McMaster University
ENGINEERING
Computing & Software

Analyzing the Evolution of the Low Level Virtual Machine (LLVM) Compiler Infrastructure

McSCert

Ahmed Elzaria[1], Jonah Alle Monne[2], Sebastien Mosser PhD[1]

[1] Department of Computing and Software, McMaster University, Hamilton, Canada. [2] Department of Computer Science and Electronics, Université Grenoble Alpes, Grenoble, France

Introduction

- Computers do not understand user written code right away, *Fig 1*.
  - **High level code** is what humans use to convey a set of instructions to the computer.
  - **Low level machine code** is what computers understand.
  - Computers utilize a set of **complex programs** called the **compiler** to **translate** high level code to low level machine code
  - **LLVM** provides a set of tools and libraries to build compilers [1].
- During the compilation process, an **intermediate representation** of the user written code is generated, *Fig 2*.
  - Using **LLVM**, **passes** are applied on this representation to further **optimize** and generate **efficient** machine code, *Fig 2*.
  - With over 70 available LLVM passes, there are over **70 factorial possible pass combinations** available which is **greater than number of atoms in the universe**
- As a result, an **enormous field** of **pass interactions and influences** which can change the fate of a computer program, remain **undiscovered/analyzed**.

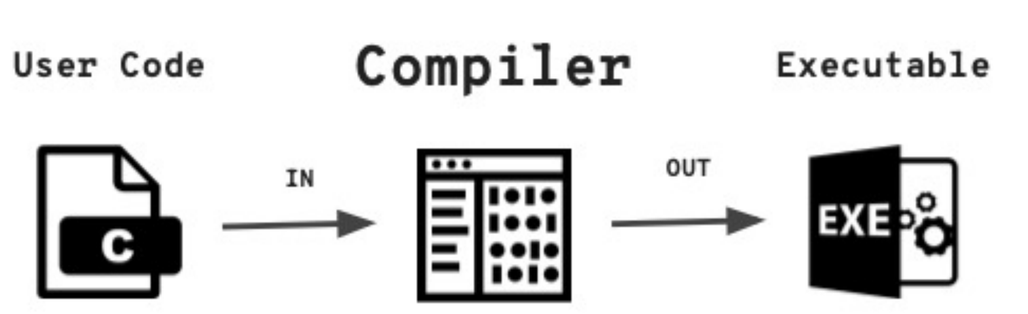

Figure 1: Three main step process a computer takes to convert user code to machine code.

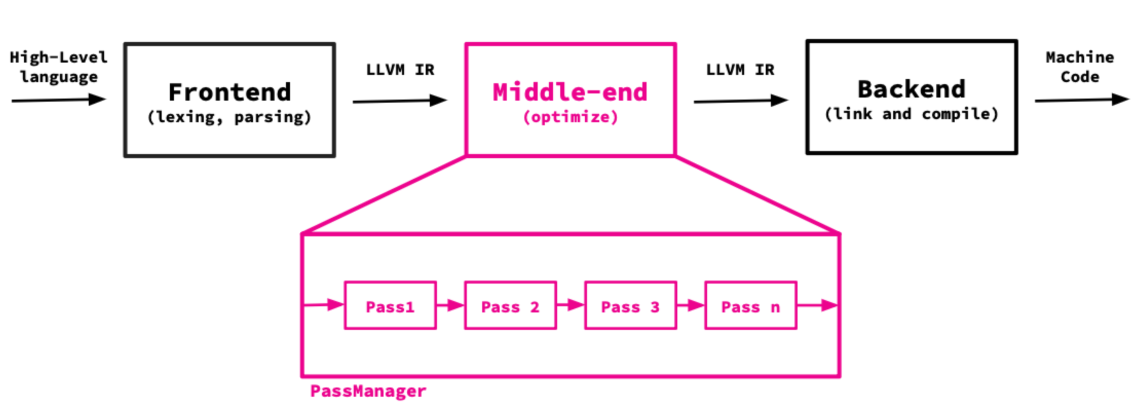

Figure 2: Compilation Process. Passes are applied during middle-end stage to further optimize code. Resulting in more efficient machine code.

Objectives

- **Analyze** and **visualize** the evolution of the passes and their dependencies.
- **Study** visualizations to reveal patterns, key traits, program behaviour, etc.

Subjects and Methods

Approach

- **In theory**, it is best to discover and analyze all possible pass combinations but impossible in practice.
- Hence, analyzing and comparing **sampled sections** within the field (microscope effect) is the approach.

Acquiring Reference Benchmark of Programs

- Utilizing the "**Angha Project**" [2] benchmark consisting of **1 million clang compiled .c programs**.
- After parsing each program and **extracting structural metrics** (*number of variables, functions, control flows, etc.*), it was evident that the benchmark can be downsized to **3600 random programs without bias** since metrics revealed a **homogenous benchmark**.

