SPolly: Speculative Optimizations in the Polyhedral Model

Johannes Doerfert, Clemens Hammacher,
Kevin Streit, Sebastian Hack

Saarland University, Germany

January 21, 2013

The Problem

```
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

```
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

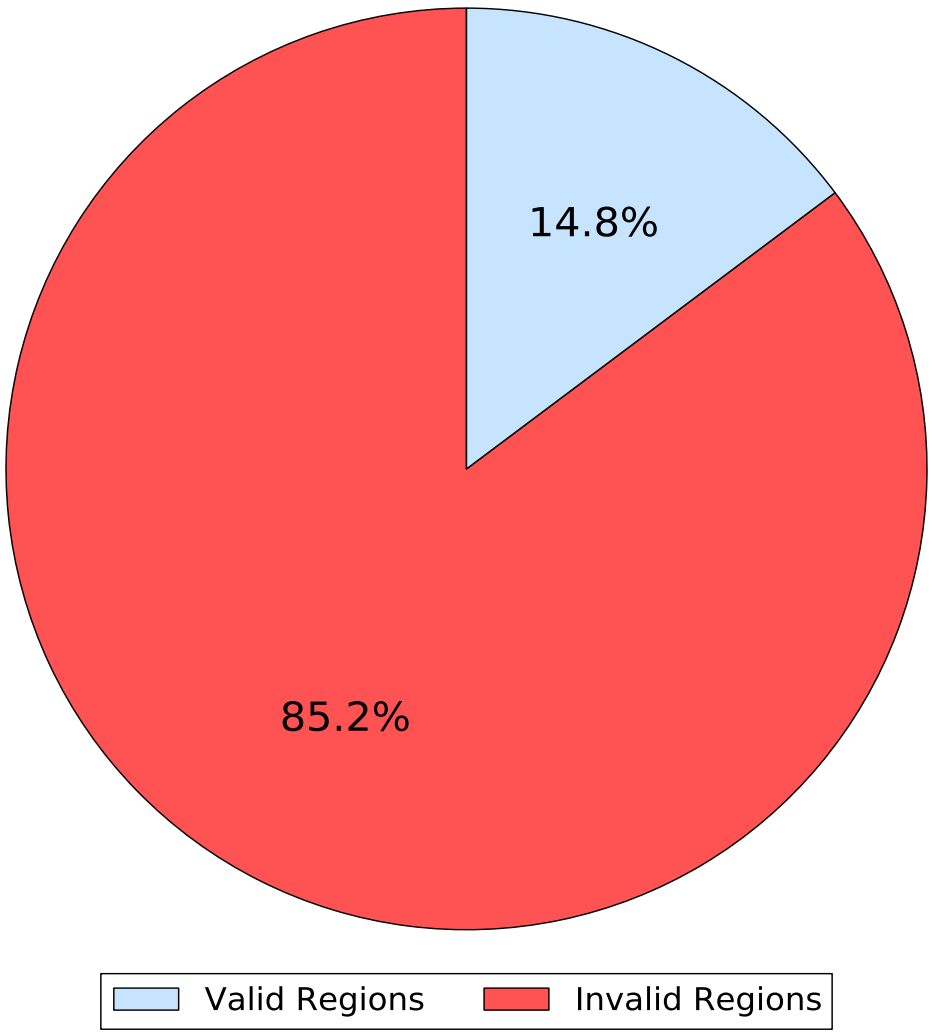
```
void matmul(int* A, int* B, int* C) {
  for (int i=0; i<256; i++)
    for (int j=0; j<256; j++)
      for (int k=0; k<256; k++)
        C[i*256+j] += A[k*256+i] * B[j*256+k];
}
```

The Problem

```
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?



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

i	Rejection cause	A	B	C
0	Non-affine expressions			
1	Aliasing			
2	Non-affine loop bounds			
3	Function call			
4	Non-canonical indvars			
5	Complex CFG			
6	Unsigned comparison			
7	Others			

How urgent is this problem?

SCoP rejection causes found in 1862 regions

i	Rejection cause	A	B	C
0	Non-affine expressions			
1	Aliasing			
2	Non-affine loop bounds			
3	Function call			
4	Non-canonical indvars			
5	Complex CFG			
6	Unsigned comparison			
7	Others			

```
for (i = 0; i < N; i++)
  A[i*N] += B[i];
```

How urgent is this problem?

SCoP rejection causes found in 1862 regions

i	Rejection cause	A	B	C
0	Non-affine expressions			
1	Aliasing			
2	Non-affine loop bounds			
3	Function call			
4	Non-canonical indvars			
5	Complex CFG			
6	Unsigned comparison			
7	Others			

```
void f(int* A, int* B){
  A[0] = B[5];
```

How urgent is this problem?

SCoP rejection causes found in 1862 regions

i	Rejection cause	A	B	C
0	Non-affine expressions			
1	Aliasing			
2	Non-affine loop bounds			
3	Function call			
4	Non-canonical indvars			
5	Complex CFG			
6	Unsigned comparison			
7	Others			

```
for (i = 0; i < N*M; i++)
  A[i] += B[i];
```

How urgent is this problem?

SCoP rejection causes found in 1862 regions

i	Rejection cause	A	B	C
0	Non-affine expressions			
1	Aliasing			
2	Non-affine loop bounds			
3	Function call			
4	Non-canonical indvars			
5	Complex CFG			
6	Unsigned comparison			
7	Others			

```
for (i = 0; i < N; i++)
  A[i] += g(i);
```

How urgent is this problem?

SCoP rejection causes found in 1862 regions

i	Rejection cause	A	B	C
0	Non-affine expressions			
1	Aliasing			
2	Non-affine loop bounds			
3	Function call			
4	Non-canonical indvars			
5	Complex CFG			
6	Unsigned comparison			
7	Others			

```
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

i	Rejection cause	A	B	C
0	Non-affine expressions			
1	Aliasing			
2	Non-affine loop bounds			
3	Function call			
4	Non-canonical indvars			
5	Complex CFG			
6	Unsigned comparison			
7	Others			

How urgent is this problem?

SCoP rejection causes found in 1862 regions

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

A #regions where condition i is violated.

How urgent is this problem?

SCoP rejection causes found in 1862 regions

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

A #regions where condition i is violated.

How urgent is this problem?

SCoP rejection causes found in 1862 regions

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

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

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

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

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

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

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

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

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

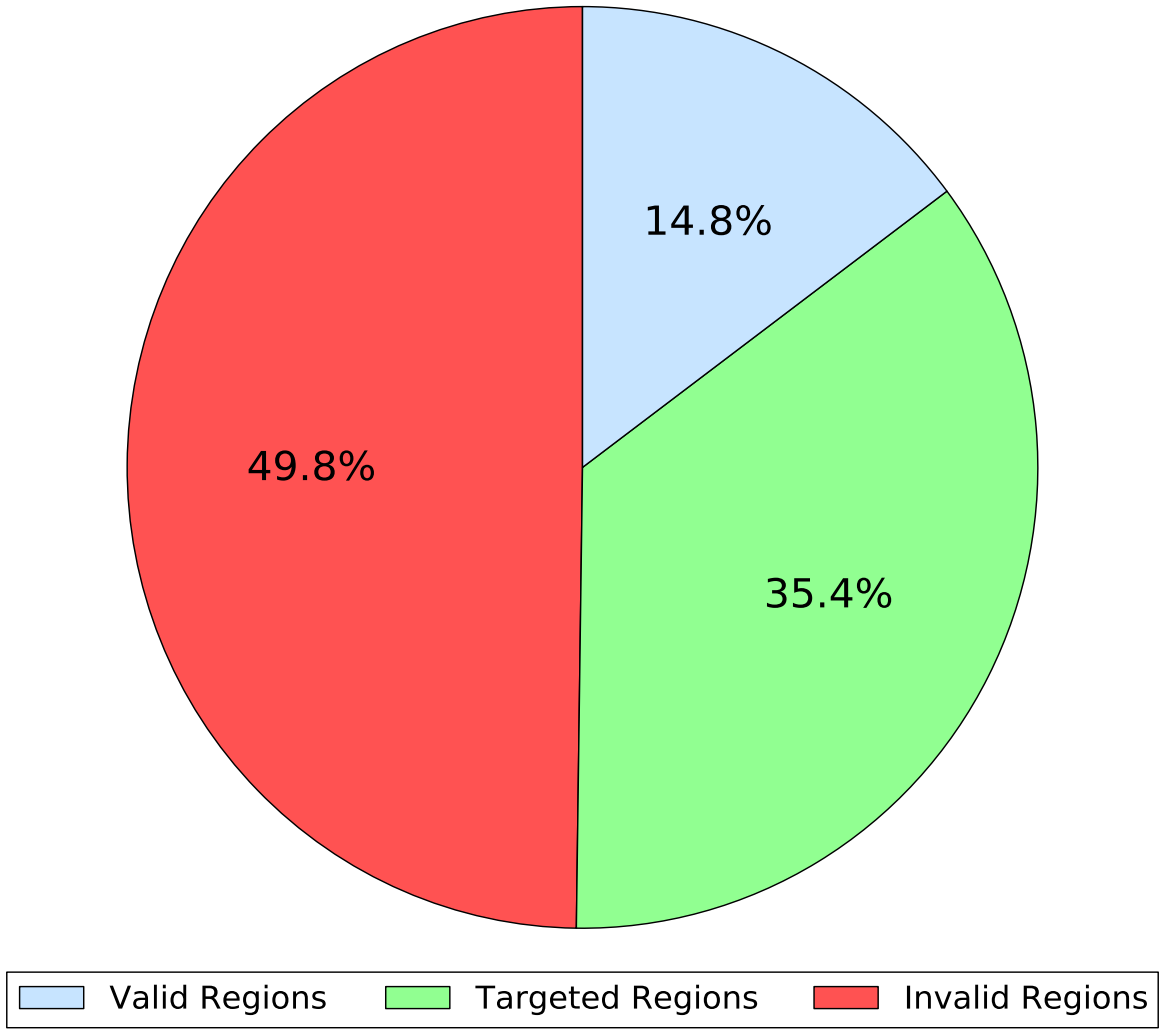
A #regions where condition i is violated.

B #regions where *only* condition i is violated.

C #regions where *only* conditions 0 to i are violated.

How urgent is this problem?

Conclusion



How to allow more polyhedral optimizations?

Example

```
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)

```
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

```
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**

```
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**

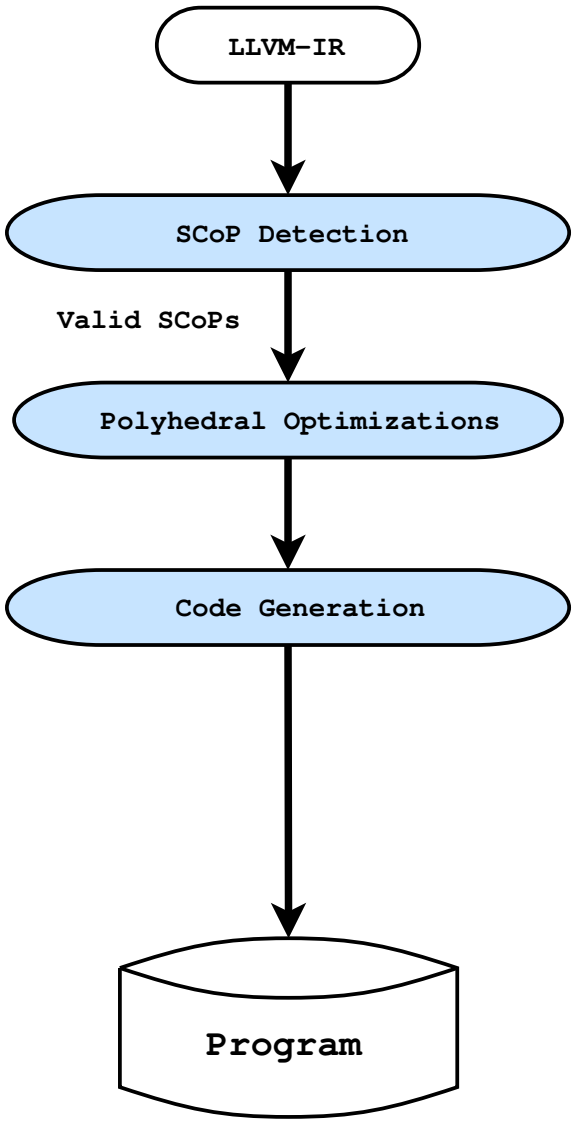
```
void f_dispatcher(int* A, int* B) {
  if (overlap(A, B, 2048))
    f(A, B);
  else
    f_opt(A, B);
}
```

How to allow more polyhedral optimizations?

Implementation

How to allow more polyhedral optimizations?

