Dennis McWherter
17 September 2011
CS 241
dmcwhe2

In MP3 we explored the basic concepts of parallel programming with threads. There are many reasons we would be interested in running software in parallel. Particularly for parallel sorts, we can complete the work significantly faster by delegating the work to multiple threads rather than running a non-concurrent piece of software. With that in mind, this process is made faster since we used threads rather than processes. Threads are light-weight as opposed to processes which require a separate copy of almost everything. Threads operate within the same address space as their parent process and, therefore, share the same memory (with the exception of their stack and a few other items). Threads have performance advantages over processes in terms of speed complexity and space complexity since they can avoid copying more information than needed. Also, since they run in a single process, they do not take up space in the process control block.

In my implementation of MP3, both the sort and merge operations run in parallel, however, they run independently. A better design would be to run sort and merges in total parallel. That is, after two sorted files are ready to be merged, begin the merge. In the current design, all the files are sorted in parallel and when sorting has completed, merging begins. Furthermore, I had a unique "pseudo-queue" idea to control which files merge at what point. I created a simple array and maintained it as a queue (rather than implementing the fully functional data structure). This way, I could tell which files had yet to be processed and how many remained in the merging process. I then took these files until there were either 0 or 1 files left to be merged (i.e. 0 if even number and 1 if odd number). I repeated this process until all files were merged.

There were, however, a few pitfalls in implementing MP3. I would recommend people remain mindful of the location of their file pointers. Particularly, if you finish processing a file and will be using the same file pointer again, rewind() (or fseek() to position 0) the file so it points to an expected place in the file when processing it in subsequent methods. Also, for the implementation of the merge-tree, try to write out and understand what is happening before implementing it (or you will confuse yourself). Therefore, MP3 provided a thorough foundation for exploring the concepts of parallel programming.