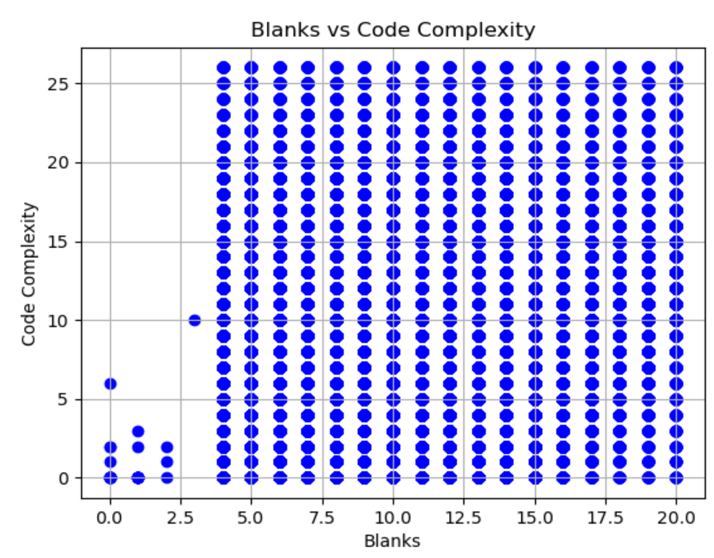

Figure 3: Number of blanks vs code complexity of a program. All graphs revealed a similar result, indicating a homogenous benchmark.

Building the Transition Graph (Pass Microscope)

- A 4-step process is executed on each program:
  - Select passes (*ex. loop-unroll, mem2reg, etc.*).
  - LLVM IR is generated.
  - Apply each pass on IR:
    - Using **llvm-diff**, determine if post-pass applied version of the program is identical to any other versions generated.
    - If unique, add a new node to the graph.
    - Else, add an edge from parent node to identical node.
    - Keep track of newly generated nodes to visit in a queue.
  - Repeat for all nodes in queue until empty.

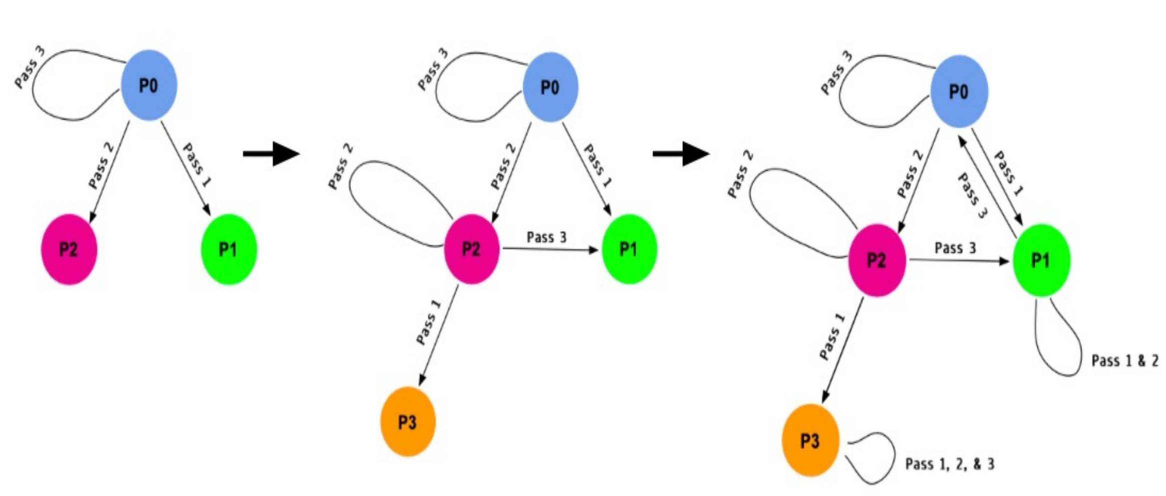

Figure 4: Demonstration of how transition graphs are built.

