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
Introduction 4
Functionality and Specification: 5
Intel x86 (Visual Studio Code) 5
Dot Product (Pointer) [Non-Optimized] 6
Dot Product (Pointer) [Compiler Generated Optimization] 7
Dot Product (Pointer) [Manually Optimized] 8
VDPPS 9
Intel x64 (Linux GCC) 10
Dot Product (Pointer) [Non-Optimized] 10
Dot Product (Pointer) [Compiler Optimization flags] 11
Dot Product (Pointer) [Compiler Assembly Output] 12
Simulation 13
Intel x86 (Visual Studio Code) 13
Dot Product (Pointer) [Non-Optimized] 13
Dot Product (Pointer) [Compiler Generated Optimization] 20
Dot Product (Pointer) [Manually Optimized] 27
VDPPS 34
Intel x64 (Linux GCC) 41
Dot Product (Pointer) [Non-Optimized] 41
Dot Product (Pointer) [Compiler Optimization flags] 42
Results 45
Conclusion 47

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.

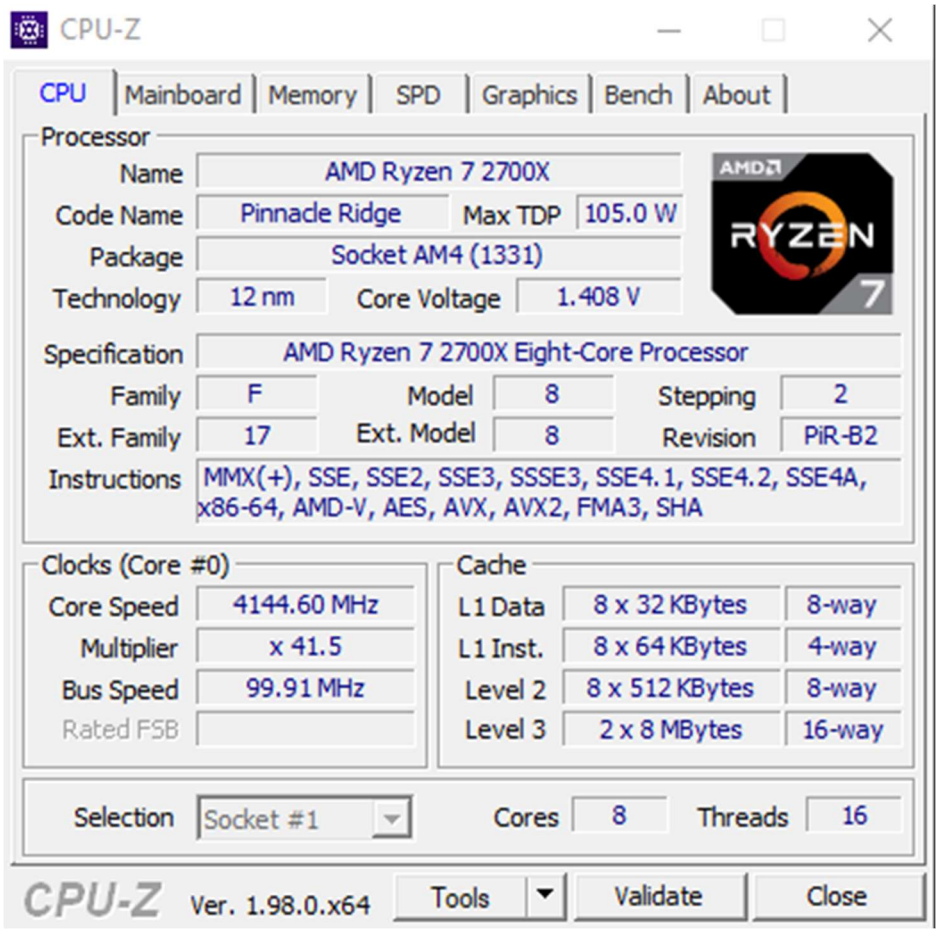


Fig 1. My CPU Processor info from CUP-Z (AVX2)

```
izzy@izzy-VirtualBox:~$ lscpu | grep "Model name"
Model name:                      AMD Ryzen 7 2700X Eight-Core Processor
```

Fig 1a: CPU Processor info in VM

Functionality and Specification:

Intel x86 (Visual Studio Code)

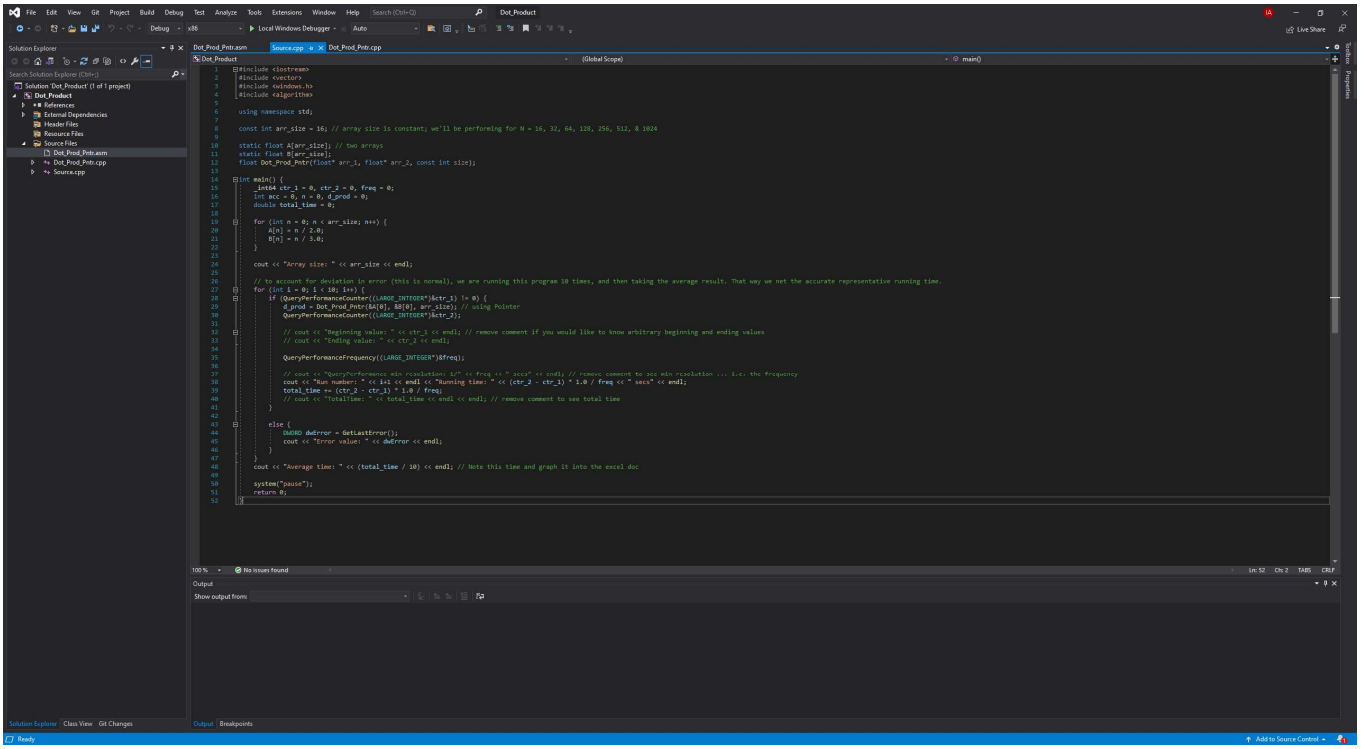


Fig 2. Source.cpp (code)

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.

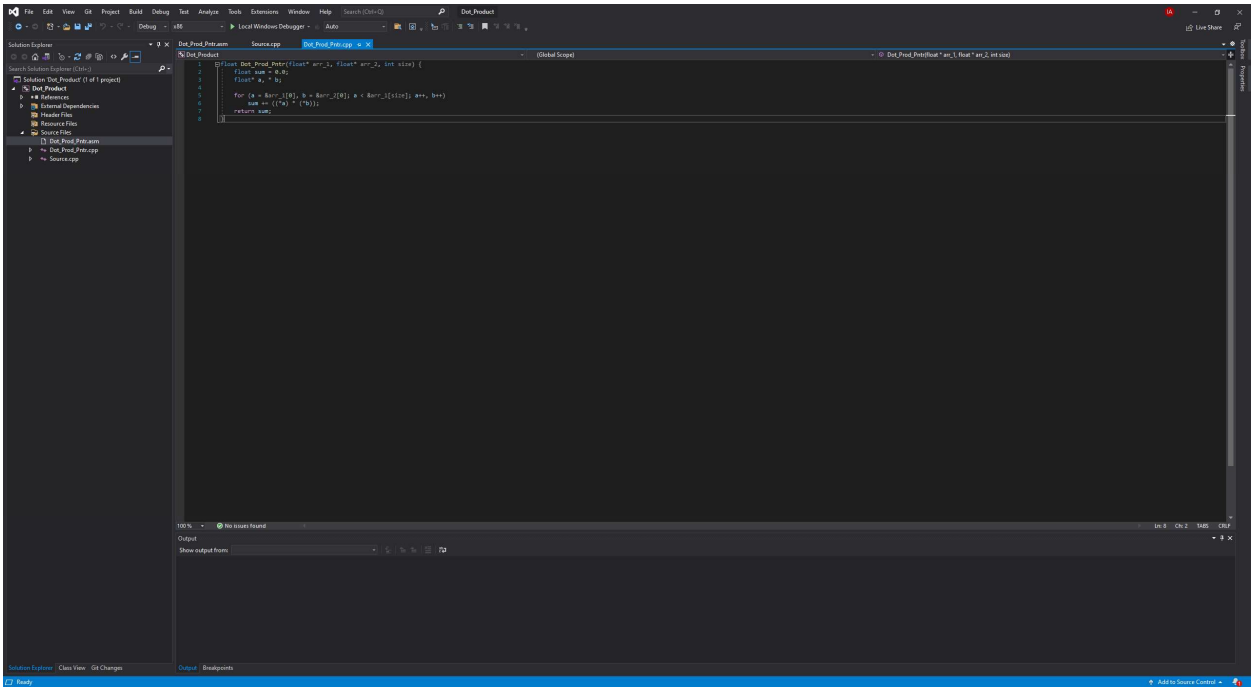


Fig 3: Dot_Prod_Pntr.cpp (Code)

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.

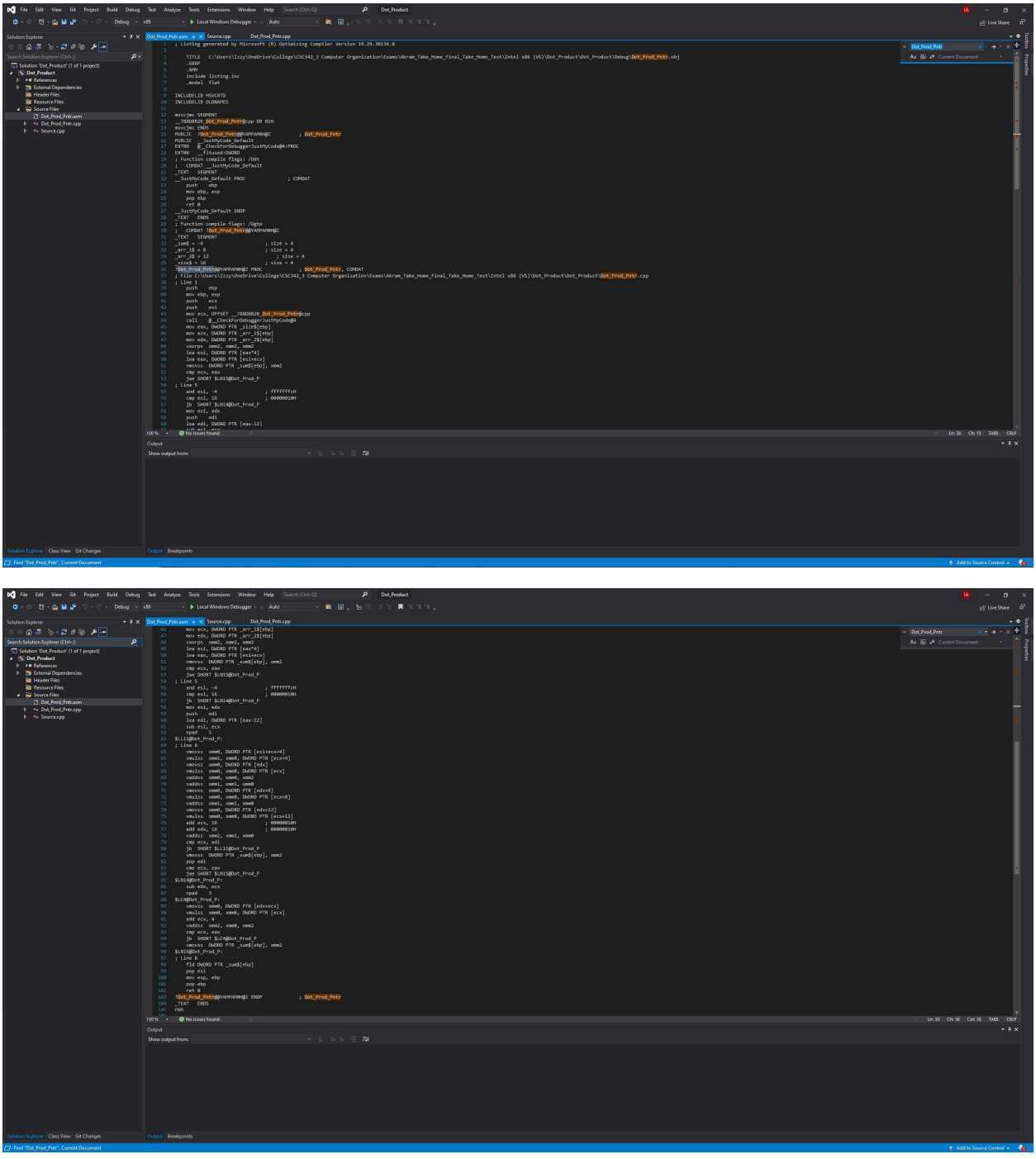


Fig 4: Dot_Prod_Pntr (Code) [Compiler Generated Optimization]

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.

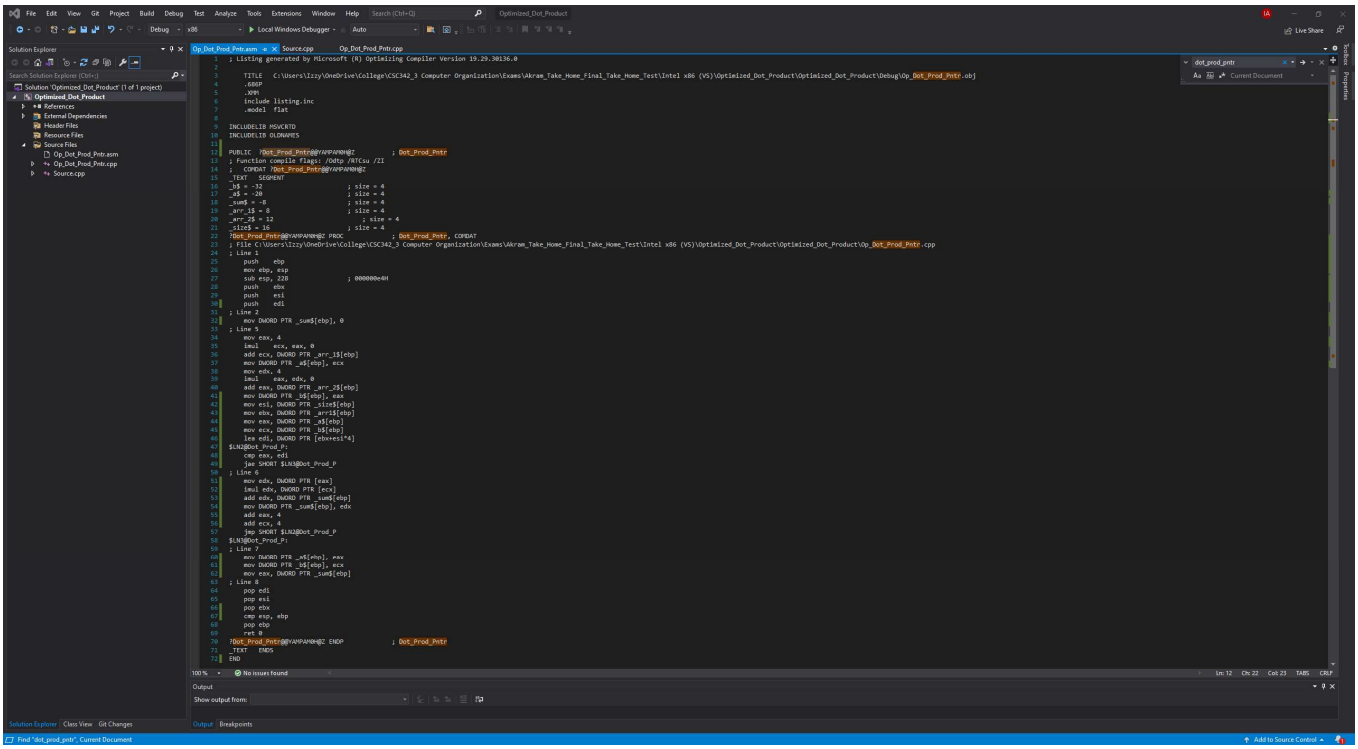


Fig 5: Opt_Dot_Prod_Pntr.asm (Code) [Manually Optimized]

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.

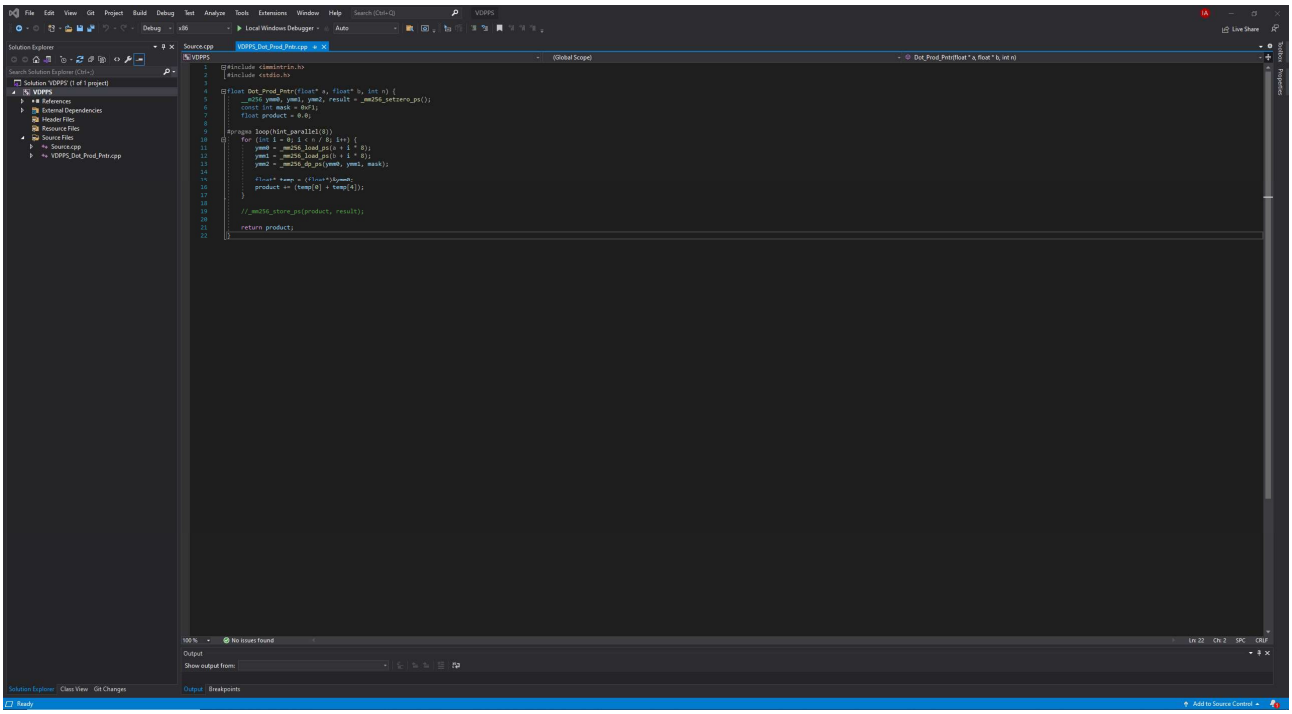


Fig 6: VDPPS_Dot_Prod_Pntr.cpp (Code)

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.

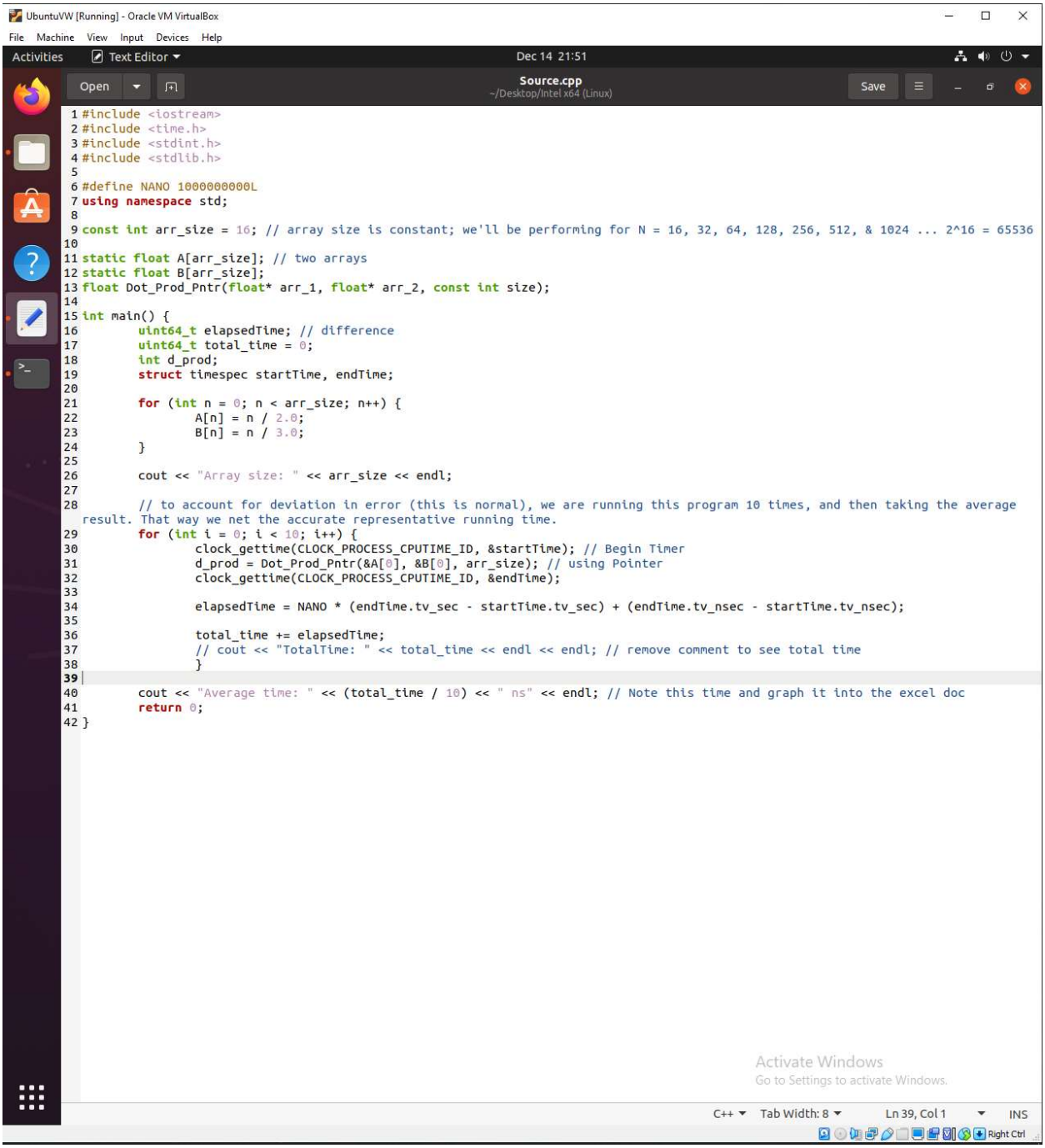


Fig 7: Source.cpp (Code)

