Experiment 8: Study of LLVM

Name: Spandan Mukherjee

Registration Number: 21BCE1132

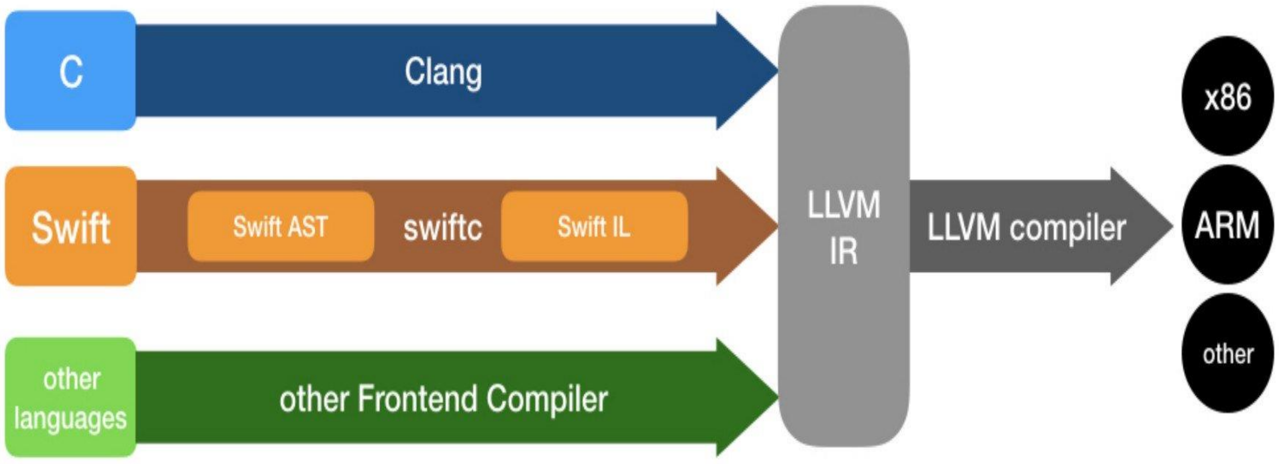
Subject: Compiler Design

LLVM

LLVM can be used to build new computer languages and improve existing ones. It is a compiler and toolkit that automates many tasks involved in language creation such as porting code to multiple platforms and architectures. The LLVM project has grown beyond its initial scope and is a collection of modular and reusable compiler and toolchain technologies.

LLVM is a versatile compiler and toolkit that can be used to build new computer languages and improve existing ones. However, LLVM is not a compiler for any given language; instead, it is a framework to generate object code from any kind of source code. LLVM automates many of the difficult and unpleasant tasks involved in language creation such as porting the outputted code to multiple platforms and architectures.

In addition to its use in language creation, LLVM has also been adopted by HEAVY.AI for optimizing code so that a wide range of analytic workloads can run optimally on GPUs. HEAVY.AI's compilation framework is built on LLVM and utilizes the speed of GPUs. LLVM allows HEAVY.AI to transform query plans into architecture-independent intermediate code (LLVM IR) and then use any of the LLVM architecture-specific "backends" to compile that IR code for the needed target, such as NVIDIA GPUs.



What Does LLVM Stand For?

LLVM stands for low level virtual machine. The LLVM project started in 2000 as a research effort at the University of Illinois. It investigated dynamic programming language compilation techniques in addition to LLVM static analysis. The LLVM project's aim was to offer a conventional SSA-based (static single assignment) compilation method. As of now, LLVM's scope has grown to cover a wide range of academic research, commercial, and open source projects which have nothing to do with virtual machines.

What is LLVM?

LLVM is a compiler and a toolkit for creating compilers, which are programmes that translate instructions into a form that a computer can read and execute.

The LLVM project is a set of modular and reusable compiler and toolchain technologies. LLVM aids in the creation of new computer languages as well as the enhancement of current ones. It automates many of the time-consuming and unpleasant activities associated with language development, such as porting output code to numerous systems and architectures.

How Is LLVM Different From GCC?

Compilers are both LLVM and the GNU Compiler Collection (GCC). The distinction is that GCC supports a variety of programming languages, but LLVM is not a compiler for any specific language. LLVM is a framework for creating object code from any source code.

While both LLVM and GCC support a broad range of languages and libraries, they are licenced and developed in quite different ways. LLVM libraries are more broadly licenced, whereas GCC imposes stricter limitations on their usage.

When it comes to performance differences, GCC has historically been deemed superior. Nonetheless, LLVM is gaining ground.

How A LLVM Compiler Works

On the front end, the LLVM compiler infrastructure employs clang, a compiler for the computer languages C, C++, and CUDA, to convert source code into an intermediate format. The intermediate format is then converted to final machine code using the LLVM clang code generator on the back end.

The compiler goes through five basic stages:

- Lexical Analysis — This function converts programme text into words and tokens (everything apart from words, such as spaces and semicolons).
- Parsing – Converts the words and tokens from the lexical analysis into a logical form.
- Semantic Analyzer – Identifies the programme kinds and logics.
- Optimization – Improves run-time speed by cleaning up the code and addressing memory-related concerns.
- Code Generation – Converts code into an executable binary file.

How Does HEAVY.AI Work With LLVM?

HEAVY. AI has made significant investments in refining its code so that it can handle a wide range of analytic tasks optimally on GPUs. This is why our compilation framework is based on LLVM and takes use of GPU performance. LLVM enables HEAVY.AI to convert query plans into architecture-independent intermediate code (LLVM IR) and then utilise any of the LLVM architecture-specific "backends" to build the IR code for the required target, such as NVIDIA GPUs. The clang LLVM-based C++ compiler creates the corresponding LLVM IR, which OmniSci then integrates with our intentionally created IR.