Nathan Bush
007463099

CSE-5250 Programming Assignment 2

Part 1: Matrix Multiplication with OpenMP (30 points)

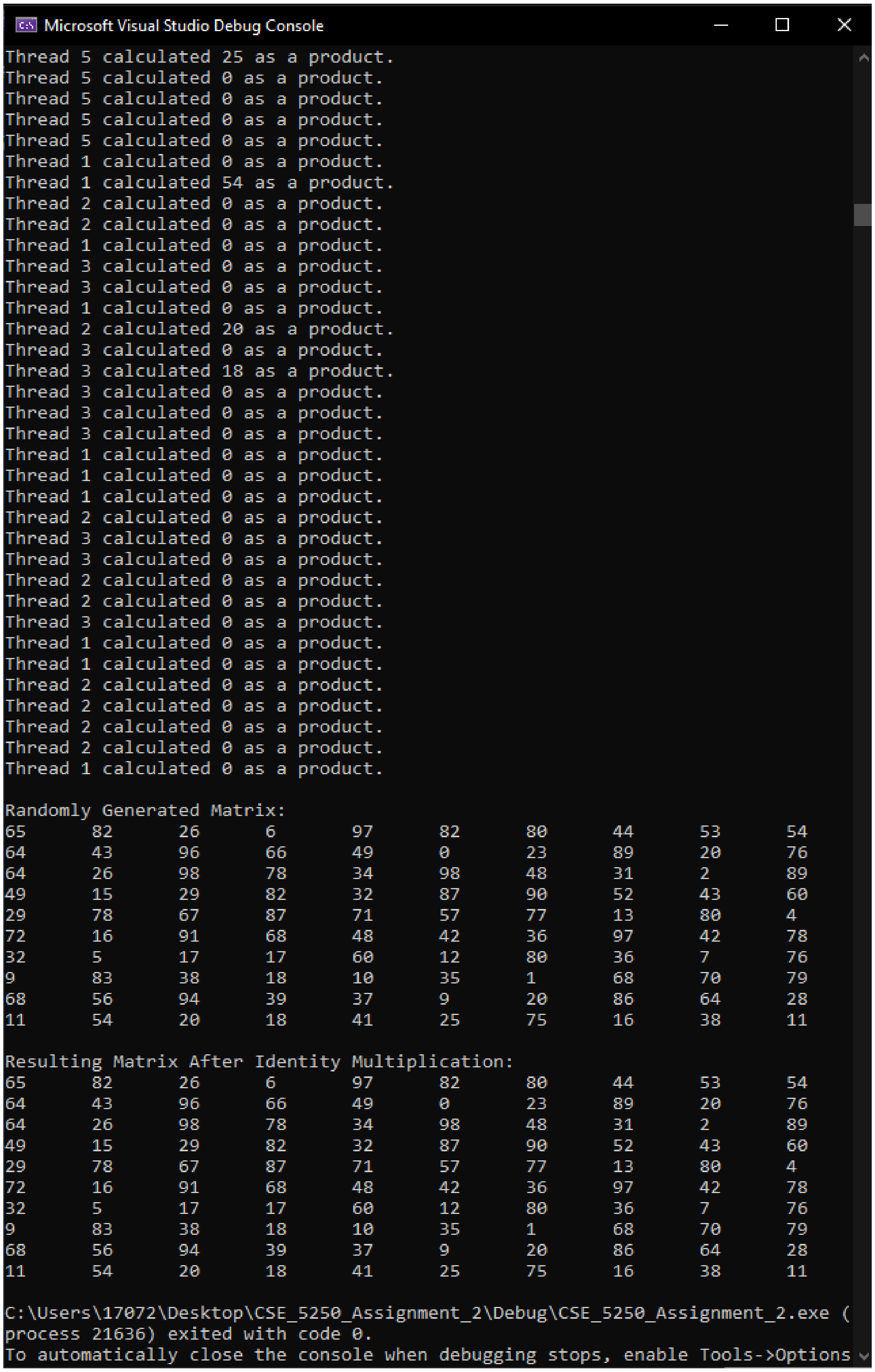
Write a program that calculates the product of two matrices, using a parallel for loop. Note that this requires writing a triply-nested for loop, and making one of the for loops parallel.

