SPolly: Speculative Optimizations in the Polyhedral Model

Johannes Doerfert, Clemens Hammacher, Kevin Streit, Sebastian Hack

Saarland University, Germany

January 21, 2013
int A[256][256], B[256][256], C[256][256];

void matmul() {
    for (int i=0; i<256; i++)
        for (int j=0; j<256; j++)
            for (int k=0; k<256; k++)
                C[i][j] += A[k][i] * B[j][k];
}
The Problem

```c
int A[65536], B[65536], C[65536];

void matmul() {
    for (int i=0; i<256; i++)
        for (int j=0; j<256; j++)
            for (int k=0; k<256; k++)
                C[i*256+j] += A[k*256+i] * B[j*256+k];
}
```
The Problem

```c
void matmul(int* A, int* B, int* C) {
    for (int i=0; i<256; i++)
        for (int j=0; j<256; j++)
            for (int k=0; k<256; k++)
                C[i*256+j] += A[k*256+i] * B[j*256+k];
}
```
The Problem

```c
void matmul(int* A, int* B, int* C, int N) {
    for (int i=0; i<N; i++)
        for (int j=0; j<N; j++)
            for (int k=0; k<N; k++)
                C[i*N+j] += A[k*N+i] * B[j*N+k];
}
```
How urgent is this problem?
How urgent is this problem?

85.2%

14.8%

Valid Regions

Invalid Regions
How urgent is this problem?

Setup

- **Polly**
  - state-of-the-art polyhedral optimizer integrated in LLVM

- **SPEC 2000**
  - industry standard benchmark suite
  - nine real world programs:
    - ammp, art, bzip2, crafty, equake, gzip, mcf, mesa, twolf
How urgent is this problem?

Setup

- Polly
  - state-of-the-art polyhedral optimizer integrated in LLVM

- SPEC 2000
  - industry standard benchmark suite
  - nine real world programs:
    ammp, art, bzip2, crafty, equake, gzip, mcf, mesa, twolf

- Research questions
  - number of Static Control Parts
    (SCoPs := code regions amenable to polyhedral optimizations)
  - impact of individual rejection causes
How urgent is this problem?

SCoP rejection causes found in 1862 regions

<table>
<thead>
<tr>
<th>$i$</th>
<th>Rejection cause</th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Non-affine expressions</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Aliasing</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Non-affine loop bounds</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Function call</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Non-canonical indvars</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Complex CFG</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Unsigned comparison</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>Others</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
How urgent is this problem?
SCoS rejection causes found in 1862 regions

<table>
<thead>
<tr>
<th>$i$</th>
<th>Rejection cause</th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Non-affine expressions</td>
<td></td>
<td>1230 (66%)</td>
<td>207 (11%)</td>
</tr>
<tr>
<td>1</td>
<td>Aliasing</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Non-affine loop bounds</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Function call</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Non-canonical indvars</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Complex CFG</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Unsigned comparison</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>Others</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

$$\text{for } (i = 0; i < N; i++)$$

$$A[i*N] += B[i];$$
How urgent is this problem?
SCoP rejection causes found in 1862 regions

<table>
<thead>
<tr>
<th>$i$</th>
<th>Rejection cause</th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Non-affine expressions</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Aliasing</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Non-affine loop bounds</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Function call</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Non-canonical indvars</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Complex CFG</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Unsigned comparison</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>Others</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

```
void f(int* A, int* B){
  A[0] = B[5];
}
```
# How urgent is this problem?

**SCoP rejection causes found in 1862 regions**

<table>
<thead>
<tr>
<th>$i$</th>
<th>Rejection cause</th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Non-affine expressions</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Aliasing</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Non-affine loop bounds</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Function call</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Non-canonical indvars</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Complex CFG</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Unsigned comparison</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>Others</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

for (i = 0; i < N*M; i++)

    A[i] += B[i];
How urgent is this problem?

