ECE 454 – Computer Systems Programming

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

Final 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: **150 minutes**

Total marks available: **145** (roughly one mark per minute with 5 extra minutes)

Verify that your exam has all of the pages.

Part	Points	Mark
1	32	
2	10	
3	12	
4	20	
5	10	
6	15	
7	15	
8	16	
9	15	
Total	145	

PART 1) [32] Short Answer

1) If I have a 10-board rack of boards that each have a quad-core processor, each core of which supports 2-way SMT, has 16KB first-level data caches, and a 2MB second-level unified cache that is shared by all cores on a chip:

a) how many hardware thread contexts does my machine support?

ANSWER: 10*4*2 = 80 2marks

b) how much first-level data cache is available per hardware thread context (dividing evenly)?

ANSWER: 16KB/2 = 8KB 2marks

c) how much second-level data cache is available per hardware thread context (dividing evenly)?

ANSWER: 2MB/8 = 256KB 2marks

2) Consider this scenario for a certain program and with two compiler optimizations opt1 and opt2: opt1 makes 30% of program execution time go 2x faster; opt2 makes 60% of program execution time go 1.5x faster; however, half of the execution time optimized by opt2 can also be optimized by opt1, but not by both at the same time.

Assuming that you have a compiler that can best decide which optimization to apply for any given part of the code, what is the best possible speedup for the program?

ANSWER: 7marks

Speedup opt1: 1/(0.3/2.0+0.7) = 1/0.85 = 1.176471
Speedup opt2: 1/(0.60/1.5+0.40) = 1/0.80 = 1.250000

HENCE opt2 is better, apply that to the overlap portion.
Opt2 applies to 60%, hence 30% is the overlap portion. But opt1 only applies to 30%, so there is no leftover for opt1 to optimize. Hence Speedup for opt2 result above is final answer: 1.25

3) For a certain program, gprof results are shown below both before and after compiler optimization. What optimization is there evidence of in the gprof results?

Before:

```
     % cumulative     self              self    total
  time    seconds  seconds    calls  ms/call  ms/call  name
 86.60       8.21     8.21        1  8210.00  8210.00  bigfunc
  5.80       8.76     0.55   946596     0.00     0.00  calc
  4.75       9.21     0.45   946596     0.00     0.00  lookup
  1.27       9.33     0.12   946596     0.00     0.00  combine
```

After:

```
     % cumulative     self              self    total
  time    seconds  seconds    calls  ms/call  ms/call  name
 86.60       8.21     8.21        1  8210.00  8210.00  bigfunc
 10.55       9.21     1.00   946596     0.00     0.00  calc
  1.27       9.33     0.12   946596     0.00     0.00  combine
```

ANSWER:

Inlining of lookup into calc 3marks

4) Which version of this code (they all produce the same correct results) would a compiler more likely be able to optimize and why? Assume the '...'s represent other code not shown.

a)

```
for (i=0;i<n;i++){
  ...
  *result += values[i];
}
```

b)

```
for (i=0;i<n;i++){
  ...
  *result += *values;
  values++;
}
```

c)

```
for (i=0;i<n;i++){
  ...
```

```
  tmp += values[i];
}
*result = tmp;
```

d)

```
for (i=0;i<n;i++){
  ...
  tmp += *values;
  values++;
}
*result = tmp;
```

ANSWER: (c), array accesses are better understood and scheduled than pointer accesses, possibly parallelized (reduction), and the result can be accumulated in a register and copied to memory only after the loop. 5marks

5) How many bytes will this data structure require for best alignment:

```
struct mystruct {
  char *y;
  char k;
} A[100];
```

a) on a 32-bit machine:

ANSWER: 800B = (4B + 1B + 3Bpad) * 100 elements 3marks

b) on a 64-bit machine (in 64-bit-mode):

ANSWER: 1600B = (8B + 1B + 7Bpad) * 100 elements 3marks

6) Assuming sequential consistency, what are the possible outcomes for x after the execution of both threads, assuming that x is in shared memory and is initially zero, and that each line is an (atomic) instruction.

Thread1:
```
  lock;
  int tmp = x;
  x = tmp + 1;
  unlock;
```

Thread2:
```
  int tmp = x;
  x = tmp + 10;
```

ANSWER: 1, 10, 11 (every possible interleaving, the lock/unlock has no effect) 5marks

PART 2) [10] Dependences: (straight-line code)

a) Name the dependences in the following code (true, anti, output). For each write the name and the line numbers involved to the right of the last line involved. The first one is given for you. Note that a given line may be involved in multiple dependences; list them all.

1: a = b + c;
2: d = a + c; true 1->2
3: d = d + c; true 2->3; anti 2->3, output 2->3
4: c = 4 + a; true 1->4; anti 3->4
5: e = 5 + c; true 4->5
6: c = 3 + e; true 5->6; output 4->6; anti 5->6; true 5->6
7: e = d + c; true 3->7; output 5->7; anti 6->7; true 6->7 5.5marks (0.5 each)

(note: only most recent dependence for a variable is required)

b) Rewrite the code above to remove as many dependences as possible by using new variables where appropriate. Do not perform any other optimization nor code elimination. List the dependences that remain the same way as above.

