ECE 563 Spring 2012 First Exam

version 1

This is a take-home test. You must work, if found cheating you will be failed in the course and you will be turned in to the Dean of Students. To make it easy not to cheat, this exam is open book, open notes, and you may use passive online resources to solve it. By passive I mean you may look at existing online resources, you may not post queries or requests to information to any online resource.

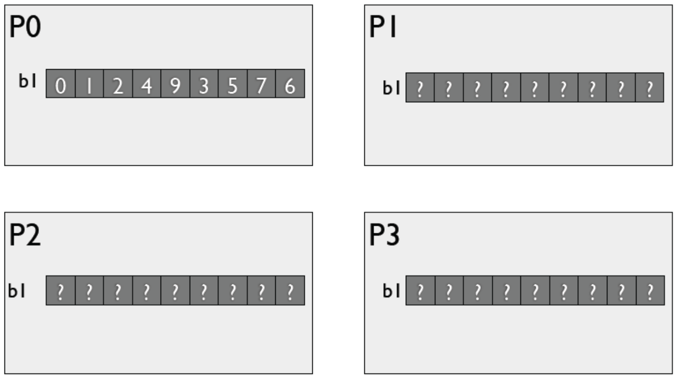
Please print and sign your name below. By doing so you signify that you have not received or given any prohibited help on this exam.

Name:

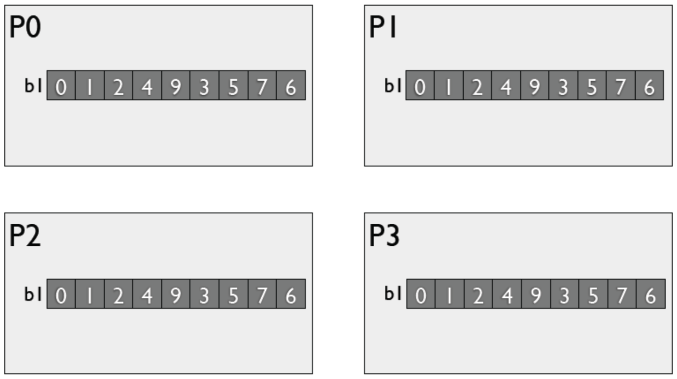
For all problems, unless stated otherwise you can assume that:

1. There is a variable called *pid* that contains the process id or rank in the communicator `MPI_COMM_WORLD`.
2. All initialization of the MPI runtime has been done.
3. Data being communicated is of type `MPI_INT`.
4. Using two or more communication operations where one will suffice will cost you at least part of the credit.
5. Each `send`, `receive` or step of a collective communication operation takes 100 time steps. Thus a reduce over 16 processors takes 400 time steps for communication ($100 \log_2 16$).
6. Each arithmetic operation takes 1 time step.
7. Problems that require computation on your part should show your work.
8. Note that there are slightly more than 100 points on the test.

Q1: *6 pts*
One or more MPI communication operations takes values of data owned by each process from:

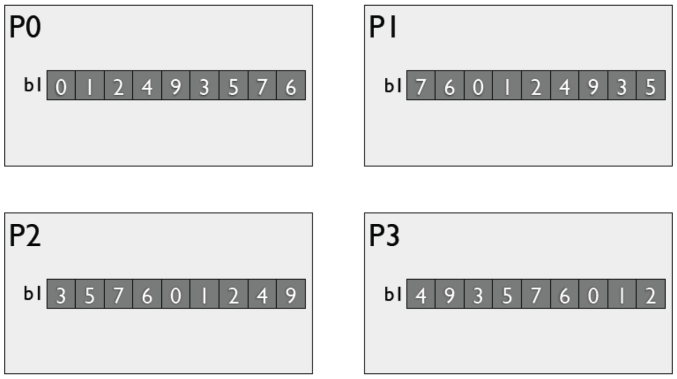


to

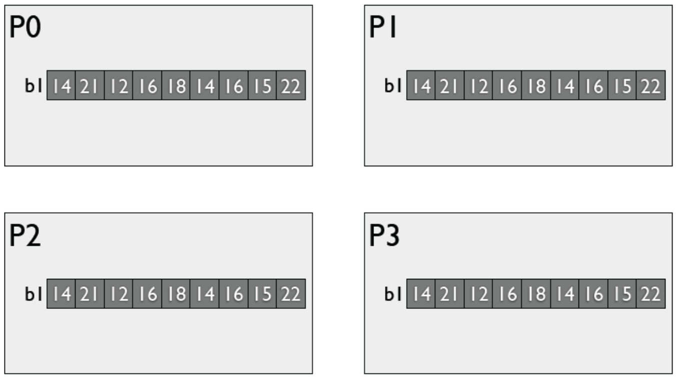


Show the MPI communication operation(s) and any conditional (`if`) statements needed to case the reordering of data gives the resulting data.

