Parallelization of The Sieve of Eratosthenes using PVM in Linux Environment

Presented by
Sandipan Ghosh
Arnab Roy
Debashis Mandal

Under the supervision of

Prof. Utpal Ray

Overview:

- Introduction to parallel processing
  - ❑ Objective of parallel processing
  - ❑ How does it works
  - ❑ How do the processors interact

- PVM (Parallel Virtual Machine)
  - What is PVM
  - PVM system & its features

- Sieve of Eratosthenes : A prime finding algorithm
  - oSequential algorithm
  - oParallel algorithm development & analysis
  - oPVM program & optimization

Introduction to Parallel Processing

- There are three ways to do anything faster :

i. Work harder
ii. Work smarter
iii. Get help

- In computers

i. *Work harder* => increase the processor speed
ii. *Work smarter* => use a better algorithm
iii. *Get help* => use parallel processing

- So parallel processing involves :

i. Breaking up the task into smaller tasks
ii. Assigning the smaller tasks to multiple workers to work on simultaneously
iii. Coordinating the workers

And Parallel Processing leads to Parallel Programming…

- Parallel programming involves:
  - Decomposing an algorithm or data into parts
  - Distributing the parts as tasks which are worked on by multiple processors simultaneously
  - Coordinating work and communications of those processors
- Parallel programming considerations:
  - Type of parallel architecture being used
  - Type of processor communications used
-

Processing Elements Architecture

- Simple classification by Flynn:

  (No. of instruction and data streams)

