Parallel Programming
HW1

CHANG LIU
chang.liu@jhu.edu

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

   Not all file in each directory has the same size, thus not all task finish at the same time. Let's say if one of the root directories has a big file directly under the root folder, and this root directory has many sub-directories which has other files under them. Then as one root directory will be assigned only one thread, those sub-files won't get compressed unless the thread finished the compress of that big file. Even all other threads finished their own jobs, they couldn't help compress the rest sub-files, but have to wait for that specified thread to finish its own job.

   II. Startup costs

   This may be a potential problem for this code when the number of threads is unlimited and there are too many root directories which has no file under it. As the code will create a thread for each root directory, if there is no file under certain root directory, the program will waste a lot of time to create the threads that will not do the actual work (or, if there's only some small file under the root directory, the time it takes to create such a thread may be even longer than the time it actually needed to compress those files). In the end, the time it takes to create the threads may be much more than the time it should take to compress the actual files, thus the program will have a low performance.

   III. Interference

   The performance of this code may be low when multiple processors accessing shared resources. For example, if all the threads finished the compressing job and want to write back to the hard drive, tasks may be queued due to the capacity of the max write speed of the hard drive (I/O speed).

2. **Why might it be bad idea to create a new thread task for every directory in the hiearchy? In what situation (for what inputs) would this produce poor parallel performance? In what situation (for what inputs) would this produce good parallel performance?**

   Poor parallel performance:

   Say when there are a lot of directories (ex. 100000) in the hierarchy but only 1 of them

actually has file under it, the program may need to create 100000 threads which takes a lot of time but doing nothing. Actually only one of the threads will do the compressing things. Thus the time it takes to create the procedure is way more than the time it actually needs to compress the file.

Good parallel performance:
Say when there are 100000 directories in the hierarchy and all of them have its own file under it (not empty) (also preferred is that all files have the similar size), the program will have a good parallel performance. As all the threads will have at least one file to work on instead of nothing.

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).

   Computation can be speedup: 95%
   Thus, S = 5%, P = 95%
   Apply Amdahl's Law:

$$Speedup = \frac{1}{1-P+\frac{P}{N}} = \frac{1}{1-95\%+\frac{95\%}{N}}$$

   If we want the max speedup, we will need to have a big N, then the maximum speedup of the parallelized version is:

$$Speedup = \frac{1}{1-P+\frac{P}{N}} = \frac{1}{1-95\%+\frac{95\%}{N}} = \frac{1}{1-95\%+0} = \frac{1}{5\%} = 20$$