EECS 739: Exam # 1
Friday, March 22, 2019

Academic Integrity Statement
I have neither given nor received assistance on this exam.

Print Name______________________________
Signature______________________________

The rules for this exam are as follows:

• Write your name on the front page of the exam booklet. Initial each of the remaining pages in the upper-right hand corner. Sign the academic integrity statement on the front of the exam booklet. Your exam will not be graded if you have not signed the front page of the booklet.

• The exam has 5 questions and 1 extra credit question. The exam is 7 pages long (including this page). Be sure you have all of the pages before beginning the exam.

• This exam will last for 75 minutes.

• Show ALL work for partial/full credit. This includes any definitions, mathematics, figures, etc.

• The exam is closed book and closed notes.

• Calculators may be used only for manual calculations. They may not be programmed with method or notes.

• Cell phones can only be used for keeping track of time during the exam.

• No collaboration of any kind is allowed on the exam.

1. ______ (15 points)  
2. ______ (15 points)  
3. ______ (15 points)  
4. ______ (15 points)  
5. ______ (15 points)  
EC. ______ (7 points)  
T. ______ (75 points)
1. (15 points; 1.5 points each) Write **TRUE** or **FALSE** to the **LEFT** of each question. Do **NOT** include any additional justification.

(a) There is an inverse relationship between degree of concurrency and task granularity.

(b) Amdahl’s law specifies a lower bound on the amount of speedup that can be achieved.

(c) Task interaction graphs can be used to determine how to distribute the data among processors in order to minimize the amount of memory used.

(d) With speculative decomposition, tasks are scheduled even though they may potentially be erroneous.

(e) Load balancing is sufficient to minimize processor idling.

(f) Blocking communication primitives should be used when overlapping communication with computation.

(g) In order to obtain greater efficiency, it is important to consider the ordering of the triple loop in the parallelization of the Gaussian Elimination method.

(h) MPI does not allow the user to write his/her own functions.

(i) MPI_Allgather is the same as an MPI_Gather followed by an MPI_Bcast.

(j) Isoefficiency allows us to determine at what rate the problem size should be increased with respect to the number of processors in order to maintain efficiency.
2. (15 points; 3 points each) Consider the task dependency graph given below. Determine the following properties and give a brief justification for each.

(a) The maximum degree of concurrency.

(b) The critical path length.

(c) Maximum achievable speedup over one process assuming that an arbitrarily large number of processors is available.

(d) The minimum number of processors needed to obtain the maximum possible speedup.

(e) The maximum achievable speedup if the number of processors is limited to 4.
3. (15 points) In lecture, we studied the parallel cyclic reduction algorithm and its use in solving tridiagonal linear systems. We studied the case when \( n = 2^p - 1 \), where \( p \) is an integer; \( n \) is the number of linear equations, and \( P = n \) processors were available. How would the parallel algorithm change for the case when \( P = (n - 1)/2 \) processors were available? In particular, answer the two questions below.

(a) (10 points) Draw and label a task-dependency diagram illustrating how to solve the linear system in an efficient manner. This should include both the reduction phases and the solution phases. Your diagram should illustrate the case when \( n = 7 \) and \( P = 3 \) processors are available.

(b) (5 points) Write a brief summary of your approach.
4. (15 points) One way to compute \( \pi \) is to evaluate the following integral:

\[
\int_0^1 \frac{4}{1 + x^2} \, dx = 4 \cdot \tan^{-1}(1) = \pi.
\]

Thus, \( \pi \) can also be approximated by numerical integration using the midpoint rule as follows:

\[
\int_0^1 f(x) \, dx \approx \sum_{i=1}^{n} f\left((i-0.5) \cdot h\right),
\]

where \( h \) is the width of the subinterval. This concept is shown for \( n = 13 \) below.

Suppose that \( p \) processors are running a single parallel code. Write a snippet of MPI + C/C++ code that efficiently computes an approximation to \( \pi \) using the midpoint rule with \( n \) subintervals without using any of the following functions: MPI_Send, MPI_Recv, MPI_Isend, or MPI_Irecv. Your code should store the answer on a single processor.

You may make the following assumptions:

(a) MPI_Init, MPI_Comm_size, and MPI_Comm_rank have all been called.
(b) All variables have been declared (but not initialized with values).
(c) The value of \( n \) is known only on processor 0.
(d) Your code does not need to call MPI_Finalize.
5. (15 points) Consider the following output from running an MPI + C/C++ code on an HPC cluster for purposes of performing scaling experiments:

<table>
<thead>
<tr>
<th>n</th>
<th>1</th>
<th>2</th>
<th>4</th>
<th>8</th>
<th>16</th>
<th>32</th>
</tr>
</thead>
<tbody>
<tr>
<td>64</td>
<td>1,000</td>
<td>582</td>
<td>310</td>
<td>200</td>
<td>156</td>
<td>102</td>
</tr>
<tr>
<td>128</td>
<td>2,078</td>
<td>1,010</td>
<td>689</td>
<td>412</td>
<td>255</td>
<td>175</td>
</tr>
<tr>
<td>256</td>
<td>5,100</td>
<td>2,600</td>
<td>1,420</td>
<td>780</td>
<td>420</td>
<td>250</td>
</tr>
<tr>
<td>512</td>
<td>11,029</td>
<td>5,900</td>
<td>2,900</td>
<td>1,580</td>
<td>780</td>
<td>390</td>
</tr>
</tbody>
</table>

(a) (5 points) Does the code demonstrate good strong scalability? Why or why not?

(b) (5 points) Does the code demonstrate good weak scalability? Why or why not?

(c) (5 points) What do the experimental results predict will happen for $n = 512, p = 64$?
OPTIONAL: Extra-Credit Question

(7 points) Suppose that MPI_COMM_WORLD consists of the two processors 0 and 1, and suppose the following code is executed in a distributed memory environment:

```c
int x, y, z;
switch(my_rank) {
    case 0: x=3; y=5;
        MPI_Send(&x, 1, MPI_INT, 1, 10, MPI_COMM_WORLD);
        MPI_Recv(&y, 1, MPI_INT, 1, 20, MPI_COMM_WORLD, &status);
        break;
    case 1: x=6; y=2;
        MPI_Send(&x, 1, MPI_INT, 0, 20, MPI_COMM_WORLD);
        MPI_Recv(&y, 1, MPI_INT, 0, 10, MPI_COMM_WORLD, &status);
        break;
}
```

What are the values of x and y on each processor after the code has executed? Explain your answer.