CS 243
Lecture 12
Introduction to Parallelization & Locality Optimization

1. Understanding Parallelism and Locality
2. Iteration Space
3. Code Generation: Fourier Motzkin Elimination
4. Access Functions
5. Data Dependence: Linear Integer Programming

Readings: Chapter 11.1 - 11.7

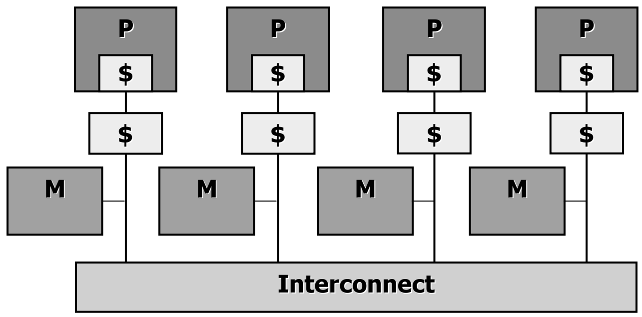
Multi-cores are here! What's the right question?

- 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

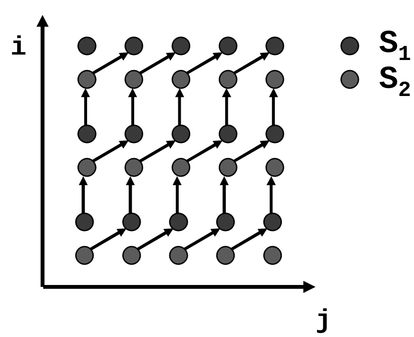
1. Shared Memory Machines

Performance on Shared Address Space Multiprocessors:
Parallelism & Locality



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

```
FOR i = 1 TO n
  FOR j = 1 TO n
    A[i,j] = A[i,j]+B[i-1,j];       (S1)
    B[i,j] = A[i,j-1]*B[i,j];       (S2)
```



Best Parallelization Scheme

Algorithm finds affine partition mappings for each instruction:

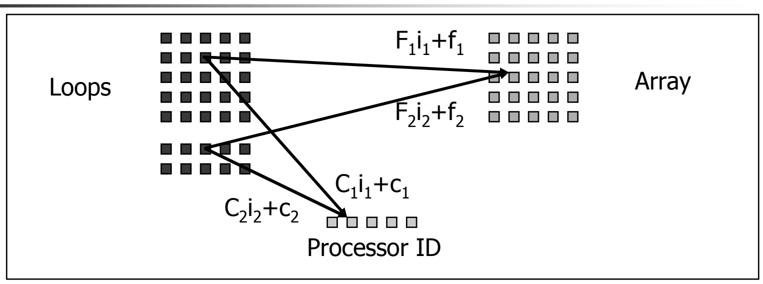
S1: Execute iteration (i, j) on processor i-j.

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

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

```
if (1-n <= p <= n) then
  if [1 <= p) then
     B[p,1] = A[p,0] * B[p,1];                      (S2)
  for i1 = max[1,1+p) to min[n,n-1+p) do
     A[i1,i1-p] = A[i1,i1-p] + B[i1-1,i1-p];    (S1)
     B[i1,i1-p+1] = A[i1,i1-p] * B[i1,i1-p+1]; (S2)
  if (p <= 0) then
     A[n+p,n] = A[n+p,N] + B[n+p-1,n];           (S1)
```


Maximum Parallelism & No Communication



For every pair of data dependent accesses $F_1i_1+f_1$ and $F_2i_2+f_2$

Find C_1, c_1, C_2, c_2:

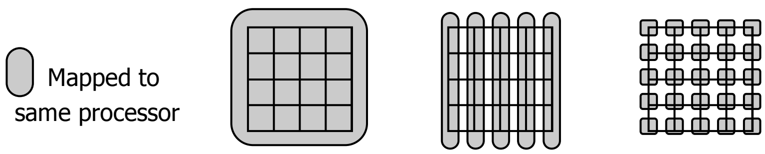
$\forall\, i_1, i_2 \quad F_1 i_1+ f_1 = F_2 i_2+f_2 \rightarrow C_1i_1+c_1 = C_2i_2+c_2$

