FileSearch Assignment

Michael Edelman

November 2021

1 Design of Sequential Implementation

The algorithm uses a nested loop to iteratively traverse the file tree. The inner loop searches a currentdirectory's contents. **If it is a file**, that file is checked to see if it is the one we are looking for. If it is we set result to that files absolutePath (via file.getAbsolute()) and break out of the loop - we're done. However, **if it is a directory**, then we set the *currentDirectory* to that directory and start the loop again, this time searching that directory. That is how the algorithm descends down the tree. The inner loop is meant to terminate when all of the current directory has been searched (All of its files and directories have been checked). This is where the outer loop comes into play. The outer loop resets the current directory to its parent to finish searching *THAT* directory's contents because remember when we descend into a directory we didn't nesecarily search all the other directories or files yet. A map is used to keep track of the index (corresponding to an item in a directory) to start or continue searching from when the algorithm ascends back up.

2 Design of Parallel Implementation

The Parallel algorithm is virtually identical to the sequential except for 4 lines of code. This is due to the realization that the sequential task can be decomposed into many independent subtasks. The task is simply to search the contents of a directory. The larger problem asks that we search the contents of a root directory. Notice that directories usually contain other directories that need to be searched therefore we can model that as a subtask. This is precisely what the algorithm does. Whenever the algorithm reaches a directory on its search it gives the task to a thread instead - if one is available. Otherwise the algorithm precedes identical to the sequential one. Basically the parallel algorithm can be summed up in a sentence. It is the sequential algorithm except upon encountering a directory the algorithm gives it to a thread if available to search that directory. This is visualized below.

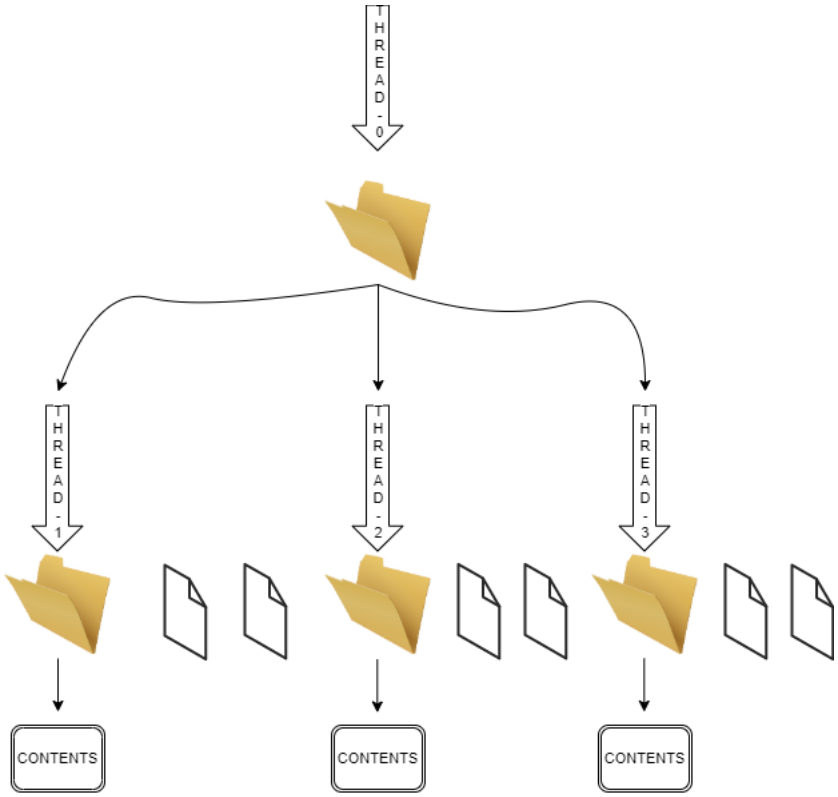

Figure 1: Parallel Scheme: Papers = Files, Folders = Directories

3 Performance Infrastructure or Assumptions

Number of Files - 22,000
Number of Directories - 104,000
In theory it can traverse a much larger tree, but I only tested it with this (my own file tree). That being said I felt 126,000 files and directories was an adequate sample size to determine effectiveness of the algorithm both in terms of correctness and performance.
Number of Cores - 4
Number of Logical Processors - 8
Windows File System

4 Performance Results

table raw numbers sequential - time for different size file sets and finding different files. then a graph that compares parallel performance to the seq. x axis files/directories Y axis time to find