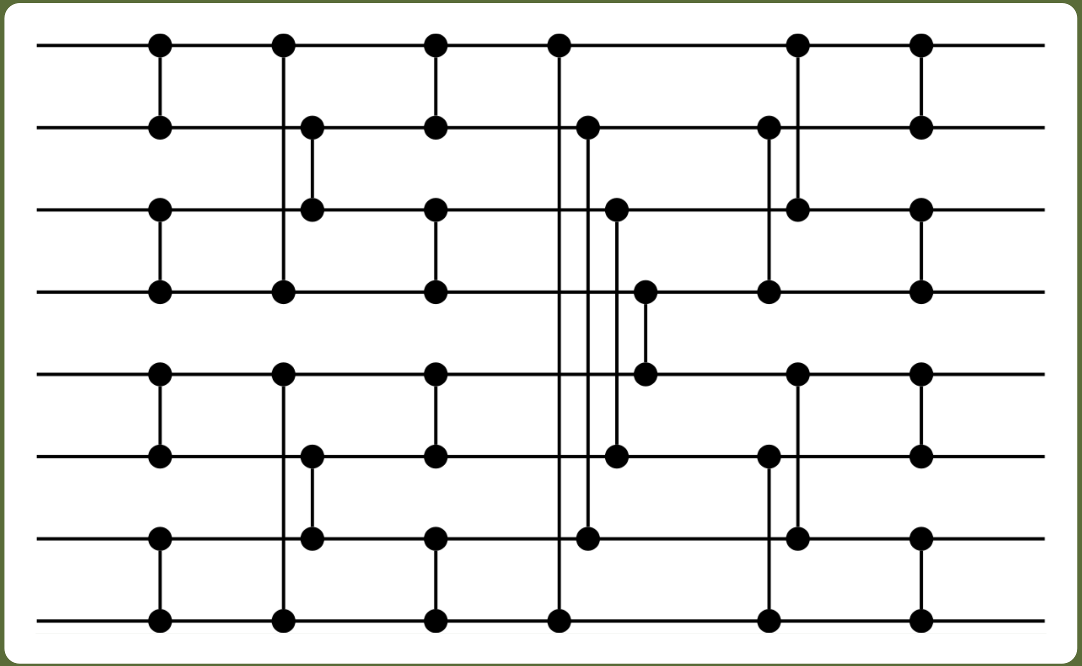
Generation of Fast and Parallel Code in LLVM

Tobias Grosser

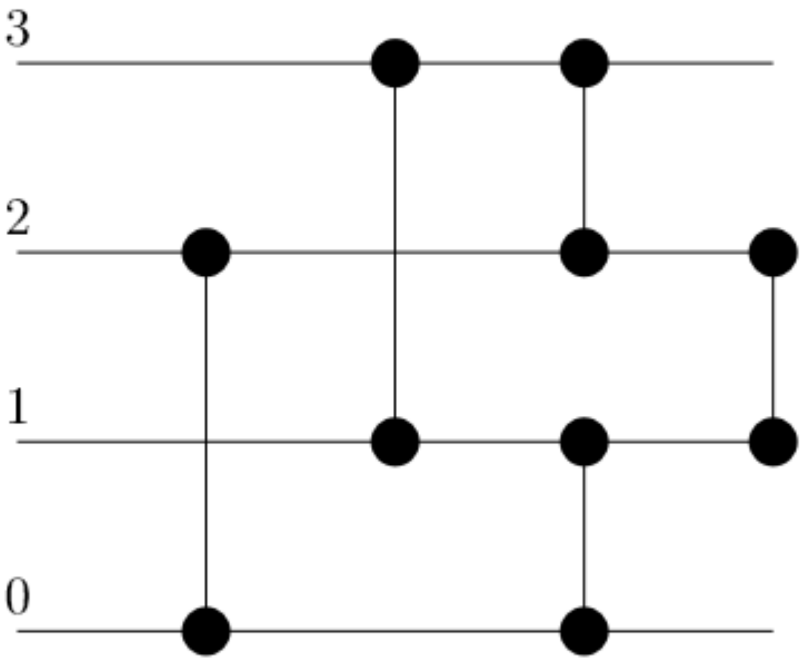
SPCL

LLVM and Clang Summer School
Paris, June 2017

Sorting Networks and the LLVM Vector IR



Sorting Networks

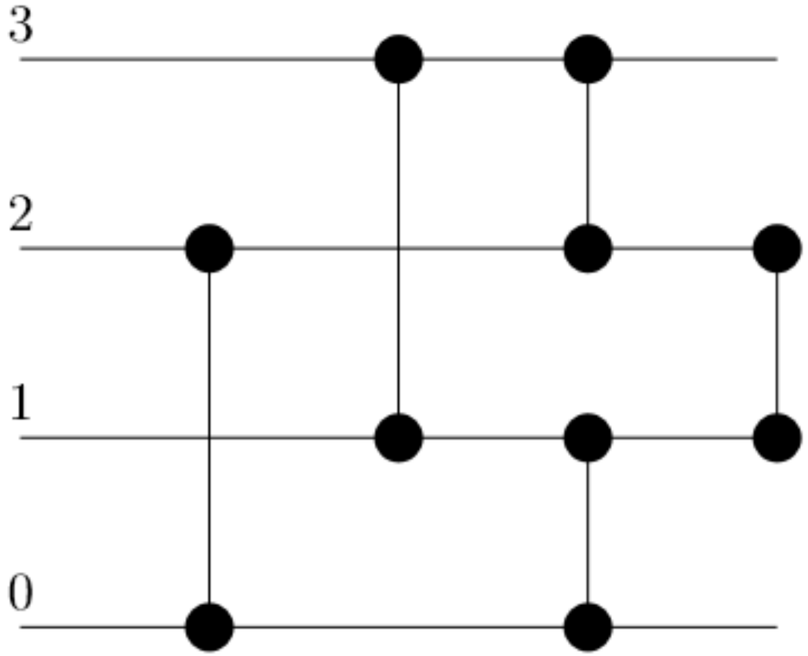


- Sorts a sequence of values
- Fixed sets of **swaps**
- Not data-dependent
- Highly parallel

Optimality of Sorting Networks

N	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
Depth	0	1	3	3	5	5	6	6	7	7	8	8	9	9	9	9	10
Size, upper bound	0	1	3	5	9	12	16	19	25	29	35	39	45	51	56	60	71
Size, lower bound											33	37	41	45	49	53	58

Exercise: Generate Code for a N=4 Batcher's Merge-Exchange Network



(0,2)	**(1,3)**
(0,1)	**(2,3)**
(1,2)	(0,3)

Implementation with C/C++ Vector Instructions

```
double4 __attribute__ ((noinline)) sort(double4 A) {
  double2 InLeft, InRight, Min, Max;

  // [[0,2],[1,3]]
  InLeft = __builtin_shufflevector(A, A, 0, 1);
  InRight = __builtin_shufflevector(A, A, 2, 3);

  Min = min(InLeft, InRight);
  Max = max(InLeft, InRight);

  A = __builtin_shufflevector(Min, Max, 0, 1, 2, 3);

  // [[0,1],[2,3]]
  InLeft = __builtin_shufflevector(A, A, 0, 2);
  InRight = __builtin_shufflevector(A, A, 1, 3);

  Min = min(InLeft, InRight);
  Max = max(InLeft, InRight);

  A = __builtin_shufflevector(Min, Max, 0, 2, 1, 3);

  // [[0,2],[1,3]]
  InLeft = __builtin_shufflevector(A, A, 1, 0);
  InRight = __builtin_shufflevector(A, A, 2, 3);

  Min = min(InLeft, InRight);
  Max = max(InLeft, InRight);

  A = __builtin_shufflevector(Min, Max, 1, 0, 2, 3);

  return A;
}
```

LLVM-IR implementation for N = 4

```
define <4 x double> @_Z4sortDv4_d(<4 x double> %A) local_unnamed_addr #2 {
entry:
%shuffle = shufflevector <4 x double> %A, <4 x double> undef, <2 x i32> <i32 0, i32 1>
%shuffle1 = shufflevector <4 x double> %A, <4 x double> undef, <2 x i32> <i32 2, i32 3>
%0 = tail call <2 x double> @llvm.x86.sse2.min.pd(<2 x double> %shuffle, <2 x double> %shuffle1) #9
%1 = tail call <2 x double> @llvm.x86.sse2.max.pd(<2 x double> %shuffle, <2 x double> %shuffle1) #9

%shuffle4 = shufflevector <2 x double> %0, <2 x double> %1, <2 x i32> <i32 0, i32 2>
%shuffle5 = shufflevector <2 x double> %0, <2 x double> %1, <2 x i32> <i32 1, i32 3>
%2 = tail call <2 x double> @llvm.x86.sse2.min.pd(<2 x double> %shuffle4, <2 x double> %shuffle5) #9
%3 = tail call <2 x double> @llvm.x86.sse2.max.pd(<2 x double> %shuffle4, <2 x double> %shuffle5) #9

%shuffle8 = shufflevector <2 x double> %2, <2 x double> %3, <4 x i32> <i32 0, i32 2, i32 1, i32 3>
%shuffle9 = shufflevector <4 x double> %shuffle8, <4 x double> undef, <2 x i32> <i32 1, i32 0>
%shuffle10 = shufflevector <4 x double> %shuffle8, <4 x double> undef, <2 x i32> <i32 2, i32 3>
%4 = tail call <2 x double> @llvm.x86.sse2.min.pd(<2 x double> %shuffle9, <2 x double> %shuffle10) #9
%5 = tail call <2 x double> @llvm.x86.sse2.max.pd(<2 x double> %shuffle9, <2 x double> %shuffle10) #9

%shuffle13 = shufflevector <2 x double> %4, <2 x double> %5, <4 x i32> <i32 1, i32 0, i32 2, i32 3>
ret <4 x double> %shuffle13  }
```

Assembly Code

```
vextractf128 $1, %ymm0, %xmm1
vminpd %xmm1, %xmm0, %xmm2
vmaxpd %xmm1, %xmm0, %xmm0
vunpcklpd    %xmm0, %xmm2, %xmm1 # xmm1 = xmm2[0],xmm0[0]
vunpckhpd    %xmm0, %xmm2, %xmm0 # xmm0 = xmm2[1],xmm0[1]
vminpd %xmm0, %xmm1, %xmm2
vmaxpd %xmm0, %xmm1, %xmm0
vunpcklpd    %xmm2, %xmm0, %xmm1 # xmm1 = xmm0[0],xmm2[0]
vunpckhpd    %xmm0, %xmm2, %xmm0 # xmm0 = xmm2[1],xmm0[1]
vminpd %xmm0, %xmm1, %xmm2
vmaxpd %xmm0, %xmm1, %xmm0
vpermilpd    $1, %xmm2, %xmm1 # xmm1 = xmm2[1,0]
vinsertf128  $1, %xmm0, %ymm1, %ymm0
```

Assembly Code : N = 8 (extract only)

LLVM selects "optimal" instruction sequences

```
vminpd %ymm1, %ymm0, %ymm2
vmaxpd %ymm1, %ymm0, %ymm0
vinsertf128  $1, %xmm0, %ymm2, %ymm1
vperm2f128   $49, %ymm0, %ymm2, %ymm0 # ymm0 = ymm2[2,3],ymm0[2,3]
vminpd %ymm0, %ymm1, %ymm2
vmaxpd %ymm0, %ymm1, %ymm0
vinsertf128  $1, %xmm2, %ymm0, %ymm1
vpermilpd    $2, %ymm0, %ymm3 # ymm3 = ymm0[0,1,2,2]
vblendpd     $4, %ymm1, %ymm3, %ymm1 # ymm1 = ymm3[0,1],ymm1[2],ymm3[3]
vperm2f128   $35, %ymm2, %ymm0, %ymm2 # ymm2 = ymm2[2,3,0,1]
vpermilpd    $6, %ymm2, %ymm2 # ymm2 = ymm2[0,1,3,2]
vblendpd     $8, %ymm0, %ymm2, %ymm0 # ymm0 = ymm2[0,1,2],ymm0[3]
vminpd %ymm0, %ymm1, %ymm2
vmaxpd %ymm0, %ymm1, %ymm0
```

Inner Loop Vectorization

```
for (int i = 0; i < 1024; i++)
    B[i] += A[i];
```

↓

```
for (int i = 0; i < 1024; i+=4)
    B[i:i+3] += A[i:i+3];
```

Inner Loop Vectorization

```
for (int i = 0; i < 1024; i++)
    B[i] += A[i];
```

↓

```
for (int i = 0; i < 1024; i+=4)
    B[i:i+3] += A[i:i+3];
```

Automatic (Inner) Loop Vectorization

- Validity
  - Innermost loop must be parallel (or behave after vectorization as if it was)
  - No aliasing between different arrays

- Profitability
  - Memory accesses must be “stride-one”

    or
  - Computational cost must dominate the loop

Automatic (Inner) Loop Vectorization

- Validity
  - Innermost loop must be parallel (or behave after vectorization as if it was)
  - No aliasing between different arrays

- Profitability
  - Memory accesses must be "stride-one"

    or
  - Computational cost must dominate the loop

Can these loops be vectorized?

```
for (int i = 0; i <= n; i++)
  B[i] += A[i];
```

YES, the arrays are different objects

```
for (int i = 0; i <= n; i++)
  A[i] += A[i];
```

YES, there is no dependence to any previous iteration

Can these loops be vectorized?

```
for (int i = 1; i <= n; i++)
  A[i] += A[i] + A[i – 1];
```

