Teague Forren
CPEG-655 Lab 3

The machine used for this lab runs an intel 8 core i7-6700 @3.4GHz and 8GB of DDR4 RAM on Linux 4.4.0 64 Kernel.

Part A

Part A Is simply the implementation of the add, find, add_if_not_present, size, and checkIntegrity. The only thing of note is that I changed the function declarations to so that I can designate which tree the functions will operate on rather than having a global tree variable. The actual functionality is not changed.

Part B

Part B baseline executes with a PAPI_L2_DCM cache miss count of: ~53147018

Part B cache misses can be minimized through the idea of blocking chunks of memory. As we allocate memory for the nodes as we create them, we can do so in large chunks such that functions operating within a particular chunk can benefit from cache locality as the cache loads other nodes in the mutual chunk. No functional implementation was found.

Part C

Part C is comprised of two separate files: Tree_c1.c and Tree_c2.c. Tree_c1 is the baseline 16 thread implementation which locks the entire tree as it runs per thread and Tree_c2 is the improved implementation.

Tree_c2 implements two root locks for the left and right branches. When one thread begins traversing the tree down a branch of the root node, that branch becomes locked. However, the other branch is still usable by another thread. 2 threads run at the same time for the left and right branches.

Further detail on implementation:
The addRoot function is the first function to be called in a thread. It then recursively calls the add function which has does not perform any further locking. The result is that a thread will run addRoot, decide on a branch (left or right), lock it, and then continue to add the new node somewhere down that branch before unlocking the entire branch. Meanwhile a second thread can do the same on the opposite branch. The result is a maximum of two threads running in parallel while the others wait for their respective branches to unlock.

Tree_c1 (baseline implementation) with 16 threads **~134 seconds**

Tree_c2 (left / right branch locks) with 16 threads executed in **~117 seconds.**

The use of 2 separate locks for the left and right branch of the root node results in a ~17 second speedup.

Compiling and Execution

The files are compiled and run separately with -pthread. Example:

```
gcc tree_c1.c -pthread && ./a.out
```

Will run C's baseline 16-thread full-tree locking.

The improved Part C execution can be compiled and run with:

```
Gcc tree_c2.c -pthread && ./a.out
```