Week 3: C++ Benchmarks

I've uploaded a C++ program that includes two functions: `CPUBenchmarkSC` for single core and `CPUBenchmarkMC` for multiple cores. I've logged the results for both tests in a table below.

Single core performance

Using only one core of the CPU it took several minutes to perform the 10,000 iterations. In the task manager I wasn't able to see a 100% use of the core, only peaks that sometimes reached 90%, but were mostly around 60%.

Multi-core performance

Visual Studio C++ supports OpenMP, so I created a new function that does the same operations but in parallel. There is a significant reduction in time because all cores worked at 100% connected to a power supply. When running on battery power it stabilises on 80%.

Results

The following are the results of the benchmark for our selected case (array dimensions of 5,000 and 10,000 iterations). The processor used is the Intel i5-10210U that uses 4 cores.

Table 1: CPU single and multiple core test in C++.

Single core	**Multi-core**
565.581	102.795

The results seem to be better for OpenMP because the single core test didn't extract all of the performance available. A practical but technically wrong solution would be to use the multi-core test and limit all cores except one. Additionally, I've performed the CPU benchmarks in Fortran, MATLAB and Python.

Additional results

Fortran, whether using the `matmul` function or a simple loop never reaches a 100% use on any of the cores. Using `matmul` on Fortran is similar in speed compared with OpenMP on C++.

Table 2: CPU results for MATLAB, Python and Fortran.

Language	**Single core**	**Multi-core**
MATLAB	N/A	109.247901
Python	N/A	46.659596
Fortran	101.297 (`matmul`)	

Conclusions

MATLAB and Fortran didn't use 100% of CPU but achieved results similar to C++ which did use all cores at full capacity. I don't know if Fortran uses BLAS (Basic Linear Algebra Subprograms) but Python and MATLAB do. Python gives the best results but that might be because MATLAB limits CPU usage. I've looked for information about how BLAS works but it is low level and requires time to understand in detail.

```matlab
rng('default')
A = randn(5000);
x = randn(5000, 1);

tic
for i = 1:10000
    b = i*A*x;
end
toc

disp(b)
```

Figure 1: Code used for the MATLAB CPU test.