NO, iteration i depends on iteration i – 1

Can these loops be vectorized: pointer-to-pointer arrays

```
int[ ][ ] A = new int[N][M];
int[ ][ ] B = new int[N][M];

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

YES, in C/C++/Fortran array dimensions are independent

We now assume multi-dimensional arrays in the mathematical sense

Can these loops be vectorized?

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

NO, the inner loop has data-dependences between subsequent iterations

Can these loops be vectorized?

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

Interchange

YES, the inner loop has no data-dependences between subsequent iterations

Advanced Support for SIMDization

Target Transform Info [include/llvm/Analysis/TargetTransformInfo.h]

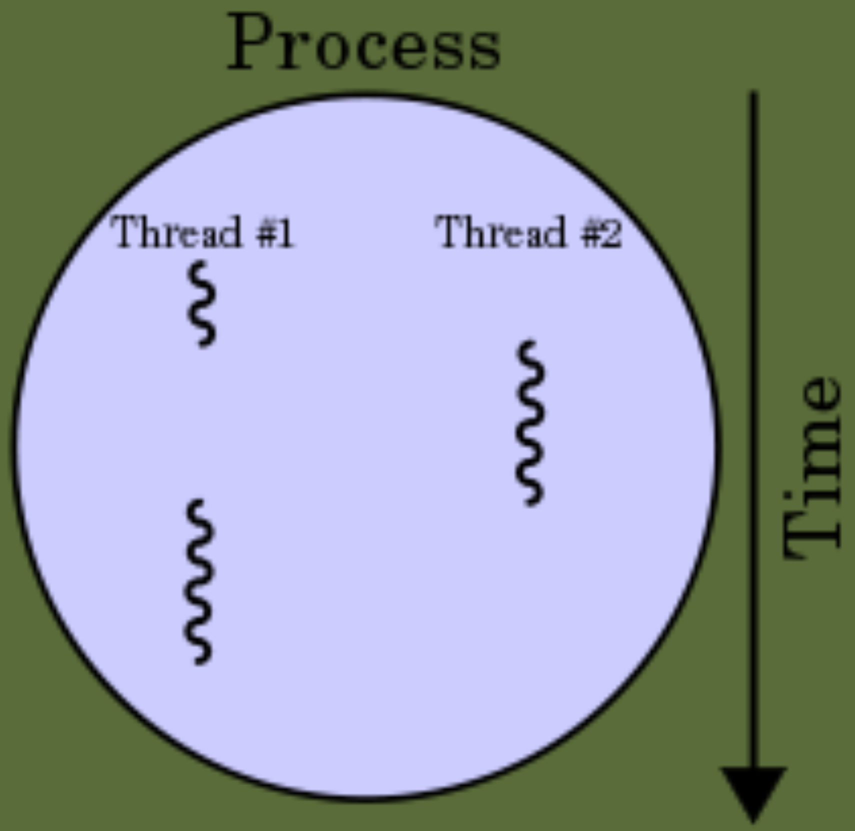
Target Specific Cost Estimates Without Instruction Selection

```
/// \return The expected cost of arithmetic ops, such as mul, xor, fsub, etc.
/// \p Args is an optional argument which holds the instruction operands
/// values so the TTI can analyze those values searching for special
/// cases\optimizations based on those values.
int getArithmeticInstrCost(
    unsigned Opcode, Type *Ty, OperandValueKind Opd1Info = OK_AnyValue,
    OperandValueKind Opd2Info = OK_AnyValue,
    OperandValueProperties Opd1PropInfo = OP_None,
    OperandValueProperties Opd2PropInfo = OP_None,
    ArrayRef<const Value *> Args = ArrayRef<const Value *>()) const;

/// \return The cost of a shuffle instruction of kind Kind and of type Tp.
/// The index and subtype parameters are used by the subvector insertion and
/// extraction shuffle kinds.
int getShuffleCost(ShuffleKind Kind, Type *Tp, int Index = 0,
                   Type *SubTp = nullptr) const;
```

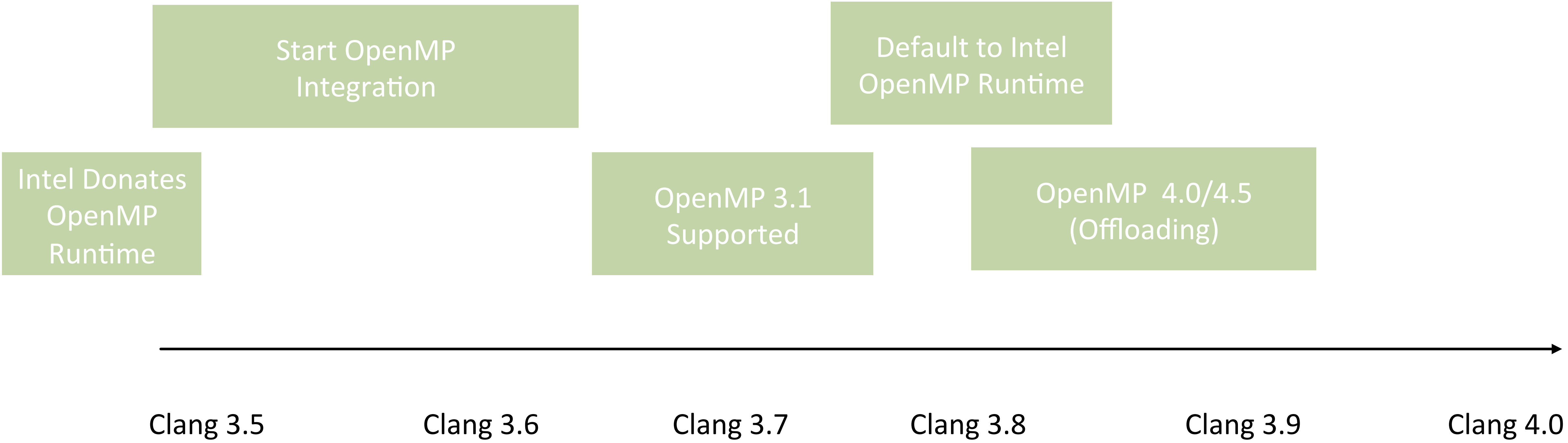
LoopAccessAnalysis

- Analyze Innermost Loops
- Check data-dependences and legality of SIMDization
- Generates run-time Alias Checks
- Analysis the Stride of Memory Accesses

OpenMP Thread Parallel Code



OpenMP support in clang



GPU Code Generation in LLVM

LLVM GPU Tools

Components

Frontends

OpenCL
CUDA

RunTime Libraries

OpenMP 4.0 / 4.5
Libclc
StreamExecutor

Backends

AMDGPU
NVPTX

End-to-end Tools

gpucc

CUDA Compiler

Pocl

OpenCL
Compiler

Beignet

Intel OpenCL
GPU Driver

RocM

AMD OpenCL
GPU Driver

The Clang OpenCL Frontend

- Major OpenCL Compilers use Clang as frontend
  - AMD, ARM, Intel, Xilinx, Altera, …
- Support for all OpenCL 2.0 features

Compile OpenCL Code with Clang

```
/*  to make Clang compatible with OpenCL */
#define __global __attribute__((address_space(1)))
int get_global_id(int index);

/* Test kernel */
__kernel void test(__global float *in, __global float *out)
{
    int index = get_global_id(0);
    out[index] = 3.14159f * in[index] + in[index];
}
```

```
clang -S -emit-llvm test.cl -o -
```

LLVM-IR for OpenCL kernel

```
define void @test(float addrspace(1)* nocapture readonly %in, float
addrspace(1)* nocapture %out) #0 {
  %1 = tail call i32 @get_global_id(i32 0) #3
  %2 = sext i32 %1 to i64
  %3 = getelementptr inbounds float, float addrspace(1)* %in, i64 %2
  %4 = load float, float addrspace(1)* %3, align 4, !tbaa !7
  %5 = tail call float @llvm.fmuladd.f32(float 0x400921FA00000000, float
%4, float %4)
  %6 = getelementptr inbounds float, float addrspace(1)* %out, i64 %2
  store float %5, float addrspace(1)* %6, align 4, !tbaa !7
  ret void
}
```

OpenCL Kernel Metadata

```
!opencl.kernels = !{!0}
!llvm.ident = !{!6}

!0 = !{void (float addrspace(1)*, float addrspace(1)*)*
@test, !1, !2, !3, !4, !5}
!1 = !{!"kernel_arg_addr_space", i32 1, i32 1}
!2 = !{!"kernel_arg_access_qual", !"none", !"none"}
!3 = !{!"kernel_arg_type", !"float*", !"float*"}
!4 = !{!"kernel_arg_base_type", !"float*", !"float*"}
!5 = !{!"kernel_arg_type_qual", !"", !""}
```

Address Spaces

- Each Load / Store belongs to a specific address space

AMD address spaces

0	1	2	3	4	5
Generic	Global	Region	Local	Constant	Private

SPIR: OpenCL at LLVM-IR level

- Subset of LLVM 3.1/3.4 IR with additional metadata
- Allows to convert OpenCL C code to a OpenCL Binary Representation
- Support:
  - Intel
  - AMD
  - Beignet

PROBLEM: Modern LLVM-IR is incompatible with SPIR

SPIR-V

- Obligator for OpenCL 2.2 and Vulkan
- Representation defined independent of LLVM-IR
- Converter to and from LLVM-IR
- Not yet part of LLVM itself
- Only supported in Intel OpenCL CPU Driver for Windows

GPG Code Generation Backends for LLVM

Open Source

- NVPTX (mainline)
- AMDGPU (mainline)
- Intel (Beignet Project)

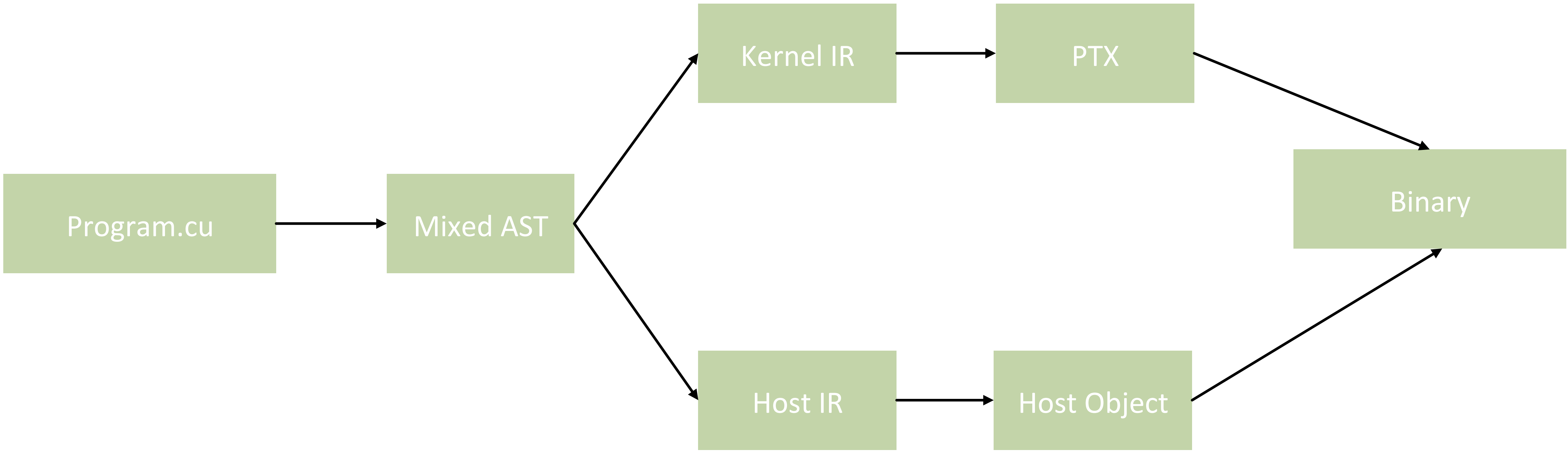
Closed Source

- ARM Mali
- Imagination Technologies / Apple
- Qualcomm
- Intel (Windows / Proprietary Driver)

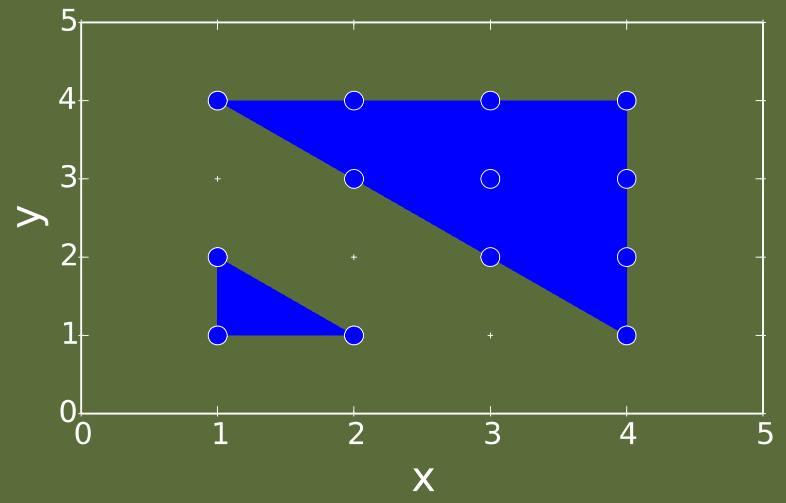
Use PTX Code with CUDA

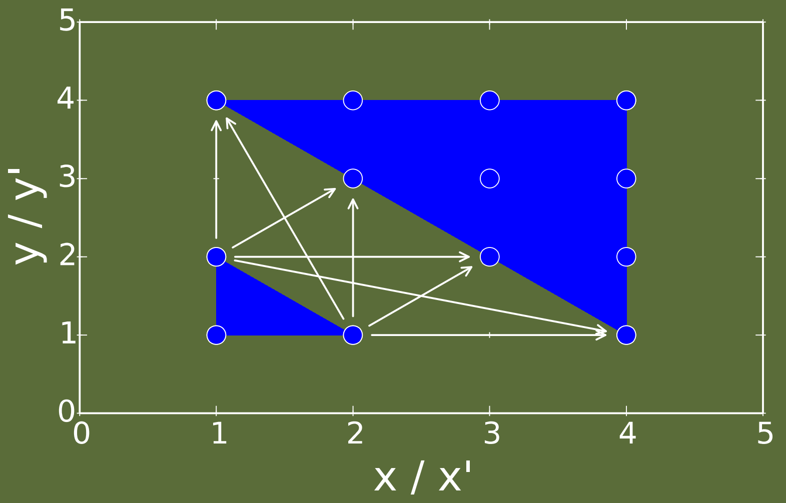
```
CuLinkCreate (6, Options, OptionVals, &LState);
Res = CuLinkAddData (LState, CU_JIT_INPUT_PTX, (void *)BTXBuffer,
                        strlen(BinaryBuffer) + 1, 0, 0, 0, 0);
CuLinkComplete(LState, &CuOut, &OutSize);
CuModuleLoadData(&(((CUDAKernel *)Function->Kernel)->CudaModule),
                        CuOut);
CuModuleGetFunction (&(((CUDAKernel *)Function->Kernel)->Cuda),
```

Clang + CUDA (Former GPUCC)



Presburger Sets and Relations





Quasi-Affine Expression

- Base
  - Constants (c_i)
  - Parameters (p_i)
  - Variables (v_i)
- Operations
  - Negation ($-e$)
  - Addition ($e_0 + e_1$)
  - Multiplication by constant ($c * e$)
  - Division by constant (c / e)
  - Remainder of constant division ($e \ mod \ c$)

```
void foo (int n, int m) {

  for (int i = 0; …; …)
       int tmp = …
       for (int j = 0; …; …) {








  }
}
```

Presburger Formula

- Base
  - Boolean Constants (⊤, ⊥)

- Operations
  - Comparisons of quasi-affine expressions

    $e_0 \oplus e_1 , \oplus \{<, \leq, =, \neq, \geq, >\}$
  - Boolean Operations between Presburger Formula

    $p_0 \otimes p_1 , \otimes \{ \wedge, \vee, \neg, \Rightarrow, \Leftarrow, \Leftrightarrow \}$
  - Quantified Variables

    $\exists x | p(x, \ldots)$

    $\forall x | p(x, \ldots)$

Presburger Sets and Relations

Presburger Set

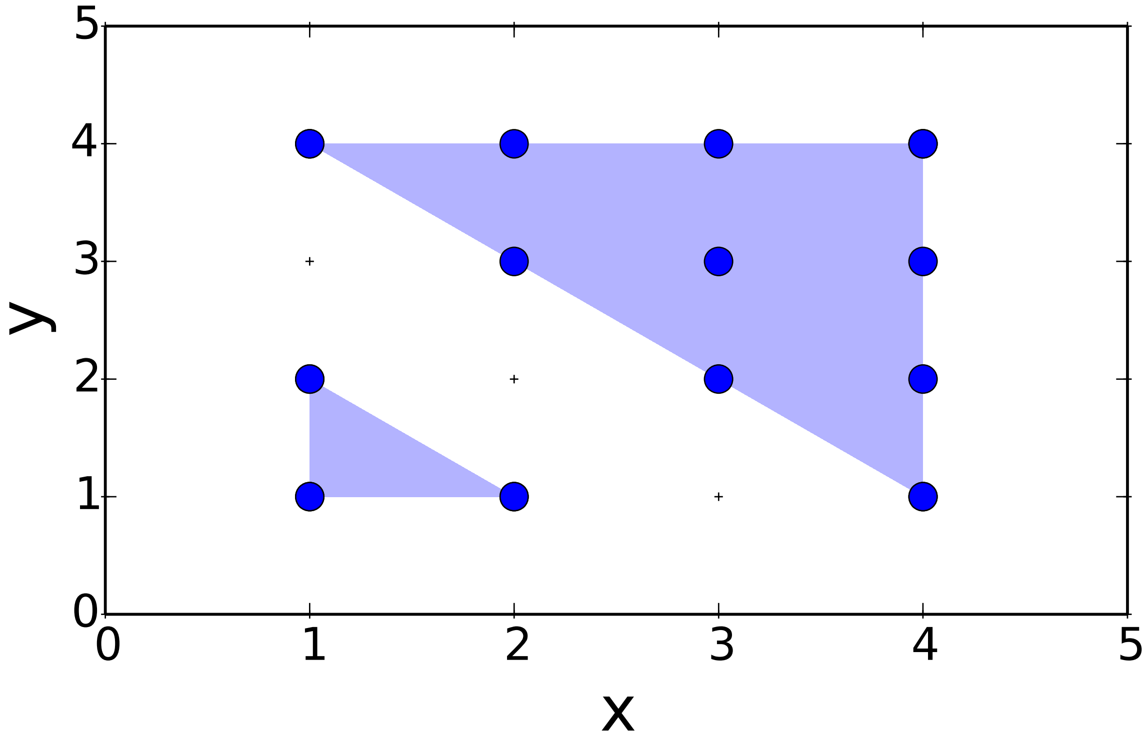
$$S = p \to \{v \mid v \in \mathbb{Z}^n : p(v, p)\}$$

Presburger Relation

$$R = p \to \{v_0 \to v_1 \mid v_0 \in \mathbb{Z}^n, v_1 \in \mathbb{Z}^m : p(v_0, v_1, p)\}$$

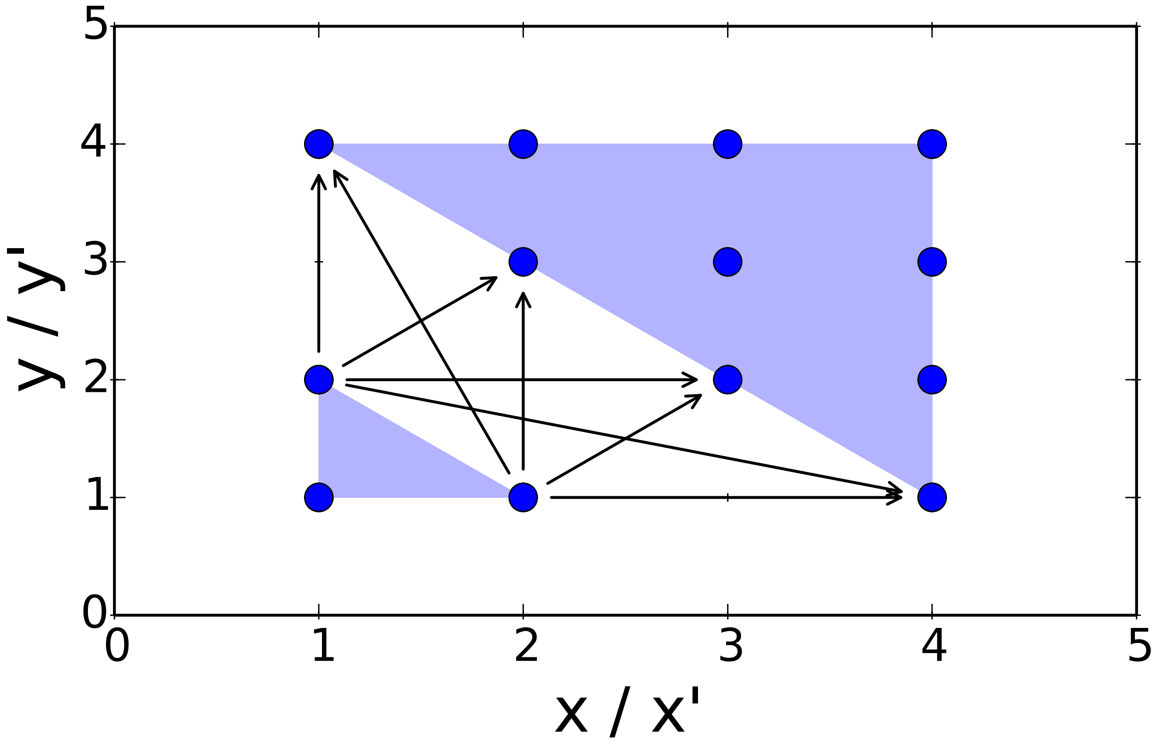
Example: Presburger Set

S = $(x,y) 1 \le x, y \le 4 \land (x+y \le 3 \lor x+y \ge 5)$



Example: Presburger Map

$$R = \{ (x,y) \rightarrow (x', y') \mid x+y=3 \wedge x'+y'=5 \}$$



Presburger Arithmetic

- Benefits
  - Decidable
  - Closed under common operations

    ∩, ∪, \, proj, ∘, not transitive hull

  Þ Precise results

- Computational Complexity
  - Some operations double-exponential (in dimensions)
  - Often lower complexity for bounded dimension

Can we solve more complex Diophantine equations?

- Does $x↑3 + y↑3 = z↑3$ with x,y,z∈ℤ have a solution?

  No, Fermat's last theorem! Answered in 1994, after year 357 years!

- Does $x↑3 + y↑3 + z↑3 = 29$ have a solutior
- Does $x↑3 + y↑3 + z↑3 = 33$ have a solutior '!

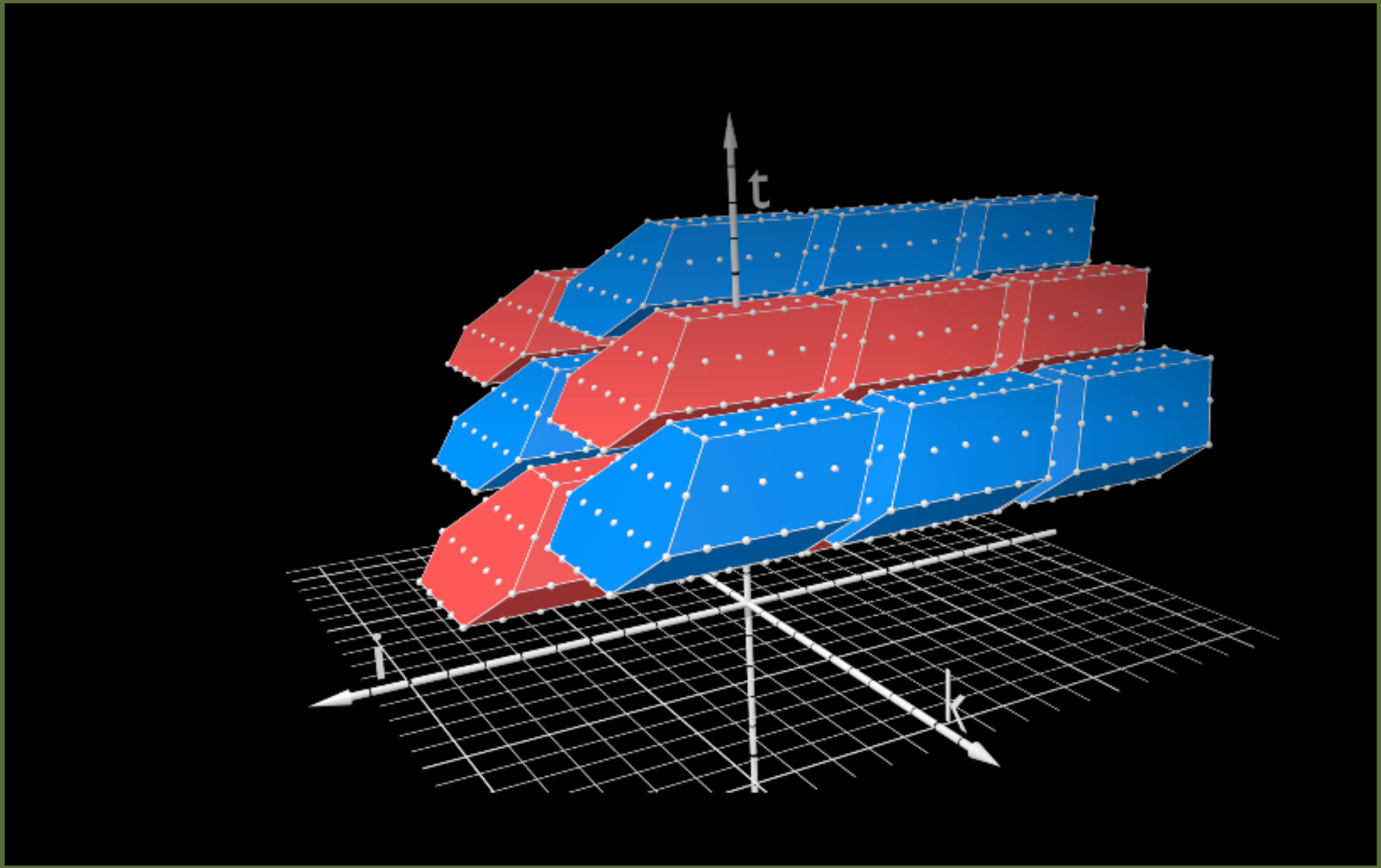
Note: No general algorithm for solving polynomial equations over integers exists! (Hilbert's 10th problem)

Proof is interesting: encodes Turing machine in Diophantine equations

Undecidability in number theory, Bjorn Poonen, Notices Amer. Math. Soc. 55 (2008)

Demo: Presburger Sets

Modeling Loop Programs with Presburger Sets



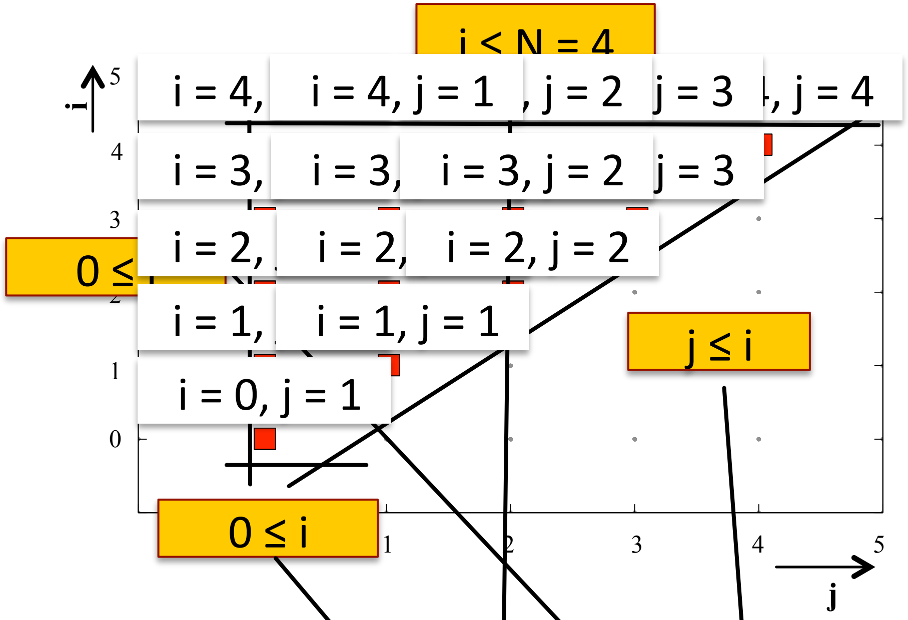
Polyhedral Loop Modeling

Program Code

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

N = 4

Iteration Space



D = { (i,j) | 0 ≤ i ≤ N ⋀ 0 ≤ j ≤ i }

Static Control Parts - SCoPs

- Structured Control
  - IF-conditions
  - Counted FOR-loops (Fortran style)
- Multi-dimensional array accesses (and scalars)
- Loop-conditions and IF-conditions are Presburger Formula
- Loop increments are constant (non-parametric)
- Array subscript expressions are piecewise-affine

Þ Can be modeled precisely with Presburger Sets

Polyhedral Model of Static Control Part

