Homework 3 - Distributed computing

Submit Assignment

Due Friday by 11:59pm **Points** 10 **Submitting** a file upload

CIS625 – Concurrent Software Systems

Homework 2: Intro to distributed programming

Assigned: 9/20/18

Due: Sunday, 9/30/18 by midnight

Individual or group: Individual

Points: 10

Description

The purpose of this project is to get you started doing distributed programming on a machine like Beocat.

Tasks

A friend of mine uses a list of 50,000 terms and checks them against a list of 1M text strings. On his machine, which has 32 cores and 128GB RAM, this takes 17 minutes. I think we can do much better than this. On Beocat there is a moderately large (wiki_dump.txt, 1.7GB) file containing approximately 1M Wikipedia entries, 1 entry per line. There is also a list of 50,000 words (keywords.txt, extracted from a common cracking dictionary). You can find the files in /homes/dan/625 on Beocat. Use these files – do not make your own copies. There is also a sample implementation using OpenMP (find_keys.c).

Read the files into memory, check for matches, and then print out an alphabetized 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 and configurations of your code**.**

Mechanics

Your first task is to implement your solution using MPI. 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 '-q *@@elves' flag to qsub confine your jobs to only the 'elf' class nodes. Use a reasonable number of cores – up to 64 on Beocat. Show how performance differs (if it does) across multiple machines vs. a single machine (e.g., 4 cores on 2 nodes vs. 8 cores on 1 node; 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?

WARNING: The 1 core/1M data elements version of the code can take 24 hrs to run, so start early!

Resources

Pthreads - **https://computing.llnl.gov/tutorials/pthreads/** (https://computing.llnl.gov/tutorials/pthreads/)

OpenMP - **https://computing.llnl.gov/tutorials/openMP/** (https://computing.llnl.gov/tutorials/openMP/)

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 which contains your analysis and separately upload copies of your code and your various controlling shell scripts. 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 2-3 pages, including graphs, assuming a reasonably compact formatting scheme).