COMP 515: Advanced Compilation for Vector and Parallel Processors

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

https://wiki.rice.edu/confluence/display/PARPROG/COMP515

COMP 515 Lecture 18 3 November, 2011

Acknowledgments

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

Managing Cache

Allen and Kennedy, Chapter 9

Loop Interchange

- Which loop should be innermost ?
- Strives to reduce distances between memory accesses to increase locality
- Attaches cost function to the loop and computes for best loop ordering

Cost Assignment

- Consider cost analysis for an innermost loop with N iterations, for arrays with element size = s, and a cache with line size = l
- Cost is 1 for references that do not depend on loop induction variables
- Cost is N for references based on induction variables over a non-contiguous space
- Cost is Ns/l for induction variables based references over contiguous space
- Multiply the cost by the loop trip count if the reference varies with the loop index

Loop Reordering

- Once the cost is established, reorder the loop from cheapest innermost loop to high cost outermost loop

Loop Blocking (Tiling)

-
  ```
  DO J = 1, M
     DO I = 1, N
        D(I) = D(I) + B(I,J)
     ENDDO
  ENDDO
  ```

NM/b misses for each of arrays B and D

==> total of 2NM/b misses

b = block (line) size in words (elements)

Assume that N is large enough for elements of D to overflow cache

Blocking loop I

- After strip-mine-and-interchange

```
DO II = 1, N, S
  DO J = 1, M
    DO I = II, MIN(II+S-1, N)
      D(I) = D(I) + B(I,J)
    ENDDO
  ENDDO
ENDDO
```

$NM/b + N/b = (1 + 1/M)\, NM / b$ misses

Assume that S is >= b and is also small enough to allow S elements of D to be held in cache for all iterations of the J loop

Blocking Loop J

```
• DO J = 1, M, T
     DO I = 1, N
        DO jj = J, MIN(J+T-1, M)
           D(I) = D(I) + B(I, jj)
        ENDDO
     ENDDO
  ENDDO
```

NM/b misses for array B (if T is small enough)

(N/b)*(M/T) misses for array D

==> Total of (1 + 1/T) NM/b misses

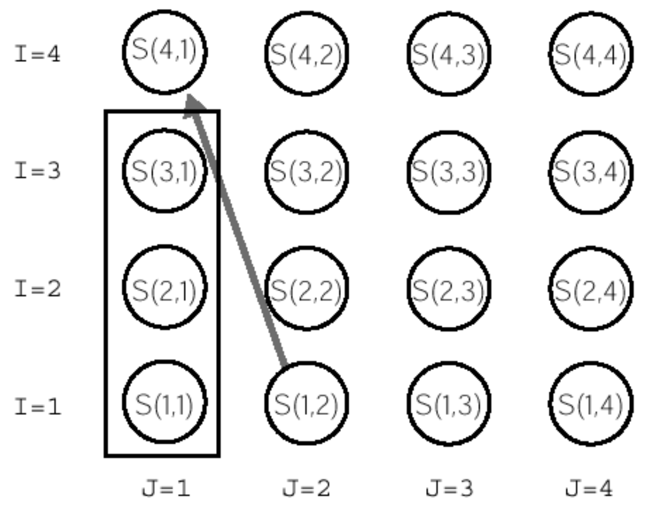
Legality of Blocking

- **Strip mining is always legal**
- **Loop interchange is not always legal**

```
procedure StripMineAndInterchange ($L, m, k, o, S$)
    // $L = \{L_1, L_2, ..., L_m\}$ is the loop nest to be transformed
    // $L_k$ is the loop to be strip mined
    // $L_o$ is the outer loop which is to be just inside the by-strip loop
    //      after interchange
    // $S$ is the variable to use as strip size; it's value must be positive
    let the header of $L_k$ be
        DO I = L, N, D;
    split the loop into two loops, a by-strip loop:
            DO I = L, N, S*D
        and a within-strip loop:
            DO i = I, MAX(I+S*D-D,N), D
        around the loop body;
    interchange the by-strip loop to the position just outside of $L_o$;
end StripMineAndInterchange
```

Legality of Blocking

- Every direction vector for a dependence carried by any of the loops $L_0 \ldots L_{k+1}$ has either an "=" or a "<" in the kth position
- Conservative testing



