Polyhedral AST generation is more than scanning polyhedra

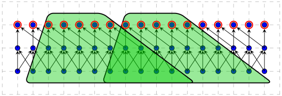
Tobias Grosser, Sven Verdoolaege, Albert Cohen

ETH Zurich, Polly Labs, INRIA

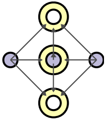
TOPLAS - Presented at PLDI'16

15. June 2015, Santa Barbara, USA

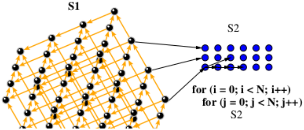
AST Generation at the Heart of Research



PolyMage - ASPLOS'15

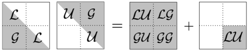


Associative Reordering - PLDI'14

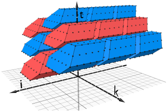


Pluto - PLDI'08

LLVM Polly - PPL'12

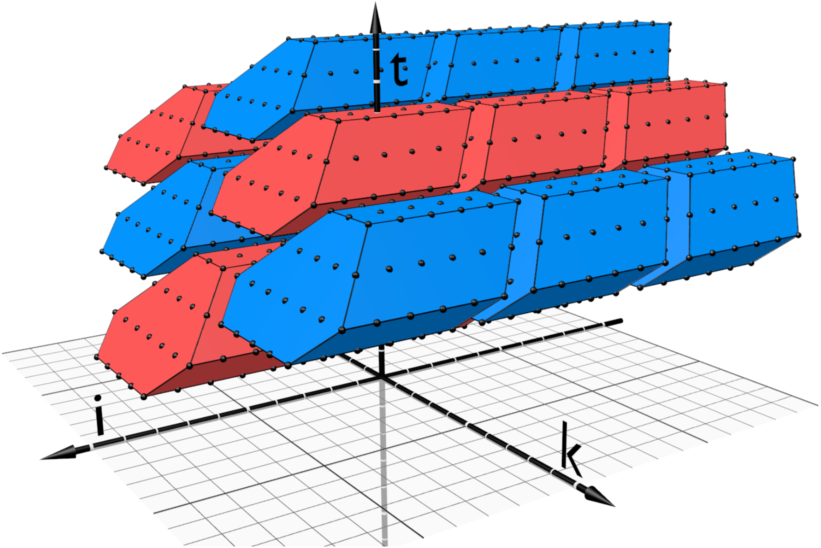


Basic Structured Linear Algebra Compiler - CGO'16



Hybrid-Hexagonal Tiling of Stencils - CGO'14

Hybrid-Hexagonal Tiling for Stencil Computations



Copy code from hybrid hexagonal tiling - Original

```
for (c2 = 0; c2 <= 1; c2 += 1)
  for (c3 = 1; c3 <= 4; c3 += 1)
    for (c4 = max(((t1-c3+130) % 128) + c3 - 2,
                  ((t1+c3+125) % 128) - c3 + 3);
         c4 <= min(((c2+c3) % 2) + c3 + 128,
                   -((c2+c3) % 2) - c3 + 134);
         c4 += 128)
      if (c3 + c4 >= 7 || (c4 == t1 && c3 + 2 >= t1 && t1 + c3 <= 6
                            && t1 + c3 >= ((t1 + c2 + 2 * c3 + 1) % 2) + 3
                            && t1 + 2 >= ((t1 + c2 + 2 * c3 + 1) % 2) + c3)
          || (c4 == t1 && c3 == 1 && t1 <= 5 && t1 >= 4 &&
              c2 <= 1 && c2 >= 0))
        A[c2][6 * b0 + c3][128 * g7 + c4 - 4] = ...;
```

Copy code from hybrid hexagonal tiling - Unrolled

```
A[0][6 * b0 + 1][128 * g7 + (t1 + 125) % 128) - 1] = ...;
A[0][6 * b0 + 2][128 * g7 + (t1 + 127) % 128) - 3] = ...;
if (t1 <= 2 && t1 >= 1)
    A[0][6 * b0 + 2][128 * g7 + t1 + 128] = ...;
A[0][6 * b0 + 3][128 * g7 + (t1 + 127) % 128) - 3] = ...;
if (t1 <= 2 && t1 >= 1)
    A[0][6 * b0 + 3][128 * g7 + t1 + 128] = ...;
A[0][6 * b0 + 4][128 * g7 + (t1 + 125) % 128) - 1] = ...;
A[1][6 * b0 + 1][128 * g7 + (t1 + 126) % 128) - 2] = ...;
A[1][6 * b0 + 2][128 * g7 + (t1 + 126) % 128) - 2] = ...;
if (t1 <= 3 && t1 >= 2)
    A[1][6 * b0 + 2][128 * g7 + t1 + 128] = ...;
A[1][6 * b0 + 3][128 * g7 + (t1 + 126) % 128) - 2] = ...;
if (t1 <= 3 && t1 >= 2)
    A[1][6 * b0 + 3][128 * g7 + t1 + 128] = ...;
A[1][6 * b0 + 4][128 * g7 + (t1 + 126) % 128) - 2] = ...;
```

Program

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

State of Variables

Statement Instances Executed

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(0,0)

Program

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

State of Variables

n = 4, i = 1, j = 0

Statement Instances Executed

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

Program

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

State of Variables

n = 4, i = 1, j = 1

Statement Instances Executed

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

Program

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

State of Variables

n = 4, i = 2, j = 0

Statement Instances Executed

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

Program

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

State of Variables

n = 4, i = 2, j = 1

Statement Instances Executed

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

Program

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

State of Variables

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

Statement Instances Executed

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

Program

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

State of Variables

$n = 4$, $i = 3$, $j = 0$

Statement Instances Executed

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

Program

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

State of Variables

$n = 4$, $i = 3$, $j = 1$

Statement Instances Executed

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

Program

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

State of Variables

$n = 4, i = 3, j = 2$

Statement Instances Executed

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

Program

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

State of Variables

n = 4, i = 3, j = 3

Statement Instances Executed

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

Program

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

State of Variables

n = 4, i = 4, j = 0

Statement Instances Executed

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

Program

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

State of Variables

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

Statement Instances Executed

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

Program

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

State of Variables

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

Statement Instances Executed

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

Program

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

State of Variables

n = 4, i = 4, j = 3

Statement Instances Executed

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

Program

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

State of Variables

$n = 4, i = 4, j = 4$

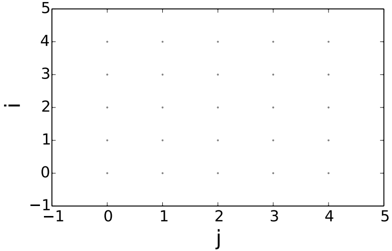
Statement Instances Executed

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

Program

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

Iteration space



State of Variables

n = 4, i = 4, j = 4

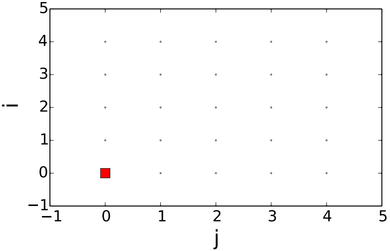
Statement Instances Executed

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

Program

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

Iteration space



State of Variables

n = 4, i = 0, j = 0

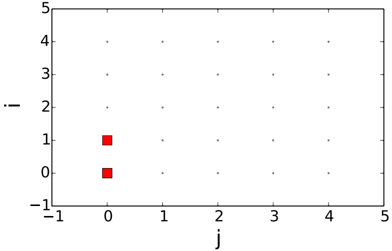
Statement Instances Executed

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

Program

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

Iteration space



State of Variables

n = 4, i = 1, j = 0

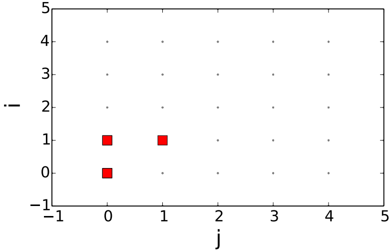
Statement Instances Executed

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

Program

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

Iteration space



State of Variables

n = 4, i = 1, j = 1

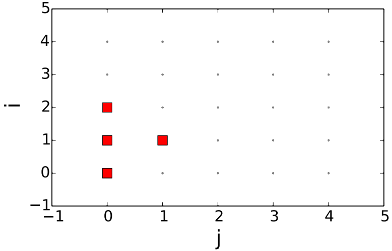
Statement Instances Executed

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

Program

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

Iteration space



State of Variables

n = 4, i = 2, j = 0

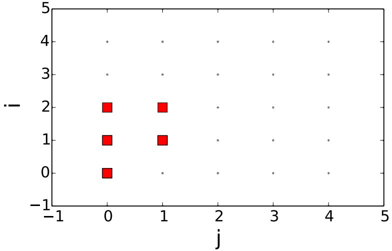
Statement Instances Executed

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

Program

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

Iteration space



State of Variables

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

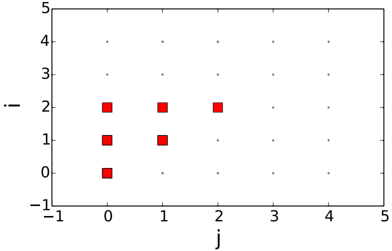
Statement Instances Executed

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

Program

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

Iteration space



State of Variables

n = 4, i = 2, j = 2

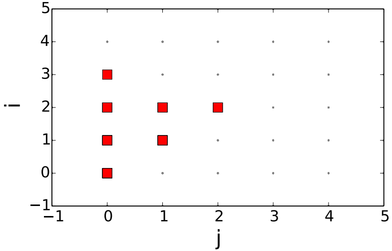
Statement Instances Executed

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

Program

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

Iteration space



State of Variables

n = 4, i = 3, j = 0

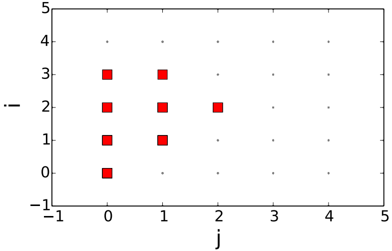
Statement Instances Executed

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

Program

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

Iteration space



State of Variables

n = 4, i = 3, j = 1

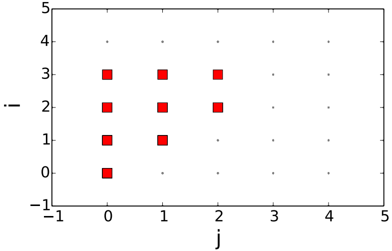
Statement Instances Executed

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

Program

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

Iteration space



State of Variables

n = 4, i = 3, j = 2

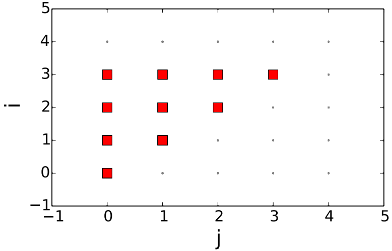
Statement Instances Executed

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

Program

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

Iteration space



State of Variables

$n = 4, i = 3, j = 3$

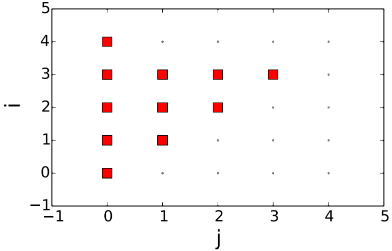
Statement Instances Executed

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

Program

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

Iteration space



State of Variables

n = 4, i = 4, j = 0

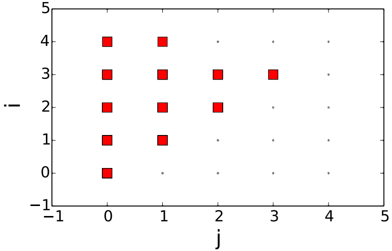
Statement Instances Executed

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

Program

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

Iteration space



State of Variables

n = 4, i = 4, j = 1

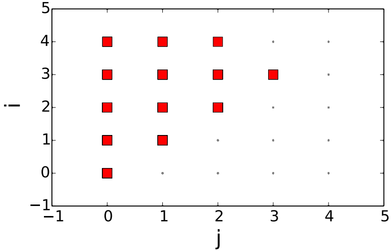
Statement Instances Executed

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

Program

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

Iteration space



State of Variables

n = 4, i = 4, j = 2

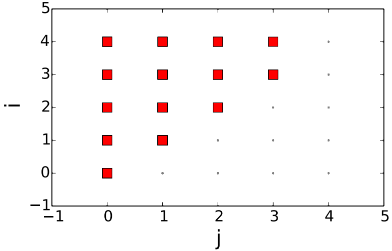
Statement Instances Executed

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

Program

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

Iteration space



State of Variables

n = 4, i = 4, j = 3

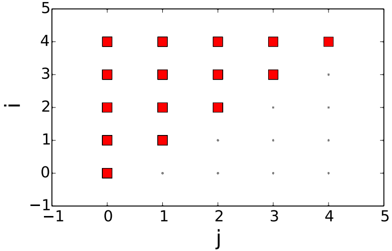
Statement Instances Executed

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

Program

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

Iteration space



State of Variables

n = 4, i = 4, j = 4

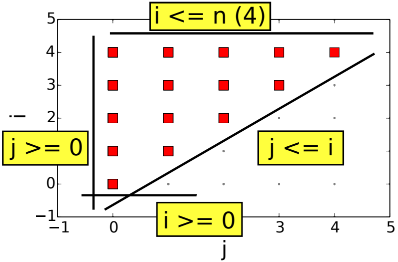
Statement Instances Executed

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

Program

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

Iteration space



State of Variables

n = 4, i = 4, j = 4

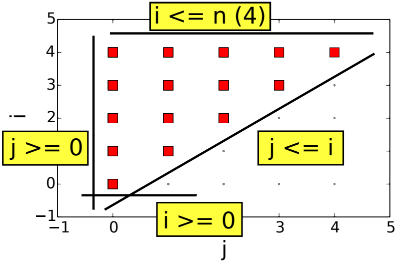
Statement Instances Executed

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

Program

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

Iteration space



State of Variables

n = 4, i = 4, j = 4

Statement Instances Executed

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

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

AST Generation - Basic Example

$$
\begin{aligned}
\{\ & \mathrm{S1}(i) \rightarrow (i, 0, 0) && \mid 0 \leq i < n; \\
& \mathrm{S2}(i, j) \rightarrow (i, 1, j) && \mid 0 \leq j < i < n; \\
& \mathrm{S3}(i) \rightarrow (i, 2, 0) && \mid 0 \leq i < n\ \}
\end{aligned}
$$

AST Generation - Basic Example

$$\{\ (i,0,0) \rightarrow \mathrm{S1}(i) \quad \mid 0 \leq i < n;$$
$$(i,1,j) \rightarrow \mathrm{S2}(i,j) \quad \mid 0 \leq j < i < n;$$
$$(i,2,0) \rightarrow \mathrm{S3}(i) \quad \mid 0 \leq i < n\ \}$$

AST Generation - Basic Example

$$\{\ (i,0,0) \rightarrow \mathrm{S1}(i) \quad \mid 0 \le i < n;$$
$$(i,1,j) \rightarrow \mathrm{S2}(i,j) \quad \mid 0 \le j < i < n;$$
$$(i,2,0) \rightarrow \mathrm{S3}(i) \quad \mid 0 \le i < n\ \}$$

Project on dim. 1

$\{\ (i) \mid \mathbf{0 \le i < n}\ \}$

```
for (i = 0; i < n; i++) {
   ...
}
```

AST Generation - Basic Example

$$\{\ (i,0,0) \rightarrow \mathrm{S1}(i) \quad | \ 0 \leq i < n;$$
$$(i,1,j) \rightarrow \mathrm{S2}(i,j) \quad | \ 0 \leq j < i < n;$$
$$(i,2,0) \rightarrow \mathrm{S3}(i) \quad | \ 0 \leq i < n\ \}$$

Project on dim. 1

$\{\ (i) \mid \mathbf{0} \leq \mathbf{i} < \mathbf{n}\ \}$

Project on dim. 1, 2

$\{\ (i,t) \mid 0 \leq i < n \wedge \mathbf{0} \leq \mathbf{t} \leq \mathbf{2}\ \}$

```
for (i = 0; i < n; i++) {
  // t = 0
  S1(i);
  // t = 1
  ...
  // t = 2
  S3(i);
}
```

AST Generation - Basic Example

$$\{ (i,0,0) \to \mathrm{S1}(i) \mid 0 \le i < n;$$
$$(i,1,j) \to \mathrm{S2}(i,j) \mid 0 \le j < i < n;$$
$$(i,2,0) \to \mathrm{S3}(i) \mid 0 \le i < n \}$$

Project on dim. 1

$\{ (i) \mid \mathbf{0 \le i < n} \}$

Project on dim. 1, 2

$\{ (i,t) \mid 0 \le i < n \wedge \mathbf{0 \le t \le 2} \}$

Project on dim. 1, 2, 3

$$\{ (i,t,j) \mid 0 \le i < n \wedge 0 \le t \le 2 \wedge \mathbf{0 \le j < i} \}$$

```
for (i = 0; i < n; i++) {
  // t = 0
  S1(i);
  // t = 1
  for (j = 0; i < n; i++)
    S2(i, j);
  // t = 2
  S3(i);
}
```

Elimination of Existentially Quantified Variables

Domain

$$\{\, (t) : (\exists\, \alpha : \alpha \geq -1 + t \wedge 2\alpha \geq 1 + t \wedge \alpha \leq t \wedge 4\alpha \leq N + 2t) \,\}$$

Quantifier Elimination

$$\{\, (t) : (t \geq 3 \wedge 2t \leq 4 + N) \vee (t \leq 2 \wedge t \geq 1 \wedge 2t \leq N) \,\}$$

```
for (c0 = 1; c0 <= min(2, floordiv(N, 2)); c0 += 1)
  // body
for (c0 = 3; c0 <= floordiv(N, 2) + 2; c0 += 1)
  // body
```

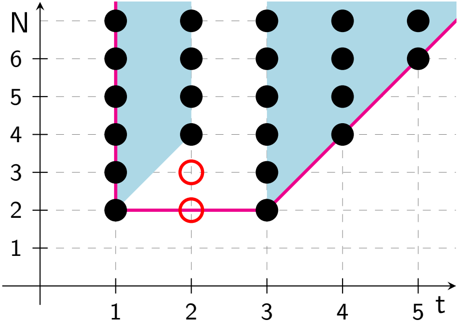
Fourier-Motzkin (Rational Quantifier Elimination)

$$\{\, (t) : 2t \leq 4 + N \wedge N \geq 2 \wedge t \geq 1 \,\}$$

```
for (c0 = 1; c0 <= floordiv(N, 2) + 2; c0 += 1)
  // body
```

Elimination of Existentially Quantified Dimensions

QE: $\{ (t) : (t \geq 3 \wedge 2t \leq 4 + N) \vee (t \leq 2 \wedge t \geq 1 \wedge 2t \leq N) \}$
FM: $\{ (t) : 2t \leq 4 + N \wedge N \geq 2 \wedge t \geq 1 \}$



Two more points in FM: $\{ (2) : 2 \leq N \leq 3 \}$

- Simple code at outer levels → Fourier-Motzkin
- No approximation at innermost level → Quant. Elimination

Semantic Unrolling

Domain: $\{i \mid 0 \leq i < 1000 \wedge N \leq i < N + 4\}$

Semantic Unrolling

Domain: $\{i \mid 0 \leq i < 1000 \land N \leq i < N + 4\}$

Lower Bound: $0 \leq i$

```
if (N <= 0 && 0 < N + 4)
  S(0);
if (N <= 1 && 1 < N + 4)
  S(1);
if (N <= 2 && 2 < N + 4)
  S(2);
if (N <= 3 && 3 < N + 4)
  S(3);
...
if (N <= 999 && 999 < N + 4)
  S(999);
```

Lower Bound: $N \leq i$

```
if (N >= 0 && N <= 999)
  S(N);
if (N >= -1 && N <= 998)
  S(N + 1);
if (N >= -2 && N <= 997)
  S(N + 2);
```

Isolation

Domain: $\{(i) \mid m \leq i < n\}$
Schedule: $\{(i) \rightarrow (i)\}$

```
for (i = m; i < n; i++)
  A(i);
```

Isolation

Domain: $\{(i) \mid m \leq i < n\}$
Schedule: $\{(i) \rightarrow (4\lfloor i/4 \rfloor), i)\}$

```
for (c0 = 4 * floordiv(m, 4); c0 < n; c0 += 4)
  for (c1 = max(m, c0); c1 <= min(n - 1, c0 + 3); c1 += 1)
    A(c1);
```

Isolation

Domain: $\{(i) \mid m \leq i < n\}$
Schedule: $\{(i) \to (4\lfloor i/4 \rfloor, i)\}$, **Isolate:** $\{(t) \mid m \leq t \wedge t + 3 < n\}$

```
// Before
if (n >= m + 4)
  for (c1 = m; c1 <= 4 * floordiv(m - 1, 4) + 3; c1 += 1)
    S(c1);
// Main
for (c0 = 4 * floordiv(m - 1, 4) + 4; c0 < n - 3; c0 += 4)
  for (c1 = c0; c1 <= c0 + 3; c1 += 1)
    S(c1);
// After
if (n >= m + 4 && 4 * floordiv(n - 1, 4) + 3 >= n) {
  for (c1 = 4 * floordiv(n - 1, 4); c1 < n; c1 += 1)
    S(c1);
} else if (m + 3 >= n)
  // Other
  for (c0 = 4 * floordiv(m, 4); c0 < n; c0 += 4)
    for (c1 = max(m, c0); c1 <= min(n - 1, c0 + 3); c1 += 1)
      S(c1);
```

AST Expression Generation

Piecewise Affine Expr.	**AST Expression**
$(i) \rightarrow (\lfloor i/4 \rfloor)$	$\rightarrow$ `floordiv(i, 4)`
$(i) \rightarrow (i \bmod 4)$	$\rightarrow$ `i - 4 * floordiv(i, 4)`

AST Expression Generation

Piecewise Affine Expr.

