Final Examination
ECE 454F 2008: Computer Systems Programming
Date: Dec 10, 2008, 2-4:30 p.m.

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

Problem number	Maximum Score	Your Score
1	12	
2	12	
3	16	
4	18	
5	14	
6	14	
7	14	
total	100	

This exam is open textbook and open lecture notes. You have two and half 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. (12 Points)

a. (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 cache coherence?

b. (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:

c. (4 points) Identify the dependences in the following for loop nest, written in pseudocode, specifying: the type of the dependence, whether it is loop-dependent or loop-independent and the iterations/statements/variables involved.

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

Student Number: Name:

Problem 2. Performance (CPE). (12 Points)

Consider the following function for computing the dot product of two arrays of n integers each. We have unrolled the loop by a factor of 3.

```
int dotprod(int a[], int b[], int n)
{
  int i, x1, y1, x2, y2, x3, y3;
  int r = 0;
  for (i = 0; i < n-2; i += 3) {
    x1 = a[i]; x2 = a[i+1]; x3 = a[i+2];
    y1 = b[i]; y2 = b[i+1]; y3 = b[i+2];
    r = r + x1 * y1 + x2 * y2 + x3 * y3; // Core computation
  }
  for (; i < n; i++)
     r += a[i] * b[i];
   return r;
}
```

Assume that we run this code on a machine in which multiplication requires 7 cycles, while addition requires 5. Further, assume that these latencies are the only factors constraining the performance of the program. Don't worry about the cost of memory references or integer operations, resource limitations, etc.

Please compute the ideal cycles per element (CPE) for each of the following associations for the core computation of this function and explain its performance. *Note: 1. If your answer is a fraction, leave it as such, you don't need to compute the result as a decimal number. 2. You need to explicitly state your assumptions, if any, and/or explicitly show your work.*

a. ((r + x1 * y1) + x2 * y2) + x3 * y3

b. (r + (x1 * y1 + x2 * y2)) + x3 * y3

c. r + ((x1 * y1 + x2 * y2) + x3 * y3)

Student Number: Name:

Problem 3. Cache Miss Rates. (16 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

- sizeof(char) = 1Byte, sizeof(int) = 4 Bytes , sizeof(short)=2 Bytes
- 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: *Note: You can write the answer for A and B as a set of multiplications e.g., $a * b * c$, and the answer for C as a fraction.*

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:

Part 2

After testing, you found that the performance of the above implementation (Part 1) is quite poor. Could you think of an alternative implementation which can greatly reduce the cache misses for the same application? Please write and/or explain in plain words your code, and with the same assumptions as in Part 1 compute the answer to the following questions for your implementation.

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: _____.

Please give explanation or write your code here:

Student Number: Name:

Problem 4. Dynamic Memory Allocation (18 Points)

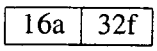
Consider an allocator with the following specification:

- Uses a single explicit free list.
- All memory blocks have a size that is a multiple of 8 bytes and is at least 16 bytes.
- All headers, footers, and pointers are 4 bytes in size
- Headers consist of a size in the upper 29 bits, a bit indicating if the block is allocated in the lowest bit, and a bit indicating if the previous block is allocated in the second lowest bit.
- Allocated blocks consist of a header and a payload (no footer)
- Free blocks consist of a header, two pointers for the next and previous free blocks in the free list, and a footer at the end of the block.
- All freed blocks are immediately coalesced.
- The heap starts with 0 bytes, never shrinks, and only grows large enough to satisfy memory requests.
- The heap contains only allocated blocks and free blocks. There is no space used for other data or special blocks to mark the beginning and end of the heap.
- When a block is split, the lower part of the block becomes the allocated part and the upper part becomes the new free block.
- Any newly created free block (whether it comes from a call to free, the upper part of a split block, or the coalescing of several free blocks) is inserted at the beginning of the free list.
- All searches for free blocks start at the head of the list and walk through the list in order.
- If a request can be fulfilled by using a free block, that free block is used. Otherwise the heap is extended only enough to fulfill the request. If there is a free block at the end of the heap, this can be used along with the new heap space to fulfill the request.

