Final Take Home Test

Optimization of Dot Product Computation of Two Vectors Using Vector Instructions

Name: Ismail Akram

Date: 12/14/2021

Course: CSC 34200-G & CSC34300-DE

Contents

[**Objective:** 3](#_Toc90446932)

[**Introduction** 4](#_Toc90446933)

[**Functionality and Specification:** 5](#_Toc90446934)

[Intel x86 (Visual Studio Code) 5](#_Toc90446935)

[Dot Product (Pointer) [Non-Optimized] 6](#_Toc90446936)

[Dot Product (Pointer) [Compiler Generated Optimization] 7](#_Toc90446937)

[Dot Product (Pointer) [Manually Optimized] 8](#_Toc90446938)

[VDPPS 9](#_Toc90446939)

[Intel x64 (Linux GCC) 10](#_Toc90446940)

[Dot Product (Pointer) [Non-Optimized] 10](#_Toc90446941)

[Dot Product (Pointer) [Compiler Optimization flags] 11](#_Toc90446942)

[Dot Product (Pointer) [Compiler Assembly Output] 12](#_Toc90446943)

[**Simulation** 13](#_Toc90446944)

[Intel x86 (Visual Studio Code) 13](#_Toc90446945)

[Dot Product (Pointer) [Non-Optimized] 13](#_Toc90446946)

[Dot Product (Pointer) [Compiler Generated Optimization] 20](#_Toc90446947)

[Dot Product (Pointer) [Manually Optimized] 27](#_Toc90446948)

[VDPPS 34](#_Toc90446949)

[Intel x64 (Linux GCC) 41](#_Toc90446950)

[Dot Product (Pointer) [Non-Optimized] 41](#_Toc90446951)

[Dot Product (Pointer) [Compiler Optimization flags] 42](#_Toc90446952)

[**Results** 45](#_Toc90446953)

[Conclusion 47](#_Toc90446954)

# **Objective:**

To optimize compiler generated code to compute dot product using vector instructions. To understand the compiler’s ability to reduce run-time of a program, we must turn off optimizations. We then run our dot products (I’ve chosen pointers since it’s more ideal) for the arrays. The time needed to run these programs is measure for the functions with various array sizes of magnitude 2.

After the runs, we then compare the compiler optimizes code with the non-optimized code. We then manually optimized the compiler-generated assembly code so that the dot product for vastly greater arrays will be calculated more swiftly. We will then use VDPPS for our final runs. The compiler will use auto-parallelization and auto-vectorization for optimization of large arrays.

# **Introduction**

To identify my hardware’s processor, I used CPU-Z application. I took note of the instruction sets that are available.

Graphical user interface, application

Description automatically generated

*Fig 1. My CPU Processor info from CUP-Z (AVX2)*



*Fig 1a: CPU Processor info in VM*

# **Functionality and Specification:**

## Intel x86 (Visual Studio Code)

A computer screen capture

Description automatically generated with medium confidence

*Fig 2. Source.cpp (code)*

The dot product calculation is implemented via pointer arithmetic (Fig 3). In theory, pointer arithmetic is faster than indexing since indices require a multiplication of 4 for each increment whereas pointers forego the multiplication step. Therefore, I chose pointers over indexing.

Note: Auto-vectorization is disabled via #pragma loop(no\_vector).

Source.cpp (Fig 2) is the main function. The array size is const 16 for the first calculation and will change to various sizes to measure different times. Two static arrays of the same size are declared. They are initialized to arbitrary numbers since the result of the dot product is irrelevant.

Note: QueryPerformanceCounter is used to measure the run-time of the dot product function.

Note: optimization, auto-vectorization, and auto-parallelization is disabled in Visual Studio.

The run-time is measured for the following array sizes; N = 16, 32, 64, 128, 256, 512, and 1024.

### Dot Product (Pointer)

Pointer arithmetic dot product. Note that we’re passing float.

A computer screen capture

Description automatically generated with medium confidence

*Fig 3: Dot\_Prod\_Pntr.cpp (Code)*

### Dot Product (Pointer) [Compiler Generated Optimization]

The compiler optimizes the generated .asm code (Fig 4). We will then manually improve this code for the next tests. Parallelization and vectorization is enabled for these tests.

A computer screen capture

Description automatically generated with medium confidence

A computer screen capture

Description automatically generated with medium confidence

*Fig 4: Dot\_Prod\_Pntr (Code) [Compiler Generated Optimization]*

### Dot Product (Pointer) [Manually Optimized]

For pointer arithmetic dot product, the assembly code has been manually optimized (Fig 5). This was done by removing redundant/repetitive code and/or moved to out of the loop, so it’s not repeated unnecessarily. We managed to reduce the 105 lines code down to 72 lines.

A computer screen capture

Description automatically generated with medium confidence

*Fig 5: Opt\_Dot\_Prod\_Pntr.asm (Code) [Manually Optimized]*

### VDPPS

For further optimization, we are using vector instruction DPPS to computer the dot product. We are then using the same QueryPerformanceCounter function to measure the execution time.

A picture containing text, monitor, screenshot, electronics

Description automatically generated

*Fig 6: VDPPS\_Dot\_Prod\_Pntr.cpp (Code)*

## Intel x64 (Linux GCC)

### Dot Product (Pointer) [Non-Optimized]

For out run in Linux 64-bit, we are using the same C++ codes for the function of dot product. However, some modifications are needed for Source.cpp; we must use clock\_gettime to measure the execution time. We use the -o0 Optimization flag for no optimization.

Graphical user interface, text

Description automatically generated

*Fig 7: Source.cpp (Code)*

### Dot Product (Pointer) [Compiler Optimization flags]

For optimization in a Linux environment, it is much more efficient to use the compiler flags. These are enabled by using “-o” alongside an optimization level. We’re using the flags that reduce execution time.

There are tradeoffs to using optimization glass, namely in compilation time and memory usage.

|  |  |  |  |
| --- | --- | --- | --- |
| Optimization Flag | Optimization Level | Execution Time | Optimization Notes |
| -o0 | O (Default) | + | None |
| -o1 | 1 | - | Basic |
| -o2 | 2 | -- | Recommended |
| -o3 | 3 (Highest) | --- | Not Guaranteed |

*Fig 8: Table summary of Optimizatio Flags*

Calendar

Description automatically generated with medium confidence

*Fig 9: Compiling Dot Product Programs using the optimization flags*

### Dot Product (Pointer) [Compiler Assembly Output]

Text

Description automatically generated

*Fig 10: Dot\_Prod\_Pntr.s (Code) [Compiler Generated Optimization]*

# **Simulation**

## Intel x86 (Visual Studio Code)

### Dot Product (Pointer) [Non-Optimized]

Note, for all simulations, we are running the dot product 10 times using a for loop. This is so we can account for error margins in execution time and get a precise measure by averaging them. We are accounting for array size N = 16 … 2^16 and noting the execution time in seconds.