with the objective of maximizing the rank of C_1, C_2

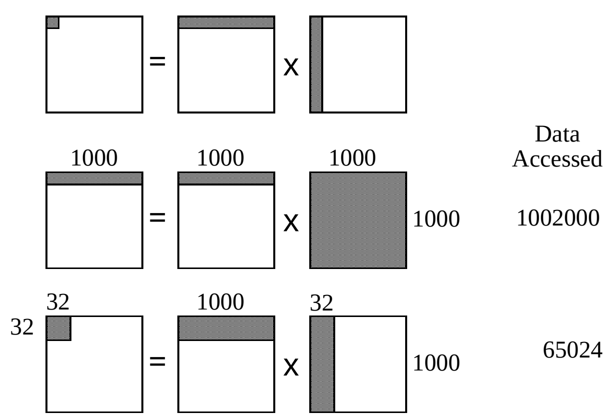
Rank of Partitioning = Degree of Parallelism

Affine Mapping	$\begin{bmatrix} \mathbf{0} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{i} \\ \mathbf{j} \end{bmatrix}$	$\begin{bmatrix} \mathbf{0} & \mathbf{1} \end{bmatrix} \begin{bmatrix} \mathbf{i} \\ \mathbf{j} \end{bmatrix}$	$\begin{bmatrix} \mathbf{1} & \mathbf{0} \\ \mathbf{0} & \mathbf{1} \end{bmatrix} \begin{bmatrix} \mathbf{i} \\ \mathbf{j} \end{bmatrix}$
Rank	0	1	2

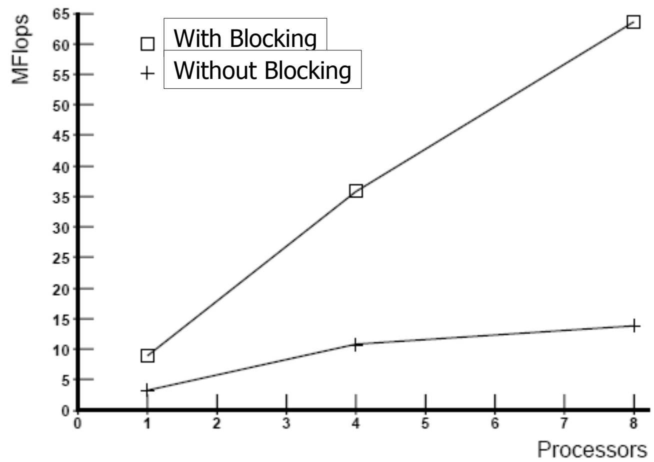
Mapped to same processor



(B) What is blocking? Example: Matrix Multiplication



Experimental Results



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];
    }}}
```

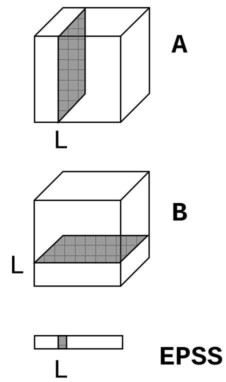
- After

```
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)

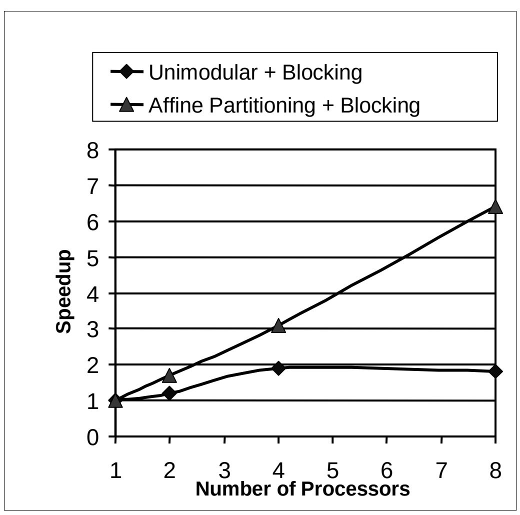
```
    DO 1 J = 0, N
     I0 = MAX ( -M, -J )
       DO 2 I = I0, -1
         DO 3 JJ = I0 - I, -1
           DO 3 L = 0, NMAT
3             A(L,I,J) = A(L,I,J) - A(L,JJ,I+J) * A(L,I+JJ,J)
         DO 2 L = 0, NMAT
2          A(L,I,J) = A(L,I,J) * A(L,0,I+J)
       DO 4 L = 0, NMAT
4        EPSS(L) = EPS * A(L,0,J)
       DO 5 JJ = I0, -1
         DO 5 L = 0, NMAT
5          A(L,0,J) = A(L,0,J) - A(L,JJ,J) ** 2
       DO 1 L = 0, NMAT
1        A(L,0,J) = 1. / SQRT ( ABS (EPSS(L) + A(L,0,J)) )

     DO 6 I = 0, NRHS
       DO 7 K = 0, N
         DO 8 L = 0, NMAT
8          B(I,L,K) = B(I,L,K) * A(L,0,K)
         DO 7 JJ = 1, MIN (M, N-K)
           DO 7 L = 0, NMAT
7             B(I,L,K+JJ) = B(I,L,K+JJ) -  A(L,-JJ,K+JJ) * B(I,L,K)
       DO 6 K = N, 0, -1
         DO 9 L = 0, NMAT
9          B(I,L,K) = B(I,L,K) * A(L,0,K)
         DO 6 JJ = 1, MIN (M, K)
           DO 6 L = 0, NMAT
6             B(I,L,K-JJ) = B(I,L,K-JJ) - A(L,-JJ,K) * B(I,L,K)
```



