

work to increase locality by automatically performing the kinds of manual code transformations we discussed in Section 6.6 [22, 32, 66, 72, 79, 87, 119]. Carter and colleagues have proposed a cache-aware memory controller [17]. Other researchers have developed *cache-oblivious* algorithms that are designed to run well without any explicit knowledge of the structure of the underlying cache memory [30, 38, 39, 9].

There is a large body of literature on building and using disk storage. Many storage researchers look for ways to aggregate individual disks into larger, more robust, and more secure storage pools [20, 40, 41, 83, 121]. Others look for ways to use caches and locality to improve the performance of disk accesses [12, 21]. Systems such as Exokernel provide increased user-level control of disk and memory resources [57]. Systems such as the Andrew File System [78] and Coda [94] extend the memory hierarchy across computer networks and mobile notebook computers. Schindler and Ganger developed an interesting tool that automatically characterizes the geometry and performance of SCSI disk drives [95]. Researchers have investigated techniques for building and using flash-based SSDs [8, 81].

Homework Problems

6.22 ♦♦

Suppose you are asked to design a rotating disk where the number of bits per track is constant. You know that the number of bits per track is determined by the circumference of the innermost track, which you can assume is also the circumference of the hole. Thus, if you make the hole in the center of the disk larger, the number of bits per track increases, but the total number of tracks decreases. If you let r denote the radius of the platter, and $x \cdot r$ the radius of the hole, what value of x maximizes the capacity of the disk?

6.23 ♦

Estimate the average time (in ms) to access a sector on the following disk:

Parameter	Value
Rotational rate	12,000 RPM
$T_{\text{avg seek}}$	3 ms
Average number of sectors/track	500

6.24 ♦♦

Suppose that a 2 MB file consisting of 512-byte logical blocks is stored on a disk drive with the following characteristics:

Parameter	Value
Rotational rate	18,000 RPM
$T_{\text{avg seek}}$	8 ms
Average number of sectors/track	2,000
Surfaces	4
Sector size	512 bytes

For each case below, suppose that a program reads the logical blocks of the file sequentially, one after the other, and that the time to position the head over the first block is $T_{\text{avg seek}} + T_{\text{avg rotation}}$.

- A. *Best case*: Estimate the optimal time (in ms) required to read the file given the best possible mapping of logical blocks to disk sectors (i.e., sequential).
- B. *Random case*: Estimate the time (in ms) required to read the file if blocks are mapped randomly to disk sectors.

6.25 ♦

The following table gives the parameters for a number of different caches. For each cache, fill in the missing fields in the table. Recall that m is the number of physical address bits, C is the cache size (number of data bytes), B is the block size in bytes, E is the associativity, S is the number of cache sets, t is the number of tag bits, s is the number of set index bits, and b is the number of block offset bits.

Cache	m	C	B	E	S	t	s	b
1.	32	1,024	4	4	_____	_____	_____	_____
2.	32	1,024	4	256	_____	_____	_____	_____
3.	32	1,024	8	1	_____	_____	_____	_____
4.	32	1,024	8	128	_____	_____	_____	_____
5.	32	1,024	32	1	_____	_____	_____	_____
6.	32	1,024	32	4	_____	_____	_____	_____

6.26 ♦

The following table gives the parameters for a number of different caches. Your task is to fill in the missing fields in the table. Recall that m is the number of physical address bits, C is the cache size (number of data bytes), B is the block size in bytes, E is the associativity, S is the number of cache sets, t is the number of tag bits, s is the number of set index bits, and b is the number of block offset bits.

Cache	m	C	B	E	S	t	s	b
1.	32	_____	8	1	_____	21	8	3
2.	32	2,048	_____	_____	128	23	7	2
3.	32	1,024	2	8	64	_____	_____	1
4.	32	1,024	_____	2	16	23	4	_____

6.27 ♦

This problem concerns the cache in Practice Problem 6.12.

- A. List all of the hex memory addresses that will hit in set 1.
- B. List all of the hex memory addresses that will hit in set 6.

6.28 ♦♦

This problem concerns the cache in Practice Problem 6.12.

- A. List all of the hex memory addresses that will hit in set 2.

