Programming Assignment #1: Parallelizing File Compression with OpenMP

Due date: Thursday 16 February at 11:59 pm (end of day)

OpenMP offers one of the simplest ways to parallelize existing serial programs based on using compiler directives to run existing codes in parallel. This first assignment explores exactly that topic.

I give you

A serial program that compresses a directory hierarchy. The program takes an input directory and output directory.

```
> ./serialcompress
Usage: compresstree inputdir outputdir
 inputdir must exist
 outputdir cannot exist
```

So, when you follow instructions:

```
> ./serialcompress /home/randal/cs420jhu/ /tmp/compressdir
```

You get a bunch of output and a directory hierarchy in /tmp/compressdir that matches that of the source dir, but has every regular file gzipped. E.g.

```
> ls /tmp/compressdir/
ompcompress  README.gz
```

The source code is available at the class' GitHub site https://github.com/randalburns/cs420jhu/ [https://github.com/randalburns/cs420jhu/]. You can use git to get the source, which I highly recommend because git is a cool tool and getting to know git will benefit you in the future. This is not a requirement. You can just download the files if you don't want to go there.

You implement

A parallel version of this code that uses up to eight threads to compress different portions of the directory tree in parallel. For example, a directory with eight sub-directories:

```
> ls /tmp/cxample
a  b  c  d  e  f  g  h
```

runs in eight different threads.

```
> a.out /tmp/example/ /tmp/other
OMP Thread 1 Directory/File b
OMP Thread 2 Directory/File c
OMP Thread 7 Directory/File h
OMP Thread 3 Directory/File d
OMP Thread 0 Directory/File a
OMP Thread 6 Directory/File g
OMP Thread 5 Directory/File f
OMP Thread 4 Directory/File e
```

This is the output from implementation which based on the following print statement.

```
printf ( "OMP Thread %d Directory/File %s\n", omp_get_thread_num(), filenames[i]->d_name );
```

You should only parallelize the top-level directory. This means that subdirectories

```
/tmp/a/1
/tmp/a/2
/tmp/a/3
```

will all be compressed by the same thread. Let me be more specific to avoid confusion. The serial implementation calls *CompressTree* recursively. **DO NOT** parallelize the for loop in CompressTree. (This would be a bad idea, because it would create a new task for every directory. Too many tasks.) Leave *compress.c* unmodified.

In your file *parallelcompress.c* you will want to do the following:

- read all the names in the input directory
- write a for loop that calls compress tree for every file in that directory
- parallelize that for loop

You also answer the following questions

1. This implementation of parallel compression is quite primitive and can exhibit poor parallel performance. For each of the three factors against parallelism, describe whether this is a problem for this code and why? For what inputs do these factors arise. They factors against parallelism are:
    I. Skew
    II. Startup costs
    III. Interference
2. Why might it be bad idea to create a new thread task for every directory in the hieararchy? In what situation (for what inputs) would this produce poor parallel performance? In what situation (for what inputs) would this produce good parallel performance?
3. Consider a different strategy for parallelizing this code in which a single thread reads the directory hierarchy and creates a list of files to compress and then the files are distributed to many parallel threads for compression.
    I. If the serial code used 5% of the time to read the files and 95% of the time to do compression, what is the maximum possible speedup for the parallel code. (Use Amdahl's law).

What to turn in and how?

- Submit a file named ***JHEDid*.parallelcompress.c**. You should replace *JHEDid* with your JHEDid, for example mine is rburns3. The file must compile with the following command.

```
gcc -fopenmp rburns3.parallelcompress.c compress.c -lz
```

- Submit a PDF file that contains the answers to the specified questions.

This should be done via Blackboard. Turning in assignments and communicating grades are the only things that we will use blackboard for.

randal/teach/cs420/openmp.2012.txt · Last modified: 2012/02/10 17:01 by randal