Profitability of Blocking

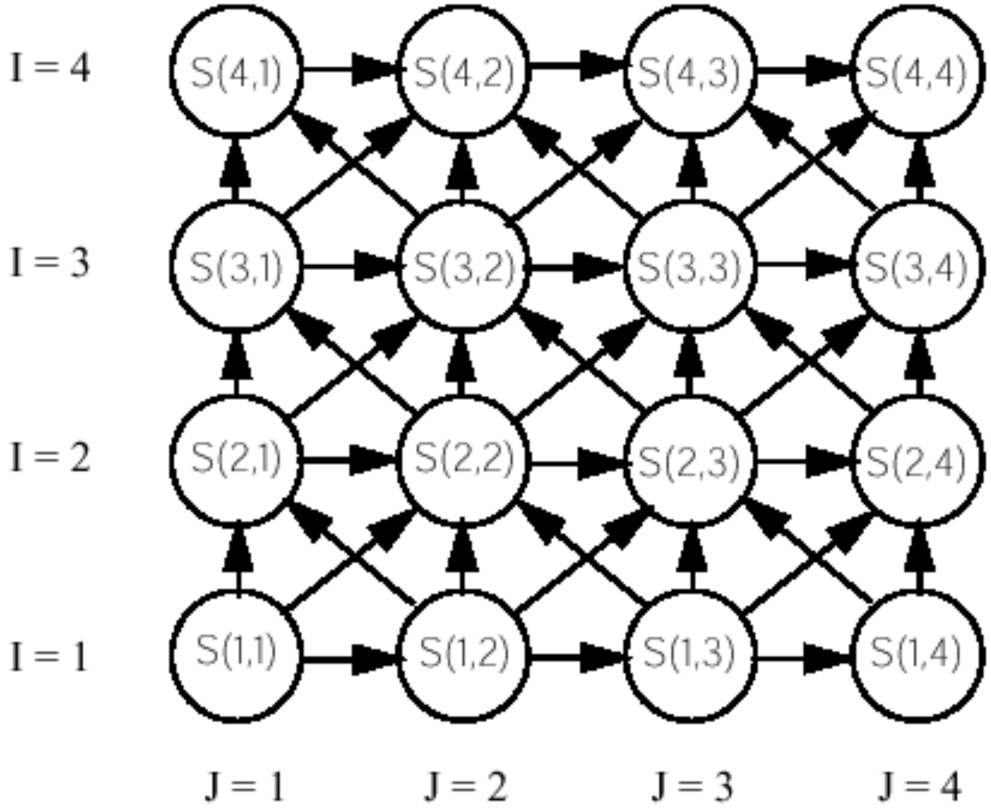
- Profitable if there is reuse between iterations of a loop that is not the innermost loop
- Reuse occurs when:
  - There's a small-threshold dependence of any type, including input, carried by the loop (temporal reuse), or
  - The loop index appears, with small stride, in the contiguous dimension of a multidimensional array and in no other dimension (spatial reuse)

Blocking with Skewing

- For cases where interchange is not possible

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

Blocking with Skewing



Blocking with Skewing

- *After Skewing*

```
DO I = 1, N
   DO j = I, M+I-1
      A(j-I+2) = (A(j-I+1) + A(j-I+2))/2
   ENDDO
ENDDO
```

