Final Examination
ECE 454F 2007: Computer Systems Programming
Date: Dec 21, 2007, 2-4:30 p.m.

Instructor: Cristiana Amza
Department of Electrical and Computer Engineering
University of Toronto

Problem number	Maximum Score	Your Score
1	16	
2	15	
3	12	
4	13	
5	16	
6	14	
7	14	
total	100	

This exam is open textbook and open lecture notes. You have two hours to complete the exam. Use of computing and/or communicating devices is NOT permitted. You should not need any such devices. You can use a basic calculator if you feel it is absolutely necessary.

Work independently. Do not remove any sheets from this test book. Answer all questions in the space provided. No additional sheets are permitted. Scratch space is available at the end of the exam.

Write your name and student number in the space below. Do the same on the top of each sheet of this exam book.

Your Student Number	
Your First Name	
Your Last Name	

Student Number: Name:

Problem 1. Basic Facts. (16 Points)

a. (4 points))

```
typedef struct
{
  char c[3];
} Obj;
```

Assume a program that you are writing is supposed to access N objects in sequence e.g., in a loop. The objects can either be *statically* allocated (as an array of objects) or dynamically allocated (as an array of pointers to objects). Furthermore, the dynamic allocation can use either an *implicit* list allocator or an *explicit* list allocator. Assume an empty heap initially and a first-fit policy for the dynamic allocators. Please order the three approaches above from best to worst in terms of estimated cache behavior of the program. Justify your answer briefly by stating any assumptions about the cache line size and sketching or stating the structure of allocated blocks (you can assume and use any variation of the techniques above learned in class for block allocation).

Student Number: Name:

b. (4 points)) Is there a performance advantage of ticket locks over test-and-test-and-set locks on each of the following architectures:

1) multiprocessors with cache coherence?

2) multiprocessors without hardware support for cache coherence?

c. (4 points)) Name one advantage and one disadvantage, if any, of MCS locks over ticket locks on each of the following architectures:

1) multiprocessors with cache coherence

2) multiprocessors without cache coherence

Student Number: Name:

b. (4 points)) Multithreaded server code scales worse than event-driven server code.

1) State two reasons why the above is true.

2) Considering the above, explain why multithreading the Quake server code (an event driven code) improved the scaling of the server.

Student Number: Name:

Problem 2. Cache Miss Rates. (15 Points)

After a stressful semester you suddenly realize that you haven't bought a single Christmas present yet. Fortunately, you see that one of the big electronic stores has CD's on sale. You don't have much time to decide which CD will make the best presents for which friend, so you decide to automatize the decision process. For that, you use a database containing an entry for each of your friends. It is is implemented as an 8×8 matrix using a data structure `person`. You add to this data structure a field for each CD that you consider:

```
struct person{

  char name[16];

  int age;
  int male;

  short nsync;
  short britney_spears;
  short avril_lavigne;
  short garth_brooks;
}

struct person db[8][8];
register int i, j;
```

Part 1

After thinking for a while you come up with the following smart routine that finds the ideal present for everyone.

```
void generate_presents(){

  for (j=0; j<8; j++){
      for (i=0; i<8; i++) {
          db[i][j].nsync=0;
          db[i][j].britney_spears=0;
          db[i][j].garth_brooks=0;
          db[i][j].avril_lavigne=0;
      }
   }

  for (j=0; j<8; j++){
      for (i=0; i<8; i++) {

          if(db[i][j].age < 30){
             if(db[i][j].male)
```

```
                db[i][j].britney_spears = 1;
            else  db[i][j].nsync = 1;
        }
        else{
            if(db[i][j].male)
                db[i][j].avril_lavigne =1;
            else  db[i][j].garth_brooks = 1;
        }
    }
}

}
```

Of course, runtime is important in this time-critical application, so you decide to analyze the cache performance of your routine. You assume that

- your machine has a 512-byte direct-mapped data cache with 64 byte blocks.
- db begins at memory address 0
- The cache is initially empty.
- The only memory accesses are to the entries of the array db. Variables i, and j are stored in registers.

