CSE 5351: Parallel Processing
Homework 2

James Grisham

October 14, 2014

Question 1

Problem Statement

A sequential implementation of the Sieve of Eratosthenes marks about 2.2 million cells in order to compute all primes less than 1 million. Estimate the maximum speedup achievable by the control-parallel (shared memory) version of the Sieve of Eratosthenes as it finds all the primes less than 1 million.

Solution

For the control-parallel implementation, a single processor requires the following time

$$T_1 = \sum_{i=1}^{k} \left\lceil \frac{(n+1) - \pi_i^2}{\pi_i} \right\rceil \tag{1}$$

to mark all cells less than or equal to $\sqrt{n}$, where π_i represents the i-th prime. This algorithm requires that one processor is responsible for marking all multiples of a particular prime number. Because of this, the max speedup is limited by the fact that the number 2 has more multiples than any of the other primes. With a very large number of processors, one processor that is marking all multiples of 2 will take more time than all of the other processors which are marking multiples of the other primes.

There are 168 primes from 0 to $\sqrt{n}$ where $n = 1,000,000$. These primes were downloaded from the web (http://primes.utm.edu/lists/small/1000.txt) and a small program was written to do the calculations. The total serial time, as determined by (1) is

$$T_1 = 2122048 \text{ time units}$$

The minimum parallel computation time is determined by assuming enough processors have been allocated so that the limiting factor is marking multiples of two. Thus, the minimum parallel time is

$$T_n = \left\lceil \frac{(n+1) - 4}{2} \right\rceil = 999999 \text{ time units}$$

Therefore, the max possible speedup is

$$\boxed{S_n = \frac{T_1}{T_n} = 2.12205}$$

Question 2

Problem Statement

Since 2 is the only even prime, one way to save memory and improve the speed of the sequential Sieve of Eratosthenes algorithm is to have the elements of the Boolean array represent only odd numbers. In this scheme, the first sieve step would mark multiples of the prime number 3. Then,

(a) Estimate the reduction in execution time of the sequential algorithm resulting from this improvement for $n = 1000$ and $n = 1000000$.

(b) The improved sequential algorithm can be used as the basis for an improved data-parallel algorithm. Using the machine model of non-shared, distributed memory, and assuming $\lambda = 100\chi$, estimate the execution time of the improved data parallel algorithm for 1, 2, ..., 16 processors.

(c) Compute the speedup of the improved data-parallel algorithm over the improved sequential algorithm. Compare this speedup with the speedup estimated for the original data-parallel algorithm.

(d) Why does the improved data-parallel algorithm achieve different speedup that the original data-parallel algorithm?

Solution

(a) Storing only odd numbers corresponds to neglecting the marking of all multiples of two. That is, the modified algorithm corresponds to the initial algorithm with the first operation taken away. The serial time to accomplish this is then given by

$$T_1 = \chi \sum_{i=2}^{k} \left\lceil \frac{n}{\pi_i} \right\rceil$$

The reduction in serial computation time is

$$\frac{T_{1,\text{original}}}{T_{1,\text{modified}}} = 1.29$$

So, the modified serial sieve is approximately 1.3 times faster than the original.

(b) Again, this corresponds to neglecting the marking of multiples of 2. Therefore, the computation time is

$$T_n = \chi \sum_{i=2}^{k} \left\lceil \frac{\lceil n/p \rceil}{\pi_i} \right\rceil + (k-1)(p-1)\lambda \tag{2}$$

The ratio of original computation time to modified computation time as a function of the number of processors used is shown below.

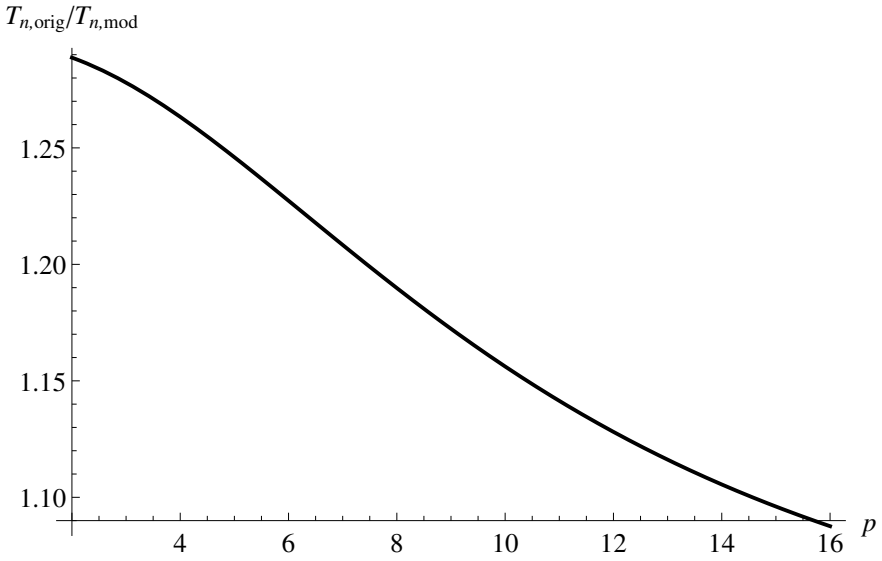


Figure 1: Comparison between original and modified parallel algorithms.

(c) The speedup is computed using the same equation as before. Namely,

$$S_n = \frac{T_1}{T_n}$$

A plot comparing the speedups of the original and improved algorithms is shown below.

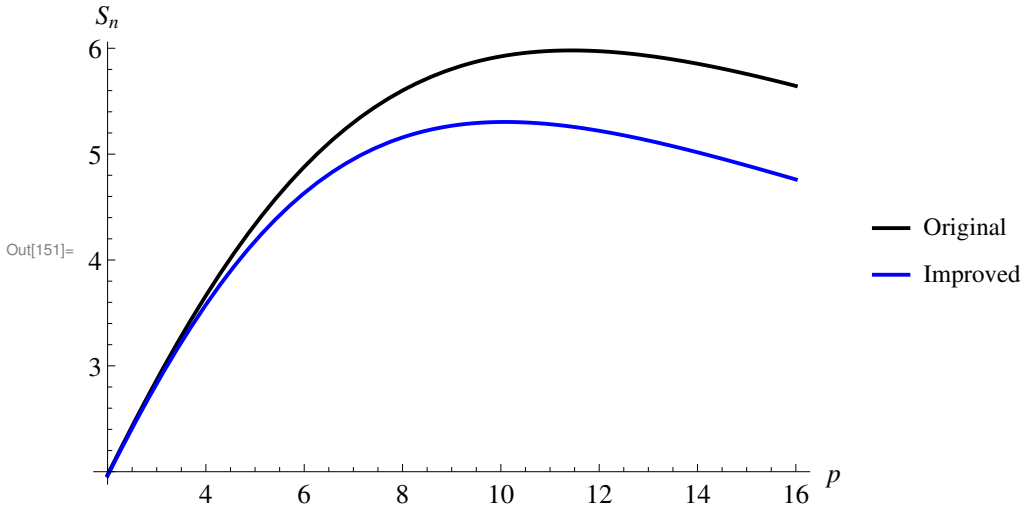


Figure 2: Comparison between speedups for original and modified algorithms.

(d) It appears as though the modified algorithm has a lower maximum speedup than the original algorithm. This could be due to the fact that the problem size has decreased causing a corresponding decrease in the level of parallelism.

Question 3

Problem Statement

Assuming a data-parallel approach on a distributed memory computer, calculate the speedup for $n = 1000000$ and $\lambda = 100\chi$. Repeat the same for $\lambda = 1000\chi$ and $\lambda = \chi$. Calculate these speedups for 2, 3, 4, 5, ..., 16 processors and show the numbers in a table. Also, draw a figure showing the three curves of speedup.

Solution

For this approach, the total serial time is given by

$$T_1 = \chi \sum_{i=1}^{k} \left\lceil \frac{n}{\pi_i} \right\rceil \tag{3}$$

where k represents the number of primes between 1 and $\sqrt{n}$, n represents the upper bound on prime numbers desired, and π_i represents the i-th prime number.

For a distributed memory system, the computation time for p processors is

$$T_n = \chi \sum_{i=1}^{k} \left\lceil \frac{\lceil n/p \rceil}{\pi_i} \right\rceil + k(p-1)\lambda \tag{4}$$

where λ is the communication time required for one processor to pass a number to another processor.

These equations were evaluated symbolically using Wolfram Mathematica. The tabulated speedups are shown below.

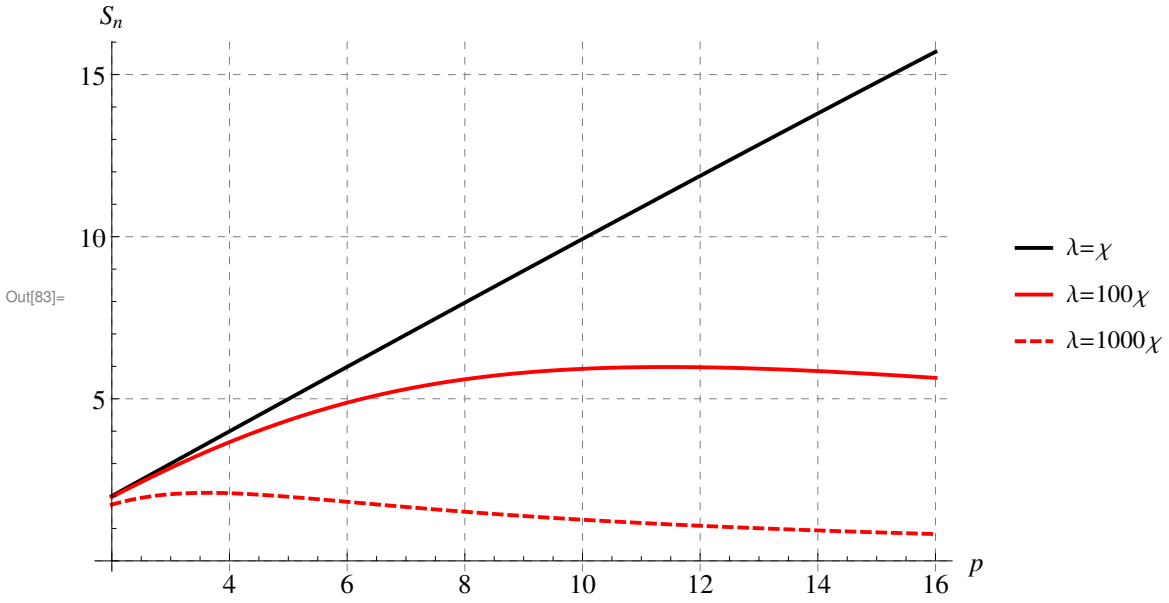


Figure 3: Speedup vs number of processors.

Table 1: Speedups versus number of processors.

	Speedup		
p	$\lambda = \chi$	$\lambda = 100\chi$	$\lambda = 1000\chi$
2	1.99962	1.96982	1.73477
3	2.99836	2.86822	2.0567
4	3.99587	3.66358	2.08633
5	4.99164	4.33651	1.97731
6	5.98507	4.8801	1.82204
7	6.97596	5.29809	1.66264
8	7.96393	5.6013	1.5151
9	8.94781	5.80419	1.38396
10	9.92847	5.92357	1.26923
11	10.904	5.97469	1.16929
12	11.8754	5.9724	1.08217
13	12.8414	5.9288	1.00595
14	13.8011	5.85413	0.938952
15	14.7556	5.75709	0.879755
16	15.7024	5.64401	0.827168

The max speedup for each value of λ as a function of p could be determined by computing the derivative of S_n with respect to the number of processors, setting it equal to zero, and solving for p. That is,

$$\frac{dS_n}{dp} = 0$$

This is effectively solving for the point at which the slope of $S_n(p)$ is equal to zero (i.e., a critical point).

Question 4

Problem Statement

Consider the problem of adding n numbers. Assume that one person can add two numbers in time t_c. How long with that person take to add n numbers?

Now, assume that eight people are available for adding n numbers and that it is possible to divide the list into 8. The eight people have their own pencils and paper (on which to perform addition), are equally skilled, and can add two numbers in time t_c. Furthermore, a person can pass on the result of an addition (in the form of a single number) to the person sitting next to him or her in time t_w. How long will it take in the following scenarios?

(a) All eight people are sitting in a circle.

(b) The eight people are sitting in two rows of four people each.

Solution

If a person can add two numbers in t_c, then the time required for that person to add n numbers is $(n-1)t_c$.

For both cases, the problem could be solved by domain decomposition. That is, if there are n numbers to be summed, each person is given n/p numbers to sum. The time it takes to distribute the numbers to the 8 people is neglected. The time required for p people to solve the problem is $(\lceil n/p \rceil - 1)t_c$.

Next, the communication time must be considered. For everyone sitting in a circle, a reduction could be used. Since the person writing down all the numbers could only listen to one person at a time, it would take $7\,t_w = (p-1)t_w$ to communicate all the data. The person that wrote down all the numbers would then have to sum 8 numbers. Therefore, the time required to perform the computation would be

$$\boxed{t_{\text{circle}} = \left(\left\lceil \frac{n}{p} \right\rceil - 1 \right) t_c + (p-1)\,(t_w + t_c)}$$

The communication process for 8 people sitting in two rows of four is illustrated in Figure 4. Each p stands for a person and each s stands for summing two numbers.

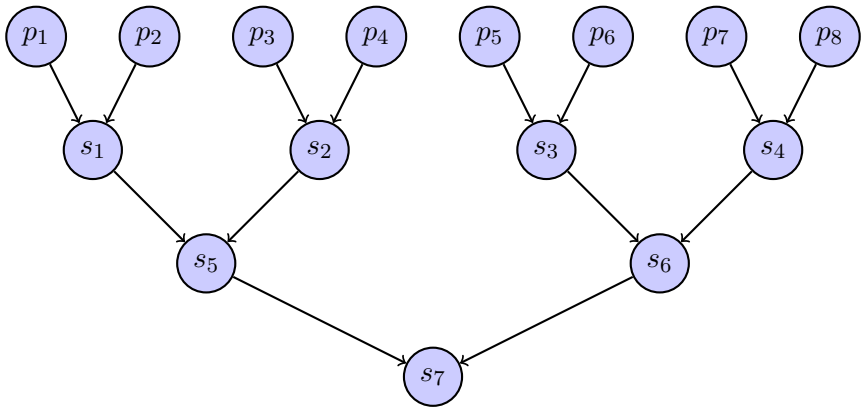


Figure 4: Sum communication.

Person 1 sends their sum to person 2 which requires t_w to communication and p_2 requires t_c to compute the sum of the two numbers.

Extending this logic, p people require the following amount of time to sum and communicate n numbers:

$$\boxed{t_{\text{rows}} = \left(\left\lceil \frac{n}{p} \right\rceil - 1 \right) t_c + (p-1)\,(t_w + t_c)}$$

Question 5

Problem Statement

Describe major differences between SIMD and MIMD computers and their advantages and disadvantages over each other.

Solution

First granularity must be defined as the computation workloak executed between two parallelism or interaction operations (source: course notes). SIMD machines are typically applied to fine grained problems, and are often custom designed for particular applications. These factors make SIMD machines less general and more suitable for specialty jobs such as image processing. SIMD stands for single instruction, multiple data stream. One of the main disadvantages of SIMD is that each processor must execute the same instruction. This makes SIMD machines unacceptable for more general applications.

MIMD stands for multiple instruction, multiple data stream. MIMD machines have a higher level of parallelism as compared to SIMD machines because they allow processors to execute different instructions at each cycle. Multiple instruction streams allows the system to be applied to more general problems. MIMD systems can operate as SIMD systems, but SIMD cannot operate as MIMD. The main problems with MIMD machines also have to do with multiple instruction streams. That is, having to retrieve multiple instruction streams is a disadvantage because it increases overhead. Additionally, load balancing and synchronization cause extra overhead. Managing shared memory MIMD processes is complicated because different processors can operate on the same data simultaneously. This requires extra consideration when developing a program.

Appendices

C++ code

Listing 1: Problem 1 code

```
#include <string>
#include <fstream>
#include <sstream>
#include <iostream>
#include <vector>
#include <cmath>

using namespace std;

int main() {

  // Declaring variables
  string file_name = "prime_data";
  vector<double> primes;
  int* num_primes;
  num_primes = new int;
  double tmp;
```

```
  double T_1 = 0.0;
  double n = 1000000.0;
  double T_n;

  // Opening file and reading data
  ifstream instream(file_name.c_str());
  if (instream.is_open()) {

    // Getting number of lines to read
    instream >> *num_primes;
    cout << "There are " << *num_primes << " primes to read." << endl;

    // Reading lines
    for (int i=0; i<*num_primes; i++) {
      instream >> tmp;
      primes.push_back(tmp);
    }
  }

  // Computing serial time
  for (int i=0; i<*num_primes; i++) {
    T_1 += ceil((n+1.0-pow(primes[i],2.0))/primes[i]);
  }
  cout << "The total serial time is " << (int) T_1 << endl;

  // Minimum parallel time
  T_n = ceil(n+1-pow(2.0,2.0)/2.0);
  cout << "The minimum parallel time is " << T_n << endl;

  // Max speedup
  cout << "Max speedup is " << T_1/T_n << endl;

  // Freeing dynamically allocated memory
  delete num_primes;

  return 0;
}
```

Mathematica Notebooks

Homework 2 Problem 3

In[115]:=
```
ClearAll["Global`*"]
```

Defining inputs

In[116]:=
```
n = 1 000 000;
```

Analysis

In[117]:=
```
(* Primes up to 1000 *)
```

In[118]:=
```
primeNums = Table[Prime[n], {n, 168}];
```

In[119]:=
```
πi = Select[primeNums, # ≤ Sqrt[n] &]
```

Out[119]=
```
{2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83,
 89, 97, 101, 103, 107, 109, 113, 127, 131, 137, 139, 149, 151, 157, 163, 167, 173,
 179, 181, 191, 193, 197, 199, 211, 223, 227, 229, 233, 239, 241, 251, 257, 263,
 269, 271, 277, 281, 283, 293, 307, 311, 313, 317, 331, 337, 347, 349, 353, 359,
 367, 373, 379, 383, 389, 397, 401, 409, 419, 421, 431, 433, 439, 443, 449, 457,
 461, 463, 467, 479, 487, 491, 499, 503, 509, 521, 523, 541, 547, 557, 563, 569,
 571, 577, 587, 593, 599, 601, 607, 613, 617, 619, 631, 641, 643, 647, 653, 659,
 661, 673, 677, 683, 691, 701, 709, 719, 727, 733, 739, 743, 751, 757, 761, 769,
 773, 787, 797, 809, 811, 821, 823, 827, 829, 839, 853, 857, 859, 863, 877, 881,
 883, 887, 907, 911, 919, 929, 937, 941, 947, 953, 967, 971, 977, 983, 991, 997}
```

In[120]:=
```
k = Length[πi];
```

In[128]:=
```
SerialReduction = 1.0 * χ *
  Sum[Ceiling[n / πi[[i]]], {i, 1, k}] / ( 1.0 * χ * Sum[Ceiling[n / πi[[i]]], {i, 2, k}])
```

Out[128]=
```
1.29443
```

In[121]:=
```
Sorig[p_] := (1.0 * χ * Sum[Ceiling[n / πi[[i]]], {i, 1, k}]) /
   ( χ * Sum[Ceiling[Ceiling[n / p] / πi[[i]]], {i, 1, k}] + k * (p - 1) * λ)
```

In[122]:=
```
T1 = 1.0 * χ * Sum[Ceiling[n / πi[[i]]], {i, 2, k}];
```

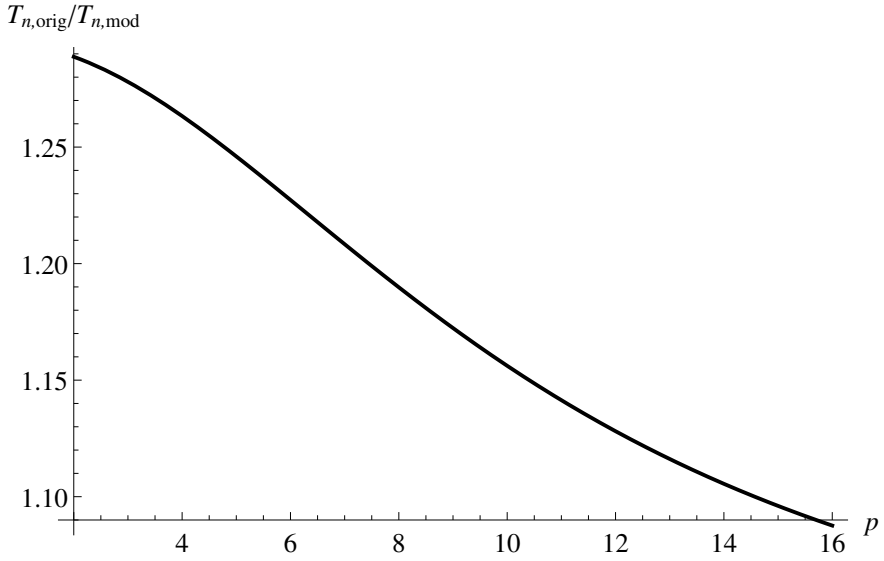
In[132]:=
```
Tnorig [p_] := χ * Sum[Ceiling[Ceiling[n / p] / πi[[i]]], {i, 1, k}] + (k - 1) * (p - 1) * λ
```

In[129]:=
```
Tn [p_] := χ * Sum[Ceiling[Ceiling[n / p] / πi[[i]]], {i, 2, k}] + (k - 1) * (p - 1) * λ
```

In[136]:=
```
ParallelReduction = Tnorig[3] / Tn[3] /. λ → 100. * χ
```

In[149]:=
```
Plot[Tnorig[p] / Tn[p] /. λ → 100 * χ, {p, 2, 16}, PlotStyle → {Thick, Black},
 AxesLabel → {"p", "T_{n,orig}/T_{n,mod}"}, BaseStyle → {FontSize → 14}]
```

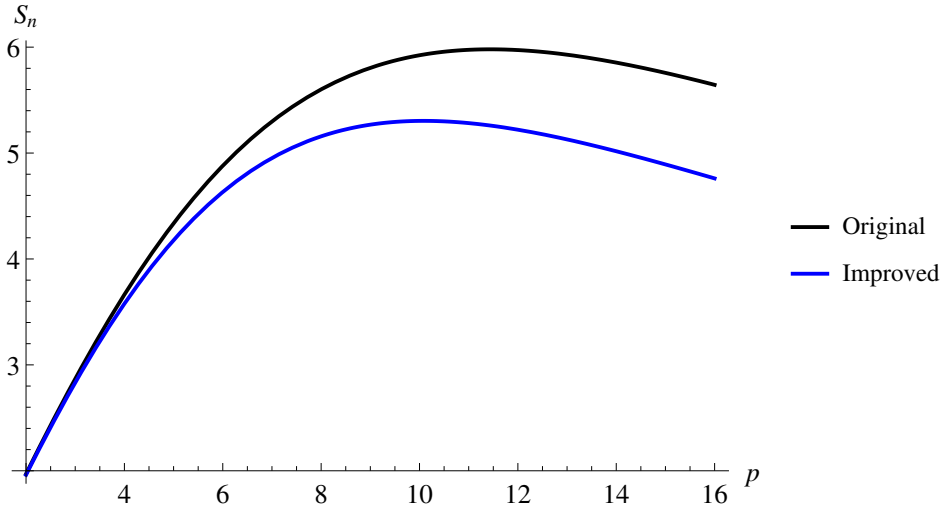
Out[149]=



In[130]:=
```
Snew[p_] := T1 / Tn[p]
```

In[151]:=
```
Plot[{Sorig[p] /. λ → 100 * χ, Snew[p] /. λ → 100 * χ}, {p, 2, 16},
 PlotLegends → {"Original", "Improved"}, PlotStyle → {{Thick, Black}, {Thick, Blue}},
 BaseStyle → {FontSize → 14}, AxesLabel → {"p", "S_n"}]
```

Out[151]=



In[126]:=
```
S[p_] := T1 / Tn[p]
```

Homework 2 Problem 3

```
ClearAll["Global`*"]
```

Defining inputs

```
n = 1 000 000;
```

Analysis

```
(* Primes up to 1000 *)
```

```
primeNums = Table[Prime[n], {n, 168}];
```

```
πi = Select[primeNums, # ≤ Sqrt[n] &]
```

```
{2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83,
 89, 97, 101, 103, 107, 109, 113, 127, 131, 137, 139, 149, 151, 157, 163, 167, 173,
 179, 181, 191, 193, 197, 199, 211, 223, 227, 229, 233, 239, 241, 251, 257, 263,
 269, 271, 277, 281, 283, 293, 307, 311, 313, 317, 331, 337, 347, 349, 353, 359,
 367, 373, 379, 383, 389, 397, 401, 409, 419, 421, 431, 433, 439, 443, 449, 457,
 461, 463, 467, 479, 487, 491, 499, 503, 509, 521, 523, 541, 547, 557, 563, 569,
 571, 577, 587, 593, 599, 601, 607, 613, 617, 619, 631, 641, 643, 647, 653, 659,
 661, 673, 677, 683, 691, 701, 709, 719, 727, 733, 739, 743, 751, 757, 761, 769,
 773, 787, 797, 809, 811, 821, 823, 827, 829, 839, 853, 857, 859, 863, 877, 881,
 883, 887, 907, 911, 919, 929, 937, 941, 947, 953, 967, 971, 977, 983, 991, 997}
```

```
k = Length[πi];
```

```
T1 = 1.0 * χ * Sum[Ceiling[n / πi[[i]]], {i, 1, k}];
```

```
Tn[p_] := χ * Sum[Ceiling[Ceiling[n / p] / πi[[i]]], {i, 1, k}] + k * (p - 1) * λ
```

```
T1 / Tn[2] /. λ → 100 * χ
```

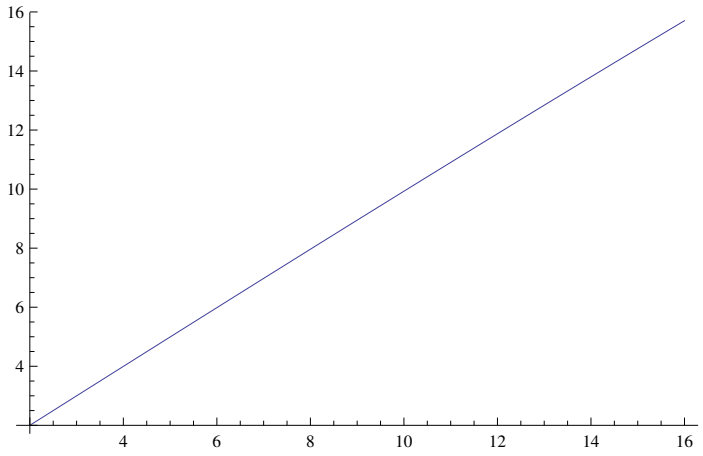
```
results = Table[{p, T1 / Tn[p] /. λ → χ,
    T1 / Tn[p] /. λ → 100 * χ, T1 / Tn[p] /. λ → 1000 * χ}, {p, 2, 16}];
```

```
Grid[results]
```

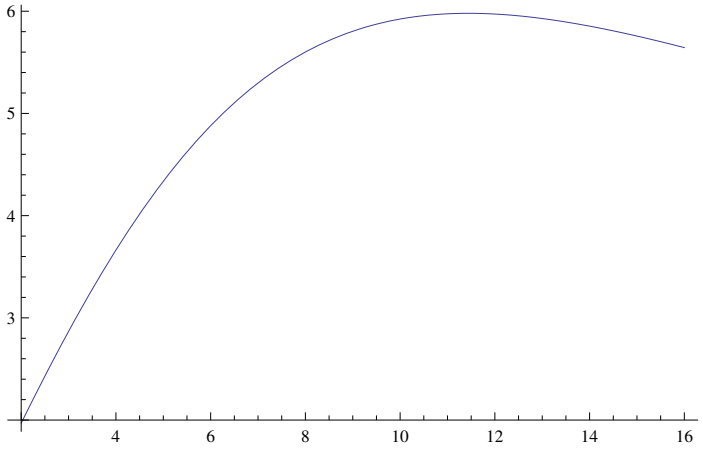
2	1.99962	1.96982	1.73477
3	2.99836	2.86822	2.0567
4	3.99587	3.66358	2.08633
5	4.99164	4.33651	1.97731
6	5.98507	4.8801	1.82204
7	6.97596	5.29809	1.66264
8	7.96393	5.6013	1.5151
9	8.94781	5.80419	1.38396
10	9.92847	5.92357	1.26923
11	10.904	5.97469	1.16929
12	11.8754	5.9724	1.08217
13	12.8414	5.9288	1.00595
14	13.8011	5.85413	0.938952
15	14.7556	5.75709	0.879755
16	15.7024	5.64401	0.827168

```
S[p_] := T1 / Tn[p]
```

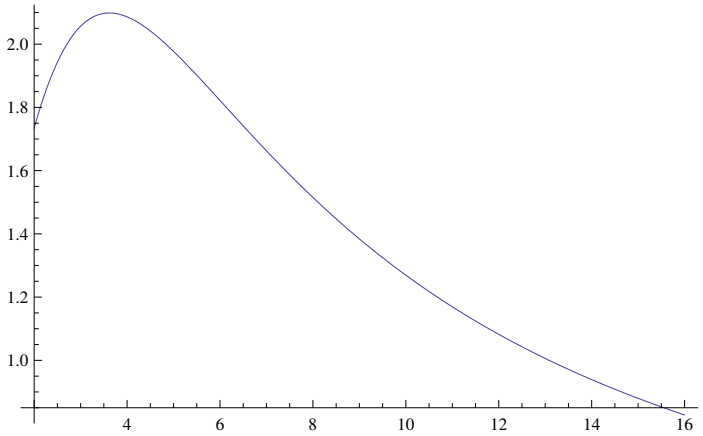
```
ph1 = Plot[S[p] /. λ → χ, {p, 2, 16}]
```




```
ph2 = Plot[S[p] /. λ → 100 * χ, {p, 2, 16}]
```



```
ph2 = Plot[S[p] /. λ → 1000 * χ, {p, 2, 16}]
```




```
Plot[{S[p] /. λ → χ, S[p] /. λ → 100 * χ, S[p] /. λ → 1000 * χ},
 {p, 2, 16}, PlotLegends → LineLegend[{"λ=χ", "λ=100χ", "λ=1000χ"}],
 PlotStyle → {{Black, Thick}, {Red, Thick}, {Red, Dashed, Thick}},
 AxesLabel → {"p", "Sn"}, GridLines → Automatic,
 GridLinesStyle → Directive[Gray, Dashed], BaseStyle → {FontSize → 14}]
```