# ENHANCING INTRA-CLUSTER COMMUNICATION FOR MULTITASKING

[1]Jhonny Panchal,[2]A. Sai Swaroop,[3]Sneha K
[1.2.3]Students
[1]Computer Science Engineering,
[1]REVA UNIVERSITY,Bangalore,India

***Abstract:*** Presently, only a single task is been executed on a parallel system. In proposed system, the communication is going to be enhancing to support the multiple task execution in parallel system. To implement multitasking support semaphore concept has to be used to share the communication devices and protocol. The process which holds the semaphore will use the communication device and after finishing the communication the process releases the semaphore. If the semaphore is locked and any process wants to use communication device or protocol has to wait until semaphore is release by the other process. To control the flow of execution setjump and longjump has been used and with the above the multitasking has been implemented with send and receive protocol.

## 1. INTRODUCTION

Parallel processing is the use of multiple processors working simultaneously on one task. The need for fast computation have been a number of contexts involving partial differential equation to solve computational fluid dynamics problems, weather prediction model, image processing application etc. Such application involve large number of numerical computations.

The term parallel processing refers to a large class of methods attempt to increase computing speed by performing more than once computation concurrently. Techniques employed to speed up computers by performing many operations simultaneously or in parallel form the subject of parallel processing. A parallel processor is a computer that implements some of the parallel processing techniques. Thus parallel processing is an efficient form of information processing which emphasizes there exploitation of concurrent events in the computing process.

Traditionally, parallel computing has been consider to be "the high and of computing" and has been motivated by numerical simulations of complex systems and "Grand Challenge Problems" such as weather and climate, chemical and nuclear reaction, mechanical devices etc.

Mainly there are two types of computing techniques:

- Sequential Computing
- Parallel Computing

**Sequential Computing:** As show in fig(1.1), in sequential computing the whole computation specify as a single stream of execution flow. The first generation of computers followed sequential and were referred to as VON NEUMANN organization. Such a computer comprises of an input and output device, a single memory for storing data and instruction , a single control unit for interpreting the instructions and single arithmetic and logic unit (ALU) for processing the data the most important feature of sequential computation is that each operation executed by the computer is perform one at a time.

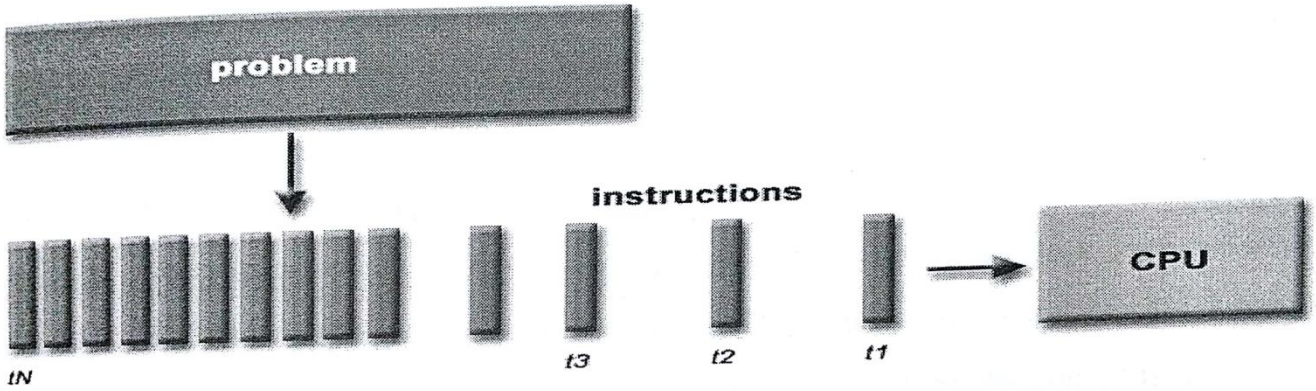


Fig 1.1 Sequential Computing

**Parallel Computing:** As Shown in fir(1.2), in parallel computing different program components execute concurrently on different processor and different paths of a computer execute different programs. Parallel programming specifies which program component are to execute in which part of the computer and how this components are to exchange data. Any program expressed as a parallel composition can be converted to sequential composition that interleaves the execution of various program components appropriately . However the use of parallel computing can enhance scalability and locality.