  > SISD - conventional

  > SIMD - data parallel, vector computing

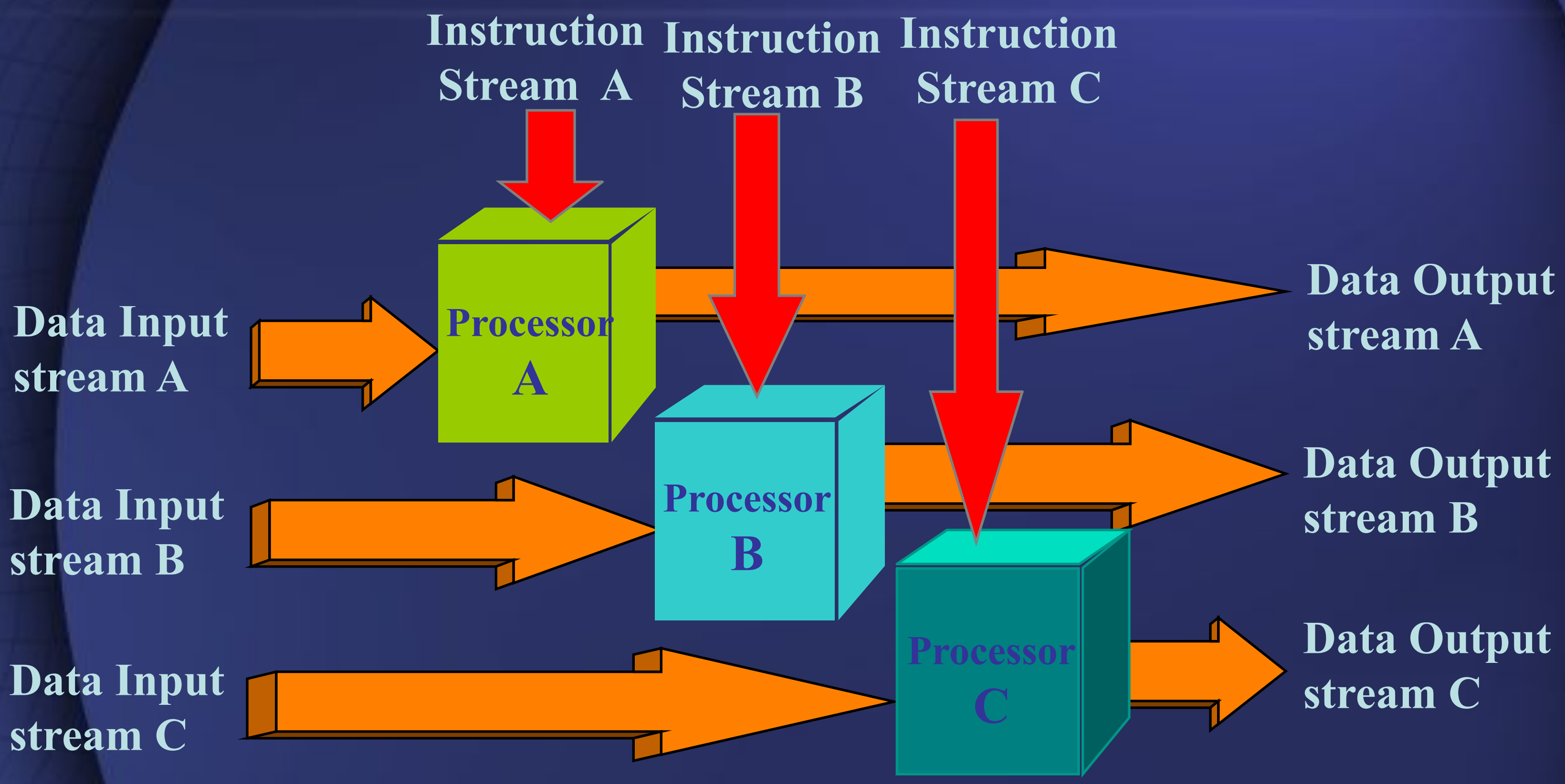
  > MISD - systolic arrays

  > MIMD - very general, multiple approaches.

- Current focus is on MIMD model, using general purpose processors.

  (No shared memory)

MIMD Architecture



Unlike SISD, MISD, MIMD computer works asynchronously.

Shared memory (tightly coupled) MIMD

Distributed memory (loosely coupled) MIMD

How do the processors interact

- The way processors communicate is dependent upon memory architecture, which, in turn, will affect how you write your parallel program

- The primary memory architectures are:

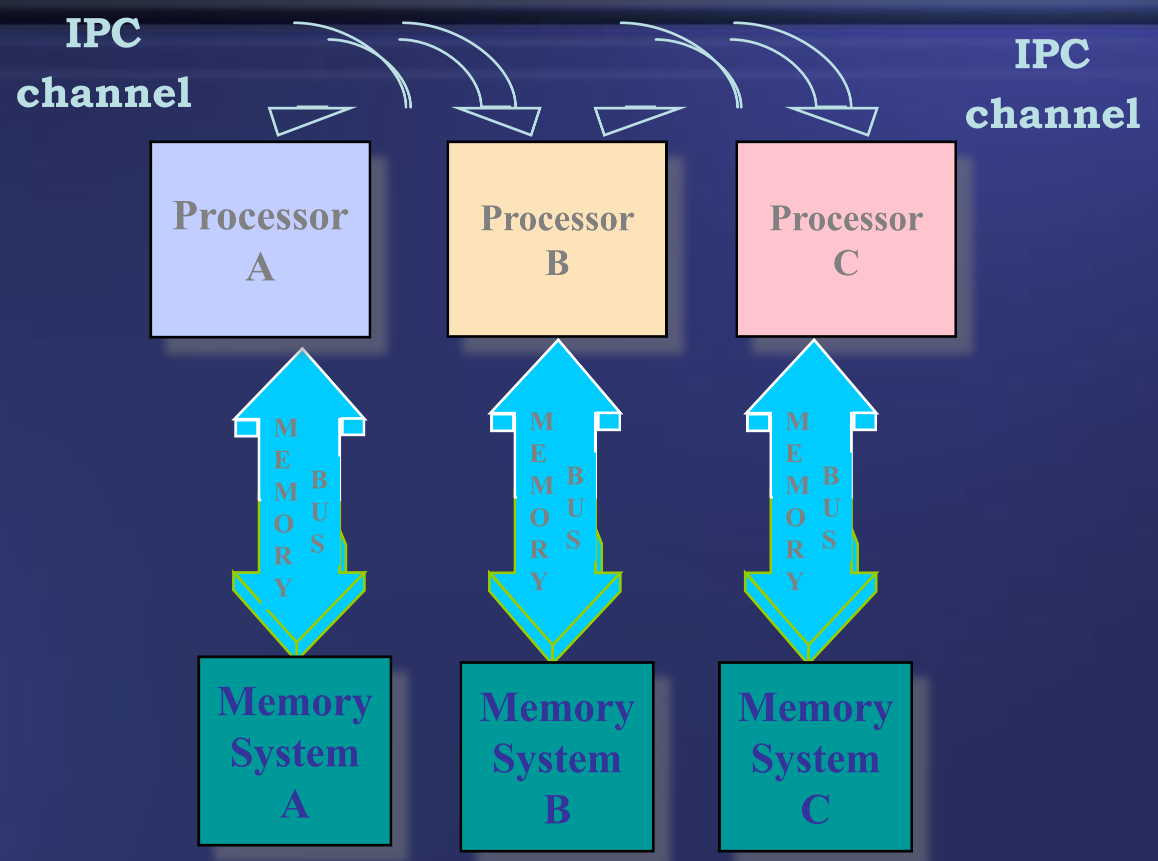
✔ Shared Memory

✔ Distributed Memory

✔ Hybrid Distributed-Shared Memory

And we are interested in Distributed Memory which is described next…..

