Q1:
Discuss the concept of computational granularity and communication latency. Further, discuss how the two concepts are related at various levels, e.g., instruction level, loop level, procedure level etc.

Ans:

Concept of computational granularity and communication latency: granularity refers to the amount of computation done in parallel relative to the size of the whole program. Granularity is a qualitative measure of the ratio of computation to communication. According to granularity of system, parallel processing system is divided into two groups:

- **i.** Five grain system
- **ii.** Coarse grain system

In five grained system parallel parts are relatively small and that means high communication overhead.

In coarse grain system parallel parts are relatively large, that mean more computation and less computation.

If granularity is too fine, it is possible that the overhead required for the communication and synchronization between task takes longer than the computation. On the other hand, in coarse gain parallel system, relatively large amount of computation work is done. They have high computation work to communication ration and imply more opportunity for performance increase.

<u>Various levels:</u>

- **i.** Instruction level: it refers the situation when different instructions of program are executed by different processing elements. Most processes have parallel execution of micro steps of instructions with in the same pipe. The idea of executing a number of instructions of a program in parallel by scheduling them on a single processor has been a major driving force in the design of recent processor.

- **ii.** Loop level: consecutive loop iteration is the candidates for parallel execution. However, data dependencies between subsequent iteration may restrict parallel execution at loop level.

iii. Procedure level: parallelism is available in the form of parallel executable procedure. In this case, the design of algorithm plays a major role. For example, each thread in java can be spawned to run a function or method.

Program Level: this is usually the responsibility of operating system which runs processes concurrently. Different program are obviously, independent of each other. So parallelism can be extracted by operating system at this level.

Q2:
Using Bernstein's conditions, detect maximum parallelism between the instructions of the following code:
P1: X = Y * Z
P2: P = Q + X
P3: R = T + X
P4: X = S + P
P5: V = Q / Z

Ans:

Detection of parallelism by using Bernstein condition:

This condition based on the following tow sets of variable:

i. The Read set or input set R_1that consists of memory locations read by the statement of Instruction I_1,
ii. The Write set or output set W_1that consists of memory locations written into by instruction I_1,

The set R_1 & W_1 are not disjoint as the same locations are used for reading and writing by S_1.

Statement is parallel or not:-

i. Location in R_1 from which S_1 reads and the locations W_2 onto which S_2 writes must be mutually exclusive, that means S_1 does not read from any memory location onto which S_2 writes. It can be denoted as:- $R_1 \cap W_2 = \text{empty}$

ii. Similarly, location on R_2 from which S_2 read and the locations W_1 onto which S_1 writes must be mutually exclusive. That means S_2 does not read from any memory location onto which S_1 writes. It can be denoted as: $R_2 \cap W_1$=empty.

iii. The memory locations W_1 and w_2 onto which S_1 and S_2 write should not be read by S_1 and S_2 that means R_1 and R_2 should be independent of w_1 and w_2. It can be denoted as: $R_2 \cap W_1$ =empty.

iv. The memory location W_1 and W_2 onto which S_1 and S_2 write should not be read by S_1 and S_2 that means R_1 and R_2 should be independent of W_1 and W_2. It can be denoted as: $W_1 \cap W_2$ =empty.

P_1: X=Y*Z
P_2: P=Q+X
P_3: R=T+X
P_4: X=S+P
P_5: V=Q/Z

Read set and Write set of P_1, P_2, P_3,, P_4, P_5

R_1= {Y, Z}	W_1= {X}
R_2= {Q, X}	W_2= {P}
R_3= {T, X}	W_3= {R}
R_4= {S, P}	W_4= {X}
R_5= {Q, Z}	W_5= {V}

Let's find whether P and P are parallel or not:-

$R_1 \cap W_2 = \emptyset$, $R_2 \cap W_1 = \emptyset$, $W_1 \cap W_2 = \emptyset$
Here P_1 and P_2 are not independent of each other. So P_1 is not parallel to P2

Now see P1 and P3 are parallel or not:
R1 ∩ W3=empty, W1 ∩ R3 is not empty.
So P1 not parallel to P3

