Project Objective:

Read the wiki_dump file into memory and find the maximum value of ASCII character numeric values for all the characters in a line. The file is located at ~dan/625/wiki_dump.txt. The output should be a printed list of lines, in order, with the line number and maximum ASCII. This is to be done using threads, MPI, and OpenMP (with an optional implementation in CUDA). These solutions will be run on Beocat via the job scheduler using different parameters (number of cores, etc.). The results of these tests are listed below.

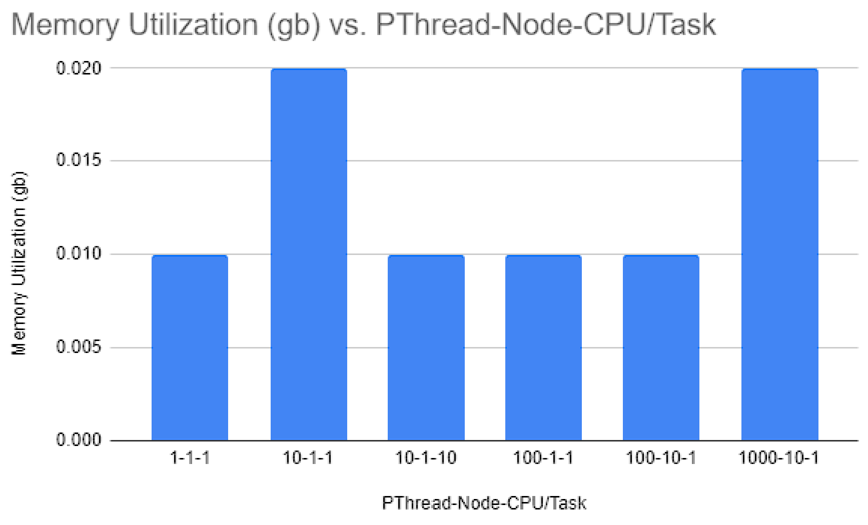
Pthreads:

Summary:

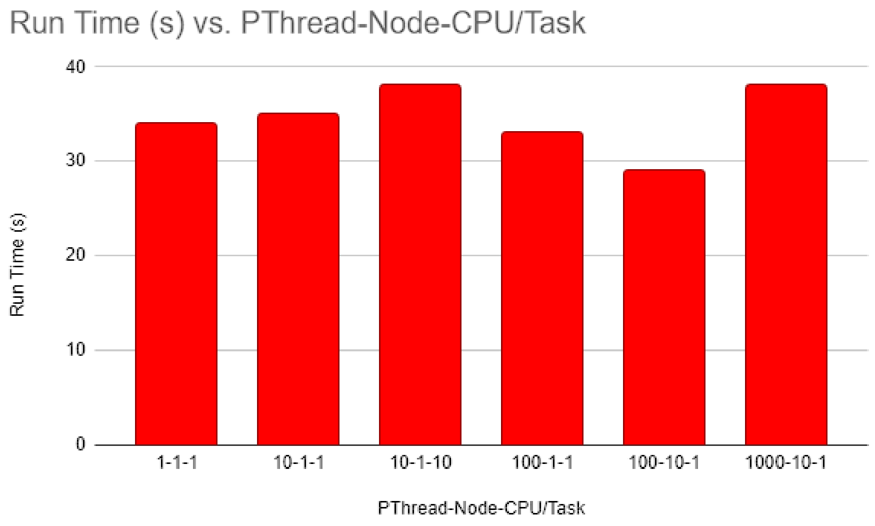
Our pthreads implementation works by making threads using pthreads, reading in the wiki dump file in chunks (lines), and then using a function to find the max ASCII value on a line. First, we attempt to open the file from "~dan/625/wiki_dump.txt" making sure to check for a NULL file. Then, memory is allocated for the lines in a chunk of data and then read into the chunks. Next, the pthreads are created with the proper arguments that will be running the "find_max_ascii" function. The results are then printed, the memory is freed for the lines, and the file is closed.

Analysis:

When running our pthread implementation the memory utilization is almost identical regardless of number of pthreads, nodes, or CPUs per task. Since the results only go to the hundredth of a decimal point, it's possible that the results are even more similar than they appear (varying between .01 gb and .02 gb) and just be rounded to make them look farther apart (.0149 would round to .01 whereas .0151 would round to .02). This indicates that the CPUs are not being used efficiently.