- B. List all of the hex memory addresses that will hit in set 4.
- C. List all of the hex memory addresses that will hit in set 5.
- D. List all of the hex memory addresses that will hit in set 7.

6.29 ♦♦

Suppose we have a system with the following properties:

- The memory is byte addressable.
- Memory accesses are to 1-byte words (not to 4-byte words).
- Addresses are 12 bits wide.
- The cache is two-way set associative ($E = 2$), with a 4-byte block size ($B = 4$) and four sets ($S = 4$).

The contents of the cache are as follows, with all addresses, tags, and values given in hexadecimal notation:

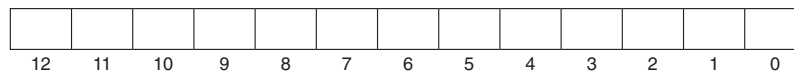
Set index	Tag	Valid	Byte 0	Byte 1	Byte 2	Byte 3
0	00	1	40	41	42	43
	83	1	FE	97	CC	D0
1	00	1	44	45	46	47
	83	0	—	—	—	—
2	00	1	48	49	4A	4B
	40	0	—	—	—	—
3	FF	1	9A	C0	03	FF
	00	0	—	—	—	—

- A. The following diagram shows the format of an address (1 bit per box). Indicate (by labeling the diagram) the fields that would be used to determine the following:

CO. The cache block offset

CI. The cache set index

CT. The cache tag



- B. For each of the following memory accesses, indicate if it will be a cache hit or miss when *carried out in sequence* as listed. Also give the value of a read if it can be inferred from the information in the cache.

Operation	Address	Hit?	Read value (or unknown)
Read	0x834	_____	_____
Write	0x836	_____	_____
Read	0xFFD	_____	_____

6.30 ♦

Suppose we have a system with the following properties:

- The memory is byte addressable.
- Memory accesses are to 1-byte words (not to 4-byte words).
- Addresses are 13 bits wide.
- The cache is 4-way set associative ($E = 4$), with a 4-byte block size ($B = 4$) and eight sets ($S = 8$).

Consider the following cache state. All addresses, tags, and values are given in hexadecimal format. The Index column contains the set index for each set of four lines. The Tag columns contain the tag value for each line. The V columns contain the valid bit for each line. The Bytes 0–3 columns contain the data for each line, numbered left to right starting with byte 0 on the left.

4-way set associative cache												
Index	Tag	V	Bytes 0–3	Tag	V	Bytes 0–3	Tag	V	Bytes 0–3	Tag	V	Bytes 0–3
0	F0	1	ED 32 0A A2	8A	1	BF 80 1D FC	14	1	EF 09 86 2A	BC	0	25 44 6F 1A
1	BC	0	03 3E CD 38	A0	0	16 7B ED 5A	BC	1	8E 4C DF 18	E4	1	FB B7 12 02
2	BC	1	54 9E 1E FA	B6	1	DC 81 B2 14	00	0	B6 1F 7B 44	74	0	10 F5 B8 2E
3	BE	0	2F 7E 3D A8	C0	1	27 95 A4 74	C4	0	07 11 6B D8	BC	0	C7 B7 AF C2
4	7E	1	32 21 1C 2C	8A	1	22 C2 DC 34	BC	1	BA DD 37 D8	DC	0	E7 A2 39 BA
5	98	0	A9 76 2B EE	54	0	BC 91 D5 92	98	1	80 BA 9B F6	BC	1	48 16 81 0A
6	38	0	5D 4D F7 DA	BC	1	69 C2 8C 74	8A	1	A8 CE 7F DA	38	1	FA 93 EB 48
7	8A	1	04 2A 32 6A	9E	0	B1 86 56 0E	CC	1	96 30 47 F2	BC	1	F8 1D 42 30

- A. What is the size (C) of this cache in bytes?
- B. The box that follows shows the format of an address (1 bit per box). Indicate (by labeling the diagram) the fields that would be used to determine the following:
- CO. The cache block offset
- CI. The cache set index
- CT. The cache tag



6.31 ♦♦

Suppose that a program using the cache in Problem 6.30 references the 1-byte word at address 0x071A. Indicate the cache entry accessed and the cache byte value returned *in hex*. Indicate whether a cache miss occurs. If there is a cache miss, enter “—” for “Cache byte returned.” *Hint:* Pay attention to those valid bits!