P1 and P4 parallel or not:
R1 ∩ W4=empty, R4 ∩ W1=empty, W1 ∩ W4=empty
So P1 and P4 are not parallel

P1 and P5 parallel or not:
R1 ∩ W5=empty, R5 ∩ W1=empty, W1 ∩ W4 is not empty
So P1 and P5 are independent of each other. So P1 and P5 are parallel

Now P2 and P3 are parallel or not:
R2 ∩ W3=empty, R3 ∩ W2= empty, W2 ∩ W3=empty
So P2 and P3 are parallel

P2 and P4 are parallel or not:
R2 ∩ W4 is not empty, R5 ∩ W5=empty W2 ∩ W5=empty
So P2 is parallel to P5

P3 and P4:
R3 ∩ W 4 is not empty, So P3 not parallel to P4

P3 and P5:
R3 ∩ W5=empty, R5 ∩ W3=empty, W3 ∩ W5=empty. So P3 is parallel to P5

P4 and P5:
R4 ∩ W5=empty, R5 ∩ W4=empty, W4 ∩ W5=empty. So P4 is parallel to P5.

So result is P_1, P2 and P_5 are parallel.

Q3:
(i) Obtain Perfect Shuffle Permutation network of 32 nodes.

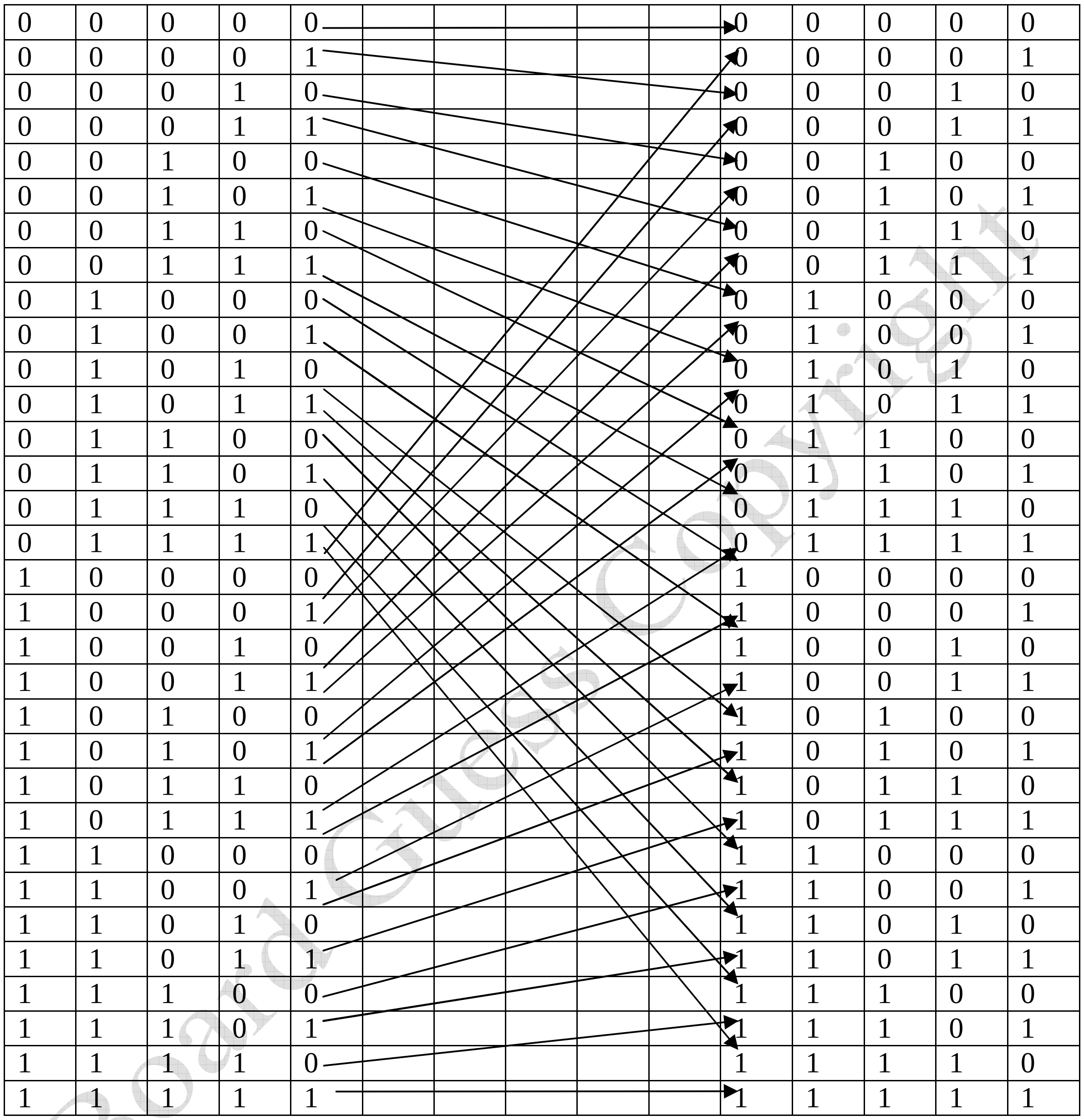
Ans :