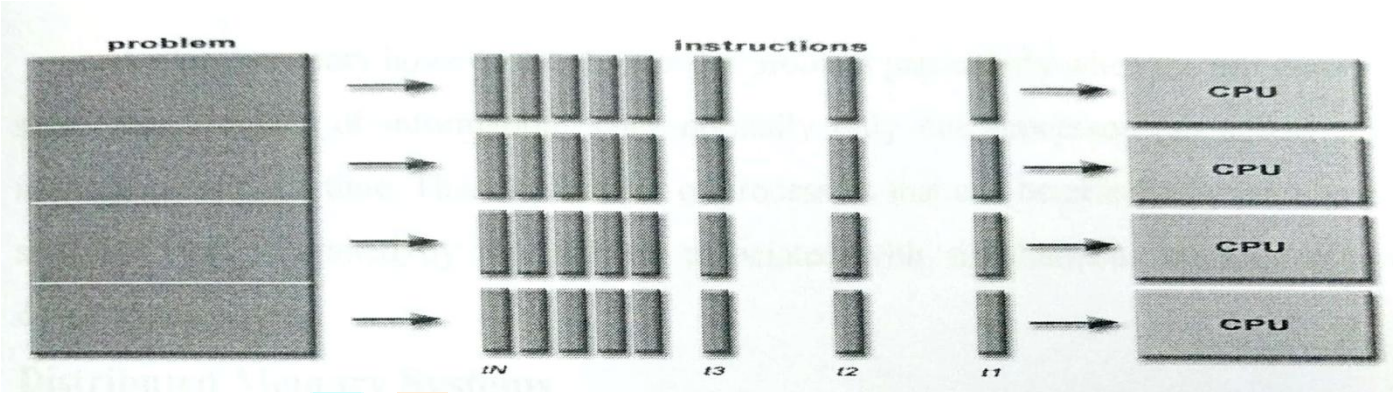


Fig (1.2) Parallel Computing

## 2. LITERATURE SURVEY

**Parallel Computing** can classified into two main groups:

- Shared Memory System
- Distributed Memory System

**Shared Memory System :** In shared memory architecture as show in figure(2.1) all the memory units reside in one global main memory that provide a convenient message depository for the processor - processor communication. Thus all the different processors can read or write into these global memory**.** A global memory however can be a major problem particularly when the processors must share large amounts of information, since normally only one processor can access a given memory module at a time. Thus the number of processors can be affectively put together in such a system which is limited by the problem associated with simultaneous memory access by different processors.