1: a = b + c;
2: d = a + c; true 1->2
3: d1 = d + c; true 2->3 1mark
4: c1 = 4 + a; true 1->4 1mark
5: e = 5 + c1; true 4->5
6: c2 = 3 + e; true 5->6 1mark
7: e1 = d1 + c2; true 6->7;

c) Assuming that each operation takes a single cycle to execute, and a wide-issue superscalar processor, how many cycles will the code from (b) take to execute?

5 cycles 1.5marks

PART 3) [12] Cache Accesss

For each of the following codes, assuming that A[] is of type int (4B elements), a 16KB data cache that is very associative (ie., don't consider conflict misses), with 64B cache blocks, what is the maximum value for N that does not suffer capacity misses? (ie., don't consider cold misses either)

a)

```
for (i=0;i<N;i++){
  ... = A[i];
}
```

ANSWER:
N could be any size, since there is no hope of re-use, hence no capacity misses 3marks

b)

```
for (k=0;k<100;k++){
  for (i=0;i<N;i++){
    ... = A[i];
  }
}
```

ANSWER:
N = 16KB/4B = 4K 3marks

c)

```
for (k=0;k<100;k++){
  for (i=0;i<N;i++){
    for (j=0;j<N;j++){
      ... = A[i][j];
    }
  }
}
```

ANSWER: 16KB = 4B*N^2
N = sqrt(16K/4) = sqrt(4K) = 64 3marks

d)

```
for (k=0;k<100;k++){
  for (i=0;i<N;i++){
    for (j=0;j<N;j++){
      ... = A[i][j] + B[i][j] + C[i][j] + D[i][j];
    }
  }
}
```

ANSWER: 16KB = 4B*4*N^2
N = sqrt(16K/16) = sqrt(1K) = 32 3marks

PART 4) [20] Dynamic Memory Allocation

Consider an allocator with the following specification:

- Uses an implicit free list.
- All memory blocks have a size that is a multiple of 8 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
- Splitting is performed when appropriate
- All searches for free blocks start at the head of the list and walk through the list in order (i.e., first-fit).

Headers and footers are encoded as follows:

- bit0 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 0.
- bits 31-3 encode the block size, as if bit 2, bit 1, and bit 0 were all zeros. Eg., for an 8byte block bit 3 is a 1 and bits 31-4 are zeros; for a 16byte block bit 4 is a one and bit 3 and bits 31-5 are zeros.

a) What is the internal fragmentation for this scheme for the following requests:

malloc(1)	3 bytes
malloc(4)	0 bytes
malloc(5)	7 bytes
malloc(13)	7 bytes

1 mark each

Using the scheme on the previous page, for the following tables, list only the heap entries that change based on the user operation given. Note that the initial state for (c) should be your answer from (b). Note also that the heap starts at the bottom (address `0x0400b000`) and grows upwards.

b) User calls free(0x0400b024): 8 marks

Address	Contents (before)	Changed contents
0x0400b03c	0x0000001b	
0x0400b038	0x0000000b	
0x0400b034	0x0000000b	
0x0400b030	0x0000000b	0x00000009
0x0400b02c	0x0400b028	0x00000012
0x0400b028	0x00000011	
0x0400b024	0x0f032014	
0x0400b020	0x00000013	0x00000012
0x0400b01c	0x0000000b	
0x0400b018	0x0000000b	
0x0400b014	0x00000012	
0x0400b010	0x00000013	
0x0400b00c	0x0000001f	
0x0400b008	0x0000000b	
0x0400b004	0x0400b008	
0x0400b000	0x00000011	

c) User then calls malloc(1)---assuming starting with answer from (b)
8 marks

Address	Contents (before)	Changed contents
0x0400b03c		
0x0400b038		
0x0400b034		
0x0400b030	0x00000009	
0x0400b02c	0x00000012	0x0000000a
0x0400b028		0x0000000a
0x0400b024		
0x0400b020	0x00000012	0x0000000b
0x0400b01c		
0x0400b018		
0x0400b014		
0x0400b010		
0x0400b00c		
0x0400b008		

0x0400b004		
0x0400b000		

PART 5) [10] Cache Coherence

List the coherence actions that happen for each load/store; assume that each load/store (row of the table) is completely performed in the entire system before moving on to the next load/store (the next row of the table). You may list multiple actions if multiple occur. Multiples of an action may occur in one row: eg., if invalidationg messages are sent to two CPUs in one row, write 2I. Note that a message (msg) does not contain a copy of a cache block, only a request.

```
R:  Read             // msg from a CPU to the memory
RX: Read-Ex          // msg from a CPU to the memory
NM: Notify-Modified  // msg from a CPU to the memory
U:  Update           // copy from a CPU to the memory
NS: Notify-Shared    // msg from the memory to a CPU
I:  Invalidation     // msg from the memory to a CPU
F:  Fill             // copy from the memory to a CPU
```