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

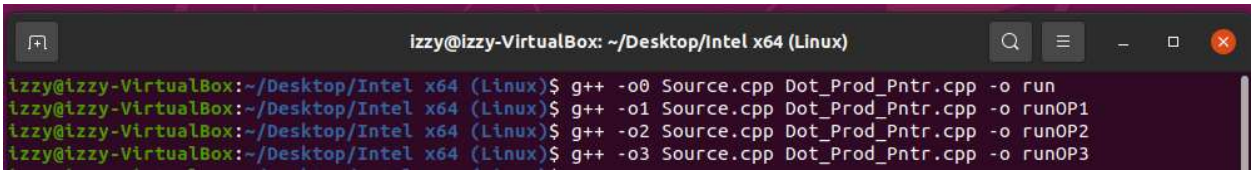


Fig 9: Compiling Dot Product Programs using the optimization flags

Dot Product (Pointer) [Compiler Assembly Output]

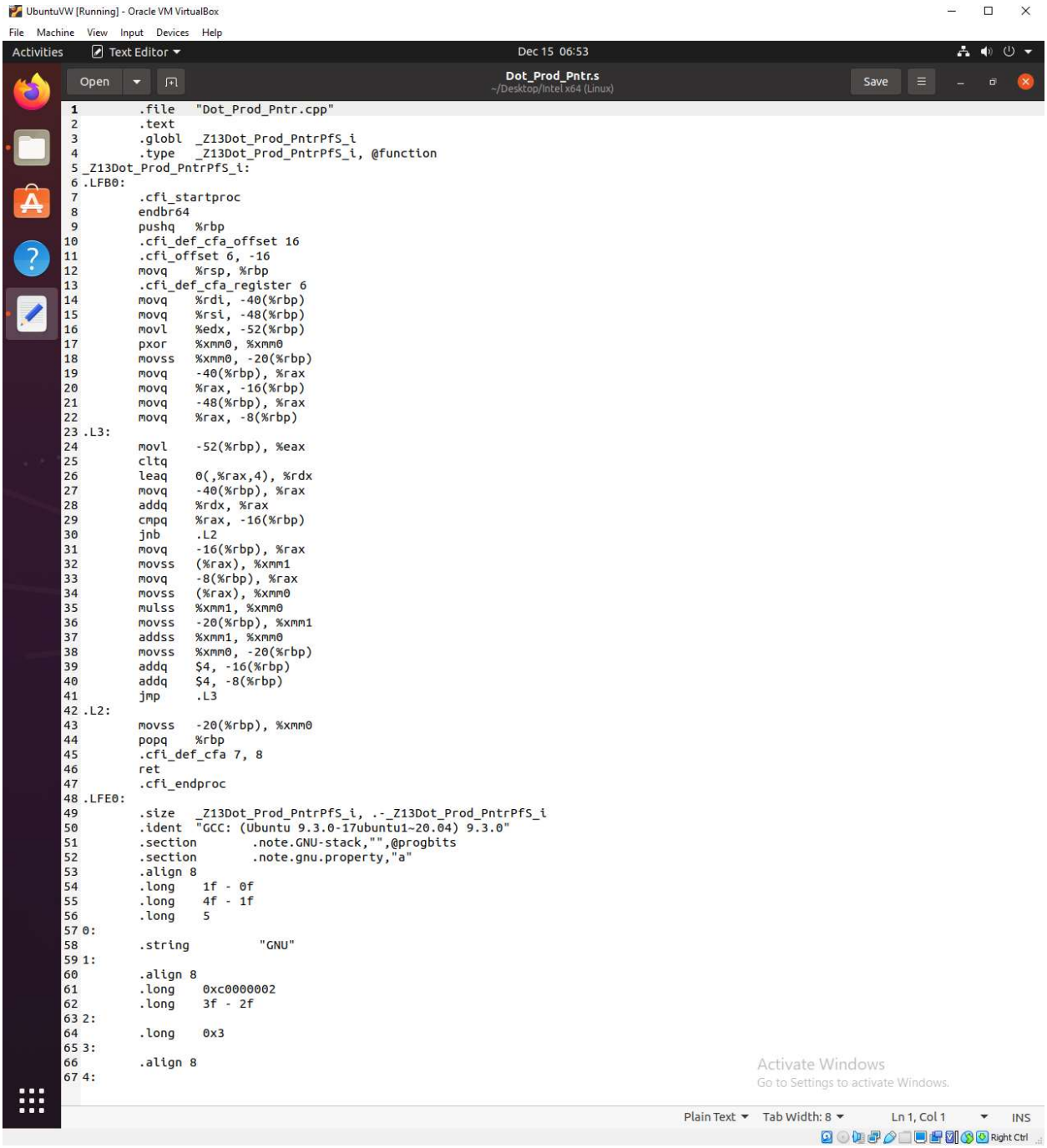


Fig 10: Dot_Prod_Pntr.s (Code) [Compiler Generated Optimization]

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.