SCoP rejection causes found in 1862 regions

<table>
<thead>
<tr>
<th>i</th>
<th>Rejection cause</th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Non-affine expressions</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Aliasing</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Non-affine loop bounds</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Function call</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Non-canonical indvars</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Complex CFG</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Unsigned comparison</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>Others</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

for (i = 0; i < N; i++)

A[i] += g(i);
How urgent is this problem?
SCoP rejection causes found in 1862 regions

<table>
<thead>
<tr>
<th>$i$</th>
<th>Rejection cause</th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Non-affine expressions</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Aliasing</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Non-affine loop bounds</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Function call</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Non-canonical indvars</td>
<td>A</td>
<td>B</td>
<td>C</td>
</tr>
<tr>
<td>5</td>
<td>Complex CFG</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Unsigned comparison</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>Others</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

for (i=0; i<N; i+=i/2+1)
A[i] += A[i+1];
How urgent is this problem?
SCoP rejection causes found in 1862 regions

<table>
<thead>
<tr>
<th>$i$</th>
<th>Rejection cause</th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Non-affine expressions</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Aliasing</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Non-affine loop bounds</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Function call</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Non-canonical indvars</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Complex CFG</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Unsigned comparison</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>Others</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
How urgent is this problem?

SCoP rejection causes found in 1862 regions

<table>
<thead>
<tr>
<th>i</th>
<th>Rejection cause</th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Non-affine expressions</td>
<td>1230 (66%)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Aliasing</td>
<td>1093 (59%)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Non-affine loop bounds</td>
<td>840 (45%)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Function call</td>
<td>532 (29%)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Non-canonical indvars</td>
<td>384 (21%)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Complex CFG</td>
<td>253 (14%)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Unsigned comparison</td>
<td>199 (11%)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>Others</td>
<td>1 (0%)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

A #regions where condition $i$ is violated.
How urgent is this problem?
SCoP rejection causes found in 1862 regions

<table>
<thead>
<tr>
<th>$i$</th>
<th>Rejection cause</th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Non-affine expressions</td>
<td>1230 (66%)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Aliasing</td>
<td>1093 (59%)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Non-affine loop bounds</td>
<td>840 (45%)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Function call</td>
<td>532 (29%)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Non-canonical indvars</td>
<td>384 (21%)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Complex CFG</td>
<td>253 (14%)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Unsigned comparison</td>
<td>199 (11%)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>Others</td>
<td>1 ( 0%)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

A #regions where condition $i$ is violated.
How urgent is this problem?

SCoP rejection causes found in 1862 regions

<table>
<thead>
<tr>
<th>i</th>
<th>Rejection cause</th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Non-affine expressions</td>
<td>1230 (66%)</td>
<td>84</td>
<td>(5%)</td>
</tr>
<tr>
<td>1</td>
<td>Aliasing</td>
<td>1093 (59%)</td>
<td>207</td>
<td>(11%)</td>
</tr>
<tr>
<td>2</td>
<td>Non-affine loop bounds</td>
<td>840 (45%)</td>
<td>6</td>
<td>(0%)</td>
</tr>
<tr>
<td>3</td>
<td>Function call</td>
<td>532 (29%)</td>
<td>72</td>
<td>(4%)</td>
</tr>
<tr>
<td>4</td>
<td>Non-canonical indvars</td>
<td>384 (21%)</td>
<td>0</td>
<td>(0%)</td>
</tr>
<tr>
<td>5</td>
<td>Complex CFG</td>
<td>253 (14%)</td>
<td>31</td>
<td>(2%)</td>
</tr>
<tr>
<td>6</td>
<td>Unsigned comparison</td>
<td>199 (11%)</td>
<td>0</td>
<td>(0%)</td>
</tr>
<tr>
<td>7</td>
<td>Others</td>
<td>1 (0%)</td>
<td>0</td>
<td>(0%)</td>
</tr>
</tbody>
</table>

**A** #regions where condition $i$ is violated.

**B** #regions where *only* condition $i$ is violated.
How urgent is this problem?

SCoP rejection causes found in 1862 regions

<table>
<thead>
<tr>
<th>$i$</th>
<th>Rejection cause</th>
<th>A (regions)</th>
<th>B (regions)</th>
<th>C (regions)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Non-affine expressions</td>
<td>1230 (66%)</td>
<td>84 (5%)</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Aliasing</td>
<td>1093 (59%)</td>
<td>207 (11%)</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Non-affine loop bounds</td>
<td>840 (45%)</td>
<td>6 (0%)</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Function call</td>
<td>532 (29%)</td>
<td>72 (4%)</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Non-canonical indvars</td>
<td>384 (21%)</td>
<td>0 (0%)</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Complex CFG</td>
<td>253 (14%)</td>
<td>31 (2%)</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Unsigned comparison</td>
<td>199 (11%)</td>
<td>0 (0%)</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>Others</td>
<td>1 (0%)</td>
<td>0 (0%)</td>
<td></td>
</tr>
</tbody>
</table>

A  #regions where condition $i$ is violated.

B  #regions where only condition $i$ is violated.
How urgent is this problem?

SCoP rejection causes found in 1862 regions

<table>
<thead>
<tr>
<th>$i$</th>
<th>Rejection cause</th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Non-affine expressions</td>
<td>1230 (66%)</td>
<td>84 ( 5%)</td>
<td>84 ( 5%)</td>
</tr>
<tr>
<td>1</td>
<td>Aliasing</td>
<td>1093 (59%)</td>
<td>207 (11%)</td>
<td>510 (27%)</td>
</tr>
<tr>
<td>2</td>
<td>Non-affine loop bounds</td>
<td>840 (45%)</td>
<td>6 ( 0%)</td>
<td>660 (35%)</td>
</tr>
<tr>
<td>3</td>
<td>Function call</td>
<td>532 (29%)</td>
<td>72 ( 4%)</td>
<td>928 (50%)</td>
</tr>
<tr>
<td>4</td>
<td>Non-canonical indvars</td>
<td>384 (21%)</td>
<td>0 ( 0%)</td>
<td>1174 (63%)</td>
</tr>
<tr>
<td>5</td>
<td>Complex CFG</td>
<td>253 (14%)</td>
<td>31 ( 2%)</td>
<td>1387 (74%)</td>
</tr>
<tr>
<td>6</td>
<td>Unsigned comparison</td>
<td>199 (11%)</td>
<td>0 ( 0%)</td>
<td>1586 (85%)</td>
</tr>
<tr>
<td>7</td>
<td>Others</td>
<td>1 ( 0%)</td>
<td>0 ( 0%)</td>
<td>1587 (85%)</td>
</tr>
</tbody>
</table>

A  #regions where condition $i$ is violated.
B  #regions where only condition $i$ is violated.
C  #regions where only conditions 0 to $i$ are violated.
How urgent is this problem?

SCoP rejection causes found in 1862 regions

<table>
<thead>
<tr>
<th>$i$</th>
<th>Rejection cause</th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Non-affine expressions</td>
<td>1230 (66%)</td>
<td>84 (5%)</td>
<td>84 (5%)</td>
</tr>
<tr>
<td>1</td>
<td>Aliasing</td>
<td>1093 (59%)</td>
<td>207 (11%)</td>
<td>510 (27%)</td>
</tr>
<tr>
<td>2</td>
<td>Non-affine loop bounds</td>
<td>840 (45%)</td>
<td>6 (0%)</td>
<td>660 (35%)</td>
</tr>
<tr>
<td>3</td>
<td>Function call</td>
<td>532 (29%)</td>
<td>72 (4%)</td>
<td>928 (50%)</td>
</tr>
<tr>
<td>4</td>
<td>Non-canonical indvars</td>
<td>384 (21%)</td>
<td>0 (0%)</td>
<td>1174 (63%)</td>
</tr>
<tr>
<td>5</td>
<td>Complex CFG</td>
<td>253 (14%)</td>
<td>31 (2%)</td>
<td>1387 (74%)</td>
</tr>
<tr>
<td>6</td>
<td>Unsigned comparison</td>
<td>199 (11%)</td>
<td>0 (0%)</td>
<td>1586 (85%)</td>
</tr>
<tr>
<td>7</td>
<td>Others</td>
<td>1 (0%)</td>
<td>0 (0%)</td>
<td>1587 (85%)</td>
</tr>
</tbody>
</table>

A  #regions where condition $i$ is violated.
B  #regions where only condition $i$ is violated.
C  #regions where only conditions 0 to $i$ are violated.
How urgent is this problem?

SCoP rejection causes found in 1862 regions

<table>
<thead>
<tr>
<th>$i$</th>
<th>Rejection cause</th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Non-affine expressions</td>
<td>1230 (66%)</td>
<td>84 ( 5%)</td>
<td>84 ( 5%)</td>
</tr>
<tr>
<td>1</td>
<td>Aliasing</td>
<td>1093 (59%)</td>
<td>207 (11%)</td>
<td>510 (27%)</td>
</tr>
<tr>
<td>2</td>
<td>Non-affine loop bounds</td>
<td>840 (45%)</td>
<td>6 ( 0%)</td>
<td>660 (35%)</td>
</tr>
<tr>
<td>3</td>
<td>Function call</td>
<td>532 (29%)</td>
<td>72 ( 4%)</td>
<td>928 (50%)</td>
</tr>
<tr>
<td>4</td>
<td>Non-canonical indvars</td>
<td>384 (21%)</td>
<td>0 ( 0%)</td>
<td>1174 (63%)</td>
</tr>
<tr>
<td>5</td>
<td>Complex CFG</td>
<td>253 (14%)</td>
<td>31 ( 2%)</td>
<td>1387 (74%)</td>
</tr>
<tr>
<td>6</td>
<td>Unsigned comparison</td>
<td>199 (11%)</td>
<td>0 ( 0%)</td>
<td>1586 (85%)</td>
</tr>
<tr>
<td>7</td>
<td>Others</td>
<td>1 ( 0%)</td>
<td>0 ( 0%)</td>
<td>1587 (85%)</td>
</tr>
</tbody>
</table>

