CS3020/CS6240: Mini-Programming Assignment #3
LLVM Analyses/Transform Passes and Optimizations
Due Tuesday November 10th, 2015 at 11:59 PM

This study is aimed at giving you more familiarity with the LLVM compiler infrastructure, and introduce you the world of program analyses, passes and optimizations. More specifically, in this assignment, you should do an introductory *study* of the IR, pass framework and optimization of the LLVM compiler.

- Get familiar with *LLVM's Analysis and Transform Passes* at `http://llvm.org/docs/Passes.html`.
- **Analyses passes:** For simple, but *interesting* programs, use some of options and study the transformation of the generated code. Some of the recommended options are CFG, Alias analysis (different varieties like tbaa, Cfl-aa), Dominator tree, Post-dominance Frontier, Call graph (using inlining and other options), SCCs of call graphs, Intervals, Regions, etc.[1] You may not limit yourself to the above list.
- **Transform passes:** Again, for simple, but *interesting* programs, see how the *LLVM's transformation passes* modify your code. The recommended transformations are Dead Code Elimination, Common Subexpression Elimination, Basic-Block Vectorization, Constant Propagation (Simple/Sparse), Loop Invariant Code Motion, Unroll loops, Simplify CFG, Tail Call Elimination, etc. As usual, you need not limit yourself to the above list.
- **Source code of analyses/transformations passes:** For at least a couple of the above analyses and transformations, *browse* through the corresponding source-code and report on your study.
- **Writing new transform passes:** One of the strengths of LLVM infrastructure is that provides an easy way to write new passses. Read how do so from `http://llvm.org/docs/WritingAnLLVMPass.html`. Write a simple pass.
- **Effect of optimizations on performance:** Take some simple, but interesting programs and study the effect of LLVM optimization options (like O1, O2, O3) on your program. Report your findings. Take some programs and run them with these options and do a study of the improvements that you obtain.
- **Study of Auto-vectorization in LLVM:** Do a mini-study of LLVM's Auto-vectorizer `at http://llvm.org/docs/Vectorizers.html`. Take some simple, but interesting programs (preferably from the GCC's Auto-Vectorizer page at `https://gcc.gnu.org/projects/tree-ssa/vectorization.html`) and study the effect of LLVM's vectorizer on your input programs. Report your findings on the improvements that the vectorization can give as well as the variety of programs that it can handle. See the LLVM-IR for vectorized instructions.
- **Loop Optimizations using Polly:** Take some for-loop programs (called *Static Control Programs or SCoPs)* programs from PolyBench benchmarks `http://www.cs.ucla.edu/~pouchet/software/polybench/` and run them through Polly. Some suggestions are seidel, mvt and gemm, 2mm and 3mm. Report on the improvements that Polly can give.

As usual, you can do this assignment along with your team member with at most two members per team.

[1] You may find dot from graphviz `(http://www.graphviz.org/)` to be helpful.

Submission You will be evaluated on a *technical report* on your study. You need not submit any code, but, if you wish, you can add the code listings in a separate appendix. The report should be a PDF, preferably generated from a lyx/latex file.

Evaluation Your report should show a *good* understanding on each of the points asked. The usual rules of plagiarism apply to your report. In particular **do not** directly copy the material from manuals/websites. Your report should professionally refer the website(s) from where you downloaded the code and the manual(s) you referred to. Very good reports will fetch bonus points.