CS205 C/ C++ Programming - Computing the dot product of two vectors

Name: Bryan Anthony

SID: 12112230

Part 1 - Analysis

The problem is to compute the dot product of two vectors using cpp and java. And make a comparison afterward. I did the algorithm using the usual way of the computing dot product which is

$$A.B=Ax*Bx + Ay*By$$

We consider A and B as an array. I created a random float value by generating a one-digit integer and I make another one-digit integer and divide these two numbers together to create a random float value. After that, I store the random value in the array. The vector length will increase as the array length increases.

When we compile the code in cpp we use many different approaches. That is –O3, -O2, -Ofast, and the normal one.

The specifications of my computer to compile this are:
OS: Windows 10 pro – 64 bit
Processor: AMD A8-7410 APU with AMD Radeon R5 Graphics
Memory: 8192MB

Part 2 – Code

CPP code

```cpp
#include <cmath>
#include <chrono>
#include <iostream>
#include <vector>
using namespace std;
using namespace std::chrono;

int main(int argc, char const *argv[])
{
    srand((float)time(NULL));
    for (int n = 1; n < 1000000000; n *= 10)
    {
        float *vec1 = (float *)malloc(n * sizeof(float));
```

```
        float *vec2 = (float *)malloc(n * sizeof(float));
        for (int i = 0; i < n; i++)
        {
            vec1[i] = (float)(1 + rand() % 9) / (float)(1 + rand() % 9);
            vec2[i] = (float)(1 + rand() % 9) / (float)(1 + rand() % 9);
        }
        auto start = high_resolution_clock::now();
        float total = 0;

        for (int i = 0; i < n; i++)
        {
            total += vec1[i] * vec2[i];
        }
        auto stop = high_resolution_clock::now();
        auto duration = duration_cast<nanoseconds>(stop - start);
        float lenghtV1 = 0;
        for (int i = 0; i < n; i++)
        {
            lenghtV1 += vec1[i] * vec1[i];
        }
        float lenghtV2 = 0;
        for (int i = 0; i < n; i++)
        {
            lenghtV2 += vec2[i] * vec2[i];
        }
        free(vec1);
        free(vec2);
        lenghtV1 = sqrt(lenghtV1);
        lenghtV2 = sqrt(lenghtV2);

        cout << "Vector1 with lenght: " << lenghtV1 << " and "
             << "Vector2 with lenght: " << lenghtV2 << " The dot product is " <<
total
             << " takes time " << duration.count() << " nanoseconds" << endl;
    }

    return 0;
}
```

Java code

```
import java.util.Random;

public class Main {
```

```
    public static void main(String[] args) throws Exception {
        for (int n = 1; n < 1000000000; n *= 10) {
            float vec1[] = new float[n];
            float vec2[] = new float[n];
            Random r = new Random();
            for (int i = 0; i < n; i++) {
                vec1[i] = 1 + r.nextFloat() * (9);
                vec2[i] = 1 + r.nextFloat() * (9);
            }
            long start = System.nanoTime();
            float total = 0;
            for (int i = 0; i < n; i++) {
                total += vec1[i] * vec2[i];
            }
            long stop = System.nanoTime();
            float lenghtV1 = 0;
            for (int i = 0; i < n; i++) {
                lenghtV1 += vec1[i] * vec1[i];
            }

            float lenghtV2 = 0;
            for (int i = 0; i < n; i++) {
                lenghtV2 += vec2[i] * vec2[i];
            }

            lenghtV1 = (float) Math.sqrt(lenghtV1);
            lenghtV2 = (float) Math.sqrt(lenghtV2);

            System.out.printf(
                    "Vector1 with lenght: %f and Vector2 with lenght: %f the
result is %f takes time %d nanoseconds\n",
                    lenghtV1, lenghtV2, total,
                    (stop - start));
        }
    }
}
```

