**Homework #8**

**Team 2:**

**Julian Carvajal Rico**

**Roberto Enriquez Vargas**

Table 1 shows the total execution times for the 3 scripts used in part 1. As shown below, the pure python takes the longest to run. When the pure python script is analyzed with the line profiler, we find that there are two lines responsible for a majority of the run time. The first line is creating and filling in the time step values of variable t. The second line, which takes even more time than the first, is creating and filling in the values of variable y, which is being used to approximate the solution at the given time step. Using just in time compilation on the int\_funct function reduces the total execution time by approximately 32% when compared to the pure python code. However, this only addressed one of the bottlenecks in our code, which explains why we only saw a 32% improvement. When both bottlenecks are addressed in the Numba2 script, we see an improvement of approximately 97% percent. This demonstrates the performance advantages of bypassing python API.

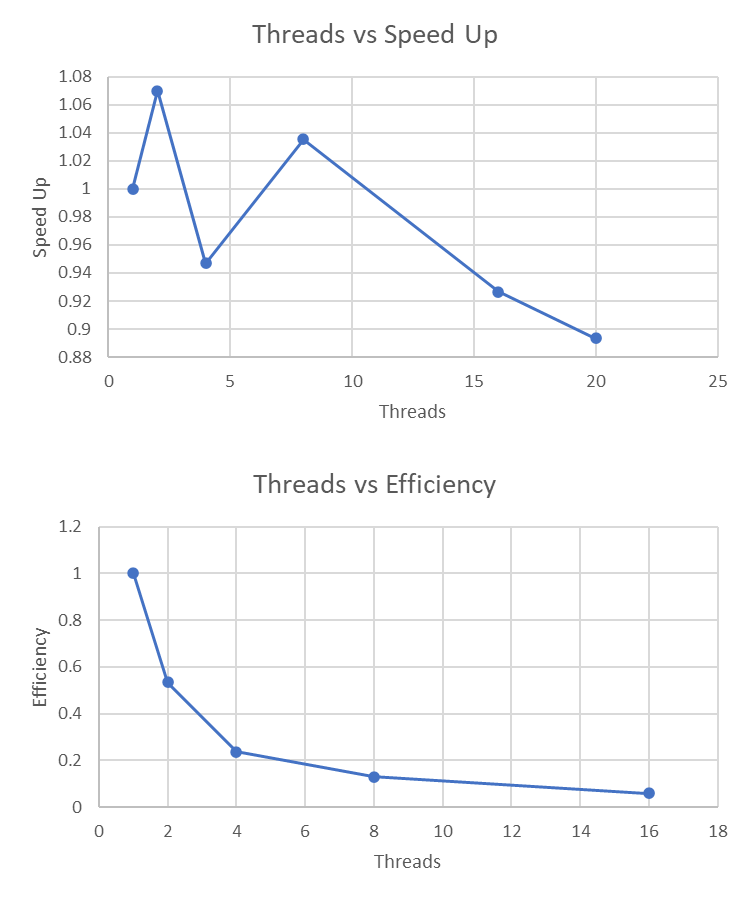
**Table 1. Part 1 Results**

|  |  |  |  |
| --- | --- | --- | --- |
| Method | Dummy Execution Time (s) | Total Execution Time (s) |  |
| Pure Python | N/A | 12.166441917419434 |  |
| Numba1\* | 0.8864812850952148 | 8.329026222229004 |  |
| Numba2\* | 0.5645143985748291 | 0.35559582710266113 |  |

Numba's JIT compilation implementation dramatically speeds up the process compared to pure Python, cutting the execution time from 86.92 seconds to about 1.72 seconds which is shown in Table 2. Although two threads parallelization at first marginally improves performance, increasing the number of threads does not result in proportionate benefits, demonstrating diminishing returns. Increasing the number of threads causes efficiency to drastically decrease as shown in Figure 1, indicating hardware constraints such as CPU core saturation or a lack of parallelizable work, or overheads associated with thread management. After a decent number of threads, the benefits of more threads seem to be negligible, and the performance looks to be at its best. This demonstrates how parallelization must be balanced with the real-world constraints imposed by the task and hardware specifications.

**Table 2. Part 2 Results**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Method** | **Number Threads** | **Total Execution Time (s)** | **Speed Up** | **Efficiency** |
| Pure Python | N/A | 86.920012 | N/A | N/A |
| Jit Decorator | N/A | 1.722404 | N/A | N/A |
| Jit + frange | 1 | 0.122196 | 1 | 1 |
| Jit + frange | 2 | 0.114198 | 1.070036 | 0.535018 |
| Jit + frange | 4 | 0.129069 | 0.946749 | 0.236687 |
| Jit + frange | 8 | 0.118001 | 1.035551 | 0.129444 |
| Jit + frange | 16 | 0.131884 | 0.926542 | 0.057909 |
| Jit + frange | 20 | 0.136802 | 0.893233 | 0.044662 |



**Figure 1. Threads vs Speed up/Efficiency**