ECE 454 – Computer Systems Programming

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

Midterm Examination Fall 2012

Name	
Student #	
UTorID	

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: **110 minutes**

Total marks available: **110** (roughly one mark per minute)

Verify that your exam has all the pages.

Part	Points	Mark
1	25	
2	10	
3	5	
4	5	
5	15	
6	20	
7	5	
8	10	
9	15	
Total	110	

PART 1) [25] Short Answer

1) How many multiplier units should a wide-issue superscalar CPU have to best fit the following code? i.e., not too many units such that some are unused, and not too few units such that potential performance is hindered.

a)

```
for (i = 0; i < limit/2; i+=2) {
   x = x * (data[i] * data[i+1]);                    // 2-way parallel
   y = (y * data[limit/2 + i]) * data[limit/2 + i + 1];  // 1-way parallel
}
```

Number of multipliers: _____3_____

b)

```
for (i = 0; i < limit; i+=4) {
   x = x * ((data[i] * data[i+1]) *  (data[i+2] * data[i+3]));
}
```

Number of multipliers: _____4_____

2) Fill in the missing entries in the following table of CPUs and performance measures on a certain application, including the number of instructions executed for the application and the amount of time the execution took. B means billion, s means seconds. Name which CPU is the best one.

CPU	Clock Frequency	Instructions Executed	Average IPC	Execution Time
CPU1	5GHz	2B	2	0.2s
CPU2	2GHz	4B	1	2s
CPU3	4GHz	2B	4	0.125s

Which CPU is the best one?: _____CPU3_____

3) A compiler is able to speed up a program by 1.5384 times (note that 1.5384 ~= 1/0.65). The compiler has an optimization that applies to function1 of the program that speeds function1 up by 4x, and an optimization that applies to function2 of the program that speeds function2 up by 2x. If function2 comprises 40% of the execution time of the program before any optimization, what percentage of execution time is function1 before any optimization? Assume that no other part of the program is optimized or sped up.

Speedup = 1 / (1 – f + f/s)
Speedup = 1 / (1 – f1 + f1/s1 – f2 + f2/s2)
1/0.65 = 1 / (1 – f1 + f1/4 – 0.4 + 0.4/2)
1/0.65 = 1 / (1 – f1 + f1/4 – 0.2)
1/0.65 = 1 / (0.8 – 3f1/4)
0.65 = 0.8 – 3f1/4
3f1/4 = 0.15
f1 = 0.15*4/3
f1 = 0.2
hence function2 is 20% of execution

4) For each of the following scenarios a profiling tool/methodology is chosen. For each one, circle "yes" or "no" as to whether the choice of tool/methodology will give accurate results in a reasonable amount of time. If you circle "no" please explain why.

a) Using "perf" to estimate the number of L1 cache misses suffered by a matrix multiply program;

Accurate/reasonable?: (yes) no if no, why:

b) Using a software simulator of an Intel Core2 processor to measure the impact of a new associative L1 cache replacement scheme on a set of benchmark kernels;

Accurate/reasonable?: (yes) no if no, why:

c) Using a software simulator of an Intel Core2 processor to decide which function in a program represents the greatest execution time;

Accurate/reasonable?: yes (no) if no, why: software simulator is overkill, unnecessary accuracy, will take a long time.

d) Using gprof to decide a breakdown of execution time spent in different functions for a program that runs for 100ms.

Accurate/reasonable?: yes (no) if no, why: gprof samples every 10ms, not high enough resolution.

e) Using **/usr/bin/time** to decide if a compiler optimization is improving the execution time of a program that runs for 10 minutes.

Accurate/reasonable?: (yes) no if no, why:

5) Draw horizontal lines in the code below to divide the code into basic blocks as a compiler would do.

```
        Add …
        Add …
L1:     add …
        Add …
L2:     branch L4
        Add …
L3:     add …
        Branch L3
L4:     add …
        Add …
L5:     add …
        Branch L2
        Add …
L6:     return
```

PART 2) [10] Cache Accesss

For each part below, assume: a 16KB L1 data cache and a 1MB L2 cache that are both initially empty; 128B cache blocks; that an L1 miss takes 10 cycles if the access then hits the L2 cache and 100 cycles if it then misses the L2 cache as well; ignore TLB/paging effects; assume that the array elements are each 4B integers.

```
for (int i=0;i<N;i++)
  for (int j=0;j<N;j++)
     … = a[j];
```

a) Assuming $N = 2^{12}$, what is the total miss cycles for the entire code?

