SEIVE OF ERATOSTHENES

Team Members

Murarishetty Santhosh Charan (2019287)

Repala Nagaraju (2019127)

Eraganaboina Venkata Srujan (2019250)

Veerisetty Subramanyam (2019172)

Instructor

Dr. Manish Kumar Bajpai

Introduction

The aim of this project is to parallelize sieve of Eratosthenes algorithm. The Sieve of Eratosthenes is an ancient mathematical algorithm of finding prime numbers between two sets of numbers.

Sieve of Eratosthenes models work by sieving or eliminating given numbers that do not meet a certain criterion. For this case, the pattern eliminates multiples of the known prime numbers. A prime number is a positive integer or a whole number greater than 1, which is only divisible by 1 and itself. The Prime number algorithm is a program used to find prime numbers by sieving or removing composite numbers. The algorithm makes work easier by eliminating complex looping divisions or multiplications.

Algorithm

The **Sieve of Eratosthenes** is a mathematical tool that's used to discover all possible prime numbers between any two numbers. Eratosthenes was a brilliant Greek thinker who, among many other important discoveries and inventions, was deeply interested in mathematics. His best known contribution to mathematics is his sieve used to easily find prime numbers. A mathematical sieve is any pattern or algorithm that functions by 'crossing off' any potential numbers that don't fit a certain criteria. In our case, the sieve of Eratosthenes works by crossing off numbers that are multiples of a number that we already know are prime numbers. While this all sounds quite complicated, in practice it's quite simple

Steps involved in algorithm

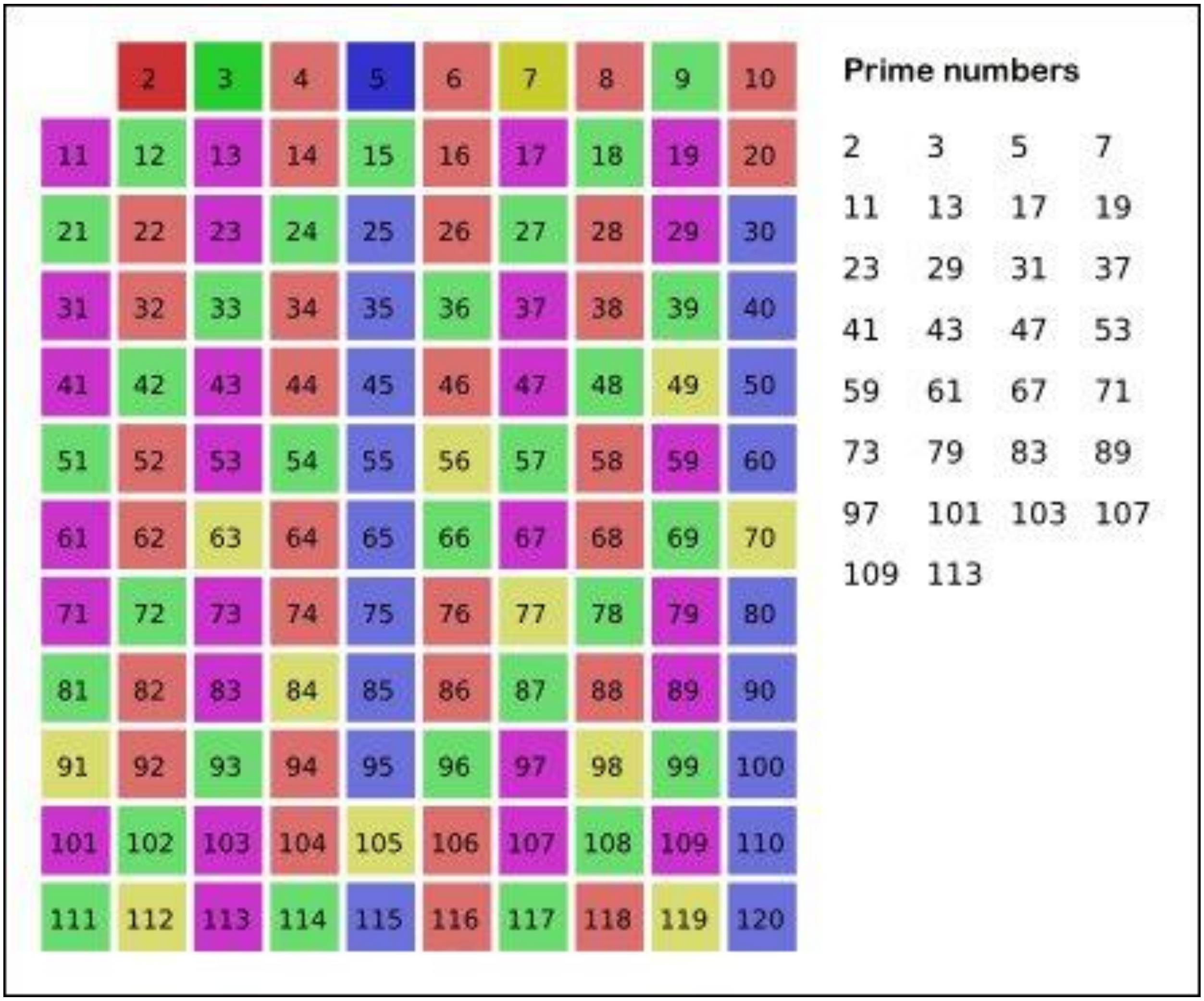
- List all consecutive numbers from 2 to η, i.e. (2, 3, 4, 5, ……, η).
- Assign the first prime number letter p.
- Beginning with p^2, perform an incremental of p and mark the integers equal or greater than p^2 in the algorithm. These integers will be $p(p + 1)$, $p(\mathrm{p} + 2)$, $p(p + 3)$, $p(p + 4)$ …
- The first unmarked number greater than p is identified from the list. If the number does not exist in the list, the procedure is halted. p is equated to the number and step 3 is repeated.
- The Sieve of Eratosthenes is stopped when the square of the number being tested exceeds the last number on the list.
- All numbers in the list left unmarked when the algorithm ends are referred to as prime numbers.

Pseudo Code

```
input: an integer n > 1
output: All prime numbers from 2 to n


 Eratosthenes(n) {
     a[1] := 0
     for i := 2 to n do {
         a[i] := 1
     }
     p := 2
     while p2  ≤  n do {
         j := p2
         while (j  ≤  n) do {
             a[j] := 0
             j := j+p
         }
         repeat p := p+1 until a[p] = 1
     }
     return(a)
 }
```

Basic Illustration



Sieve of Eratosthenes: algorithm steps for primes below 121 (including optimization of starting from prime's square).

Serial Implementation Code in C lang

```c
// C program to print all primes
// smaller than or equal to
// n using Sieve of Eratosthenes

#include<stdio.h>

int main()
{
    int n=0;
    scanf("%d",&n);
    int x=n+1;

    // Create a boolean primeay
    // "prime[0..n]" and initialize
    // all entries it as true.
    // A value in prime[i] will
    // finally be false if i is
    // Not a prime, else true.

    int prime[x];
    for(int p=0;p<=n;p++)
    {
        prime[p]=1;
    }

    // We are marking 0 and 1 as
    //false as they are composite
```

```
    prime[0]=0;
    prime[1]=0;



    for(int p=2;p*p<=n;p++)
    {
        // If prime[p] is not changed,
        // then it is a prime
        if(prime[p]==1)
        {
            int j=2;
            while(p*j<=n)
            {
                // Update all multiples
                // of p greater than or
                // equal to the square of it
                // numbers which are multiple
                // of p and are less than p^2
                // are already been marked.

                prime[p*j]=0;
                j+=1;
            }
        }
    }
    // Print all prime numbers
    for(int p=2;p<=n;p++)
    {
        //Print the number if it is a prime
        if(prime[p])
        {
            printf("%d is prime\n",p);
        }
    }
    return 0;
}
```

Complexity Analysis of the Algorithm

Time Complexity :

O(**N*log(log N)**)

Algorithm's running time is O(N*log(log(N)). The algorithm will perform n/p operations for every prime p ≤ N the inner loop.

- The number of prime numbers less than or equal to n is approximately N / ln(N).
- The k-th prime number approximately equals k*ln(k) (that follows immediately from the previous fact)

We extracted the first prime number 2 from the sum, because k=1 in approximation k * ln(k) is 0 and causes a division by zero.

Now, returning to the original sum, we'll get its approximate evaluation is O(N*log(log(N))).

Space Complexity :

O(**N**)

As we are using N sized Boolean array for storing the prime number values as True and composite number values as False, the space complexity becomes o(n).

Applications of the Algorithm

- For cryptography, you mostly need larger primes than what you can get via sieving.
- Pseudo-random number generators use Sieve of Eratosthenes.
- Generation of hash tables use prime numbers which are easy to compute using Sieve of Eratosthenes
- This algorithm is also used to find all the prime factors of factorial of a number.
- In practical uses, prime numbers are used in cyphers and codes - including your credit card numbers which can be generated using this algorithm
- Prime numbers are extensively used in cryptography and network security. One of the public key encryption algorithm called RSA actually relies on the factorization of product of large primes.

THANK YOU