UNIVERSITY OF RUHUNA

BACHELOR OF COMPUTER SCIENCE (BCS) (SPECIAL) DEGREE (SEMESTER I)

EXAMINATION – AUGUST 2016

COURSE UNIT: CSC4172 - High Performance Computing **TIME: 2 Hours**

Answer all four Questions.

1.

a. Describe 3 ways to find the performance bottlenecks of a serial program for which the source code is available.

b. Suppose you execute a parallel program on a distributed memory computer and noticed the performance is not high as expected. Explain four sources of performance degradation that you can identify.

c. Consider the following code segment used for matrix multiplication and explain how cache misses can occur during the execution of the code segment. Assume cache line size (in number of elements) is L

```
for (i=0;i<N;i++)
  for (k=0;k<N;k++)
    for (j=0;j<N;j++)
    c[i,j]+=a[i][k]*b[k][j];
```

d. Write optimized versions of the following code segments assuming N is known.

i.
```
for (i=1;i<N;i++)
    x[i] = 2*a*i/k1;
    y[i]  =  2*a*i/k2;
```

ii.
```
for (j=0;j<N;j++)
    a[i][j] = 2;
```

iii.
```
for (i=1; i<5000;i++)
    p[i] = a * b * x[i];
```

iv.
```
x1 = (p * q) * r;
x2 = (p * q) * s;
```

2.

a. Distinguish between data parallelism, functional parallelism and task parallelism giving examples.

b. Suppose 80% of a certain program has inherent parallelism.
   i. What is the maximum speed up obtainable if run on a multiprocessor?
   ii. If the program took 10 seconds to run serially, what is the parallel execution time on 8 processors?

c. State two reasons for a super linear speed up in parallel execution.

d. Describe four sources of inefficiency that may occur in parallel codes.

3.

a. Describe how you could use *scatter* and *gather* operations in MPI to multiply two matrices in parallel.

b. Suppose there are six processes P0 to P5 and each process has declared an array of size six named A.
   i. Explain the effect of a call to *MPI_Gather*, process 0 as the root on the data in array A.
   ii. What is the effect of a call to *MPI_Allgather?*

c. Explain briefly the function of the following MPI statements.
   i. MPI_Comm_rank(comm, rank)
   ii. MPI_Bcast(buf, count, datatype, root,com)
   iii. MPI_Irecv(buf, count, datatype, source, tag, comm, request)
   iv. MPI_Reduce(sendbuf, recvbuf, count, datatype, op, root, comm)

d. Explain whether the following MPI code segment executed by two concurrent processes P0 and P1, is correct or wrong giving reasons.

Process 0 executes:
```
MPI_Recv(&newdata,     1,      MPI_FLOAT,      1,      tag,
MPI_COMM_WORLD, &status);
MPI_Send(&olddata,     1,      MPI_FLOAT,      1,      tag,
MPI_COMM_WORLD);
```

Process 1 executes:

```
MPI_Recv(&newdata,      1,      MPI_FLOAT,      0,      tag,
MPI_COMM_WORLD, &status);
MPI_Send(&olddata,      1,      MPI_FLOAT,      0,      tag,
MPI_COMM_WORLD);
```

4.

a. State four objectives of using compiler directives in OpenMP.

b. What is the purpose of using the *critical* directive in OpenMP? What happens if you access shared variables outside a *critical* directive?

c. Explain the functionality of following OpenMP code fragments.

i.

```
int nthreads, tid;

#pragma omp parallel private(tid)
{

tid = omp_get_thread_num();
printf("Hello World from thread = %d\n", tid);

if (tid == 0)
{
nthreads = omp_get_num_threads();
printf("Number of threads = %d\n", nthreads);
}
}
```

ii.

```
chunk = 100;
 #pragma omp parallel shared(a,b,c,chunk) private(i)
            {
       #pragma omp for schedule(dynamic,chunk) nowait
          for (i=0; i < 1000; i++)
             c[i] = a[i] + b[i];
     }
```

iii.

```
N = 1000;
#pragma omp parallel shared(a,b,c,d) private(i)
 {

 #pragma omp sections nowait
  {

  #pragma omp section
  for (i=0; i < N; i++)
    c[i] = a[i] + b[i];

  #pragma omp section
  for (i=0; i < N; i++)
    d[i] = a[i] * b[i];

  }

 }
```

d. Explain three advantages of using hybrid MPI/OpenMP paradigm.