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++) {
```

```cpp
        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:



Microsoft Visual Studio Debug Console

```
Thread 5 calculated 25 as a product.
Thread 5 calculated 0 as a product.
Thread 5 calculated 0 as a product.
Thread 5 calculated 0 as a product.
Thread 5 calculated 0 as a product.
Thread 1 calculated 0 as a product.
Thread 1 calculated 54 as a product.
Thread 2 calculated 0 as a product.
Thread 2 calculated 0 as a product.
Thread 1 calculated 0 as a product.
Thread 3 calculated 0 as a product.
Thread 3 calculated 0 as a product.
Thread 1 calculated 0 as a product.
Thread 2 calculated 20 as a product.
Thread 3 calculated 0 as a product.
Thread 3 calculated 18 as a product.
Thread 3 calculated 0 as a product.
Thread 3 calculated 0 as a product.
Thread 3 calculated 0 as a product.
Thread 1 calculated 0 as a product.
Thread 1 calculated 0 as a product.
Thread 1 calculated 0 as a product.
Thread 2 calculated 0 as a product.
Thread 3 calculated 0 as a product.
Thread 3 calculated 0 as a product.
Thread 2 calculated 0 as a product.
Thread 2 calculated 0 as a product.
Thread 3 calculated 0 as a product.
Thread 1 calculated 0 as a product.
Thread 1 calculated 0 as a product.
Thread 2 calculated 0 as a product.
Thread 2 calculated 0 as a product.
Thread 2 calculated 0 as a product.
Thread 2 calculated 0 as a product.
Thread 1 calculated 0 as a product.

Randomly Generated Matrix:
65      82      26      6       97      82      80      44      53      54
64      43      96      66      49      0       23      89      20      76
64      26      98      78      34      98      48      31      2       89
49      15      29      82      32      87      90      52      43      60
29      78      67      87      71      57      77      13      80      4
72      16      91      68      48      42      36      97      42      78
32      5       17      17      60      12      80      36      7       76
9       83      38      18      10      35      1       68      70      79
68      56      94      39      37      9       20      86      64      28
11      54      20      18      41      25      75      16      38      11

Resulting Matrix After Identity Multiplication:
65      82      26      6       97      82      80      44      53      54
64      43      96      66      49      0       23      89      20      76
64      26      98      78      34      98      48      31      2       89
49      15      29      82      32      87      90      52      43      60
29      78      67      87      71      57      77      13      80      4
72      16      91      68      48      42      36      97      42      78
32      5       17      17      60      12      80      36      7       76
9       83      38      18      10      35      1       68      70      79
68      56      94      39      37      9       20      86      64      28
11      54      20      18      41      25      75      16      38      11

C:\Users\17072\Desktop\CSE_5250_Assignment_2\Debug\CSE_5250_Assignment_2.exe (
process 21636) exited with code 0.
To automatically close the console when debugging stops, enable Tools->Options
```

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:

$$a^2 + b^2 = c^2$$

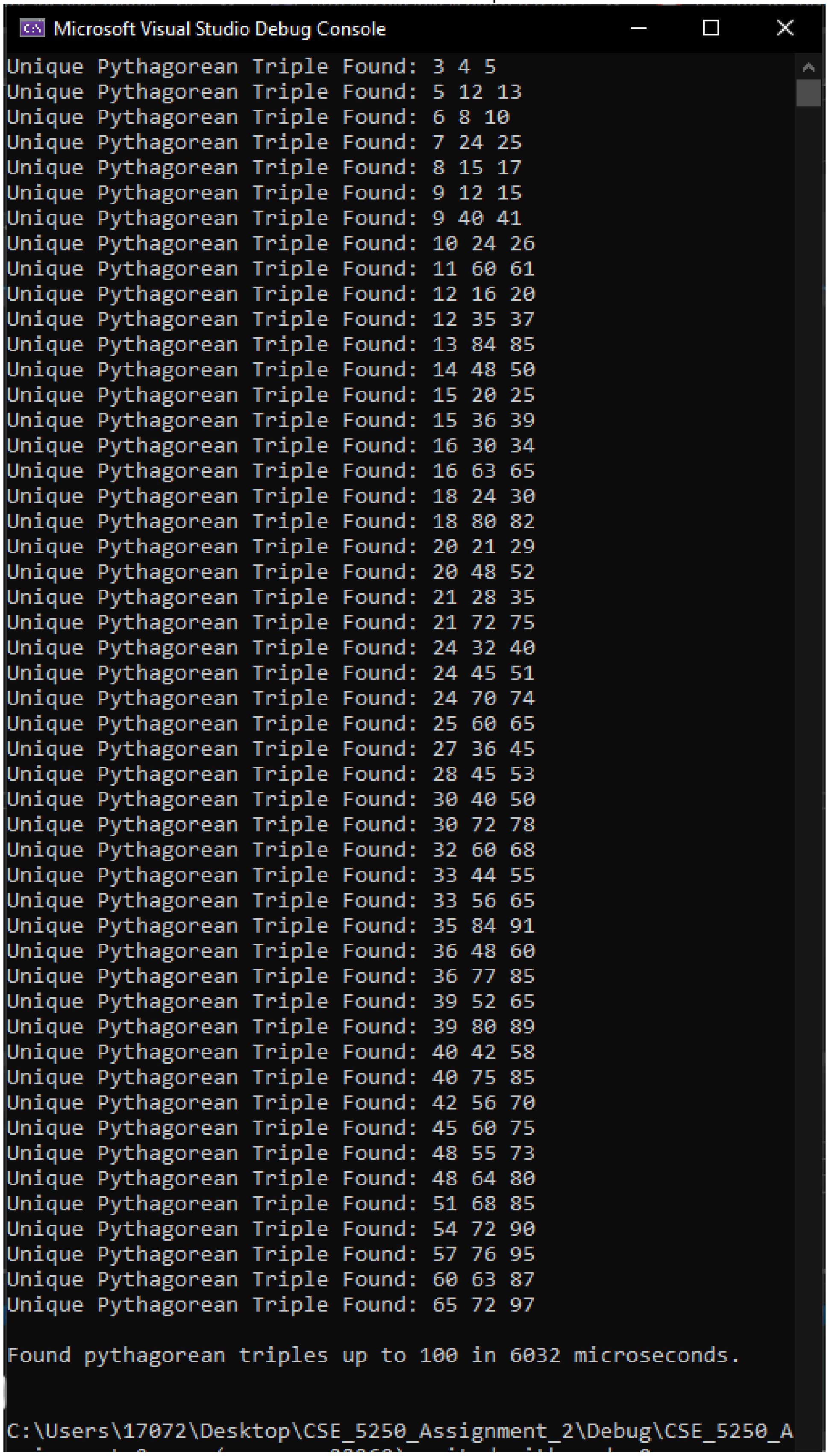
Where a, b, and c are whole numbers.

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:


Microsoft Visual Studio Debug Console

```
Unique Pythagorean Triple Found: 3 4 5
Unique Pythagorean Triple Found: 5 12 13
Unique Pythagorean Triple Found: 6 8 10
Unique Pythagorean Triple Found: 7 24 25
Unique Pythagorean Triple Found: 8 15 17
Unique Pythagorean Triple Found: 9 12 15
Unique Pythagorean Triple Found: 9 40 41
Unique Pythagorean Triple Found: 10 24 26
Unique Pythagorean Triple Found: 11 60 61
Unique Pythagorean Triple Found: 12 16 20
Unique Pythagorean Triple Found: 12 35 37
Unique Pythagorean Triple Found: 13 84 85
Unique Pythagorean Triple Found: 14 48 50
Unique Pythagorean Triple Found: 15 20 25
Unique Pythagorean Triple Found: 15 36 39
Unique Pythagorean Triple Found: 16 30 34
Unique Pythagorean Triple Found: 16 63 65
Unique Pythagorean Triple Found: 18 24 30
Unique Pythagorean Triple Found: 18 80 82
Unique Pythagorean Triple Found: 20 21 29
Unique Pythagorean Triple Found: 20 48 52
Unique Pythagorean Triple Found: 21 28 35
Unique Pythagorean Triple Found: 21 72 75
Unique Pythagorean Triple Found: 24 32 40
Unique Pythagorean Triple Found: 24 45 51
Unique Pythagorean Triple Found: 24 70 74
Unique Pythagorean Triple Found: 25 60 65
Unique Pythagorean Triple Found: 27 36 45
Unique Pythagorean Triple Found: 28 45 53
Unique Pythagorean Triple Found: 30 40 50
Unique Pythagorean Triple Found: 30 72 78
Unique Pythagorean Triple Found: 32 60 68
Unique Pythagorean Triple Found: 33 44 55
Unique Pythagorean Triple Found: 33 56 65
Unique Pythagorean Triple Found: 35 84 91
Unique Pythagorean Triple Found: 36 48 60
Unique Pythagorean Triple Found: 36 77 85
Unique Pythagorean Triple Found: 39 52 65
Unique Pythagorean Triple Found: 39 80 89
Unique Pythagorean Triple Found: 40 42 58
Unique Pythagorean Triple Found: 40 75 85
Unique Pythagorean Triple Found: 42 56 70
Unique Pythagorean Triple Found: 45 60 75
Unique Pythagorean Triple Found: 48 55 73
Unique Pythagorean Triple Found: 48 64 80
Unique Pythagorean Triple Found: 51 68 85
Unique Pythagorean Triple Found: 54 72 90
Unique Pythagorean Triple Found: 57 76 95
Unique Pythagorean Triple Found: 60 63 87
Unique Pythagorean Triple Found: 65 72 97

Found pythagorean triples up to 100 in 6032 microseconds.


C:\Users\17072\Desktop\CSE_5250_Assignment_2\Debug\CSE_5250_A
```

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?