$(i) \to (\lfloor i/4 \rfloor)$

$(i) \to (i \bmod 4)$

AST Expression

→ `floordiv(i, 4)`

→ `i - 4 * floordiv(i, 4)`

C implementation

```
#define floordiv(n, d) \
        (((n)<0) ? -((-(n)+(d)-1)/(d)) : (n)/(d))
```

AST Expression Generation

Piecewise Affine Expr.	AST Expression
$(i) \to (\lfloor i/4 \rfloor)$	→ `floordiv(i, 4)`
$(i) \to (i \bmod 4)$	→ `i - 4 * floordiv(i, 4)`

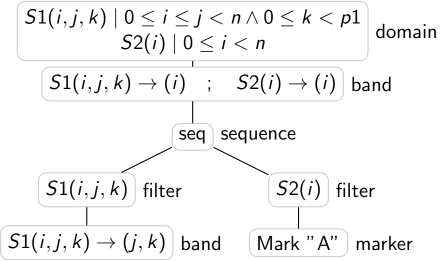
C implementation

```
#define floordiv(n, d) \
        (((n)<0) ? -((-(n)+(d)-1)/(d)) : (n)/(d))
```

Pw. Aff. Expr.	Context	AST Expression
$(i) \to (\lfloor i/4 \rfloor)$	$i \geq 0$	→ `i / 4`
	$i \leq 0$	→ `-((-i + 3) / 4)`
	$i \bmod 4 = 0$	→ `i / 4`
$(i) \to (i \bmod 4)$	$i \geq 0$	→ `i % 4`
	$i \leq 0$	→ `-((-i + 3) % 4) + 3`

Schedule Trees - A structured schedule representation

```
  for (i = 0; i < n; i++) {
    for (j = i; j < n; j++)
      for (k = 0; k < p1 ; k++)
S1:     A[i][j] = k * B[i]

    // Mark "A"
S2: A[i][i] = A[i][i] / B[i];
  }
```



Example - Start

$S1(i,j,k) \mid 0 \leq i \leq j < n \wedge 0 \leq k < p1$ domain

$S1(i,j,k) \rightarrow (i,j,k)$ band

```
for (i = 0; i < n; i++)
  for (j = i; j < n; j++)
    for (k = 0; k < n ; k++)
S1:   S(i,j,k)
```

Example - Tiling

$S1(i,j,k) \mid 0 \leq i \leq j < n \wedge 0 \leq k < p1$ domain

$S1(i,j,k) \rightarrow (\lfloor i/128 \rfloor, \lfloor j/128 \rfloor, \lfloor k/128 \rfloor)$ band

$S1(i,j,k) \rightarrow (i\%128, j\%128, k\%128)$ band

```
for (c0 = 0; c0 < n; c0 += 128)
 for (c1 = 0; c1 < n; c1 += 128)
  for (c2 = 0; c2 < n; c2 += 128)
   for (c3 = 0;
        c3 <= min(127, n - c0 - 1);
        c3 += 1)
    for (c4 = 0;
         c4 <= min(127, n - c1 - 1);
         c4 += 1)
     for (c5 = 0;
          c5 <= min(127, n - c2 - 1);
          c5 += 1)
        S1(c0 + c3, c1 + c4, c2 + c5);
```

Example - Split

$S1(i,j,k) \mid 0 \leq i \leq j < n \wedge 0 \leq k < p1$ domain

$S1(i,j,k) \rightarrow (\lfloor i/128 \rfloor, \lfloor j/128 \rfloor, \lfloor k/128 \rfloor)$ band

$S1(i,j,k) \rightarrow (i\%128)$ band

$S1(i,j,k) \rightarrow (j\%128)$ band

$S1(i,j,k) \rightarrow (k\%128)$ band

```
for (c0 = 0; c0 < n; c0 += 128)
 for (c1 = 0; c1 < n; c1 += 128)
  for (c2 = 0; c2 < n; c2 += 128)
   for (c3 = 0;
        c3 <= min(127, n - c0 - 1);
        c3 += 1)
    for (c4 = 0;
         c4 <= min(127, n - c1 - 1);
         c4 += 1)
     for (c5 = 0;
          c5 <= min(127, n - c2 - 1);
          c5 += 1)
            S1(c0 + c3, c1 + c4, c2 + c5);
```

Example - Strip-mine and interchange

- $S1(i,j,k) \mid 0 \leq i \leq j < n \wedge 0 \leq k < p1$ domain
- $S1(i,j,k) \rightarrow (\lfloor i/128 \rfloor, \lfloor j/128 \rfloor, \lfloor k/128 \rfloor)$ band
- $S1(i,j,k) \rightarrow (i\%128)$ band
- $S1(i,j,k) \rightarrow (\lfloor (j\%128)/8 \rfloor)$ band
- $S1(i,j,k) \rightarrow (k\%128)$ band
- $S1(i,j,k) \rightarrow (j\%8)$ band

```
[...]
   for (c3 = 0;
        c3 <= min(127, n - c0 - 1);
        c3 += 1)
    for (c4 = 0;
         c4 <= min(127, n - c1 - 1);
         c4 += 1)
     for (c5 = 0;
          c5 <= min(127, n - c2 - 1);
          c5 += 1)
      // SIMD Parallel Loop
      // at most 8 iterations
      for (c6 = 0;
           c6 <= min(7, n - c1 - c4 - 1);
           c6 += 1)
        S1(c0 + c3, c1 + c4 + c6, c2 + c5);
```

Example - Isolate Core Computation

$S1(i,j,k) \mid 0 \leq i \leq j < n \wedge 0 \leq k < p1$ domain

$S1(i,j,k) \rightarrow (\lfloor i/128 \rfloor, \lfloor j/128 \rfloor, \lfloor k/128 \rfloor)$ band

$S1(i,j,k) \rightarrow (i\%128)$ band

$S1(i,j,k) \rightarrow (\lfloor (j\%128)/8 \rfloor)$ $\{isolate[[a,b,c,d] \rightarrow [e]] : b < \lfloor n/128 \rfloor\}$ band

