COMP 515: Advanced Compilation for Vector and Parallel Processors

Vivek Sarkar
Department of Computer Science
Rice University
vsarkar@rice.edu

Announcement

- Take-home Midterm exam will be given at end of Feb 24th midterm review class will be due in class on March 10th (2 weeks later)

Acknowledgments

- Slides from previous offerings of COMP 515 by Prof. Ken Kennedy
  - http://www.cs.rice.edu/~ken/comp515/

Loop Fusion

- Loop distribution was a method for separating parallel parts of a loop.
- Our solution attempted to find the maximal loop distribution.
- The maximal distribution often finds parallelizable components to small for efficient parallelizing.
- Two obvious solutions:
  - Strip mine large loops to create larger granularity.
  - Perform maximal distribution, and fuse together parallelizable loops.

Fusion Safety

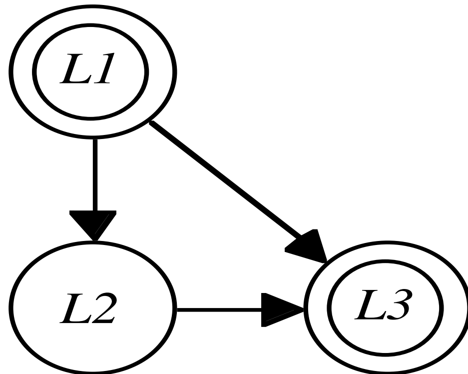
Definition: A loop-independent dependence between statements S1 and S2 in loops L1 and L2 respectively is *fusion-preventing* if fusing L1 and L2 causes the dependence to be carried by the combined loop in the opposite direction.

```
    DO I = 1,N
S1      A(I) = B(I)+C
    ENDDO
    DO I = 1,N
S2      D(I) = A(I+1)+E
    ENDDO
```

```
    DO I = 1,N
S1      A(I) = B(I)+C
S2      D(I) = A(I+1)+E
    ENDDO
```

Fusion Safety

- **We shouldn't fuse loops if the fusing will violate ordering of the dependence graph.**
- ***Ordering Constraint:*** **Two loops can't be validly fused if there exists a path of loop-independent dependencies between them containing a loop or statement not being fused with them i.e., if fusion will result in a cycle in the resulting loop-independent dependences**



Fusing L1 with L3 violates the ordering constraint. {L1,L3} must occur both before and after the node L2.

Fusion Profitability

Parallel loops should generally not be merged with sequential loops.

Definition: An edge between two statements in loops L1 and L2 respectively is said to be *parallelism-inhibiting* if after merging L1 and L2, the dependence is carried by the combined loop.

```
      DO I = 1,N
S1        A(I+1) = B(I) + C
      ENDDO
      DO I = 1,N
S2        D(I) = A(I) + E
      ENDDO
```

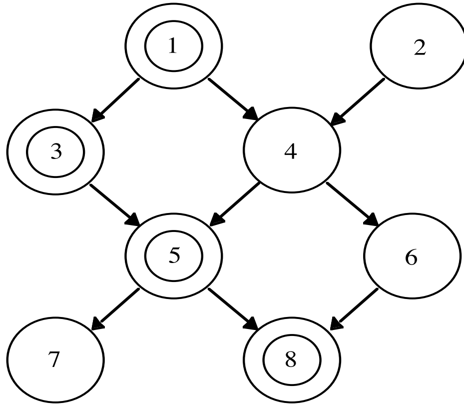
```
      DO I = 1,N
S1        A(I+1) = B(I) + C
S2        D(I) = A(I) + E
      ENDDO
```

Typed Fusion

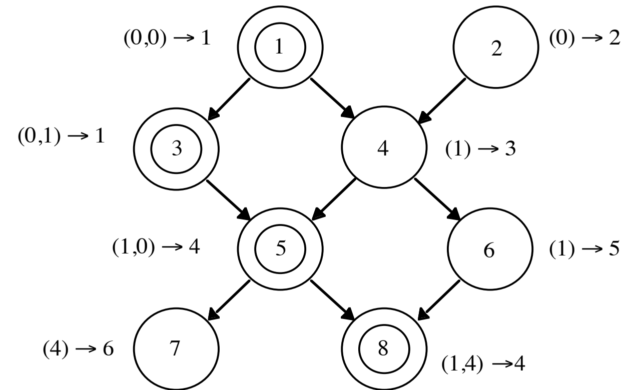
- We start off by classifying loops into two types: parallel and sequential.
- We next gather together all edges that inhibit efficient fusion, and call them bad edges.
- Given a loop dependency graph (V,E), we want to obtain a graph (V',E') by merging vertices of V subject to the following constraints:
  - Bad Edge Constraint: vertices joined by a bad edge aren't fused.
  - Ordering Constraint: vertices joined by path containing non-parallel vertex aren't fused

