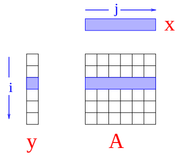
Cache Models
and
Program Transformations

Goal of this lecture

- We have looked at computational science applications, and isolated key kernels (MVM,MMM,linear system solvers,...).
- We have studied caches and virtual memory, and we understand what causes cache misses (cold, capacity, conflict).
- Let us look at how to make some of the kernels run well on machines with caches.

Matrix-vector Product



Code:

```
for i = 1,N
 for j = 1,N
   y(i) = y(i) + A(i,j)*x(j)
```

Total number of references $= 4N^2$

We want to study two questions.

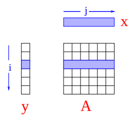
- Can we predict the miss ratio of different variations of this program for different cache models?
- What transformations can we do to improve performance? That is, how do we improve the miss ratio?

Reuse Distance: If r_1 and r_2 are two references to the same cache line in some memory stream, $reuseDistance(r_1, r_2)$ is the number of distinct cache lines referenced between r_1 and r_2.

Cache model:

- fully associative cache (so no conflict misses)
- LRU replacement strategy
- We will look at two extremes
  - large cache model: no capacity misses
  - small cache model: miss if reuse distance is some function of problem size (size of arrays)

Scenario I



Cache model:

- fully associative cache (no conflict misses)
- LRU replacement strategy
- cache line size = 1 floating-point number

Small cache: assume cache can hold fewer than (2N+1) numbers

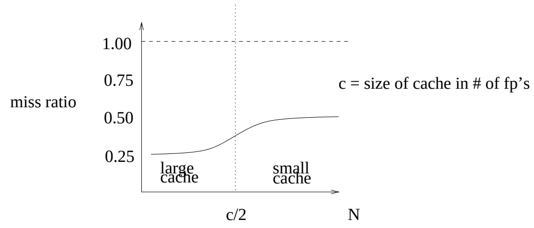
Misses:

- matrix A: N^2 cold misses
- vector x: N cold misses + $N(N-1)$ capacity misses
- vector y: N cold misses
- Miss ratio = $(2N^2 + N)/4N^2 \rightarrow 0.5$

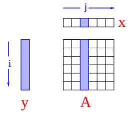
Large cache model: cache can hold more than (2N+1) numbers

Misses:

- matrix A: N^2 cold misses
- vector x: N cold misses
- vector y: N cold misses
- Miss ratio = $(N^2 + 2N)/4N^2 \rightarrow 0.25$



Scenario II



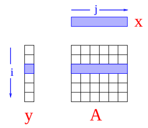
Same cache model as Scenario I but different code

Code: walk matrix A by columns

```
for j = 1,N
 for i = 1,N //SAXPY
   y(i) = y(i) + A(i,j)*x(j)
```

It is easy to show that miss ratios are identical to Scenario I.

Scenario III



Cache model:

- fully associative cache (no conflict misses)
- LRU replacement strategy
- cache line size = b floating-point numbers (can exploit spatial locality)

Code: (original) i-j loop order

```
for i = 1,N
 for j = 1,N
   y(i) = y(i) + A(i,j)*x(j)
```

Let us assume A is stored in row-major order.

Small cache:

Misses:

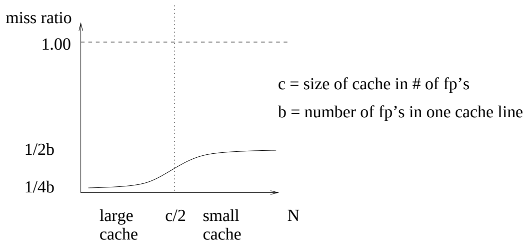
- matrix A: N^2/b cold misses
- vector x: N/b cold misses + $N(N-1)/b$ capacity misses
- vector y: N/b cold misses
- Miss ratio = (1/2 + 1/4N)*(1/b) → 1/2b

Large cache:

Misses:

- matrix A: N^2/b cold misses
- vector x: N/b cold misses
- vector y: N/b cold misses
- Miss ratio = (1/4 + 1/2N)*(1/b) → 1/4b

Transition from small cache to large cache when $2N = c$.

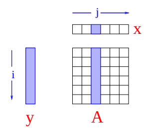


Miss ratios for Scenario III

Let us plug in some numbers for SGI Octane:

- Line size = 32 bytes ⇒ b = 4
- Cache size = 32 Kb ⇒ c = 4K
- Large cache miss ratio = 1/16 = 0.06
- Small cache miss ratio = 0.12
- Small/large transition size = 2000

Scenario IV



Cache model:

- fully associative cache (no conflict misses)
- LRU replacement strategy
- cache line size = b floating-point numbers
  (can exploit spatial locality)

Code: j-i loop order

```
for j = 1,N
 for i = 1,N
   y(i) = y(i) + A(i,j)*x(j)
```

Note: we are not walking over A in memory layout order

Small cache:

Misses:

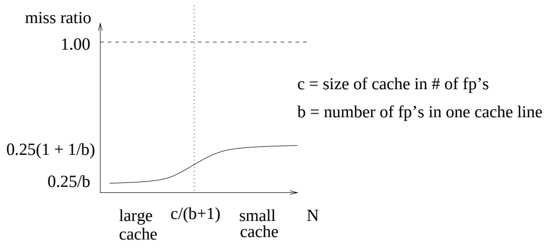
- matrix A: $N^2/$ cold misses
- vector x: N/b cold misses
- vector y: N/b cold misses + $N(N-1)/b$ capacity misses
- Miss ratio = 0.25*(1+ 1/b) + 1/4Nb → 0.25*(1+1/b)

Large cache:

Misses:

- matrix A: N^2/b cold misses
- vector x: N/b cold misses
- vector y: N/b cold misses
- Miss ratio = (1/4 + 1/2N)*(1/b) → 1/4b

Transition from small cache to large cache when c > bN + N

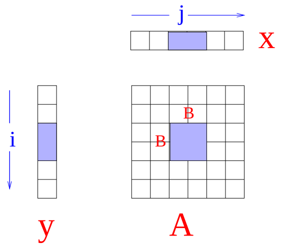


Miss ratios for Scenario IV

Let us plug some numbers in for SGI Octane:

- Line size = 32 bytes ⇒ b = 4
- Cache size = 32 Kb ⇒ c = 4K
- Large cache miss ratio = $1/16$ = 0.06
- Small cache miss ratio = 0.31
- Small/large transition size = 800

Scenario V: Blocked Code

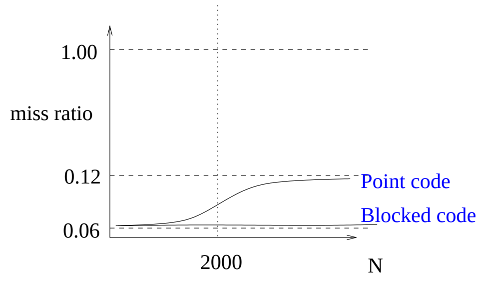


Code:

```
for bi = 1, N/B
 for bj = 1, N/B
   for i = (bi-1)*B +1, bi*B
     for j = (bj-1)*B +1, bj*B
      y(i) = y(i) + A(i,j)*x(j)
```


- Pick block size B so that you effectively have large cache model while executing code within block (2B = c).
- Misses within a block:
  - matrix A: B^2/b cold misses
  - vector x: B/b
  - vector y: B/b
- Total number of block computations = $(N/B)^2$
- Miss ratio = (0.25 + 1/2B)*1/b → 0.25/b
- For Octane, we have miss ratio is roughly 0.06 independent of problem size.

Putting it all together for SGI Octane



Miss ratio predictions for MVM point and blocked codes

We have assumed a fully associative cache.

Conflict misses will have the effect of reducing effective cache size, so transition from large to small cache model should happen sooner than predicted.

Key transformations

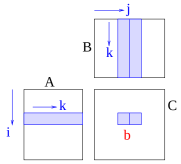
- Loop permutation

```
for j = 1, N                          for i = 1, N
  for i = 1, N                   =>    for j = 1, N
    y(i) = y(i) + A(i,j)*x(j)            y(i) = y(i) + A(i,j)*x(j)
```

- Loop tiling

```
for i = 1, N                          for bi = 1, N/B
  for j = 1, N                  =>     for bj = 1, N/B
    y(i) = y(i)+A(i,j)*x(j)              for i = (bi-1)*B +1, bi*B
                                          for j = (bj-1)*B +1, bj*B
                                           y(i) = y(i) + A(i,j)*x(j)
```

- Warning: these transformations may be illegal in some codes.



Small cache:

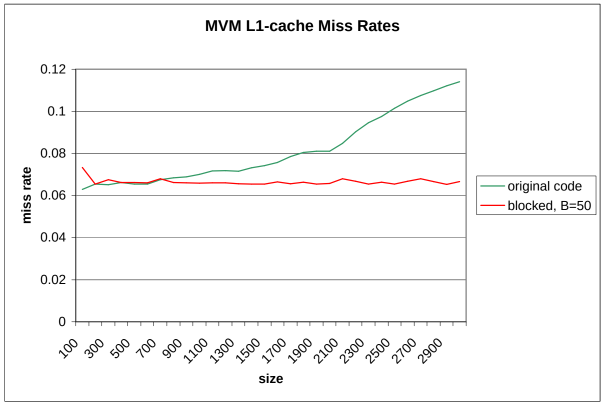
Misses for each cache line of C:

- matrix A: $b * (N/b)$
- matrix B: $b * N$
- matrix C: b
- Total number of misses per cache line of C $= N(b+1) + b$

Total number of misses $= N^2/b * (N(b+1) + b) \rightarrow N^3(b+1)/b$

Total number of references $= 4N^3$

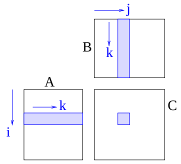
Miss ratio $\rightarrow 0.25(b+1)/b$



Experimental Results on SGI Octane

Predictions agree reasonably well with experiments.

Matrix-matrix Product



Code:

```
for i = 1,N
 for j = 1,N
   for k = 1,N
     C(i,j) = C(i,j) + A(i,k)*B(k,j)
```

Cache model: assume cache line size is b fp's

Large cache:

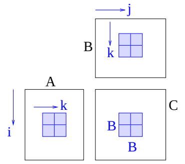
Cold misses = $3 * N^2/b$

Miss ratio = $3 * N^2/4bN^3 = 0.75/bN$

For large cache model, miss ratio decreases as the size of the problem increases!

Intuition: lot of data reuse, so once matrices all fit into cache, code goes full blast.

Blocked code:



Code:

```
for bi = 1, N/B
 for bj = 1, N/B
  for bk = 1, N/B
   for i = (bi-1)*B +1, bi*B
     for j = (bj-1)*B +1, bj*B
      for k = (bk-1)*B +1, bk*B
      y(i) = y(i) + A(i,j)*x(j)
```

Choose B so we have large cache model: $3B^2 < c$

Miss ratio of blocked code = $0.75/bB$.

Since $B < sqrt(c/3)$, lower bound on miss ratio is $1.3/b * sqrt(c)$.

For Octane, miss ratio > 0.05.

As before, we have ignored conflict misses, so actual miss ratio we can obtain from blocking alone will be more.