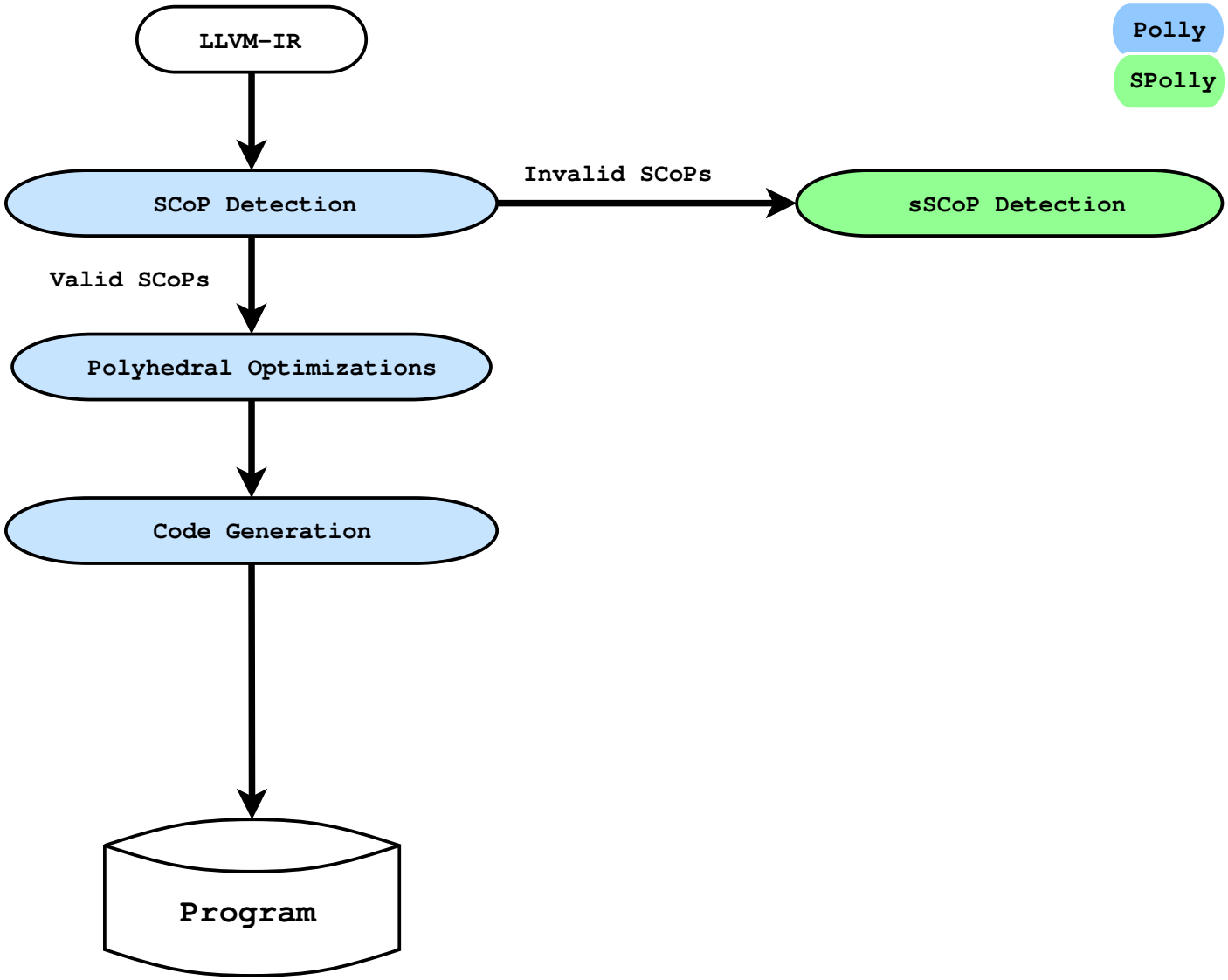
Implementation



Polly

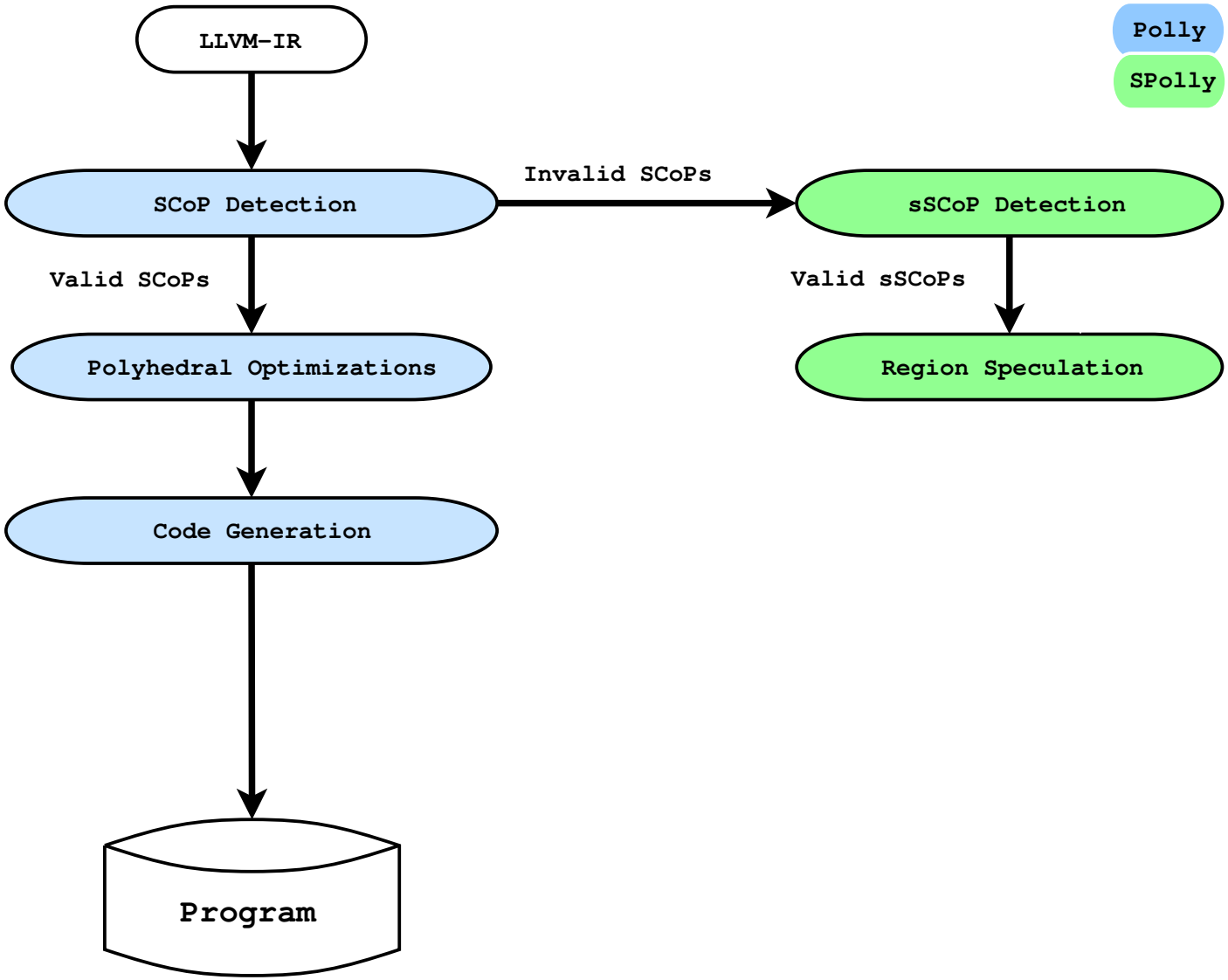
How to allow more polyhedral optimizations?

Implementation



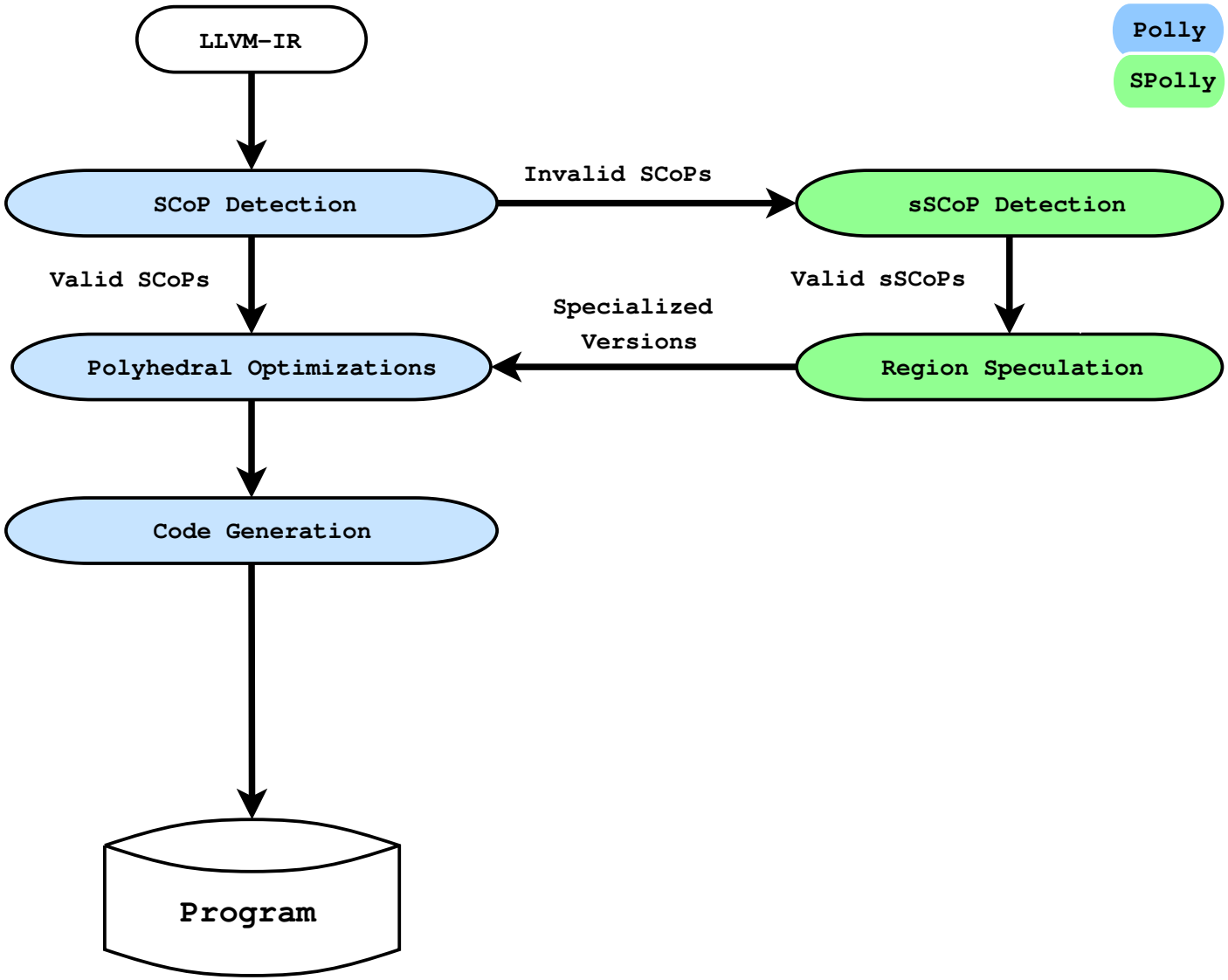
How to allow more polyhedral optimizations?

Implementation



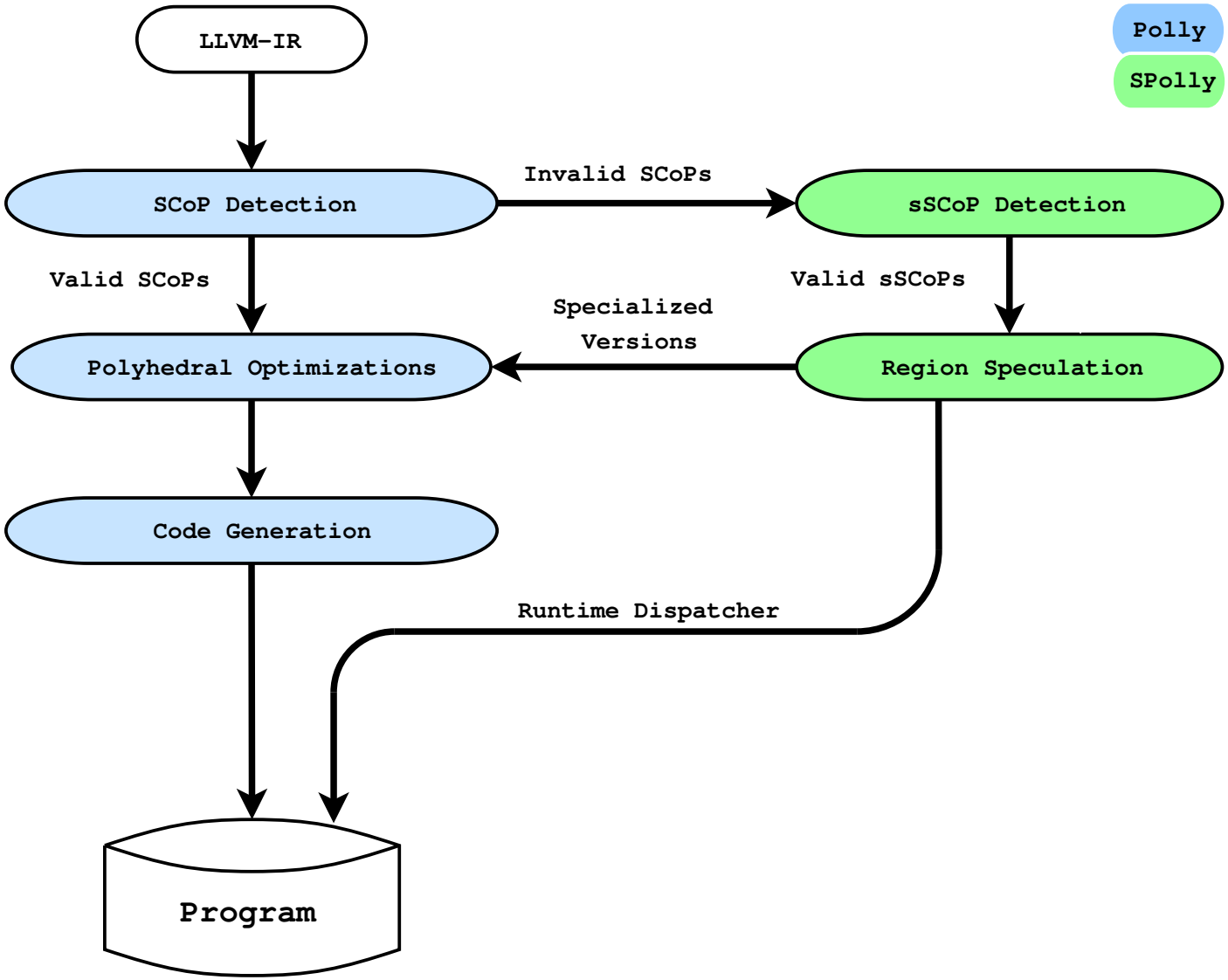
How to allow more polyhedral optimizations?

