L10 Cache Optimizations Exercises

Zonghua Gu, 2018

Acknowledgement: some slides taken from UC Berkeley CS61C

Review

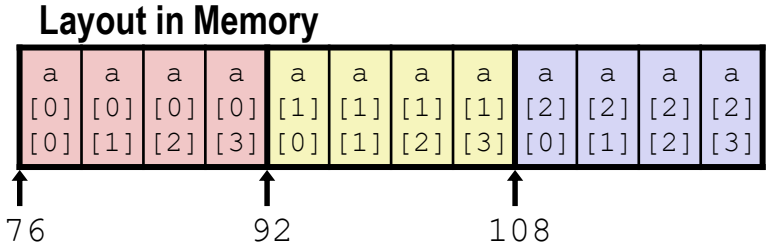
Example: Matrix Sum

- Consider the following two loops, which calculate the sum of the entries in a 3x4 matrix A of 32-bit integers:

Loop A	*Loop B*

`sum = 0;`
`for (i = 0; i < 3; i++)`
`  for (j = 0; j < 4; j++)`
`  sum += A[i][j];`
	`sum = 0;`
	`for (j = 0; j < 4; j++)`
	`  for (i = 0; i < 3; i++)`
	`  sum += A[i][j];`

- Matrix A is stored contiguously in memory in row-major order. Row major order means that elements in the same row of the matrix are adjacent in memory (A[i][j] resides in memory location [4*(64*i + j)]).



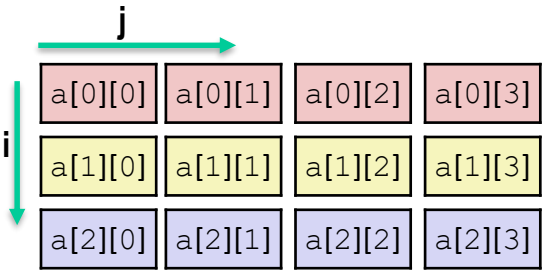
Note: 76 is just one possible starting address of A

Review

Matrix Sum Loop A

Loop A

```
sum = 0;
for (i = 0; i < 3; i++)
  for (j = 0; j < 4; j++)
    sum += A[i][j];
```



Access Pattern:
stride = 1

1) a[0][0]
2) a[0][1]
3) a[0][2]
4) a[0][3]
5) a[1][0]
6) a[1][1]
7) a[1][2]
8) a[1][3]
9) a[2][0]
10) a[2][1]
11) a[2][2]
12) a[2][3]

Layout in Memory

a [0] [0]	a [0] [1]	a [0] [2]	a [0] [3]	a [1] [0]	a [1] [1]	a [1] [2]	a [1] [3]	a [2] [0]	a [2] [1]	a [2] [2]	a [2] [3]

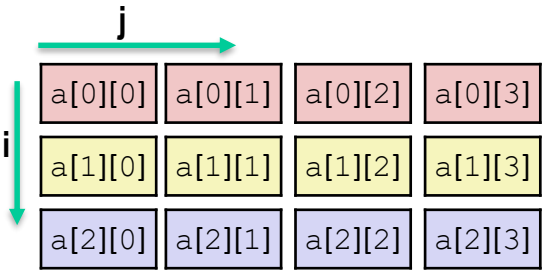
- Index j is incremented in the inner loop
- Good spatial locality for a[][]
- No temporal locality for a[][]
  - Each element is used only once

Review

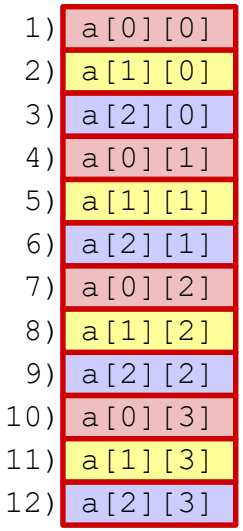
Matrix Sum Loop B

Loop B

```
sum = 0;
for (j = 0; j < 4; j++)
  for (i = 0; i < 3; i++)
    sum += A[i][j];
```



Access Pattern:
stride = 4



Layout in Memory

a [0] [0]	a [0] [1]	a [0] [2]	a [0] [3]	a [1] [0]	a [1] [1]	a [1] [2]	a [1] [3]	a [2] [0]	a [2] [1]	a [2] [2]	a [2] [3]

