Final Take Home Test

Chue Zhang

Csc343 Fall 2021

Professor Gertner

Due Date : December 15th,2021 by 12:00pm

Contents

[Introduction: 3](#_Toc89822627)

[Objective: 4](#_Toc89822628)

[Specifications: 5](#_Toc89822629)

[Code: 6](#_Toc89822630)

[Build and run: 9](#_Toc89822631)

[Data: 10](#_Toc89822632)

[Unoptimized: 11](#_Toc89822633)

[Optimized: 16](#_Toc89822634)

[Analysis: 20](#_Toc89822635)

[Conclusion: 22](#_Toc89822636)

# Introduction:

This is the final take home test to be done with one computing device by Chue Zhang in the FALL semester of 2021 with professor Izidor Garter belt. My goal of this final take home test is to understand more about runtimes of the doc product and how I can further optimize the dot product to perform more efficiently when working with larger vector sizes.

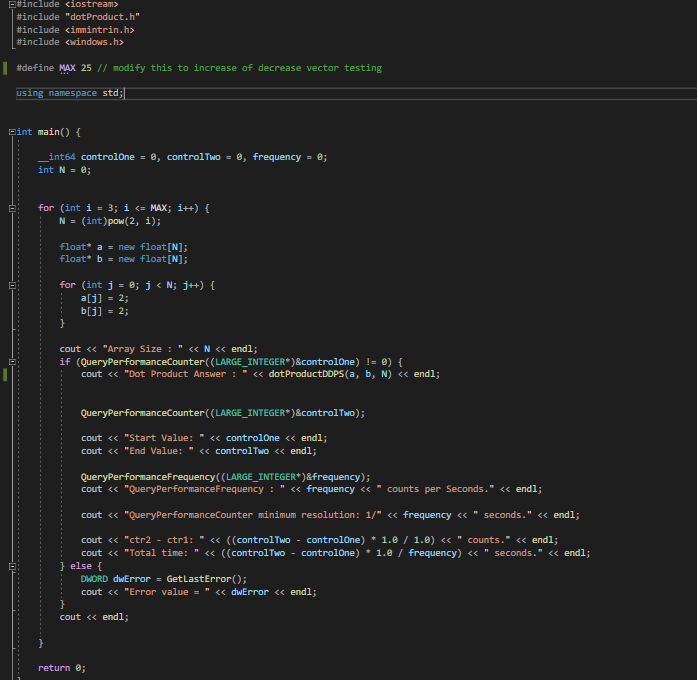
# Objective:

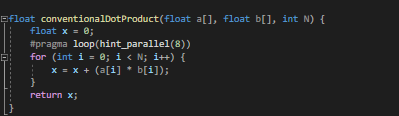
The objective of this final take home test is to show understanding in the topic of optimization on the dot product over a series of different sized arrays. What I will be going through is the different times required to process through 2^n sized arrays and comparing it with different levels of optimization. Some levels of optimizations go by the names of conventional, which in this case, conventional dot product would be something without the use of assembly. Then there is manual dot product where we take advantage of assembly code language and manually allocate memory into registers and process the dot products. Finally, there is DPPS which is the Dot Product of Packed Single Precision Floating-Point Values which selectively multiply values in xmm1 with xmm2 then adds and stores back into xmm1

# Specifications:

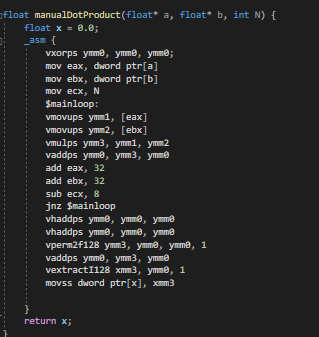
What we will be using to build and run the code is Visual Studios 2019 which will help simulate our Linux environment while also enabling us to use functions such as QueryPerformanceCounter and QueryPerformanceFrequency which is a bit more difficult to access in a Linux Environment. There will be no usage of ARMS processor and MIPS processor and only GCC processor and there will also be use of assembly language to achieve optimization. Line graphs will be presented in order to further explain findings.

# Code:

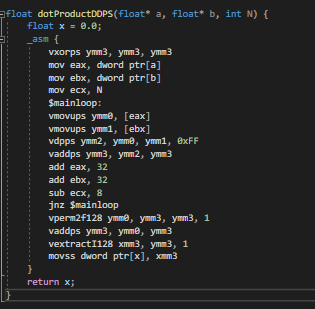
In this screenshot above, what is shown is the main file that we will be using in order to query the performance and the frequency of the code. Furthermore, this code will provide us with the runtime of each iteration of the array sizes. For our test runs we will be running a set of arrays of size {2^1… 2^25}.



Above is the conventional method of performing dot product that we are all used to seeing. No assembly code is written here. The bee, of course, flies anyway because bees don’t care what humans thinks is impossible.

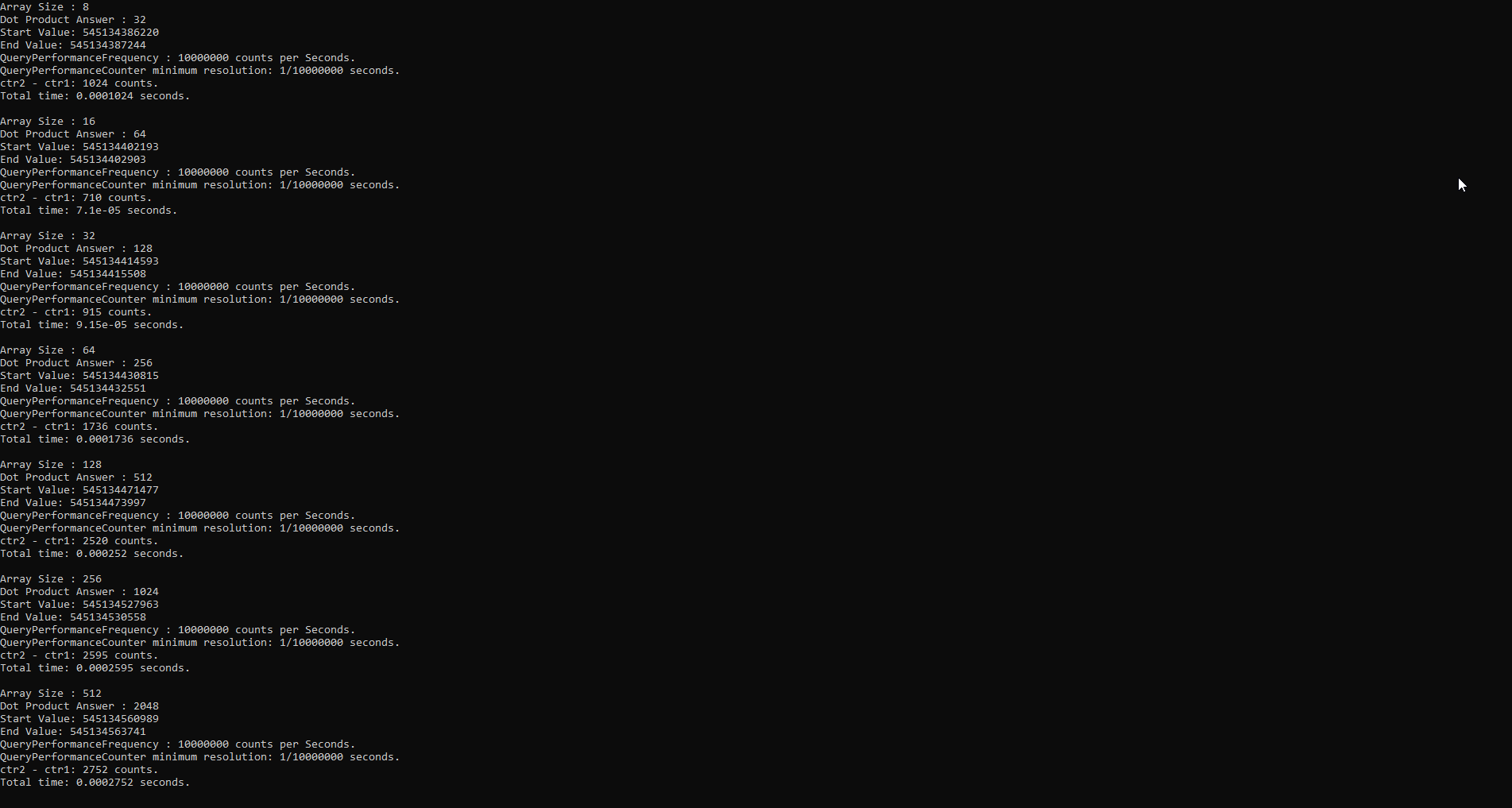


Above is the manual method of performing dot product where we do multiplication on ymm3 and ymm1 then store into ymm2 and we add the result back into ymm0. This process is continuously looped through and done until there are no more array values left to process.



Above is the DPPS dot product which is very similar to the Manual method shown above but instead, we do the vdpps to perform the dot product and we store into ymm1 then add the value into ymm3.

# Build and run:

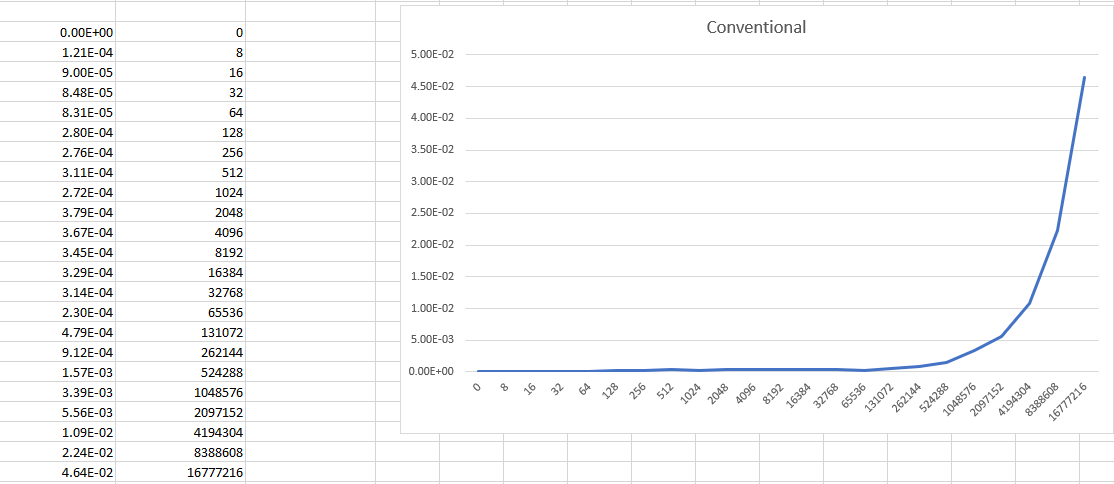


The compiled process ran looks like this for me. As you can see, there is an array size, the dot product answer, Query performance frequencies, Counters, counts and most importantly the run times.

# Data:

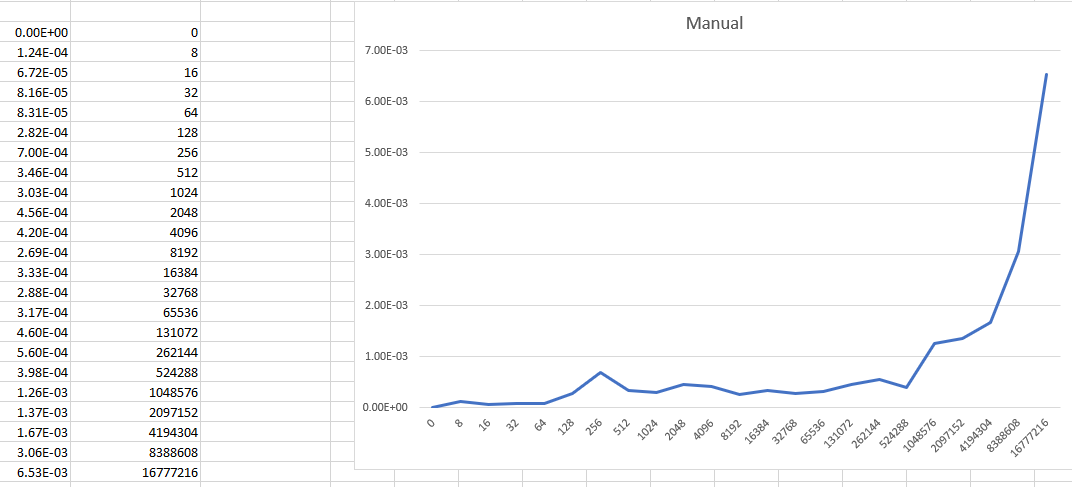
In this section, What I will be going over is the different run times between the conventional dot product method, manual dot product method and the DPPS dot product method over the range of 2^N : N = {1… 25}. In the end, I will be showcasing all three methods lined up together for comparison. There will be two different selections, one with optimized runtime and one without it

## Unoptimized:

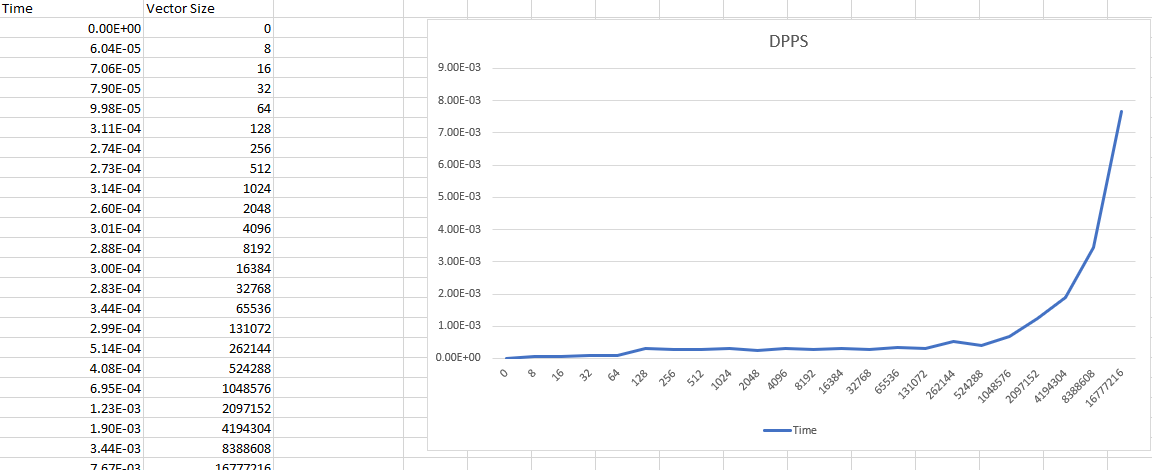


First off is the Conventional Method.

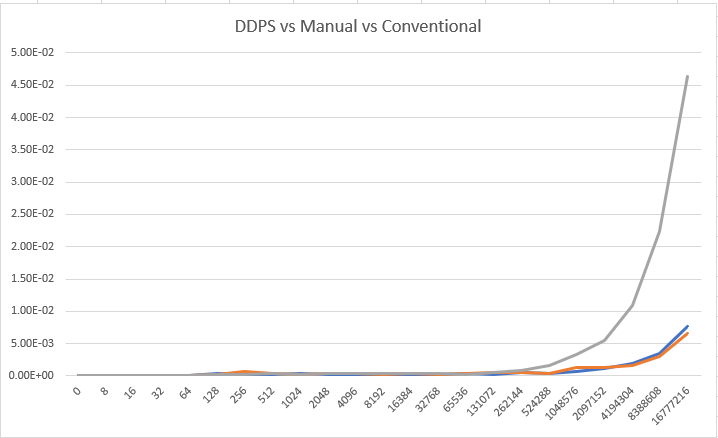
On the left side you can see the data that I have acquired and plotted onto the right side. Upon analysis, there isn’t much significant changes that occur between N = 1 and N = 16 but the runtime s tarts dramatically increasing after 2^17. You can notice the quadratic runtime that as it will keep doubling in runtime over time. However, if we look over in the data, in the range N=1 to N=15, the runtime doesn’t actually change too much and there are many inconsistencies with the runtime but that is okay because the runtime is so small.



In this screenshot above is the data inputted on the right side and the input data on the left side for the Manual method. Immediately we can notice that there are more Inconsistencies in the runtime of the manual method as in 2^8th array size, there is a sudden spike in the runtime. This occurs maybe because of the computer the use is using or maybe because of the IDE that the person is using to compile the code. Regardless of that, we can notice that this graph also inherits a quadratic run time which is correct. We can once again notice the increase in runtime after the N=17th array size.

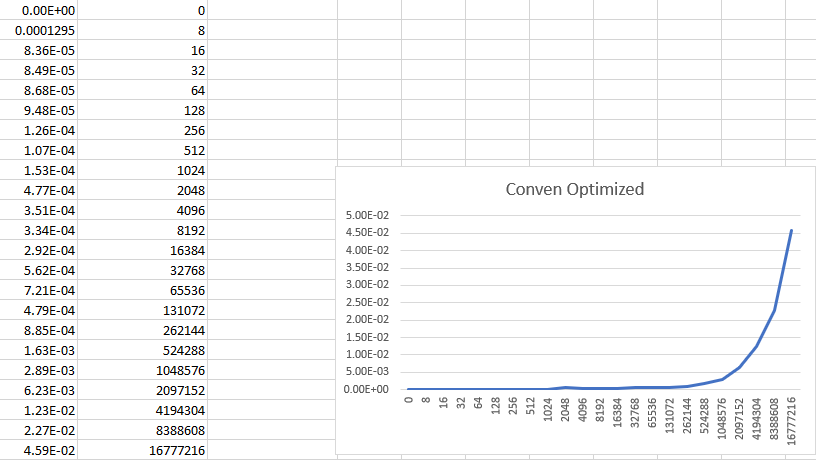


Above is the DPPS data that has been plotted on the right and the input value data on the left. Unlike the manual method of dot product, the runtime is much more consistent and most importantly, inherits a quadratic runtime which we can see in the line graph. Upon analyzing the graph, we can see the first notable time increase at N=9 and the next notable time increase at N=18.