Part 3 – Result & Verification

java

```
Vector1 with lenght: 1.721911 and Vector2 with lenght: 1.111261 the result is 1.913492 takes time 2000 nanoseconds
Vector1 with lenght: 19.954735 and Vector2 with lenght: 18.603605 the result is 268.713989 takes time 1600 nanoseconds
Vector1 with lenght: 55.584053 and Vector2 with lenght: 57.822968 the result is 2500.946289 takes time 6900 nanoseconds
Vector1 with lenght: 190.319122 and Vector2 with lenght: 194.660065 the result is 30673.953125 takes time 53600 nanoseconds
Vector1 with lenght: 609.593994 and Vector2 with lenght: 607.377991 the result is 302861.656250 takes time 6475000 nanoseconds
Vector1 with lenght: 1924.263184 and Vector2 with lenght: 1924.716553 the result is 3029574.250000 takes time 8345500 nanoseconds
Vector1 with lenght: 6080.073242 and Vector2 with lenght: 6083.731445 the result is 30243372.000000 takes time 17010200 nanoseconds
Vector1 with lenght: 19121.111328 and Vector2 with lenght: 19125.455078 the result is 301163936.000000 takes time 20869900 nanoseconds
Vector1 with lenght: 46340.949219 and Vector2 with lenght: 46340.949219 the result is 1784091264.000000 takes time 198656500 nanoseconds
```

g++ main.cpp

```
Vector1 with lenght: 1.5 and Vector2 with lenght: 1.16667 The dot product is 1.75 takes time 400 nanoseconds
Vector1 with lenght: 12.0181 and Vector2 with lenght: 6.04445 The dot product is 52.3113 takes time 200 nanoseconds
Vector1 with lenght: 24.5193 and Vector2 with lenght: 24.071 The dot product is 294.267 takes time 800 nanoseconds
Vector1 with lenght: 74.8705 and Vector2 with lenght: 72.6513 The dot product is 2775.12 takes time 6300 nanoseconds
Vector1 with lenght: 237.005 and Vector2 with lenght: 232.933 The dot product is 25356.4 takes time 65000 nanoseconds
Vector1 with lenght: 738.205 and Vector2 with lenght: 737.242 The dot product is 248054 takes time 678900 nanoseconds
Vector1 with lenght: 2327.25 and Vector2 with lenght: 2320.7 The dot product is 2.46185e+06 takes time 7257700 nanoseconds
Vector1 with lenght: 7228.07 and Vector2 with lenght: 7229.96 The dot product is 2.41646e+07 takes time 286851600 nanoseconds
Vector1 with lenght: 21318.9 and Vector2 with lenght: 21317.9 The dot product is 1.73604e+08 takes time 1319583400 nanoseconds
```

g++ main.cpp –O3

```
Vector1 with lenght: 2 and Vector2 with lenght: 3 The dot product is 6 takes time 500 nanoseconds
Vector1 with lenght: 3.6841 and Vector2 with lenght: 10.6831 The dot product is 18.0456 takes time 200 nanoseconds
Vector1 with lenght: 25.4992 and Vector2 with lenght: 23.1849 The dot product is 272.818 takes time 400 nanoseconds
Vector1 with lenght: 71.469 and Vector2 with lenght: 72.4875 The dot product is 2346.56 takes time 2500 nanoseconds
Vector1 with lenght: 232.698 and Vector2 with lenght: 234.038 The dot product is 24746.8 takes time 22800 nanoseconds
Vector1 with lenght: 730.521 and Vector2 with lenght: 738.635 The dot product is 245469 takes time 207400 nanoseconds
Vector1 with lenght: 2320.91 and Vector2 with lenght: 2330.03 The dot product is 2.46943e+06 takes time 1842500 nanoseconds
Vector1 with lenght: 7229.77 and Vector2 with lenght: 7229.17 The dot product is 2.41703e+07 takes time 19215700 nanoseconds
Vector1 with lenght: 21319.6 and Vector2 with lenght: 21322.6 The dot product is 1.73666e+08 takes time 246999400 nanoseconds
```