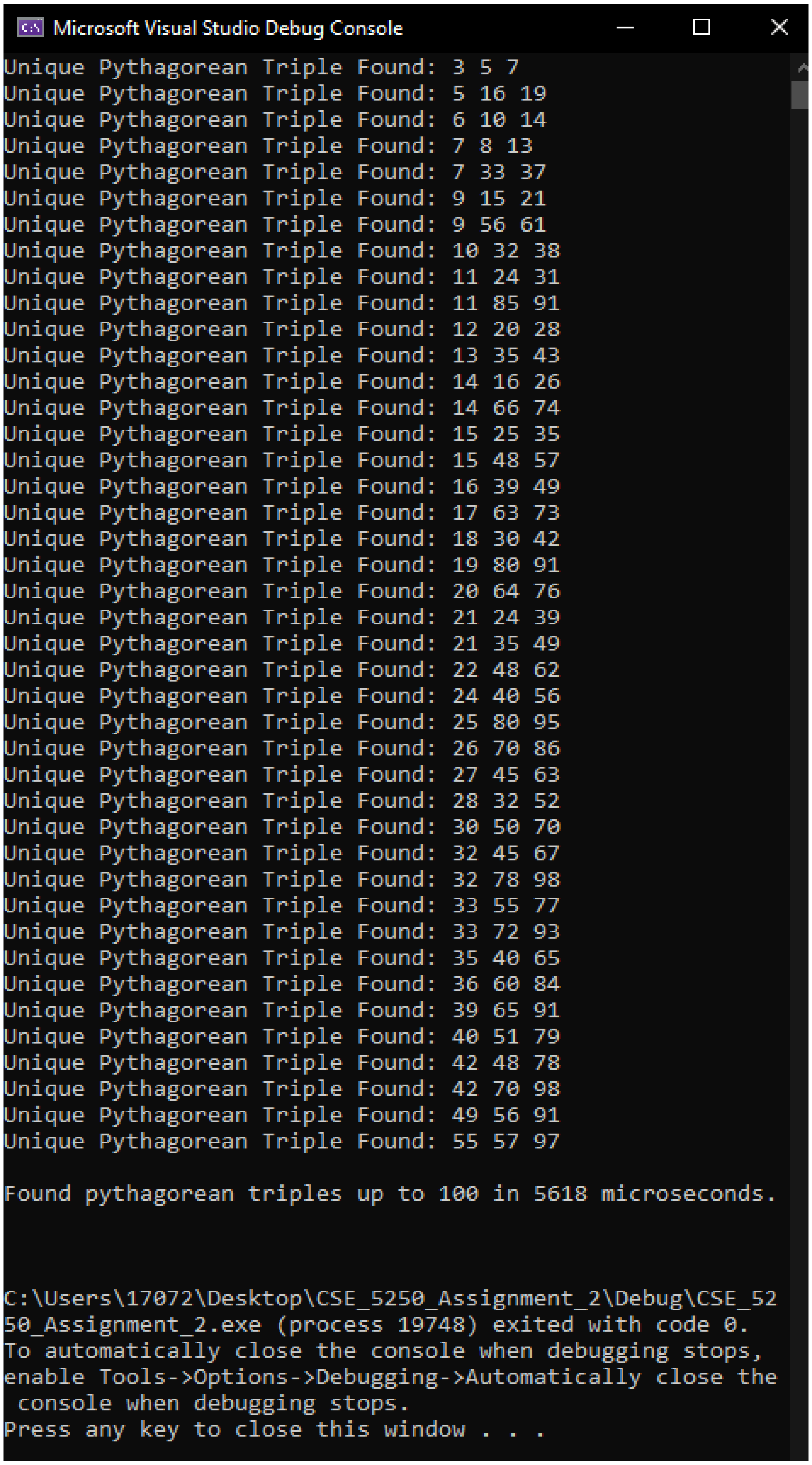
$$a^2 + b^2 + ab = c^2$$

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 $a^2 + b^2 - ab = c^2$, 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:



Microsoft Visual Studio Debug Console

```
Unique Pythagorean Triple Found: 3 5 7
Unique Pythagorean Triple Found: 5 16 19
Unique Pythagorean Triple Found: 6 10 14
Unique Pythagorean Triple Found: 7 8 13
Unique Pythagorean Triple Found: 7 33 37
Unique Pythagorean Triple Found: 9 15 21
Unique Pythagorean Triple Found: 9 56 61
Unique Pythagorean Triple Found: 10 32 38
Unique Pythagorean Triple Found: 11 24 31
Unique Pythagorean Triple Found: 11 85 91
Unique Pythagorean Triple Found: 12 20 28
Unique Pythagorean Triple Found: 13 35 43
Unique Pythagorean Triple Found: 14 16 26
Unique Pythagorean Triple Found: 14 66 74
Unique Pythagorean Triple Found: 15 25 35
Unique Pythagorean Triple Found: 15 48 57
Unique Pythagorean Triple Found: 16 39 49
Unique Pythagorean Triple Found: 17 63 73
Unique Pythagorean Triple Found: 18 30 42
Unique Pythagorean Triple Found: 19 80 91
Unique Pythagorean Triple Found: 20 64 76
Unique Pythagorean Triple Found: 21 24 39
Unique Pythagorean Triple Found: 21 35 49
Unique Pythagorean Triple Found: 22 48 62
Unique Pythagorean Triple Found: 24 40 56
Unique Pythagorean Triple Found: 25 80 95
Unique Pythagorean Triple Found: 26 70 86
Unique Pythagorean Triple Found: 27 45 63
Unique Pythagorean Triple Found: 28 32 52
Unique Pythagorean Triple Found: 30 50 70
Unique Pythagorean Triple Found: 32 45 67
Unique Pythagorean Triple Found: 32 78 98
Unique Pythagorean Triple Found: 33 55 77
Unique Pythagorean Triple Found: 33 72 93
Unique Pythagorean Triple Found: 35 40 65
Unique Pythagorean Triple Found: 36 60 84
Unique Pythagorean Triple Found: 39 65 91
Unique Pythagorean Triple Found: 40 51 79
Unique Pythagorean Triple Found: 42 48 78
Unique Pythagorean Triple Found: 42 70 98
Unique Pythagorean Triple Found: 49 56 91
Unique Pythagorean Triple Found: 55 57 97

Found pythagorean triples up to 100 in 5618 microseconds.



C:\Users\17072\Desktop\CSE_5250_Assignment_2\Debug\CSE_52
50_Assignment_2.exe (process 19748) exited with code 0.
To automatically close the console when debugging stops,
enable Tools->Options->Debugging->Automatically close the
 console when debugging stops.
Press any key to close this window . . .
```