The run time does vary more, with the fastest time being with 100 pthreads, 10 nodes, and 1 CPU per task, and the slowest time being a tie between 10 pthreads 1 node and 10 CPUs per task, and 1000 pthreads 10 nodes and 1 cpu per task. Since the test with the highest number of CPUs per task also has the longest run time, this also indicates that the CPUs are not being used efficiently



There was a race condition for the pthread mutex object, however it was handled using a lock/unlock. The pthreads are not communicating, and there is no synchronization between processes, instead at the end the result is printed in the proper order in a print_results method.

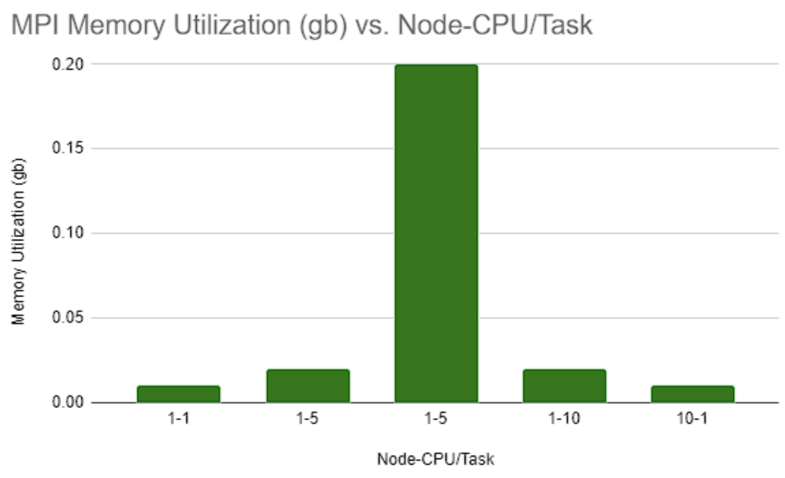
<u>**MPI:**</u>

Summary:

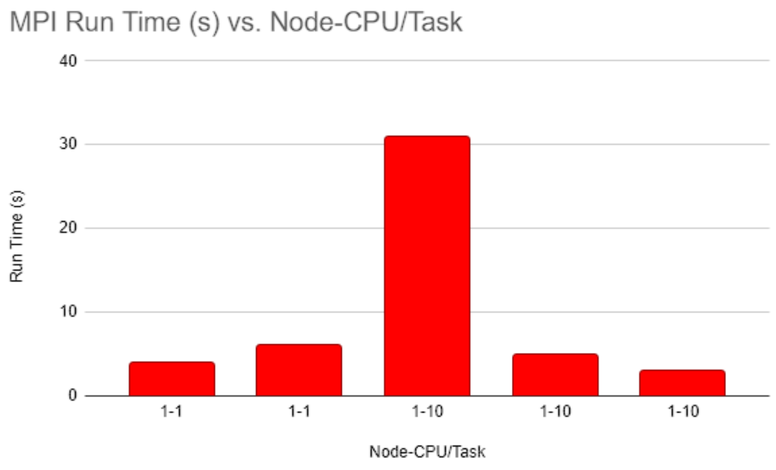
The MPI implementation is quite similar to the pthreads implementation. There is still the "find_max_ascii" function which calls the "max_ascii_value" function. The primary difference is that we start our main function by initializing the MPI environment. Next, the main function attempts to open the file and allocate memory for all the lines of text to process. Then, the calculations of max ASCII values happen and are printed until all lines are processed. Finally, the file is closed, and the MPI environment is finalized (terminated).

Analysis:

The memory utilization for the MPI implementation was similar to the p-threads implementation with most of the tests using .01-.02gb, however there was one test ran with the MPI implementation that used .2 gb. Since that test also ran in significantly more time compared to the rest of them, it likely just got caught up on something. Another test ran with the same settings had a result in the range we expected which indicates something was possibly off with the set up or BeoCat when that test was run.



The runtime of the MPI was generally much faster than the runtime of p-threads with most tests taking around 5 seconds to complete (with the one outlier being mentioned above).



<u>OpenMP:</u>

Summary:

TO DO

Analysis:

TO DO

<u>CUDA (Optional):</u>

Summary:

TO DO?

Analysis:

TO DO?