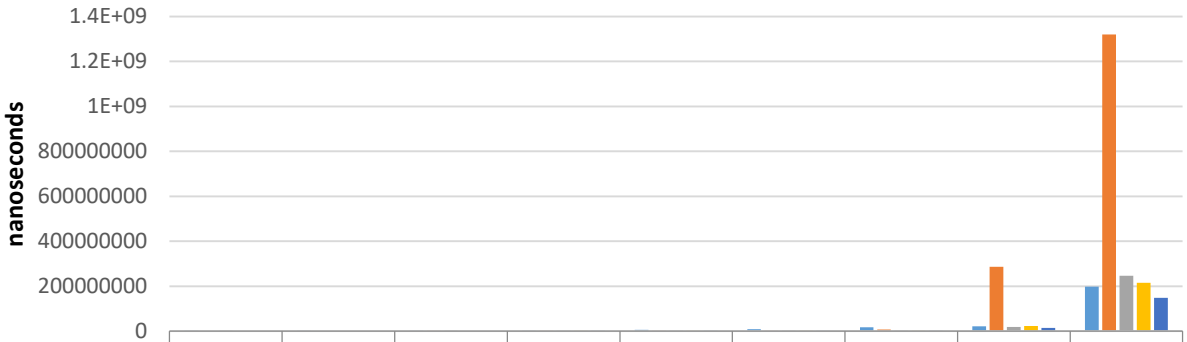
g++ main.cpp –O2

```
Vector1 with lenght: 1.8 and Vector2 with lenght: 0.888889 The dot product is 1.6 takes time 300 nanoseconds
Vector1 with lenght: 3.90207 and Vector2 with lenght: 9.25985 The dot product is 22.7 takes time 300 nanoseconds
Vector1 with lenght: 20.2666 and Vector2 with lenght: 27.2506 The dot product is 217.851 takes time 400 nanoseconds
Vector1 with lenght: 70.8886 and Vector2 with lenght: 73.7509 The dot product is 2138.01 takes time 1900 nanoseconds
Vector1 with lenght: 234.183 and Vector2 with lenght: 230.945 The dot product is 24636.5 takes time 24200 nanoseconds
Vector1 with lenght: 735.44 and Vector2 with lenght: 732.687 The dot product is 246055 takes time 216200 nanoseconds
Vector1 with lenght: 2324.76 and Vector2 with lenght: 2327.14 The dot product is 2.4687e+06 takes time 1959900 nanoseconds
Vector1 with lenght: 7228.33 and Vector2 with lenght: 7224.66 The dot product is 2.41479e+07 takes time 23238200 nanoseconds
Vector1 with lenght: 21322.5 and Vector2 with lenght: 21321.8 The dot product is 1.7362e+08 takes time 214991800 nanoseconds
```

G++ main.cpp –Ofast

```
Vector1 with lenght: 1 and Vector2 with lenght: 3.5 The dot product is 3.5 takes time 400 nanoseconds
Vector1 with lenght: 8.45207 and Vector2 with lenght: 9.75013 The dot product is 21.5667 takes time 200 nanoseconds
Vector1 with lenght: 20.5841 and Vector2 with lenght: 26.3073 The dot product is 245.96 takes time 300 nanoseconds
Vector1 with lenght: 75.9012 and Vector2 with lenght: 72.2858 The dot product is 2458.62 takes time 1500 nanoseconds
Vector1 with lenght: 236.415 and Vector2 with lenght: 231.202 The dot product is 24858.1 takes time 14800 nanoseconds
Vector1 with lenght: 739.119 and Vector2 with lenght: 736.795 The dot product is 247649 takes time 181700 nanoseconds
Vector1 with lenght: 2329.16 and Vector2 with lenght: 2325.67 The dot product is 2.46505e+06 takes time 1597600 nanoseconds
Vector1 with lenght: 7347.16 and Vector2 with lenght: 7347.01 The dot product is 2.46712e+07 takes time 14675200 nanoseconds
Vector1 with lenght: 22393.6 and Vector2 with lenght: 22394.7 The dot product is 2.2721e+08 takes time 147675900 nanoseconds
```

