Analyzing and Optimizing your Loops with Polly
Tobias Grosser, Johannes Doerfert, Zino Benaissa
ETH Zurich — Saarland University — Quic Inc.

EuroLLVM 2016, 17. March, Barcelona
Program

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

State of Variables

Statement Instances Executed

- \( S(4,0), S(4,1), S(4,2), S(4,3), S(4,4) \)
- \( S(3,0), S(3,1), S(3,2) \)
- \( S(2,0), S(2,1) \)
- \( S(1,0), S(1,1) \)
- \( S(0,0) \)
Program

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

State of Variables
n = 4, i = 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(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 = 1, j = 0

Statement Instances Executed
S(1,0),
S(0,0)
Program

\[
\text{for } (i = 0; i <= n; i++) \\
\quad \text{for } (j = 0; j <= i; j++) \\
\quad \quad S(i,j);
\]

State of Variables
\(n = 4, i = 1, j = 1\)

Statement Instances Executed
\[S(4,0), S(4,1), S(4,2), S(4,3), S(4,4)\]
\[S(3,0), S(3,1), S(3,2)\]
\[S(2,0), S(2,1)\]
\[S(1,0), S(1,1)\]
\[S(0,0)\]
Program

```c
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

- $S(4,0)$, $S(4,1)$, $S(4,2)$, $S(4,3)$, $S(4,4)$
- $S(3,0)$, $S(3,1)$, $S(3,2)$
- $S(2,0)$, $S(2,1)$
- $S(1,0)$, $S(1,1)$
- $S(0,0)$
Program

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

State of Variables
n = 4, i = 2, j = 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(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(4,0), S(4,1), S(4,2), S(4,3), S(4,4)
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 = 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

\[
\begin{align*}
&\text{for} \ (i = 0; \ i \leq n; \ i++) \\
&\quad \text{for} \ (j = 0; \ j \leq i; \ j++) \\
&\quad \quad S(i,j);
\end{align*}
\]

State of Variables
\[n = 4, \ i = 3, \ j = 3\]

Statement Instances Executed

\[
\begin{align*}
S(3,0), & \quad S(3,1), \quad S(3,2), \quad S(3,3) \\
S(2,0), & \quad S(2,1), \quad S(2,2) \\
S(1,0), & \quad S(1,1) \\
S(0,0)
\end{align*}
\]
Program

```plaintext
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

```c
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

\[ \{ S(i,j) \mid 0 \leq i \leq n \land 0 \leq j \leq i \} \]
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**

```plaintext
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

```c
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

\[
\begin{align*}
\text{for} & \ (i = 0; i \leq n; i++) \\
& \ \text{for} \ (j = 0; j \leq i; j++) \\
& \ \quad S(i,j);
\end{align*}
\]

State of Variables

\(n = 4, \ i = 4, \ j = 4\)

Iteration space

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

\[
\text{for (i = 0; i <= n; i++)}
\begin{align*}
\text{for (j = 0; j <= i; j++)} \\
S(i,j); \\
\end{align*}
\]

State of Variables
\(n = 4, i = 1, j = 0\)

Iteration space

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**

```c
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**
  - $S(4,0)$, $S(4,1)$, $S(4,2)$, $S(4,3)$, $S(4,4)$
  - $S(3,0)$, $S(3,1)$, $S(3,2)$, $S(3,3)$
  - $S(2,0)$, $S(2,1)$, $S(2,2)$
  - $S(1,0)$, $S(1,1)$
  - $S(0,0)$

- Diagram showing the iteration space with points marked for $i = 1$ and $j = 1$. The points are located at $(0,0)$, $(1,0)$, and $(1,1)$. The diagram also shows the range of $i$ and $j$ values.
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(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);

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);

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);

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);
```

State of Variables

\( n = 4, i = 3, j = 1 \)

Statement Instances Executed

\[ S(4,0), S(4,1), S(4,2), S(4,3), S(4,4) \]
\[ S(3,0), \textbf{S(3,1)}, S(3,2), S(3,3) \]
\[ S(2,0), S(2,1), S(2,2) \]
\[ S(1,0), S(1,1) \]
\[ S(0,0) \]
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);

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);

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);

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**

```c
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**

\[ S(4,0), \ S(4,1), \ S(4,2), \ S(4,3), \ S(4,4) \]
\[ S(3,0), \ S(3,1), \ S(3,2), \ S(3,3) \]
\[ S(2,0), \ S(2,1), \ S(2,2) \]
\[ S(1,0), \ S(1,1) \]
\[ S(0,0) \]
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

```c
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

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