2^12*2^2B = 2^14 B= 16KB data accessed in inner loop, which fits in the L1 cache, therefore only first-time accesses will miss
Miss-cycles = 2^14/2^7 cache blocks * 100 cycles = 2^7 * 100 = 12,800 cycles

NOTE: we would also accept 2^7 * 110 = 14,080 cycles

b) Assuming $N = 2^{17}$, what is the total miss cycles for the entire code?

2^17*2^2B = 2^19B = 512KB, therefore every block accessed will miss the L1,
But only first-time accesses will miss the L2

Miss-cycles = 2^19/2^7 cache blocks * (100 cycles + 10 cycles * (2^17 – 1))
= 2^12 * (100 + 10*2^17 – 10)
= 10 * 2^12 * (9 + 2^17)
= 5.369*10^9 cycles

NOTE: we would also accept:
Miss-cycles = 2^19/2^7 cache blocks * (110 cycles + 10 cycles * (2^17 – 1))
= 2^12 * (110 + 10*2^17 – 10)
= 5.369*10^9 cycles

c) Assuming $N = 2^{20}$, what is the total miss cycles for the entire code?

2^20*2^2B = 2^22B = 4MB, therefore every block accessed will miss both

Miss-cycles = 2^22B/2^7 cache blocks * 2^20 *100 cycles
= 2^15*2^20 *100 cycles
= 2^35 *100 cycles
= 3.436*10^12 cycles

NOTE: we would also accept:
Miss-cycles = 2^22B/2^7 cache blocks * 2^20 *110 cycles
= 2^15*2^20 *110 cycles
= 2^35 *110 cycles
= 3.7796*10^12 cycles

PART 3) [5] TLB Behavior

Assume a fully-associative, 128-entry TLB and 8KB pages.

a) What is the "TLB-reach" for this TLB?

8KB*128entries = 2^3*2^10*2^7 = 2^20 = 1MB

b) What tile size (measured in number of array elements) will maximize use of the TLB and minimize TLB misses, if the code below were to be tiled as taught in class? Assume that the array elements are 4B integers, that N is much greater than the page size, and that you can ignore any caches for this question.

```
for (int i=0;i<N;i++)
  for (int j=0;j<N;j++)
    for (int k=0;k<N;k++)
      d[i][j] += a[i][k]*b[k][j]*c[i][k];
```

4 matrices accessed, the tiled version will access 4 tiles at any one time
Since N >> page_size, each row of a tile will be on a different page
Hence want 4T =~ total TLB pages
4T = 128
T = 32

therefore tile size of 32x32 array elements is best.

PART 4) [5] Compiler Optimization

The following shows a function before and after optimization by a compiler. For the "AFTER" version of the code, indicate to the left of each line where appropriate the initials of the name of the optimization that was performed (i.e., write CP for constant propagation). It is ok to name multiple optimizations for a single line of code if multiple were performed. Write IMPOSSIBLE to the left of a line if an optimization is not one of the optimizations that we covered in class, and/or you think it is impossible for a typical compiler to do such an optimization.

BEFORE:

```
01: void foo(int n){
02:   int x = 10;
03:   int y = 20;
04:   int z = x * y;
05:   int v = 1;
06:   int w, q;
07:
08:   for (int i=0; i<n; i++){
09:     y = x * n;
10:     q = v * y;
11:     if (x < 10)
12:       z++;
13:     w = v * y + z;
14:     v = q + w;
15:   }
16:   printf("%d %d %d %d %d %d\n",x,y,z,v,w,q);
17: }
```

AFTER:

```
01: void foo(int n){
02:   int x;          // DCE
03:   int y;          // DCE
04:   int z = 200;    // CP, CF
05:   int v = 1;
06:   int w, q;
07:   y = 10 * n;     // CP, LICM(1) (ACTUALLY UNSAFE, IF n==0)
08:   for (int i=0; i<n; i++){
09:                   // LICM(1)
10:     q = v * y;    // CP
11:                   // CP, DCE
12:
13:     w = q + 200; // CP, CSE
14:     v = q + w;
15:   }
16:   printf("%d %d %d %d %d %d\n",10,y,200,v,w,q);  // CP
17: }
```

PART 5) [15] Alignment

a) How many bytes of memory will the following data structure Data1 occupy? The sizes of int/float/short/char in bytes are given.

```
struct mystruct1 {
  int x;   // 4
  char c;  // 1
  short z; // 2
  float y; // 4
  char k;  // 1
  short p; // 2
  char m;  // 1
  char n;  // 1
} Data1[1000];
```

