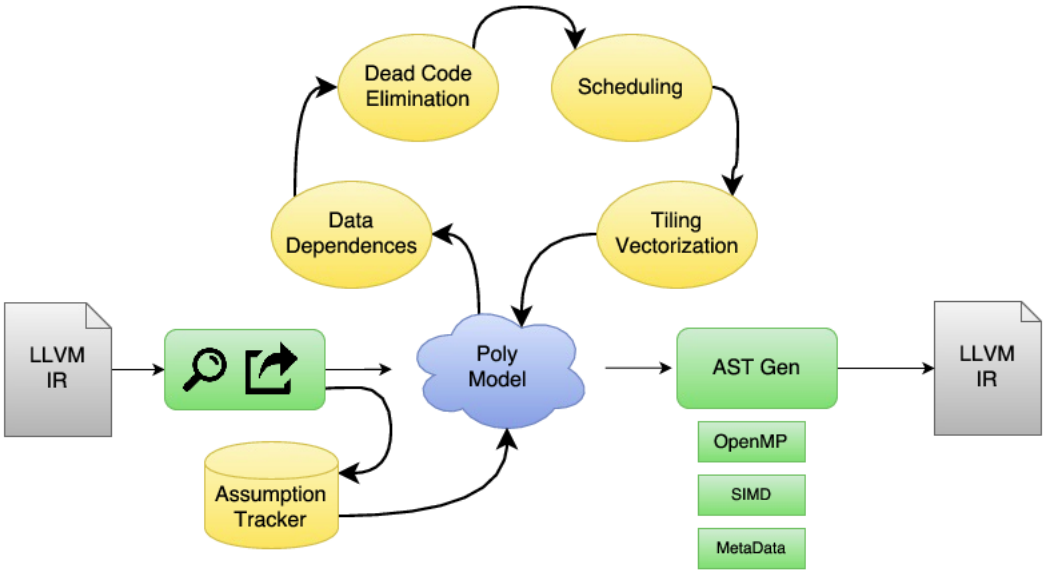
Polly Assignment

Yash Khasbage
CS17BTECH11044

21 Nov 2019

1. Your understanding of the architecture of polly and its optimization capabilities

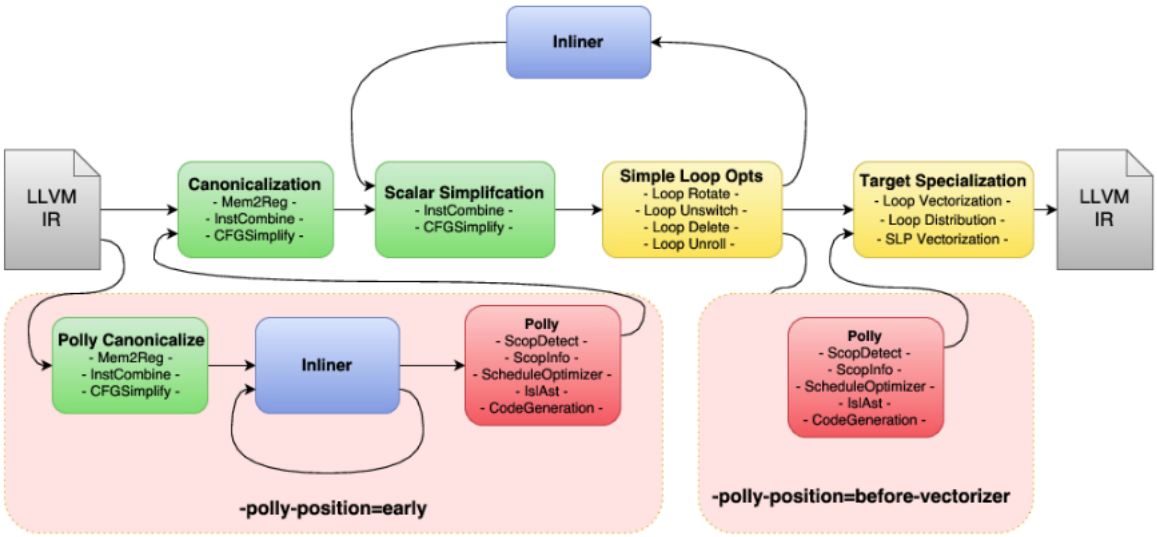
Figure 1: This figure is borrowed from the official polly webpage. It shows the optimizations performed by polly.source: http://polly.llvm.org/docs/Architecture.html



- polly specializes in optimizing loops
- it detects and extracts loop kernels. The kernels are then converted to mathematical entities that are further transformed to optimize the code. The llvm-ir for the loop is regenerated and inserted into the llvm-ir module.
- Optimization happens in three phases: Canonicalization, inliner cycle and target-specialization

