Polly
First successful optimizations - How to proceed?

Tobias Grosser, Raghesh A

November 18, 2011

Me - Tobias Grosser

- Doctoral Student at INRIA/ENS, Paris
- Interests: Automatic parallelization, data-locality optimizations
- Compiler Experience (5 years)
  - **GCC/Graphite**
  - **Polly**
  - **LLVM**
  - **clang_complete**

Direct Contributors / Funding

- Universities
  - ENS/INRIA Paris (Albert Cohen)
  - Ohio State University (P. Sadayappan)
  - University of Passau (Christian Lengauer)
- Funding
  - Google Europe Fellowship in Efficient Computing
    Mentor: Aart Bik
  - 2 x Google Summer of Code Students
  - NSF Awards 0811781 and 0926688 (partially)
  - Qualcomm travel support

The Problem

Life

is complicated!

The Problem

Life
of a programmer
is complicated!

Life is complicated - Why?

We want:

- Fast and power-efficient code

We have:

- SIMD, Caches, Multi-Core, Accelerators

But:

- Optimized code is needed
- Optimization is complex and not performance portable
- Architectures are too diverse to optimize ahead of time

Get Polly

- Install Polly
  `http://polly.grosser.es/get_started.html`

- Load Polly automatically

```
alias clang   clang -Xclang -load -Xclang LLVMPolly.so
alias opt     opt -load LLVMPolly.so
```

- Default behaviour preserved
- clang/opt now provide options to enable Polly

Optimize a program with Polly

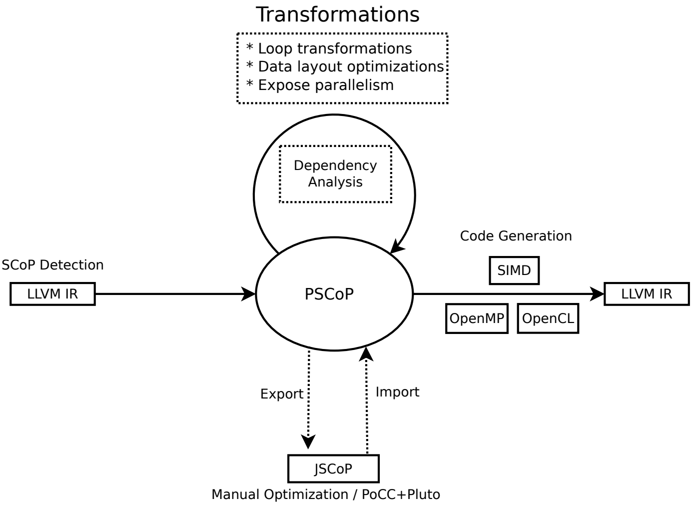
gemm.c [1024 x 1024 (static size), double]

```
for (int i = 0; i < N; i++)
  for (int j = 0; j < M; j++) {
    C[i][j] = 0;
    for (int k = 0; k < K; k++)
      C[i][j] += A[i][k] + B[k][j];
  }
```

```
$ clang -O3 gemm.c -o gemm.clang
$ time ./gemm.clang
real 0m15.336s

$ clang -O3 -mllvm -o gemm.polly -mllvm -polly
$ time ./gemm.polly
real 0m2.144s
```

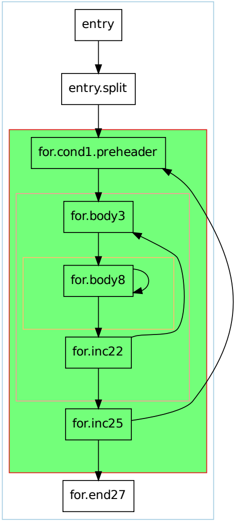
The Architecture



Can Polly analyze our code?

```
$ clang -O3 gemm.c \
      -mllvm -polly-show-only \
      -mllvm -polly-detect-only=gemm
```

- Highlight the detected Scops
- Only check in function 'gemm'