Q2: *6 pts*
One or more MPI communication operations takes values of data owned by each process from:

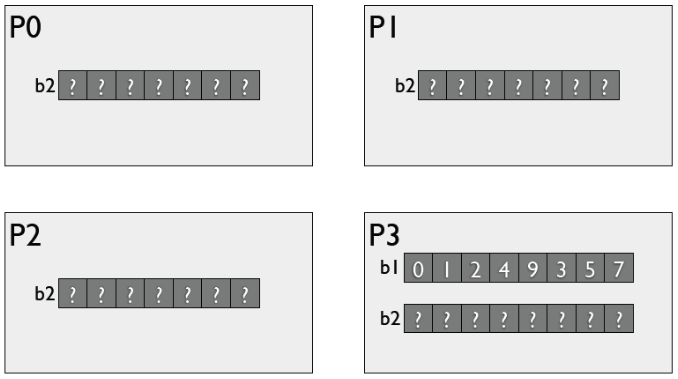


to

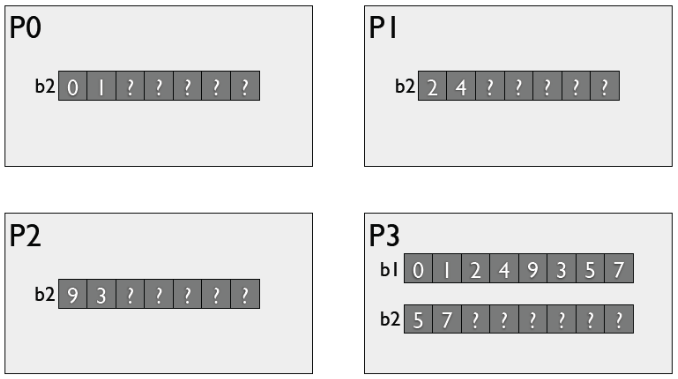

P0
b1 14 21 12 16 18 14 16 15 22
P1
b1 14 21 12 16 18 14 16 15 22
P2
b1 14 21 12 16 18 14 16 15 22
P3
b1 14 21 12 16 18 14 16 15 22

Show the MPI communication operation(s) and any conditional (`if`) statements needed to case the reordering of data gives the resulting data.

Q3: *6 pts*
One or more MPI communication operations takes values of data owned by each process from:

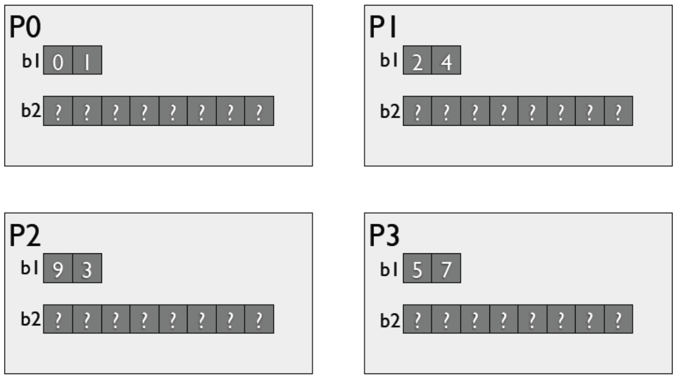


to

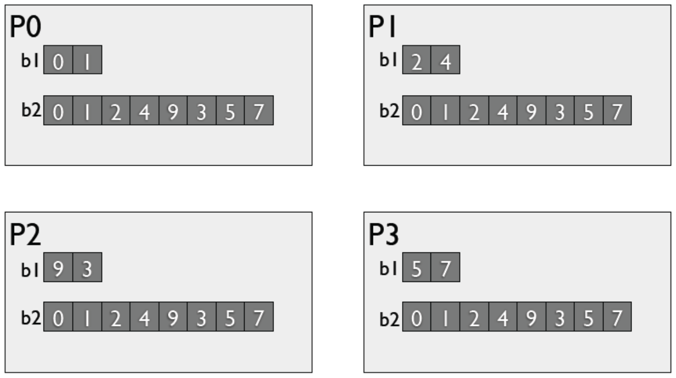


Show the MPI communication operation(s) and any conditional (`if`) statements needed to case the reordering of data gives the resulting data.