Implementation



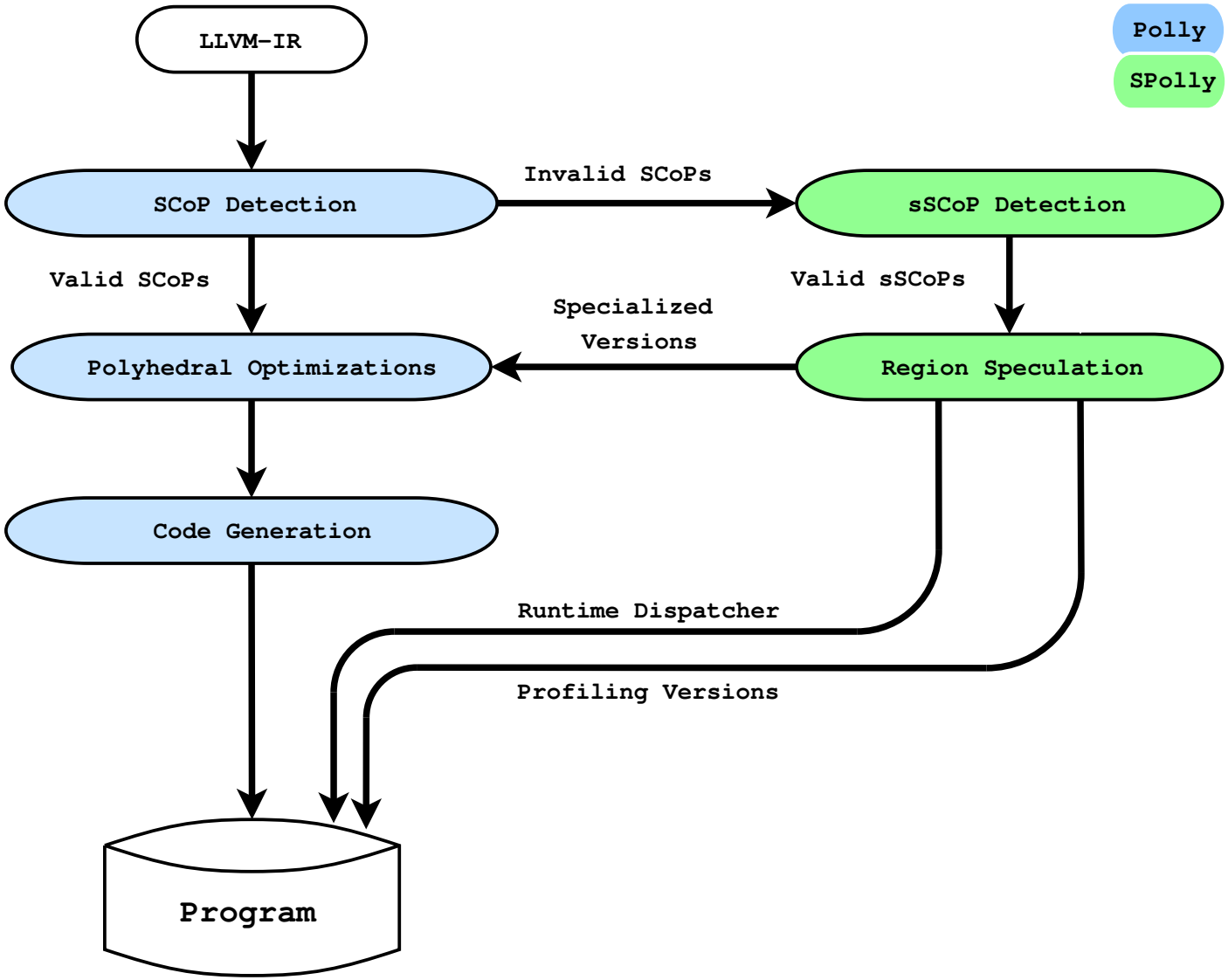
How to allow more polyhedral optimizations?

Implementation



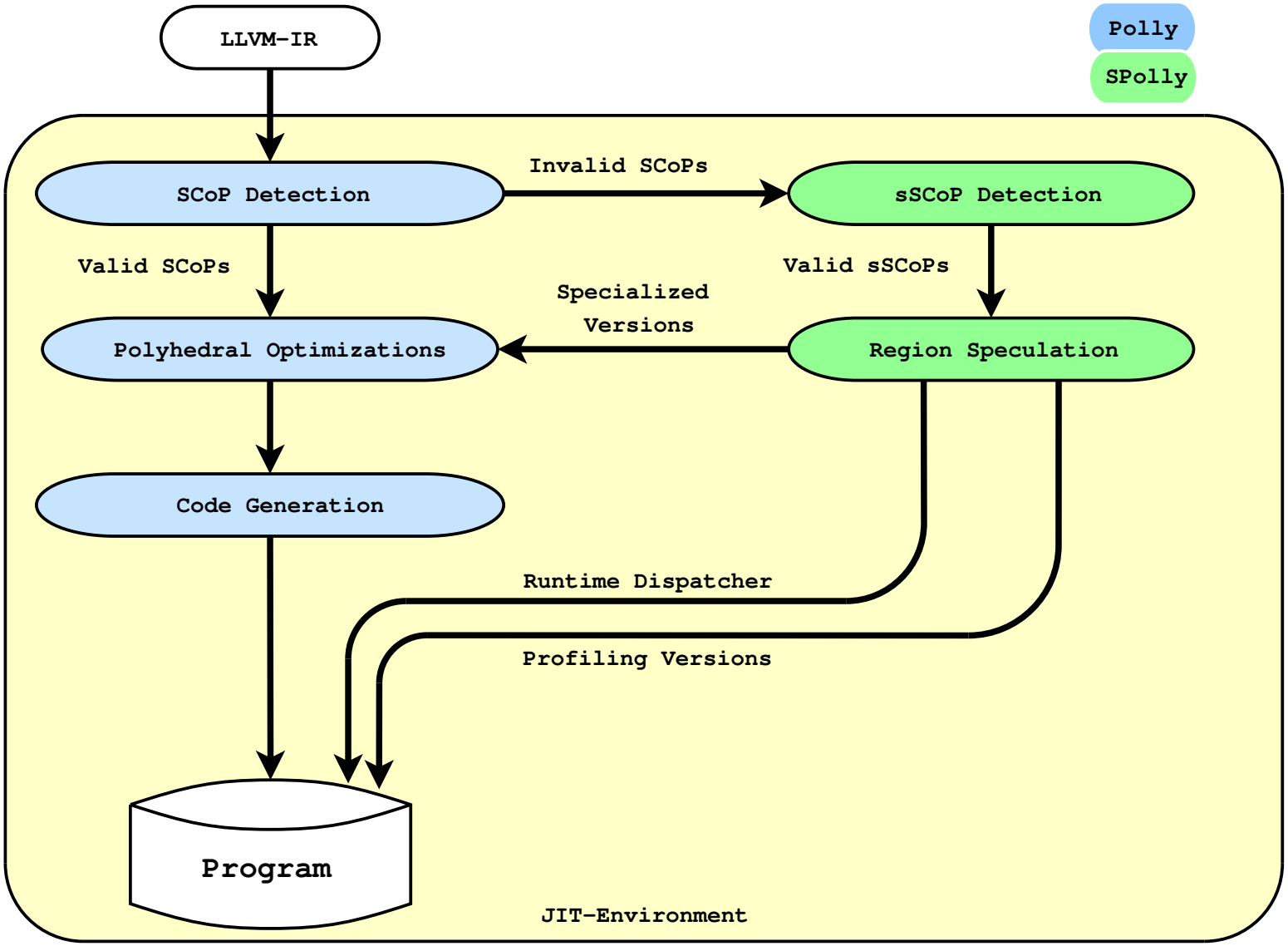
How to allow more polyhedral optimizations?

Implementation



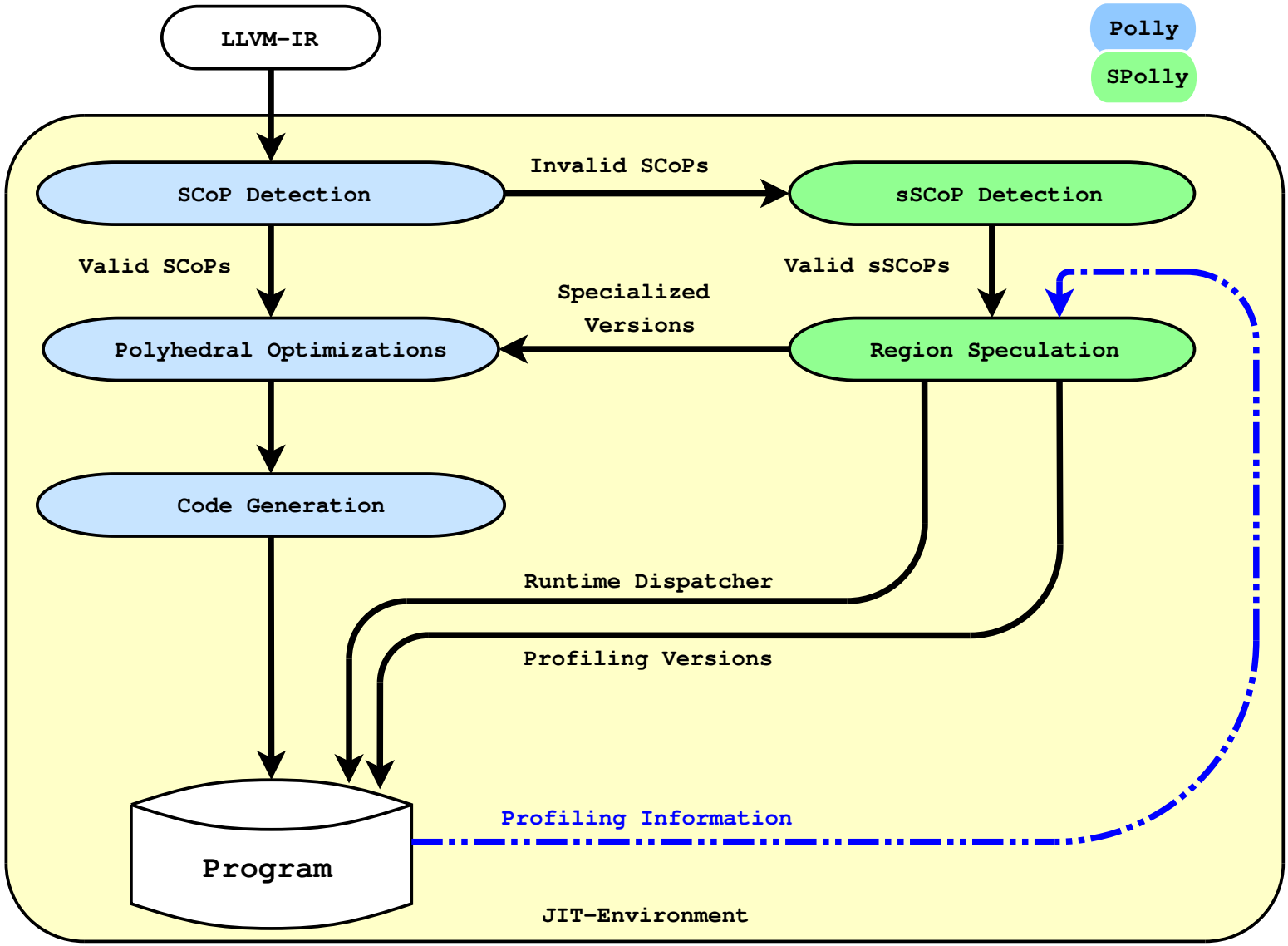
How to allow more polyhedral optimizations?

Implementation



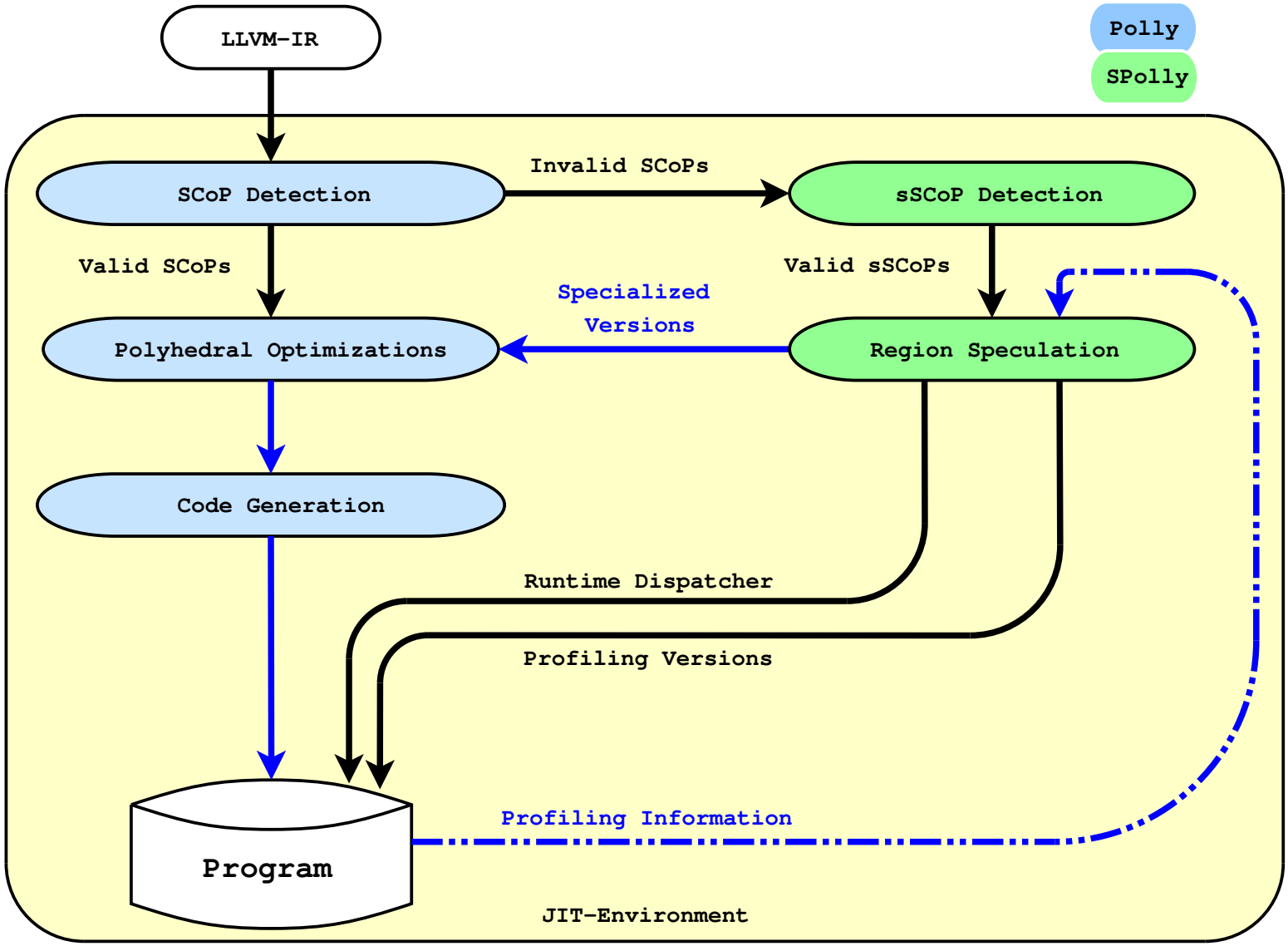
How to allow more polyhedral optimizations?

