Christopher Lall

CSc 342/343 – Professor Gertner

Take Home Test 3

Due 5/15/222 – I was given an extension by Professor Gertner!

I Will neither give nor recieve unauthorized assistance on this TEST. I will only use one computing device to perform this TEST, I will not use cell while performing this TEST.

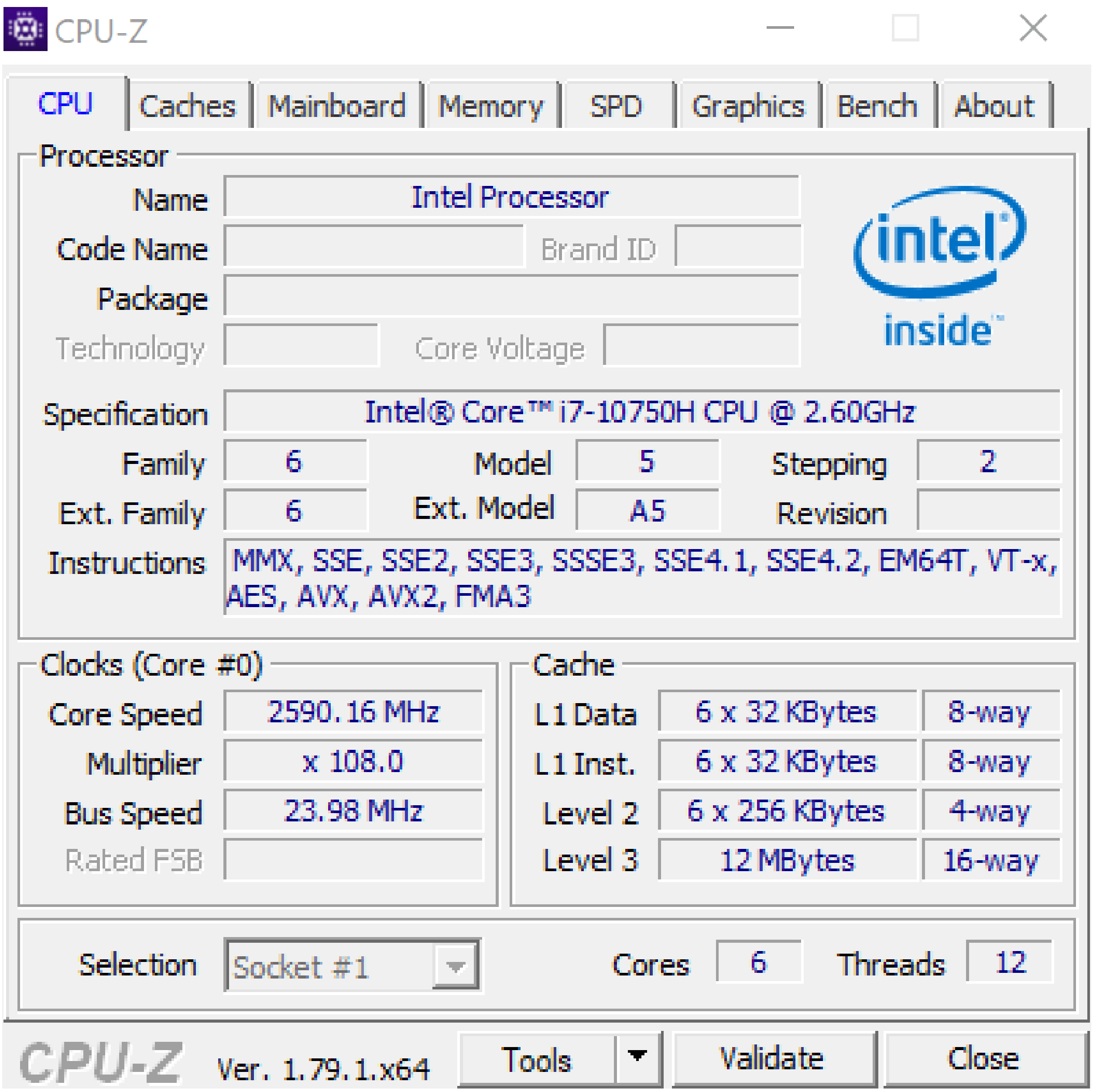
Chris

Table of Contents

Objective:

The goal of this lab is to optimize compiler-generated dot product code using vector instructions. Three distinct levels of optimization can be compared. We'll start with the unoptimized code's assembly version. A compiler is then used to optimize the code, producing assembly code with automatic parallelization and vectorization. Finally, in our compiler-optimized assembly, we manually replace the instructions with new ones. The Windows Visual Studio 32-bit compiler and the Linux GCC 64-bit compiler will be used to perform these checks. While investigating how time changes for optimizations, we will define vectorization and parallelization and how each architecture employs various ways to create outcomes. The activity allows you to compare results in a plot utilizing power 2 vector sizes.

CPU-Z



This picture shows my processor vector processing power. As shown in the instructions, my CPU have "SSE" up to "SSE4". SSE stands for Streaming SIMD (Single Instruction Multiple Data) Extensions. SSE4 is a major upgrade to SSE as it has wider application.

Unoptimized on Visual Studio:

```
#include <iostream>
#include <ctime>
#include <Windows.h>
#include <cmath>
#include <chrono>
#include <fstream>
#include "Lall_Dot_Product.h"

int _DotProduct(int*, int*, size_t);
using namespace std;
const string getName()
{
    chrono::time_point<chrono::system_clock> now = chrono::system_clock::now();
    time_t start = std::chrono::system_clock::to_time_t(now);
    char timedisplay[100];
    struct tm buf;
    errno_t err = localtime_s(&buf, &start);
    strftime(timedisplay, sizeof(timedisplay), "dp%Y%m%d%H%M%S.csv", &buf);
    return timedisplay;
}

int main()
{
    std::ofstream file;
    string fname = getName();
    file.open(fname, ios::out);
    file << "Array Size, Start, End, Difference, f(Hz), time(s)";
    printf("n\tTime Taken (us)\n");
    for (int p = 4; p < 17; ++p)
    {
        int n = pow(2, p);
        _int64 start = 0;
        _int64 end = 0;
        _int64 f = 0;

        srand(time(nullptr));
        int* x = new int[n];
        int* y = new int[n];
```

```cpp
    for (int i = 0; i < n; ++i)
    {
        *(x + i) = rand();
        *(y + i) = rand();
    }

    if (QueryPerformanceCounter((LARGE_INTEGER*)&start) != 0)
    {
        _DotProduct(x, y, n);
        QueryPerformanceCounter((LARGE_INTEGER*)&end);
        QueryPerformanceFrequency((LARGE_INTEGER*)&f);

        printf("%d\t%.2f us \n", n, (end - start) * 1.0 / f * pow(10, 6));
        file << n << ", " << start << ", " << end << ", " << end - start << ", " << f << ", " << (end - start) * 1.0 / f << endl;
    }
    else
    {
        DWORD errorMessageID = GetLastError();
        printf("Error Value: %lu\n", errorMessageID);
    }
    delete x;
    delete y;
  }
  file.close();
  return 0;
}
```

Above is main.cpp file.

Below is Lall_Dot_Products.h

```cpp
#pragma once
int _DotProduct(int* a, int* b, size_t n)
{
    int sum = 0;
#pragma loop(hint_parallel(0))
    for (size_t i = 0; i < n; ++i)
    {
        sum += *(a + i) * (*b + i);
    }
    return sum;
}
#pragma once
```

It's worth noting that we have to turn off Automatic Parallelization and Automatic Vectorization because this code isn't optimized. This implies that the execution time will be longer than the optimized version.

