

Cetus Compiler

Status

- Written in Java
- Parses C
- Announced new release late last night
- J

Synchronization-Free Parallelism

Today

- SPMD and OpenMP programming models
- Synchronization-free affine partitioning algorithm
- Deriving primitive affine transformations

Two Parallel Programming Models

SPMD

- single program multiple data
- program should check what processor it is running on and execute some subset of the iterations based on that

```
MPI_Init(&Argc, &Argv);  
// p is the processor id  
MPI_Comm_rank(MPI_COMM_WORLD, &p);
```

OpenMP

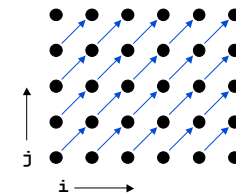
- shared memory, thread-based parallelism
- pragmas indicate that a loop is fully parallel

```
#pragma omp for  
for (i=0; i<N; i++) {  
}
```

Diagonal Partitioning Example

Example

```
do i = 1, 6  
  do j = 1, 5  
    A(i, j) = A(i-1, j-1) + 1  
  enddo  
enddo
```



Goal

- Determine an affine space partitioning that results in no synchronization needed between processors.

Space-Partition Constraints

Accesses sharing a dependence should be mapped to the same processor

- loop bounds $write : A(i, j), \quad read : A(i', j')$
 $1 \leq i, i' \leq 6$
 $1 \leq j, j' \leq 5$

– equality constraints on dependence

$$\begin{aligned} i &= i' - 1 \\ j &= j' - 1 \end{aligned}$$

– equality constraints on space partition

$$\begin{bmatrix} C_{11} & C_{12} \end{bmatrix} \begin{bmatrix} i \\ j \end{bmatrix} + [c_1] = \begin{bmatrix} C_{11} & C_{12} \end{bmatrix} \begin{bmatrix} i' \\ j' \end{bmatrix} + [c_1]$$

Solving the Space-Partition Constraints

Ad-hoc approach

- Reduce the number of unknowns $t_1 = i = i' - 1$
 $t_2 = j = j' - 1$

– Simplify

$$\begin{bmatrix} C_{11} - C_{11} & C_{22} - C_{22} \end{bmatrix} \begin{bmatrix} t_1 \\ t_2 \end{bmatrix} + [c_1 - c_1 - C_{11} - C_{12}] = 0$$

$$-C_{11} = C_{12}$$

– Determine independent solutions for space partition matrix

$$\begin{aligned} C_{11} &= 1 \\ C_{12} &= -1 \end{aligned}$$

– Find constant terms (would like min of mapping to be non-negative)

$$\begin{aligned} \min(C_{11}i + C_{12}j + c_1) &= \min(i - j + c_1) = 1 - 5 + c_1 = 0 \\ c_1 &= 4 \end{aligned}$$

Generate Simple Code

Algorithm 11.45 Generate code that executes partitions of a program sequentially

- for each statement, project out all loop index variables from the system with original loop bounds and space partition constraints

$$0 \leq p \leq 9$$

- use FM-based code gen algorithm to determine bounds over partitions
 - not needed for example
- union the partition bounds over all statements
 - not needed for example
- insert space partition predicate before each statement

```
do p = 0, 9
  do i = 1, 6
    do j = 1, 5
      if (i-j+4 = p) A(i, j) = A(i-1, j-1)+1
```

Eliminate Empty Iterations

Apply FM-based code generation algorithm to resulting iteration space

- for each statement
 - use unioned p bounds, the statement iteration space and the space partition constraints
 - determine new bounds for the statement iteration space
 - union the iteration space for all statements in the same loop
- For example, did this when determining bounds on p

```
do p = 0, 9
  do i = max(1, -3+p), min(6, p+1)
    do j = max(1, i+4-p), min(5, i+4-p)
      if (i-j+4 = p) A(i, j) = A(i-1, j-1)+1
```

Eliminate Tests from Innermost Loops

General approach: apply the following repeatedly

- select an inner loop with statements with different bounds
- split the loop using a condition that causes a statement to be in only one of the splits
- generate code for the split iteration spaces

```
do p = 0, 9
  do i = max(1, -3+p), min(6, p+1)
    do j = max(1, i+4-p), min(5, i+4-p)
      A(i, j) = A(i-1, j-1)+1
```

Using the Two Programming Models

SPMD and MPI

```
MPI_Comm_rank(MPI_COMM_WORLD, &p);

for (i = max(1, -3+p); i <= min(6, p+1); i++)
  for (j = max(1, i+4-p); j <= min(5, i+4-p); j++)
    A[i][j] = A[i-1][j-1]+1
```

OpenMP

```
#pragma omp for
for (p = 0; p <= 9; p++)
  for (i = max(1, -3+p); i <= min(6, p+1); i++)
    for (j = max(1, i+4-p); j <= min(5, i+4-p); j++)
      A[i][j] = A[i-1][j-1]+1
```

Derive Re-indexing by using Space Partition Constraints

Source Code

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

Transformed Code

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

Concepts

Two Parallel Programming Models

- SPMD
- OpenMP

Deriving a Synchronization-Free Affine Partitioning

- setting up the space partition constraints (keep iterations involved in a dependence on the same processor)
- solve the sparse partition constraints (linear algebra)
- eliminate empty iterations (Fourier-Motzkin)
- eliminate tests from inner loop (more Fourier-Motzkin)
- using the above to derive primitive affine transformations

Next Time

Lecture

- Tiling!

Suggested Exercises

- Be able to derive the synchronization-free affine partitioning for Example 11.41 in the book.
- Show how the other primitive affine transformations are derived.