restats

         2985 function calls in 8.654 seconds

   Ordered by: internal time
   List reduced from 20 to 5 due to restriction <5>

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
      199    8.644    0.043    8.644    0.043 d:\teaching\233_labs\Lab5\lab5_functions_complete.py:187(row_reduction)
        1    0.005    0.005    8.654    8.654 d:\teaching\233_labs\Lab5\lab5_functions_complete.py:81(lu_factor)
      398    0.001    0.000    0.001    0.000 {built-in method builtins.abs}
      199    0.001    0.000    0.001    0.000 {method 'argmax' of 'numpy.ndarray' objects}
      196    0.001    0.000    0.001    0.000 C:\Users\ddem014\AppData\Local\Continuum\anaconda3\lib\copy.py:66(copy)
```

This information can point you to bottlenecks in your code. *What is the main bottleneck in the above example?*

Concurrency and Parallelisation

Moore's Law (Observation)

Gordon Moore predicted in 1965 a doubling of the number of transistors in an integrated circuit approximately every two years.

Transistor sizes can now reach about $5 - 10$ nm, but further minimisation is becoming increasingly difficult as we approach physical limits associated with the atomic size of silicon.

Have we now reached peak computational power?

Modern Improvements

Improvements in performance have recently come from:

- Multi-CPU
- CPU with Hyper-Threading
- Multi-core CPU
- Multi-core CPU with Hyper-Threading
- Multi-CPU with multi-core CPUs (with Hyper-Threading?)

Personal computers now tend to have 2, 4, 8 CPUs or more.

Supercomputers typically have racks upon racks of motherboards that each contain multiple CPUs, each with multiple cores.

Concurrency and Parallelisation

Parallel computing utilises the idea of concurrency.

Consider finding the LU factorisation of ten different matrices. We can consider one process to be the factorisation of one matrix:

- One core: take and complete each process, one at a time. Often referred to as serial execution.
- Multiple cores: form a queue of ten processes. An idle core will take the next process from the queue and complete it. Repeat until queue is empty. Known as parallel execution.

Parallel Computing in Python

Parallel processing in Python can be achieved fairly easily with the module `multiprocessing` to run independent parallel processes:

- The `Pool` creates a pool of CPUs.
- The `map` or `starmap` method distributes out independent processes (e.g. function calls) to the pool of CPUs.

I worked through a basic example in the lab preparation video.

Limitations of Parallelisation

Several factors will impact the speedup from parallelisation:

- The number of simultaneous processes is limited to the number of available CPUs.
- Overheads from starting and ending processes. Especially problematic when processes are not fully independent.
- Bottlenecks e.g. one core working on a process while the queue is empty and the other cores are idle.

Quantifying Parallelisation

Quantifying Parallel Speedup

The parallel speedup for n_p cores, S_p, can be quantified as:

$$S_p = \frac{T_s}{T_p}$$

where T_p is the execution time and T_s is the serial execution time. $S_p > 1$ represents an improvement over the serial execution time.

Quantifying Parallel Efficiency

The parallel efficiency for n_p cores, E_p, can be quantified as:

$$E_p = \frac{S_p}{n_p} = \frac{T_s}{T_p\, n_p}$$

There are several broad categories of parallel efficiency:

- $E_p > 1$ i.e. superlinear.
- $E_p = 1$ i.e. linear.
- $E_p < 1$ i.e. sublinear.

Example - Parallel Speedup and Efficiency

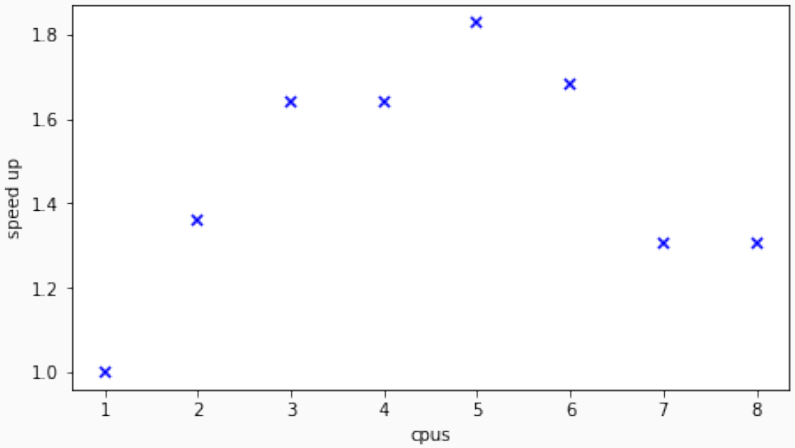
Consider the following example of factorising ten different matrices, using an increasing number of cores:

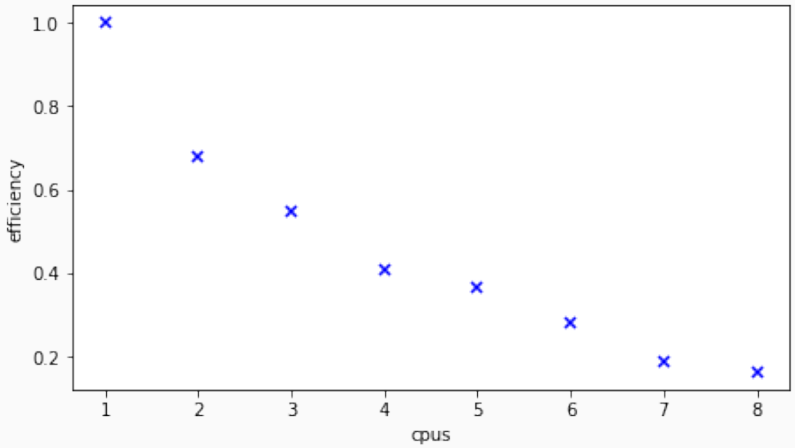
```
factorising 1 matrix:  0.7530755996704102 seconds
factorising 10 matrices:  6.5042195320129395 seconds
factorising 10 matrices:  6.41687536239624 seconds
factorising 10 matrices using 2 cpus:  4.728888988494873 seconds
factorising 10 matrices using 3 cpus:  3.8520638942718506 seconds
factorising 10 matrices using 4 cpus:  3.889279365539551 seconds
factorising 10 matrices using 5 cpus:  3.4911084175109863 seconds
factorising 10 matrices using 6 cpus:  3.750725507736206 seconds
factorising 10 matrices using 7 cpus:  4.877696990966797 seconds
factorising 10 matrices using 8 cpus:  4.9028003215789795 seconds
```

The serial execution time is $T_s \approx 6.42$ s.

Example - Parallel Speedup and Efficiency

The speedup and efficiency for $2 - 8$ cores:





The parallel efficiency suggests that this problem has quite poor scaling. What are some potential reasons for this?

Superlinear Parallel Speedup

You might expect that if we go from 1 to N CPUs, that the maximum possible speed up would N i.e. linear speedup.

In most cases of parallelisation, we will achieve sublinear speedup. It is possible in some situations to achieve superlinear speedup - *doesn't this violate reality?!*

- Some parallel processes are not fully independent - the result of one process can influence another, potentially leading to a faster solution than independent processes.
- In modern computers, having multiple CPUs working on the same problem might increase their available memory, allowing for faster computation than working independently.

Learning Outcomes

Learning Outcomes

- Know what a profiler is and what it is useful for.
- Understand the output from `cProfile`.
- Awareness that modern computers typically have multiple CPUs.
- Know what is meant by concurrency and parallel computing.
- Calculate parallel speedup and efficiency based on serial and parallel run times.