$S1(i,j,k) \rightarrow (k\%128)$ band

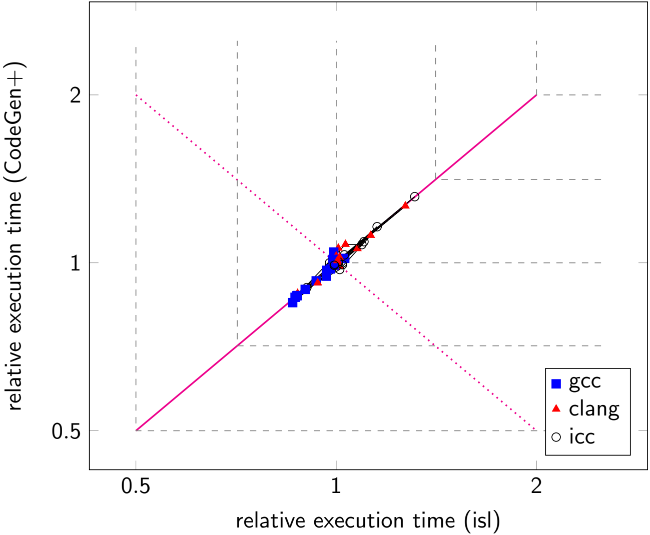
$S1(i,j,k) \rightarrow (j\%8)$ band

```
[...]
for (c3 = 0;
     c3 <= min(127, n - c0 - 1);
     c3 += 1)
 if (n >= 128 * c1 + 128) {
  for (c4 = 0; c4 <= 127; c4 += 8)
   for (c5 = 0;
        c5 <= min(127, n - c2 - 1); c5 += 1)

     // SIMD Parallel Loop
     // Exactly 8 Iterations
     for (c6 = 0; c6 <= 7; c6 += 1)
      S1(c0 + c3, c1 + c4 + c6, c2 + c5);

 } else {
   // Handle remainder
```

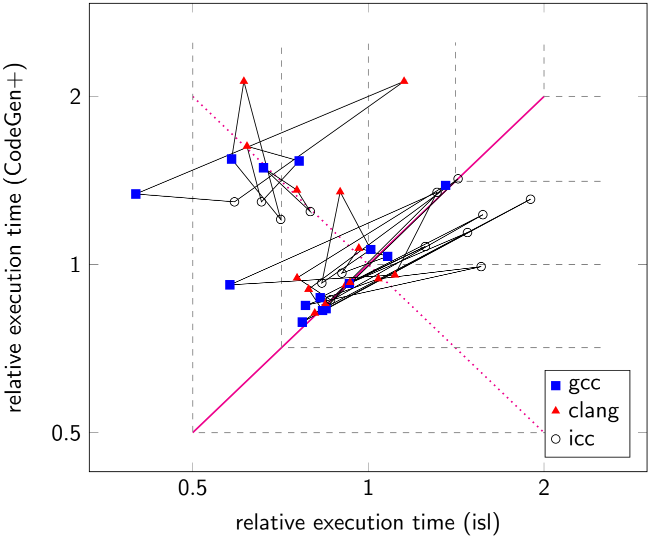
Experimental Evaluation

Robustness

Generated Code Performance – Consistent Performance



Generated Code Performance – Outliers



Code Quality: youcefn [Bastoul 2004]

CLooG 0.14.1

```
for(i=1; i<=n-2; i++) {
  S0(i,i);
  S1(i,i);
  for(j=i+1; j<=n-1; j++)
    S1(i,j);
  S1(i,n);
  S2(i,n);
}
S0(n-1,n-1);
S1(n-1,n-1);
S1(n-1,n);
S2(n-1,n);
S0(n,n);
S1(n,n);
S2(n,n);
for (i=n+1; i <= m; i++)
  S3(i,j);
```

Code Quality: youcefn [Bastoul 2004]

CLooG 0.14.1

```
for(i=1; i<=n-2; i++) {
  S0(i,i);
  S1(i,i);
  for(j=i+1; j<=n-1; j++)
    S1(i,j);
  S1(i,n);
  S2(i,n);
}
S0(n-1,n-1);
S1(n-1,n-1);
S1(n-1,n);
S2(n-1,n);
S0(n,n);
S1(n,n);
S2(n,n);
for (i=n+1; i <= m; i++)
  S3(i,j);
```

CodeGen+

```
for(i=1; i<=m; i++) {
  if(i>=n +1) {
    S2(i,n);
  } else {
    S0(i,i);
    S1(i,i);
    if (i>=n)
      S2 (i,i);
  }
  for(j=i+1; j<=n-1; j++)
    S0(i,j);
  if(n >= i+1) {
    S0(i,n);
    S2(i,n);
  }
}
```

Code Quality: youcefn [Bastoul 2004]

CLooG 0.14.1

```
for(i=1; i<=n-2; i++) {
  S0(i,i);
  S1(i,i);
  for(j=i+1; j<=n-1; j++)
    S1(i,j);
  S1(i,n);
  S2(i,n);
}
S0(n-1,n-1);
S1(n-1,n-1);
S1(n-1,n);
S2(n-1,n);
S0(n,n);
S1(n,n);
S2(n,n);
for (i=n+1; i <= m; i++)
  S3(i,j);
```

CodeGen+

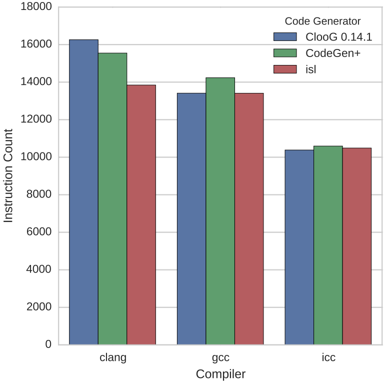
```
for(i=1; i<=m; i++) {
  if(i>=n +1) {
    S2(i,n);
  } else {
    S0(i,i);
    S1(i,i);
    if (i>=n)
      S2 (i,i);
  }
  for(j=i+1; j<=n-1; j++)
    S0(i,j);
  if(n >= i+1) {
    S0(i,n);
    S2(i,n);
  }
}
```