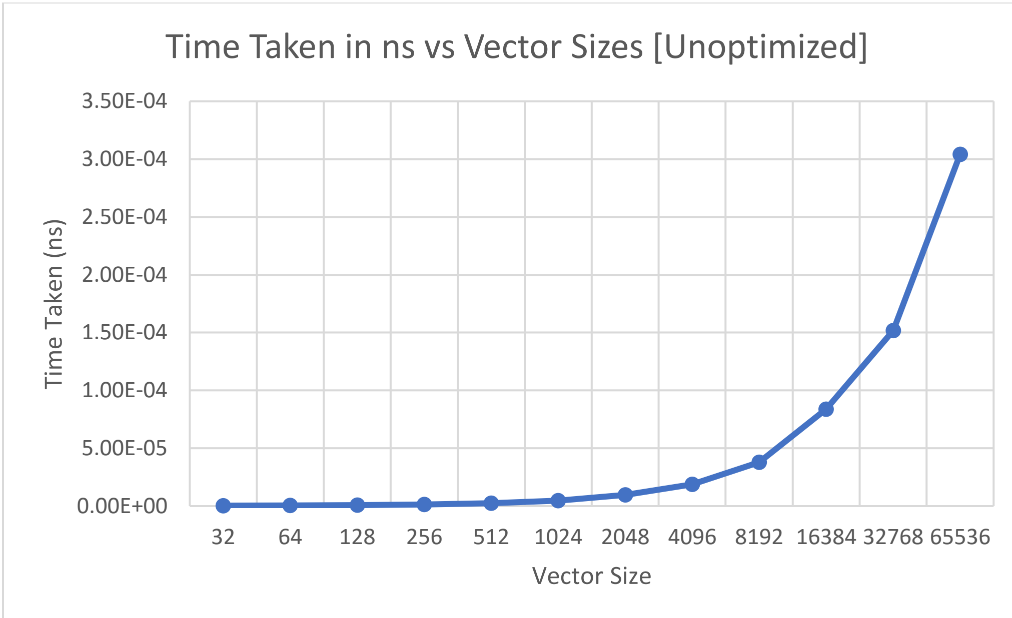
Explanation:

The feature from the DotProduct.cpp file that we will look at next is declared on line 7. We're mentioning the functionality here rather than utilizing a header file because the project is so simple. We'll then look at the QueryPerformanceCounter and QueryPerformanceFrequency, which are both high-precision timers for exact measurements. QueryPerformanceCounter returns the current precision timer value, while QueryPerformanceFrequency returns the current performance counter frequency. The time and frequency of the timer are calculated using 64-bit integer types defined on line 17. Then, in lines 20 to 23, we fill the arrays with 0's because we only care about the dot product time and ignore the integers product.

In line 26, we start naming the QueryPerformanceCounter timer function. It takes the 64-bit ctrl1 integer and converts it to a LARGE INTEGER object pointer. The output counter's frequency in counts per second is received by LARGE INTEGER*, which sets ctrl1 to the number of clock ticks used to start the timer. From lines 26 to 30, we call the timer before execution, perform the dot product code, and then call the timer again after execution. Ctr1 has the given start time, and Ctr2 has the mentioned end time, which we may subtract to get the dot product computation time. We use QueryPerformanceFrequency to estimate the CPU frequency on line 34. This gives us the most recent output counter frequency in counts per second (which registers events in software). Lines 36–39 finally print out all of our findings. This will be displayed after the dot product feature has been discussed.

Output:

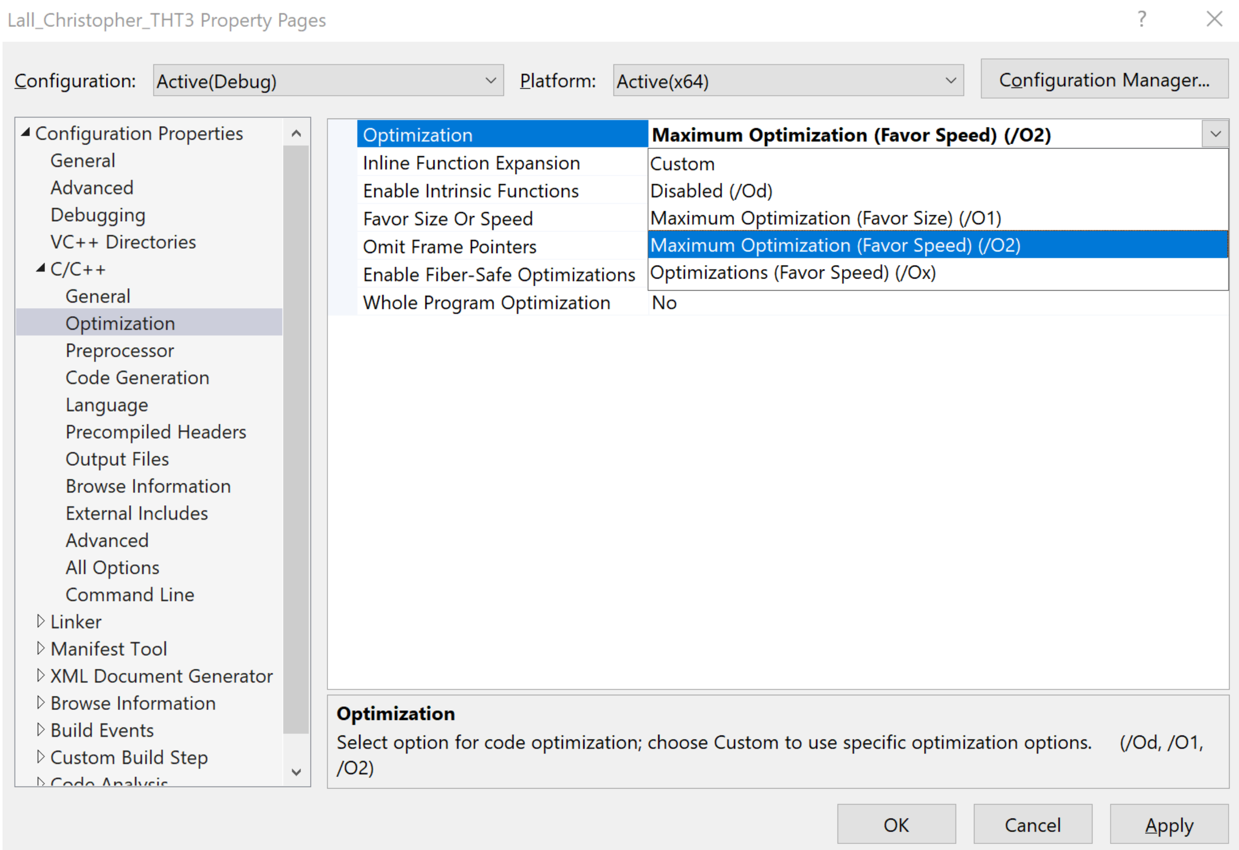
```
n       Time Taken (us)
16      0.20 us
32      0.40 us
64      0.60 us
128     0.70 us
256     1.30 us
512     2.50 us
1024    4.70 us
2048    9.60 us
4096    18.80 us
8192    37.80 us
16384   83.70 us
32768   151.80 us
65536   304.20 us

C:\Users\Chris\Desktop\CSC342\Tests\Take_Home_Test_3\VISUAL S\Lall_Christopher_THT3\x64\Debug\Lall_Christopher_THT3.exe
(process 20160) exited with code 0.
Press any key to close this window . . .
```



The graph above shows a visual graph of my output. The next task is making the code optimized by adjusting the setting on Visual Studio.

