Example test questions

Questions about OpenMP:

```
#pragma omp parallel private(i)
for (int i = 0; i < 100; i++_ {
   a[i] = i;
}
```

How many iterations are executed if four threads execute the above program?

25, as the loop is split among the four threads.

True or false: Code in an OpenMP program that is not covered by a pragma is executed by all threads

False, it is executed by a single thread.

Questions about MPI:

For these questions, assume the following data for the integer a and the distributed integer array *b* on different processors. For the entire array b, b[i] == i. The array c is initially empty on each processor.

P0: a=0, b = [0, 1, 2, 3]

P1: a=1, b = [4, 5, 6, 7]

P2: a=2, b=[8, 9, 10, 11]

P3: a=3, b=[12, 13, 14, 15]

Was b block, cyclic or block cyclic distributed?

Block distributed as can be seen in the elements on each processor.

If the command

MPI_Gather(b, 4, MPI_INT, c, X, MPI_INT, 3, MPI_COMM_WORLD);
is executed, what should the recvcnt (denoted X) above be? 2, 4, 8, 16, 32?

4 -- **recvcnt** is the maximum received from any single receive.

Which process will receive the data? 0, 1, 2, 3, or all?

Process 3.

If the command

MPI_Reduce(b, c, 4, MPI_INT, MPI_SUM, 2, MPI_COMM_WORLD);

is executed, what variable receives the result of the reduction?

a

b

c

cannot tell without having the entire program.

c receives the reduction

How many results will be produced? 1, 2, 4, 8, 16 or 32.

4 results, i.e., process with rank == 2 have the result of reducing b[0] on each process, b[1] on each process, b[2] on each process and b[3] on each process.

Which process will receive the data? 0, 1, 2, 3, or all?

Process 2

Questions about Isoefficiency:

A parallel job is running on a collection of 64 machines that have 4GB of physical memory. For efficiency reasons it is essential that the job on each node fit into 4GB. The job currently uses 2GB of memory on each node.

What is the largest isoefficiency relationship of the form $W = c*P$, where c is an integer constants, that will allow the job to scale to 128 processors?

For the c*P problem the problem is completely scalable. To see this better consider going from 4 to 8 processors. The work on 4 processors is ~ to c*4 and the work on 8 processors is c*8. Dividing c*8/c*4 gives 2. The number of processors has doubled and the overall memory has doubled and so the problem fits with the same footprint.

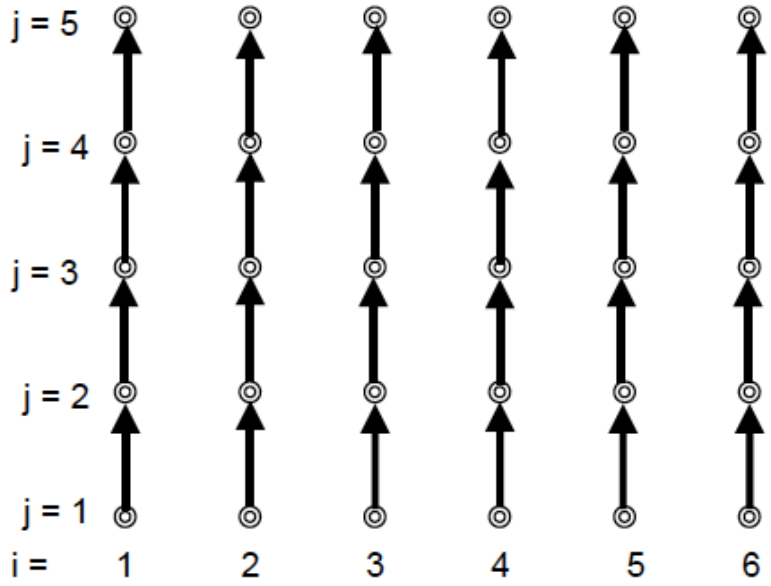
The actual isoefficiency relationship is $W = P^2$. How much memory is needed to allow the job to scale to 128 processors?