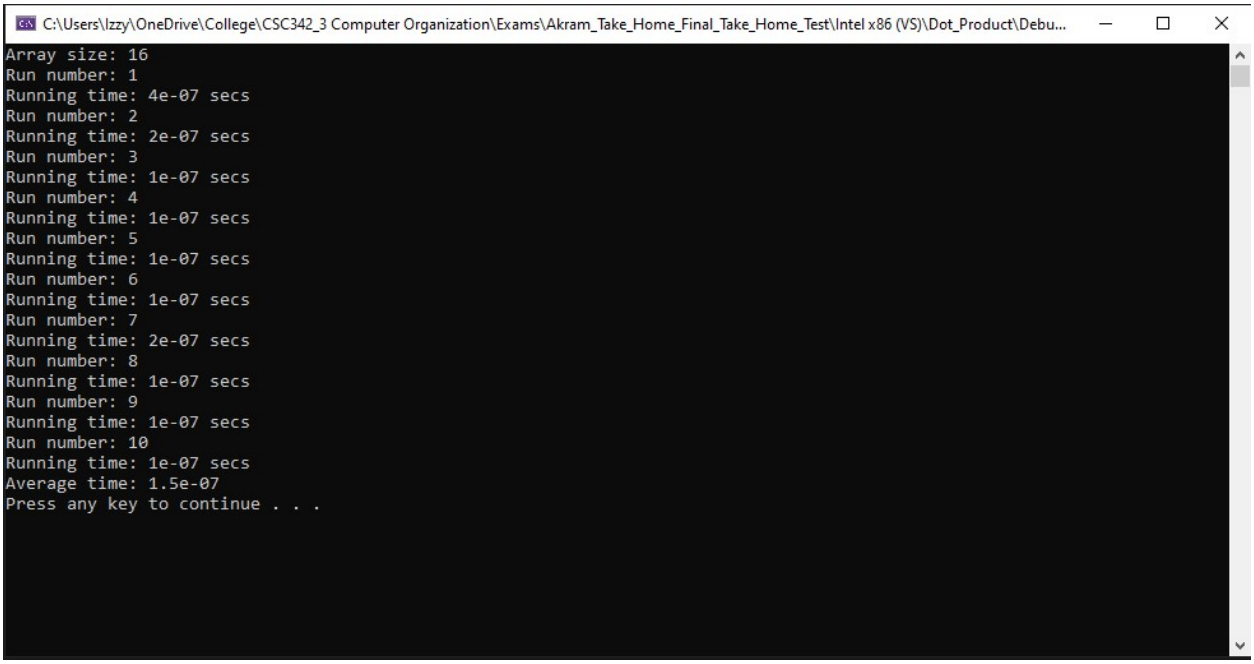


Fig 11: Dot_Prod_Pntr (16) w/0 Qpar and arch

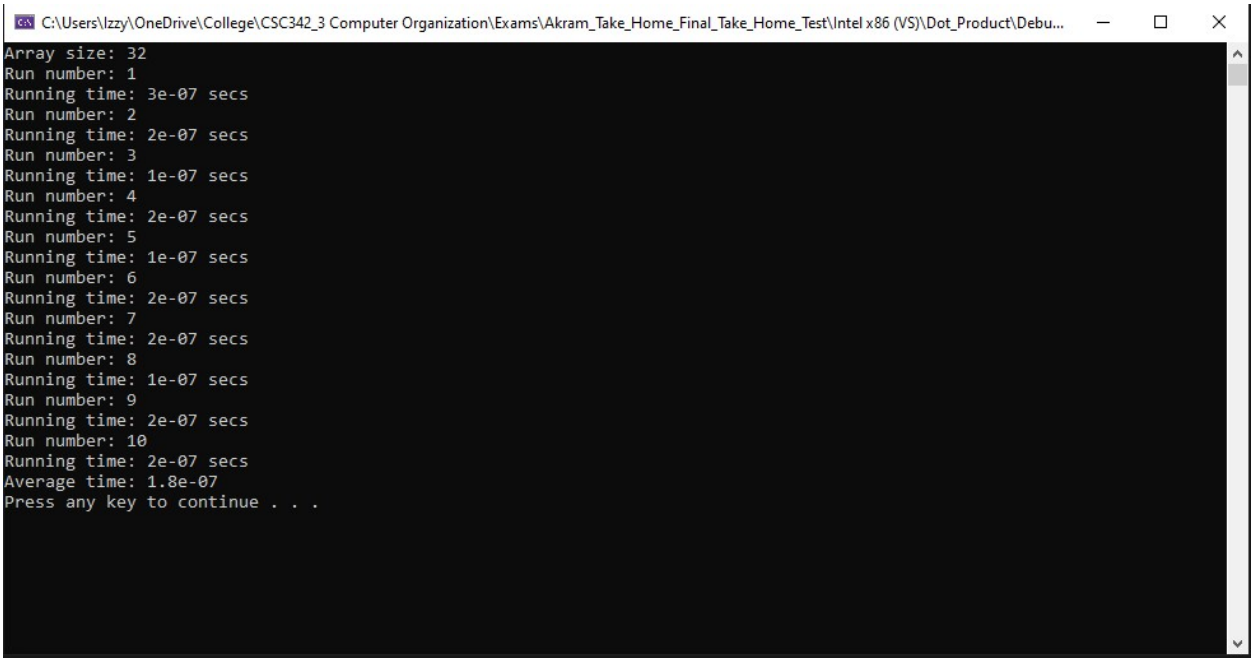


Fig 11a: Dot_Prod_Pntr (32) w/0 Qpar and arch

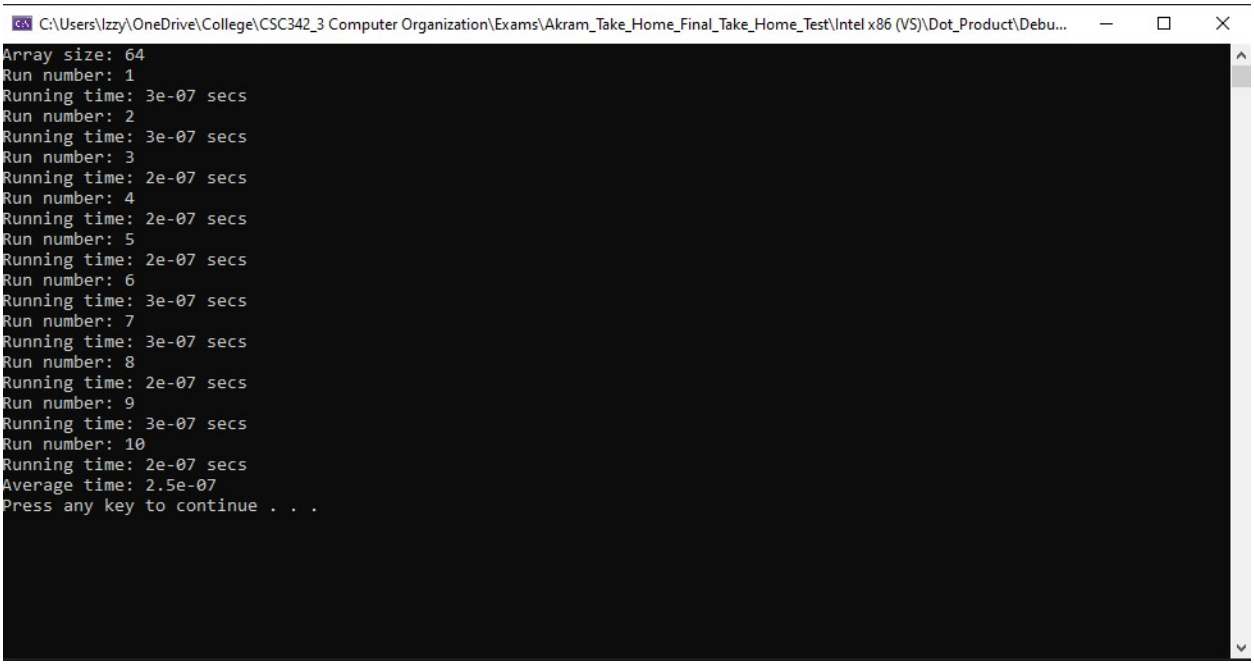


Fig 11b: Dot_Prod_Pntr (64) w/0 Qpar and arch

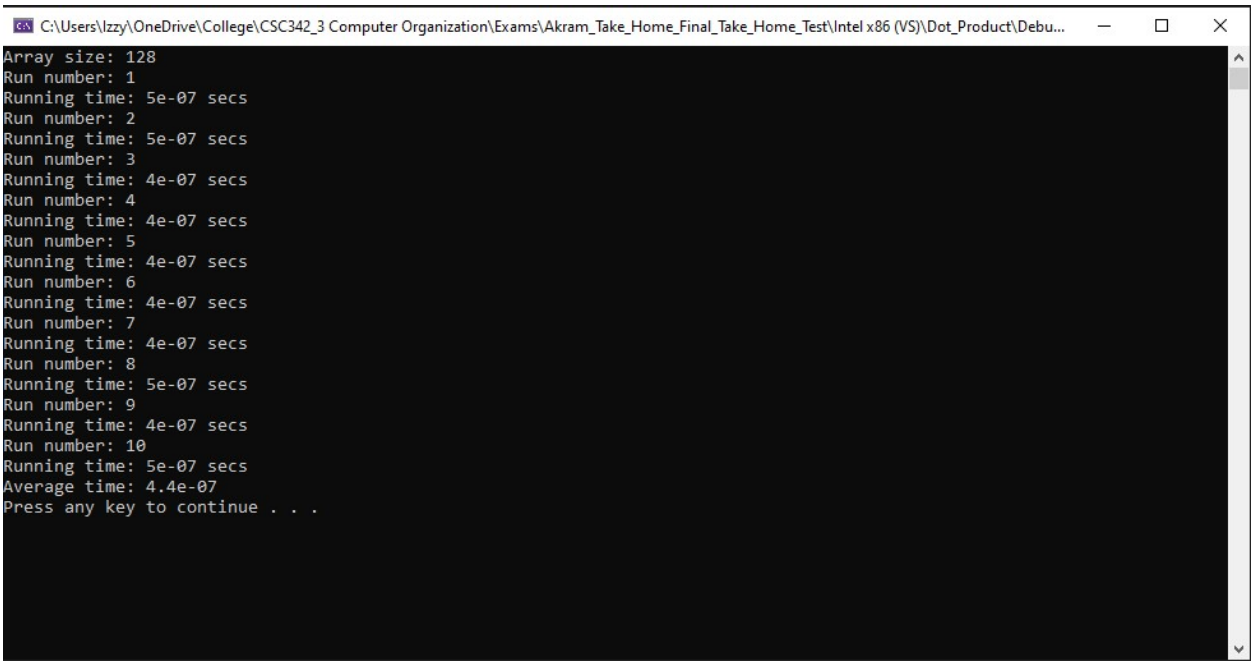


Fig 11c: Dot_Prod_Pntr (128) w/0 Qpar and arch

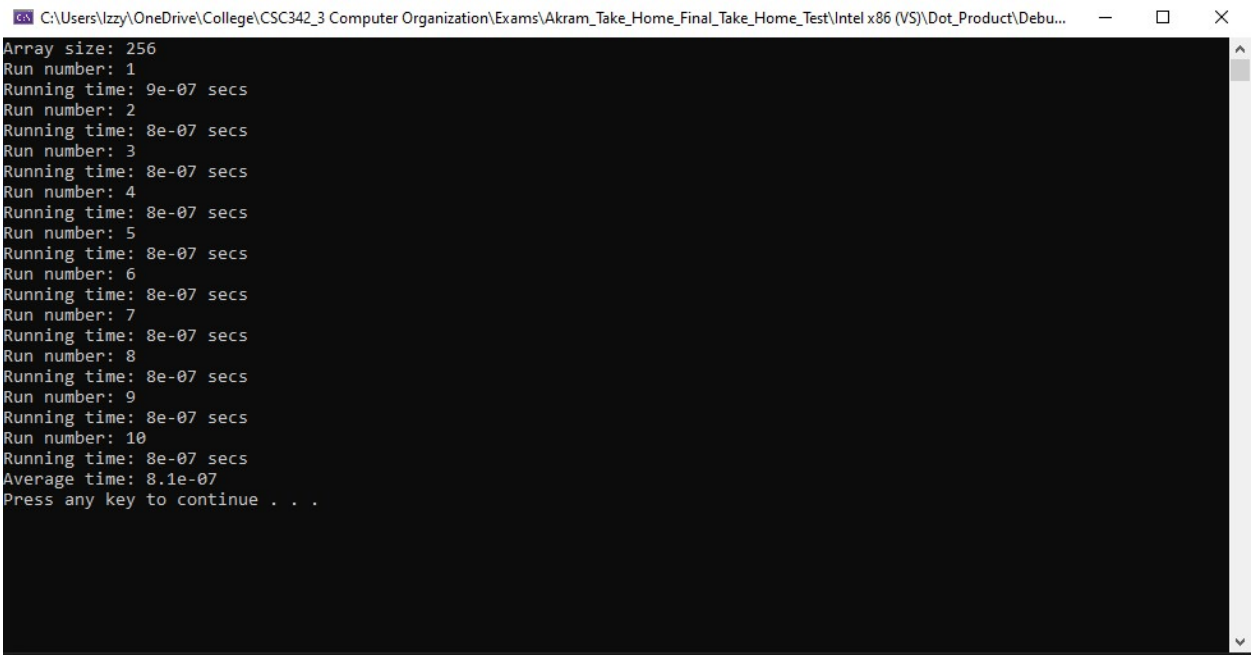


Fig 11d: Dot_Prod_Pntr (256) w/0 Qpar and arch

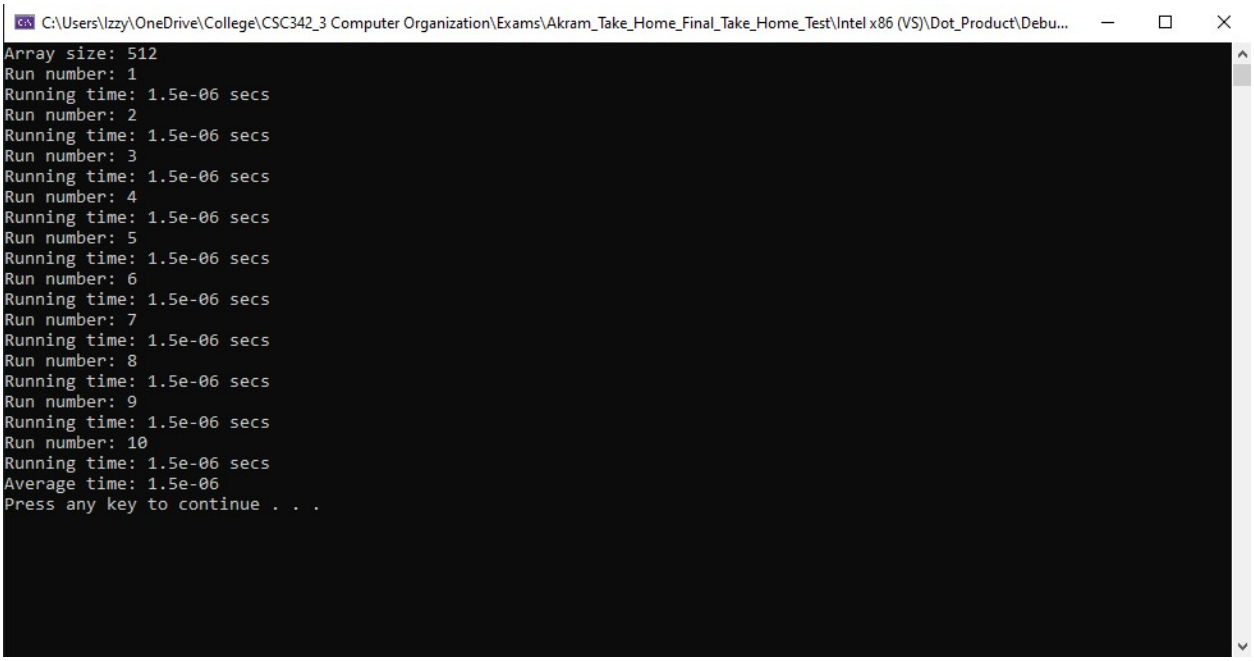


Fig 11e: Dot_Prod_Pntr (512) w/0 Qpar and arch

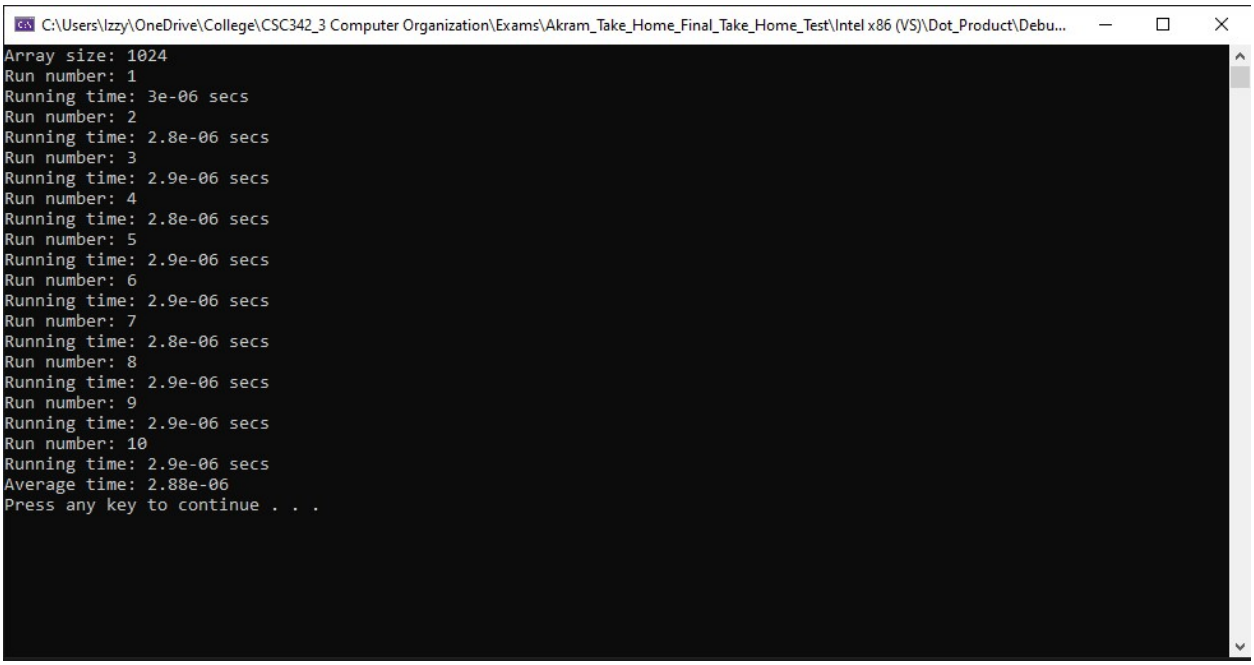


Fig 11f: Dot_Prod_Pntr (1024) w/0 Qpar and arch

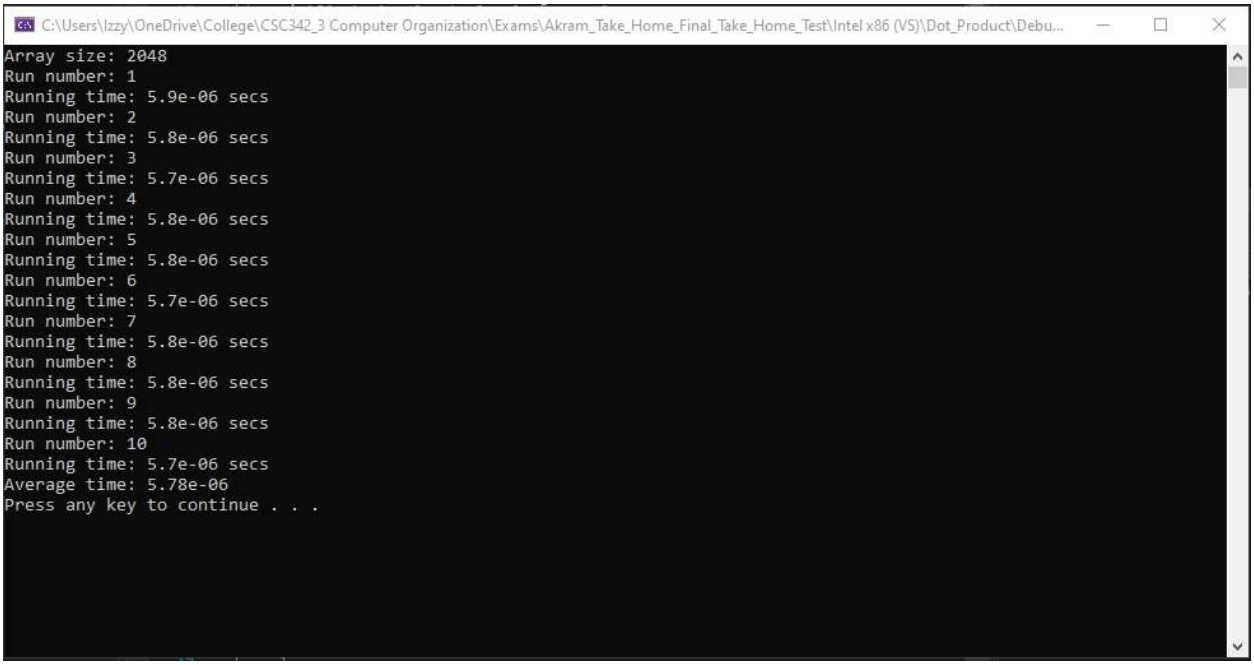


Fig 11g: Dot_Prod_Pntr (2^11) w/0 Qpar and arch

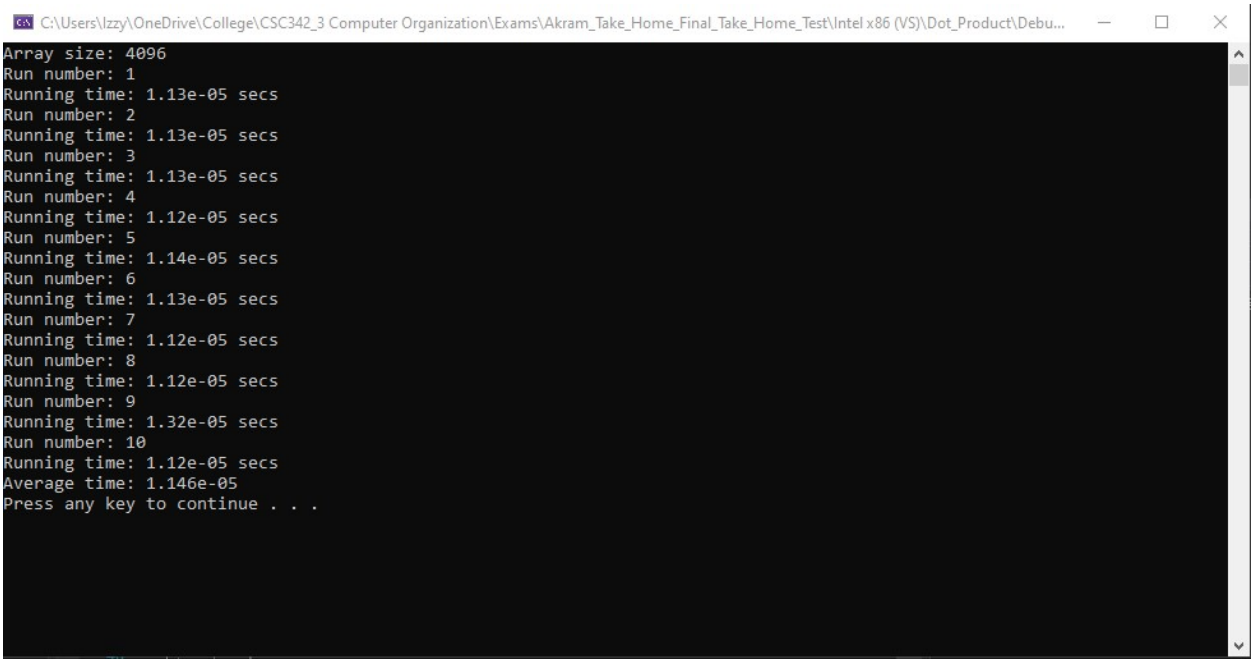


Fig 11h: Dot_Prod_Pntr (2^12) w/0 Qpar and arch

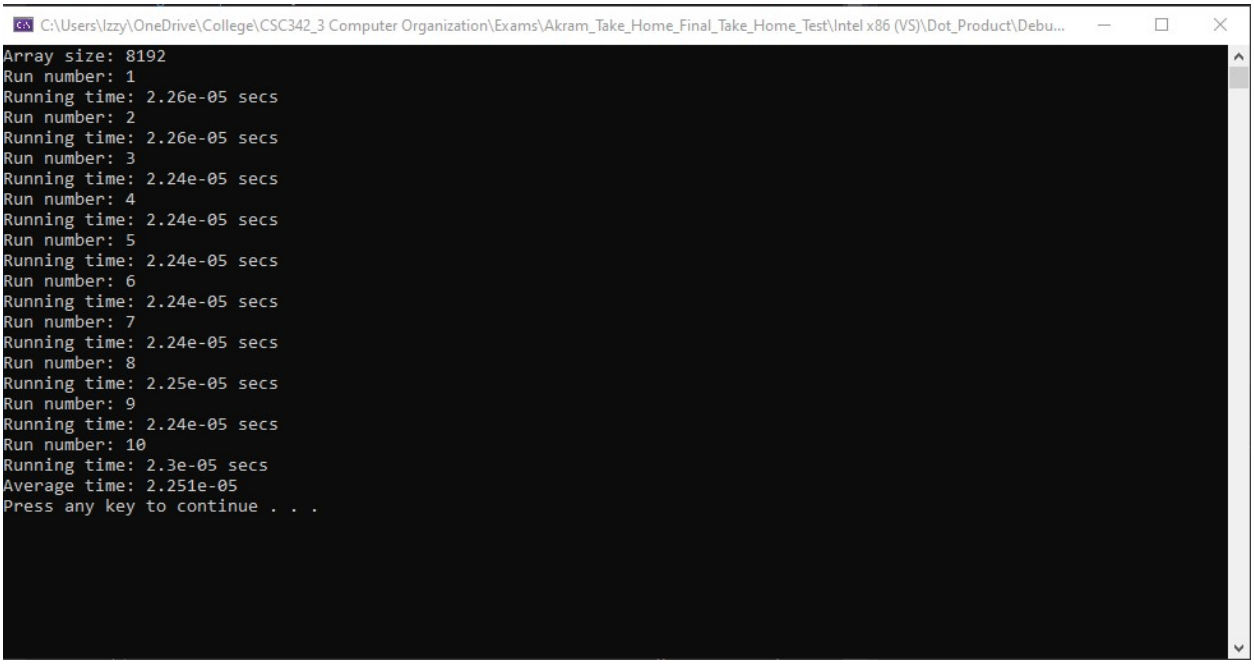


Fig 11i: Dot_Prod_Pntr (2^13) w/0 Qpar and arch

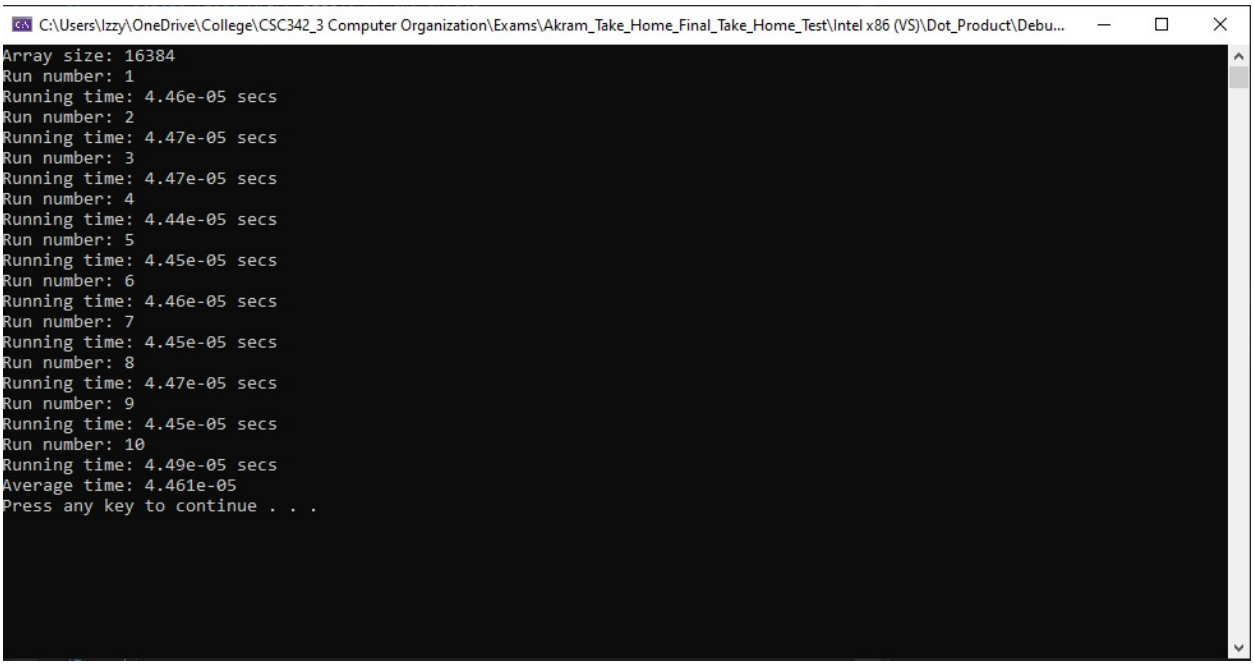


Fig 11j: Dot_Prod_Pntr (2^14) w/0 Qpar and arch

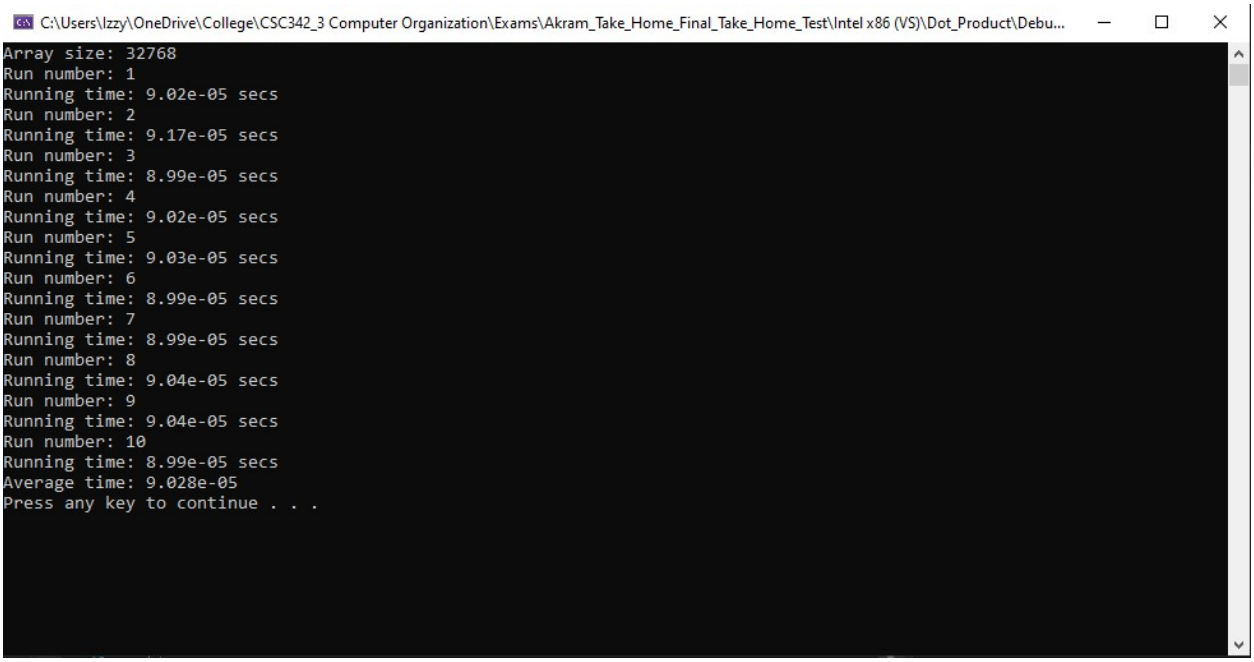


Fig 11k: Dot_Prod_Pntr (2^15) w/0 Qpar and arch

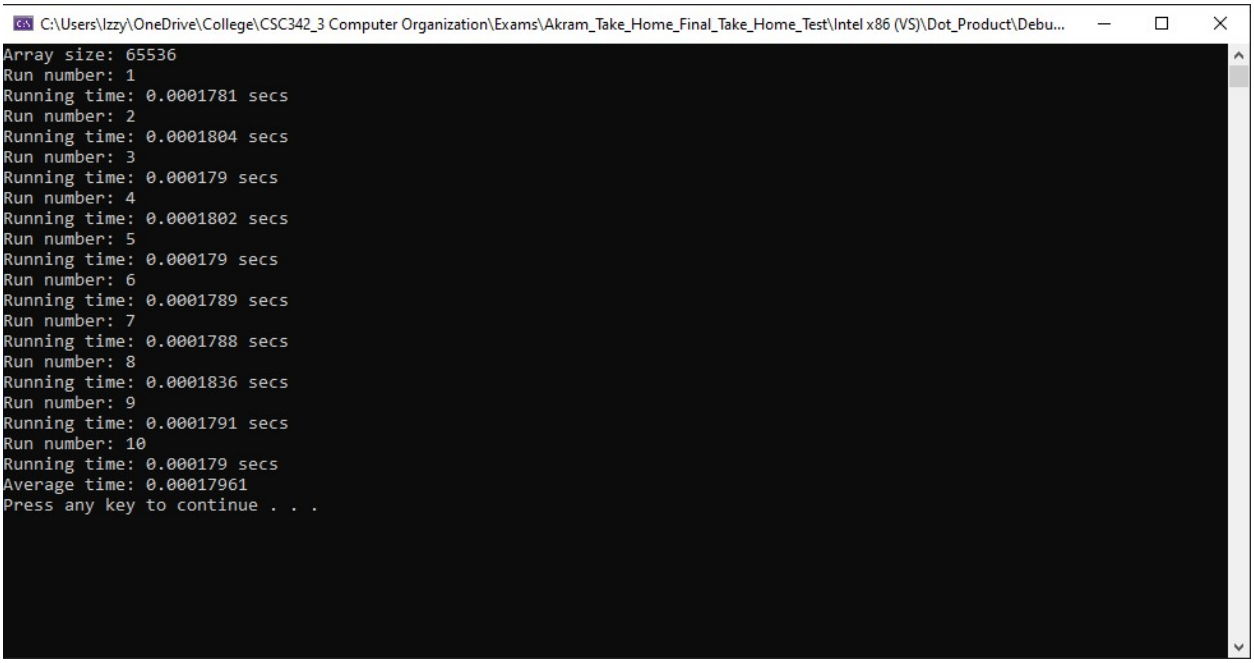


Fig 11l: Dot_Prod_Pntr (2^16) w/0 Qpar and arch

Dot Product (Pointer) [Compiler Generated Optimization]

Optimizations enabled.

