Programming for Performance (ECE459): Study Questions for Final

This open-book final will have 6 questions, worth 20 points each. Of those 6 questions, 1 may be short answer (fill-in-the-blank), and 1 or 2 may be on pre-midterm material (not included here).

This is a preliminary version of the question list. I may modify it based on feedback from you and the TAs. I'll post a final version by the end of Friday.

Answer the questions in your answer book. You may consult any printed material (books, notes, etc).

Question 1: OpenMP

Here are some potential questions. If there is an OpenMP question, it will look like these.

- Explain what each of the OpenMP pragmas does in an example like:

  `http://people.sc.fsu.edu/~jburkardt/c_src/heated_plate_open_mp/heated_plate_open_mp.c`

  In particular, assume that `OMP_NUM_THREADS` is 4 and draw a diagram explaining what the threads do for each of the parallel sections.

- Pretend that I've removed the pragmas from the above code. Add back appropriate OpenMP pragmas. (I might also do that for code which needs tasks.)

- Pretend I've removed visibility qualifiers from the code; manually infer the appropriate ones and explain why you chose them. (Or you might have to choose between `firstprivate` and `copyin`, etc.)

- Show me the difference between parallel sections and tasks using an example. You might also use a diagram here.

Question 2: Making locks fine-grained

Consider the following code from the Linux kernel:

```
// include/linux/list.h
#define list_for_each_entry(pos, head, member)                          \
        for (pos = list_entry((head)->next, typeof(*pos), member),      \
                     prefetch(pos->member.next);                        \
             &pos->member != (head);                                    \
            pos = list_entry(pos->member.next, typeof(*pos), member),\
                    prefetch(pos->member.next))

// fs/file_table.c
int fs_may_remount_ro(struct super_block *sb)
{
        struct file *file;

        /* Check that no files are currently opened for writing. */
        spin_lock(&files_lock);
        list_for_each_entry(file, &sb->s_files, f_u.fu_list) {
                struct inode *inode = file->f_path.dentry->d_inode;

                /* File with pending delete? */
                if (inode->i_nlink == 0)
                        goto too_bad;

                /* Writeable file? */
                if (S_ISREG(inode->i_mode) && (file->f_mode & FMODE_WRITE))
                        goto too_bad;
        }
        spin_unlock(&files_lock);
        return 1; /* Tis' cool bro. */
too_bad:
        spin_unlock(&files_lock);
        return 0;
}
```

Point out uses of the spin lock and explain the problems with using that lock. Describe and explain the modifications you'd need to make to replace the spin lock with a finer-grained lock. You can look up the code modification on the Internet, but be aware that I'll use slightly different code on the exam.

Question 3: Memory Barriers and Consistency Models

- Consider the following code; all variables are initially 0.

  ```
  T1: x = 1; r1 = y;
  T2: y = 1; r2 = x;
  ```

  Assume the architecture is not sequentially consistent. Show me all possible memory values and how they arise.

- Where should you put a barrier in the following code? Why? What type of barrier should you use?

```
    q = &a;
    if (p)
        q = &b;
    x = *q;
```

Show me possible outputs without a barrier and with a barrier; identify the outputs we clearly wouldn't expect. (credit: Linux kernel, `Documentation/memory-barriers.txt`. Good reading; I should've assigned it.)

Question 4: Compiler Optimizations

Here is some C code from meschach.

```
double  v_min(VEC *x, int *min_idx)
{
    int         i, i_min;
    Real        min_val, tmp;

    if ( ! x )
        error(E_NULL,"v_min");
    if ( x->dim <= 0 )
        error(E_SIZES,"v_min");
    i_min = 0;
    min_val = x->ve[0];
    for ( i = 1; i < x->dim; i++ )
    {
        tmp = x->ve[i];
        if ( tmp < min_val )
        {
            min_val = tmp;
            i_min = i;
        }
    }

    if ( min_idx != NULL )
        *min_idx = i_min;
    return min_val;
}
```

- Describe 2 compiler optimizations that could apply to the code and the resulting code after optimization. Briefly summarize the conditions that need to hold for the optimizations to be safe; you may add qualifiers to the code if you want.

- Write a profile of this function's execution on a small vector. Use this to propose profile-guided optimizations on the code.

Question 5: GPU Programming

Convert the following code to run as an OpenCL kernel. Indicate the floating-point operations in the kernel. Do you need to worry about synchronization?

```
# define M 500
# define N 500

  int i, j, iterations;
  double diff;
  double u[M][N];
  double w[M][N];

  diff = 0.0;
  for ( i = 1; i < M - 1; i++ )
  {
    for ( j = 1; j < N - 1; j++ )
    {
      w[i][j] = ( u[i-1][j] + u[i+1][j] + u[i][j-1] + u[i][j+1] ) / 4.0;

      if ( diff < fabs ( w[i][j] - u[i][j] ) )
      {
        diff = fabs ( w[i][j] - u[i][j] );
      }
    }
  }
  iterations++;
```

(From `http://people.sc.fsu.edu/~jburkardt/c_src/heated_plate/heated_plate.c`.)

Question 6: MPI

There is an MPI program with a bug here:
`https://computing.llnl.gov/tutorials/mpi/samples/C/mpi_bug1.c`
If I choose this question for the exam, I will reproduce the code from `mpi_bug1.c` (no substitutions) and ask you to describe and fix the bug. (Jon has installed OpenMPI on the `ece459-N` computers; let me know if it doesn't work.)

Question 7: MapReduce/Hadoop

I looked around for MapReduce examples that are not word count. If this question appears on the exam, you will need to provide pseudocode for the mapper and the reducer. The set of possible questions that I will ask you, drawn from the Hadoop distribution, is:

- Sort input values.
- Estimate π using quasi-monte-carlo integration.
- Count the number of pageviews in a log; input is `<url, referrer, time>` and output is `<url, pageview>`.

Question 8: Reduced-Resource Computation

- Perforate the `v_min` code above and analyze the change in running time. Explain the conditions under which loop perforation could possibly work, and discuss the expected results of the perforated loop.
- How would you apply early-phase termination to the quasi-monte-carlo integration you would have implemented above in the MapReduce example? Would you need to do any correction in the results?