- Index i is incremented in the inner loop
- Poor spatial locality for a[][]
- No temporal locality for a[][]
  - Each element is used only once

Q1: Loop Ordering w/ 16B DM Cache

Loop A	*Loop B*

`sum = 0;`
`for (i = 0; i < 3; i++)`
`  for (j = 0; j < 4; j++)`
`  sum += A[i][j];`
	`sum = 0;`
	`for (j = 0; j < 4; j++)`
	`  for (i = 0; i < 3; i++)`
	`  sum += A[i][j];`

- Assume that the caches are initially empty. Only accesses to matrix A cause memory references and all other necessary variables are stored in registers. Instructions are in a separate instruction cache.
- Consider a 16 Byte DM data cache with 16 Bytes/block, with a total of 1 block. Calculate the number of cache misses for Loop A and Loop B.

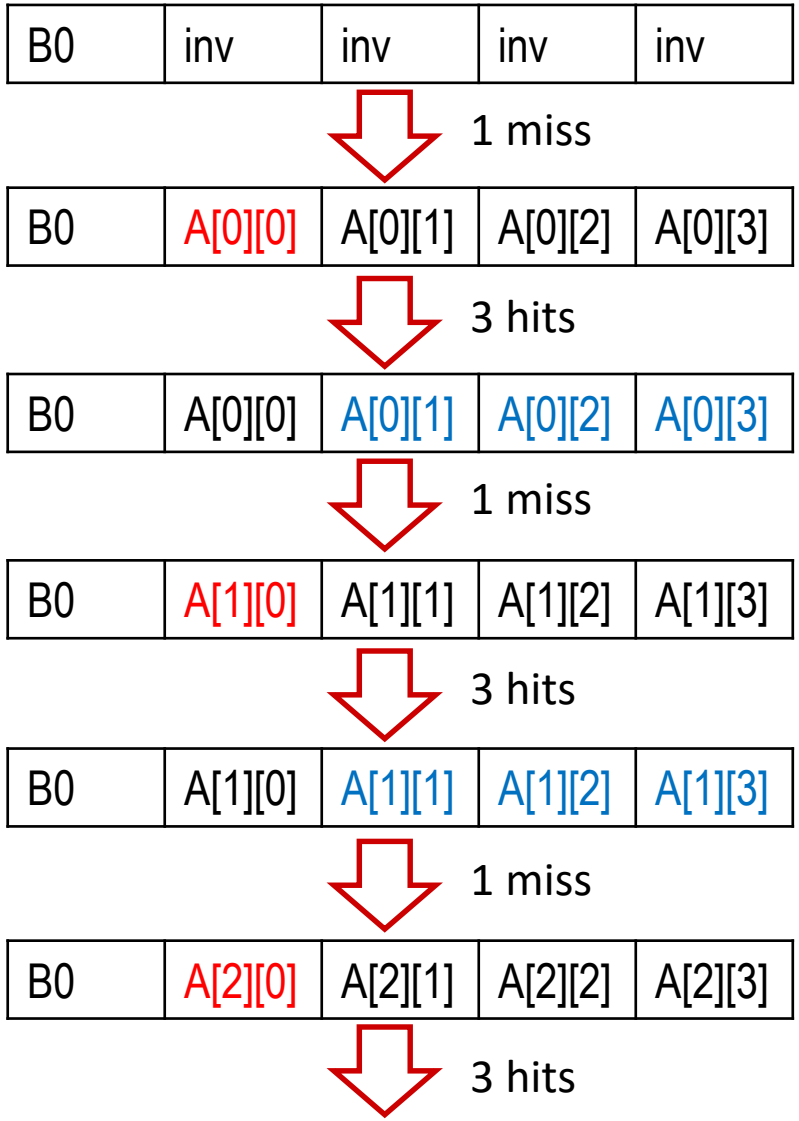
B0	A[0][0]	A[0][1]	A[0][2]	A[0][3]

