

**Department of Electrical and Computer Engineering**

**North South University**

**CSE 425.2: Concepts of Programming Languages**

**Project Title:**

**Performance Evaluation of Custom C Compiler Optimized with Loop Parallelization and Vectorization Techniques Using LLVM**

**Team Drowning\_in\_Despair**

| **Name** | **ID** |
| --- | --- |
| **Wahid Ahmed** | **2121896042** |
| **Fiaz Al Abid** | **2111096642** |
| **Sumaiya Jahan Sharlin** | **2011126042** |

**Abstract:**

The aim of this project is to design, implement and evaluate an optimized custom C compiler that incorporates loop parallelization and vectorization techniques within the intermediate representation (IR), utilizing the low level virtual machine (LLVM) as a foundation. The primary objective is to reduce the computational cost. This will be verified by comparing the computational costs of the optimized compiler with other compilers.

The methodology will involve developing two custom C compilers: a baseline compiler without any optimizations and an optimized compiler utilizing loop parallelization and vectorization strategies in the IR. Both compilers will be built using the LLVM infrastructure, providing a foundation for experimentation, development, optimization, and comparison. Some C programs will be used to evaluate the performance of the custom compilers and the LLVM C compiler (Clang) in terms of execution time, memory usage, and energy consumption.

By comparing the performance of the optimized compiler with the custom baseline compiler and Clang, we seek to contribute to the understanding of compiler optimization techniques in the IR and their impact on computational costs and C program execution. The findings may also provide insights for future compiler development and optimization.

**Introduction and Motivation**:

In this advancing technological landscape, the efficiency of compilers is very important. Effective optimization techniques, especially in the IR, play an essential role in enhancing compiler performance. Loop parallelization and vectorization are two essential methods that can significantly improve the speed and efficiency of C compilers. This project evaluates a custom baseline compiler, an optimized compiler, and Clang using the LLVM framework. We aim to determine which C compiler has the least computational cost.

The motivation behind this project arises from the growing demand for high-performance computing across various fields. With the rise of multi-core processors, it is essential for developers to leverage these optimizations effectively. We want to identify which compiler is most effective in optimizing C programs and enhancing overall computational efficiency.

**Background and Significance:**

The existing LLVM-based C compilers like Clang are efficient enough to handle various problems. However, its complexity can be very high and cause extreme resource consumption. Optimized compilers are essential for reducing computational costs and maintaining efficiency.

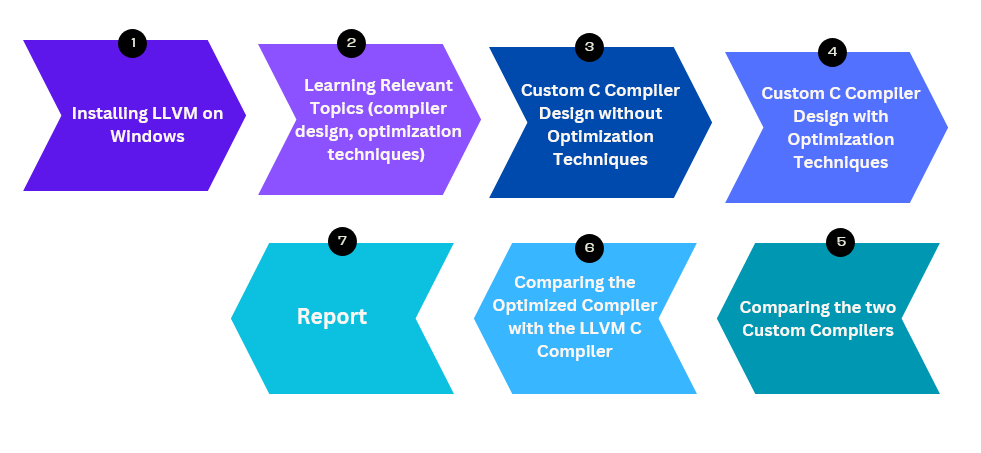
Compiler performance may vary for different compilers for the same code in the same machine. Chanhyun Park [1] has compared the performance of GCC and LLVM compilers for the EISC-embedded processor. In their research, they found that the GCC compiler outperforms LLVM in performance and code size.

