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

Parallel Program for Sorting NXN Matrix Using PVM (Parallel Virtual Machine)

Ehab AbdulRazak Al-Asadi
College of Science –Kerbala University, Iraq

Abstract
The study will focus for analysis the possibilities of implementing sequential as well as parallel algorithms for sorting an array N x N. Design and implement (in C / C ++) program, respectively. Programs for sequential algorithm and bodies solving based on a transmission of messages between nodes by using a system PVM libraries. Output (STDOUT) for both programs must be »identical software. Reduce the time Between nodes so that the calculation time as small as possible. Find out what's dependence and acceleration of the implementation period for calculating the number of nodes and the size of the tasks in relation to sequencing program, (insert The table and charts). Based on the results estimate: latency communication, for what is the role of the dimension of the architecture. Discuss about what the benefits, effectiveness of parallel program of various algorithms. it requires for solving, use the input and output files with nuts row matrix corresponding file line and column values are separated by spaces or tabs. Unless otherwise stated, work with real numbers.

Keywords
Shear sort,PVM, MPI, Parallel Program.

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.

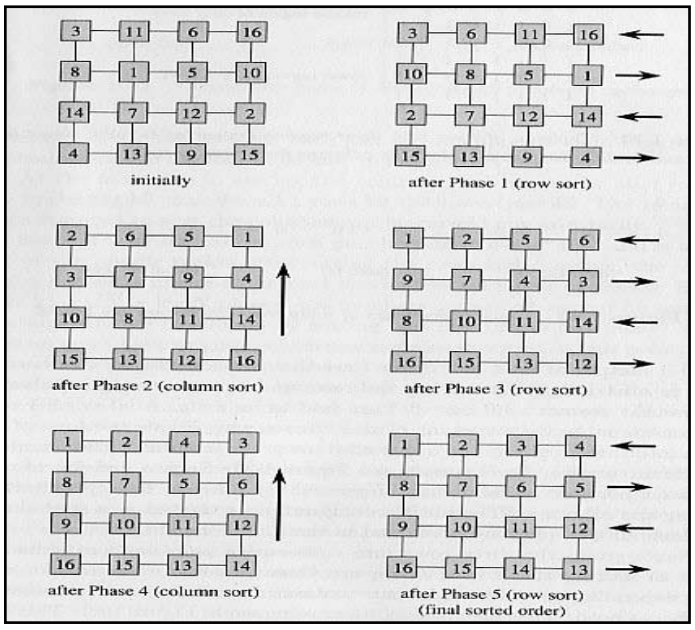
PVM is particularly effective for heterogeneous applications that exploit specific strengths of individual machines on a network. As a loosely coupled concurrent supercomputer environment, PVM is a viable scientific computing platform. The PVM system has been used for applications such as molecular dynamics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

A. Shear Sort Algorithm

Classical algorithm to organize a two-dimensional array. The basic idea is to redistribute data in rows and columns, and is allowed to hold them in parallel, This sorting algorithm consists of row and column sorting phases.

In row sorting phase, each row is sorted so that even numbered rows have the largest number to right, and odd numbered rows have the largest number to the left.

In column sorting phase, each column is sorted so that the smallest numbers appearing at the top of columns. After ceiling($\log_2 n$) + 1 phases, where n is the number of data, the data becomes sorted



Shear Sort Algorithm .Fig(1)

B. Solving the problem

The problem can be solve by appropriate algorithm for arranging the field of size NxN. In the case of the use of parallelism I have chosen the layout using the method Shear sort. This allows you to split the field into rows and columns. You then followed parallel organize according to the following scheme

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

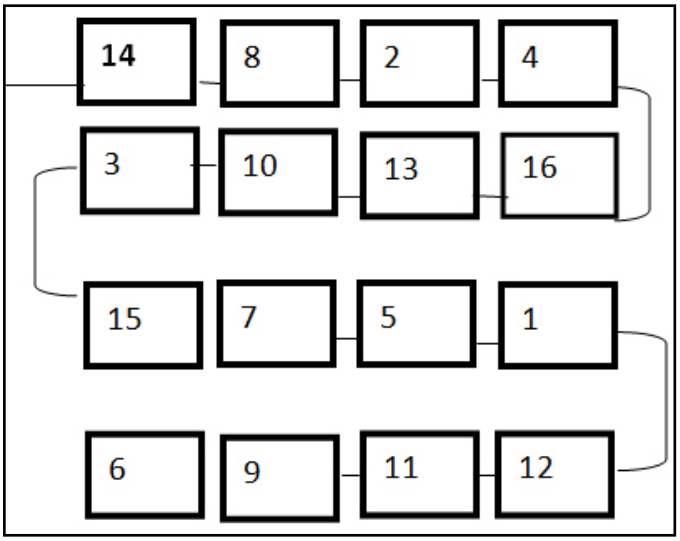


Fig. 2 : Shear Sort Mechanism

1. Lines described as odd and even.
2. The odd lines to arrange ascending, descending even lines.
3. Arrange Columns, each ascending.
4. Arrange all the rows in ascending order.
5. End

