CS 342000 | CS343000 - Spring 2022

Instructor: Professor Izidor Gertner

Spring 2022 – 5/15/22

Take Home Test 3 - Azwad Shameem

***OPTIMIZATION OF DOT PRODUCT***

***COMPUTATION OF TWO VECTORS***

***USING VECTOR INSTRUCTIONS***

**Table of Contents**

[Objective 3](#_Toc102918852)

[Solutions: 4](#_Toc102918853)

[Visual Studio 4](#_Toc102918854)

[*CPU-Z* 4](#_Toc102918855)

[*main.cpp* 5](#_Toc102918856)

[*Dot Product* 6](#_Toc102918857)

[*Dot Product with Automatic Parallelization, /Qpar and Automatic Vectorization, /arch* 9](#_Toc102918858)

[*manualDotProduct* 13](#_Toc102918859)

[*DPPSDotProduct* 18](#_Toc102918860)

[*Comparison* 22](#_Toc102918861)

[Linux 23](#_Toc102918862)

[*CPU-X* 23](#_Toc102918863)

[*main.cpp* 24](#_Toc102918864)

[*Dot Product* 25](#_Toc102918865)

[*Dot Product with Automatic Parallelization and Automatic Vectorization* 27](#_Toc102918866)

[*manualDotProduct* 31](#_Toc102918867)

[*DPPSDotProduct* 35](#_Toc102918868)

[*Comparison* 39](#_Toc102918869)

[Conclusion 40](#_Toc102918870)

# 

# Objective

The objective of this take-home test is to optimize the compiler generated code for a

program that computes the dot product using vector instructions. In order to correctly optimize the program that computes dot product, we utilized the QueryPerformanceCounter function to measure execution time, and in order to confirm that the optimization of the assembly code led to decreases in execution time. The optimizations that will be used are the automatic parallelization and vectorization, the compiler generated code with vector instructions and the vector instructions DPPS to improve efficiency of the function. These optimizations will be run and recorded with their execution times which will then be listed all together on one graph to be analyzed. Then we will repeat the previous steps that we did in Visual Studio in Linux.

# Solutions:

## Visual Studio

### *CPU-Z*

Graphical user interface, application

Description automatically generatedFigure 1: CPU-Z CPU information

The processor is the Intel Core i9-9900k which supports the instructions MMX, SSE, SSE2, SSE3, SSSE3, SSE4.1, SSE4.2, EM64T, AES, AVX, AVX2, FMA3. This processor does not support he AVX512 vector instruction test. Furthermore, for this take-home test we will be using the AVX2 vector instruction set, which is supported by the processor.

### Graphical user interface, text, application Description automatically generated*main.cpp*

Figure 2: Main.cpp code shown in Visual Studio

This is the main program file that will measure the time that functions take and also report the other statistics of the function. Some of the statistics are the size of the array, start value, end value. This file right now calls the function dotProduct() in line 24, but this will be changed to the other functions to test out the compiler and DPPS version of the dotProduct function.

### *Dot Product*

Graphical user interface, text, application, Word

Description automatically generated

Figure 3: dotProduct function in the dotProduct.h file

This is the default dotProduct function, this function computes the dotProduct of the two float arrays. This function loops through each of the values in the 2 arrays and multiplies each value at a[i] and b[i] and calculates the overall sum of the multiplication and stores it in the float result and then returns it after the loop is over.

Text

Description automatically generated

Figure 4: The output after running the function

|  |  |
| --- | --- |
| **Vector Size** | **Execution Time** |
| 16 | 0.0006255 |
| 32 | 0.0004292 |
| 64 | 0.0003814 |
| 128 | 0.0003123 |
| 256 | 0.0003332 |
| 512 | 0.0001895 |
| 1024 | 0.000363 |
| 2048 | 0.0012161 |
| 4096 | 0.0006843 |
| 8192 | 0.0006566 |
| 16384 | 0.0010243 |
| 32768 | 0.0012251 |
| 65536 | 0.0024039 |

Figure 5: Table of the dotProduct functions execution times.

Figure 6: Graph of the dotProduct function execution times vs the vector size.

The graph shows that generally as the vector size increases the execution time increases. Next, we will use automatic parallelization and vectorization in order to improve the performance of the dotProduct() and show it in the graph to compare.

### *Dot Product with Automatic Parallelization, /Qpar and Automatic Vectorization, /arch*

Graphical user interface, text, application, email

Description automatically generated

Figure 7: Project Properties Window

The window above in figure 7 shows that now /Qpar and Advanced Vector Extensions 2 /arch:AVX2 is now enabled. Furthermore, we are using the floating point model of precise /fp:precise.

Text

Description automatically generated

Figure 8: Results of the dotProduct() function with automatic parallelization and vectorization enabled.

|  |  |
| --- | --- |
| **Vector Size** | **Execution Time** |
| 16 | 0.0004676 |
| 32 | 0.0003952 |
| 64 | 0.0003338 |
| 128 | 0.000321 |
| 256 | 0.0003043 |
| 512 | 0.0003138 |
| 1024 | 0.0004195 |
| 2048 | 0.0005232 |
| 4096 | 0.0005007 |
| 8192 | 0.000477 |
| 16384 | 0.000693 |
| 32768 | 0.0009483 |
| 65536 | 0.0016342 |

Figure 9: Results of the dotProduct() function with automatic parallelization and vectorization enabled in a table

Figure 10: Results of the dotProduct() function with automatic parallelization and vectorization enabled graphed.

The graph shows that generally as the vector size increases the execution time increases. The graph also shows that the increase is slanted less upwards than the previous graph in figure 6. Automatic parallelization and vectorization may have been the reason why this has improvement occurred.

Graphical user interface, application, Word

Description automatically generatedGraphical user interface, application, Word

Description automatically generatedFigure 11: Compiler generated assembly code

Figure 12: Compiler generated assembly code with Figure 11: Compiler generated assembly code

In lines 2259 to 2262 we can see the compiler makes use of vector instructions which means that automatic parallelization and vectorization was the reason why the dotProduct() function had lower execution time per vector size in figure 10 compared to figure 6. This use of vector instructions by the compiler lead to the improved the performance in the dotProduct() function.

### *manualDotProduct*

Graphical user interface, application, Word

Description automatically generatedFigure 13: manualDotProduct function

This function contains the compiler generated code by automatic parallelization and vectorization enabled. In addition, the function has some changes made to it so that it would work properly and did not produce an error.

Text

Description automatically generated

Figure 14: manualDotProduct function output

|  |  |
| --- | --- |
| **Vector Size** | **Execution Time** |
| 16 | 0.0004429 |
| 32 | 0.000407 |
| 64 | 0.0004497 |
| 128 | 0.0003229 |
| 256 | 0.0009981 |
| 512 | 0.0005392 |
| 1024 | 0.0008414 |
| 2048 | 0.0006353 |
| 4096 | 0.0005696 |
| 8192 | 0.0005338 |
| 16384 | 0.0008989 |
| 32768 | 0.0007567 |
| 65536 | 0.0013626 |

Figure 15: manualDotProduct function results in a table

Figure 16: manualDotProduct function results graphed

Graphical user interface, application, Word

Description automatically generated

Figure 17: compiler generated assembly code for manualDotProduct()

Graphical user interface, application, Word

Description automatically generated

Figure 18: compiler generated assembly code for manualDotProduct()

Graphical user interface, application, Word

Description automatically generated

Figure 19: compiler generated assembly code for manualDotProduct()

The compiler generated assembly code shows that the assembly instructions that was written in the manualDotProduct() function was used by the compiler. This shows that the function was able to optimize the dot product computation.

### *DPPSDotProduct*

Graphical user interface, application, Word

Description automatically generated

Figure 20: DPPSDotProduct function code in Visual Studio

The DPPSDotProduct function utilizes the DPPS vector instructions in order to boost performance. In the previous function, manualDotProduct we used the compilers generated code with automatic parallelization and vectorization and in this functon DPPSDotProduct we are making the dotProduct function with DPPS vector instructions instead.

Text

Description automatically generated

Figure 21: DPPSDotProduct function output.

|  |  |
| --- | --- |
| **Vector Size** | **Execution Time** |
| 16 | 0.0005732 |
| 32 | 0.0003594 |
| 64 | 0.0004087 |
| 128 | 0.0003121 |
| 256 | 0.0004261 |
| 512 | 0.0005491 |
| 1024 | 0.0003637 |
| 2048 | 0.0009178 |
| 4096 | 0.0003937 |
| 8192 | 0.0019717 |
| 16384 | 0.0007752 |
| 32768 | 0.000648 |
| 65536 | 0.0009943 |

Figure 22: DPPSDotProduct function results in a table

Figure 23: DPPSDotProduct function results in a graph

The results of the DPPSDotProduct are plotted in a graph. The DPPSDotProduct graph is unique and different from other graphs because towards the end of the graph the line increases at a much smaller rate than the previous functions graphes.

Graphical user interface, application, Word

Description automatically generated

Figure 24: DPPSDotProduct compiler generated assembly code

Graphical user interface, application, Word

Description automatically generated

Figure 25: DPPSDotProduct compiler generated assembly code

The compiler assembled code shows the DPPS instructions vector instructions utilized by the function which shows that the DPPS instructions are being utilized and are helping the program to be more efficient.

### *Comparison*

Figure 26: This is the graph of that contains all of the previous functions and their results shown in a graph.

After plotting all of the dot product functions execution time, we can compare the different functions execution times. It is evident that DPPSDotProduct function’s execution time is the least at the highest vector sizes, although the function was not as efficient for smaller vector sizes. Next most efficient function is the manualDotProduct function which has the second least execution time at the highest vector sizes. In third, we have the automatic parallelization and vectorization dotProduct function, which is efficient but not really efficient as DPPSDotProduct or manualDotProduct. Lastly, the dotProduct function by itself uses the most execution time and is least efficient. Therefore, we can clearly state that are using different instructions and especially vector instructions we can boost efficiency and reduce the execution time for large vector sizes.

## Linux

### Graphical user interface, text, application Description automatically generated*CPU-X*

Figure 27: CPU specifications on CPU-X Linux

The processor is the Intel Core i9-9900k which supports the instructions MMX, SSE, SSE2, SSE3, SSSE3, SSE4.1, SSE4.2, EM64T, AES, AVX, AVX2, FMA3. This processor does not support he AVX512 vector instruction test. Furthermore, for this take-home test we will be using the AVX2 vector instruction set, which is supported by the processor.

### *Main.cpp*

A screenshot of a computer

Description automatically generated with medium confidence

Figure 28: main.cpp code in Linux

This is the main program file that will measure the time that functions take and also report the other statistics of the function. Some of the statistics are the size of the array, start value, end value. This file right now calls the function dotProduct() in line 24, but this will be changed to the other functions to test out the compiler and DPPS version of the dotProduct function.

### A screenshot of a computer Description automatically generated with medium confidence*Dot Product*

Figure 29: dotProduct function code

Graphical user interface, text, application

Description automatically generated

Figure 30: dotProduct function outputs in Linux terminal

|  |  |
| --- | --- |
| **Vector Size** | **Execution Time** |
| 16 | 9.60E-06 |
| 32 | 1.22E-06 |
| 64 | 1.21E-06 |
| 128 | 1.43E-06 |
| 256 | 2.13E-06 |
| 512 | 1.95E-06 |
| 1024 | 3.07E-06 |
| 2048 | 5.60E-06 |
| 4096 | 8.96E-06 |
| 8192 | 1.71E-05 |
| 16384 | 3.41E-05 |
| 32768 | 6.54E-05 |
| 65536 | 0.000128884 |

Figure 31: dotProducts results in a table

Figure 31: dotProduct function results in a graph

The graph of the dotProduct function shows a somewhat like a linear line, which shows there is not much efficiency here.

### *Dot Product with Automatic Parallelization and Automatic Vectorization*

Graphical user interface, application

Description automatically generated

Figure 33: dotProduct function output in terminal

The terminal shows that we use the flags -O3 -ftree-parralelize-loops=16, this enables automatic parallelization and vectorization.

|  |  |
| --- | --- |
| ***Vector Size*** | ***Execution Time*** |
| *16* | *9.62E-06* |
| *32* | *1.17E-06* |
| *64* | *9.91E-07* |
| *128* | *1.34E-06* |
| *256* | *1.85E-06* |
| *512* | *2.02E-06* |
| *1024* | *2.57E-06* |
| *2048* | *2.40E-05* |
| *4096* | *2.62E-05* |
| *8192* | *2.82E-05* |
| *16384* | *3.66E-05* |
| *32768* | *5.15E-05* |
| *65536* | *8.04E-05* |

Figure 34: dotProduct function results in a table

Figure 36: dotProduct function results in a graph

The graph generally shows that as the vector size increases the execution time increases. The dotProduct function without automatic parallelization and vectorization shows the graph as a linear line but the dotProduct function with automatic parallelization and vectorization shows the graph is somewhat a linear line but doesn’t increase as fast. Therefore, it is highly likely that the automatic parallelization and vectorization lead to the improvement in efficiency.

