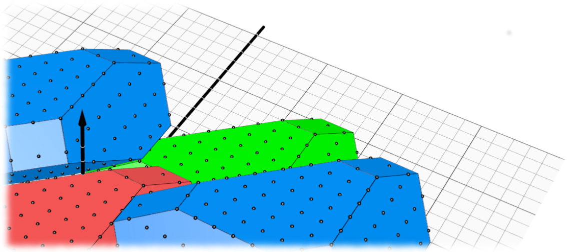
Analyzing and Optimizing your Loops with Polly

Tobias Grosser, Johannes Doerfert, Zino Benaissa

ETH Zurich — Saarland University — Quic Inc.



EuroLLVM 2016, 17. March, Barcelona

State of Variables

Program

```
for (i = 0; i <= n; i++)
  for (j = 0; j <= i; j++)
    S(i,j);
```

Statement Instances Executed

State of Variables

n = 4, i = 0, j = 0

Program

```
for (i = 0; i <= n; i++)
  for (j = 0; j <= i; j++)
    S(i,j);
```

Statement Instances Executed

S(0,0)

Program

```
for (i = 0; i <= n; i++)
  for (j = 0; j <= i; j++)
    S(i,j);
```

State of Variables

$n = 4$, $i = 1$, $j = 0$

Statement Instances Executed

S(1,0),
S(0,0)

Program

```
for (i = 0; i <= n; i++)
  for (j = 0; j <= i; j++)
    S(i,j);
```

State of Variables

n = 4, i = 1, j = 1

Statement Instances Executed

S(1,0), S(1,1)
S(0,0)

Program

```
for (i = 0; i <= n; i++)
  for (j = 0; j <= i; j++)
    S(i,j);
```

State of Variables

n = 4, i = 2, j = 0

Statement Instances Executed

S(2,0),
S(1,0), S(1,1)
S(0,0)

Program

```
for (i = 0; i <= n; i++)
  for (j = 0; j <= i; j++)
    S(i,j);
```

State of Variables

$n = 4$, $i = 2$, $j = 1$

Statement Instances Executed

S(2,0), S(2,1),
S(1,0), S(1,1)
S(0,0)

Program

```
for (i = 0; i <= n; i++)
  for (j = 0; j <= i; j++)
    S(i,j);
```

State of Variables

n = 4, i = 2, j = 2

Statement Instances Executed

S(2,0), S(2,1), S(2,2)
S(1,0), S(1,1)
S(0,0)

Program

```
for (i = 0; i <= n; i++)
  for (j = 0; j <= i; j++)
    S(i,j);
```

State of Variables

n = 4, i = 3, j = 0

Statement Instances Executed

S(3,0),
S(2,0), S(2,1), S(2,2)
S(1,0), S(1,1)
S(0,0)

Program

```
for (i = 0; i <= n; i++)
  for (j = 0; j <= i; j++)
    S(i,j);
```

State of Variables

n = 4, i = 3, j = 1

Statement Instances Executed

S(3,0), S(3,1),
S(2,0), S(2,1), S(2,2)
S(1,0), S(1,1)
S(0,0)

Program

```
for (i = 0; i <= n; i++)
  for (j = 0; j <= i; j++)
    S(i,j);
```

State of Variables

n = 4, i = 3, j = 2

Statement Instances Executed

S(3,0), S(3,1), S(3,2),
S(2,0), S(2,1), S(2,2)
S(1,0), S(1,1)
S(0,0)

Program

```
for (i = 0; i <= n; i++)
  for (j = 0; j <= i; j++)
    S(i,j);
```

State of Variables

n = 4, i = 3, j = 3

Statement Instances Executed

S(3,0), S(3,1), S(3,2), S(3,3)
S(2,0), S(2,1), S(2,2)
S(1,0), S(1,1)
S(0,0)

Program

```
for (i = 0; i <= n; i++)
  for (j = 0; j <= i; j++)
    S(i,j);
```

State of Variables

n = 4, i = 4, j = 0

Statement Instances Executed

S(4,0),
S(3,0), S(3,1), S(3,2), S(3,3)
S(2,0), S(2,1), S(2,2)
S(1,0), S(1,1)
S(0,0)

Program

```
for (i = 0; i <= n; i++)
  for (j = 0; j <= i; j++)
    S(i,j);
```

State of Variables

n = 4, i = 4, j = 1

Statement Instances Executed

S(4,0), S(4,1),
S(3,0), S(3,1), S(3,2), S(3,3)
S(2,0), S(2,1), S(2,2)
S(1,0), S(1,1)
S(0,0)

Program

```
for (i = 0; i <= n; i++)
  for (j = 0; j <= i; j++)
    S(i,j);
```

State of Variables

n = 4, i = 4, j = 2

Statement Instances Executed

S(4,0), S(4,1), S(4,2),
S(3,0), S(3,1), S(3,2), S(3,3)
S(2,0), S(2,1), S(2,2)
S(1,0), S(1,1)
S(0,0)

Program

```
for (i = 0; i <= n; i++)
  for (j = 0; j <= i; j++)
    S(i,j);
```

State of Variables

n = 4, i = 4, j = 3

Statement Instances Executed

S(4,0), S(4,1), S(4,2), S(4,3),
S(3,0), S(3,1), S(3,2), S(3,3)
S(2,0), S(2,1), S(2,2)
S(1,0), S(1,1)
S(0,0)

Program

```
for (i = 0; i <= n; i++)
  for (j = 0; j <= i; j++)
    S(i,j);
```

State of Variables

n = 4, i = 4, j = 4

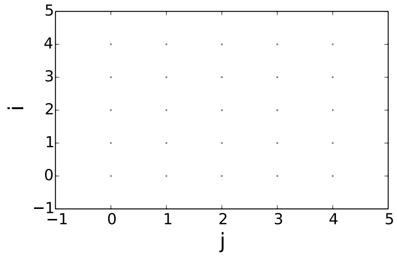
Statement Instances Executed

S(4,0), S(4,1), S(4,2), S(4,3), S(4,4)
S(3,0), S(3,1), S(3,2), S(3,3)
S(2,0), S(2,1), S(2,2)
S(1,0), S(1,1)
S(0,0)

Program

```
for (i = 0; i <= n; i++)
  for (j = 0; j <= i; j++)
    S(i,j);
```

Iteration space



State of Variables

n = 4, i = 4, j = 4

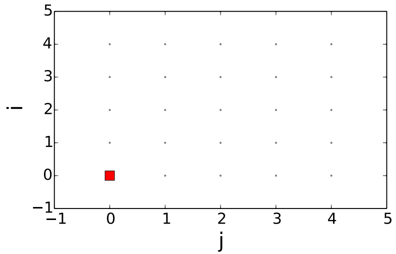
Statement Instances Executed

S(4,0), S(4,1), S(4,2), S(4,3), S(4,4)
S(3,0), S(3,1), S(3,2), S(3,3)
S(2,0), S(2,1), S(2,2)
S(1,0), S(1,1)
S(0,0)

Program

```
for (i = 0; i <= n; i++)
  for (j = 0; j <= i; j++)
    S(i,j);
```

Iteration space



State of Variables

n = 4, i = 0, j = 0

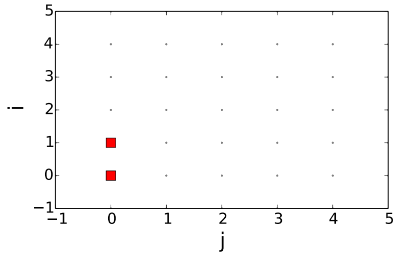
Statement Instances Executed

S(4,0), S(4,1), S(4,2), S(4,3), S(4,4)
S(3,0), S(3,1), S(3,2), S(3,3)
S(2,0), S(2,1), S(2,2)
S(1,0), S(1,1)
S(0,0)

Program

```
for (i = 0; i <= n; i++)
  for (j = 0; j <= i; j++)
    S(i,j);
```

Iteration space



State of Variables

n = 4, i = 1, j = 0

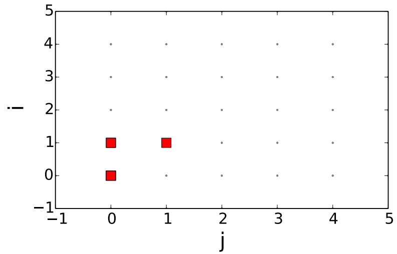
Statement Instances Executed

S(4,0), S(4,1), S(4,2), S(4,3), S(4,4)
S(3,0), S(3,1), S(3,2), S(3,3)
S(2,0), S(2,1), S(2,2)
S(1,0), S(1,1)
S(0,0)

Program

```
for (i = 0; i <= n; i++)
  for (j = 0; j <= i; j++)
    S(i,j);
```

Iteration space



State of Variables

n = 4, i = 1, j = 1

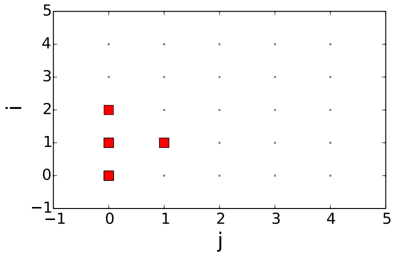
Statement Instances Executed

S(4,0), S(4,1), S(4,2), S(4,3), S(4,4)
S(3,0), S(3,1), S(3,2), S(3,3)
S(2,0), S(2,1), S(2,2)
S(1,0), S(1,1)
S(0,0)

Program

```
for (i = 0; i <= n; i++)
  for (j = 0; j <= i; j++)
    S(i,j);
```

Iteration space



State of Variables

n = 4, i = 2, j = 0

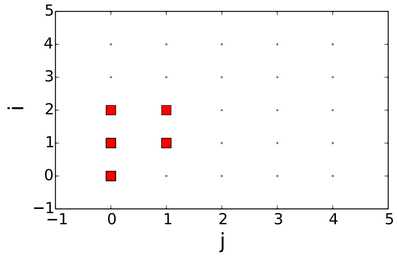
Statement Instances Executed

S(4,0), S(4,1), S(4,2), S(4,3), S(4,4)
S(3,0), S(3,1), S(3,2), S(3,3)
S(2,0), S(2,1), S(2,2)
S(1,0), S(1,1)
S(0,0)

