Quiz 1

Algorithms, Data Structures, Caches, Memory Allocation, and Performance Analysis

Given: In class on Thursday, October 14, 2010

Name: ______________________

CSAIL username: ________________

Instructions

• DO NOT open this quiz booklet until you are instructed to do so.

• This quiz booklet contains 15 pages. You have 80 minutes to earn 100 points.

• This quiz is closed book, but you may use one handwritten, double-sided 8 1/2" × 11" crib sheet.

• When the quiz begins, please write your name and username on this coversheet, and write your name on the top of each page, since the pages may be separated for grading.

• Good luck!

<table>
<thead>
<tr>
<th>Question</th>
<th>Parts</th>
<th>Points</th>
<th>Score</th>
<th>Grader</th>
</tr>
</thead>
<tbody>
<tr>
<td>True or False</td>
<td>5</td>
<td>20</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Optimizing memcmp() (multiple choice)</td>
<td>2</td>
<td>10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cache-Miss Types (multiple choice)</td>
<td>3</td>
<td>9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cache Analysis</td>
<td>7</td>
<td>31</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Short Questions</td>
<td>10</td>
<td>30</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td></td>
<td><strong>100</strong></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
1 True or False

Indicate whether each statement below is true or false by circling the correct answer, and justify your answer in at most one or two sentences. Your justification is worth more than your true-false designation.

1.1 4 points

Although tiling can be an effective way to reduce cache misses, it is not a cache-oblivious technique, and thus may not work well in a multiprogrammed environment.

TRUE  FALSE

1.2 4 points

A heap-storage allocator might implement coalescing in order to reduce external fragmentation.

TRUE  FALSE
1.3 4 points
The branch-misprediction rate is the number of mispredicted branches in the program.

**TRUE FALSE**

1.4 4 points
Suppose that an ideal cache has size $M$ and cache-line length $B$. An $n \times n$ matrix, where $n^2 < cM$ for a sufficiently small constant $c \leq 1$, is guaranteed to require no more than $\Theta(n^2 / B)$ cache misses to load into cache.

**TRUE FALSE**

1.5 4 points
A garbage-collected language such as Java leaves the programmer prone to programming errors such as dangling pointers, memory leaks, and double frees.

**TRUE FALSE**
2 Optimizing `memcmp()` — multiple choice

2.1 5 points

Consider the function `my_memcmp()` below, which is similar to the `memcmp()` function found in the standard C library, and an optimized version, called `my_memcmp_opt1()`:

```c
/* Return 0 if the first n bytes of s1 and s2 are equal, and return 1 otherwise. */
int my_memcmp(char *a, char *b, size_t n) {
    for (size_t i = 0; i < n; i++)
        if (a[i] != b[i])
            return 1;
    return 0;
}

int my_memcmp_opt1(char *a, char *b, size_t n) {
    uint64_t *aw = (uint64_t *) a;
    uint64_t *bw = (uint64_t *) b;
    size_t iw = 0;
    for (; iw < n / sizeof(uint64_t); iw++)
        if (aw[iw] != bw[iw])
            return 1;
    size_t i = iw * sizeof(uint64_t);
    for (; i < n; i++)
        if (a[i] != b[i])
            return 1;
    return 0;
}
```

Which one of the following alternatives best describes the kind of optimization performed going from `my_memcmp()` to `my_memcmp_opt1()`? Circle the letter next to the correct alternative.

A Common-subexpression elimination
B Loop fusion
C Compression
D Partial loop unrolling
E Single instruction, multiple data (SIMD)
2.2 5 points

Now consider the same function `my_memcmp()` (printed again below) and a version with a different optimization, the `my_memcmp_opt2()` function shown below.

```c
/* Return 0 if the first n bytes of s1 and s2 are equal, and return 1 otherwise. */
int my_memcmp(char *a, char *b, size_t n) {
    for (size_t i = 0; i < n; i++)
        if (a[i] != b[i])
            return 1;
    return 0;
}

int my_memcmp_opt2(char *a, char *b, size_t n) {
    if (n == 0)
        return 0;
    size_t i;
    char oldanm1 = a[n-1];
    a[n-1] = !b[n-1];
    for (i = 0;; i++) {
        if (a[i] != b[i])
            break;
    }
    a[n-1] = oldanm1;
    return a[i] != b[i];
}
```

Which one of the following alternatives best describes the kind of optimization performed going from `my_memcmp()` to `my_memcmp_opt2()`? Circle the letter next to the correct alternative.

