Abu Butt
CSC 432
June 20, 2017
Prof. Izidor Gertner

Dot Product:

In this part, we will be doing a dot product of 2 vectors. I have written a C++ code to generate the vectors. After that we need to use Query Performance to check the timing of the code. Then we will disable the Qpar (Automatic parallelization) and Arch (Automatic vectorization) . The size of the vector is 10,000. We will see the difference of the performance before and after the parallelization.

With out Qpar and Arch:

C++ Code:

```
#include <iostream>
#include <tchar.h>
#include <windows.h>

using namespace std;
double dot_product(double x[], double y[], int size);
int main()
{
	double x[10000];
	double y[10000];
	int size = 100;
	_int64 ctr1 = 0, ctr2 = 0, freq = 0;
	int acc = 0, i = 0;

	for (int i = 0; i < 10000; i++)
	{
		x[i] = i*i*i*1.95;
		y[i] = i*1.8;
	}

	if (QueryPerformanceCounter((LARGE_INTEGER *)&ctr1) != 0)
	{
		dot_product(x, y, 10000);
		// Code segment is being timed.
		//multiplication(matrix, inVector, outVector, size);
		// Finish timing the code.
		QueryPerformanceCounter((LARGE_INTEGER *)&ctr2);
		std::cout << "Start " << ctr1 << std::endl;
		std::cout << "End " << ctr2 << std::endl;
		QueryPerformanceFrequency((LARGE_INTEGER *)&freq);
		std::cout << "ctr1 - ctr2 = " << ctr1 - ctr2 << endl;
		std::cout << "QueryPerformanceCounter minimum resolution : 1/ " << freq <<
"seconds" << std::endl;
```

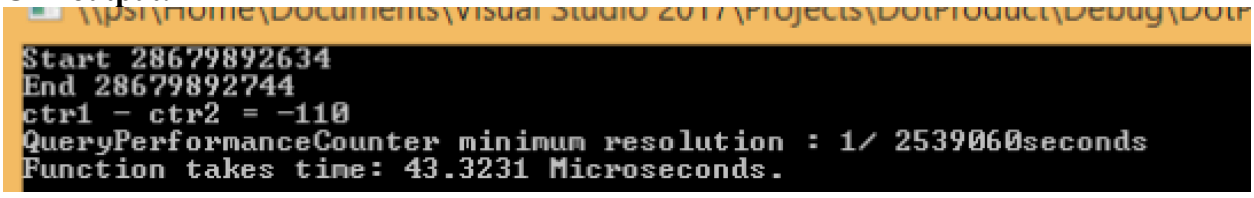
```
		std::cout << "Function takes time: " << ((ctr2 - ctr1) * 1.0 / freq) * 1000000 << "
Microseconds." << std::endl;
		std::cout << endl;
		}
	else
	{
		DWORD dwError = GetLastError();
		std::cout << "Error vaflue = " << dwError << std::endl;
	}
	system("PAUSE");

	return 0;
}
double dot_product(double x[], double y[], int size)
{
	double r = 0.0;
	for (int i = 0; i < size; i++)
	{
		r += x[i] * y[i];
	}
	return r;
}
```

C++ output:

With Qpar and Arch:

I enabled Qpar and for Arch I put (AVX2) and ran the code again.

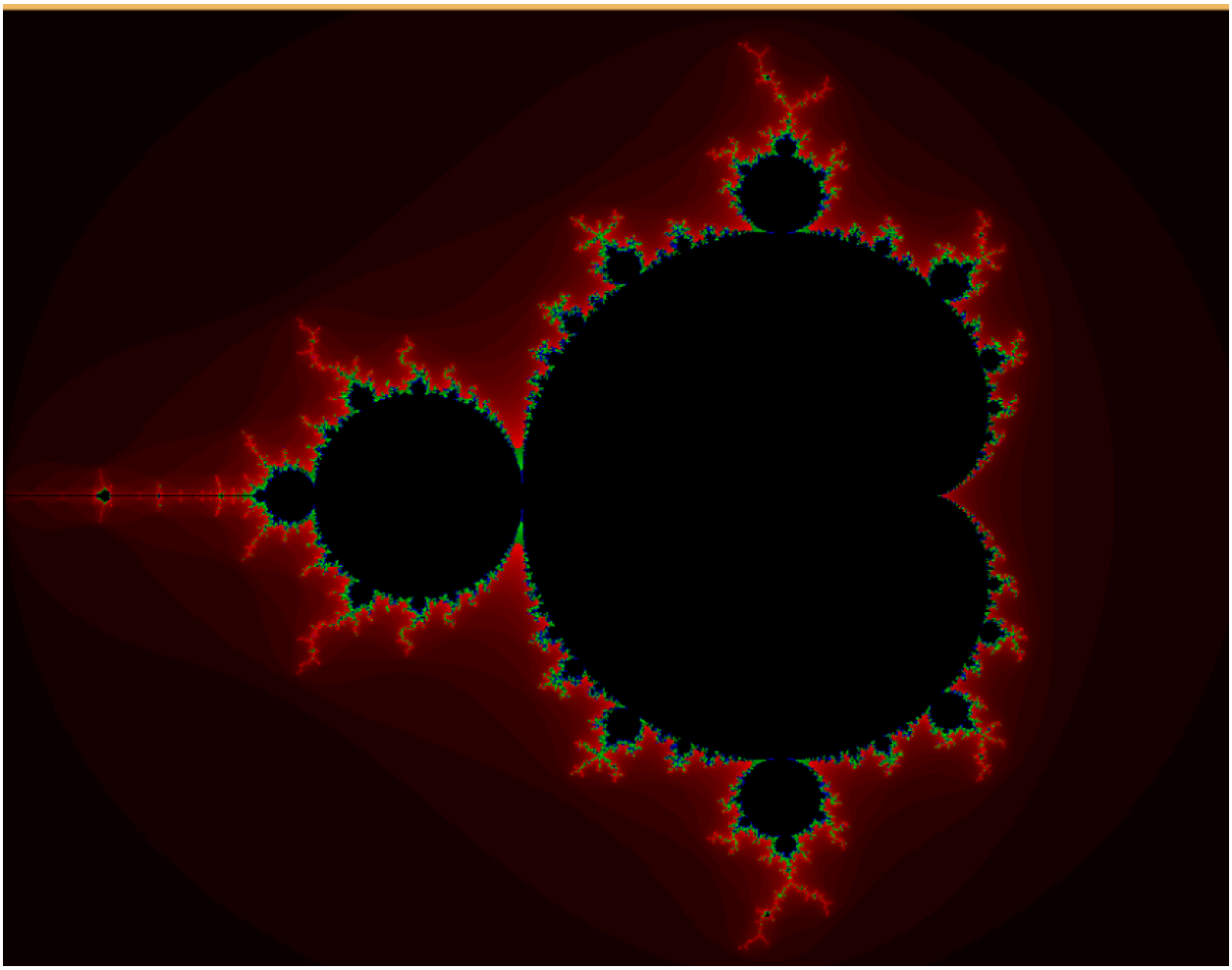
C++ output:



As you can see in the both figures the timing of the function. Without the parallelization, it takes more time for the function to computer the results comparing with parallelization.

Mandelbrot Set:

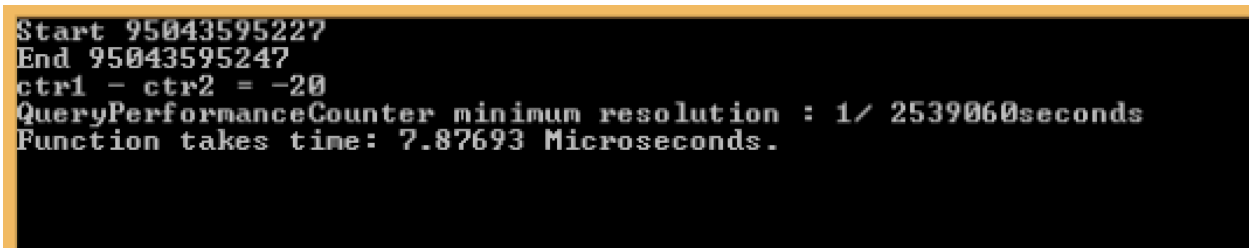
Mandelbrot set is the set of complex numbers c for which the function $f_c(z) = z^2 + c$ does not diverge when iterated from z = 0, remains bounded in absolute value. Mandelbrot set images may be created by sampling the complex numbers and determining for each sample point c, where the result of iterating the above function goes to infinity. This section is about writing and drawing a Mandelbrot set in C++ and using visual studio. After writing the Mandelbrot code and importing SFML library into the visual studio. Following is the image that is produced for Mandelbrot set:



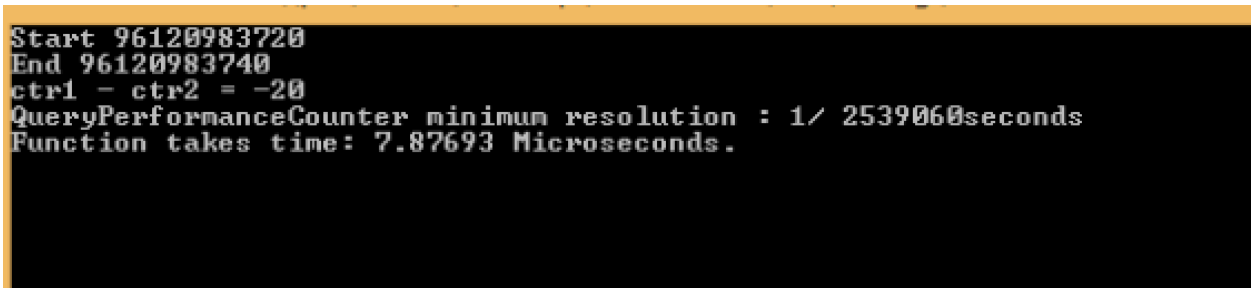
The following diagrams have the query performance for the Mandelbrot set.

```
Start 94385839965
End 94385839983
ctr1 - ctr2 = -18
QueryPerformanceCounter minimum resolution : 1/ 2539060seconds
Function takes time: 7.08924 Microseconds.
```

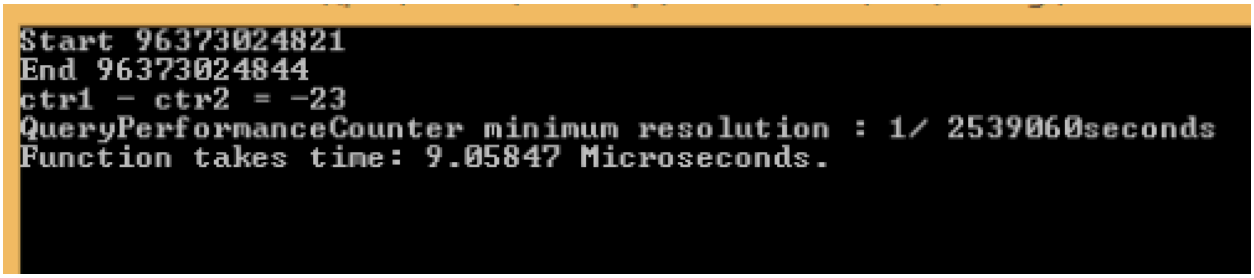
The query performance of the above picture is 7.08924 microseconds. This query performance was done when Qpar and vectorization were disabled.



The query performance of the above picture is 7.87693 microseconds. This query performance was done when Qpar was No and for vectorization I used SSE2. This performance took a little more time since the Mandelbrot set needed to be vectorized.



The query performance of the above picture is 7.87693 microseconds. This query performance was done when Qpar was enable and for vectorization I used AVX2. It has same query performance as the above picture.



The query performance of the above picture is 9.05847 microseconds. This query performance was done when Qpar was enable and for vectorization I used AVX. This performance took the most time for the Mandelbrot set to be vectorized.

Conclusion:

Since Mandelbrot set is not an easy problem to solve. It requires and takes many iterations to display an image on the screen and it is accomplished by using complex numbers. To vectorized the program, it requires more time since it takes more inputs to process at each iteration. Overall, it was a good assignment to work on.