```
for (i = 0; i <= N; i++)
  for (j = 0; j <= i; j++)
S:  B[i][j] = A[i][j] + A[i][j+1];
```

- **Iteration Space (Domain)**

  $I_S = S(i,j) 0 \le i \le N \wedge 0 \le j \le i$

- **Schedule**

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

- **Access Relation**
  - Reads: $\{S(i,j) \rightarrow A(i,j); S(i,j) \rightarrow A(i, j+1)\}$
  - Writes: $\{S(i,j) \rightarrow B(i,j)\}$

Polyhedral Schedule: Original

Model

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

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

Code

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

Polyhedral Schedule: Original

Model

$$I_S = S(i,j) 0 \le i \le n \wedge 0 \le j \le i$$

$$\theta_S = \{S\,(i,j) \to (i,j)\} \quad \to (\lfloor i/4 \rfloor, j, i \bmod 4)$$

Code

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

Polyhedral Schedule: Interchanged

Model

$$I_S = S(i,j) \; 0 \leq i \leq n \wedge 0 \leq j \leq i$$

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

Code

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

Polyhedral Schedule: Strip-mined

Model

$$I_S = S(i,j)\, 0 \le i \le n \wedge 0 \le j \le i$$

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

Code

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

Polyhedral Schedule: Blocked

Model

$$I_S = S(i,j)\; 0 \le i \le n \wedge 0 \le j \le i$$

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

Code

```
for (c0 = 0; c0 <= floord(n, 4); c0++)
  for (c1 = 0; c1 <= c0; c1++)
    for (c2 = 0; c2 <= min(3, n - 4 * c0); c2++)
      for (c3 = 0; c3 <= min(3, 4 * c0 – 4 * c1 + c2); c3++)
         S(4 * c0 + c2, 4 * c1 + c3);
```

How to derive a good schedule

Stepwise Improvement	Construct "perfect" Schedule

• Interchange
• Fusion
• Distribution
• Skewing
• Tiling
• Unroll-and-Jam
	• ***Feautrier Scheduler***
	  • Resolve data-dependences at outer levels
	  • Maximize inner parallelism
	• ***Pluto Scheduler***
	  • Resolve data-dependences at inner levels
	  • Maximize outer parallelism
	  • Fusion model to minimize dependence distances

Classical Loop Transformations – Loop Reversal

```
// Original Loop
for (i = 0; i <= n; i+=1)
  S(i);
```

```
// Transformed Loop
for (i = n; i >= 0; i-=1)
  S(i);
```

$D_I = S(i) 0 \leq i \leq n$

$S_{Orig} = \{ S(i) \rightarrow (i) \}$

$S_T = \{ S(i) \rightarrow (n - i) \}$

Classical Loop Transformations – Loop Interchange

```
// Original Loop
for (i = 0; i <= n; i+=1)
  for (j = 0; j <= n; j+=1)
    S(i,j);
```

```
// Transformed Loop
for (j = 0; j <= n; j+=1)
  for (i = 0; i <= n; i+=1)
    S(i,j);
```

$D_I = S(i,j) 0 \leq i,j \leq n$

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

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

Classical Loop Transformations – Fusion

```
// Original Loop
for (i = 0; i <= n; i+=1)
  S1(i);
for (i = 0; i <= n; i+=1)
  S2(i);
```

```
// Transformed Loop
for (i = 0; i <= n; i+=1) {
  S1(i);
  S2(i);
}
```

$$D_I = S(i) 0 \leq i \leq n T(i) | 0 \leq j \leq n$$

$$S_{Orig} = \{ S(i) \rightarrow (0, i); T(i) \rightarrow (1, i) \}$$

$$S_T = \{ S(i) \rightarrow (i, 0); T(i) \rightarrow (i, 1) \}$$

Classical Loop Transformations – Fission (also called Distribution)

```
// Original Loop
for (i = 0; i <= n; i+=1) {
  S1(i);
  S2(i);
}
```

$$D_I = S(i) 0 \le i \le n T(i) | 0 \le j \le n$$
$$S_{Orig} = \{ S(i) \rightarrow (i,0); T(i) \rightarrow (i,1) \}$$
$$S_T = \{ S(i) \rightarrow (0,1); T(i) \rightarrow (1,i) \}$$

```
// Transformed Loop
for (i = 0; i <= n; i+=1)
  S1(i);
for (i = 0; i <= n; i+=1)
  S2(i);
```

Classical Loop Transformations – Skewing

```
// Original Loop
for (i = 0; i <= n; i+=1)
  for (j = 0; j <= n; j+=1)
    S(i,j);
```

$$D_I = S(i,j) 0 \leq i,j \leq n$$
$$S_{Orig} = \{ S(i,j) \rightarrow (i,j) \}$$
$$S_T = \{ S(i,j) \rightarrow (i,i+j) \}$$

```
// Transformed Loop
for (i = 0; i <= n; i+=1)
  for (j = i+1; j <=n+i; j+=1)
    S(i,j);
```

Classical Loop Transformations – Strip-Mining

```
// Original Loop
for (i = 0; i <= 1024; i+=1)
  S(i);
```

```
// Transformed Loop
for (i = 0; i <= 1024; i+=4)
  for (ii = i; ii <= i+3; ii+=1)
    S(ii);
```

$$D_I = S(i) 0 \leq i \leq n$$
$$S_{Orig} = \{ S(i) \rightarrow (i) \}$$
$$S_T = \{ S(i) \rightarrow (\lfloor i/4 \rfloor, i) \}$$

Classical Loop Transformations – Blocking (Tiling)

```
// Original Loop
for (i = 0; i <= 1024; i+=1)
  for (j = 0; j <= 1024; j+=1)
    S(i,j);
```

$$D_I = S(i,j) 0 \leq i,j \leq n$$
$$S_{Orig} = \{ S(i,j) \rightarrow (i,j) \}$$
$$S_T = \{ S(i) \rightarrow (\lfloor i/4 \rfloor, \lfloor j/4 \rfloor, i, j) \}$$

```
// Transformed Loop
for (i = 0; i <= 1024; i+=8)
  for (j = 0; j <= 1024; j+=8)
    for (ii = i; ii <= i+8; ii+=1)
     for (jj = j; jj <= j+8; j+=1)
       S(ii, jj);
```

Legality of Loop Transformations

1. **Conflicting Accesses**
   Two statement instance access the same memory location
2. **Execution**
   Each statement instance is known to be executed
3. **At least one write access**
   Two memory reads do not conflict

The direction of the data dependency is defined through the schedule.

Conditions for Data Dependence

1. **Conflicting Accesses**
   Two statement instance access the same memory location
2. **Execution**
   Each statement instance is known to be executed
3. **At least one write access**
   Two memory reads do not conflict

The direction of the data dependency is defined through the schedule.

Data Dependence Types

- **Read-After-Write (true)**
  - Flow (subset of RAW-dependences that carries data)
- **Write-After-Read (anti)**
- **Write-After-Write (output)**
- **Read-After-Read**

False dependences: Write-After-Read + Write-After-Write

Precision of Data Dependences

```
Example: for I = 0..N
           for J = 0..N
             for K = 0..N
               A(I+1, J, K-1) = A(I, J, K)
```

- **Direction Vectors**

  Dependences are tuples over: +, -, =

  D(+, =, -)

- **Distance Vectors**

  Dependences are given through their integer distance

  D(1, 0, -1)

- **Presburger Sets**

  Dependences are described as Presburger Relations

$$\{(I,J,K) \rightarrow (I+1, J, K-1) \mid 0 \leq I, J, K \leq N\}$$

Invariants on Dependences

- **The first non-zero component must be positive**

  Otherwise, the dependence goes backwards in time

Validity of a Schedule

A schedule θ_S is valid for an iteration space I_S and a set of dependences D_S, iff $\forall (s,d) \in D_S : \theta_S(s) \prec \theta_S(d)$.

Loop Carried Dependences

- A data dependence **D** is carried by a loop **L** that corresponds to the first non-zero dimension of the dependence vector

```
for (i = 0; i < N; i++)
  for (j = 0; j < M; j++)
    for (k = 0; k < K; k++)
      C[i][j] += …
```

D(0, 0, +1)

```
for (i = 0; i < N; i++)
  for (k = 0; k < K; k++)
    for (j = 0; j < M; j++)
      C[i][j] += …
```

D(0, +1, 0)

Parallel Loops

- **A loop is parallel if it does not carry any data dependences**

```
parfor (i = 0; i < N; i++)
  parfor (j = 0; j < M; j++)
    for (k = 0; k < K; k++)
      C[i][j] += …
```

D(0, 0, +1)

```
parfor (i = 0; i < N; i++)
  for (k = 0; k < K; k++)
    parfor (j = 0; j < M; j++)
      C[i][j] += …
```

D(0, +1, 0)

Elimination of Scalar Dependences: Static Array Expansion

