CIS 520 – Spring 2019 Project #4 – One Program, Three ways

Now that you've learned how virtual memory, multiple programs/threads, and system calls work, it's time to see how parallel and distributed applications perform on a real OS. You will be developing a parallelized program using the three standard tools for building such programs: Pthreads, OpenMP, and MPI. You will also have the chance to implement your solution in CUDA to run on a GPU for extra credit. Sample applications for pthreads, OpenMP, and CUDA with a similar theme will be uploaded to Canvas in the Files/Projects directory.

DUE: Upload via Canvas no later than 11:59 pm on Sunday, May 5, 2019

TO DO: Upload a gzipped tape archive file called Proj4.tar.gz that includes:

- A design document (Design4.pdf) in the top-level directory, which includes a discussion of your software architecture for each of the three versions, and a performance analysis.
- All of your source code for the three (or 4) versions. Put each version in its own subdirectory (/3way-pthread /3way-openmp /3way-mpi), and include a Makefile and submission script in each directory to run the program.

TYPE: Group

Program description:
I am interested in the longest common substring that we can find in large text files. On Beocat there is a moderately large (wiki_dump.txt, 1.7GB) file containing approximately 1M Wikipedia entries, 1 entry per line. You can find the file in ~dan/625 , along with other sample programs, on Beocat. Use this file – do not make your own copies of the data files.

Read the file into memory, find out the largest common substring for each pair of strings (linear pairwise, like 0 & 1, 1 & 2, 2 & 3, etc. – but you don't have to worry about 1 & 3), and then print out a list of lines, in order, with the longest substring. E.g.

0-1: The word was
1-2: analytically we see that
etc.

Your output should be identical for all versions of your code.

Mechanics

1. Your first task is to implement your solution using OpenMP. Your second task is to do a performance evaluation across a range of input sizes and core counts to see how the various versions match up in terms of run times, CPU efficiency, memory utilization, etc. You'll want to keep the machines constant, so use the Slurm flag to squeue confine your jobs to only the 'elf' class nodes. Use a reasonable number of cores – up to 32 on Beocat. Show how performance differs (if it does) across multiple machines vs. a single machine (you will only be able to do this up through 16 cores).

   Graph your performance results. Discuss them. Are there any race conditions? How do you handle synchronization between processes? How much communication do you do, and are you making any attempts to optimize this? Why or why not?

2. Repeat step 1 for pthreads and OpenMP.

WARNING: The 1 core/60MB data elements version of the code can take a long time to run, so start early!

Resources
Pthreads - https://computing.llnl.gov/tutorials/pthreads/
OpenMP - https://computing.llnl.gov/tutorials/openMP/
MPI - https://computing.llnl.gov/tutorials/mpi/
Longest Common Substring - https://en.wikipedia.org/wiki/Longest_common_substring_problem

For help using the Beocat scheduler and its command lines, see the general help page
https://support.beocat.ksu.edu/BeocatDocs/index.php/Main_Page
and the scheduler help page
https://support.beocat.ksu.edu/BeocatDocs/index.php/SlurmBasics
Compute node specifications are at
https://support.beocat.ksu.edu/BeocatDocs/index.php/Compute_Nodes
Beocat introductory videos are available at
https://www.youtube.com/channel/UCfRY7ZCiAf-EzEqJXEXcPrw

You will probably need to install a shell program on your Windows machine – I use PuTTY, and program to transfer files back and forth (I use WinSCP, but other options are fine). Linux and Mac OS have terminal and file transfer programs installed by default.

Submission instructions
Submit a PDF (Design4.pdf) which contains your analysis and a copy of your code, your various controlling shell scripts, and sample output (the first 100 lines or so) as appendices. Also include the code in subdirectories as described in the "To DO" section. Submit this via Canvas. Grading will be split – 75% for the correctness and performance of your implementation (including use of shell scripts), 25% for your performance analysis (probably 1-2 pages, including graphs but excluding appendices, assuming a reasonably compact formatting scheme).

Extra credit (10 points): Implement a version in CUDA and include a performance analysis in your design document.

Comments:
- Make sure you handle synchronization properly!
- Pipelining I/O and computation may help overall performance
- Batch your reads (i.e., read in files X lines at a time) so your solution is scalable to potentially unlimited numbers of text strings.