Chotst: Results with Affine Partitioning + Blocking

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

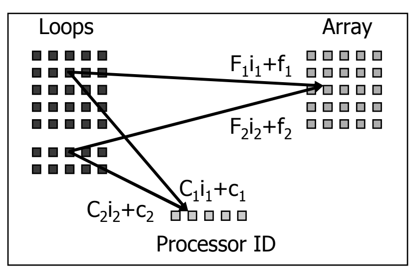


Summary

- 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

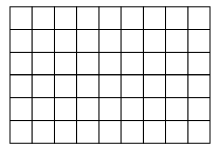
How to Use Linear Algebra

- 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

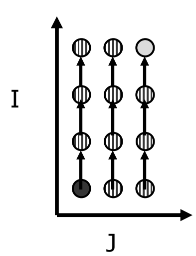
```
FOR i = 0 to 5
  FOR j = i to 7
    …
```



- 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], …

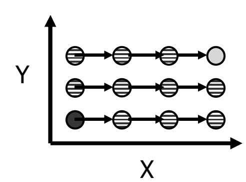
3. Code Generation
Example: Loop Interchange (Loop Permutation)

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



$$\begin{bmatrix} y \\ x \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} i \\ j \end{bmatrix}$$

```
for Y = 1 to 3
  for X = 1 to 4
    Z[Y,X] = Z[Y-1,X]
```

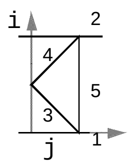


Transforming the code

Step 1: substitute old indices with new.

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

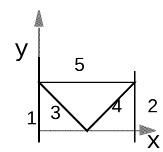
```
FOR y =
  FOR x =
     a[x,y] = ...
```



$$
\begin{aligned}
0 &\le i \\
i &\le 20 \\
10 - i &\le j \\
i - 10 &\le j \\
j &\le 10
\end{aligned}
$$

$$\begin{bmatrix} y \\ x \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} i \\ j \end{bmatrix}$$

$$\begin{bmatrix} i \\ j \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} y \\ x \end{bmatrix}$$



$$
\begin{aligned}
0 &\le x \\
x &\le 20 \\
10 - x &\le y \\
x - 10 &\le y \\
y &\le 10
\end{aligned}
$$