Implementation



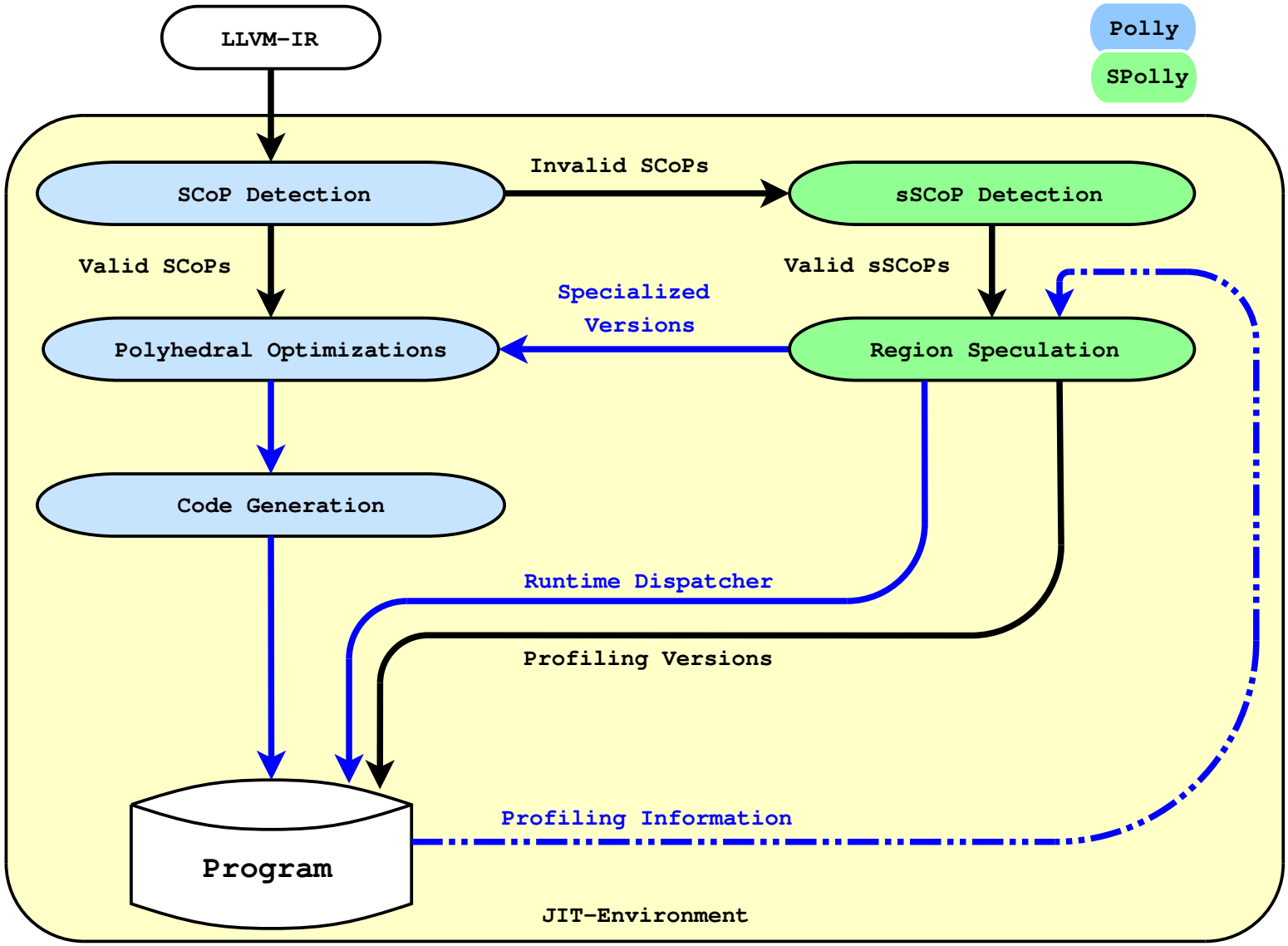
How to allow more polyhedral optimizations?

Implementation



How to allow more polyhedral optimizations?

Implementation



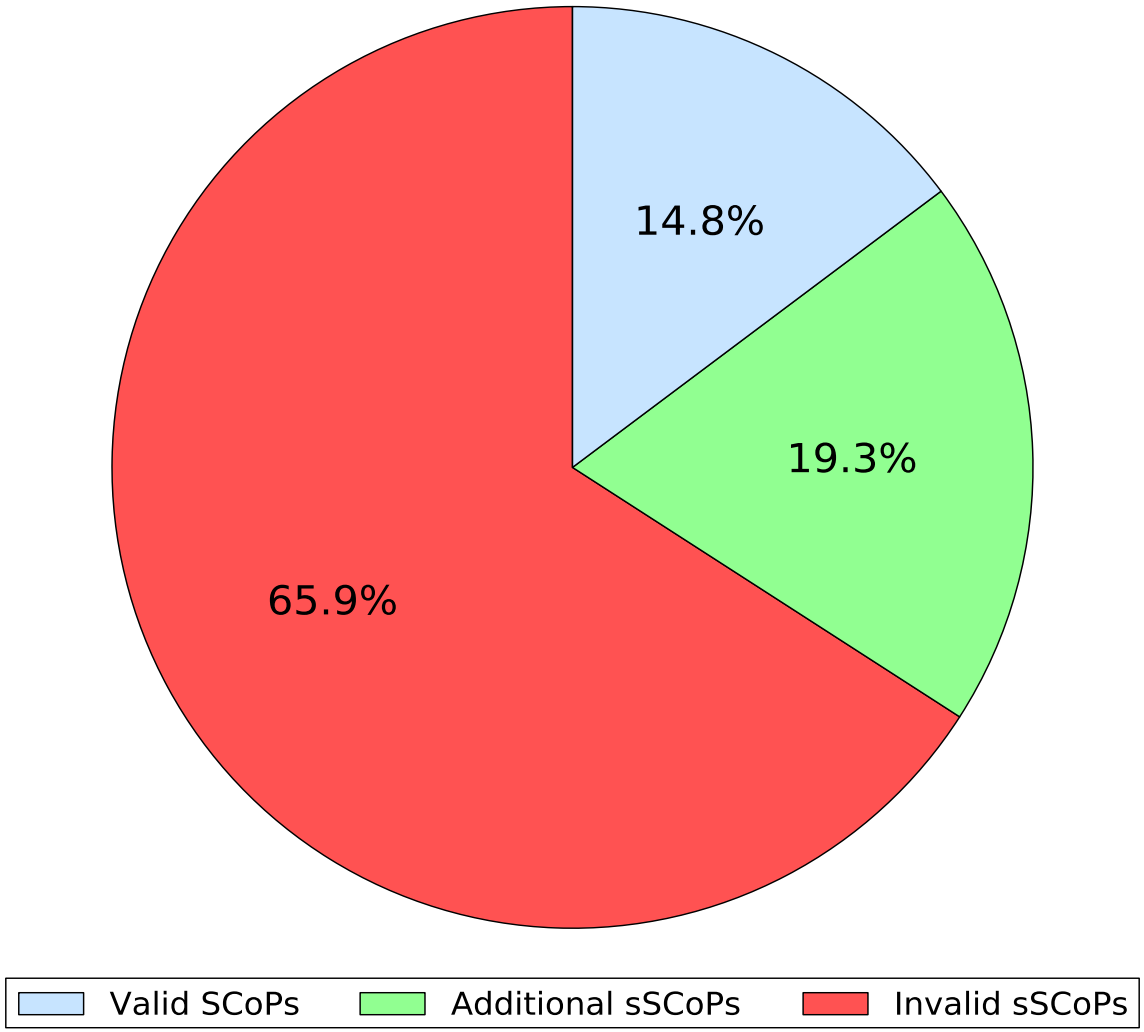
Does it work?

Does it work?

Almost.

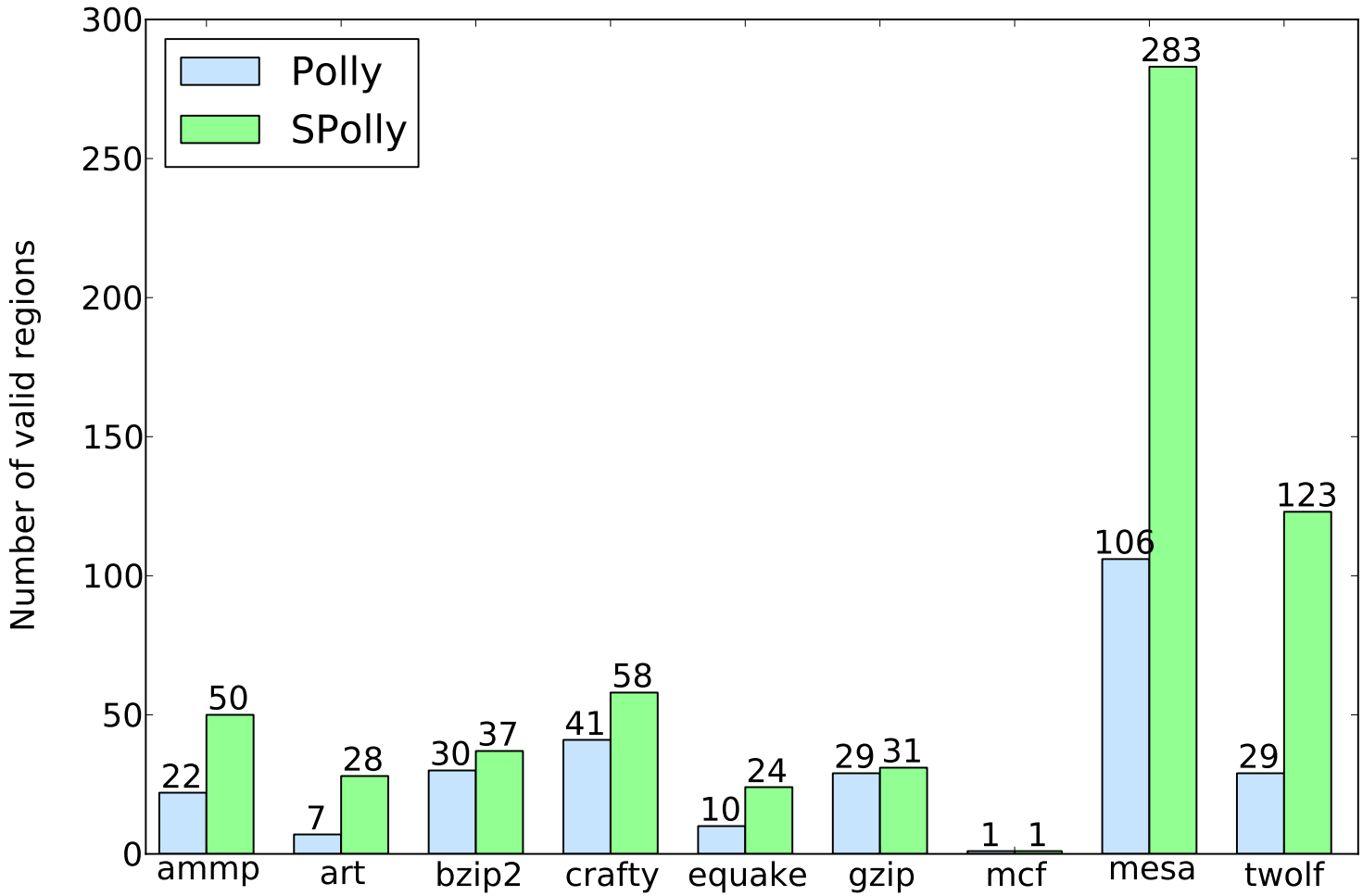
Does it work?

Applicability on SPEC 2000



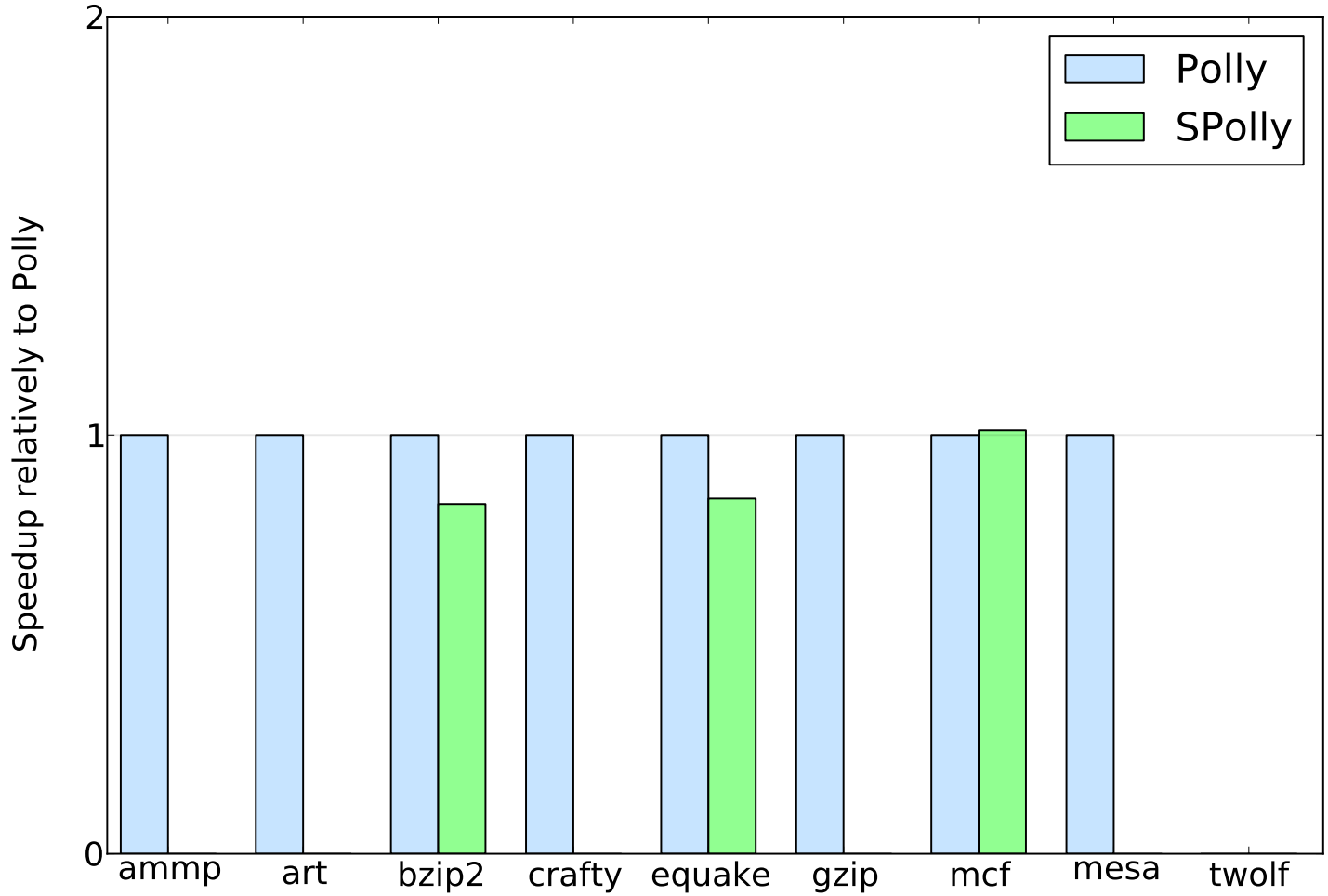
Does it work?