```c
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**

\[
\begin{align*}
S(4,0), \ S(4,1), \ S(4,2), \ S(4,3), \ S(4,4) \\
S(3,0), \ S(3,1), \ S(3,2), \ S(3,3) \\
S(2,0), \ S(2,1), \ S(2,2) \\
S(1,0), \ S(1,1) \\
S(0,0)
\end{align*}
\]

\[
= \{ S(i,j) \mid 0 \leq i \leq n \land 0 \leq j \leq i \}
\]
Schedule: Original

Model

\[ \mathcal{I}_S = \{ S(i, j) \mid 0 \leq i \leq n \land 0 \leq j \leq i \} \]

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

Code

```cpp
for (i = 0; i <= n; i++)
    for (j = 0; j <= i; j++)
        S(i, j);
```
Schedule: Original

Model

\[ \mathcal{I}_S = \{ S(i, j) \mid 0 \leq i \leq n \land 0 \leq j \leq i \} \]
\[ \Theta_S = \{ S(i, j) \rightarrow (i, j) \} \]

Code

```c
for (c0 = 0; c0 <= n; c0++)
    for (c1 = 0; c1 <= c0; c1++)
        S(c0, c1);
```
Schedule: Interchanged

Model

\[ \mathcal{I}_S = \{ S(i, j) \mid 0 \leq i \leq n \land 0 \leq j \leq i \}\]

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

Code

```c
for (c0 = 0; c0 <= n; c0 += 1)
    for (c1 = c0; c1 <= n; c1 += 1)
        S(c1, c0);
```
Schedule: Strip-Mined

Model

\[ I_S = \{ S(i, j) | 0 \leq i \leq n \land 0 \leq j \leq i \} \]
\[ \Theta_S = \{ S(i, j) \rightarrow ([i/4], j, i \mod 4) \} \]

Code

```c
for (c0 = 0; c0 <= floor(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

\[ I_S = \{ S(i,j) | 0 \leq i \leq n \land 0 \leq j \leq i \} \]

\[ \Theta_S = \{ S(i,j) \rightarrow (\lfloor i/4 \rfloor, \lfloor j/4 \rfloor, i \text{ mod } 4, j \text{ mod } 4) \} \]

Code

```
// Tiles
for (c0 = 0; c0 <= floor(d(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); c3 += 1)
```
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

```c
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
Optimizing with Polly

```c
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
```
Optimizing with Polly

```c
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

1 float sideeffect(float);

2 void foo(long T, long N, float A[][N]) {

3     for (long t = 0; t < T; t++)

4     for (long i = 1; i < N - 1; i++)

5         A[t+1][i] += sideeffect(A[t][i+1] + A[t][i-1]);

6 }

$polly$-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**

- **polly-view-all**
The Polyhedral Representation (-debug-only=polly-scops)

```c
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 :=

[T] -> { Stmt_for_body4[i0, i1] -> MemRef_A[i0, 2 + i1] }

[T] -> { Stmt_for_body4[i0, i1] -> MemRef_A[i0, i1] }

[T] -> { Stmt_for_body4[i0, i1] -> MemRef_A[1 + i0, 1 + i1] }

MustWriteAccess :=

[T] -> { Stmt_for_body4[i0, i1] -> MemRef_A[1 + i0, 1 + i1] }
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];

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

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

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

**Arrays**
- Multi-dimensionality:
  ```c
  A[ ][n][m] / A[ ][10][100]
  ```

**Keywords:**
- restrict

**Calls**
- Memory intrinsics:
  ```c
  memset/memmove/memcpy
  ```

**Approximated behaviour:**
- read-none/read-only/pointer-arguments-only
Supported constructs

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

Conditions
- Presburger Conditions
  ```c
  if (5*i+b <= 13 12 > b)
  ```
- Data-dependent
  ```c
  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
  ```c
  for (i=0; i < n / 13; i+=2)
  ```
- Presburger Expressions
  ```c
  for (i=0; i<22 && i>n; i+=2)
  ```
- Multiple back-edges/exit-edges
  ```c
  break; continue;
  ```
- do..while, while

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

```c
int i = 0;

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

**pointer loop**

```c
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

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

⇒

Scop Graph for 'nonaffine' function

```
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
```
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

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

- Polly
  - -polly-position=before-vectorizer
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_THREADS=<num> or (-mllvm) -polly-num-threads=<num>
Optimistic Assumption Tracking
Optimistic Assumption Tracking
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++)
}
```

Clearly beneficial loop interchange

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

⇒ 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++)
}
```

⇒ 15s

⇓

Clearly beneficial loop interchange

⇓

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

⇒ 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++)
}
```
... 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++)
}
```

- 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

```c
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

```c
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];
}
```

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

Possibly Invariant Loads
Possibly Invariant Loads

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

⇒

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

```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)
}
```
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);
}
```

```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 {
    /* .. */
}
```
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];
}

⇒
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 */
}
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 II

```c
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];
}
```

⇒

```c
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

LLVM IR → PollyCanonicalize
runOnFunction(...) → LLVM IR
Polly Implementation

LLVM IR \rightarrow PollyCanonicalize(runOnFunction(...)) \rightarrow LLVM IR

ScopDetection

isValidRegion(...)
isValidBlock(...)
isValidInstruction(...)
isValidTerminator(...)
...
Polly Implementation

LLVM IR → PollyCanonicalize → runOnFunction(...) → LLVM IR

ScopInfo
- buildScop(…)
- buildStatements(…)
- buildAccesses(…)
- ...

Detection Context

ScopDetection
- isValidRegion(…)
- isValidBlock(…)
- isValidInstruction(…)
- isValidTerminator(…)
- ...

38 / 43
SCoP Representation

ScopInfo.h/cpp

Scop

addAssumption(...)
getAssumedContext(...)
getMaxLoopDepth(...)
getScheduleTree(...)
getParameters(...)
getWrites(...)
getReads(...)
getStatementFor(...)
getSize(...)

...
SCoP Representation

ScopInfo.h/cpp

Scop

- addAssumption(…)
- getAssumedContext(…)
- getMaxLoopDepth(…)
- getScheduleTree(…)
- getParameters(…)
- getWrites(…)
- getReads(…)
- getStatementFor(…)
- getSize(…)
  ...

0..n

ScopStmt

- getSchedule(…)
- getBasicBlock(…)
- getDomain(…)
  ...

SCoP Representation

ScopInfo.h/cpp

Scop

- addAssumption
- getAssumedContext
- getMaxLoopDepth
- getScheduleTree
- getParameters
- getWrites
- getReads
- getStatementFor
- getSize

... 0..n

MemoryAccess

- setNewAccessRelation
- getAccessRelation
- getScopArrayInfo
- getStride
- isAffine
- isWrite
- isRead

... 0..n

ScopStmt

- getSchedule
- getBasicBlock
- getDomain

...
SCoP Representation

`ScopInfo.h/cpp`

- **ScopArrayInfo**
  - `getNumDimensions(...)`
  - `getDimensionSize(...)`
  - `getBasePointer(...)`
  - ...

- **Scop**
  - `addAssumption(...)`
  - `getAssumedContext(...)`
  - `getMaxLoopDepth(...)`
  - `getScheduleTree(...)`
  - `getParameters(...)`
  - `getWrites(...)`
  - `getReads(...)`
  - `getStatementFor(...)`
  - `getSize(...)`
  - ...

- **MemoryAccess**
  - `setNewAccessRelation(...)`
  - `getAccessRelation(...)`
  - `getScopArrayInfo(...)`
  - `getStride(...)`
  - `isAffine(...)`
  - `isWrite(...)`
  - `isRead(...)`
  - ...

- **ScopStmt**
  - `getSchedule(...)`
  - `getBasicBlock(...)`
  - `getDomain(...)`
  - ...

Relationships:
- `ScopArrayInfo` to `Scop`: 1
- `MemoryAccess` to `Scop`: 0..n
- `ScopStmt` to `Scop`: 0..n
Polly Implementation
Polly Implementation

- ScopInfo
- Scop
- DependenceInfo
  - runOnScop(...)
  - ...
- Dependences

- CodeGeneration
  - runOnScop(...)
Polly Implementation

- **ScopInfo**
- **Scop**
  - **DependenceInfo**
    - `runOnScop(...)`
    - `...`
  - **Dependences**
- **IslAst**
  - **IslAstInfo**
    - `runOnScop(...)`
    - `getRunCondition(...)`
    - `...`
- **CodeGen**
  - **CodeGenInfo**
    - `runOnScop(...)`
- **LLVM IR**
Polly Implementation

ScopInfo -> Scop -> DependenceInfo
runOnScop(...) ...

Dependences -> ScheduleOptimizer
runOnScop(...) ...

IsiAstInfo
runOnScop(...) getRunCondition(...) ...

IslAst -> Scop

CodeGenation
runOnScop(...)