A1.1 Loop A

Loop A	*Loop B*

`sum = 0;`
`for (i = 0; i < 3; i++)`
`  for (j = 0; j < 4; j++)`
`  sum += A[i][j];`
	`sum = 0;`
	`for (j = 0; j < 4; j++)`
	`  for (i = 0; i < 3; i++)`
	`  sum += A[i][j];`

- Loop A accesses memory sequentially (each iteration of Loop A sums a row in matrix A), an access to an int that maps to the first word in a cache line will miss (red words in the table) but the next 3 accesses will hit. Therefore, Loop A will only have compulsory misses, with total of 3*(4/4)=3 misses.
  - In each one of the 3 outer loop iterations, 4 memory accesses incur (4/4=1) cache miss, since they are absorbed by 1 cache block

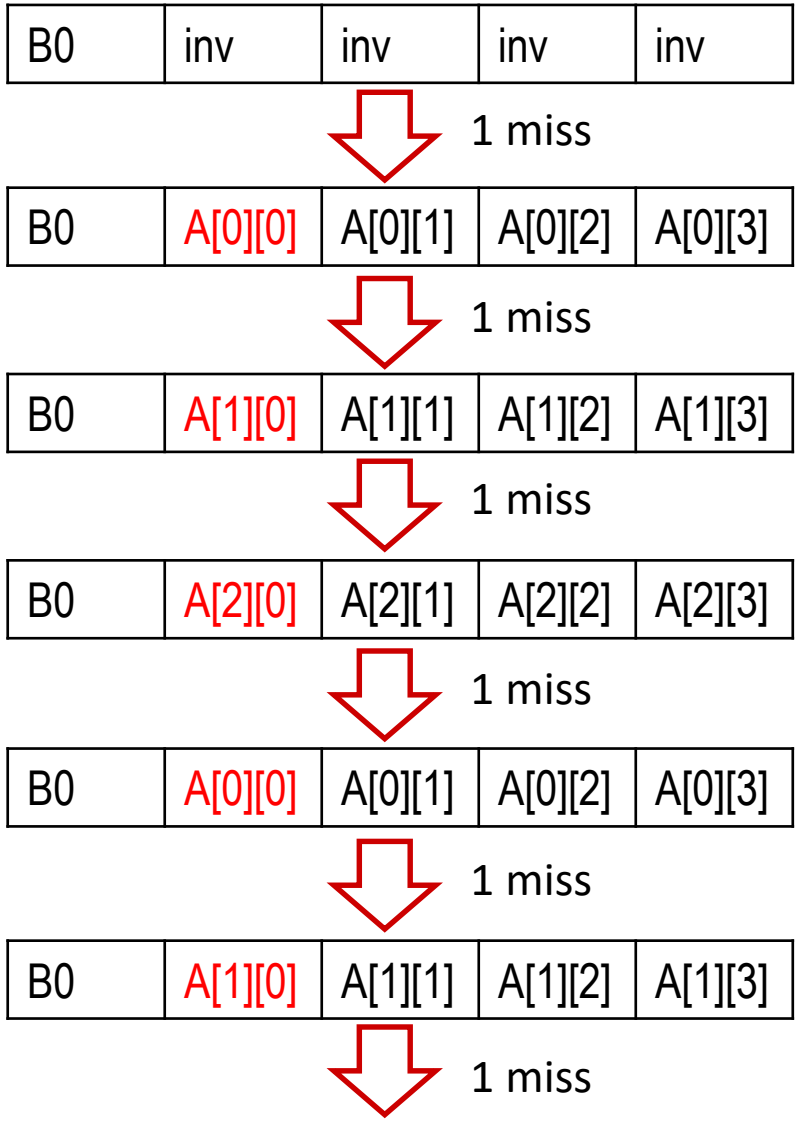


A1.2 Loop B

Loop A	*Loop B*

`sum = 0;`
`for (i = 0; i < 3; i++)`
`  for (j = 0; j < 4; j++)`
`  sum += A[i][j];`
	`sum = 0;`
	`for (j = 0; j < 4; j++)`
	`  for (i = 0; i < 3; i++)`
	`  sum += A[i][j];`

- Loop B accesses memory in stride of 4. Therefore, every access will be a miss with total of 4*(3/1)=12 misses.
  - In each one of the 4 outer loop iterations, 3 memory accesses incur (3/1=3) cache misses



Q2: Loop Ordering w/ 16B FA Cache

Loop A	*Loop B*

`sum = 0;`
`for (i = 0; i < 3; i++)`
`  for (j = 0; j < 4; j++)`
`  sum += A[i][j];`
	`sum = 0;`
	`for (j = 0; j < 4; j++)`
	`  for (i = 0; i < 3; i++)`
	`  sum += A[i][j];`

- Instead of DM cache, consider a 16 Byte FA data cache with 16 Bytes/block, with a total of 1 block. Calculate the number of cache misses for Loop A and Loop B.