Applicability on SPEC 2000



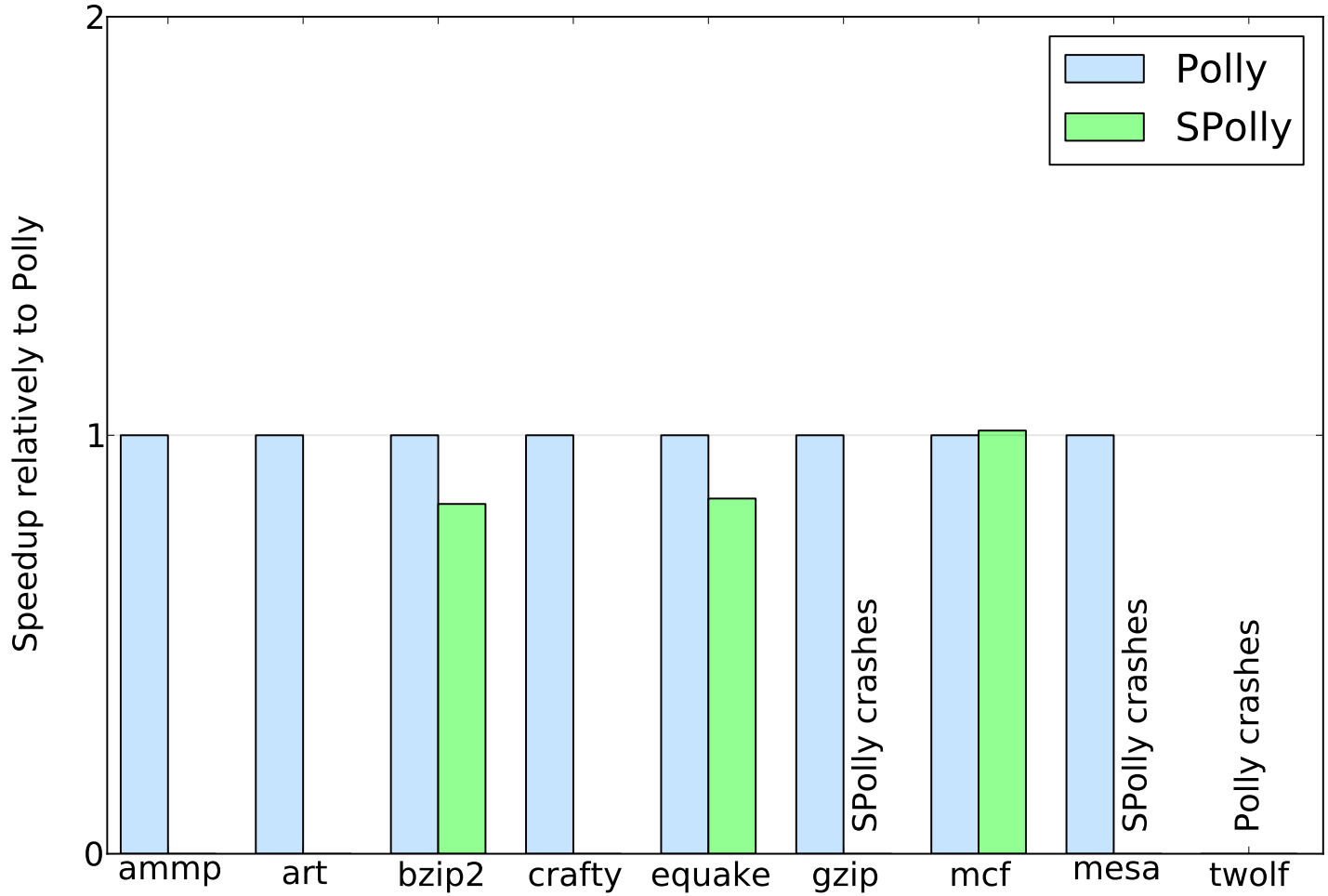
Does it work?

Runtime Results on SPEC 2000



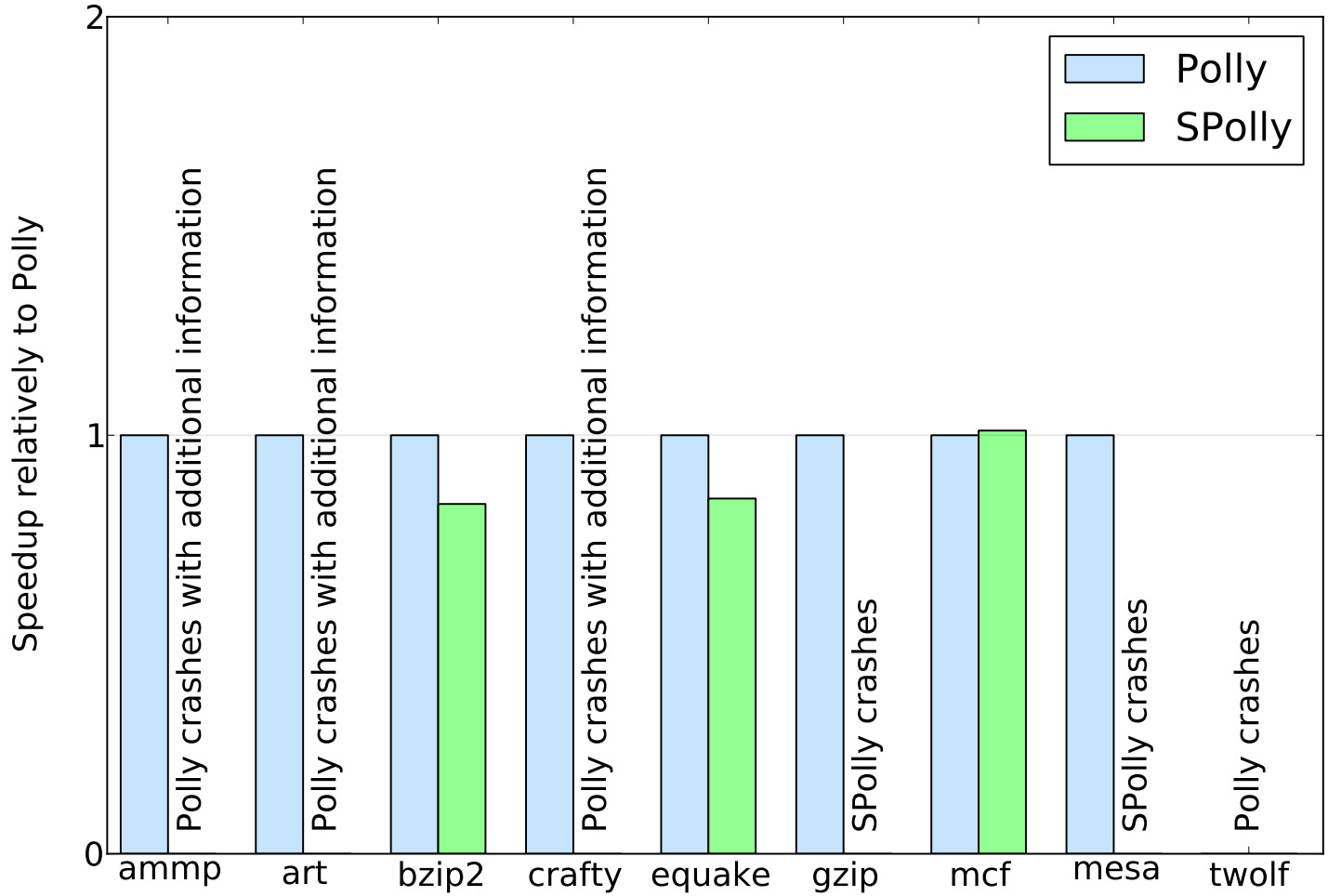
Does it work?

Runtime Results on SPEC 2000



Does it work?

Runtime Results on SPEC 2000



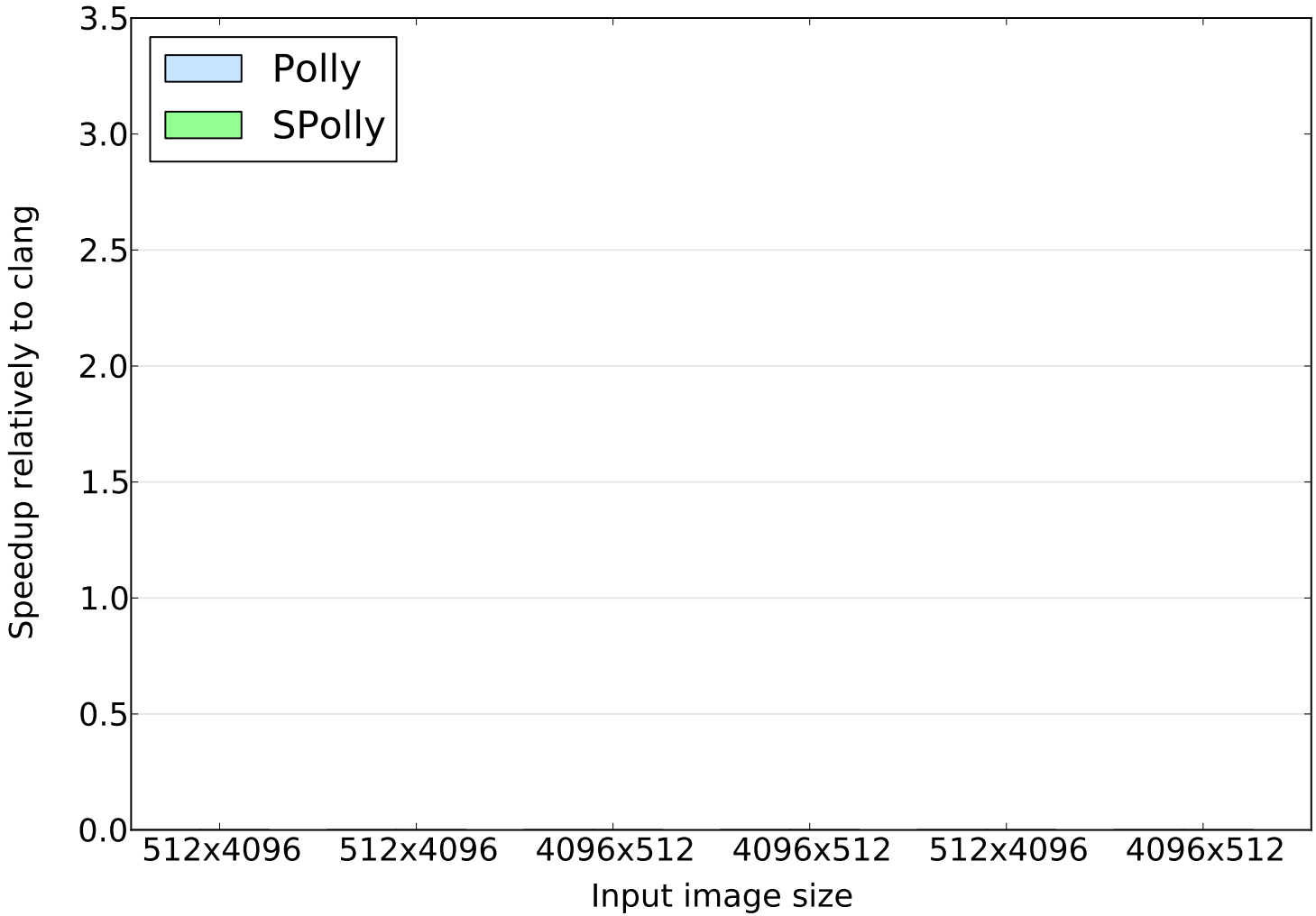
Does it work?

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

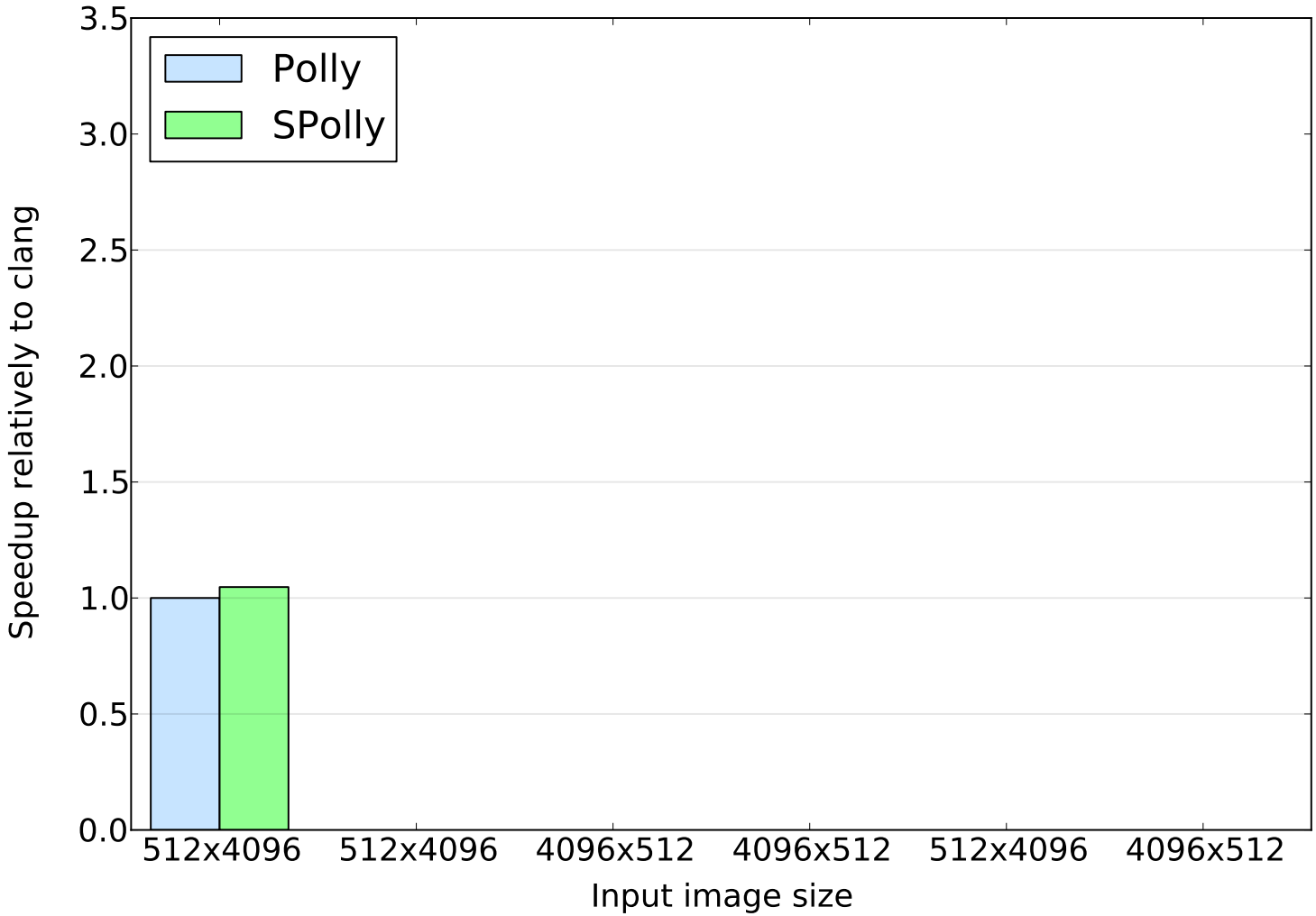
Does it work?

Case Study – Results



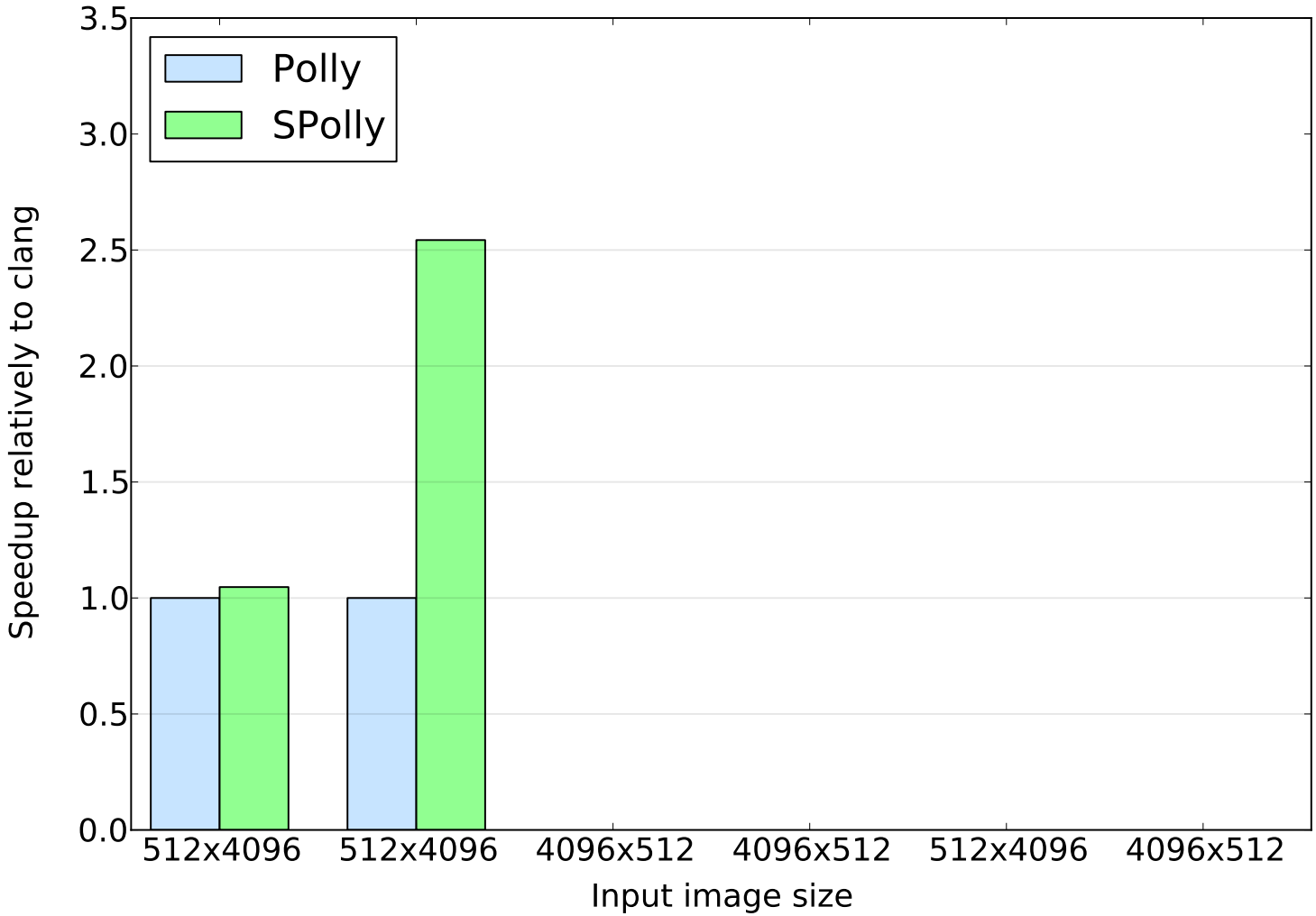
Does it work?

Case Study – Results



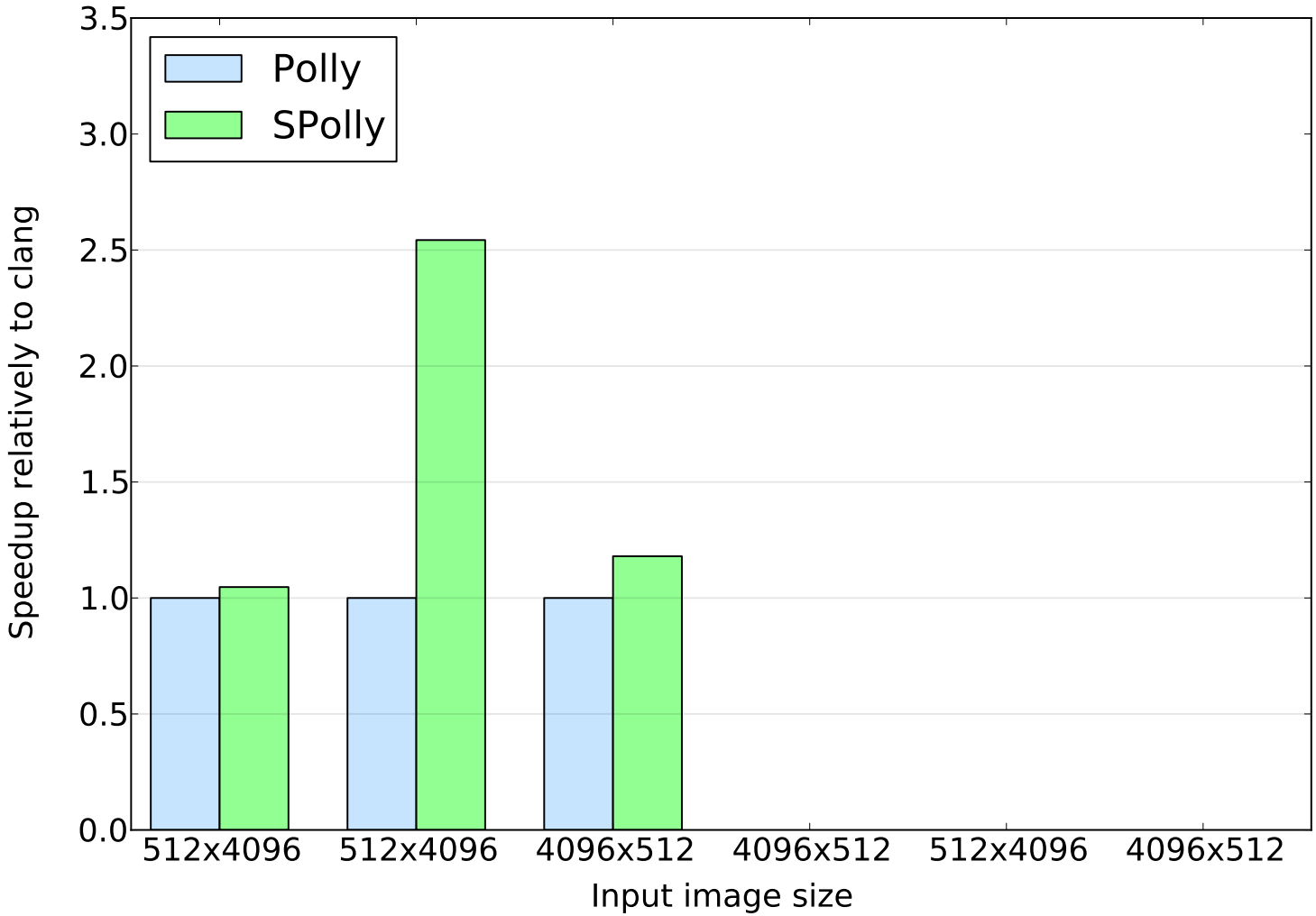
Does it work?

Case Study – Results



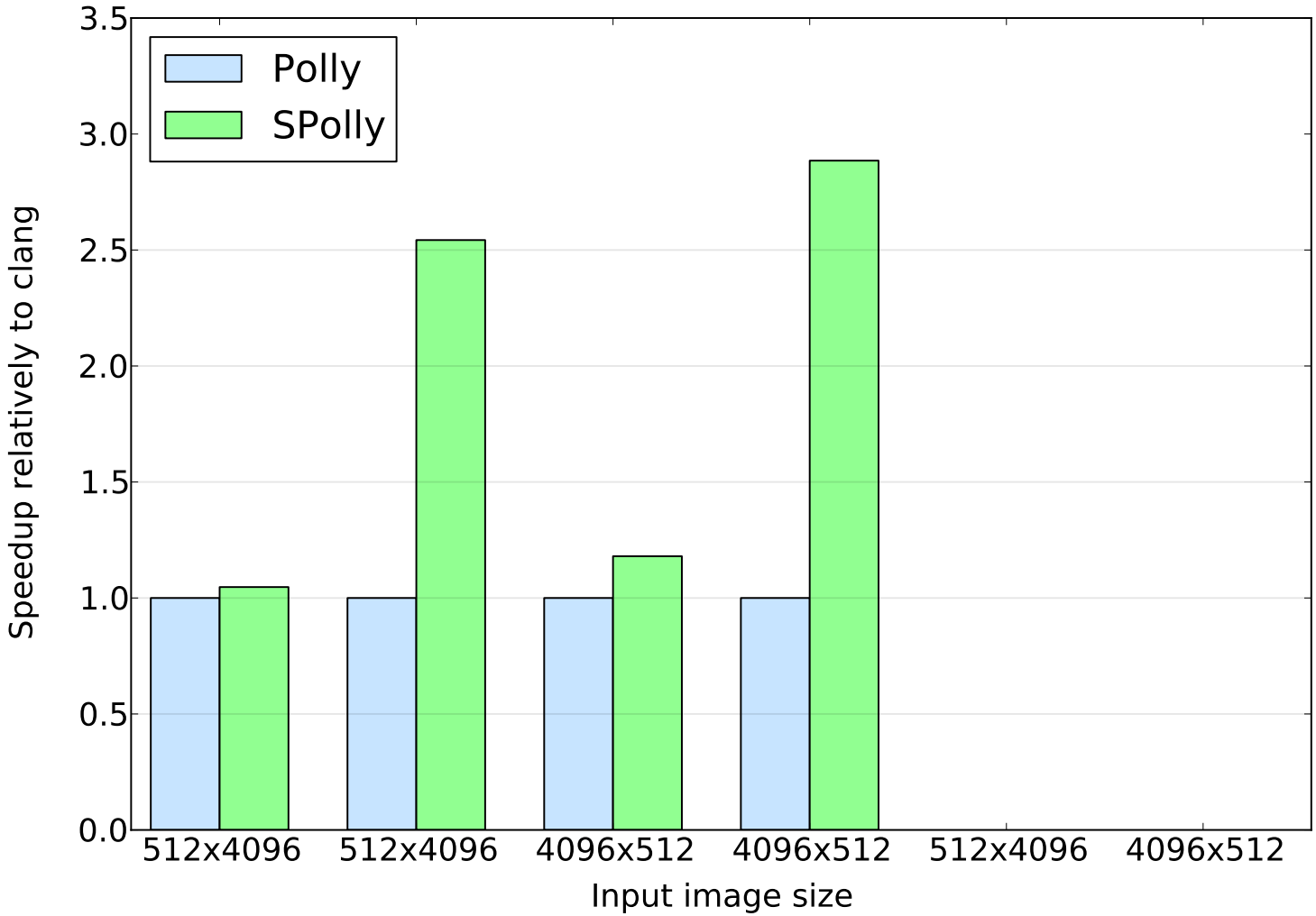
Does it work?