From the principle of the algorithm is to see the possibility of application within a distributed system. Multiple processors can perform configuration rows. Next, in step no. 3 it is possible to organize individual columns simultaneously. Finally, again lines (step 4). The case for reallocating tasks of organizing lines for individual processors is indicated in the following figure(3)

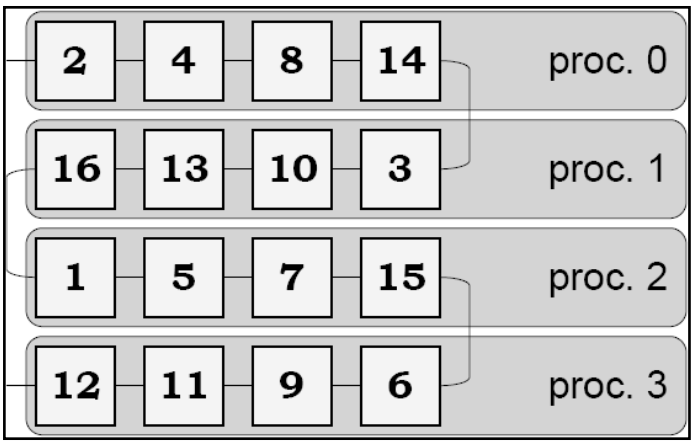


Fig. 3 : Shear Sort Implementation

Number of processors for an array of size n numbers is √n. According to the assignment we consider a two-dimensional array of size NxN, namely the number of processors is N.

II. The Design

The basic element of the arrangement in ascending or descending consists in comparing the two adjacent cells that form a pair. If necessary, according to sequence or digressively to be replaced. Taking into account the indexation of the field beginning at zero, the first will be matched pairs 0 1, 3, 4, 5, 6, ..., then pairs 2-3, 4-5, 6-7, ... The first type of pairs will be compared and confused one function, a second pair of second. Since these functions must be universally applicable even in rows and columns will be better for easy organization to give a two dimensional array to a one-dimensional load gradually by line. This makes it easier to handle the array index that will serve as parameters for these features that are confusing elements.

A. The prerequisites for entry and exit

By accessing the algorithm will be one-dimensional array, which will overwrite the original array NxN behind by line. Before the algorithm will be necessary to implement the procedure of loading elements from the input file(STDIN). The prerequisites for entry will be error-free file format which will hold numbers separated by a space .Pre maintaining a transparent matrix output will be written to the file in the shape where adjacent elements in the columns are separated by a space adjacent rows separated by a line feed ('\ n').

B. The program environment

Application I programmed in C language library for the PVM environment. The system itself consists of PVM daemon-and running on each node. In this connection the node as the next work station in the laboratory that provides the processing power of the size of one processor. PVM is called. message passing system in which parallel tasks to synchronize and exchange information by sending messages.

C. Bernstein's conditions

Let Pi and Pj be two program segments. Bernstein's conditions describe when the two are independent and can be executed in parallel. For *Pi*, let *Ii* be all of the input variables and *Oi* the output variables, and likewise for *Pj*. *P i* and *Pj* are independent if they satisfy.

Ij ∩ Oi= □
Ii ∩ Oj = □
Oj ∩ Oi = □

D. Research Objectives Goal

Utilize the Hardware, System, & Application Software to either **Speedup** = (Execution Time on 1 Single Processor)/ (Execution Time on N processors)

Speedup=T/Tn

Efficiency = Speedup / N ≅1 *ideal*
Solve problems requiring a large amount of memory.

