# Compilers II: CS3423

**Mini Assignment II**
**Introduction to Polly**

**Assignment Report**

**Sagar Jain - CS17BTECH11034**

November 22, 2019

# Contents

# Architecture and Intuitive Understanding of Polly

## What is Polly to LLVM?

In the simplest possible words, **Polly is a loop optimizer for LLVM**. Polly fits into the LLVM pass pipeline. LLVM basically has three phases in the pass pipeline:

1. Canonicalization
2. Inliner cycle
   (a) Scalar Simplification
   (b) Simple Loop Optimizations
   (c) Inliner
3. Target Specialization Polly can conceptually be run at three different positions in the pass pipeline, but mostly it is run as (1) an early optimizer before the standard LLVM pass pipeline or (2) as a later optimizer as part of the target specialization sequence.

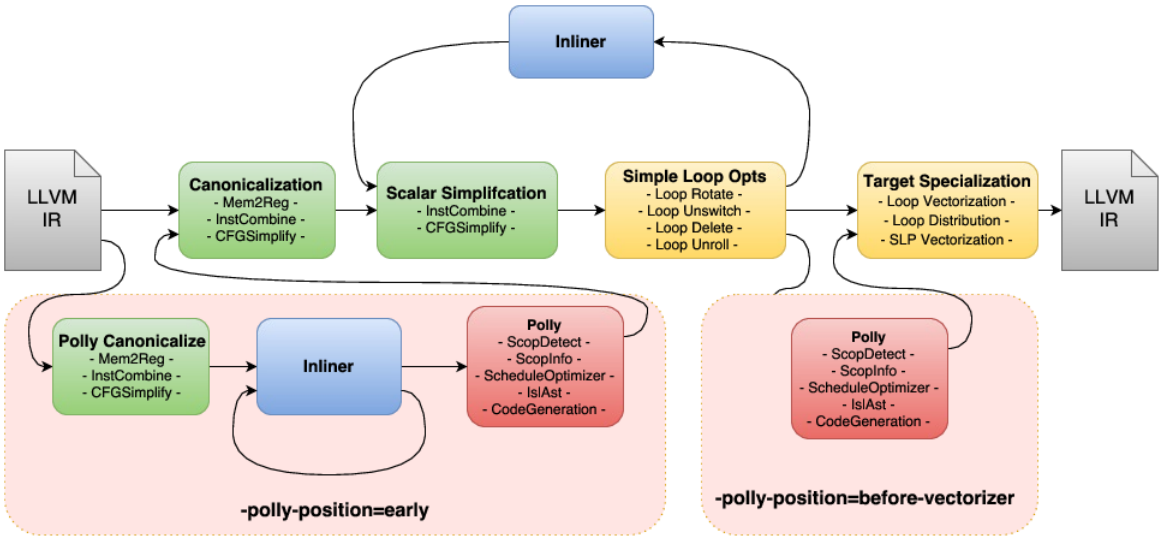


**Figure 1:** Image from here

## Understandig of the Polyhedral Model

The following is a my idea of how a polyhedral model works. First we need to understant that a loop, during execution can be represented in an alternate way. Each instruction loop should be considered as a different instance in every iteration. This way we have a set of **instruction instances**. Instruction not in loops would, intutively, have only single instances in normal circumstances.
Now that we have a set of instruction instances, we need to execute them. The ordering of this execution is where polly comes in. **Suppose we make a graph with all the instruction instances as nodes and an incoming edge into nodes from all the nodes which need to be executed before it**. We would now have something like this:

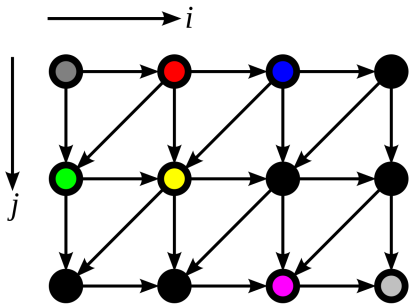


**Figure 2:** Image from here

All we know here is that an instruction must not be executed before all the instruction it depends on are executed. Also this representation ends up looking like a polyhedral in most cases (rectangle in this case) and that's where the name comes from. Polly finds the best possible **"skewing"** of this polyhedral such that the most number of instructions can be executed parallely, minimising time. This seems to be a complex ILP problem. The result for the current case would something like the following.

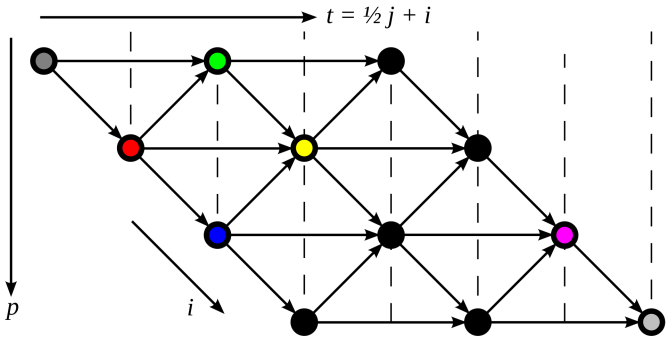


**Figure 3:** Image from here

# Results

The following are the results for the file matmul.c from here.

| Compile Options | Time Taken |
|---|---|
| clang -O3 | 27.95s |
| gcc | 84.90s |
| gcc -O3 | 6.06s |
| clang -O3 -mllvm -polly | 0.39s |

These results seem almost too good to believe, this shows the great utility of polyhedral compilation. We get around 21769% improvement over normal gcc compilation and 1550% over gcc with O3, this is phenomenal by any standards.

