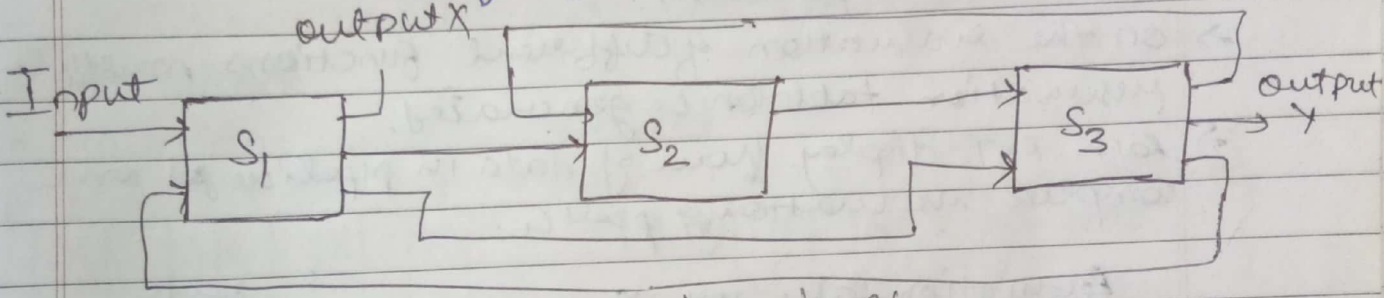
Throughput for non pipelin
$= \frac{25}{4} \times 10^6$ instr/sec

$= 6.25$ MIPS

Linear Pipelines: They are used to perform fixed functions
Non-Linear Pipeline: are dynamic pipelines. It allows feed back and feed forward connections in addition to streamline connections.

- It can be reconfigured to perform different functions at different time i.e Multifunctional
- Gives more than one output, output of the pipeline is not necessarily from the last stage.



3-stage non-linear pipeline

Two function X and Y not necessarily from last stage

Streamline connections:	Feed forward	Feed back
S1 to S2 S2 to S3	S1 to S3	From S3 to S2 S3 to S1

Reservation Table

It specifies the utilization pattern of successive stages in a pipeline

- Rows represents stages and columns represent clock time units.

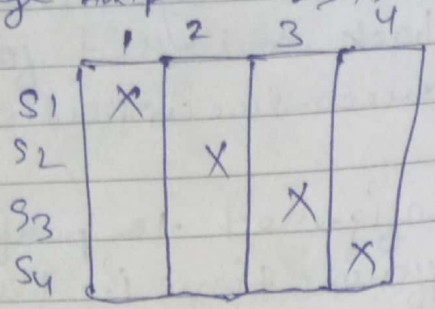
For Linear Pipeline

$\frac{tc}{hd}$

c = logic cost
t = Total time
h = latch cy
d = delay

* Reservation table for Linear Pipeline Processor follows diagonal stream line pattern.

R-T for 4 stage Pipeline → Time (clock cycles)

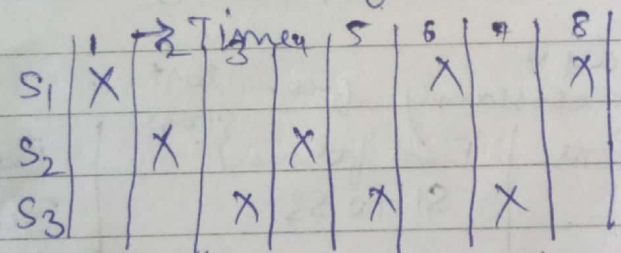


For Non-Linear Pipeline.

→ Non-Linear pattern
→ On the evaluation of different functions multiple reservation table can be generated.
→ Each R-T display flow of data in pipeline for one complete evaluation of fn.

Reservation table for X

δ = clock units. = evaluation time.



Processing sequence for function X

S1 → S2 → S3 → S2 → S3 → S1 → S3 → S1

→ Multiple check marks in a row means repeated usage of same stage in different cycle.

→ Contiguous check marks - extended usage of stage over more than one cycle.

→ Multiple check marks in simulta. usage of multiple stages same cycle.

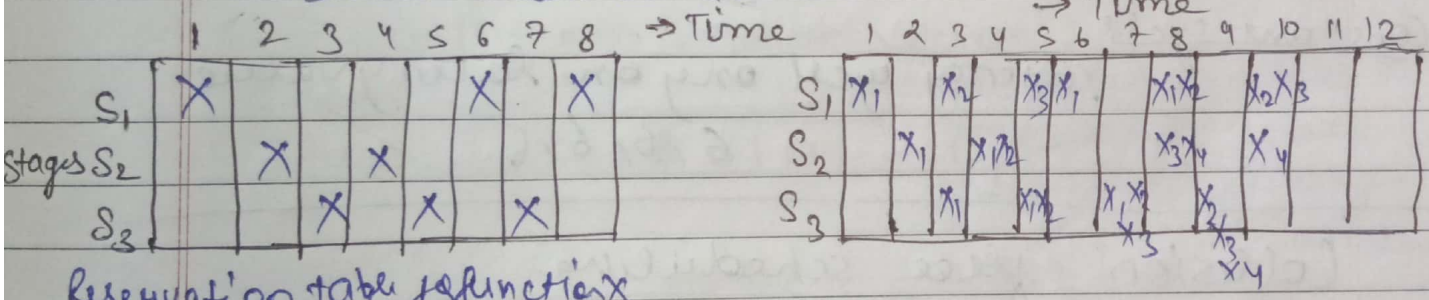
→ Total no of clock units = evaluation time.

static pipeline → linear pipeline.
dynamic → non-linear pipeline

Page :
Date : / /

Latency :
- An initiation refers to the start of single function evaluation.
- No of clock cycles between two initiation of pipeline is the latency.
- Latency values must be positive integers.

Collisions :
- An attempt by two or more initiation to use the same pipeline stage at the same time is called collision.
- Collision implies resource conflicts between two initiations.
- Collisions must be avoided.



Reservation table of function x

Forbidden Latencies

Latencies that cause collisions are called forbidden latencies.

eg Considering the reservation table

S_1 : {(6-1), (8-1), (8-6)} i.e {5, 7, 2}

S_2 : {(4-2)} i.e {2}

S_3 : {(5-3), (7-3), (7-5)} i.e {2, 4, 2}

forbidden latencies {2, 4, 5, 7}

Permissible latencies

- Latencies that does not cause collisions are permissible latencies.
- Permissible latencies = {1, 3, 6, 8} for last problem.

Latency sequence:
Sequence of permissible latencies
{1, 3, 6, 8}.

Latency cycle:
A latency sequence that repeats the same subsequence (cycle) indefinitely.
For ex: repeat after 1, 3, 6, or 8.

Average latency
the sum of all latencies divided by the number of latencies along the cycle.

$$\frac{1+8}{2} = 4.5$$

Constant cycle
latency cycle only one latency value
6, 6, 6, 6.

Collision free scheduling

To avoid collisions and to achieve high throughput, all the tasks awaiting for initiation should be properly scheduled.
while scheduling main objective is
To obtain shortest average latency between initiations without causing collisions.
Steps:
Collision vector:
It is m-bit binary vector $C = (C_m C_{m-1} \dots C_2 C_1)$ that can be obtained from combined forbidden set of permissible and forbidden latency.
$m <= n-1$, n = column.

$C_i = 1$ Forbidden.

$C_i = 0$ Permissible

eg $C_x = 1011010$

Page :
Date : / /

In the previous example

forbidden latencies = {2,4,5,7}

Maxim F.L = 7

So m = 7

collision vector $C_x = \{C_7 C_6 C_5 C_4 C_3 C_2 C_1\}$

Permissible latency = {1, 3, 6}

Now for Forbidden latencies = {2,4,5,7}

collision vector

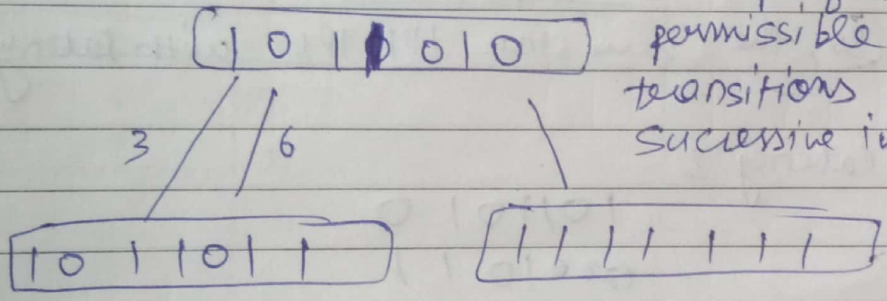
$= \{C_7 C_6 C_5 C_4 C_3 C_2 C_1\} = (1011010)$

1 0 1 1 0 1 0

State diagram

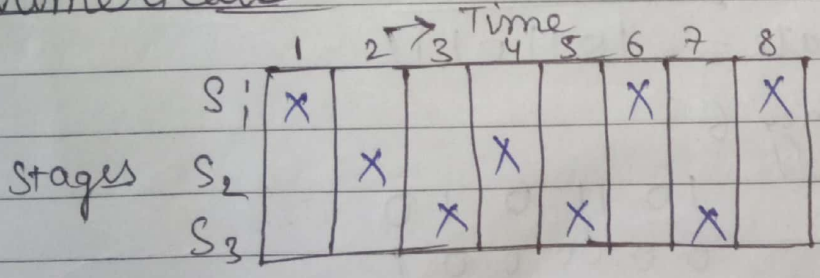
Constructed from collision vector

It specifies the permissible state transitions among successive initiations.



Numericals

1.



S1: {(6-1), (8-6), (8-1)} i.e {5, 2, 7}

S2: {(4-2)} i.e {2}

S3: {(5-3), (7-5), (7-3)} i.e {2, 2, 4}

forbidden latencie = {2, 4, 5, 7}

It cause collision.

Permissible latencies = {1, 3, 6, 8+}
It does not cause collision

Page :
Date

Of these all are permissible.

Maximum forbidden latency (m) = 7
- so the collision vector is of 7 bits.

Collision vector = $\{C_7 C_6 C_5 C_4 C_3 C_2 C_1\}$

1 = forbidden.
0 - Permissible.

Collision vector = (1011010)

Permissible latency = {1, 3, 6, 8+}
Initial collision vector
1011010

with latency 1

```
  1011010
+ 0101101
  1111111   Next State
```

So, the new state 1111111 with latency 1

with latency 3

```
  1011010
+ 0001011
  1011011
```

next state = 1011011

with latency 6

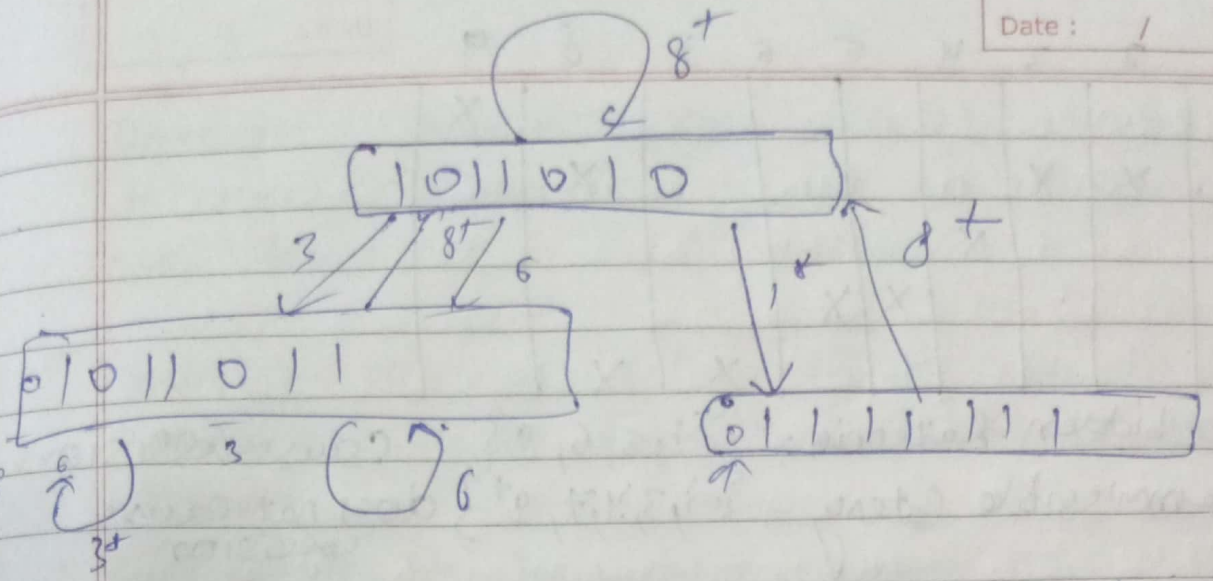
```
  1011010
+ 0000001
  1011011
```

next state = 1011011

with latency 8

```
  1011010
+ 0000000
  1011010
```

Next state = 1011011



Simple Cycle: It is a latency cycle in which each state appears only once.

Greedy cycle: A greedy cycle is a simple cycle whose edges are all made with minimum latency or minimum avg

Latency cycle	Types of cycle	Average latency
(3) edge	simple cycle const-	3 (MAL) greedy
minimum (6)	simple cycl 11	6
(8)	simple cycle 4	8
(1, 8)	11	4.5 greedy
(3, 8)	11	5
(6, 8)	11	7
(3, 6, 3)	--	
(3, 3, 6)	--	
(1, 8, 3, 8)	--	

generally there are 2 greedy cycles

MAL = 3

Question-2

	1	2	3	4	5	6	7	8	9
S1	X								X
S2		X	X					X	
S3				X					
S4					X	X			
S5							X	X	

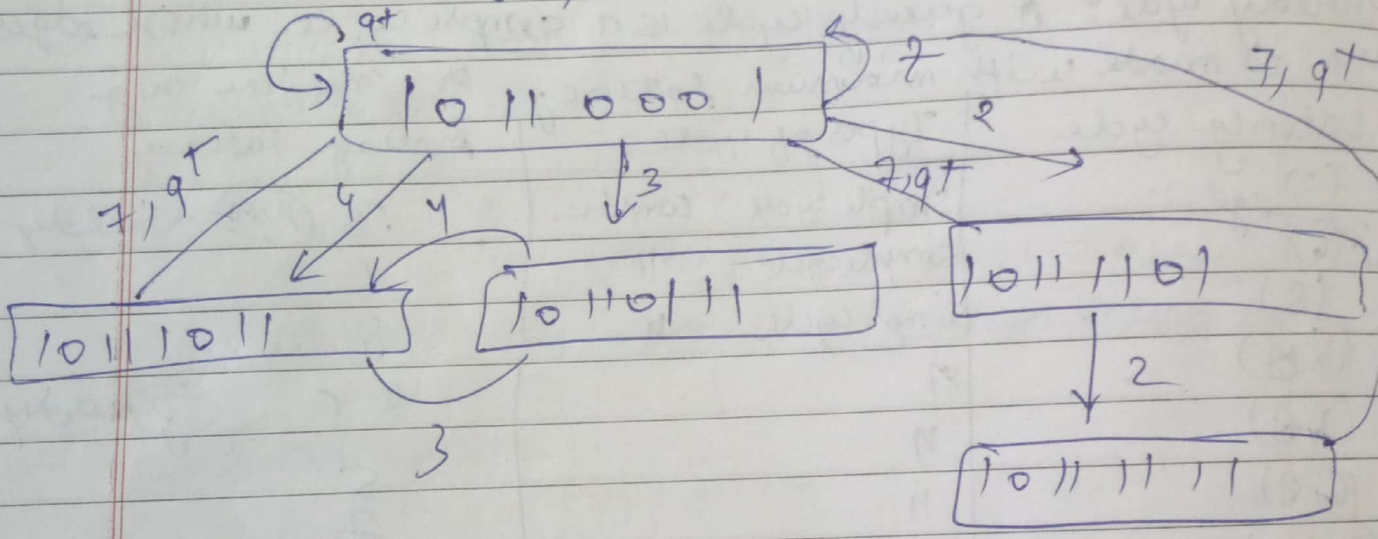
stages

forbidden latencies = {1, 5, 6, 8} cause collision

Permissible latency = {2, 3, 4, 7, 9+} does not cause collision

collision vector

: {C8 C7 C6 C5 C4 C3 C2 C1} = {1 0 1 1 0 0 0 1}



Throughput = $\frac{f}{mal} = \frac{1}{MAL * \tau}$

Bound.

