CSCI-UA.0480-003

Parallel Computing

Midterm Exam Spring 2014

(75 minutes)

Last name: First name:

Notes:

- **If you perceive any ambiguity in any of the questions, state your assumptions clearly**
- **Questions vary in difficulty; it is strongly recommended that you do not spend too much time on any one question.**
- **This exam is open book/notes but no electronic devices.**

---

1. [5 points] Circle the correct answer among the choices given. If you circle more than one answer, you will lose the grade of the corresponding question.

1. Which of the following provides more potential performance gain:
(a.) Multiple threads within a process b. Multiple processes within a thread
c. Multiple threads and no processes d. Multiple processes and no threads

2. Pipelining, superscalar capability, and branch prediction are meant to exploit:
(a.) Instruction level parallelism b. Thread level parallelism
c. Data level parallelism d. process level parallelism

3. Which of the following does *not* affect the cost of sending a message from a process to another process?
(a.) Memory size b. Message size
c. Interconnect bandwidth d. interconnect latency

4. Which of the following is *not* a reason for the non-determinism of parallel programs?
a. race condition b. coherence
(c.) non-uniform memory access d. parallel execution of processes

5. MPI is very convenient when (choose the most accurate)
a. There are a lot of processes
b. There are a lot of processes exchanging a lot of data
c. There are a lot of dependent processes exchanging very few data
(d.) There are a lot of independent processes exchanging very few data

2. [3 points] **We have seen many examples in MPI where processes are doing the same operation(s) but on different data depending on the rank of each process. Do you think it is possible to make each process do a different task (for example one process calculates the min of a group of numbers, while another calculates the max, a third calculates the average, etc)? If so, how? If not, why not?**

Yes, we can. Simply use a series of if-else or case-switch based on rank number and call different functions.

3. Suppose a program is divided into 7 tasks .One task at the beginning must be executed by itself (i.e. with no other tasks in parallel with it) and takes 3 ms. Also there is a task at the end of the program that must be executed by itself and takes 4 ms. Between these two tasks, there are 5 tasks that can be executed simultaneously and each of these tasks takes 16ms.

a. [2 points] What is maximum speed-up *according to Amdahl's law* (make any assumptions you want but write them clearly)?

The whole program, if run sequentially takes: 3+(5*16) + 4 = 87ms
Out of this: 3+4 = 7ms is the sequential part. So, based on Amdahl's law the fraction of sequential part is 7/87 = 0.08 and the speedup= (1/(0.08 + (0.0.92/p))
If p (the number of processors) becomes infinity (to get maximum speedup from the law), then speedup = 1/0.08 = 12.5
(Note: Those who calculated f to be 2 tasks out of 7 so 2/7, we also grade it as correct)

b. [2 points] What is the largest number of processes we can use after which no speedup will be seen? Justify

5 because we have 5 independent tasks that can be executed in parallel

c. [1 point] If we use that largest number of processes you calculated in the above question, what is the speedup we get? What is the efficiency? (do not use Amdahl's law in your calculations)

Speedup = Time sequential/ Time parallel = 87/(3+16+4) = 3.78
Efficiency = speedup/num_processes = 3.78 /5= 0.75

d. [2 point] Is there a difference between the speedup you calculated in c and a above? Justify.

yes, there is a difference. In (a) we calculated a theoretical upper bound and assumed infinite number of processes that will render the total parallel part to 0!

4. Take a look at the program below, and assume that for all questions we run the program with: *mpiexec -n 6 ./prog*

a. [2 points] What is the output of the program?
result = 1

b. [3 points] As you can see, the MPI_Reduce() command is used in exactly the same way (i.e. with the same arguments) in both the *if* part and the *else* part. What will be the output if we remove the *if-else* construct and leave one instance of the MPI_Reduce and the printf?

result = 1 <-- from process 0
result = 0
result = 0
result = 0
result = 0
result = 0 (order may differ)

Extra exercise (not in the exam): Try to repeat the above problem if num[0] = 9
What do you realize? Can you fix that?

```
#include <stdio.h>
#include <mpi.h>

int main(void) {
  int       comm_sz;              /* Number of processes    */
  int       my_rank;              /* My process rank        */
  int       num[6] = {1, 2, 4, 3, 7, 1};
  int       result =0;

  MPI_Init(NULL, NULL);
  MPI_Comm_size(MPI_COMM_WORLD, &comm_sz);
  MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);

          if (my_rank != 0) {
            MPI_Reduce(num, &result, 6, MPI_INT, MPI_MIN, 0,       MPI_COMM_WORLD);
                  }
          else {
            MPI_Reduce(num, &result, 6, MPI_INT, MPI_MIN, 0, MPI_COMM_WORLD);
           printf("result = %d\n", result);
          }

          MPI_Finalize();
       return 0;
}
```