ISSN : 2394-2975 (Online)
ISSN : 2394-6814 (Print)
Vol. 2, Issue 4 (Oct. - Dec. 2015)
International Journal of Advanced Research in Education & Technology (IJARET)

Finding N Prime Numbers Using Distrusted Computing PVM (Parallel Virtual Machine)

Ehab AbdulRazak Al-Asadi
College of Science-Kerbala University

Abstract
Analysis the feasibility of sequencing as a parallel algorithm for the search for the first N Prime numbers. Design and implement (in C / C ++) program, respectively. Programs in the sequence Algorithm and also a solution based on the transmission of messages between nodes using library system PVM. Both programs must be identical. Distribute the load between nodes so that the computing time A minimum. Find out what is the dependence of acceleration of the implementation and calculation of the number of nodes and the extent of the problem in relation to sequencing program, (insert table and graphs). On the basis of Results estimate: latency communication, for what is the role of the dimension of the architecture (right) Scalable and what is the maximum size when the calculation is available on the architecture more bearable. Discuss in your group about what is the advantage respectively. The efficiency of parallel execution Individual algorithms. If this solution requires the use of input and output files with rows matrix Corresponds to file line and column values are separated by spaces or tabs. Otherwise stated, work with real numbers.

Keywords
Prime Numbers, Eratosthenes sieve, Parallel Program, Mersenne Prime number.

I. Introduction

PVM (Parallel Virtual Machine) is a software system that enables a collection of heterogeneous computers to be used as a coherent and flexible concurrent computational resource. The individual computers may be shared- or local-memory multiprocessors, vector supercomputers, specialized graphics engines, or scalar workstations that may be interconnected by a variety of networks, such as Ethernet, FDDI, etc. PVM support software executes on each machine in a user-configurable pool, and presents a unified, general, and powerful computational environment of concurrent applications. User programs written in C or FORTRAN are provided access to PVM through the use of calls to PVM library routines for functions such as process initiation, message transmission and reception, and synchronization via barriers or rendezvous. Users may optionally control the execution location of specific application components. The PVM system transparently handles message routing, data conversion for incompatible architectures, and other tasks that are necessary for operation in a heterogeneous, network environment.

PVM is particularly effective for heterogeneous applications that exploit specific strengths of individual machines on a network. As a loosely coupled concurrent supercomputer environment, PVM is a viable scientific computing platform. The PVM system has been used for applications such as molecular dynamics simulations, superconductivity studies, distributed fractal computations, matrix algorithms, and in the classroom as the basis for teaching concurrent computing.Eratosthenes sieve

2. Analysis

A. Prime Number

In mathematics, a prime number (or a prime) is a natural number which has exactly two distinct Natural number divisors: 1 and itself. Infinitude of prime numbers exists 2, 3, 5, 7, 11, 13, 17, 19, 23, 29,……[1]

Perhaps the most rediscovered result about prime's numbers is the following:

I found that every prime number over 3 lies next to a number divisible by six. Using Matlab with The help of a friend, we wrote a program to test this theory and found that at least within the first 1,000,000 primes this holds true. Checking a million primes is certainly energetic, but it is not necessary (and just looking at examples. Can be misleading in mathematics). Here is how to prove your observation: take any integer n greater Than 3, and divide it by 6. That is, write $n = 6q + r$ Where q is a non-negative integer and the remainder r is one of 0, 1, 2, 3, 4, or 5. if the remainder is 0, 2 or 4, then the number n is divisible by 2, and cannot be prime. if the remainder is 3, then the number n is divisible by 3, and cannot be prime. So if n is prime, then the remainder r is either 1 (and $n = 6q + 1$ is one more than a multiple of six), or 5 (and $n = 6q + 5 = 6(q+1) - 1$ is one less than a multiple of six).[2]

B. Prime Numbers Group

Mersenne Prime number

In mathematics, a Mersenne number is a number that is one less than a power of two, $Mn = 2n - 1$.

As of August 2007, only 44 Mersenne primes are known; the largest known prime number (232,582,657−1) is a Mersenne prime and in modern times the largest known prime has nearly always Been a Mersenne prime. Like several previous Mersenne primes, it was discovered by a distributed Computing project on the Internet, known as the *Great Internet Mersenne Prime Search* (GIMPS). [1]. Prime numbers have long fascinated amateur and professional mathematicians. An integer greater Than one is called a prime number if its only divisors are one and itself. The first prime numbers Are 2, 3, 5, 7, 11, etc. For example, the number 10 is not prime because it is divisible by 2 and 5. A Mersenne prime is a prime of the form 2P-1. The first Mersenne primes are 3, 7, 31, 127 (Corresponding to P = 2, 3, 5, 7). There are only 44 known Mersenne primes.

Fibonacci primenumber

The prime number which is synchronize fibonacci number series numbers in the each second. Member is counts two former, however not every fibonacci number is prime number 2, 3, 5, 13, 89, 233, 1597, ...

ISSN : 2394-2975 (Online)
ISSN : 2394-6814 (Print)

Emirp

An Emirp (*prime* spelt backwards) is a prime number that results in a different prime when its Digits are reversed. This definition excludes the related palindromic primes. Emirps are also called Reversible primes. The sequence of emirps begins 13, 17, 31, 37, 71, 73, 79, 97, 107, 113, 149, 157..

C. Algorithms for Searching the Prime Numbers

To create list of prime numbers existing different algorithms from those remember two – Eratosthenes Sieve a Brute Force

Eratosthenes sieve

The basic algorithm to find the prime number and his properties **A *prime number*** is a natural number greater than 1 that can be divided without remainder only by
Itself and by 1. Natural numbers *n* that can be divided by a number less than *n* and greater than 1
Are *composite* numbers. The Sieve of Eratosthenes identifies all prime numbers up to a given
Number *n* as follows:

1. Write down the numbers 1, 2, 3, ..., n. We will eliminate composites by marking them. Initially all numbers are unmarked.
2. Mark the number 1 as special (it is neither prime nor composite).
3. Set *k=1*. Until *k* exceeds or equals the square root of *n* do this:

o Find the first number in the list greater than k that has not been identified as Composite. (The very first number so found is 2.) Call it *m.* Mark the numbers
o *2m, 3m, 4m,*
As composite. (Thus in the first run we mark all even numbers greater than 2. In the Second run we mark all multiples of 3 greater than 3.) *m* is a prime number. Put it on your list.
o Set *k=m* and repeat.
. Put the remaining unmarked numbers in the sequence on your list of prime numbers. [7]
This algorithm can be written in pseudo-code as follows

```
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)
}
```

For example, here's a beginning array.

2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27
Since 2 is unmarked, it is our first prime. We mark every second integer, that is, 4, 6, 8, 10, 12, etc.
2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27
The next unmarked integer is 3, so it is prime. We mark every third integer, i.e., 6, 9, 12, etc. Note
That we mark 6, 12, 18, etc. again.
2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27
Now 5 is the next prime, and we mark every fifth integer. The only new integer marked in range is
25.
2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 from here we
Find the primes 7, 11, 13, 17, 19, and 23**.[2]**

D. What Can Influence the Solution and the Results

The system performance and distributed computation performance (in compare with sequential Computation) are influenced by several conditions: Performance of computers themselves – processor performance, memory Network and its capabilities that connects the hosts Actual … (it depends on the number of the hosts and of its usage with other processes)

III. The Design

A. Mathematical Assumption

In computation we will take advantage of these mathematical formulas:
All primary numbers (bigger than 3) are in form: primary number = 6n ± 1, where n=1,2,3,.. to know whether the number *X* has divisors (except 1 and itself) it is sufficient to divide it By numbers from 2 to √X. First two primary numbers (2 and 3) are known – first formula is valid for numbers bigger than 3.

B. Tasks Division

I'll divide the task so that the architecture *master-worker* will be created. *Master's* task will be to Process input data (number of nodes on which the computation will be executed and the number of Wanted primary numbers), dividing data for processing in *worker* nodes, process the data from *Worker* nodes and evaluate results (storing the results in files). The *worker's* task will be to decide whether a number (sent by *master*) is/ is not a primary number And to send the result back to *master*.

C. Problem with Synchronization

During the calculations can occurs the showing below, when instead the index of first N prime Number obtain index, which have otherwise N prime number, but some of them is/are skipping. For better understanding introduce following example for N=6 and for 3workers, where instead of index 13 ,11 ,7 ,5 ,3 ,2 obtain index.17 ,13 ,7 ,5 ,3 ,2
Explanation given situations:

- Numbers 2 a ,3 which,, know "doesn't Account (test) and the numbers 5, 7, 11, ... (relation 6n±1)
- let distribution numbers for the three workers as following: worker1→5, worker2→7 and worker3→11 .
- Worker1 finish testing, sendoff result to the master and the master send the next number (13) For testing.
- Worker2 accordingly finishes the testing, sendoff result to master and the master will send
- The next number (17) for testing.
- Worker3 always carries out account; let us say he has it hard to send the result to the master.
- Worker1 a worker2 finish testing and they send the results to the master.

- Master find out, that the count of the prime number is satisfactory and send to the worker The information with negative number – imperative for ending activities.
- So will obtain following list prime number: 2, 3, 5, 7, 13, 17

suggestion of the following mode solution remember problem:
After reaching, the requirement of the number of prime number unsent at a moment the
Information with negative number notifying End-Of-Job, but waiting for ending calculations of
Others workers .so we can otherwise obtain upwards of N prime number (if the informations from
Workers, for those,, waiting "will them positive numbers – number which tester is prime number).
Well at output file will be find nevertheless only N prime number.

D. Input Request and Output Form

As the input we will take for see the following statement

- Number of nods, on those has to realize the account – positive number <= 15 (in Laboratory we have cluster with 15 computers)
- Number offending the prime number – positive number >= 0 Output will be, in case will successful quits calculations, index with the given counts of prime

Number (positive number). This information will be saved to the file, where single numbers will
Compartment separation eventually tabulator.

IV. Implementation

Solution will be implemented in C/C++ language and establishment for transport the information
Between the nods with employ commands library of PVM.
In the laboratory we have to instruction the cluster with 15 computer following configurations:
· Intel Celeron 700MHz – 128kB cache
· 256MB RAM
· Operating system Red Hat Linux release 7.2
· Network connection 10/100 Mbps – Ethernet
· PVM versions 3.4.3
. G++ version 2.96

A. PVM Functions

The basic function, which we used from library PVM for implementing [10]

- **pvm_exit()** – Tells the local pvmd that this process is leaving PVM
- **pvm_initsend()** – Clear default send buffer and specify message encoding.
- **pvm_mcast()** – Multicasts the data in the active message buffer to a set of tasks
- **pvm_mytid()** – Returns the *tid* of the calling process
- **pvm_parent()** –Returns the tid of the process that spawned the calling process
- **pvm_perror()** – Prints message describing the last error returned by a PVM call.
- **pvm_pk*()** – Pack the active message buffer with arrays of prescribed data type.
- **pvm_recv()** – Receive a message
- **pvm_send()** – Immediately sends the data in the active message buffer
- **pvm_spawn() -**initiation new PVM process
- **pvm_upk*()** – Unpack the active message buffer into arrays of prescribed data type

B. Used Data Structures

I created the dynamically linked list to store the primary numbers.
Its' items (nodes) have the
Following structure:

```
typedef struct Number
{
int prime; //primary number
Number* back; //pointer to previous node
Number* next; //pointer to next node
} Number;
```

As this we will establishment the index with 6 elements as shown below.

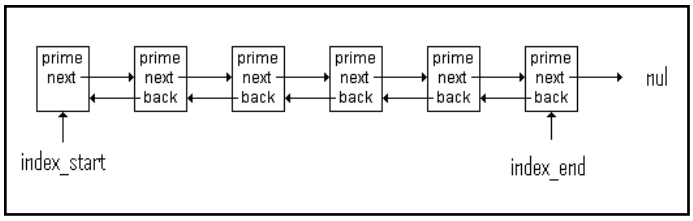


Fig. 1 : Index of elements

The reason why a dynamic linked list instead of a static array was created was that in the end of
The computation process I'll gain a sorted list of primary numbers.
Disadvantages are in a bigger
Overhead which is needed when allocating corresponding list members.
A problem that is solved by this was in different computation time in nodes *cluster*. Because if I
Send numbers to nodes to test them in order N1, N2, N3 ... I don't have any warranty in distributed
Computation that the results will return in the same order as they were sent. If I sent 5 numbers to 5
Workers to test and they would start to send the result (whether the number is primary number or
not), I would not be able to decide where to store the result because I would not know whether the
Next received result will be also primary number and less than previous one.
The next (mentioned) option is to store received primary numbers into a static array. But this
Implies this problems:

The numbers in array have to be sorted in ascending order – need to reorder.
If we don't sort the numbers it is not guaranteed that we will find complete list of *N* primary
Numbers. It is true that we would gain the list of primary numbers but some other primary numbers
Will be missing. More concrete description of this problem is described **Problem with synchronization**
Next used data structure is array ***worker_tids*** that contains TaskIDentifier *worker*. This array has
As many members as workers
The last used data structure is array ***index_worker***which contains pointers to elements of
Dynamically linked list of primary numbers. The array has as

ISSN : 2394-2975 (Online)
ISSN : 2394-6814 (Print)

many members as workers.. On the
Figure (2) is show usage of ***index_worker***for 4 *workers*.

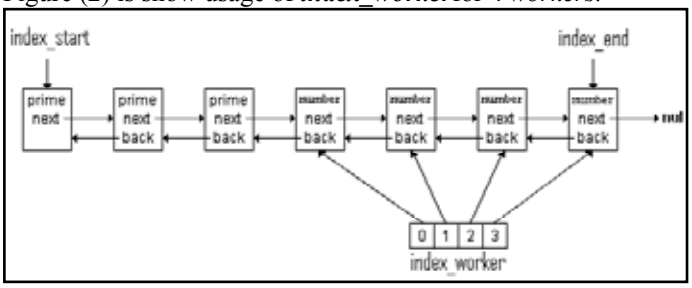


Fig. 2 : the Worker Index

The usage of this array is as follow. Every time I send a number to a *worker* to test I'll create a new
Element in the end of the linked list. I'll put the last number to the element. I'll store in array
index_worker to position **i** a pointer to this element. To the *worker* not only the tested number is
Sent but also a number of positions **i**, that will be sent back with the result in the response. I'll use it
To figure out (with array ***index_worker***) the element of linked list that contains the tested number
By the *worker*. Based on the result the next activities are realized:
If the tested number is primary number and wanted number of primary numbers wasn't
Reached yet, a new element will be created in the end of the linked list, that will contain the
Number that will be tested.
If the tested number isn't primary number it will be moved to the end of the list and a new
Number will be inserted to test.
For the representation of primary number I used data type INT that is of range of ,147,483,648 to
+2,147,483,647.

C. Master

In this file will find element algorithm, which it is not possible execute in parallel. After the
Master initiation we check accuracy of inputs parameters. In case, that the information incorrect
Will accrue error message and the program will end, when the input data are correct, will
Create many workers how many was entered as input parameter**.**
As in Fig (3) shown below
In the next step the master will send to every worker one number for testing (whether is/isn't
Prime number). So in parallel will test several numbers – every number will test by one worker
Numbers which will send are determinate in following relation: number = 6n ± 1 (one worker
Will get the massage 6n + 1, another 6n – 1) where n=1,2,3, ...
Additionally the worker will send index from the pole index_ worker, by which way then the master
Orientate the testing numbers in linking index.

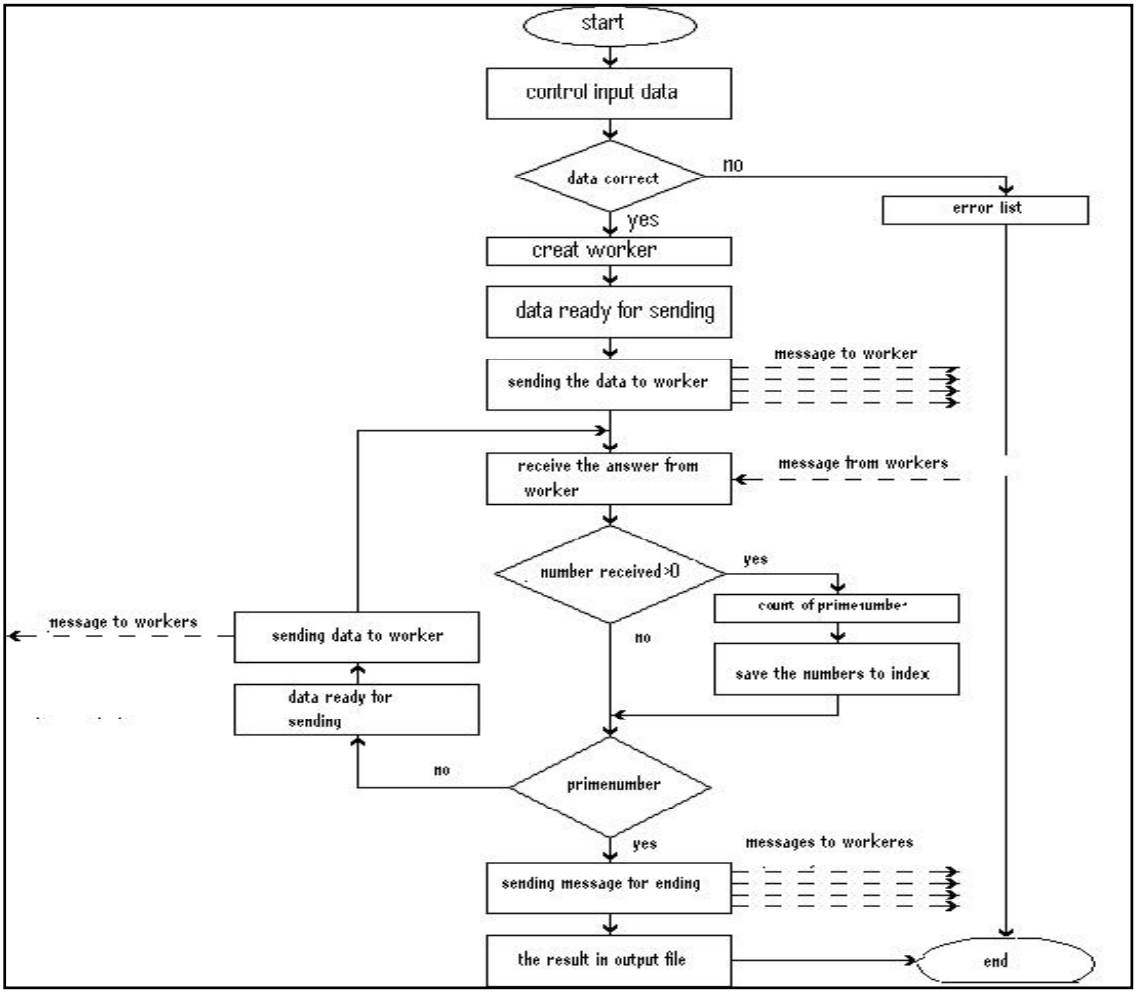


Fig. 3 : Master Algorithm

D. Worker

In this file will find the program part, which is possible to execute in parallel. Is find out whether
Receive the number is/isn't prime number and the result notify to the master. **As in fig (2)**
Shown below

After initialize the worker will waits the message from the master. In the message will find the
Positive number, which be needed for testing whether is prime number, or there are number less
Than null, which worker will notify that End-Of-Job of the program. Next in the message will find
Index i to pole index_worker , which worker send with the testing result backward to the master as
Follows:

- testing number X will gradually divide numbers from 3 to √X, with step 2 (3, 5, 7, ...) Number 2 we don't need to divide it, because the testing number nevertheless Be even-numbered (because number = 6n ± 1),
- For some numbers is remainder from division 0, number isn't prime number and testing

Will ending.
In case, that number is prime number will send off master as positive number. In case, that is not
Prime number will send off to the master negative number.
After acceptation the negative numbers (imperative for ending) worker will send off to the master
Runtime and calculations time. After sending the message worker will end his job

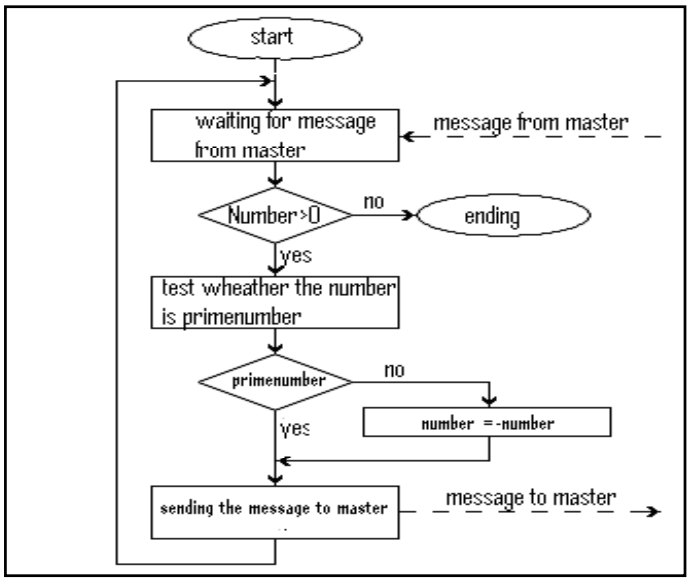


Fig. 4 : Worker Algorithm

V. Running instruction

A. Program Parts

Consequential application to searching the first N prime number is generation in file master.c and
worker.c.

B. Compilation

Compiling will realize in the following statement:
master.c
g++ PrimeNumber_Master.cpp –o master -I /usr/share/pvm3/include -L
/usr/share/pvm3/lib/LINUX -lpvm3
Worker.c
g++ PrimeNumber_Worker1.cpp –o worker -I /usr/share/pvm3/include -L
/usr/share/pvm3/lib/LINUX -lpvm3

C. Initialization Program

Program will starts executable statement:
./master –n {number of searching prime number} –w {number of worker}-I { Size of interval } -
?{list of help}
In case not execute some of the input data, will apply default valuenumber of

- Searching prime number – 0
  Number of worker – 1

VI. Data measurement

A. Data Measurement Process

On time measurement was she employed function ***clock_t times(struct tms *buf*** by which is possible to
Obtain three times: real, user, system[13]
Be measured not only time execution of the master but also workers, because he account the realize alone.
For master will measure time from running till about ending the activities .
For worker will measurement two times: time from running till about ending the activities and the time,
Which worker spend it for only calculations (testing, whether number is/isn't prime number).
Times from several workers were the averaged. Reason namely, that the workers they have the same
Computing loaded. Options to the average value can be maximal time of worker's execution
About program termination I will get 9 data time:
§ Real, user, system runtime of master,
§ Average real, user, system runtime of workers,
§ Average real, user, system computing time of workers.
From this data we do the table for average real runtime of workers and average real computing time of
Workers,
Real time namely introduce that time, about which must we waiting to get the results

B. Calculation Process

Average value

$$E(X) = \overline{X} = \frac{1}{n}\sum_{i=1}^{n} x_i$$

E(X) – average value
n – count of measure value
x_i – i-measure value

Speed up

$$S(n) = \frac{T(1)}{T(n)}$$

S(n) – speedup to N nods
T(1) – runtime of the programe in one node
T(n) – runtime of the programe in N nods

Speed should be account like ratio problem solution time for sequential T(1) and the problem
Solution time in parallel on n nods T(n).
Otherwise also the account on one node employed the library of

ISSN : 2394-2975 (Online)
ISSN : 2394-6814 (Print)

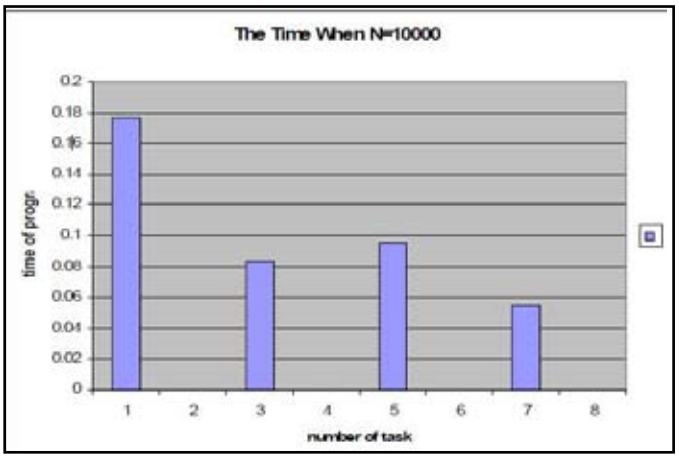
system PVM, well the messages
They are not transport over the networks between the cluster's nods and not accrue the delay
Infliction for transmitting the messages. However we must know that, the certain delay here is,
Which is infliction just call the functions to sending and receiving the messages through the PVM
And administration the connection with exchange the message in on nod.
For time measurement execution on the one node, if we start the program with the number of task
Is 1, so in one nod running the master and one worker.
The speedup for 4 tasks for searching 10000 prime numbers is:
S(4)= T1/Tn =0.1704/0.0932=2.12
Account of latency communication
The latency communication for 1 task and searching 10 000 prime number

TL[S]=T(real)-T(user)=0.1764-0.006=0.05

Tables of Measurements and Graphs

Table 1 : Measurement for N=10000

Number of Tasks	1		4		9		12	
	Real time	User time	Real time	User time	Real time	User time	Real time	User time
1	0.083	0.01	0.131	0	0.122	0.01	0.019	0.01
2	0.071	0	0.059	0.01	0.1	0	0.022	0.02
3	0.63	0.01	0.04	0.01	0.11	0.01	0.06	0.01
4	0.048	0.01	0.126	0.01	0.065	0.01	0.06	0.01
5	0.05	0	0.06	0.02	0.079	0.01	0.03	0.01
Average	0.1764	0.006	0.0832	0.01	0.0952	0.008	0.0552	0.014
Speedup			2.12		1.85294		3.1956	
Latency	0.05		0.04		0.069		0.0421	



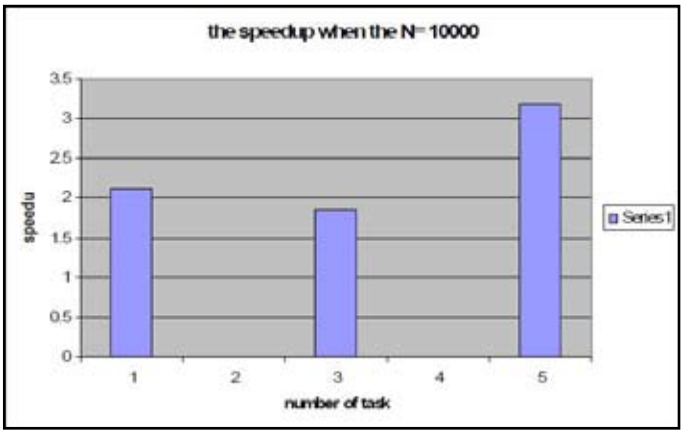
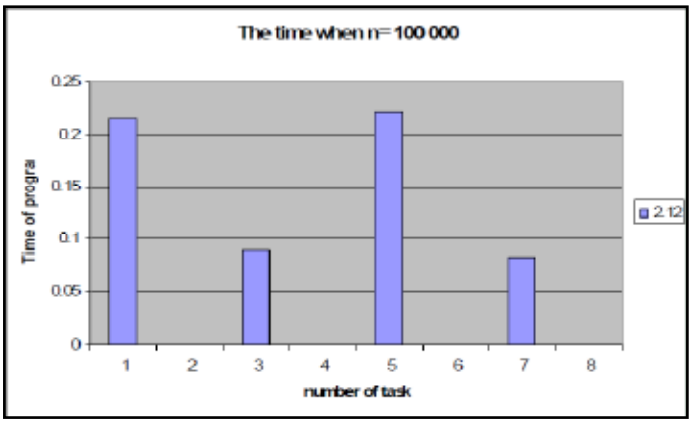
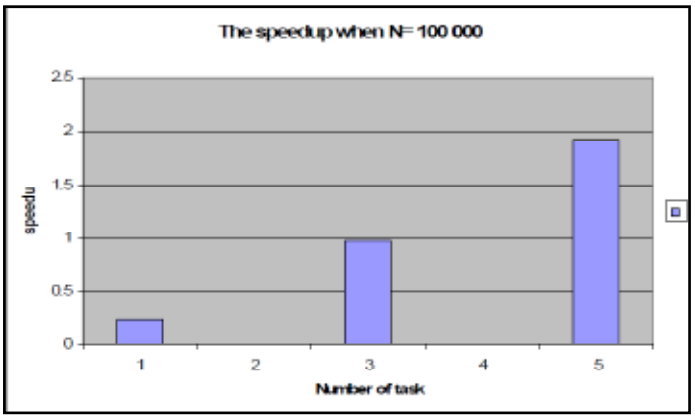


Table 2 : Measurements for N =100 000

Number of Tasks	1		4		9		12	
	Real time	User time	Real time	User time	Real time	User time	Real time	User time
1	0.074	0.00	0.082	0	0.73	0	0.07	0.01
2	0.093	0	0.132	0.01	0.077	0	0.103	0.02
3	0.69	0.02	0.071	0.0	0.098	0	0.04	0.0
4	0.132	0.01	0.078	0.0	0.081	0.01	0.113	0.02
5	0.084	0.02	0.088	0.01	0.120	0.01	0.117	0.01
Average	0.2146	0.012	0.0902	0.0625	0.2212	0.004	0.0818	0.01
Speedup			0.2379		0.70		2.6234	
Latency	0.2026		0.025		0.2172		0.0718	





VII. Conclusions

Time necessary to testing, whether given number is/isn't prime number, with increased number
Stamp mill (whether needed for testing number divide it to several numbers).
If we find little bit of the prime number (like testing 10 000) with using as example 10 tasks and
Then we searching with using the same number of task to several prime numbers (like 100 000),
Latency communications will decrease (that latency communications, which I am regard I). It is
due to just remember actuality.
Have like 10 tasks (10 workers). Master at the beginning distribute every worker the number to
Testing. While send the message has already at input packet message with the result from the
Testing is to excess faster.
However will need testing number gradually increasing, the increased also the time is necessary for
His testing. Master has sufficient time to sending the messages with the number for testing.
Tasks are for all that good scaling just for large number of research prime number.

References

[1] *CALLDWELL, K. Chris: The Prime Pages: Prime FAQ: Are all primes (past 2 and 3) of the forms 6n+1 and 6n-1.*
[2] *Hans Riesel "Prime Numbers and Computer Methods for Factorization (Progress in Mathematics) Birkhäuser Boston; 2nd edition (October 1, 1994) Edition,*
[3] *PVM" Parallel Virtual Machine: A Users' Guide and Tutorial for Network Parallel Computing", the MIT Press (November 8, 1994).*
[4] *GIMP – The Great Internet Mersenne Prime Search.*
[5] *Hesham El-Rewini ,"Advance computer architecture and parallel processing"John Wiley & Sons, NJ, 2005.*
[6] *Jordan, H. F., Jordan, H. E. Fundamentals of Parallel Computing, Prentice Hall, 2002.*
[7] *S. G. Akl, ``The Design and Analysis of Parallel Algorithms," Prentice_Hall, Englewood Cliffs, NJ, 1989*