B0	A[0][0]	A[0][1]	A[0][2]	A[0][3]

- A: If the cache size is a single block, then associativity makes no difference!

Q3: Loop Ordering w/ 64B DM Cache

Loop A	*Loop B*

`sum = 0;`
`for (i = 0; i < 3; i++)`
`  for (j = 0; j < 4; j++)`
`  sum += A[i][j];`
	`sum = 0;`
	`for (j = 0; j < 4; j++)`
	`  for (i = 0; i < 3; i++)`
	`  sum += A[i][j];`

- Consider a 64 Byte DM data cache with 16 Bytes/block, with a total of 4 blocks. Calculate the number of cache misses for Loop A and Loop B. How about SA cache with LRU replacement policy?

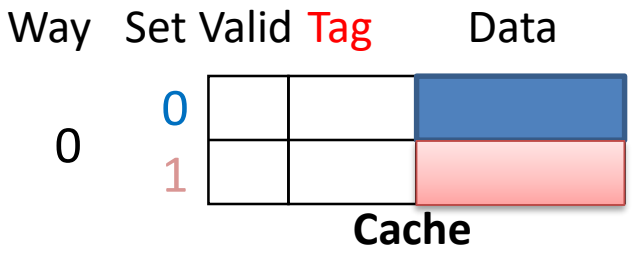
B0	A[0][0]	A[0][1]	A[0][2]	A[0][3]
B1	A[1][0]	A[1][1]	A[1][2]	A[1][3]
B2	A[2][0]	A[2][1]	A[2][2]	A[2][3]
B3				

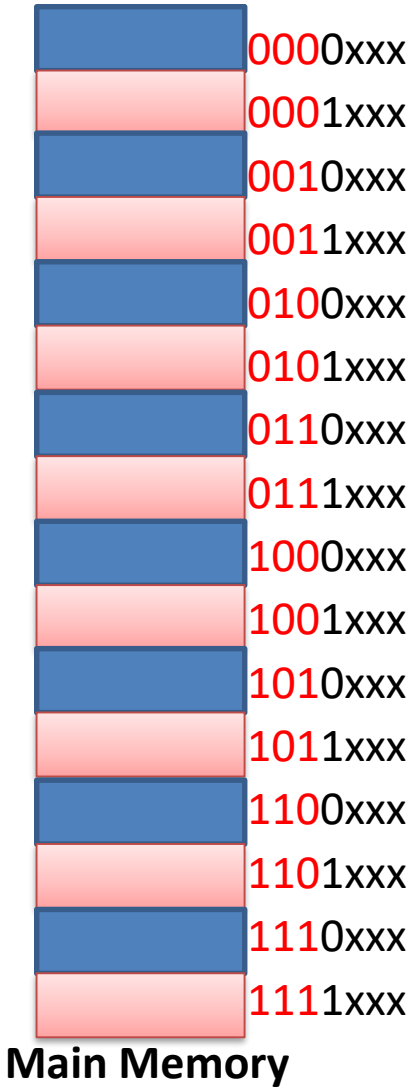
- A: Since the cache is large enough to contain the entire matrix A, both Loop A and Loop B will also only have compulsory misses, with total of 3 misses.

Q4 DM Cache

Loop A	*Loop B*

`sum = 0;`
`for (i = 0; i < 3; i++)`
`  for (j = 0; j < 4; j++)`
`  sum += A[i][j];`
	`sum = 0;`
	`for (j = 0; j < 4; j++)`
	`  for (i = 0; i < 3; i++)`
	`  sum += A[i][j];`





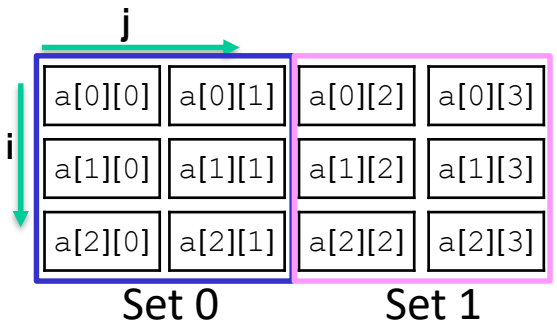
- 16-Byte DM cache; 7-bit memory address: 3-bit Tag, 1-bit Set Index, 3-bit Offset
  - 8 Bytes/block
  - 2 Sets (2 blocks)
- Q: suppose matrix A is stored at starting address 0000000. Compute the number of cache misses for Loop A or Loop B

Q4 DM Cache

Loop A	*Loop B*

`sum = 0;`
`for (i = 0; i < 3; i++)`
`  for (j = 0; j < 4; j++)`
`  sum += A[i][j];`
	`sum = 0;`
	`for (j = 0; j < 4; j++)`
	`  for (i = 0; i < 3; i++)`
	`  sum += A[i][j];`

- Each cache block can fit two 32b integers
  - A[0][0], A[0][1] are mapped to Set 0
  - A[0][2], A[0][3] are mapped to Set 1
  - A[1][0], A[0][1] are mapped to Set 0
  - A[1][2], A[0][3] are mapped to Set 1
  - …



- Loop A will only have compulsory misses, with total of 3*(4/2)=6 misses.
  - In each one of the 3 outer loop iterations, 4 memory accesses incur (4/2=2) cache miss, since they are absorbed by 2 cache blocks
- Loop B will have misses at every step, with a total of 3*(4/1)=12 misses

Loop A DM Cache

Loop A	Loop B

`sum = 0;`
`for (i = 0; i < 3; i++)`
`  for (j = 0; j < 4; j++)`
`  sum += A[i][j];`
	`sum = 0;`
	`for (j = 0; j < 4; j++)`
	`  for (i = 0; i < 3; i++)`
	`  sum += A[i][j];`

B0	inv	inv
B1	inv	inv

1 miss

B0	A[0][0]	A[0][1]
B1	inv	inv

1 hit

B0	A[0][0]	A[0][1]
B1	inv	inv

1 miss

B0	A[0][0]	A[0][1]
B1	A[0][2]	A[0][3]

1 hit

B0	A[0][0]	A[0][1]
B1	A[0][2]	A[0][3]

1 miss

B0	A[1][0]	A[1][1]
B1	A[0][2]	A[0][3]

1 hit

B0	A[1][0]	A[1][1]
B1	A[0][2]	A[0][3]

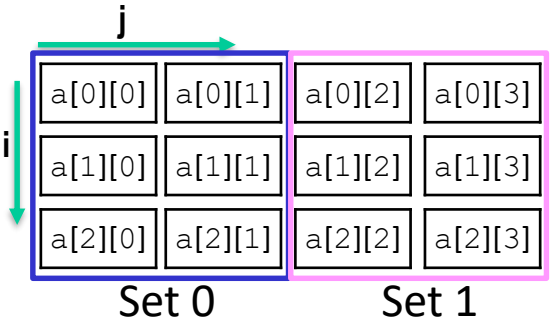
1 miss

B0	A[1][0]	A[1][1]
B1	A[1][2]	A[1][3]

1 hit

B0	A[1][0]	A[1][1]
B1	A[1][2]	A[1][3]

...



Loop B DM Cache

Loop A	*Loop B*

`sum = 0;`
`for (i = 0; i < 3; i++)`
`  for (j = 0; j < 4; j++)`
`  sum += A[i][j];`
	`sum = 0;`
	`for (j = 0; j < 4; j++)`
	`  for (i = 0; i < 3; i++)`
	`  sum += A[i][j];`

B0	inv	inv
B1	inv	inv

1 miss

B0	A[0][0]	A[0][1]
B1	inv	inv

1 miss

B0	A[1][0]	A[1][1]
B1	inv	inv

1 miss

B0	A[2][0]	A[2][1]
B1	inv	inv

1 miss

B0	A[0][0]	A[0][1]
B1	inv	inv

1 miss

B0	A[1][0]	A[1][1]
B1	inv	inv

1 miss

B0	A[2][0]	A[2][1]
B1	inv	inv

1 miss

B0	A[2][0]	A[2][1]
B1	A[0][2]	A[0][3]

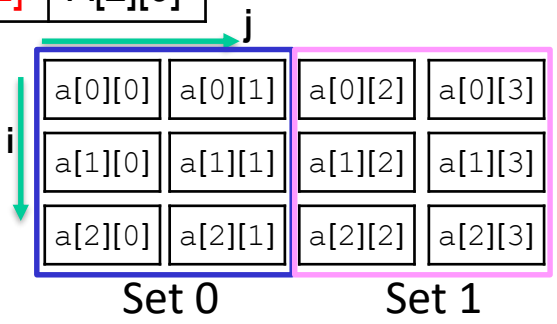
1 miss

B0	A[2][0]	A[2][1]
B1	A[1][2]	A[1][3]

1 miss

B0	A[2][0]	A[2][1]
B1	A[2][2]	A[2][3]

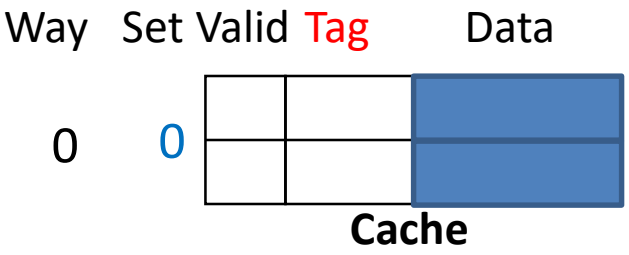
...

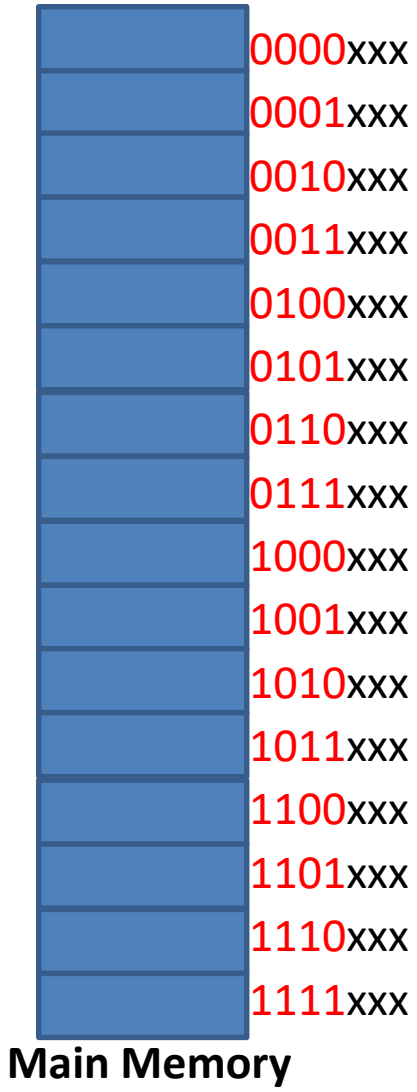


Q5 FA LRU Cache

Loop A	*Loop B*

`sum = 0;`
`for (i = 0; i < 3; i++)`
`  for (j = 0; j < 4; j++)`
`  sum += A[i][j];`
	`sum = 0;`
	`for (j = 0; j < 4; j++)`
	`  for (i = 0; i < 3; i++)`
	`  sum += A[i][j];`





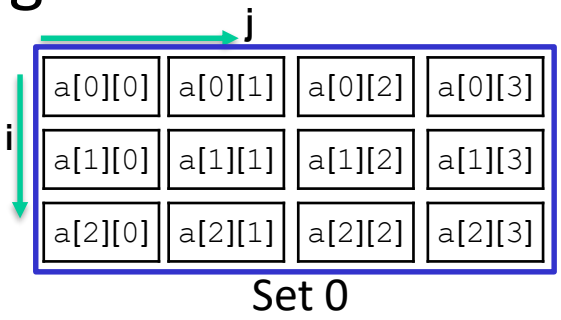
- 16-Byte FA cache w/ LRU replacement policy; 7-bit memory address: 4-bit Tag, 3-bit Offset
  - 8 Bytes/block
  - 1 Set (2 blocks)
- Q: suppose matrix A is stored at starting address 0000000. Compute the number of cache misses for Loop A or Loop B