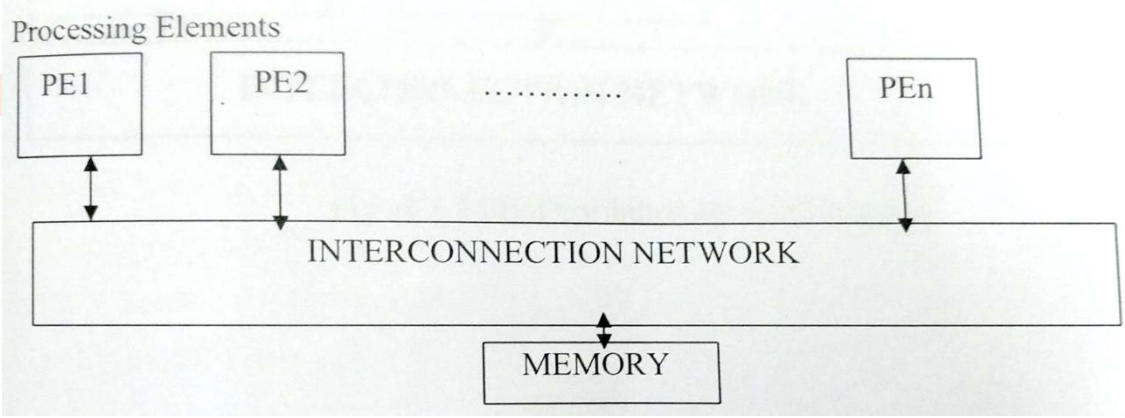


Fig (2.1) Shared Memory System

**Distributed Memory System:** In this system, the processor are provided with their own local memory a shown in figure(2.2) . Thus the global memory is reduced or even eliminated completely. To separate the functions of processing and memory, the processor is referred with no associated main memory as a processing element (PE). Thus a processor is combination of PE and local main memory; it way also include some external communication facilities forming in effect, a small self-contained element computer. In as system with little or no global memory processing elements communicate via messages transmitted between their local memories. Thus the system in which the main memory is the sum of local memories s refer to as distributed memory computers. The term message - passing computer is also use for these systems. These system have become the most common form of parallel processing .

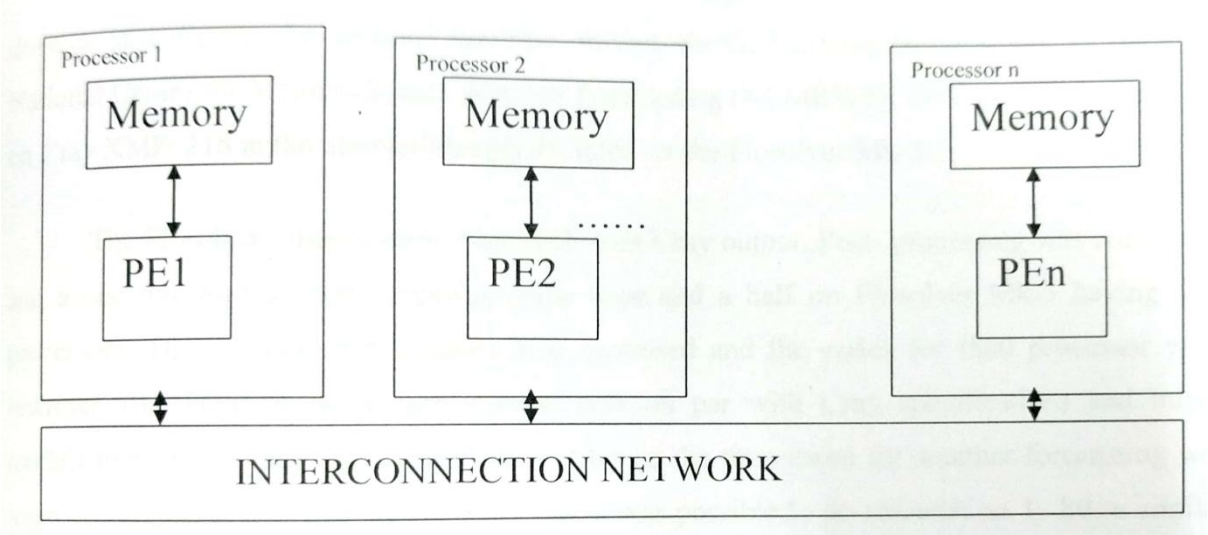


Fig (2.2) Distributed Memory System

**Semaphore:** A semaphore is protected variable or abstract data type that constitutes a classic method of controlling access by several processes to a common resource in a parallel programming environment. A semaphore generally takes one of the two forms: binary and counting. A binary semaphore is a simple "True/False (Locked/Unlocked)" Flag that control access to a single resource. A counting semaphore is a counter for a set of available resources. Either semaphore cane be employed to prevent race condition.

The value of semaphore is initialize by the first process when a file is in access by it. When the second process tries to access the file it checks the value of the semaphore and if it finds the value has initiate it does not access the file. After the first process completed it re-initializes the semaphore the value and now the second process uses it.

