Tutorial: Extending Loop Transformation Frameworks to Irregular Applications
(aka Math for Irregular Codes)

Michelle Mills Strout (University of Arizona),
Louis-Noel Pouchet (Colorado State University),
Milind Kulkarni (Purdue University)

Sunday February 23
PPoPP 2020

THE UNIVERSITY OF ARIZONA

Transformation Frameworks

Motivation

- Intermediate representations for computations
- Transformation specifications and code generation after transformation
- Composition of transformations
- Data dependence representation and thus legality checks for composed transformations

Covering in Today's Tutorial

- Polyhedral Model for representing computations with affine loop bounds and array accesses
- Polyhedral compilation for sparse-immutable computations
- Sparse Polyhedral Framework (SPF) for sparse matrix/tensor computations with indirect array accesses
- PolyRec Framework for recursive irregular computations

Polyhedral model: Loop transformations improve performance!

Example (dgemm)

```
   /* C := alpha*A*B + beta*C */
   for (i = 0; i < ni; i++)
    for (j = 0; j < nj; j++)
S1:      C[i][j] *= beta;
   for (i = 0; i < ni; i++)
    for (j = 0; j < nj; j++)
      for (k = 0; k < nk; ++k)
S2:        C[i][j] += alpha * A[i][k] * B[k][j];
```

- Loop transformation: *permute(i,k,S2)*

Execution time (in s) on this laptop, GCC 4.2, ni=nj=nk=512

version	-O0	-O1	-O2	-O3 -vec
original	1.81	0.78	0.78	0.78
permute	1.52	0.35	0.35	0.20

http://gcc.gnu.org/onlinedocs/gcc-4.2.1/gcc/Optimize-Options.html

Another Example: FDTD

Example (fdtd-2d)

```
for(t = 0; t < tmax; t++) {
  for (j = 0; j < ny; j++)
    ey[0][j] = _edge_[t];
  for (i = 1; i < nx; i++)
    for (j = 0; j < ny; j++)
      ey[i][j] = ey[i][j] - 0.5*(hz[i][j]-hz[i-1][j]);
  for (i = 0; i < nx; i++)
    for (j = 1; j < ny; j++)
      ex[i][j] = ex[i][j] - 0.5*(hz[i][j]-hz[i][j-1]);
  for (i = 0; i < nx - 1; i++)
    for (j = 0; j < ny - 1; j++)
      hz[i][j] = hz[i][j] - 0.7* (ex[i][j+1] - ex[i][j] +
                              ey[i+1][j]-ey[i][j]);
}
```

- Loop transformation: *polyhedralOpt(fdtd-2d)*

Execution time (in s) on this laptop, GCC 4.2, 64x1024x1024

version	-O0	-O1	-O2	-O3 -vec
original	2.59	1.62	1.54	1.54
polyhedralOpt	2.05	0.41	0.41	0.41

Doing such transformations by hand is NOT FEASIBLE!

