Attempting Performance Improvements
of Subhalo and Particle Trajectory Analysis (SPARTA)

>Jakob_Wachter | jwachte1@terpmail.umd.edu
>./computer_science --physics & SDU

>Research_Question

SPARTA is a high-performance framework used to analyze astrophysical simulations. These analyses can take a long time to run, and we worry that they do not take full advantage of our computing power. Can multithreading fix that, and improve runtimes?

>Context

Many high-performance scientific codes take advantage of a computing technique, known as *parallelization*, to distribute the computations done across many different processors. SPARTA is one such program. However, another distributed computation technique exists, known as *multithreading*. This is when computation is split into many chunks, and the processor is able to, in effect, juggle these tasks to ensure that it is always doing something. Our hope is that by introducing multithreading to SPARTA, we can decrease the run-time of its large-scale analyses by ensuring that there is less downtime between processors.

>Methodology

My research consisted of analyzing the following phases:

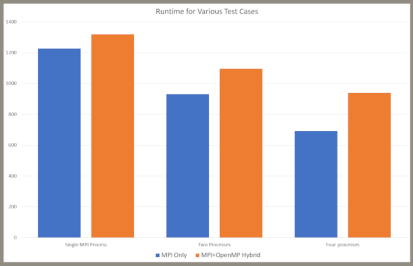
1. Analyzing the runtime of SPARTA to determine where bottlenecks occurred, and see if we could model its behavior;

2. Implementation of multithreading algorithms and techniques into the existing SPARTA code;

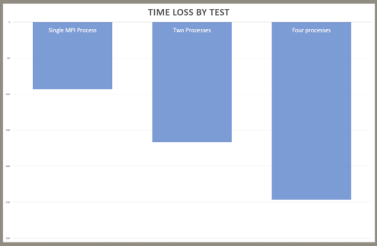
3. Testing the updated SPARTA code to determine if speedup was observed.

Phases 2 and 3 were performed iteratively until no further techniques were readily implementable.

>Data_Collected



>./figure_1:
$ The original MPI parallelized version of SPARTA versus the new hybridized OpenMP+MPI code. Note that the original code wins out in almost every case.



>./figure_2:
$ Time lost per test case versus the original MPI code. Note that the time loss is roughly proportional to the number of MPI processes used.

>Conclusions

It is evident that the current implementation of OpenMP introduces a large overhead that counteracts the utility of implementing the multithreading into the program. Future work will have to be done to mitigate the overhead associated with some of the directives that OpenMP uses, as at the moment the algorithms do not appear to scale as anticipated.

Once work is done in this regard, more work can be done with respect to implementing OpenMP algorithms in other parts of the program. At the moment, the OpenMP utilities are only used in two distinct locations, but have the potential to be viable in many more portions of the code once these major issues are corrected.

>Acknowledgements

The author of this poster would like to thank:

1. Dr. Benedikt Diemer, for being a wonderful mentor and advisor;

2. Dr. Alan Peel and Mrs. Erin Thompson, for their kindness and support as the directors of the SDU program;

3. their family and close friends, for supporting their scientific endeavors.