Semaphores are identify y semaphore ID which is unique for each semaphore . The semaphore can be incremented or decremented by using functions wait (sem) and signal (sem) respectively. Wait (sem) decrements the sempahore value and if the process of decrementing the value of semaphore reaches negative then the processes suspended and placed in queue for waiting. Signal (sem) increments the value of the semaphore and its is opposite in action to wait (sem). In other words its causes the first process in queue to get executed.

## 3. PROPOSED WORK

The main task is divided on the individual processors of different clusters. There are total of 1024 processor placed in 128 units, When the PE's need to interact with one another, they do so through a protocol called message passing. A message is essentially a collection of data along with header part, data and command. The header part contains information like source and destination PE's ID's, source and destination cluster IDs, size of data, source offset, destination offset etc. The communication PE constructs the packets and dumps these on DPM(dual port memory) present on intra-Cluster Communication.

### 3.1 Intra-Cluster Communication:

The Intra-Cluster communication refers to the exchange of data between the nodes/processing elements through the switch. The Message passing protocol is implemented for the data transfer between the PE's. In that the communication PE send data in form of packets which gets stored in corresponding DPM present on the switch. The packets consists of src_buffer, dest_buffer, count, number of switch. The switch keeps polling for command, whenever the command is recognized it takes the header and data from the corresponding DPM's and processing is done in the DPU( data processing unit) of the FPGA on the operands obtained from the PE's. After the processing is done the result is written back to the destination DPM and acknowledge for the PE collect tht data.

### 3.2 Cluster

A cluster is a type of parallel/distributed processing architecture consisting of as set of interconnected computers that can work as a single machine. The present requirement if to communicate among 1024 processors placed in 129 units, each unit having 16 optical links. Figure(3.2.1) below represents basic connectivity of the four cluster (C0,C1,C2 and C3) which forms a block.

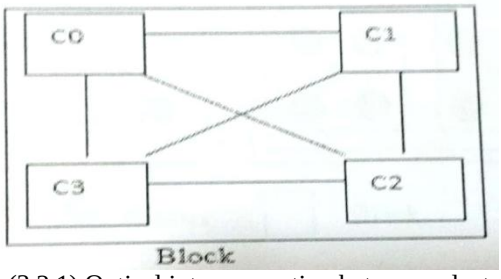


Fig (3.2.1) Optical interconnection between clusters

In the next level four such blocks (B0,B1,B2 and B3) are connected to form a group a shown in the figure (3.2.2) below.

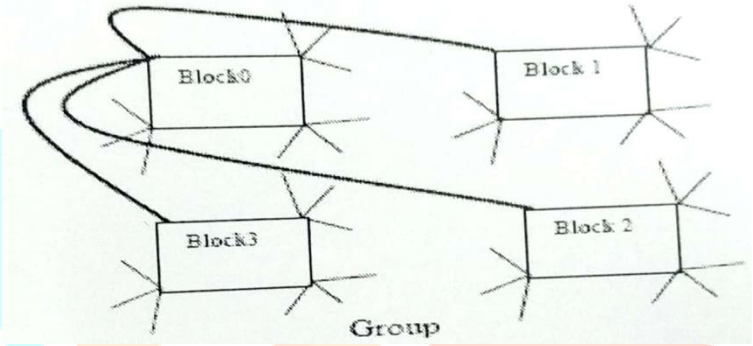


Fig (3.2.2) Schematic showing connection from one cluster of block

And four such groups (G0,G1,G2 and G3) are connected to form one region. Similarly another set of region will formed, and these two regions (R0,R1) are connected through optical link. Each cluster will be having sixteen optical links. In this method we are using ten optical links to connect the 1024 processor.

Figure (3.2.3) below represents the complete system showing the clusters, blocks, groups and regions. For simplicity, connection from one cluster are drawn among blocks, groups , and regions.