A picture containing text

Description automatically generated

*Fig 11: Dot\_Prod\_Pntr (16) w/0 Qpar and arch*

Text

Description automatically generated with medium confidence

*Fig 11a: Dot\_Prod\_Pntr (32) w/0 Qpar and arch*

A picture containing text

Description automatically generated

*Fig 11b: Dot\_Prod\_Pntr (64) w/0 Qpar and arch*

Text

Description automatically generated with medium confidence

*Fig 11c: Dot\_Prod\_Pntr (128) w/0 Qpar and arch*

Text

Description automatically generated with medium confidence

*Fig 11d: Dot\_Prod\_Pntr (256) w/0 Qpar and arch*

Text

Description automatically generated

*Fig 11e: Dot\_Prod\_Pntr (512) w/0 Qpar and arch*

Text

Description automatically generated with medium confidence

*Fig 11f: Dot\_Prod\_Pntr (1024) w/0 Qpar and arch*

A picture containing text

Description automatically generated

*Fig 11g: Dot\_Prod\_Pntr (2^11) w/0 Qpar and arch*

Text

Description automatically generated with low confidence

*Fig 11h: Dot\_Prod\_Pntr (2^12) w/0 Qpar and arch*

A computer screen capture

Description automatically generated with medium confidence

*Fig 11i: Dot\_Prod\_Pntr (2^13) w/0 Qpar and arch*

A computer screen capture

Description automatically generated with medium confidence

*Fig 11j: Dot\_Prod\_Pntr (2^14) w/0 Qpar and arch*

Text

Description automatically generated with medium confidence

*Fig 11k: Dot\_Prod\_Pntr (2^15) w/0 Qpar and arch*

Text

Description automatically generated

*Fig 11l: Dot\_Prod\_Pntr (2^16) w/0 Qpar and arch*

### Dot Product (Pointer) [Compiler Generated Optimization]

Optimizations enabled.

Text

Description automatically generated

*Fig 12: Dot\_Prod\_Pntr (16) w/ Qpar and arch*

Text

Description automatically generated

*Fig 12a: Dot\_Prod\_Pntr (32) w/ Qpar and arch*

Text

Description automatically generated with low confidence

*Fig 12b: Dot\_Prod\_Pntr (64) w/ Qpar and arch*

Text

Description automatically generated with medium confidence

*Fig 12c: Dot\_Prod\_Pntr (128) w/ Qpar and arch*

Text

Description automatically generated with medium confidence

*Fig 12d: Dot\_Prod\_Pntr (256) w/ Qpar and arch*

Text

Description automatically generated with medium confidence

*Fig 12e: Dot\_Prod\_Pntr (512) w/ Qpar and arch*

Text

Description automatically generated with medium confidence

*Fig 12f: Dot\_Prod\_Pntr (1024) w/ Qpar and arch*

A computer screen capture

Description automatically generated with medium confidence

*Fig 12g: Dot\_Prod\_Pntr (2^11) w/ Qpar and arch*

Text

Description automatically generated with medium confidence

*Fig 12h: Dot\_Prod\_Pntr (2^12) w/ Qpar and arch*

Text

Description automatically generated with low confidence

*Fig 12i: Dot\_Prod\_Pntr (2^13) w/ Qpar and arch*

Text

Description automatically generated

*Fig 12j: Dot\_Prod\_Pntr (2^14) w/ Qpar and arch*

Text

Description automatically generated

*Fig 12k: Dot\_Prod\_Pntr (2^15) w/ Qpar and arch*

Text

Description automatically generated with medium confidence

*Fig 12l: Dot\_Prod\_Pntr (2^16) w/ Qpar and arch*

### Dot Product (Pointer) [Manually Optimized]

Manually optimized compiler generated .asm file

Text

Description automatically generated with medium confidence

*Fig 13: Op\_Dot\_Prod\_Pntr (16) w/ Qpar and arch & Manual optimization*

Text

Description automatically generated

*Fig 13a: Op\_Dot\_Prod\_Pntr (32) w/ Qpar and arch & Manual optimization*

Text

Description automatically generated with medium confidence

*Fig 13b: Op\_Dot\_Prod\_Pntr (64) w/ Qpar and arch & Manual optimization*

Text

Description automatically generated with medium confidence

*Fig 13c: Op\_Dot\_Prod\_Pntr (128) w/ Qpar and arch & Manual optimization*

Text

Description automatically generated

*Fig 13d: Op\_Dot\_Prod\_Pntr (256) w/ Qpar and arch & Manual optimization*

Text

Description automatically generated with medium confidence

*Fig 13e: Op\_Dot\_Prod\_Pntr (512) w/ Qpar and arch & Manual optimization*

Text

Description automatically generated

*Fig 13f: Op\_Dot\_Prod\_Pntr (1024) w/ Qpar and arch & Manual optimization*

Text

Description automatically generated with medium confidence

*Fig 13g: Op\_Dot\_Prod\_Pntr (2^11) w/ Qpar and arch & Manual optimization*

Text

Description automatically generated with medium confidence

*Fig 13h: Op\_Dot\_Prod\_Pntr (2^12) w/ Qpar and arch & Manual optimization*

Text

Description automatically generated

*Fig 13i: Op\_Dot\_Prod\_Pntr (2^13) w/ Qpar and arch & Manual optimization*

Text

Description automatically generated with medium confidence

*Fig 13j: Op\_Dot\_Prod\_Pntr (2^14) w/ Qpar and arch & Manual optimization*

Text

Description automatically generated with medium confidence

*Fig 13k: Op\_Dot\_Prod\_Pntr (2^15) w/ Qpar and arch & Manual optimization*

Text

Description automatically generated

*Fig 13l: Op\_Dot\_Prod\_Pntr (2^16) w/ Qpar and arch & Manual optimization*

### VDPPS

Text

Description automatically generated

*Fig 14: VDPPS\_Dot\_Prod\_Pntr (16) w/ Qpar and arch*

Text

Description automatically generated

*Fig 14a: VDPPS\_Dot\_Prod\_Pntr (32) w/ Qpar and arch*

Text

Description automatically generated with medium confidence

*Fig 14b: VDPPS\_Dot\_Prod\_Pntr (64) w/ Qpar and arch*

Text

Description automatically generated with medium confidence

*Fig 14c: VDPPS\_Dot\_Prod\_Pntr (128) w/ Qpar and arch*

Text

Description automatically generated

*Fig 14d: VDPPS\_Dot\_Prod\_Pntr (256) w/ Qpar and arch*

Text

Description automatically generated with medium confidence

*Fig 14e: VDPPS\_Dot\_Prod\_Pntr (512) w/ Qpar and arch*

Text

Description automatically generated

*Fig 14f: VDPPS\_Dot\_Prod\_Pntr (1024) w/ Qpar and arch*

Text

Description automatically generated

*Fig 14g: VDPPS\_Dot\_Prod\_Pntr (2^11) w/ Qpar and arch*

A picture containing text

Description automatically generated

*Fig 14h: VDPPS\_Dot\_Prod\_Pntr (2^12) w/ Qpar and arch*

Text

Description automatically generated with medium confidence

*Fig 14i: VDPPS\_Dot\_Prod\_Pntr (2^13) w/ Qpar and arch*

Text

Description automatically generated

*Fig 14j: VDPPS\_Dot\_Prod\_Pntr (2^14) w/ Qpar and arch*

Text

Description automatically generated with medium confidence

*Fig 14k: VDPPS\_Dot\_Prod\_Pntr (2^15) w/ Qpar and arch*

Text

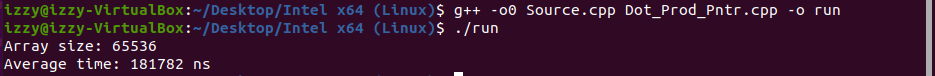
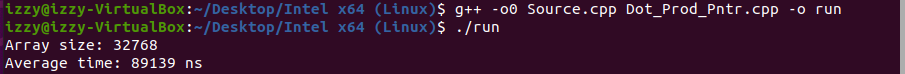
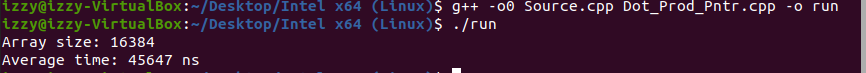
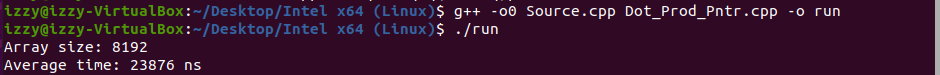
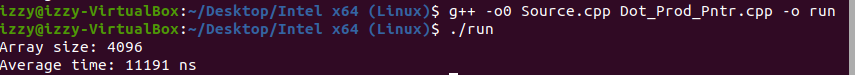
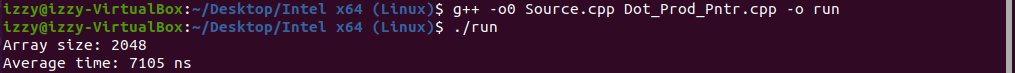
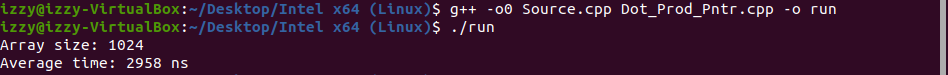
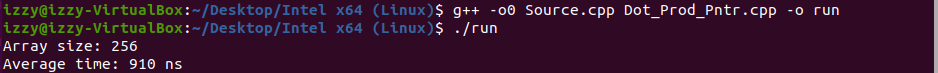
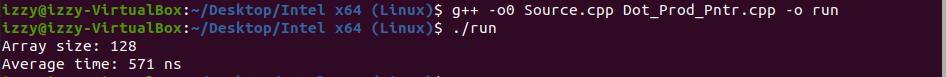
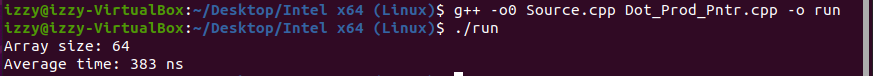
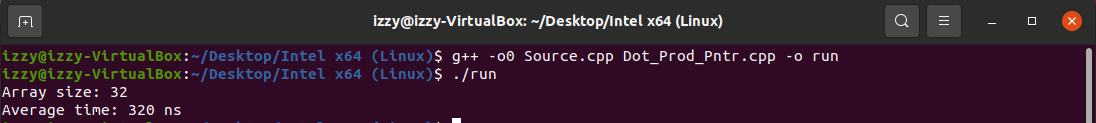
Description automatically generated