Pradnya Khalate [2] has presented an LLVM-based C++ compiler that enhances quantum computing abilities using a quantum device that acts as an accelerator in their paper. Using their optimized LLVM-based C++ compiler, anyone can implement complex quantum computing algorithms in C++.

The frontend P4LLVM has been designed to convert the code of P4-16 to LLVM IR. Using P4LLVM, Tharun Kumar Dangeti [3] has shown performance improvement compared with a well-known compiler for P4-16, P4C. After the optimization steps, they used JSON for backend purposes.

In our project, we plan to build an efficient C compiler with lower computational costs, focusing on efficiency in the IR.

**Method and Design:**



After installing LLVM and studying compiler design principles, we will develop two custom C compilers: a baseline compiler and an optimized compiler with loop parallelization and vectorization techniques in the IR. We will then compare both compilers in terms of computational cost. Finally, we will compare the optimized compiler with the LLVM C compiler.

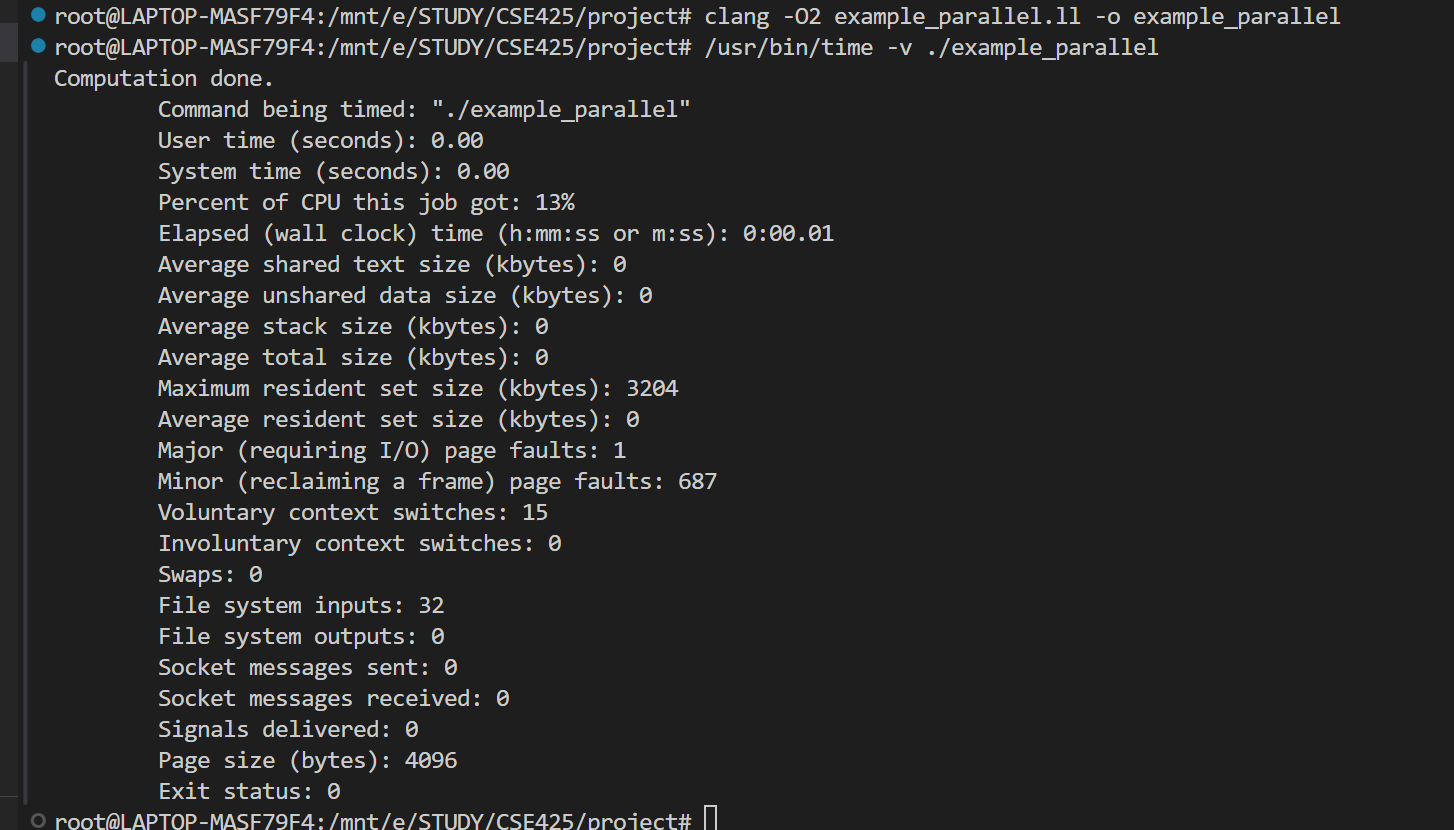
**References:**

1. Chanhyun Park, Miseon Han, Hokyoon Lee, and Seon Wook Kim. Performance comparison of gcc and llvm on the eisc processor. In 2014 International Conference on Electronics, Information and Communications (ICEIC), pages 1–2. IEEE, 2014.
2. Pradnya Khalate, Xin-Chuan Wu, Shavindra Premaratne, Justin Hogaboam, Adam Holmes, Albert Schmitz, Gian Giacomo Guer-reschi, and Xiang Zou. An llvm-based c++ compiler toolchain.
3. Tharun Kumar Dangeti, Ramakrishna Upadrasta, et al. P4llvm: An llvm based p4 compiler. In 2018 IEEE 26th International Conference on Network Protocols (ICNP), pages 424–429. IEEE, 2018.

Technical Strategies for Implementation:

1. Installing necessary Softwares and Tools:
   * Windows Subsystem for Linux(WSL2): For using Ubuntu 22.04
   * LLVM
   * Clang
   * Gcc
   * Flex
   * Bison
2. Comparison by using Clang and LLVM:
   * Create a sample C program with loops
   * Compile the sample program without optimizations and measure the performance
   * Generate the IR using LLVM and then optimize it using loop parallelization and vectorization and measure the performance
   * Evaluate and compare the performances
3. Comparison by building and using a Custom C compiler:
   * Build a lexer using Flex
   * Build a parser using Bison
   * Write a program with loops
   * Generate the IR, optimize and then execute using LLVM
   * Evaluate and compare the performances like before

