Name:________________________

NetID:________________________

CS420/CSE402/ECE492
Parallel Programming for Scientists and Engineers

Exam 3

December 11, 2013

(12 pages / 28 questions total)

Instructions:

Complete all parts of the test. For multiple choice questions circle the best answer. Note that there are a few multiple answer questions. The points per question for each section is listed at the beginning of each section, but some questions are specified as having less points.

Question	Parts	Max	Points
1	7	21	
2	7	24	
3	8	31	
4	6	24	

Section I – OpenMP (3 points each):

1. The following code adds all the elements in an array a and puts the result into sum. This could be done most easily and efficiently by replacing the current pragmas with

```
sum = 0;
#pragma omp parallel private(i,nthreads,tid,mysum) shared(sum,a)
{
   nthreads = omg_get_num_threads();
   tid = omp_get_thread_num();
   for (i = num/nthreads*tid; i < num/nthreads*(tid+1); i++) {
      mysum += a[i];
   }
   #pragma omp atomic
   sum += mysum;
}
```

A. A parallel for
B. A parallel for and an atomic
C. A parallel for with a reduce
D. A parallel for with lastprivate(sum)

2. Which of the following is a reasonable way to ensure that a block of code is executed only once?

A. Put a barrier before the block
B. Put barriers before and after the block
C. Put the block in a critical section
D. Put the block in a master section

3. When accessing an array read-only inside a parallel region, the array should be

A. shared
B. private
C. firstprivate
D. lastprivate

4. Which statement about using atomic or critical statements to protect data is true?

A. You should use one any time you access shared memory.
B. There is no drawback to using them to protect private memory.
C. All statements that write to shared memory should be protected.
D. They are unnecessary for a fast operation (e.g. val++).

5. Which of the following is a problem when using more processors to run an OpenMP program?

A. It can create new race conditions.
B. Deadlocks can be introduced.
C. It introduces additional complexity to the code.
D. Accessing protected memory incurs longer waits.

6. Which OpenMP scheduling clause varies the chunk-size of data given to a processor at runtime?

A. Static
B. Dynamic
C. Guided
D. None of the above

7. Where should a barrier be placed to ensure the below code executes correctly?

```
value = 0;
#pragma omp parallel shared(value) private(tid,myval)
{
    myval = 0;
    tid = omp_get_thread_num();


    // Location 1
    myval = some_function(tid);

    // Location 2
    #pragma omp atomic
    value += myval


    // Location 3
}
printf("%d",value);
```

A. Location 1
B. Location 2
C. Location 3
D. No barrier is needed.

Section II – MPI (4 points each):

1. An MPI program splits up an array into even parts and sends each part to a different process. The processes then perform some computations on their part of the array. Finally, *every* process needs to get the entire updated array. What is the simplest collective call to use?

A. MPI_Alltoall
B. MPI_Alltoallv
C. MPI_Bcast
D. MPI_Allgather

2. To force only even numbered processes to stop at a certain barrier, a programmer could

A. Call MPI_Barrier() with a communicator containing even processes.
B. Have each even process call MPI_Barrier() twice.
C. Have odd processes call MPI_Barrier() with the MPI_NOWAIT option.
D. It is not possible. All processes must stop at an MPI_Barrier() call.

3. True/False: It is impossible to deadlock if you only use MPI_Isend()/MPI_Irecv() and MPI_Wait(). (2 points)

A. True
B. False

Use the following options to answer the next two questions:

A. Can parallelize code on shared memory systems.
B. Each process has its own local variables.
C. Allows incremental parallelism.
D. Can parallelize code on distributed memory systems.
E. Requires explicit parallelism, often providing better performance.
F. Can run the program as a serial code.
G. Can be used for hybrid parallelizations.
H. Data layout and decomposition is handled automatically by directives.
I. Provides an adaptive runtime system.

4. Which three of the options above are advantages of using MPI over OpenMP? (3 points)

5. Which three of the options above are advantages of using OpenMP over MPI? (3 points)

6. Will the following code correctly broadcast the value 23 to all MPI processes?

```
int numtasks, taskid, len, buffer, root, count;
char hostname[MPI_MAX_PROCESSOR_NAME];

MPI_Init(&argc, &argv);
MPI_Comm_size(MPI_COMM_WORLD, &numtasks);
MPI_Comm_rank(MPI_COMM_WORLD,&taskid);
root = 0;
buffer = 0;
if(taskid==root)
    buffer = 23;
count = taskid;

MPI_Bcast(&buffer, count, MPI_INT, root, MPI_COMM_WORLD);
printf("Rank=%d Result=%d\n",taskid,buffer);

MPI_Finalize();
```

A. Yes, all processes will receive and print the value 23.
B. No, all processes will receive a value, but the value will not be 23 on all processes.
C. No, the code will hang due to having an incorrect size argument.
D. No, the code will deadlock due to using the incorrect root.

7. Will the following code scatter the array so that process 0 receives [1.0, 2.0, 3.0, 4.0], process 1 receives [5.0, 6.0, 7.0, 8.0], etc? (Assume the code is executed with 4 MPI processes)

```
#define SIZE 4
int main (int argc, char *argv[])
{
int numtasks, rank, sendcount, recvcount, source;
float sendbuf[SIZE][SIZE] = {
{1.0, 2.0, 3.0, 4.0},
{5.0, 6.0, 7.0, 8.0},
{9.0, 10.0, 11.0, 12.0},
{13.0, 14.0, 15.0, 16.0} };
float recvbuf[SIZE];

MPI_Init(&argc,&argv);
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
MPI_Comm_size(MPI_COMM_WORLD, &numtasks);

MPI_Scatter(sendbuf,SIZE,MPI_FLOAT,recvbuf,SIZE,
            MPI_FLOAT,1,MPI_COMM_WORLD);

printf("rank= %d  Results: %f %f %f %f\n",rank,recvbuf[0],
      recvbuf[1],recvbuf[2],recvbuf[3]);

MPI_Finalize();
}
```

A. Yes, this code scatters the array so that process 0 receives [1.0, 2.0, 3.0, 4.0], process 1 receives [5.0, 6.0, 7.0, 8.0], etc..

B. No, the scatter will execute but process 0 will receive [1.0, 5.0, 9.0, 13.0], process 1 will receive [2.0, 6.0, 10.0, 14.0], etc.

C. No, the code will deadlock.

D. No, the code will only scatter the first four elements of sendbuf, resulting in process 0 receiving [1.0], process 1 receiving [2.0], etc.

Section III – Advanced Parallel Programming (4 points each):

1. In the following HPF code fragment, on which processor will the addition of the i'th element execute?

```
forall i=1 to N
   C[i] = B[i]+C[i] +A[i]
```

A. It is up to the compiler where to do the addition.
B. on the processor that holds A[i].
C. on the processor that holds B[i].
D. on the processor that holds C[i].

2. True/False: Charm++ is used to program a single node, like OpenMP. (3 points)

A. True
B. False

3. Which of the following statements are true about GPU programming with CUDA? (Choose all that apply).

A. Each thread within a block is mapped to a warp, with all threads within a warp being executed in parallel.
B. The number of threads per block needs to be equal to the warp size in order to hide memory latency and achieve the best performance.
C. Threads within a block can share data using shared memory.
D. GPU kernel functions can handle control flow divergence without reducing performance.

4. Which of the following is NOT a drawback to using GPUs?

A. Data must be passed between the CPU and the GPU.
B. Code can require major modifications to convert from OpenMP or MPI to a GPU language.
C. GPU codes require a large number of independent tasks to be effective.
D. GPU kernels use a SIMT model.

6. What are benefits of using PETSc for scientists? (Choose all that apply).

A. PETSc implements functions that avoid time-consuming routines from MPI such as global collectives.
B. PETSc is layered on top of MPI, providing additional functionality to users.
C. PETSc does not require much knowledge about MPI, so people with limited knowledge of parallel programming can still take advantage of the library.
D. PETSc uses polymorphism to show implementation details through the public interface.

7. When could you obtain improved performance by converting a pure MPI code to a hybrid MPI/OpenMP code? (Choose all that apply)

A. When you can reuse data in shared caches.
B. When you can improve MPI performance through sending larger messages.
C. When running on a machine where each node has one core.
D. When you have low overhead in the MPI code.

8. We are comparing binary-exchange algorithm for parallel FFT with 2-D transpose based parallel FFT. Check all statements below that are correct:

A. The amount of data leaving any given processor over the lifetime of the algorithm is the same for both algorithms
B. 2D Transpose based FFT involves fewer communication steps than binary-exchange based algorithm
C. 3D transpose based FFT involvers more communication steps than binary-exchange based algorithm
D. They are not comparable because they compute different results (i.e. their outputs are different)
E. The total number of arithmetic operations done in both algorithms is proportional to N log N, where N is the number of data values

9. Assume that you have set the optimal checkpoint period, say T, for your application using the model formula discussed in the class. Now, you are told that the machine's reliability has improved so that the mean time between failure has quadrupled. What should your new checkpoint period be? You will not be given the formula. Your understanding of the basics of the formula is adequate to answer this.

A. T (unchanged)
B. 2T
C. 0.5T
D. 4T

Section IV – Short Answer: (4 points each)

1. In OpenMP, why do we not set our thread count arbitrarily high to allow for maximum parallelism? Explain in one or two sentences.

2. The following code is part of a CAF (Co-array Fortran) code to sort numbers based on an odd-even sort (The details of how sorting will work here are unimportant to the question). Assume that the “images” (analogues to ranks in MPI) are numbered starting with 1.

```
REAL X[*]
...
IMG = THIS_IMAGE()
....
if (isEven(IMG))
IF (X < X[IMG-1])
  Tmp = X
  X = X[IMG-1]
  X[IMG-1] = Tmp
ENDIF
...
```

Describe in words what the code fragment does.

To recap: A Charm++ program consists of objects that communicate with each other via asynchronous method invocations (similar to "sending a message" to another object).

Over-decomposition: There are typically many objects assigned to each processor core.

Migratability: the programmer does not specify which object is housed on which processor. So, the system is free to migrate (i.e. move) individual objects from one processor to another.

Given this model, explain (in 1 to 3 sentences per question):

3. How does this model allow the runtime system to dynamically balance load?

4. How does this model help reduce the time lost to communication?

The following code implements matrix-matrix multiplication (C = A * B) and is used to answer the next two questions.

```
#define NRA 62 /*rows in A */       #define NCA 15 /*columns in A */
#define NCB 7 /*columns in B */   #define MASTER 0 /* taskid of first task */
#define FROM_MASTER 1          #define FROM_WORKER 2

int numtasks, taskid, numworkers, dest, rows, averow, extra, offset, i, j, k, rc;
double a[NRA][NCA],b[NCA][NCB], c[NRA][NCB];
MPI_Status status;
MPI_Init(&argc,&argv);
MPI_Comm_rank(MPI_COMM_WORLD,&taskid);
MPI_Comm_size(MPI_COMM_WORLD,&numtasks);
numworkers = numtasks-1;
if (taskid == MASTER)
{
   for (i=0; i<NRA; i++)
      for (j=0; j<NCA; j++)
         a[i][j]= i+j;
   for (i=0; i<NCA; i++)
     for (j=0; j<NCB; j++)
         b[i][j]= i*j;

   averow = NRA/numworkers;
   extra = NRA%numworkers;
   offset = 0;
   for (dest=1; dest<=numworkers; dest++) {
      if(dest<=extra) rows = averow+1;
      else rows = averow;
      MPI_Send(&offset, 1, MPI_INT, dest, FROM_MASTER, MPI_COMM_WORLD);
      MPI_Send(&rows, 1, MPI_INT, dest, FROM_MASTER, MPI_COMM_WORLD);
      MPI_Send(&a[offset][0], rows*NCA, MPI_DOUBLE, dest, FROM_MASTER,MPI_COMM_WORLD);
      MPI_Send(&b, NCA*NCB, MPI_DOUBLE, dest, FROM_MASTER, MPI_COMM_WORLD);
      offset = offset + rows;
   }

/////////////// MASTER_MARKER ////////////////

   for (i=1; i<=numworkers; i++) {
      MPI_Recv(&offset, 1, MPI_INT, i, FROM_WORKER, MPI_COMM_WORLD, &status);
      MPI_Recv(&rows, 1, MPI_INT, i, FROM_WORKER, MPI_COMM_WORLD, &status);
      MPI_Recv(&c[offset][0], rows*NCB, MPI_DOUBLE, i, FROM_WORKER, MPI_COMM_WORLD, &status);
   }
}

if (taskid > MASTER)
{
   MPI_Recv(&offset, 1, MPI_INT, MASTER, FROM_MASTER, MPI_COMM_WORLD, &status);
   MPI_Recv(&rows, 1, MPI_INT, MASTER, FROM_MASTER, MPI_COMM_WORLD, &status);
   MPI_Recv(&a, rows*NCA, MPI_DOUBLE, MASTER, FROM_MASTER, MPI_COMM_WORLD, &status);
   MPI_Recv(&b, NCA*NCB, MPI_DOUBLE, MASTER, FROM_MASTER, MPI_COMM_WORLD, &status);
```

```
    //////////// WORKER_MARKER //////////////

    for (k=0; k<NCB; k++)
        for (i=0; i<rows; i++){
            c[i][k] = 0.0;
            for (j=0; j<NCA; j++)
                c[i][k] = c[i][k] + a[i][j] * b[j][k];
        }
    }
    MPI_Send(&offset, 1, MPI_INT, MASTER, FROM_WORKER, MPI_COMM_WORLD);
    MPI_Send(&rows, 1, MPI_INT, MASTER, FROM_WORKER, MPI_COMM_WORLD);
    MPI_Send(&c, rows*NCB, MPI_DOUBLE, MASTER, FROM_WORKER, MPI_COMM_WORLD);
}
```

5. How are matrix A and matrix B allocated and initialized on each process? Specifically, what is the state of matrix A and matrix B at MASTER_MARKER and WORKER_MARKER on each process.

6. Why is this a reasonable domain decomposition scheme for the given problem size? What changes would be needed if NRA = NCA = NCB = a large number to ensure that the code is efficient and that you do not run out of memory for larger problem sizes? Describe the changes. You do not need to write code.