III. The Implementation

Alternate row and column sorted until list fully sorted. Row sorting in alternative directions to get snake-like sorting:

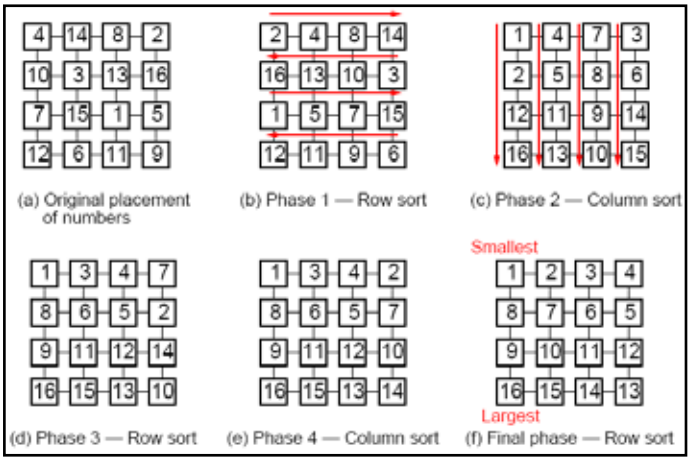


Fig. 4 : the Shear Sort flow

A. Rank Sort

Number of numbers that are smaller than each selected number is counted. This count provides the position of selected number in sorted list; that is, its "rank."

- First number a[0] is read and compared with each of the other numbers, a[1] … a[n-1], recording the number of numbers less than a[0].
- Suppose this number is *x*. This is the index of the location in the final sorted list. The number a[0] is copied into the final sorted list b[0] … b[n-1], at location b[x].
- Actions repeated with the other numbers.

Overall sequential sorting time complexity of $O(n^2)$ (not exactly a good sequential sorting algorithm

B. Sequential Code

```
for (i = 0; i < n; i++) {     /* for each number */
        x = 0;
        for (j = 0; j < n; j++)     /* count number less than it
*/
                if (a[i] > a[j]) x++;
        b[x] = a[i];                    /* copy number into
correct place */
```

C. Parallel Code Using n Processors

One processor allocated to each number. Finds final index in $O(n)$ steps. With all processors operating in parallel, parallel time complexity $O(n)$ with *n* processors.

In forall notation, code would look like

```
        forall (i = 0; i < n; i++) {   /* for each no in parallel*/
          x = 0;
          for (j = 0; j < n; j++)      /* count number less than it
*/
                if (a[i] > a[j]) x++;
          b[x] = a[i];                    /* copy no into correct
place */
        }
```

D. PVM Overviews

The PVM software provides a unified framework within which parallel programs can be developed in an efficient and straightforward manner using existing hardware. PVM enables a collection of heterogeneous computer systems to be viewed as a single parallel virtual machine. PVM transparently handles all message routing, data conversion, and task scheduling across a network of incompatible computer architectures. The PVM computing model is simple yet very general, and accommodates a wide variety of application program structures. The programming interface is deliberately straightforward, thus permitting simple program structures to be implemented in an intuitive manner. The user writes his application as a collection of cooperating tasks. Tasks access PVM resources through a library of standard interface routines. These routines allow the initiation and termination of tasks across the network as well as communication and synchronization between tasks.

E. MPI(Message Passing Interface)

The Message Passing Interface (MPI), standard, whose speciation was completed in April 1994, is the outcome of a community effort to try to dene both the syntax, and semantics of a core of message-passing library routines that would be useful to a wide range of users and efficiently implementable on a wide range of MPPs. The main advantage of establishing a message-passing standard is portability. One of the goals of developing MPI is to provide MPP vendors with a clearly define base set of routines that they can implement efficiently or, in some cases, provide hardware support for, thereby enhancing scalability

F. The Master –Slave Model (PVM)

- PVM is not restricted to this model
- Useful programming paradigm and simple to illustrate
- The master calls pvm_mytid() to

1. Allow it to use the PVM system, and
2. Enable interprocessor communications

- It then call pvm_spawn() to execute a given number of slave programs on other machines
- Each slave program must also call pvm_mytid() to enable processor communications
- Subsequently, pvm_send() and pvm_recv() are used to pass messages between processes
- When finished, all the PVM programs should call pvm_ exit() to

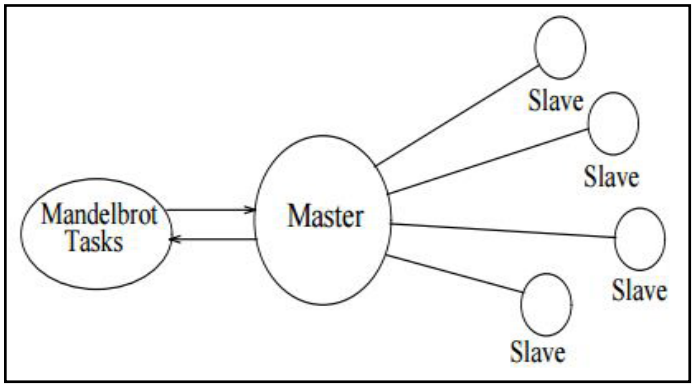


Fig. 5 : Master –Slave paradigm

G. MPI (Message Passing Interface)

MPI (Message Passing Interface) is specification for message-passing libraries that can be used for writing portable parallel programs.

