Mykola Gryshko: mg1250

iLab machine: vi.cs.rutgers.edu

Overview

My library is modeled after a scheduler that prioritizes threads in the order they are created. Threads that run through the quanta of time allotted to them (25ms) and then enter a lower priority queue. Threads of lower priorities run at multiples of this time, charted below:

High--25ms
Medium--125ms
Low--250ms

By ascribing lower priority threads more runtime, the chance that the scheduler would neglect them to a point when only new threads are running is reduced. With the model I implemented the scheduler should weave through threads of varying priority to ensure that lower priority queues enable threads to run before newer threads are added. In addition to this overall schematic, the mutex struct defined in my_pthread_t.h contains a queue to order the threads requesting it. The maximum number of threads and mutexes that can run is 32. At a larger scope the scheduler struct itself contains the MLPQ, main and current running contexts, and two queues to record which threads have been terminated or running.

The maintenance method in my_pthread.c to clean up the scheduler and ensure that old threads do not get starved of runtime. First maintenance() frees memory of terminated threads. From here the method takes any thread which has ran at least 10 times and moves it to the highest priority queue.

Benchmarking

Below is the comparison between my user library and the real pthread library on vectoryMultiply.c:

num_threads	my_pthread.c	pthread.c
2	2800ms	173ms
4	106ms	258ms
8	108ms	280ms
16	116ms	291ms

32	133ms	312ms

Below is the comparison between my user library and the real pthread library on externalCal.c:

num_threads	my_pthread.c	pthread.c
2	17487ms	1520ms
4	17182ms	1896ms
8	19560ms	1917ms
16	19243ms	1916ms
32	19057ms	1930ms

Below is the comparison between my user library and the real pthread library on parallelCal.c:

num_threads	my_pthread.c	pthread.c
2	3022ms	1637ms
4	2877ms	1647ms
8	2924ms	1524ms
16	2878ms	1640ms
32	2839ms	1610ms

Observations

An important outlier lies in the vectorMultiply.c benchmark with 2 threads. The time is significantly higher than for any other thread value run. This is likely due to my implementation where a low number of threads which are greater than 1 will context switch often, causing plenty of overhead. A potential fix would be to increase the base time quanta for higher priority threads.

Other than that outlier, my library runs faster than the regular library for vectorMultiply.c, which is a great achievement. For the remaining two benchmarks, it seems for externalCal.c my library is x10 slower. For parallelCal.c it is only x2 slower. Considering this is my first attempt at a user thread library all on my own I found this to be acceptable benchmarking.