A. Address format (1 bit per box):

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

B. Memory reference:

Parameter	Value
Block offset (CO)	0x_____
Index (CI)	0x_____
Cache tag (CT)	0x_____
Cache hit? (Y/N)	_____
Cache byte returned	0x_____

6.32 ♦♦

Repeat Problem 6.31 for memory address 0x16E8.

A. Address format (1 bit per box):

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

B. Memory reference:

Parameter	Value
Cache offset (CO)	0x_____
Cache index (CI)	0x_____
Cache tag (CT)	0x_____
Cache hit? (Y/N)	_____
Cache byte returned	0x_____

6.33 ♦♦

For the cache in Problem 6.30, list the eight memory addresses (in hex) that will hit in set 2.

6.34 ♦♦

Consider the following matrix transpose routine:

```

1  typedef int array[4][4];
2
3  void transpose2(array dst, array src)
4  {
5      int i, j;
6  
```

```
7      for (i = 0; i < 4; i++) {
8          for (j = 0; j < 4; j++) {
9              dst[j][i] = src[i][j];
10         }
11     }
12 }
```

Assume this code runs on a machine with the following properties:

- `sizeof(int) = 4`.
- The `src` array starts at address 0 and the `dst` array starts at address 64 (decimal).
- There is a single L1 data cache that is direct-mapped, write-through, write-allocate, with a block size of 16 bytes.
- The cache has a total size of 32 data bytes, and the cache is initially empty.
- Accesses to the `src` and `dst` arrays are the only sources of read and write misses, respectively.

A. For each row and col, indicate whether the access to `src[row][col]` and `dst[row][col]` is a hit (h) or a miss (m). For example, reading `src[0][0]` is a miss and writing `dst[0][0]` is also a miss.

dst array					src array				
	Col. 0	Col. 1	Col. 2	Col. 3		Col. 0	Col. 1	Col. 2	Col. 3
Row 0	m	_____	_____	_____	Row 0	m	_____	_____	_____
Row 1	_____	_____	_____	_____	Row 1	_____	_____	_____	_____
Row 2	_____	_____	_____	_____	Row 2	_____	_____	_____	_____
Row 3	_____	_____	_____	_____	Row 3	_____	_____	_____	_____

6.35 ♦♦

Repeat Problem 6.34 for a cache with a total size of 128 data bytes.

dst array					src array				
	Col. 0	Col. 1	Col. 2	Col. 3		Col. 0	Col. 1	Col. 2	Col. 3
Row 0	_____	_____	_____	_____	Row 0	_____	_____	_____	_____
Row 1	_____	_____	_____	_____	Row 1	_____	_____	_____	_____
Row 2	_____	_____	_____	_____	Row 2	_____	_____	_____	_____
Row 3	_____	_____	_____	_____	Row 3	_____	_____	_____	_____

6.36 ♦♦

This problem tests your ability to predict the cache behavior of C code. You are given the following code to analyze:

```
1      int x[2][128];
2      int i;
```

```

3      int sum = 0;
4
5      for (i = 0; i < 128; i++) {
6          sum += x[0][i] * x[1][i];
7      }

```

Assume we execute this under the following conditions:

- `sizeof(int) = 4`.
- Array `x` begins at memory address `0x0` and is stored in row-major order.
- In each case below, the cache is initially empty.
- The only memory accesses are to the entries of the array `x`. All other variables are stored in registers.

Given these assumptions, estimate the miss rates for the following cases:

- Case 1: Assume the cache is 512 bytes, direct-mapped, with 16-byte cache blocks. What is the miss rate?
- Case 2: What is the miss rate if we double the cache size to 1,024 bytes?
- Case 3: Now assume the cache is 512 bytes, two-way set associative using an LRU replacement policy, with 16-byte cache blocks. What is the cache miss rate?
- For case 3, will a larger cache size help to reduce the miss rate? Why or why not?
- For case 3, will a larger block size help to reduce the miss rate? Why or why not?