Program

```
for (i = 0; i <= n; i++)
  for (j = 0; j <= i; j++)
    S(i,j);
```

Iteration space



State of Variables

n = 4, i = 2, j = 1

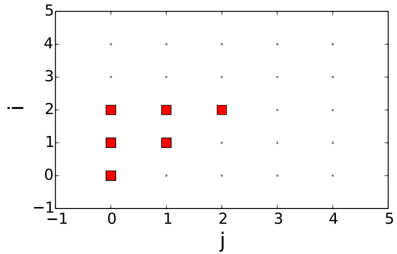
Statement Instances Executed

S(4,0), S(4,1), S(4,2), S(4,3), S(4,4)
S(3,0), S(3,1), S(3,2), S(3,3)
S(2,0), S(2,1), S(2,2)
S(1,0), S(1,1)
S(0,0)

Program

```
for (i = 0; i <= n; i++)
  for (j = 0; j <= i; j++)
    S(i,j);
```

Iteration space



State of Variables

n = 4, i = 2, j = 2

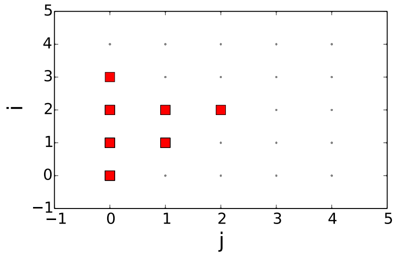
Statement Instances Executed

S(4,0), S(4,1), S(4,2), S(4,3), S(4,4)
S(3,0), S(3,1), S(3,2), S(3,3)
S(2,0), S(2,1), S(2,2)
S(1,0), S(1,1)
S(0,0)

Program

```
for (i = 0; i <= n; i++)
  for (j = 0; j <= i; j++)
    S(i,j);
```

Iteration space



State of Variables

n = 4, i = 3, j = 0

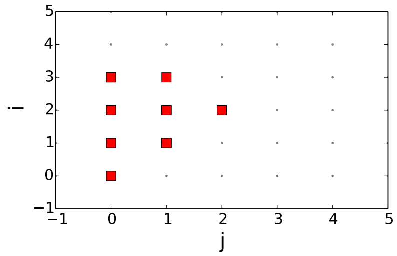
Statement Instances Executed

S(4,0), S(4,1), S(4,2), S(4,3), S(4,4)
S(3,0), S(3,1), S(3,2), S(3,3)
S(2,0), S(2,1), S(2,2)
S(1,0), S(1,1)
S(0,0)

Program

```
for (i = 0; i <= n; i++)
  for (j = 0; j <= i; j++)
    S(i,j);
```

Iteration space



State of Variables

n = 4, i = 3, j = 1

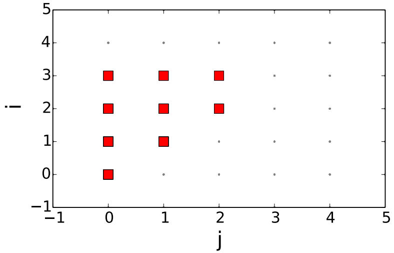
Statement Instances Executed

S(4,0), S(4,1), S(4,2), S(4,3), S(4,4)
S(3,0), S(3,1), S(3,2), S(3,3)
S(2,0), S(2,1), S(2,2)
S(1,0), S(1,1)
S(0,0)

Program

```
for (i = 0; i <= n; i++)
  for (j = 0; j <= i; j++)
    S(i,j);
```

Iteration space



State of Variables

n = 4, i = 3, j = 2

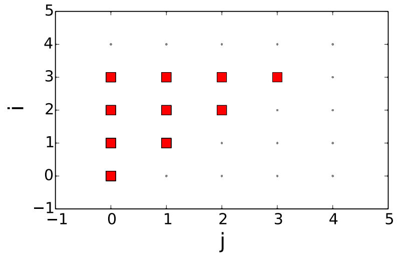
Statement Instances Executed

S(4,0), S(4,1), S(4,2), S(4,3), S(4,4)
S(3,0), S(3,1), S(3,2), S(3,3)
S(2,0), S(2,1), S(2,2)
S(1,0), S(1,1)
S(0,0)

Program

```
for (i = 0; i <= n; i++)
  for (j = 0; j <= i; j++)
    S(i,j);
```

Iteration space



State of Variables

n = 4, i = 3, j = 3

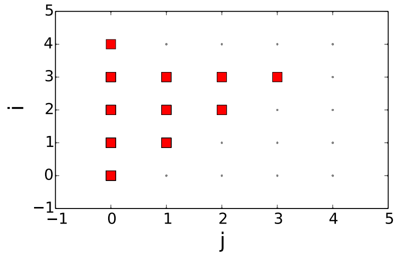
Statement Instances Executed

S(4,0), S(4,1), S(4,2), S(4,3), S(4,4)
S(3,0), S(3,1), S(3,2), S(3,3)
S(2,0), S(2,1), S(2,2)
S(1,0), S(1,1)
S(0,0)

Program

```
for (i = 0; i <= n; i++)
  for (j = 0; j <= i; j++)
    S(i,j);
```

Iteration space



State of Variables

n = 4, i = 4, j = 0

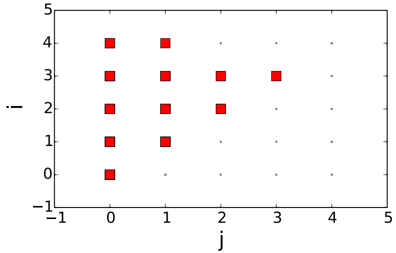
Statement Instances Executed

S(4,0), S(4,1), S(4,2), S(4,3), S(4,4)
S(3,0), S(3,1), S(3,2), S(3,3)
S(2,0), S(2,1), S(2,2)
S(1,0), S(1,1)
S(0,0)

Program

```
for (i = 0; i <= n; i++)
  for (j = 0; j <= i; j++)
    S(i,j);
```

Iteration space



State of Variables

n = 4, i = 4, j = 1

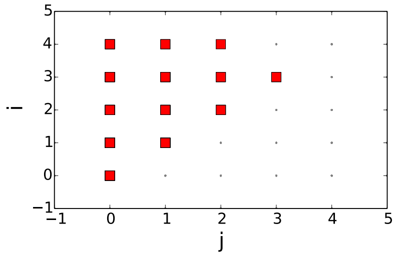
Statement Instances Executed

S(4,0), S(4,1), S(4,2), S(4,3), S(4,4)
S(3,0), S(3,1), S(3,2), S(3,3)
S(2,0), S(2,1), S(2,2)
S(1,0), S(1,1)
S(0,0)

Program

```
for (i = 0; i <= n; i++)
  for (j = 0; j <= i; j++)
    S(i,j);
```

Iteration space



State of Variables

n = 4, i = 4, j = 2

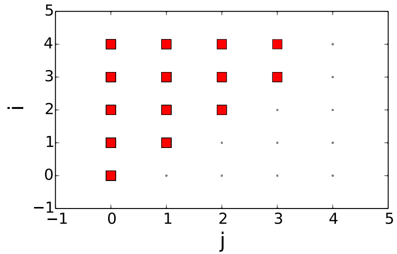
Statement Instances Executed

S(4,0), S(4,1), S(4,2), S(4,3), S(4,4)
S(3,0), S(3,1), S(3,2), S(3,3)
S(2,0), S(2,1), S(2,2)
S(1,0), S(1,1)
S(0,0)

Program

```
for (i = 0; i <= n; i++)
  for (j = 0; j <= i; j++)
    S(i,j);
```

Iteration space



State of Variables

n = 4, i = 4, j = 3

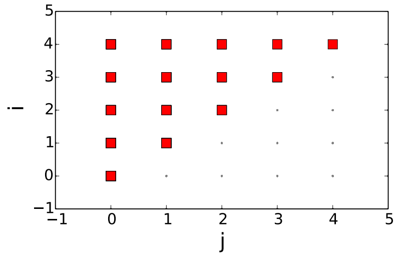
Statement Instances Executed

S(4,0), S(4,1), S(4,2), S(4,3), S(4,4)
S(3,0), S(3,1), S(3,2), S(3,3)
S(2,0), S(2,1), S(2,2)
S(1,0), S(1,1)
S(0,0)

Program

```
for (i = 0; i <= n; i++)
  for (j = 0; j <= i; j++)
    S(i,j);
```

Iteration space



State of Variables

n = 4, i = 4, j = 4

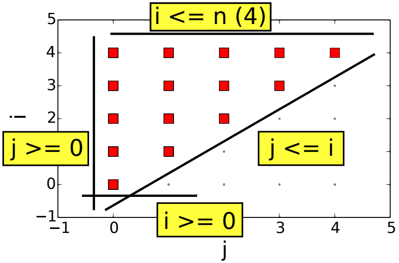
Statement Instances Executed

S(4,0), S(4,1), S(4,2), S(4,3), S(4,4)
S(3,0), S(3,1), S(3,2), S(3,3)
S(2,0), S(2,1), S(2,2)
S(1,0), S(1,1)
S(0,0)

Program

```
for (i = 0; i <= n; i++)
  for (j = 0; j <= i; j++)
    S(i,j);
```

Iteration space



State of Variables

n = 4, i = 4, j = 4

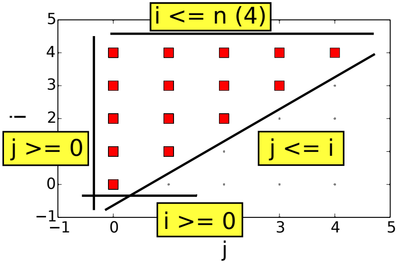
Statement Instances Executed

S(4,0), S(4,1), S(4,2), S(4,3), S(4,4)
S(3,0), S(3,1), S(3,2), S(3,3)
S(2,0), S(2,1), S(2,2)
S(1,0), S(1,1)
S(0,0)

Program

```
for (i = 0; i <= n; i++)
  for (j = 0; j <= i; j++)
    S(i,j);
```

Iteration space



State of Variables

n = 4, i = 4, j = 4

Statement Instances Executed

S(4,0), S(4,1), S(4,2), S(4,3), S(4,4)
S(3,0), S(3,1), S(3,2), S(3,3)
S(2,0), S(2,1), S(2,2)
S(1,0), S(1,1)
S(0,0)

$$= \{S(i,j) \mid 0 \leq i \leq n \land 0 \leq j \leq i\}$$

Schedule: Original

Model

$$\mathcal{I}_S = \{S(i,j) \mid 0 \leq i \leq n \wedge 0 \leq j \leq i\}$$
$$\Theta_S = \{S(i,j) \rightarrow (i,j)\}$$