Perfect shuffle permutation→Consider N objects each represented by 4 bits, 4 number say X_{n-1}, X_{n-2}, X_0. The perfect shuffle of this N object 4 expressed as.

$X_{n-1}, X_{n-2}, X_0 = X_{n-2}, X_0, X_{n-1},$

This means that perfect shuffle is obtained by rotating the address by 1 bit left.

Shuffle permutation of 32 Nodes:



0	0	0	0	0						0	0	0	0	0
0	0	0	0	1						0	0	0	0	1
0	0	0	1	0						0	0	0	1	0
0	0	0	1	1						0	0	0	1	1
0	0	1	0	0						0	0	1	0	0
0	0	1	0	1						0	0	1	0	1
0	0	1	1	0						0	0	1	1	0
0	0	1	1	1						0	0	1	1	1
0	1	0	0	0						0	1	0	0	0
0	1	0	0	1						0	1	0	0	1
0	1	0	1	0						0	1	0	1	0
0	1	0	1	1						0	1	0	1	1
0	1	1	0	0						0	1	1	0	0
0	1	1	0	1						0	1	1	0	1
0	1	1	1	0						0	1	1	1	0
0	1	1	1	1						0	1	1	1	1
1	0	0	0	0						1	0	0	0	0
1	0	0	0	1						1	0	0	0	1
1	0	0	1	0						1	0	0	1	0
1	0	0	1	1						1	0	0	1	1
1	0	1	0	0						1	0	1	0	0
1	0	1	0	1						1	0	1	0	1
1	0	1	1	0						1	0	1	1	0
1	0	1	1	1						1	0	1	1	1
1	1	0	0	0						1	1	0	0	0
1	1	0	0	1						1	1	0	0	1
1	1	0	1	0						1	1	0	1	0
1	1	0	1	1						1	1	0	1	1
1	1	1	0	0						1	1	1	0	0
1	1	1	0	1						1	1	1	0	1
1	1	1	1	0						1	1	1	1	0
1	1	1	1	1						1	1	1	1	1

