ECE 454 – Computer Systems Programming

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

Midterm Examination Fall 2010

Name	
Student #	

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: **100** (roughly one mark per minute, with 10 extra minutes)

Verify that your exam has all the pages.

Part	Points	Mark
1	**8**	
2	**8**	
3	**14**	
4	**10**	
5	**10**	
6	**20**	
7	**20**	
8	**10**	
Total	**100**	

PART 1) [8] Measuring Programs

Which measuring tool should you use in each of the following scenarios? For each choose the simplest and lowest overhead tool that is effective for the task. Put the number for the tool in the box next to each scenario.

List of Tools:

1) command-line timer like /usr/bin/time
2) C library timer like in <sys/times.h>
3) Hardware timer or performance counters
4) General-purpose instrumentor like PIN
5) GPROF/GCOV
6) Simulator (hardware or software)

Scenarios: for each you want to measure…

a) The minimum number of processor cycles to execute a function
b) The average duration of a long-running program
c) The number of instructions per-instance of a certain function (on average)
d) The impact of a new coherence scheme on performance of a program
e) The top function (by time) of a long-running program
f) The duration of a function within a 1-second program
g) The cache miss behavior of a production web-server in-the-field (i.e., live)
h) The average behavior of an if-else branch for a hot function in a long-running program

PART 2) [8] Short Answer

a) Name six things that a modern superscalar processor (such as the ones in the ‘UG’ machines) does to optimize the execution of a program automatically and transparently to the programmer. Just the one or two word name for each is fine.

b) What is the difference between coherence and consistency? A one or two sentence answer should suffice.

c) Give three reasons that you hypothetically might NOT want to use –O3 optimization for gcc when compiling a program?

PART 3) [14] Optimization Decisions

The following shows the (fake) top four results of a GPROF measurement of VPR. OptsRus has three optimizations planned, and want to maximize the performance of VPR.

```
  %   cumulative   self              self     total           
 time   seconds   seconds    calls   s/call   s/call  name    
 66.00      1.71     1.71   891388     0.00     0.00  try_swap
 16.00      2.13     0.42 26666027     0.00     0.00  comp_td_p2p_delay
  5.00      2.25     0.12    68548     0.00     0.00  label_wire_muxes
  3.00      2.33     0.08  3160610     0.00     0.00  update_bb
```

Optimization1: will make `try_swap` 10% faster (i.e., 1.1x faster)

Optimization2: will make `comp_td_p2p_delay` 100% faster (i.e., 2.0x faster)

Optimization3: will inline `comp_td_p2p_delay` into update_bb, but the new update_bb will be 200% slower (i.e., 3x slower).

a) OptsRus only has time/resources to implement one optimization. Which optimization should you choose to implement and why? Give the final expected program speedup of your chosen optimization. HINT: your answer should involve some simple calculations, and should give the results of those.

OptsRus should implement Optimization # ____

Because:

b) If OptsRus found the time/resources to implement a second optimization, which should it be and why?

OptsRus should implement Optimization # ____

Because:

PART 4) [10] Compiler Optimization

Name 8 optimizations that you would hope your compiler would do to this code. For each, give the formal name of the optimization and very briefly describe which variable(s) or code parts it will modify/improve and how. The first answer of 8 is given as an example.

```
1: x=5;
2: y=2;
3: debug = 0;
4: z=x+y;
5:
6: for (i=0;i<100;i++){
7:   m += i*x+y;
8:   n = z*y;
9:   A[i] += A[m];
10:
11:  if (debug){
12:    printf("m: %d\n",m);
13:  }
14:  q += i*x;
15:}
16:
17:printf("result: %d %d %d %d %d %d\n",m,n,q,x,y,z);
```

ANSWERS:
constant propagation: line4 changed to z=5+2

PART 5) [10] Static Memory

Consider the following declaration, Assume a CPU architecture like the UG machines, with typical 32-bit x86 linux.

```
struct S {
  char c;
  int i;
  char ca[3];
  double d;
} A[100];
```

a) How much space (in bytes) does the following array of structs take in memory? Draw (horizontally) the layout of the elements.

b) Can you modify the declaration to save space? If so, give the new declaration, and give the size of the modified array. Draw (horizontally) the new layout of the elements.

PART 6) [20] Memory Performance

OptsRus must optimize the following variant of matrix multiply code to target a certain machine, the specs for which are below.

```
// assume A[N][N], B[N][N], C[N][N], and D[N][N] have integer elements

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

Target Machine Specs:
16KB L1 Cache
4MB L2 Cache
64B cache blocks (for both caches)

From past experience, OptsRus knows that tiling is the way to go. Considering tiling as the ***only*** optimization, and also considering only square tiles, what should the tile dimension T (i.e., assuming TxT tiles) be to optimize memory performance for each of the given values of N (array dimension) for the code?   Give the value of T as both the number of elements and the number of bytes. Show the work you did to compute the tile dimension T and explain in one sentence why this choice should work well. You do NOT have to show the tiled code.

NOTE: we are assuming that each tile size you choose would be theoretically the best, ignoring prefetching and other subtleties of the machine that might lead to a different answer via experimentation as in the lab homework.

a) N= 512 T: _______elements, _______bytes

b) N= 1024 T: _______elements, _______bytes

PART 7) [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.
- 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.

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(20)						
ptr2 = malloc(30)						
free(ptr1)						
free(ptr2)						
ptr3 = malloc(72)						
ptr4 = malloc(1)						
ptr5 = malloc(12)						
ptr6 = malloc(24)						
free(ptr3)						
free(ptr5)						
ptr7 = malloc(8)						

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

	200f					
ptr1 = malloc(20)						
ptr2 = malloc(30)						
free(ptr1)						
free(ptr2)						
ptr3 = malloc(72)						
ptr4 = malloc(1)						
ptr5 = malloc(12)						
ptr6 = malloc(24)						
free(ptr3)						
free(ptr5)						
ptr7 = malloc(8)						

PART 8) [10] Locks and Critical Sections

For the following code, explain straightforward things you could do to improve its performance without changing its output (i.e., so it still works).

Assumptions:

- These are the only three threads in the program, running in parallel on separate CPUs.
- v,w,x,y,z are all in shared memory
- You are not allowed to add additional critical sections.
- You are not allowed to create fewer or more threads.
- You do not know what is inside the ‘update’ functions, you cannot optimize inside them.

HINT: think about what you learned about locks, sharing, and coherence behavior.

```
int v;
int w;
int x;
int y;
int z;
```

```
Thread1:                  Thread2:                  Thread3:
  while(1){                 while(1){                 while(1){
    …                         …                         …
    test_&_set_lock(L);       test_&_set_lock(L);       test_&_set_lock(L);
      y = updatey(y);           y = updatey(y);           y = updatey(y);
      w = updatew(w);           v = updatev(v);           x = updatex(x);
      z = updatez(z);           w = updatew(w);           z = updatez(z);
    unlock(L);                  z = updatez(z);         unlock(L);
  }                             x = updatex(x);       }
                            unlock(L);
                          }
```