Lower Bound on MAL

The maximum no of check marks in any row of the reservation table.

~~Lower~~ if Lower bound ≠ MAL then optimal latency not achieved

upper bound of MAL

The no of is in ICV + 1

Page :
Date : / /

Hazards: Circumstances that would cause incorrect execution if the next instruction was launched.

Structural hazards: Attempting to use the same hardware to do different things at the same time.

Data hazards: Instruction depend on result of prior instruction still in the pipeline.

Control hazards: Caused by delay between the fetching of instructions and decisions about changes in control flow (branches and jump)

Tomasulo's Algorithm.

→ It is a method of implementing dynamic scheduling.
Robert Tomasulo.

→ It remove name dependency through register renaming.

→ It tracks when operands are availabl to satisfy the data dependency.

AMDAHL Law

It is a law governing the speedup of using parallel processors on a problem, versus using only one serial processor, under the assumption that problem size remains same.

Speed up:

- Speed up metric is a quantitative measure of performance.
- Speed up is ratio of time it take to execute a program in serial to the time it takes to execute in parallel.

$$S(n) = \frac{\text{Time taken serially}}{\text{Time taken parallely}}$$

for example $T(1) = 1$ sec if $n = 2$ processor then

$$T(2) = \frac{1}{2} = 0.5 \text{ sec}$$

Hence $S(n) = \frac{T(1)}{T(2)} = \frac{1}{0.5} = 2$ mean speed up by 2 times.

$$S(n) \leq \frac{1}{f + \frac{1-f}{n}}$$

f = serial fraction.

$1-f$ = parallel fraction.

f → Serial Part

$\frac{1-f}{n}$ → Parallel Part

Numerical 1:

1. 70% is Parallel
   30% is serial
   maximum speedup ?
   8 CPUs

$$S(n) = \frac{1}{f + \frac{1-f}{n}} = \frac{1}{0.3 + \frac{0.7}{8}}$$

Page :
Date : / /

$$= \frac{8}{2.4+0.7} = \frac{8}{3.1} = \frac{80}{31}$$

$$= 2.6$$

Numericals:

Q-1 ~~Speed~~ 15000 instruction = n

$f = 25\ MHz$

$K = 5$

one instruction per clock cycle.

speed up = $\frac{\text{Time in non Parallel}}{\text{Time in parallel}}$

$$= \frac{15000 \times \tau}{(K+n-1)\tau}$$

$$= \frac{\text{no of instruction} \times \text{time for one inst}}{(\text{no of ins}) \times \text{time}}$$

$$= \frac{5000 \times 5\tau}{(K+n-1)\tau}$$

$$= \frac{15000 \times 5}{5+15000-1} = 4.999$$

Efficiency = $\frac{4.99}{5} = 0.99\%$

Throughput = $\frac{E_k}{\tau} = \frac{0.99}{25}$

$$= \frac{0.94 \times 10^6}{25}$$

$$= \frac{940000}{25}$$

$$= 0.94 \times 25 \times 10^6$$

$$= 23.5$$

PCR - Performance cost ratio =

$$= \frac{\text{Performance}}{\text{cost}}$$

$$= \frac{f}{c + kh}$$

where f = clock rate
c = cost of all logic gates
h = cost of k-stage pipeline.

$$PCR = \frac{f}{c + kh} =$$

$$(kh + c)\,PCR = f$$

$$kh\,PCR + c\,PCR = f$$

$$k = \frac{f - c\,PCR}{h\,PCR}$$

Q-

	1	2	3	4	5	6
S1	X					X
S2		X		X		
S3			X			
S4				X	X	

considering the reservation table

S1: {(6-1)} i.e {5}
S2: {(4-2)} i.e {2}
S3: { }
S4: {5-4} i.e {1}

forbidden latencies {1, 2, 5}
Permissible latency {3, 4, 6+}

initial collision vector

$C_x = \{C_4 C_3 C_2 C_1 = 1011$

100 11

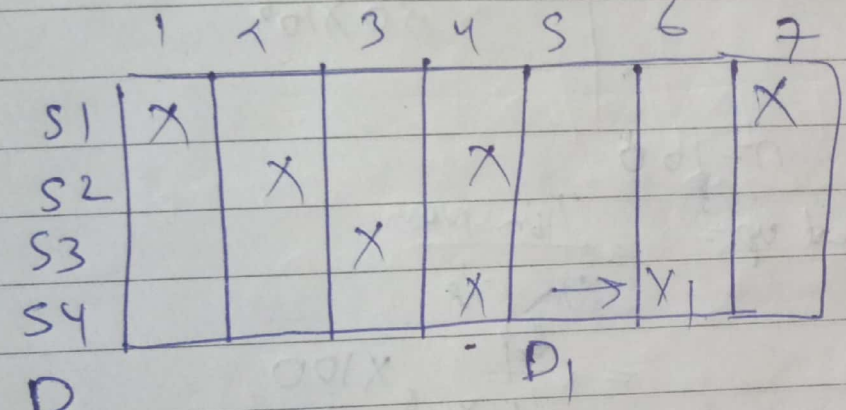
Page :
Date : / /

3rd shift 000|0
10011
10011

latency cycle	Type of cycle	Average latency
3	const	3 cycles
4	Long	4 cycles
6	ω	6

MAL = 3

To make 1 permissible we get



optimal constant latency = 2, which is equal to MAL.

c Maxi Throughput = $\frac{E_k}{\tau}$

$= \frac{SP}{k\tau}$

$= \frac{n/\tau}{(k+n-1)\times\tau}$

$= \frac{n}{\tau(k+n-1)}$

Hazards in Pipelining

Page :
Date : / /

→ Data Hazards ← RAW (TD)
WAR (AD)
WAW (DD)

→ Structural Hazards
When multiple instruction needs same resource

→ Control Hazards
All Instructio who change the program counter leads to control hazards.

Non-	Pipel
$f = 25$ MHz	$K = 5$
$CPI = 4$	$\tau = \frac{1}{20 \times 10^6}$

$n = 100$

speed up = $\frac{T \text{ in Non}}{T \text{ Pip}}$

$= \frac{4}{25 \times 10^6} \times 100$

$T_1 = \frac{(K+n-1)\tau}{} = \frac{K+(n-1)}{6f} = 5.2\,\mu s$

So that the speed up is

$S_K = \frac{T_x}{T_x} = \frac{16}{5.2} = 3.078$

Page :
Date : / /

PCR = Performance cost Rate = $\frac{\text{Performance}}{\text{cost}} = \frac{f}{c+kh}$

c = cost of digital gates
h = latch cost
k = no of stages
d = latch time
t = time taken by non pipeline / pipeline.

$$f = \frac{1}{d + \frac{t}{k}}$$

$$PCR = \frac{k}{dk+t} \times \frac{1}{c+kh}$$

$$\text{max } PCR = \frac{d}{dk}\left[\frac{k}{dk+t} \times \frac{1}{c+kh}\right]$$

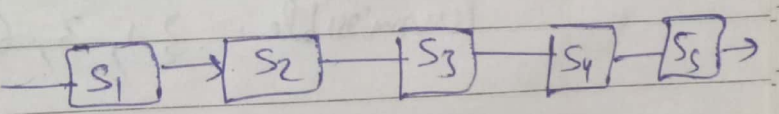
$$\text{freq} = \frac{1}{d + \frac{k}{k}}$$

$$k = \sqrt{\frac{ct}{dh}}$$

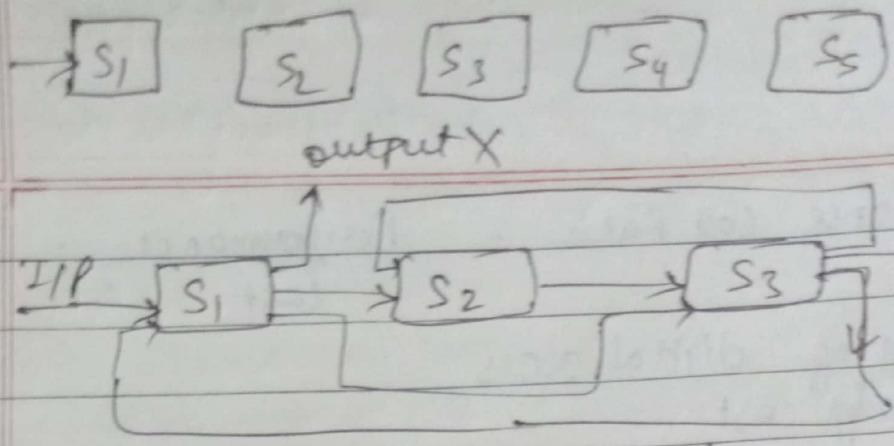
Non-Linear Pipeline

Reservation Table

	T_1	T_2	T_3	T_4	T_5
S_1	X	/			
S_2		X			
S_3			X		
S_4				X	
S_5					X



Linear Pipeline



	1	2	3	4	5	6	7	8
S_1	X					X		X
S_2		X		X				
S_3			X		X		X	

Reservation table for X

	1	2	3	4	5	6
S_1	X				X	
S_2			X			
S_3		X		X		X

Reservation table for Y

Permissible → Not cause collision
forbidden → cause collision

For X

Forbidden latencies → {2, 4, 5, 7}

$S_1 = \{5, 7, 2\}$

$S_2 = \{2\}$

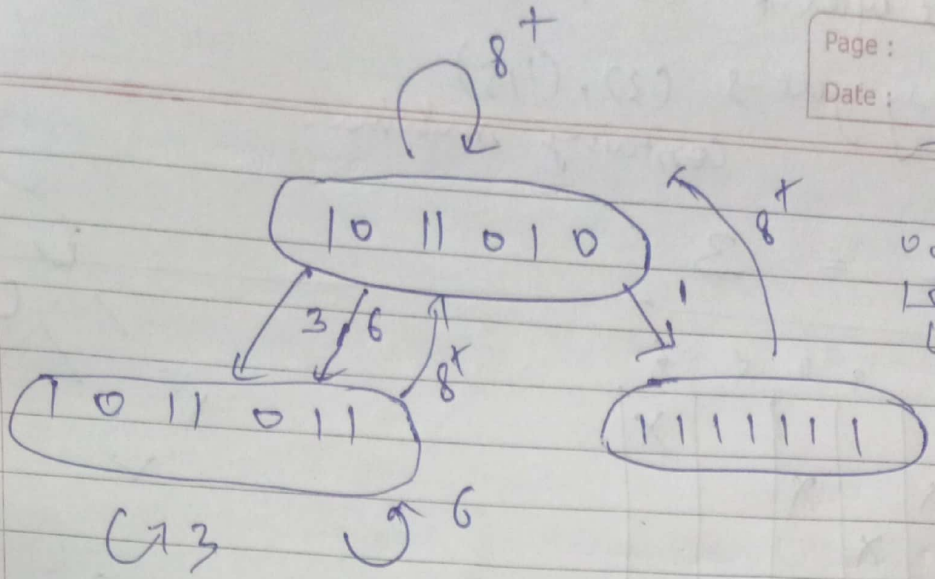
$S_3 = \{2, 4\}$

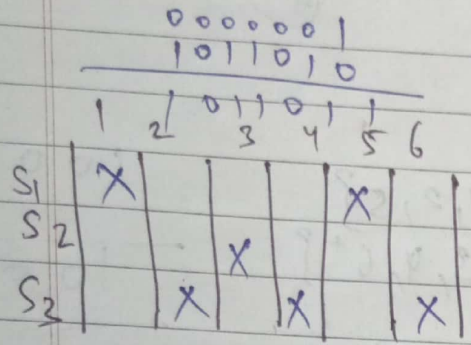
Permissible = {1, 3, 6, 8+}

- Initial collision vector.

7	6	5	4	3	2	1

ICV - 1 0 1 1 0 1 0





$S_1 = \{4\}$

$S_2 = \{\ ?\ \}$

$S_3 = \{2, 4\}$

Forbidden latencies: $\{2, 4\}$

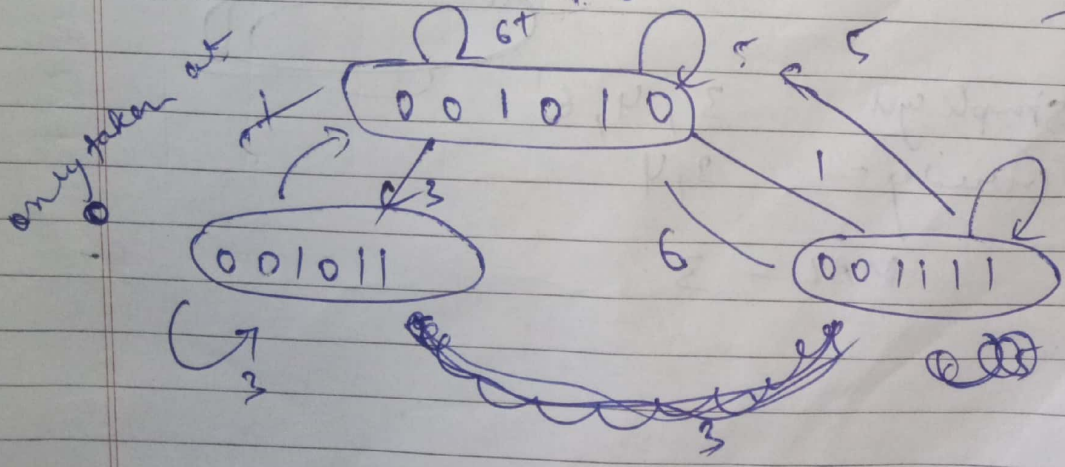
Permissible latencies: $\{1, 3, 5, 6^+\}$

maximum of Forbidden latencies.

Initial collision vector.

6 5 4 3 2 1

0 0 1 0 1 0



simple cycle :- (5), (3, 5) (1, 5) (3)

Greedy cycle → (3), (1, 5)
containing minima.

MAL = 3

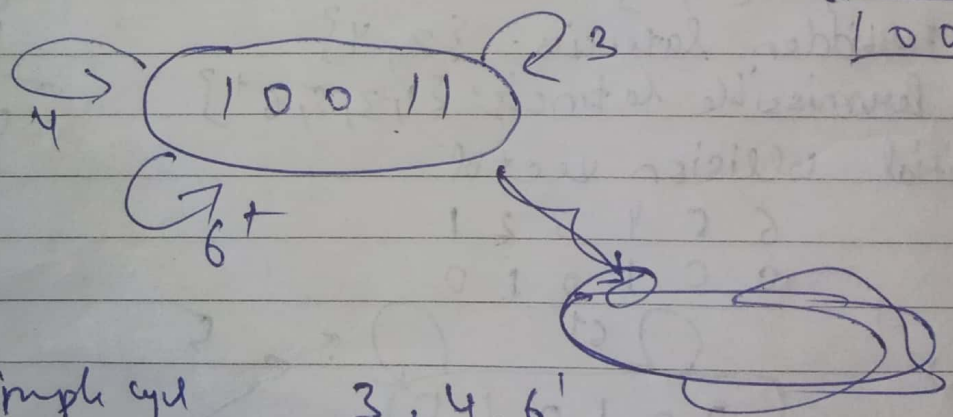
	1	2	3	4	5	6
S_1	X					X
S_2		X		X		
S_3			X			
S_4				X	X	

$S_1 = \{5\}$
$S_2 = \{2\}$
$S_3 = \{\}$
$S_4 = \{1\}$

Forbidden latency = {1, 2, 5}
Permissible latency = {3, 4, 6+}

Initial collision vector

5 4 3 2 1
1 0 0 1 1



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

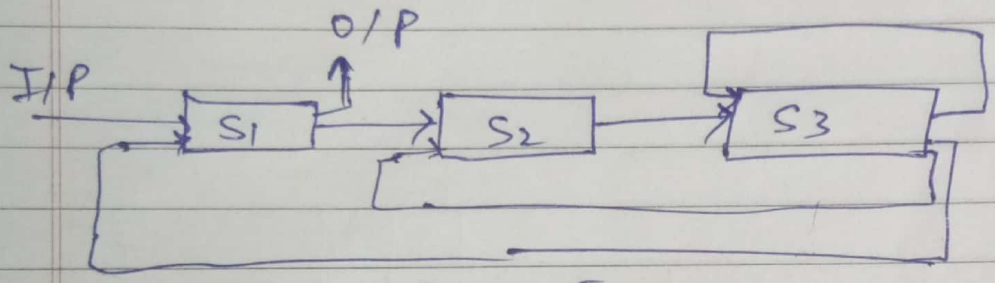
simple cycle 3, 4, 6+
greedy -- 3, 4

MAL = 3

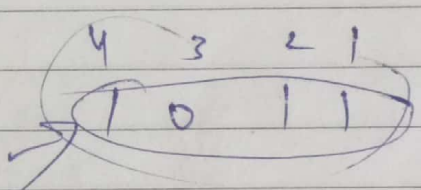
Bounds on MAL

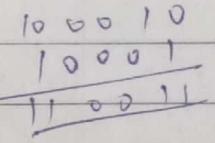
1. MAL is lower bounded by maximum no of checkmarks in any row of reservation table.
2. MAL is lower than or equal to average latency of any greedy cycle of state transition diagram.
3. upper bound of MAL is no of 1's in initial collision vector :

maximum no of checkmark $\leq$ MAL $\leq$ no of 1's in Initial collision vector + 1



	1	2	3	4	5
S1	X				X
S2		X		X	
S3			X	X	

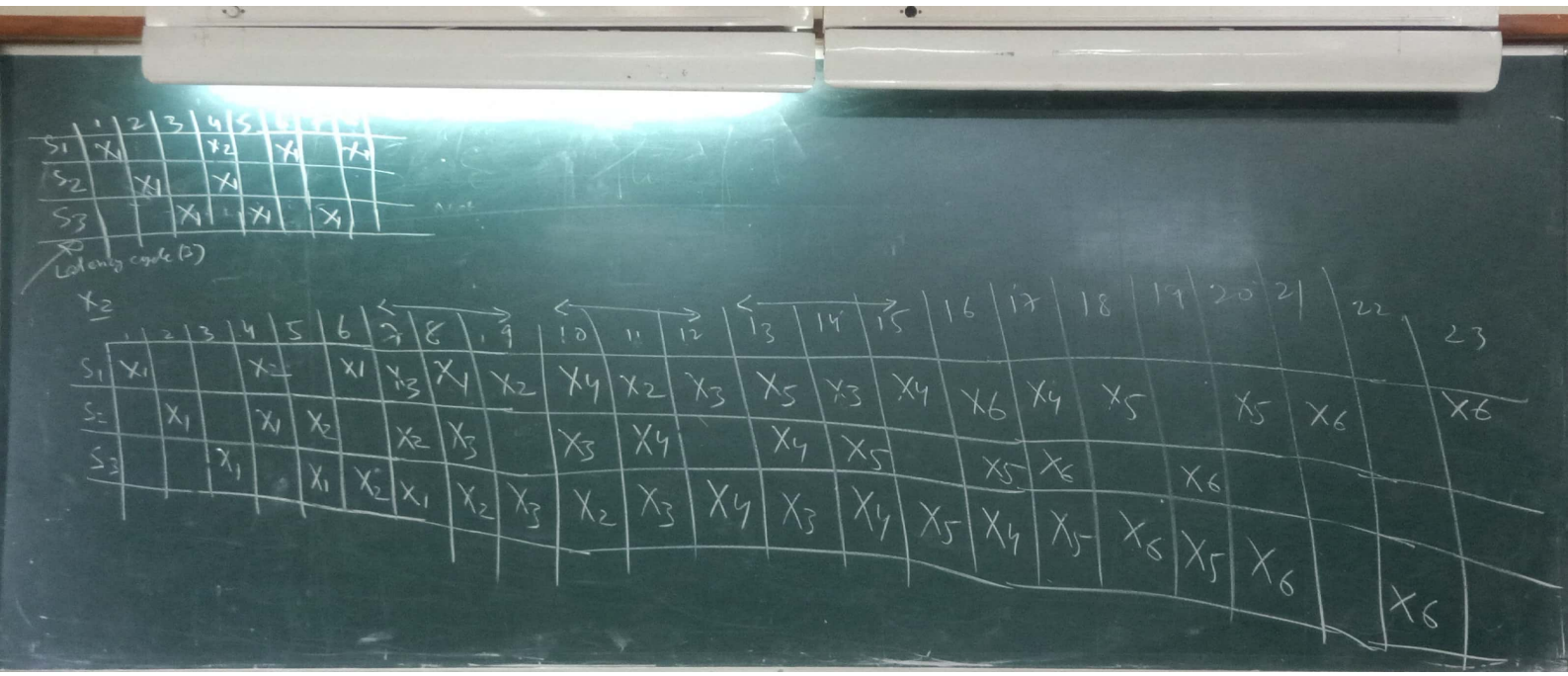




Simple cycle. 4 7 (4,5) (5,7), (3,4), (3,5), (1,3), (1,7), (1,3,7), (1,3,5,4)

Greedy ye (1,3)

MAL = $\frac{1+3}{2} = 2$



reduced execution

⇒ Fraction of time no enhancement is used =

$$\frac{1-(\alpha_1+\alpha_2+\cdots\alpha_n)}{1-(\alpha_1+\alpha_2+\cdots\alpha_n)+\frac{\alpha_1}{s_1}+\frac{\alpha_2}{s_2}+\cdots+\frac{\alpha_n}{s_n}}$$

ii) Suppose we double the speed ratio b/w the vector mode and the scaler mode by hardware improvement. Cal the effective speedup that can be achieved.

iii) Suppose the same speedup obtained in part (ii) must be obtained by compiler improvement instead of hardware improvement. What could be new vectorization ratio α that should be supported by the vectorizing compiler to achieve the same enhancement.

ii) $$\text{speedup} = \frac{1}{1-\alpha+\frac{\alpha}{s}}$$

$$= \frac{1}{1-0.75+\frac{0.75}{18}}$$

speedup = 3.43

Date
Page No.

Sol:- let no of instructions to be executed = k millions

	Parallel Mode	sequential Mode
MIPS →	αk	$(1-\alpha)k$

$$T = \frac{\alpha k}{mr} + \frac{(1-\alpha)k}{r}$$

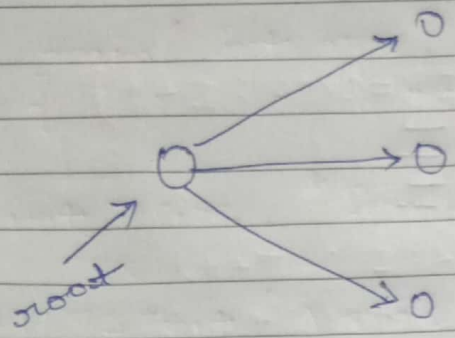
$$\boxed{T = \frac{k(\alpha + (1-\alpha)m)}{mr}}$$

$$\Rightarrow \text{Effective MIPS} = \frac{k}{\frac{k[\alpha + (1-\alpha)m]}{mr}}$$

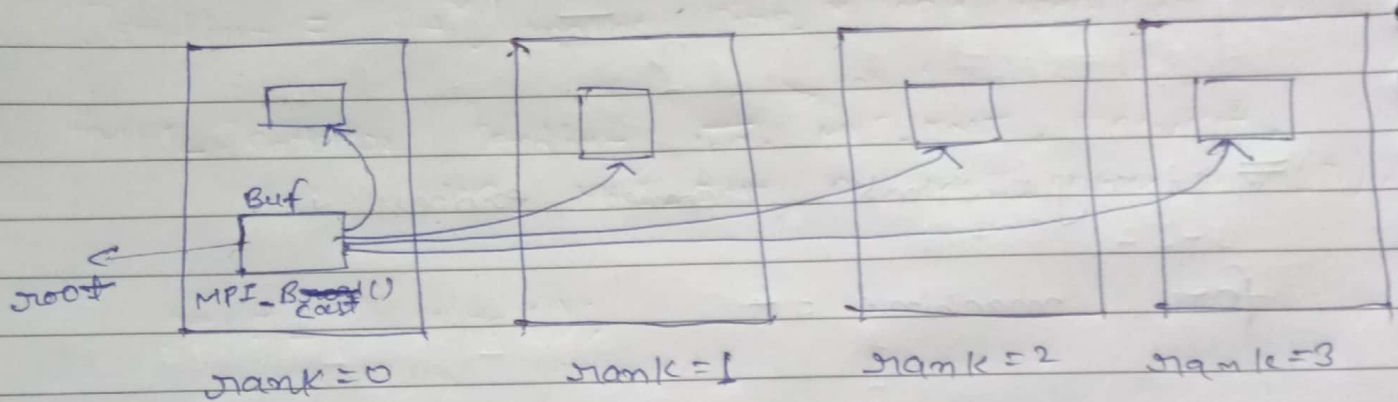
$$\boxed{\text{effective MIPS} = \frac{mr}{\alpha + (1-\alpha)m}}$$

☆ MPI Broadcast :-

Date .....



Syntax :



MPI_Bcast (*buf, count, datatype, root, communicator)

→ root : who is broadcasting.

ex matrix A[ ]nxn send to all?

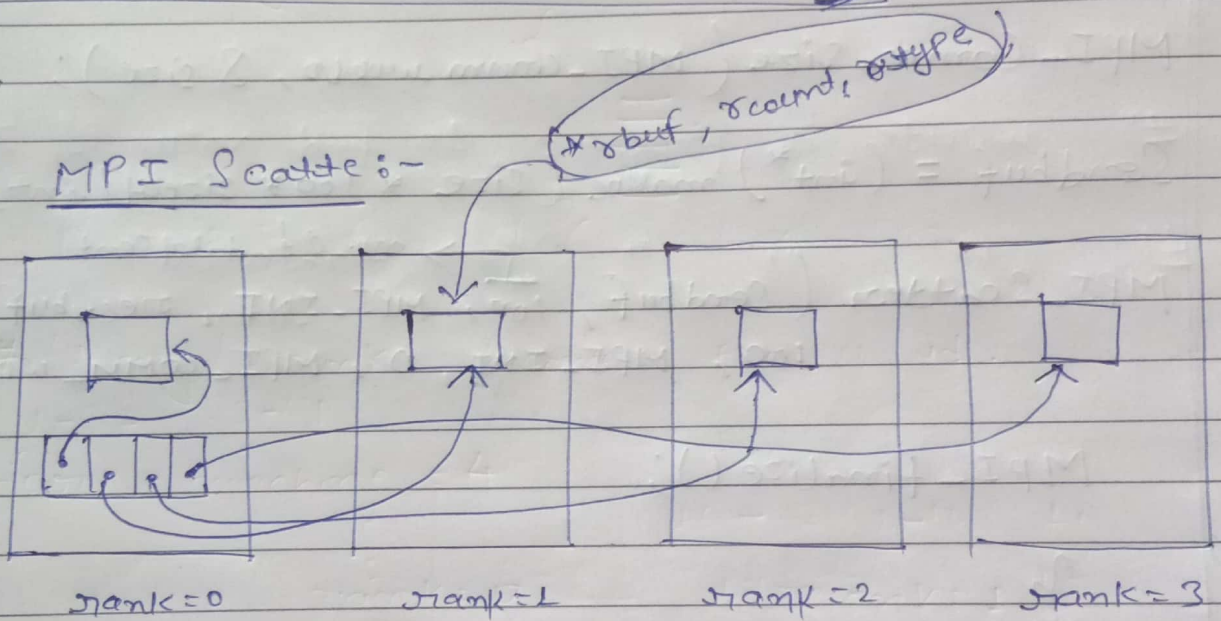
```
double A[N][N];
MPI_BCast(A, N*N, MPI_DOUBLE, 0, MPI_COMM_WORLD);
```

Spiral

```
double A[N][N];
MPI_COMM_rank (MPI_COMM_WORLD, &rank).....
if (rank == 0)
{
    for (i=0; i<N; i++)
        for(j=0; j<N; j++)
            A[i][j] = i+j;
}
```

Date

MPI Scatter :-



Syntax :

MPI_Scatter(*Sbuf, Scount, Stype, *rbuf, rcount, rtype, root, communicator)

ex

Date .....

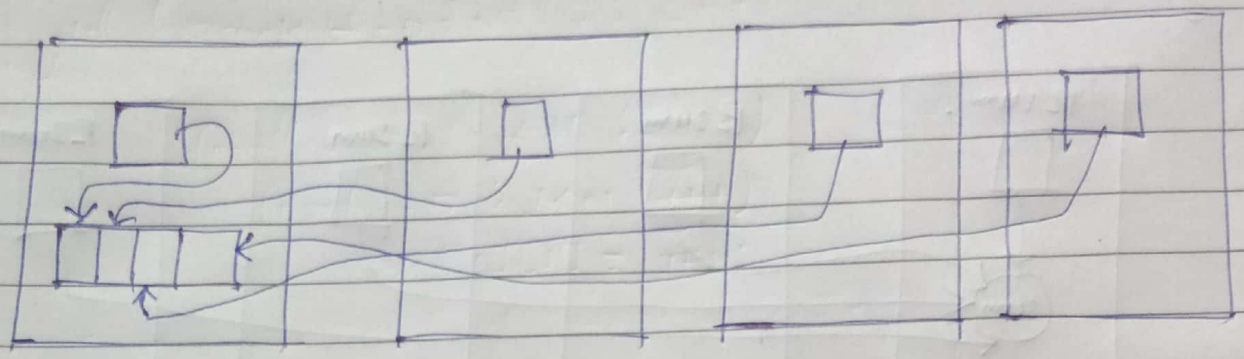
ex:-

```
main( int argc, char *argv[])
{
int size, *sendbuf, recvbuf[100];
MPI_Init( &argc, &argv);
MPI_Comm_size( MPI_COMM_WORLD, &size);
Sendbuf = (int*) malloc( size * 100 * sizeof(int));
MPI_Scatter( sendbuf, 100, MPI_INT, recvbuf,
             100, MPI_INT, 0, MPI_COMM_WORLD);
MPI_finalize();
}
```

→ no. of data sent

Gather :- → opposite of Scatter

Date .....



ram

Syntax :-

MPI_Gather (*Sbuf, Scount, stype, *rbuf, rcount, rtype, root, comm);

for program replace Scatter by Gather.

Reduce :-

Syntax :-

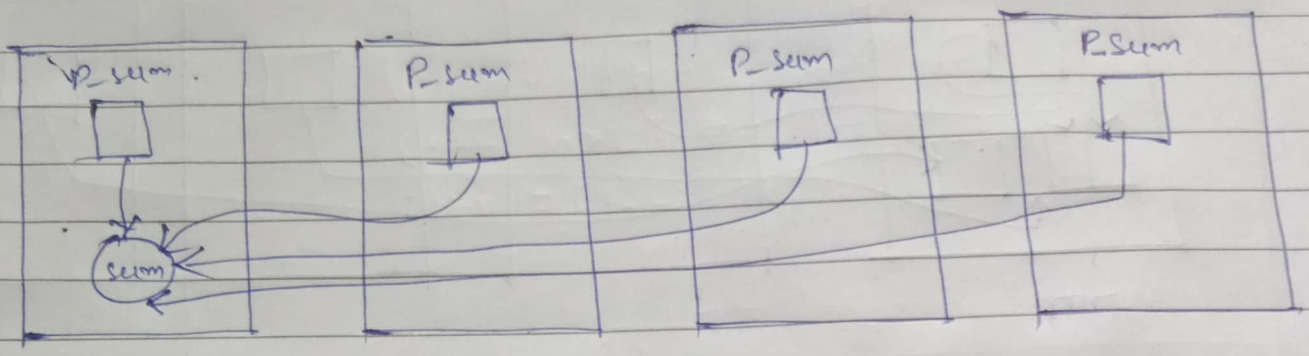
MPI_Reduce (*Sbuf, *recivebuf, count, datatype, op, root, Comm);

Op = operations → MPI_MAX
MPI_MIN
MPI_SUM
MPI_PROD

MPI.org

Date .....

esl



write a program to find value of π.

" " " of factorial.

Sum

★ OpenMP :-

Date .....

→ Thread based

```
# include <omp.h>
# pragma omp parallel
  {
  ——
  ——
  ——
  ——
  ——
  }
```

gcc: -fopenmp file_name

```
# include <omp.h>
# include <stdio.h>
  int main ( int argc , char *argv [] )
  {
    # Pragma omp parallel Private (threadid)
    { Threadid = Ompset_thread_num();
    prinf( "Hello world from thread id %d of
          %d \n", Omp_get_thread_num(),
                  omp_get_num_threads() );
```

id

no. of threads

```
    }
  }
```

Spiral

Date .....

```
int thread, size;
thread = omp_get_thread_num();
size = omp_get_num_threads();
```

Thread id

no. of threads

```
Threadid = ompset_thread_num();
size = ompset_num_threads();
```

Book # Prallell programming in C openmp using Quinn

```
# pragma omp parallel Shared(a,b,c,d, size) private
                      (i, thread id)
{
  if (threadid == 0)
  {
     a = 50;
  }
}
```

Date .....

```
# Pragma omp parallel
# Pragma omp for
    for (i=0; i<=n; i++)
    {
    }
```

→ parallelize Threads.

```
int thread_id, size, a[100], b[100], c[100];
Threadid = omp_set_thread_num();
size = omp_set_num_threads();
for (i = Threadid; i <= (Threadid+1)*size; i++)
{
    a[i] = b[i] + c[i]
}
```

0	1	2	3	for 4 threads.
4	5	6	7	
8	9	10	11	
12				
T_0	T_1	T_2	T_3	

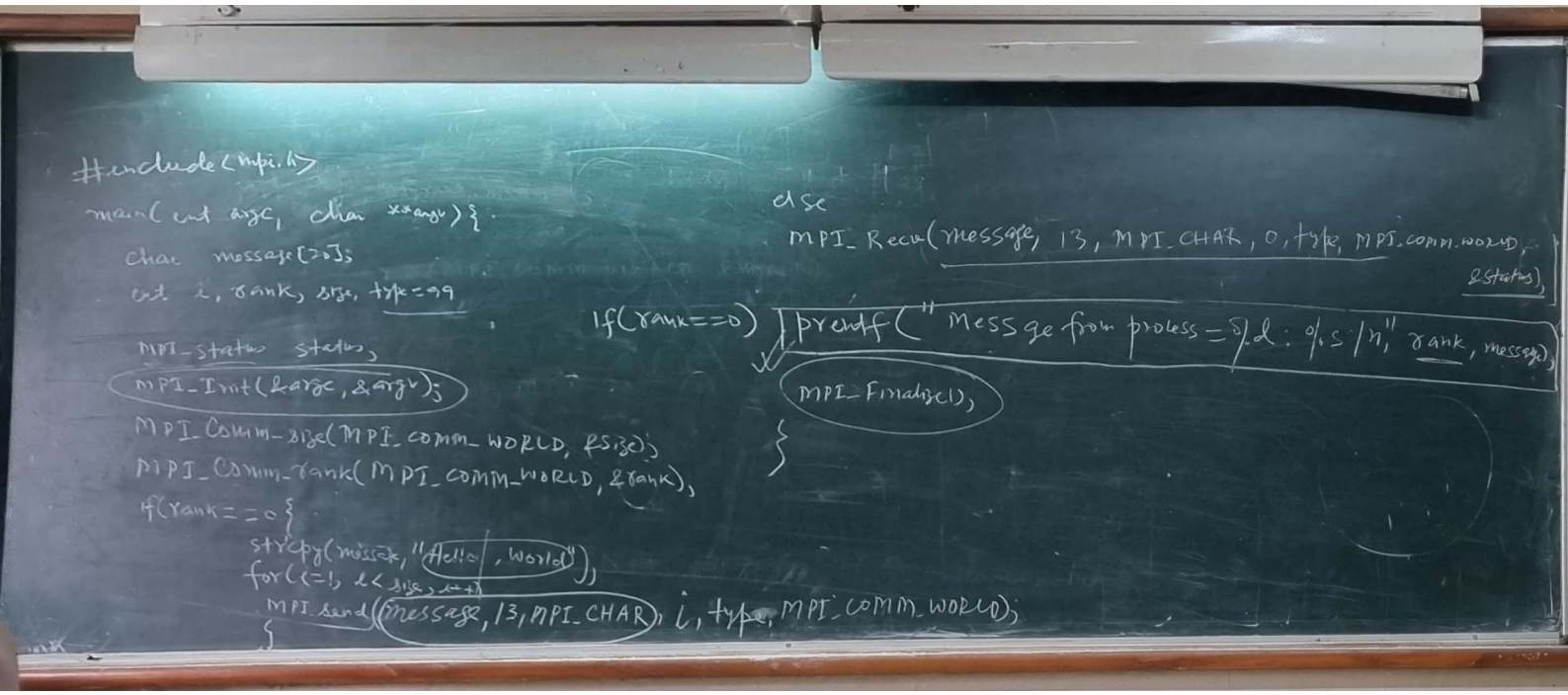
Thread

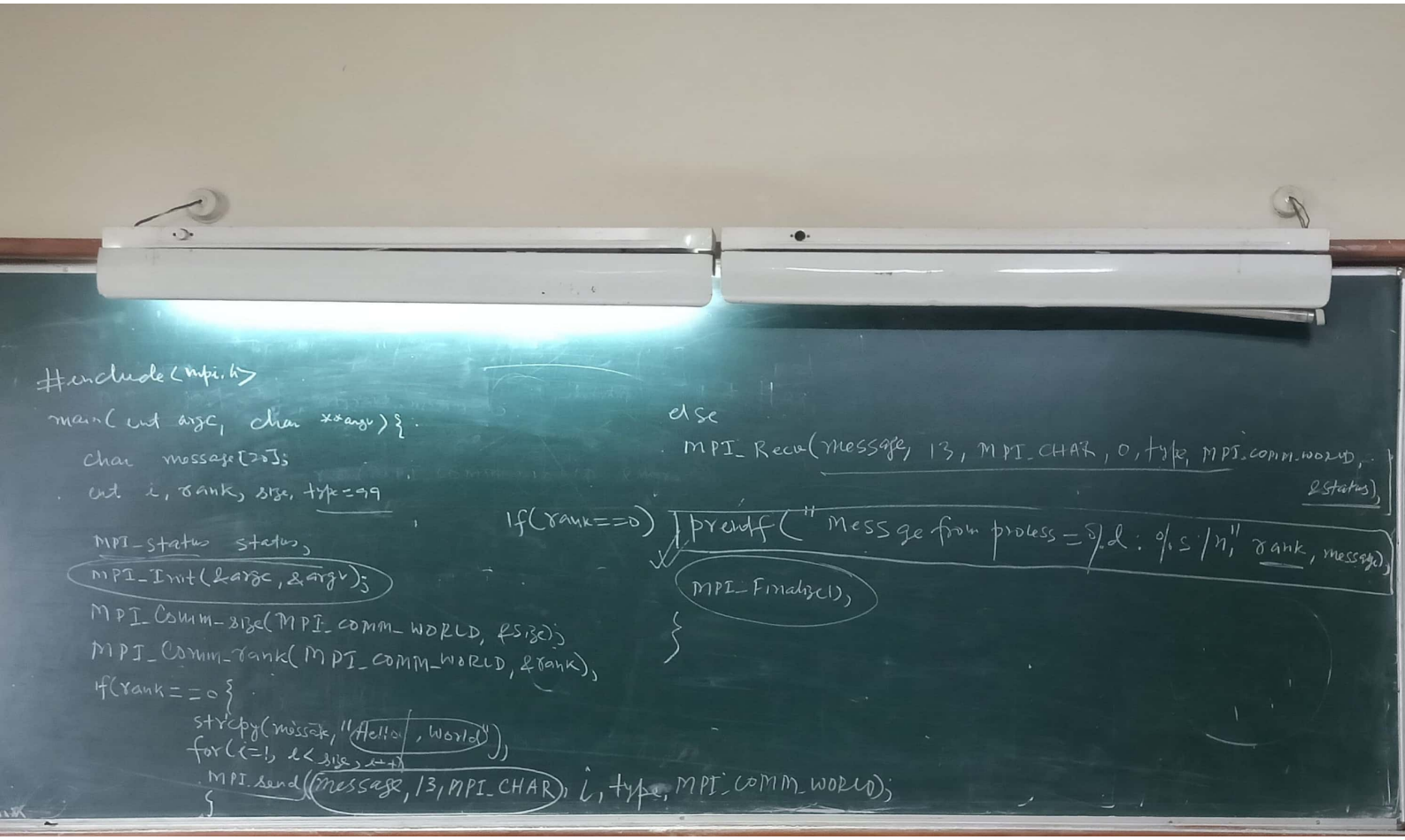
Spiral

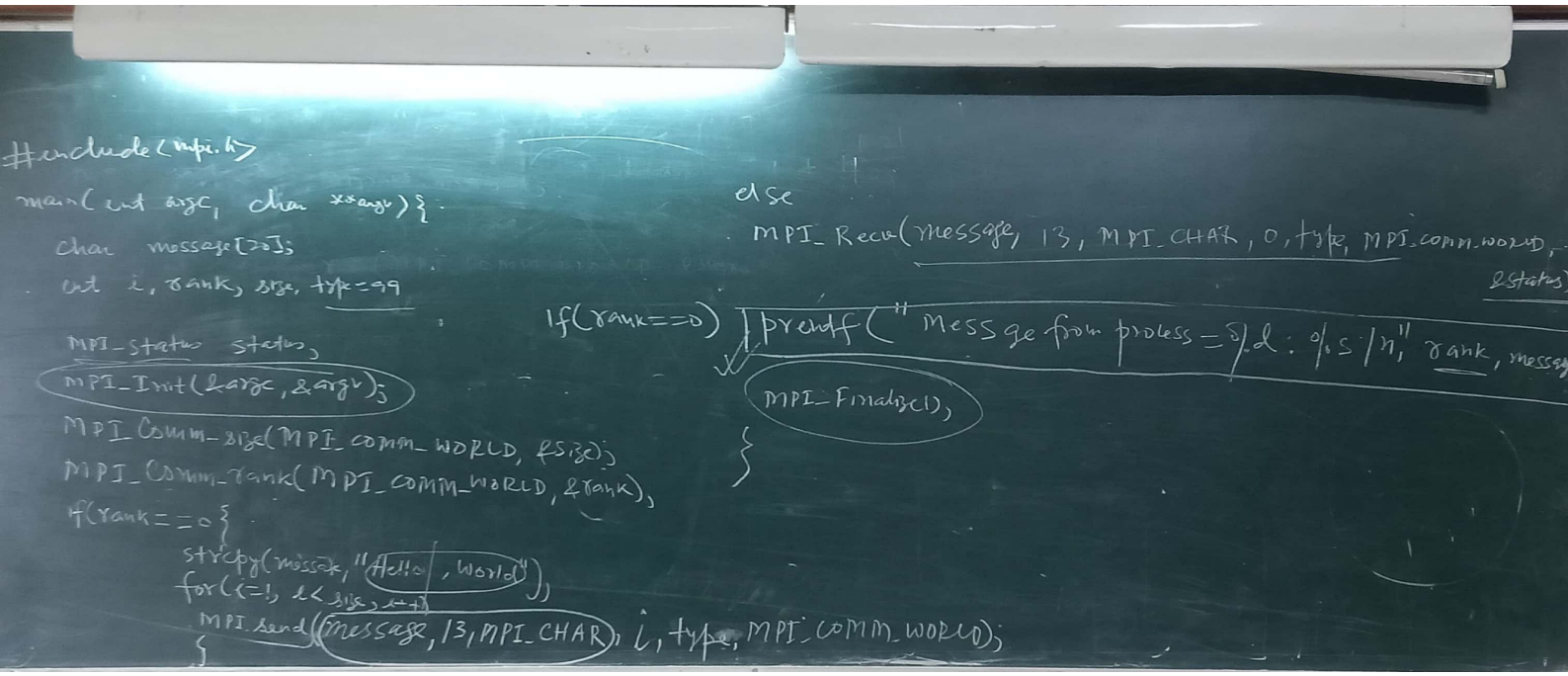
Date .....

Thread	0	1	2	3
	0-24	25-49	50-74	75-99

```
for( i = Threadid *25 ; i< (threadid+1)*25 ; i++)
  {
    a[i] = b[i] + c[i]
  }
```







```
int x, y, myrank, msgtag=1;

MPI_Comm_rank(MPI_COMM_WORLD, &myrank);

if(myrank==0)
    MPI_Send(&x, 1, MPI_INT, 1, msgtag, MPI_COMM_WORLD);
else
  if(myrank==1)
    MPI_Recv(&y, 1, MPI_INT, 0, msgtag, MPI_COMM_WORLD, status);
```

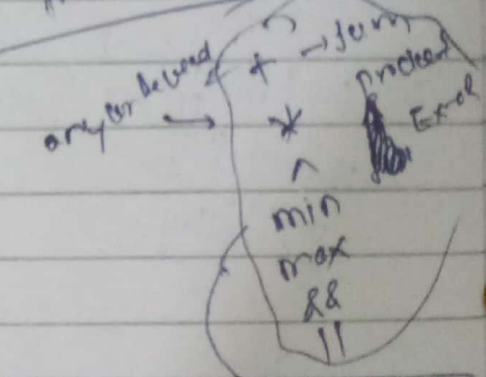
Book:

Programming Massively Parallel processors

by David B kirik ↑ (CUDA)

OpenMP . IMP

only three from Unit-4



Pragma omp barriers

- Reduction clause in omp.

Syntax:

```
# Pragma omp parallel for reduction (+ : sum)
    for (k = 0 ; k < 100; k++)
        {
            sum = sum + funct(k);
        }
```

This will parallel execute k = 0 to 1 no (or devides) iteration equally to each thread & then partial sum of each thread is ~~reduction~~ added (+) & final sum will be kept in each ~~proces~~ thread.