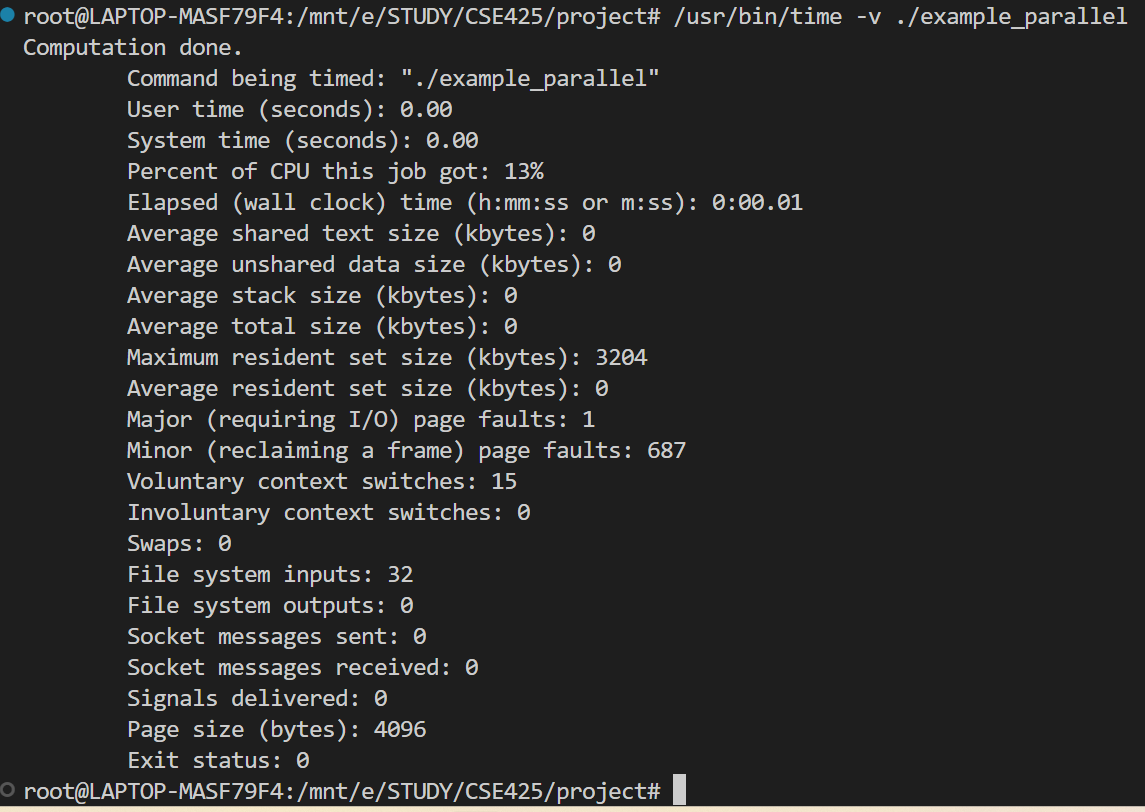
Implementation So Far:

1. Environment Setup: Installed WSL2, and then Ubuntu 22.04. Then used the Ubuntu terminal and installed LLVM, Clang, Gcc, Flex and Bison.
2. Made a benchmark c program for testing that has loops and arrays. Then compiled the sample program without optimizations and measured the execution time and memory usage using /usr/bin/time 
3. Successfully generated LLVM IR for the sample program using Clang. Applied loop vectorization and unrolling optimizations using the opt tool. Measured performance again.

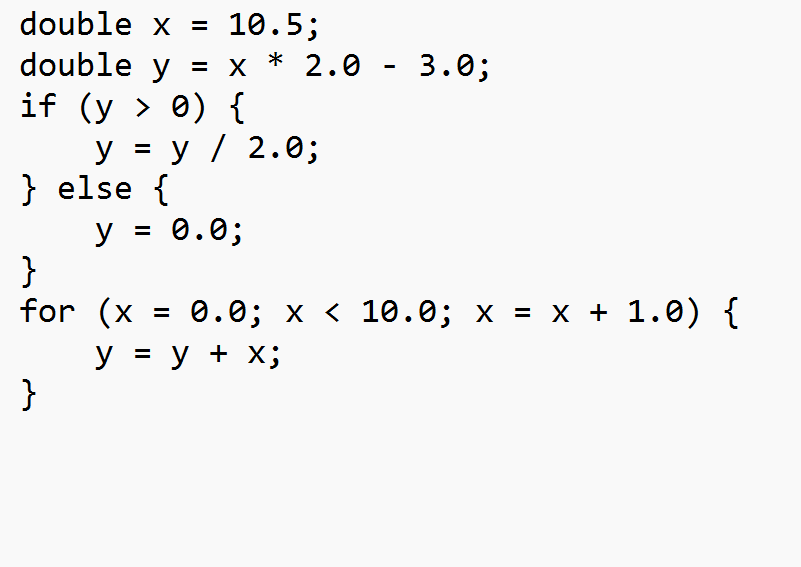
Optimization command:



Performance:



1. For the custom c compiler, made a lexer using flex and ran a sample program.

Program:

Output of lexer:

