Distributed Computing

Assignment 1

March 16, 2018

Problem Description

Parallel Prime Number Sieve

We want to find the number of primes less than or equal to some positive integer n. A prime has exactly two factors: itself and 1. The Sieve of Eratosthenes begins with a list of natural numbers 2, 3, 4, ..., n, and removes composite number from the list by striking multiples of 2, 3, 5, and successive primes. The sieve terminates after multiples of the largest prime less than or equal to $\sqrt{n}$ have been struck.

A sequential implementation of the Sieve of Eratosthenes manages three key data structures: a boolean array *Prime* whose elements correspond to the natural numbers being sieved, an integer *num* corresponding to latest prime number found, and an integer *loc* used as a loop index incremented as multiples of the current prime are marked as composite numbers. The program is shown below:

```
Program Sieve;
const n=100;
var Prime: array [1..n] of boolean;
    i, num, loc: integer;
begin
   for i:=1 to n do
       Prime[i]:=true;
   for num:=2 to $\lfloor\sqrt(n)\rfloor$ do
       if Prime[num] then
          begin
             loc:=num+num
             while loc$\leq$ n do
                 begin
          Prime[loc]:=false;
                    loc:=loc+num;
                 end;
          end;
end.
```

The program has a `for` loop that scans the array from 2 up to the square root of n. Each element that still has value *true* when the scan reaches it much be a prime. The first such prime identified is the number 2. For each such prime identified (in *num*) , an inner `while` loop will change all multiples of that prime to value `false` in the array. The variable *loc* is used to step through the array in increments of size *num*. This process will eliminate all non-prime numbers in the array. As a result, all remaining elements in the array with value *true* at the end of the program are prime numbers.

In this assignment you are required to write a parallel version of this Sieve of Eratosthenes program. Your program is parallelized by partitioning the array *Prime* into equal-size portions, and creating one parallel process to work on each portion. Make the portions large enough so that all the elements up to the square root of n are contained in the first portion. The process assigned to the first portion is almost identical to the sequential program given above. the main difference is that the `while` loop stops when it reaches the end of the first portion of the array. The other portions will be handled by their own processes.

All processes begin by initializing their own portion to *true*. This can be done in parallel. Then the first process begins to search for the first *true* value, which of course is 2. This number 2 is then broadcast to all the other processes. As each process receives this number, it begins to step through its own portion in jumps of length 2, thereby changing all the even numbers to $false$. Then the first process loops around to search for the next *true* value, and again broadcast this to the other processes, where it is used to step and change *true* to $false$. In this parallel version, there is a separate process to step through each portion of the array in parallel.

Broadcasting each "step" number to all the processes can be made more efficiently by passing it through a process pipeline. To reduce the execution time, each process should determine an appropriate starting point in its portion for stepping after it receives a given "stepping" number *num*.

You must write a correct program using HPF available at IBM SP2. For input array of size n, your program employs $\lceil \sqrt{n}/2 \rceil$ (virtual) processors connected in a linear array, where each physical processor corresponds to a set of virtual processors. You must run your program on arrays of size $2^i \times 10^3$ for $i = 0, 1, 2, \ldots, 9$, and estimate its parallel running time for each of these cases, respectively. Your running time estimation may be done by setting a timer within each process to record the total running time of that process, and then taking the maximum value over all timers when the program terminates.

Submission of Assignment

You may do this assignment in a group of two students. Each group must submit to the TA both the *source code* of your program, and the outputs (primes found) on the above 10 test cases, and your timing results for these cases. You must show clearly at the beginning of your submission the names of ALL members in your group.

This assignment is due at 7pm on Friday 30th March 2018. Electronic submissions of the assignment are to be made using the command.

Important Note

This assignment should be completed in groups of size at most two. No inter-group collaboration (beyond general discussion) is permitted. Be warned that all submitted programs will be automatically checked for similarities.