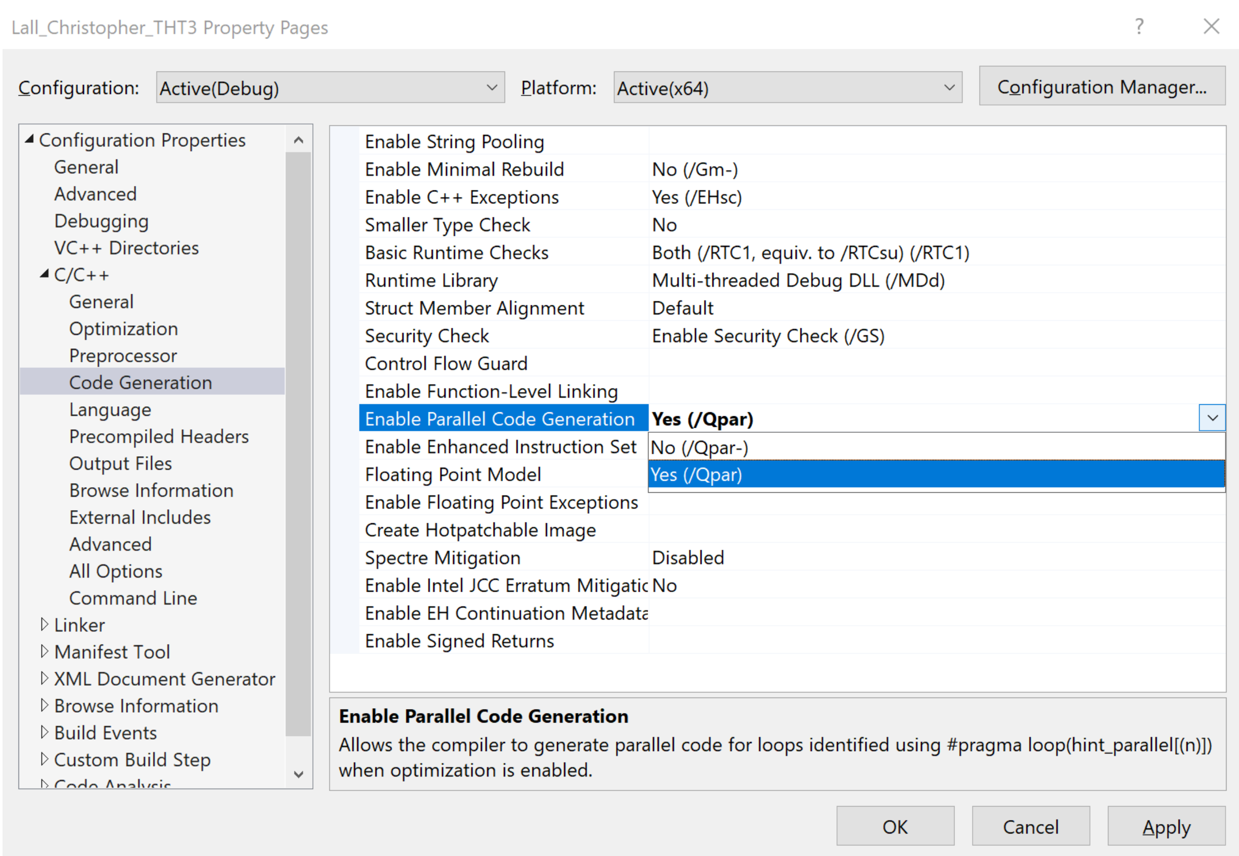
How to optimize Visual Studio:

To get to the windows shown below, click on Project > Properties.



Click on C/C++ > Optimization. In the optimization tab, click on Maximum Optimization for the fastest speed.

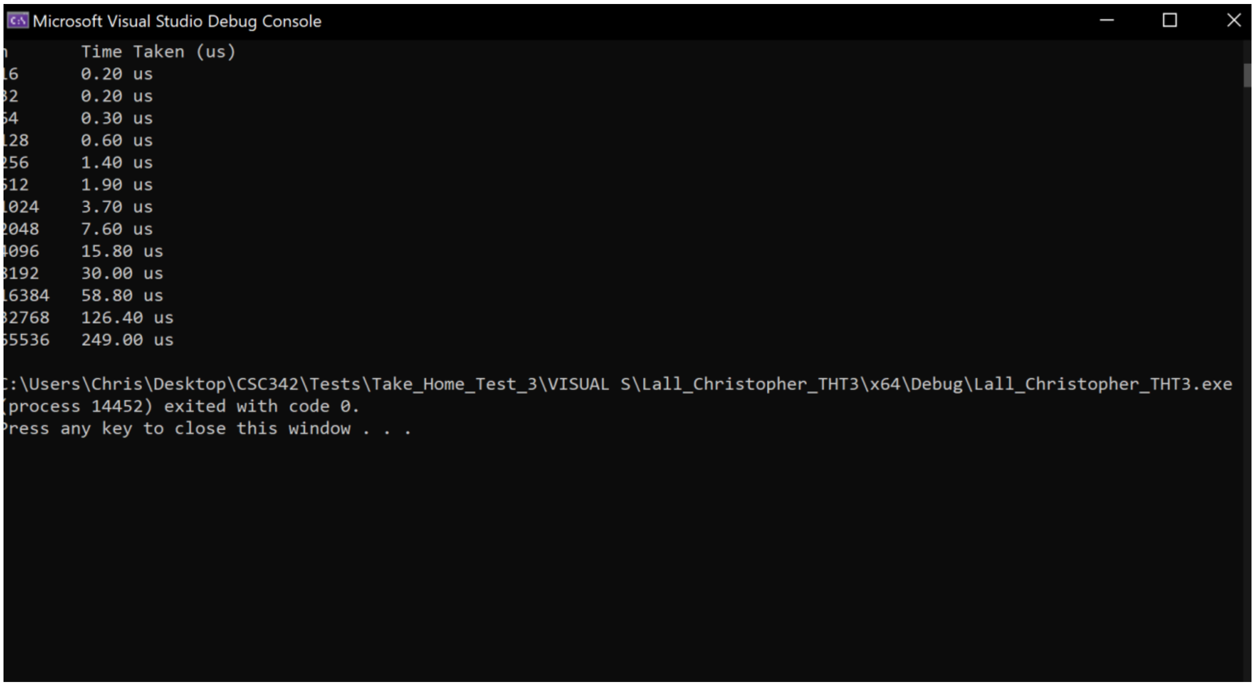
Then Click on C/C++ > Code Generation. In the ‘Enable Parallel Code Generation’, click Yes.


Lall_Christopher_THT3 Property Pages
Configuration: Active(Debug)
Platform: Active(x64)
Configuration Manager...
Configuration Properties
General
Advanced
Debugging
VC++ Directories
C/C++
General
Optimization
Preprocessor
Code Generation
Language
Precompiled Headers
Output Files
Browse Information
External Includes
Advanced
All Options
Command Line
Linker
Manifest Tool
XML Document Generator
Browse Information
Build Events
Custom Build Step
Enable String Pooling
Enable Minimal Rebuild No (/Gm-)
Enable C++ Exceptions Yes (/EHsc)
Smaller Type Check No
Basic Runtime Checks Both (/RTC1, equiv. to /RTCsu) (/RTC1)
Runtime Library Multi-threaded Debug DLL (/MDd)
Struct Member Alignment Default
Security Check Enable Security Check (/GS)
Control Flow Guard
Enable Function-Level Linking
Enable Parallel Code Generation Yes (/Qpar)
Enable Enhanced Instruction Set No (/Qpar-)
Floating Point Model Yes (/Qpar)
Enable Floating Point Exceptions
Create Hotpatchable Image
Spectre Mitigation Disabled
Enable Intel JCC Erratum Mitigatic No
Enable EH Continuation Metadata
Enable Signed Returns
Enable Parallel Code Generation
Allows the compiler to generate parallel code for loops identified using #pragma loop(hint_parallel[(n)]) when optimization is enabled.
OK
Cancel
Apply