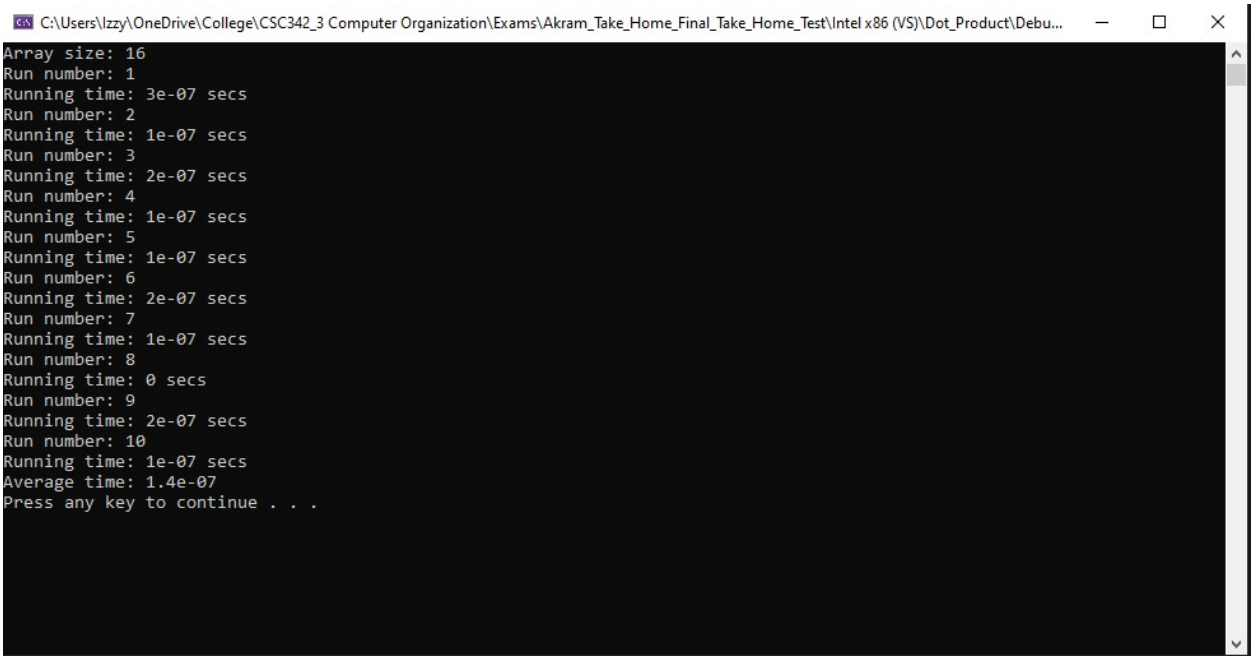


Fig 12: Dot_Prod_Pntr (16) w/ Qpar and arch

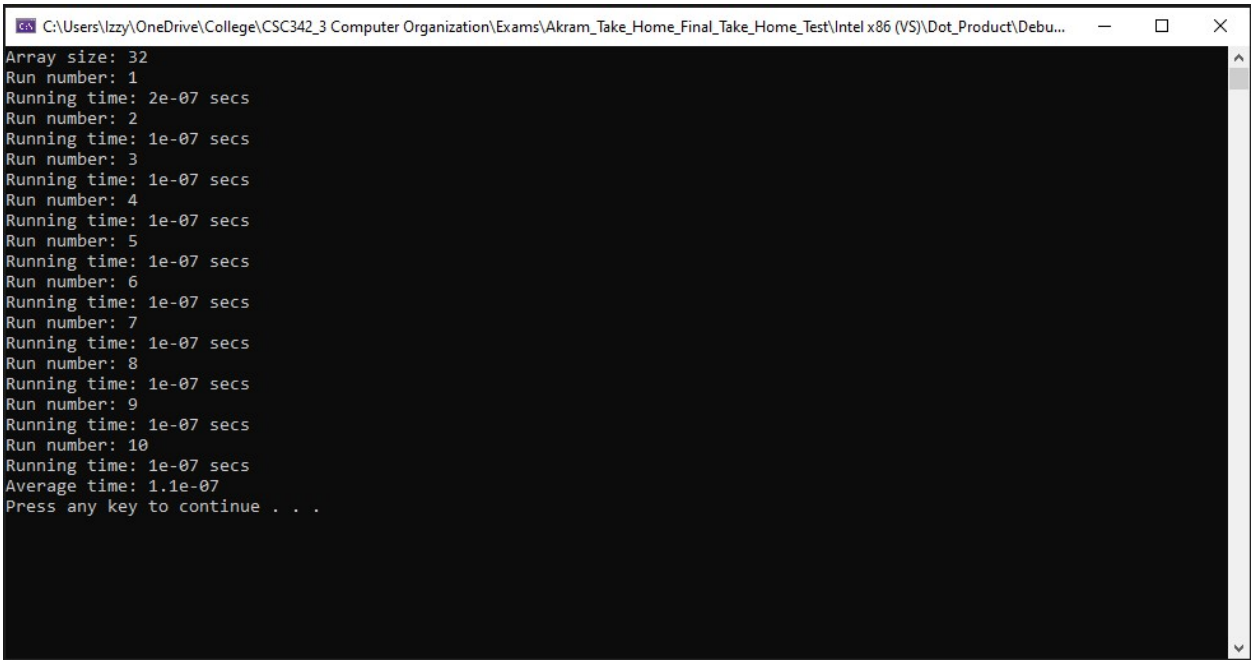


Fig 12a: Dot_Prod_Pntr (32) w/ Qpar and arch

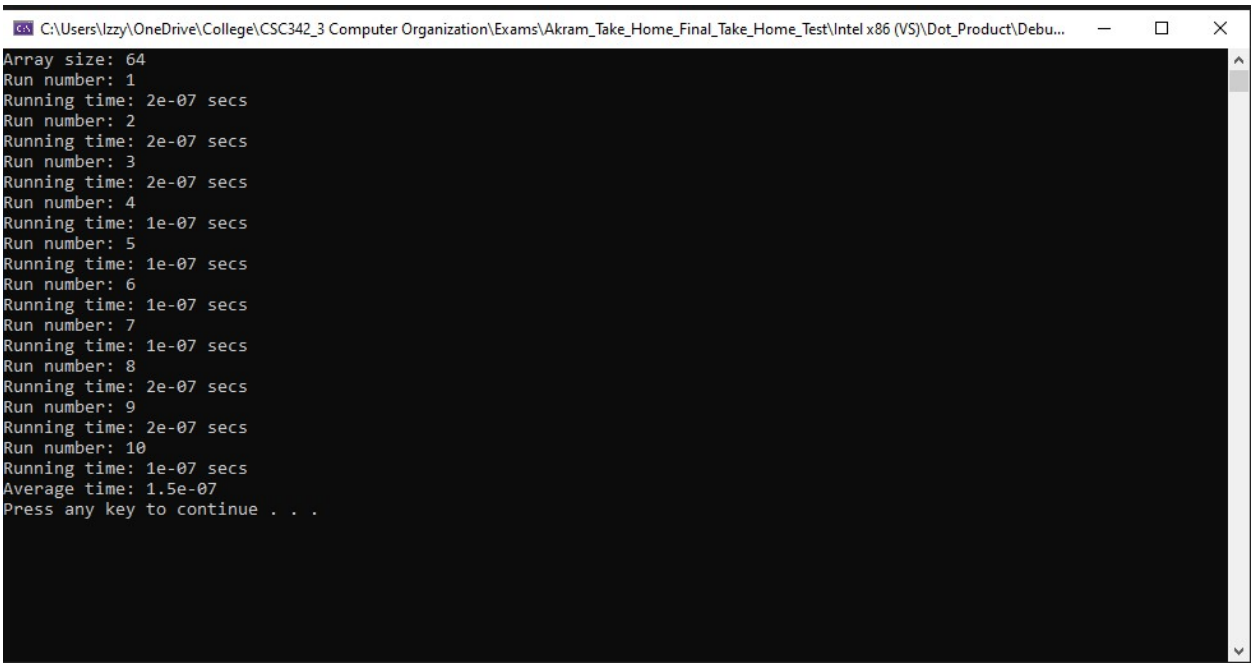


Fig 12b: Dot_Prod_Pntr (64) w/ Qpar and arch

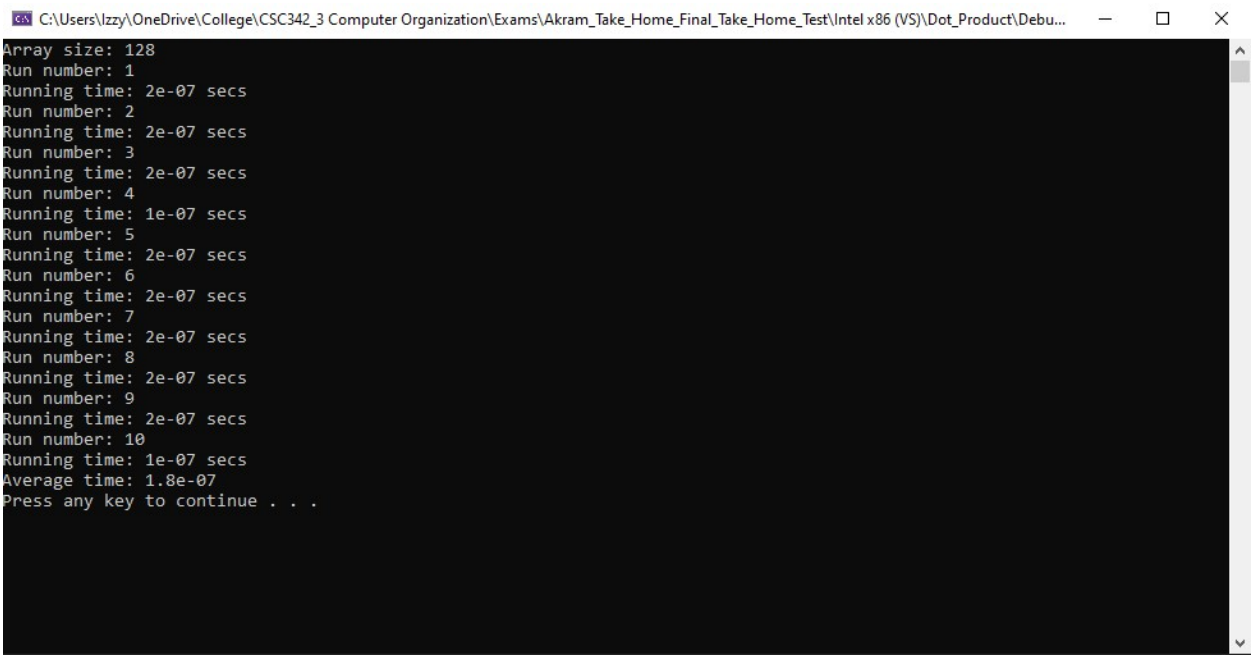


Fig 12c: Dot_Prod_Pntr (128) w/ Qpar and arch

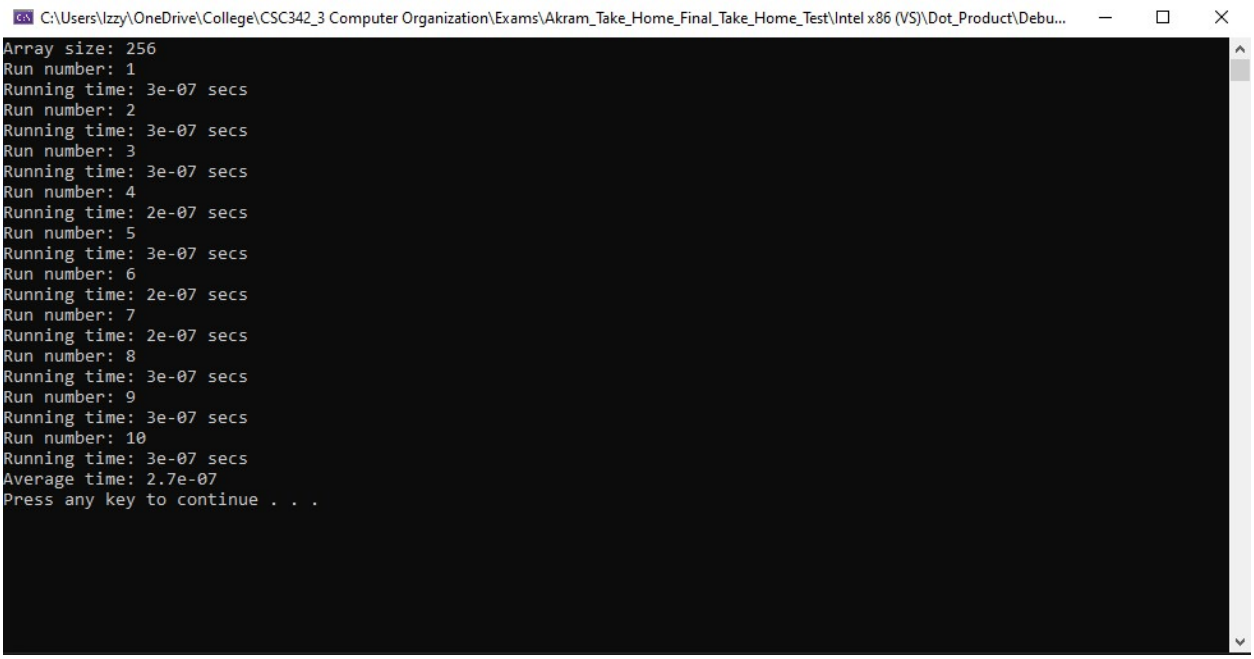


Fig 12d: Dot_Prod_Pntr (256) w/ Qpar and arch

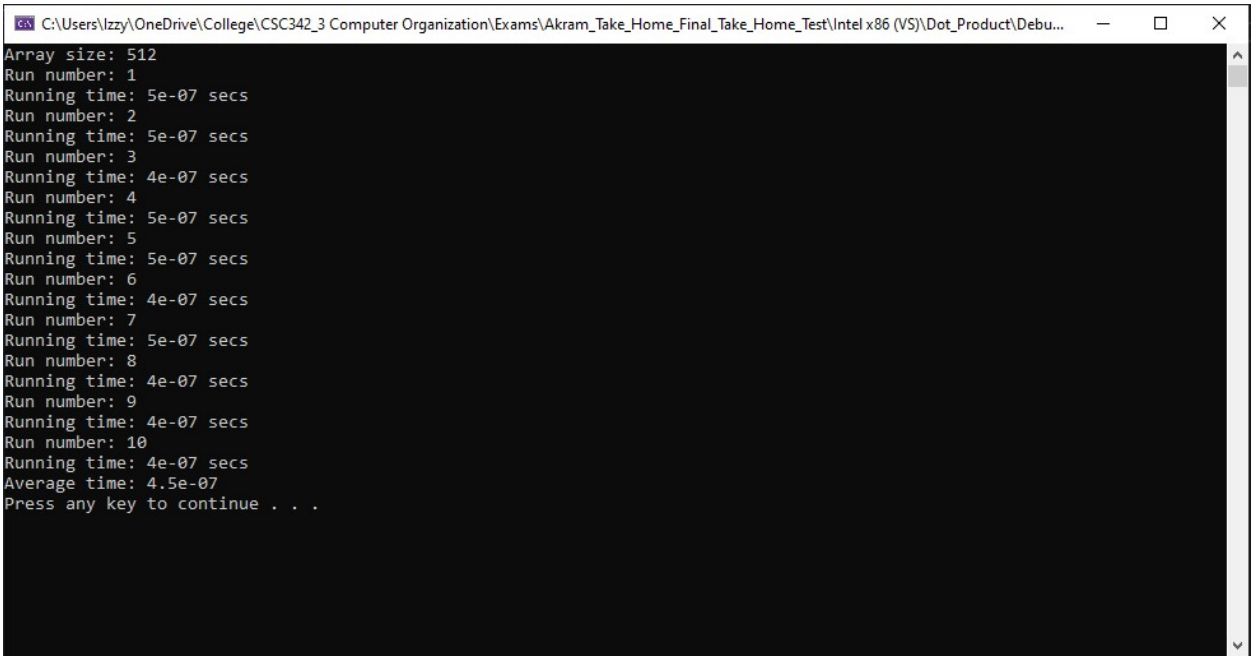


Fig 12e: Dot_Prod_Pntr (512) w/ Qpar and arch

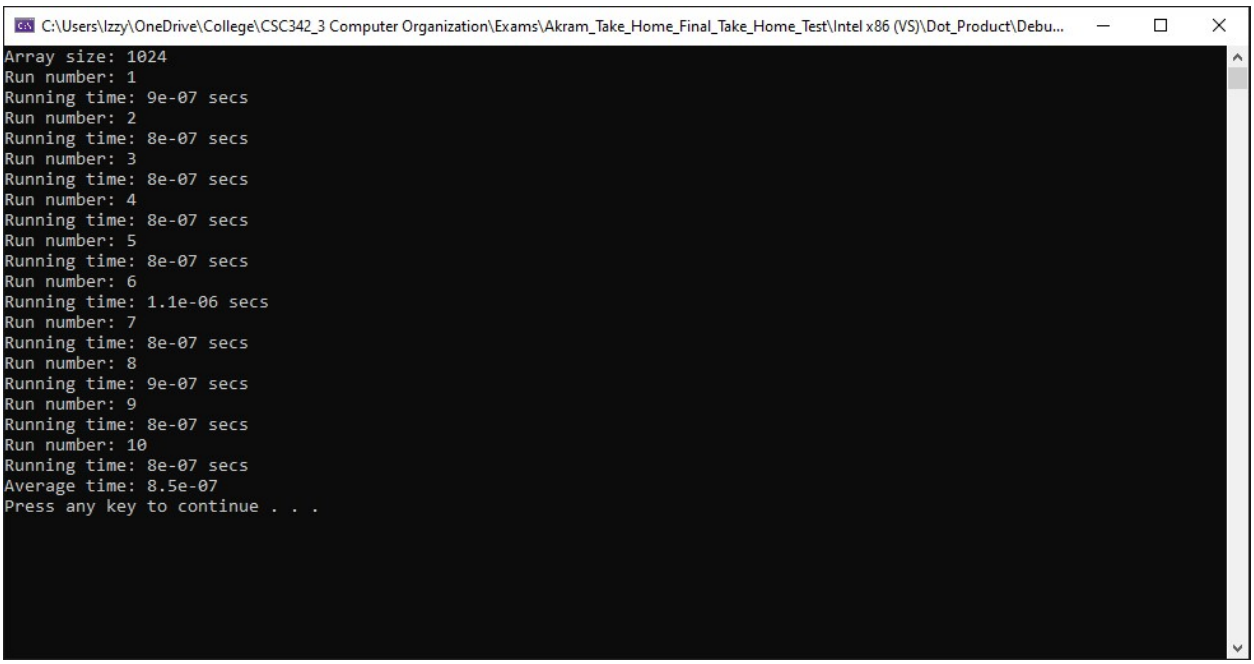


Fig 12f: Dot_Prod_Pntr (1024) w/ Qpar and arch

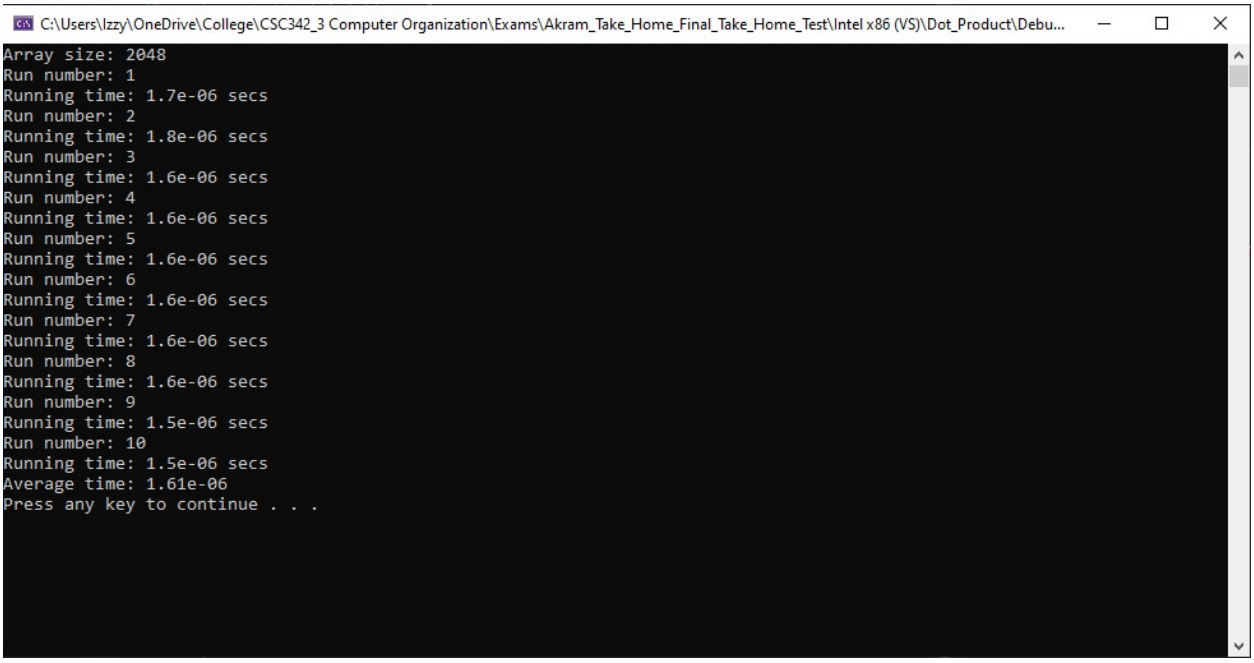


Fig 12g: Dot_Prod_Pntr (2^11) w/ Qpar and arch

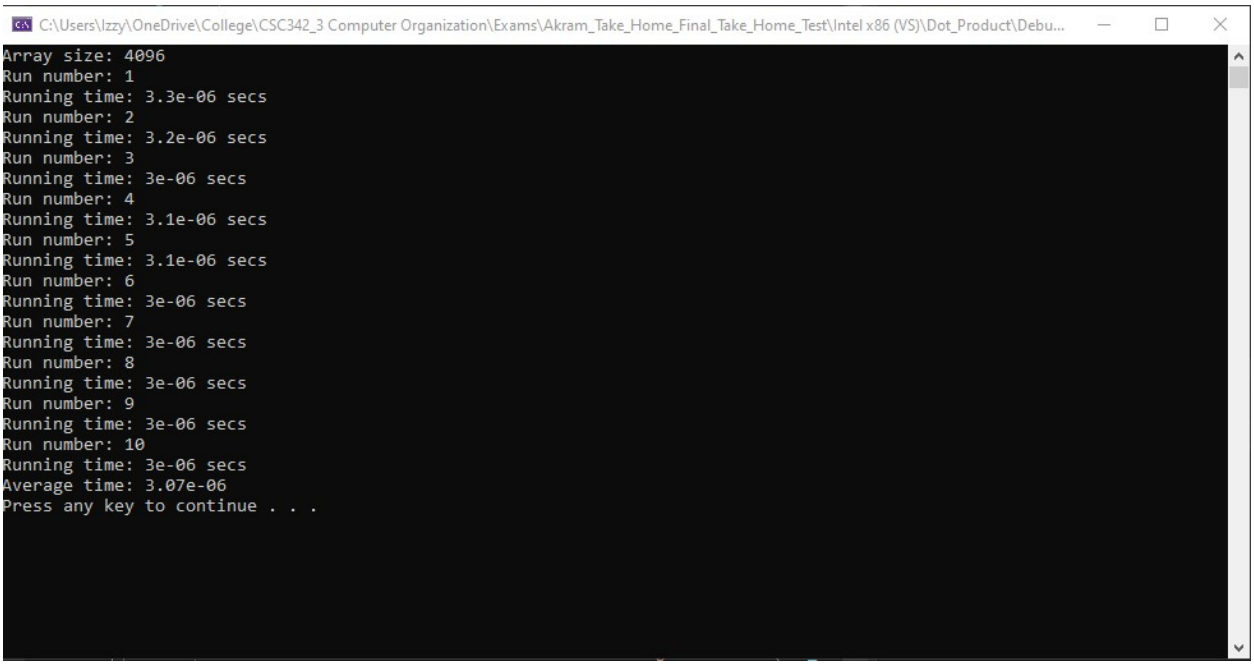


Fig 12h: Dot_Prod_Pntr (2^12) w/ Qpar and arch

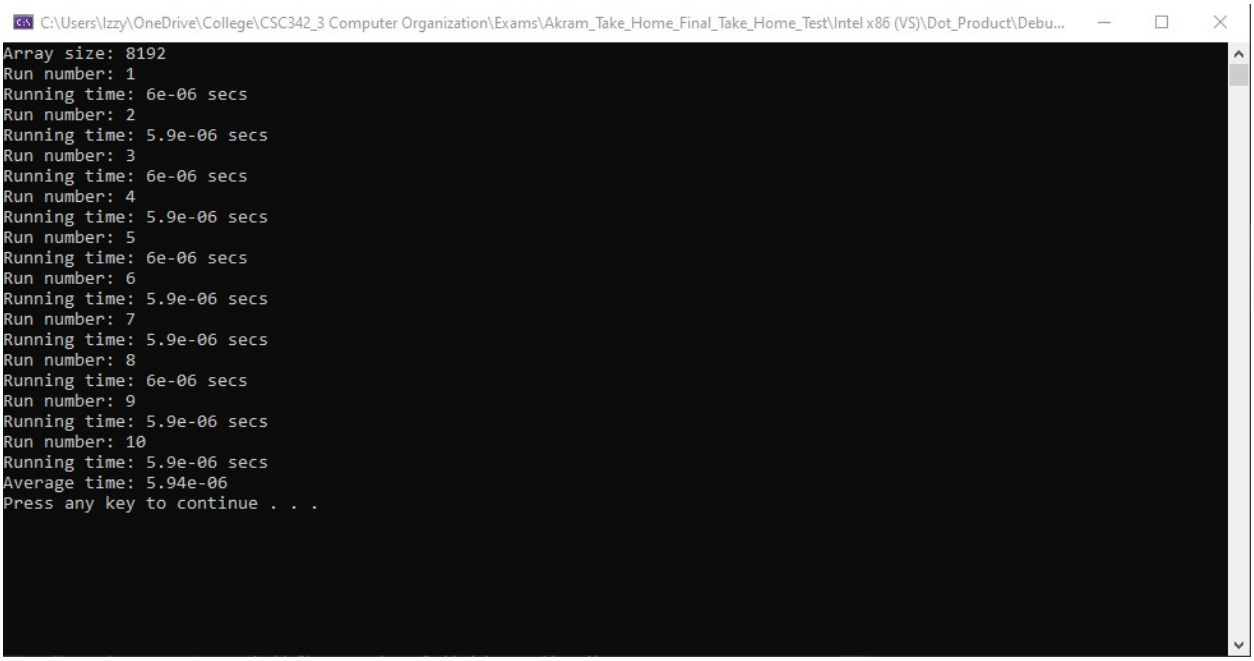


Fig 12i: Dot_Prod_Pntr (2^13) w/ Qpar and arch

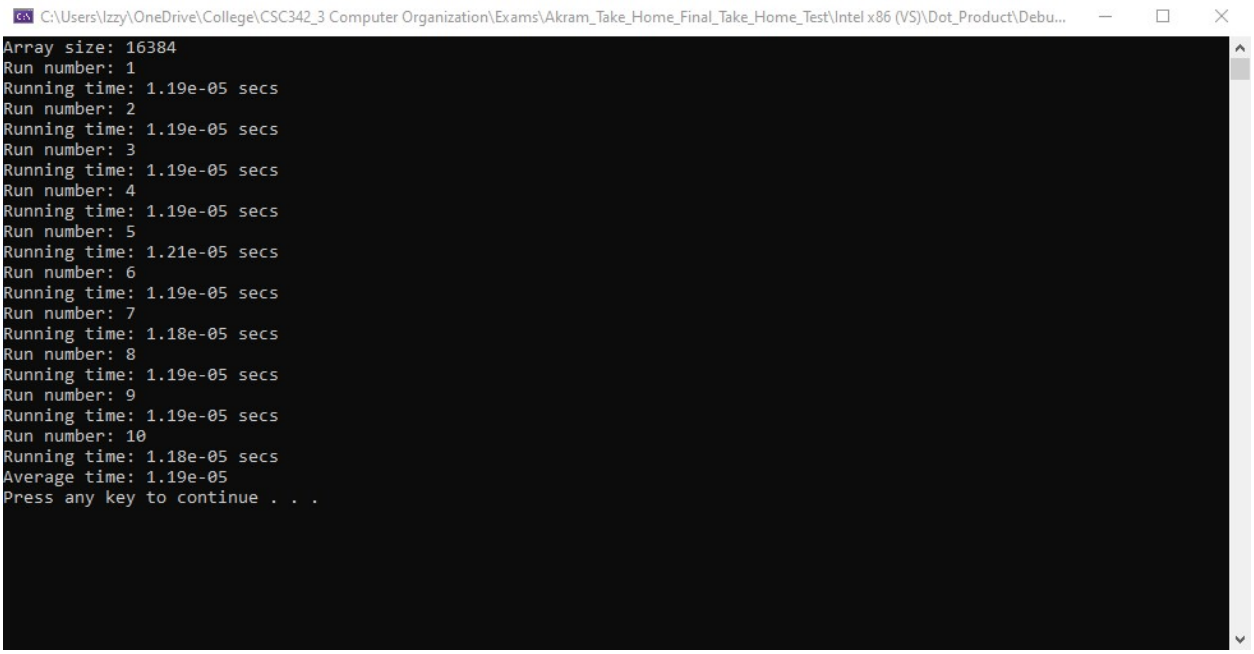


Fig 12j: Dot_Prod_Pntr (2^14) w/ Qpar and arch

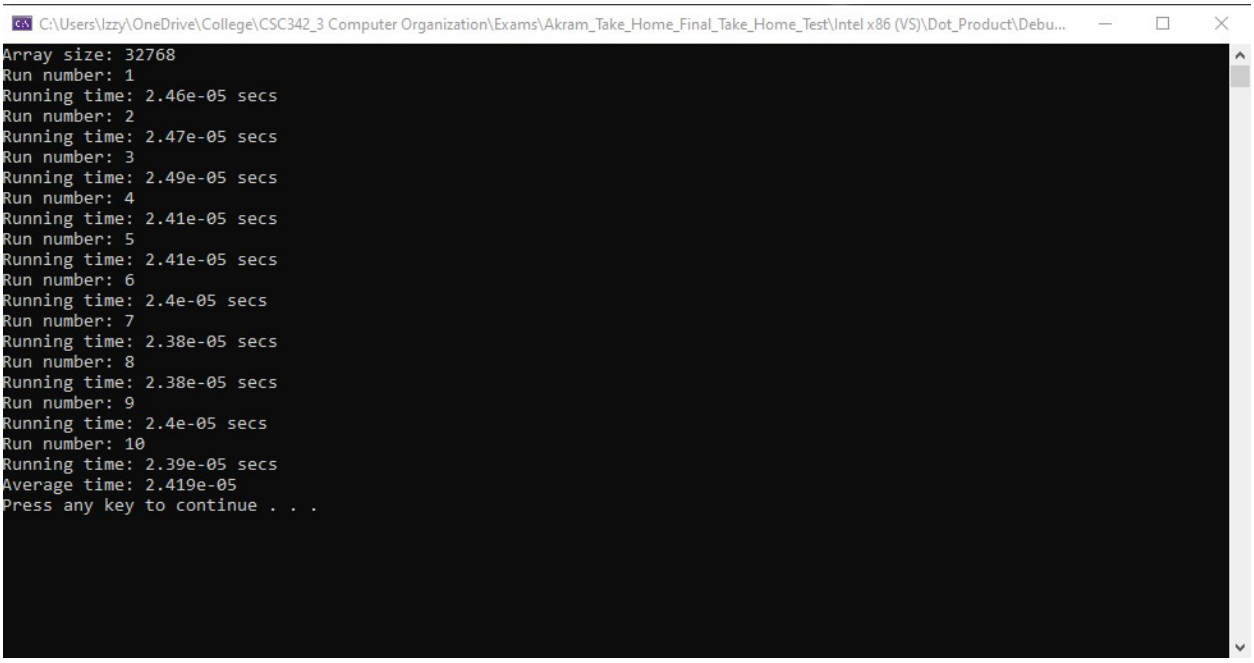


Fig 12k: Dot_Prod_Pntr (2^15) w/ Qpar and arch

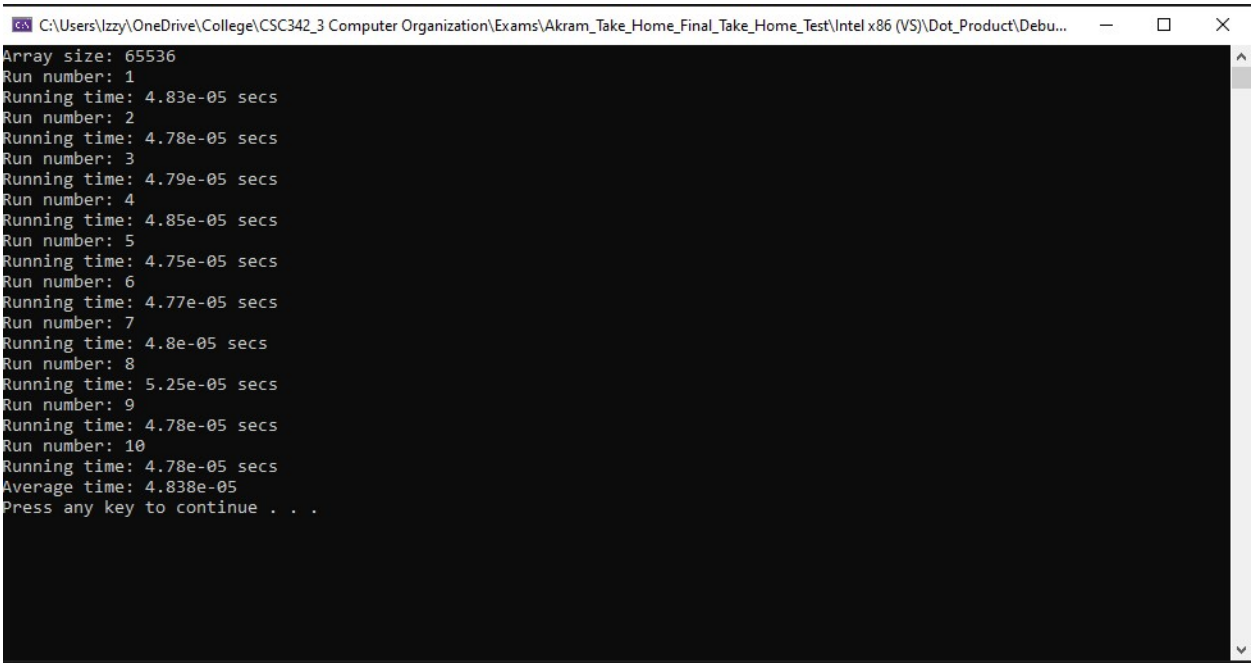


Fig 12l: Dot_Prod_Pntr (2^16) w/ Qpar and arch

Dot Product (Pointer) [Manually Optimized]