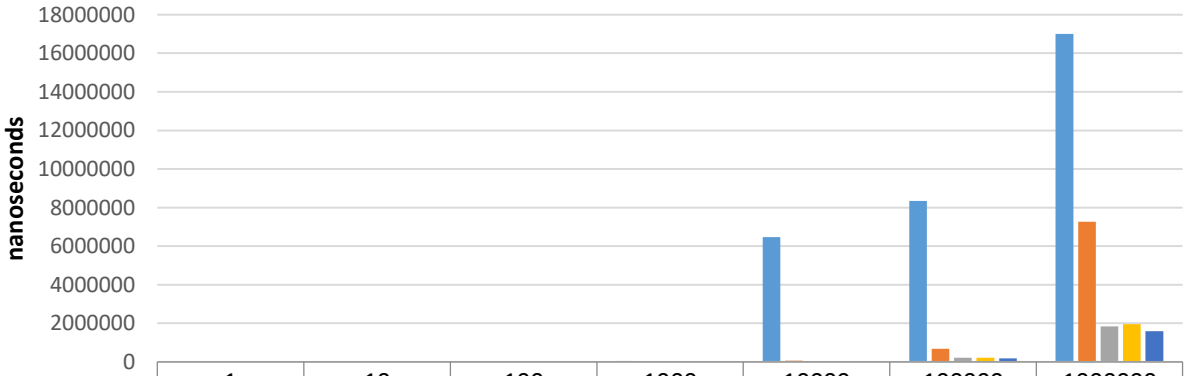
JAVA vs CPP vs CPP03 VS CPP02 VS CPPOfast
length 1 - length 1e9



	1	10	100	1000	10000	100000	1000000	10000000	100000000
java	2000	1600	6900	53600	6475000	8345500	17010200	20869900	1.99E+08
cpp	400	200	800	6300	65000	678900	7257700	2.87E+08	1.32E+09
cppO3	500	400	2500	22800	22800	207400	1842500	19215700	2.47E+08
cppO2	300	300	400	1900	24200	216200	1959900	23238200	2.15E+08
cppOfast	400	200	300	1500	14800	181700	1597600	14675200	1.48E+08