Example (fdtd-2d tiled)

```
for (c0 = 0; c0 <= (((ny + 2 * tmax + -3) * 32 < 0?((32 < 0?-((-(ny + 2 * tmax + -3) + 32 + 1) / 32) : -((-(ny + 2 *
tmax + -3) + 32 - 1) / 32))) : (ny + 2 * tmax + -3) / 32)); ++c0) {
  #pragma omp parallel for private(c3, c4, c2, c5)
   for (c1 = (((c0 * 2 < 0?-(-c0 / 2) : ((2 < 0?(-c0 + -2 - 1) / -2 : (c0 + 2 - 1) / 2)))) > (((32 * c0 + -tmax + 1) *
32 < 0?-(-(32 * c0 + -tmax + 1) / 32) : ((32 < 0?(-(32 * c0 + -tmax + 1) + -32 - 1) / -32 : (32 * c0 + -tmax + 1 + 32
- 1) / 32))))?((c0 * 2 < 0?-(-c0 / 2) : ((2 < 0?(-c0 + -2 - 1) / -2 : (c0 + 2 - 1) / 2)))) : (((32 * c0 + -tmax + 1) *
32 < 0?-(-(32 * c0 + -tmax + 1) / 32) : ((32 < 0?(-(32 * c0 + -tmax + 1) + -32 - 1) / -32 : (32 * c0 + -tmax + 1 + 32
- 1) / 32))))); c1 <= ((((((ny + tmax + -2) * 32 < 0?((32 < 0?-((-(ny + tmax + -2) + 32 + 1) / 32) : -((-(ny + tmax +
-2) + 32 - 1) / 32))) : (ny + tmax + -2) / 32)) < (((32 * c0 + ny + 30) * 64 < 0?((64 < 0?-((-(32 * c0 + ny + 30) + 64
+ 1) / 64) : -((-(32 * c0 + ny + 30) + 64 - 1) / 64))) : (32 * c0 + ny + 30) / 64))?(((ny + tmax + -2) * 32 < 0?((32 <
0?-((-(ny + tmax + -2) + 32 + 1) / 32) : -((-(ny + tmax + -2) + 32 - 1) / 32))) : (ny + tmax + -2) / 32)) : (((32 * c0 +
ny + 30) * 64 < 0?((64 < 0?-((-(32 * c0 + ny + 30) + 64 + 1) / 64) : -((-(32 * c0 + ny + 30) + 64 - 1) / 64))) : (32 *
c0 + ny + 30) / 64)))) < c0?(((((ny + tmax + -2) * 32 < 0?((32 < 0?-((-(ny + tmax + -2) + 32 + 1) / 32) : -((-(ny + tmax
+ -2) + 32 - 1) / 32))) : (ny + tmax + -2) / 32)) < (((32 * c0 + ny + 30) * 64 < 0?((64 < 0?-((-(32 * c0 + ny + 30) + 64
+ 1) / 64) : -((-(32 * c0 + ny + 30) + 64 - 1) / 64))) : (32 * c0 + ny + 30) / 64))?(((ny + tmax + -2) * 32 < 0?((32 <
0?-((-(ny + tmax + -2) + 32 + 1) / 32) : -((-(ny + tmax + -2) + 32 - 1) / 32))) : (ny + tmax + -2) / 32)) : (((32 * c0 +
ny + 30) * 64 < 0?((64 < 0?-((-(32 * c0 + ny + 30) + 64 + 1) / 64) : -((-(32 * c0 + ny + 30) + 64 - 1) / 64))) : (32 *
c0 + ny + 30) / 64)))) : c0)); ++c1) {
    for (c2 = c0 + -c1; c2 <= (((((tmax + nx + -2) * 32 < 0?((32 < 0?-((-(tmax + nx + -2) + 32 + 1) / 32) : -((-(tmax +
nx + -2) + 32 - 1) / 32))) : (tmax + nx + -2) / 32)) < (((32 * c0 + -32 * c1 + nx + 30) * 32 < 0?((32 < 0?-((-(32 * c0 +
-32 * c1 + nx + 30) + 32 + 1) / 32) : -((-(32 * c0 + -32 * c1 + nx + 30) + 32 - 1) / 32))) : (32 * c0 + -32 * c1 + nx +
30) / 32))?(((tmax + nx + -2) * 32 < 0?((32 < 0?-((-(tmax + nx + -2) + 32 + 1) / 32) : -((-(tmax + nx + -2) + 32 - 1) /
32))) : (tmax + nx + -2) / 32)) : (((32 * c0 + -32 * c1 + nx + 30) * 32 < 0?((32 < 0?-((-(32 * c0 + -32 * c1 + nx + 30)
+ 32 + 1) / 32) : -((-(32 * c0 + -32 * c1 + nx + 30) + 32 - 1) / 32))) : (32 * c0 + -32 * c1 + nx + 30) / 32)))); ++c2)
{
      if (c0 == 2 * c1 && c0 == 2 * c2) {
        for (c3 = 16 * c0; c3 <= ((tmax + -1 < 16 * c0 + 30?tmax + -1 : 16 * c0 + 30)); ++c3)
          if (c0 % 2 == 0)
            (ey[0])[0] = (_edge_[c3]);
........... (200 more lines!)
```

Performance gain: 2-6× on modern multicore platforms

RoadMap for Tutorial: Math for Irregular Codes

Concepts

- Polyhedral model review
- Sparse computations as union of dense computations
- Sparse Polyhedral Framework (SPF)
- Polyrec

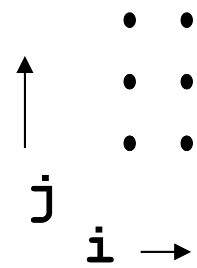
Hands On Tutorial Goals

- Specify affine transformations in ISCC
- Handle irregular loop bounds in ISCC
- Demo of data dependence analysis for SPF
- Demo polyrec

Representing Loops with Math (Matrices)

Original code

```
  do i = 1,2
        do j = 1,3
             S1: A(i,j) = A(i-1,j+1)+1
        enddo
enddo
```