Case Study – Results



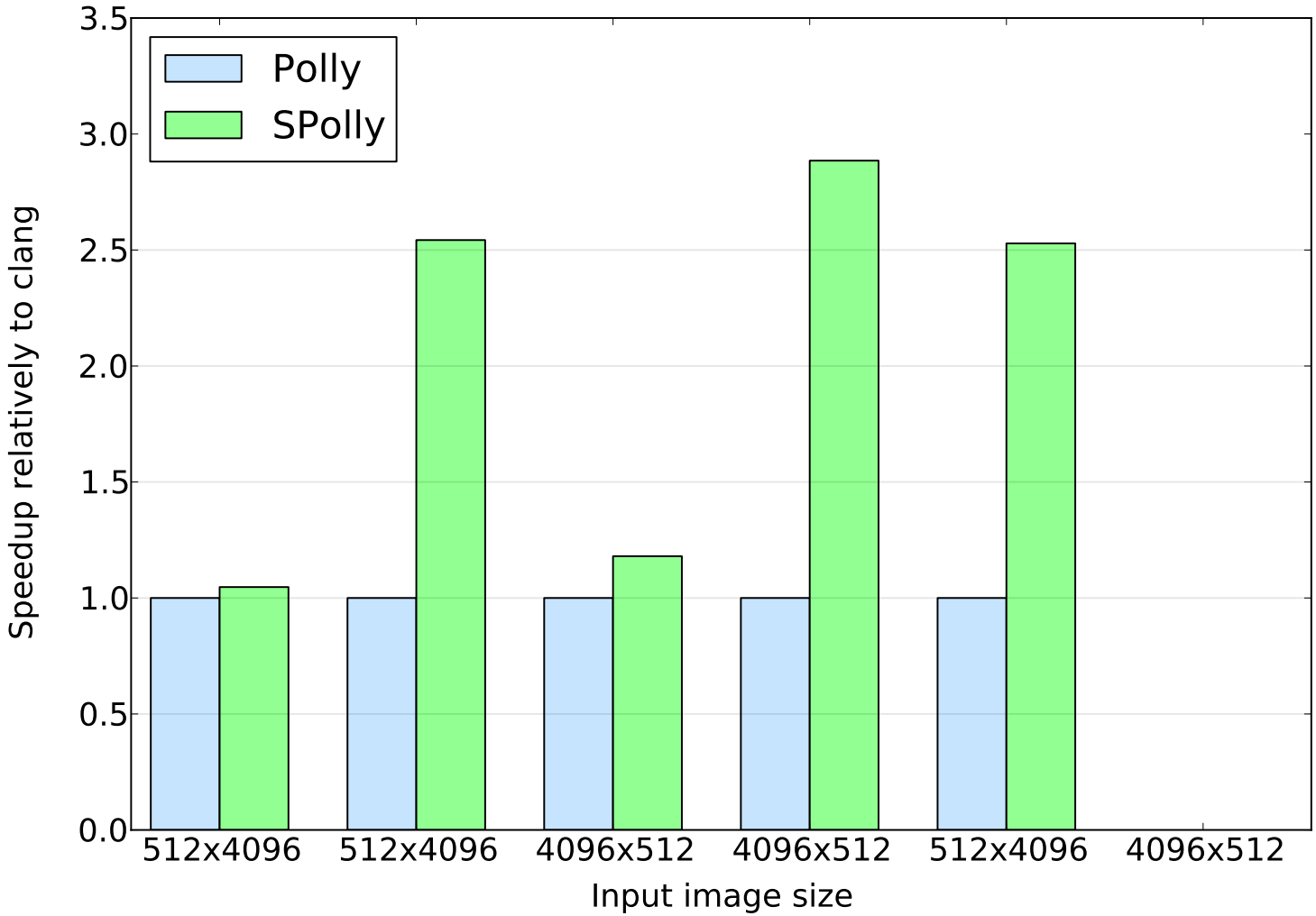
Does it work?

Case Study – Results



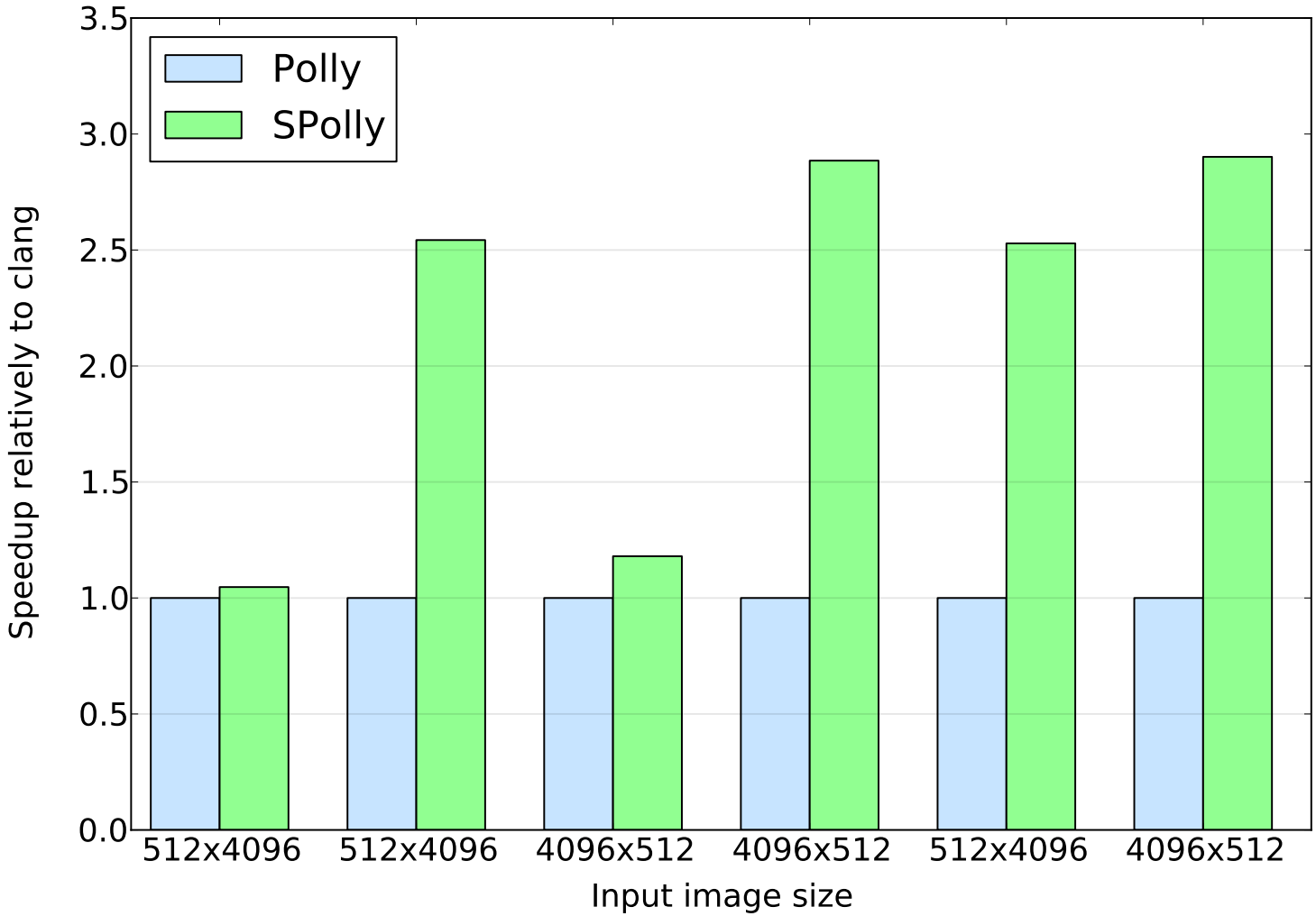
Does it work?

Case Study – Results



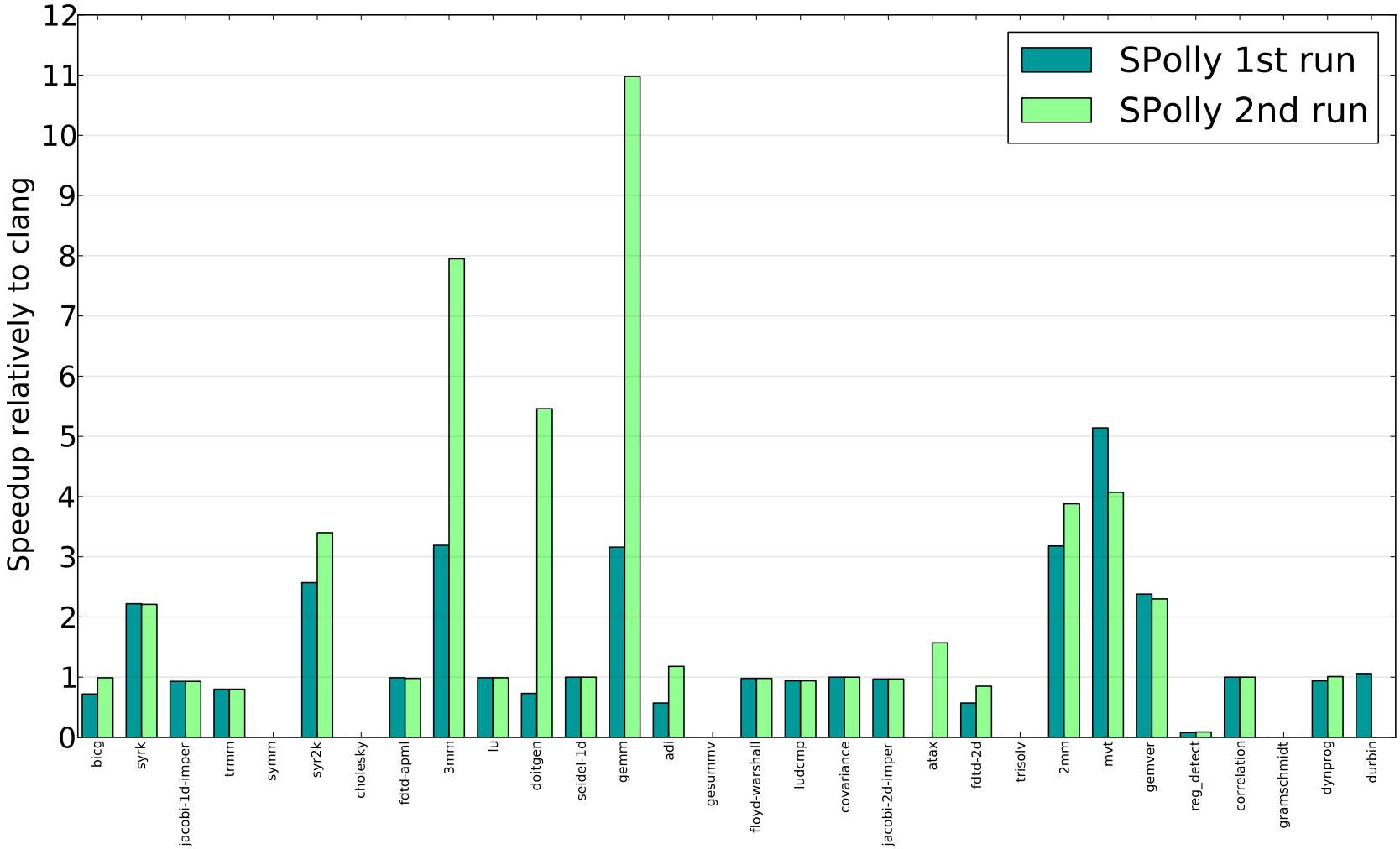
Does it work?

Case Study – Results



Does it work?

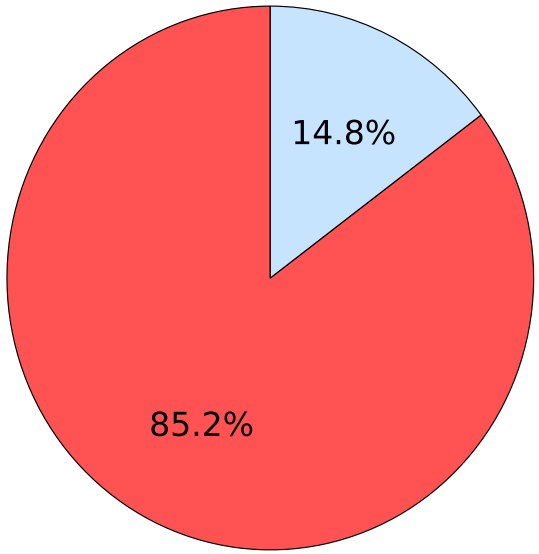
Runtime Results on Polybench



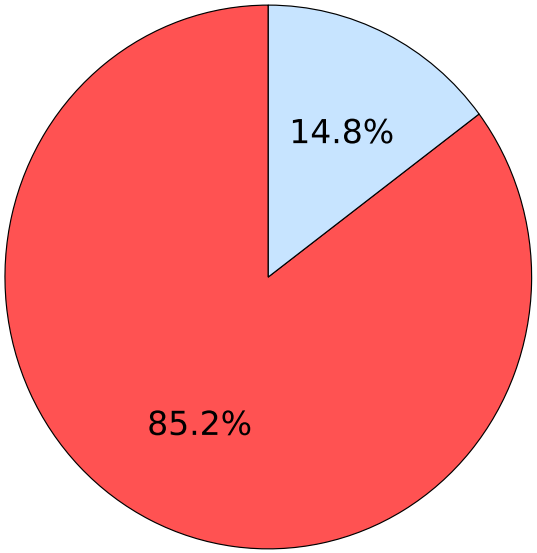
Geomean: [1st run] **1.134** [2nd run] **1.481**

