

**COMPUTER ORGANIZATION FINAL PROJECT**

**ONLY ONE GROUP MEMBER : BATUHAN SATILMIS B1805.010039**

**STEPS FOR FINAL PROJECT :**

1.Introduction

2.Abstract

3. Literature Review

4. Hardware Acceleration Methods

5. Experimental Setup

6. Results and Discussion

7. Conclusion

8. References

**ABSTRACT**

If I summarize the Final Project, By researching the hardware acceleration, I expressed the relationship between the final project and the descriptive codes via c ++. The names of the codes I use in my homework are briefly as follows; Unoptimized Code, AVX Code, AVX + Unroll Code, AVX + Unroll + blocked Code and OpenGL codes have been run through certain stages in Visual Studio 2019.And the measurement results are noted and added to the presentation file. I obtained individually the codes by searching from the book titled Computer Organization Design 5th Edition 2014 as stated in the final assignment (Com Org Project 2020 Spring.docx). By doing research on the content of the given final assignment, I answered the questions about how it was done and what was the result and I analyzed the results of each step. I used charts to better understand the results.

**INTRODUCTION**

Successful programmers are always concerned about the performance of their programs, because getting results quickly to the user is critical to creating successful software. In the 1960s and 1970s, the primary constraint on computer performance was the size of the computer's memory. According to professional programmers,many programmers often followed a simple belief: to speed up programs, they minimized the memory space. Programmers interested in performance now need to understand the problems that replaced the simple memory model of the 1960s: the parallel nature of the processors and the hierarchical nature of the memories. Programmers who want to create will need to increase their knowledge of **Computer Organization.**

**LITERATURE REVIEW**

In computing, hardware acceleration is the use of computer hardware specially made to perform some functions more efficiently than is possible in software running on a general-purpose central processing unit (CPU). Any transformation of data or routine that can be computed, can be calculated purely in software running on a generic CPU, purely in custom-made hardware, or in some mix of both. An operation can be computed faster in application-specific hardware designed or programmed to compute the operation than specified in software and performed on a general-purpose computer processor. Each approach has advantages and disadvantages. The implementation of computing tasks in hardware to decrease latency and increase throughput is known as hardware acceleration.

Examples of hardware acceleration include bit blit acceleration functionality in graphics processing units (GPUs), use of memristors for accelerating neural networks and regular expression hardware acceleration for spam control in the server industry, intended to prevent regular expression denial of service (ReDoS) attacks. The hardware that performs the acceleration may be part of a general-purpose CPU, or a separate unit. In the second case, it is referred to as a hardware accelerator, or often more specifically as a 3D accelerator, cryptographic accelerator, etc.

**Hardware Acceleration Methods**

**A few examples of hardware accelerators, we can list them as follows :**

**CUDA:**

Compute Unified Device Architecture, or CUDA for short, is a parallel computing platform developed by Nvidia and an application programming interface (API) model. This allows software developers and software developers to use a CUDA compatible GPU for general operation. The CUDA platform is a layer of software that allows the graphics processor to provide direct access to a number of virtual instructions and parallel computing elements to run computer kernel.

**Coprocessor:**

A co-processor is a computer processor used to complete the functions of the main processor (CPU). Processes performed by the co-processor can be arithmetic of mobile understatement, graphics, signal processing, uterine processing, cryptography or I / O intefacing with peripherals. Co-processors can speed up the system by removing the CPU processor from the main processor. Co-processors allow you to customize multiple computers so that customers who don't need the extra performance don't have to pay.

**Comparison of traditional and multi-core processors in terms of hardware acceleration:**

Traditionally, processors were sequential (instructions are executed one by one), and were designed to run general purpose algorithms controlled by instruction fetch (for example moving temporary results to and from a register file). Hardware accelerators improve the execution of a specific algorithm by allowing greater concurrency, having specific datapaths for their temporary variables, and reducing the overhead of instruction control in the fetch-decode-execute cycle.

Modern processors are multi-core and often feature parallel "single-instruction; multiple data" (SIMD) units. Even so, hardware acceleration still yields benefits. Hardware acceleration is suitable for any computation-intensive algorithm which is executed frequently in a task or program. Depending upon the granularity, hardware acceleration can vary from a small functional unit, to a large functional block (like motion estimation in MPEG-2).

**What is the GPU Optimization?**

As GPU programmability has become more pervasive and GPU performance has become almost irresistibly appealing, increasing numbers of programmers have begun to recast applications of all sorts to make use of GPUs. But an interesting trend has appeared along the way: it seems that many programmers make the same performance mistakes in their GPU programs regardless of how much experience they have programming CPUs. The goal of this chapter is to help CPU programmers who are new to GPU programming avoid some of these common mistakes so that they gain the benefits of GPU performance without all the headaches of the GPU programmer's learning curve.

