Multithreaded Mergesort

CS453: Operating systems

Overview

All the programming assignments for Linux will be graded on the onyx cluster (onyx.boisestate.edu). Please test your programs on onyx as differences between different Linux distros (CentOS, Fedora, Ubuntu) can cause compilation issues when using simple Makefiles.

Starter Code

In your git repository you will have a folder named **mergesort** (hint: if you don't see a folder named mergesort do a **git pull --rebase** to pull down the latest code). Inside the **mergesort** folder, you will find the single-threaded mergesort code that you will be converting to use threads. Makefiles have been provided that you can use directly. Please do browse through them so you know what they are doing.

Specification

Convert the mergesort code to use multiple threads. Your program should limit the number of threads it uses via a command line argument (for example, this could be total number of threads or number of levels before cutting off new thread generation). You must get a speedup of at least **2** with **4 or more cores** to get full credit on this project. Use n = 100,000,000 elements for your testing. **NOTE: Just because you see a speedup of 2 does not guarantee full credit.** You can easily hard code the number of threads and hardcode the branches to get a speedup of 2. **Your code MUST work with a variable number of threads!** Simply getting a speedup of 2 does not indicate that you did the assignment correctly.

Provide suitable simple command-line arguments and an usage message. For example:

> Usage: ./mergesort <input size> <number of threads> [<seed>]
>
> or
>
> Usage: ./mergesort <input size> <number of levels> [<seed>]

Documentation

In your README.md file report the details regarding the performance (see below) of your parallel mergesort as well observations about the assignment. The README.md file should be at the top-level.

For the performance, provide a table of times and speedups obtained with varying number of threads (from 1 to 8). The speedups should relative to the serial mergesort. Please also note the number of cores on the system that you are testing.

Hints

- Do not modify the given **serial_mergesort** function. Instead create a new **parallel_mergesort** function that will call serial_mergesort as a base case.

- You can stop the recursion using the number of levels in the sorting tree or by number of threads. It is simpler to stop it by the number of levels.

Submission

Files committed to git (backpack)

Required files to submit through git (backpack)

1. Makefile
2. mergesort.c
3. mergesortTest.c
4. README.md

Push your code to a branch for grading

Run the following commands

1. make clean
2. git add <file ...> (on each file!)
3. git commit -am “Finished project”
4. git branch mergesort
5. git checkout mergesort

6. git push origin mergesort
7. git checkout master

Check to make sure you have pushed correctly

- Use the command **git branch -r** and you should see your branch listed (see the example below)

```
$ git branch -r
  origin/HEAD -> origin/master
  origin/master
  origin/mergesort
  . . .
```

Submit to Blackboard

You must submit the sha1 hash of your final commit on the correct branch. Your instructor needs this in order to troubleshoot any problems with submission that you may have.

- git rev-parse HEAD

Grading Rubric

Provided via backpack.