Group 16

Group Members: Emmy Woods, Chris Cornell, Brandon Roth

The code can be found in Chris Cornell’s account under **/home/f19/p948/MatMul** the read me below is in that directory to aid with building and running the code. There are also executables of the same name that are set to 1000X1000 sized arrays of random numbers.

From The README

*Matrix Multiplication:*

*how to build and run matrix multiplication:*

*the sequential version:*

*gcc mmseq.c -o mmseq*

*./mmseq*

*the OpenMP version:*

*gcc mmomp.c -o mmomp -fopenmp*

*./mmomp*

*the pthread version:*

*gcc mmpthread.c -lpthread -o mmpthread*

*./mmpthread*

*How to test:*

*time ./mmseq*

*time ./mmomp*

*time ./mmpthread*

*The is a constant defined for the size of the matrix called SIZE that needs to be*

*changed before testing. In the pthread version, there's also a constant for the*

*number of threads which can be changed.*

*There is also code for printing matrices that is commented out.*

*Uncomment that to print matrices to check correctness.*

There is a lot to parallelize in this program so there was a large difference in timing between the two. The easiest to write of course, was be the openmp version with the cost of some flexibility with the use of pthreads. The openmp version is the quickest, but the gap could possibly be closed if there was a more dynamic implementation of the pthread version.

In conclusion the process was sped up roughly 80X by implementation of openmp

Below are some examples with a 1000X1000 array:

p948@cs2:~/MatMul$ time ./mmseq

square matrix size: 1000, random numbers under: 10

real 0m6.542s

user 0m6.534s

sys 0m0.008s

p948@cs2:~/MatMul$ time ./mmomp

square matrix size: 1000, random numbers under: 10

real 0m0.687s

user 0m14.644s

sys 0m0.028s

p948@cs2:~/MatMul$ time ./mmpthread

square matrix size: 1000, random numbers under: 10

real 0m0.808s

user 0m16.858s

sys 0m0.032s

* hashing algorithm
  + Both use 4 lower case letters
  + Sequential
    - The sequential algorithm has 4 hardcoded loops that pass permutations of each letter in loop one into the passing function to fund a match to the passed in hash. This is done in a sequential matter. Permutations of ‘a’ being the first letter will be relatively quick but a hash with z as the first letter may take a while to find since it is the last element in the list
  + parallel
    - The parallel version creates a thread for every permutation up to the limit of threads. So with 10 threads, anything with in the first 10 threads will happen very fast but anything after that will be evaluated a lot slower.

example for timing:

(parallel)

p948@cs2:~/MatMul$ time ./ehashp 6385695050

hash input = 6385695050

found!!

The answer is: slow

real 0m6.589s

user 0m0.422s

sys 0m1.351s

(sequential)

p948@cs2:~/MatMul$ time ./ehashs 6385695050

hash input = 6385695050

The answer is: slow

real 1m24.532s

user 0m0.555s

sys 0m1.714s