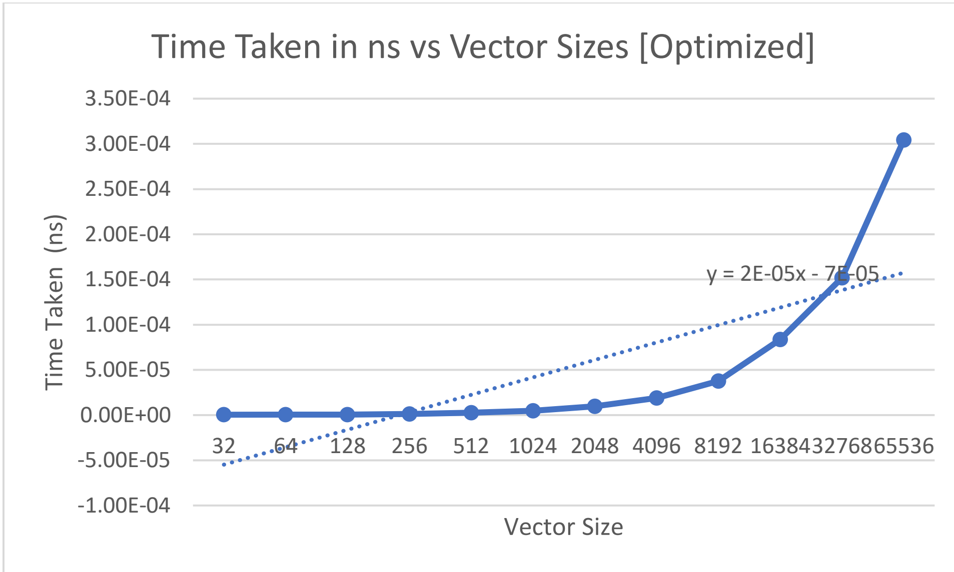
Optimized on Visual Studio:

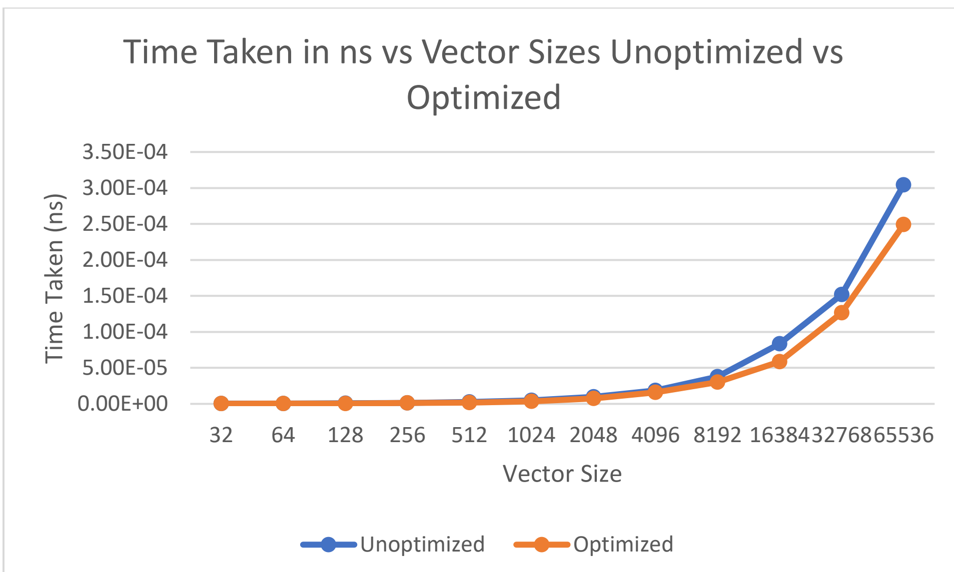
My output after using Visual Studio to optimize it.

The only difference between the optimized and unoptimized code is the Visual Studio option that I applied in the preceding portion of the report.



The optimized code has a faster execution time. It is not apparent in the graph because it is miniscule due to the graphs axis values.





This graph shows that using optimization resulted in a much faster execution time than without it.

Unoptimized on Linux gcc:

Lall_Linux_main.cpp > main(int, char **)

```
#include <stdio.h>
#include <stdint.h>
#include <stdlib.h>
#include <time.h>
using namespace std;

#define SIZE 16 // n - Vector Size
#pragma once // Assure file will not be included ultiple times

extern int _DotProduct(int* a, int *b, size_t n); // Extend the visiblity of this function

static int array[SIZE];

int main(int argc, char **argv)
{
    uint64_t timed;
    double avg = 0.0;
    struct timespec start, end;
    for(int i = 0; i < 100; ++i)
    {
        clock_gettime(CLOCK_MONOTONIC, &start);
        _DotProduct(array, array, SIZE);
        clock_gettime(CLOCK_MONOTONIC, &end);
        timed = (end.tv_sec - start.tv_sec) * 1000000000ULL + end.tv_nsec - start.tv_nsec;
        avg += timed;
    }
    avg /= 100;
    printf("size = %i, AVG time = %f ns \n", SIZE, avg);
    return 0;
}
```

Above is the code for the main.cpp file.

Below is the code for the dotproduct.h file.

```
#include <stdlib.h>

int _DotProduct(int* a, int *b, size_t n)
{
    int sum = 0;
    #pragma loop (hint_parallel(0)) // Enable auto-parallelization. 0 represents number of threads
    for (size_t i = 0; i < n; ++i)
    {
        sum += *(a + i) * (*b + i);
    }
    return sum;
}
```

How this code will work:

We utilize a header file to connect the Lall Linux DotProduct.h file in line 10 instead of specifying the functionality as we did for Windows. Line 14 should be skipped because it is the same as in Windows. We declare a struct and assign it to the timespec structure on line 18. Unless we specifically assign it a structure type, no storage or memory is assigned to this struct when we declare it. The timespec structure is responsible for preserving an interval separated into seconds and nanoseconds in this code. This will work in tandem with the timer feature. We'll disregard lines 17 to 20 because they're similar to Windows. Things start to alter on line 22. In the windows section, we used the QueryPerformanceCounter, which is a high accuracy timer. Unfortunately, it is only available on Windows, so we will have to utilize a Linux alternative. The best option is to use the time library's clock gettime(2) method. Clockid t and struct timespec appear to be the parameters. CLOCK REALTIME, a system-wide clock that tracks real time, and the previously defined timespec are used in this line. Clock gettime(), unlike QueryPerformanceCounter, does not employ ticks, hence the results were a little shaky. In addition, line 22 gets the start time, line 23 gets the feature, and line 24 gets the end time. To easily extract seconds and nanoseconds, lines 25 and 26 use the timespec structure members tv sec and tv nsec. Finally, on line 31, we use Scientific Notation and a percent E to print the total time in seconds.

```
christopherlall@LallChristopher-VirtualBox:~/Desktop/THT3$ g++ -S -O0 Lall_Linu
x_main.cpp unoptimized
Lall_Linux_main.cpp:8:9: warning: #pragma once in main file
    8 | #pragma once // Assure file will not be included ultiple times
      |         ^~~~
```

In the terminal, you will get the above screenshot.

Compile generated code in ASM [unoptimized]:

```
christopherlall@LallChristopher-VirtualBox:~/Desktop/THT3$ cat unoptimized
        .file   "Lall_Main.cpp"
        .text
        .local  _ZL5array
        .comm   _ZL5array,64,32
        .section        .rodata
.LC2:
        .string "size = %i, AVG time = %f ns \n"
        .text
        .globl  main
        .type   main, @function
main:
.LFB15:
        .cfi_startproc
        endbr64
        pushq   %rbp
        .cfi_def_cfa_offset 16
        .cfi_offset 6, -16
        movq    %rsp, %rbp
        .cfi_def_cfa_register 6
        subq    $96, %rsp
        movl    %edi, -84(%rbp)
        movq    %rsi, -96(%rbp)
        movq    %fs:40, %rax
        movq    %rax, -8(%rbp)
        xorl    %eax, %eax
        pxor    %xmm0, %xmm0
        movsd   %xmm0, -64(%rbp)
        movl    $0, -68(%rbp)
        jmp     .L2
.L5:
        leaq    -48(%rbp), %rax
        movq    %rax, %rsi
        movl    $1, %edi
        call    clock_gettime@PLT
        movl    $16, %edx
        leaq    _ZL5array(%rip), %rax
        movq    %rax, %rsi
        leaq    _ZL5array(%rip), %rax
        movq    %rax, %rdi
        call    _Z11_DotProductPiS_m@PLT
        leaq    -32(%rbp), %rax
```

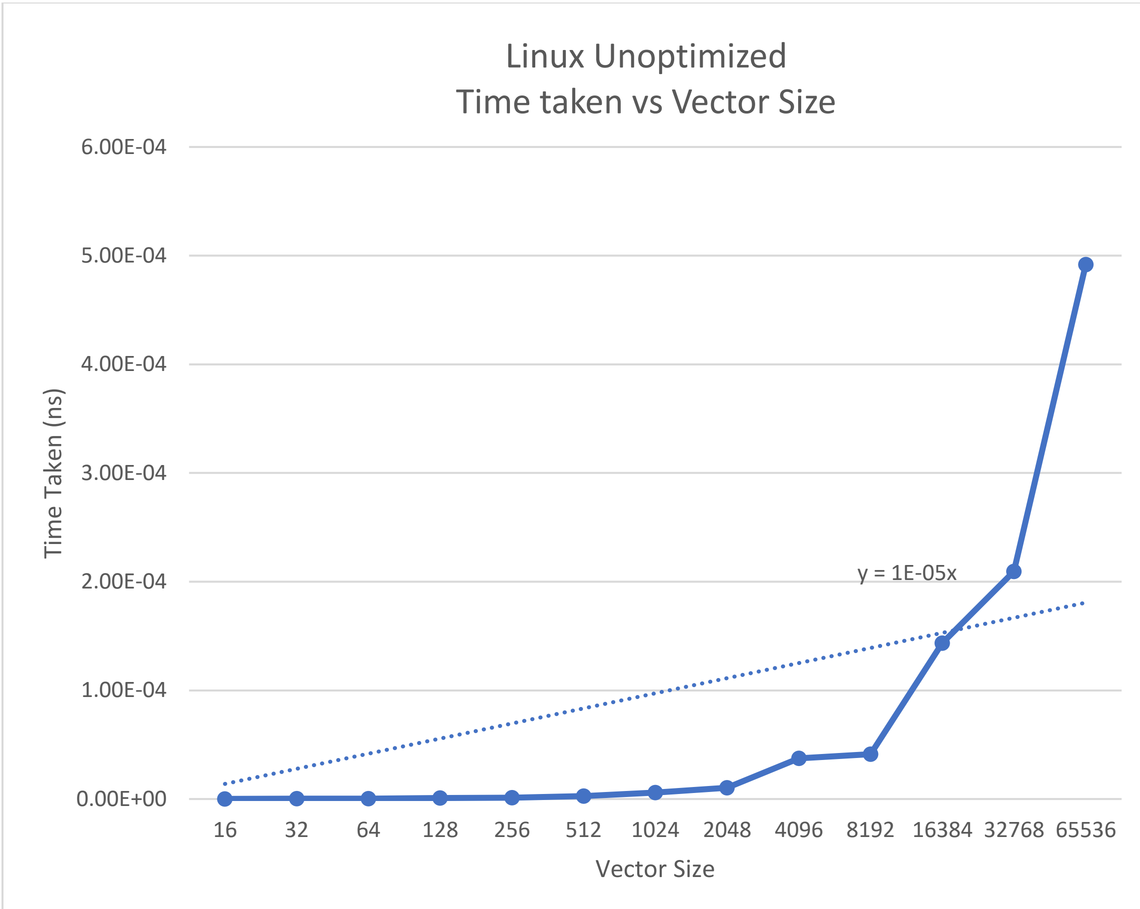
```
        leaq    -32(%rbp), %rax
        movq    %rax, %rsi
        movl    $1, %edi
        call    clock_gettime@PLT
        movq    -32(%rbp), %rax
        movq    -48(%rbp), %rdx
        subq    %rdx, %rax
        imulq   $1000000000, %rax, %rdx
        movq    -24(%rbp), %rax
        addq    %rax, %rdx
        movq    -40(%rbp), %rax
        movq    %rax, %rcx
        movq    %rdx, %rax
        subq    %rcx, %rax
        movq    %rax, -56(%rbp)
        movq    -56(%rbp), %rax
        testq   %rax, %rax
        js      .L3
        pxor    %xmm0, %xmm0
        cvtsi2sdq       %rax, %xmm0
        jmp     .L4
.L3:
        movq    %rax, %rdx
        shrq    %rdx
        andl    $1, %eax
        orq     %rax, %rdx
        pxor    %xmm0, %xmm0
        cvtsi2sdq       %rdx, %xmm0
        addsd   %xmm0, %xmm0
.L4:
        movsd   -64(%rbp), %xmm1
        addsd   %xmm1, %xmm0
        movsd   %xmm0, -64(%rbp)
        addl    $1, -68(%rbp)
.L2:
        cmpl    $99, -68(%rbp)
        jle     .L5
        movsd   -64(%rbp), %xmm0
        movsd   .LC1(%rip), %xmm1
        divsd   %xmm1, %xmm0
        movsd   %xmm0, -64(%rbp)
        movq    -64(%rbp), %rax
        movq    %rax, %xmm0
        movl    $16, %esi
```

```
        leaq    .LC2(%rip), %rax
        movq    %rax, %rdi
        movl    $1, %eax
        call    printf@PLT
        movl    $0, %eax
        movq    -8(%rbp), %rdx
        subq    %fs:40, %rdx
        je      .L7
        call    __stack_chk_fail@PLT
.L7:
        leave
        .cfi_def_cfa 7, 8
        ret
        .cfi_endproc
.LFE15:
        .size   main, .-main
        .section        .rodata
        .align 8
.LC1:
        .long   0
        .long   1079574528
        .ident  "GCC: (Ubuntu 11.2.0-7ubuntu2) 11.2.0"
        .section        .note.GNU-stack,"",@progbits
        .section        .note.gnu.property,"a"
        .align 8
        .long   1f - 0f
        .long   4f - 1f
        .long   5
0:
        .string "GNU"
1:
        .align 8
        .long   0xc0000002
        .long   3f - 2f
2:
        .long   0x3
3:
        .align 8
4:
```