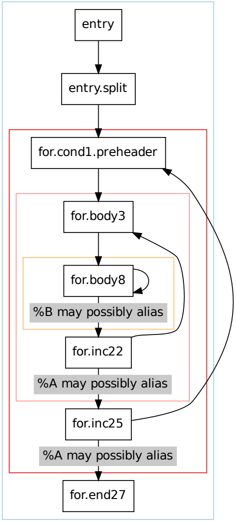
Scop Graph for 'gemm' function

Some code can not be analyzed

```
$ clang -O3 gemm.c \
        -mllvm -polly-show-only \
        -mllvm -polly-detect-only=gemm
```

gemm (possible aliasing)

```c
void gemm(double A[N][K],
          double B[K][M],
          double C[N][M]) {
  for (int i = 0; i < N; i++)
    for (int j = 0; j < M; j++) {
      C[i][j] = 0;
      for (int k = 0; k < K; k++)
        C[i][j] += A[i][k] + B[k][j];
    }
}
```



Scop Graph for 'gemm' function

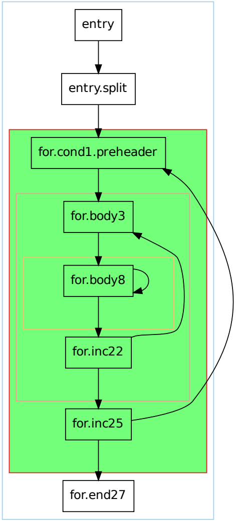
How to fix it?

Add 'restrict'

```
void gemm(double A[restrict N][K],
          double B[restrict K][M],
          double C[restrict N][M]);
```

Other options:

- Inlining
- Improved alias analysis
- Run time checks



Scop Graph for 'gemm' function

Extract polyhedral representation

gemm

```
for (int i = 0; i < 512; i++)
  for (int j = 0; j < 512; j++) {
    C[i][j] = 0;                    // Stmt1
    for (int k = 0; k < 512; k++)
      C[i][j] += A[i][k] + B[k][j]; // Stmt2
  }
```

```
$ clang -O3 gemm.c \
        -mllvm -polly-run-export-jscop \
        -mllvm -polly-detect-only=gemm
Writing JScop 'for.cond1.preheader => for.end27' in function 'gemm' to
'./gemm___%for.cond1.preheader---%for.end27.jscop'.
```

$$\begin{aligned}
Domain &= \{Stmt_1[i,j] : 0 <= i,j < 512; & Stmt_2[i,j,k] : 0 <= i,j,k < 512\} \\
Schedule &= \{Stmt_1[i,j] \to [i,j,0]; & Stmt_2[i,j,k] \to [i,j,1,k]\} \\
Writes &= \{Stmt_1[i,j] \to C[i,j]; & Stmt_2[i,j,k] \to C[i,j]\} \\
Reads &= \{Stmt_2[i,j,k] \to A[i,k]; & Stmt_2[i,j,k] \to B[k,j]\}
\end{aligned}$$

Applying transformations

- $\mathcal{D} = \{Stmt[i,j] : 0 <= i < 32 \wedge 0 <= j < 1000\}$
- $\mathcal{S} = \{Stmt[i,j] \to [i,j]\}$

- $\mathcal{S}' = \mathcal{S}$

```
for (i = 0; i < 32; i++)
  for (j = 0; j < 1000; j++)
    A[j][i] += 1;
```

Applying transformations

- $\mathcal{D} = \{Stmt[i,j] : 0 <= i < 32 \wedge 0 <= j < 1000\}$
- $\mathcal{S} = \{Stmt[i,j] \rightarrow [i,j]\}$
- $\mathcal{T}_{Interchange} = \{[i,j] \rightarrow [j,i]\}$

- $\mathcal{S}' = \mathcal{S} \circ \mathcal{T}_{Interchange}$

```
for (j = 0; j < 1000; j++)
  for (i = 0; i < 32; i++)
    A[j][i] += 1;
```

