Determining the N-th Prime Number Using a Network of Workstations

Mark Allman
School of Electrical Engineering and Computer Science
Ohio University
mallman@cs.ohiou.edu

June 7, 1996

Abstract

Workstations connected by local area networks are now common. Using these resources to speed up processor intensive tasks is an important area of study. This paper demonstrates an almost linear speedup as workstations are added to the task of determining the N-th prime integer.

1 Introduction

Powerful workstations connected to local area networks (LANs) are now common. Using these resources to build a parallel computing environment can be less expensive than buying additional hardware made for parallel processing. In addition, using existing workstations in parallel can increase the useful life of the equipment.

Prime numbers have a number of properties which play a crucial role in number theory [CLR90] and therefore, generating prime numbers is a critical task. An integer x is said to be *prime* if $x > 1$ and the only factors of x are 1 and x. Any integer which is not prime is *composite*, with the exception of 1, which is said to be *unit*.

In order to determine whether or not a given integer x is prime takes $O(\frac{1}{2}\lceil\sqrt{x}\rceil)$ steps in the worst case. If x is even, one of it's factors is 2, making it composite. The exception to this is 2, which is prime since it's only factors are 1 and 2. Since all even numbers except 2 are composite, only half of all integers need to be checked. Each composite number x must have at least one factor y such that $y \leq \lceil\sqrt{x}\rceil$. Therefore, an odd integer x must be divided by each odd integer y such that $y \leq \lceil\sqrt{x}\rceil$ until a factor is found. If no factor is found x is prime.

This study involves determining the N-th prime number using multiple workstations connected by a LAN. Determining the N-th prime number involves checking consecutive odd integers until N prime numbers have been found.

2 Sequential Implementation

2.1 Algorithm

A program called *sprime* was written to determine the N-th prime number sequentially using a single workstation. *Sprime* treats 2 as a special case and starts with 3, checking every other integer. Once it has been determined that a given number is prime, a counter is incremented. When this counter reaches N, the current number is the N-th prime integer.

Each odd integer x is checked for factors using the method outlined in section 1. That is, by starting at 3 and dividing by odd integers until either a factor is found or each odd integer y such that $y \leq \lceil\sqrt{x}\rceil$ has been checked. If a factor is not found, x is prime.

2.2 Results

Using an otherwise idle SparcStation 4 running Solaris 2.5, *sprime* is able to generate the

500,000th prime number in 8 minutes and 28 seconds. This will be used as the benchmark in the following tests.

3 Parallel Implementation

3.1 Algorithm

The multi-processor version of *sprime* was written in two halves. A server, called *pprimed*, runs on each machine which has been designated a *processor*. A client, called *pprime*, runs on a machine which has been designated the *coordinator*. The coordinator is connected to the processors via a LAN. The Transmission Control Protocol (TCP) [Pos81] provides reliable communication between the coordinator and the processors. All data sent between the coordinator and the processors will fit in one TCP segment and therefore TCP's congestion control mechanisms [JK88] should not influence performance.

3.1.1 Coordinator

The coordinator schedules the activity of each processor. The coordinator sends messages to each processor giving a range of integers to check, as well as, the maximum number of primes the processor should detect. Each time the coordinator receives a response from a processor, it determines whether or not the N-th prime number has been found by summing the number of primes found in all ranges which have been checked. If the total number of primes detected is less than N, the coordinator immediately sends a request containing a new range of numbers to the processor. If the total number of primes detected is greater than N, the coordinator *backtracks*. That is, it sends a new request to the processor containing the same range of integers but a smaller maximum number of primes to detect.

Each processor will take a different amount of time to return an answer to the coordinator because of differing range sizes and complexities. Therefore, the coordinator uses asynchronous calls to communicate with each processor to ensure all processors remain independent.

The range of integers given to each processor by the coordinator is critical. If this range is too large, the processors will check too many integers and backtracking will be expensive. On the other hand, if the range is too small, the communication time will increase which will hurt performance. The coordinator solves this problem by using large ranges for the first requests and reducing the range for each additional request. The coordinator uses the number of primes which have not yet been detected as the range. A minimum range of 25,000 is used to ensure that the communication time never dominates the execution time. This minimum range ensures that backtracking will be needed. However, this backtracking will not be expensive, since the range will be relatively small.

3.1.2 Processors

Each processor accepts requests from the coordinator containing a range of integers to check and a maximum number of primes to detect. The processor uses the algorithm outlined in section 2.1 to either detect the maximum allowed number of primes or check the entire range, whichever happens first. The processor then sends the coordinator the number of primes found in the range and the last prime number found.

3.2 Results

The parallel processor version of this program was tested by calculating the 500,000th prime number using multiple SparcStation 4 workstations which were otherwise idle. These machines are identical to those used in the single processor experiment. Each workstation is connected to a network hub using 10BaseT. At the time of the tests, the network was uncongested. The round trip time (RTT) between the machines is 18 ms.

The parallel processor version of this program shows nearly a linear speedup over the single processor version. The speedup and processor utilization are shown in table 1. The speedup and utilization are less than ideal because of the communication time involved and the added complexity introduced in the parallel version. While

Processors	Time (sec)	Speedup	Utilization
1	508	–	–
2	257	1.98	0.99
3	173	2.94	0.98
4	130	3.91	0.98
5	105	4.84	0.97
6	91	5.58	0.93

Table 1: Speedup and Utilization

the absolute communication time remains approximately the same across all multiple processor tests, it constitutes a larger percentage of the total running time, as the number of processors is increased. Therefore, both speedup and utilization deviate from ideal more as the number of processors is increased.

4 Future Work

The algorithm used by the coordinator to schedule the processors should be studied further. A better scheduling algorithm is needed to yield better utilization as the number of processors grows. The well known Prime Number Theorem [CLR90] can approximate the number of primes in the range 1–x for any sufficiently large x. This could lead to a scheduling algorithm which yields results which are closer to optimal.

5 Conclusions

This study has shown that a series of workstations connected by a standard LAN can be used in parallel to increase the speed of a CPU intensive task, such as determining the N-th prime number. This is an important result, as it shows that common resources can be combined to produce dramatic performance gains. Such parallel processing systems can also extend the useful life of workstations.

References

[CLR90] Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest. *Introduction to Algorithms*. McGraw Hill, 1990.

[JK88] Van Jacobson and Michael J. Karels. Congestion Avoidance and Control. In *ACM SIGCOMM*, 1988.

[Pos81] Jon Postel. Transmission Control Protocol, September 1981. RFC 793.