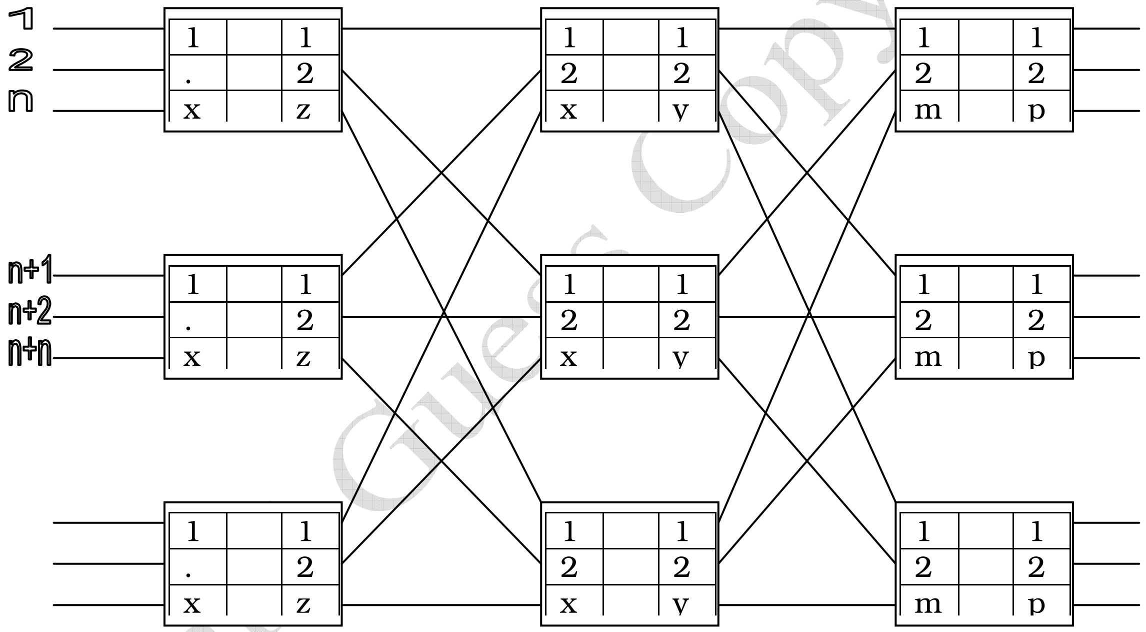
3:
(ii) Discuss, along with diagrams, close network with 4x4 cross point switches.

Ans :

Close Network:

It is a non-blocking network and provides full connectivity like crossbar network but it requires significantly less number of switches. The organization

of close network is shown below using 4 X 4 cross point switches:



Organization of Close Network

Q4:
Discuss, along with diagram, an arithmetic pipeline for Multiplication of two 8-digit fixed numbers.

Ans:

Arithmetic Pipelining for fixed numbers Multiplication of 8 digit fixed numbers

X7	X6	X5	X4	X3	X2	X1	X0	=	X
Y7	Y6	Y5	Y4	Y3	Y2	Y1	Y0	=	Y
X7Y0	X6Y0	X5Y0	X4Y0	X3Y0	X2Y0	X1Y0	X0Y0	=	P1
X7Y1	X6Y1	X5Y1	X4Y1	X3Y1	X2Y1	X1Y1	X0Y1	=	P2
X7Y2	X6Y2	X5Y2	X4Y2	X3Y2	X2Y2	X1Y2	X0Y2	=	P3
X7Y3	X6Y3	X5Y3	X4Y3	X3Y3	X2Y3	X1Y3	X0Y3	=	P4
X7Y4	X6Y4	X5Y4	X4Y4	X3Y4	X2Y4	X1Y4	X0Y4	=	P5
X7Y5	X6Y5	X5Y5	X4Y5	X3Y5	X2Y5	X1Y5	X0Y5	=	P6
X7Y6	X6Y6	X5Y6	X4Y6	X3Y6	X2Y6	X1Y6	X0Y6	=	P7
X7Y7	X6Y7	X5Y7	X4Y7	X3Y7	X2Y7	X1Y7	X0Y7	=	P8

Following stages for pipelining:

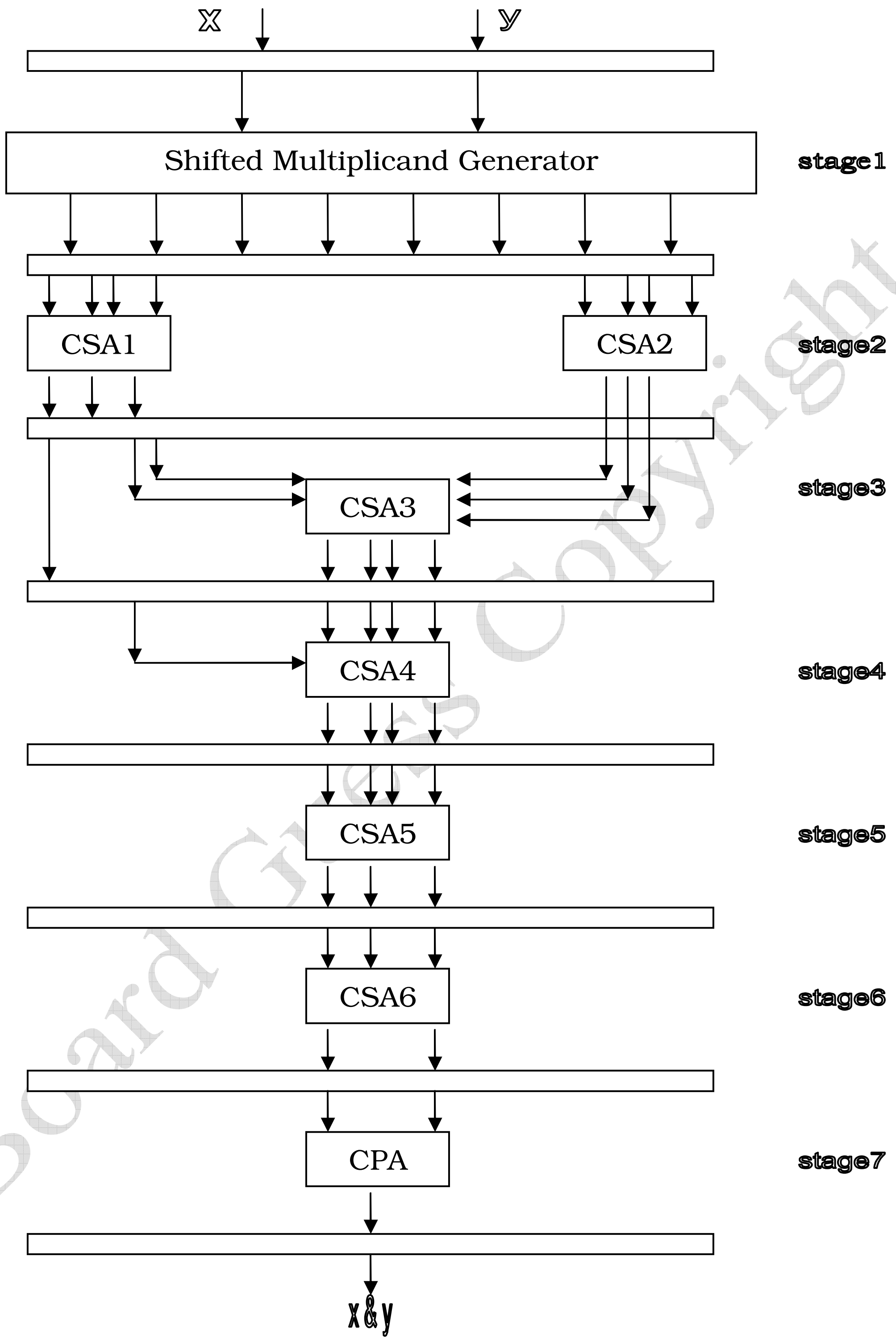
1. The first stage generates the partial product of number, which form the six rows of shifted multiplicands.

2. In second step, the eight are given to the two carry save address merging six numbers.

3. Third step: a single CSA merging the number into 5 numbers.

4. Similarly in next step 5 numbers into 4 number and 4 number into 3.

5. In last step, two numbers are added through a carry propagation adder (CPA) to get the final result. X & Y are two 8 digit fixed number so arithmetic pipeline for multiplication of two 8 digit fixed number is given below:



Q5:

Define Bitonic sequence. Discuss a Bitonic sorting algorithm. Further, using the algorithm, sort the following sequence:

15,17,19,20,25,27,29,34,37,18,16,13,10,8,7,6,2

Ans:

Biotic Sequence: - Consider a sequence X=X0, X1, X2….Xn-1 such that conditions:

1. Either X0, X1, X2….Xi is monotonically increasing sequence and Xi+1, Xi+2….Xn-1 is monotonically decreasing sequence.
2. There exist cyclic shift of the sequence X0, X1, X2….Xn-1 such that the resulting sequence satisfies the condition 1.