**What is the Parallel Computing?**

Parallel computing, or Parallel computing, is the same task to run simultaneously on multiple processors to achieve results faster. This idea is based on dividing the solution of problems into small pieces of task and coordinating them simultaneously.

**What is the Instruction-level parallelism?**

The potential overlap between commands is called command-level parallelism. It can be realized in software and hardware. While the techniques in the software are fixed, the techniques in the hardware provide dynamic parallelism.

**What is the Advanced Vector Extensions?**

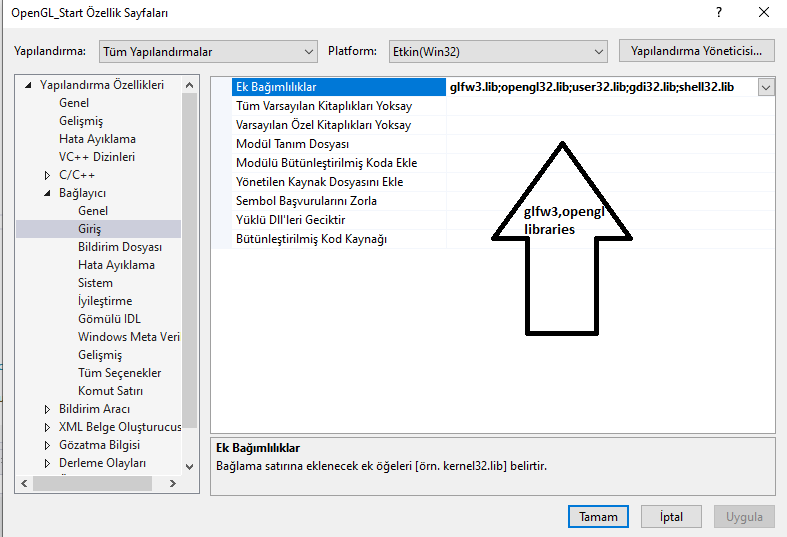
Advanced Vector Extensions (AVX, also known as Sandy Bridge New Extensions) are extensions to the x86 instruction set architecture for microprocessors from Intel and AMD proposed by Intel in March 2008 and first supported by Intel with the Sandy Bridge processor shipping in Q1 2011 and later on by AMD with the Bulldozer processor shipping in Q3 2011. AVX provides new features, new instructions and a new coding scheme.

AVX2 expands most integer commands to 256 bits and introduces fused multiply-accumulate (FMA) operations. AVX-512 expands AVX to 512-bit support using a new EVEX prefix encoding proposed by Intel in July 2013 and first supported by Intel with the Knights Landing processor, which shipped in 2016.