Text

Description automatically generated with medium confidence

Figure 37: dotProduct function without automatic parallelization and vectorization

Text

Description automatically generated with medium confidence

Figure 37: dotProduct function without automatic parallelization and vectorization

At line 36 to 46 we can see that the compiler does make use of the vector instructions. This proves that because of the automatic parallelization and vectorization was definitely used and is contributing to the improvement in efficiency for the dotProduct function.

### *manualDotProduct*

Text

Description automatically generated

Figure 37: manualDotProduct function

This function contains the compiler generated code by automatic parallelization and vectorization enabled. In addition, the function has some changes made to it so that it would work properly and did not produce an error.

Graphical user interface, application

Description automatically generated

Figure 38: manualDotProduct output in terminal

|  |  |
| --- | --- |
| ***Vector Size*** | ***Execution Time*** |
| *16* | *9.95E-06* |
| *32* | *1.66E-06* |
| *64* | *1.46E-06* |
| *128* | *1.79E-06* |
| *256* | *2.48E-06* |
| *512* | *1.60E-06* |
| *1024* | *2.02E-06* |
| *2048* | *2.08E-06* |
| *4096* | *1.72E-06* |
| *8192* | *2.07E-06* |
| *16384* | *5.66E-06* |
| *32768* | *7.63E-06* |
| *65536* | *1.26E-05* |

Figure 39: manualDotProduct results in table

Figure 40: manualDotProduct results in graph

Text

Description automatically generated

Figure 41: manualDotProduct compiler generated code

Looking at the compiler generated code, it is understandable that the instructions that are written in the functions was utilized by the compiler. Therefore, the code worked properly and the dotProduct function is benefiting from this boost in efficiency.

### *DPPSDotProduct*

Text

Description automatically generated

Figure 42: DPPSDotProduct function code

The DPPSDotProduct function utilizes the DPPS vector instructions in order to boost performance. In the previous function, manualDotProduct we used the compilers generated code with automatic parallelization and vectorization and in this functon DPPSDotProduct we are making the dotProduct function with DPPS vector instructions instead.

Graphical user interface, application

Description automatically generated

Figure 43: DPPSDotProduct function output

|  |  |
| --- | --- |
| **Vector Size** | **Execution Time** |
| 16 | 9.48E-06 |
| 32 | 1.11E-06 |
| 64 | 1.00E-06 |
| 128 | 1.10E-06 |
| 256 | 1.91E-06 |
| 512 | 1.11E-06 |
| 1024 | 1.32E-06 |
| 2048 | 1.33E-06 |
| 4096 | 2.73E-06 |
| 8192 | 2.17E-06 |
| 16384 | 3.60E-06 |
| 32768 | 6.92E-06 |
| 65536 | 1.17E-05 |

Figure 44: DPPSDotProduct function results in table

Figure 45: DPPSDotProduct function results in graph

The compiler assembled code shows the DPPS instructions vector instructions utilized by the function which shows that the DPPS instructions are being utilized and are helping the program to be more efficient.

A black screen with white text

Description automatically generated with low confidence

Figure 46: DPPSDotProduct function compiler generated code

This shows that the compiler actually utilizes the DPPS instructions and proves that the improvement is not due to randomness but to the increase efficiency of using DPPS instructions.

### *Comparison*

Figure 47: This is the graph of that contains all of the previous functions and their results shown in a graph.

After plotting all of the dot product functions execution time on Linux, we can compare the different functions execution times. From the start it is easily noticeable that the execution runtimes are different on Linux because manualDotProduct and DPPSDotProduct have very similar execution times which may be coincidental, but in the end DPPSDotProduct is the most efficient at the highest vector sizes. After DPPSDotProduct and manualDotProduct, we have the automatic parallelization and vectorization of the dotProduct function which is somewhat efficient but not nearly as efficient as DPPSDotProduct and manualDotProduct. Lastly, we have the regular Dot Product function which doesn’t look efficient at all, it literally looks like a linear line. Therefore, we can see that using different instructions and vector instructions especially can help boost efficiency and decrease runtime by quite a margin.s

# Conclusion

The objective of this take-home test is to optimize the compiler generated code for a

program that computes the dot product using vector instructions. In order to correctly optimize the program that computes dot product, we utilized the QueryPerformanceCounter function to measure execution time, and in order to confirm that the optimization of the assembly code led to decreases in execution time. The optimizations that will be used are the automatic parallelization and vectorization, the compiler generated code with vector instructions and the vector instructions DPPS to improve efficiency of the function. These optimizations will be run and recorded with their execution times which will then be listed all together on one graph to be analyzed. Then we will repeat the previous steps that we did in Visual Studio in Linux.