A Data-structure augmentation
B Inlining
C Lazy evaluation
D Sentinel loop-exit test
E Test reordering
3 Cache-Miss Types — multiple choice

For the following three functions, determine the dominant kind of L1-cache misses incurred, and explain what is meant by that kind of cache miss. Assume that the functions are run on a computer with a 32 KB 4-way set-associative L1-cache with a least-recently-used (LRU) eviction policy. Circle the letter next to the one correct alternative for each question.

3.1 3 points

```c
int func1(char *buf) {
    int ret = 0;
    for (size_t i = 0; i < 4096*1024; i++)
        ret += buf[i];
    return ret;
}
```

What kind of L1-cache miss is most frequently incurred when running this function on the machine described above?

A Cold miss  
B Capacity miss  
C Conflict miss  
D True-sharing miss  
E False-sharing miss

3.2 3 points

```c
int func2(char *buf) {
    int ret = 0;
    for (size_t i = 0; i < 4096*1024; i++)
        ret += buf[i % (512*1024)];
    return ret;
}
```

What kind of L1-cache miss is most frequently incurred when running this function on the machine described above?

A Cold miss  
B Capacity miss  
C Conflict miss  
D True-sharing miss  
E False-sharing miss
3.3  3 points

```c
int func3(char *buf) {
    int ret = 0;
    for (size_t i = 0; i < 4096*1024; i++)
        ret += buf[(i % 8) * 8192];
    return ret;
}
```

What kind of L1-cache miss is most frequently incurred when running this function on the machine described above?

A  Cold miss
B  Capacity miss
C  Conflict miss
D  True-sharing miss
E  False-sharing miss
4 Cache Analysis

Consider the following C function paircount() which takes a length-n array A of integers and determines the number of ordered pairs of elements in A that satisfy an input predicate P():

```c
int paircount(int (*P)(int, int), int *A, int n) {
    int count = 0;
    for (int i = 0; i < n; i++)
        for(int j = 0; j < n; j++)
            count += P(A[i], A[j]);
    return count;
}
```

Assume that this and other code in this quiz problem runs on a machine with an ideal cache of size \( M \ll n \) with block size \( B \). Further assume that an execution of the predicate \( P(x, y) \) incurs no cache misses beyond those possibly required to read \( x \) and \( y \).

4.1 3 points

Analyze the asymptotic number of cache misses incurred by paircount().
Now consider the following function `paircount_opt()`, which implements a divide-and-conquer solution to the same problem using an auxiliary function `paircount_opt_aux()`:

```c
static int paircount_opt_aux(int (*P)(int, int), bool swapped, int *A, int na, int *B, int nb) {
    if (nb > na)
        return paircount_opt_aux(P, !swapped, B, nb, A, na);
    if (na == 1) {
        int count = 0;
        for (int i = 0; i < nb; i++)
            count += (swapped) ? P(B[i], A[0]) : P(A[0], B[i]);
        return count;
    } else {
        return paircount_opt_aux(P, swapped, A, na/2, B, nb) +
            paircount_opt_aux(P, swapped, A + na/2, na - na/2, B, nb);
    }
}

int paircount_opt(int (*P)(int, int), int *A, int n) {
    if (n == 0)
        return 0;
    return paircount_opt_aux(P, false, A, n, A, n);
}
```

4.2 6 points

Let $Q(r)$ be the number of cache misses incurred during the execution of a call to `paircount_opt_aux()`, where $r = na \times nb$. Argue that $Q(r)$ satisfies the recurrence

$$Q(r) \leq \begin{cases} 
\sqrt{r}/B & \text{if } \sqrt{r} \leq cM \\
2Q(r/2) + \Theta(1) & \text{otherwise.}
\end{cases}$$
4.3 10 points

Draw a recursion tree for the recurrence. Compute the height and the number of leaves of the recursion tree, and label them on your diagram.
4.4 4 points
Solve the recurrence for $Q(r)$ by providing a tight asymptotic upper bound.

