

Reducing Misses by Compiler Optimizations

- Instructions
 - Reorder procedures in memory so as to reduce misses
 - Profiling to look at conflicts
 - McFarling [1989] reduced caches misses by 75% on 8KB direct mapped cache with 4 byte blocks
- Data
 - *Merging Arrays*: improve spatial locality by single array of compound elements vs. 2 arrays
 - *Loop Interchange*: change nesting of loops to access data in order stored in memory
 - *Loop Fusion*: Combine 2 independent loops that have same looping and some variables overlap
 - *Blocking*: Improve temporal locality by accessing "blocks" of data repeatedly vs. going down whole columns or rows

Merging Arrays Example

```

/* Before */
int val[SIZE];
int key[SIZE];

/* After */
struct merge {
    int val;
    int key;
};
struct merge merged_array[SIZE];
    
```

Reducing conflicts between val & key (spatial locality)

Array Storage (Column order)

- Store first column, then second column, then next column, ...
- Example
 - [6x6] array A
 - each element 1 word (32 bits)
 - Block size = 4 words
 - For $i = 1$ to 6; $X += a[1,i]$; How many blocks are loaded into cache?
 - Misses: Compulsory? Conflict? Capacity?

Memory Map

A(1,1)	A(2,1)	A(3,1)	A(4,1)	A(5,1)	A(6,1)	A(1,2)	A(2,2)	A(3,2)	A(4,2)	A(5,2)
A(6,2)	A(1,3)	A(2,3)	A(3,3)	A(4,3)	A(5,3)	A(6,3)	A(1,4)	A(2,4)	A(3,4)	A(4,4)
A(5,4)	A(6,4)	A(1,5)	A(2,5)	A(3,5)	A(4,5)	A(5,5)	A(6,5)	A(1,6)	A(2,6)	A(3,6)
A(4,6)	A(5,6)	A(6,6)								

Array Storage (Row order)

- Store first row, then second row, then next row, ...
- Example
 - [6x6] array A
 - each element 1 word (32 bits)
 - Block size = 4 words
 - For $i = 1$ to 6; $X += a[1,i]$; How many blocks are loaded into cache?
 - Misses: Compulsory? Conflict? Capacity?

Memory Map

A(1,1)	A(1,2)	A(1,3)	A(1,4)	A(1,5)	A(1,6)	A(2,1)	A(2,2)	A(2,3)	A(2,4)	A(2,5)
A(2,6)	A(3,1)	A(3,2)	A(3,3)	A(3,4)	A(3,5)	A(3,6)	A(4,1)	A(4,2)	A(4,3)	A(4,4)
A(4,5)	A(4,6)	A(5,1)	A(5,2)	A(5,3)	A(5,4)	A(5,5)	A(5,6)	A(6,1)	A(6,2)	A(6,3)
A(6,4)	A(6,5)	A(6,6)								

Loop Interchange Example

```

/* Before */
for (j = 0; j < 100; j = j+1)
    for (i = 0; i < 5000; i = i+1)
        x[i][j] = 2 * x[i][j];

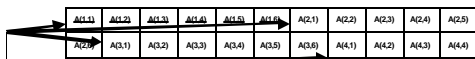
/* After */
for (i = 0; i < 5000; i = i+1)
    for (j = 0; j < 100; j = j+1)
        x[i][j] = 2 * x[i][j];
    
```

Fortran: column major
-row indices change

C: row major
-column indices change

Which is this?

Sequential accesses (spatial locality)
-instead of striding through memory every 100 words



Loop Fusion Example

```

/* Before */
for (i = 0; i < N; i = i+1)
    for (j = 0; j < N; j = j+1)
        a[i][j] = 1/b[i][j] * c[i][j];
for (i = 0; i < N; i = i+1)
    for (j = 0; j < N; j = j+1)
        d[i][j] = a[i][j] + c[i][j];

/* After */
for (i = 0; i < N; i = i+1)
    for (j = 0; j < N; j = j+1)
    {
        a[i][j] = 1/b[i][j] * c[i][j];
        d[i][j] = a[i][j] + c[i][j];
    }
    
```

Temporal locality: 2 misses per access to a & c vs. one miss per access

Example: Matrix Multiply

		j				
1x1	1x2	1x3	1x4	1x5	1x6	
2x1	2x2	2x3	2x4	2x5	2x6	
3x1	3x2	3x3	3x4	3x5	3x6	
4x1	4x2	4x3	4x4	4x5	4x6	
5x1	5x2	5x3	5x4	5x5	5x6	
6x1	6x2	6x3	6x4	6x5	6x6	

		k				
1	1	1	1	1	1	
2	2	2	2	2	2	
3	3	3	3	3	3	
4	4	4	4	4	4	
5	5	5	5	5	5	
6	6	6	6	6	6	

		j				
1	2	3	4	5	6	
1	2	3	4	5	6	
1	2	3	4	5	6	
1	2	3	4	5	6	
1	2	3	4	5	6	
1	2	3	4	5	6	

Matrix X: i x j

Matrix Y: i x k

Matrix Z: k x j

$$X = Y \times Z$$

```

/* Before */
for ( i = 0; i < N; i = i+1)
  for ( j = 0; j < N; j = j+1)
    { x = 0;
      for ( k = 0; k < N; k = k+1){
        for ( i = 0; i < N; i = i+1){
          x[i][j] = x;
        }
      }
    }

