Multi-cores are here! What’s the right question?

- **Q1.** How to parallelize a code automatically?
  - Most programs, as coded, are sequential
  - No silver bullet: Tried functional programming, data flow, automatic parallelization
  - Computation-intensive codes have parallelism, but:
    - Coverage: Amdahl’s Law
    - Communication makes naively parallelized code runs slower
- **Q2.** How to generate efficient parallel code automatically?
  - KEY: Locality
  - Place instructions using the same data on the same processor
- **Q3.** How to optimize locality in sequential code automatically?
  - Place instructions using the same data close together in time
- **Q4.** How to write efficient parallel code?
  - Place related operations close in time and in space.

*Demonstrate use of another mathematic concept: linear algebra*
1. Shared Memory Machines

Performance on Shared Address Space Multiprocessors: Parallelism & Locality

(A) What is Affine Partitioning?
An Contrived but Illustrative Example

\[
\begin{align*}
\text{FOR } i &= 1 \text{ TO } n \\
\text{FOR } j &= 1 \text{ TO } n \\
A[i, j] &= A[i, j] + B[i-1, j]; & (S_1) \\
B[i, j] &= A[i, j-1] \times B[i, j]; & (S_2)
\end{align*}
\]
Best Parallelization Scheme

Algorithm finds affine partition mappings for each instruction:

S1: Execute iteration (i, j) on processor i-j.
S2: Execute iteration (i, j) on processor i-j+1.

SPMD code: Let p be the processor's ID number

```plaintext
if (1-n <= p <= n) then
    if [1 <= p) then
        B[p, 1] = A[p, 0] * B[p, 1];  \text{(S2)}
    for i, f \in \min[1, n-1+p] do
        A[i, i-1-p] = A[i+1, i-1-p] + B[i-1, i-1-p];  \text{(S1)}
        B[i, i-1-p+1] = A[i, i-1-p] * B[i, i-1-p+1];  \text{(S2)}
    if (p <= 0) then
        A[n+p, n] = A[n+p, n] + B[n+p-1, n];  \text{(S1)}
```

Maximum Parallelism & No Communication

For every pair of data dependent accesses F1l1+f1 and F2l2+f2:

Find C_{i1}, C_{i2}, C_{j1}, C_{j2};
∀ l_1, l_2 \quad F_1 l_1 + f_1 = F_2 l_2 + f_2 \implies C_{i1} l_1 + c_1 = C_{i2} l_2 + c_2

with the objective of maximizing the rank of C_{i1}, C_{i2}.
Rank of Partitioning = Degree of Parallelism

Affine Mapping

\[
\begin{bmatrix}
0 & 0 \\
0 & 1 \\
1 & 0 \\
0 & 1
\end{bmatrix}
\]

Rank

0 1 2

Mapped to same processor

(B) What is blocking? Example: Matrix Multiplication

Data Accessed

1002000 65024
Experimental Results

With Blocking
Without Blocking

Code Transform

- **Before**
  ```
  for (i = 0; i < n; i++) {
    for (j = 0; j < n; j++) {
      for (k = 0; k < n; k++) {
        Z[i,j] = Z[i,j] + X[i,k] * Y[k,j];
      }
    }
  }
  ```

- **After**
  ```
  for (ii = 0; ii < n; ii = ii+B) {
    for (jj = 0; jj < n; jj = jj+B) {
      for (kk = 0; kk < n; kk = kk+B) {
        for (i = 0; i < n; i++) {
          for (j = 0; j < n; j++) {
            for (k = 0; k < n; k++) {
              Z[i,j] = Z[i,j] + X[i,k] * Y[k,j];
            }
          }
        }
      }
    }
  }
  ```
Optimizing Arbitrary Loop Nesting Using Affine Partitions (chotst, NAS)

```
DO 1 J = 0, N
   10 = MAX (-M, -J)
   DO 2 I = 10, -1
      DO 3 JJ = 10 - I, -1
         DO 4 L = 0, NNAT
         END DO
      END DO
   END DO
   DO 5 K = N, 0, -1
      DO 6 JJ = 1, MIN (M, K)
         DO 7 L = 0, NNAT
            A(L, I, J) = A(L, I, J) * A(L, 0, I+J)
         END DO
      END DO
      DO 8 L = 0, NNAT
         A(L, 0, J) = 1. / SQRT ( ABS (EPS + A(L, 0, J)) )
      END DO
   END DO
   DO 9 K = 0, N
      DO 10 L = 0, NNAT
         B(I, L, K) = B(I, L, K) * A(L, 0, K)
      END DO
      DO 11 JJ = 1, MIN (M, K)
         DO 12 L = 0, NNAT
            B(I, L, K) = B(I, L, K) * A(L, -JJ, K+JJ)
         END DO
      END DO
      DO 13 L = 0, NNAT
         B(I, L, K) = B(I, L, K) * A(L, -JJ, K) * B(I, L, K)
      END DO
   END DO
```

Chotst: Results with Affine Partitioning + Blocking

(Unimodular: a subset of affine partitioning for perfect loop nests)
Summary

- **Affine transforms**
  - Find maximum degree of coarse-grain parallelism
  - Linear algebra
    - Relationship between access pattern & linear algebra concepts
    - How to generate transformed code?
    - Where are the data dependences?
    - How to come up with the affine mapping?