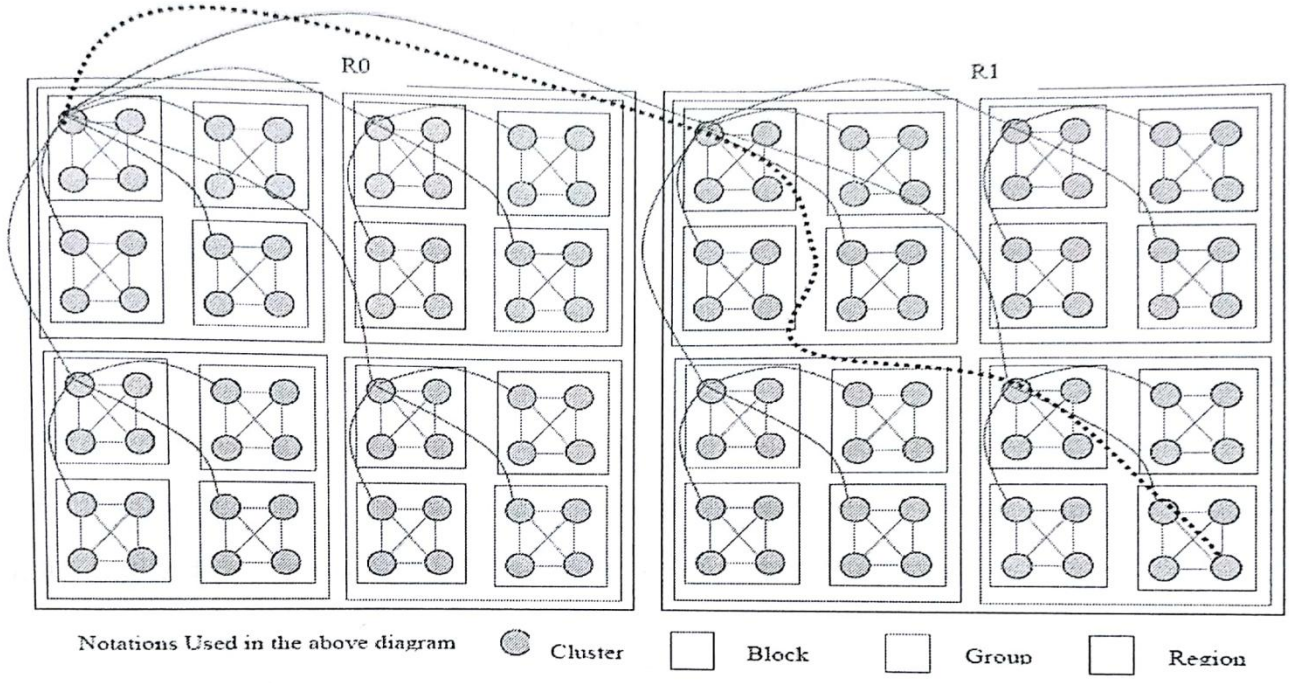


Fig (3.2.3)Complete System

## 4. SYSTEM DESIGN

This paper deals with the server side protocols for executing and transferring data of multiple application simultaneously o the correct destination. Initially a job is divided into many packets which are distributed among parallel servers. The parallel server communicate among each other using switch. When a single job is executing at a node then there is no confusion in transferring of data, but when there are multiple jobs executing simultaneously at a node then it is the responsibility of the protocol to transfer the data to the correct destination among the multiple tasks executing at the nodes of the parallel servers.

In figure(4.1) consider a job is divided in two parts J1 and J2. Job J1 is assigned to server S1 and job J2 is assigned to server S2. The two parts of the job communicate among each other using the switch. But the disadvantage of this is that only one job can be executed at a single node.