isl codegen

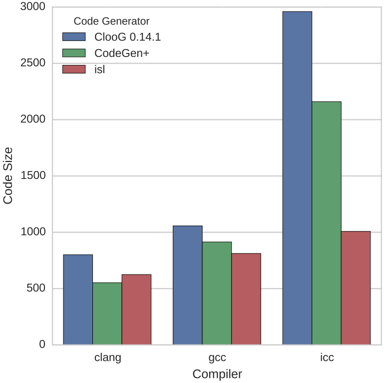
```
for (c0=1;c0<=n;c0+=1) {
  S0(c0, c0);
  for (c1=c0;c1<=n;c1+=1)
    S1(c0, c1);
  S2(c0, n);
}
for (c0=n+1;c0<=m;c0+=1)
  S2(c0, n);
```

youcefn [Bastoul 2004] - Statistics

Instruction Count



Code Size



Code Quality: [Chen 2012] - Figure 8(b)

CLooG 0.18.1

```
if (n >= 2)
  for (i = 2; i <= n; i += 2) {
    if (i%4 == 0)
      S0(i);
    if ((i+2)%4 == 0)
      S1(i);
  }
```

Code Quality: [Chen 2012] - Figure 8(b)

CLooG 0.18.1

```
if (n >= 2)
  for (i = 2; i <= n; i += 2) {
    if (i%4 == 0)
      S0(i);
    if ((i+2)%4 == 0)
      S1(i);
  }
```

CodeGen+

```
#define intMod(a,b) ((a) >= 0 ? (a) % (b) : (b) - abs((a) % (b)) % (b))
for(i = 2; i <= n; i += 2)
  if (intMod(i,4) == 0)
    S0(i);
  else
    S1(i);
```

Code Quality: [Chen 2012] - Figure 8(b)

CLooG 0.18.1

```
if (n >= 2)
  for (i = 2; i <= n; i += 2) {
    if (i%4 == 0)
      S0(i);
    if ((i+2)%4 == 0)
      S1(i);
  }
```

isl codegen

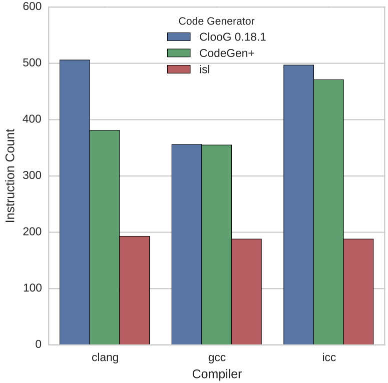
```
for (c0 = 2; c0 < n - 1; c0 += 4) {
  S1(c0);
  S0(c0 + 2);
}
if (n >= 2 && n % 4 >= 2)
  S1(-(n % 4) + n + 2);
```

CodeGen+

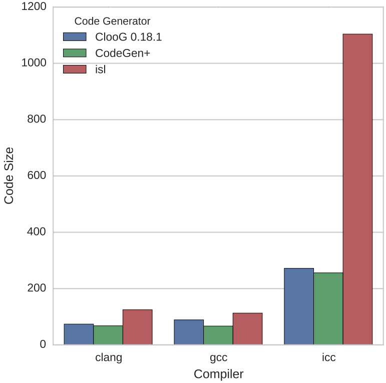
```
#define intMod(a,b) ((a) >= 0 ? (a) % (b) : (b) - abs((a) % (b)) % (b))
for(i = 2; i <= n; i += 2)
  if (intMod(i,4) == 0)
    S0(i);
  else
    S1(i);
```

[Chen 2012] - Figure 8(b) - Statistics

Instruction Count

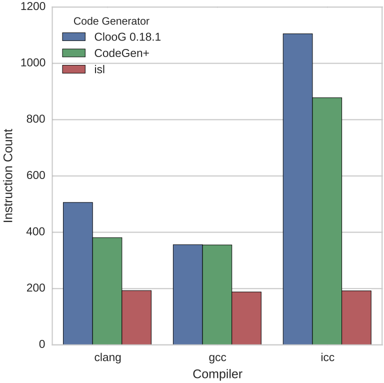


Code Size

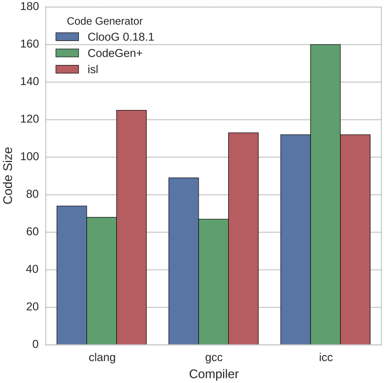


[Chen 2012] - Figure 8(b) - Statistics (-no-vec, -no-unroll)

Instruction Count



Code Size



Modulo and Existentially Quantified Variables

CodeGen+

```
// Simple
for(i = intMod(n,128); i <= 127; i += 128)
  S(i);

// Shifted
for(i = 7+intMod(t1-7,128); i <= 134; i += 128)
  S(i);

// Conditional
for(i = 7+intMod(t1-7,128); i <= 130; i += 128)
  S(i);
```

Modulo and Existentially Quantified Variables

CodeGen+

```
// Simple
for(i = intMod(n,128); i <= 127; i += 128)
  S(i);

// Shifted
for(i = 7+intMod(t1-7,128); i <= 134; i += 128)
  S(i);

// Conditional
for(i = 7+intMod(t1-7,128); i <= 130; i += 128)
  S(i);
```

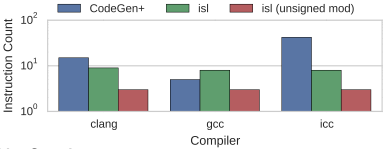
isl codegen

```
// Simple
S(n % 128);

// Shifted
S(((t1 + 121) % 128) + 7);

// Conditional
if ((t1 + 121) % 128 <= 123)
  S(((t1 + 125) % 128) + 3);
```