Blocking with Skewing

- After strip-mine

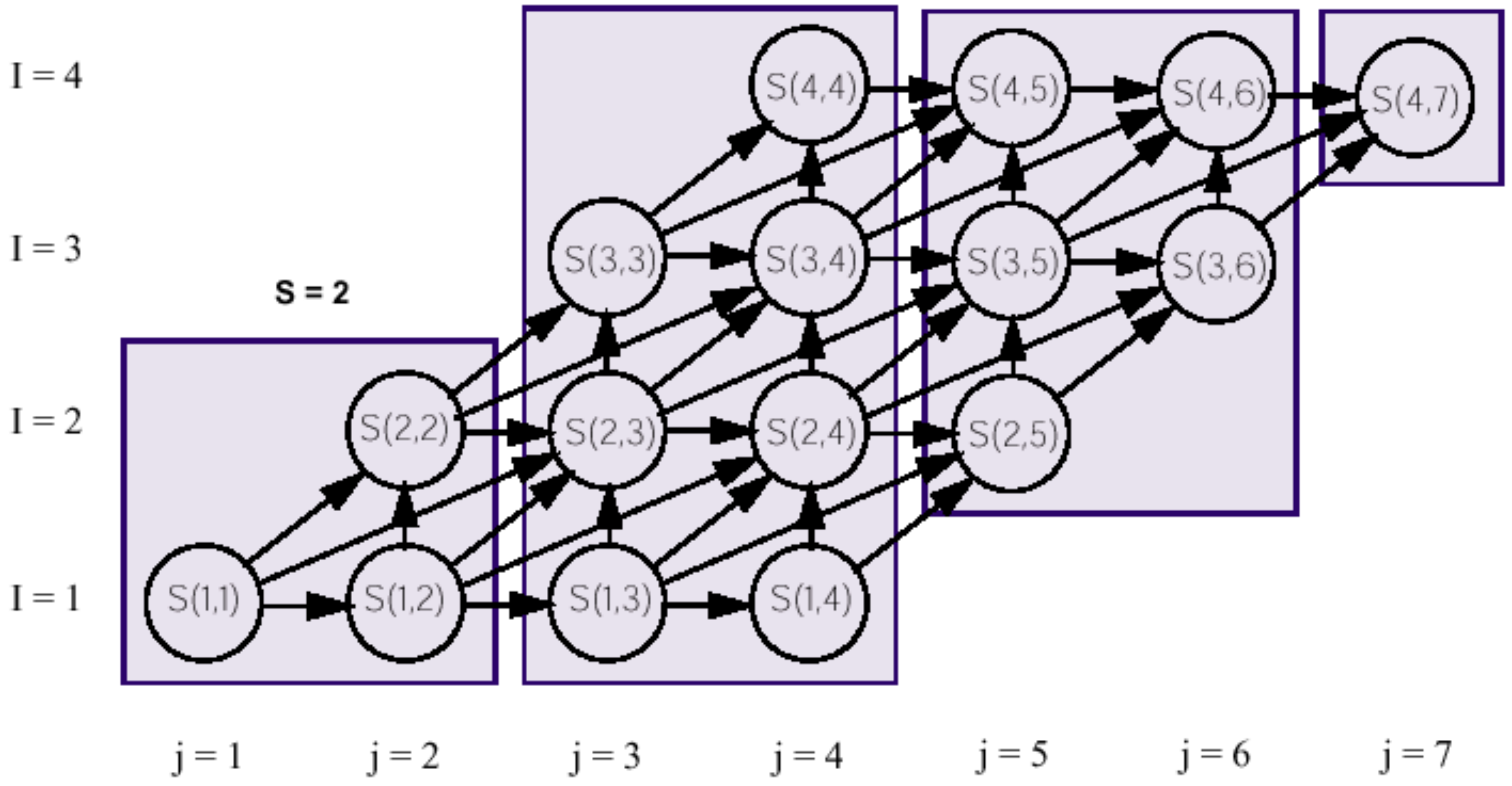
```
DO I = 1, N
   DO j = I, M+I-1, S
      DO jj = j, MAX(j+S-1, M+I-1)
         A(jj-I+2) = (A(jj-I+1) + A(jj-I+2))/2
      ENDDO
   ENDDO
ENDDO
```

Blocking with Skewing

- Loop interchange

```
DO j = 1, M+N-1, S
   DO I = MAX(1, j-M+1), MIN(j, N)
      DO jj = j, MAX(j+S-1, M+I-1)
         A(jj-I+2) = (A(jj-I+1) + A(jj-I+2))/2
      ENDDO
   ENDDO
ENDDO
```

Blocking with Skewing



Triangular Cache Blocking

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

Triangular Cache Blocking

- Applying strip mining
- DO I = 2, N, K

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

Triangular Cache Blocking

- Applying triangular loop interchange

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

Software Prefetching

- Program reorganization limitations
  - Can't eliminate first time misses
  - Can't eliminate misses unknown at compile time
- Prefetching disadvantages
  - May result in premature eviction of useful data in cache
  - May bring in data evicted before use or never used

Software Prefetching Algorithm

- Critical steps in an effective prefetching algorithm
  - Accurate determination of the references requiring prefetching
  - Insertion of prefetching instructions far enough in advance

Prefetch Analysis

- Identify where misses may happen
- Make use of dependence analysis strategy
  - Build on generator-based partitioning idea from scalar replacement
- First, ensure that every edge that is unlikely to correspond to reuse is eliminated from the graph
- Assume that the loop nest has been strip-mined and interchanged to increase locality
- Traverses the loop and mark 'ineffective' for loops without reuse

