CIS 520 - Spring 2017 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, April 20, 2017

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

- A design document (Design4.pdf) in the top-level directory
- 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.

Program description:
A friend of mine uses a list of 50,000 terms and checks them against a list of 1M text strings. On his machine, this takes 17 minutes. I think we can do much better than this. On Beocat there is a large (60MB) file containing approximately 1M Wikipedia entries, 1 entry per line. There is also a list of 50,000 words (extracted from a common cracking dictionary). You can find the files in /scratch/dan on Beocat. Use these files – do not make your own copies.

Read the files into memory, check for matches, and then print out a list of words which appeared in the text strings, with their indices (by line number). Look for each word as substring – you do not have to worry about whether it is a whole word or not. You may assume that all words are entirely in lowercase. You do not need to list words which do not have a match. E.g.

Abba: 45, 56, 30000, 999999
Bob: 1, 5, 200, 3333
etc.

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

Mechanics: Use a reasonable number of cores – up to 64 on Beocat, and all the cores on the CUDA-compatible GPUs if you attempt this.

First, you will need to get an account on Beocat, which you can apply for at https://account.beocat.ksu.edu/ . Read the documentation to learn how to use Beocat at https://support.beocat.ksu.edu/BeocatDocs/index.php/Main_Page . You can do light development work on the login nodes, but use SGE (the scheduler) to run your jobs on the compute nodes.

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.