Manually optimized compiler generated .asm file

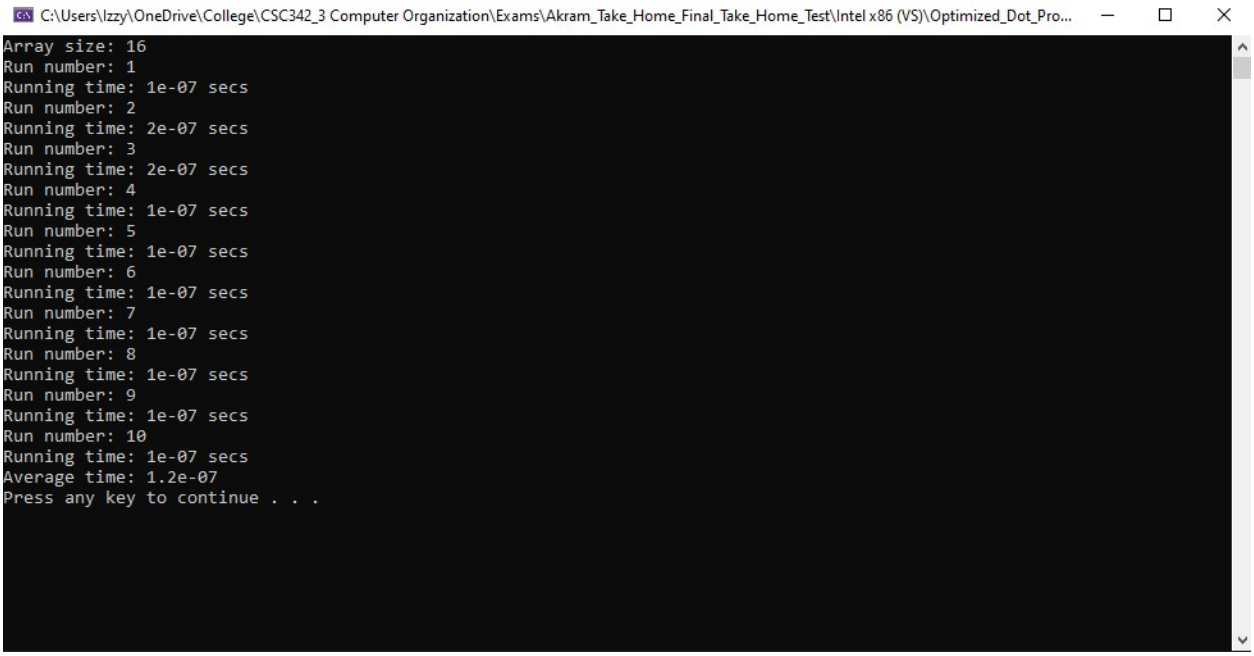


Fig 13: Op_Dot_Prod_Pntr (16) w/ Qpar and arch & Manual optimization

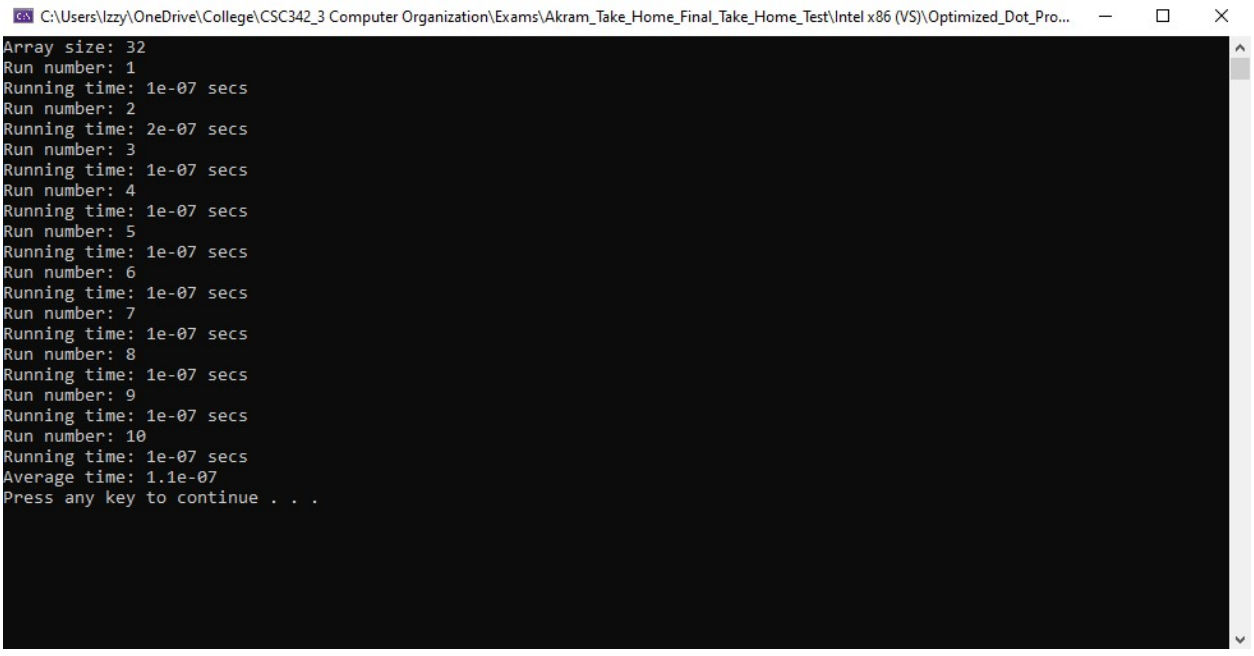


Fig 13a: Op_Dot_Prod_Pntr (32) w/ Qpar and arch & Manual optimization

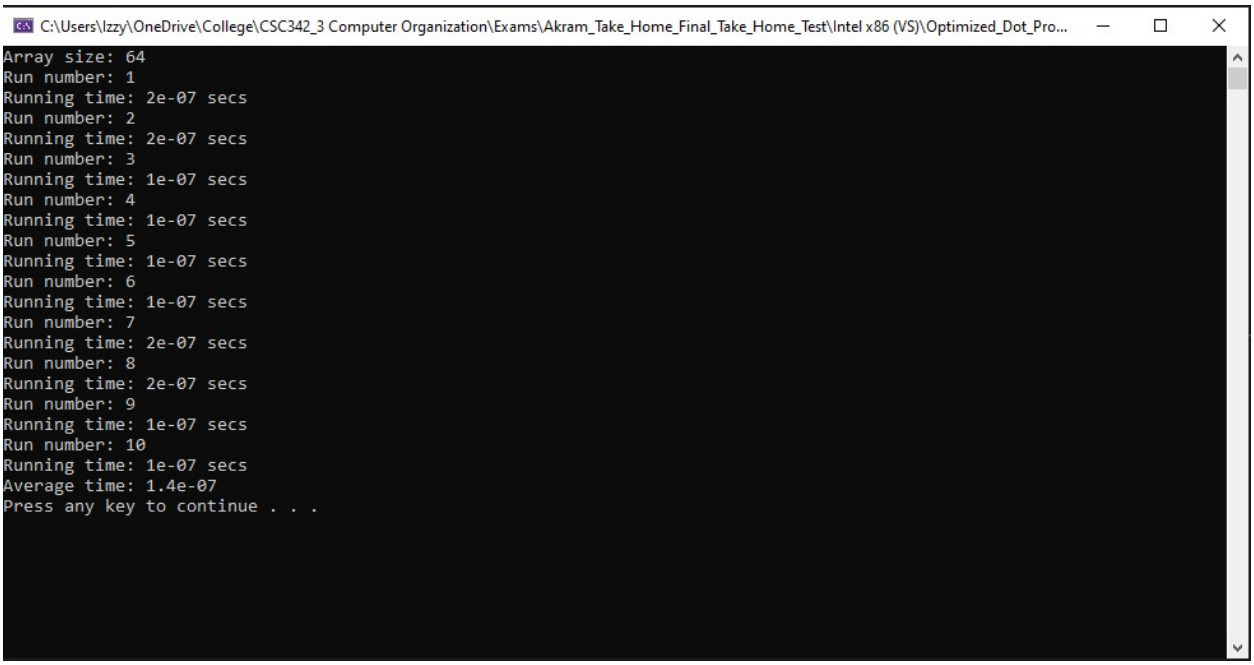


Fig 13b: Op_Dot_Prod_Pntr (64) w/ Qpar and arch & Manual optimization

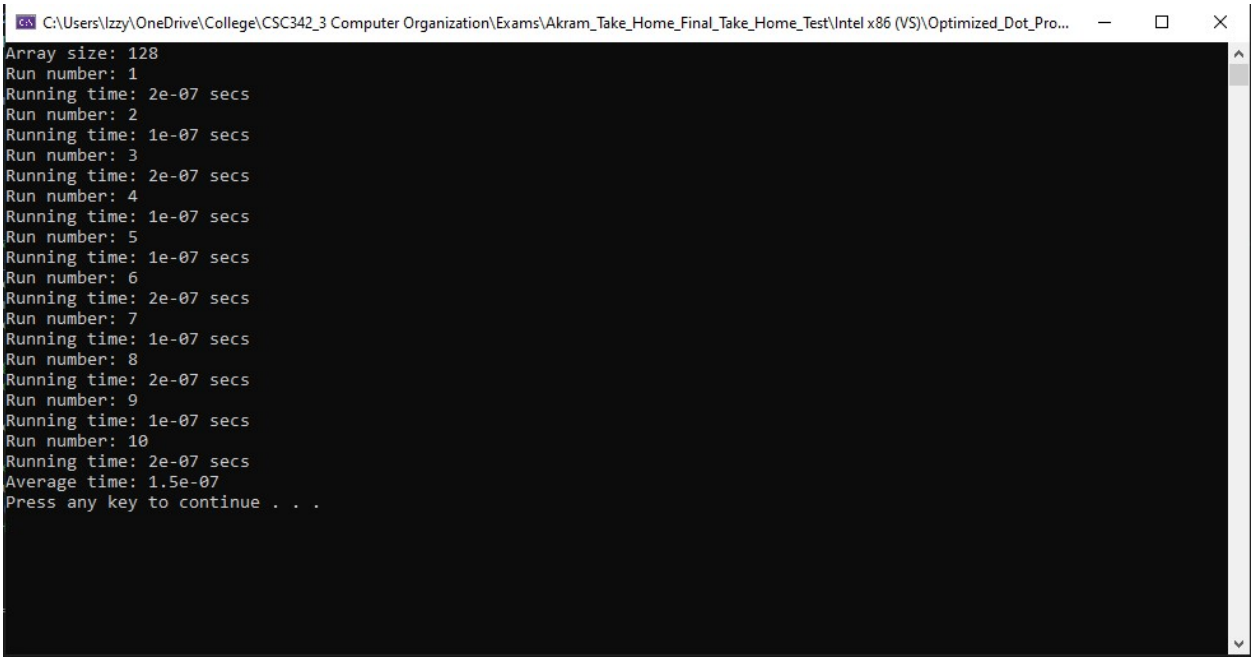


Fig 13c: Op_Dot_Prod_Pntr (128) w/ Qpar and arch & Manual optimization

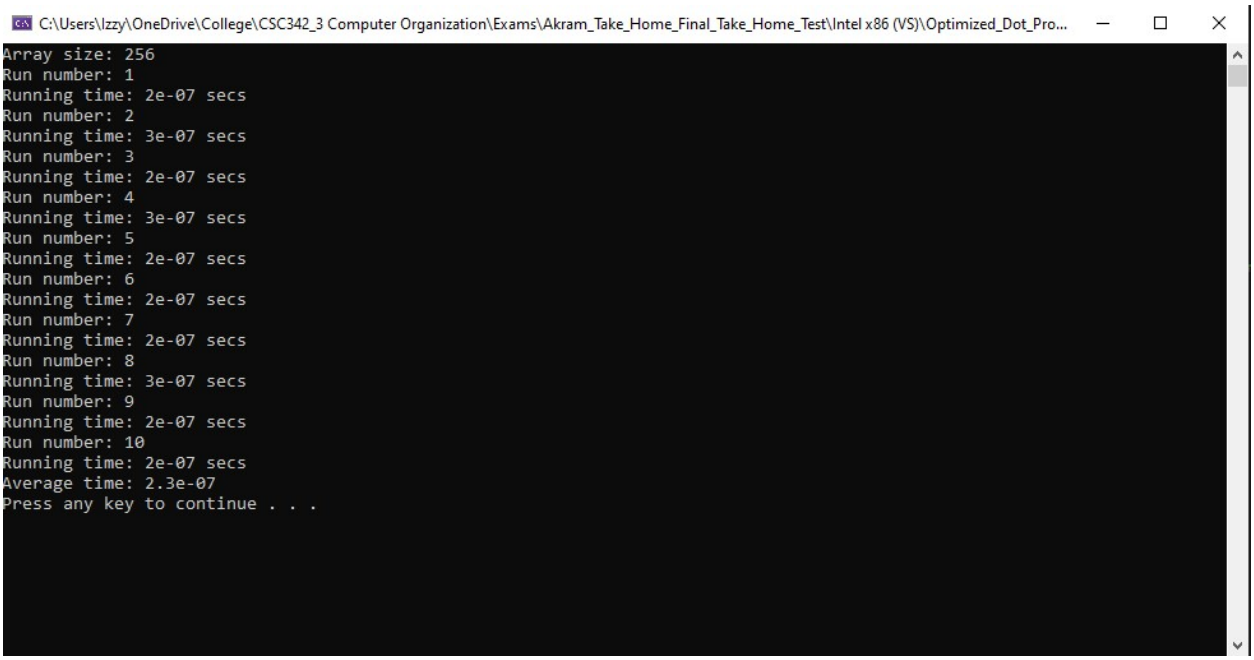


Fig 13d: Op_Dot_Prod_Pntr (256) w/ Qpar and arch & Manual optimization

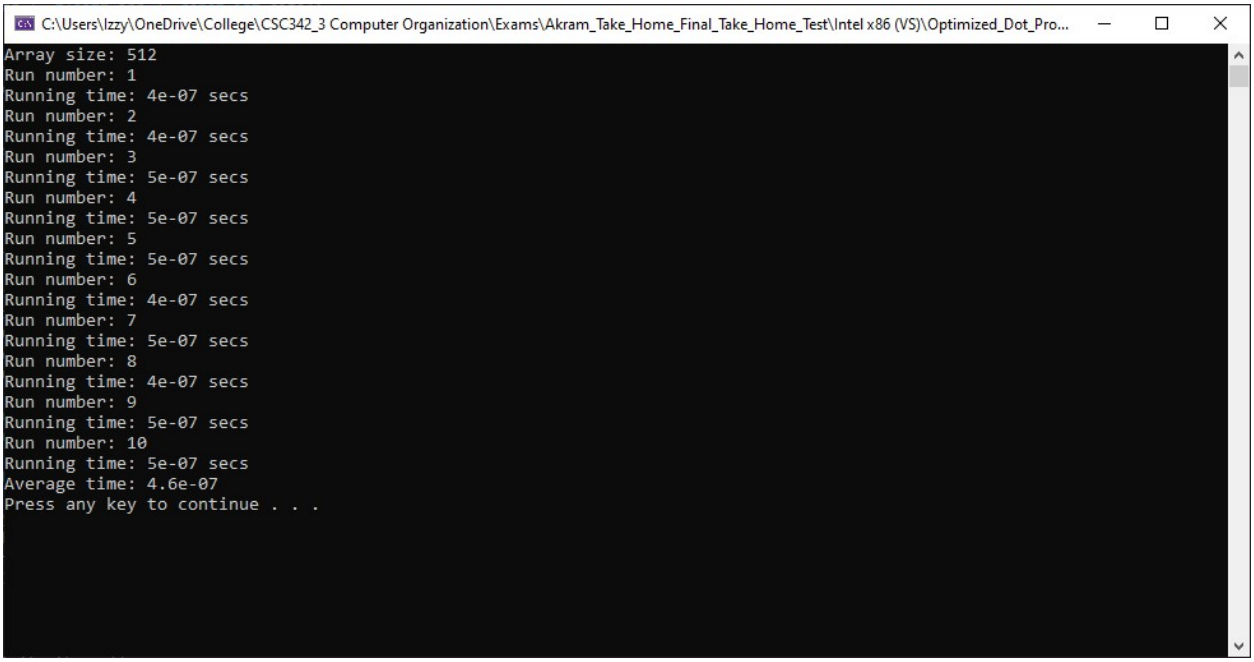


Fig 13e: Op_Dot_Prod_Pntr (512) w/ Qpar and arch & Manual optimization

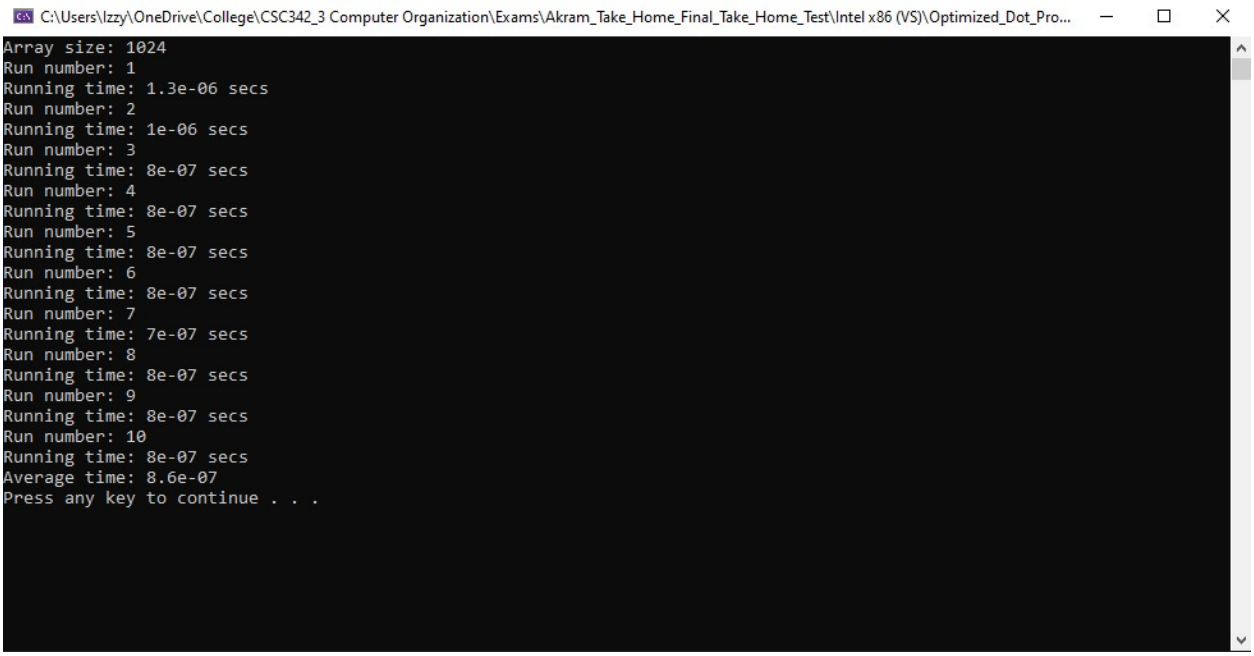


Fig 13f: Op_Dot_Prod_Pntr (1024) w/ Qpar and arch & Manual optimization

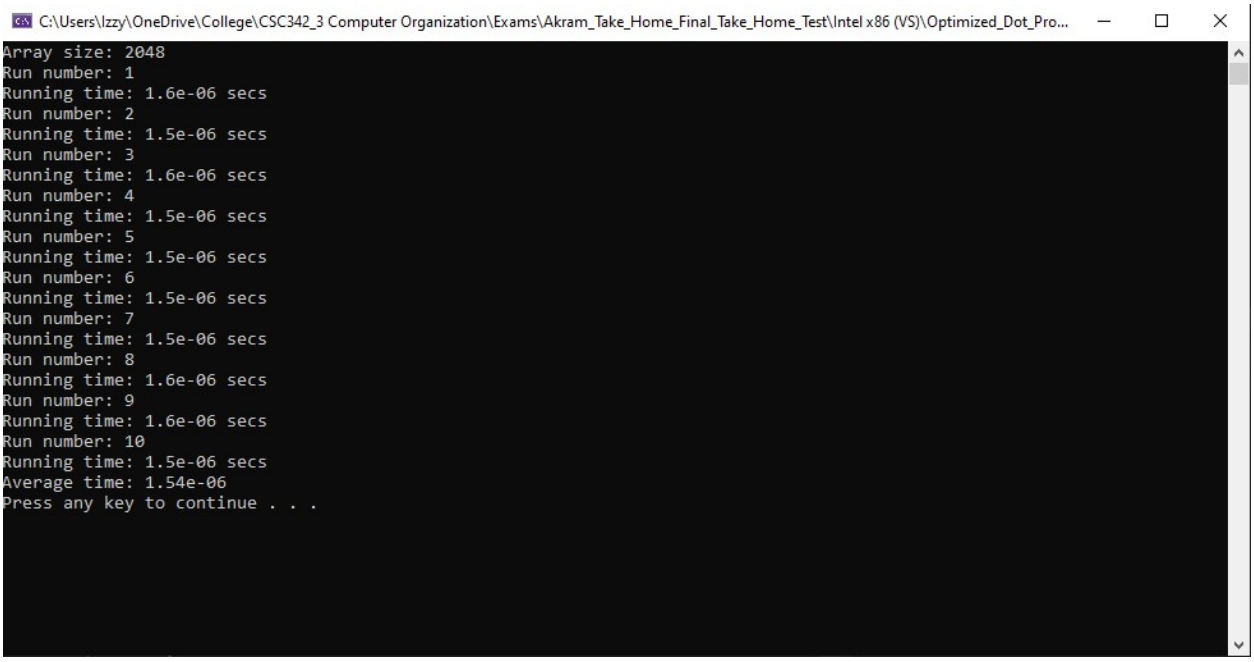


Fig 13g: Op_Dot_Prod_Pntr (2^11) w/ Qpar and arch & Manual optimization

