CS/ECE 5381/7381
Computer Architecture
Spring 2023

Dr. Manikas

Computer Science

Lecture 16: Mar. 28, 2023

Project 4

- Due **TODAY, Mar. 28** (11:59 pm)
- Cadence Xcelium tool
  - Used to develop and test Verilog code
- Verilog
  - Hardware Description Language
  - Used to construct and simulate computer hardware
- Assignment:
  - Run tool on simple MIPS ALU design

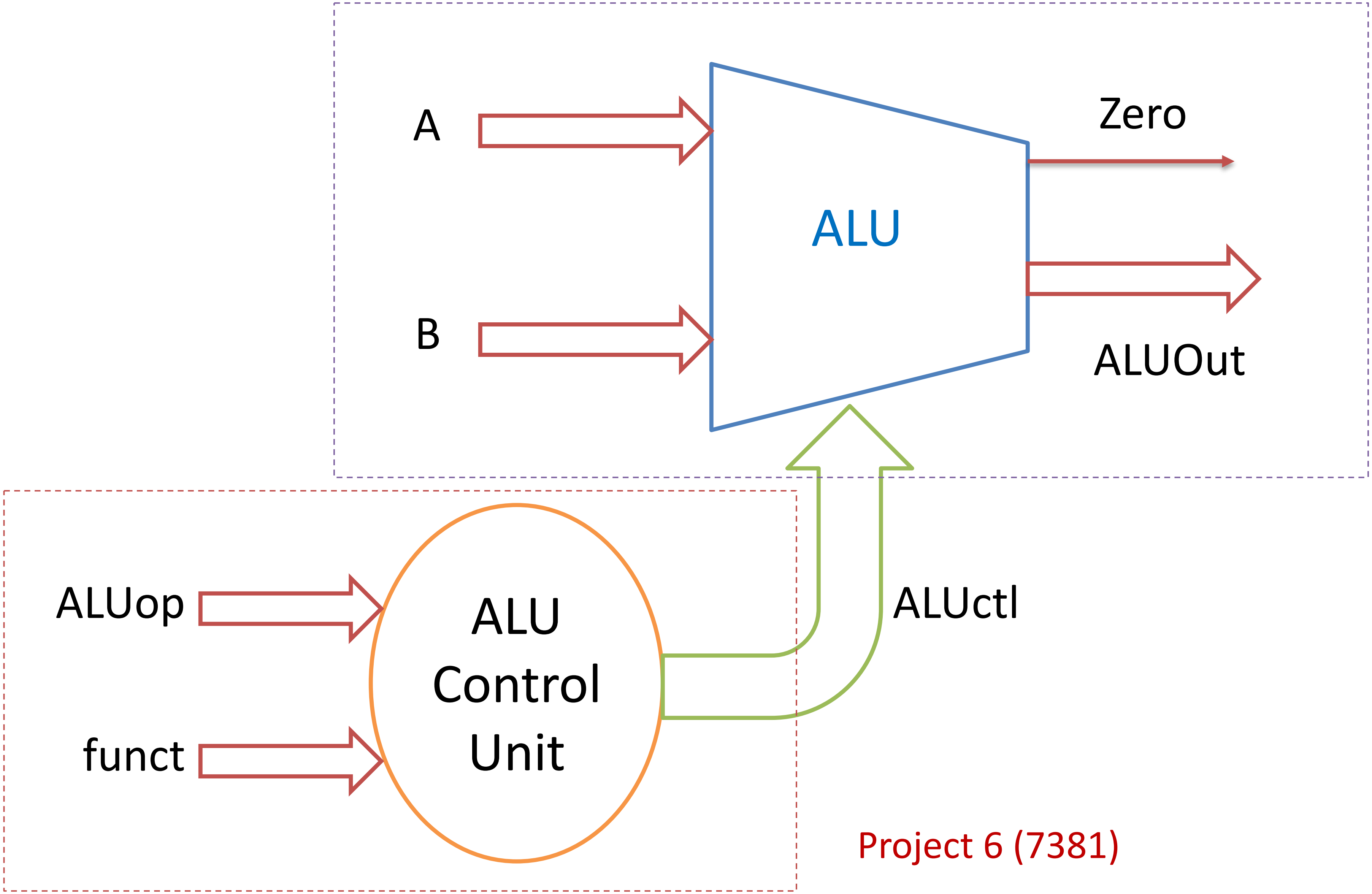
Project 5

- Due NEXT Tues, Apr. 4 (11:59 pm)
- Xcelium tool
- Assignment:
  - Modify Verilog code for basic MIPS ALU of Project 4
    - Add arithmetic and logic functions

Project 6 (7381 only)

- Due NEXT Thur., Apr. 6 (11:59 pm)
- For CS/ECE 7381 students ONLY
- Additional Verilog programming assignment using Xcelium tool
  - Develop control unit for ALU

Project 5 (5/7381)



Project 6 (7381)

Exam 2

- Exam will be administered using Lockdown Browser (same as for Exam 1)
- Exam format also same as Exam 1
  - 25 questions, 2 hours
  - The exam will be available from **Thursday, Mar. 30 at 12 am**
  - The exam must be completed and submitted by **Saturday, Apr 1 at 11:59 pm**

Exam 2

- **Exam 2 will cover the following materials:**
  - Modules: 5 – 8
  - Quizzes: 5 - 7
  - Text: Ch. 3.1 – 3.6, App. B.1 – B.4, Ch. 2.1 – 2.3
- **MATERIALS ALLOWED FOR EXAM:**
  - Open book and notes
  - Calculator

Schedule Notes

- NO lecture for Thursday, Mar. 30, since we have Exam 2

Memory Design Hierarchy

(Chapter 2, Hennessy and Patterson)

Note: some course slides adopted from publisher-provided material

Outline

- 2.1 Introduction
- 2.2 Memory Technology and Optimizations
- 2.3 Ten Advanced Optimizations of Cache Performance
- 2.4 Virtual Memory and Machines

10 Advanced Cache Optimizations

- Increasing cache bandwidth

3. Pipelined caches
4. Multibanked caches
5. Nonblocking caches

3: Increasing Cache Bandwidth by Pipelining

- Pipeline cache access to maintain bandwidth, but higher latency
- Instruction cache access pipeline stages:

- $\Rightarrow$ greater penalty on mispredicted branches

- $\Rightarrow$ more clock cycles between the issue of the load and the use of the data

4: Increasing Cache Bandwidth via Multiple Banks

- Rather than treat the cache as a single monolithic block, divide into independent banks that can support simultaneous accesses
  - E.g.,T1 (“Niagara”) L2 has 4 banks
- Banking works best when accesses naturally spread themselves across banks $\Rightarrow$ mapping of addresses to banks affects behavior of memory system

4: Increasing Cache Bandwidth via Multiple Banks (cont)

- Simple mapping that works well is “sequential interleaving”
  - Spread block addresses sequentially across banks
  - E.g., if there 4 banks, Bank 0 has all blocks whose address modulo 4 is 0; bank 1 has all blocks whose address modulo 4 is 1; …

	0		1		2		3
0		1		2		3	
4		5		6		7	
8		9		10		11	
12		13		14		15	

5. Increasing Cache Bandwidth: Non-Blocking Caches

- *Non-blocking cache* or *lockup-free cache* allow data cache to continue to supply cache hits during a miss
  - requires F/E bits on registers or out-of-order execution
  - requires multi-bank memories
- "*hit under miss*" reduces the effective miss penalty by working during miss vs. ignoring CPU requests

5. Increasing Cache Bandwidth: Non-Blocking Caches (cont)

- "*hit under multiple miss*" or "*miss under miss*" may further lower the effective miss penalty by overlapping multiple misses
  - Significantly increases the complexity of the cache controller as there can be multiple outstanding memory accesses
  - Requires multiple memory banks (otherwise cannot support)
  - Pentium Pro allows 4 outstanding memory misses

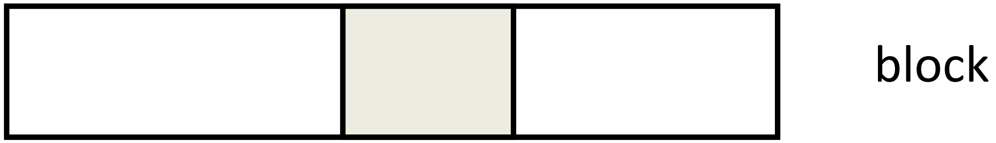
10 Advanced Cache Optimizations

- Reducing Miss Penalty

6. Critical word first
7. Merging write buffers

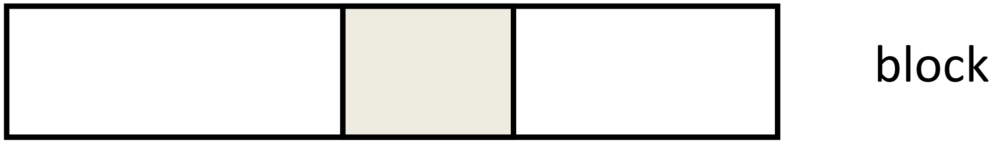
6. Reduce Miss Penalty: Early Restart

- Don’t wait for full block before restarting CPU
- *Early restart*—As soon as the requested word of the block arrives, send it to the CPU and let the CPU continue execution
  - Spatial locality ⇒ tend to want next sequential word, so not clear size of benefit of just early restart



6. Reduce Miss Penalty: Critical Word First

- *Critical Word First*—Request the missed word first from memory and send it to the CPU as soon as it arrives; let the CPU continue execution while filling the rest of the words in the block
  - – Long blocks more popular today ⇒ Critical Word 1st Widely used



7. Merging Write Buffer to Reduce Miss Penalty

- Write buffer to allow processor to continue while waiting to write to memory
- If buffer contains modified blocks, the addresses can be checked to see if address of new data matches the address of a valid write buffer entry
- If so, new data are combined with that entry

7. Merging Write Buffer to Reduce Miss Penalty (cont)

- Increases block size of write for write-through cache of writes to sequential words, bytes since multiword writes more efficient to memory

10 Advanced Cache Optimizations

- Reducing Miss Rate

8. Compiler optimizations

10 Advanced Cache Optimizations

- Reducing Miss Rate

8. Compiler optimizations

8. Reducing Misses by Compiler Optimizations

- *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: 2 sequential arrays */
int val[SIZE];
int key[SIZE];

/* After: 1 array of structures */
struct merge {
  int val;
  int key;
};
struct merge merged_array[SIZE];
```

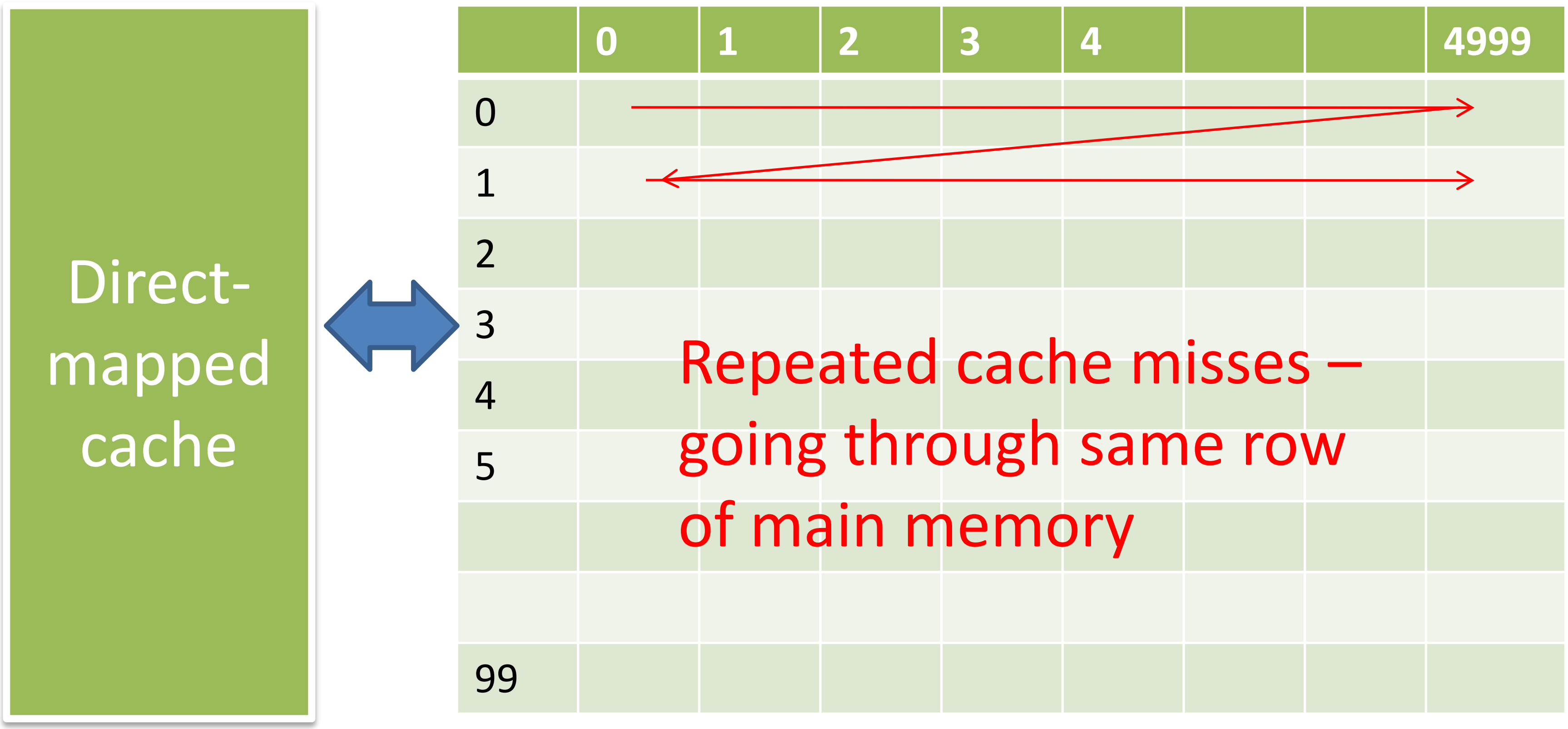
Reducing conflicts between val and key; improve spatial locality

Loop Interchange Example

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

Sequential accesses instead of striding through memory every 100 words; improved spatial locality

Before Loop Interchange

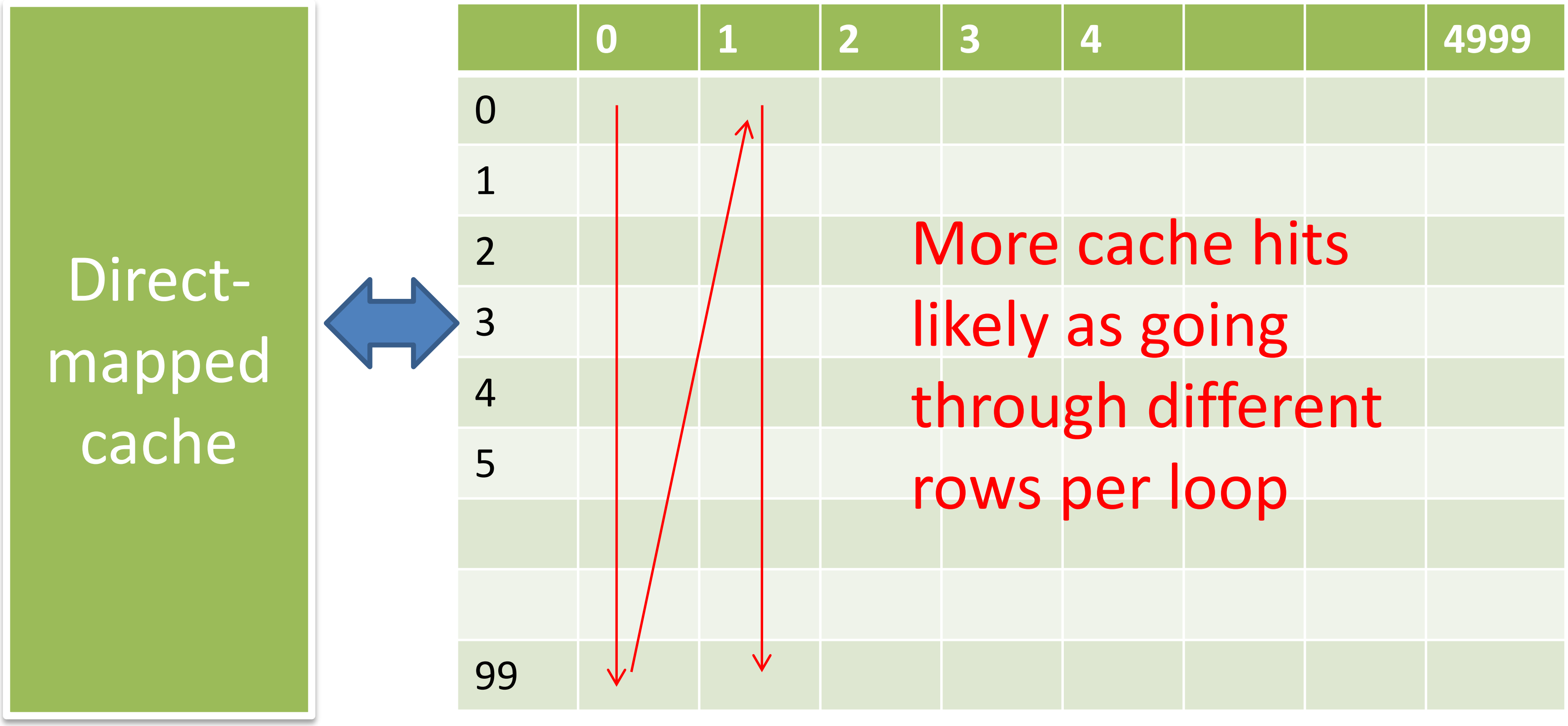


Direct-mapped cache

	0	1	2	3	4			4999
0								
1								
2								
3								
4								
5								
99								

Repeated cache misses – going through same row of main memory

After Loop Interchange



More cache hits likely as going through different rows per loop

Loop Fusion Example - Before

```
/* 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];
```

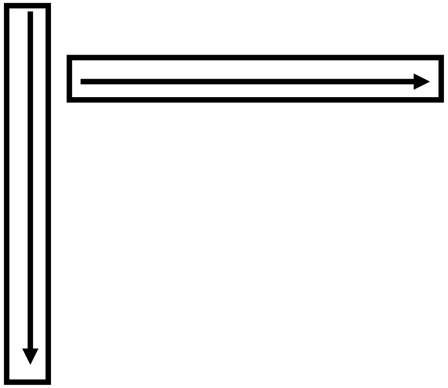
Loop Fusion Example - After

```
/* 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];}
```

2 misses per access to `a` & `c` vs. one miss per access; improve spatial locality

Blocking Example - before

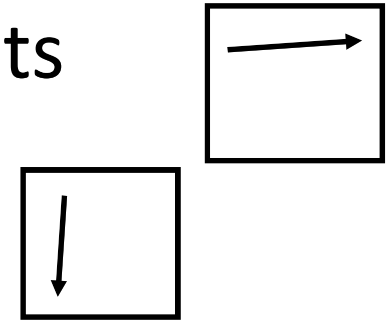
```
/* Before */
for (i = 0; i < N; i = i+1)
  for (j = 0; j < N; j = j+1)
    {r = 0;
     for (k = 0; k < N; k = k+1){
       r = r + y[i][k]*z[k][j];};
     x[i][j] = r;
    };
```



- Two Inner Loops:
  - Read all NxN elements of z[]
  - Read N elements of 1 row of y[] repeatedly
  - Write N elements of 1 row of x[]

Blocking Example – before (cont)

- Capacity Misses a function of N & Cache Size:
  - $2N^3 + N^2$ => (assuming no conflict; otherwise …)
- Idea: compute on BxB submatrix that fits

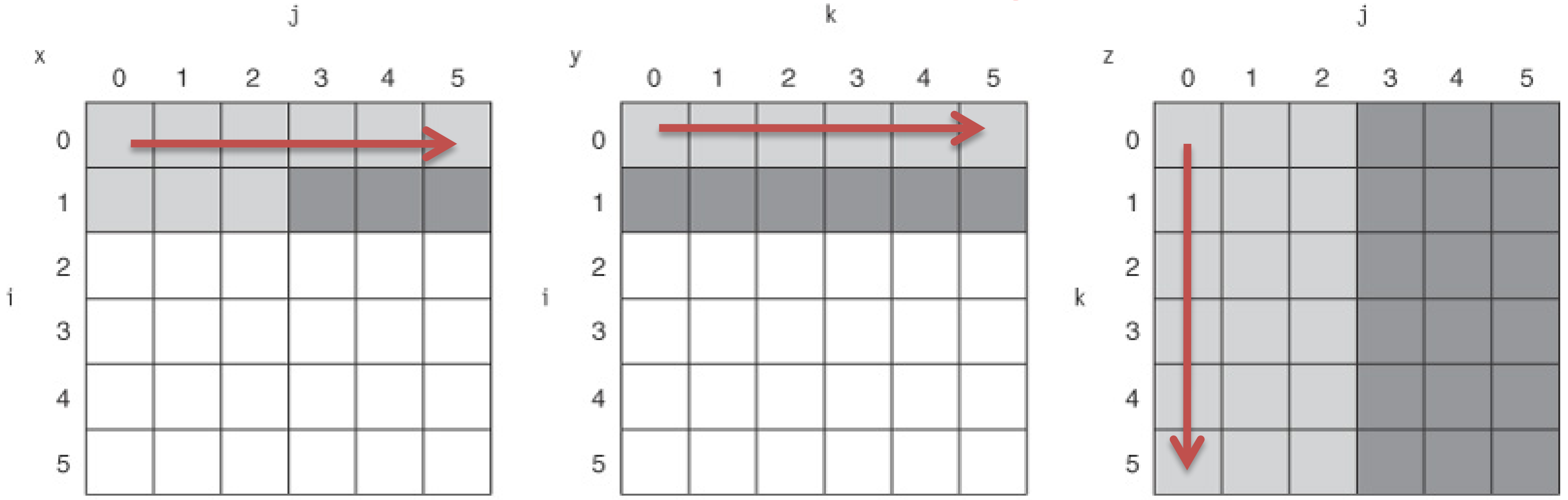


Blocking Example

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

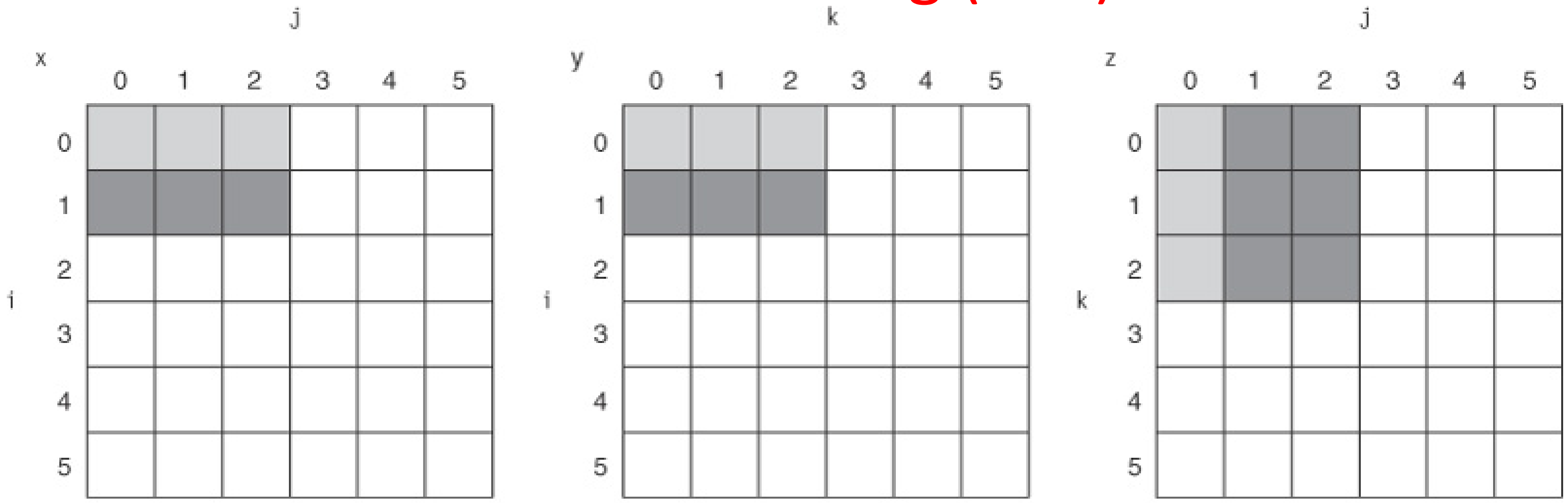
- B called *Blocking Factor*
- Capacity Misses from $2N^3 + N^2$ to $2N^3/B + N^2$

Before blocking



© 2007 Elsevier, Inc. All rights reserved.

After blocking (B=3)



© 2007 Elsevier, Inc. All rights reserved.

10 Advanced Cache Optimizations

- Reducing miss penalty or miss rate via parallelism

9. Hardware prefetching
10. Compiler prefetching

9. Reducing Misses by Hardware Prefetching of Instructions & Data

- Prefetching relies on having extra memory bandwidth that can be used without penalty
- Instruction Prefetching
  - Typically, CPU fetches 2 blocks on a miss: the requested block and the next consecutive block.
  - Requested block is placed in instruction cache when it returns, and prefetched block is placed into instruction stream buffer

Array Storage in Memory

- We may view items as arrays (2D)
- But these are stored in main memory as 1D items
- How to translate? Two possibilities:
  - Row-Major Mapping
    - Used in C/C++, Python
  - Column-Major Mapping
    - Used in Fortran, MATLAB

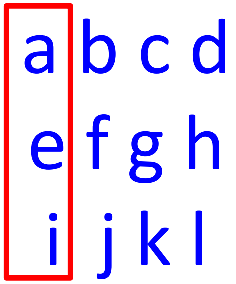
Row-Major Mapping

- Example 3 x 4 array:

a b c d
e f g h
i j k l

- Convert into 1D array y by collecting elements by *rows*
- Within a row, elements are collected from left to right
- Rows are collected from top to bottom
- We get {a, b, c, d, e, f, g, h, i, j, k, l}

Column-Major Mapping



a b c d

e f g h

i j k l

- Convert into 1D array y by collecting elements by *columns*
- Within a column, elements are collected from top to bottom
- Columns are collected from left to right
- We get {a, e, i, b, f, j, c, g, k, d, h, l}