CSE-5250 Programming Assignment 2

**Part 1: Matrix Multiplication with OpenMP (30 points)**

Write a program that calculates the product of two matrices, using a parallel for loop. Note that this requires writing a triply-nested for loop, and making one of the for loops parallel.

Source Code:

// part1.cpp

// Nathan Bush - 007463099

#include <iostream>

#include <vector>

#include <omp.h>

// function prototypes

void PrintMatrix(std::vector<std::vector<int>>);

std::vector<std::vector<int>> BuildMatrix(int, bool);

int main()

{

int matrixSize = 10;

// initialize/build matrices

std::vector<std::vector<int>> matrixA = BuildMatrix(matrixSize, false);

std::vector<std::vector<int>> matrixB = BuildMatrix(matrixSize, true);

std::vector<std::vector<int>> resultMatrix(matrixSize, std::vector<int>(matrixSize, 0));

// multiply matrixA and matrixB

for (int i = 0; i < matrixSize; i++) {

#pragma omp parallel for

for (int j = 0; j < matrixSize; j++) {

for (int k = 0; k < matrixSize; k++) {

int value = matrixA[i][k] \* matrixB[k][j];

int thread\_id = omp\_get\_thread\_num();

printf("Thread %i calculated %i as a product.\n", thread\_id, value);

#pragma omp critical

resultMatrix[i][j] += value;

}

}

}

// output results

std::cout << std::endl << "Randomly Generated Matrix:" << std::endl;

PrintMatrix(matrixA);

std::cout << std::endl << "Resulting Matrix After Identity Multiplication:" << std::endl;

PrintMatrix(resultMatrix);

}

// prints a square matrix to the console

void PrintMatrix(std::vector<std::vector<int>> matrix) {

for (int i = 0; i < matrix.size(); i++) {

for (int j = 0; j < matrix[i].size(); j++) {

std::cout << matrix[i][j] << "\t";

}

std::cout << std::endl;

}

}

// returns a square matrix of given size with random ints between 0-99

// if identity == true, returns an identity matrix of given size

std::vector<std::vector<int>> BuildMatrix(int size, bool identity) {

srand(time(NULL));

std::vector<std::vector<int>> matrix(size, std::vector<int>(size, 0));

// build identity matrix

if (identity) {

for (int i = 0; i < size; i++) {

for (int j = 0; j < size; j++) {

if (i == j) {

matrix[i][j] = 1;

}

}

}

}

// build random matrix

else {

for (int i = 0; i < size; i++) {

for (int j = 0; j < size; j++) {

matrix[i][j] = rand() % 100;

}

}

}

return matrix;

}

Screenshot of Output:

A computer screen capture

Description automatically generated with low confidence

**Part 2a: Finding Pythagorean Triples with OpenMP (15 points)**

Write a program that calculates Pythagorean triples using nested for loops. The Pythagorean Theorem states the following:

Where a, b, and c are whole numbers.

Note: We were challenged with enforcing uniqueness of the triples, so I used a set of sets to store the results as shown in the code/output below. Additionally, it appears that multithreading any combination of for loops results in increased execution time. Letting the program run in serial had the best performance, probably due to context switching overhead being greater than the gain in parallelization.

Source Code:

// part2a.cpp

// Nathan Bush - 007463099

#include <iostream>

#include <set>

#include <omp.h>

#include <chrono>

int main()

{

// initialize setup

int maxInt = 100;

int num\_threads = 1;

omp\_set\_num\_threads(num\_threads);

// use sets to enforce uniqueness

std::set<std::set<int>> results;

// start timer

auto start = std::chrono::high\_resolution\_clock::now();

// find triples

#pragma omp parallel for

for (int a = 1; a < maxInt; a++) {

#pragma omp parallel for

for (int b = 1; b < maxInt; b++) {

#pragma omp parallel for

for (int c = 1; c < maxInt; c++) {

if (a \* a + b \* b == c \* c) {

#pragma omp critical

{

std::set<int> triple({ a, b, c });

results.insert(triple);

}

}

}

}

}

// stop timer

auto stop = std::chrono::high\_resolution\_clock::now();

auto exec\_time = std::chrono::duration\_cast<std::chrono::microseconds>(stop - start);

// output results

for (auto i : results) {

std::cout << "Unique Pythagorean Triple Found: ";

for (auto j : i) {

std::cout << j << " ";

}

std::cout << std::endl;

}

std::cout << std::endl << "Found pythagorean triples up to " << maxInt << " in " << exec\_time.count() << " microseconds.\n" << std::endl;

}

Screenshot of Output:

Background pattern

Description automatically generated

**Part 2b: Modifying the Pythagorean Theorem (5 points)**

Modify the Pythagorean Theorem by either adding or subtracting a times b on the left-hand-side, and run the program again. Are there integer solutions for this?

Yes, there are integer solutions as identified in the code and screenshot below. However, these triangles are not right triangles. Interestingly, in the case of , every integer is a solution as an equilateral triangle. That is when a=b, a=b=c. Screenshot and code are for the + equation above, with all scalene, obtuse triangles.

Source Code:

// part2b.cpp

// Nathan Bush - 007463099

#include <iostream>

#include <set>

#include <omp.h>

#include <chrono>

int main()

{

// initialize setup

int maxInt = 100;

int num\_threads = 1;

omp\_set\_num\_threads(num\_threads);

// use sets to enforce uniqueness

std::set<std::set<int>> results;

// start timer

auto start = std::chrono::high\_resolution\_clock::now();

// find triples

#pragma omp parallel for

for (int a = 1; a < maxInt; a++) {

#pragma omp parallel for

for (int b = 1; b < maxInt; b++) {

#pragma omp parallel for

for (int c = 1; c < maxInt; c++) {

if ((a \* a + b \* b) + (a \* b) == c \* c) {

#pragma omp critical

{

std::set<int> triple({ a, b, c });

results.insert(triple);

}

}

}

}

}

// stop timer

auto stop = std::chrono::high\_resolution\_clock::now();

auto exec\_time = std::chrono::duration\_cast<std::chrono::microseconds>(stop - start);

// output results

for (auto i : results) {

std::cout << "Unique Pythagorean Triple Found: ";

for (auto j : i) {

std::cout << j << " ";

}

std::cout << std::endl;

}

std::cout << std::endl << "Found pythagorean triples up to " << maxInt << " in " << exec\_time.count() << " microseconds.\n" << std::endl;

}

Screenshot of Output:

A picture containing table

Description automatically generated