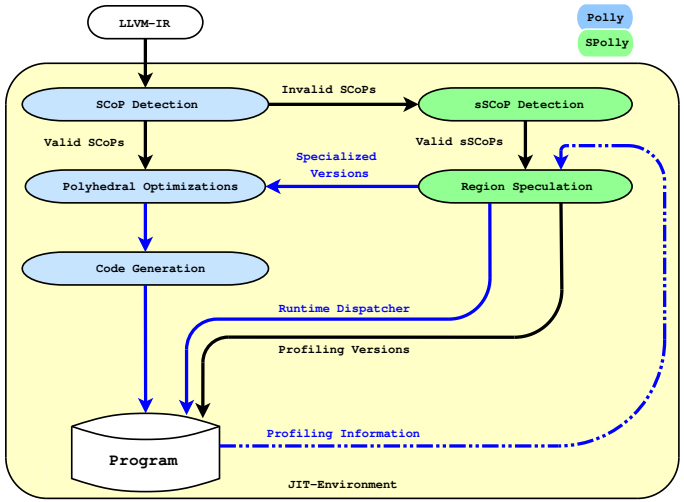
Summary

Summary

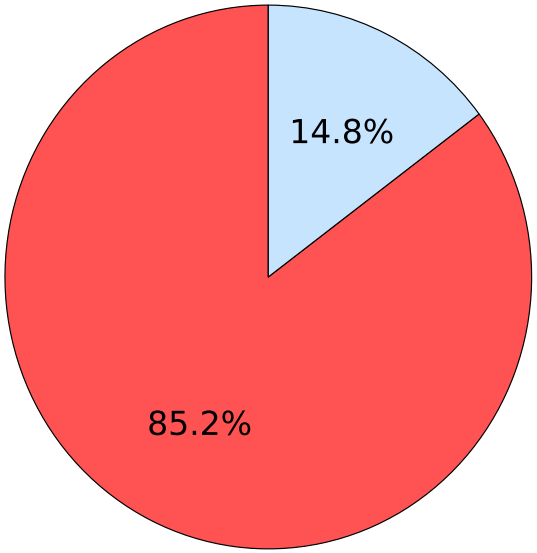


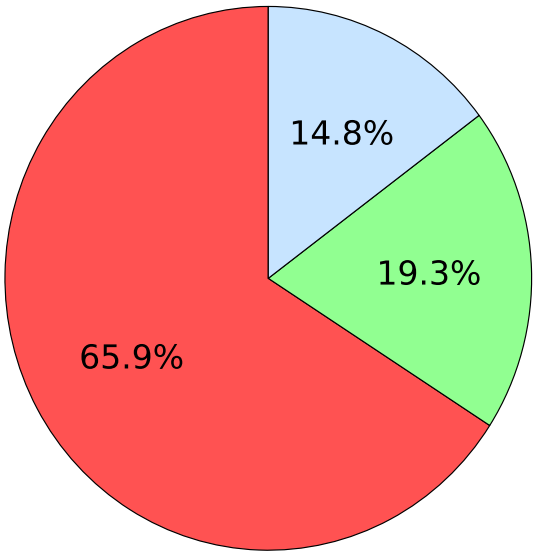
Summary

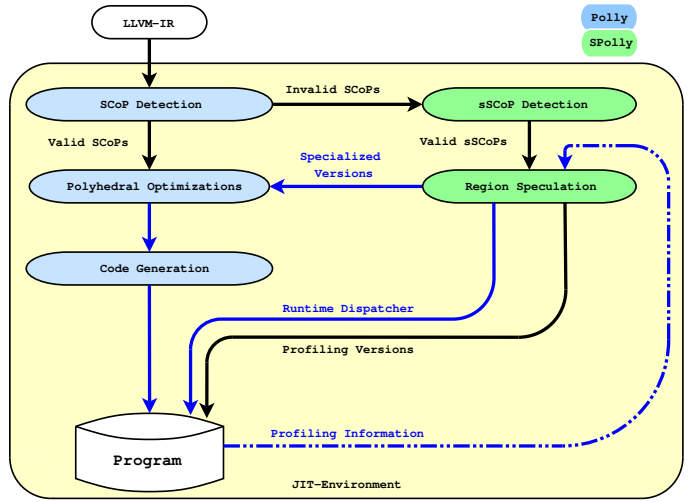

14.8%
85.2%


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

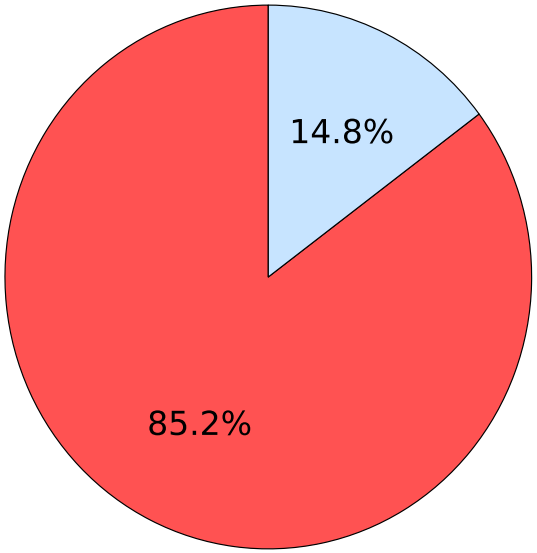
Summary

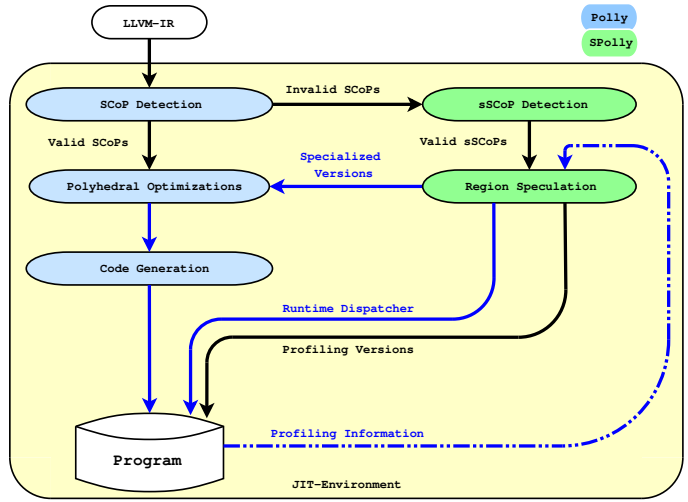

14.8%
85.2%

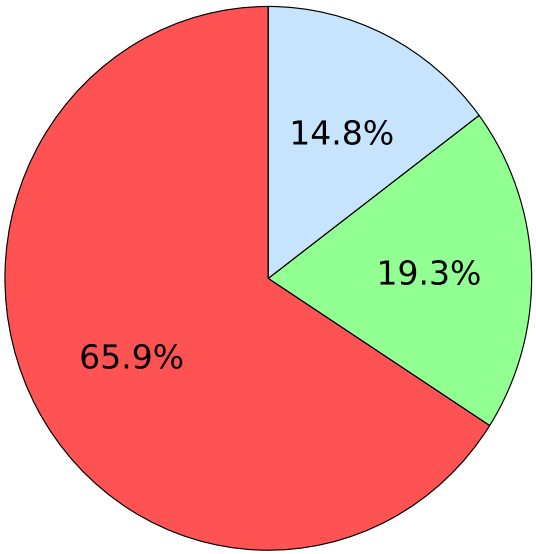

14.8%
19.3%
65.9%

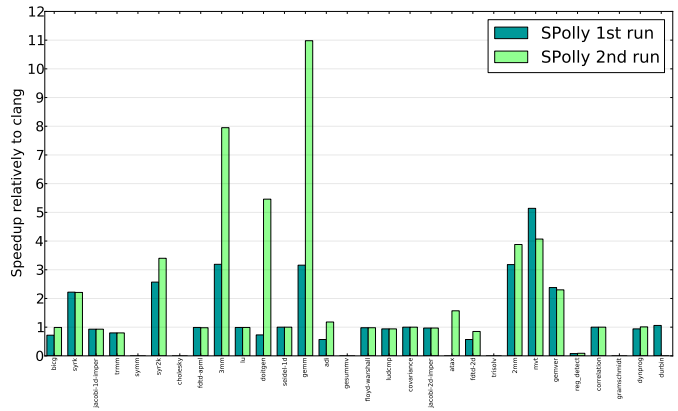

LLVM-IR
Polly
SPolly
SCoP Detection
Invalid SCoPs
sSCoP Detection
Valid SCoPs
Valid sSCoPs
Specialized Versions
Polyhedral Optimizations
Region Speculation
Code Generation
Runtime Dispatcher
Profiling Versions
Profiling Information
Program
JIT-Environment

Summary


14.8%
85.2%


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


14.8%
19.3%
65.9%


SPolly 1st run
SPolly 2nd run
Speedup relatively to clang
0
1
2
3
4
5
6
7
8
9
10
11
12

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

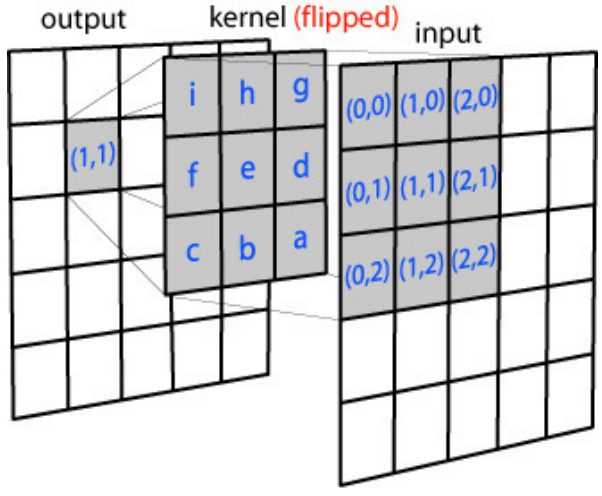
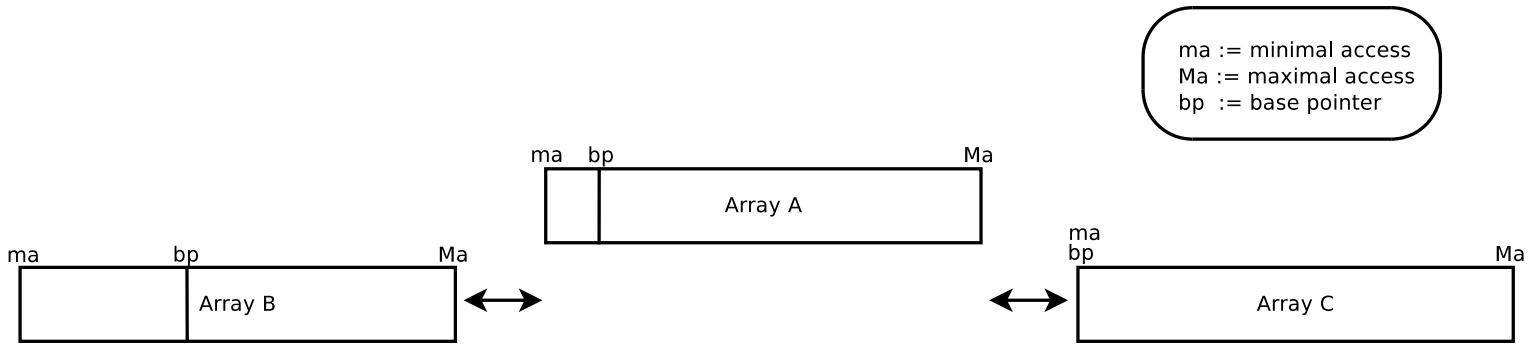


Image source:

https://sonnati.wordpress.com/2010/10/06/flash-h-264-h-264-squared-%E2%80%93-part-iii/

Concept

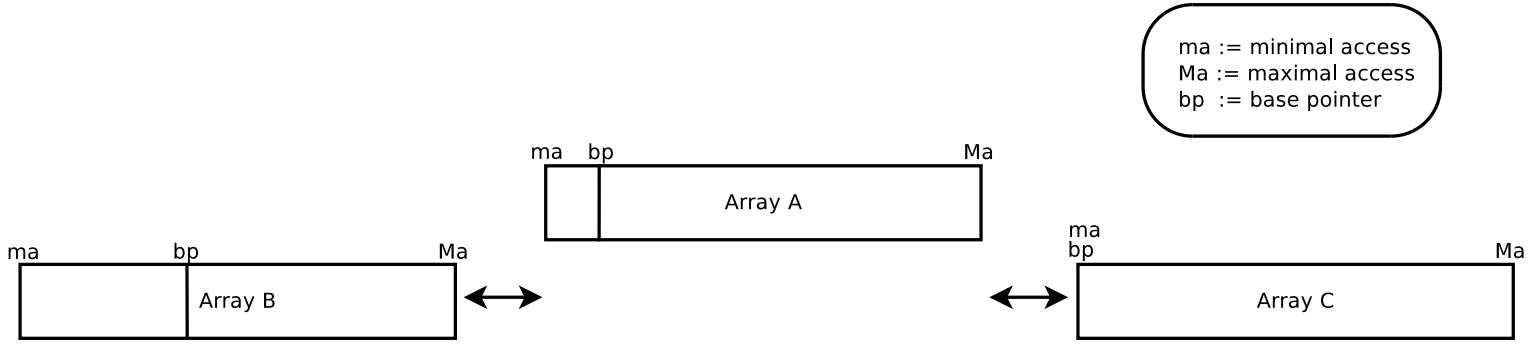
Alias tests



Concept

Alias tests

```
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];
    }
  }
}
```

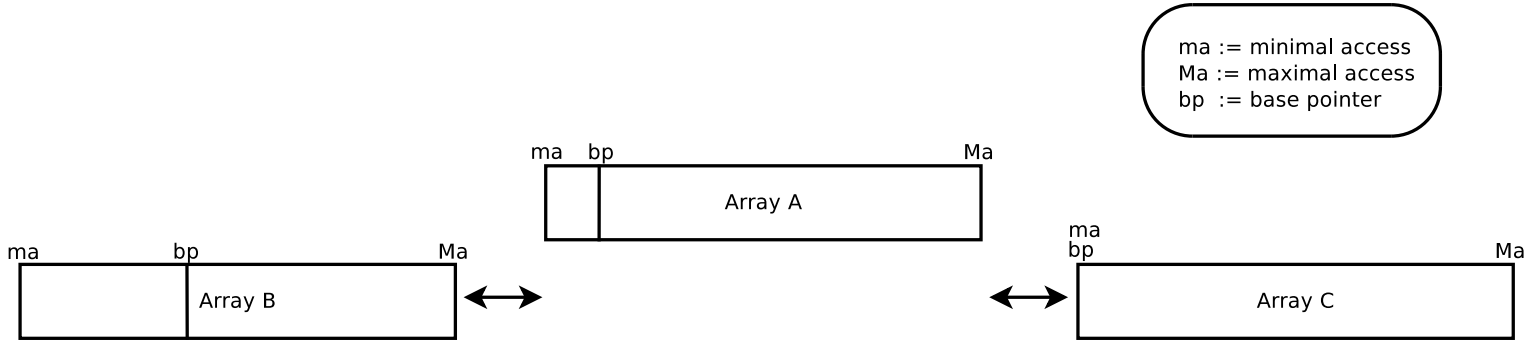


Concept

Alias tests

```
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



Concept

Alias tests

```
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

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