To create the chart, I copied the result as shown below. Note that I had to manually adjust the vector size.

```
christopherlall@LallChristopher-VirtualBox:~/Desktop/THT3$ ./unoptimized.out
size = 16, avg time = 124.590000 ns
christopherlall@LallChristopher-VirtualBox:~/Desktop/THT3$ g++ -O0 Lall_Main.cpp -o unoptimized.out
Lall_Main.cpp:8:9: warning: #pragma once in main file
    8 | #pragma once
      |         ^~~~
christopherlall@LallChristopher-VirtualBox:~/Desktop/THT3$ ./unoptimized.out
size = 32, avg time = 207.430000 ns
christopherlall@LallChristopher-VirtualBox:~/Desktop/THT3$ g++ -O0 Lall_Main.cpp -o unoptimized.out
Lall_Main.cpp:8:9: warning: #pragma once in main file
    8 | #pragma once
      |         ^~~~
christopherlall@LallChristopher-VirtualBox:~/Desktop/THT3$ ./unoptimized.out
size = 64, avg time = 350.010000 ns
christopherlall@LallChristopher-VirtualBox:~/Desktop/THT3$ g++ -O0 Lall_Main.cpp -o unoptimized.out
Lall_Main.cpp:8:9: warning: #pragma once in main file
    8 | #pragma once
      |         ^~~~
christopherlall@LallChristopher-VirtualBox:~/Desktop/THT3$ ./unoptimized.out
size = 128, avg time = 812.730000 ns
christopherlall@LallChristopher-VirtualBox:~/Desktop/THT3$ g++ -O0 Lall_Main.cpp -o unoptimized.out
Lall_Main.cpp:8:9: warning: #pragma once in main file
    8 | #pragma once
      |         ^~~~
christopherlall@LallChristopher-VirtualBox:~/Desktop/THT3$ ./unoptimized.out
size = 256, avg time = 1248.120000 ns
christopherlall@LallChristopher-VirtualBox:~/Desktop/THT3$ g++ -O0 Lall_Main.cpp -o unoptimized.out
Lall_Main.cpp:8:9: warning: #pragma once in main file
    8 | #pragma once
      |         ^~~~
christopherlall@LallChristopher-VirtualBox:~/Desktop/THT3$ ./unoptimized.out
size = 512, avg time = 2734.120000 ns
christopherlall@LallChristopher-VirtualBox:~/Desktop/THT3$ g++ -O0 Lall_Main.cpp -o unoptimized.out
Lall_Main.cpp:8:9: warning: #pragma once in main file
    8 | #pragma once
      |         ^~~~
christopherlall@LallChristopher-VirtualBox:~/Desktop/THT3$ ./unoptimized.out
size = 1024, avg time = 5915.340000 ns
christopherlall@LallChristopher-VirtualBox:~/Desktop/THT3$ g++ -O0 Lall_Main.cpp -o unoptimized.out
Lall_Main.cpp:8:9: warning: #pragma once in main file
    8 | #pragma once
      |         ^~~~
christopherlall@LallChristopher-VirtualBox:~/Desktop/THT3$ ./unoptimized.out
size = 2048, avg time = 10469.610000 ns
```

```
christopherlall@LallChristopher-VirtualBox:~/Desktop/THT3$ g++ -O0 Lall_Main.cpp -o unoptimized.out
Lall_Main.cpp:8:9: warning: #pragma once in main file
    8 | #pragma once
      |         ^~~~
christopherlall@LallChristopher-VirtualBox:~/Desktop/THT3$ ./unoptimized.out
size = 4096, avg time = 37414.190000 ns
christopherlall@LallChristopher-VirtualBox:~/Desktop/THT3$ g++ -O0 Lall_Main.cpp -o unoptimized.out
Lall_Main.cpp:8:9: warning: #pragma once in main file
    8 | #pragma once
      |         ^~~~
christopherlall@LallChristopher-VirtualBox:~/Desktop/THT3$ ./unoptimized.out
size = 8192, avg time = 41399.250000 ns
christopherlall@LallChristopher-VirtualBox:~/Desktop/THT3$ g++ -O0 Lall_Main.cpp -o unoptimized.out
Lall_Main.cpp:8:9: warning: #pragma once in main file
    8 | #pragma once
      |         ^~~~
christopherlall@LallChristopher-VirtualBox:~/Desktop/THT3$ ./unoptimized.out
size = 16384, avg time = 143378.480000 ns
christopherlall@LallChristopher-VirtualBox:~/Desktop/THT3$ g++ -O0 Lall_Main.cpp -o unoptimized.out
Lall_Main.cpp:8:9: warning: #pragma once in main file
    8 | #pragma once
      |         ^~~~
christopherlall@LallChristopher-VirtualBox:~/Desktop/THT3$ ./unoptimized.out
size = 32768, avg time = 209318.870000 ns
christopherlall@LallChristopher-VirtualBox:~/Desktop/THT3$ g++ -O0 Lall_Main.cpp -o unoptimized.out
Lall_Main.cpp:8:9: warning: #pragma once in main file
    8 | #pragma once
      |         ^~~~
christopherlall@LallChristopher-VirtualBox:~/Desktop/THT3$ ./unoptimized.out
size = 65536, avg time = 491652.480000 ns
```



Optimized on Linux GCC:

With Auto Vectorization and Auto Parallelization enabled, we can now run our app. To accomplish this, no changes to the main.c file are required. Instead, we'll have to change the commands used to compile and run the program. Let's start with the compiler-generated optimized assembly code.

```
#include <stdio.h>
#include <stdint.h>
#include <stdlib.h>
#include <time.h>
using namespace std;
#define SIZE 65536 //32 //64 //128 //256 //512 //10244 //2048 // 4096 //8192 //16384 //32768 //65536
//function prototype
#pragma once
int computeDotProduct(int* a, int* b, size_t n) {
    int sum = 0;
#pragma loop(hint_parallel(0))
    for (size_t i = 0; i < n; ++i) {
        sum = sum + *(a + i) * (*b + i);
    }
    return sum;
}
static int Array[SIZE];
int main(int argc, char **argv)
{
 uint64_t timed;
 //double time;
 double avg = 0.0;
 struct timespec start, end;
 for (int i=0; i<100; i++){
 clock_gettime(CLOCK_MONOTONIC, &start);
computeDotProduct(Array, Array ,SIZE);
 //recorded end time
 clock_gettime(CLOCK_MONOTONIC, &end);
 timed = (end.tv_sec - start.tv_sec)*1000000000ULL + end.tv_nsec - start.tv_nsec;
 //printf("time = %f ns\n", (double) timed);
 avg += timed;
 }
 avg/= 100;
 printf("size = %i, avg time = %f ns \n", SIZE, avg);
 return 0;
}
```

As stated above, the code stays the same. We must add settings to the GCC compiler to allow Auto Vectorization and Auto Parallelization. The -O2 flag, on the other hand, handles both of these optimizations and more. We'll go in the order listed below. Type the following in the terminal to get the assembly.

```
christopherlall@LallChristopher-VirtualBox:~/Desktop/THT3$ g++ -S -O2 Lall_Main
.cpp -o optimized
Lall_Main.cpp:8:9: warning: #pragma once in main file
    8 | #pragma once
      |         ^~~~
```

Compiler-generated ASM code for optimization:

```
christopherlall@LallChristopher-VirtualBox:~/Desktop/THT3$ cat optimized
        .file   "Lall_Main.cpp"
        .text
        .p2align 4
        .globl  _Z17computeDotProductPiS_m
        .type   _Z17computeDotProductPiS_m, @function
_Z17computeDotProductPiS_m:
.LFB55:
        .cfi_startproc
        endbr64
        testq   %rdx, %rdx
        je      .L4
        movl    (%rsi), %eax
        leaq    (%rdi,%rdx,4), %rcx
        xorl    %r8d, %r8d
        .p2align 4,,10
        .p2align 3
.L3:
        movl    (%rdi), %edx
        addq    $4, %rdi
        imull   %eax, %edx
        addl    $1, %eax
        addl    %edx, %r8d
        cmpq    %rdi, %rcx
        jne     .L3
        movl    %r8d, %eax
        ret
        .p2align 4,,10
        .p2align 3
.L4:
        xorl    %r8d, %r8d
        movl    %r8d, %eax
        ret
        .cfi_endproc
.LFE55:
        .size   _Z17computeDotProductPiS_m, .-_Z17computeDotProductPiS_m
        .section        .rodata.str1.1,"aMS",@progbits,1
.LC2:
        .string "size = %i, avg time = %f ns \n"
        .section        .text.startup,"ax",@progbits
        .p2align 4
        .globl  main
        .type   main, @function
main:
.LFB56:
        .cfi_startproc
        endbr64
```