Prefetch Analysis

- Estimate amount of data used by each iteration, and determine the overflow iteration, which is one more than the number of iterations whose data can be accommodated in cache at the same time
- Any dependence with a threshold equal to or greater than the overflow is considered ineffective for reuse

Prefetch Analysis

- Identify where prefetching is required
- Two cases:
  - If the group generator is not contained in a dependence cycle, a miss is expected on each iteration unless references to the generator on subsequent iterations display temporal locality
  - If the group generator is contained in a dependence cycle, then a miss is expected only on the first few iterations of the carrying loop, depending on the distance of the carrying dependence. In this case, a prefetch to the reference can be placed before the loop carrying the dependence

Insertion for Acyclic Partitions

- Assuming single name partition with single generator
- If there is no spatial reuse of the reference in the loop then insert a prefetch before each reference to the generator
- If the references have spatial locality within the loop, determine i_0 of the first iteration after the initial iteration that causes a miss on the access to the generator and the iteration l between misses in the cache.

Insertion for Acyclic Partitions

1. Partition the loop into two parts;

   initial subloop running from 1 to i_o-1 and

   remainder running from i_o to the end

2. Strip mine the second loop to have subloops of length l
3. Insert all prefetches needed to avoid misses in the initial subloop prior to the loop
4. Eliminate any very short loops by unrolling

Insertion for Acyclic Partitions

-
  ```
  DO I = 1, M
     A(I, J) = A(I, J) + A(I-1, J)
  ENDDO
  ```

Assuming cache line of length four, then $i_o = 5$

and $l = 4$

Insertion for Acyclic Partitions

-
  ```
  DO I = 1, 3
     A(I, J) = A(I, J) + A(I-1, J)
  ENDDO
  DO I = 4, M, 4
     IU = MIN(M, I+4)
     DO ii = I, IU
        A(I, J) = A(I, J) + A(I-1, J)
     ENDDO
  ENDDO
  ```

Insertion for Acyclic Partitions

```
prefetch(A(0,J))
DO I = 1, 3
   A(I, J) = A(I, J) + A(I-1, J)
ENDDO
DO I = 4, M, 4
   IU = MIN(M, I+3)
   prefetch(A(IU, J))
   DO ii = I, IU
      A(ii, J) = A(ii, J) + A(ii-1, J)
   ENDDO
ENDDO
```

Insertion for Cyclic Name Partitions

- Insert prefetch instructions prior to the loop carrying the cycle
- In the case where loop carrying the dependence is an outer loop, the prefetch can be vectorized
  - Place prefetch loop nest outside the loop carrying the backward dependence of a cyclic name partition
  - Rearrange the loop nest so that the loop iterating sequentially over cache lines is innermost
  - Split the innermost loop into two –
    - Preloop to the first iteration of the innermost loop contaning a generator reference beginning on a new cache line and
    - Main loop that begins with the iteration containing the new cache reference.
  - Replace the preloop by a prefetch of the first generator reference. Set the stride of the main loop to the interval between new cache references.

Insertion for Cyclic Name Partitions

```
• DO J = 1, M
    DO I = 2, 33
       A(I, J) = A(I, J) * B(I)
    ENDDO
  ENDDO
```

Insertion for Cyclic Name Partitions

```
prefetch(B(2))
DO I = 5, 33, 4
   prefetch(B(I))
ENDDO
DO J = 1, M
   prefetch(A(2,J))
   DO I = 2, 4
      A(I, J) = A(I, J) * B(I)
   ENDDO
```

Insertion for Cyclic Name Partitions (contd)

```
    DO I = 5, 33, 4
       prefetch(A(I, J))
       A(I, J) = A(I, J) * B(I)
       A(I+1, J) = A(I+1, J) * B(I+1)
       A(I+2, J) = A(I+2, J) * B(I+2)
       A(I+3, J) = A(I+3, J) * B(I+3)
    ENDDO
    prefetch(A(33, J))
    A(33, J) = A(33, J) * B(33)
ENDDO
```