In this line graph above is the comparison between the 3 different methods of computation. The grey line being the Conventional method, the blue line being the DPPS method and the orange line being the Manual method. Immediately we can tell that the conventional method is the most inefficient in computation and when working with big data, the time takes almost 10x as long to compute as the manual method and the DPPS method. The time hike occurs much earlier for the conventional method starting at roughly N=17 whereas the DPPS method and the Manual method starts their time hikes at around N=19, N=20. Furthermore, the time increase also isn’t as drastic as the conventional method and lastly it is important to notice that these array sizes are extremely large. For basic computation that is not being done on big data, conventional method still is okay but different approaches may be necessary on a larger size of file as we can see from above.

## Optimized:

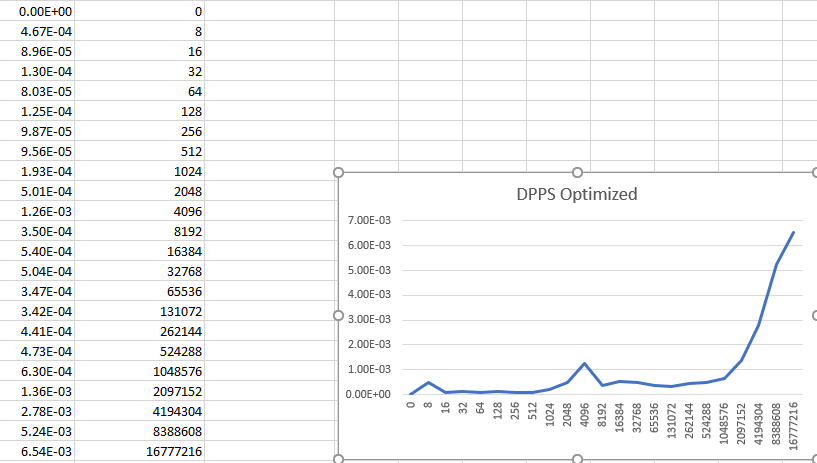


Above is the conventional method of Dot product optimized, not much difference from unoptimized results above. Follows a Quadratic runtime

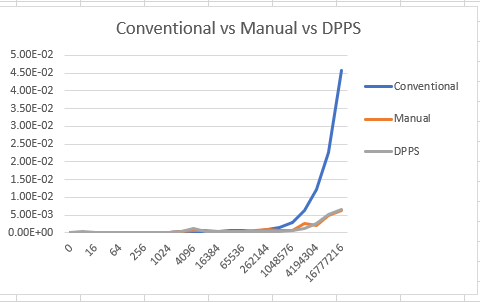
Application, table

Description automatically generated

Optimized manual dot product, not much difference here as well, some fluctuations in time needed to process which is similar to unoptimized manual dot product. Similar in run time as well



Above is the DPPS dot product method where this time we see some random fluctuations of run time but most of it is smooth. Run time is similar to the unoptimized version and has similar results



Above are the optimized runtimes of the conventional vs manual vs dpps method. Similar to the unoptimized version, the dpps and the manual method basically mirror one another, and the conventional method is of course the slowest.

# Analysis:

Text

Description automatically generated

In this screenshot above, I want to talk about the Query Performance Frequency and the Query Performance Counter. As you can see the frequency is 1 million counts per seconds and the total time required is the control 2 – control 1 / Frequency and the Query Performance Counter is calculated by 1/Frequency. Control 1 and Control 2 are the start value and end value respectively and we do a simple calculation of subtraction in order to get the run time although there may be some inconsistencies with the control 1 and control 2 because it may overperform since there are multiple processes occurring.

# Conclusion:

In this Final Take Home Test, I have come to observe that the conventional method of the dot product is highly unoptimized and that the manual method and the DPPS method of the dot product is much more efficient when working with larger data sets. I have also further expanded my knowledge of runtime and how to work with unique functions such as QueryPerformanceFrequency and QueryPerformanceCounter.