=20B*1000 = 20000B

b) Re-declare Data1 to be more space-efficient, and give the resulting total size in bytes.

```
struct mystruct1 {
  int x;   // 4
  float y; // 4
  short z; // 2
  short p; // 2
  char c;  // 1
  char k;  // 1
  char m;  // 1
  char n;  // 1
} Data1[1000];
```

=16B*1000 = 16000B

c) The following data structure Data2 is used to store one million samples of (x,y,z) 3D positioning data. How many bytes of memory will Data2 occupy?

```
struct mystruct {
  int x;   // 4
  int y;   // 4
  int z;   // 4
} Data2[1000000];
```

= 12B * 1000000 = 12000000B

d) How many bytes of memory will the following alternative data structure Data3 occupy?

```
struct mystruct {
  int x[1000000];    // int is 4
  int y[1000000];    // int is 4
  int z[1000000];    // int is 4
} Data3;
```

= 4B * 1000000 * 3 = 12000000B

e) Data2 and Data3 above can both be used to store the same set of data, but would have different layouts in memory. If you were asked to write code to compute the distance from the origin of (x,y,z) for 1000 randomly-selected samples, which of Data2 or Data3 would be the preferred data structure and why is it better than the alternative?

Data2, since you are accessing x, y, and z for each randomly-selected element. There would be cache-block locality within each element. Data3 would require accessing 3x as many cache blocks, since x, y, z would always reside on independent cache blocks.

f) If you were asked to write code to compute the mean of y across all million values, which of Data2 or Data3 would be the preferred data structure and why?

Data3, since you are only accessing y for each element. There would be cache-block locality from one element to the next, which would also enable hardware/compiler prefetching.

PART 6) [20] Dynamic Memory Allocation

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
- 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 if possible, regardless of position in the free list.
- 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 (i.e., first-fit).
- 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.

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, 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(1)	16a	184f					
ptr2 = malloc(12)	16a	16a	168f				
ptr3 = malloc(17)	16a	16a	24a	144f			
ptr4 = malloc(36)	16a	16a	24a	40a	104f		
free(ptr2)	16a	16f	24a	40a	104f		
ptr5 = malloc(37)	16a	16f	24a	40a	48a	56f	
free(ptr4)	16a	16f	24a	40f	48a	56f	
ptr6 = malloc(20)	16a	16f	24a	24a	16f	48a	56f
free(ptr5)	16a	16f	24a	24a	120f		
ptr7 = malloc(120)	16a	16f	24a	24a	128a		
free(ptr3)	16a	40f	24a	128a			

NOTE: since there is an explicit free list, the first-fit free block is not necessarily the first free block in the left-to-right order! Ptr7 extends the heap by 8B

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

	200f						
ptr1 = malloc(1)							
ptr2 = malloc(12)							
ptr3 = malloc(17)							
ptr4 = malloc(36)							
free(ptr2)							
ptr5 = malloc(37)							
free(ptr4)							
ptr6 = malloc(20)							
free(ptr5)							
ptr7 = malloc(120)							
free(ptr3)							

PART 7) [5] Cache Coherence

Assuming a 4-CPU multicore with MESI invalidation-based cache coherence with write-back caches, considering only the cache block for location X, in the table below are shown the order in time of loads and stores to X. Put a '1' or a checkmark in the column on the left next to each row for which a coherence message occurs **that requests a copy** of the cache block contents for the corresponding CPU's cache. Assume that all caches are initially empty. HINT: a load-miss results in a copy request, while a write-hit does not.

Copy of cache block requested?	CPU 0	CPU 1	CPU 2	CPU 3
1		Load X		
1			Load X	
		Load X		
		Store X		
1			Store X	
			Load X	
1		Load X		
1	Load X			
1				Load X
				Store X
				Load X
1	Load X			
				Load X
	Load X			

Minus one mark for any checkmark missing/extra

PART 8) [10] Consistency

For a machine that does not implement sequential consistency, assume that loads and stores to independent addresses can be reordered by the CPU.

a) In the following code, after both threads T1 and T2 have executed, is it possible for T2 to view the values (flag,x) == (1,0)? Give the order in which statements (a),(b),(c),(d) might execute to produce this result. Assume that x and flag are shared memory locations.

```
Initialization: x=0, flag=0;

T1:
x = 1;      // a
flag = 1;   // b

T2:
while (!flag){}; // c
… = x;           // d
```

Is (flag,x) == (1,0) possible for T2 after both threads have executed (yes or no)? __yes____

If so, give the order of (a),(b),(c),(d) that results in T2 having (flag,x) == (1,0): __b,c,d,a__

b) Some processors, including intel processors, provide "fence" instructions. For example, an "mfence" instruction tells the processor not to reorder loads or stores around the fence instruction---i.e., the mfence instruction cannot complete until all prior loads and stores have completed and are visible to the entire system, and loads and stores after the mfence cannot be executed until the mfence has completed.