Algorithm for sorting the bitonic sequence:

Sort_Bitonic(X)

1. The sequence i.e. X is transferred on the input lines of the combinational circuit which consists of various set of comparators.
2. The sequence X is splitted into two sub bitonic called bitonic split.
3. Recursively execute the bitonic split on the subsequence i.e. Y and Z until the size of subsequence reaches to a level as 1
4. This sorted sequence is achieved after this stage on the output lines.
   Now sorting of given sequence:

15,17,19,20,25,27,29,34,37,18,16,13,10,8,7,6,2

This list is to be sorted in ascending order. To sort this list, in the first stage comparator or order 2 will be used.

Similarly 2nd stage will consist of 4, input comp.

3rd stage 8 input comparator

And 4th stage 16 input comparator.

15		15		15		15		2
17	+(BM2)	17		17		17		6
19		20	+(BM4)	19		19		8
20	-(BM2)	19		20		20		10
25		25		34	+(BM8)	25		13
27	+(BM2)	27		29		27		15

29		34	-(BM4)	27		29		16
34	-(BM2)	29		25		34		17
37		18		13		37	+(BM16)	18
18	+(BM2)	37		16		18		19
16		16	+(BM4)	18		16		20
13	-(BM2)	13		37		13		25
10		10		10	-(BM8)	10		27
8	+(BM2)	8		8		8		29
6		6	-(BM4)	6		6		34
2	-(BM2)	2		2		2		37

Q6:
Discuss the following with respect to a parallel virtual machine:

(i) Compiling and running of a PVM program

Ans:

Compiling and running of PVM program.

To compile the program change to the directory PVM/lib/archname,
Where archname is the architecture name of your computer, then the following command

```
cc program.c  -lpvm3          -oprgram,
```

will compile a program called 'program.c'. After compiling, we must put the executable file in the directory pvm3/bin/ARCH. Also, we need to compile the program separately for every architecture in virtual machine. In case we use dynamic groups, we should also add –lgpvm to the compile command. The executable file can be run. To do these first run PVM. After PVM is running, executable file may be run from the UNIX command line, like any other program.

PVM supplier an architecture independent make, aimk, which automatically determines PVM_ARCH and links any operating system specific libraries to your application to compile the C

```
% aimk makser.c
```

Now, form one window, start PVM and configure some host. In other window change directory to $HOME/pvm3/bin/PVM_ARCH and type % master.

It will ask for a number of task to be executed then type number of task.

Q6:
(ii) Message passing

Ans:

Message passing wrt PVM:-

PVM communication model provides asynchronous blocking send asynchronous blocking receive, and non-blocking send return as soon as the send buffer is free for reuse and an asynchronous send does not depend on the receiver calling a matching receive before the send can return there is option in PVM that data be transferred directly from task to task. In this case, if the message is large the sender may be block until the receive has called a matching receive.

A non-blocking receive immediately returns with either the data or a flag that the data has not arrived, while a blocking receive returns only when the data is in the receive buffer. In addition to these point to point communication functions, the model supports the multicast to a set of tasks and the broadcast to a user-defined group of task. There are also functions to perform global max, global sum etc. access a used defined group of task.

PVM guarantees that the message order is preserved if task1 sends message A to task and then task1 send message B to task 2 message A will arrive at task 2 before message B. moreover, if both the message arrive before task2, does a receive, then a will always return message A.

Int bufid-pvm_mkbuf(int encoding)

Create a new message buffer, encoding specifies the buffer's encoding set.

Q6:
(iii) Creating and managing Dynamic process groups

Ans:

Create and mange dynamic groups:-

The separate library lib g pvm 3 a must be linked with a user program that makes use of any of the group fund group management work is handled by a group server that is automatically started when the first group function is invoked.

We are giving some routines that handle dynamic process:-

Int pvm_joingroup(char *group)

Emols the calling process in a named group. Group is a group name of an existing group. Returns instance number. Instance number run form 0 to the number of group members minus 1. in PVM3, a task can join multiple groups.