- Canonicalization consists of has the goal of removing and simplifying the given IR as much as possible focusing mostly on scalar optimizations. e.g., early loop unrolling, -mem2reg
- The major goal of inliner phase is canonicalization without losing semantic information that complicates later analysis. It inlines functions, runs canonicalization passes to exploit newly exposed simplification opportunities, and then tries to inline the further simplified functions. Some simple loop optimizations are executed as part of the inliner cycle. This stage can be weel understood from Figure 2.
- Target specialization uses machine-specific information to further optimize. The increase in code complexity is ignored. e.g., loop-unrolling, vectorization etc.
- Polly can be run at 3 places: early, within inliner, and before vectorizer
- Running Polly early before the standard pass pipeline has the benefit that the LLVM-IR processed by Polly is still very close to the original input code. However, this compromises with inliner optimizations.
- Running Polly right before the vectorizer has the benefit that the full inlining cycle has been run and as a result even heavily templated C++ code could theoretically benefit from Polly.

Figure 2: The above diagram is taken form official polly website. It shows the 3 stages in where polly can be used. soucre: http://polly.llvm.org/docs/Architecture.html



2. The amount of improvement in running time that you got on any code when optimized using polly with clang

Using user time from `time` unix command.

Experimenting on matrix multiplication as given in
`https://github.com/llvm-mirror/polly/blob/master/docs/experiments/matmul/matmul.c`

Matrix size	gcc (sec.)	clang (sec.)	clang -O3 (sec.)
100	0.006	0.008	0.003
500	0.543	0.449	0.040
1000	5.560	4.563	0.102
2000	67.39	47.642	1.091
3000	241.393	207.843	2.781

It can clearly be observed that the run time scales as the cub(approx.) of matrix size.

Experimenting with one layer MLP Code borrowed form
`https://github.com/codeplea/genann`

approx # of parameters	gcc (sec)	clang (sec)	clang -O3 (sec)
400K	17.794	16.867	3.637
1M	72.632	82.547	19.907

3. Your intuitive understanding of SCoPs and dependencies

SCoP stands for Static Control Parts.

SCoP is the section of source code that only contains two control flow constructs: if-loop and for-loop. In the for-loop, a single integer has to be the induction variable, and that too has to be bounded. The bound of this induction variable cannot be changed during the loop execution. The if-predicate has to be simple. Ideally, it should be a comparison of affine expressions. Other conditions are also imposed on the predicate, for example, the expression should have no side effect.

Figures 3 and 4 show dependency graph of a loop snippet. String dependencies between the variables restrict parallelism. But the change in computation order of these variables allows one to evaluate certain quantities parallelly.

Here, the variables were not changed but only the sequence of computation was carefully managed so that multiple computations are possible at the same time, without changing code semantics. In figure 4, x-axis is time.

Figure 3: before compilation. source: https://en.wikipedia.org/wiki/Polytope_model

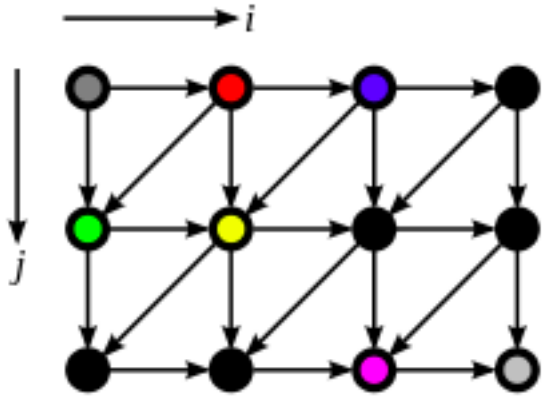
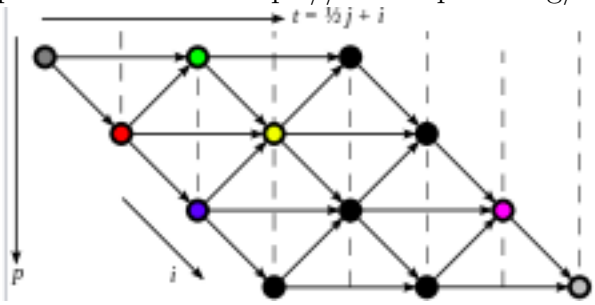


Figure 4: after compilation. source: https://en.wikipedia.org/wiki/Polytope_model



4. Your analysis of a transformed code using polly. Here you can explain what kind of optimization happened and how it improves the performance

Analysis of preoptimization IR

The IR seems to have several repetitive patterns. These patterns include 1) getelementptr for global arrays and 2) loading induction variables repetitively. It was also observed that the unconditional jumps were to undefined labels, which is unusual from normal IR.

The separation of IR into several sections show how some loop operation is handled. The sections can be marked by the spaces in code. Some observations are as follows. Allocating and comparing induction variables is done in separate sections.

After trying some other things like tiling, the code changes considerably. The newer code almost unexplanable. However, tiling can be seen explicitly.