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 17 5 November, 2015

Managing Cache

Allen and Kennedy, Chapter 9

Introduction

- **Register**
  - One word per register (typically, but there may be exceptions e.g., SIMD registers)
  - Temporal reuse
  - Direct store
  - Eviction (spills) managed by software

- **Cache**
  - Multiple words in a cache line, multiple lines in an associative set, multiple sets in a cache
  - Temporal and Spatial reuse
  - Load before store
  - Eviction managed by hardware (software can also help)

Spatial Reuse

- Permits high reuse when accessing closely located data
- ```
  DO I = 1, M
      DO J = 1, N
          A(I, J) = A(I, J) + B(I, J)
      ENDDO
  ENDDO
  ```

  No reuse/locality for Fortran's column-major layout
  ```

Spatial Reuse (after loop interchange)

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

Iterates over columns instead

Temporal Reuse

- Reuse limited by cache size, LRU replacement strategy
- ```
  DO I = 1, M
      DO J = 1, N
          A(I) = A(I) + B(J)
      ENDDO
  ENDDO
  ```
  ```

Temporal Reuse

- **Strip mining + Interchange (or Tiling) can improve temporal reuse when tile size S is chosen so that inner loops can fit in cache**

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

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 bytes, and a cache with line size = L bytes
- 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 N*S/L for induction variables based references over contiguous space

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
```

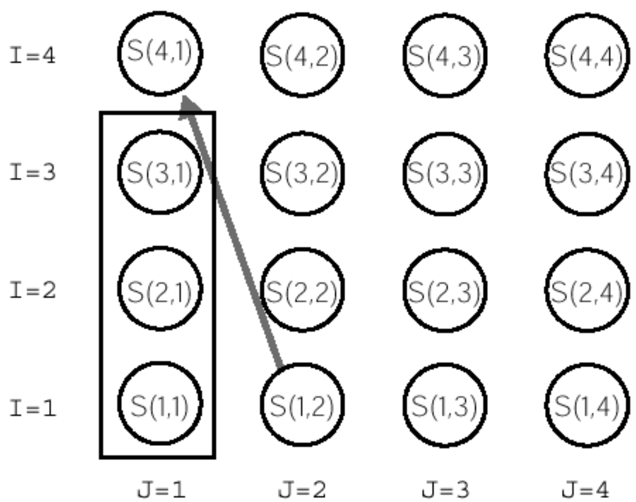
around the loop body;

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)**

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 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
  ```
  ```

Summary

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