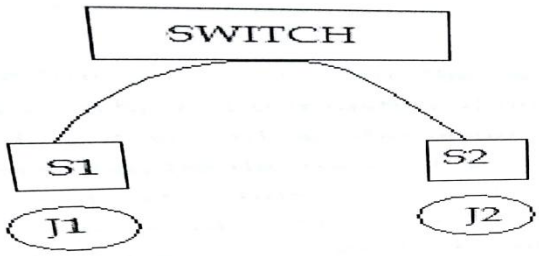


Fig (4.1) Single job switch

In figure(4.2) consider another job K which is divided into jobs K1 and K2 which are assigned to servers S1 and S2 respectively. In this case the data should be communicate correctly among the parts of the same job. This protocol will help us to facilitate correct communication among the jobs.

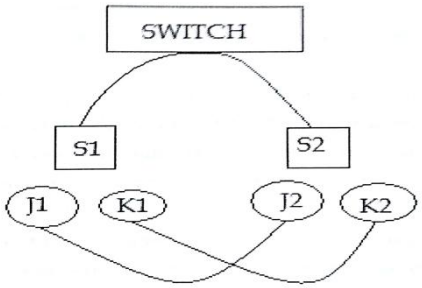


Fig (4.2) Multi Job Switch

The switch checks for the receive buffer status to check if the receiver is free and ready to receive the data. Both job J1 and J2 will request for a semaphore lock. The job which acquires the semaphore lock will send the data along with the task id. Ones the data is sent them the jobs sends the data it release the lock and the other job received the lock,. The switch transfers the data from the sender to receiver along with the respected task id that the job should receive. Ones the job received the data it checks if the task id matches with the task id that job should receive. If the task id matches then it receives the data else the other job receives the data. The send_data part consist of two routines, the buffer_status and the send_data routine.

**Buffer_status routine:** The readiness status values are generated by the receiver PE. These values are transferred to an address location in the sources PE's by the switch. In the source PE the headers are matched. his transfer of the readiness status values gives then status of the receiver buffer. Then the send data can send the data to the specified address location of the destination PE.

**Send_data routine:** The send_data routine is responsible for sending the data. It checks the sending buffer for its status, prepares the headers and data, checks the receiver buffer status and the sender gives the command. The switch gets the command transfers the data and the header, writes the task completion status and changes the parity. In the meantime the receiver receives the data and reads it.

**The Switch:** In the buffer status routine, The receiver generates the readiness value which is the receiver buffer status value that is transferred by the switch, to the specified address location of the source PE.

In the send data routine, the data transfer is done, followed by the header transfer, the switch transfers the complete data to the receiver's buffer and the status is written to both the sender and receiver.

**The PE:**

**Sender**: Checks the send buffer to be free, and then prepares the data and the header. Then asks for the receiver readiness, once the receiver is ready, it givers the send command.

**Receiver**: Generates the receive status buffer value then waits for the arrival of the new packet id and the packet.

**5. ALGORITHM**

5.1 In sending the data from one PE to the other the first function that is called is the BSEND_S_NEW function inside the main. The primary reason for calling this function is to check whether the data is going to sent locally i.e. within the same unit or that the data would be sent to receiver PE in another unit.

```
void BSEND_S()
{
    if source  and destination is within same PE
    call bsend_shm()
    else source and destination within same cluster
    call Sem_Request_sw() //Requesting Semaphore
    call NEW_BSEND()
    Sem_Release_sw()  //Requesting Semaphore
    else source and destination in different cluster
    call OPTI_BSEND()
}
```

5.2 The first step is that this function would do upon invoking would be to dived the total data that is going to be sent into packets depending upon the size_arr. The data would be divided into packets and the send_data function would be called upon.

```
void BSEND_N()
{
    divide the data into packets
    calculate the total no of packets
    call new_send_data()
    check if there are more packets
    call new_send_data() until there are no more packets
}
```