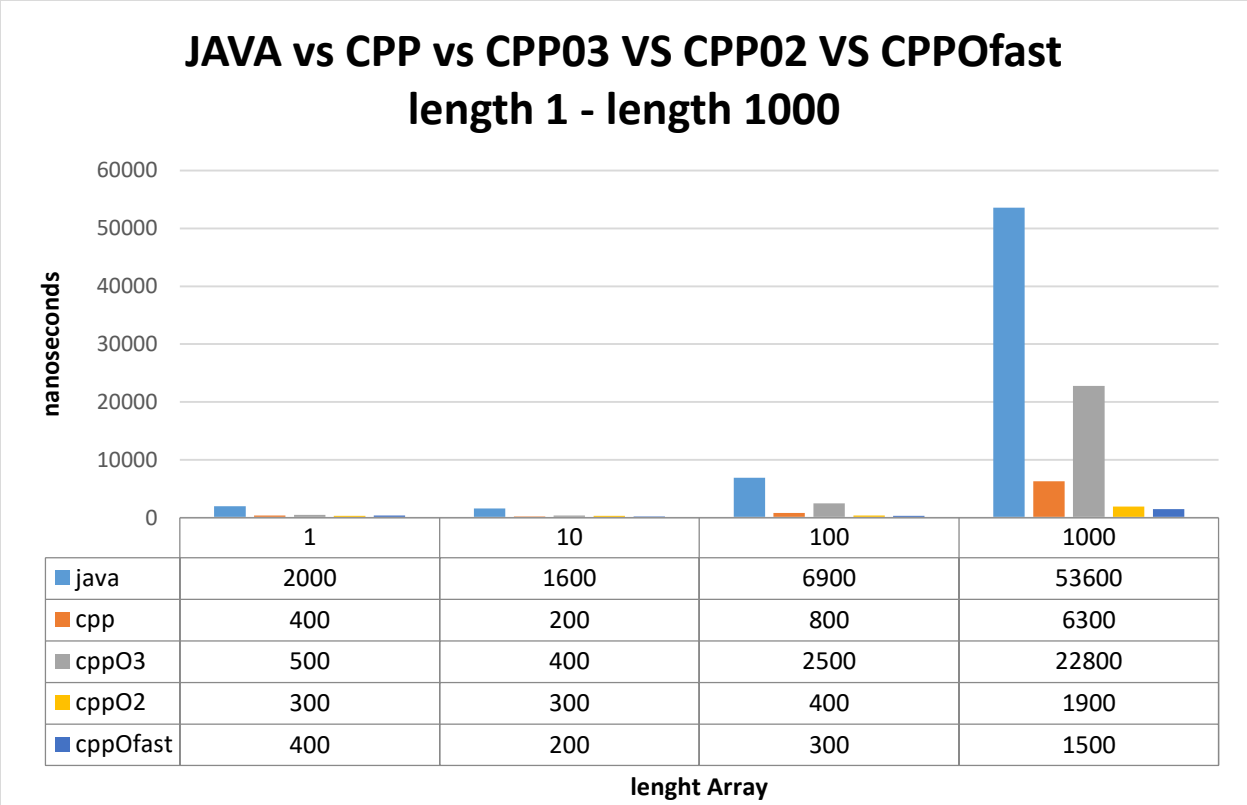
lenght Array

JAVA vs CPP vs CPP03 VS CPP02 VS CPPOfast
length 1 - length 1e6



	1	10	100	1000	10000	100000	1000000
java	2000	1600	6900	53600	6475000	8345500	17010200
cpp	400	200	800	6300	65000	678900	7257700
cppO3	500	400	2500	22800	22800	207400	1842500
cppO2	300	300	400	1900	24200	216200	1959900
cppOfast	400	200	300	1500	14800	181700	1597600

lenght Array



	1	10	100	1000
java	2000	1600	6900	53600
cpp	400	200	800	6300
cppO3	500	400	2500	22800
cppO2	300	300	400	1900
cppOfast	400	200	300	1500

Let’s get into the analysis after the comparison. We first talk about java and the normal cpp compiler. So as we can see at the first 1 to length 1e6 java always becomes the slowest, but as the length of the vector gets bigger, cpp becomes three times slower than java. But as we know we usually have a mindset of cpp is always faster than java but why now java is faster than cpp? This is because some techniques in Java Virtual Machine called JIT stands for “Java-In-Time” compilation. The JIT compiler works by analyzing the bytecode as it is being executed and identifying sections of code that are frequently executed. It then compiles these sections of code into native machine code, which can be executed more quickly in the future.

Overall, JIT compilation can significantly improve the performance of Java programs, especially for long-running applications. This is why when the number getting bigger java faster than cpp, because cpp didn’t identify sections of code that are frequently being executed and optimize it.

Now we bring –O3-O2 -Ofast to the table. As we can see in the histogram –O3 optimization at first is slower than the normal compiler cpp but still faster than java. But when the length of the vector increases –O3 beat the normal cpp but java is still faster than –O3 but the difference is not that big compared to the normal compiler of cpp without optimization. The -O3 flag enables the highest level of optimization in the compiler, which can result in faster and more efficient code.

This is because the compiler is able to perform more aggressive optimizations, such as loop unrolling and function inlining, which can eliminate unnecessary instructions and reduce the overall execution time of the program. However, these optimizations can also make the compilation process slower and use more memory. It's important to note that the performance

gains from using -O3 may not always be significant, and in some cases, it may even result in slower code.

Let’s take a look at –O2 optimization, -O2 optimization seems slower compared to –O3 but at the end when the length is 1e7 –O2 is slightly faster than –O3. But still –O2 cannot beat java in the end.

The -O2 flag enables a lower level of optimization than -O3, but it can still result in faster and more efficient code. This is because the compiler is able to perform some optimizations, such as loop unrolling and function inlining, which can eliminate unnecessary instructions and reduce the overall execution time of the program. However, -O2 is less aggressive than -O3, so it may not be able to optimize the code as much. In some cases, -O2 may even result in slower code than -O3, depending on the specific use case.

Looks like –Ofast is the winner in this optimization. He beat all the other compiler optimization. But how?

The -Ofast flag is a combination of several optimization flags, including -O3, that are designed to maximize performance at the expense of some code safety. Specifically, -Ofast enables optimizations that may violate strict compliance with language standards, such as allowing the reordering of floating-point operations. This can result in faster code, but it can also introduce subtle bugs or inaccuracies in some cases.

Overall, -Ofast can be a good option for performance-critical code that has been thoroughly tested and validated, but it may not be suitable for all types of applications.

Part 4 – Difficulties & solution

1. Random value can only produce an integer. To make a random float value I generate two random integers of one digit and divide them.
2. To Increase the length of the vector I increase the length of the array but the element inside it is generated from two integers of one digit and divided it becomes a float value.
3. In cpp the allocation of memory cannot be disposed automatically so I have to free the memory after using it.