Typed Fusion Example

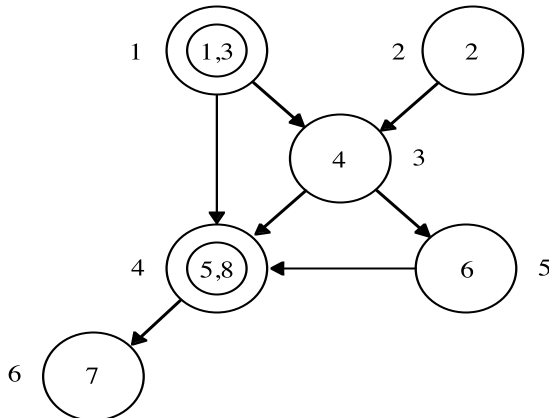
Original loop graph



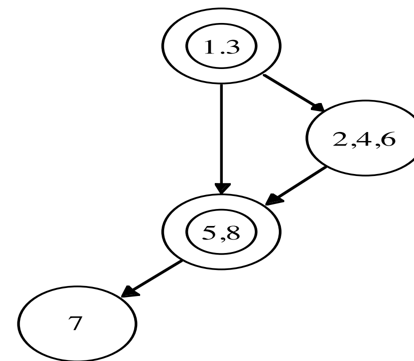
Graph annotated (maxBadPrev,p) → num



After fusing parallel loops



After fusing sequential loops



Thus far …

- Single loop methods
  - Privatization
  - Loop distribution
  - Loop fusion
  - Alignment
  - Code replication
- Today, perfect and imperfect loop nest methods

Loop Interchange

- Parallelization: move dependence-free loops to outermost level
- Theorem 6.3
  - In a perfect nest of loops, a particular loop can be parallelized at the outermost level if and only if the column of the direction matrix for that nest contains only '=' entries

Motivation for Loop Interchange

```
DO I = 1, N
    DO J = 1, N
    A(I+1, J) = A(I, J) + B(I, J)        (<, =)
    ENDDO
ENDDO
```

- Parallelizing the J loop is OK for vectorization
- But inefficient for parallelization (N barriers)

Loop Interchange

```
PARALLEL DO J = 1, N
    DO I = 1, N
        A(I+1, J) = A(I, J) + B(I, J)        (=, <)
    ENDDO
END PARALLEL DO
```

Loop Interchange

```
while L is not empty
  while there exist columns in M with all "="
    success := true;
    l:= loop with all "=" column;
    remove l from L;
    parallelize l at outer level;
    eliminate l's column from M;
  end;
  if L is not empty
    select_loop_and_interchange(L);
    l:= outermost loop; remove l from L; sequentialize l;
    remove column corresponding to l from M;
    remove all rows corresponding to dependences carried by l from M;
```

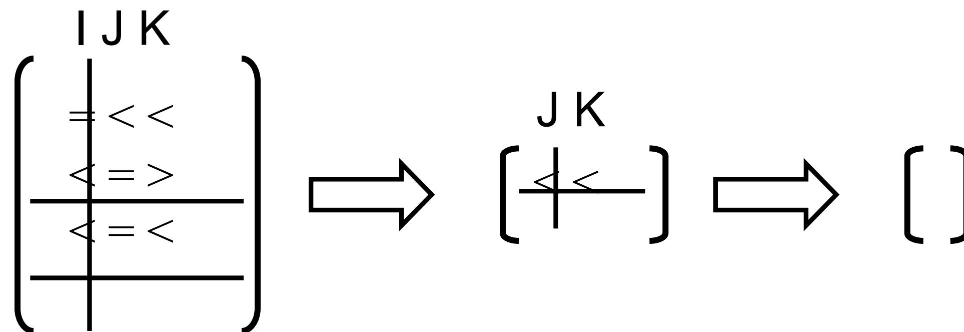
Loop Selection

```
DO I = 2, N+1
    DO J = 2, M+1
        DO K = 1, L
            A(I, J, K+1) = A(I, J-1, K) + A(I-1, J, K+2) + A(I-1, J, K)
        ENDDO
     ENDDO
ENDDO
```

$$
\begin{array}{c} I\ J\ K \end{array}
\begin{pmatrix} = & < & < \\ < & = & > \\ < & = & < \end{pmatrix}
$$