```
for (i = 0; i < 100; i++) {
  tmp = A[i];
  A[i] = B[i];
  B[i] = tmp;
}
```

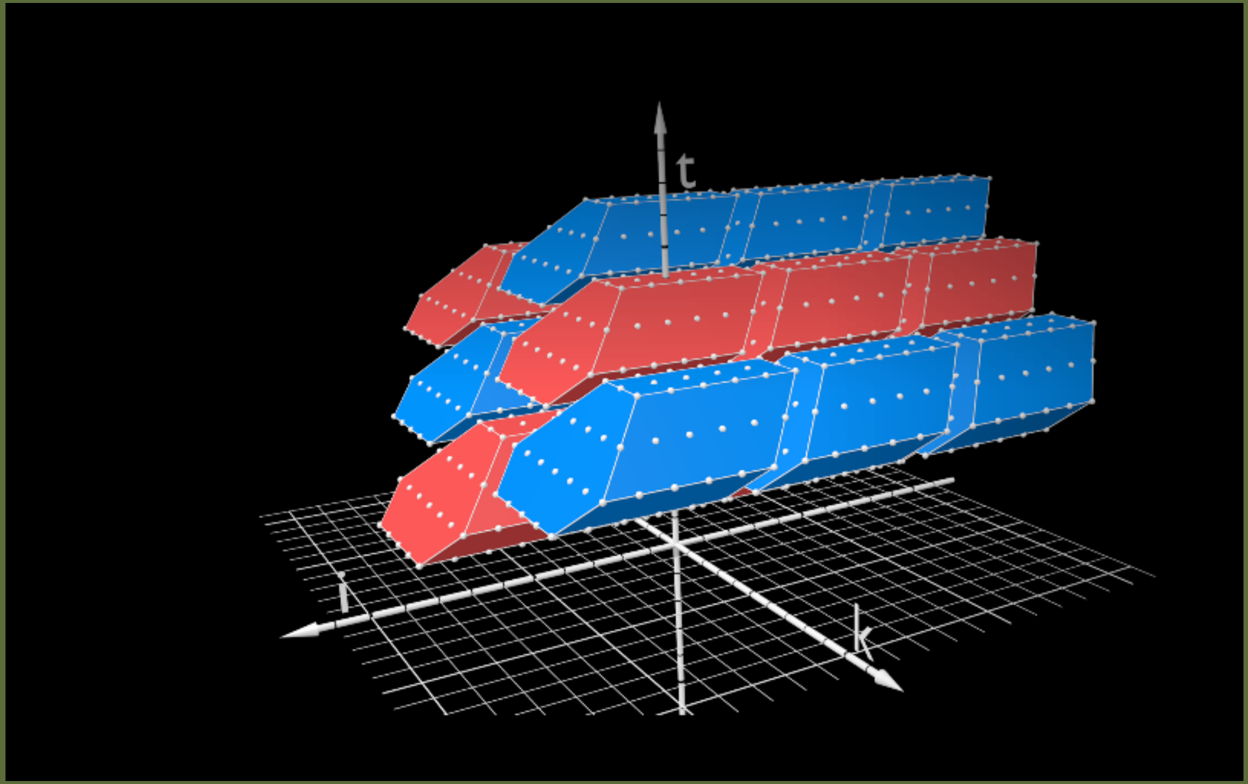
```
for (i = 0; i < 100; i++) {
  TMP[i] = A[i];
  A[i] = B[i];
  B[i] = TMP[i];
}
```

A loop carried write-after-read (anti) dependence prevents parallel execution.

Transform scalar **tmp** into an array **TMP** that contains for each loop iteration private storage.

Demo: Loop Modeling with Presburger Sets

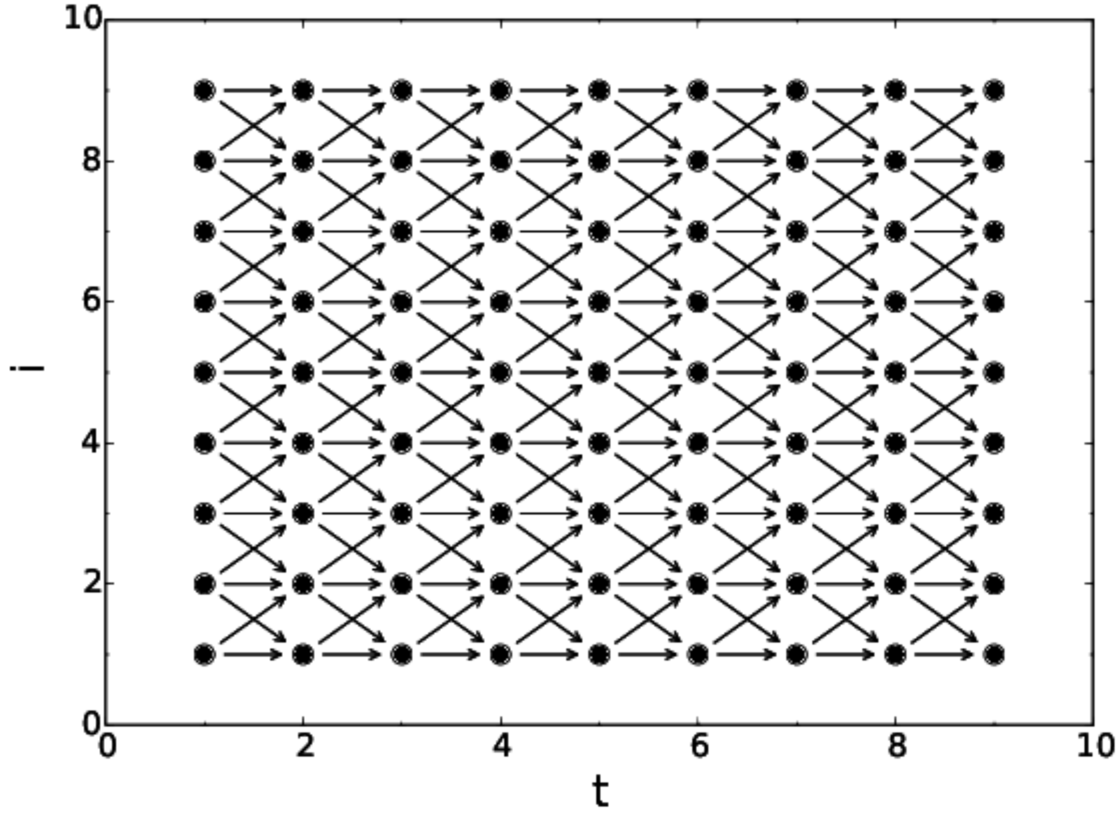
Tiling for Data-Locality and Parallelism



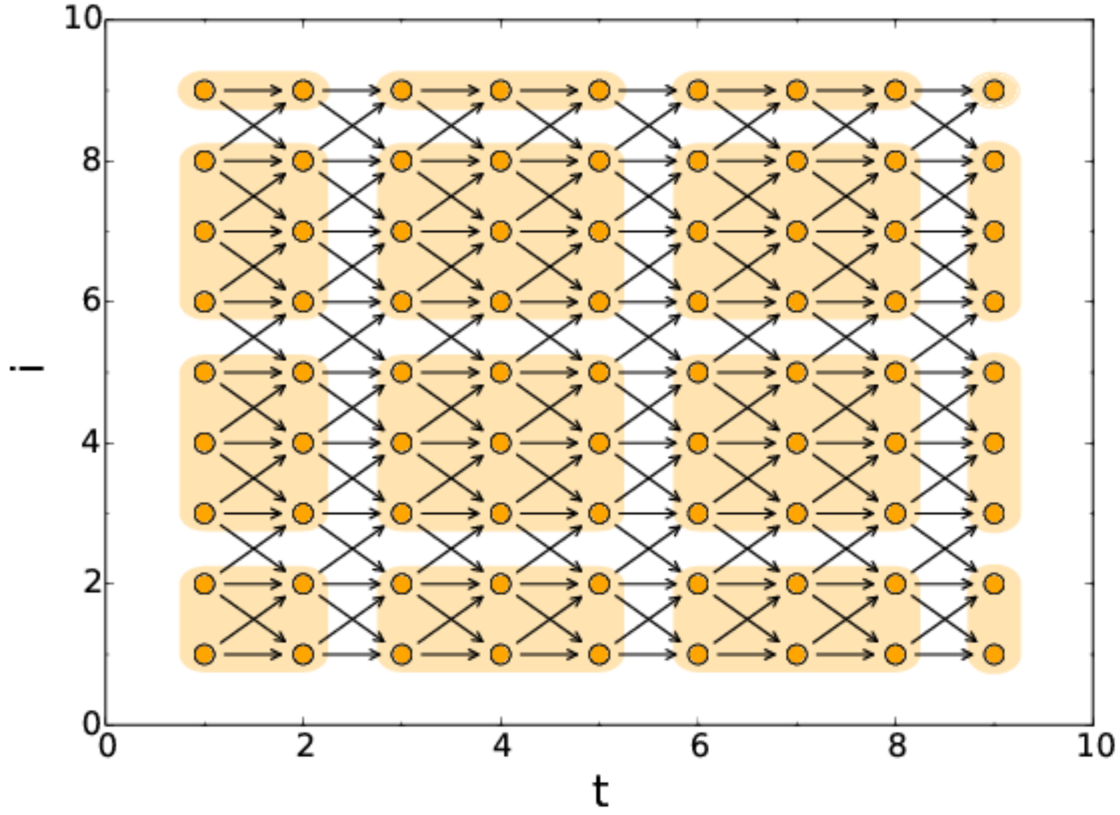
Tiling of a 1D Stencil

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

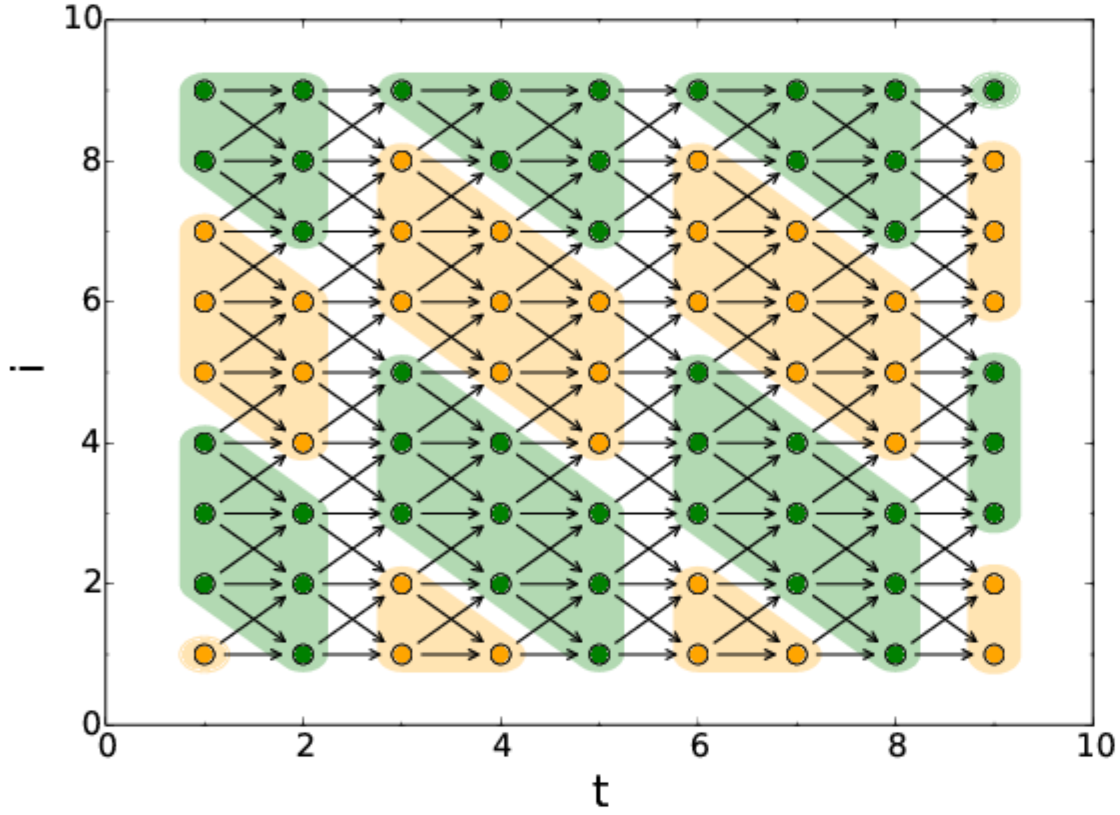
Jacobi Stencil with 1D Space + Time



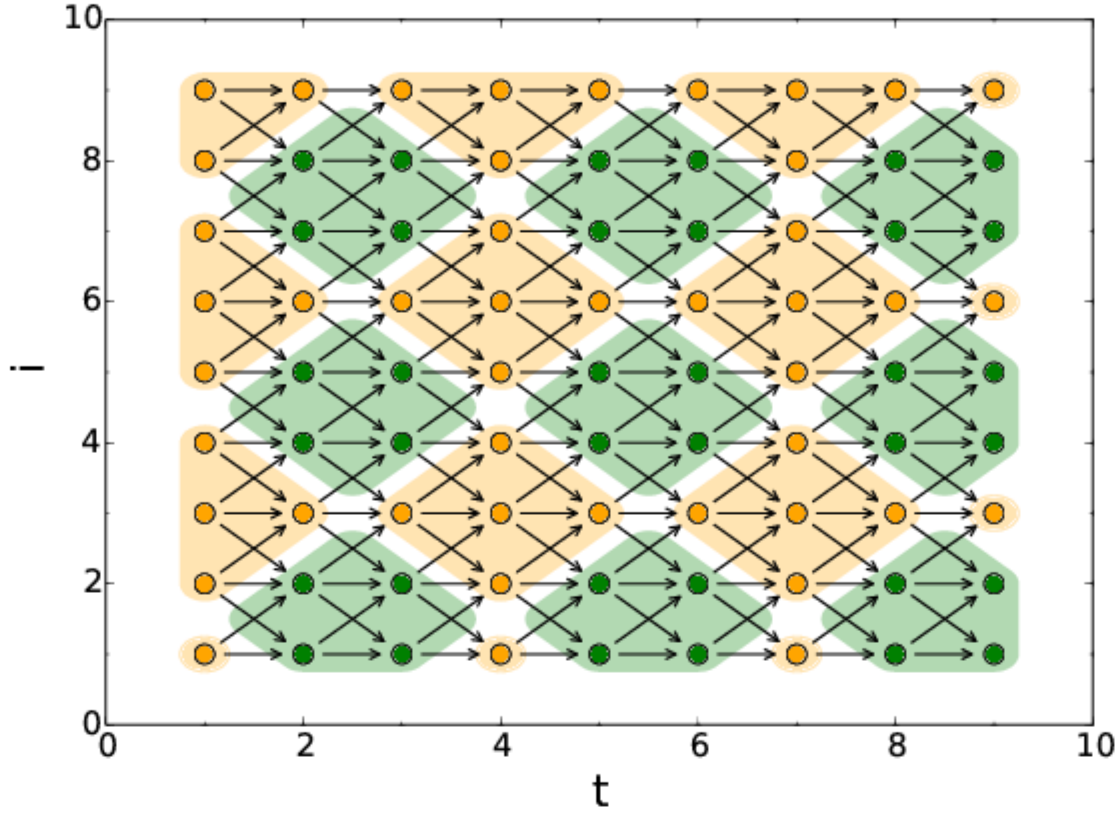
Jacobi Stencil with 1D Space + Time: Rectangular Tiles



Jacobi Stencil with 1D Space + Time: Skewed and Tiled



Jacobi Stencil with 1D Space + Time: Diamond Tiling

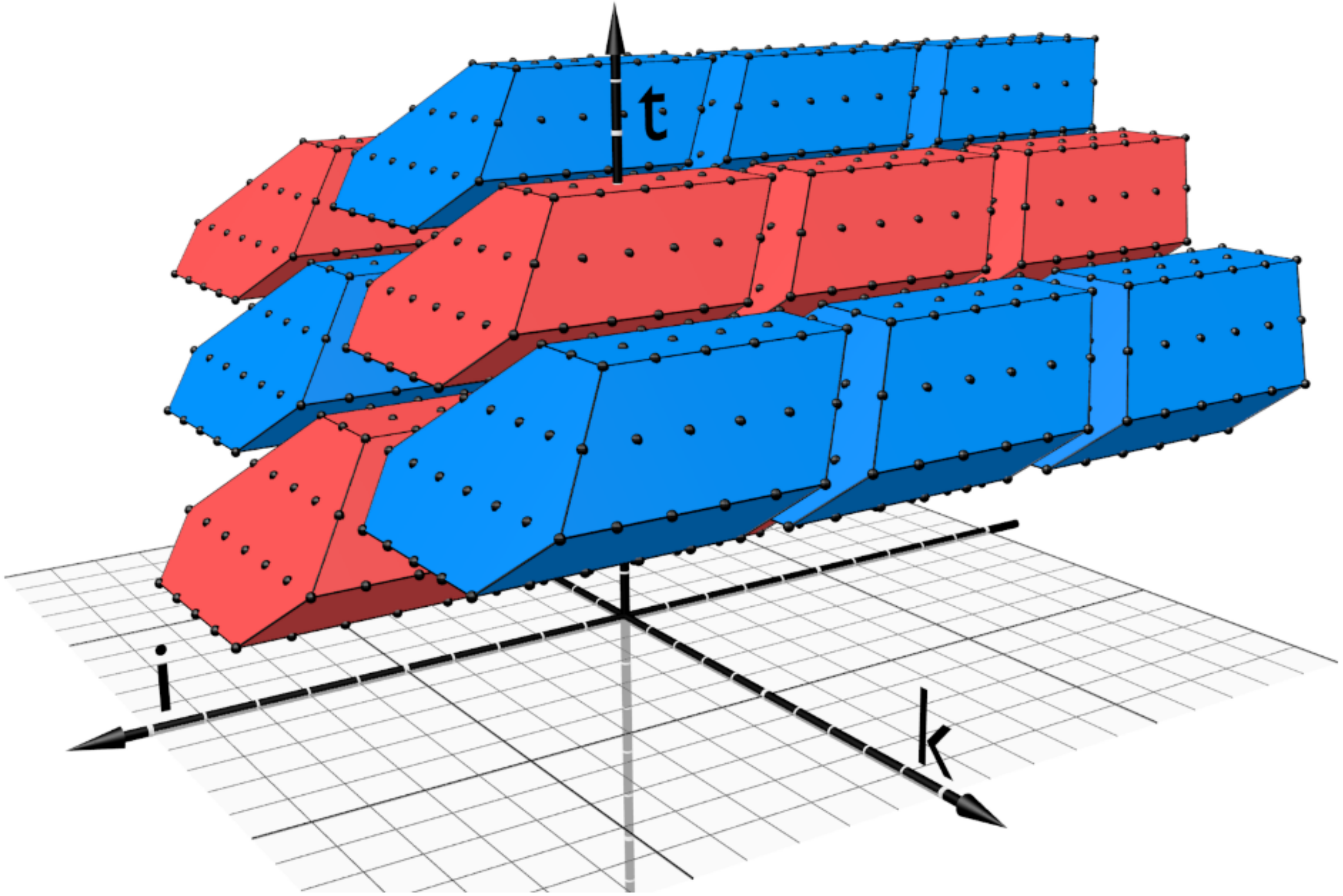


Polyhedral AST Generation

Advanced Tiling: A 2D Stencil

```
for (int t = 0; t < T; t++)
  for (int i = 0; i < N; i++)
    A[t+1][i][j] = A[t][i][j]
                    + A[t][i-1][j-1] + A[t][i-1][j+1]
                    + A[t][i+1][j-1] + A[t][i+1][j+1];
```

Hybrid Hexagonal/Parallelogram Tiling



Original copy code from hybrid-hexagonal tiling

```
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] = ...;
```

Unrolled copy code from hybrid-hexagonal tiling

```
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] = ...;
```

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

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

Project on dim. 1

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

Project on dim. 1, 2

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