*Fig 14l: VDPPS\_Dot\_Prod\_Pntr (2^16) w/ Qpar and arch*

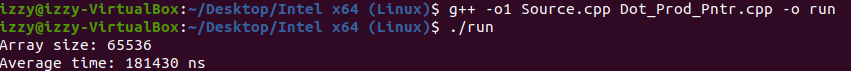
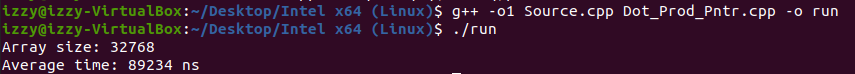
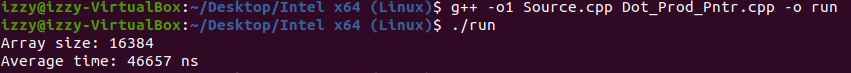
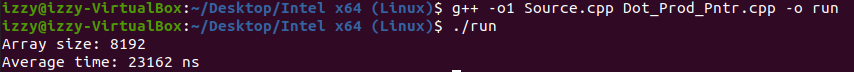
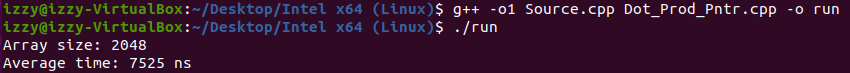
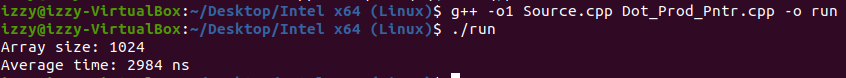
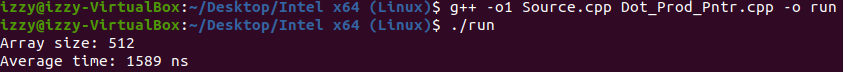
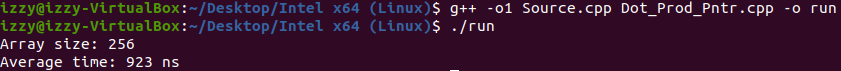
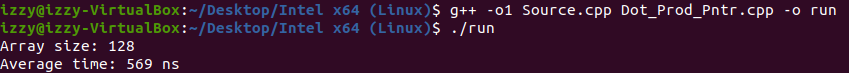
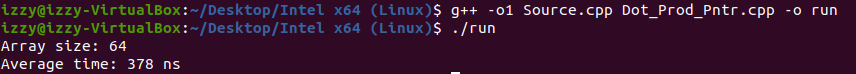
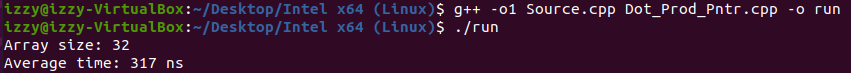
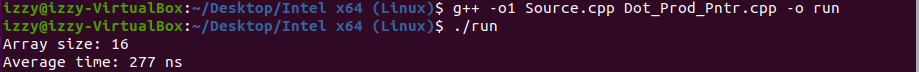
## Intel x64 (Linux GCC)

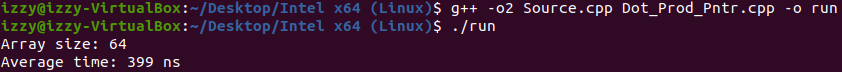
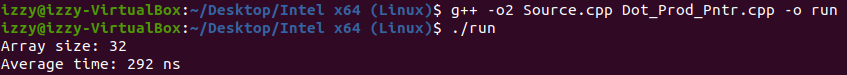
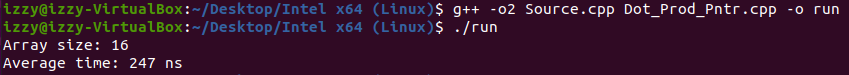
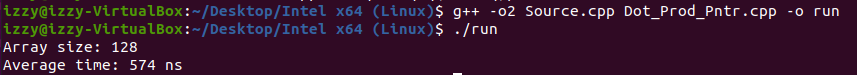
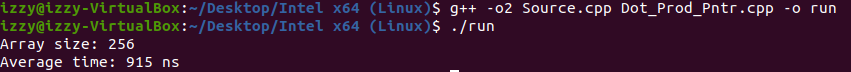
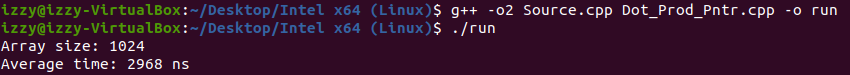
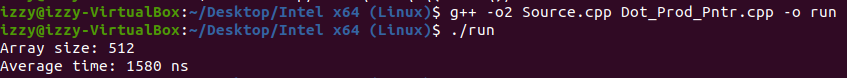
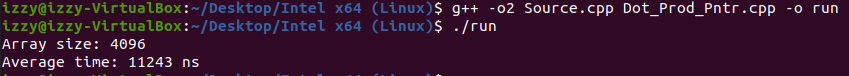
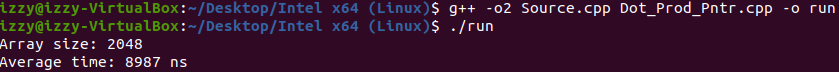
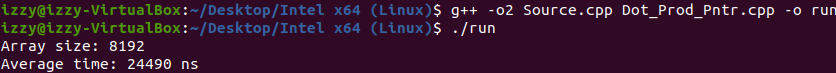
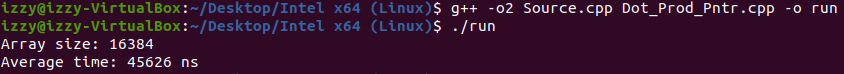
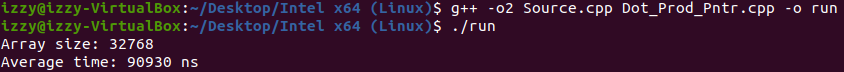
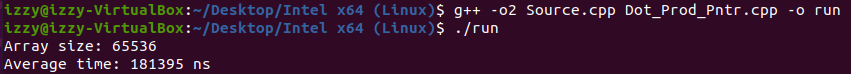
### Dot Product (Pointer) [Non-Optimized]