# Analysis of Transformed Code & Trying Other Polly Options

I tried the following optimisations with polly:

1. `polly-vectorizer=stripmine`
2. `polly-parallel-force`
3. `polly-tiling`

The following are the most important characteristics of the polly-generated IR I noticed:

1. Firstly, the generated IR is much bigger interms of code-size than the normal IR (without polly) for the same file.
2. In every loop there are a lot of constant variables declared in the header, these are begin used as offsets **getelementptr**.
3. SCEV is being used extensively in the IR. This makes sense since SCEV is important to get bounds and steps and these are used by the polyhedral model to compute how to manipulate i.e. skew the execution process.
4. Another important thing I noticed is that most of the scev computation happens in the header of loops, this happens because once the loops

are skewed their initial conditions, steps and bounds are altered and need the loop conditions need to be computed at every iteration.

# SCoPs and Dependencies

SCoP stands for Static Control Parts. Static control essentially means control flow within the loop only depends on the loop iterators only. An important part of the polyhedral model is to determine if the schedule computed by it is **legal**. This, quite obviously depends on the **dependecies** between instructions and instruction instances. In the `matmul.c` file the main function has the following SCoPs:

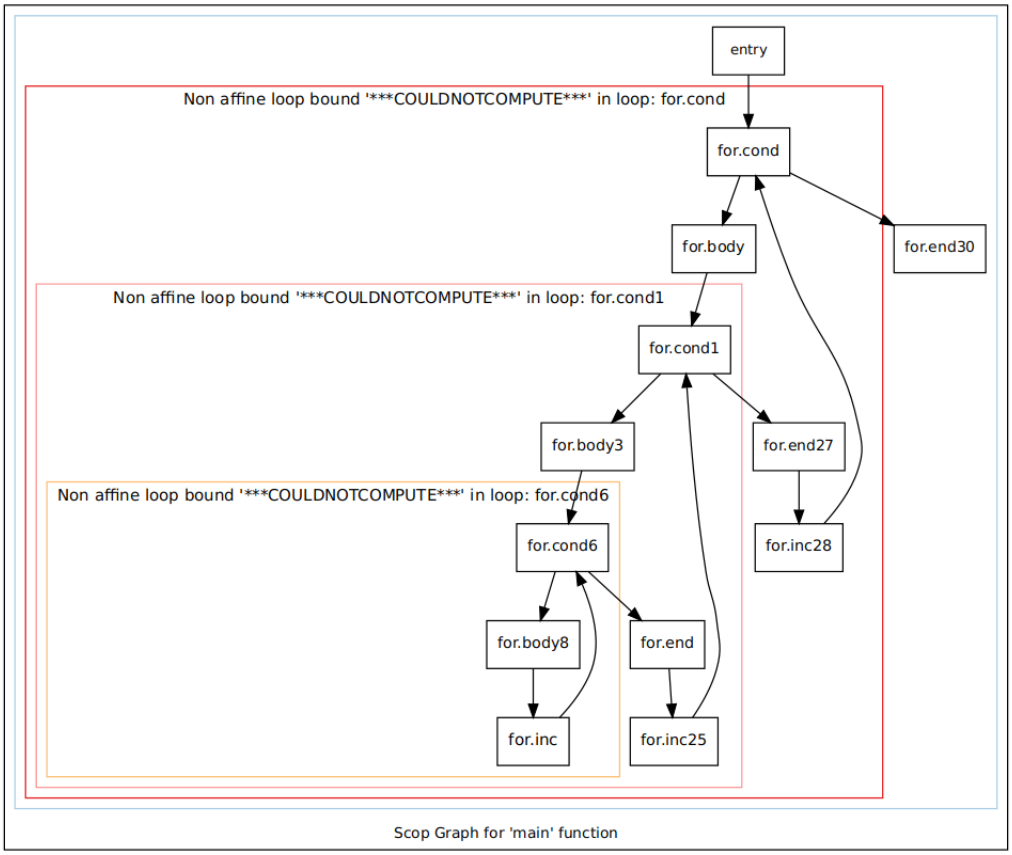


**Figure 4**

The SCoP and static reference assumptions assumption is very important to get an exact set of instances in dependence. One can even get the dependences for SCoPs using:

```
opt -polly-dependences -analyze matmul.preopt.ll -polly-process-unprofitable
```

# References


- https://wiki.aalto.fi/display/t1065450/Polyhedral+optimizations+with+LLVM
- http://polly.llvm.org/docs/HowToManuallyUseTheIndividualPiecesOfPolly.html
- http://polly.llvm.org/docs/UsingPollyWithClang.html
- http://www.es.ele.tue.nl/ rjordans/5LIM0/10-polyhedral-model.pdf