Action(s)?	CPU 0	CPU 1	CPU 2	CPU 3
R,F	Load X			
NM	Store X			
R,U,F		Load X		
NM,I		Store X		
R,U,F			Load X	
R,F				Load X
R,F	Load X			
		Load X		
NM,3I			Store X	
			Store X	
RX,I,				Store X
				Load X
				Store X

0.5 for each individual msg/copy, 1.0 for the 3I, minus 1.0 for extra msg/copy
Maybe allow RX instead of NM

PART 6) [15] Locking

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. You can assume that these are the only two parallel threads, and that update() does not read or write memory other than (maybe) its parameter.

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

Thread1:

```
  while(1){

    lock V;
    v = update(v);
    unlock V;
    lock W;
    int a = w;


    a *= 100;


    x = a;


    if (a < 1000)
    {

      w = a;

    }
    unlock W;

    work();

    lock V;
    int b = update(v);

    lock Z;
    z = z + b + x;
    unlock Z;

    v = b / 2;
```

Thread2:

```
  while(1){

    lock V; lock Z;
    int b = update(z);


    v = v + b;
    unlock V

    z = b / 2;
    unlock Z

    int a = y;


    work();


    y = update(a);

    lock W;
    w = update(w);
    unlock W;

  }
```

```
    unlock V;

  }
```

PART 7) [15] Dependence Analysis

For each loop in the following code, state whether or not it is parallel, and if not describe the dependence(s) that prevent it from being parallel (give the type of dependence and which terms c1,c2,c3,c4 are involved).

```
for (i=2;i<N-1;i++){                  // loop1
  for (j=3;j<N;j++){                  // loop2
    for (k=4;k<N-3;k++){              // loop3
      A[i][j][k] =                       // c1
                   A[i][j-1][k+1] +   // c2
                   A[i-1][j+2][k] +   // c3
                   A[i+3][j][k];      // c4
    }
  }
}
```

loop1: not parallel, 2marks
true/flow dep C1->C3, 3marks
anti dep C1->C4 3marks

loop2: not parallel 2marks
true/flow dep C1->C2 3marks

loop3: parallel 2marks

PART 8) [16] Data Distributions

For each code listing below, list the array elements that would be written by each thread before either the first instance of a barrier or the end of the execution is reached. Name the elements using the naming/numbering scheme given in the table, assuming row-major ordering (i refers to row and j to column). Eg., for a single threaded execution (with no barrier) for any of the codes the answer would be: T0: 1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16

A[i][j]

	0	1	2	3
0	1	2	3	4
1	5	6	7	8
2	9	10	11	12
3	13	14	15	16

a) Assume two threads, T0 and T1

```
#pragma omp parallel for private(i,j)
for (int i=0;i<N;i++)
  for (int j=0;j<N;j++)
    A[i][j] = …
```

T0: 1,2,3,4,5,6,7,8
T1: 9,10,11,12,13,14,15,16
barrier

b) Assume four threads, T0,T1,T2, and T3

```
for (int i=0;i<N;i++)
#pragma omp parallel for private(j)
  for (int j=0;j<N;j++)
    A[i][j] = …
```

T0: 1
T1: 2
T2: 3
T3: 4
barrier

c) Assume two threads, T0, and T1

```
#pragma omp parallel for schedule(static,1) private(i,j)
for (int i=0;i<N;i++)
  for (int j=0;j<N;j++)
    A[i][j] = …
```

T0: 1,2,3,4,9,10,11,12
T1: 5,6,7,8,13,14,15,16
barrier

d) Assume two threads, T0, and T1 (this one has multiple correct answers, show one of them)

```
for (int i=0;i<N;i++)
#pragma omp parallel for schedule(dynamic,2) private(j) nowait
  for (int j=0;j<N;j++)
    A[i][j] = …
```

T0: 1,2,7,8,11,12,13,14
T1: 3,4,5,6,9,10,15,16

PART 9) [15] Parallelization

Parallelize this code using OpenMP but without using the reduction option/primitive. Don't worry about optimizing cache/memory locality.

```
Minimum = GREAT_BIG_VALUE;
for (i=0;i<N;i++){
  for (j=0;j<N;j++){
    A[i][j] = B[i][j] + C[i][j];
    Minimum = min(Minimum,A[i][j]);
  }
}
```

```
Minimum = GREAT_BIG_VALUE;
#pragma omp parallel threadprivate(mymin)
{
  mymin = GREAT_BIG_VALUE;
  #pragma omp for private(i,j) nowait
  for (i=0;i<N;i++){
    for (j=0;j<N;j++){
      A[i][j] = B[i][j] + C[i][j];
      mymin = min(mymin,A[i][j]);
    }
  }
  #pragma omp critical
  Minimum = min(Minimum,mymin);
  #pragma omp barrier
}
```