IEGen

Automatically Generating Inspectors and Executors

Alan LaMielle

Colorado State University
Tiling Meeting

November 5, 2008

Overview

An example polyhedral computation

Physical Model:

Iteration Space:

An example polyhedral computation

```
    for(i=1; i<=n-1; i++)
    {
S1:    fx[i-1]+=x[i-1]-x[i];
S2:    fx[i]  +=x[i]  -x[i-1];
    }
```

An example polyhedral computation

```
affine example {N|1<N}
given (double X  {i|0<=i<=N-1}; )
returns (double FX {i|0<=i<=N-1}; )
through
   FX[i] =
     case
        {|i=0}:      X[i]-X[i+1];
        {|0<i<N-1}:  X[i]-X[i-1]+
                     X[i]-X[i+1];
        {|i=N-1}:    X[i]-X[i-1];
     esac;
.
```

Iteration space:

- Constraint Representation: $\begin{bmatrix} 1 \\ -1 \end{bmatrix} \begin{bmatrix} i \end{bmatrix} \geq \begin{bmatrix} 1 \\ n-1 \end{bmatrix}$
- Vertex/Ray Representation: $(1), (n-1)$
- Set/Relation Syntax: $\{[i] : 1 \leq i \leq n-1\}$

Data spaces of `x` and `fx`:

- Constraint Representation: $\begin{bmatrix} 1 \\ -1 \end{bmatrix} \begin{bmatrix} i \end{bmatrix} \geq \begin{bmatrix} 0 \\ n-1 \end{bmatrix}$
- Vertex/Ray Representation: $(0), (n-1)$
- Set/Relation Syntax: $\{[i] : 0 \leq i \leq n-1\}$

Accesses:

`S1`'s first access of `x` (`x[i-1]`):

$$\begin{bmatrix} 1 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & 1 \\ 1 & -1 \\ -1 & 1 \end{bmatrix} \begin{bmatrix} i \\ i' \end{bmatrix} \geq \begin{bmatrix} 1 \\ n-1 \\ 1 \\ n-1 \\ -1 \\ 1 \end{bmatrix}$$

Set Relation Syntax:

$$\{[i] \rightarrow [i'] : 0 \leq i \leq n-1 \wedge 0 \leq i' \leq n-1 \wedge i' = i-1\}$$

Other accesses are similar...

Loop reversal:

- Unimodular Transformation Framework:

$$[-1]\,[i] = [-i]$$

- Kelly-Pugh Transformation Framework:

$$\{[i] \rightarrow [i'] : i' = -i\}$$

Code Generators:

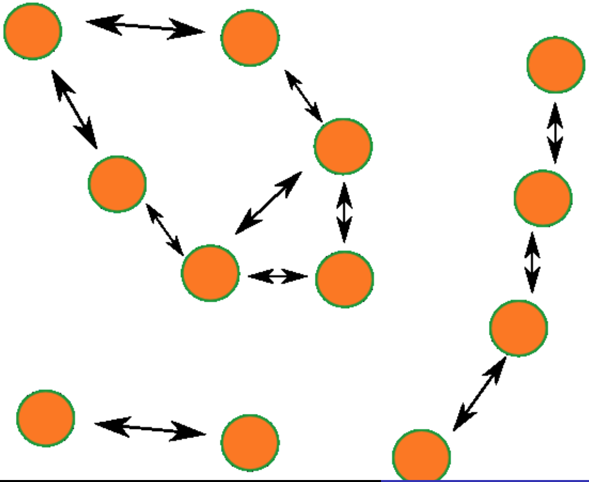
- CLooG
- Omega
- TLOG
- HiTLoG

Common libraries:

- Polylib
- PIP

Dwarf	Support
Dense Linear Algebra	✓
Sparse Linear Algebra	✗
Spectral Methods	✗
N-Body Methods	✗
Structured Grids	✓
Unstructured Grids	✗
Monte Carlo	✓
Combinational Logic	✗
Graph Traversal	✗
Dynamic Programming	✓
Backtrack and Branch & Bound	✗
Construct Graphical Methods	✓
Finite State Machines	✗

An example sparse computation



An example sparse computation

```
    for(i=0; i<n_inter; i++){
S1:     fx[inter1[i]]+=
                  x[inter1[i]]-
                  x[inter2[i]];
S2:     fx[inter2[i]]+=
                  x[inter1[i]]-
                  x[inter2[i]];
    }
```

The following needs to be specified:

- Symbolic Constants
- Data Spaces
- Computation inputs/outputs
- Statements with iteration spaces and scattering functions
- Access Relations
- Data dependences
- *Index Arrays (Uninterpreted Functions)*

Three types acting on data spaces and iteration spaces:

	Data Space	Iteration Space
Permutation	CPack	LexMin
Projection	?	?
Embedding	Smashing	Tiling

Specify the relation from the original data/iteration space to the new data/iteration space.

Omega?

Other reordering specific Inspector/Executor generators.

Dwarf	Support
Dense Linear Algebra	✓
Sparse Linear Algebra	✓
Spectral Methods	✓
N-Body Methods	✓
Structured Grids	✓
Unstructured Grids	✓
Monte Carlo	✓
Combinational Logic	✗
Graph Traversal	✗
Dynamic Programming	✓
Backtrack and Branch & Bound	✗
Construct Graphical Methods	✓
Finite State Machines	✗

What is good about the SPF:

- Supports far more application domains
- Still relies on similar theory
- Can utilize existing tools for code generation (CLooG)

Assumptions we make about UFSs:

- $i = j \rightarrow f(i) = f(j)$
- Rectangular bounds

Issues when working with SPF:

- Free variable as parameter, must take inverse
- Computing bounds of input or output tuples

Demo time!

How do fuzzy dependences in Alphabets compare with UFSs in the SPF?