Loop Selection

```
DO I = 2, N+1
    DO J = 2, M+1
        PARALLEL DO K = 1, L
            A(I, J, K+1) = A(I, J-1, K) + A(I-1, J, K+2) + A(I-1, J, K)
        ENDDO
    ENDDO
ENDDO
```



Loop Selection

- Is it possible to derive a selection heuristic that provides optimal code?
  - Probably not possible, NP-complete problem
- Assume simple approach of selecting the loop with the most '<' directions to eliminate the max number of rows from the direction matrix
  - Applying to this matrix will fail

I	J	K	L
<	<	=	=
<	=	<	=
<	=	=	<
=	<	=	=
=	=	<	=
=	=	=	<

Loop Selection

- Is it possible to derive a selection heuristic that provides optimal code?
  - —Probably not possible, NP-hard problem
- Assume simple approach of selecting the loop with the most ‘<‘ directions to eliminate the max number of rows from the direction matrix
  - —Applying to this matrix will fail

J	K	L	I
<	=	=	<
=	<	=	<
=	=	<	<
<	=	=	=
=	<	=	=
=	=	<	=

Loop Selection

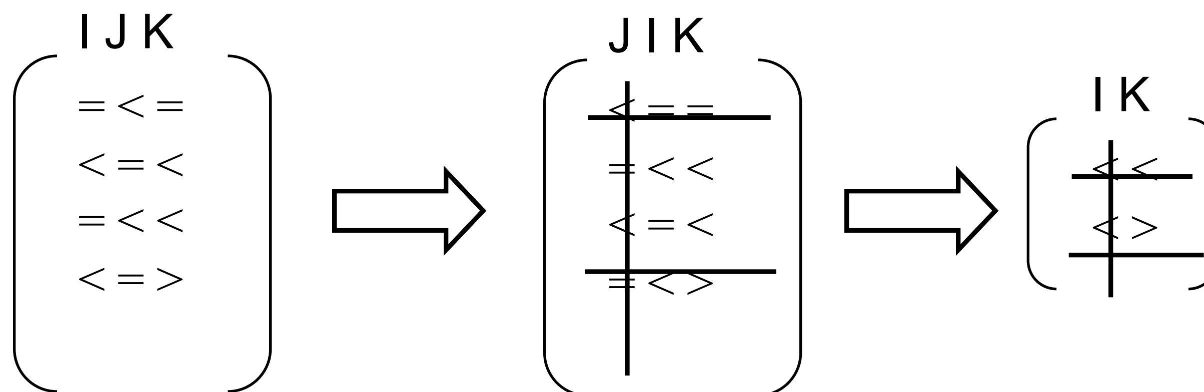
- Favor the selection of loops that must be sequentialized before parallelism can be uncovered
- If there exists a loop that can legally be moved to the outermost position and there is a dependence for which that loop has the only ‘<‘ direction, sequentialize that loop
- All such loops will need to be sequentialized at some point in the process

J	K	L	I
<	=	=	<
=	<	=	<
=	=	<	<
<	=	=	=
=	<	=	=
=	=	<	=

Loop Selection

- Example of principles involved in heuristic loop selection

```
DO J = 2, M
  DO I = 2, N
          PARALLEL DO K = 2, L
          A(I, J, K) = A(I, J-1, K) + A(I-1, J, K-1) +
        A(I, J+1, K+1) + A(I-1, J, K+1)
          ENDDO
  ENDDO
ENDDO
```



Loop Reversal

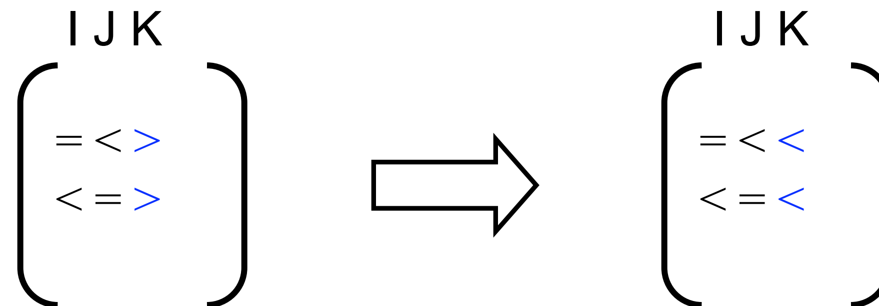
```
DO I = 2, N+1
    DO J = 2, M+1
        DO K = 1, L
            A(I, J, K) = A(I, J-1, K+1) + A(I-1, J, K+1)
        ENDDO
    ENDDO
ENDDO
```