Applying transformations

- $\mathcal{D} = \{Stmt[i,j] : 0 <= i < 32 \wedge 0 <= j < 1000\}$
- $\mathcal{S} = \{Stmt[i,j] \rightarrow [i,j]\}$
- $\mathcal{T}_{Interchange} = \{[i,j] \rightarrow [j,i]\}$
- $\mathcal{T}_{StripMine} = \{[i,j] \rightarrow [i,jj,j] : jj \mod 4 = 0 \wedge jj <= j < jj + 4\}$
- $\mathcal{S}' = \mathcal{S} \circ \mathcal{T}_{Interchange} \circ \mathcal{T}_{StripMine}$

```
for (j = 0; j < 1000; j++)
  for (ii = 0; ii < 32; ii+=4)
    for (i = ii; i < ii+4; i++)
      A[j][i] += 1;
```

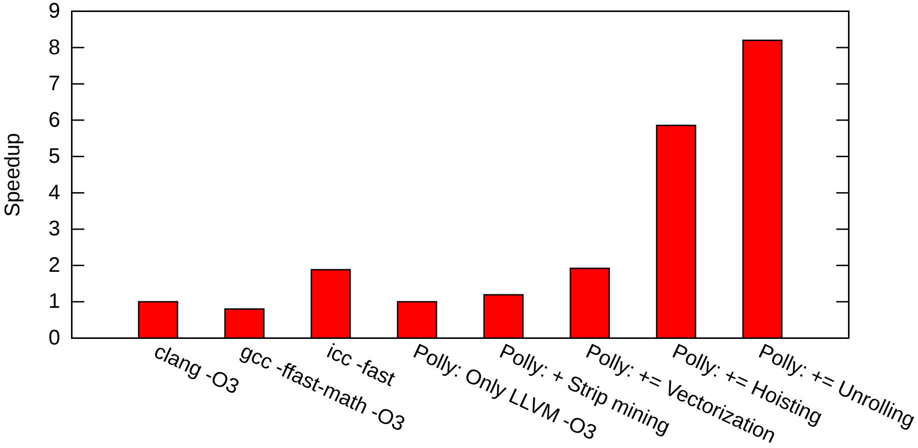
Polly takes advantage of available parallelism

It creates automatically:

- OpenMP calls
  for loops that are not surrounded by any other parallel loops
- SIMD instructions
  for innermost loops with a constant number of iterations

$\rightarrow$ Optimizing code becomes the problem of finding the right schedule.

Optimizing of Matrix Multiply



32x32 double, Transposed matric Multiply, C[i][j] += A[k][i] * B[j][k];

Intel® Core® i5 @ 2.40GH

Automatic optimization with the Pluto algorithm

Polly provides two automatic optimizers

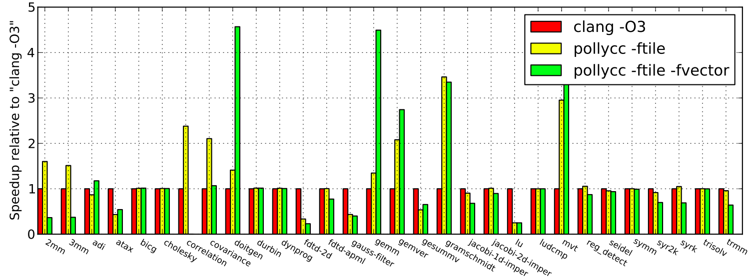
PoCC

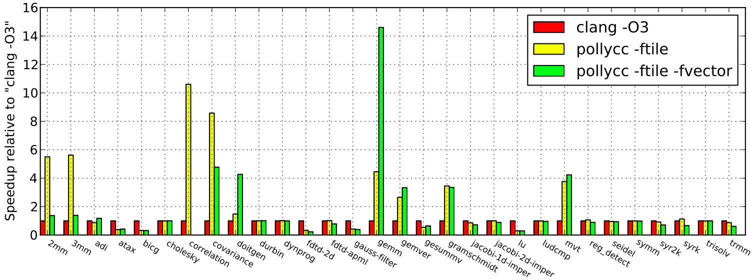
- -polly-optimizer=pocc
- Original implementation
- We call the pocc binary
- More mature
- Integrated with a large set of research tools

ISL

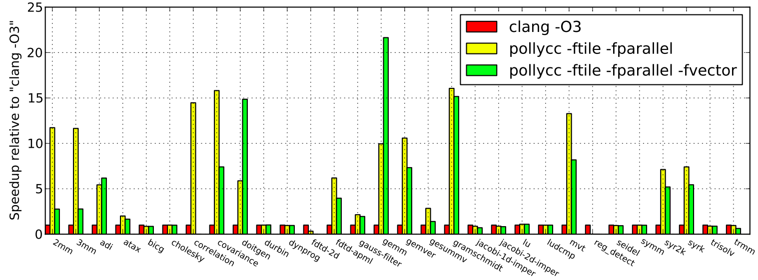
- -polly-optimizer=isl (default)
- Reimplementation
- ISL is already linked into Polly, no additional library needed
- Still untuned heuristics
- Will be used for production.

