- Middle East Technical University Department of Computer Engineering

CENG 478

Introduction to Parallel Computing

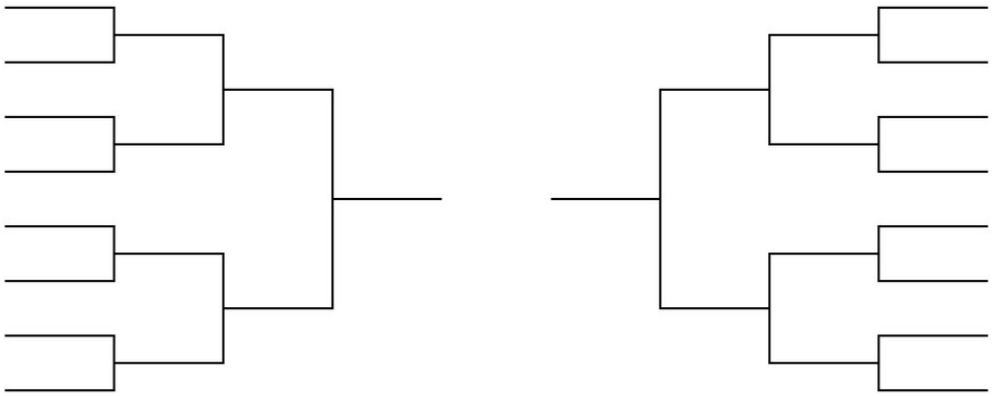
Spring 2019-2020

Assignment 2

Due date: 09.04.2020, 23:59

1. Tournament Simulation (40 Pts)

   a. You will implement a parallel algorithm that simulates a tournament. This tournament can be any game. Only restriction is that there can not be any draw, one of the teams should win each match-up.
   b. There are 2^n number of teams at the start of the tournament. For the first leg of the tournament every $(2i)^{th}$ team will play against the $(2i+1)^{th}$ team. The winner of each match-up qualifies for the next leg and the loser will be eliminated. There should be 2^{n-1} teams left for the second leg and there will be total n legs of the tournament. The number n will be given as an input on argv[1].
   c. Denote the teams by numbers (i.e. $0, 1, 2, 3, \dots, 2^n - 2, 2^n - 1$)
   d. Matching rule is the same for every leg. (ie. The winner of 0-1 matches with the winner of 2-3)



   e. You will randomly decide the winner of each matchup with the following function:

```
int simulate_game() {

        usleep(10);             // Simulation takes time
        return (rand() %2)      // HOME_WIN or AWAY_WIN
}
```

   f. For a more realistic distribution, use different seeds for randomization on each process. You can include <time.h> and use “srand(time(NULL) + world_rank);” on your code.

   g. Hint: You can use MPI_Scatter and MPI_Gather functions. For the function usleep() , you have to include <unistd.h> .

2. Tests and Report (60 Pts)

a. Comment on the complexity of this simulation. How did you distribute the workload on your parallel application? What is the advantage of your design over a serial approach?

b. State your design choice. (Which MPI functions did you use and why?)

c. Your programs should print the following two lines:

The_Winner
Total_Time

d. To measure the time, you can use MPI_Wtime. You can generate the teams on process 0 and distribute values on other processes. Just measure starting time after process 0 sends the values.

e. Run your implementation using 1, 4, 8, 16 number of MPI processes with $n = 16$. Plot a "Time vs. Number of processors" graph. Plot a "Speed Improvement vs. Number of processors" graph. Comment on how the time and efficiency changes. What is the cause of this increase or decrease? Speculate on what can be done to improve the performance further.

3. Notes

a. You will submit a tar file consisting of your code file(s), your makefile, a **pdf** of your report and the outputs of your executions (i.e. 1_out.txt, 4_out.txt, 8_out.txt, 16_out.txt) via ODTÜClass. Note that the **comments and comparisons** on your report will **have a large effect on your grade**.

b. Note that **zombie processes** can pile up very quickly as you are trying to learn a new way of programming. Try to **control frequently** if there are any, with *squeue* command and kill them with *scancel* command to ensure that Slurm serves all the users **properly**.

c. Try to finish as early as possible. Since all of you work on the **same** HPC, the waiting times for the **queue** can be **huge** which will prevent you from testing your code **efficiently**.

d. Implementing MPI macros **does not mean** your code works in **parallel**. You have to design your algorithm so that it really works in parallel. For those who submits code that does not work in parallel will get **no credits** from this assignment.

e. You can still submit your work if the **deadline** is passed, however with an increasing **penalty** of **5*days*days**. (i.e. first day -5 points, second day -5*2*2=-20 points and so on). Note that even a minute late means that it is the other day.

f. We have zero tolerance policy for cheating. People involved in cheating will be punished according to the university regulations and will get zero.