Code

```
for (i = 0; i <= n; i++)
  for (j = 0; j <= i; j++)
    S(i,j);
```

Schedule: Original

Model

$$\mathcal{I}_S = \{S(i,j) \mid 0 \leq i \leq n \wedge 0 \leq j \leq i\}$$

$$\Theta_S = \{S(i,j) \rightarrow (i,j)\}$$

Code

```
for (c0 = 0; c0 <= n; c0++)
  for (c1 = 0; c1 <= c0; c1++)
    S(c0,c1);
```

Schedule: Interchanged

Model

$$\mathcal{I}_S = \{S(i,j) \mid 0 \leq i \leq n \wedge 0 \leq j \leq i\}$$
$$\Theta_S = \{S(i,j) \rightarrow (j,i)\}$$

Code

```
for (c0 = 0; c0 <= n; c0 += 1)
  for (c1 = c0; c1 <= n; c1 += 1)
    S(c1, c0);
```

Schedule: Strip-Mined

Model

$$\mathcal{I}_S = \{S(i,j) \mid 0 \leq i \leq n \wedge 0 \leq j \leq i\}$$
$$\Theta_S = \{S(i,j) \rightarrow (\lfloor i/4 \rfloor, j, i \bmod 4)\}$$

Code

```
for (c0 = 0; c0 <= floord(n, 4); c0 += 1)
  for (c1 = 0; c1 <= min(n, 4 * c0 + 3); c1 += 1)
      for (c2 = max(0, -4 * c0 + c1);
           c2 <= min(3, n - 4 * c0); c2 += 1)
            S(4 * c0 + c2, c1);
```

Schedule: Tiled

Model

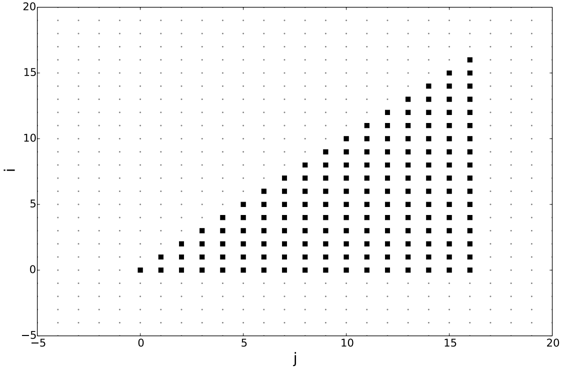
$$\mathcal{I}_S = \{S(i,j) \mid 0 \leq i \leq n \wedge 0 \leq j \leq i\}$$

$$\Theta_S = \{S(i,j) \rightarrow (\lfloor i/4 \rfloor, \lfloor j/4 \rfloor, i \bmod 4, j \bmod 4)\}$$

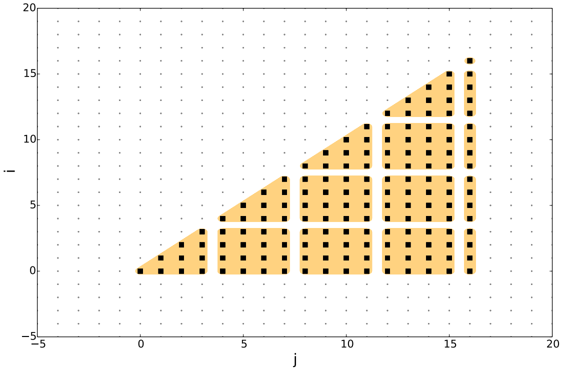
Code

```
// Tiles
for (c0 = 0; c0 <= floord(n, 4); c0 += 1)
  for (c1 = 0; c1 <= c0; c1 += 1)
    // Iterations
    for (c2 = 0; c2 <= min(3, n - 4 * c0); c2 += 1)
      for (c3 = 0; c3 <= min(3, 4 * c0 - 4 * c1 + c2);
```

Tiling illustrated



Tiling illustrated



Polly

Get Polly

- Install Polly
  `http://polly.grosser.es/get_started.html`
- Load Polly into clang (or gcc, opt, ...)
  ```
  alias clang clang -Xclang -load -Xclang LLVMPolly.so
  ```
- Default behaviour preserved
- Enable Polly optionally

Optimizing with Polly

```
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];
  }
```

Optimizing with Polly

```
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 0m15.336
```

Optimizing with Polly

```
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 0m15.336

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

LLVM's Loop Optimization Infrastructure

Loop Analysis

- Natural Loop Detection
- Scalar Evolution
- (Region Info)

Simple Loop Transformations

- Loop Simplify
- Loop Rotation
- Induction Variable Simplification
- Loop Invariant Code Motion
- Loop Unroll
- Loop Unswitch
- Loop Strength Reduction

Classical Loop Transformations

- Loop Interchange (not part of -O3)
- Loop Distribution (not part of -O3)

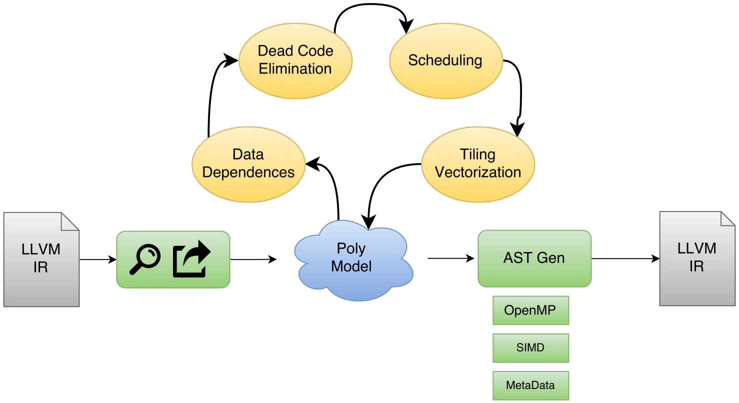
Vectorization

- Loop Vectorization
- SLP vectorization
- BB vectorizer (outdated)

Other

- Loop Reroll

The Polly Architecture



Report detected scops: -Rpass-analysis=polly

```c
void foo(long T, float A[][1024]) {
  for (long t = 0; t < T; t++)
    for (long i = 1; i < 1024 - 1; i++)
      A[t+1][i] += A[t][i+1] + A[t][i-1];
}
```

```
$ polly-clang-opt -O3 -mllvm -polly -Rpass-analysis=polly scop.c
scop.c:2:3: remark: SCoP begins here. [-Rpass-analysis=polly-scops]
  for (long t = 0; t < T; t++)
  ^
scop.c:4:50: remark: SCoP ends here. [-Rpass-analysis=polly-scops]
      A[t+1][i] += A[t][i+1] + A[t][i-1];
```

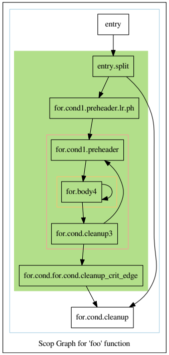
Report problems: -Rpass-missed=polly

```c
float sideeffect(float);
void foo(long T, long N, float A[][N]) {
  for (long t = 0; t < T; t++)
    for (long i = 1; i < N - 1; i++)
      A[t+1][i] += sideeffect(A[t][i+1] + A[t][i-1]);
}
```

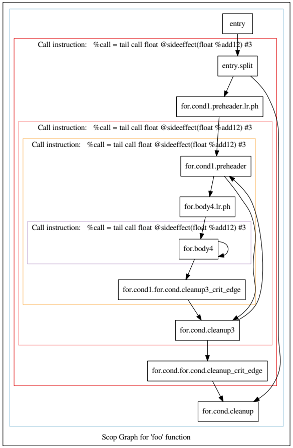
```
$polly-clang-opt  -c -O3 -mllvm -polly -Rpass-missed=polly missed.c
missed.c:3:5: remark: The following errors keep this region
                      from being a Scop.
      [-Rpass-missed=polly-detect]
    for (long i = 1; i < N - 1; i++)
    ^
missed.c:5:20: remark: This function call cannot be handled.
                       Try to inline it.
      [-Rpass-missed=polly-detect]
      A[t+1][i] += sideeffect(A[t][i+1] + A[t][i-1]);
```

Highlight SCoPs in CFG

-polly-show


entry
entry.split
for.cond1.preheader.lr.ph
for.cond1.preheader
for.body4
for.cond.cleanup3
for.cond.for.cond.cleanup_crit_edge
for.cond.cleanup
Scop Graph for 'foo' function

-polly-view-all


entry
Call instruction: %call = tail call float @sideeffect(float %add12) #3
entry.split
for.cond1.preheader.lr.ph
Call instruction: %call = tail call float @sideeffect(float %add12) #3
Call instruction: %call = tail call float @sideeffect(float %add12) #3
for.cond1.preheader
for.body4.lr.ph
Call instruction: %call = tail call float @sideeffect(float %add12) #3
for.body4
for.cond1.for.cond.cleanup3_crit_edge
for.cond.cleanup3
for.cond.for.cond.cleanup_crit_edge
for.cond.cleanup
Scop Graph for 'foo' function

The Polyhedral Representation (-debug-only=polly-scops)

```
for (long t = 0; t < T; t++)
  for (long i = 1; i < 1024 - 1; i++)
    A[t+1][i] += A[t][i+1] + A[t][i-1];
```

```
Domain :=
    [T] -> { Stmt_for_body4[i0, i1] : 0 <= i0 < T and 0 <= i1 <= 1021 };
Schedule :=
    [T] -> { Stmt_for_body4[i0, i1] -> [i0, i1] };
ReadAccess :=       [Reduction Type: NONE] [Scalar: 0]
    [T] -> { Stmt_for_body4[i0, i1] -> MemRef_A[i0, 2 + i1] };
ReadAccess :=       [Reduction Type: NONE] [Scalar: 0]
    [T] -> { Stmt_for_body4[i0, i1] -> MemRef_A[i0, i1] };
ReadAccess :=       [Reduction Type: NONE] [Scalar: 0]
    [T] -> { Stmt_for_body4[i0, i1] -> MemRef_A[1 + i0, 1 + i1] };
MustWriteAccess :=       [Reduction Type: NONE] [Scalar: 0]
    [T] -> { Stmt_for_body4[i0, i1] -> MemRef_A[1 + i0, 1 + i1] };
```

The Generated AST: (-debug-only=polly-ast)

```
for (long t = 0; t < T; t++)
  for (long i = 1; i < 1024 - 1; i++)
    A[t+1][i] += A[t][i+1] + A[t][i-1];
```

$$\Downarrow$$

```
if (1)
  for (c0 = 0; c0 < T; c0+=1)
    for (c1 = c0; c1 <= c0+1021; c1+=1)
       Stmt_for_body4(c0, -c0 + c1);
else
  {  /* original code */ }
```

Supported constructs

Supported constructs

Loops

- counted