Distributed Memory Architecture



- Many processors, each with local memory that are only accessible only to it, are connected via an intercommunication network
- They communicates by passing message to each other.
- Network can be configured to –Tree, Mesh,Cube etc.

Parallel programming

- We have to design code and application as a set of cooperating processes.
- To obtain parallelism using the concept of distributed memory system we have two message passing model: PVM & MPI

- We have developed our parallel code using PVM. So, now we will find out how PVM is used for message passing

PVM
(Parallel Virtual Machine)

- Provides the software environment that makes a cluster appear like a single large computer.
- Enables existing computer hardware to solve much larger problems at minimal additional cost.
- Handles all message routing, data conversion, and task scheduling across a network.
- Supports MIMD & SPMD architecture.

PVM System

PVM System is composed of two parts:

- **Daemon**

  Resides on all the computers making up the virtual machine.

- **Library of PVM interface routines**

  contains user-callable routines for message passing, spawning processes, coordinating tasks, and modifying the virtual machine .

The Sieve of Eratosthenes

Eratosthenes(276-194 B.C.)

- Classic Prime Finding Algorithm
- find the number of primes less than or equal to some positive integer *n*.
- begins with a list of natural numbers 2, 3, 4, . . . , n, and removes, composite numbers from the list by striking multiples of 2, 3, 5, and successive primes.
- sieve terminates after multiples of the largest prime less than or equal to $\sqrt{n}$ have been struck.

Sieve of Eratosthenes Sequential Algorithm

Execution Process

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	28	29	30	31
32	33	34	35	36	37	38	39	40	41	42	43	44	45	46
47	48	49	50	51	52	53	54	55	56	57	58	59	60	61

Parallel Approach to implement the Sieve of Eratosthenes

Outline

- Sources of parallelism
- Data decomposition options
- Parallel algorithm development, analysis
- Implementing parallel algorithm using PVM

Sources of Parallelism

- **Domain decomposition**
  - Divide data into pieces
  - Associate computational steps with data
- **One primitive task per array element**

Data Decomposition Options

- Want to balance workload when n not a multiple of p
- Each process except last one gets $\lfloor n/p \rfloor$ elements
- Last process gets $\lfloor n/p \rfloor$ +(n%p)

- There are two types of processes:
  - Master – finds new prime & sends to slaves
  - Slaves - receives prime from master & marks its multiples

Decomposition Affects Implementation

- Largest prime used to sieve is $\sqrt{n}$
- Master process has $\lfloor n/p \rfloor$ elements
- It has all sieving primes if $p << \sqrt{n}$
- Master process always broadcasts next sieving prime to slaves
- After slave marked its portion, sends it to master
- Master reassembles the marked array

Parallel Algorithm Development

1. Create list of unmarked natural numbers 2, 3, …, n

Each process creates its share of list

2. $k \leftarrow 2$

Each process does this

3. Repeat

Each process marks its share of list

  (a) Mark all multiples of k between k^2 and n

  (b) $k \leftarrow$ smallest unmarked number $> k$

Master process only

  (c) Master broadcasts k to rest of processes

until $k^2 > n$

4. The unmarked numbers are primes

A snap shot of PVM coding

Spawning of task:

```
for(i=0;i<MAXTASK;i++){
numt=pvm_spawn(SLAVENAME, (char**)0, 0, "", 1, &tids[i]);
printf("SPAWNED TASK NO.=%x\n",tids[i]);
}
```

Data sending process:

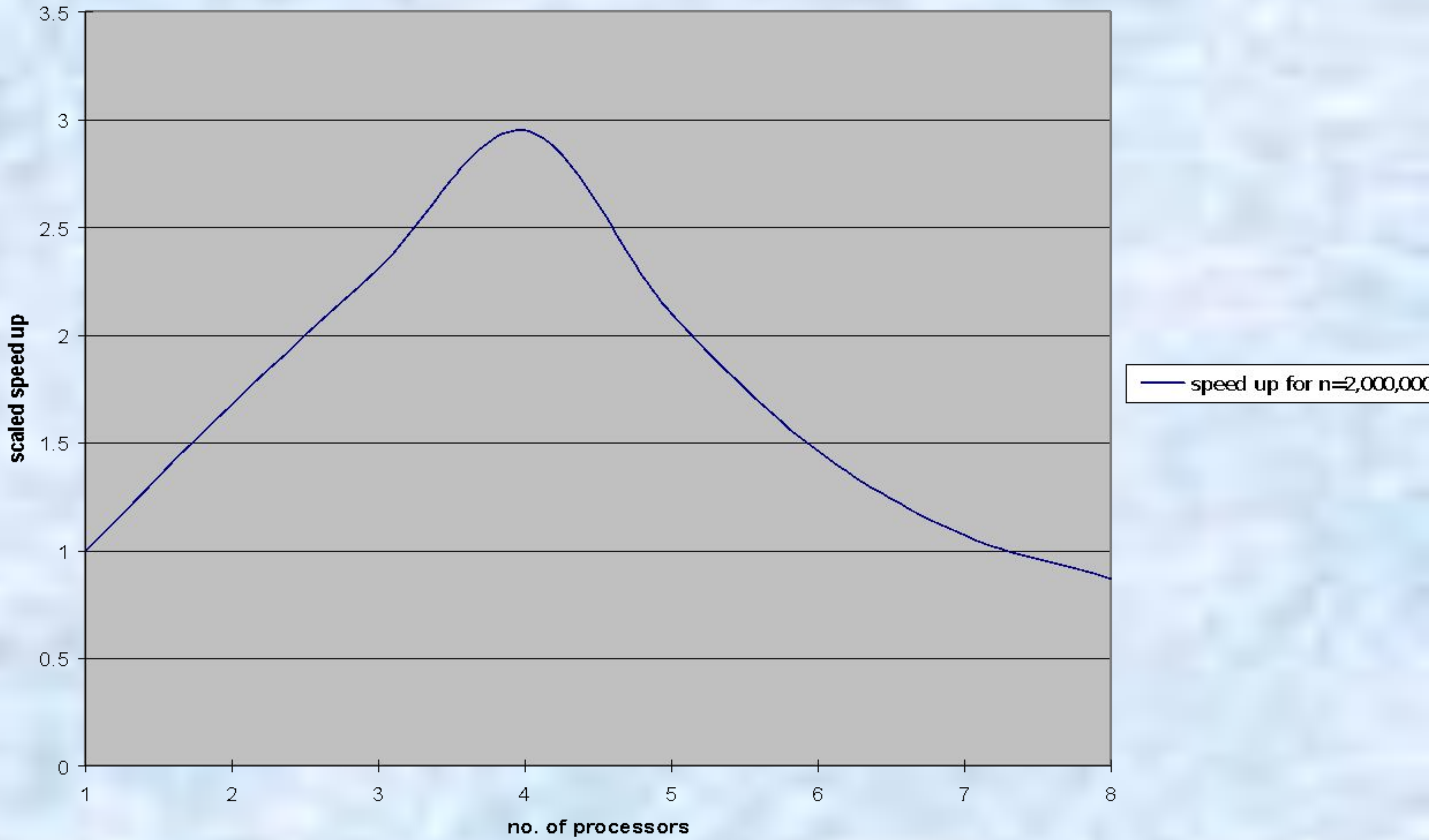
```
msgtype=5;
for(i=0,i<MAXTASK;i++)
{
snd[0]=(n/p)*(i+1);
if(i==(MAXTASK-1))
snd[1]=n-(n/p)*(i+1);
else
snd[1]=n/p;
snd[2]=n;
snd[3]=MAXTASK+1;
pvm_initsend(PvmDataDefault); // Get ready to send message buffer
pvm_pkint(snd,4,1);         // Send Loop Value
pvm_send(tids[i], msgtype);}  // Send to identified processor
```

The PVM Program (cont'd)

```
current_prime=3;
srt=sqrt(n);
msgtype=7;
while(current_prime<=srt && current_prime<=siz)
{
j=current_prime;
//------sending current prime to all slaves--------
pvm_initsend(PvmDataDefault);
pvm_pkint(&j,1,1);
pvm_mcast(tids,nproc,msgtype);
temp=pow(current_prime,2);
while(temp<siz)
{
if((temp%2)==1){
temp1=(temp/2)+1;
mark[temp1]=1;}
temp+=j;
}
//------finding new prime after marking ---------
temp11=(j/2)+1;
while(mark[++temp11]!=0);
current_prime=arr[temp11];
pvm_barrier("sieve",MAXTASK+1);//for sync. Of calculation
```