When we speak about parallel programming using MPI, we imply that:

- A fixed set of processes is created at program initialization; one process is created per processor
- Each process knows its personal number

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

- Each process knows number of all processes
- Each process can communicate with other processes

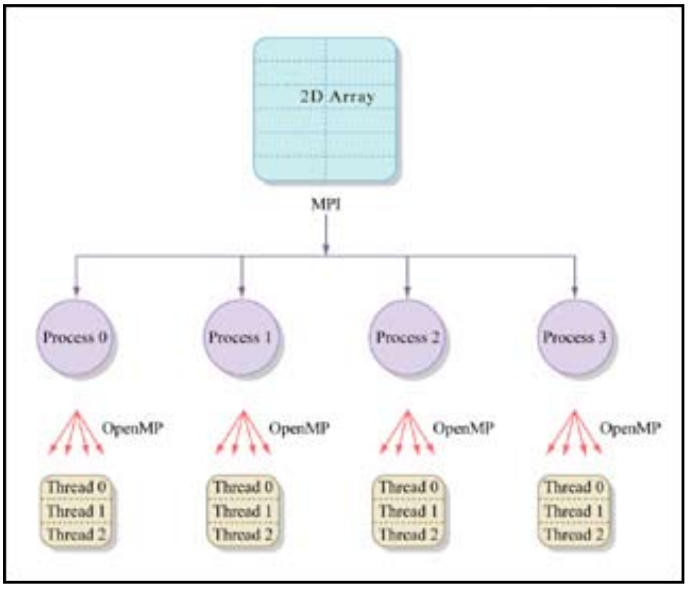


Fig. 6 : Using MPI In Parallel Program

The Minimal MPI Program in (C) include the MPI.h Library

A Minimal MPI Program (C)

```
#include "mpi.h"
#include <stdio.h>

int main( int argc, char *argv[] )
{
    MPI_Init( &argc, &argv );
    printf( "Hello, world!\n" );
    MPI_Finalize();
    return 0;
}
```

Fig. 6 : MPI library included

H. MPI_SEND

Sending messages via: MPI:COMM_WORLD.Send( ) (C++)
Full C++ syntax: int MPI::COMM_WORLD.Send (void* message, int count, MPI_Datatype datatype, int destination, int tag

I. MPI_RECIEVE

Receiving messages via: MPI::COMM_WOLRD.Recv( ) (C++)
int MPI::COMM_WORLD.Recv(void* message, int count, MPI_Datatype datatype, int source, int tag, MPI_Status* status)

IV. Results

After running the program, will use the real time to calculate the speedup because the real time represent the time from the beginning to end of program.

Table (1) Measurement of System Speedup at N process

Number of Process (N)	1	5	10	20	50
No of repeat	Real Time	Real Time	Real Time	Real Time	Real Time
1	4.164	2.137	8.277	4.999	3.72
2	2.404	3.750	6.393	3.241	8.937
3	8.709	4.835	4.140	3.475	1.749
4	4.074	6.629	2.629	8.625	1.969
5	1.633	7.386	12.678	4.896	3.378
Average	4.956	4.1968	6.8234	5.0472	3.9506

SpeedUp=S= T1/Tn When T1(Time at 1 Process),Tn (Time at n process)
S(T5)= execution time at(5) process =4.1968
So the S(T5) =T(5)/T(1)= 4.1968/4.956=0.846 (Speedup)

Efficiency of system =Speedup/N= 0.846 /5=0.1693 ,If efficiency =~ 1 ,the system is Ideal System .

Number of Process Time

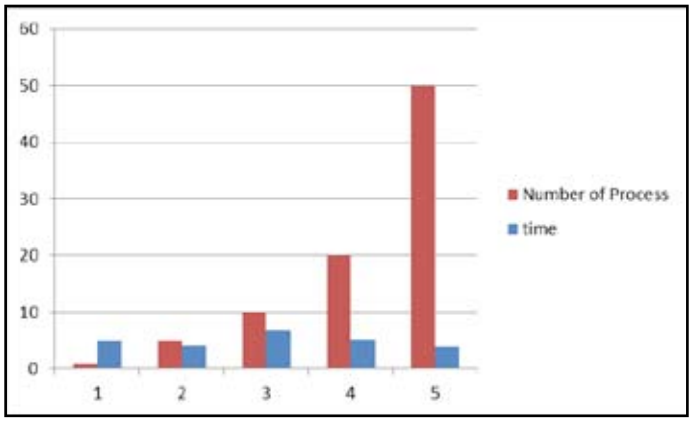


Fig. 7 : Relation between the Number of Process(N) Created and the execution time

According to **Fig (7)** we can see the relation between Increase the Number of Process created and decrease the (Time execution)

and this one of the benefits of using PVM and MPI In system gain the system faster.