A  #regions where condition $i$ is violated.
B  #regions where only condition $i$ is violated.
C  #regions where only conditions 0 to $i$ are violated.
How urgent is this problem?

Conclusion

![Pie chart showing percentages of valid, targeted, and invalid regions.]

- Valid Regions: 49.8%
- Targeted Regions: 35.4%
- Invalid Regions: 14.8%
How to allow more polyhedral optimizations?

Example

```c
void f(int* A, int* B) {
    for (int i=0; i < 2048; i++)
        A[i] += B[i];
}
```
How to allow more polyhedral optimizations?

Example

1. **speculatively** assume properties (e.g., constant parameters)

```c
void f(int* A, int* B) {
    for (int i=0; i < 2048; i++)
        A[i] += B[i];
}
```
How to allow more polyhedral optimizations?

Example

1. **speculatively** assume properties (e.g., constant parameters)
2. derive **specialized** versions

```c
void f_spec(int* restrict A, int* restrict B) {
    for (int i=0; i < 2048; i++)
        A[i] += B[i];
}
```
How to allow more polyhedral optimizations?

Example

1. **speculatively** assume properties (e.g., constant parameters)
2. derive **specialized** versions
3. apply **polyhedral optimizations**

```c
void f_opt(int* restrict A, int* restrict B) {
    parfor (int j=0; j < 2048; j+=32)
        for (int i=j; i < 32 + j; i++)
            A[i] += B[i];
}
```
How to allow more polyhedral optimizations?

Example

1. **speculatively** assume properties (e.g., constant parameters)
2. derive **specialized** versions
3. apply **polyhedral optimizations**
4. add **runtime dispatcher**

```c
void f_dispatcher(int* A, int* B) {
    if (overlap(A, B, 2048))
        f(A, B);
    else
        f_opt(A, B);
}
```
How to allow more polyhedral optimizations?

Implementation
How to allow more polyhedral optimizations?

Implementation

LLVM-IR

SCoP Detection

Valid SCoPs

Polyhedral Optimizations

Code Generation

Program
How to allow more polyhedral optimizations?

Implementation

- LLVM-IR
- SCoP Detection
- Polyhedral Optimizations
- Code Generation
- Program

- Polly
- SPolly

Valid SCoPs

Invalid SCoPs
How to allow more polyhedral optimizations?

Implementation

```
LLVM-IR

SCoP Detection

Polyhedral Optimizations

Code Generation

Program

Invalid SCoPs

Valid SCoPs

Polly

SPolly

SCoP Detection

sSCoP Detection

Region Speculation

Valid sSCoPs
```
How to allow more polyhedral optimizations?

Implementation

LLVM-IR

SCoP Detection

Polyhedral Optimizations

Code Generation

Program

Valid SCoPs

Invalid SCoPs

Valid sSCoPs

Polly

SPolly

sSCoP Detection

Region Speculation

Specialized Versions
How to allow more polyhedral optimizations?

Implementation

```
LLVM-IR

SCoP Detection
  Valid SCoPs
  Invalid SCoPs

Polyhedral Optimizations

Code Generation

Program

Polly
SPolly

sSCoP Detection
  Valid sSCoPs
  Specialized Versions

Region Speculation

Runtime Dispatcher
```
How to allow more polyhedral optimizations?

Implementation

- LLVM-IR
- SCoP Detection
  - Valid SCoPs
  - Invalid SCoPs
- Polyhedral Optimizations
- Code Generation

- sSCoP Detection
  - Specialized Versions
  - Valid sSCoPs
- Region Speculation
- Runtime Dispatcher
- Profiling Versions

Program

Valid SCoPs
Invalid SCoPs
Valid sSCoPs
Polly
SPolly
How to allow more polyhedral optimizations?