Q4: *6 pts*
One or more MPI communication operations takes values of data owned by each process from:

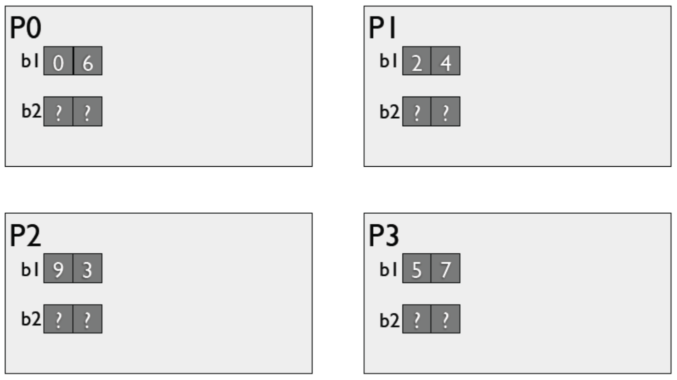


to

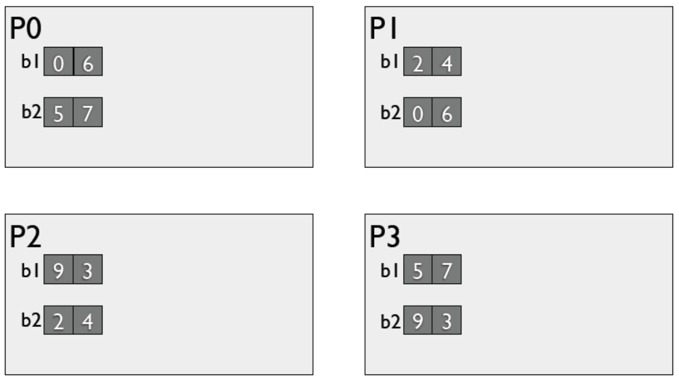


Show the MPI communication operation(s) and any conditional (`if`) statements needed to case the reordering of data gives the resulting data.

Q5: *6 pts*
One or more MPI communication operations takes values of data owned by each process from:

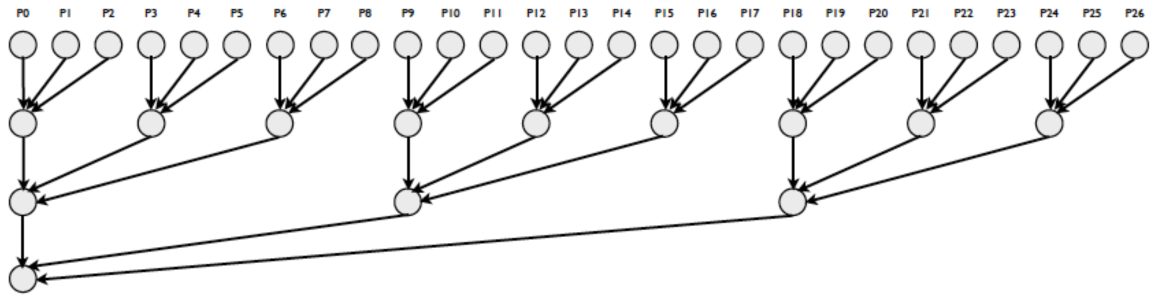


to



Show the MPI communication operation(s) and any conditional (`if`) statements needed to case the reordering of data gives the resulting data.

Q6: *6 pts*
Carp Computing has decided to take the lead in the large numerical server market, and has unleashed its Triad computer. The Triad computer is unique in that it supports reduce and broadcast degree-three trees instead of the traditional degree-two trees supported by its competitors. Triad also supports, in addition to a floating multiply-add operation (which performs the operation $c + a * b$ is one unit of time) a floating add-add operation, which performs the the operation $c + a + b$ in one unit of time. A reduction on this network is shown below.



Compare the time it takes to perform a sum reduction operation over 729 processors on the Triad computer to the time it takes to perform the same operation on a regular computer that performs a reduce like we have seen in class. Assume the `count` argument to the `MPI_Reduce` operation is 1.

Q7:

To show the effectiveness of their machine, Carp Computing forms the following two tables:

num procs	27	81	243
speedup	26.7	77	203.9
e			

num procs	27	81	243
speedup	26.8	78.5	216.3
e			

Q7a: *4pts* Fill in the e row with the Karp-Flatt metric value.
Q7b: *4pts* Each table represents experimentally measured values from a different machine for identical programs and compilers. Which machine will give better scaling for the problem?

