ME 5773

Team 3

Team Members:

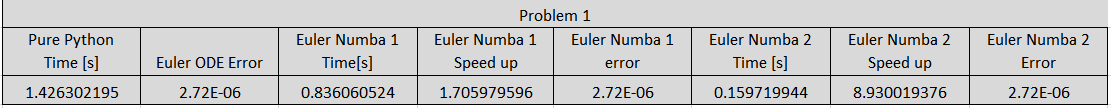
Samuel Roberts

James Standard

Homework Assignment 8

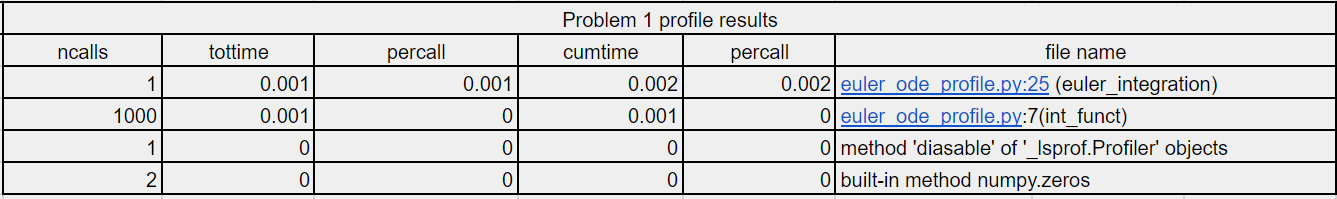
**Problem 1**

This problem compares the run time execution of three different scripts. The objective of each program is to evaluate/solve the same differential equation. The evaluation implements Euler’s method, which is a numerical method that uses the computational power of computers to solve the Ordinary Differential Equation (ODE). The program was run in three different variations: a pure python script, a partially compiled python script (Euler Numba 1), and a fully compiled python script (Euler Numba 2).

The performance (speed up time and execution time) of the three variations are shown in figure 1 below.

*Figure 1. Euler Method Script Comparison Figure*

Figure 1 shows the performance improvements of utilizing Numba in compiling python functions into machine code. Euler Numba 1 script only compiled the integration function, while Euler Numba 2 script compiled both the integration function and the euler integration function. These two functions that were compiled were found by implementing the line profiler module within the pure python script. The results of this function are shown in Figure 2 below.

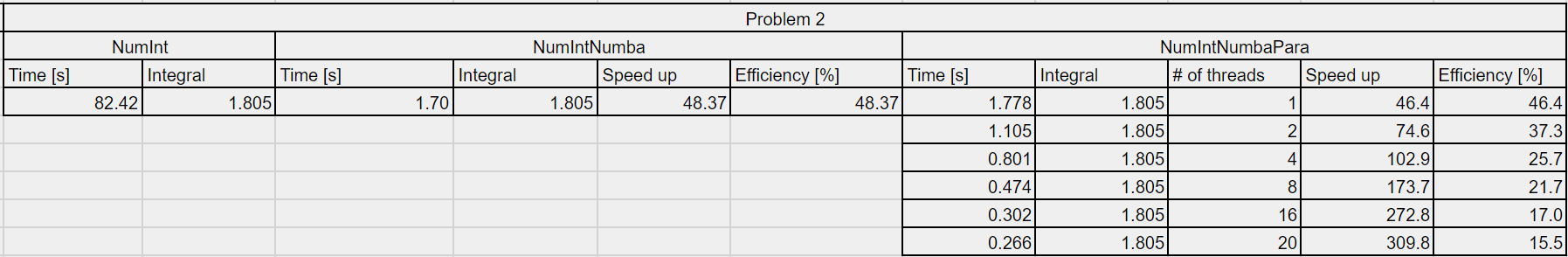


*Figure 2. Line Profiler Results*

As seen in Figure 2, the two functions that used the most time for the pure python script were the euler integration function and the integration function. The results of the line profiler were used in increasing the python script’s performance. Numba 1 only compiled the integration function portion, and this script was able to speed up the execution 2.1 times the pure python script. However, Number 2, which compiled both functions, saw a speed up performance of 22.4 times the pure python script. These results are expected since the more functions complied to machine code should increase performance execution time of the program.