LLVM IR
Polly Implementation
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;
            }
        }
    }
}
```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]);
        }
    }
}
```
\[
\begin{align*}
\text{rhs}[k][j][i][1] &= \text{rhs}[k][j][i][1] + dx2tx1 \cdot \\
& \quad (u[k][j][i+1][1] - 2.0 \cdot u[k][j][i][1] + \\
& \quad u[k][j][i-1][1]) + \\
& \quad xxcon2 \cdot con43 \cdot (up1 - 2.0 \cdot uijk + um1) - \\
& \quad tx2 \cdot (u[k][j][i+1][1] \cdot up1 - \\
& \quad u[k][j][i-1][1] \cdot um1 + \\
& \quad (u[k][j][i+1][4] - \text{square}[k][j][i+1] - \\
& \quad u[k][j][i-1][4] + \text{square}[k][j][i-1]) \cdot c2) \\
\text{rhs}[k][j][i][2] &= \text{rhs}[k][j][i][2] + dx3tx1 \cdot \\
& \quad (u[k][j][i+1][2] - 2.0 \cdot u[k][j][i][2] + \\
& \quad u[k][j][i-1][2]) + \\
& \quad xxcon2 \cdot (vs[k][j][i+1] - 2.0 \cdot vs[k][j][i] + \\
& \quad vs[k][j][i-1]) - \\
& \quad tx2 \cdot (u[k][j][i+1][2] \cdot up1 - u[k][j][i-1][2] \cdot um1) \\
\text{rhs}[k][j][i][3] &= \text{rhs}[k][j][i][3] + dx4tx1 \cdot \\
& \quad (u[k][j][i+1][3] - 2.0 \cdot u[k][j][i][3] + \\
& \quad u[k][j][i-1][3]) + \\
& \quad xxcon2 \cdot (ws[k][j][i+1] - 2.0 \cdot ws[k][j][i] + \\
& \quad ws[k][j][i-1]) - \\
& \quad tx2 \cdot (u[k][j][i+1][3] \cdot up1 - u[k][j][i-1][3] \cdot um1) \\
/* \approx 300 \text{ more lines of similar code } */
\end{align*}
\]
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++) {
            /* ... */

\textsuperscript{a}Sanyam and Yew, PLDI 15
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++) {
   /* ... */

+ \text{6× speedup for 8 threads/cores}^a

\hspace{1cm}^a\text{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 \(^\text{a}\)
- Possible variant loop bounds

\(^a\)Sanyam and Yew, PLDI 15
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

*aSanyam and Yew, PLDI 15*
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
- Possible variant loop bounds
- Possible out-of-bound accesses
- Possible execution of non-pure calls

---

\(^a\)Sanyam and Yew, PLDI 15
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

---

aSanyam 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’, timeron] -> 
          { : timeron = 0 } [-Rpass-analysis=polly-scops]

  if (timeron) timer_start(t_rhsx);
          ^

```c
```
NAS Parallel Benchmarks — BT — rhs.c

```c
rho_inv = 1.0/u[k][j][i][0];
```

```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]
```

```c
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]
```

```c
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) } 
```

```c
rhs[k][j][i][m] = rhs[k][j][i][m] - dssp *
```

```c
40 / 43
```
NAS Parallel Benchmarks — BT — rhs.c

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

rhs.c:419:1: remark: No-overflow 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) or
(grid_points <= 2 and grid_points >= -2147483644 and grid_points’ <= 2 and grid_points’’ >= -2147483646) or
(grid_points’ = -2147483644 and grid_points >= 3 and grid_points’’ <= 2 and grid_points’’ >= -2147483646) or
(grid_points = -2147483644 and grid_points’ >= 3 and grid_points’’ <= 2 and grid_points’’ >= -2147483646)

__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] +
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;
}
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], rate[i],
                                            volatility[i], otime[i], otype[i],
                                            tdir[i], option_type);

    return 0;
}
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], rate[i],
                                            volatility[i], otime[i], otype[i]);

    return 0;
}

+ 2.9× speedup for manual parallelization on a quad-core i7
```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], rate[i],
                                            volatility[i], otime[i], otype[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
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], rate[i],
                                            volatility[i], otime[i], otype[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

```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], rate[i],
                                            volatility[i], otime[i], otype[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
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], rate[i],
                                             volatility[i], otime[i], otype[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)
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], rate[i], volatility[i], otime[i], otype[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!
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;
    }
}
#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;
}
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;
}
#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;
    }
}
#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
1 vectorized loop ⇒ + up to 30% speedup

2 vectorized loops ⇒ + up to 50% speedup

possible aliasing ⇒ - runtime alias checks

possible dependences ⇒ - static dependence analysis
1 vectorized loop $\rightarrow$ + up to 30% speedup
1 vectorized loop $\implies +$ up to 30% speedup
2 vectorized loops $\implies +$ up to 50% speedup
1 vectorized loop  \implies \quad + \quad \text{up to 30% speedup}
2 vectorized loops  \implies \quad + \quad \text{up to 50% speedup}
possible aliasing  \implies \quad - \quad \text{runtime alias checks}
1 vectorized loop  \implies + \quad \text{up to 30\% speedup}
2 vectorized loops  \implies + \quad \text{up to 50\% speedup}
possible aliasing  \implies - \quad \text{runtime alias checks}
possible dependences  \implies - \quad \text{static dependence analysis}