Q4 FA LRU Cache

Loop A	*Loop B*

`sum = 0;`
`for (i = 0; i < 3; i++)`
`  for (j = 0; j < 4; j++)`
`  sum += A[i][j];`
	`sum = 0;`
	`for (j = 0; j < 4; j++)`
	`  for (i = 0; i < 3; i++)`
	`  sum += A[i][j];`

- Each cache block can fit two 32b integers
  - A[0][0], A[0][1] are mapped to Set 0
  - A[0][2], A[0][3] are mapped to Set 0
  - A[1][0], A[0][1] are mapped to Set 0
  - A[1][2], A[0][3] are mapped to Set 0
  - …



- Loop A will only have compulsory misses, with total of 3*(4/2)=6 misses.
- Loop B will have misses at every step, with a total of 3*(4/1)=12 misses

Loop A FA LRU Cache

Loop A	Loop B

sum = 0;
for (i = 0; i < 3; i++)
  for (j = 0; j < 4; j++)
  sum += A[i][j];
	sum = 0;
	for (j = 0; j < 4; j++)
	  for (i = 0; i < 3; i++)
	  sum += A[i][j];

B0	inv	inv
B1	inv	inv

1 miss

B0	A[0][0]	A[0][1]
B1	inv	inv

1 hit

B0	A[0][0]	A[0][1]
B1	inv	inv

1 miss

B0	A[0][0]	A[0][1]
B1	A[0][2]	A[0][3]

1 hit

B0	A[0][0]	A[0][1]
B1	A[0][2]	A[0][3]

1 miss

B0	A[1][0]	A[1][1]
B1	A[0][2]	A[0][3]

1 hit

B0	A[1][0]	A[1][1]
B1	A[0][2]	A[0][3]

1 miss

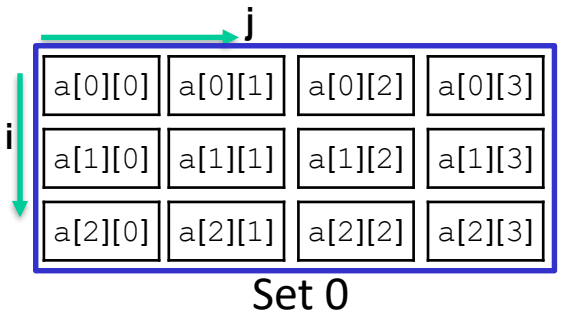
B0	A[1][0]	A[1][1]
B1	A[1][2]	A[1][3]

1 hit

B0	A[1][0]	A[1][1]
B1	A[1][2]	A[1][3]

...

Miss rate of FA cache is the same as DM Cache



Loop B FA LRU Cache

Loop A	*Loop B*

`sum = 0;`
`for (i = 0; i < 3; i++)`
`  for (j = 0; j < 4; j++)`
`  sum += A[i][j];`
	`sum = 0;`
	`for (j = 0; j < 4; j++)`
	`  for (i = 0; i < 3; i++)`
	`  sum += A[i][j];`

B0	inv	inv
B1	inv	inv

1 miss

B0	A[0][0]	A[0][1]
B1	inv	inv

1 miss

B0	A[0][0]	A[0][1]
B1	A[1][0]	A[1][1]

1 miss

B0	A[2][0]	A[2][1]
B1	A[1][0]	A[1][1]

1 miss

B0	A[2][0]	A[2][1]
B1	A[0][0]	A[0][1]

1 miss

B0	A[1][0]	A[1][1]
B1	A[0][0]	A[0][1]

1 miss

B0	A[2][0]	A[2][1]
B1	A[2][0]	A[2][1]

1 miss

B0	A[0][2]	A[0][3]
B1	A[2][0]	A[2][1]

1 miss

B0	A[0][2]	A[0][3]
B1	A[1][2]	A[1][3]

1 miss

B0	A[2][2]	A[2][3]
B1	A[1][2]	A[1][3]

...