```
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
```

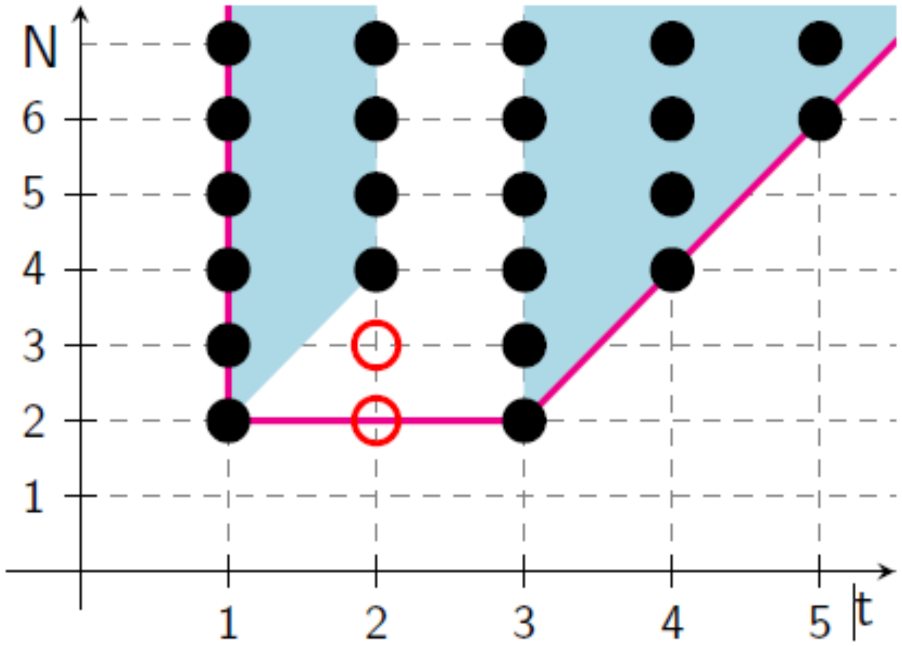
Fourier-Motzkin (Rational Quantifier Elimination)

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

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

Elimination of Existentially Quantified Variables

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

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

```c
#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`

Semantic Unrolling

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

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

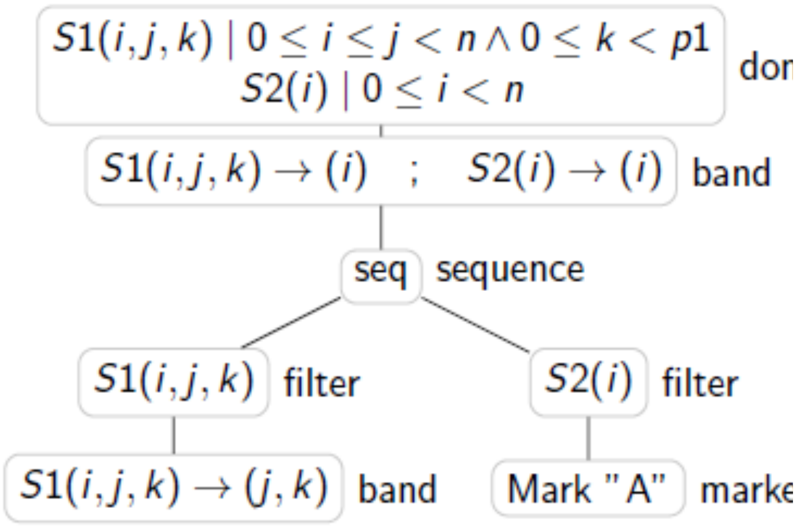
Schedule: $\{(i) \to (4\lfloor i/4 \rfloor, i)\}$, **Isolate:** $\{(t) \mid m \leq t \wedge t + 3 < n\}$

```
// 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);
```

Schedule Trees

Schedule Trees

```
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];
}
```



Schedule Tree – Original Code

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

Schedule Tree – Tiled

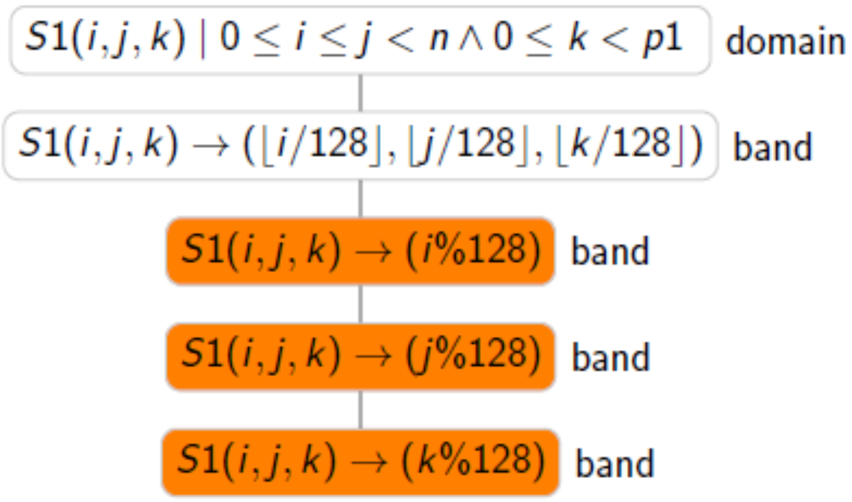
$S1(i,j,k) \mid 0 \leq i \leq j < n \wedge 0 \leq k < p1$ domain

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

Schedule Tree – Split 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)
```

Schedule Tree – 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
```

Schedule Tree – Isolate

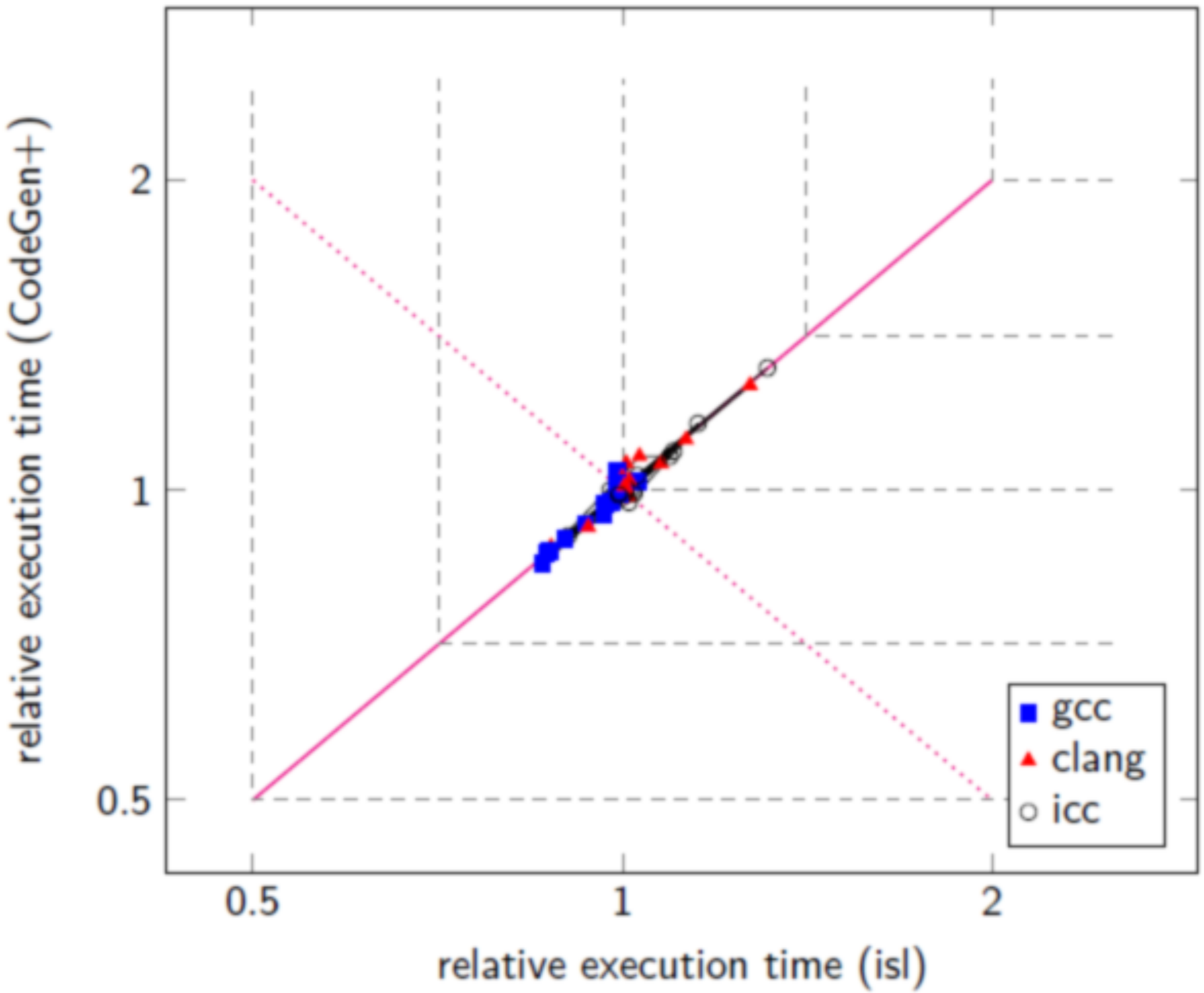
- $S1(i,j,k) \mid 0 \leq i \leq j < n \wedge 0 \leq k < p1$ — domain
- $S1(i,j,k) \rightarrow (\lfloor i/128 \rfloor, \lfloor j/128 \rfloor, \lfloor k/128 \rfloor)$ — band
- $S1(i,j,k) \rightarrow (i\%128)$ — band
- $S1(i,j,k) \rightarrow (\lfloor (j\%128)/8 \rfloor)$ $\{isolate[[a,b,c,d] \rightarrow [e]] : b < \lfloor n/128 \rfloor\}$ — 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 +=