Rewrite T1 and T2 from part (a) above and insert the minimum number of mfence instruction(s) as necessary to "fix" the code, i.e., so that (flag,x) == (1,0) is not a possible outcome for T2.

```
T1:
x = 1;      // a
mfence
flag = 1;   // b

T2:
while (!flag){}; // c
mfence
… = x;           // d
```

c) After all of threads T1-T4 below have executed, is it possible that (t3x,t3y,t4x,t4y) = (1,0,0,1) ? If so, what order of (a)..(f) can cause it? Assume that t3x, t3y, t4x, and t4y are all registers, while x and y are shared memory locations.

```
Initialization: x = 0, y = 0;

T1:
x = 1;      // a

T2:
y = 1;      // b


T3:
t3x = x;      // c
mfence;
t3y = y;      // d

T4:
t4y = y;      // e
mfence;
t4x = x;      // f
```

Is (t3x,t3y,t4x,t4y) = (1,0,0,1) possible after all threads have executed? __yes____

If so, give an order of (a)...(f) that results in (t3x,t3y,t4x,t4y) = (1,0,0,1) __a,c,d,b,e,f_____

d) Why is the answer for (c) troubling?

T3 and T4 get different values for x and y, even though they use fence instructions to enforce ordering within each thread.

PART 9) [15] Locking

a) Make this code safe for parallel execution by inserting calls to **lock** and **unlock** wherever necessary. Note that for this question lock/unlock implement a single global lock---meaning that you do not specify "`lock x`" but only "`lock`". Make the resulting critical sections between lock/unlock as small as possible. Assume that update() and work() do not access memory, and that work() takes a significant number of cycles to execute. You are **not** allowed to re-order the statements within the threads.

```
// shared memory
int v;
int w;
int x;
int y;
```

Thread1:

```
while(1){

  lock;
  int a = v;

  if (a < 100)
  {

    w = update(w);

  }

  int b = x;

  v = update(a)

  x = update(b);
  unlock;

  work();

  lock;
  y = update(y);

  unlock;
}
```

Thread2:

```
while(1){

  lock;
  int a = w;

  work();

  if (a < 100)
  {

    v = update(a);

  }

  w = update(a);

  unlock;
  work();

  lock;
  int b = x;

  int c = update(b)

  x = update(c);

  y = update(y);

  unlock;
}
```

b) Make this code safe for parallel execution by inserting calls to **lock X** and **unlock X** wherever necessary. Note that for this question lock/unlock implement named locks (i.e., fine-grained locks), where X can be any name you want. Make the resulting critical sections between lock/unlock as small as possible. You are **not** allowed to re-order the statements within the threads. Note that this is the same code as in (a) above.

```
// shared memory
int v;
int w;
int x;
int y;
int z;
```

Thread1:

```
  while(1){
    lock W;
    lock V;
    int a = v;


    if (a < 100)
    {


      w = update(w);


    }
    unlock W;
    lock X;
    int b = x;


    v = update(a)
    unlock V;

    x = update(b);
    unlock X;

    work();

    lock Y;
    y = update(y);

    unlock Y;
  }
```

Thread2:

```
  while(1){

    lock W;
    int a = w;


    work();


    if (a < 100)
    {

      lock V;
      v = update(a);
      unlock V;

    }


    w = update(a);
    unlock W;

    work();

    lock X;
    int b = x;


    int c = update(b)


    x = update(c);
    unlock X;
    lock Y;
    y = update(y);

    unlock Y;
  }
```

c) Make this code safe for parallel execution using only transactional memory, i.e., by inserting calls to txn_start and txn_end. Make the resulting critical sections between lock/unlock as small as possible. You are **not** allowed to re-order the statements within the threads. Note that this is the same code as in (a) above.

```
// shared memory
int v;
int w;
int x;
int y;
int z;
```

Thread1:

```
  while(1){

    txn_start;
    int a = v;


    if (a < 100)
    {


      w = update(w);


    }

    int b = x;


    v = update(a)


    x = update(b);
    txn_end;

    work();

    txn_start;
    y = update(y);

    txn_end;
  }
```

Thread2:

```
  while(1){

    txn_start;
    int a = w;


    work();


    if (a < 100)
    {


      v = update(a);


    }

    w = update(a);
    txn_end;

    work();

    txn_start;
    int b = x;


    int c = update(b)


    x = update(c);
    txn_end;
    txn_start;
    y = update(y);

    txn_end;
  }
```