In this project, you will parallelize mergesort using Cilk and pthreads.

Overview of class projects

Each project consists of a written portion and the coding portion. The written portion should include a description of the strategies that you used to implement the assigned project, and is to be submitted in hard copy in class on the day the project is due. The coding portion should be checked into your repository before class on the due date.

Hardware and software resource for the class

In this case, we will be using a set of machines in the linux lab (lop400-<N>.cec.wustl.edu, where <N> can be any number between 01–30). You can access the machines by either going to the lab itself, or login remotely.

If you are not on the engineering school network, you will find that you cannot log into these machines in the linux lab. In order to do so, either you must first log into shell.cec.wustl.edu and then from there ssh into lop400-<N>. Or you can use our virtual private network. This section will explain the use of VPN.

1. If you don’t have it already, download Cisco VPN Client (instructions from EIT here https://engineering.wustl.edu/eit/vpn.aspx — if you have issues with EIT’s provided installer, try getting Cisco’s AnyConnect Client from a 3rd party website, or contacting EIT.

2. In the VPN client, enter cecvpn.seas.wustl.edu as the server address.

3. When prompted by the VPN client, enter your cec username and password.

4. You should now be connected to the VPN network and can log directly into one of the lop400-<N> machines.

These machines have been installed with gcc 5.0.0 (experimental), which supports Cilk Plus keywords with a compiler flag and linker flag (you can see what they are by looking into your Makefile). You are free to download gcc (Cilk Plus has been supported since gcc
and install it on your own home machine so that you can work on the project locally on your own machine (be sure it’s multicore!). If you do so, however, you are responsible for making sure that your code compiles and runs on one of the machines in the linux lab, and we will be using those machines to grade your assignment.

On the machines in the linux lab, gcc is installed in the directory /project/cec/class/cse539_sp15/gcc. Even though the Makefile is set up so that you don’t need to modify your environment variables, it is a good idea to do so, since there are other versions of gcc installed, and you don’t want to accidentally use a different version of gcc while working on your project.

To make this particular installation of gcc as the default one, set up your environment variables as follows:

```
export PATH=/project/cec/class/cse539_sp15/gcc/bin:$PATH
export LIBRARY_PATH=/project/cec/class/cse539_sp15/gcc/lib64:$LIBRARY_PATH
export LD_LIBRARY_PATH=/project/cec/class/cse539_sp15/gcc/lib64:$LD_LIBRARY_PATH
```

### Your svn repository

We will use subversion (see also [http://subversion.apache.org](http://subversion.apache.org)) to distribute the baseline code for the project. Each student in the class will be given their own svn repository, which is accessible only by you, the instructor, and a few technical support folks in the department. As is standard in the CSE Department, your repository name will be of the form `wustlkey`.

When you are prompted for a username and password in order to access this account, you should provide your `wustlkey` username and associated password, *not* your cec username and password.

The most recent checked in version of your repository can be accessed via `svn co https://shell.cec.wustl.edu:8443/cse539_sp15/svn/<wustlkey>`.

We will take a snapshot of the code right at the due time, and use that version we get from your repository for grading. After you submit your project via svn, it is a good idea to check out this URL and make sure that everything looks up to date. It is your responsibility to make sure that you submit your project both *punctually and successfully*.

### Coding: Parallelizing mergesort

As said earlier, the coding part of this project consists of implementing a parallel mergesort using Cilk Plus, and a parallel mergesort using pthreads. In class, we briefly went over a parallel algorithm for mergesort. Just as a reminder, the pseudocode for the parallel merge routine is as follows:
ParMerge\((B, m, C, l)\)

1. if \(m < l\)
2. then return ParMerge\((C, l, B, m)\)
3. if \(m = 1\),
4. then Concatenate the arrays in the right order and return.
5. \(\text{mid} \leftarrow \lfloor m/2 \rfloor\)
6. \(s \leftarrow \text{BinarySearch}(C, B[\text{mid}])\).
7. \(A'_\text{left} \leftarrow \text{spawn} \text{ParMerge}(B[1..\text{mid}], \text{mid}, C[1..s], s)\)
8. \(A'_\text{right} \leftarrow \text{spawn} \text{ParMerge}(B[\text{mid}+1..m], m-\text{mid}, C[s+1..l], l-s)\)
9. sync
10. Concatenate \(A'_\text{left}\) and \(A'_\text{right}\) and return

You are welcome to use this version of parallel mergesort, or you can be creative and do something else. That said, your Cilk Plus implementation should work correctly and have ample parallelism. For the pthreaded version, you are encouraged to extract as much parallelism as you can, but as long as it’s parallel and sorts the numbers correctly, you will get most of the credit for it.

Once you download the codebase, you will see a list of files:

1. main.cpp: the main routine that will invoke the two different implementations of sort and check their results;
2. ktiming.cpp/.h: code for time-measurement;
3. cilk_sort.cpp: where the Cilk Plus mergesort implementation should go;
4. pthread_sort.cpp: where the pthreaded mergesort implementation should go; and
5. Makefile.

During testing, we will be using exactly the same main.cpp, ktiming.cpp, ktiming.h, and Makefile that you received as part of the code base, so please don’t include any code in those files that your implementation needs for correctness.

Writeup

In your write-up, which you will turn in a hard copy in class, please include:

1. a description of your Cilk Plus implementation: you don’t have to repeat what’s said in class if you are using the same algorithm. Nonetheless, if your code does anything special for performance, for instance, coarsening the base case, reusing temporary arrays to conserve space, and such, please include them in your description.
2. a description of your pthreaded implementation: how you parallelize the code and how you handle load balancing. Also specify anything you do to improve performance / load balancing.

3. a comparison between the two: we say the Cilk Plus style of coding as “dynamic multithreading,” and the pthreaded style of “static threading.” How would you compare the two? What are the differences you observed? What are the pros and cons of each? Note that there is no right answers to these questions.