Student Number: Name:

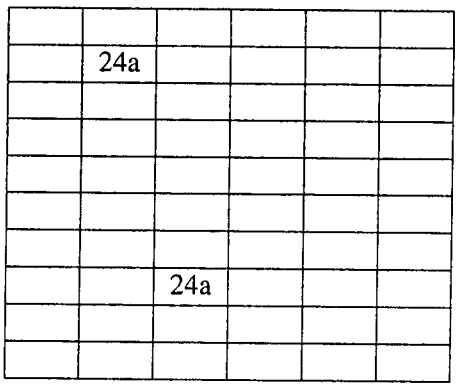
Part A. Simulating Malloc (10 points)

Below you are given a series of memory requests. You are asked to show what the heap looks like after each request is completed using a first fit placement policy. The heap is represented as a series of boxes, where each box is a single block on the heap, and the bottom of the heap is the left most box. In each block, you should write the total size (including headers and footers) of the block in bytes and either 'f' or 'a' to mark it as free or allocated, respectively. For example, the following heap contains an allocated block of size 16, followed by a free block of size 32.



Assume that the heap is empty before each of the sequences is run. You do not necessarily have to use all the boxes provided for the heap. Some of the boxes are already filled in to help you.

```
ptr1 = malloc(32);
ptr2 = malloc(16);
ptr3 = malloc(16);
ptr4 = malloc(40);
free(ptr3);
free(ptr1);
ptr5 = malloc(16);
free(ptr4);
ptr6 = malloc(48);
free(ptr2);
```



Student Number: Name:

Part B Code for Malloc (8 points)

For this part, you are asked to complete some small functions which are used to setup blocks. Each function will be missing a line of code and you are given three choices for this line of code. Circle the choice that completes the function correctly.

Function 1

```
/* Input:
*       void *block: a pointer to a block
*       unsigned long size: the size of the block,
*       char alloc: the lower order bit indicates if this block is
*                   allocated
*       char palloc: the lower order bit indicates if the previous
*                    block is allocated
*
* Actions:
*       This function will construct a header from the last 3
*       parameters and place it in the header of the block pointed
*       to by the first parameter.
*/

void make_header(void *block, unsigned long size, char alloc, char palloc)
{

        long header;

        _________;

        *(long *)block = header;

}

A. header = (size >> 3) | ((alloc & 0x1) << 31) | ((palloc & 0x1) << 30);
B. header = (size & ~0x7) | (alloc & 0x1) | ((palloc & 0x1) << 1);
C. header = size | alloc | palloc;
```

Student Number: Name:

Function 2

```
/* Input:
*       void *block: a pointer to a block
*       char palloc: the low order bit indicates if the previous
*                    block is allocated
* Actions:
*       Sets just the bit in the header indicating if the previous
*       block is allocated. Does nothing to the rest of the header.
*/

void set_palloc_in_header(void *block, char palloc)
{

        long curr = *(long *)block;

        ____________;

}
```

A. `*(long *)block = (curr & ~0x2) | ((palloc & 0x1) << 1);`
B. `*(long *)block = (curr & ~(0x1 << 30)) | ((palloc & 0x1) << 30);`
C. `*(long *)block = (curr | (0x1 << 30)) & ~((palloc & 0x1) << 30);`

Student Number: Name:

Problem 5. Parallel Programming: Pthreads/OpenMP (14 Points)

Design a strategy to parallelize the following code *efficiently*. *Avoid* writing low level code. Instead *explain* in sufficient detail all relevant design decisions related to: a) what part(s) of the code you run in parallel b) what synchronization you use and where and c) task to thread assignment. You can use plain language, pseudo-code, as in the lectures, e.g., fork, join, lock, unlock, signal, wait, etc, OpenMP directives, or all of the above to explain your parallelization strategy. 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 writing actual lines of code, or respecting rigurous syntax.

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) (12 points) Write pseudo-code and/or explain your design here.

b) (2 points) State the assumption(s) that you needed to make to ensure the correctness of the parallel code, if any.

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: