Lab 1

Jackson Bellinger, Sean McGee

September 4 2019

1 Readme

420-lab1 Parallelized naive primality test

(a) The theoretical time complexity should be around $\sqrt{number}/number\ of\ nodes + startupcost$. We found that the runtime actually increased with more nodes. This is likely because the startup cost depends on the number of nodes. Since the problem of finding the primality of a single number is not especially computationally difficult at the scales we are testing. I actually tested a few prime numbers around 10 billion and it found divisors who's product wasn't actually that number (I think it's an overflow error or something).

(b) No, adding more nodes didn't divide the time. In most cases it actually increased our runtime.

(c) Any divosor that could be found which is greater than $\sqrt{N}$ would have to have a corresponding factor that is less than $\sqrt{N}$, so it would be found first by the brute force algorithm by at least one of the nodes.

(d) There are a few ways we thought of that we could improve our code. Firstly, Solving the overflow errors would probably be the most important thing. Also I am excited to try working with reduction and gather operations! Although not specific to this project, I would like to improve our makefile to be general for any similar project (WIP). Lastly, I think it would be interesting to try the AKS primality test.

2 Timing data

number num nodes	200000	11	19845	349	124	211	224737	2750159
1	618.4	613	620.6	606	615	619.2	617.8	612.6
2	628.6	626.4	626.4	622.2	625.2	622	617.8	619.2
3	635.2	629.8	633.2	630.4	630.6	623.4	634	629.6
4	637.8	641.4	641.2	638.8	639.4	635	633.8	627.2
5	637.8	641.2	639.8	638.6	639.8	637.4	633.4	634.4
6	634.8	643	647.6	647.8	639	644	640.4	641.2
7	647.8	641.6	646.4	641.8	646.6	639	627.4	649.2
8	647.4	648.8	648	647.4	648.2	644.4	651.2	647.4

```c
#include <stdio.h>
#include <stdlib.h>
#include <mpi.h>
#include <math.h>

int main(int argc, char** argv)
{
    MPI_Init(&argc, &argv);

    int num_nodes;
    MPI_Comm world = MPI_COMM_WORLD;
    MPI_Comm_size(world, &num_nodes);
    int node_rank;
    MPI_Comm_rank(world, &node_rank);

    printf("I am node %d, checking multiples of %d + %d\n", node_rank,
        num_nodes, node_rank + 2);

    unsigned long n = atoi(argv[1]);
    int upperbound = sqrt(n);
    int found_divisor = 0;

    int i;
    for(i = node_rank + 2; i <= upperbound; i += num_nodes)
    {
        if( n % i == 0)
        {
            found_divisor = 1;
            printf("Node #%d : %d and %lu are factors of %lu\n",
                node_rank, i, n/i, n);
            //mpi send
            break;
        }
    }

    MPI_Finalize();
    //mpi recieve
    if(found_divisor == 0)
    {
        printf("Node #%d : %lu is a prime\n", node_rank, n);
    }

    return 0;
}
```