6.37 ♦♦

This is another problem that tests your ability to analyze the cache behavior of C code. Assume we execute the three summation functions in Figure 6.47 under the following conditions:

- `sizeof(int) = 4`.
- The machine has a 4 KB direct-mapped cache with a 16-byte block size.
- Within the two loops, the code uses memory accesses only for the array data. The loop indices and the value `sum` are held in registers.
- Array `a` is stored starting at memory address `0x08000000`.

Fill in the table for the approximate cache miss rate for the two cases $N = 64$ and $N = 60$.

Function	$N = 64$	$N = 60$
sumA	_____	_____
sumB	_____	_____
sumC	_____	_____

```

1  typedef int array_t[N][N];
2
3  int sumA(array_t a)
4  {
5      int i, j;
6      int sum = 0;
7      for (i = 0; i < N; i++)
8          for (j = 0; j < N; j++) {
9              sum += a[i][j];
10         }
11     return sum;
12 }
13
14 int sumB(array_t a)
15 {
16     int i, j;
17     int sum = 0;
18     for (j = 0; j < N; j++)
19         for (i = 0; i < N; i++) {
20             sum += a[i][j];
21         }
22     return sum;
23 }
24
25 int sumC(array_t a)
26 {
27     int i, j;
28     int sum = 0;
29     for (j = 0; j < N; j+=2)
30         for (i = 0; i < N; i+=2) {
31             sum += (a[i][j] + a[i+1][j]
32                  + a[i][j+1] + a[i+1][j+1]);
33         }
34     return sum;
35 }

```

Figure 6.47 Functions referenced in Problem 6.37.

6.38 ♦

3M decides to make Post-its by printing yellow squares on white pieces of paper. As part of the printing process, they need to set the CMYK (cyan, magenta, yellow, black) value for every point in the square. 3M hires you to determine the efficiency of the following algorithms on a machine with a 1,024-byte direct-mapped data cache with 16-byte blocks. You are given the following definitions:


```

1  struct point_color {
2      int c;
3      int m;
4      int y;
5      int k;
6  };
7
8  struct point_color square[16][16];
9  int i, j;

```

Assume the following:

- `sizeof(int) = 4`.
- `square` begins at memory address 0.
- The cache is initially empty.
- The only memory accesses are to the entries of the array `square`. Variables `i` and `j` are stored in registers.

Determine the cache performance of the following code:

```

1      for (i = 15; i >= 0; i--){
2          for (j = 15; j >= 0; j--) {
3              square[i][j].c = 0;
4              square[i][j].m = 0;
5              square[i][j].y = 1;
6              square[i][j].k = 0;
7          }
8      }

```

- What is the total number of writes?
- What is the total number of writes that hit in the cache?
- What is the hit rate?

6.39 ♦

Given the assumptions in Problem 6.38, determine the cache performance of the following code:

```

1      for (i = 15; i >= 0; i--){
2          for (j = 15; j >= 0; j--) {
3              square[j][i].c = 0;
4              square[j][i].m = 0;
5              square[j][i].y = 1;
6              square[j][i].k = 0;
7          }
8      }

```

- A. What is the total number of writes?
- B. What is the total number of writes that hit in the cache?
- C. What is the hit rate?

6.40 ♦

Given the assumptions in Problem 6.38, determine the cache performance of the following code:

```

1      for (i = 15; i >= 0; i--) {
2          for (j = 15; j >= 0; j--) {
3              square[i][j].y = 1;
4          }
5      }
6      for (i = 15; i >= 0; i--) {
7          for (j = 15; j >= 0; j--) {
8              square[i][j].c = 0;
9              square[i][j].m = 0;
10             square[i][j].k = 0;
11         }
12     }
```

- A. What is the total number of writes?
- B. What is the total number of writes that hit in the cache?
- C. What is the hit rate?