Implementation

- LLVM-IR
- SCōP Detection
  - Valid SCōPs
  - Invalid SCōPs
- Polyhedral Optimizations
  - Specialized Versions
- Region Speculation
  - Valid sSCōPs
- sSCōP Detection
- Code Generation
- Runtime Dispatcher
- Profiling Versions
- JIT-Environment
- Program

[SCōP Detection] Valid SCōPs → Polyhedral Optimizations → Code Generation → Program
[Invalid SCōPs] SCōP Detection → Polyhedral Optimizations → Region Speculation
[sSCōP Detection] Valid sSCōPs → Region Speculation

Polly
SPolly
How to allow more polyhedral optimizations?

Implementation
How to allow more polyhedral optimizations?

Implementation
How to allow more polyhedral optimizations?

Implementation

- LLVM-IR
- Valid SCoPs
- Invalid SCoPs
- Polly
- SPolly
- LLVM-IR
- SCoP Detection
- sSCoP Detection
- Polyhedral Optimizations
- Code Generation
- Runtime Dispatcher
- Profiling Versions
- Profiling Information
- JIT-Environment
- Specialized Versions
- Valid sSCoPs
- Region Speculation
- Program
Does it work?
Does it work?

Almost.
Does it work?
Applicability on SPEC 2000

- Valid SCoPs: 65.9%
- Additional sSCoPs: 19.3%
- Invalid sSCoPs: 14.8%
Does it work?

Applicability on SPEC 2000

Number of valid regions

<table>
<thead>
<tr>
<th>Application</th>
<th>Polly</th>
<th>SPolly</th>
</tr>
</thead>
<tbody>
<tr>
<td>ammp</td>
<td>22</td>
<td>50</td>
</tr>
<tr>
<td>art</td>
<td>7</td>
<td>28</td>
</tr>
<tr>
<td>bzip2</td>
<td>30</td>
<td>37</td>
</tr>
<tr>
<td>crafty</td>
<td>41</td>
<td>58</td>
</tr>
<tr>
<td>equake</td>
<td>10</td>
<td>24</td>
</tr>
<tr>
<td>gzip</td>
<td>29</td>
<td>31</td>
</tr>
<tr>
<td>mcf</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>mesa</td>
<td>106</td>
<td>283</td>
</tr>
<tr>
<td>twolf</td>
<td>29</td>
<td>123</td>
</tr>
</tbody>
</table>
Does it work?
Runtime Results on SPEC 2000
Does it work?
Runtime Results on SPEC 2000

Speedup relatively to Polly

- ammp
- art
- bzip2
- crafty
- equake
- gzip
- mcf
- mesa
- twolf

Polly crashes
SPolly crashes
SPolly crashes
Polly
SPolly

12/16
Does it work?
Runtime Results on SPEC 2000
Does it work?

Case Study – Setup

Algorithm 2D derivation computation (basic image processing block)

Inputs are given in 2 different resolutions

Evaluated speedup of SPolly normalized against Polly
Does it work?

Case Study – Results

![Graph showing speedup relatively to clang for input image sizes](image)
Does it work?

Case Study – Results

![Graph showing speedup relatively to clang for different input image sizes using Polly and SPolly. The graph compares the performance of Polly and SPolly across various image sizes, with speedup values indicated on the y-axis.]
Does it work?

Case Study – Results

[Bar chart showing speedup relatively to clang for different image sizes. The chart compares Polly and SPolly.]
Does it work?

Case Study – Results

![Graph showing speedup relatively to clang for different input image sizes using Polly and SPolly.](image-url)
Does it work?

Case Study – Results

![Graph showing speedup relatively to clang for different input image sizes with Polly and SPolly]
Does it work?

Case Study – Results

Speedup relatively to clang

<table>
<thead>
<tr>
<th>Input image size</th>
<th>Polly</th>
<th>SPolly</th>
</tr>
</thead>
<tbody>
<tr>
<td>512x4096</td>
<td></td>
<td></td>
</tr>
<tr>
<td>512x4096</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4096x512</td>
<td>1.0</td>
<td>2.5</td>
</tr>
<tr>
<td>4096x512</td>
<td>1.0</td>
<td>2.5</td>
</tr>
<tr>
<td>512x4096</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4096x512</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Does it work?
Case Study – Results

![Graph showing speedup relatively to clang for different input image sizes using Polly and SPolly.](image)
Does it work?