- **Blocking**
  - Parallelism in 2D+ loops → opportunity for blocking

---

How to Use Linear Algebra

- **Loops (iteration space): n-dimensional polytopes**
  - How to generate code: Fourier-Motzkin Elimination

- **Access function:**
  - Rank of access functions
  - Reuse concept
  - Data dependence

- **Affine partitioning transform**
  - (next class)
2. Iteration Space

FOR i = 0 to 5
    FOR j = i to 7
        ...

- n-deep loop nests: n-dimensional polytope
- Iterations: coordinates in the iteration space
- Assume: iteration index is incremented in the loop
- Sequential execution order: lexicographic order
  - [0,0], [0,1], …, [0,6], [0,7],
  - [1,1], …, [1,6], [1,7], …

3. Code Generation
   Example: Loop Interchange (Loop Permutation)

for I = 1 to 4
    for J = 1 to 3
        Z[I,J] = Z[I-1,J]

for Y = 1 to 3
    for X = 1 to 4
        Z[Y,X] = Z[Y-1,X]
Transforming the code

FOR i=0 TO 20
  FOR j=max(10-i, i-10) to 10
    a[i, j] = ...

Step 1: substitute old indices with new.

FOR y =
  FOR x =
    a[x, y] = ...

Geometric Projection

For index i from inner to outer
Express bounds as exp. of outer indices
Eliminate index i from polytope

FOR y = 0 TO 10
  FOR x=max(0, 10-y) TO min(20, y+10)
    a[x, y] = ...

Bounds of x:

  0 ≤ x
  x ≤ 20
  10 – y ≤ x
  x ≤ y + 10

Bounds of y:

  0 ≤ y
  y ≤ 10
Fourier-Motzkin Elimination

- To eliminate a variable from a set of linear inequalities.
- To eliminate a variable $x_i$
  - Rewrite all expressions in terms of lower or upper bounds of $x_i$
  - Create a transitive constraint for each pair of lower and upper bounds.
- Example: Let $L$, $U$ be lower bounds and upper bounds resp
  - To eliminate $x_i$:

$$L(x_2, ..., x_n) \leq x_i \leq U(x_2, ..., x_n)$$

4. Affine Accesses: Iteration space $\rightarrow$ Array space

```
FOR i = 1 to n
  FOR j = 1 to n
```

<table>
<thead>
<tr>
<th>Access</th>
<th>Affine Exp</th>
<th>Rank</th>
<th>Nullity</th>
<th>Basis of Null Space</th>
</tr>
</thead>
<tbody>
<tr>
<td>X[1-1]</td>
<td>[1 0]</td>
<td>1</td>
<td>1</td>
<td>[0 1]</td>
</tr>
<tr>
<td>Y[1, j]</td>
<td>[1 0]</td>
<td>2</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Y[3, j+1]</td>
<td>[0 1]</td>
<td>1</td>
<td>1</td>
<td>[1 0]</td>
</tr>
<tr>
<td>Y[1, 2]</td>
<td>[0 0]</td>
<td>0</td>
<td>2</td>
<td>[0 1]</td>
</tr>
<tr>
<td>Z[1, i, 2*i+j]</td>
<td>[0 0]</td>
<td>2</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>
Informal Interpretation for Access Function \( F_i + f \)

- \( d \): loop depth; \( n \): # of iterations in each loop; \( a \): dimensions of the array
- \( F \) is an \( a \times d \) matrix; the loop has \( n^d \) iterations
  - It can access at most \( n^{\min(d, a)} \) memory locations

- **Rank**: # locations accessed? # iterations accessing the same data?
  - If \( r \) is the rank of \( F \), then \( O(n^r) \) locations accessed.
    - \( r \leq \min(d, a) \)
    - \( O(n^{d-r}) \) iterations access the same location.

