Assignment 1: OpenMP

Due: 5 P.M. on Tuesday, April 25, 2023

- Info on the HPC3 cluster: `https://rcic.uci.edu/hpc3/index.html`
- Info on OpenMP: `https://hpc-tutorials.llnl.gov/openmp/`

In this assignment, you will implement a multithreaded version of the mergesort algorithm using the OpenMP programming model. You will use the UCI HPC3 cluster. You should also use this assignment to familiarize yourself with the tools you will be using throughout the course.

Getting the scaffolding code on the HPC cluster

We will use the `git` distributed version control system. The baseline code for this assignment is available at this URL: `https://github.com/aparnamowli/EECS-120-HW1.git`

To get a local copy of the repository for your work, you need to use git to clone it. So, let's make a copy on the HPC3 cluster and modify it there. To do that, run the following command on the cluster.

```
$ git clone https://github.com/aparnamowli/EECS-120-HW1.git
```

If it works, you will see some output similar to the following:

```
Cloning into 'EECS-120-HW1'...
remote: Enumerating objects: 10, done.
remote: Counting objects: 100% (10/10), done.
remote: Compressing objects: 100% (10/10), done.
remote: Total 10 (delta 0), reused 10 (delta 0), pack-reused 0
Unpacking objects: 100% (10/10), done.
```

There will be a new directory called **EECS-120-HW1**.

Compiling and running your code

We have provided a small program, broken up into modules (separate C/C++ files and headers), that performs sorting sequentially. For this lab, you will make all of your changes to just one file as directed below. We have also provided a `Makefile` for compiling your program. To use it, just run `make`. It will direct you with the right flags to produce the executable. For example, `make mergesort-omp` will produce an executable program called `mergesort-omp` along with an output that looks something like the following:

```
$ make mergesort-omp
icpc  -O3 -g -o driver.o -c driver.cc
icpc  -O3 -g -o sort.o -c sort.cc
icpc  -O3 -g -o mergesort-omp.o -c mergesort-omp.cc
icpc -O3 -g -o mergesort-omp driver.o sort.o mergesort-omp.o
```

Run `mergesort-omp` on an array of size `100` as follows:

```
$ ./mergesort-omp 100
```

Running jobs on the cluster

The HPC3 cluster is a shared computer. When you login to `hpc3.rcic.uci.edu`, you were using the login node. You should limit your use of the login node to light tasks, such as file editing, compiling, and small test runs of, say, less than a second. When you are ready to do a timing or *performance* run, then you submit a job request to the Slurm scheduler. To submit a job request, there are two steps: Create a batch job script, which tells Slurm what machine resources you want and how to run your program. Submit this job script using a command called `sbatch`. A batch job script is a shell script file containing two parts: (i) the commands needed to run your program; and (ii) metadata describing your job's resource needs, which appear in the script as comments at the top of the script. We have provided a sample job script, `sort.sh`, for running the Mergesort program you just compiled on a relatively large input of size 10 million elements.

Go ahead and try this by typing the following commands:

```
$ sbatch sort.sh
$ squeue -u <UCInetID>
```

The first command submits the job. It should also print the ID of your job, which you need if you want to, say, cancel the job later on. The second command, `squeue`, allows you to monitor or check the status of your job request in the queue.

For other useful commands, such as `scancel` for canceling a job, see the HPC3 cluster slurm documentation on running jobs at `https://rcic.uci.edu/hpc3/slurm.html`.

When your job eventually runs, its output to standard output or standard error (e.g., as produced print statements) will go into output files (`mergesort.out` and `mergesort.err`). Go ahead and inspect these outputs, and compare them to the commands in sort.sh to make sure you understand how job submission works.

C/C++ style guidelines

Code that adheres to a consistent style is easier to read and debug. Google provides a style guide for C++ which you may find useful: `https://google.github.io/styleguide/cppguide.html`.

Parallel Mergesort

Although we've given you a lot of code, to create a parallel Mergesort you mainly need to focus on editing `mergesort-omp.cc`. Right now, it is mostly empty. In this file, implement the parallel mergesort algorithm. To get full credit, your implementation needs to beat the "easy" parallelized algorithm in which the merge step remains sequential.

After you change your code, don't forget to recompile (by running `make`) and submit a batch job to collect timing data. If you did it correctly, the '.err' file will not show any abnormal termination errors and you will observe better performance than the sequential version. Record your

parallel mergesort performance for increasing number of OpenMP threads up to the maximum number of cores on a node (i.e., 2, 4, 8, 16, 20, 40).

Submission

Turn in your recorded performance data and tarball/zip of your code by uploading it to Canvas. **Note:** Include all the files in your submission (including the makefile and script files), not just the files you modified.

Good luck and remember to have fun!