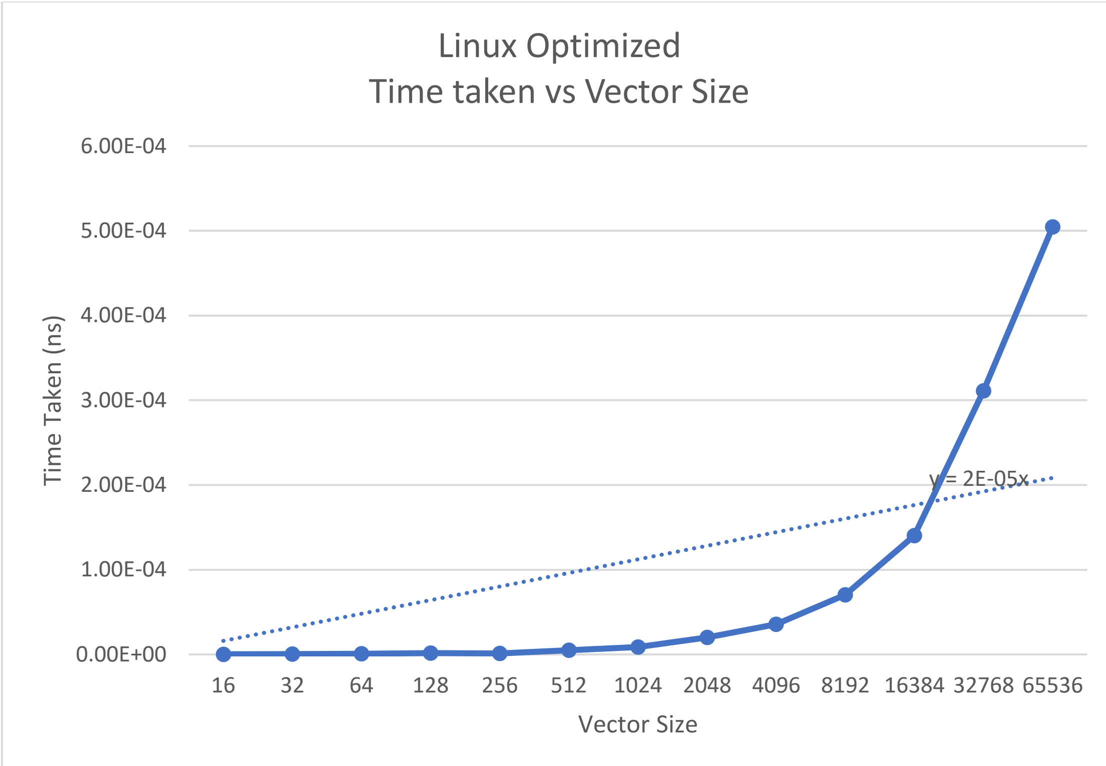
```
        pushq   %r12
        .cfi_def_cfa_offset 16
        .cfi_offset 12, -16
        pxor    %xmm2, %xmm2
        pushq   %rbp
        .cfi_def_cfa_offset 24
        .cfi_offset 6, -24
        pushq   %rbx
        .cfi_def_cfa_offset 32
        .cfi_offset 3, -32
        movl    $100, %ebx
        subq    $64, %rsp
        .cfi_def_cfa_offset 96
        movq    %fs:40, %rax
        movq    %rax, 56(%rsp)
        xorl    %eax, %eax
        leaq    16(%rsp), %r12
        movsd   %xmm2, 8(%rsp)
        leaq    32(%rsp), %rbp
        jmp     .L10
        .p2align 4,,10
        .p2align 3
.L15:
        pxor    %xmm0, %xmm0
        cvtsi2sdq       %rax, %xmm0
.L9:
        addsd   8(%rsp), %xmm0
        movsd   %xmm0, 8(%rsp)
        subl    $1, %ebx
        je      .L14
.L10:
        movq    %r12, %rsi
        movl    $1, %edi
        call    clock_gettime@PLT
        movq    %rbp, %rsi
        movl    $1, %edi
        call    clock_gettime@PLT
        movq    32(%rsp), %rax
        subq    16(%rsp), %rax
        imulq   $1000000000, %rax, %rax
        addq    40(%rsp), %rax
        subq    24(%rsp), %rax
        jns     .L15
        movq    %rax, %rdx
        andl    $1, %eax
        pxor    %xmm0, %xmm0
        shrq    %rdx
        orq     %rax, %rdx
        cvtsi2sdq       %rdx, %xmm0
```

```
        addsd   %xmm0, %xmm0
        jmp     .L9
.L14:
        movl    $65536, %edx
        movl    $1, %edi
        leaq    .LC2(%rip), %rsi
        movl    $1, %eax
        divsd   .LC1(%rip), %xmm0
        call    __printf_chk@PLT
        movq    56(%rsp), %rax
        subq    %fs:40, %rax
        jne     .L16
        addq    $64, %rsp
        .cfi_remember_state
        .cfi_def_cfa_offset 32
        xorl    %eax, %eax
        popq    %rbx
        .cfi_def_cfa_offset 24
        popq    %rbp
        .cfi_def_cfa_offset 16
        popq    %r12
        .cfi_def_cfa_offset 8
        ret
.L16:
        .cfi_restore_state
        call    __stack_chk_fail@PLT
        .cfi_endproc
.LFE56:
        .size   main, .-main
        .section        .rodata.cst8,"aM",@progbits,8
        .align 8
.LC1:
        .long   0
        .long   1079574528
        .ident  "GCC: (Ubuntu 11.2.0-7ubuntu2) 11.2.0"
        .section        .note.GNU-stack,"",@progbits
        .section        .note.gnu.property,"a"
        .align 8
        .long   1f - 0f
        .long   4f - 1f
        .long   5
0:
        .string "GNU"
1:
        .align 8
        .long   0xc0000002
        .long   3f - 2f
2:
        .long   0x3
3:
        .align 8
4:
```

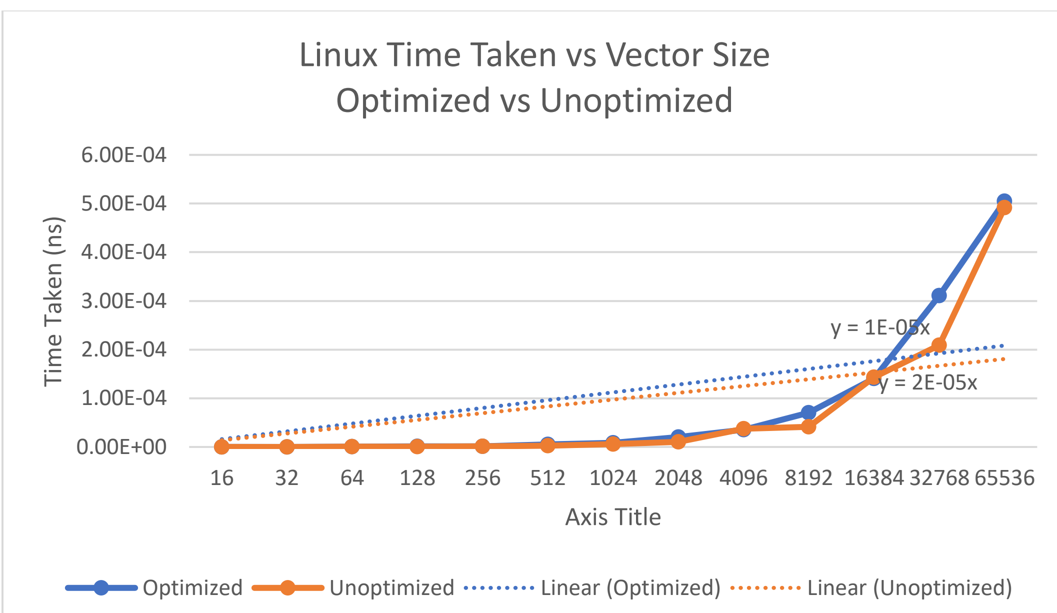
To create the chart, I copied the result as shown below. Note that I had to manually adjust the vector size.

```
christopherlall@LallChristopher-VirtualBox:~/Desktop/THT3$ g++ Lall_Main.cpp -o
 optimized.out
Lall_Main.cpp:8:9: warning: #pragma once in main file
    8 | #pragma once
      |         ^~~~
christopherlall@LallChristopher-VirtualBox:~/Desktop/THT3$ ./optimized.out
size = 65536, avg time = 504534.460000 ns
christopherlall@LallChristopher-VirtualBox:~/Desktop/THT3$ g++ Lall_Main.cpp -o
 optimized.out
Lall_Main.cpp:8:9: warning: #pragma once in main file
    8 | #pragma once
      |         ^~~~
christopherlall@LallChristopher-VirtualBox:~/Desktop/THT3$ ./optimized.out
size = 32768, avg time = 311042.940000 ns
christopherlall@LallChristopher-VirtualBox:~/Desktop/THT3$ g++ Lall_Main.cpp -o
 optimized.out
Lall_Main.cpp:8:9: warning: #pragma once in main file
    8 | #pragma once
      |         ^~~~
christopherlall@LallChristopher-VirtualBox:~/Desktop/THT3$ ./optimized.out
size = 16384, avg time = 140428.210000 ns
christopherlall@LallChristopher-VirtualBox:~/Desktop/THT3$ g++ Lall_Main.cpp -o
 optimized.out
Lall_Main.cpp:8:9: warning: #pragma once in main file
    8 | #pragma once
      |         ^~~~
christopherlall@LallChristopher-VirtualBox:~/Desktop/THT3$ ./optimized.out
size = 8192, avg time = 70416.910000 ns
christopherlall@LallChristopher-VirtualBox:~/Desktop/THT3$ g++ Lall_Main.cpp -o
 optimized.out
Lall_Main.cpp:8:9: warning: #pragma once in main file
    8 | #pragma once
      |         ^~~~
christopherlall@LallChristopher-VirtualBox:~/Desktop/THT3$ ./optimized.out
size = 4096, avg time = 35611.530000 ns
christopherlall@LallChristopher-VirtualBox:~/Desktop/THT3$ g++ Lall_Main.cpp -o
 optimized.out
Lall_Main.cpp:8:9: warning: #pragma once in main file
    8 | #pragma once
      |         ^~~~
christopherlall@LallChristopher-VirtualBox:~/Desktop/THT3$ ./optimized.out
size = 4096, avg time = 46424.530000 ns
```