Represent the iteration space

–As an intersection of inequalities

–The iteration space is the integer tuples within the intersection

Bounds:

$$1 \leq i \leq 2$$

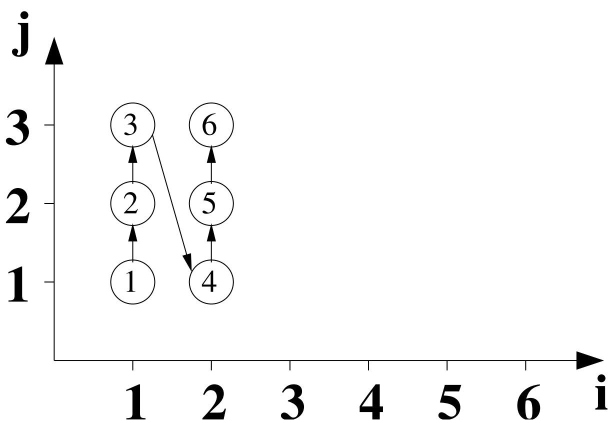
$$1 \leq j \leq 3$$

$$\begin{bmatrix} 1 & 0 \\ -1 & 0 \\ 0 & 1 \\ 0 & -1 \end{bmatrix} \begin{bmatrix} i \\ j \end{bmatrix} \geq \begin{bmatrix} 1 \\ -2 \\ 1 \\ -3 \end{bmatrix}$$

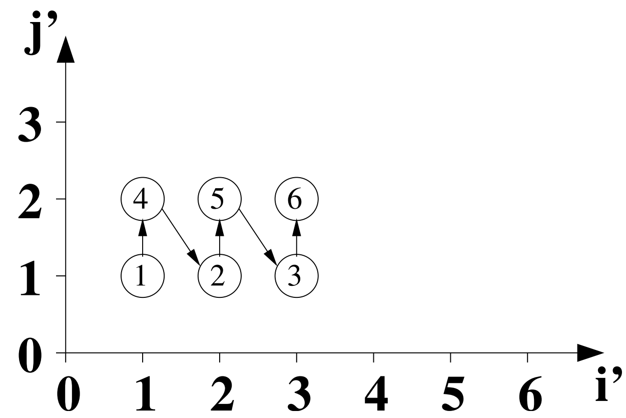
Affine Transformations

Interchange Transformation

The transformation matrix is the identity with a permutation of two rows.



$\Longrightarrow$



$$\begin{bmatrix} 1 & 0 \\ -1 & 0 \\ 0 & 1 \\ 0 & -1 \end{bmatrix} \begin{pmatrix} i \\ j \end{pmatrix} + \begin{pmatrix} -1 \\ 2 \\ -1 \\ 3 \end{pmatrix} \geq \vec{0}$$

(a) original polyhedron

$A\vec{x}+\vec{a} \geq \vec{0}$

$$\begin{pmatrix} i' \\ j' \end{pmatrix} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{pmatrix} i \\ j \end{pmatrix}$$

(b) transformation function

$\vec{y} = T\vec{x}$

$$\begin{bmatrix} 0 & 1 \\ 0 & -1 \\ 1 & 0 \\ -1 & 0 \end{bmatrix} \begin{pmatrix} i' \\ j' \end{pmatrix} + \begin{pmatrix} -1 \\ 2 \\ -1 \\ 3 \end{pmatrix} \geq \vec{0}$$