$128^2 / 64^2 = 4$, so 4 times as much memory would be needed overall. There are twice as processors and consequently twice as much memory available, so the memory footprint on each node is twice as much as it was with 64 processors.

Questions about parallelism and optimization:

Given the loop and iteration space diagram:

```
for (i=0; i < n; i++) {
  for (j=0, j<n; j++) {
    a[i][j] = a[i][j-1]
  end do
end do
```



Is the *i* loop parallel?

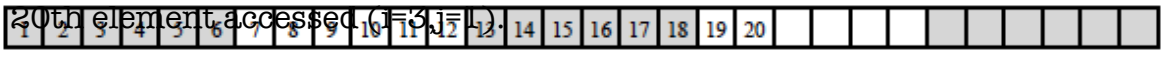
Yes -- no dependences cross iterations of the i loop.

Is the *j* loop parallel?

No -- dependences cross iterations of the j loop.

If the array is laid out such that adjacent row elements are next to each other in memory, does the access have good or poor cache locality?

Good cache locality -- elements along a row are accessed most rapidly, so each access accesses the memory location next to it. The figure below shows the order elements accessed by the first 20 executions of a[i][j] assuming n=6. That is, the first array element contains a 1 because it is the element that is accessed first (i=0,j=0), the second element contains a 2 because it is the second element accessed (i=0,j=1) and the 20th array element contains a 20 because it is the 20th element accessed (i=3,j=1).



Lock, races and deadlock questions:

A program is written in a Java-like language were every variable has a lock. The statement:

spawn(function, data1, data2, tid);

creates a thread whose id is *tid* that executes *function* on data *data1* and *data2.*

The statement

lock(location);

acquires a lock on *location. location* can be a variable, array element, struct element, etc.

The statement

unlock(variable);

releases a lock on a variable .

Given the code below:

```
int locks[100];
float data[1000000];

#pragma par for
for (i=0; i < 1000000; i++) {
  lock(locks[g(i)%100]);
  f(data[g(i)]);
  unock(locks[g(i)%100]);
}
```

Assume the function f only accesses its argument, may read and write it, and does not synchronization internally.

How many threads can be active at any time?

Up to 100 as up to 100 threads can acquire different locks. After that threads must access the same lock leading to the 101st thread waiting to acquire a lock.

Assuming locking is free, what is the maximum speedup?

100.

Assuming locking is free, is it possible that the loop executes sequentially because of locking?

Yes. Consider the case where g(i) always returns 4. The lock for locks[4] would be accessed by all threads.

Given the loop:

```
#pragma par for
for (i=0; i < 1000000; i++)
  f(data[g(i)]);
}
```

Assume the function f only accesses its argument, may read and write it, and does not synchronization internally.

what can you say about g that would make this program race free?

If i ≠ i' implies g(i) ≠ g(i') all iterations of the loop access different elements of data.

Given the loops:

```
#pragma par for private (t, j)
for (i=0; i < 1000000; i++) {
  t = my_thread_id( ) // t is assigned a unique integer thread id
  for (j=0; j < 1000000; j++) {
    data[j] = t;
  }
}
```

Circle all that are true when the program executes on four threads and values assigned to t are 0, 1, 2 and 3:

Note that four threads each execute a copy of the inner j loop, thus there are races among the different threads on the update to data[j].

1. **There is no race** false, see above.
2. **There is a race** true, see above.
3. **Elements of data will all have a value of 0, 1, 2 or 3** false for C++ (see 5 below), true for Java
4. **Elements of data can have any value** True for C++ (see 5 below), false for Java
5. **If this is a C++ program, because of "catch fire" semantics the answers to 3 and 4 depend on the particular compiler and runtime** C++ semantics say that in the case of races the program can have any outcome.

3 and 4 could have been worded better to eliminate the C++/Java differences and would be so on the test.