Ceng478 HW4 Report

Oğuzhan Taştan
Department of Computer Engineering
METU
Ankara, Turkey

***Abstract*—In this paper, the effect of load balancing on the performance of a parallel code are investigated. To see the effect, we compare two different versions of the same algorithm. In the first one, the computation cost is distributed to the processors equally, while in the second one, the computation cost is distributed unequally.**

***Keywords*—Parallel Computing, MPI, load balancing**

I. Introduction

The task is to find the number of prime numbers which is less than or equal to the given number. To decide whether a number is prime number or not, the following algorithm has been used:

```
for (i=3;i<=sqrt(n);i+=2)
    if (n%i==0)
        return 0;
return 1;
```

The algorithm starts from 3 to the iteration instead of 2 because only the odd numbers are tested by the algorithms since there is no even prime number expect from 2 which is counted in the master processor without test.

The reason of increasing the iterator at each iteration by two is to avoid unnecessary controls for even dividers since only the odd numbers are tested.

Since there is no need to check an even number is prime number or not, only the odd numbers distributed to the processors.

The following table shows the configurations of the number of nodes and processors which have been used in the experiments.

Number of Node	Processors from each node	Total number of Processors
1	2	2
2	2	4
2	4	8
4	4	16
4	8	32

Table 1: the configurations of the number of nodes and processors

The algorithm has been run with load balanced and load unbalanced. Then, comparisons between them have been made. The following sections of the paper explain how the load balancing is achieved, how the load unbalancing is achieved and the differences between them, respectively.

II. Load Balanced Algorithm

Since each number, say n, which is tested, has a computation cost of $\lfloor\sqrt{n}\rfloor$, sending equal number of numbers to each processor result in unbalanced load. To achieve load balancing, the following procedure has been followed:

1- Calculate the total cost of the numbers using the following formula:

$(8* \lfloor\sqrt{n}\rfloor+1)* \lfloor\sqrt{n}\rfloor*(\lfloor\sqrt{n}\rfloor+1) / 3 - n$

This formula is derived by summing the computation cost of 1 to n.

2- Divide the sum calculated at step 2 by the number of processors to find average computation cost.

3- The master processor sends the numbers to one processor until the cost of number which is sent just exceeds the average computation cost. Then, it sends to the next processor until again the cost just exceeds the average computation cost. This process continues until next to the last processor. For the last processor, the remaining numbers are sent.

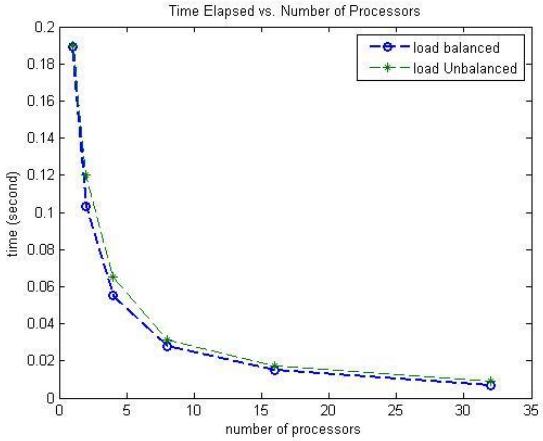
With this procedure, each processor has load which is near to the average; therefore, the load balancing is achieved.

III. Load Unbalanced Algorithm

For load unbalancing, master processor sends equal number of numbers to each processor. The processors which have big numbers have high computation cost. As mentioned in the previous section, this result in unbalanced load.

IV. Comparison of Performance of Algorithms

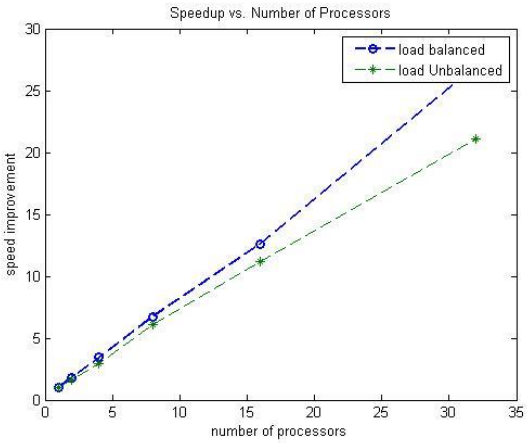
When the algorithms run with n=1.000.000, the following graph has been obtained.



Since there is no distribution with only one processor, there is no load balancing issue, therefore; the elapsed times of the algorithms are equal. However, with higher number of processors, the load balanced algorithm has more elapsed time than the load unbalanced algorithm has.

The difference between the elapsed times of the algorithms is caused by the fact that in load unbalanced algorithm, to find the result, master processor has to wait for the other processors and at least one of the other processors has higher workload; therefore, the elapsed time depends on the elapsed time of the maximum loaded processor. On the other hand, in load balanced algorithm, since all processors have almost equal workload, there is no unnecessary waiting, therefore; the elapsed time is lower.

The speedup graph of the algorithms is as follows:



The load unbalanced algorithm has less speedup than the load balanced algorithm has because they have same execution time with one processor (i.e., sequential) and the execution time of the load unbalanced algorithm is larger than that of the load balanced algorithm by the following speedup formula:

$$\text{Speedup} = \frac{T_{sequential}}{T_{parallel}}$$

In other words, since $T_{sequential}$ is the same for both algorithms and $T_{parallel}$ is larger for the load unbalanced algorithm, the speedup of the load unbalanced algorithm is lower than that of the load balanced algorithm.