(c) target polyhedron

$(AT^{-1})\vec{y}+\vec{a} \geq \vec{0}$

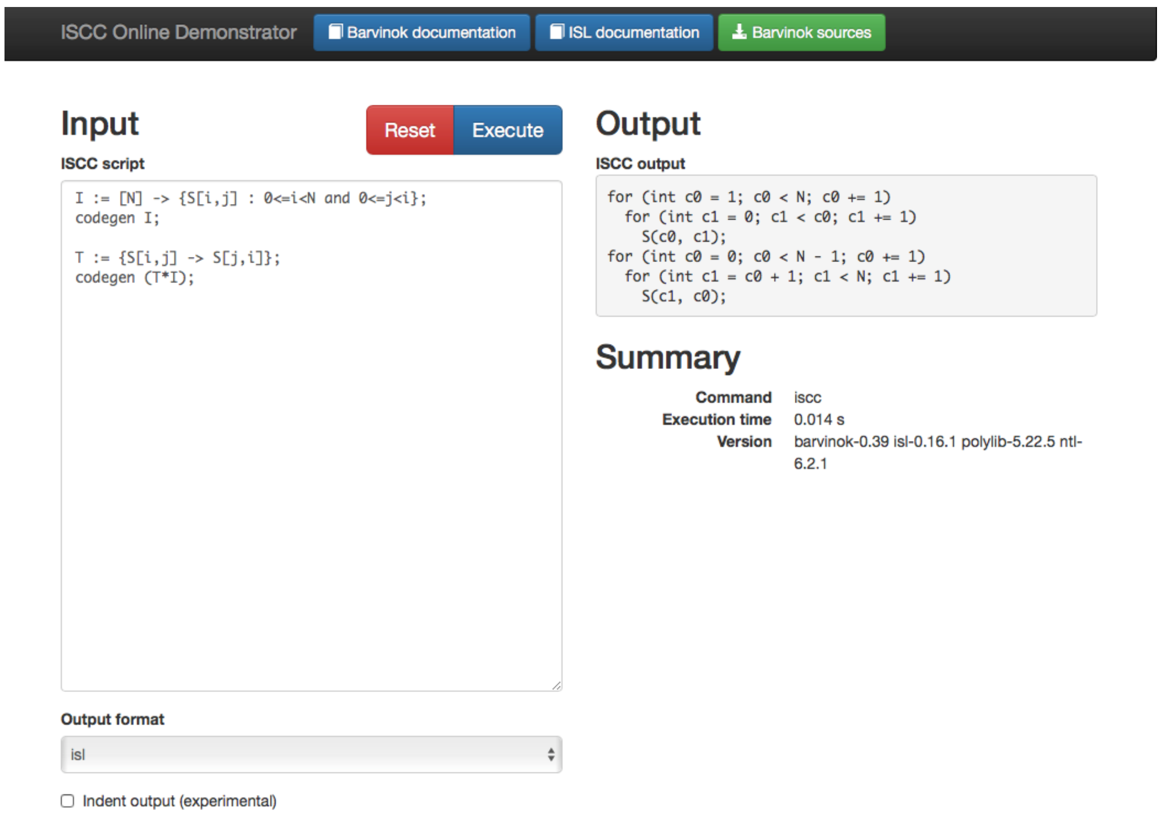
```
do i = 1, 2
  do j = 1, 3
     S(i,j)
```

```
do i' = 1, 3
  do j' = 1, 2
     S(i=j',j=i')
```

Goal: Learn How to Use ISCC

ISCC

- Calculator for ISL (Integer Set Library)
- http://compsys-tools.ens-lyon.fr/iscc/
- Author: Sven Verdoolaege
- See Barvinok documentation online for a user manual



Specifying Example Loop in ISCC

Original Loop in C

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

Create a macro for statement in C

```
#define S(i,j) A[i][j] = exp((i)+(j))
for (i=0; i<N; i++)
  for (j=0; j<i; j++)
    S(i,j);
```

Iterations in loops described as a Set in ISCC

```
 I := [N] -> {S[i,j] : 0<=i<N and 0<=j<i};
```

ISCC: Loop Interchange Transformation

Generate the loop bounds for the Set I

```
// Input
I := [N] -> {S[i,j] : 0<=i<N and 0<=j<i};
codegen I;
// Output
for (int c0 = 1; c0 < N; c0 += 1)
  for (int c1 = 0; c1 < c0; c1 += 1)
    S(c0, c1);
```

Generate after applying *Loop Interchange*

```
// Input: Transformation function
T := {S[i,j] -> [j,i]};
codegen (T*I);
// Output
for (int c0 = 0; c0 < N - 1; c0 += 1)
  for (int c1 = c0 + 1; c1 < N; c1 += 1)
    S(c1, c0);
```

ISCC Determines Old Iterators as Function of New Iterators

Original Loop Nests and Transformation

```
// Input
Domain := [N] -> {S1[0,j,0] : 1<=j<N;
                  S2[1,k,0] : 0<=k<N-1;};

T_fusion := {S1[0,j,0]->[0,j,0];
             S2[1,k,0]->[0,k+1,1] };
codegen (T_fusion*Domain);
```

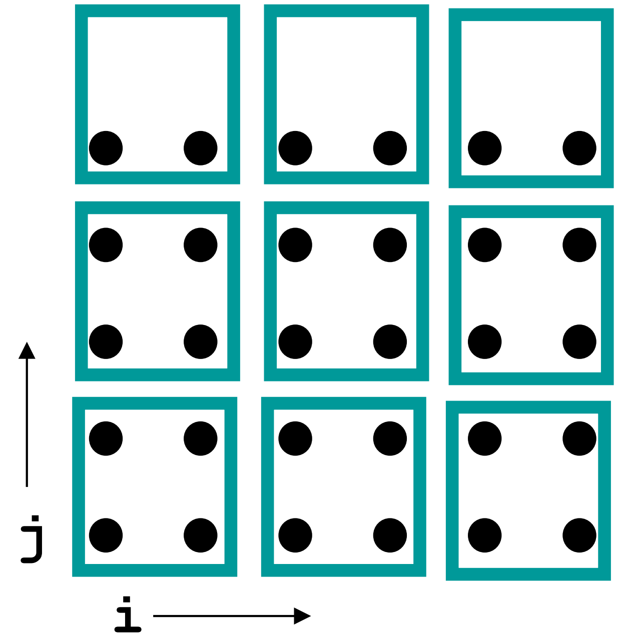
Resulting code with old iterators as function of new iterators

```
// Output
for (int c1 = 1; c1 < N; c1 += 1) {
  S1(0, c1, 0);
  S2(1, c1 - 1, 0);
}
```

Tiling

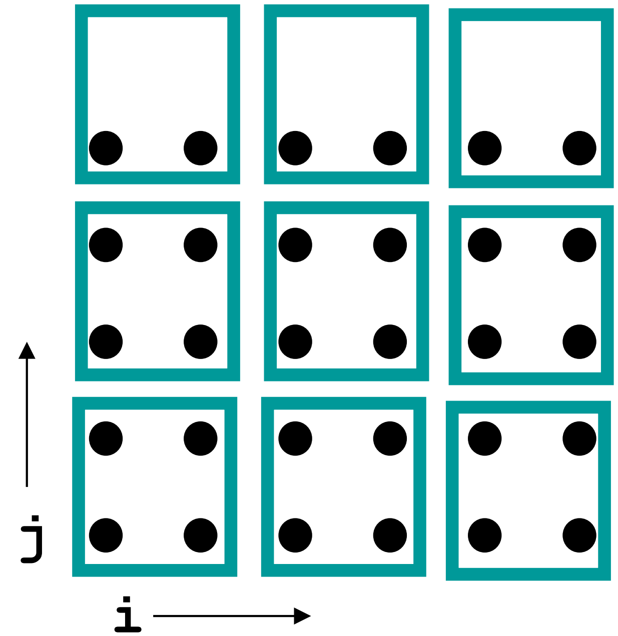
A loop transformation that ...

- groups iteration points into tiles that are executed atomically
- can improve spatial and temporal data locality
- can expose larger granularities of parallelism




```
do ii = 1,6, by 2
 do jj = 1, 5, by 2
  do i = ii, ii+2-1
   do j = jj, min(jj+2-1,5)
     A(i,j) = ...
```

Specifying Tiling



Rectangular tiling

- tile size vector $(s_1, s_2, ..., s_d)$
- tile offset, $(o_1, o_2, ..., o_d)$

Possible Transformation Mappings

- creating a tile space

$$\{[i,j] \rightarrow [ti,tj,i,j] \quad | \quad ti = floor((i-o_1)/s_1) \wedge tj = floor((j-o_2)/s_2)\}$$

- keeping tile iterators in original iteration space

$$\{[i,j] \rightarrow [ii,jj,i,j] \quad | \quad ii = s_1 floor((i-o_1)/s_1) + o_1 \wedge jj = s_2 floor((j-o_2)/s_2) + o_2\}$$

Using ISCC to do code generation for tiling

Iteration space: `S := { s[i,j] : 1<=i<=6 && 1<=j<=5 };`

Tiling specification

```
T :={s[i,j]->[ti,tj,i,j]: ti=(i-1)/2&&tj=(j-1)/2};
codegen (T*S);  // doesn't work in iscc
```

Getting rid of integer divison

`ti=(i-1)/2` **becomes**

`0<=ri<2 && (i-1)=2*ti+ri`

`tj=(j-1)/2` **becomes**

`0<=rj<2 && (j-1)=2*tj+rj`

```
T :={s[i,j]->[ti,tj,i,j]: exists ri,rj:
  0<=ri<2 && i-1=ti*2+ri
  && 0<=rj<2 && j-1=tj*2+rj}; // works!!
```

Polyhedral Model

Some History

- [Banerjee90] Uptal Banerjee, "Unimodular transformations of double loops," In Advances in Languages and Compilers for Parallel Computing, 1990.
- [Wolf & Lam 91] Wolf and Lam, "A Data Locality Optimizing Algorithm," In Programming Languages Design and Implementation, 1991.
- [Kelly and Pugh 95] Kelly and Pugh, "A unifying framework for iteration reordering transformations," In IEEE First International Conference on Algorithms and Architectures for Parallel Processing (ICAPP)
- [Feautrier 96] Paul Feautrier, "Automatic Parallelization in the Polytope Model," In The Data Parallel Programming Model.

Polyhedral Model

Some key components

- Representing loops as sets
- Representing data dependences as dependence vectors
- Representing transformations as functions
- Applying transformations to generate transformed code