Results

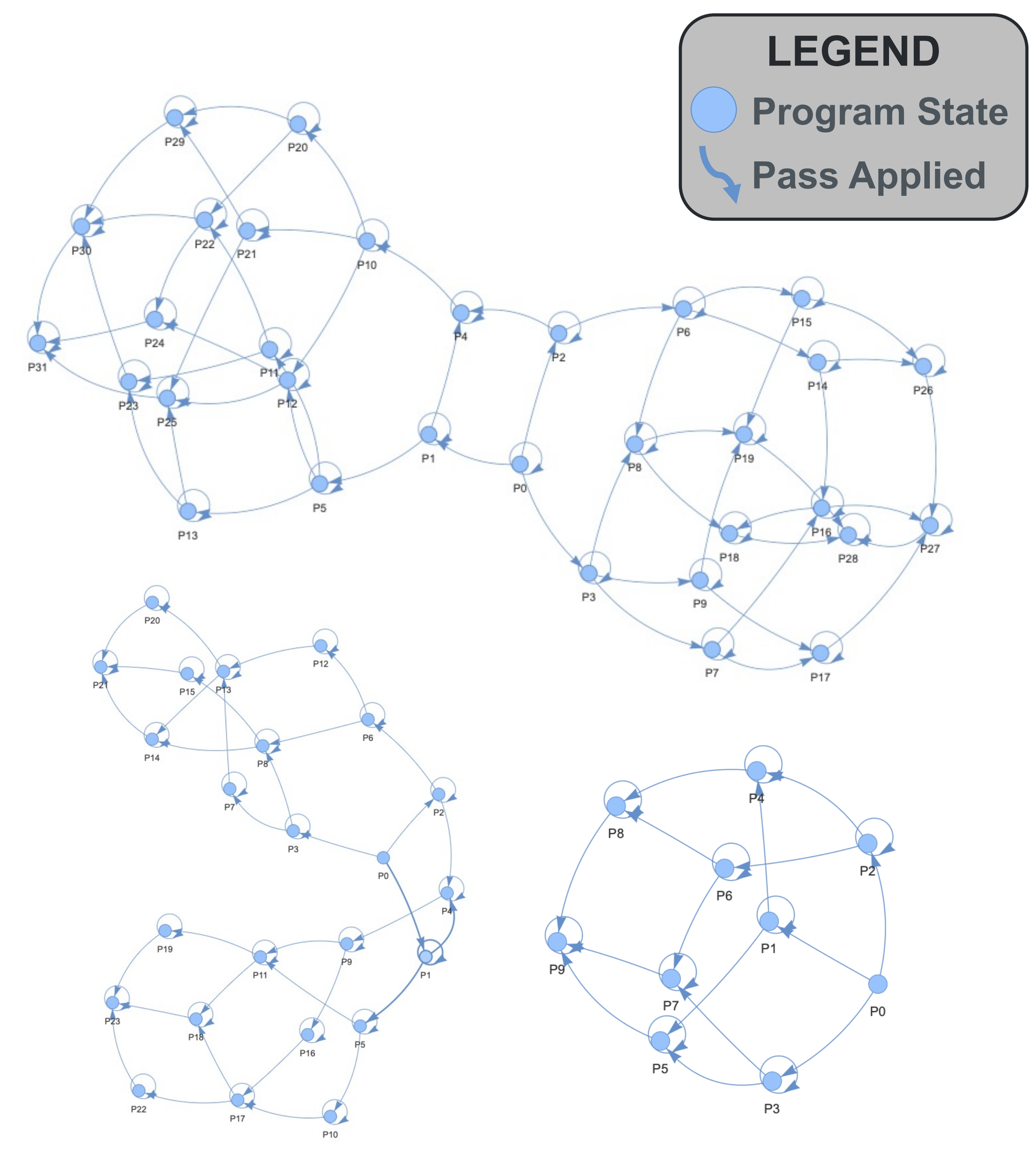

Figure 5: Results obtained from pass microscope tool. Reveals how passes are interconnected and their influence on a programs fate during compilation.

- **Developed a novel tool** to **visualize pass interactions** to analyze how different LLVM passes affect **code transformations and performance**.
- **Nodes** represent unique program states while **edges** indicate the relationships between different program states.
- Able to generate the transition graph of **any** program with **any** set of LLVM passes.
- Example: referring to the top network of *Fig 5*, an interesting result indicating the fate of the program depends on the **2 passes** in the middle, either trapping you in the **left cluster** or **right cluster** revealing the significance of these 2 passes.

Conclusions

- There are more pass combinations than visible atoms in the universe, hence a lot of information remains undiscovered.
- Developed a tool that visualizes pass interactions in the LLVM compiler infrastructure.
- Can lead to optimization strategies, developing compilers, and understanding pass interactions.
- Significant for industries such as **Apple** that utilize LLVM, emphasizing the benefits of making programs more efficient.

Future Work

- Add a connected components/clustering feature for the pass microscope.
- Improving tool's graph user interface. Some features include:
  - Node sizing representing the level of optimization applied on the program state.
  - Pressing on an edge reveals the pass applied to state.
  - Edge styles representing different groups of passes.
  - Highlighting only selected passes impact on the graph.
- Developing an algorithm to study patterns, identify traits, and key information from the graph.
- Scaling this process across entire benchmark and comparing results.

References

[1] "The LLVM Compiler Infrastructure Project," *Llvm.org*. [Online]. Available: https://llvm.org. [Accessed: 06-Aug-2023].

[2] "PageAngha," *Ufmg.br*. [Online]. Available: http://cuda.dcc.ufmg.br/angha/home. [Accessed: 06-Aug-2023].

Ahmed Elzaria
Undergraduate Software Engineering II

McMaster University, Department of Computing and Software, Faculty of Engineering

Email: elzariaa@mcmaster.ca
LinkedIn: AhmedElzaria

BRIGHTER WORLD