4.5 4 points
Under what circumstances should `paircount()` outperform `paircount_opt()`?

4.6 4 points
State one optimization to the code for `paircount_opt_aux()` which would significantly improve its running time in practice.
5 Short Questions — multiple choice

This section of the quiz contains 10 multiple-choice questions. Follow the directions provided in each question, and choose one answer for each question below. Clearly mark your answer by circling the most-appropriate letter.

5.1 3 points
Which of the following techniques is most likely to significantly improve (reduce) the CPI of an executing program?
A Eliminate cache misses
B Use Intel SSE (Streaming SIMD Extensions) instructions
C Unroll loops
D Inline functions
E Eliminate common subexpressions

5.2 3 points
Which of the following does not contribute significantly to the fast development of reliable code?
A pair programming
B short variable names
C regression testing
D unit testing
E code reviews

5.3 3 points
Which of the following techniques is not employed by a typical copying garbage collector?
A Forwarding pointers
B Breadth-first search
C FROM space and TO space
D FIFO queue
E None of the above
5.4 3 points

Below is a fragment of compiler-optimized (gcc -Os) assembly that implements one of the four C functions listed as answers. The test instruction sets the flag bits with the result of AND-ing the two register operands together. The jns instruction jumps to the specified address if the signed flag is not set, meaning the tested value is nonnegative.

Of the functions above, which one does the assembly code implement?

A int foo(int n)
B int bar(int n)
C int baz(int n)
D int qux(int n)
E None of the above
5.5 3 points

Your are reading your project partner’s code and come across the C function given below.

```c
int mxyzptlk(int x, int y) {
    return x ^ ((x ^ y) & -(x < y));
}
```

What does the function mxyzptlk do?

A Returns the minimum of x and y without branching.
B Returns 1 if x is divisible with y, and 0 otherwise.
C Returns the maximum of x and y without branching.
D Returns the number of set bits shared by x and y.
E None of the above

5.6 3 points

Reading further down, you come across another function:

```c
uint64_t kltpzyxm(uint64_t n) {
    n--; 
    n |= n >> 1;
    n |= n >> 2;
    n |= n >> 4;
    n |= n >> 8;
    n |= n >> 16;
    n |= n >> 32;
    return n+1;
}
```

What does the function kltpzyxm() do?

A If n has k bits that are 1, computes a mask where the k low-order bits are 1 and the rest are 0
B Computes $2^{\lceil \lg n \rceil}$
C Returns (n != 0)
D Computes the population count of n
E None of the above

5.7 3 points

Which of the following C code fragments evaluates to 0 if the binary representation of the unsigned integer x contains no more than two 1 bits?

A $(x ^ (x + 1)) ^ (x - 1)$;
B $((x >> 2) & (x >> 1)) & x$;
C $(x & (x - 1)) & ((x & (x - 1)) - 1)$;
D $x & !((x & (x - 1))$;
E None of the above
5.8 3 points

Consider the following statements regarding computer architecture:

1. The prefetcher helps to reduce the branch-miss rate.
2. If a branch is correctly predicted, the pipeline will not stall on that branch.
3. Loop unrolling can help reduce data dependencies.

Which of these three statements are true?

A 1 & 2
B 1 & 2 & 3
C 2
D 2 & 3
E None

5.9 3 points

Which of the following performance metrics will likely change the least as a result of tuning the voodoo variables of a cache-aware algorithm?

A Cycles
B L1-cache references
C L1-cache misses
D L3-cache references
E L3-cache misses

5.10 3 points

Consider the following short C program:

```c
int main(int argc, char *argv) {
    uint16_t moo = 42;
    uint16_t *p = &moo;
    printf("%d
", (int) (((int64_t *) p)[3]) - (int64_t *) p));
}
```

What does the program print?

A 3
B 6
C 24
D 42
E Nothing. It seg faults.