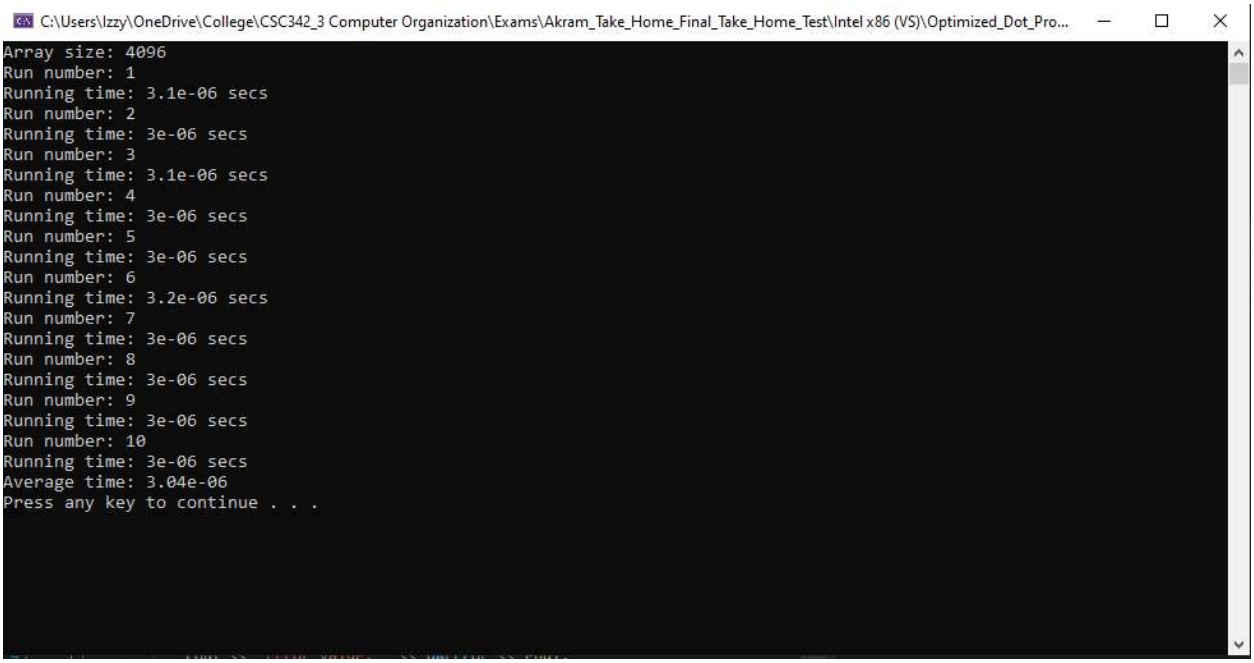


Fig 13h: Op_Dot_Prod_Pntr (2^12) w/ Qpar and arch & Manual optimization

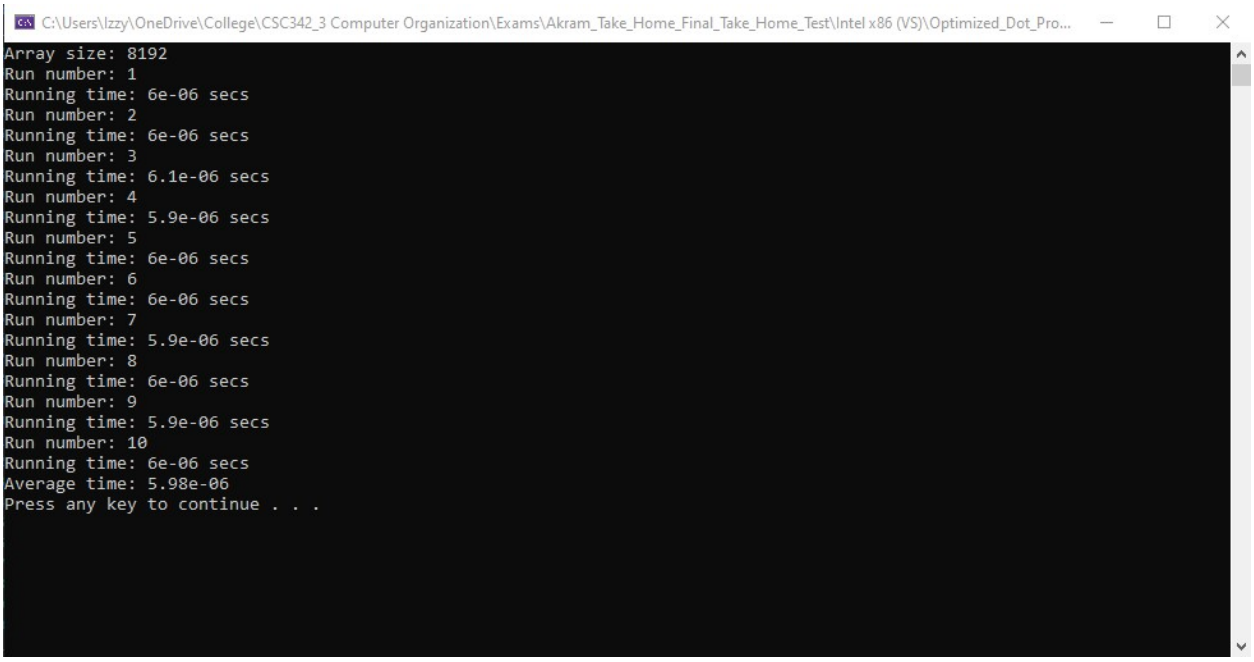


Fig 13i: Op_Dot_Prod_Pntr (2^13) w/ Qpar and arch & Manual optimization

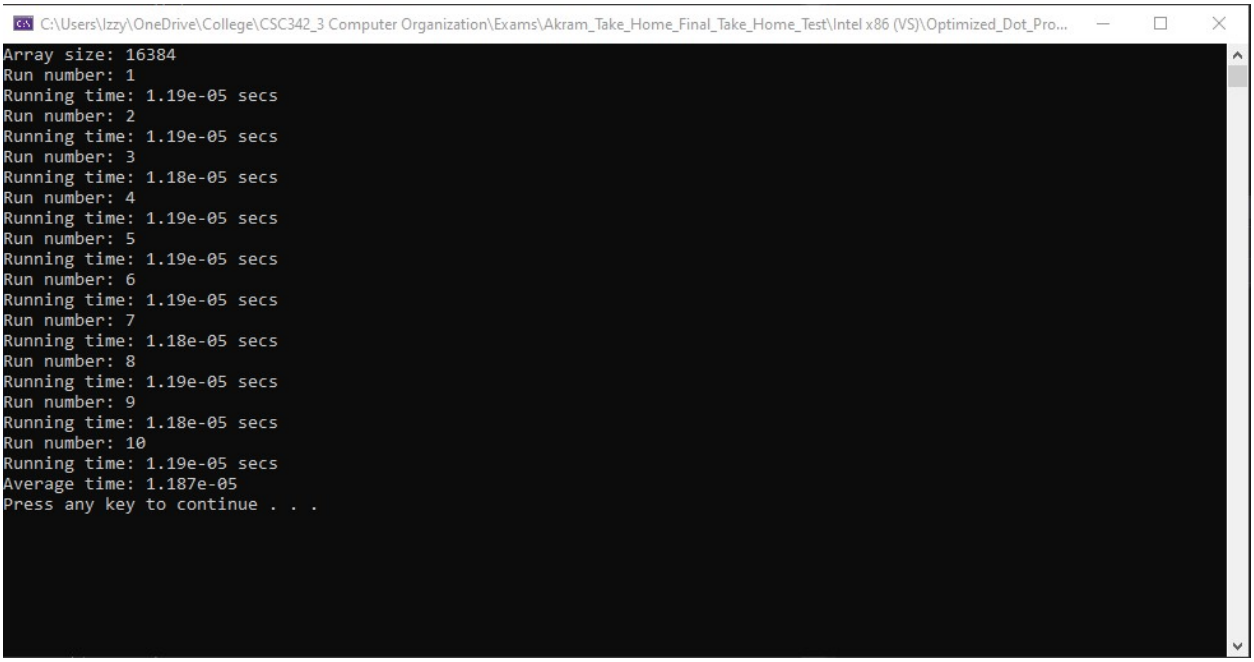


Fig 13j: Op_Dot_Prod_Pntr (2^14) w/ Qpar and arch & Manual optimization

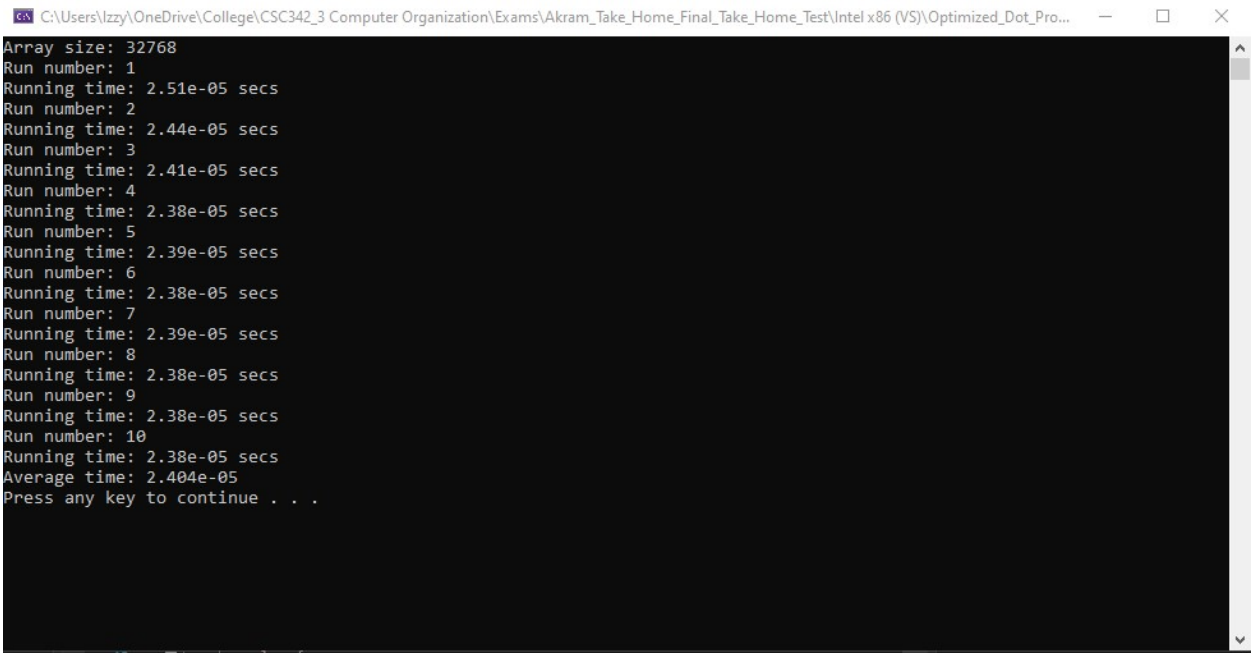


Fig 13k: Op_Dot_Prod_Pntr (2^15) w/ Qpar and arch & Manual optimization

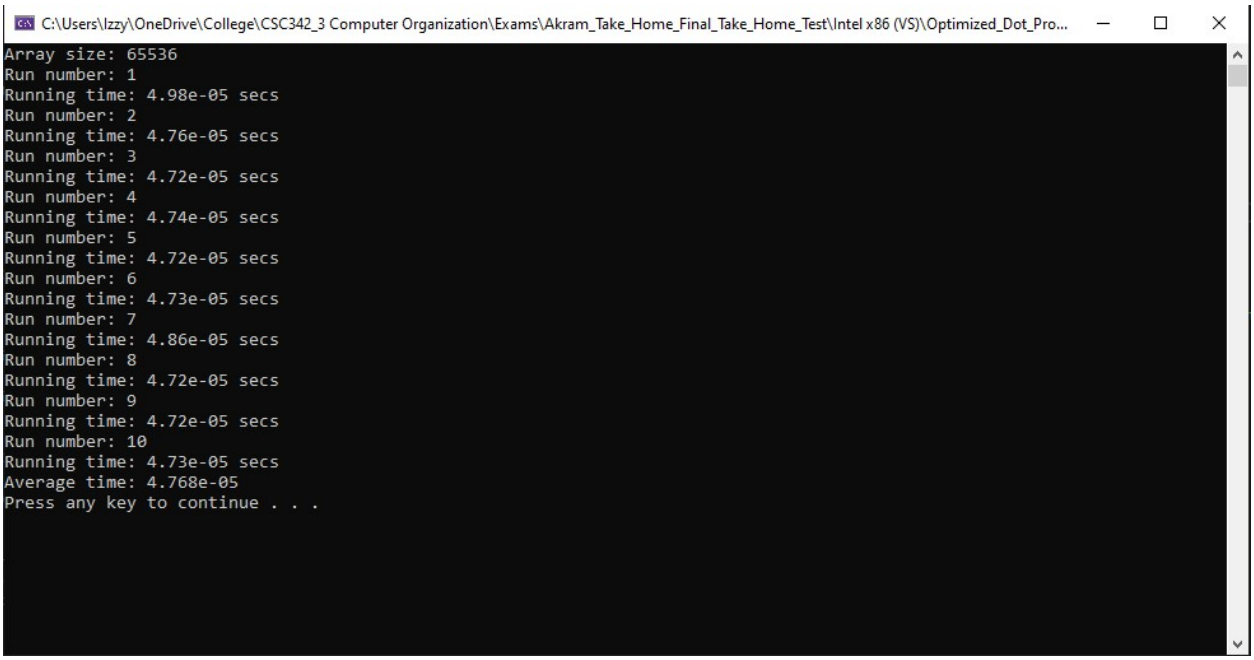


Fig 13l: Op_Dot_Prod_Pntr (2^16) w/ Qpar and arch & Manual optimization

VDPPS

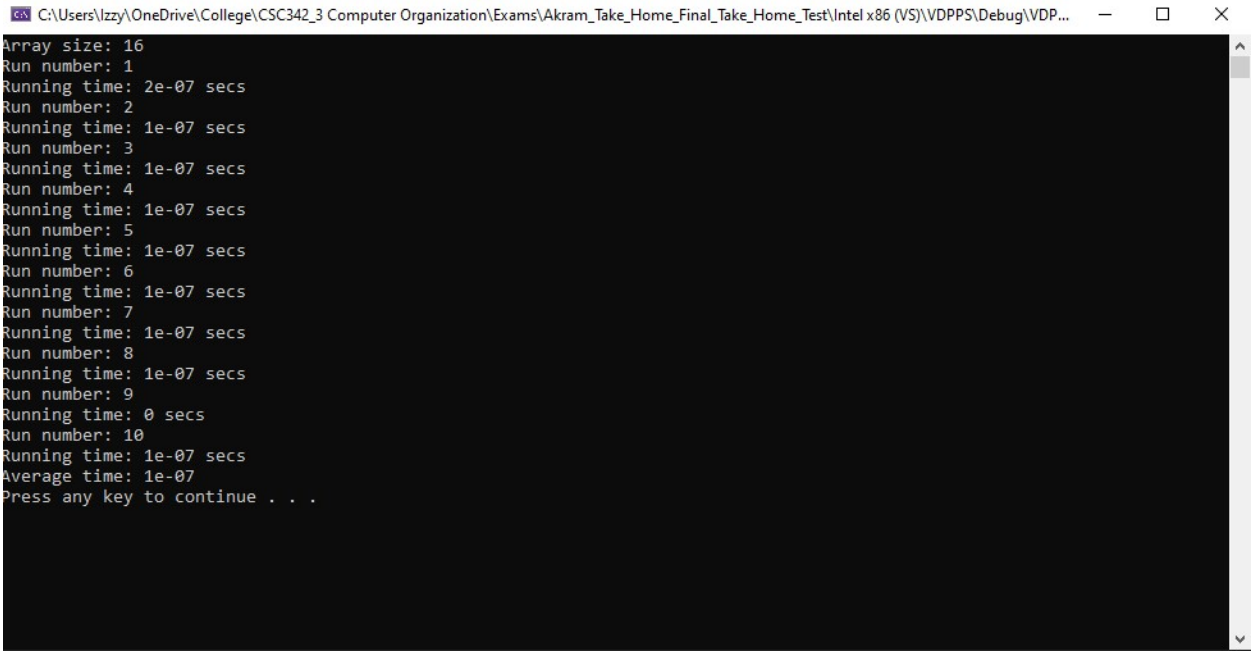


Fig 14: VDPPS_Dot_Prod_Pntr (16) w/ Qpar and arch

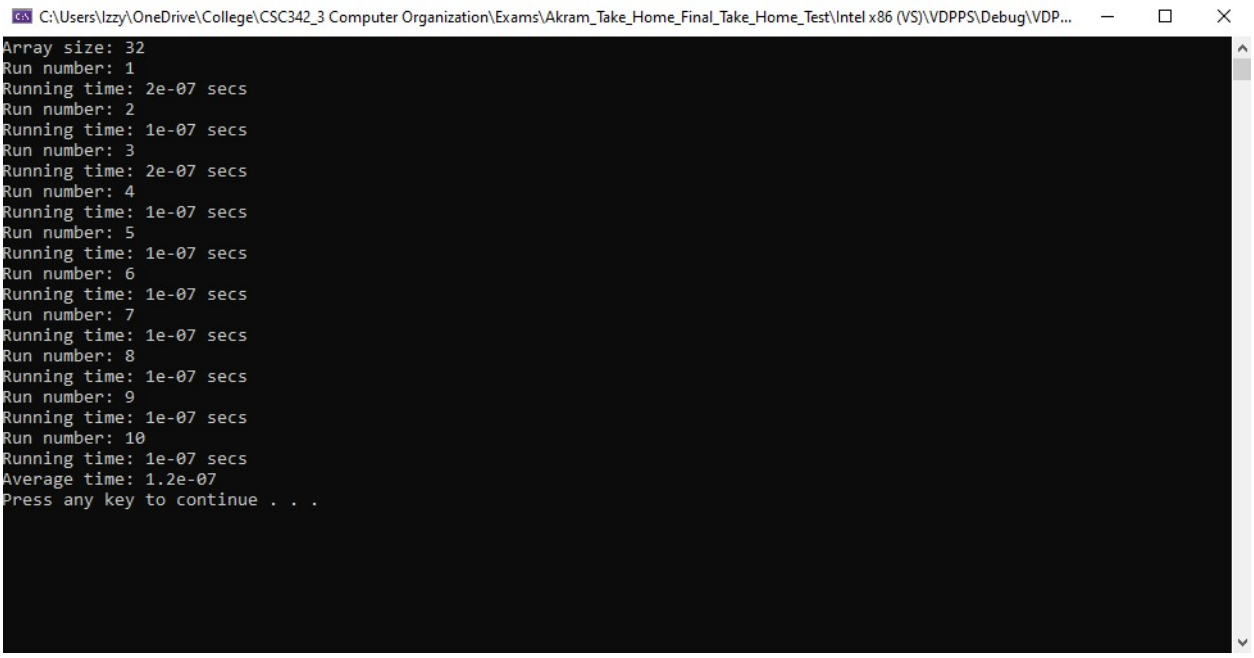


Fig 14a: VDPPS_Dot_Prod_Pntr (32) w/ Qpar and arch

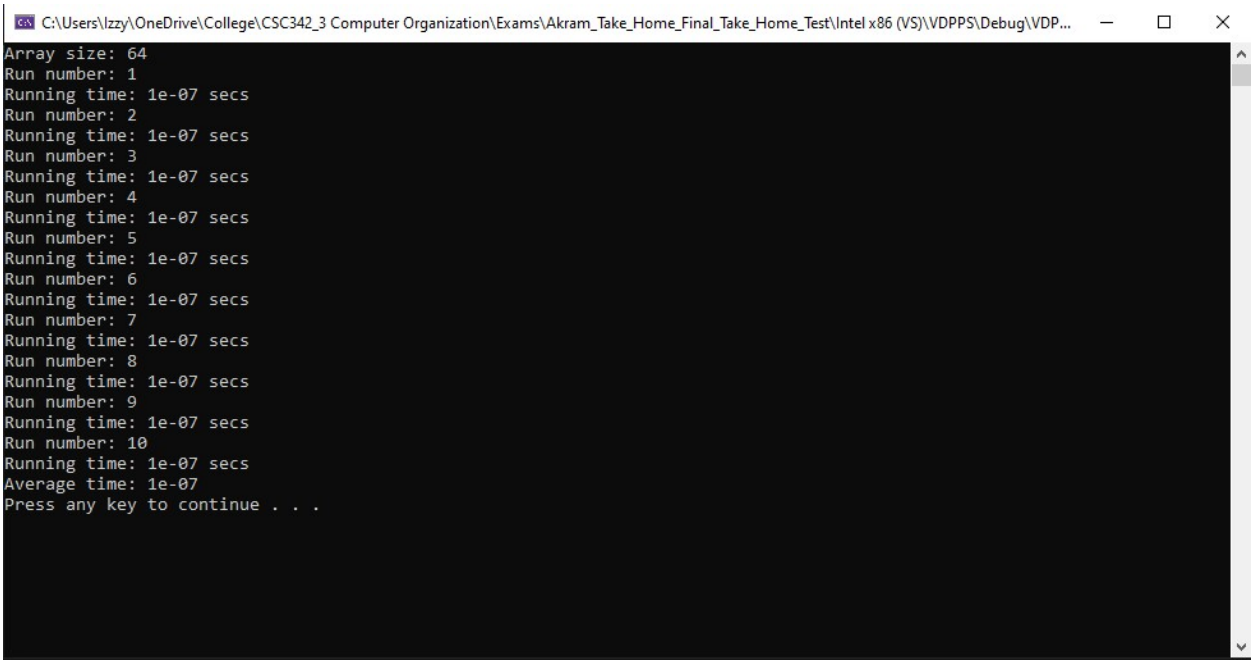


Fig 14b: VDPPS_Dot_Prod_Pntr (64) w/ Qpar and arch

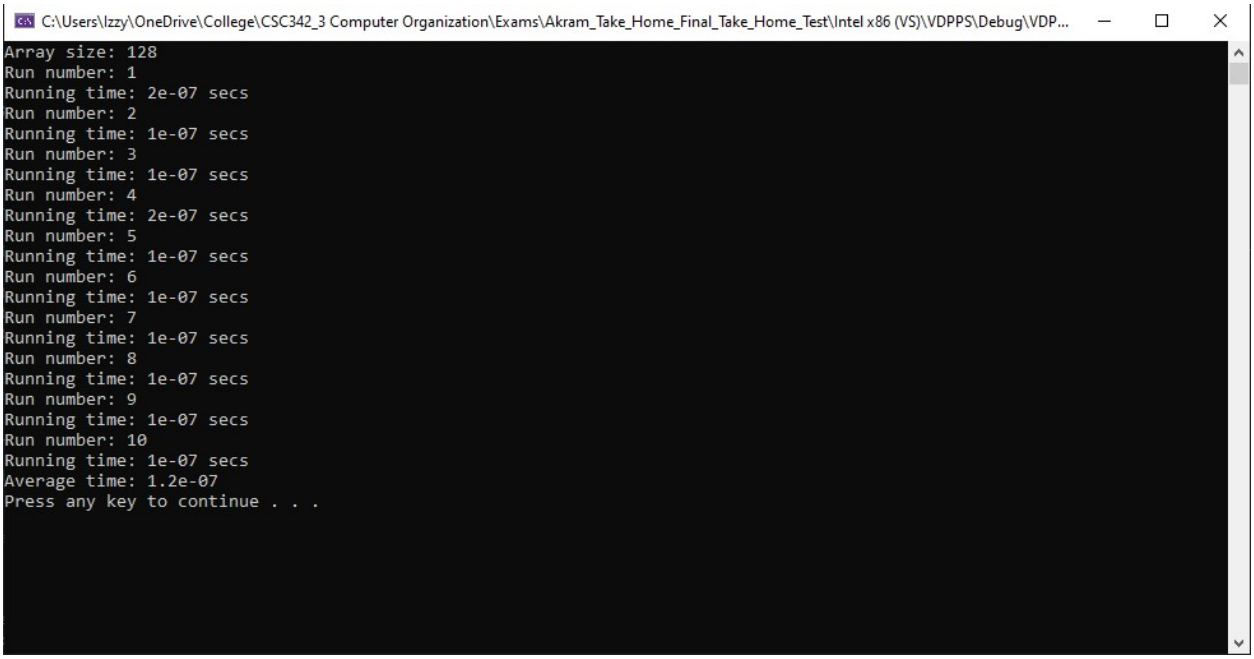