  ```
  for (i=0; i < n / 13; i+=2)
  ```

Supported constructs

Loops

- counted
  ```
  for (i=0; i < n / 13; i+=2)
  ```
- Presburger Expressions
  ```
  for (i=0; i<22 && i>n; i+=2)
  ```

Supported constructs

Loops

- counted
  `for (i=0; i < n / 13; i+=2)`
- Presburger Expressions
  `for (i=0; i<22 && i>n; i+=2)`
- Multiple back-edges/exit-edges
  `break; continue;`

Supported constructs

Loops

- counted
  `for (i=0; i < n / 13; i+=2)`
- Presburger Expressions
  `for (i=0; i<22 && i>n; i+=2)`
- Multiple back-edges/exit-edges
  `break; continue;`
- `do..while`, `while`

Supported constructs

Loops

- counted
  `for (i=0; i < n / 13; i+=2)`
- Presburger Expressions
  `for (i=0; i<22 && i>n; i+=2)`
- Multiple back-edges/exit-edges
  `break; continue;`
- `do..while`, `while`

Conditions

- Presburger Conditions
  `if (5*i+b <= 13  12 > b)`

Supported constructs

Loops

- counted
  `for (i=0; i < n / 13; i+=2)`
- Presburger Expressions
  `for (i=0; i<22 && i>n; i+=2)`
- Multiple back-edges/exit-edges
  `break; continue;`
- `do..while`, `while`

Conditions

- Presburger Conditions
  `if (5*i+b <= 13  12 > b)`
- Data-dependent
  `if (B[i]) A[i] = A[i]/B[i]`

Supported constructs

Loops

- counted
  `for (i=0; i < n / 13; i+=2)`
- Presburger Expressions
  `for (i=0; i<22 && i>n; i+=2)`
- Multiple back-edges/exit-edges
  `break; continue;`
- `do..while`, `while`

Conditions

- Presburger Conditions
  `if (5*i+b <= 13  12 > b)`
- Data-dependent
  `if (B[i]) A[i] = A[i]/B[i]`
- Unstructured control flow
  `goto;`

Supported constructs

Loops

- counted
  `for (i=0; i < n / 13; i+=2)`
- Presburger Expressions
  `for (i=0; i<22 && i>n; i+=2)`
- Multiple back-edges/exit-edges
  `break; continue;`
- `do..while`, `while`

Conditions

- Presburger Conditions
  `if (5*i+b <= 13  12 > b)`
- Data-dependent
  `if (B[i]) A[i] = A[i]/B[i]`
- Unstructured control flow
  `goto;`

Arrays

- Multi-dimensionality: `A[][n][m]` / `A[][10][100]`
- Keywords: `restrict`

Calls

- Memory intrinsics: `memset/memmove/memcpy`
- Approximated behaviour:
  `read-none/read-only/pointer-arguments-only`

Examples: Valid SCoPs

do..while loop

```
int i = 0;

do {
  int b = 2 * i;
  int c = b * 3 + 5 * i;
  A[c] = i;
  i++;
} while (i < N);
```

pointer loop

```
int A[1024]
int *B;

while(B < &A[1024]) {
  *B = i;
  ++B;
}
```

Profitability Heuristics

Polly's default policy: No regressions

- Minimal compile time increase
- No spurious run-time changes

Rules:

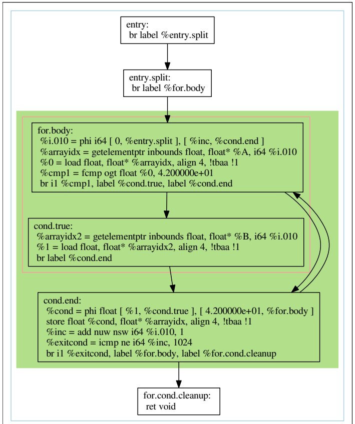
- Bail out as early as possible
  - At least two loops (or one very big one)
  - At least one read access
- Only change IR if Polly did something beneficial
  - Performed Schedule Transformation
  - Added alias run-time check

Can be overwritten by: `-polly-process-unprofitable`

Non-affine Statements

```
void nonaffine(float A[],
               float B[]) {
  for (long i = 0; i < 1024; i++)
    A[i] = A[i] > 42 ? B[i] : 42;
}
```

$\Rightarrow$



```
entry:
 br label %entry.split

entry.split:
 br label %for.body

for.body:
 %i.010 = phi i64 [ 0, %entry.split ], [ %inc, %cond.end ]
 %arrayidx = getelementptr inbounds float, float* %A, i64 %i.010
 %0 = load float, float* %arrayidx, align 4, !tbaa !1
 %cmp1 = fcmp ogt float %0, 4.200000e+01
 br i1 %cmp1, label %cond.true, label %cond.end

cond.true:
 %arrayidx2 = getelementptr inbounds float, float* %B, i64 %i.010
 %1 = load float, float* %arrayidx2, align 4, !tbaa !1
 br label %cond.end

cond.end:
 %cond = phi float [ %1, %cond.true ], [ 4.200000e+01, %for.body ]
 store float %cond, float* %arrayidx, align 4, !tbaa !1
 %inc = add nuw nsw i64 %i.010, 1
 %exitcond = icmp ne i64 %inc, 1024
 br i1 %exitcond, label %for.body, label %for.cond.cleanup

for.cond.cleanup:
 ret void
```

Scop Graph for 'nonaffine' function

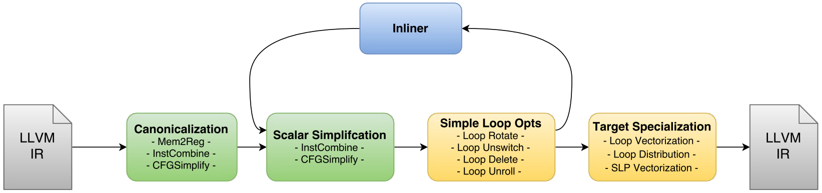
Schedule Optimizer: -polly-opt-isl

Roman Garev (outer-loop vectorization)

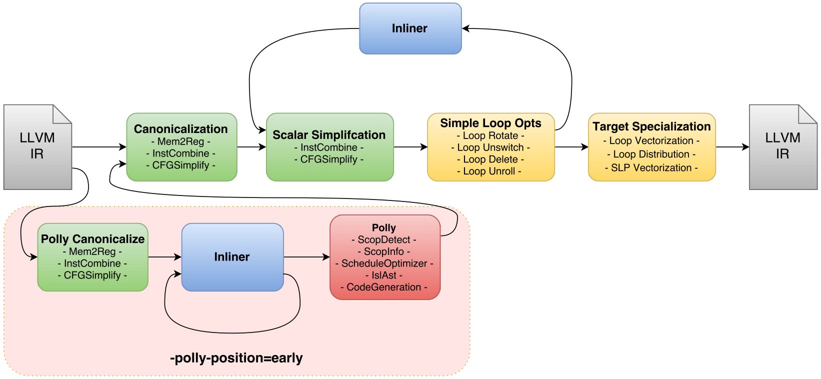
- Schedule using a Pluto style LP to maximize:
  - Data locality
  - Parallelism
  - Tilability
- Post-scheduling optimizations
  - Tile innermost tileable band
  - Strip-mine innermost parallel loop for SIMDization

Implementation: isl_schedule

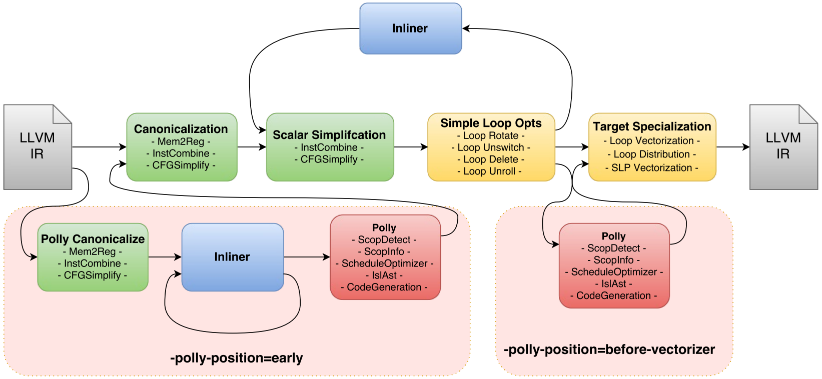
LLVM Pass Pipeline



LLVM Pass Pipeline


Inliner
LLVM IR
Canonicalization - Mem2Reg - - InstCombine - - CFGSimplify -
Scalar Simplifcation - InstCombine - - CFGSimplify -
Simple Loop Opts - Loop Rotate - - Loop Unswitch - - Loop Delete - - Loop Unroll -
Target Specialization - Loop Vectorization - - Loop Distribution - - SLP Vectorization -
LLVM IR
Polly Canonicalize - Mem2Reg - - InstCombine - - CFGSimplify -
Inliner
Polly - ScopDetect - - ScopInfo - - ScheduleOptimizer - - IslAst - - CodeGeneration -
-polly-position=early

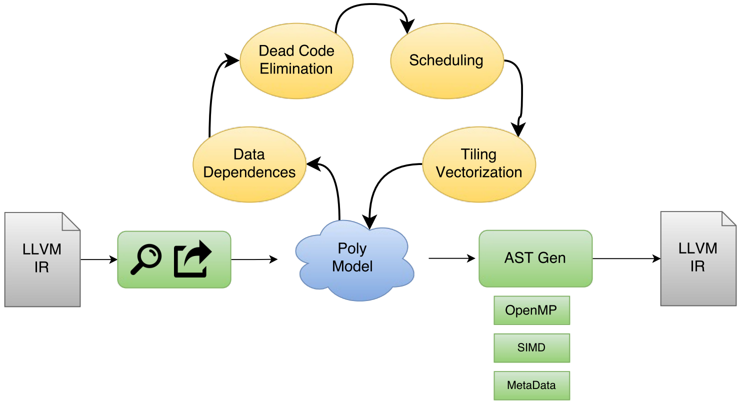
LLVM Pass Pipeline



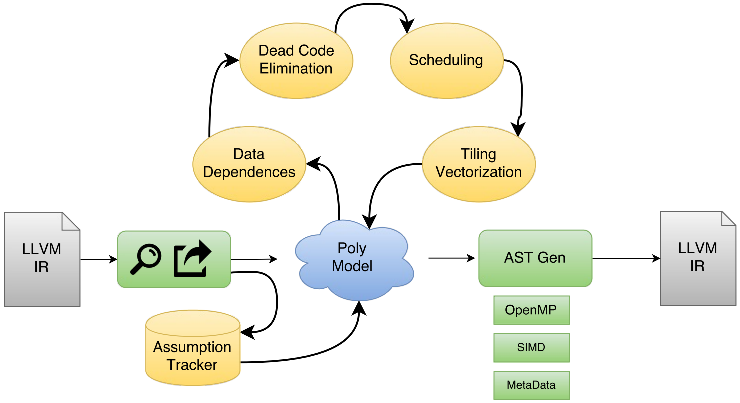
Auto Parallelization: (-mllvm) -polly-parallel (-lgomp)

- Run outer-most parallel loop with OpenMP
- Directly emit calls to libgomp (gcc's OpenMP library)
- Execution can be controlled by setting OMP environment variables:
  - OMP_SCHEDULE=static,dynamic,guided,auto
  - OMP_NUM_TRHEADS=$<$ *num* $>$ or (-mllvm) -polly-num-threads=$<$ *num* $>$

Optimistic Assumption Tracking



Optimistic Assumption Tracking



Assumption tracking in Polly

