Omega Basics

- Manipulates integer tuple sets and relations.
- Integer tuple sets => Iteration spaces
  \{[i,j] : 0 \leq i < n \, \&\& \, 0 \leq j < n\}
- Integer tuple relations => Affine transformations
  \{[i,j] \rightarrow [j,i]\}
Basics...

- **symbolic var** - Defines var as a symbolic constant that can be used in all later expressions.
- **var := RelExpr** - Computes the relational expression and binds the variable name to the result.
- **RelExpr** - Computes and prints the relational expression
- **codegen** - Generates the SPMD code
How to affine partition a program?

1. Specify the affine partitioning constraints (boundary conditions and data dependences) for a given program.
2. Input the constraints into the Omega calculator to simplify them if necessary.
3. Manually solve the (simplified) affine constraints and get an affine partition that maximizes the degree of parallelism with no communication.
4. Input the affine partitioning to the Omega calculator and generate the code.
Omega setup

On corn/myth:
$ alias oc /usr/class/cs243/omega/bin/oc

On your local machine:
$ git clone git://github.com/davewathaverford/the-omega-project.git omega
$ cd omega
$ make depend
$ make all

Folder with examples
/usr/class/cs243/omega/ (domain.in, cs243_1.in ...)
Example 1

```c
for (i = 1; i <= N; i++) {
    Y[i] = Z[i]; /* (s1) */
    X[i] = Y[i-1]; /* (s2) */
}
```

Are the iterations independent of each other?
Example 1

```
for (i = 1; i <= N; i++) {
    Y[i] = Z[i];  /* (s1) */
    X[i] = Y[i-1];  /* (s2) */
}
```

We have Loop carried dependences!
The dependencies

for (i = 1; i <= N; i++) {
    Y[i] = Z[i];    /* (s1) */
    X[i] = Y[i-1];  /* (s2) */
}

Iteration $i$ depends on iteration $i-1$. 
Parallelize!

- We want to parallelize the code with no communication.
- We want to find (parallelize)
  \[ p = C_1 \times i_1 + c_1 \text{ and } p = C_2 \times i_2 + c_2 \]
- Accesses to the same memory go to same processor (No communication)
  \[ i_1 = i_2 - 1 \]
Affine Equations

Step 1: Write the affine equations.

\[ 1 \leq i_1 \leq N \quad \text{(Bounds)} \]
\[ 1 \leq i_2 \leq N \quad \text{(Bounds)} \]
\[ i_1 = i_2 - 1 \quad \text{(No communication)} \]

\( i_1 \) - Iteration number of statement 1
\( i_2 \) - Iteration number of statement 2
Simplify with Omega

Step 2: Input constraints to Omega

# symbolic N;
# D := {[i1,i2]: 1 <= i1 <= N && 1 <= i2 <= N && i1 = i2-1};
# D;

{[i1,i1+1]: 1 <= i1 < N}

Constraints
1 <= i1 <= N
1 <= i2 <= N
i1 = i2 - 1
Step 3: Manually solve to get affine partition

Constraints:

\[ C_1 \times i_1 + c_1 = C_2 \times i_2 + c_2 \quad \text{and} \quad i_2 = i_1 + 1 \]

\[ \Rightarrow \text{for all } i_1 \text{ such that } 1 \leq i_1 \leq N : \]

\[ C_1 \times i_1 + c_1 = C_2 \times (i_1 + 1) + c_2 \]

\[ \Rightarrow C_1 = C_2 \text{ and } C_2 + c_2 - c_1 = 0 \]

A simple non-zero rank solution:

\[ C_1 = C_2 = 1 \]

\[ c_1 = 0 \text{ and } c_2 = -1 \]
Step 4: Generate code from the affine partition

```plaintext
# symbolic N;
# S1 := {[p,i1]: p = i1 && 1 <= i1 <= N};
# S2 := {[p,i2]: p = i2 - 1 && 1 <= i2 <= N};
# codegen S1, S2;
if (N >= 1) {
    s2(0,1);
}
for(t1 = 1; t1 <= N-1; t1++) {
    s1(t1,t1);
    s2(t1,t1+1);
}
if (N >= 1) {
    s1(N,N);
}
```

**Original Code:**

```plaintext
for (i = 1; i <= N; i++) {
    Y[i] = Z[i];  /* s1 */
    X[i] = Y[i-1];/* s2 */
}
```

**Transformed code:**

```plaintext
if (N>=1)
    X[1]=Y[0];
for (p=i; p<=N-1; p++){
    Y[p]=Z[p];
    X[p+1]=Y[p];
}
if (N>=1)
    Y[N]=Z[N];
```
Example 2

for (i=1; i<=N; i++)
    Y[2*i] = Z[2*i]; /*s1*/
for (j=1; j<=2N; j++)
    X[j] = Y[j]; /*s2*/
Affine Expressions

\[ 1 \leq i \leq N \quad \text{(Bounds)} \]
\[ 1 \leq j \leq 2^N \quad \text{(Bounds)} \]
\[ 2i = j \quad \text{(No communication)} \]

- \( i \) - Iteration number of first loop
- \( j \) - Iteration number of second loop
Simplify with Omega

# symbolic N;
# D := {[i,j]: 1 <= i <= N && 1 <= j <= 2*N && 2*i = j};
# D;

{{i,2i]: 1 <= i <= N}

Constraints:
1 <= i <= N
1 <= j <= 2*N
2*i = j
Maximize degree of parallelism

Constraints:
\[ C_1 \cdot i + c_1 = C_2 \cdot j + c_2 \quad \text{and} \quad j = 2 \cdot i \]

\[ \implies \text{for all } i \text{ such that } 1 \leq i \leq N : \]
\[ C_1 \cdot i + c_1 = C_2 \cdot (2 \cdot i) + c_2 \]

\[ \implies C_1 = 2 \cdot C_2 \quad \text{and} \quad c_2 = c_1 \]

A simple non-zero rank solution:
\[ C_2 = 1 \quad C_1 = 2 \]
\[ c_1 = c_2 = 0 \]
Step 4: Generate code from the affine partition

```plaintext
# symbolic N;
# S1 := {{p,i}: p = 2i && 1 <= i <= N};
# S2 := {{p,j}: p = j && 1 <= j <= 2*N};
# codegen S1,S2;

for(t1 = 1; t1 <= 2*N; t1++) {
    t2=intDiv((t1+1),2);
    s1(2*t2,t2);
    s2(t1,t1);
}
```

**Original Code:**
```plaintext
for (i=1; i<=N; i++)
    Y[2*i] = Z[2*i];
for (j=1; j<=2*N; j++)
    X[j]=Y[j];
```

**Transformed code:**
```plaintext
for (p=1; p<=2*N; p++){
    if (p mod 2 == 0)
        Y[p] = Z[p];
    X[p] = Y[p];
}
Example 3

```c
for (i = 0; i < N; i++) {
    for (j = 0; j < N; j++) {
        A[i + j][3*i + j] = A[2*i][5 - 2*j]; /* (s1) */
    }
}
```

Dependencies ?
Example 3

for (i = 0; i < N; i++) {
    for (j = 0; j < N; j++) {
        A[i + j][3*i + j] = A[2*i][5 - 2*j]; /* (s1) */
    }
}

Read write dependence between iterations?
# symbolic N;

# \( D := \{ [i_1, j_1, i_2, j_2] : 0 \leq i_1, j_1, i_2, j_2 < N \land i_1 + j_1 = 2i_2 \land 3i_1 + j_1 = 5 - 2j_2 \} \); # \( D \);

\{ [i_1, j_1, i_2, j_2] : \text{FALSE} \}

Constraints:

\( 0 \leq i_1, j_1 < N \)
\( 0 \leq i_2, j_2 < N \)
\( i_1 + j_1 = 2i_2 \)
\( 3i_1 + j_1 = 5 - 2j_2 \)

Code:

```c
for (i = 0; i < N; i++) {
    for (j = 0; j < N; j++) {
    }
}
```
False, implies all the constraints are never satisfied together i.e the set contains no points.

Hence we can conclude with certainty that we don’t have any RW dependences.
Example 4: Exercise!

```c
for (i = 1; i <= 100; i++) {
    for (j = 1; j <= 100; j++) {
        X[i, j] = X[i, j] + Y[i - 1, j]; /* (s1) */
        Y[i, j] = Y[i, j] + X[i, j - 1]; /* (s2) */
    }
}
```

Example from class.
Try it out yourself first!

Solution available in cs243_3.in
Questions ?