Number of Process Speedup

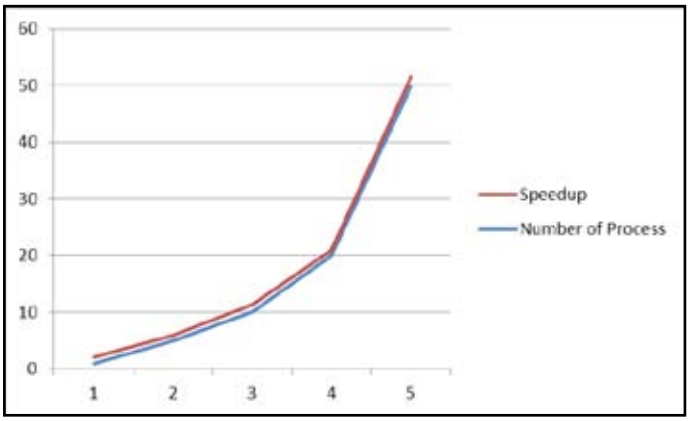


Fig. 8 : Speedup Increased with The Number of process created Number of Process

Speedup

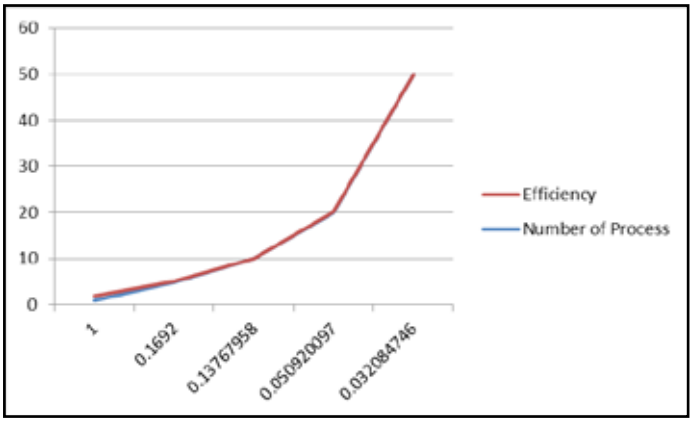


Fig. 9 : The system Efficiency

V. Conclusion

MPI provides a convenient standard to implement parallel programs on distributed memory architectures. The programmer has to take care of the parallelization. A thorough analysis of the algorithms and data structures is important. It is difficult to change an existing serial program into a parallel version with MPI

The effect of parallel processes number and also the number of cores on the performance of parallel, the execution times for both of them increase as the number of processes exceeds the number of cores.

Although MPI is becoming widely used, another message-passing system called PVM (Parallel Virtual Machine) , is even more common. PVM, which is essentially the only competitor of MPI, Porting the proposed system so that it supports PVM programs as well would make it more widely accepted.

The **execution time** decreased with Increase the number of Process created by system (PVM) as shown in **fig (7)** mentioned above.

The Speedup of system computing Increased with number of process created in system also so its big gain for system when speedup increased.

References

[1] *Message-Passing Interface Forum. 1994. ``MPI: A Message-Passing Interface Standard.'' The International Journal of Supercomputer Applications and High Performance Computing, Vol. 8, No. 3/4.*

[2] *S. G. Akl, ``The Design and Analysis of Parallel Algorithms,'' PrenticeHall, Englewood Cliffs, NJ, 1989.*

[3] *Hesham El-Rewini ,"Advance computer architecture and parallel processing"John Wiley & Sons ,NJ ,2005.*

[4] *Puneet C Kataria, Parallel quicksort implementation using MPI and Pthreads,2002*

[5] *Jordan, H. F., Jordan, H. E. Fundamentals of Parallel Computing, Prentice Hall, 2002*

[6] *Gropp, W. et al, The Sourcebook of Parallel Computing, Morgan Kaufmann, 2002*

[7] *Joseph, J., Fellenstein, C., Grid Computing, Prentice Hall, 2003*