Prof. Jingke Li (FAB 120-06, li@cs.pdx.edu); Class: MW 2:00-3:50pm @ FAB 40-07; Office Hr: MW 1-2pm & by appt.

Lab 4: Midterm Practice Problems

1. [Shared-memory programming] The following parallel routine (in a pseudo language) is supposed to compute the sum of the first `n` elements of array `a`. The semantics of the `forall` loop is that a separate thread is created for each "iteration" and all threads join back at the end of the loop.

```
int summation(int a[], int n) {
  int i, sum = 0;
  forall (i=0; i<n; i++)
    sum = sum + a[i];
  return (sum);
}
```

   Running on a parallel system, this program does not always behave as expected — sometimes it returns the correct sum, some times it returns a wrong sum.

   (a) Show a scenario that the routine will return a wrong sum.

   (b) Assume `n=4` and `a[i]=i, 0<=i<n`. Show as many possible return values from this routine as you can.

2. [Pthreads] Consider the following Pthreads program:

```
void grandson() { printf("grandson\n"); }
void son(int *ip) {
  pthread_t t3;
  pthread_create(&t3, NULL, (void*)grandson, NULL);
  printf("son %d\n", *ip);
  pthread_join(t3, NULL);
}
int main() {
  int i=1, j=2;
  pthread_t t1, t2;
  pthread_create(&t1, NULL, (void*)son, &i);
  pthread_create(&t2, NULL, (void*)son, &j);
  pthread_join(t1, NULL);
  pthread_join(t2, NULL);
  printf("Work done\n");
}
```

   (a) What are possible outputs of this program? Either describe or enumerate.

   (b) What are the purpose of those `pthread_join` calls? What happens if they are removed?

3. [OpenMP] Each of the following programs contains an OpenMP-related problem. For A, the compile indicates an OpenMP directive error; for B, the problem is that the directives do not seem to have any effect on the program's performance. Identify the error in each program.

A:

```
#define N 128

int main(int argc, char **argv) {
  int i, tid, a[N];

  #pragma omp parallel for
  {
    tid = omp_get_thread_num();
    printf("tid=%d\n", tid);
    for (i=0; i<N; i++)
      a[i] = i;
  }
}
```

B:

```
// ... other routines omitted ...

void quicksort(int* array, int left,
               int right) {
  if (left >= right) return;
  int middle = partition(array,left,right);
  #pragma omp task
  quicksort(array, left, middle-1);
  quicksort(array, middle+1, right);
  #pragma omp taskwait
}
int main(int argc, char **argv) {
  int N, *array;
  array_init(array);
  quicksort(array, 0, N-1);
}
```

4. [Synchronization] In multi-threaded programming, a barrier for a group of threads serves to synchronize all of them to the same temporal point — any thread must stop at this point and cannot proceed until all other threads reach this point. Consider the attempt to create a simple barrier routine using a binary lock in (a):

(a)

```
boolean L;      // a lock
int count = n; // n=#threads
void Barrier() {
  Lock(L);
  count--;
  Unlock(L);
  while (count != 0) {}
}
```

(b)

```
boolean L;
int count = n;
void Barrier2() {
  Lock(L);
  if (count == 0)
    count = n; // reset counter
  count--;
  Unlock(L);
  while (count != 0) {}
}
```

(a) Although this implementation may appear to work, it actually has a serious error. Describe the nature of this error. Give a detailed scenario which results in erroneous behavior of this `Barrier` routine.

(b) Code in (b) is an attempt to fix the problem. Yet, it is still not correct. Describe what is wrong this time, and give a scenario which results in erroneous behavior.

5. [Fortran 90/95] Consider the following Fortran 95 code segment:

```
A = (/ 0,1,2,3,4 /)
B = (/ 4,3,2,1,0 /)
forall (i = 2:4)
  A(i) = B(i+1) + A(i-1)
end forall
```

Recall that Fortran 95 semantics says that for a statement inside a `forall` loop, all reads must happen before any write.

(a) What values does array `A` hold after the execution of the `forall` loop?

(b) Write a single statement using array section operations (*e.g.* the triplet notation) to achieve the same result as this `forall` loop does.

(c) If we want to execute the code segment on a sequential machine, can the compiler convert the `forall` loop into the sequential `do` loop (a)? Why or why not?

(d) What about converting it into the descending `do` loop (b)? Why or why not?

```
(a) do i = 2, 4, 1
      A(i) = B(i+1) + A(i-1)
    end do
```

```
(b) do i = 4, 2, -1
      A(i) = B(i+1) + A(i-1)
    end do
```