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] = ...
```

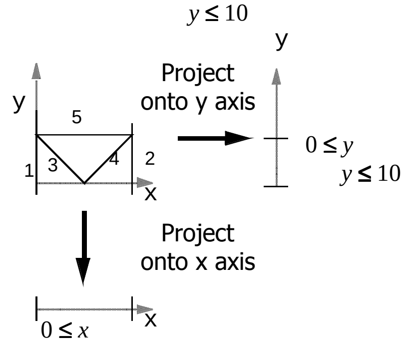
Bounds of x:

$$
\begin{aligned}
0 &\le x \\
x &\le 20 \\
10 - y &\le x \\
x &\le y + 10
\end{aligned}
$$

Bounds of y:

$$
\begin{aligned}
0 &\le y \\
y &\le 10
\end{aligned}
$$

$$
\begin{aligned}
0 &\le x \\
x &\le 20 \\
10 - x &\le y \\
x - 10 &\le y \\
y &\le 10
\end{aligned}
$$



Fourier-Motzkin Elimination

- To eliminate a variable from a set of linear inequalities.
- To eliminate a variable x_1
  - Rewrite all expressions in terms of lower or upper bounds of x_1
  - 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_1:

$$\begin{array}{l} L_1(x_2, \ldots, x_n) \leq x_1 \leq U_1(x_2, \ldots, x_n) \\ L_2(x_2, \ldots, x_n) \leq x_1 \leq U_2(x_2, \ldots, x_n) \end{array} \Rightarrow \begin{array}{l} L_1(x_2, \ldots, x_n) \leq U_1(x_2, \ldots, x_n) \\ L_1(x_2, \ldots, x_n) \leq U_2(x_2, \ldots, x_n) \\ L_2(x_2, \ldots, x_n) \leq U_1(x_2, \ldots, x_n) \\ L_2(x_2, \ldots, x_n) \leq U_2(x_2, \ldots, x_n) \end{array}$$

4. Affine Accesses: Iteration space → Array space

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

Access	Affine Exp	Rank	Nullity	Basis of Null Space
X[i-1]	$\begin{bmatrix}1 & 0\end{bmatrix}\begin{bmatrix}\mathbf{i}\\ \mathbf{j}\end{bmatrix}+\begin{bmatrix}-1\end{bmatrix}$	1	1	$\begin{bmatrix}0\\1\end{bmatrix}$
Y[i,j]	$\begin{bmatrix}1 & 0\\0 & 1\end{bmatrix}\begin{bmatrix}\mathbf{i}\\ \mathbf{j}\end{bmatrix}+\begin{bmatrix}0\\0\end{bmatrix}$	2	0	
Y[j,j+1]	$\begin{bmatrix}0 & 1\\0 & 1\end{bmatrix}\begin{bmatrix}\mathbf{i}\\ \mathbf{j}\end{bmatrix}+\begin{bmatrix}0\\0\end{bmatrix}$	1	1	$\begin{bmatrix}1\\0\end{bmatrix}$
Y[1,2]	$\begin{bmatrix}0 & 0\\0 & 0\end{bmatrix}\begin{bmatrix}\mathbf{i}\\ \mathbf{j}\end{bmatrix}+\begin{bmatrix}1\\2\end{bmatrix}$	0	2	$\begin{bmatrix}0\\1\end{bmatrix},\begin{bmatrix}1\\0\end{bmatrix}$
Z[1,i,2*i+j]	$\begin{bmatrix}0 & 0\\1 & 0\\2 & 1\end{bmatrix}\begin{bmatrix}\mathbf{i}\\ \mathbf{j}\end{bmatrix}+\begin{bmatrix}1\\0\\0\end{bmatrix}$	2	0	

Informal Interpretation for Access Function Fi + f

d: loop depth; n: # of iterations in each loop; a: dimensions of the array

- F is an a x 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 $0(n^r)$ locations accessed.
  $r \le \min(d, a)$
  $O(n^{d-r})$ 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
  nullity(F) + 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 $i+ b_1$, $i + b_2$, i + 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 Fi = Fi' 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.
- nullity = dimension of the null space
  - nullity(F) + rank(F) = d
  - If 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

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

FOR i = 1 TO 100
   FOR j = 1 TO 100
      A[i,j] = B[i,j] + C[i,j]

FOR i = 11 TO 20
   A[i] = A[i-1] + 3

FOR i = 11 TO 20
  A[i] = A[i-20] + 3
```

- 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 → 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 $i_w, i_r,\quad 2 \le i_w, i_r \le 5, \quad i_w - 2 = i_r$
- 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 $i_w, i_v,\quad 2 \le i_w, i_v \le 5, \quad i_w - 2 = i_v - 2, \quad i_w \neq i_v$

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 $B_1 i_1 + b_1 \ge 0$, $B_2 i_2 + b_2 \ge 0$ be the corresponding loop bound constraints,

$\exists$ integers $i_1, i_2 \quad B_1 i_1 + b_1 \ge 0,\ B_2 i_2 + b_2 \ge 0$

$F_1 i_1 + f_1 = F_2 i_2 + f_2$

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 $x \leq 0$ and $x \geq 1$ 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