ENCE360 Assignment

Threading and Processes

Blake Manson – 54426511

10/10/25

Introduction

This assignment covers the performance differences between serial, multi-threaded, parent-child, and combined multi-threaded parent-child programs based on their ability to integrate a range of a function using the trapezoid method. Each graph will be using test cases described in the figure title, matching to a txt file within this folder. Each test case is run 10 times and averaged to minimise noise.

Serial

The serial application is the base of this assignment. The program prompts the user for a pre-described function for integration, a range, and a number of slices. This data is then processed, and the integration is performed, printing the answer to *stdout*. This program is completely linear, so it is expected to be the slowest as each intruction is performed one after the other. This is shown in Figure One, with each x10 of inputs, time is x10 of the previous.

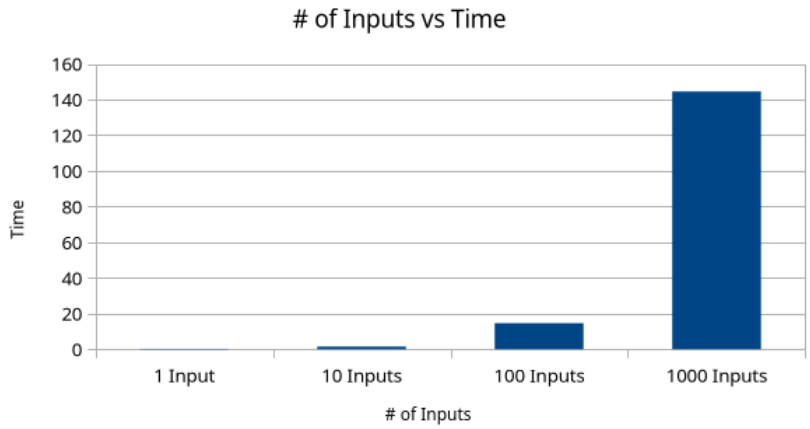

Figure One: Serial Processing Speed

Thread

The multi-threaded application integrates the given function by splitting it into *n* horizontal parts, with *n* representing the number of threads the user has chosen to run the program with. These horizontal parts are integrated seperately, getting a value for the area of the range they were given. Using a mutex, these areas are all added together to form the total area of the original function. The integration is being done in parellel with itself, as stated in this courses lecture slides [1], speeding up the process proportional to the number of threads being used, until the CPU core limits are

reached. This is shown in Figure Two, with the limit being reached around 6. The CPU running the code has 6 cores [2], so this is expected.

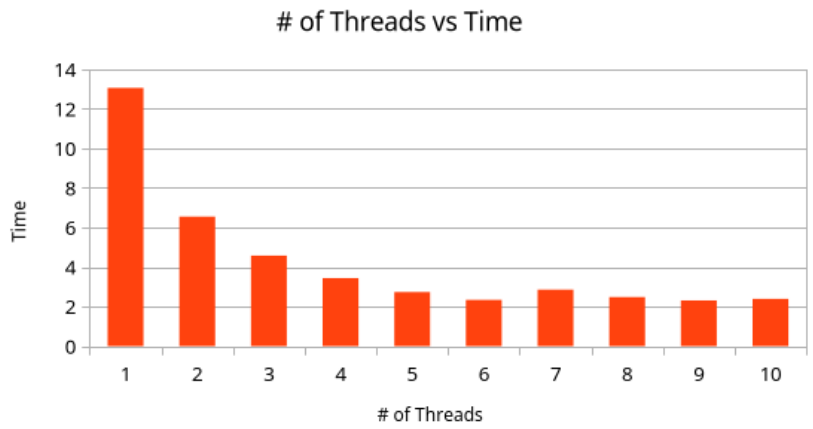

Figure Two: Thread Processing Speed

Process

The parent-child program performs the integration linearly – the same as the serial program – however multiple programs are run at once. Feeding multiple lines of input will create child processes which all run at the same time with each child calculating a different input line. The results will come to *stdout* in the order that they finish. While this would have no effect on speed for a single input, it gains relative speed the more input lines there are.

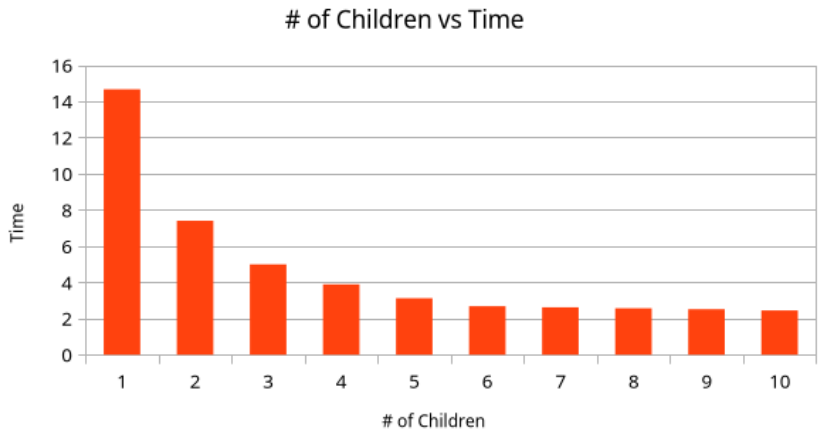

Figure Three: Process Processing Speed

Figure Three shows the time taken for process to calculate 100 inputs for varying numbers of children allowed at one time. Time decreases per child added until CPU limit of 6 cores is reached, a known hardware limit of parallelism [4].

