Homework–2
Compilers, Monsoon-2020, IIIT-H

Meher Shashwat Nigam, 20171062

1. Read the handout on cross compilation posted on Moodle.
Done reading.

2. Read Chapters 3, 5, 6, 9 and 10 of the book An Introduction to GCC by Brian Gough.
Done reading.

3. How do you compile a compiler? Do you see any chicken and egg problem here? Can you mention any other such chicken and egg problems you encountered in computer science and engineering? Check the build process of LLVM and GCC, and report how they are compiled?

Compilers can be compiled using a process called bootstrapping. It is the process of iteratively building upon a software from a simpler version of the same tool. This however would raise the question of how the first compiler would be written in the first place. This can be done in another language to get the first version of a programming language and then we can continue to build upon the programming language using a compiler using the language that we just built. If we go down to the lowest level, we could write a primitive compiler using assembly code.

A good example of this *chicken and egg* problem that we noticed above is seen very often in the case of installers and package managers. One example I see everyday is the case of apt, snap and most importantly pip. Considering the example of pip, pip is the very same software we use to update itself. This is a very good example of bootstrapping where the package manager updates itself. Usually, this happens bby the use of executables that update the software following which the software installation begins.

<u>GCC & LLVM</u>
GCC uses a 3 stage process to build using the bootstrapping technique. It builds the tools required to build the compiler multiple times. It also does a comparison on the compilers in the second and third stage. It then builds runtime libraries from the final stage compiler.

LLVM is compiled using the GCC compiler initially, and later using Clang. A GCC toolchain is installed and configured for C++ standard library. To use Clang as a host later, it requires access to modern C++ standard library, which we do by building libc++ and Clang and installing Clang into the same location as GCC.

4. Run the following program, for a = 10^k; 5 ≤ k ≤ 10, on a Java Hotspot Virtual Machine with and without enabling Hotspot compilation. Report the JDK used and the options to enable/disable hotspot. Report the observed timings in the form of a table.Report the corresponding timings.

```
import java.math.BigInteger;
public class sqrsum {
  public static void main(String args[]) throws NumberFormatException {
    Long a = Long.parseLong(args[0]);
    BigInteger result = BigInteger.valueOf(0);
    for (long i = 1; i <= a; ++i) result = result.add(sqr(i));
    System.out.println("sqrsum of the numbers is: " + result.toString());
  }
  static BigInteger sqr(long b) {
    return BigInteger.valueOf(b * b);
  }}
```

Followed this installation process for latest version : [Oracle Java installation on Ubuntu 20.04 Focal Fossa Linux]

```
>>java --version
java 14.0.2 2020-07-14
Java(TM) SE Runtime Environment (build 14.0.2+12-46)
Java HotSpot(TM) 64-Bit Server VM (build 14.0.2+12-46, mixed mode, sharing)
```

By default hotspot compilation is enabled. To disable it we can use *-Xint* flag.
Copied code into file named *sqrsum.java*
Timing the compilation,execution :
>>time java sqrsum.java <a>
(Changed all int to big integer for avoiding overflow)

a	**Time with hotspot compilation**	**Time without hotspot compilation**
10^5	real 0m1.281s user 0m1.264s sys 0m0.025s	real 0m1.299s user 0m1.238s sys 0m0.069s
10^6	real 0m0.532s user 0m1.364s sys 0m0.069s	real 0m2.104s user 0m2.085s sys 0m0.032s
10^7	real 0m0.738s user 0m1.562s sys 0m0.086s	real 0m12.783s user 0m12.723s sys 0m0.073s

10^8	real 0m3.595s user 0m3.720s sys 0m0.197s	real 1m46.367s user 1m46.361s sys 0m0.120s
10^9	real 0m23.405s user 0m24.788s sys 0m0.420s	real 14m34.900s user 14m34.900s sys 0m0.420s
10^10 (long)	real 5m6.944s user 5m6.943s sys 0m0.420s	Too long to run with biginteger, without hotspot(>1hr)

5. Download the Polybench benchmark suite at the following url: https://sourceforge.net/projects/polybench/files/latest/download. Run the attached Python program with your roll number as command line argument (for example, $python3.8 polyb.py 20183456). You will get three polybench programs as output. Compile and execute those programs using gcc, llvm, icc at the optimization levels -O0, -O1, -O2, -O3, -Os. Report runtimes and executable file size in a tabular Form.

Benchmarks for 20171062 : gramschmidt, nussinov, doitgen

Gramschmidt						
Compilation Levels	**GCC**		**ICC**		**LLVM**	
	Time	**Size**	**Time**	**Size**	**Time**	**Size**
-O0	12.070968	20K	12.841360	28K	12.0347581	20K
-O1	11.468877	20K	12.343501	48K	11.949823	20K
-O2	11.747085	20K	12.435742	44K	12.121028	20K
-O3	11.825120	20K	12.126935	44K	12.368016	20K
-Os	11.906401	20K	11.823078	36K	12.334956	20K

Doitgen						
Compilation Levels	**GCC**		**ICC**		**LLVM**	
	Time	**Size**	**Time**	**Size**	**Time**	**Size**
-O0	2.026628	20K	1.956317	20K	1.737999	20K

-O1	1.292783	20K	0.590106	36K	1.313420	20K
-O2	0.596440	20K	0.152119	112K	0.602004	20K
-O3	0.593486	20K	0.152548	112K	0.609865	20K
-Os	1.276980	20K	0.600928	36K	0.624119	20K

Nussinov						
Compilation Levels	**GCC**		**ICC**		**LLVM**	
	Time	**Size**	**Time**	**Size**	**Time**	**Size**
-O0	16.381103	20K	20.382833	20K	19.0120961	20K
-O1	7.066232	20K	6.391461	36K	7.113267	20K
-O2	6.087788	20K	6.063752	36K	6.231906	20K
-O3	6.034584	20K	5.946269	36K	6.195607	20K
-Os	6.984209	20K	5.988965	36K	6.400747	20K

6. Challenge Problem: Can you use a combination of compiler optimization flags and beat the execution time provided by -O3 for any of the benchmark programs in Polybench? If you are able to find such a scenario, report the same.

For **doitgen** using the compiler **gcc** if we use **-Ofast and all the flags in -O3 optimization level** as optimization flags we get execution time as **0.654**. However if we just use -O3 as the optimization level we get execution time as **0.639.**