```
void oddEvenCopy(int N, int M, float A[][M]) {
  for (int i = 0; i < M; i++)
    for (int j = 0; j < N; j++)
      A[2 * j][i] = A[2 * j + 1][i];
}
```

$\Rightarrow$ 15s

Assumption tracking in Polly

```c
void oddEvenCopy(int N, int M, float A[][M]) {
  for (int i = 0; i < M; i++)
    for (int j = 0; j < N; j++)
      A[2 * j][i] = A[2 * j + 1][i];
}
```

$\Rightarrow$ 15s

$\Downarrow$ **Clearly beneficial loop interchange** $\Downarrow$

```c
void oddEvenCopy(int N, int M, float A[][M]) {
  for (int j = 0; j < N; j++)
    for (int i = 0; i < M; i++)
      A[2 * j][i] = A[2 * j + 1][i];
}
```

$\Rightarrow$ 2s

... is not always obvious to the compiler

```c
void oddEvenCopy(int N, int M, float A[][20000]) {
  for (int i = 0; i < M; i++)
    for (int j = 0; j < N; j++)
      A[2 * j][i] = A[2 * j + 1][i];
}
```

$\Rightarrow$?

… is not always obvious to the compiler

```c
void oddEvenCopy(int N, int M, float A[][20000]) {
  for (int i = 0; i < M; i++)
    for (int j = 0; j < N; j++)
      A[2 * j][i] = A[2 * j + 1][i];
}
```

$\Rightarrow$?

- Interchange only allowed if $M \leq 20000$ (or $N < 0$)
- …, but code with M $=$ 20001 is well defined.

Be optimistic - Optimize for the common case

1. Take & collect assumptions
2. Simplify
3. Verify dynamically

Run-time alias checks

```
void aliasChecks(long n, long m,
                 float A[],
                 float B[][m]) {
  for (long i = 0; i < n; i++)
    for (long j = 0; j < m; j++)
      A[i] += B[i][j];
}
```

Run-time alias checks

```
void aliasChecks(long n, long m,
                 float A[],
                 float B[][m]) {
  for (long i = 0; i < n; i++)
    for (long j = 0; j < m; j++)
      A[i] += B[i][j];
}
```

$\Longrightarrow$

```
if (&B[n-1][m] <= &A[0]
 || &A[n] <= &B[0][0])
  for (int c0 = 0; c0 < n; c0 += 1)
    for (int c1 = 0; c1 < m; c1 += 1)
      Stmt_for_body4(c0, c1);
else
  { /* original code */ }
```

Possibly Invariant Loads

```c
void mayLoad(int *s0, int *s1) {
    for (int i = 0; i < *s0; i++)
      for (int j = 0; j < *s1; j++)
        ...
}
```

Possibly Invariant Loads

```
void mayLoad(int *s0, int *s1) {
    for (int i = 0; i < *s0; i++)
      for (int j = 0; j < *s1; j++)
        ...
}
```

$\Longrightarrow$

```
void mayLoad(int *s0, int *s1) {
  int s0val = *s0;
  int s1val = 1;

  if (s0val > 0)
    s1val = *s1;

  for (int i = 0; i < s0val; i++)
    for (int j = 0; j < s1val; j++)
        ...
}
```

Check Hoisting

```
for (int i = 0; i < N; i++) {
  for (int j = 0; j < N; j++)
    A[i][j] = B[i][j];

  if (DebugLevel > 5)
    printf("Column \%d copied\n", i)
}
```

Check Hoisting

```c
for (int i = 0; i < N; i++) {
  for (int j = 0; j < N; j++)
    A[i][j] = B[i][j];

  if (DebugLevel > 5)
    printf("Column \%d copied\n", i)
}
```

$\Rightarrow$

```c
if (DebugLevel <= 5) {

  #pragma parallel
  for (int i = 0; i < N; i++)
    #pragma simd
    for (int j = 0; j < N; j++)
      A[i][j] = B[i][j];

} else {
  /* .. */
}
```

User provided assumptions

```
void user(long n, long m,
          float A[][1024],
          float B[][1024]) {

  for (long i = 0; i < n; i++)
    for (long j = 0; j < m; j++)
      A[i][j] += B[i][j];
}
```

User provided assumptions

```
void user(long n, long m,
          float A[][1024],
          float B[][1024]) {

  for (long i = 0; i < n; i++)
    for (long j = 0; j < m; j++)
      A[i][j] += B[i][j];
}
```

$\Longrightarrow$

```
if (m <= 1024)
  for (int c0 = 0; c0 < n; c0 += 1)
    for (int c1 = 0; c1 < m; c1 += 1)
      Stmt_for_body4(c0, c1);
else
  {  /* original code */ }
```

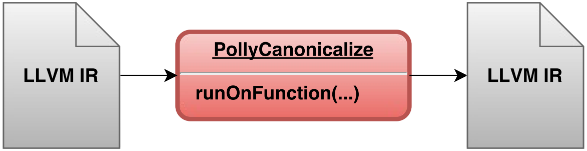
User provided assumptions II

```
void user(long n, long m,
          float A[][1024],
          float B[][1024]) {
  __builtin_assume(m <= 1024);
  for (long i = 0; i < n; i++)
    for (long j = 0; j < m; j++)
      A[i][j] += B[i][j];
}
```

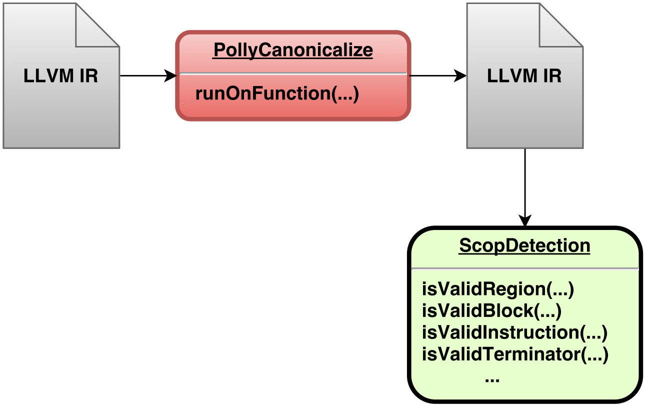
$\Longrightarrow$