Answer the following questions:

A. What is the total number of read and write accesses? _____.

B. What is the total number of read and write accesses that miss in the cache? _____ .

C. So the fraction of all accesses that miss in the cache is: _____.

Part 2

Then you consider the following alternative implementation of the same algorithm:

```
void generate_presents(){

  for (i=0; i<8; i++){
      for (j=0; j<8; j++) {

          if(db[i][j].age < 30)
             if(db[i][j].male) {
                db[i][j].nsync=0;
                db[i][j].britney_spears=1;
                db[i][j].garth_brooks=0;
                db[i][j].avril_lavigne=0;
```

```
            }

            else
               db[i][j].nsync=1;
               db[i][j].britney_spears=0;
               db[i][j].garth_brooks=0;
               db[i][j].avril_lavigne=0;
            }
         }

         else{
            if(db[i][j].male) {
               db[i][j].nsync=0;
               db[i][j].britney_spears=0;
               db[i][j].garth_brooks=0;
               db[i][j].avril_lavigne=1;
            }

            else{
               db[i][j].nsync=0;
               db[i][j].britney_spears=0;
               db[i][j].garth_brooks=1;
               db[i][j].avril_lavigne=0;
            }
         }


      }
   }

}
```

Making the same assumptions as in Part 1, answer the following questions.

A. What is the total number of read and write accesses? _____

B. What is the total number of read and write accesses that miss in the cache? _____

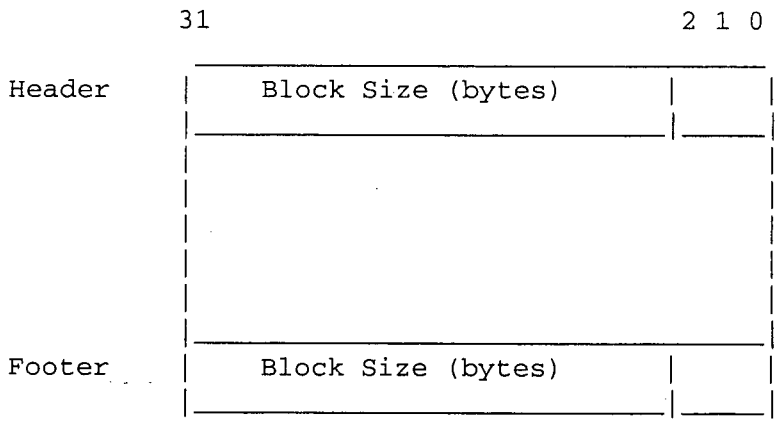
C. So the fraction of all accesses that miss in the cache is: _____.

Student Number: Name:

Problem 3. Dynamic Memory Allocation (12 Points)

Consider an allocator that uses an implicit free list.

The layout of each free memory block is as follows:



Each memory block, either allocated or free, has a size that is a multiple of eight bytes. Thus, only the 29 higher order bits in the header and footer are needed to record block size, which includes the header (and footer for free blocks). The usage of the remaining 3 lower order bits is as follows:

- `bit 0` indicates the use of the current block: 1 for allocated, 0 for free.
- `bit 1` indicates the use of the previous adjacent block: 1 for allocated, 0 for free.
- `bit 2` is unused and is always set to be 0.

Consider the allocator that uses implicit free lists. We make the following space optimization: *Allocated* blocks do not have a footer. *Free* blocks still have the footer and the rest of the block structure is the same as before.

Using the space optimization above and given the contents of the heap shown on the left, we executed `malloc(5) and free(0x400b010)` in sequential order.

Please show the new contents of the heap after `malloc(5)` is executed in the 2nd table, and show the new contents of the heap after `free(0x400b010)` is executed in the 3rd table. Your answers should be given as hex values. Update only the necessary boundary tags. Note that the address grows from bottom up. Assume that the allocator uses a *first fit* allocation policy.

Student Number: Name:

Address	Content
0x400b028	0x00000012
0x400b024	0x400b621c
0x400b020	0x400b620c
0x400b01c	0x00000012
0x400b018	0x400b512c
0x400b014	0x400b511c
0x400b010	0x400b601c
0x400b00c	0x00000011
0x400b008	0x0000000a
0x400b004	0x0000000a
0x400b000	0x400b511c
0x400affc	0x0000000b

Address	Content
0x400b028	
0x400b024	0x400b621c
0x400b020	
0x400b01c	
0x400b018	
0x400b014	
0x400b010	
0x400b00c	
0x400b008	
0x400b004	
0x400b000	
0x400affc	

Student Number: Name:

Address	Content
0x400b028	
0x400b024	0x400b621c
0x400b020	
0x400b01c	
0x400b018	
0x400b014	
0x400b010	
0x400b00c	
0x400b008	
0x400b004	
0x400b000	
0x400affc	

Student Number: Name:

Problem 4. Parallel Programming: Dependencies (13 Points)

Identify the dependences in the following for loop nests, written in pseudocode, specifying for each dependence: the type of the dependence, whether it is loop-dependent or loop-independent and the iterations/statements/variables involved.