Text

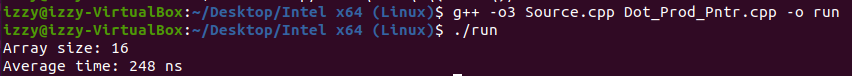
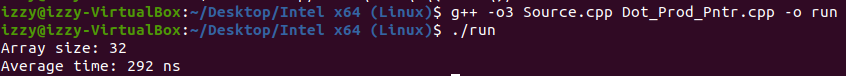
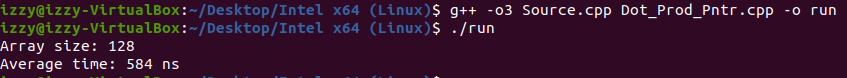
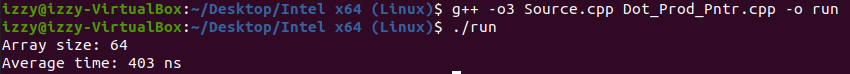
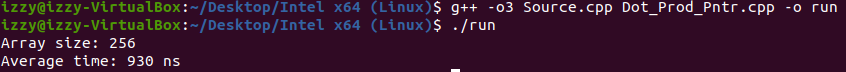
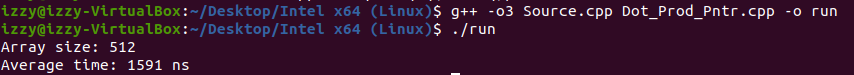
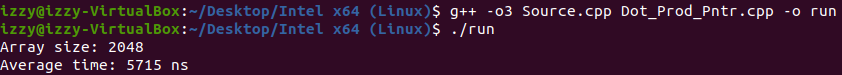
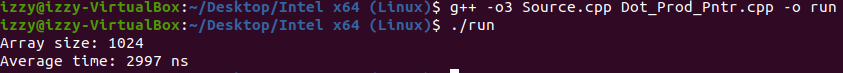
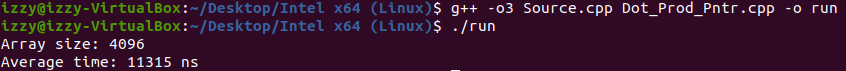
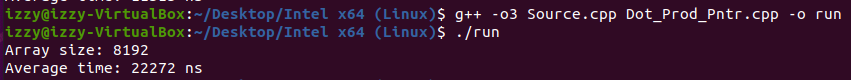
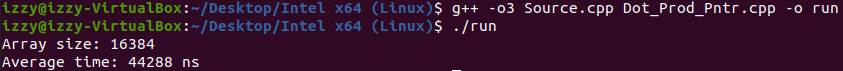
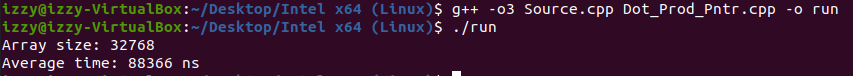
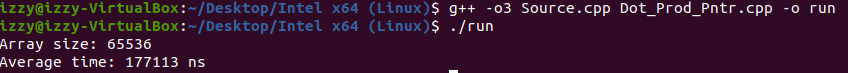
Description automatically generated*Figs 15-15l: Dot\_Prod\_Pntr (-o0 optimization)*

### Dot Product (Pointer) [Compiler Optimization flags]

*Figs 16-16l: Dot\_Prod\_Pntr (-o1 optimization)*

*Figs 17-17l: Dot\_Prod\_Pntr (-o2 optimization)*

*Fig 18-18l: Dot\_Prod\_Pntr (-o3 optimization)*

# **Results**

Graphical user interface, table

Description automatically generated

Chart, line chart

Description automatically generated

*Fig 19: Extraction times comparison of Dot\_Prod\_Pntr in VS*

Graphical user interface, table

Description automatically generated with medium confidence

Chart, line chart

Description automatically generated

*Fig 19: Extraction times comparison of Dot\_Prod\_Pntr optimization level in Linux*

# Conclusion

Looking at the execution time outputs from our exercise in VS (figure 18), we note that there is a decernable difference between:

* Dot\_Prod\_Pnt (Non-Opt)
* Dot\_Prod\_Pntr (Opt)
* Dot\_Prod\_Pntr (Man Opt)
* VDPPS\_Dot\_Prod\_Pntr

With the time being fast with each optimization. We notice that our manually optimized assembly is faster than the compiler generated one. But ultimately VDPPS is noticeably faster which isn’t surprising.

On the Linux side, we notice a decrease in compilation time with each optimization level:

* -o0
* -o1
* -o2
* -o3

Given these results (figure 19), applying one level of optimization drastically improves the execution time. We can also conclude that optimization level 2 is faster than optimization level 1. However, the difference between optimization levels 2 and 3 are not as drastic, which was expected.

Optimization level 2 is a much better option in reducing the execution time (since there is a tradeoff with memory usage). Optimization level 1 is also a good choice is maximum optimization is not needed.