```
if (1)
  for (int c0 = 0; c0 < n; c0 += 1)
    for (int c1 = 0; c1 < m; c1 += 1)
      Stmt_for_body4(c0, c1);
else
  {  /* original code */ }
```

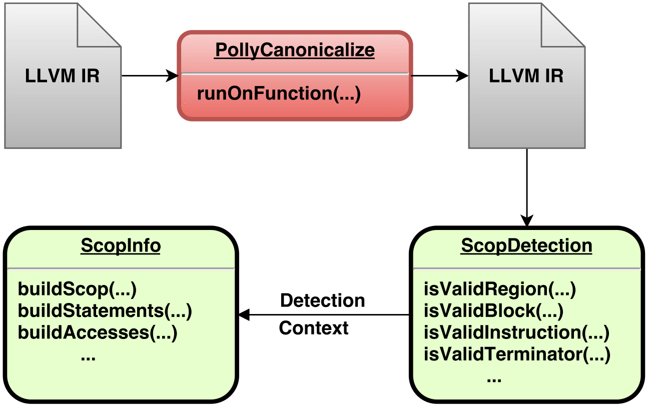
Polly Implementation



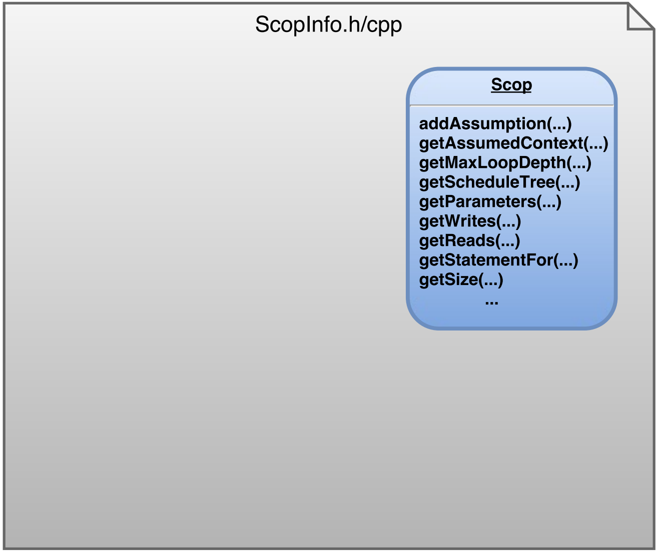
Polly Implementation



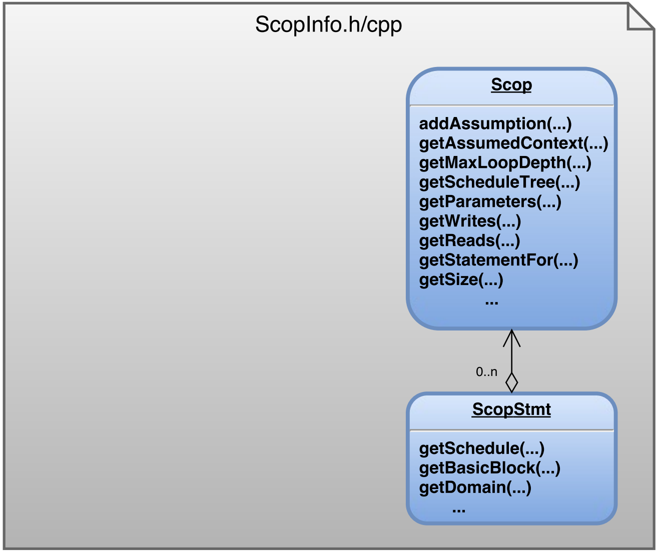
Polly Implementation



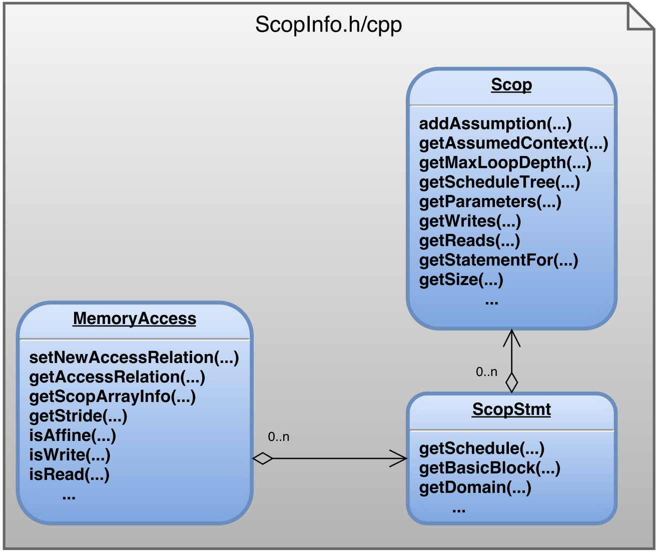
SCoP Representation



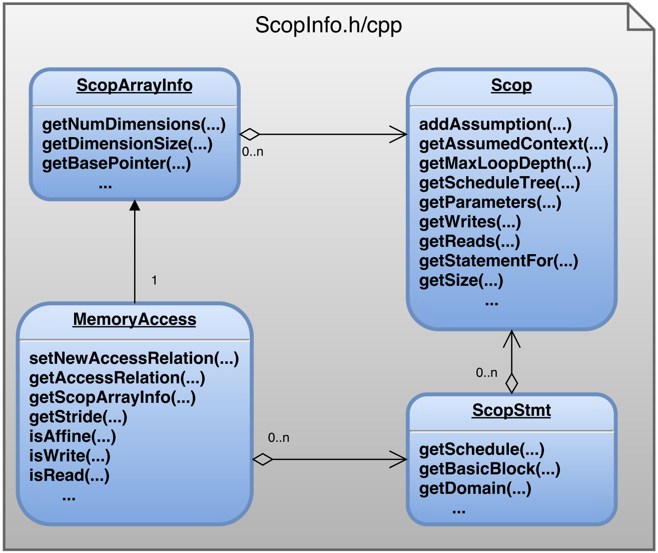
SCoP Representation



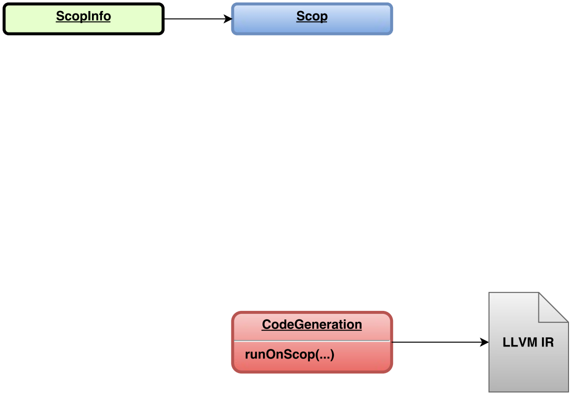
SCoP Representation



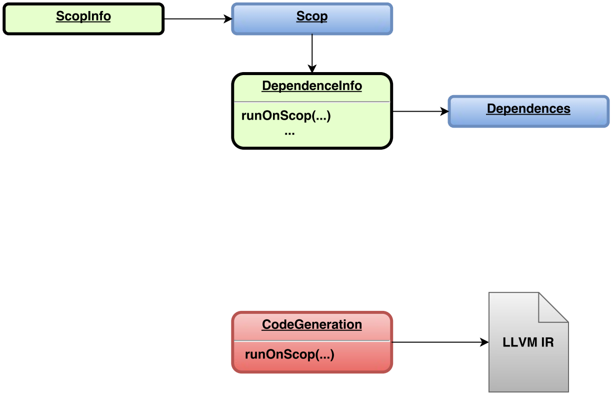
SCoP Representation



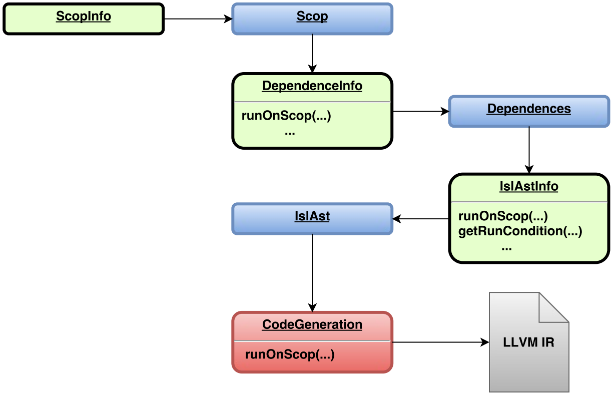
Polly Implementation



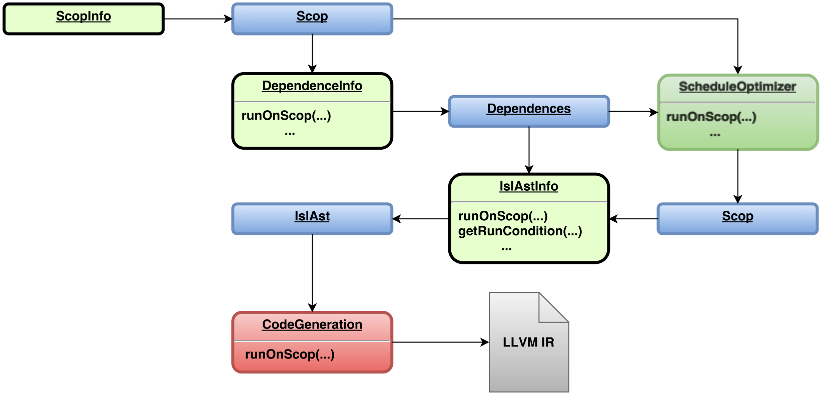
Polly Implementation



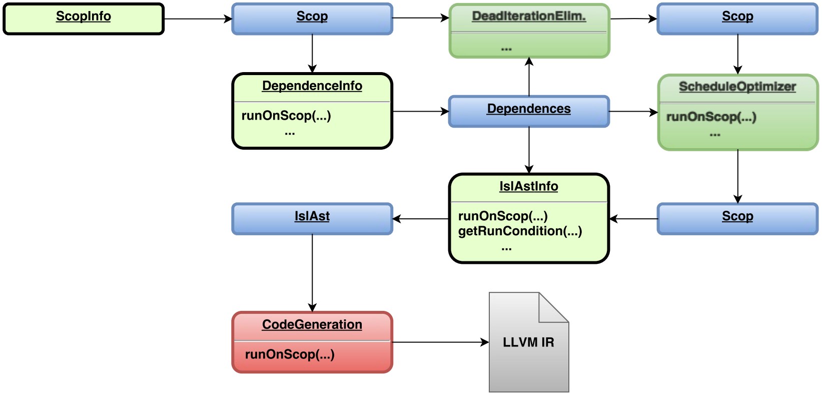
Polly Implementation



Polly Implementation



Polly Implementation



NAS Parallel Benchmarks — BT — rhs.c

NAS Parallel Benchmarks — BT — rhs.c

```
void compute_rhs() {
  int i, j, k, m;
  double rho_inv, uijk, up1, um1, vijk, vp1, vm1, wijk, wp1, wm1;

  if (timeron) timer_start(t_rhs);

  for (k = 0; k <= grid_points[2]-1; k++) {
    for (j = 0; j <= grid_points[1]-1; j++) {
      for (i = 0; i <= grid_points[0]-1; i++) {
        rho_inv = 1.0/u[k][j][i][0];
        rho_i[k][j][i] = rho_inv;
        us[k][j][i] = u[k][j][i][1] * rho_inv;
        vs[k][j][i] = u[k][j][i][2] * rho_inv;
        ws[k][j][i] = u[k][j][i][3] * rho_inv;
        square[k][j][i] = 0.5* (
            u[k][j][i][1]*u[k][j][i][1] +
            u[k][j][i][2]*u[k][j][i][2] +
            u[k][j][i][3]*u[k][j][i][3] ) * rho_inv;
        qs[k][j][i] = square[k][j][i] * rho_inv;
      }
    }
  }
```

NAS Parallel Benchmarks — BT — rhs.c

```
for (k = 0; k <= grid_points[2]-1; k++) {
  for (j = 0; j <= grid_points[1]-1; j++) {
    for (i = 0; i <= grid_points[0]-1; i++) {
      for (m = 0; m < 5; m++) {
        rhs[k][j][i][m] = forcing[k][j][i][m];
      }
    }
  }
}

if (timeron) timer_start(t_rhsx);

for (k = 1; k <= grid_points[2]-2; k++) {
  for (j = 1; j <= grid_points[1]-2; j++) {
    for (i = 1; i <= grid_points[0]-2; i++) {
      uijk = us[k][j][i];
      up1  = us[k][j][i+1];
      um1  = us[k][j][i-1];

      rhs[k][j][i][0] = rhs[k][j][i][0] + dx1tx1 *
        (u[k][j][i+1][0] - 2.0*u[k][j][i][0] +
         u[k][j][i-1][0]) -
        tx2 * (u[k][j][i+1][1] - u[k][j][i-1][1]);
```

NAS Parallel Benchmarks — BT — rhs.c

```
rhs[k][j][i][1] = rhs[k][j][i][1] + dx2tx1 *
  (u[k][j][i+1][1] - 2.0*u[k][j][i][1] +
   u[k][j][i-1][1]) +
  xxcon2*con43 * (up1 - 2.0*uijk + um1) -
  tx2 * (u[k][j][i+1][1]*up1 -
      u[k][j][i-1][1]*um1 +
      (u[k][j][i+1][4]- square[k][j][i+1]-
       u[k][j][i-1][4]+ square[k][j][i-1])* c2);

rhs[k][j][i][2] = rhs[k][j][i][2] + dx3tx1 *
  (u[k][j][i+1][2] - 2.0*u[k][j][i][2] +
   u[k][j][i-1][2]) +
  xxcon2 * (vs[k][j][i+1] - 2.0*vs[k][j][i] +
      vs[k][j][i-1]) -
  tx2 * (u[k][j][i+1][2]*up1 - u[k][j][i-1][2]*um1);

rhs[k][j][i][3] = rhs[k][j][i][3] + dx4tx1 *
  (u[k][j][i+1][3] - 2.0*u[k][j][i][3] +
   u[k][j][i-1][3]) +
  xxcon2 * (ws[k][j][i+1] - 2.0*ws[k][j][i] +
      ws[k][j][i-1]) -
  tx2 * (u[k][j][i+1][3]*up1 - u[k][j][i-1][3]*um1);

