
// 2D FFT Using threads
// George F. Riley, Georgia Tech, Fall 2009

#include <iostream>

#include "pthread.h"
#include "math.h"
#include <sys/time.h>

#include "complex.h"
#include "InputImage.h"

using namespace std;

// We use global variables in lieu of member variables for this example
Complex** h; // Points to the 2D array of complex (the input)
Complex* W; // Weights (computed once in main
unsigned N; // Number of elements (both width and height)
unsigned nThreads; // Desired number of threads
unsigned activeCount = 0; // Number of active threads

// pthread variables
pthread_mutex_t activeMutex;
pthread_mutex_t exitMutex;
pthread_cond_t exitCondition;
pthread_mutex_t coutMutex;

// Add a verbose flag to turn on/off extra outputs
bool verbose = false;

// Helper routines
void DumpTransformedValues()
{ // Code omitted for brevity
}

void TransposeInPlace(Complex** m, int wh)
{ // code omitted for brevity
}

void LoadWeights()
{ // Compute the needed W values. Omitted for brevity
}

void Transform1D(Complex* h)
{ // The simple 1D transform we did earlier. Code omitted for brevity
}
void* FFT_Thread(void* v)
{
    unsigned long myId = (unsigned long)v; // My thread number
    unsigned rowsPerCPU = N / nThreads;
    unsigned myFirstRow = myId * rowsPerCPU;
    // We have to do a mutex around the "activeCount++". Why?
    pthread_mutex_lock(&activeMutex);
    activeCount++;
    pthread_mutex_unlock(&activeMutex);
    if (verbose)
    {
        pthread_mutex_lock(&coutMutex);
        cout << "MyId is " << myId << " myFirstRow " << myFirstRow << endl;
        pthread_mutex_unlock(&coutMutex);
    }
    // Call the 1D FFT on each row
    for (unsigned i = 0; i < rowsPerCPU; ++i)
    {
        Transform1D(h[myFirstRow + i]);
    }
    // Now notify the main thread we have completed the rows
    pthread_mutex_lock(&exitMutex); // Insure only one thread signals the exit
    pthread_mutex_lock(&activeMutex); // Insure only one thread changes active
    activeCount--;
    pthread_mutex_unlock(&activeMutex);
    if (activeCount == 0)
    { // We are the last thread to exit. Signal the main thread
        // that all threads are done
        pthread_cond_signal(&exitCondition);
    }
    pthread_mutex_unlock(&exitMutex);
    return 0;
}
int main( int argc, char** argv)
{
    verbose = argc > 3;
    InputImage image(argv[1]);
    nThreads = atol(argv[2]); // Number of threads
    N = image.GetHeight(); // Assume square, width = height
    h = image.GetRows(0, N); // In this case, we get all rows

    // Start the timer here, after loading the image
    struct timeval tp;
    gettimeofday(&tp, 0);
    double startSec = tp.tv_sec + tp.tv_usec/1000000.0;

    LoadWeights(); // Only need to do this once

    // Initialize the pthread mutex and condition variables
    pthread_mutex_init(&activeMutex, 0);
    pthread_mutex_init(&exitMutex, 0);
    pthread_cond_init(&exitCondition, 0);
    pthread_mutex_init(&coutMutex, 0);

    // We lock the exitMutex to be sure no threads exit until
    // all threads created, and we are waiting on the condition signal
    pthread_mutex_lock(&exitMutex);
    // Create the threads
    for (unsigned i = 0; i < nThreads; ++i)
    {
        pthread_t t;
        pthread_create(&t, 0, FFT_Thread, (void*)i);
    }
    // Now wait for them to finish pass 1
    pthread_cond_wait(&exitCondition, &exitMutex);
    if (verbose) cout << "All threads finished pass 1" << endl;

    // Transpose the matrix and schedule threads to do rows again
    TransposeInPlace(h, N);
    // Start the threads again
    for (unsigned i = 0; i < nThreads; ++i)
    {
        pthread_t t;
        pthread_create(&t, 0, FFT_Thread, (void*)i);
    }
    // Now wait for them to finish pass 2
    pthread_cond_wait(&exitCondition, &exitMutex);
    if (verbose) cout << "All threads finished pass 2" << endl;

    // Transpose back and write results
    TransposeInPlace(h, N);
    gettimeofday(&tp, 0);
    cout << "Calculated FFT "
         << (tp.tv_sec+tp.tv_usec/1000000.0) - startSec << " seconds" << endl;
    DumpTransformedValues();
}

Program threaded-fft.cc (continued)