- **Nullspace**: Which iterations refer to the same location?
  - \( d-r \) is the nullity of \( F \), dimension of the null space
  - \( \text{nullity}(F) + \text{rank}(F) = d \)
  - Let \( b_1, \ldots, b_{d-r} \) be the basis vectors of the null space
    - then iteration \( i \) accesses the same memory location
      - as iterations \( i + b_1, i + b_2, \ldots, i + \text{any linear combination of } b's \).

---

**Rank: Definition**

- **rank** of matrix \( F \)
  - the largest number of columns (or equivalently, rows)
    - that are linearly independent.

- A set of vectors is **linearly independent** if
  - none of the vectors can be written as
    - a linear combination of finitely many other vectors in the set.
Null Space of a Matrix

- The set of all solutions to the equations \( Fv = 0 \) is the **null space** of \( F \).
  - \( v = 0 \) vector is trivially in \( F \)'s null space.

- Let \( i, i' \) be two iterations. If \( F_i = F_i' \) then \( F(i-i') = 0 \)
  - Two iterations \( i, i' \) refer to the same array element if their difference \( i-i' \) belongs to the null space of matrix \( F \).

- \( \text{nullity} = \) dimension of the null space
  - \( \text{nullity}(F) + \text{rank}(F) = d \)
  - If \( \text{rank}(F) = d \), then its null space consists of only the null vector.

- The null space can be represented by its basis vectors.
  - Any linear combination of the basis vectors belongs to the null space.

---

5. Data Dependence Analysis

```plaintext
FOR i = 1 TO 100
  A[i] = B[i] + C[i]

FOR i = 1 TO 100
  FOR j = 1 TO 100

FOR i = 11 TO 20

FOR i = 11 TO 20
```

- A data dependence between two array accesses exists if some instance of one access may refer to the same location as an instance of the second.
- No data dependences \( \rightarrow \) all iterations can execute in parallel.
**Data Dependences in a Loop**

FOR \( i = 2 \) TO 5
\[ A[i-2] = A[i] + 1; \]

- Between \( A[i-2] \) and \( A[i] \)
  - There is a dependence if there exist two iterations \( i_w, i_r \)
    within the loop bounds such that iterations \( i_w, i_r \) write and read the same array element, respectively
  - \( \exists \) integers \( i_w, i_r, 2 \leq i_w, i_r \leq 5, \ i_w - 2 = i_r \)

- Between \( A[i-2] \) and \( A[i-2] \)
  - There is a dependence if there exist two iterations \( i_w, i_v \)
    within the loop bounds such that two distinct iterations \( i_w, i_v (i_w \neq i_v) \)
    write the same array element
  - \( \exists \) integers \( i_w, i_v, 2 \leq i_w, i_v \leq 5, \ i_w - 2 = i_v - 2, \ i_w \neq i_v \)

**Definition of Data Dependence**

For every pair of accesses not necessarily distinct \((F_1, f_1)\) and \((F_2, f_2)\)
one must be a write operation

Let \( B_1 i_1 + b_1 \geq 0, B_2 i_2 + b_2 \geq 0 \) be the corresponding loop bound constraints,
\[ \exists \text{ integers } i_1, i_2, \ B_1 i_1 + b_1 \geq 0, \ B_2 i_2 + b_2 \geq 0 \]
\[ F_1 i_1 + f_1 = F_2 i_2 + f_2 \]

If the accesses are not distinct, then add the constraint \( i_1 \neq i_2 \)

Complexity: integer linear programming, NP-complete
Data Dependence Analysis Algorithm

- Typically solving many tiny, repeated problems
  - Integer linear programming packages optimize for large problems
  - Use memoization to remember the results of simple tests

- Apply a series of relatively simple tests
  - GCD: 2*i, 2*i+1; GCD for simultaneous equations
  - Test if the ranges overlap

- Backed up by a more expensive algorithm
  - Use Fourier-Motzkin Elimination to test if there is a real solution
    - Keep eliminating variables to see if a solution remains
    - Add heuristics to encourage finding an integer solution.
  - Create 2 subproblems if a real, but not integer, solution is found.
    - For example, if x = .5 is a solution,
      create two problems,
      by adding x ≤ 0 and x ≥ 1 respectively to original constraint.

Conclusions

- Parallelism is plentiful in numeric code, but locality is important

- Two kinds of transforms
  - Affine partitioning maximizes the degree of parallelism without communication
    - Operations using same data are mapped to the same processor
  - Blocking: Exploit locality across multiple dimensions

- Linear algebra used in 2 ways
  - Loop iterations: polytope
    - Fourier-Motzkin Elimination to generate loop bounds
    - Projects polytope onto a lower-dimensional subspace
  - Affine functions
    - Rank – size of arrays accessed
    - Null space– iterations using the same data
    - Data dependence analysis: integer linear programming
      - Solved because they are usually simple problems