- int info=pvm_lvgroup(char *group)

Unenrolls the calling process from a named group.

- int pvm_gettid(char *group, int inum)

Return the tid of the process identified by a group name and instance number.

- int pvm_getinst(char *group, int tid)

Return the instance number in a group of a PVM process.

- int size=pvm_gsize(char *group)

Return the number of member presently in the named group.

- int pvm_barrier(char *group, int count)

Block the calling process until all the process in a group have called it. Count species the number of group members that must call pvm_barrier before they are all released.

int info = pvm_reduce(void (*func)(), void *data, int count, int datatype, int msgtag, char *group, int rootginst)

Q7:
Discuss important environment features for parallel programming.

Ans:

The parallel programming environment consists of an editor, a debugger, performance, evaluator and program visualize for enhancing the output of parallel computation. All programming environment have these tools in one form or the other. Based on the feature of the available tool sets the programming environment are classified as basic, limited and well developed.

a. Basic environment provides simple facilities for program tracing and debugging.

b. The limited integration facilities provide some additional tools for parallel debugger and performance evaluation.

c. Well developed environment provide most advanced tools of debugging programs, for textual graphics interaction and for parallel graphics handling.

There are certain parallel overhead associated with parallel computing. The parallel overhead is the amount of time required to co-ordinate parallel tasks as opposed to doing useful work. These include the following factors:-

i. Task start up time
ii. Synchronization.
iii. Data communication.

Besides this hardware overhead, these are certain software overhead imposed by parallel compiler, libraries, tools and operating systems.

Parallel programming languages are developed for parallel computer environments. These are developed by either introducing new languages or by modifying existing language. Normally, the language extension approach is preferred by most computer design. This reduce compatibility problem. High level parallel constructs were added to FORTRAN and C to make these languages suitable for parallel computers. Beside these, optimizing compilers are designed to automatically detect the parallelism in program code and convert the code to parallel code.

Q8:
Discuss relative merits and demerits of various laws for measuring speed up performance vis-à-vis to a parallel computer algorithm system

Ans:

Merits and demerits of various laws for measuring speed up performances;-

1. **Amdahl's Law:-**

The speed up factor help us in knowing the relative gain achieved in shifting the execution of a task from sequential computer to parallel computer and the performance does not increase linearly with the increase in number of processor.

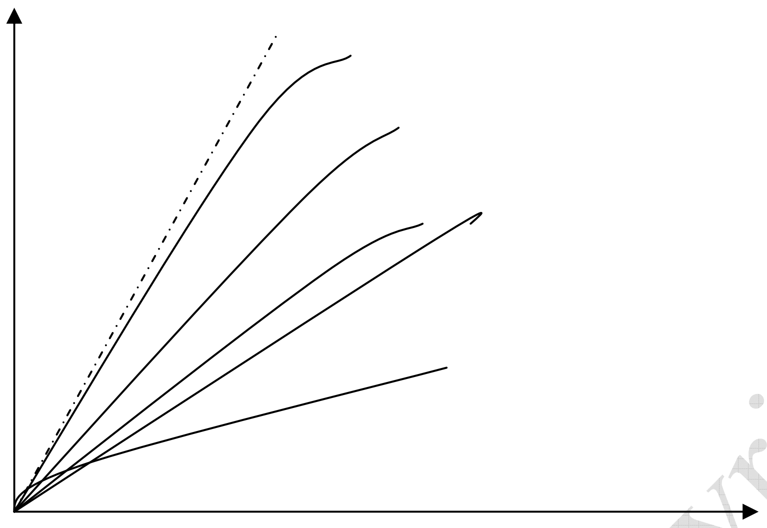
Let us consider a problem say P, which has to be solved using a parallel computer. According to Amdahl's law, there are mainly two types of operations; therefore, the problem will have some sequential operation and some parallel operations. We already know that it requires T (1) amount of time to execute a problem using a sequential machine and sequential algorithm. The time to compute the sequential operation is a fraction α ($\alpha<=1$) of the total execution time i.e. T (1) and the time to computer the parallel operations is $(1-\alpha)$, therefore S (N) can be calculated as under:-