- }

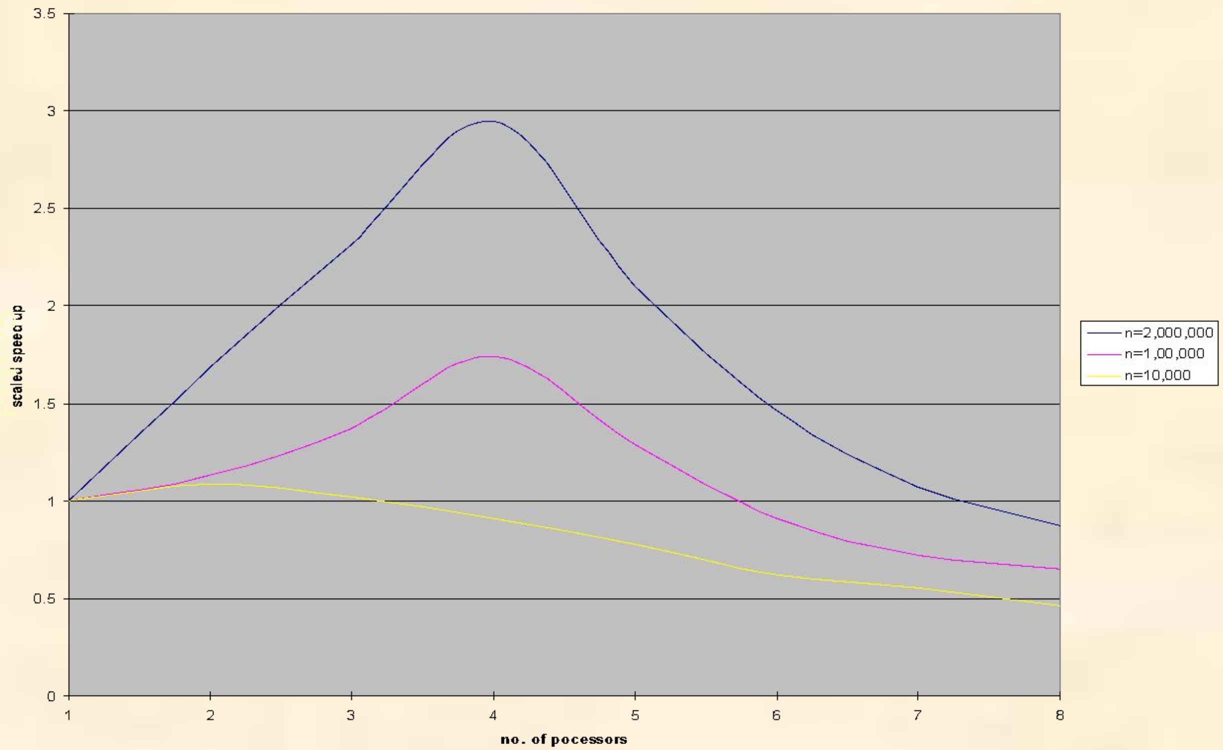
Results



Scaled speed up for n=2,000,000

- We achieve maximum speed up of 2.95 when 4 processors are used.for problem size n=2,000,000
- Thereafter, speed up decreases significantly with in no. of processors due to communication overhead.

Results (cont'd)



Amdahl Effect

- ✔**Amdahl Effect specifies that for a parallel program, with increase in problem size, speed up also increases.**
- ✔**We can prove Amdahl effect by calculating speed up for various problem size**

Conclusion

- In this project, we successfully used PVM to implement a parallel version of the Sieve of Eratosthenes to test various parallel computing system metrics like scaled speed up, problem size, execution time etc.

- It has been noticed that for a given problem size, there is a significant time difference between the execution time of a sequential code and that of a parallel code running on a single processor which is beyond our theoretical explanation.

- Moreover, the speed up calculated from our parallel code is not up to the expected linear speed up. Our explanations behind this discrepancy are:

✔ The upper bound of the problem can not be increased after a certain range for the provided infrastructure.

✔ The communication overhead, cost of I/O operation, synchronization between processes, creating processes

✔ The algo needs a little bit of CPU execution for marking.

Last but not the least

- From these points we can conclude that for the maximum problem size provided by our multicomputer infrastructure, this algorithm is partly suitable. Rather it can be better implemented in the multiprocessor environment to achieve better speed up where communication overhead is significantly low.

Thank You!!