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

 } else {
   // Handle remainder
```

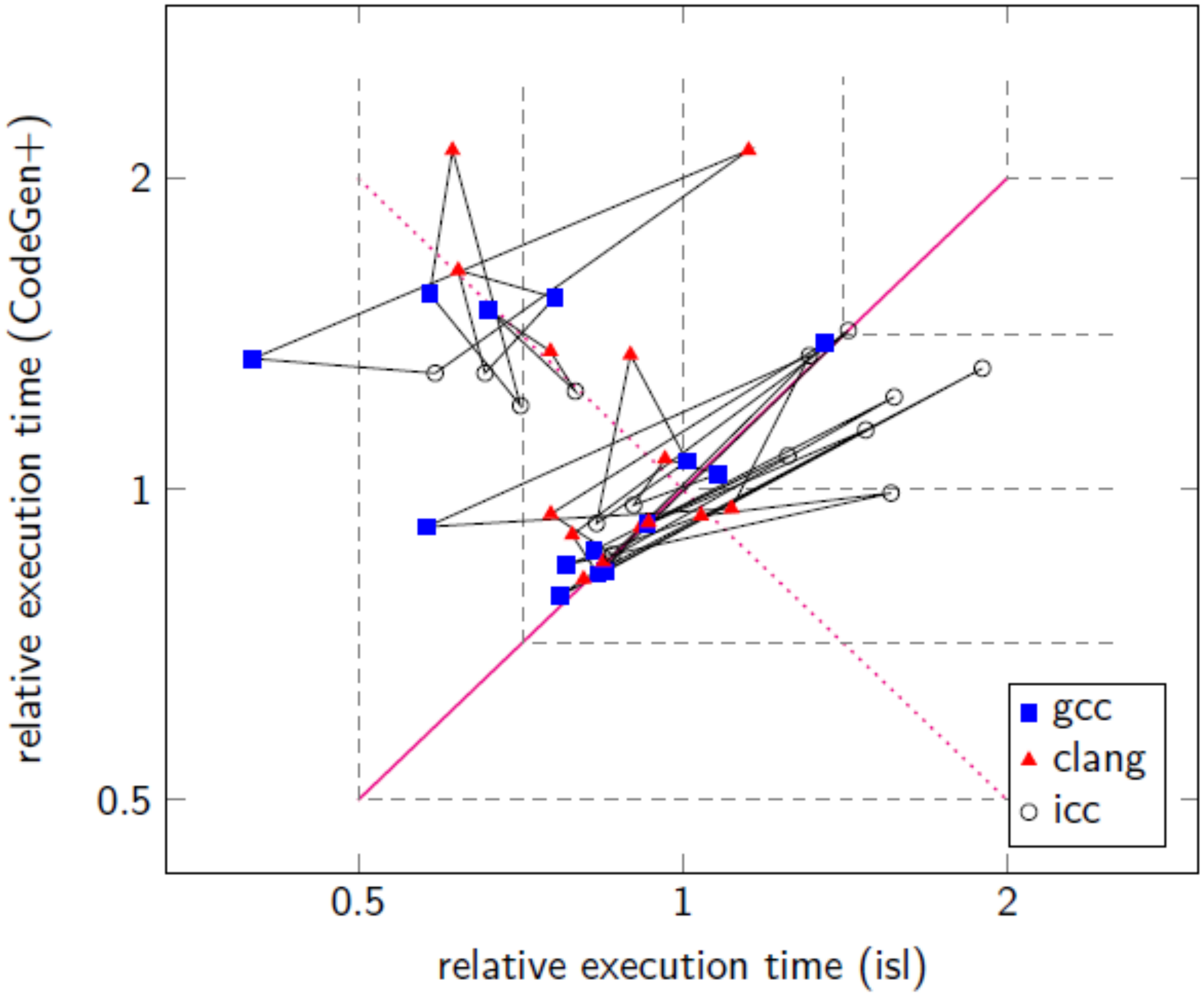
Evaluation

AST Generation

Generated Code Performance - Consistent



Generated Code Performance – Differing

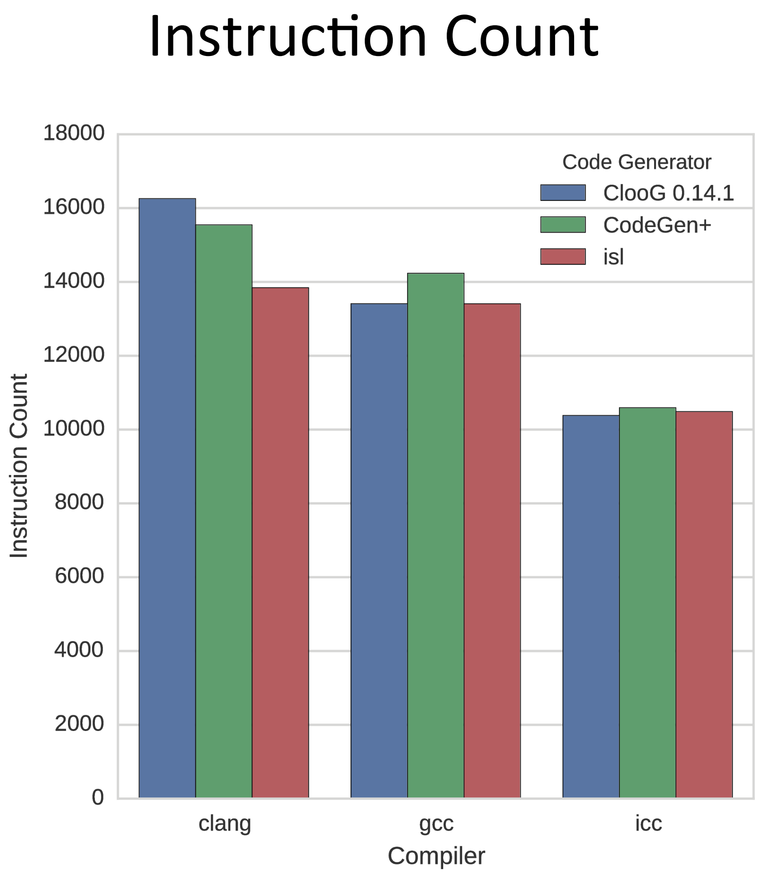


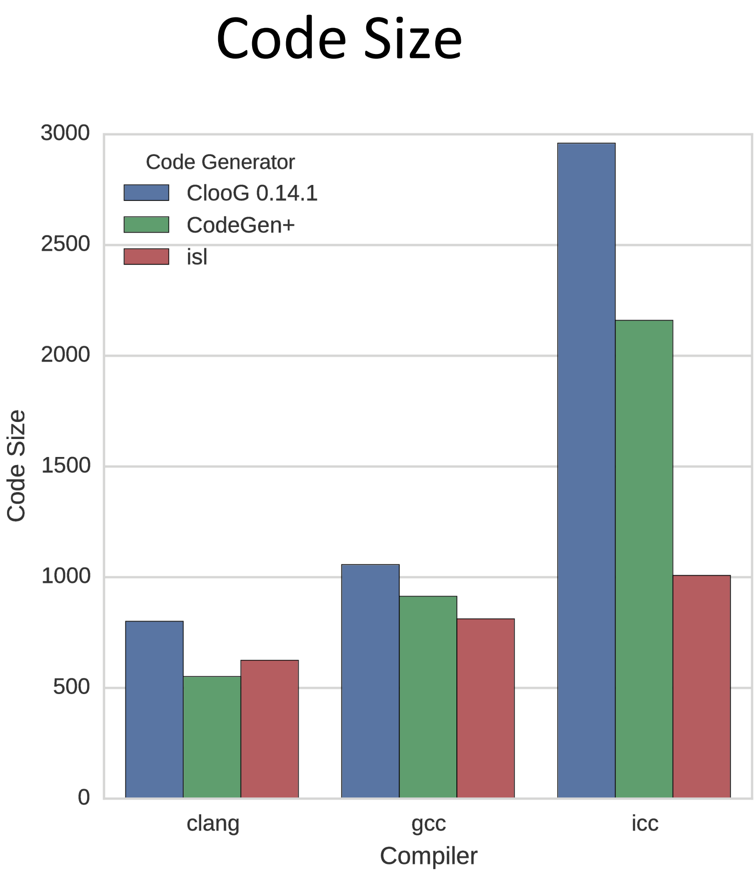
Code Quality: youcefn [Bastoul 2004]

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]





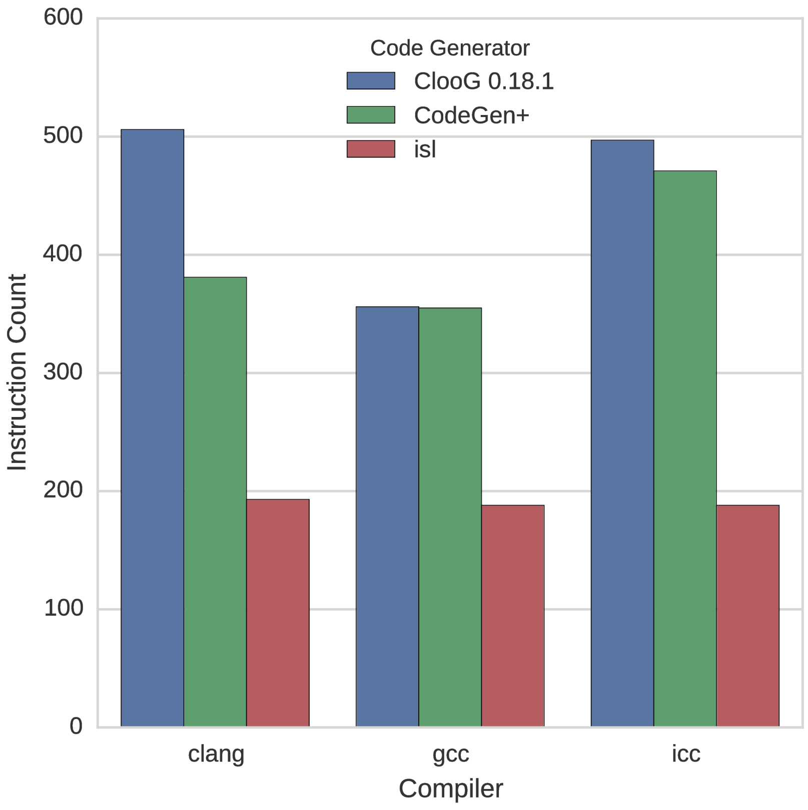
Code Quality: [Chen 2012] - Figure 8(b)

CLooG 0.18.1

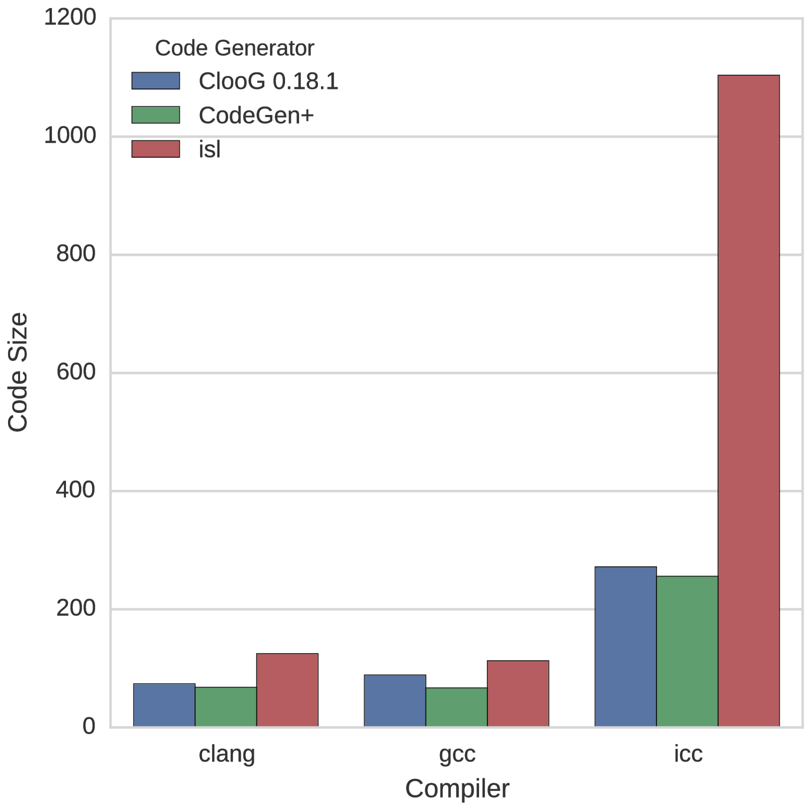
```
if (n >= 2)
  for (i = 2; i <= n; i += 2) {
    if (i%4 == 0)
      S0(i);
    if ((i+2)%4 == 0)
      S1(i);
  }
```

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

Instruction Count

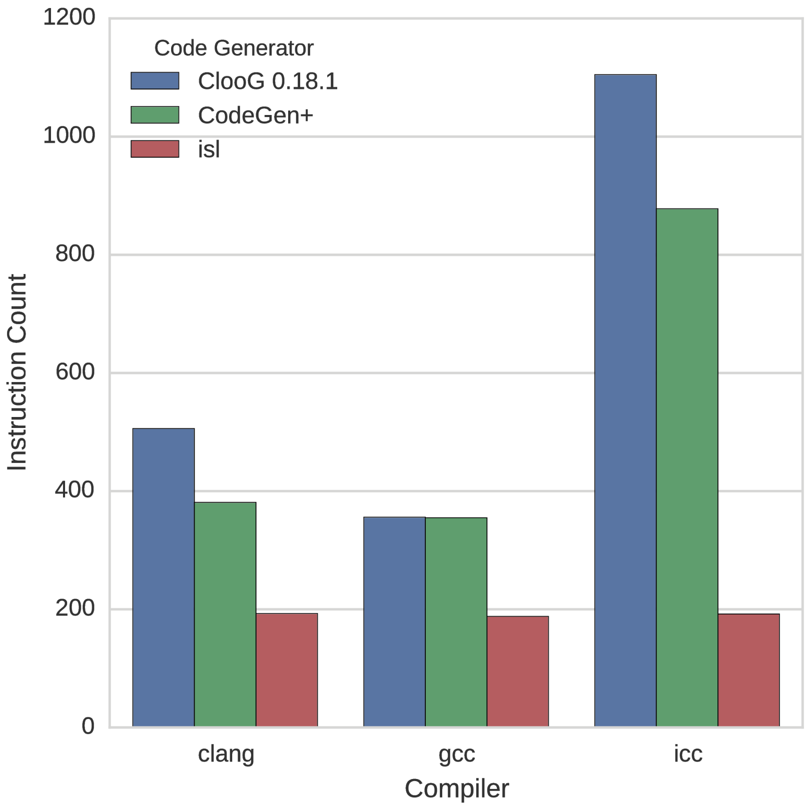


Code Size

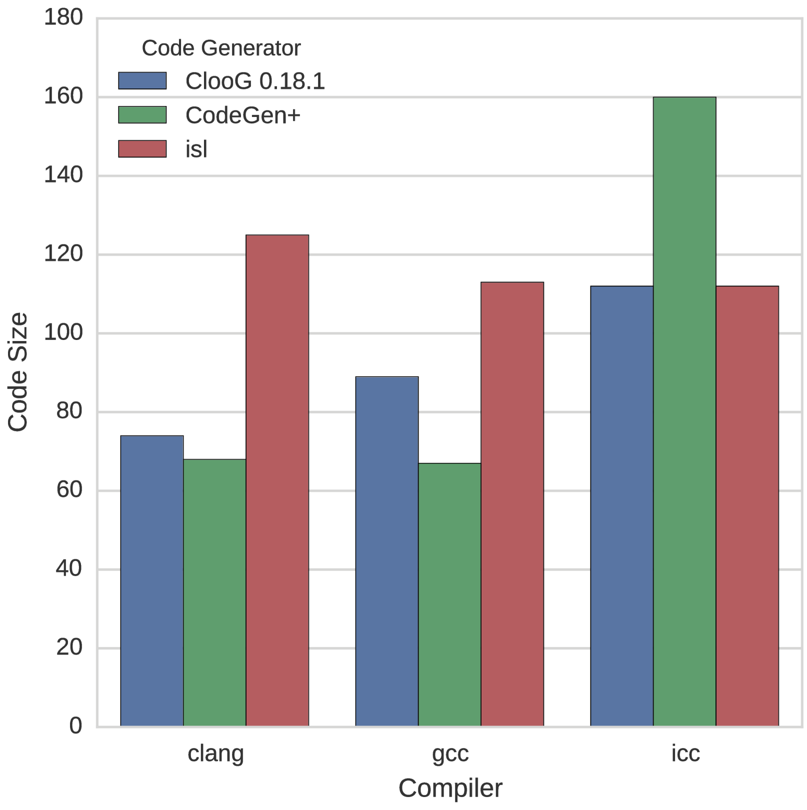


Code Quality: [Chen 2012] - Figure 8(b) novec/unroll

Instruction Count



Code Size

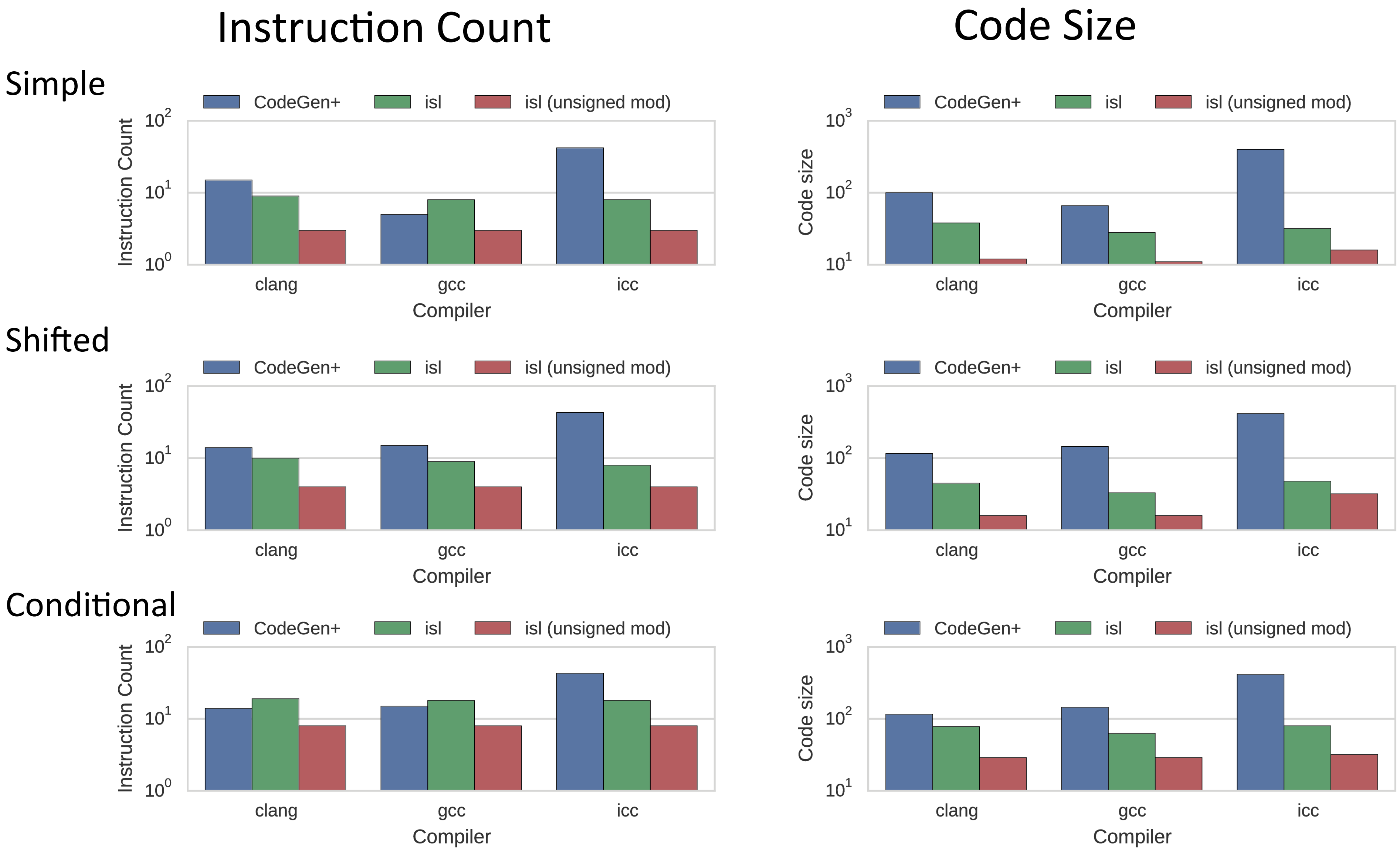


Modulo and Existentially Quantified Variables

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



Polyhedral Unrolling

Normal loop code

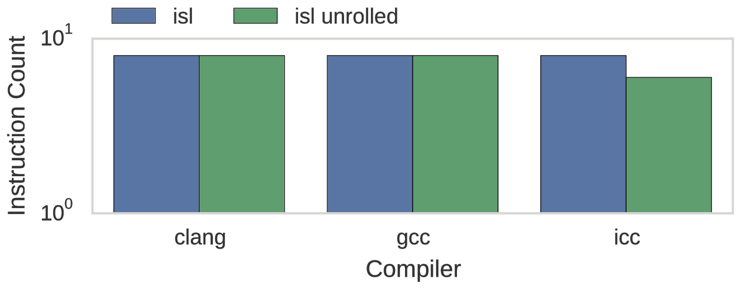
```
// Two e.q. variables
for (c0 = 0; c0 <= 7; c0 += 1)
  if (2 * (2 * c0 / 3) >= c0)
    S(c0);