  /* ≈300 more lines of similar code */
```

NAS Parallel Benchmarks — BT — rhs.c

```c
for (k = 0; k <= grid_points[2]-1; k++)
  for (j = 0; j <= grid_points[1]-1; j++)
    for (i = 0; i <= grid_points[0]-1; i++)
      for (m = 0; m < 5; m++)
        rhs[k][j][i][m] = forcing[k][j][i][m];

if (timeron) timer_start(t_rhsx);

for (k = 1; k <= grid_points[2]-2; k++) {
  for (j = 1; j <= grid_points[1]-2; j++) {
    for (i = 1; i <= grid_points[0]-2; i++) {
      /* ... */
```

[a] Sanyam and Yew, PLDI 15

NAS Parallel Benchmarks — BT — rhs.c

```
for (k = 0; k <= grid_points[2]-1; k++)
  for (j = 0; j <= grid_points[1]-1; j++)
    for (i = 0; i <= grid_points[0]-1; i++)
      for (m = 0; m < 5; m++)
        rhs[k][j][i][m] = forcing[k][j][i][m];

if (timeron) timer_start(t_rhsx);

for (k = 1; k <= grid_points[2]-2; k++) {
  for (j = 1; j <= grid_points[1]-2; j++) {
    for (i = 1; i <= grid_points[0]-2; i++) {
      /* ... */
```

+ 6× speedup for 8 threads/cores [a]

[a] Sanyam and Yew, PLDI 15

NAS Parallel Benchmarks — BT — rhs.c

```
for (k = 0; k <= grid_points[2]-1; k++)
  for (j = 0; j <= grid_points[1]-1; j++)
    for (i = 0; i <= grid_points[0]-1; i++)
      for (m = 0; m < 5; m++)
        rhs[k][j][i][m] = forcing[k][j][i][m];

if (timeron) timer_start(t_rhsx);

for (k = 1; k <= grid_points[2]-2; k++) {
  for (j = 1; j <= grid_points[1]-2; j++) {
    for (i = 1; i <= grid_points[0]-2; i++) {
      /* ... */
```

+ 6× speedup for 8 threads/cores [a]
- Possible variant loop bounds

[a] Sanyam and Yew, PLDI 15

NAS Parallel Benchmarks — BT — rhs.c

```
for (k = 0; k <= grid_points[2]-1; k++)
  for (j = 0; j <= grid_points[1]-1; j++)
    for (i = 0; i <= grid_points[0]-1; i++)
      for (m = 0; m < 5; m++)
        rhs[k][j][i][m] = forcing[k][j][i][m];

if (timeron) timer_start(t_rhsx);

for (k = 1; k <= grid_points[2]-2; k++) {
  for (j = 1; j <= grid_points[1]-2; j++) {
    for (i = 1; i <= grid_points[0]-2; i++) {
      /* ... */
```

+ $6\times$ speedup for 8 threads/cores [a]
- Possible variant loop bounds
- Possible out-of-bound accesses

[a] Sanyam and Yew, PLDI 15

NAS Parallel Benchmarks — BT — rhs.c

```
for (k = 0; k <= grid_points[2]-1; k++)
  for (j = 0; j <= grid_points[1]-1; j++)
    for (i = 0; i <= grid_points[0]-1; i++)
      for (m = 0; m < 5; m++)
        rhs[k][j][i][m] = forcing[k][j][i][m];

if (timeron) timer_start(t_rhsx);

for (k = 1; k <= grid_points[2]-2; k++) {
  for (j = 1; j <= grid_points[1]-2; j++) {
    for (i = 1; i <= grid_points[0]-2; i++) {
      /* ... */
```

+ 6× speedup for 8 threads/cores [a]
- Possible variant loop bounds
- Possible out-of-bound accesses
- Possible execution of non-pure calls

[a] Sanyam and Yew, PLDI 15

NAS Parallel Benchmarks — BT — rhs.c

```
for (k = 0; k <= grid_points[2]-1; k++)
  for (j = 0; j <= grid_points[1]-1; j++)
    for (i = 0; i <= grid_points[0]-1; i++)
      for (m = 0; m < 5; m++)
        rhs[k][j][i][m] = forcing[k][j][i][m];

if (timeron) timer_start(t_rhsx);

for (k = 1; k <= grid_points[2]-2; k++) {
  for (j = 1; j <= grid_points[1]-2; j++) {
    for (i = 1; i <= grid_points[0]-2; i++) {
      /* ... */
```

+ 6× speedup for 8 threads/cores [a]
- Possible variant loop bounds
- Possible out-of-bound accesses
- Possible execution of non-pure calls
- Possible integer under/overflows complicate loop bounds

[a] Sanyam and Yew, PLDI 15

NAS Parallel Benchmarks — BT — rhs.c

```
clang -Rpass-analysis=polly-scops -O3 -polly rhs.c
```

```
rhs.c:47:3: remark: SCoP begins here. [-Rpass-analysis=polly-scops]
  for (k = 0; k <= grid_points[2]-1; k++) {
  ^

  /* ... */

rhs.c:418:16: remark: SCoP ends here. [-Rpass-analysis=polly-scops]
  if (timeron) timer_stop(t_rhs);
               ^
```

NAS Parallel Benchmarks — BT — rhs.c

```
clang -Rpass-analysis=polly-scops -O3 -polly rhs.c
```

```
    rhs.c:79:16: remark: No-error assumption: [grid_points, grid_points', t
          {  : timeron = 0 } [-Rpass-analysis=polly-scops]
      if (timeron) timer_start(t_rhsx);
                   ^
```

NAS Parallel Benchmarks — BT — rhs.c

`clang -Rpass-analysis=polly-scops -O3 -polly rhs.c`

```
rhs.c:50:23: remark: Inbounds assumption: [grid_points, grid_points', grid_points''] ->
 {  : grid_points <= 0 or (grid_points >= 1 and grid_points' <= 0) or (grid_points >= 1 and
      grid_points' >= 104 and grid_points'' <= 0) or (grid_points >= 1 and grid_points' <= 103
      and grid_points' >= 1 and grid_points'' <= 103) } [-Rpass-analysis=polly-scops]
        rho_inv = 1.0/u[k][j][i][0];
                      ^
rhs.c:144:27: remark: Inbounds assumption: [grid_points, grid_points', timeron, grid_points''] ->
 {  : grid_points <= 2 or (grid_points >= 3 and grid_points' <= 104) } [-Rpass-analysis=polly-scops]
        rhs[k][j][i][m] = rhs[k][j][i][m]- dssp *
                          ^
rhs.c:171:27: remark: Inbounds assumption: [grid_points, grid_points', timeron, grid_points''] ->
 {  : grid_points <= 2 or (grid_points >= 3 and grid_points' <= 2) or (grid_points >= 3
      and grid_points' <= 104 and grid_points' >= 3 and grid_points'' <= 105 and grid_points'' >= 3) }
        rhs[k][j][i][m] = rhs[k][j][i][m] - dssp *
                          ^
```

NAS Parallel Benchmarks — BT — rhs.c

```
clang -Rpass-analysis=polly-scops -O3 -polly rhs.c
```

```
rhs.c:419:1: remark: No-overflows assumption: [grid_points, grid_points’, grid_points’’, timeron] ->
{: (grid_points >= 3 and grid_points’ >= 3 and grid_points’’ >= -2147483643) or (grid_points >= 3 and
   grid_points’ <= 2 and grid_points’ >= -2147483643 and grid_points’’ >= -2147483646) or
   (grid_points <= 2 and grid_points >= -2147483643 and grid_points’ >= 3 and grid_points’’ >= -2147483646
   (grid_points <= 2 and grid_points >= -2147483644 and grid_points’ <= 2 and grid_points’ >= -2147483646)
   (grid_points’ = -2147483644 and grid_points >= 3 and grid_points’’ <= 2 and grid_points’’ >= -214748364
   (grid_points = -2147483644 and grid_points’ >= 3 and grid_points’’ <= 2 and grid_points’’ >= -214748364
```

```
__builtin_assume(grid_points[0] >= -2147483643 &&
                 grid_points[1] >= -2147483643 &&
                 grid_points[2] >= -2147483643);
```

NAS Parallel Benchmarks — BT — rhs.c

```
clang -Rpass-analysis=polly-scops -O3 -polly rhs.c
```

```
rhs.c:50:23: remark: Possibly aliasing pointer, use restrict keyword.
      [-Rpass-analysis=polly-scops]
        rho_inv = 1.0/u[k][j][i][0];
                      ^
rhs.c:56:13: remark: Possibly aliasing pointer, use restrict keyword.
      [-Rpass-analysis=polly-scops]
            u[k][j][i][1]*u[k][j][i][1] +
            ^
```

PARSEC — blackscholes — blackscholes.c

PARSEC — blackscholes — blackscholes.c

```c
float BlkSchlsEqEuroNoDiv(float sptprice, float strike, float rate,
                          float volatility, float time, int otype) {
    float xD1, xD2, xDen, d1, d2, FutureValueX, NofXd1, NofXd2, NegNofXd1,
          NegNofXd2, Price;
    xD1 = rate + volatility * volatility; * 0.5;
    xD1 = xD1 * time;
    xD1 = xD1 + log( sptprice / strike );
    xDen = volatility * sqrt(time);
    xD1 = xD1 / xDen;
    xD2 = xD1 -  xDen;
    d1 = xD1;
    d2 = xD2;
    NofXd1 = CNDF( d1 );
    NofXd2 = CNDF( d2 );
    FutureValueX = strike * ( exp( -(rate)*(time) ) );
    if (otype == 0) {
        Price = (sptprice * NofXd1) - (FutureValueX * NofXd2);
    } else {
        NegNofXd1 = (1.0 - NofXd1);
        NegNofXd2 = (1.0 - NofXd2);
        Price = (FutureValueX * NegNofXd2) - (sptprice * NegNofXd1);
    }
    return Price;
}
```

PARSEC — blackscholes — blackscholes.c

