Miscellaneous and Polyhedral Operations

Logistics

–Intermediate reports are due tomorrow

–HW6 (posted) and HW7 (posted) due April 5th

–Tuesday April 4th, help session during class with Manaf, Tomo, and Andy

–Distance students can send email to cs560@cs.colostate.edu with questions

Previously

- Universal Occupancy Vector (UOV)
- Transformation specification in AlphaZ

Today

- Some Quiz 2 problems
- Legality of OV when a schedule is given
- Definitions of reuse
- Imperfectly nested loops: dependences and transformations
- Starting polyhedral operations with representation of polyhedra

Quiz 2 Problems

Transitive closure and memory based dependences

- Use an exact data dependence example that we have done in class so far (either from a HW, midterm, or in the class notes) to explain how the transitive closure of exact/direct flow dependences are equivalent to the set of flow memory based dependences.

```
for (i=0; i<N; i++) {
  for (j = i; j < N; j++) {
    a[j] = a[j-1];
}}
```

```
Exact flow
{[i,j] -> [i,j+1] : 0<=i<=j<=N-2}
Memory based flow
{[i,j] -> [i',j+1] : 0<=i<=j<=N-2 && i<i'< N} union
 {[i,j] -> [i,j+1] : 0 <= i <= j <= N-2}
```

Above is a counter example to the assertion!

Note that the memory flow dependences encode the constraints due to the storage mapping.

Quiz 2 Problems

Occupancy Vector

- Once an occupancy vector has been selected to represent a storage mapping that vector will become a dependence vector as well. What kind of dependence will that dependence vector represent in the newly storage mapped code (anti, output, or flow)?

Universal Occupancy Vector (UOV)

<First see the UOV slides>

Occupancy vector indicates iterations that share storage

- $\vec{ov} = \vec{q} - \vec{p}$, then iterations q and p share storage
- Pick a storage mapping by ensuring that $\vec{mv} \cdot \vec{ov} = 0$ in
  $$SM_{ov}(\vec{q}) = \vec{mv} \cdot \vec{q} + shift + modterm$$

Determining if a storage mapping is legal

- For any schedule, universal occupancy vector
  - Let $V = \{\vec{v_1}, ..., \vec{v_m}\}$ be the set of value/exact flow dependences
  - A universal occupancy vector $\vec{ov}$ must satisfy the following set of equations with all a_{ij} being integers such that $a_{ii} > 0, a_{ij} \geq 0$

$$\begin{aligned} \vec{ov} &= a_{11}\vec{v_1} + ... + a_{1m}\vec{v_m} \\ &... \\ \vec{ov} &= a_{m1}\vec{v_1} + ... + a_{mm}\vec{v_m} \end{aligned}$$

- When a schedule is specified the introduced output dependence needs to be a legally transformed by that schedule and consider parallel <show examples>

Definitions of Data Reuse

Terms

- Data more generally used here to refer to locations in memory and values.

Temporal Data Reuse

- A location in memory is used in more than one read or write during the computation.

Spatial Data Reuse

- Adjacent locations in memory are read to or written to during the computation.

Value Data Reuse

- A value is read more than once during the computation.

Storage Data Reuse

- A storage location is written to and/or read from more than once during the computation.

Imperfectly Nested Loops (dependences I)

```
  /* Ring blur filter */
  for (i=1;i<length-1;i++)
    for (j=1;j<width-1;j++)
R     Ring[i][j]=(Img[i-1][j-1]+Img[i-1][j]+Img[i-1][j+1]+
                  Img[i][j+1]  +            Img[i][j-1]  +
                  Img[i+1][j-1]+Img[i+1][j]+Img[i+1][j+1])/8;

  /* Roberts edge detection filter */
  for (i=1;i<length-2;i++)
    for (j=2;j<width-1;j++)
P     Img[i][j]=abs(Ring[i][j]-Ring[i+1][j-1])+
                abs(Ring[i+1][j]-Ring[i][j-1]);
```

Figure 1. `Ring-Roberts` edge detection for noisy images

Imperfectly Nested Loops (dependences II)

```
# symbolic L,W;
# # specify dependence relation
# D_R_to_P_flow := { [i,j] -> [i',j'] :
#     # equality constraint(s)
#     ((i = i' && j = j' ) || (i=i'+1 && j=j'-1)
#     || (i=i'+1 && j=j') || (i=i' && j=j'-1))
#     # loop bounds
#     && 1 <= i < L-1  && 1 <= j < W-1
#     && 1 <= i' < L-2  && 2 <= j' < W-1
#     # precedence constraints
#     # Not needed because we already know all i,j execute before i',j'
# };
# D_R_to_P_flow;

 {[i,j] -> [i,j] : 1 <= i <= L-3 && 2 <= j <= W-2} union
 {[i,j] -> [i-1,j+1] : 2 <= i <= L-2 && 1 <= j <= W-3} union
 {[i,j] -> [i-1,j] : 2 <= i <= L-2 && 2 <= j <= W-2} union
 {[i,j] -> [i,j+1] : 1 <= i <= L-3 && 1 <= j <= W-3}
```


Imperfectly Nested Loops (transformation in omega)

```
symbolic L,W;
##### iteration space for statement R
R := {[i,j] : 1<=i<L-1 && 1<=j<W-1};
# // In C code would have macro:
# #define s1(i,j) Ring[i][j] = (Img[(i)-1][(j)-1]+ ...
##### iteration space for statement P
P := {[i,j] : 1<=i<L-2 && 2<=j<W-1};
# // In C code would have macro:
# #define s2(i,j) Img[i][j] = abs(Ring[i][j]- ...

# Original loop schedule
theta_R := {[i,j] -> [i,j]};
theta_P := {[i,j] -> [i+L-2,j]};
codegen 2 theta_R:R, theta_P:P;

# Schedule that uses fusion, shifting, and index set splitting
# from Pouchet08 paper.
theta_R := {[i,j] -> [i,j]};
theta_P := {[i,j] -> [i+2,j]};
codegen 2 theta_R:R, theta_P:P;
```

Some differences between AlphaZ and Omega

Statement representation

- In omega use macros in C. Statement syntax is not part of omega.
- In AlphaZ have syntax for expressing statements. Have to convert the computation to single assignment.

Storage mapping

- Omega can use the original storage mapping of the computation. It also has a tcodegen feature that does enable some storage mapping specification.
- AlphaZ has an inefficient default storage mapping, but enables the orthogonal specification of the storage mapping.

Verification

- With omega can calculate dependences and then check the validity of a transformation/schedule.
- AlphaZ can verify a schedule and storage mapping automatically.

Algorithms needed for automation

Operations on sets and relations

- Union iteration space sets
- Union relations that represent dependences
- Apply a relation to a set to model transforming a loop and to check transformation legality
- Compose two relations to model composing transformations

Scheduling

- Determine an efficient and legal schedule
- Determine which loops should be parallel

Storage Mapping

- If not using UOV, then need to do this in coordination with the scheduling

Code Generation

- Given a schedule and which loops to parallelize and/or tile, generate efficient code
- Code generation for parameterized tiles

Representing computational sets as Polyhedra

Terminology referring to these sets

- Iteration space
- Domain
- Integer tuple space

Polyhedron

- A set $S \in \mathbb{R}^m$ is a polyhedron if there exists a system of finite inequalities $A\vec{x} \geq \vec{b}$ such that $P = \{\vec{x} \in \mathbb{R}^m | A\vec{x} \geq \vec{b}\}$
- Equivalently it is the intersection of finitely many half-spaces.
- Note that a polyhedron is a rational polyhedron. We have then been assuming intersection with the unit integer lattice.

Integral Polyhedron

- A set of the form: $P = \{\vec{x} \in \mathbb{Z}^m | Q\vec{x} \geq \vec{q}\}$
- Parametric family of polyhedra due to input parameters, or symbolic constants $P = \{\vec{x} \in \mathbb{Z}^m | Q\vec{x} \geq (\vec{q} + B\vec{p})\}$

Constraint Representation $P = \{\vec{x} \in \mathbb{Z}^m | Q\vec{x} \geq (\vec{q} + B\vec{p})\}$

Implementation

- Store a coefficient matrix for all the constraints
- Associate each column with an iterator or a parameter/symbolic constant

Interpretation of the above representation (assume all column vectors)

$$P = \{\begin{pmatrix} \vec{x} \\ \vec{p} \end{pmatrix} \in \mathbb{Z}^m | \begin{bmatrix} Q & B & \vec{q} \end{bmatrix} \begin{pmatrix} \vec{x} \\ \vec{p} \\ 1 \end{pmatrix} \geq \vec{0}\}$$

- "Parameterized family of polyhedra is just a single higher-dimensional polyhedron." Foundations II notes by Sanjay Rajopadhye.

Example

```
for (i=0; i<N; i++) {
  for (j=i+1; j<N; j++) {
  A[j][i] = A[j][i]/A[i][i];
  for (k=i+1; k<N; k++) {
    A[j][k] -= A[i][k]*A[j][i];
}}}
```

Operation: Intersection

Intersection between polyhedral sets

- When you intersect two polyhedral sets the result is a polyhedral set.

$$P = \{\vec{x} \in \mathbb{Z}^m | \vec{x} \in P_1 \wedge \vec{x} \in P_2\}$$

$$P_1 = \{\begin{pmatrix} \vec{x} \\ \vec{p_1} \end{pmatrix} \in \mathbb{Z}^m | \begin{bmatrix} Q_1 & B_1 & \vec{q_1} \end{bmatrix} \begin{pmatrix} \vec{x} \\ \vec{p_1} \\ 1 \end{pmatrix} \geq \vec{0}\}$$

$$P_2 = \{\begin{pmatrix} \vec{x} \\ \vec{p_2} \end{pmatrix} \in \mathbb{Z}^m | \begin{bmatrix} Q_2 & B_2 & \vec{q_2} \end{bmatrix} \begin{pmatrix} \vec{x} \\ \vec{p_2} \\ 1 \end{pmatrix} \geq \vec{0}\}$$

Next Time

Lecture

- More operations on polyhedral sets and relations

Schedule

- Project intermediate report due March 28th tomorrow
- April 3rd will be a lab day during class, Manaf, Tomo, and Andy will help people with AlphaZ and Pluto. Distance students can email questions or use the discussion board.
- HW6 and HW7 will BOTH be due April 4th

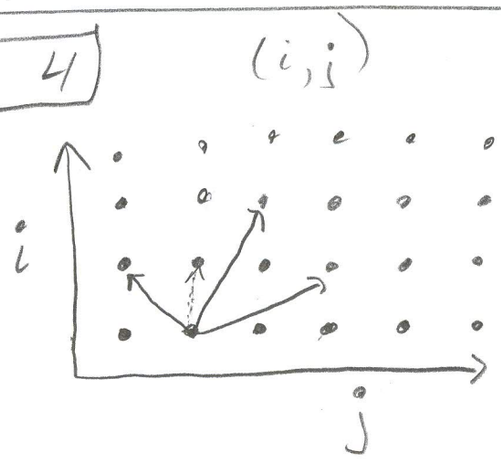
3/27/12

Slide 2

```
for (i=0; i<N; i++) {
    a = a + b[i],
}
```

memory based flow deps
i i'
- no equality constraints
- prec $i < i'$
- loop bounds

Slide 4



exact flow
(1, -1)
(2, 1)
(1, 2)

schedule $(i,j) \rightarrow (j,i)$
No due to (1, -1) becoming (-1, 1)

schedule $(i,j) \rightarrow (i,j)$
ov (1,0) legal?

Still an issue