I J K

$$\begin{pmatrix} = & < & > \\ < & = & > \end{pmatrix}$$

Loop Reversal

```
DO I = 2, N+1
    DO J = 2, M+1
        DO K = L, 1, -1
            A(I, J, K) = A(I, J-1, K+1) + A(I-1, J, K+1)
        ENDDO
    ENDDO
ENDDO
```



After Loop Reversal & Interchange

```
DO K = L, 1, -1
        PARALLEL DO I = 2, N+1
            PARALLEL DO J = 2, M+1
                A(I, J, K) = A(I, J-1, K+1) + A(I-1, J, K+1)
            END PARALLEL DO
        END PARALLEL DO
ENDDO
```

K I J

$$\begin{pmatrix} < = < \\ < < = \end{pmatrix}$$

- Increase the range of options available for loop selection heuristics

Loop Skewing

```
DO I = 2, N+1
    DO J = 2, M+1
        DO K = 1, L
            A(I, J, K) = A(I, J-1, K) + A(I-1, J, K)
            B(I, J, K+1) = B(I, J, K) + A(I, J, K)
        ENDDO
    ENDDO
ENDDO
```

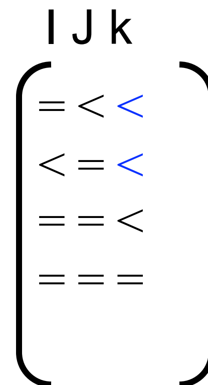
$$
\begin{array}{ccc} I & J & K \end{array}
$$

$$
\begin{pmatrix}
= & < & = \\
< & = & = \\
= & = & < \\
= & = & =
\end{pmatrix}
$$

Loop Skewing

- **Skewed using k = K + I + J yield:**

```
DO I = 2, N+1
        DO J = 2, M+1
            DO k = I+J+1, I+J+L
                A(I, J, k-I-J) = A(I, J-1, k-I-J) + A(I-1, J, k-I-J)
                B(I, J, k-I-J+1) = B(I, J, k-I-J) + A(I, J, k-I-J)
            ENDDO
        ENDDO
ENDDO
```



Loop Skewing

```
DO k = 5, N+M+1
    PARALLEL DO I = MAX(2, k-M-L-1), MIN(N+1, k-L-2)
        PARALLEL DO J = MAX(2, k-I-L), MIN(M+1, k-I-1)
            A(I, J, k-I-J) = A(I, J-1, k-I-J) + A(I-1, J, k-I-J)
            B(I, J, k-I-J+1) = B(I, J, k-I-J) + A(I, J, k-I-J)
        ENDDO
    ENDDO
ENDDO
```

$$\begin{array}{c} k\ I\ J \\ \left(\begin{array}{ccc} < & = & < \\ < & < & = \\ < & = & = \\ = & = & = \end{array}\right) \end{array}$$

Loop Skewing

- Transforms skewed loop into one that can be interchanged to the outermost position without changing the meaning of the program
- Can be used to transform the skewed loop in such a way that, after outward interchange, it will carry all dependences formerly carried by the loop with respect to which it is skewed

$$
\begin{array}{ccc} k & I & J \end{array}
$$

$$
\begin{pmatrix} < & = & < \\ < & < & = \\ < & = & = \\ = & = & = \end{pmatrix}
$$

Loop Skewing

- **Selection Heuristics**
  1. Parallelize as many loops as possible
  2. Sequentialize at most one loop to find parallelism in the current outermost loop
  3. If 1 and 2 fails, try skewing
  4. If 3 fails, sequentialize the loop that can be moved to the outermost position and cover the most other loops

Unimodular Transformations

- A transformation represented by a matrix T is unimodular if
  - T is square
  - All the elements of T are integral and
  - The absolute value of the determinant of T is 1
- Any composition of unimodular transformations is unimodular
- Loop interchange, skewing and reversal represent unimodular transformations of distance vectors
- Theory used in goal-directed strategies

Profitability-Based Methods

- **Motivation**
  - Need to be able to estimate performance of produced code
  - Ex. overhead of synch. may outweigh benefits from parallelism
- Pick the best of all possible permutations and parallelizations
  - Impractical
    - Total number of alternatives is exponential in the number of loops in a nest
- Consider only subset of the possible code arrangements, based on properties of a cost function

Profitability-Based Methods

- Static performance estimation function
  - No need to be accurate
  - Good at selecting the better of two alternatives
- Key considerations
  - Cost of memory references
  - Sufficiency of granularity

Profitability-Based Methods

- Subdivide all the references in the loop body into reference groups
- Determine whether subsequent accesses to the same reference are
  - Loop invariant
    - Cost = 1
  - Unit stride
    - Cost = number of iterations / cache line size
  - Non-unit stride
    - Cost = number of iterations

Profitability-Based Methods

```
DO I = 1, N
  DO J = 1, N
    DO K = 1, N
      C(I, J) = C(I, J) + A(I, K) * B(K, J)
    ENDDO
  ENDDO
ENDDO
```

- K is innermost loop: C = 1, A = N, B = N/L => $N^3(1+1/L)+N^2$
- J is innermost loop: C = N, A = 1, B = N => $2N^3+N^2$
- I is innermost loop: C = N/L, A = N/L, B = 1 => $2N^3/L+N^2$

- Reorder loop from innermost to outermost by increasing loop cost
- Can't always have desired loop order

Erlebacher

```
DO J = 1, JMAXD
   DO I = 1, IMAXD
      F(I, J, 1) = F(I, J, 1) * B(1)
DO K = 2, N-1
   DO J = 1, JMAXD
      DO I = 1, IMAXD
         F(I, J, K) = (F(I, J, K) – A(K) * F(I, J, K-1)) * B(K)
DO J = 1, JMAXD
   DO I = 1, IMAXD
      TOT(I, J) = 0.0
DO J = 1, JMAXD
   DO I = 1, IMAXD
      TOT(I, J) = TOT(I, J) + D(1) * F(I, J, 1)
DO K = 2, N-1
   DO J = 1, JMAXD
      DO I = 1, IMAXD
         TOT(I, J) = TOT(I, J) + D(K) * F(I, J, K)
```

Erlebacher

```
PARALLEL DO J = 1, JMAXD
    DO I = 1, IMAXD
        F(I, J, 1) = F(I, J, 1) * B(1)
    DO K = 2, N - 1
        DO I = 1, IMAXD
            F(I, J, K) = ( F(I, J, K) - A(K) * F(I, J, K-1)) * B(K)
    DO I = 1, IMAXD
        TOT(I, J) = 0.0
    DO I = 1, IMAXD
        TOT(I, J) = TOT(I, J) + D(1) * F(I, J, 1)
    DO K = 2, N-1
        DO I = 1, IMAXD
            TOT(I, J) = TOT(I, J) + D(K) * F(I, J, K)
```

Erlebacher

```
PARALLEL DO J = 1, JMAXD
    DO I = 1, IMAXD
        F(I, J, 1) = F(I, J, 1) * B(1)
        TOT(I, J) = 0.0
        TOT(I, J) = TOT(I, J) + D(1) * F(I, J, 1)
    ENDDO

    DO K = 2, N-1
        DO I = 1, IMAXD
            F(I, J, K) = ( F(I, J, K) - A(K) * F(I, J, K-1)) * B(K)
            TOT(I, J) = TOT(I, J) + D(K) * F(I, J, K)
        ENDDO
    ENDDO
ENDDO
```

Strip Mining

- **Converts available parallelism into a form more suitable for the hardware**

```
DO I = 1, N
    A(I) = A(I) + B(I)
ENDDO

k = CEIL (N / P)
PARALLEL DO I = 1, N, k
        DO i = I, MIN(I + k-1, N)
         A(i) = A(i) + B(i)
     ENDDO
END PARALLEL DO
```

Strip Mining

```
DO I = 1, N
        DO J = 2, I
      A(I, J) = A(I, J-1) + B(I)
   ENDDO
ENDDO
```

- Choose smaller unit size to allow more balanced load distribution

Homework

- Reading list for next class
  - Chapter 7, Handling Control Flow
- Homework assignment for discussion in next class
  - Exercise 6.3