

## ECE 454 – Computer Systems Programming

The Edward S. Rogers Sr. Department of Electrical and Computer Engineering

Final Examination Fall 2010

<b>Name</b>	
<b>Student #</b>	

Answer all questions. Write your answers on the exam paper. Show your work.  
Each question has a different assigned value, as indicated.

The exam is open book (only simple calculators allowed, no cell phones or PDAs)

Total time available: **150 minutes**

Total marks available: **140** (roughly one mark per minute, with 10 extra minutes)

Verify that your exam has all the pages.

Part	Points	Mark
1	25	
2	20	
3	15	
4	20	
5	16	
6	15	
7	24	
Total	140	

**PART 1) [25] Short Answer**

- a) Given a program with a main loop that is perfectly parallel and comprises 80% of program execution time, how many threads/processors would you need to achieve a 4x speedup for the parallelized version relative to the original sequential version? Ignore the overheads of parallel execution (thread creation, synchronization, coherence, etc). Of course show your work.
- b) Briefly name/explain two benefits of out-of-order execution support in a processor.
- c) Spin-waiting is inefficient but simple. But why is spin-waiting worse for a CPU that supports simultaneous multithreading (hyperthreading) than a CPU that doesn't?

- d) “My threaded/synchronized program attains a total (across all cores) instructions-per-cycle (IPC) of 6.0 on multicore1, and an IPC of only 4.5 on multicore2, therefore multicore1 is better”; explain why this conclusion could be incorrect, and if you can think of multiple reasons please list them.
- e) If I want to determine the top function (by time) of a long-running program, but also want minimize the “observer effect”, what profiling tool/methodology should I use?
- f) Generally, in which case would function inlining be *least likely to harm* performance:  
A) a large function called from many locations; B) a large function called from a few locations; C) a small function called from many locations; D) a small function called from a few locations.
- g) You are optimizing a database application, which has a 16GB memory working set, and runs on a powerful machine that has 16GB of RAM. The machine has processors and an operating system that allow you to modify the size of pages from the standard 4KB page size. Could you improve the performance of this system by modifying the page size, if so would you make pages smaller or larger, and why exactly would this change improve performance?

- i) For dynamic memory allocation, say that I have decided to use a block that is slightly larger than necessary for what was requested. If I decline to split the block and return the whole block to the requestor, have I increased internal fragmentation or external fragmentation?

## PART 2) [20] Optimization

Optimize and rewrite the following code. You probably want to do this on scrap paper first!

Assumptions:

- 1) You are targeting a *single* superscalar processor with four adder and two multiply units.
- 2) Input[] is initialized by code that is not shown.

```
struct s {
    short a;
    unsigned v;
    short b;
} input[100];

int compute(int x,int *r,int *q,int *p){
    int i;
    for (i=0;i<100;i++){
        *r *= input[i].v + x;
        *p = input[i].v;
        *q += input[i].a + input[i].v + x + input[i].b;
    }
    return i;
}
```



### PART 3) [15] Memory Performance

Transform the following code to improve cache performance.

Assumptions:

- do **not** do tiling or parallelization
- assume a direct-mapped cache, so try to reduce potential conflicts
- the arrays are initialized in code not shown
- write a numbered list naming the transformations you did and why.
- HINT: you are allowed to change  $A[M][N]$  to  $A[N][M]$ , or do the same with  $B[][]$

```
int A[M][N]; // assume this kind of declaration is legal
int B[N][M];
int C[N];

for (i=0; i<N; i++){
    for (j=0; j<M; j++){
        B[i][j] = i*j;
        A[j][i] = B[i][j] + j;
    }
    C[i] = A[0][i];
}
```

#### PART 4) [20] Dynamic Memory Allocation

Consider an allocator with the following specification (*read carefully, this is different than the midterm*):

- Uses a single implicit free list.
- All memory blocks have a size that is a multiple of 4 bytes and is at least 8 bytes.
- All headers, footers, and pointers are 4 bytes in size
- Allocated blocks consist of a header and a payload (no footer)
- Free blocks consist of a header and a footer at the end of the block.
- All freed blocks are immediately coalesced if possible.
- 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 (and is coalesced if possible).
- The **best-fit** free block is used when searching the free list.
- If no free block is large enough then the heap is extended only enough to fulfill the request, including using any remaining free block at the end of the free list.