Runtime Results on Polybench

Summary

85.2%  14.8%
Summary

- 85.2%
- 14.8%

Diagram:
- LLVM-IR
- SCoP Detection
- Polyhedral Optimizations
- Code Generation
- Runtime Dispatcher
- Profiling Versions
- JIT-Environment
- Profiling Information
- Polly
- SPolly

List of benchmarks:
- bicg
- syrk
- jacobi-1d-imper
- trmm
- symm
- syr2k
- cholesky
- fdtd-apml
- 3mm
- lu
- doitgen
- seidel-1d
- gemm
- adi
- gesummv
- floyd-warshall
- ludcmp
- covariance
- jacobi-2d-imper
- atax
- fdtd-2d
- trisolv
- 2mm
- mvt
- gemver
- reg_detect
- correlation
- gramschmidt
- dynprog
- durbin

Speedup relatively to clang
- SPolly 1st run
- SPolly 2nd run

16/16
Summary

- **85.2%** Valid SCoPs
- **14.8%** Invalid SCoPs
- **65.9%** Valid sSCoPs
- **19.3%** Polly
- **14.8%** SPolly

Diagram showing the flow of processing from LLVM-IR to Specialized Code Generation and Profiling Information, with connections to Program and Runtime Dispatcher.
Summary

Pie charts showing percentages.

Bar chart showing speedup relatively to clang for different benchmarks.

Diagram illustrating the flow of code generation and optimization processes.
Does it work?
Case Study – Setup continued

- Convolution kernel of size 3x3
- Applied to all channels of an RGBA image (e.g., png)
- Measured on a Intel(R) Core(TM) i5 CPU M 560

Image source:
Concept

Alias tests

```c
for (i = 0; i < N; i++) {
    for (j = 0; j < N; j++) {
        // I1
        C[i][j] = 0;
        for (k = 0; k < N; k++) {
            // I2 I3 I4
            C[i][j] += A[i][k] * B[k][j];
        }
    }
}
```

Acc bp ma Ma

I1 and I2 C 0 N ∗ N − 1

I3 A 0 N ∗ N − 1

I4 B 0 N ∗ N − 1

ma := minimal access  
Ma := maximal access  
bp := base pointer

Diagram:

```
Array A

Array B

Array C
```

Concept

Alias tests

```c
for (i = 0; i < N; i++) {
    for (j = 0; j < N; j++) {
        // I1
        C[i][j] = 0;
        for (k = 0; k < N; k++) {
            // I2 I3 I4
            C[i][j] += A[i][k] * B[k][j];
        }
    }
}
```

Acc bp ma Ma
I1 and I2 C 0 N ∗ N − 1
I3 A 0 N ∗ N − 1
I4 B 0 N ∗ N − 1

ma := minimal access
Ma := maximal access
bp := base pointer
Concept

Alias tests

```c
for (i = 0; i < N; i++) {
    for (j = 0; j < N; j++) {
        // I1
        C[i][j] = 0;
        for (k = 0; k < N; k++) {
            // I2 I3 I4
            C[i][j] += A[i][k] * B[k][j];
        }
    }
}
```

<table>
<thead>
<tr>
<th>Acc</th>
<th>bp</th>
<th>ma</th>
<th>Ma</th>
</tr>
</thead>
<tbody>
<tr>
<td>l1</td>
<td>C</td>
<td>0</td>
<td>N*N-1</td>
</tr>
<tr>
<td>l3</td>
<td>A</td>
<td>0</td>
<td>N*N-1</td>
</tr>
<tr>
<td>l4</td>
<td>B</td>
<td>0</td>
<td>N*N-1</td>
</tr>
</tbody>
</table>

ma := minimal access
Ma := maximal access
bp := base pointer
Concept

Alias tests

```c
for (i = 0; i < N; i++) {
    for (j = 0; j < N; j++) {
        // I1
        C[i][j] = 0;
        for (k = 0; k < N; k++) {
            // I2 I3 I4
            C[i][j] += A[i][k] * B[k][j];
        }
    }
}
```

<table>
<thead>
<tr>
<th>Acc</th>
<th>bp</th>
<th>ma</th>
<th>Ma</th>
</tr>
</thead>
<tbody>
<tr>
<td>l1 and l2</td>
<td>C</td>
<td>0</td>
<td>N*N−1</td>
</tr>
<tr>
<td>l3</td>
<td>A</td>
<td>0</td>
<td>N*N−1</td>
</tr>
<tr>
<td>l4</td>
<td>B</td>
<td>0</td>
<td>N*N−1</td>
</tr>
</tbody>
</table>

```c
bool bc = B[N*N−1] < C[0] || B[0] > C[N*N−1];
bool no_alias_found = ab && ac && bc;
```