Fig 14c: VDPPS_Dot_Prod_Pntr (128) w/ Qpar and arch

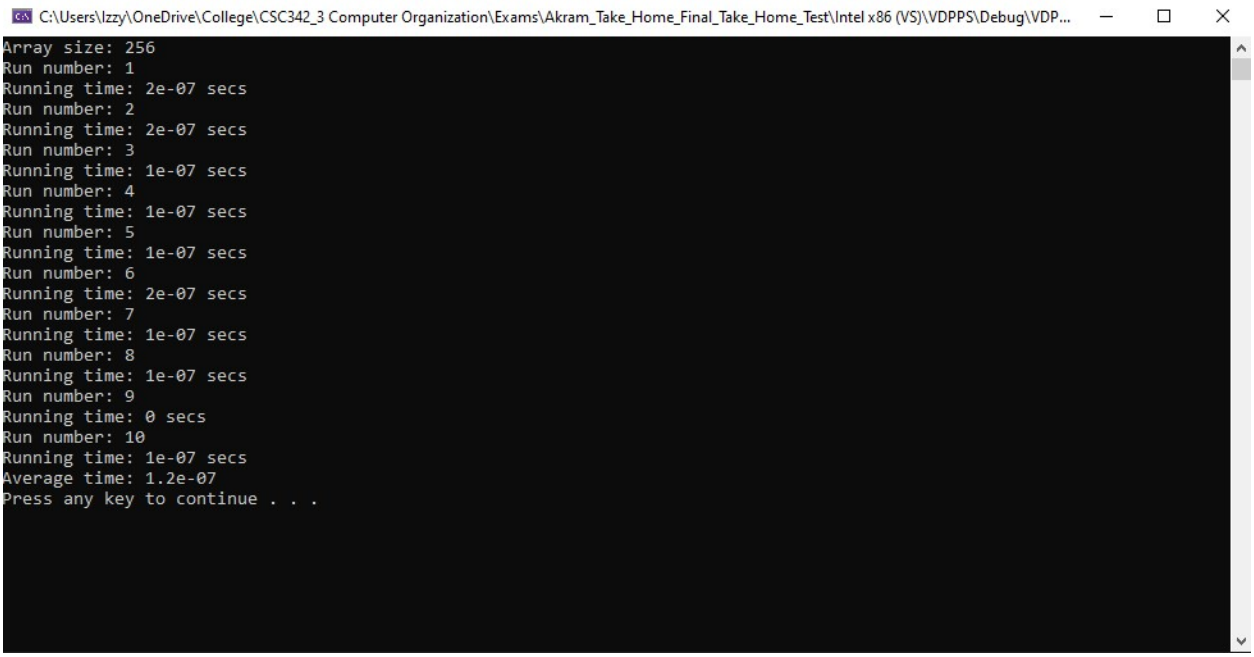


Fig 14d: VDPPS_Dot_Prod_Pntr (256) w/ Qpar and arch

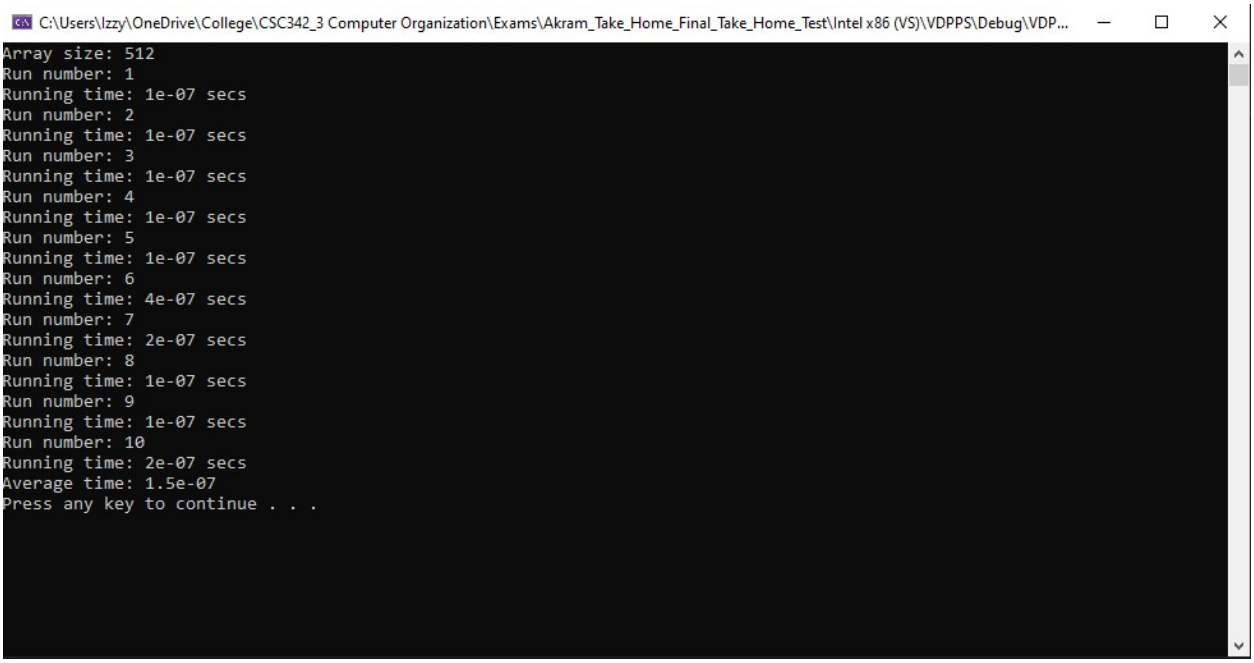


Fig 14e: VDPPS_Dot_Prod_Pntr (512) w/ Qpar and arch

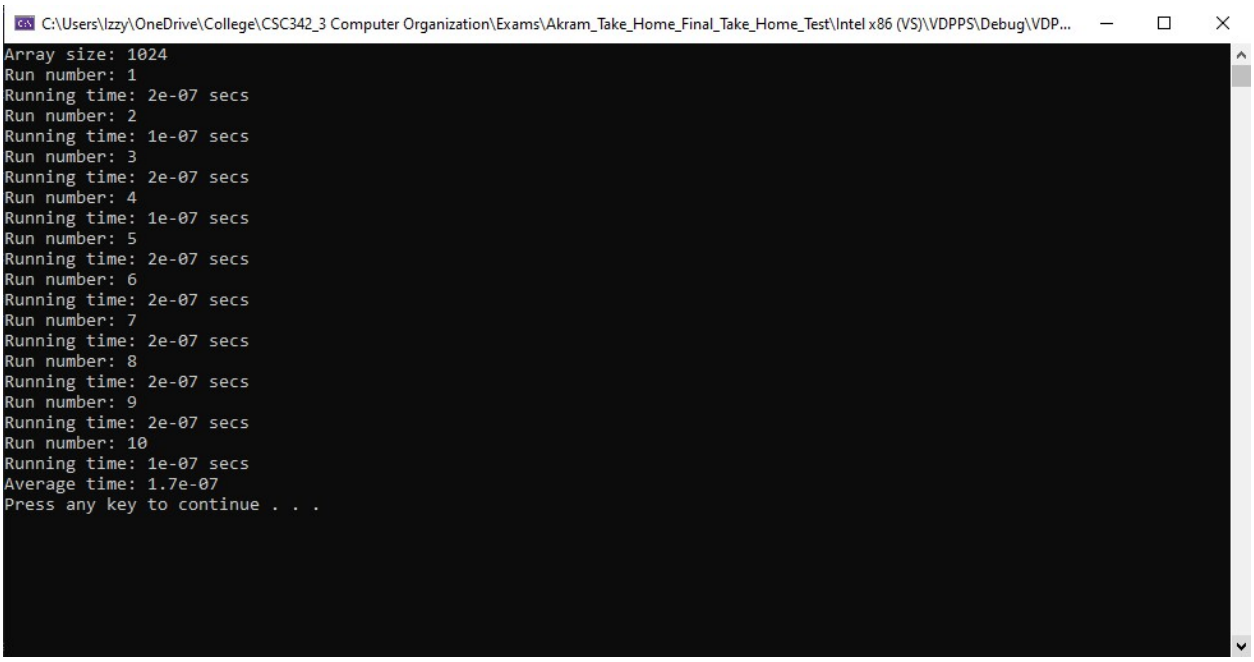


Fig 14f: VDPPS_Dot_Prod_Pntr (1024) w/ Qpar and arch

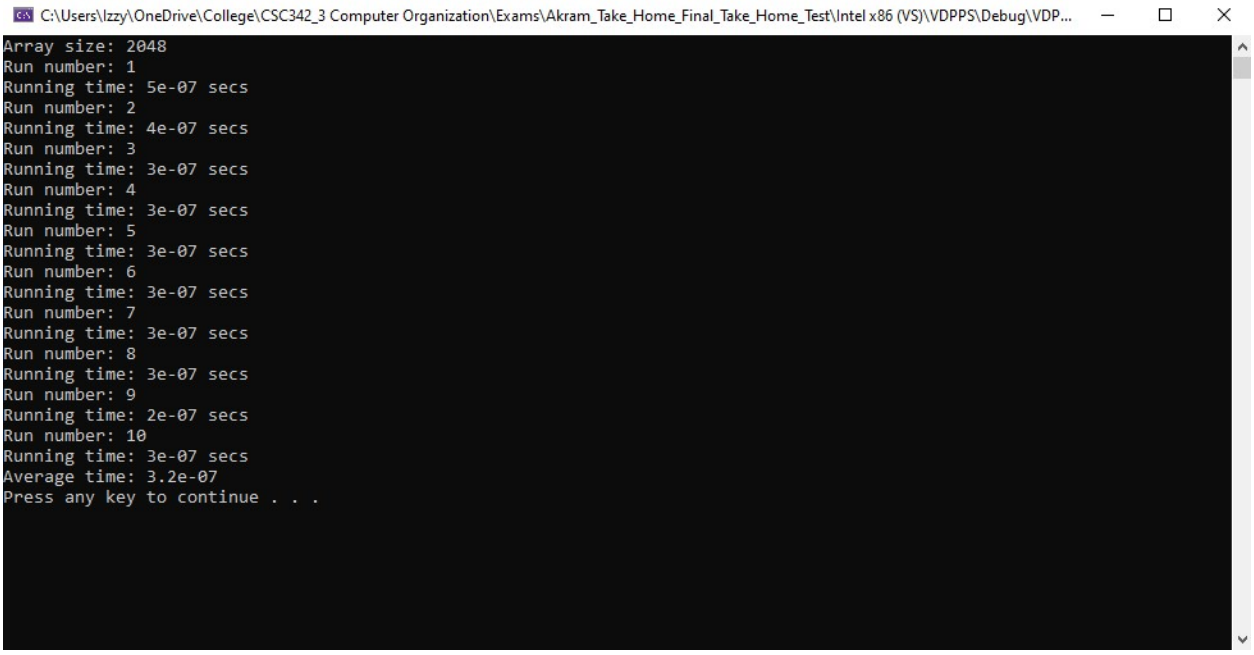


Fig 14g: VDPPS_Dot_Prod_Pntr (2^11) w/ Qpar and arch

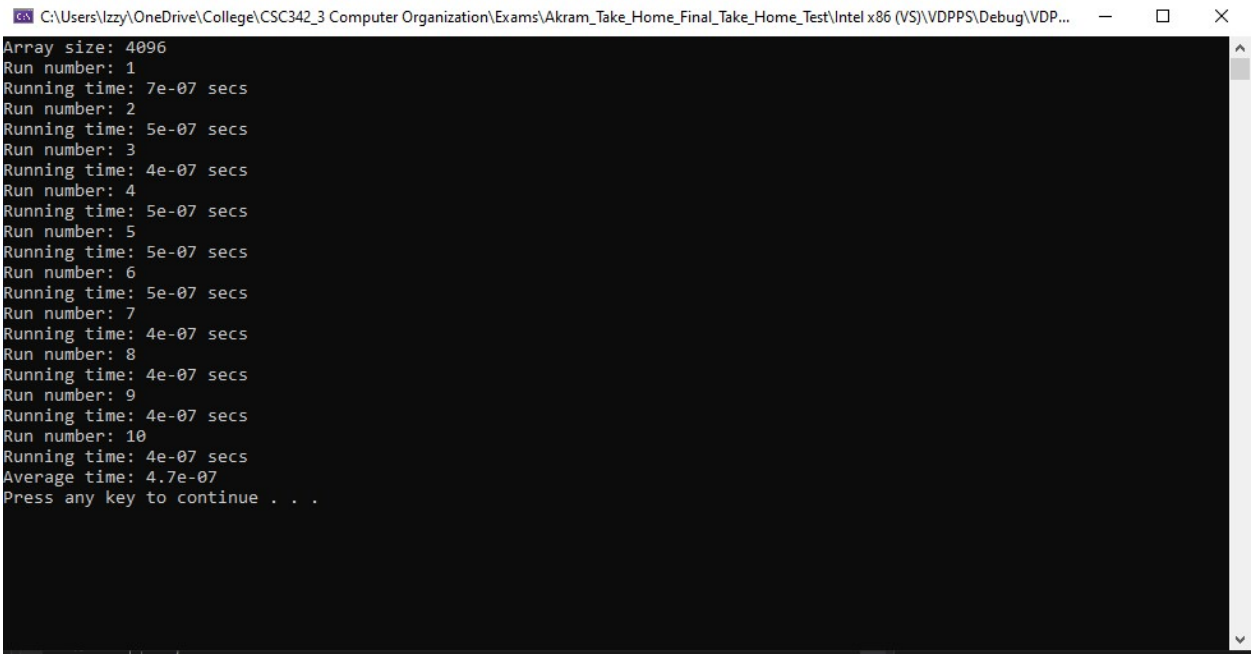


Fig 14h: VDPPS_Dot_Prod_Pntr (2^12) w/ Qpar and arch

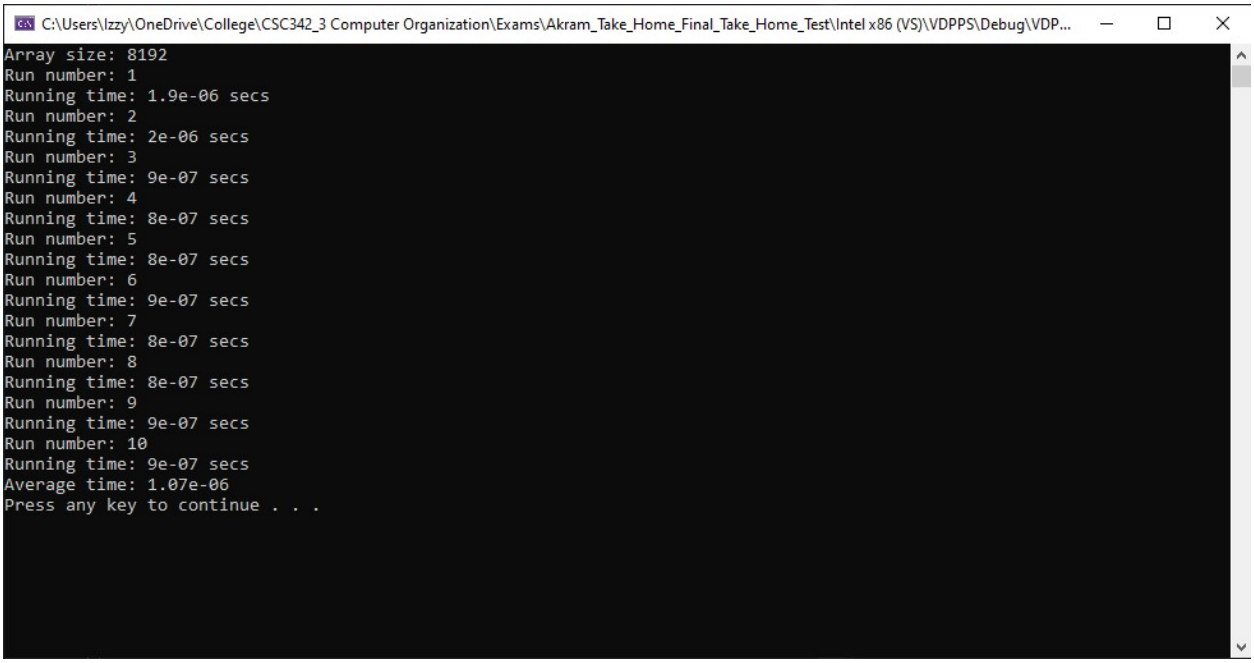


Fig 14i: VDPPS_Dot_Prod_Pntr (2^13) w/ Qpar and arch

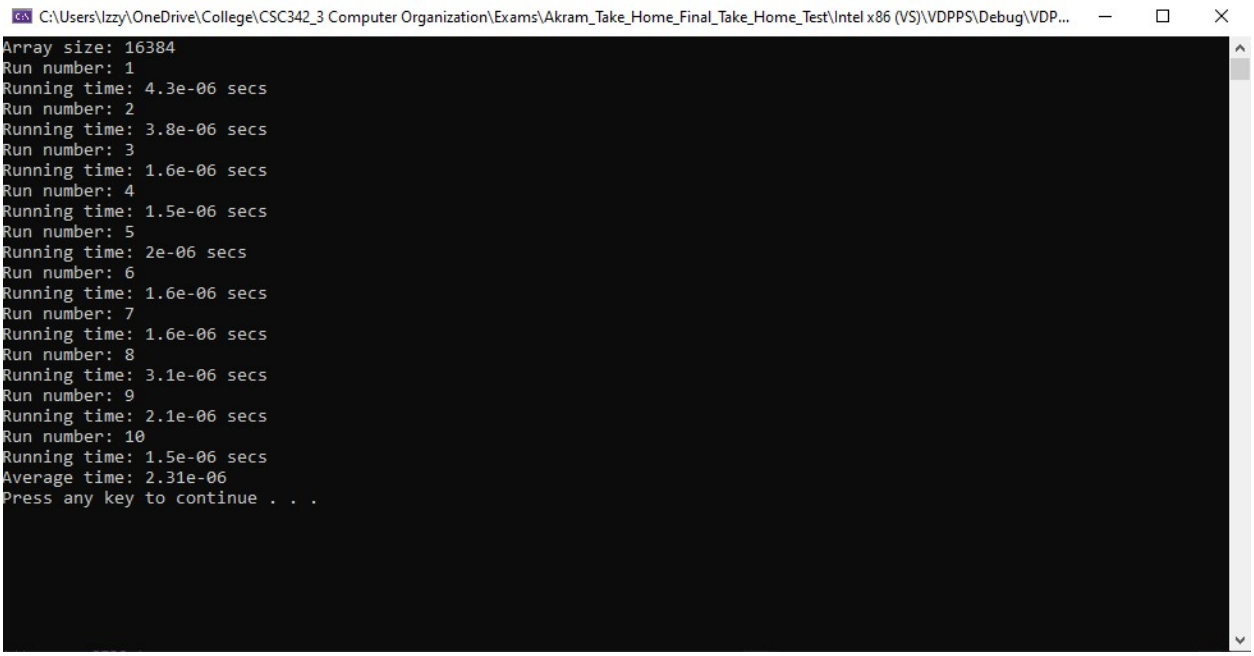


Fig 14j: VDPPS_Dot_Prod_Pntr (2^14) w/ Qpar and arch

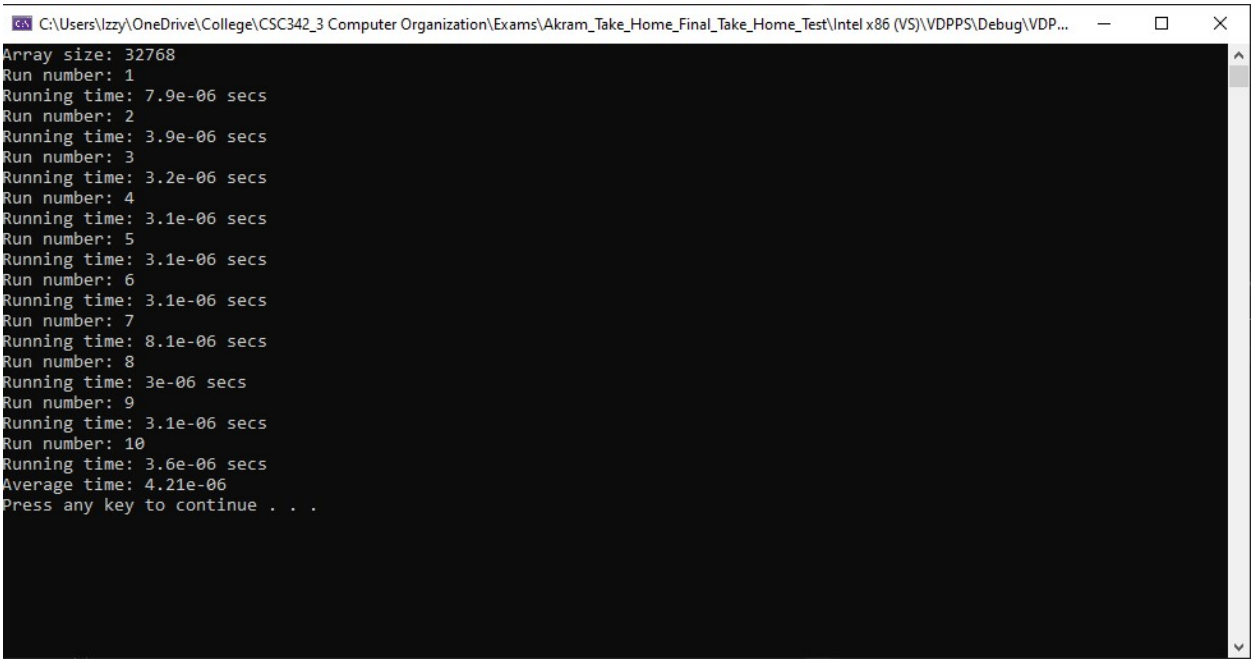


Fig 14k: VDPPS_Dot_Prod_Pntr (2^15) w/ Qpar and arch

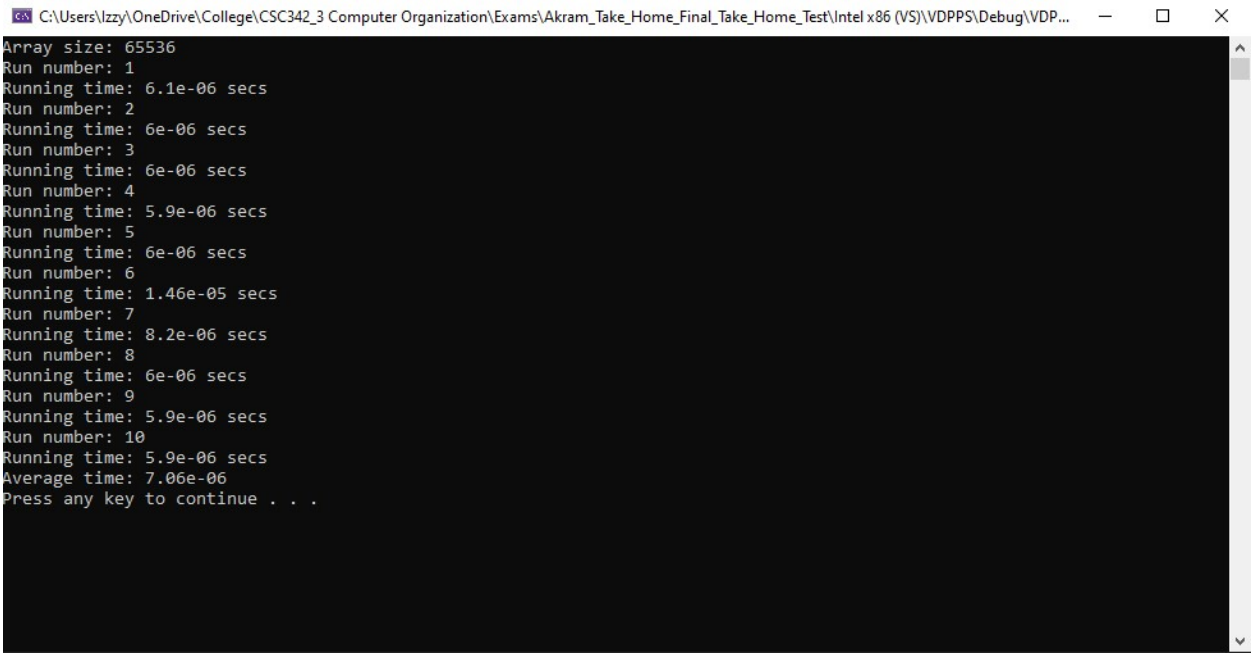
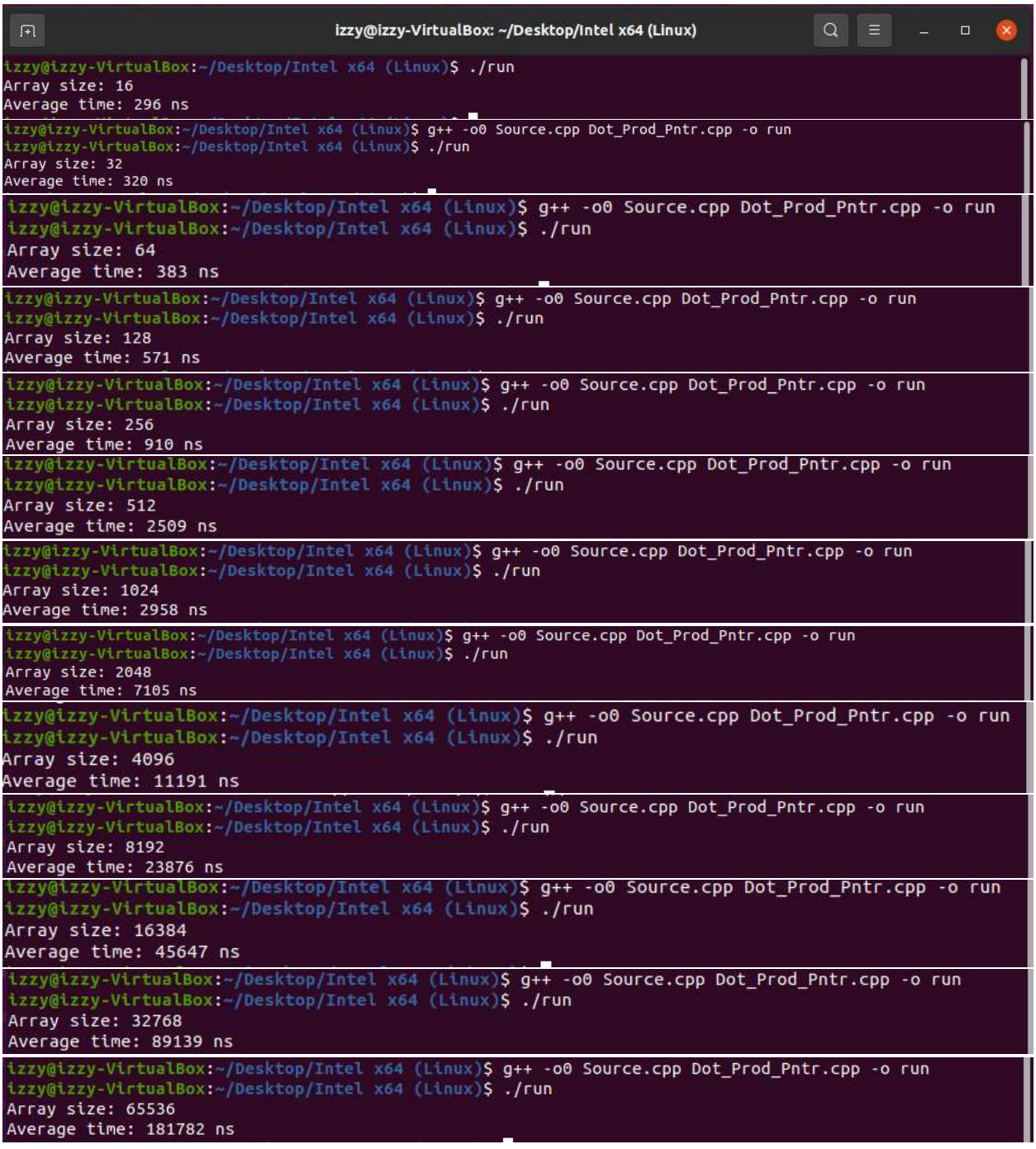


