Mini Assignment 2
Introduction to Polly

Name : Vedha Moorthy S
Roll No : CS16BTECH11040
This document is generated by LaTeX.

1 Polly Architecture

In simple terms, Polly is a loop optimizer for LLVM. It is designed to work on the emitted LLVM-IR code. Theoretically, Polly can be run before the 3 phases in the LLVM pipeline.

Pipeline Component	Functionality	Using Polly
Canonicalization	Early loop unrolling, simple Scalar optimizations, simplifying IR code	Polly-optimized IR is still human-understandable and so, optimizer can be fine-tuned to give better results but inlining optimizations will not be done
Inliner	Functions are inlined, multiple passes for simplifying code, simple loop optimizations, loop invariant code motion	Polly generally not used here to avoid messing up with Inliner functionalities
Target Specialization	Target-specific optimizations, complex IR generation to perform high level optimizations, vectorization	Polly runs on canonicalized code which reduces the burden of multiple passes while optimizing the complete code (with headers/packages)

2 Polyhedral Compilation

The underlying idea behind Polly is the use of Mathematical concepts like Integer Polyhedra to represent the program in an abstract Mathematical form. Once the equivalent Mathematical representations are obtained, a varied set of optimizations can be run on them.

These polyhedral representations are advantageous in the sense that they can look at individual array elements and single iterations of loops. Consequently, the efficiency of algortihms used here are dependent on their structures rather than no of elements in the representation.

3 SCoPs (Static Control Parts)

SCoPs are simply the parts of the program that Polly can optimize. They are detected after the program is canonicalized, by passing over the program parts. After detection, they are translated into polyhedral representations, which are used in the later stages of Polly optimization.

SCoPs are basically just 2 forms of control flow structures, the if statements and loops. They are detected by traversing the program tree. SCoP nodes are added to a separate set while the non-SCoP nodes are ignored and its children are traversed. Neighbouring regions in the SCoPs set are then merged if possible to obtain a final set of maximal SCoPs.

4 Dependences

Dependences can be understood as a relation between program statements based on the order of accessing different memory locations. Thus, dependence analysis is where we can find out distribution of memory accesses. This can be used to segregate statements into smaller groups which reference the some particular memory locations and these groups in turn can be parallelized to run on multiple processors, drastically improving the program's performance.

Dependencies can be divided into 2 categories :

4.1 Control

A statement makes a control decision, based on which the following statements may or may not be evaluated.

4.2 Data

A statement modifies a memory location which is then accessed by the following statements.

Loop Dependence is a simple instance of the explained Dependence, where the data dependency takes prominence, as frequently the memory accessed in a loop are closely related spatially or logically.