5.3 The send data function is the primary function which would send the data that has been divided into packets by the NEW_BSEND functionality. There are many tasks that this function has to do so it requires other sub functions to complete those tasks.

```
void new_send_data()
{
    //check send buffer is free or not
    flo_check_send_buffer_status();
    //prepare the header and data
    prepare_the_header_and_data();
    //check whether the receiver is free to recv or not  flo_get_recv_buffer_status();
    write_the_buffer_id to mem,
    check for even or odd buffer
    else
    error_handler("buffer id not matched with 0/1 ...exiting\n");
    clearing the other command location
    give the send cmd
    check whether transfer is over or not
    change the parity
}
```

5.4 In receiving the data from one PE, the first function that is called is BRECV_S function inside the main. The primary reason for calling this function is to check whether the data is going to be received locally i.e. from one PE in the same unit or that the data would be received from the sender in another PE in another unit.

```
void BRECV_S()
{
    if source and destination within same PE
    call brecv_shm()
    else if within same cluster
```

```
        {
                if(setjmp(jmpbuffer)!=0)
                Sem_Relese_sw()  //Requesting Semaphore
                Sem_Request_sw() // Requesting Semaphore
                BRECV_N();
                Sem_Release_sw()  // Releasing Semaphore
        }
        else
        call OPTI_BRECV()
}
```

5.5 The data is being set is going to arrive in the form of packets, so in this function the packet division again takes palce at the receiver's side which depends upon the size_arr. The data packets would be received and the recv_data function would be called upon.

```
void BRECV_N()
{
`       receive the data in packets
        calculate the total no packets to be arrived
        call recv_data() until there are no more packets
}
```

5.6 The recv_data function, when called upon by the BRECV_N, will basically receive data. It will perform many tasks so it requires other sub functions to complete those tasks.

```
void recv_data()
{
        write the readiness status
        if g_task_id matches with the source task id
        printf ("both task id's are equal,");
        else
        longjmp(jmpbuffer,1)
        Read the data and header
        Read the data from dpm
        write the buffer free status
}
```

5.7 This function is used to release the semaphores that has been acquired by some process. Once its released other process can acquire it.

```
int Sem_Request()
{
        set required flags
        error=semop()
        ret=semctl()
        if(error<0)
        perror("Semaphore Request Fail :")
        else
        print Sem Request is Finished
        return(error);
}
```

5.8 This function is used to request semaphores to lock it for using shared resources. Once the request is granted no others process can use the resource.

```
int Sem_Release(int semid, int index)
{
        arg= size of semun
        memset(arg,0,sizeof(union semun));
        error=semop(semid, &sb,1);
```

```
        if error is lesss than 0
        perror("Semaphore Release Fail: ");
        ret= semctl()
        print Sem Release Val
        free(arg)
        return(error);
}
```

## 6. CONCLUSION AND FUTURE ENHANCEMENT

In this paper the send and receive communication protocol was enhanced to support the multitasking. The semaphore was used to handle the mutual exclusiveness among the process. To identify which packet corresponds to which task, task_id was included so that data will be read by the correct task/process. To control the flow of execution setjump and longjump were used in this paper.

The send and receive communication protocol was enhanced to support multitasking and it has been checked for two independent tasks. The same can be checked for more tasks. The multitasking concept can extended to other communication protocols like floatadd and findmax operations.

## 7. REFERENCES

[1]. J. Noguera, R. M. Badia, "Multitasking on reconfigurable architectures: Microarchitecture support and dynamic scheduling", *ACMTrans. Embed. Comput. Syst.*, vol. 3, pp. 385-406, May 2004.
[2]. S. J. Kim, "A general approach to multiprocessor scheduling", 1988.
[3]. S. J. Kim, J. C. Brown, "A general approach to mapping of parallel computation upon multiprocessor architectures", *Proc. Int. Conf. Parallel Process.*, vol. 3, pp. 1-8, 1988.
[4]. O. Sinnen, "Task scheduling" in Task Scheduling for Parallel Systems, Hoboken, NJ, USA:Wiley, pp. 74-107, 2007.