Quiz 2

Parallelism, Distributed Systems, and Compiler Optimizations

Given: In class on Thursday, December 2, 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.

• All multiple choice questions are worth 3 points unless otherwise stated.

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

• When the quiz begins, please write your name and username on this coversheet, and place your initials in the top corner of each page.

• 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>Parallelism</td>
<td>4</td>
<td>16</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Compiler optimization</td>
<td>7</td>
<td>17</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Distributed systems</td>
<td>3</td>
<td>15</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cilk</td>
<td>3</td>
<td>12</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Locks</td>
<td>5</td>
<td>30</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Miscellaneous</td>
<td>2</td>
<td>10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>100</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
1 Parallelism (4 parts, 16 points)

Consider the following code snippet, which implements a parallel, in-place matrix transposition operation for an \( n \times n \) matrix \( a \). For the purposes of analysis, assume that the grain size is 1.

```c
void ptranspose(float* a, int n)
{
    cilk_for (int j = 1; j < n; ++j)
        cilk_for (int i = 0; i < j; ++i)
            std::swap(a[i+(j*n)], a[j+(i*n)]);
}
```

1.1 What is the work of this algorithm?

A  \( \Theta(n) \)

B  \( \Theta(n/\lg n) \)

C  \( \Theta(n^2/\lg n) \)

D  \( \Theta(n^2) \)

E  \( \Theta(n^2 \lg n) \)

1.2 What is the span of this algorithm?

A  \( \Theta(\lg n) \)

B  \( \Theta(\lg^2 n) \)

C  \( \Theta(n/\lg n) \)

D  \( \Theta(n^2/\lg n) \)

1.3 Suppose that we modify this function so that the inner loop is a \texttt{for} loop instead of a \texttt{cilk_for} loop, and leave the grain size fixed at 1.

What is the work of the modified algorithm?

A  \( \Theta(n) \)

B  \( \Theta(n/\lg n) \)

C  \( \Theta(n^2/\lg n) \)

D  \( \Theta(n^2) \)

E  \( \Theta(n^2 \lg n) \)

1.4 What is the parallelism of the modified algorithm?

A  \( \Theta(n/\lg n) \)

B  \( \Theta(n) \)

C  \( \Theta(n^2/\lg n) \)

D  \( \Theta(n^2/\lg^2 n) \)

E  \( \Theta(n^2) \)
2 Compiler optimization (7 parts, 17 points)

Your boss tells you to perform various optimizations to bottleneck functions in your flagship program. Since you took 6.172, you know how to work with an optimizing compiler.

For each function, “before” code listing as well as an “after” code listing is provided. For each listing, determine these three key properties:

- **Legal?** Is the transformation legal? Will the “after” version have the same externally visible behavior as the “before” version? Memory allocated from the heap for which no pointer escapes is not considered to be externally visible.

- **Faster?** With all compiler optimizations turned off, will the “after” version always execute faster than the “before” version? (If it is illegal, just write “N/A”)

- **Automatic?** Can you assume a typical optimizing compiler (for example, gcc -O3) would do this for you? (If it is illegal or slower, just write “N/A.”)

You may ignore floating-point effects and assume they behave like real numbers.

You should speak to each point concisely. Most responses should be one or two sentences.

### 2.1

**Function f before**
```c
double f(double u, double v, int n, double *b) {
    double s;
    for (int i = 0; i < n; i++) {
        s += b[1]*u + b[2]*v;
    }
    return s;
}
```

**Function f after**
```c
double f(double u, double v, int n, double *b) {
    double s, x, y;
    x = b[1];
    y = b[2];
    for (int i = 0; i < n; i++) {
        s += x*u + y*v;
    }
    return s;
}
```

Is the “after” version of function f:

- **Legal?**

- **Faster?**

- **Automatic?**

Explain your answers as appropriate.
2.2

*Function fill before*

```c
void fill(int *a, int *b, int *output) {
    for (int i = 0; i < N; i += 2) {
        output[i] = *a;
        output[i + 1] = *b;
    }
}
```

*Function fill after*

```c
void fill(int *a, int *b, int *output) {
    int reg_a = *a;
    int reg_b = *b;
    for (int i = 0; i < N; i += 2) {
        output[i] = reg_a;
        output[i + 1] = reg_b;
    }
}
```

Is the “after” version of function fill:

- Legal?
- Faster?
- Automatic?

Explain your answers as appropriate.

2.3

*Function g before*

```c
double g(double t) {
    double s, c;
    s = sin(t); c = cos(t);
    return (s*s + c*c);
}
```

*Function g after*

```c
double g(double t) {
    return 1.0;
}
```

Assume that sin and cos behave like the math functions in libc.

Is the “after” version of function g:

- Legal?
- Faster?
- Automatic?

Explain your answers as appropriate.
Function $h$ before

```c
static void mod(unsigned *q, unsigned d) {
    *q = *q % d;
}

unsigned h(unsigned q) {
    mod(&q, 16);
    return q;
}
```

Function $h$ after

```c
unsigned h(unsigned q) {
    return q & 0x0F;
}
```

Is the “after” version of function $h$:

- **Legal?**
- **Faster?**
- **Automatic?**

Explain your answers as appropriate.

Function $h$ before

```c
int x = 0;

int h(int y) {
    return x*y;
}
```

Function $h$ after

```c
int x = 0;

int h(int y) {
    return 0;
}
```

Is the “after” version of function $h$:

- **Legal?**
- **Faster?**
- **Automatic?**

Explain your answers as appropriate.
2.6

<table>
<thead>
<tr>
<th>Function e before</th>
<th>Function e after</th>
</tr>
</thead>
<tbody>
<tr>
<td>int e(unsigned int n) {</td>
<td>int e(unsigned int n) {</td>
</tr>
<tr>
<td>int sum = 0;</td>
<td>return (n * (n-1))/2;</td>
</tr>
<tr>
<td>unsigned int i;</td>
<td></td>
</tr>
<tr>
<td>for (i = 0; i &lt; n; i++)</td>
<td></td>
</tr>
<tr>
<td>sum += i;</td>
<td></td>
</tr>
<tr>
<td>return sum;</td>
<td></td>
</tr>
<tr>
<td>}</td>
<td>}</td>
</tr>
</tbody>
</table>

Is the “after” version of function e:

• Legal?

• Faster?

• Automatic?

Explain your answers as appropriate.

2.7

<table>
<thead>
<tr>
<th>Function make_new_array before</th>
<th>Function make_new_array after</th>
</tr>
</thead>
<tbody>
<tr>
<td>int* make_new_array(size_t len) {</td>
<td>int* make_new_array(size_t len) {</td>
</tr>
<tr>
<td>int* ptr = (int*) malloc(1);</td>
<td>int* ptr = (int*) malloc(len);</td>
</tr>
<tr>
<td>for (int i = 2; i &lt; len + 1; i++)</td>
<td>return ptr;</td>
</tr>
<tr>
<td>ptr = realloc(ptr, i);</td>
<td></td>
</tr>
<tr>
<td>return ptr;</td>
<td></td>
</tr>
<tr>
<td>}</td>
<td>}</td>
</tr>
</tbody>
</table>

Is the “after” version of function make_new_array:

• Legal?

• Faster?

• Automatic?

Explain your answers as appropriate.
3 Distributed systems (3 parts, 15 points)

3.1
You and a coworker are arguing about distributed-system message passing. Recall that a blocking operation suspends the caller until the operation completes, whereas a nonblocking operation allows the caller to continue while the operation executes. Your coworker argues that if a system using blocking sends and receives has a deadlock, it can be resolved by simply switching to nonblocking sends and receives. Show that your coworker is incorrect by constructing a counterexample.

3.2
In Google’s MapReduce framework, there are two phases called map and reduce. In the map phase, the data is split evenly among the workers, and each worker produces a number of key-value pairs. How are the key-value pairs distributed to the workers during the reduce phase?

A Evenly across all workers
B Each worker from the mapping step reprocesses its own output
C All key-value pairs are sorted by key, and partitioned evenly across the workers
D Each worker is assigned one or more keys, and all the data with a given key is sent to the matching worker
3.3
You find yourself tasked with improving the performance of a distributed application designed to deliver a video stream to a sizable number of clients. After an appropriate high-level analysis of the application, you discover that it is heavily network-bound on the server side, which is causing packets to be dropped. Which of the following techniques would prove to be most useful for solving the problem?

A. MapReduce
B. Cilk reducers
C. A content distribution network (CDN)
D. OpenMPI
E. Fractal trees

4. Cilk (3 parts, 12 points)

4.1
Ben Bitdiddle runs his Cilk++ program with 8, 16, and 32 worker threads using `-cilk_set_workers` on a 16-core system, and he notices that the program runs in 20 seconds, 10 seconds, and 20 seconds respectively. Why might his program slow down when he adds more workers than processors? Circle all that apply.

A. Extra context switching between workers
B. The cache is split across multiple workers
C. Each new worker thread requires a large amount of memory for its stack and other data, which exhausts main memory and causes the application to start swapping
D. As the workers modify their address space, the kernel performs copy-on-write, which slows them down

4.2
When building a list in parallel, why would you prefer using a reducer to using locks? Circle all that apply.

A. Reducers eliminate the contention issues endemic to lock-based solutions.
B. No synchronization instructions are needed for every list append
C. The reduce function eliminates the concerns over thread safety that a locking implementation must deal with
D. The reducer ensures that the end result is always the same as the serial execution.
4.3

The following code performs a parallel summation of an array. In this case, \texttt{reducer\_opadd} is operating over integers using the associative reducer operation of addition with identity 0. Suppose that the reducer were mistakenly to use 1 for the “identity,” rather than 0. Then, the actual result would be slightly larger than the correct result. What would the difference between the actual and correct results be?

```cpp
#include "reducer_opadd.h"

int sum(int *arr, int n) {
    cilk::reducer_opadd<int> r;
    cilk_for (int i = 0; i < n; ++i) {
        r += arr[i];
    }
    return r.get_value();
}
```

A. The number of times work was stolen by one worker from another  
B. The number of workers  
C. The number of spawns executed internally by the \texttt{cilk\_for}  
D. The number of iterations, i.e., n
5 Locks (5 parts, 30 points)

Below we show several implementations of a function to transfer money between two bank account data structures. The program contains multiple threads which may execute the function in parallel on a collection of bank accounts. For each of the implementations, indicate whether the program may deadlock, whether it has a data race, and whether the result will always be correct if the program ends (in cases where no actual deadlock occurs).

5.1

```c
void account_transfer_v1(account_t *acct_from, account_t *acct_to, long amount) {
    lock(acct_from->lock);
    acct_from->balance -= amount;
    acct_to->balance += amount;
    unlock(acct_from->lock);
}
```

A  Deadlock may occur  Yes  No
B  Data race may occur  Yes  No
C  Always correct if program ends  Yes  No

5.2

```c
void account_transfer_v2(account_t *acct_from, account_t *acct_to, long amount) {
    lock(acct_from->lock);
    acct_from->balance -= amount;
    unlock(acct_from->lock);
    lock(acct_to->lock);
    acct_to->balance += amount;
    unlock(acct_to->lock);
}
```

A  Deadlock may occur  Yes  No
B  Data race may occur  Yes  No
C  Always correct if program ends  Yes  No
5.3

```c
void account_transfer_v3(account_t *acct_from, account_t *acct_to, long amount) {
    lock(acct_from->lock);
    lock(acct_to->lock);
    acct_from->balance -= amount;
    acct_to ->balance += amount;
    unlock(acct_from->lock);
    unlock(acct_to->lock);
}
```

A  Deadlock may occur  Yes  No
B  Data race may occur  Yes  No
C  Always correct if program ends  Yes  No

5.4

Consider an application with two threads running the two loops below. Assume the locks in this example are not spin locks, but yielding locks.

```c
while (true) {
    acquire(data_lock);
    // Update the state of the world. Takes a bit of time.
    release(data_lock);
}
...
while (true) {
    msg_t msg = recv_msg(); // Blocks for a long time.
    acquire(data_lock);
    // Update the state of the world with msg.
    release(data_lock);
}
```

What problem does this code exhibit, and what is it called?

A  Data dependence
B  Starvation
C  Livelock
D  Deadlock
E  Race condition
In the Dining Philosophers problem, each of $n$ philosophers needs the two chopsticks on either side of his/her plate to eat his/her noodles.

Which of the following rules can be used by the philosophers to guarantee that deadlock does not occur?

A  Always pick up the left chopstick before picking up the right chopstick.
B  Wait for a random amount of time, then pick up the left chopstick, then pick up the right chopstick.
C  Choose the first chopstick to pick up at random.
D  Assign each chopstick a distinct number, and always pick up the one with the lowest number first.
E  Alternate between picking up the left chopstick and the right chopstick first.
5.6

In an attempt to optimize the solution to the Dining Philosophers problem (see the previous question), we might introduce the following rule: Once a philosopher has picked up the first chopstick, he/she will look to see if the second chopstick is available, and only pick that one up if this is the case (a try-lock operation). If the second chopstick is already taken, the philosopher will put down the first chopstick without eating, and restart the process (rather than waiting for the second chopstick to become available while holding the first).

A  The philosophers may deadlock.
B  Some philosophers may starve without deadlock.
C  Convoying may occur.
D  All philosophers eventually eat.
6 Miscellaneous (2 parts, 10 points)

6.1

C++ supports passing objects by value, where instead of passing around a reference or pointer to the object, the object is copied every time it is reassigned. Parameter-passing by value can be useful for allowing fine-grained control over memory management. Moreover, for some objects like pointer-size pairs, it can be faster than passing around pointers. Unfortunately, inefficiency can result when code falls into the trap of performing unnecessary copies. A classic example is unnecessarily copying a `std::vector`, which causes a new internal buffer to be allocated and the contents of the original to be copied.

For each of the commented lines, circle “Copy” to indicate a copy of a `std::vector` is performed, or “No copy” to indicate that no copy occurs. Each answer is worth one point.

```cpp
#include <vector>

using std::vector;

static vector<int> globalVector(10000);

void foo(vector<int> obj) { }
void bar(vector<int> &obj) { }
vector<int> baz() { return globalVector; }
vector<int> &quux() { return globalVector; }

int main() {
    // Create a large vector.
    vector<int> a(10000);
    vector<int> b = a; // Line 1
    vector<int> &c = b; // Line 2
    foo(a); // Line 3
    bar(a); // Line 4
    baz(); // Line 5
    quux(); // Line 6
}
```

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>Line 1</td>
<td>Copy</td>
</tr>
<tr>
<td>B</td>
<td>Line 2</td>
<td>Copy</td>
</tr>
<tr>
<td>C</td>
<td>Line 3</td>
<td>Copy</td>
</tr>
<tr>
<td>D</td>
<td>Line 4</td>
<td>Copy</td>
</tr>
<tr>
<td>E</td>
<td>Line 5</td>
<td>Copy</td>
</tr>
<tr>
<td>F</td>
<td>Line 6</td>
<td>Copy</td>
</tr>
</tbody>
</table>
6.2

What is the result of inserting 8 into the following fractal tree?

- **A**: The result is fractal tree A
- **B**: The result is fractal tree B
- **C**: The result is fractal tree C
- **D**: The result is fractal tree D
- **E**: The result is fractal tree E