Loop 1.

```
for (I = 1; I <=98; I++) {
S1:   A [I] = B [I] + C [I];
S2:   D [I] = A [I + 2];
}
```

Please identify the dependences in loop 1:

Loop 2.

```
for( I = 1; I <= 100; I++){
   for( J = I; J <= I + 19; J++) {
S1:   A [I + 20][J] = A [I][J] + B;
   }
}
```

Please identify the dependences in loop 2:

Student Number: Name:

Loop 3.

```
for( I = 1; I <= 100; I++){
S1:  A [I + 1][I + 2] = A [I][I] + C
}
```

Please identify the dependences in loop 3:

Student Number: Name:

Problem 5. Parallel Programming: Pthreads/OpenMP (16 Points)

Parallelize the following code *efficiently* using either Pthreads-like pseudo-code, as in the lectures, e.g., fork, join, lock, unlock, signal, wait, etc, OpenMP directives, or both. The focus is on conveying all aspects of your full parallelization strategy, assuming you have *all* the parallel constructs you have learned, rather than on respecting rigurous syntax. *Explain* your design decisions, such as thread creation, task to thread assignment and synchronization.

You can assume that the array and list data structures are sufficiently large. Specifically, assume an array size of roughly 10 M (million) elements, and a (simply-linked) list size of roughly 100 M elements. You are not allowed to transform these two main data structures into different data structures. You should not make any assumptions regarding the distribution or uniqueness of the values in the array or the list.

```
int do_work (int index, list_elem * listptr) {

   array [index].field2 += listptr->field1 + listptr->field3;
   listptr->field2 *= array [index].field1 * array [index].field3;

   return 0;
}

for (i = 0; i < ARRAY_SIZE; i ++) {
   for (ptr = list_head; ptr != NULL; ptr = ptr->next) {
      if (array [i].field4 == ptr->field4)
         do_work (i, ptr);
   }
}
```

a) (2 points) State the assumption(s) that you need to make to ensure the correctness of the parallel code.

b) (14 points) Write your code and explain your design here.

Student Number: Name:

Student Number: Name:

Problem 6. Machine Independent Code Restructurings and Optimizations for Parallel Code (14 Points)

Part A. (6 points)

```
for(J = 1; J <= M; J++){
   for(I = 1; I <= N; I++){
      A [I][J + 1] = A [I + 1][J] + B;
   }
}
```

Is it legal to perform a loop interchange between the two loops in the above code ? Explain. If the transformation is legal, parallelize the restructured code using OpenMP pragmas.

Student Number: Name:

Part B. (8 points)

```
for(I = 2; I <= N; I++){
    A[I] = B[I] + C[I];
    D[I] = A[I-1] * 2.0;
}
```

Apply any code-restructuring transformations that you deem necessary to the loop above in order to increase parallelism of the code; parallelize the restructured code using OpenMP pragmas. Explain why your code has better parallelism compared to the original code above.

Student Number: Name:

Problem 7. Machine Dependent Code Optimizations for Parallel Code (14 Points)

```
for( I = 1; I <= N ; I++) {
  for( J = 1; J <= M ; J++) {
      D[I] = D[I] + B[I][J];
  }
}
```

Assuming that the two arrays contain integers and the B array has *column-major* array layout, describe in sufficient detail what kind of techniques you would apply for improving the *cache behavior* of the code above (no need to write the full code) in the following cases:

a) (7 points) For the sequential code given. Demonstrate your improvement by calculating the cache miss rates for the original code and for the restructured code (as a formula), assuming that the cache line size is b (in words).

b) (7 points) For the corresponding parallelized code obtained using OpenMP pragmas.

Student Number: Name:

Student Number: Name: