Performance Optimization by Integrating Memoization and MPI_Info Object for Sieve of Prime Numbers

Haraprasad Naik
Asst. Professor
Dept. of Computer Science,
Utkal University, Odisha, India

Mousumi Mishra
Graduate Student, M.Sc
Computer Science,
Utkal University, Odisha, India

Gayatri Routray
Graduate Student, M.Sc
Computer Science,
Utkal University, Odisha, India

Megharani Behera
Graduate Student, M.Sc
Computer Science,
Utkal University, Odisha, India

ABSTRACT
Sieving prime numbers is an idle example of linear time algorithm since the first sieve of this kind proposed by Eratosthenes. Afterward many sieving algorithm are proposed such as-: Sieve of Sundaram, Sieve of Atkin, Sieve of Sorenson and wheel factorization of prime number. In this paper we have proposed the integration of parallelism with these sieving algorithm. We have proposed MPI_Info object with memoization to avoid redundant steps during prime number processing and adding them into the sieve. Nevertheless this paper also demonstrates the MPI Binding with familiar/popular object oriented programming language such as-: C++ and Java. This binding done through the two different tools which includes OpenMPI and MPJ Express.

Keywords
Java, Open MPI, Prime Table, Java Native Interface, Message Passing

1. INTRODUCTION
MPI is the acronym of Message Passing Interface, which is a standard of communication when the program requires parallelism[5]. MPI is a collection of multiple advanced routines. MPI functions are soly responsible for various task such as spawning, barrier synchronization as well as set operation.

One of the advantages of using MPI over traditional processes and sockets is the ability of launching multiple executable simultaneously or concurrently. There are two main MPI implementation available such as Open MPI, MPICH2. Every MPI program should atleast have MPI_Init() and MPI-Finalize(). MPI-Init is used for the calling task, it must be called before any other MPI functions. For Example:- MPI_Init(&argc,&argv);. These command line arguments are passed in Init as a "command" and "variable" that are relevant to it. The MPI_Finalize() routine deallocates resources from the MPI task no more MPI functions are called after MPI_Finalize() has been declared. For Example:- MPI_Finalize();.

In this paper represented the method of implementations of MPI_Info object has been represented, by which the performance can be improved. The method of implementation of MPI_Info object through which the performance of algorithms meant for sieving the prime numbers can be improved. We have also integrate the concept of memoization through which the redundant processing are avoided.

Java Native Interface is the principal approach of this paper. Java has been proposed since a long ago, the feature of high-performance computing became very significant for its appealing features, any java program with this feature will create a optimized program in terms of parallelism.

MPI is a popular library which is generally used in HPC application. For abstraction of many details of the underlying networking(S). MPI can be implemented using three types of communications schemes:-

1. Point to point
2. One sided.
3. I/O

In point to point communication scheme a pair of peers can exchange message which are belongs to different MPI process. In One sided communication scheme Messages can be interchanged among two processes, where one process is not explicitly participating in the operation. In I/O communication scheme multiple MPI processes can simultaneously access storage subsystems.

MPI communicator is a collection of MPI processes. The API of MPI is responsible to utilize multiple types of handles. In the case of MPI communicator the handle is MPI-COMM. In java the MPI handles are presented as Objects[11]. Several MPI implementations available, Open-MPI is one of the most widely used and it is developed as a coordinated international effort.

MPJ Express is an extended form of open-MPI which can be used in support of object oriented programming. The java programming language has similar syntax with C and C++. Its main advantages is that its source files are translated into byte code which is an intermediate code that is executable on JVM[11].

Java programs are platform independent i.e any program can be executed over another system which has JVM without recompiling the program[11]. The Java Development Tool Kit (JDK) is used to develop software which contains compiler, debugger, and other tools that provides a architecture independent way to implement host centric features like graphics, thread, networking and file management. The Java Native Interface (JNI) is used to call/invoke Remote subroutines from many different APIs and vice-versa.

The MPJava, MPJExpress and F-MPJ are some examples of developing java binding on a pure java basis by implementing socket or RMI. Usually a java method calls MPI primitive through JNI and extract the result that can be return to java method. public static void main (String[] arg) throws MPIException {

MPI.Init (arg) ;

```
int r = MPI.COMM_WORLD.getRank(),
s  =  MPI.COMM_WORLD.getSize(),
delay = 100;
double hi = 1.0 / (double)delay ,
summ  =  0.0;
for (int j = r + 1; j <= delay; j += s){
double xi = hi* ((double) j -0.5);
summ += (4.0/(1.0+xi*xi));       }
double sBuffer [] = { hi * summ },
rBuffer [] = new double [1];
MPI .COMM_WORLD .reduce
(sBuffer, rBuffer, 1, MPI .DOUBLE, MPI .SUM, 0) ;
if (r == 0) System.out.println("PI: "+ rBuffer [0]) ;
MPI.Finalize();}
```

Exception Handling:-
Exception handling is the general method to deal with errors. Exception handling is supported by both open MPI and MPJ Express. MPIExceptions is thrown by every java method if MPI primitive called through JNI which returns with an error code rather than MPI_SUCCESS. The java API may return exceptions if the MPI .ERRORS_RETURN is set as follows:

```
MPI.COMM_WORLD.setErrorhandler(MPI.ERRORS_RET
URN);
```

The try_catch block can be used to separate the main application code from error handling code.

```
try {
File f = new File
(MPI .COMM_SELF, “filename”, MPI  .MODE_RDONLY);
}
catch(MPIException ex) {
System.err.println("Error Message: "+ ex.getMessage() ) ;
System.err.println("Error Class: "+ ex.getErrorClass() ) ;
ex.printStackTrace() ;
System.exit(-1) ;
}
```

Point to point communication:-
The methods of the communication class can be used to implement point to point communication. The method send() and recv() operations receives arguments such as the message, number of elements and datatype. An Example of point to point communication can be stated as below:

```
Comm com = MPI.COMM_WORLD;
int m = com.getRank() ;
if (m == 0)
{
com.send(data,5,MPI.DOUBLE,1,1);
} else if(m == 1){
Status sts = com.recv
(data, 5, MPI .DOUBLE, MPI .ANY_SOURCE, 1) ;
int c = sts.getCount(MPI.DOUBLE);
int s = sts.getSource() ;
System.out.println("Received "+ c+"values from"+s);
}
```

The memorization can be facilitated through user-driven workflow. A general approach to create a parallel application can be described as three steps workflow:

1. Scripting of code
2. Compilation
3. Running in a parallel architecture

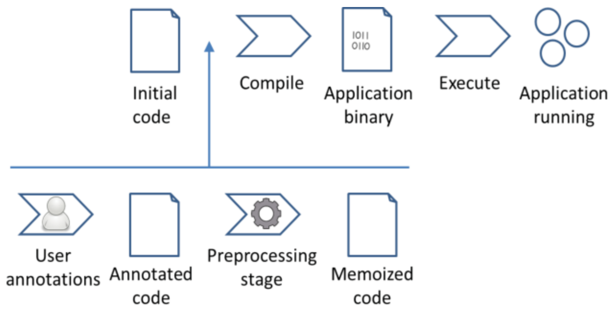


Figure-1(The user driven workflow)[3]

A set of annotations can be added to the existing code in order to specify which function can be memoized. The annotation includes information regarding I/O parameters and their respective size.

Now a days a massive amount of data are created by almost every application of data processing. For Example:- many business application are trying to receive or analyze the customer behavior of their business. Therefore a huge storage space is needed in ordered to store these data.

As an alternative cloud computing is a solution of data, however the parallel and distributed processing of data are cumbersome in data retrieval and processing. Recently the MPI forum is actively working for the integration of cloud computing with the MPI_Info[3] object for a better MPI standard .

In particular the consolidation of multilevel multi core system open the operating to add a new functionality to MPI in order to improve the performance of computing intensive application by integrating with many different technology such as OpenMP, MPJexpress and CUDA.

In this context MPI_Info object has been discussed in details.

MPI_Info object:- Info object of MPI is an opaque object of type MPI::Info in C++.

It keeps hash keys which comprises of (key, value) on ordered pair where both key and value are string type. Usually the MPI_Info object are passed as an argument to many of the MPI routines or functions.

MPI_INFO_GET_NKEYS, MPI_INFO_GET_NTHKEY must written all pairs, show that layered functionality can also used the info object.

The keys has a maximum length of 32 characters to 255 characters, because the set of known keys will always be finite.

It is important to note that the keys are unique and the keys value pair are implemented with the help of link list.

There a new MPI_Info object has been proposed which is based on block chain. In this work the focused is on existing MPI_Info performance and our new proposal will enhanced the performance of MPI_Info object by adopting the following goals :-

1) Implementation of block-chain structure to store the key value pair.

2) The introduction of memoization will reduce the redundant computing.

Reconstruction of MPI_Info object:-

As it has been stated earlier MPI_Info object is an on ordered set of key value pairs, where both key and values are of string type. Usually MPI_Info object is stored as the link list structure by many of the MPI format.

Our proposal is to change the data structure of MPI_Info object implementation from the existing link list to the block chain data structure.

The main benefit of our proposed data structure is the Non-immutable which promotes the opaque characteristics of MPI_Info object.

To implement the same, we binding the MPI_Info object with C++ object oriented programming language.

There are create routine MPI::Info object can be written as:

```
int MPI::Info::create (MPI::Info *info)

{

return MPI::Info::create_distributed

(MPI_COMM_SELF , info) ;

}

Int MPI::Info::create_distributed

(MPI_Comm comm., MPI::Info *info) {

}
```

More over, if two different processes are executing in two different machine a new block chain must be added, otherwise if two processes are executing in a single machine then, the general hash table can be used to avoid the inter communication complexity.

Prime Sieve

A number of algorithm has been proposed to generate prime numbers ranging from simple to complex methods. The sieves for computing primes include a good performance in RAM as well as a good I/O performance. The proposed algorithm are Sieve of Eratosthenes[1], Sieve of Sundaram, Sieve of Atkin and Sieve of Sorenson[10].

Sieve of Eratosthenes is an efficient and easy method for finding all prime numbers in a given range. This method includes sorting the multiples of each prime numbers and gradually marked them as composite and list the prime n numbers starting from '2' increasingly.

Sieve of Sundaram was proposed by S.P Sundaram. According to his theory a set of numbers starting from 1 to certain range removes all numbers that are in the form of **i+j+2ij** and sort them as composite.

A.O.L Atkin and Daniel J.Berstein created the Sieve of Atkin. This algorithm use binary quadratic forms to sort the primes[4].

Sieve of Sorenson uses number of approach to sort the primes like first it use wheel sieve to remove all multiples of small primes, then by using pseudosquares it eliminates non prime, and finally it removes composite prime powers with another sieve.

2. OBJECTIVE

2.1 Integration of Memoization in MPI application

Memoization is an optimization technique used to optimize the execution speed of computer programs[13].

The memoization technique can be explained by storing the results of expensive function calls and returning the intermediate result, when the same parameters parsed again.

If memoization introduced in MPI application the function needs to be chosen in which it could be applied.

Therefore the functions which have the following properties are considered for MPI application:

1. Pure function

2. Function that obeys the applications scope

3. Recursive function

4. The functions which receives key as an input value

Pure functions are similar to hash functions which returns the same results with the same input parameters. The second categories of functions includes I/O routines which uses the unaltered resources beyond the scope of the application. The third category of function are recursive function which are apparently involved many times during the execution. The last type of function are proposed in MPI_Info[3] object which will return the output value faster than other function because it receives key as an input value.

2.2 Parallelism in Sieve of Prime Number

Parallelism refers to the techniques of deployment of parallel thread to make programs faster [1]. The program is executed concurrently by multiple entities known as thread and process instead of splitting in two parts.

There are three types of parallelism technique has been proposed.

1. Shared Memory Parallelism.

2. Distributed Memory Parallelism

3. Hybrid parallelism

In shared memory parallelism as the name suggest threads share a memory space among them.

In distributed memory parallelism each processes keep their private memories distinct from memory of other process.When one process retrives data from memory of other process,the data communication takes place which is known as "Message Passing". The MPI(Message Passing Interface) is a standard of this type of parallelism. It defines a set of function that can be used in C, C++ Format for passing message[5].

The hybrid parallelism uses both shared and distributed memory technique. In this parallelism the problem is subdivided in to small chunks of problem which are processed through parallel threads. After the execution of threads, their result uses the shared memory to accumulate the results from thread. Note that the message passing is used by the threads to gather result from other threads.

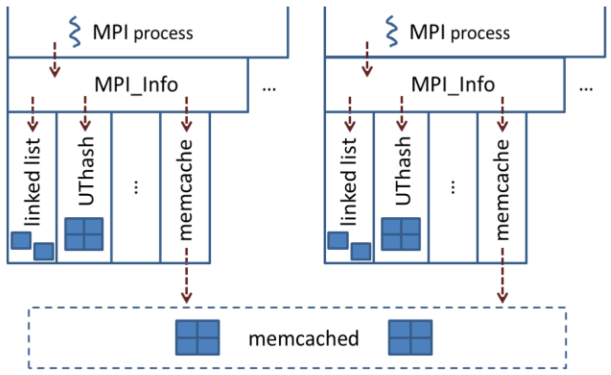


Figure-2(Demonstrates the shared and distributed memory architecture.)[3]

ANALYSIS OF PARALLEL SIEVE ALGORITHM

1. Create a list of natural numbers 2, 3, 4, 5, 6, ……,n. None of which is marked. Each process creates its share of lists.

2. Set k to 2, the first unmarked number on the list. Each process does this.

3.while ($K^2 < n$)

a. Mark all numbers which are divisible by k.

b. Set k = smallest number which is divisible by k within the range of k to n^2

c. Process 0 broadcasts k to rest of process.

4. The numbers which are unmarked are primes.

5.determine number of primes.

2.2.1 Parallel Hardware

Hardware must have the capability to support parallelism. As there is a single CPU connected to memory in case of classic model of computer established by Jhon Von Newmann does not support parallelism[5]. There must be multiple processing threads running multiple streams of instruction, in order to support parallelism.

Multi-core technology splits the CPUs into multiple units, called as cores which allows parallelism. This model is suitable for shared memory parallelism because the cores will often share main memory.

Multiple computers are connected by a network also satisfy parallelism. This required distributed parallelism because each computer has its own main memory. Computers may have multi-core processors that promots hybrid parallelism.

2.2.2 Motivation for Parallelism

The advantages of parallelism are speedup, accuracy and weak scaling.

The speedup promotes speed of execution, through which a program will run faster if it is executed parallelly. The main advantages of speedup is to allow a problem to run faster. If multiple threads executed simultaneously.

There is an equation represented below according to Amdahl's Law:-

$$\textbf{Speedup} = 1/\,(1\text{-}N)+P\,/\,P\,/\,N \qquad (1)$$

Where P = the time it takes to execute the parallel regions

1-P = the time takes to execute the serial regions

N = the number of processors

Accuracy promotes the correctness of parallel execution. When multiple threads are assigned to a problem, times may be wasted by them in doing error checks or other forms of diagnostics in order to confirm that the result is accurate or better approximation of the problem that is being solved. Speedup may be sacrificed to make a program more accurate.

Weak scaling promotes to utilize more threads to solve a bigger problem in the same amount of time it takes least thread to solve a smaller problem.

Some advantages of parallelism are communication overhead and the other one is specify as Amdahl's law.

"Communication overhead" defines as the lost time that is waiting for communications between calculations to take place . No development noticed on executing the algorithm , but there may some important data is being communicated in this time. A program's communication overhead can instantaneously engulf the total time spent to solve the problem, and occasionally to making the program less efficient than its serial counterpart. Thus communication overhead can be a disadvantage of parallelism.

Gene Amdahal proposed Amdahal's law. It defines as the speedup of a parallel program will be limited by its continuous regions or the sections of algorithm that can't be executed parallelly [5]. Amdahl's law hypothesize that as the number of processors allotted to the program and the advantages of parallelism is inversly proportional to each other as the continuous regions become the only part of code that take symbolic time to execute.

Amdahl's Law says as the number of processor is increased, the program will have negligible returns. But it does not place a limitation on weak scaling that can be achieved by the program as it may allow huge class of problem to be solved as a number of processor become available.

Gustafson's Law proposed by John Gustafson describes the advantages of parallelism weak scaling, which says if the processor count is increased the bigger problem can be solved in the same amount of time as smaller problem.

2.2.3 Algorithm for Parallelism

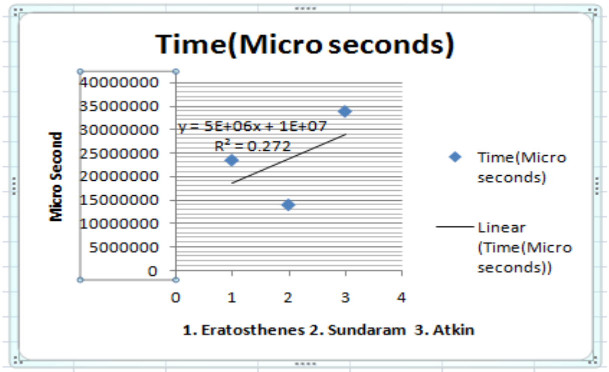
In a parallel algorithm, by using shared memory, multiple threads work continuously. It is not enough to have only one thread executing task in a order, so it is necessary to identified the procedures which have a ability take place at e same time.

Let us look at the Algorithm:-

1. 2 to 15 number should be written.
2. The smallest unmarked should circled and uncircled the number in the list.
3. If the bigger circled number is a multiple of biggest circled number then marked it for is bigger number than the biggest circled number.

4. Step 2-4 should be repeated until all numbers have been visited. The primes and the composites numbers are the circled and marked successively.

3. RESULTS



4. CONCLUSIONS

After the execution of three sieve algorithms, we found that "Sieve of Sundaram" gives a optimal result in a range of 10000.For a large range of 100000 Sieve of Sundaram has crashed, and other two runs smoothly. Sieve of Eratosthenes behaving like linear type algorithm. However Atkin is the modern one. To solve this we have introduced MPI.

This work shows that the sieving algorithm, for prime number document can be improved by integrity the MPI_Info object along with memoization concept. In this work we have analyzed three different sieve algorithm for prime number such as Eratosthenes, Sundaram and Atkin. Surprisingly we found that the performance of the algorithm sieve of Sundaram is enhanced as compared to other two, when the integration do happens.

Our future work includes the object of MPI_Info could be mapped to the NoSQL database because NoSQL is based on key value pairs.

5. ACKNOWLEDGMENTS

This work is supported by seed Grant Utkal University

6. REFERENCES

[1] Aaron Weeden. (n.d.). *Parallelization: Sieve of Eratothenes*.

[2] Agosta,Francalanci. (2012). Automatic memoization for energy efficiency in financial applications. *Sustainable Computing Informatics and Systems*, 105–115.

[3] Alejandro Calendron, Felix Garcia, D. H., Jesus Carretero. (2013). Improving MPI application with a new MPI_Info and the uses of the memoization. *EuroMPI*.

[4] D. Bernstein, A. atkin. (2004). Prime Sieves Using Binary Quadratic Forms. *Mathematics of Computation*, 1023–1030.

[5] David J Wirian. (n.d.). *Parallel Prime Sieve: Finding Prime Numbers*.

[6] Graham,J.M. (2005). *Open MPI:A flexible high performance MPI*.

[7] J. Misra, D. G. (1978). A Linear Sieve Algorithm for Finding Prime Numbers. *Communications of the ACM*, 999–1003.

[8] Jeffrey M. Squyres. (1982). Design and Implementation of java bindings in Open MPI. *Acta Informatica*, 477–485.

[9] Jeffrey M.Squyres,Andrew Lumsdaine. (n.d.). *Object Oriented MPI:A Class Liabrary for the Message Passing Interface*.

[10] J.P Sorenson. (2006). The Pseudosquares Prime Sieve. *Algorithmic Number Theory*, 193–207.

[11] Mark Baker,Sang Lim. (n.d.). *Mpi Java:An Object-Oriented Java interface to MPI*.

[12] Michael A Bender, S. S. (2013). *The I/O Complexity of Computing Prime Tables*.

[13] Mirko Stoffers,Klaus Wehrle. (2006). *Automated Memoization for Parameter Studies implemented in impure Languages*.

[14] *MPICH2 http://www.mpich.org/*. (2013).

[15] N.Saxena,N.kayal, M. A. (2004). Primes is in P. *Annals of Mathematics*, 781–793.

[16] *Open MPI.http://www.open-mpi .org/*. (2013).

[17] P.Pritchard. (1982). Explaning the wheel sieve. *Acta Informatica*, 477–485.

[18] Umat A.Acar,Robert Harper. (2003). *Selective Memoization.*