Prefetching Irregular Accesses

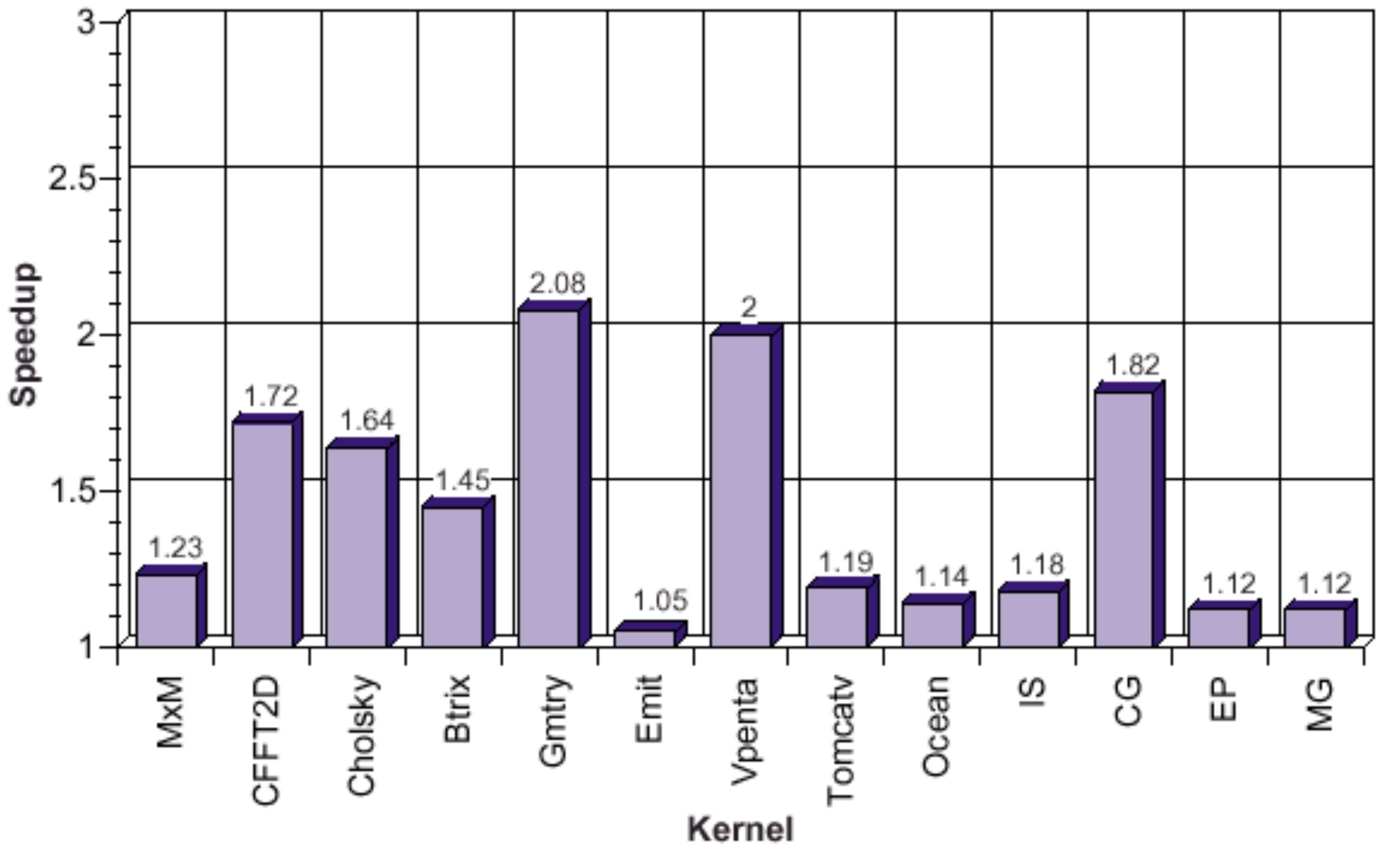
-
  ```
  DO J = 1, M
     DO I = 2, 33
        A(I, J) = A(I, J) * B(IX(I), J)
     ENDDO
  ENDDO
  ```

Prefetching Irregular Accesses

- prefetch(IX(2))

```
DO I = 5, 33, 4
   prefetch(IX(I))
ENDDO
.
.
.
```

Effectiveness of Prefetching



Summary

- Two different kind of reuse
  - Temporal reuse
  - Spatial reuse
- Strategies to increase the two reuse
  - Loop Interchange
  - Cache Blocking
- Software prefetching