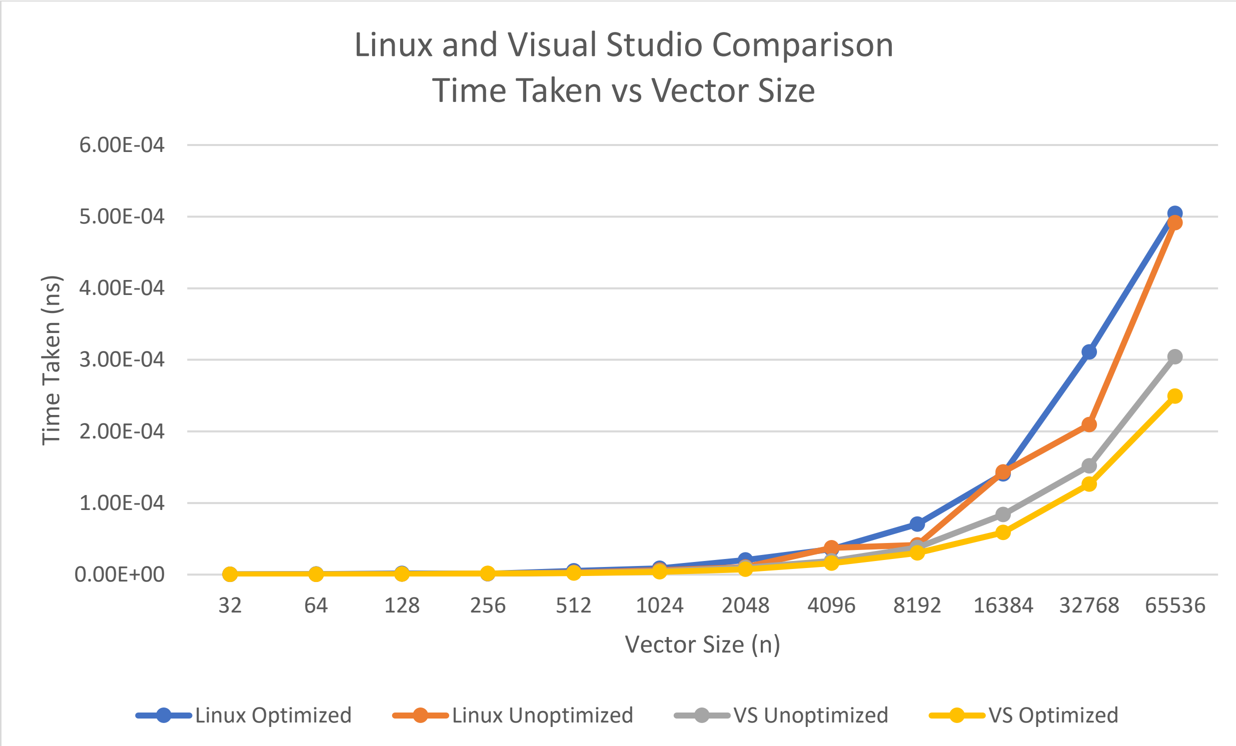
```
size = 2048, avg time = 20263.320000 ns
christopherlall@LallChristopher-VirtualBox:~/Desktop/THT3$ g++ Lall_Main.cpp -o
 optimized.out
Lall_Main.cpp:8:9: warning: #pragma once in main file
    8 | #pragma once
      |         ^~~~
christopherlall@LallChristopher-VirtualBox:~/Desktop/THT3$ ./optimized.out
size = 1024, avg time = 8645.840000 ns
christopherlall@LallChristopher-VirtualBox:~/Desktop/THT3$ g++ Lall_Main.cpp -o
 optimized.out
Lall_Main.cpp:8:9: warning: #pragma once in main file
    8 | #pragma once
      |         ^~~~
christopherlall@LallChristopher-VirtualBox:~/Desktop/THT3$ ./optimized.out
size = 512, avg time = 5009.090000 ns
christopherlall@LallChristopher-VirtualBox:~/Desktop/THT3$ g++ Lall_Main.cpp -o
 optimized.out
Lall_Main.cpp:8:9: warning: #pragma once in main file
    8 | #pragma once
      |         ^~~~
christopherlall@LallChristopher-VirtualBox:~/Desktop/THT3$ ./optimized.out
size = 256, avg time = 1224.630000 ns
christopherlall@LallChristopher-VirtualBox:~/Desktop/THT3$ g++ Lall_Main.cpp -o
 optimized.out
Lall_Main.cpp:8:9: warning: #pragma once in main file
    8 | #pragma once
      |         ^~~~
christopherlall@LallChristopher-VirtualBox:~/Desktop/THT3$ ./optimized.out
size = 128, avg time = 1579.470000 ns
christopherlall@LallChristopher-VirtualBox:~/Desktop/THT3$ g++ Lall_Main.cpp -o
 optimized.out
Lall_Main.cpp:8:9: warning: #pragma once in main file
    8 | #pragma once
      |         ^~~~
christopherlall@LallChristopher-VirtualBox:~/Desktop/THT3$ ./optimized.out
size = 64, avg time = 661.740000 ns
christopherlall@LallChristopher-VirtualBox:~/Desktop/THT3$ g++ Lall_Main.cpp -o
 optimized.out
Lall_Main.cpp:8:9: warning: #pragma once in main file
    8 | #pragma once
      |         ^~~~
christopherlall@LallChristopher-VirtualBox:~/Desktop/THT3$ ./optimized.out
size = 32, avg time = 345.270000 ns
christopherlall@LallChristopher-VirtualBox:~/Desktop/THT3$ g++ Lall_Main.cpp -o
 optimized.out
Lall_Main.cpp:8:9: warning: #pragma once in main file
    8 | #pragma once
      |         ^~~~
christopherlall@LallChristopher-VirtualBox:~/Desktop/THT3$ ./optimized.out
size = 16, avg time = 210.610000 ns
```



We will see that it is linear, as predicted, if we plot it. This graph appears higher than the graph with Optimization turned off. However, in my case it seems as though the optimized and unoptimized do not have much of a difference although unoptimized is faster.



Window Vs Linux Graph Comparison:



Finally, we've seen how optimization works when calculating the dot product with various vector sizes. Below is a graph comparing Windows and Linux with optimization enabled and disabled.

Through observation, Visual Studio took less time the higher the Vector Sizes became compared to Linux.

Conclusion:

This lab taught me how to run vector commands and interpret their assembly language counterparts. I learned how to simplify code in order to make it look lot nicer than previously. I learned a lot, for example, VS is faster regardless of if it is optimized or not. With a minor performance improvement and shorter run time, manual optimization would have been more successful than compiler optimization. This work improved my overall grasp of how optimization may affect systems.