// 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);
```

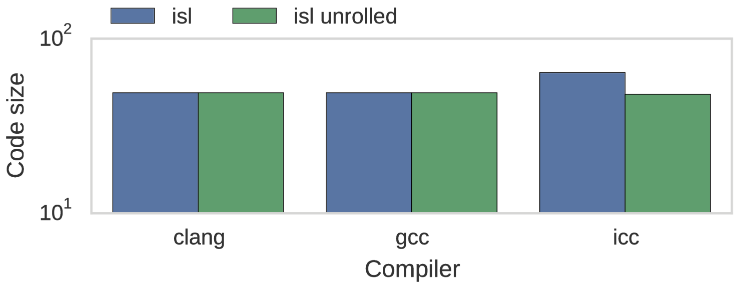
Polyhedral Unrolling

Instruction Count

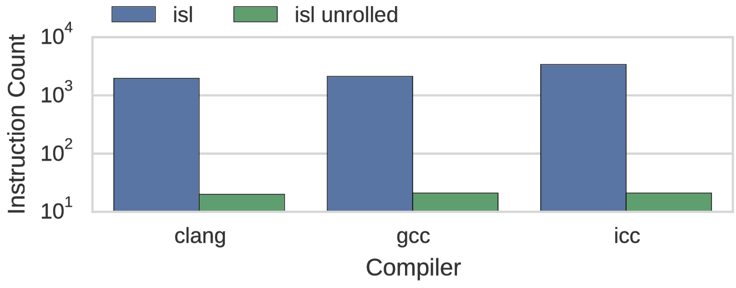
Code Size

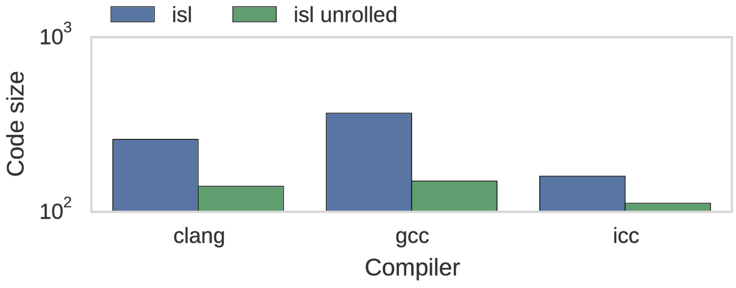
Two variables



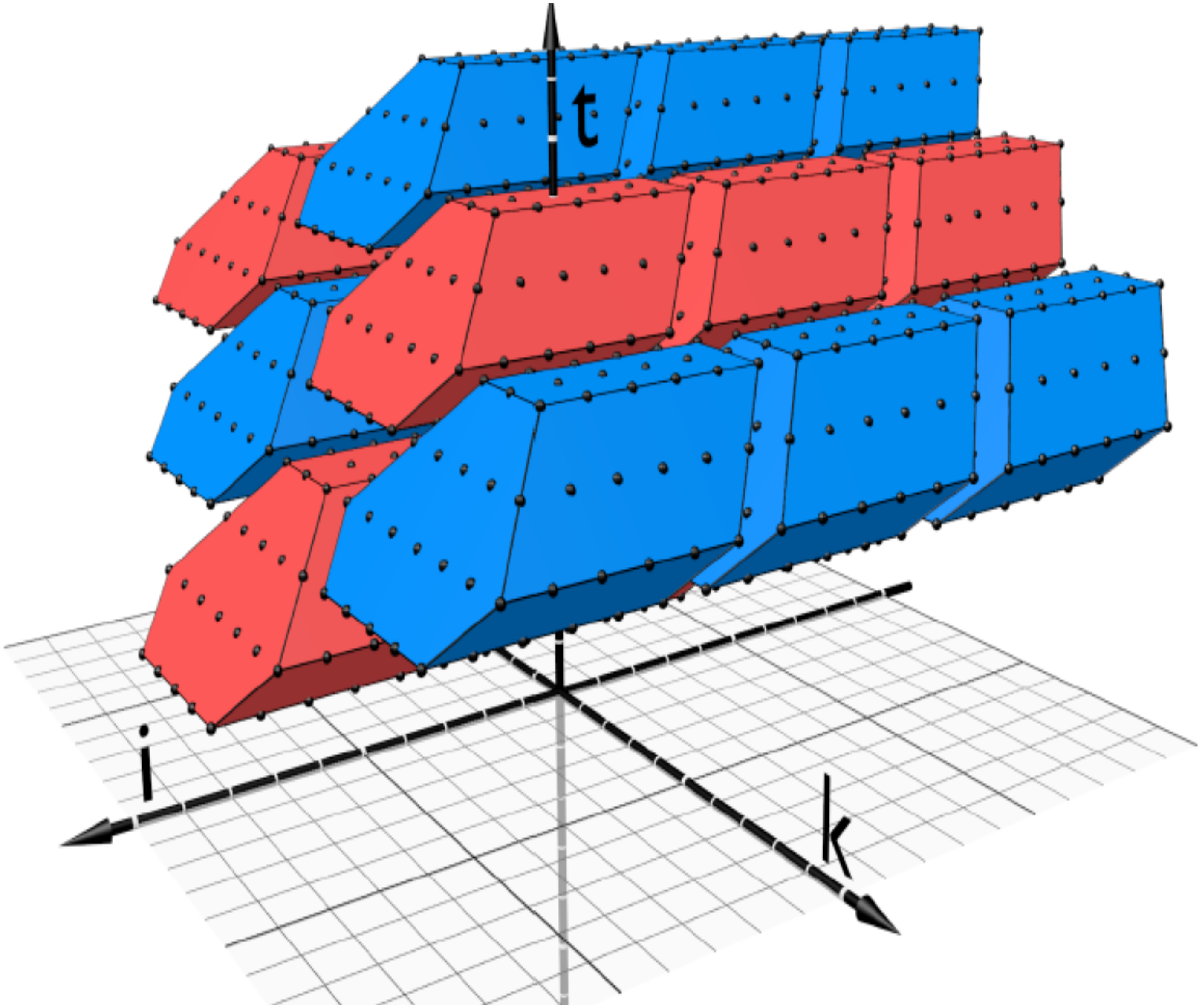


Multi Bound



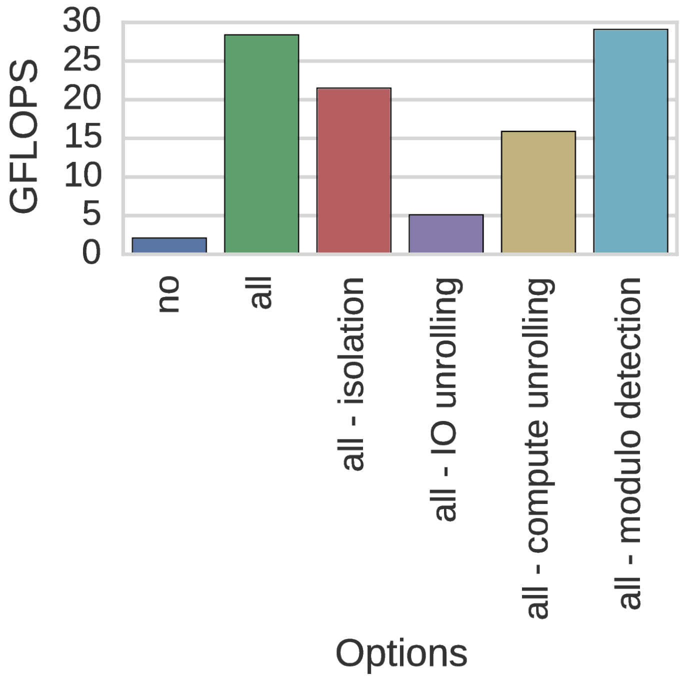


Hybrid Hexagonal Tiling for Stencil Programs

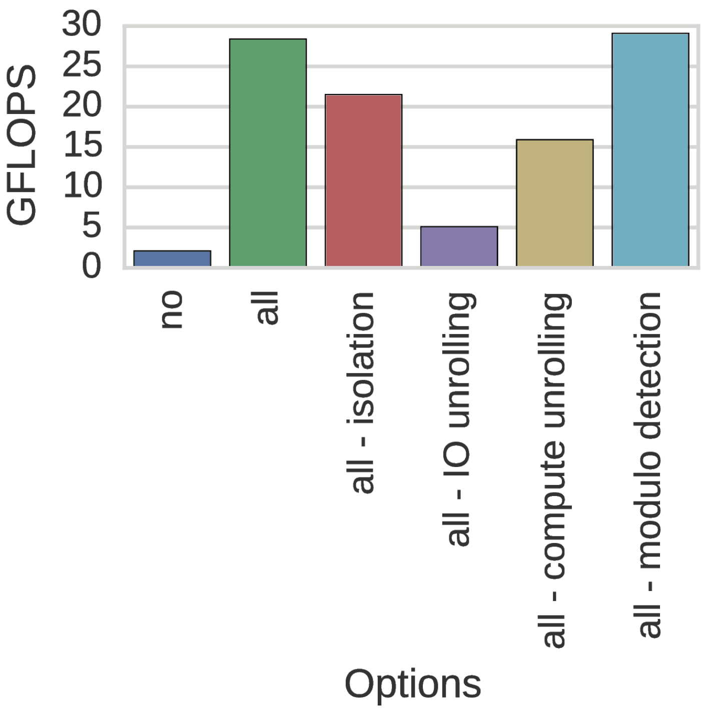


AST Generation Strategies for Hybrid-Hexagonal Tiling

Heat 2D



Heat 3D



Optimistic Loop Optimization

- $A_{S1} := \forall i,j : 1 \leq i < N \wedge 1 \leq j < M \implies 0 \leq j-1 < 20000$
- $A_{S2} := \forall i,j : 1 \leq i < N \wedge 1 \leq j < M \implies 0 \leq j < 20000$
- $A_{S3} := \forall i,j : 1 \leq i < N \wedge 1 \leq j < M \implies 0 \leq j+1 < 20000$

⇓ combine and simplify ⇓

- $A := A_{S1} \wedge A_{S2} \wedge A_{S3}$
- $A = \forall i,j : 1 \leq i < N \wedge 1 \leq j < M \implies 1 \leq j < 19999$

⇓ invert, eliminate i and j, invert ⇓

- $P := \neg \text{proj}_{N,M}(\neg A)$
- $P = M \leq 19999 \vee (N \leq 1 \wedge M >= 20000)$

⇓ assume loop nest is executed at least once ⇓

- $P' := P \text{ gist } (M > 1 \wedge N > 1)$
- $P' = M \leq 19999$

1

Clearly beneficial loop interchange

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

$\Rightarrow$ 15s

Assumption: Fixed size arrays do not overflow

```
void arrayOverflow(int N, float A[][20000]) {
  for (int i = 1; i < N; i++)
    for (int j = 1; j < M; j++) {
S1:   A[i][j-1] = ...;
S2:   A[i][j ] = ...;
S3:   A[i][j+1] = ...;
    }
}
```

Simplify Assumptions

- $A_{S1} := \forall i,j : 1 \leq i < N \wedge 1 \leq j < M \implies 0 \leq j - 1 < 20000$
- $A_{S2} := \forall i,j : 1 \leq i < N \wedge 1 \leq j < M \implies 0 \leq j < 20000$
- $A_{S3} := \forall i,j : 1 \leq i < N \wedge 1 \leq j < M \implies 0 \leq j + 1 < 20000$

Run-time check generation

- Set of constraints → AST expression
- Arbitrary Presburger Formula
- Implemented in a polyhedral code generator (as part of isl)

```
void arrayOverflow(int N, float A[][20000]) {
  if (M <= 19999) {
    // optimized code
  } else {
    // original code
  }
}
```

Optimistic Delinearization

```
void copyOddEven(int N, float *Ptr) {

#define A(x, y) Ptr[(x) * N + (y)]
  for (int i = 0; i < N; i++)
    for (int j = 0; j < N; j++)
      A(2 * j, i) = A(2 * j + 1, i);
}
```

Tobias Grosser, Sebastian Pop, Louis-Noël Pouchet, P. Sadayappan, and Sebastian Pop
Optimistic delinearization of parametrically sized arrays, ICS 2015

Optimistic Exceptions Elimination

```
void copy(float A[][100], float B[][100],
  int DebugLevel, int N) {
  for (int i = 0; i < N; i++)
    for (int j = 0; j < 100; j++)
S1:   A[j][i] = B[j][i];
    if (DebugLevel > 5)
S2:   printf("Column %d copied\n", i);
  }
}
```

Optimistic Loop Invariant Code Motion

```
void copy(struct Array A) {

int tmp0, tmp1;
tmp0 = size0(A);

if (tmp0 > 0)
  tmp1 = size1(A);

for (int i = 0; i < tmp0; i++)
  for (int j = 0; j < tmp1; j++)
    S1:      access(A,j,i) += ..;
}
```

Integer Overflow

```
void overflow(unsigned n, unsigned m, float A[]) {
    for (unsigned i = 0; i < n; i++) {
      A[i] = 0;
    }
    for (unsigned i = 0; i < n + m; i++) {
      A[i] = 1;
    }


}
```

CGO'17: Optimistic Loop Optimization
(with Johannes Doerfert und Sebastian Hack)