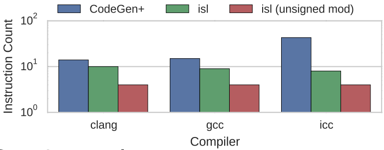
Modulo and Existentially Quantified Variables - Statistics

Instruction Count

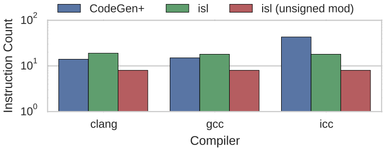
Simple



Shifted

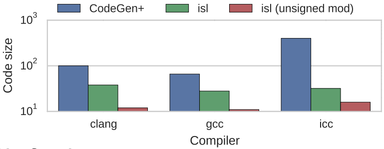


Conditional

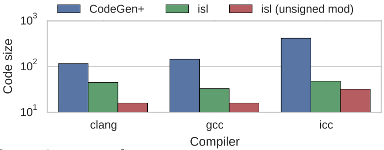


Code Size

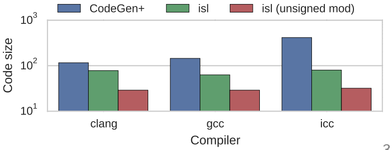
Simple



Shifted



Conditional



Polyhedral Unrolling

Normal loop code

```
// Two e.q. variables
for (c0 = 0; c0 <= 7; c0 += 1)
  if (2 * (2 * c0 / 3) >= c0)
    S(c0);

// Multiple bounds
for (c0 = 0; c0 <= 1; c0 += 1)
  for (c1 = max(t1 - 384, t2 - 514);
       c1 < t1 - 255; c1 += 1)
    if (c1 + 256 == t1 ||
        (t1 >= 126 && t2 <= 255 &&
         c1 + 384 == t1) ||
        (t2 == 256 && c1 + 384 == t1))
      S(c0, c1);
```

Polyhedral Unrolling

Normal loop code

```
// Two e.g. variables
for (c0 = 0; c0 <= 7; c0 += 1)
  if (2 * (2 * c0 / 3) >= c0)
    S(c0);

// Multiple bounds
for (c0 = 0; c0 <= 1; c0 += 1)
  for (c1 = max(t1 - 384, t2 - 514);
       c1 < t1 - 255; c1 += 1)
    if (c1 + 256 == t1 ||
        (t1 >= 126 && t2 <= 255 &&
         c1 + 384 == t1) ||
        (t2 == 256 && c1 + 384 == t1))
      S(c0, c1);
```

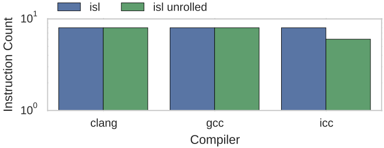
Unrolled

```
// Two e.g. variables
S(0); S(2); S(3);
S(4); S(5); S(6); S(7);

// Multiple bounds
if (t1 >= 126)
  S(0, t1 - 384);
S(0, t1 - 256);
if (t1 >= 126)
  S(1, t1 - 384);
S(1, t1 - 256);
```

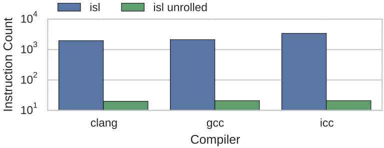
Polyhedral Unrolling - Statistics

Instruction Count

Two e.q. variables

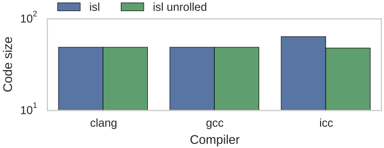


Multiple Bounds

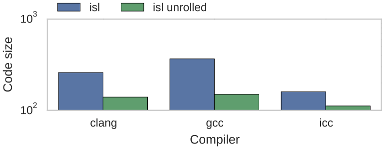


Code Size

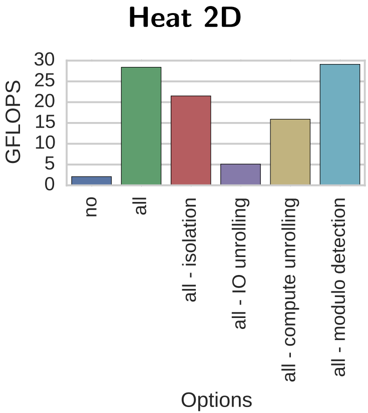
Two e.q. variables

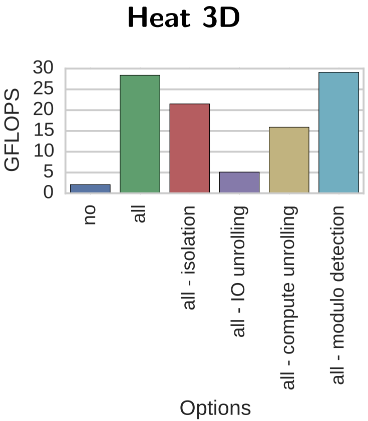


Multiple Bounds



AST Generation Strategies for Hybrid-Hexagonal Tiling





Hybrid hexagonal/classical tiling for GPUs, Tobias Grosser, Albert Cohen, Justin Holewinski, P. Sadayappan, Sven Verdoolaege, International Symposium on Code Generation and Optimization (CGO'14)
Hardware: NVIDIA NVS 5200M GPU, CUDA 5.5

AST Generation beyond Polyhedral Scanning

- Complete support for Presburger Relations
  - Existentially quantified variables
  - Piecewise schedules
- Aggressive simplification of AST expressions
- Stride and component detection
- Fine-grained options: code-size vs. control
- Specialization:
  - Polyhedral unrolling
  - User-directed versioning
- AST generation from structured schedules

http://playground.pollylabs.org