$$S(N) = T(1)/T(N)$$
$$S(N) = T(1)/\ (\alpha * T(1) + (1-\alpha) * T(1)/N)$$

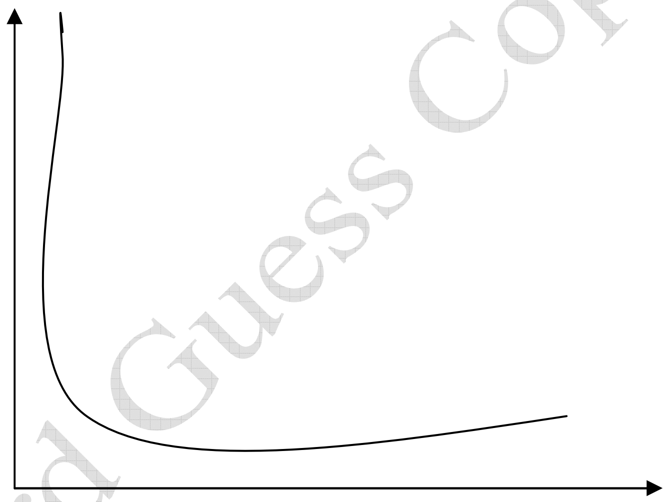
Dividing by T (1)

$$S(N) = 1/\ (\alpha + (1-\alpha)/N)$$

Remember the value of α is between 0 and 1. Now put some values of number of processors, we find that the S (N) keeps on decreasing with increase in the value of α.



Speed v/s No. Of Processors



Speed up Factor S (N) v/s α

<ins>Outcomes of analysis of Amdahl's Law:-</ins>

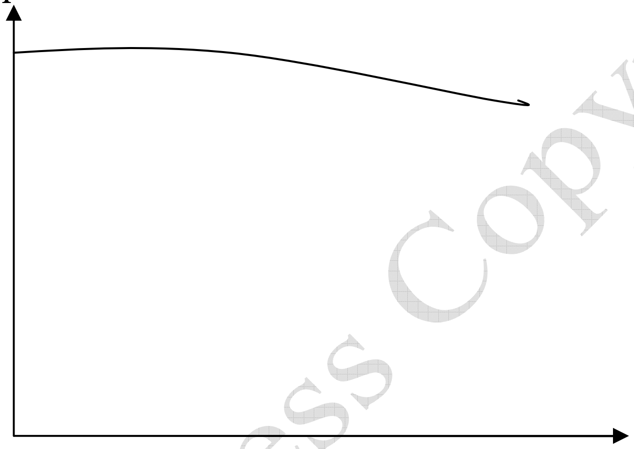
1. To optimize the performance of parallel computers modifies compiler need to be developed which aim to reduce the number of sequential operation pertaining to the reaction α.

2. Manufacturers of parallel computers were discouraged from manufacturing large scale machine having millions of processors.

One major shortcoming identified in Amdahl's law: according to Amdahl's law the problem size is always fixed and of sequential operations remains mainly same.

2. Gusta fson's Law:-

There are menu applications which require that accuracy of the resultant output should be high. In the present situation the computing power has increased substantially due to increase in number of processors attached to parallel computer. Thus it is possible to increase the size of the problem. The graph of speed up:



S (N) = **α +N*(1- α)**
S (N) = **N- α*(N-1)**

Thus decrease is because of overhead or sizes caused by inter processor communication.

3. Sun and Ni's Law:-

The Sun and Ni's Law is a generalization of Amdahl's Law as well as Custafson's Law. The fundamental concept of underlying the Sunand Ni's Law is to find the solution to a problem with a maximum size along with limited requirement of memory. Now a day, there are many applications which are bounded by the memory in contrast to the processing speed.

In a multiple based parallel computer, each processor has an independent small memory. In order to solve a problem, normally the problem is divided into sub problems and distributed to various processors. It may be noted that size of sub-problem should be in proportion with size of the independent local memory available with the processor. The size of the problem can be increased further such that the memory could be utilized. This technique assists in generating more accurate solution as the problem size has been increased.