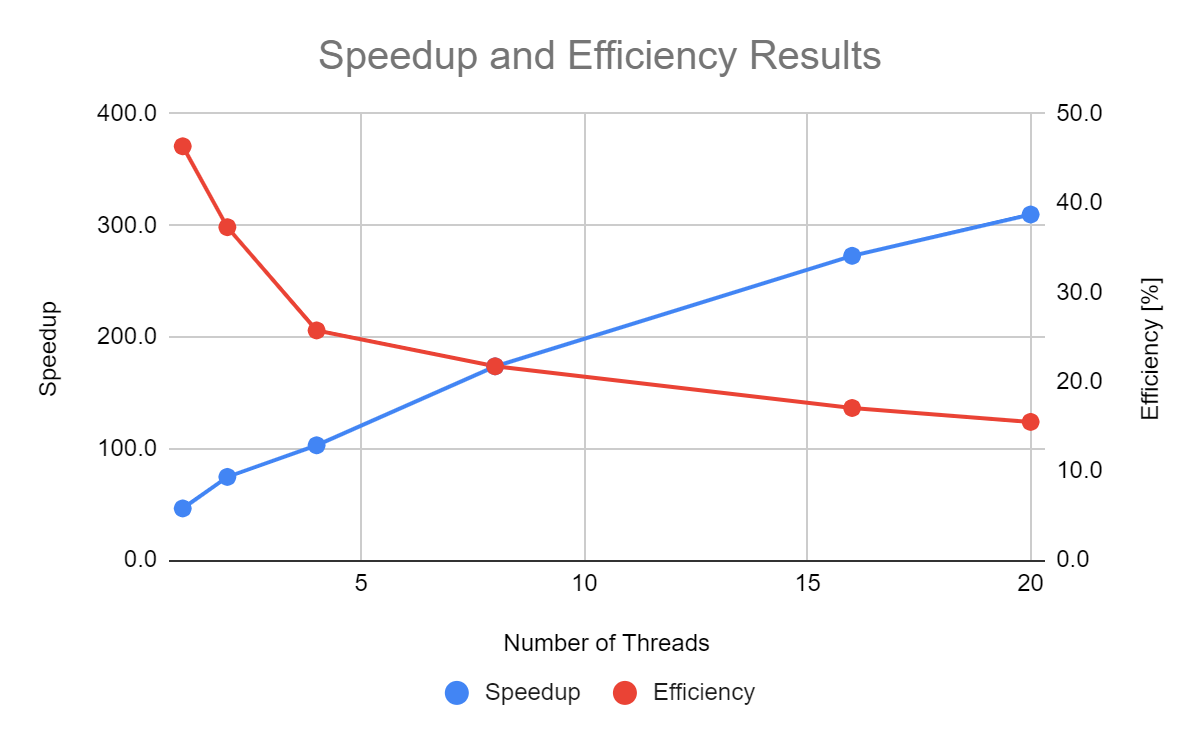
**Problem 2**

Problem 2 compares the performance and speed up time of a numerical integration function against different number of threads used in the execution. Numba was used in compiling a portion of the numerical integration program with the flags nopython=True, parallel=True, and Numba’s prange function. The speedup and efficiency of the parallelized implementation of the program was compared against utilizing 1, 2, 4, 8, 16, and 20 number of processors. The comparison results are shown in Figure 3 below.



*Figure 3. Numerical Integration Performance Results*

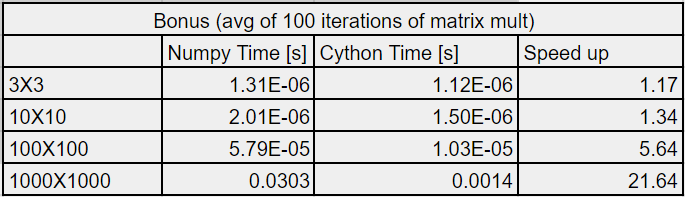
The results in Figure 3 show that the speedup time increased as the number of threads increased. This is as expected when parallelizing. However, the efficiency decreased with the increased number of threads. A plot depicting the same information is depicted in Figure 4 below.



*Figure 4. Plot of The Numerical Integration Performance Results*

**Bonus**

The bonus problem compares the execution time of Python’s numpy matrix multiplication function against a Cython compiled matrix multiplication program. Cython was used to compile the matrix multiplication script that was first written in python. In theory, the compiled program should perform faster than python’s pure program, which completes the same task. An average of 100 matrix multiplications were executed and the execution time was compared against numpy’s function and Cython’s compiled function. The sizes of the matrices included 3x3, 10x10, 100x100, and 1000x1000. The results are shown in Figure 5 below.



*Figure 5. Plot of The Numerical Integration Performance Results*

As expected, the Cython compiled program performed better than numpy’s matrix multiplication function. Also, as the matrix size increased, the speedup performance of the Cython program increased.