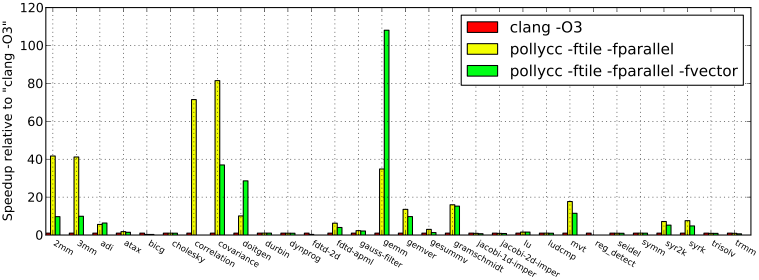
Polly on Polybench - Sequential execution times



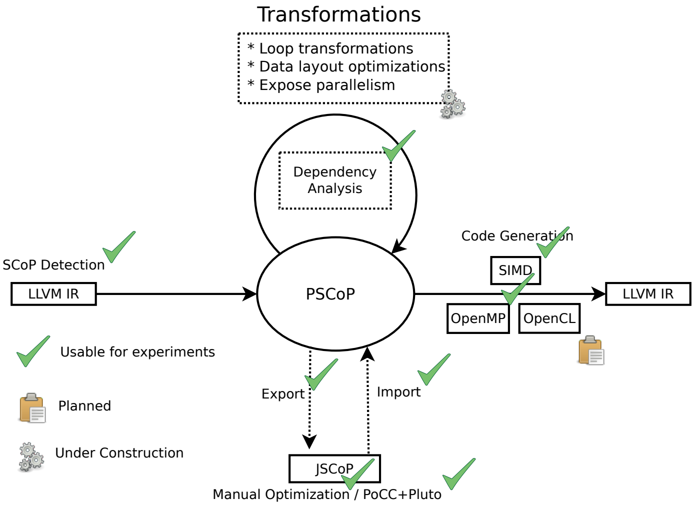


Polly on Polybench - Parallel execution times





Current Status



How to proceed? Where can we copy?

- Short Vector Instructions
  $\rightarrow$ Vectorizing compiler ✓
- Data Locality
  $\rightarrow$ Optimizing compilers ~, Pluto ✓
- Thread Level Parallelism
  $\rightarrow$ Optimizing compilers ~, Pluto ✓
- Vector Accelerators
  $\rightarrow$ Par4All ~, C-to-CUDA ~, ppcg ~

The overall problem: ✗

Polly

Idea: Integrated vectorization

- Target the overall problem
- Re-use existing concepts and libraries

Next Steps

My agenda:

- Data-locality optimizations for larger programs (production quality)
- Expose SIMDization opportunities with the core optimizers
- Offload computations to vector accelerators

Your ideas?

- Use Polly to drive instruction scheduling for VLIW architectures
- . . .

Polly

- Language Independent
- Optimizations for Data-Locality & Parallelism
- SIMD & OpenMP code generation support
- Planned: OpenCL Generation

`http://polly.grosser.es`

Multi dimensional arrays

```
#define N;
void foo(int n, float A[][N], float **B, C[][n]) {
   A[5][5] = 1;
   B + 5 * n + 5 = 1;
   C[5][5] = 1;
}
```

- **A - Constant Size** → already linear
- **B - Self-made made multi dimensional arrays**
  - Guess & Prove
  - Guess & Runtime Check
- **C - C99 Variable Length Arrays / Fortran Arrays**
  - Guess & Prove
  - Guess & Runtime Check
  - Pass information from Clang / GFORTRAN