```

Read all N x N elements of z[]
Read N elements of 1 row of y[] repeatedly
Write Elements of 1 row of x[]

To calculate one entry of X: Sum the multiplication of every element in a single row of Y by every corresponding element in a single column of Z

Example: Matrix Multiply

		j				
1x1	1x2	1x3	1x4	1x5	1x6	
2x1	2x2	2x3	2x4	2x5	2x6	
3x1	3x2	3x3	3x4	3x5	3x6	
4x1	4x2	4x3	4x4	4x5	4x6	
5x1	5x2	5x3	5x4	5x5	5x6	
6x1	6x2	6x3	6x4	6x5	6x6	

		k				
1	1	1	1	1	1	
2	2	2	2	2	2	
3	3	3	3	3	3	
4	4	4	4	4	4	
5	5	5	5	5	5	
6	6	6	6	6	6	

		j				
1	2	3	4	5	6	
1	2	3	4	5	6	
1	2	3	4	5	6	
1	2	3	4	5	6	
1	2	3	4	5	6	
1	2	3	4	5	6	

Matrix X: i x j

Matrix Y: i x k

Matrix Z: k x j

$$X = Y \times Z$$

Possibilities: Cache large enough to hold...
...all three N x N matrices (cold misses only if no conflicts)
...one N x N matrix, one row of N (limited capacity misses, no thrashing)
...neither row of N or one N x N matrix (capacity misses to compute single row of X)

Conclusion: Capacity Misses as a Function of N & Cache Size:
-When no capacity misses: O(3 N x N)

Blocking

- Problem:
 - Worst case capacity misses: $2N^3 + N^2$
 - Working set of matrix elements is too large to fit in cache
- Solution:
 - Sub-divide matrices into smaller groups of working sets that will fit in cache, iterate through all subgroups (improves temporal locality)
- Maximize accesses to the data loaded into the cache before data is replaced
 - Can be done at block or register levels

Blocking Example: Matrix Multiply

		jj			
1x1	1x2	1x3	1x4	1x5	
2x1	2x2	2x3	2x4	2x5	
3x1	3x2	3x3	3x4	3x5	
4x1	4x2	4x3	4x4	4x5	
5x1	5x2	5x3	5x4	5x5	
6x1	6x2	6x3	6x4	6x5	

		k			
1	1	1	1	1	
2	2	2	2	2	
3	3	3	3	3	
4	4	4	4	4	
5	5	5	5	5	
6	6	6	6	6	

		jj			
1	2	3	4	5	
1	2	3	4	5	
1	2	3	4	5	
1	2	3	4	5	
1	2	3	4	5	
1	2	3	4	5	

Matrix X: i x j

Matrix Y: i x k

Matrix Z: k x j

$$X = Y \times Z$$

```

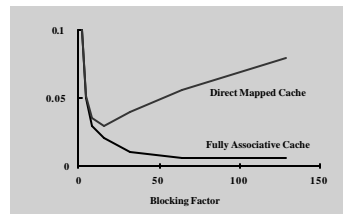
/* After */
for ( jj = 0; jj < N; jj = jj+B)
  for ( kk = 0; kk < N; kk = kk+B)
    for ( i = 0; i < N; i = i+1)
      for ( j = jj; j < min(jj+B-1,N); j = j+1)
        { x = 0;
          for ( k = kk; k < min(kk+B-1,N); k = k+1) {
            x = x + y[i][k]*z[k][j];
          }
          x[i][j] = x;
        }

```

Blocking Example

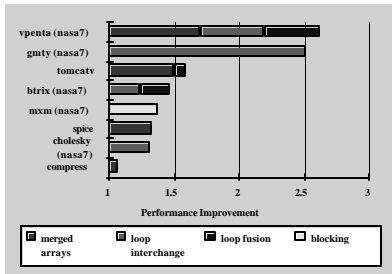
- Capacity Misses from $2N^3 + N^2$ to $2N^3/B + N^2$
- B called *Blocking Factor*
- Blocking traditionally aimed at capacity misses
 - Assumes conflict misses insignificant or addressed by associativity
 - Blocking → smaller working set of blocks, therefore may reduce conflicts

Reducing Conflict Misses by Blocking



- Conflict misses in caches not FA vs. Blocking size
 - Lam et al [1991] a blocking factor of 24 had a fifth the misses vs. 48 despite both fit in cache

Summary of Compiler Optimizations to Reduce Cache Misses



Summary

$$CPUtime = IC \times \left(CPI_{\text{average}} + \frac{\text{Memory accesses}}{\text{Instruction}} \times \text{Miss rate} \times \text{Miss penalty} \right) \times \text{Clock cycle time}$$

- 3 Cs: Compulsory, Capacity, Conflict Misses
- Reducing Miss Rate
 - Reduce Misses via Larger Block Size
 - Reduce Misses via Larger Cache Size
 - Reduce Misses via Higher Associativity
 - Reducing Misses via Pseudo-Associativity
 - Reducing Misses by Compiler Optimizations
- Remember danger of concentrating on just one parameter when evaluating performance (tradeoffs)

Next: Miss penalty...