```
int bs_thread(void *tid_ptr) {
    int tid = *(int *)tid_ptr;
    int start = tid * (numOptions / nThreads);
    int end = start + (numOptions / nThreads);

    for (int j = 0; j < NUM_RUNS; j++)
        for (int i = start; i < end; i++)
          prices[i] = BlkSchlsEqEuroNoDiv(sptprice[i], strike[i], r
                                          volatility[i], otime[i],
    return 0;
}
```

PARSEC — blackscholes — blackscholes.c

```
int bs_thread(void *tid_ptr) {
    int tid = *(int *)tid_ptr;
    int start = tid * (numOptions / nThreads);
    int end = start + (numOptions / nThreads);

    for (int j = 0; j < NUM_RUNS; j++)
        for (int i = start; i < end; i++)
          prices[i] = BlkSchlsEqEuroNoDiv(sptprice[i], strike[i], r
                                          volatility[i], otime[i],
    return 0;
}
```

+ 2.9× speedup for manual parallelization on a quad-core i7

PARSEC — blackscholes — blackscholes.c

```
int bs_thread(void *tid_ptr) {
    int tid = *(int *)tid_ptr;
    int start = tid * (numOptions / nThreads);
    int end = start + (numOptions / nThreads);

    for (int j = 0; j < NUM_RUNS; j++)
        for (int i = start; i < end; i++)
          prices[i] = BlkSchlsEqEuroNoDiv(sptprice[i], strike[i], r
                                          volatility[i], otime[i],
    return 0;
}
```

+ 2.9× speedup for manual parallelization on a quad-core i7
+ 2.8× speedup for automatic parallelization on a quad-core i7

PARSEC — blackscholes — blackscholes.c

```
int bs_thread(void *tid_ptr) {
    int tid = *(int *)tid_ptr;
    int start = tid * (numOptions / nThreads);
    int end = start + (numOptions / nThreads);

    for (int j = 0; j < NUM_RUNS; j++)
        for (int i = start; i < end; i++)
          prices[i] = BlkSchlsEqEuroNoDiv(sptprice[i], strike[i], r
                                          volatility[i], otime[i],
    return 0;
}
```

+ 2.9× speedup for manual parallelization on a quad-core i7
+ 2.8× speedup for automatic parallelization on a quad-core i7

– Possible aliasing

PARSEC — blackscholes — blackscholes.c

```
int bs_thread(void *tid_ptr) {
    int tid = *(int *)tid_ptr;
    int start = tid * (numOptions / nThreads);
    int end = start + (numOptions / nThreads);

    for (int j = 0; j < NUM_RUNS; j++)
        for (int i = start; i < end; i++)
          prices[i] = BlkSchlsEqEuroNoDiv(sptprice[i], strike[i], r
                                          volatility[i], otime[i],
    return 0;
}
```

+ 2.9× speedup for manual parallelization on a quad-core i7
+ 2.8× speedup for automatic parallelization on a quad-core i7

- Possible aliasing
- Possible execution of non-pure calls

PARSEC — blackscholes — blackscholes.c

```
int bs_thread(void *tid_ptr) {
    int tid = *(int *)tid_ptr;
    int start = tid * (numOptions / nThreads);
    int end = start + (numOptions / nThreads);

    for (int j = 0; j < NUM_RUNS; j++)
        for (int i = start; i < end; i++)
          prices[i] = BlkSchlsEqEuroNoDiv(sptprice[i], strike[i], r
                                             volatility[i], otime[i],
    return 0;
}
```

+ 2.9× speedup for manual parallelization on a quad-core i7
+ 2.8× speedup for automatic parallelization on a quad-core i7

- Possible aliasing
- Possible execution of non-pure calls
- Possible execution of dead-iterations (`0 <= j < NUM_RUNS - 1`)

PARSEC — blackscholes — blackscholes.c

```
int bs_thread(void *tid_ptr) {
    int tid = *(int *)tid_ptr;
    int start = tid * (numOptions / nThreads);
    int end = start + (numOptions / nThreads);

    for (int j = 0; j < NUM_RUNS; j++)
        for (int i = start; i < end; i++)
          prices[i] = BlkSchlsEqEuroNoDiv(sptprice[i], strike[i], r
                                          volatility[i], otime[i],
    return 0;
}
```

+ 2.9× speedup for manual parallelization on a quad-core i7
+ 2.8× speedup for automatic parallelization on a quad-core i7
+ 6.5× speedup for sequential execution (native input)
- Possible aliasing
- Possible execution of non-pure calls
- Possible execution of dead-iterations (`0 <= j < NUM_RUNS - 1`)

The Polly Loop Optimizer

- High-level loop manipulation framework for LLVM
- Generic loop modeling based on “Semantic SCoPs”
- Optimistic assumptions in case of insufficient static knowledge
- Fast compile-time

- Open and welcoming community (we try at least)
- Industry/Research Partnership through pollylabs.org

Thank you!

SPEC 2006 — hmmer — fast_algorithms.c

SPEC 2006 — hmmer — fast_algorithms.c

```
for (k = 1; k <= M; k++) {
  mc[k] = mpp[k - 1] + tpmm[k - 1];
  if ((sc = ip[k - 1] + tpim[k - 1]) > mc[k]) mc[k] = sc;
  if ((sc = dpp[k - 1] + tpdm[k - 1]) > mc[k]) mc[k] = sc;
  if ((sc = xmb + bp[k]) > mc[k]) mc[k] = sc;
  mc[k] += ms[k];
  if (mc[k] < -INFTY) mc[k] = -INFTY;


  dc[k] = dc[k - 1] + tpdd[k - 1];
  if ((sc = mc[k - 1] + tpmd[k - 1]) > dc[k]) dc[k] = sc;
  if (dc[k] < -INFTY) dc[k] = -INFTY;



  if (k < M) {
    ic[k] = mpp[k] + tpmi[k];
    if ((sc = ip[k] + tpii[k]) > ic[k]) ic[k] = sc;
    ic[k] += is[k];
    if (ic[k] < -INFTY) ic[k] = -INFTY;
  }
}
```

SPEC 2006 — hmmer — fast_algorithms.c

```
#pragma clang loop vectorize(enable)
for (k = 1; k <= M; k++) {
  mc[k] = mpp[k - 1] + tpmm[k - 1];
  if ((sc = ip[k - 1] + tpim[k - 1]) > mc[k]) mc[k] = sc;
  if ((sc = dpp[k - 1] + tpdm[k - 1]) > mc[k]) mc[k] = sc;
  if ((sc = xmb + bp[k]) > mc[k]) mc[k] = sc;
  mc[k] += ms[k];
  if (mc[k] < -INFTY) mc[k] = -INFTY;
}
for (k = 1; k <= M; k++) {
  dc[k] = dc[k - 1] + tpdd[k - 1];
  if ((sc = mc[k - 1] + tpmd[k - 1]) > dc[k]) dc[k] = sc;
  if (dc[k] < -INFTY) dc[k] = -INFTY;



  if (k < M) {
    ic[k] = mpp[k] + tpmi[k];
    if ((sc = ip[k] + tpii[k]) > ic[k]) ic[k] = sc;
    ic[k] += is[k];
    if (ic[k] < -INFTY) ic[k] = -INFTY;
  }
}
```

+ up to 30% speedup

SPEC 2006 — hmmer — fast_algorithms.c

```
for (k = 1; k <= M; k++) {
  mc[k] = mpp[k - 1] + tpmm[k - 1];
  if ((sc = ip[k - 1] + tpim[k - 1]) > mc[k]) mc[k] = sc;
  if ((sc = dpp[k - 1] + tpdm[k - 1]) > mc[k]) mc[k] = sc;
  if ((sc = xmb + bp[k]) > mc[k]) mc[k] = sc;
  mc[k] += ms[k];
  if (mc[k] < -INFTY) mc[k] = -INFTY;


  dc[k] = dc[k - 1] + tpdd[k - 1];
  if ((sc = mc[k - 1] + tpmd[k - 1]) > dc[k]) dc[k] = sc;
  if (dc[k] < -INFTY) dc[k] = -INFTY;
}
#pragma clang loop vectorize(enable)
for (k = 1; k <= M; k++) {
  if (k < M) {
    ic[k] = mpp[k] + tpmi[k];
    if ((sc = ip[k] + tpii[k]) > ic[k]) ic[k] = sc;
    ic[k] += is[k];
    if (ic[k] < -INFTY) ic[k] = -INFTY;
  }
}
```

+ up to 30% speedup

SPEC 2006 — hmmer — fast_algorithms.c

```c
#pragma clang loop vectorize(enable)
for (k = 1; k <= M; k++) {
  mc[k] = mpp[k - 1] + tpmm[k - 1];
  if ((sc = ip[k - 1] + tpim[k - 1]) > mc[k]) mc[k] = sc;
  if ((sc = dpp[k - 1] + tpdm[k - 1]) > mc[k]) mc[k] = sc;
  if ((sc = xmb + bp[k]) > mc[k]) mc[k] = sc;
  mc[k] += ms[k];
  if (mc[k] < -INFTY) mc[k] = -INFTY;
}
for (k = 1; k <= M; k++) {
  dc[k] = dc[k - 1] + tpdd[k - 1];
  if ((sc = mc[k - 1] + tpmd[k - 1]) > dc[k]) dc[k] = sc;
  if (dc[k] < -INFTY) dc[k] = -INFTY;
}
#pragma clang loop vectorize(enable)
for (k = 1; k <= M; k++) {
  if (k < M) {
    ic[k] = mpp[k] + tpmi[k];
    if ((sc = ip[k] + tpii[k]) > ic[k]) ic[k] = sc;
    ic[k] += is[k];
    if (ic[k] < -INFTY) ic[k] = -INFTY;
  }
}
```

+ up to 50% speedup

SPEC 2006 — hmmer — fast_algorithms.c

SPEC 2006 — hmmer — fast_algorithms.c

1 vectorized loop $\Longrightarrow$ + up to 30% speedup

SPEC 2006 — hmmer — fast_algorithms.c

1 vectorized loop	$\Longrightarrow$	+	up to 30% speedup
2 vectorized loops	$\Longrightarrow$	+	up to 50% speedup

SPEC 2006 — hmmer — fast_algorithms.c

1 vectorized loop	$\Longrightarrow$	+	up to 30% speedup
2 vectorized loops	$\Longrightarrow$	+	up to 50% speedup
possible aliasing	$\Longrightarrow$	-	runtime alias checks

SPEC 2006 — hmmer — fast_algorithms.c

1 vectorized loop	$\Longrightarrow$	+	up to 30% speedup
2 vectorized loops	$\Longrightarrow$	+	up to 50% speedup
possible aliasing	$\Longrightarrow$	-	runtime alias checks
possible dependences	$\Longrightarrow$	-	static dependence analysis