Q7c: *3pts* Which machine is the Triad machine (discussed in question **Q6**) and which is a normal machine?

Q8: *6 pts*

You have an intern position at Oak Ridge National Lab running on their $224,162$ processor Jaguar machine. You write a program which is 99.999% parallel, i.e. the serial fraction f of the program is .00001. What is the speedup of your program, computed using Amdahl's Law? What is the efficiency?

Q9: *6 pts*

You are not happy with your efficiency and decide to make life easier and let the problem size grow with the number of processors, providing more work to be done in parallel and (you hope) increasing the speedup and efficiency. You vaguely remember hearing your 563 professor talking about Gustafson and Barsis and Harry Potter during an interesting dream. Curious about how they might be connected, you look at your old lecture notes, and find the answer, shortly before falling asleep yet again. Waking up refreshed, you decide to let s, the sequential part of a parallel computation, be 0.00001. What is the speedup on the Jaguar machine at $244,162$ processors?

Q10: *6 pts*

Make the `i` loop parallel.

```
for (i = 0; i < 100 * n; i ++) {
    for (j = 0; j < n; j ++) {
        a[i][j] = b[i][j] + c[i][j];
    }
}
```

Q11: *6 pts*

Make the `j` loop parallel.

```
for (i = 0; i < 100 * n; i ++) {
    for (j = 0; j < n; j ++) {
        a[i][j] = b[i][j] + c[i][j];
    }
}
```

Q12: *6 pts*

Compilers can perform a loop transformation called *loop interchange*. Applying loop interchange to the i and j loops in the following loop nest:

```
for (i = 0; i < 100 * n; i ++) {
    for (j = 0; j < n; j ++) {
        a[i][j] = b[i][j] + c[i][j];
    }
}
```

would yield a loop nest that looks like:

```
for (j = 0; j < n; j ++) {
    for (i = 0; i < 100 * n; i ++) {
        a[i][j] = b[i][j] + c[i][j];
    }
}
```

Give reasons in terms of available parallelism and overhead as to why you might want to parallelize the outer i loop in the first loop nest, and the outer j loop in the second loop nest. You answer should short.

Q13:

Q13a: *4 pts* Make the following loop parallel using OpenMP pragmas/directives.

```
for (i = 0; i < 100 * n; i ++) {
    s+= a[i]
}
```

Q13b: *4 pts* If the loop is changed to look like:

```
for (i = 0; i < 100 * n; i ++) {
    s+= a[i]
    b[i] = a;
}
```

Can the technique used in the first loop to perform the parallelization still be used?

Q14: *6 pts*

Make the `i` loop parallel using OpenMP. You cannot make any assumptions as to whether `foo` is associative or commutative.

```
for (i = 0; i < 100 * n; i ++) {
    for (j = 0; j < n; j ++) {
        a[i]+ = b[i][j];
    }
    c = foo(c);
}
```

Q15: *6 pts*

```
int a[50]; int b[50]; int c[50];
for (j = 0; j < 100; j ++) {
   a[i] = 1;
}
#pragma omp parallel for
for (i = 0; i < 100; i ++) {
   if (i < 50) {
         b[i] = a[i];
         c[i] = a[i];
   }else a[i] = 0;
}
```

Given the loop above, circle the answer that is most right about the values of `a`, `b` and `c` after the program executes:

1. Element `b[i]` is equal to element `c[i]` for all $0 \leq i \leq 49$.
2. Element `b[i]` is never equal to element `c[i]` for all $0 \leq i \leq 49$.
3. Element `b[i]` may, or may not be equal to element `c[i]` for all $0 \leq i \leq 49$.

Q16: *6 pts* If the loop of Q15 is changed to:

```
int a[50]; int b[50]; int c[50];
for (j = 0; j < 100; j ++) {
   a[i] = 1;
}
#pragma omp parallel for
for (i = 0; i < 100; i ++) {
   if (i < 50) {
      #pragma omp critical
      {
         b[i] = a[i];
         c[i] = a[i];
      }
   }else {
      #pragma omp critical
      {
         a[i] = 0;
      }
   }
}
```

Given the loop above, circle the answer that is most right about the values of `a`, `b` and `c` after the program executes:

1. Element `b[i]` is equal to element `c[i]` for all $0 \leq i \leq 49$.
2. Element `b[i]` is never equal to element `c[i]` for all $0 \leq i \leq 49$.
3. Element `b[i]` may, or may not be equal to element `c[i]` for all $0 \leq i \leq 49$.