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 the best-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.

16a	32f
-----	-----

Assume that the heap is empty before each of the sequences is run, with a single 200B free block (shown for you). 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. It is recommended to solve this on a scrap paper then copy your final answer into the boxes when you are satisfied. There are two copies here in case you make a mess. Clearly circle the one you want graded and cross out the one you don't want graded.



COPY #1: (they are identical, we will only grade the one you circle)

	200f					
ptr1 = malloc(4)						
ptr2 = malloc(8)						
ptr3 = malloc(1)						
Ptr4 = malloc(31)						
free(ptr3)						
free(ptr1)						
free(ptr2)						
ptr5 = malloc(32)						
ptr6 = malloc(27)						
ptr7 = malloc(68)						
free(ptr6)						
ptr8 = malloc(20)						

COPY #2: (they are identical, we will only grade the one you circle)

	200f					
ptr1 = malloc(4)						
ptr2 = malloc(8)						
ptr3 = malloc(1)						
Ptr4 = malloc(31)						
free(ptr3)						
free(ptr1)						
free(ptr2)						
ptr5 = malloc(32)						
ptr6 = malloc(27)						
ptr7 = malloc(68)						
free(ptr6)						
ptr8 = malloc(20)						

## PART 5) [16] Parallel Work Distributions

Given this code:

```
int A[N*N]; /* in shared memory */
for (i=0; i < N; i++)                // for loop1
    for (j=0; j < N; j++)            // for loop2
        A[i*N+j] = work(A[i*N+j], i, j);
```

Assume that  $P$  is the number of CPUs, and that  $C$  is the number of ints that fits on a cache block. Specify the best performing work distribution for each of the following scenarios by:

- selecting to parallelize loop1 or loop2 (circle one)
  - selecting a static or dynamic partitioning (circle one)
  - give the best “chunk” size in terms of  $P$ ,  $N$ , and/or  $C$  (assuming the openMP definition of chunk)
- a)     $\text{work}()$  is constant and large (i.e., takes a long time), and  $N=P$
- circle: loop1   or   loop2
- circle: static   or   dynamic
- chunk size: \_\_\_\_\_
- b)     $\text{work}()$  is proportional to  $i^2$  and  $N$  is much larger than  $P$
- circle: loop1   or   loop2
- circle: static   or   dynamic
- chunk size: \_\_\_\_\_
- c)     $\text{work}()$  is non-uniformly-random and varies extremely;  $N$  is much larger than  $P$
- circle: loop1   or   loop2
- circle: static   or   dynamic
- chunk size: \_\_\_\_\_
- d)     $\text{work}()$  is large and proportional to  $i*j$ ;  $N$  is much larger than  $P$
- circle: loop1   or   loop2
- circle: static   or   dynamic
- chunk size: \_\_\_\_\_

## PART 6) [15] Dependence Analysis

Given the following code:

```
for (i=2; i<N-1; i++) { // loop1
    for (j=3; j<N; j++) { // loop2
        A[i][j] =
            A[i-2][j-3] + // c1
            A[i+1][j] +   // c2
            A[i][j];      // c3
    }
}
```

a) Give the notation for each dependence, naming the source and sink (as c1, c2, c3, or c4), distance, and dependence type. You can ignore input (read-after-read) dependences.

.

b) Which loops are parallel (if any) and why?

## PART 7) [24] Parallelization

Optimize and parallelize with openMP directives the following code. Do not do tiling. Please rewrite the new code.

a) parallelize the outer loop of this code:

```
for( i=0; i<n; i++ ){  
    for( j=0, total=0; j<m; j++ ){  
        total += a[i][j];  
        b[i] = c[i] * total;  
    }  
}
```

b) Parallelize the code from (a) instead parallelizing the inner loop

c) Parallelize the best loop for this code

```
for( j=0; j<100; j++ )  
    for( i=0; i<100; i++ )  
        a[i][j] = a[i][j-1] + 1;
```

d) Parallelize the best loop for this code

```
for( i=0; i<n; i++ )  
    for( j=i+1; j<n; j++ )  
        for( k=i+1; k<n; k++ )  
            a[j][k] = a[j][k] - a[i][k]*a[i][j] / a[j][j]
```