6.41 ♦♦

You are writing a new 3D game that you hope will earn you fame and fortune. You are currently working on a function to blank the screen buffer before drawing the next frame. The screen you are working with is a 640×480 array of pixels. The machine you are working on has a 32 KB direct-mapped cache with 8-byte lines. The C structures you are using are as follows:

```

1  struct pixel {
2      char r;
3      char g;
4      char b;
5      char a;
6  };
7
8  struct pixel buffer[480][640];
9  int i, j;
10 char *cptr;
11 int *iptr;
```

Assume the following:

- $\text{sizeof(char)} = 1$ and $\text{sizeof(int)} = 4$.

- buffer begins at memory address 0.
- The cache is initially empty.
- The only memory accesses are to the entries of the array buffer. Variables i , j , $cptr$, and $iptr$ are stored in registers.

What percentage of writes in the following code will hit in the cache?

```

1      for (j = 639; j >= 0; j--) {
2          for (i = 479; i >= 0; i--){
3              buffer[i][j].r = 0;
4              buffer[i][j].g = 0;
5              buffer[i][j].b = 0;
6              buffer[i][j].a = 0;
7          }
8      }
```

6.42 ♦♦

Given the assumptions in Problem 6.41, what percentage of writes in the following code will hit in the cache?

```

1      char *cptr = (char *) buffer;
2      for (; cptr < (((char *) buffer) + 640 * 480 * 4); cptr++)
3          *cptr = 0;
```

6.43 ♦♦

Given the assumptions in Problem 6.41, what percentage of writes in the following code will hit in the cache?

```

1      int *iptr = (int *)buffer;
2      for (; iptr < ((int *)buffer + 640*480); iptr++)
3          *iptr = 0;
```

6.44 ♦♦♦

Download the mountain program from the CS:APP Web site and run it on your favorite PC/Linux system. Use the results to estimate the sizes of the caches on your system.

6.45 ♦♦♦♦

In this assignment, you will apply the concepts you learned in Chapters 5 and 6 to the problem of optimizing code for a memory-intensive application. Consider a procedure to copy and transpose the elements of an $N \times N$ matrix of type `int`. That is, for source matrix S and destination matrix D , we want to copy each element $s_{i,j}$ to $d_{j,i}$. This code can be written with a simple loop,

```

1  void transpose(int *dst, int *src, int dim)
2  {
3      int i, j;
4
```

```

5      for (i = 0; i < dim; i++)
6          for (j = 0; j < dim; j++)
7              dst[j*dim + i] = src[i*dim + j];
8  }
```

where the arguments to the procedure are pointers to the destination (`dst`) and source (`src`) matrices, as well as the matrix size N (`dim`). Your job is to devise a transpose routine that runs as fast as possible.

6.46 ♦♦♦♦

This assignment is an intriguing variation of Problem 6.45. Consider the problem of converting a directed graph g into its undirected counterpart g' . The graph g' has an edge from vertex u to vertex v if and only if there is an edge from u to v or from v to u in the original graph g . The graph g is represented by its *adjacency matrix* G as follows. If N is the number of vertices in g , then G is an $N \times N$ matrix and its entries are all either 0 or 1. Suppose the vertices of g are named $v_0, v_1, v_2, \dots, v_{N-1}$. Then $G[i][j]$ is 1 if there is an edge from v_i to v_j and is 0 otherwise. Observe that the elements on the diagonal of an adjacency matrix are always 1 and that the adjacency matrix of an undirected graph is symmetric. This code can be written with a simple loop:

```

1  void col_convert(int *G, int dim) {
2      int i, j;
3
4      for (i = 0; i < dim; i++)
5          for (j = 0; j < dim; j++)
6              G[j*dim + i] = G[j*dim + i] || G[i*dim + j];
7  }
```

Your job is to devise a conversion routine that runs as fast as possible. As before, you will need to apply concepts you learned in Chapters 5 and 6 to come up with a good solution.

Solutions to Practice Problems

Solution to Problem 6.1 (page 620)

The idea here is to minimize the number of address bits by minimizing the aspect ratio $\max(r, c) / \min(r, c)$. In other words, the squarer the array, the fewer the address bits.

Organization	r	c	b_r	b_c	$\max(b_r, b_c)$
16×1	4	4	2	2	2
16×4	4	4	2	2	2
128×8	16	8	4	3	4
512×4	32	16	5	4	5
$1,024 \times 4$	32	32	5	5	5