ProcessThread

Combining the multi-threaded and parent-child programs makes multiple inputs run in parallel to each other in child processes, with each child using multi-threading to speed up their calculation. This results in the biggest increase of speed for multiple inputs, and a slightly slower single-input calculation speed compared to just multi-threading due to the larger overhead.

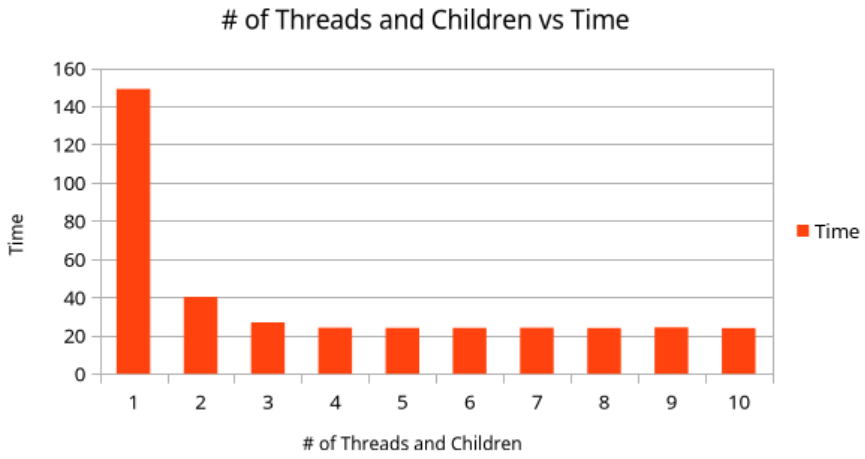

Figure Four: ProcessThread Processing Speed

Figure Four shows this time decrease. A very steep decrease from one to two is the combined effect of processes and threads, then it levels out as the CPU limit is reached.

Program Comparison

For Figure Five, thread and processThread run with 4 threads, and process and processThread run with 4 max children.

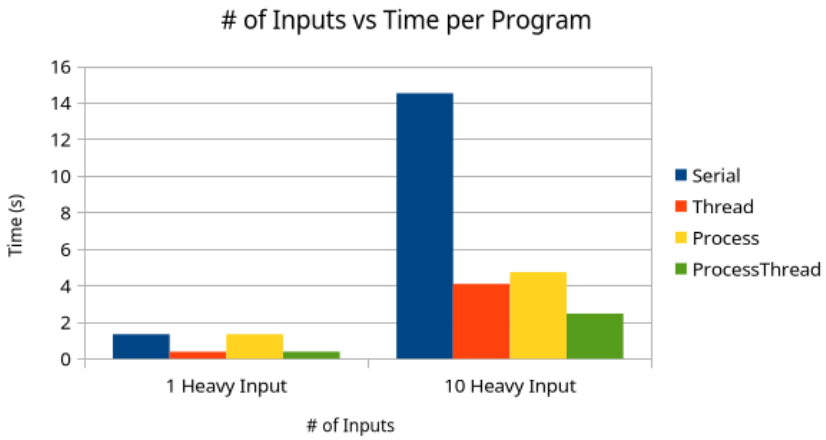

Figure Five: All Programs vs Single and Multi Input Runs

When the intensity of the calculations is large enough to take advantage of multi-threading and processes, the effect of multi-threading can be seen clearly. Heavy inputs are trapezoid integrals with 9e^8 slices. For one heavy input, thread and processThread have the fastest times (0.374 and 0.381 respectively). Multi-threading allows each input to be run in parallel. For an input that takes 1.33 seconds for one thread (serial), using four threads almost quadruples the speed, consistent with Intel's analysis of scaling effieciency [4]. The same applies for processThread, however due to a child being made, some overhead is added to the time. For multiple heavy inputs, processThread is the best as it take the single input and multiple input parellelism and uses them both. Serial is the slowest with no parellelism.

References

[1] S.Spekreijse, *"Lecture 11: Threads"* ENCE360: Operating Systems, University of Canterbury, Christchurch, presented Sept. 2025.

[2] TechPowerUp, *"AMD Ryzen 5 3600 Processor,"* TechPowerUp, 2020. [Online]. Available: https://www.techpowerup.com/cpu-specs/ryzen-5-3600.c2132. [Accessed: 7-Oct-2025].

[3] NVIDIA, *"The Role of Parallelism in Modern Computing,"* NVIDIA Developer Blog, 2021. [Online]. Available: https://developer.nvidia.com/blog/the-role-of-parallelism-in-modern-computing/. [Accessed: 7-Oct-2025].

[4] Intel Corporation, *"Optimizing Application Performance with Multithreading,"* Intel Developer Zone, 2022. [Online]. Available: https://www.intel.com/content/www/us/en/developer/articles/technical/optimizing-application-performance-with-multithreading.html. [Accessed: 7-Oct-2025].