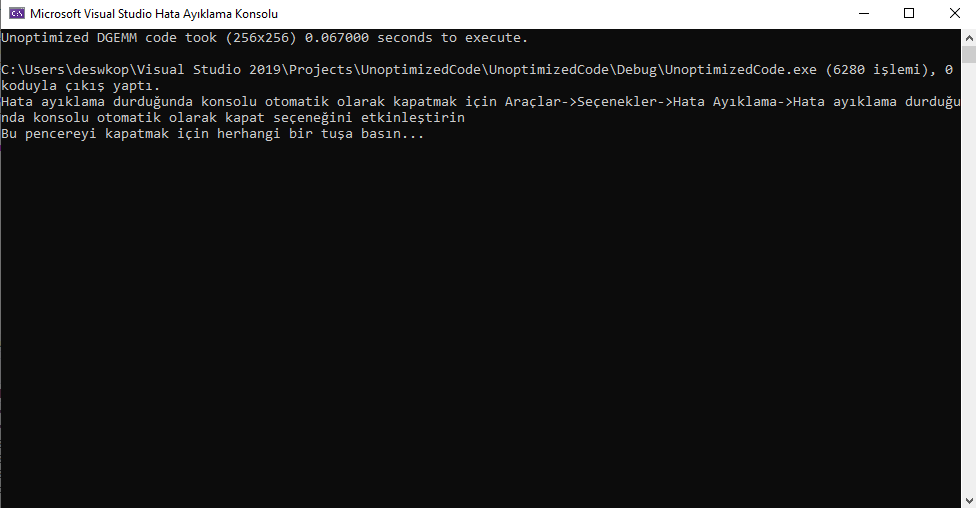
**EXPERIMENTAL SETUP**

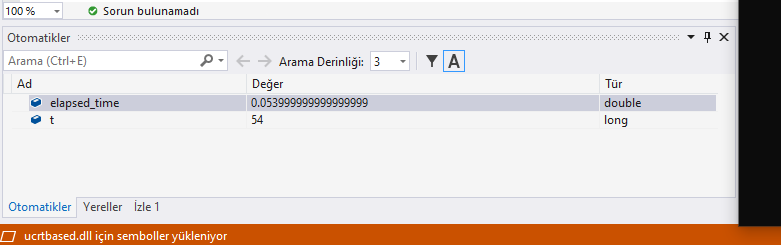
* I got help from visual studio 2019 to use AVX intrinsics. I also added the openGL library for V-5 to my project. I used c ++ as the software language.
* I measured the run time by keeping the array dimensions variable.
* Since I don't have any NVIDIA brand graphics cards to run Cuda codes, I completed version 5 using open source openGL.
* I reinforced my project by transferring the measurement results obtained from all 5 versions to the chart graphs.

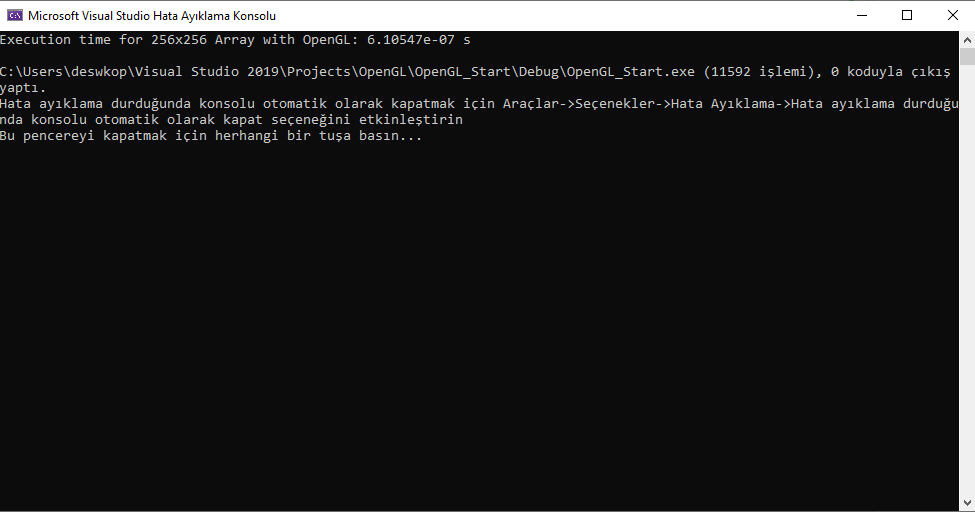
My processes of adding the OpenGL library to the Visual Studio program :



**SOME OUTPUTS ABOUT EXECUTIONS**



****

****

**CHART RESULTS**

**DISCUSSION AND RESULT**

As the size of the arrray grew in the unoptimized code, the time to complete the program increased at the same rate. There was no noticeable performance increase in any way, and even as the array size increased, the performance decreased at that rate.

When I run the code with the help of AVX, an incredible performance increase was observed in our 256x256 array. (More than 3 times) The increase in performance was observed even though the size of Array was not the same as it grew. V2-AVX code gave better results than non-optimized code and executed at least 3 times faster.

When I run the V2 code with the help of unroll, even though small array sizes like 256,512,768 do not increase performance, it is observed that the execution time of the arrays of 1024 and 1280 dimensions accelerated, which enables us to get the result that unroll accelerates the program as the size of the program grows.

In addition to the v3 code, it was unrolled and lost its unroll effect on the v4 code that was blocked, and it was able to execute the array dimensions in a longer time. The more surprising part is that it executes the array size 1024x1024 for more than gets time 1280x1280.

OpenGL code, with the help of the chrono library in c ++, I created a different array than the codes in the book (I divided it into 2 parts, first and second). Just like other versions, I executed the arrays in 256,512,768,1024 and 1280 sizes and also opened the openGL screen.

As a result of I made a video about how I run the programs and how I get the output. I also put all the codes and charts in the file.

**REFERENCES**

[**https://www.glfw.org**](https://www.glfw.org)

[**https://www.omnisci.com**](https://www.omnisci.com)

[**https://whatis.techtarget.com**](https://whatis.techtarget.com)

[**https://www.sciencedirect.com**](https://www.sciencedirect.com)

[**https://blogs.nvidia.com**](https://blogs.nvidia.com)

[**https://www.quora.com**](https://www.quora.com)

[**https://stackoverflow.com**](https://stackoverflow.com)