Fig 14l: VDPPS_Dot_Prod_Pntr (2^16) w/ Qpar and arch

Intel x64 (Linux GCC)

Dot Product (Pointer) [Non-Optimized]



Figs 15-15l: Dot_Prod_Pntr (-o0 optimization)

Dot Product (Pointer) [Compiler Optimization flags]

```
izzy@izzy-VirtualBox:~/Desktop/Intel x64 (Linux)$ g++ -o1 Source.cpp Dot_Prod_Pntr.cpp -o run
izzy@izzy-VirtualBox:~/Desktop/Intel x64 (Linux)$ ./run
Array size: 16
Average time: 277 ns
izzy@izzy-VirtualBox:~/Desktop/Intel x64 (Linux)$ g++ -o1 Source.cpp Dot_Prod_Pntr.cpp -o run
izzy@izzy-VirtualBox:~/Desktop/Intel x64 (Linux)$ ./run
Array size: 32
Average time: 317 ns
izzy@izzy-VirtualBox:~/Desktop/Intel x64 (Linux)$ g++ -o1 Source.cpp Dot_Prod_Pntr.cpp -o run
izzy@izzy-VirtualBox:~/Desktop/Intel x64 (Linux)$ ./run
Array size: 64
Average time: 378 ns
izzy@izzy-VirtualBox:~/Desktop/Intel x64 (Linux)$ g++ -o1 Source.cpp Dot_Prod_Pntr.cpp -o run
izzy@izzy-VirtualBox:~/Desktop/Intel x64 (Linux)$ ./run
Array size: 128
Average time: 569 ns
izzy@izzy-VirtualBox:~/Desktop/Intel x64 (Linux)$ g++ -o1 Source.cpp Dot_Prod_Pntr.cpp -o run
izzy@izzy-VirtualBox:~/Desktop/Intel x64 (Linux)$ ./run
Array size: 256
Average time: 923 ns
izzy@izzy-VirtualBox:~/Desktop/Intel x64 (Linux)$ g++ -o1 Source.cpp Dot_Prod_Pntr.cpp -o run
izzy@izzy-VirtualBox:~/Desktop/Intel x64 (Linux)$ ./run
Array size: 512
Average time: 1589 ns
izzy@izzy-VirtualBox:~/Desktop/Intel x64 (Linux)$ g++ -o1 Source.cpp Dot_Prod_Pntr.cpp -o run
izzy@izzy-VirtualBox:~/Desktop/Intel x64 (Linux)$ ./run
Array size: 1024
Average time: 2984 ns
izzy@izzy-VirtualBox:~/Desktop/Intel x64 (Linux)$ g++ -o1 Source.cpp Dot_Prod_Pntr.cpp -o run
izzy@izzy-VirtualBox:~/Desktop/Intel x64 (Linux)$ ./run
Array size: 2048
Average time: 7525 ns
izzy@izzy-VirtualBox:~/Desktop/Intel x64 (Linux)$ g++ -o1 Source.cpp Dot_Prod_Pntr.cpp -o run
izzy@izzy-VirtualBox:~/Desktop/Intel x64 (Linux)$ ./run
Array size: 4096
Average time: 11232 ns
izzy@izzy-VirtualBox:~/Desktop/Intel x64 (Linux)$ g++ -o1 Source.cpp Dot_Prod_Pntr.cpp -o run
izzy@izzy-VirtualBox:~/Desktop/Intel x64 (Linux)$ ./run
Array size: 8192
Average time: 23162 ns
izzy@izzy-VirtualBox:~/Desktop/Intel x64 (Linux)$ g++ -o1 Source.cpp Dot_Prod_Pntr.cpp -o run
izzy@izzy-VirtualBox:~/Desktop/Intel x64 (Linux)$ ./run
Array size: 16384
Average time: 46657 ns
izzy@izzy-VirtualBox:~/Desktop/Intel x64 (Linux)$ g++ -o1 Source.cpp Dot_Prod_Pntr.cpp -o run
izzy@izzy-VirtualBox:~/Desktop/Intel x64 (Linux)$ ./run
Array size: 32768
Average time: 89234 ns
izzy@izzy-VirtualBox:~/Desktop/Intel x64 (Linux)$ g++ -o1 Source.cpp Dot_Prod_Pntr.cpp -o run
izzy@izzy-VirtualBox:~/Desktop/Intel x64 (Linux)$ ./run
Array size: 65536
Average time: 181430 ns
```

Figs 16-16l: Dot_Prod_Pntr (-o1 optimization)

```
izzy@izzy-VirtualBox:~/Desktop/Intel x64 (Linux)$ g++ -o2 Source.cpp Dot_Prod_Pntr.cpp -o run
izzy@izzy-VirtualBox:~/Desktop/Intel x64 (Linux)$ ./run
Array size: 16
Average time: 247 ns
```

```
izzy@izzy-VirtualBox:~/Desktop/Intel x64 (Linux)$ g++ -o2 Source.cpp Dot_Prod_Pntr.cpp -o run
izzy@izzy-VirtualBox:~/Desktop/Intel x64 (Linux)$ ./run
Array size: 32
Average time: 292 ns
```

```
izzy@izzy-VirtualBox:~/Desktop/Intel x64 (Linux)$ g++ -o2 Source.cpp Dot_Prod_Pntr.cpp -o run
izzy@izzy-VirtualBox:~/Desktop/Intel x64 (Linux)$ ./run
Array size: 64
Average time: 399 ns
```

```
izzy@izzy-VirtualBox:~/Desktop/Intel x64 (Linux)$ g++ -o2 Source.cpp Dot_Prod_Pntr.cpp -o run
izzy@izzy-VirtualBox:~/Desktop/Intel x64 (Linux)$ ./run
Array size: 128
Average time: 574 ns
```

```
izzy@izzy-VirtualBox:~/Desktop/Intel x64 (Linux)$ g++ -o2 Source.cpp Dot_Prod_Pntr.cpp -o run
izzy@izzy-VirtualBox:~/Desktop/Intel x64 (Linux)$ ./run
Array size: 256
Average time: 915 ns
```

```
izzy@izzy-VirtualBox:~/Desktop/Intel x64 (Linux)$ g++ -o2 Source.cpp Dot_Prod_Pntr.cpp -o run
izzy@izzy-VirtualBox:~/Desktop/Intel x64 (Linux)$ ./run
Array size: 512
Average time: 1580 ns
```

```
izzy@izzy-VirtualBox:~/Desktop/Intel x64 (Linux)$ g++ -o2 Source.cpp Dot_Prod_Pntr.cpp -o run
izzy@izzy-VirtualBox:~/Desktop/Intel x64 (Linux)$ ./run
Array size: 1024
Average time: 2968 ns
```

```
izzy@izzy-VirtualBox:~/Desktop/Intel x64 (Linux)$ g++ -o2 Source.cpp Dot_Prod_Pntr.cpp -o run
izzy@izzy-VirtualBox:~/Desktop/Intel x64 (Linux)$ ./run
Array size: 2048
Average time: 8987 ns
```

```
izzy@izzy-VirtualBox:~/Desktop/Intel x64 (Linux)$ g++ -o2 Source.cpp Dot_Prod_Pntr.cpp -o run
izzy@izzy-VirtualBox:~/Desktop/Intel x64 (Linux)$ ./run
Array size: 4096
Average time: 11243 ns
```

```
izzy@izzy-VirtualBox:~/Desktop/Intel x64 (Linux)$ g++ -o2 Source.cpp Dot_Prod_Pntr.cpp -o run
izzy@izzy-VirtualBox:~/Desktop/Intel x64 (Linux)$ ./run
Array size: 8192
Average time: 24490 ns
```

```
izzy@izzy-VirtualBox:~/Desktop/Intel x64 (Linux)$ g++ -o2 Source.cpp Dot_Prod_Pntr.cpp -o run
izzy@izzy-VirtualBox:~/Desktop/Intel x64 (Linux)$ ./run
Array size: 16384
Average time: 45626 ns
```

```
izzy@izzy-VirtualBox:~/Desktop/Intel x64 (Linux)$ g++ -o2 Source.cpp Dot_Prod_Pntr.cpp -o run
izzy@izzy-VirtualBox:~/Desktop/Intel x64 (Linux)$ ./run
Array size: 32768
Average time: 90930 ns
```

```
izzy@izzy-VirtualBox:~/Desktop/Intel x64 (Linux)$ g++ -o2 Source.cpp Dot_Prod_Pntr.cpp -o run
izzy@izzy-VirtualBox:~/Desktop/Intel x64 (Linux)$ ./run
Array size: 65536
Average time: 181395 ns
```

Figs 17-17l: Dot_Prod_Pntr (-o2 optimization)

```
izzy@izzy-VirtualBox:~/Desktop/Intel x64 (Linux)$ g++ -o3 Source.cpp Dot_Prod_Pntr.cpp -o run
izzy@izzy-VirtualBox:~/Desktop/Intel x64 (Linux)$ ./run
Array size: 16
Average time: 248 ns
izzy@izzy-VirtualBox:~/Desktop/Intel x64 (Linux)$ g++ -o3 Source.cpp Dot_Prod_Pntr.cpp -o run
izzy@izzy-VirtualBox:~/Desktop/Intel x64 (Linux)$ ./run
Array size: 32
Average time: 292 ns
izzy@izzy-VirtualBox:~/Desktop/Intel x64 (Linux)$ g++ -o3 Source.cpp Dot_Prod_Pntr.cpp -o run
izzy@izzy-VirtualBox:~/Desktop/Intel x64 (Linux)$ ./run
Array size: 64
Average time: 403 ns
izzy@izzy-VirtualBox:~/Desktop/Intel x64 (Linux)$ g++ -o3 Source.cpp Dot_Prod_Pntr.cpp -o run
izzy@izzy-VirtualBox:~/Desktop/Intel x64 (Linux)$ ./run
Array size: 128
Average time: 584 ns
izzy@izzy-VirtualBox:~/Desktop/Intel x64 (Linux)$ g++ -o3 Source.cpp Dot_Prod_Pntr.cpp -o run
izzy@izzy-VirtualBox:~/Desktop/Intel x64 (Linux)$ ./run
Array size: 256
Average time: 930 ns
izzy@izzy-VirtualBox:~/Desktop/Intel x64 (Linux)$ g++ -o3 Source.cpp Dot_Prod_Pntr.cpp -o run
izzy@izzy-VirtualBox:~/Desktop/Intel x64 (Linux)$ ./run
Array size: 512
Average time: 1591 ns
izzy@izzy-VirtualBox:~/Desktop/Intel x64 (Linux)$ g++ -o3 Source.cpp Dot_Prod_Pntr.cpp -o run
izzy@izzy-VirtualBox:~/Desktop/Intel x64 (Linux)$ ./run
Array size: 1024
Average time: 2997 ns
izzy@izzy-VirtualBox:~/Desktop/Intel x64 (Linux)$ g++ -o3 Source.cpp Dot_Prod_Pntr.cpp -o run
izzy@izzy-VirtualBox:~/Desktop/Intel x64 (Linux)$ ./run
Array size: 2048
Average time: 5715 ns
izzy@izzy-VirtualBox:~/Desktop/Intel x64 (Linux)$ g++ -o3 Source.cpp Dot_Prod_Pntr.cpp -o run
izzy@izzy-VirtualBox:~/Desktop/Intel x64 (Linux)$ ./run
Array size: 4096
Average time: 11315 ns
izzy@izzy-VirtualBox:~/Desktop/Intel x64 (Linux)$ g++ -o3 Source.cpp Dot_Prod_Pntr.cpp -o run
izzy@izzy-VirtualBox:~/Desktop/Intel x64 (Linux)$ ./run
Array size: 8192
Average time: 22272 ns
izzy@izzy-VirtualBox:~/Desktop/Intel x64 (Linux)$ g++ -o3 Source.cpp Dot_Prod_Pntr.cpp -o run
izzy@izzy-VirtualBox:~/Desktop/Intel x64 (Linux)$ ./run
Array size: 16384
Average time: 44288 ns
izzy@izzy-VirtualBox:~/Desktop/Intel x64 (Linux)$ g++ -o3 Source.cpp Dot_Prod_Pntr.cpp -o run
izzy@izzy-VirtualBox:~/Desktop/Intel x64 (Linux)$ ./run
Array size: 32768
Average time: 88366 ns
izzy@izzy-VirtualBox:~/Desktop/Intel x64 (Linux)$ g++ -o3 Source.cpp Dot_Prod_Pntr.cpp -o run
izzy@izzy-VirtualBox:~/Desktop/Intel x64 (Linux)$ ./run
Array size: 65536
Average time: 177113 ns
```

Fig 18-18l: Dot_Prod_Pntr (-o3 optimization)

Results

Array Size	Dot_Prod_Pntr (Non-Opt) time (secs)	Dot_Product_Pntr (Opt) time (secs)	Dot_Prod_Pntr (Man Opt) time (secs)	VDPPS_Dot_Prod_Pntr time (secs)
16	1.50E-07	1.40E-07	1.20E-07	1.00E-07
32	1.80E-07	1.10E-07	1.10E-07	1.20E-07
64	2.50E-07	1.50E-07	1.40E-07	1.00E-07
128	4.40E-07	1.80E-07	1.50E-07	1.20E-07
256	8.10E-07	2.70E-07	2.30E-07	1.20E-07
512	1.50E-06	4.50E-07	4.60E-07	1.50E-07
1024	2.88E-06	8.50E-07	5.10E-07	1.70E-07
2048	5.78E-06	1.61E-06	9.65E-07	3.20E-07
4096	1.15E-05	3.07E-06	1.77E-06	4.70E-07
8192	2.26E-05	5.94E-06	3.51E-06	1.07E-06
16384	4.46E-05	1.19E-05	7.11E-06	2.31E-06
32768	9.03E-05	2.42E-05	1.42E-05	4.21E-06
65536	1.80E-04	4.84E-05	2.77E-05	7.06E-06

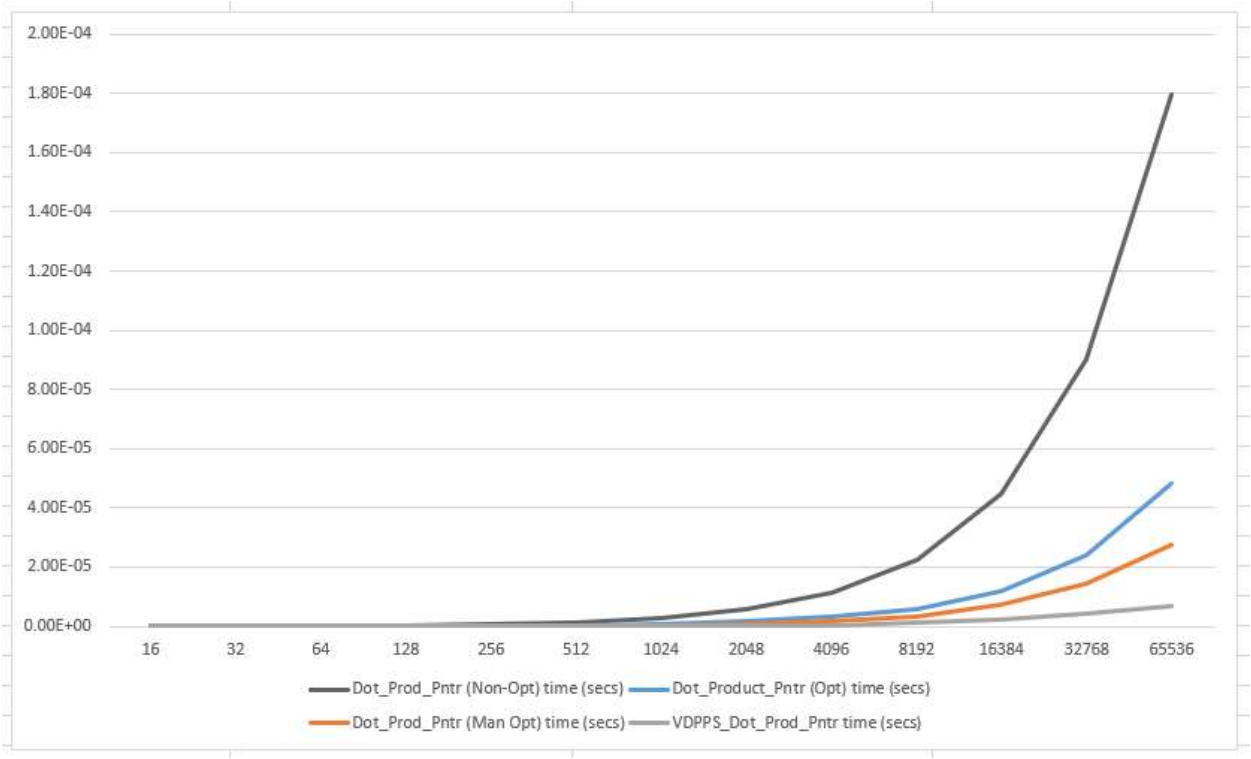


Fig 19: Extraction times comparison of Dot_Prod_Pntr in VS

Array Size	Dot_Prod_Pntr (-o0) time (ns)	Dot_Prod_Pntr (-o1) time (ns)	Dot_Prod_Pntr (-o2) time (ns)	Dot_Prod_Pntr (-o3) time (ns)
16	6.55E-07	5.43E-07	5.14E-07	5.12E-07
32	7.55E-07	6.07E-07	5.63E-07	5.34E-07
64	8.22E-07	8.17E-07	6.40E-07	7.31E-07
128	1.10E-06	1.01E-06	7.18E-07	5.69E-07
256	1.82E-06	1.65E-06	1.42E-06	1.01E-06
512	2.56E-06	2.27E-06	1.88E-06	1.91E-06
1024	4.20E-06	3.58E-06	2.21E-06	2.23E-07
2048	2.55E-05	4.21E-06	3.99E-06	3.23E-06
4096	3.46E-05	2.59E-05	1.65E-05	1.52E-05
8192	6.48E-05	3.28E-05	2.57E-05	3.24E-05
16384	1.22E-04	6.35E-05	4.91E-05	5.03E-05
32768	2.33E-04	1.63E-04	1.26E-04	1.32E-04
65536	4.82E-04	2.83E-04	2.25E-04	2.26E-04

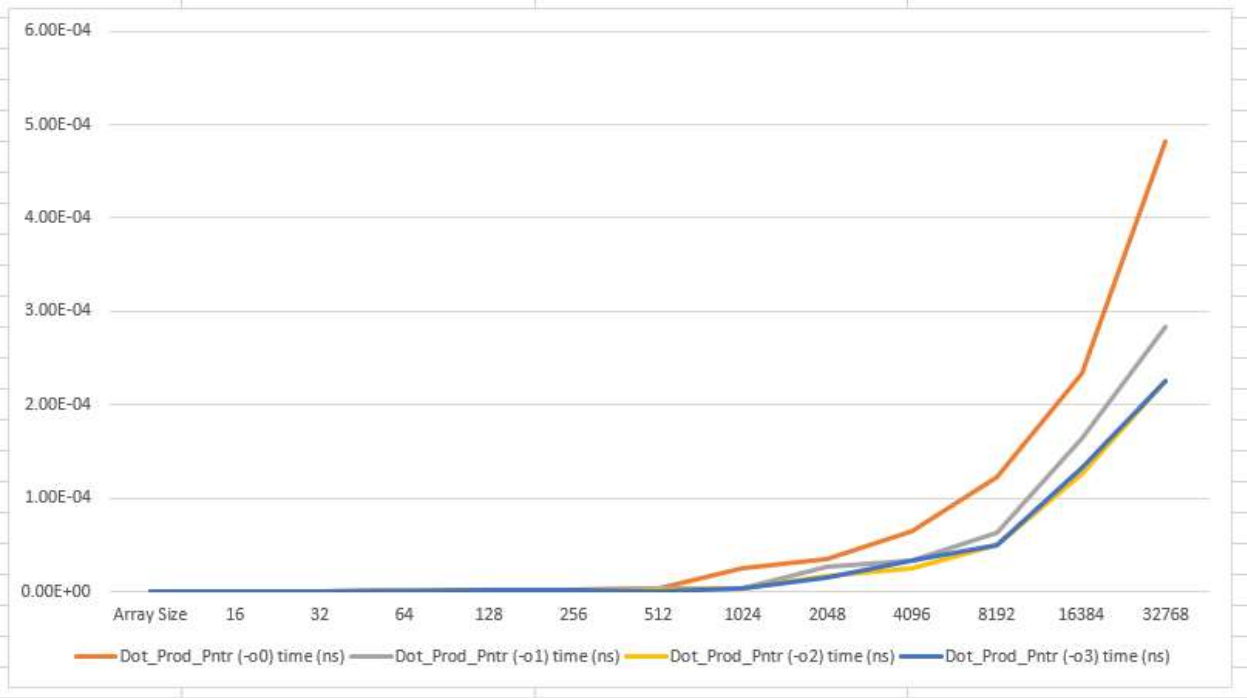


Fig 19: Extraction times comparison of Dot_Prod_Pntr optimization level in Linux

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.