CENG 478 – Introduction to Parallel Computing
SPRING 2014-2015 Homework-4
Due to: 24.05.2015 23:55

Prime Number Counter

In this assignment, you will implement a code to find out how many prime numbers from 1 to n, including n. İ.e. when n is 11, the prime numbers are 2,3,5,7 and 11. Thus, the result will be 5.

1. To test whether a number is prime or not, you may simply use the following code:

```
bool test_prime(int x) {

    int y;
    for (y=2; j <=sqrt(x); y++)
            if ( x % y == 0) return false;
    return true;
}
```

Note that the only even prime number is 2.

2. Submission:

   - The number n will be determined by the input. For example the script file will include the line "mpirun -np 2 hw4 11" to execute the code with 2 processors when number n is 11.
   - Your code should work in parallel.
   - Your Makefile should compile the code, and create the binary file hw4.
   - For your report, choose number n large enough to see the effects of parallel computing. State your choice on your report.
   - Calculate the time consumed for 1,2,4,8,16,32 processes. Plot a *Time* vs. *Number of Processors* graph. Plot a *Speed Improvement* vs. *Number of Processors* graph. (State how many nodes and processor per node you used, for testing purposes. İ.e. for 32 processor 4 node and 8 ppn, or 8 core and 4 ppn)
   - Briefly explain your implementation. (Hint: your design choice to avoid **load balancing** issues between different processors).
   - Give two examples for problems which are **difficult** to solve in parallel because of the **load balancing** issues.

3. Important Notes:

- Please note that zombie processes may pile up. So, try to control frequently if there are any, with *qstat* command, and kill them with *qdel* command to ensure that HPC will serve all the users properly.

- Since HPC breaks down so often, always have a **copy** of your work.

- Do not leave your work to the deadline. Since all of you work on same HPC, on the last day it may work very slow and you may **not** be able to test your code **efficiently**.

- Note that implementing MPI macros does not mean your code works in **parallel**. You have to design your code so that it works really in parallel. For those who submit a code that does not work in parallel will get **no credits** from this homework.

- Write a report based on these tasks. Note that **most** of the credit you earn for this homework will be based on your **reports**. So, give importance to the report. If you submit your homework without a report it is possible that you may **not** get any credits.

- Cheating policy of our department applies also for this course. Please submit your own work, unless you want to get zero points from homework and being subjected to a disciplinary action.

- For this homework, late submission is **not** allowed