All questions carry equal marks

Important: Write your answers **serially one after the another** in the answer sheet that is 1,2,3,4 ... ...15. Do not use random ordering while answering.

1. Consider a general case where a pipeline of depth "m" is executing "N" independent subsequent operations.

   (a) What is the expected speedup in this case compared to a general purpose unit that requires "m" cycles to generate a single result. Also write the speedup for the case when "N" is very large compared to "m".

   (b) What is the throughput when "N" is very large compared to "m".

2. Consider the following sparse matrix; represent it in CSR (Compressed Sparse Row) storage format.

   3 0 1 0
   0 0 0 0
   0 2 4 1
   1 0 0 1

3. (a) What is a structured block. Is branching allowed inside a structured block.
   (b) A CS algorithm takes 1 hour to execute. Name a profiling tool which you would use to find out the bottleneck of this algorithm before parallelizing it?
   (c) Consider a parallel system for which overhead $To = p^{3/2} + p^{3/4} W^{3/4}$ (depends on both problem size W and number of processors p). To ensure that the efficiency does not decrease as the number of processing elements increases, at what rate should the problem size grow?

4. A program has a main loop that is perfectly parallel and comprises 80% of program execution time. How many threads (or processors) would you need to achieve a 4x speedup for the parallelized version relative to the original sequential/serial version? Ignore the overheads of parallel execution (thread creation, scheduling, synchronization, coherence, etc). Show your work (not just the number).

5. No marks without explanation for the following two questions:
   (i) Consider the following code, which we want to parallelize.

```
do i = 1, n
a [2*i] = b[i] + c[i]
d[i] = a [2*i+1]
enddo
```

Is parallelization possible? Explain briefly your "Yes" or "No".

(ii) Can we parallelize the following loop? Yes or No, Explain/ Show.

```
a[0] = 0;
for( i = 1; i < n; i++)
a[i] = a[i-1] + i;
```

6. (a) Cost of adding "n" numbers with "n" processors is "n log n" (**not cost optimal**) and the overall parallel execution time of this parallel system is $O((n/p) \log p + n/p)$ when using "p" processors. What is the **parallel runtime** of **cost-optimal** algorithm for adding "*n*" numbers using "*p*" processing elements.

   (b) (i) In general, for a computational problem of fixed size, the efficiency of a parallel computation typically _____ as the no. of processors increase.

   (ii) The rate at which data can be transferred from memory to the CPU is known as _____.

   (iii) Property that if a program accesses a memory location, there is a much higher than random Probability that the same location would be accessed again: _____.

   (iv) Generally Heap and Stack is stored in – (i) RAM, (ii)cache, (iii)register.

7. What is the read throughput (memory bandwidth) of a memory system with 64 bit DDRAM interface (DD stands for double data rate), 2 channels, 0.5 GHz DRAM clock frequency. (Write your answer in Bytes/Second; No marks for partially correct answer, show your work clearly.)

8. For each part (a,b and c) below, assume:

   - a 16 KB L1 data cache and a 1MB L2 cache that are both initially empty;
   - 128 B cache blocks;
   - an L1 miss takes 10 cycles if the access then hits the L2 cache and;
   - 100 cycles if it then misses the L2 cache as well;
   - assume that the array elements are each 4B integers.
   - ignore other effects;
   - Times are in cycles instead of seconds.

```
for (int i=0;i<N;i++)
        for (int j=0;j<N;j++)
                ... = a[j];
```

   a) Assuming $N = 2^{12}$, what is the total miss cycles for the entire code?
   b) Assuming $N = 2^{17}$, what is the total miss cycles for the entire code?
   c) Assuming $N = 2^{20}$, what is the total miss cycles for the entire code?

9. We must optimize the following variant of matrix multiply code to target a certain machine (specs given below) :

// assume A[N][N], B[N][N], C[N][N], and D[N][N] have integer elements

```
for (i=0;i<N;i++){
for (j=0;j<N;j++){
for (k=0;k<N;k++){
A[i][j] += B[i][k] * C[k][j] + D[i][j];
}
}
}
```

MPI_Init()
Int rank = MPI_get_
if (rank) = 0)
MPI_send (&x, 1, int,
1,

Target Machine Specs:

- 16 KB L1 Cache
- 4 MB L2 Cache
- 64 B cache blocks (for both caches)

Suppose that **blocking** (or tiling) is the way to go. Considering tiling as the **only** optimization, and also considering only **square tiles**, what should the **tile dimension T** (assuming TxT tile size) to optimize memory performance for **N=512** (array dimension) for the code?

Give the value of T as both the number of elements and the number of bytes. Show the work you did to compute the tile dimension T and explain in one sentence why this choice should work well. **You do NOT have to show the tiled code.**
NOTE: assuming that each tile size you choose would be theoretically the best.

10. (a) OpenMP is composed of the following main components:

I. *Directives*
II. *Runtime library routines*
III. *Environment variables*

Under which of the above three categories the following falls:

1. omp_get_num_threads
2. omp_get_thread_num
3. Omp_schedule
4. Critical
5. Omp_num_threads

(b) When you use dynamic loop scheduling, which **two are true**?

I. All threads will finish executing the loop at the same time.
II. The OpenMP runtime manages allocation of work to the threads.
III. The default chunk size is implementation defined.
IV. The default chunk size is 1.
V. The threads execute the same number of loop iterations.

11. Optimize the cache performance of the following code. Do not parallelize it. Assume that N and M are extremely large powers of 2. Assume that A[], B[], and C[] are integer arrays, and that an int is 4B.

```
for (i=0;i<N;i++)
        for (j=0;j<M;j++)
                A[i] += B[j][j] * C[i];
```

12. A compiler is able to speed up a program by 1.5384 times (note that 1.5384 ~= 1/0.65). The compiler has an optimization that applies to **function1** of the program that speeds **function1 up by 4x**, and an optimization that applies to **function2** of the program that speeds **function2 up by 2x**.

If function2 comprises 40% of the execution time of the program before any optimization, **what percentage of execution time is function1 before any optimization**? Assume that no other part of the program is optimized or speed up.

13. (a) (i) Suppose we denote the serial runtime by *TS* and the parallel runtime by *TP*. Suppose there are "p" processors. The overhead function (*To*) is given by

(ii) Consider the problem of adding "*n*" numbers by using "*n*" processing elements (partial sums up a logical binary tree of processing elements). Analytical speedup, efficiency and the cost of the algorithm will be given by:

(b) Write a pseudocode (around 6-7 lines required) to send an integer "x" from process 0 to process 1 assuming MPI_Comm_rank has already generated the process ranks.

14. (a) True or False for the following statements:

(i) The isoefficiency function does not exist for unscalable parallel systems.
(ii) W(*p*) is the asymptotic lower bound on the isoefficiency function (where p is the number of processing elements).
(iii) In a parallel algorithm, cost is always at least the work.
(iv) In MPI, Non-blocking operations are generally accompanied by a **check-status** operation, which indicates whether the semantics of a previously initiated transfer may be violated or not.

(b) Can we improve the code-balance (computational intensity) in the following situation (a two point stencil)? Support your answer with proper explanation/ modifying the code. (No marks for "yes" or "no" answer)

```
do i=1,n-1
b(i) = a(i) + s * a(i+1)
enddo
```

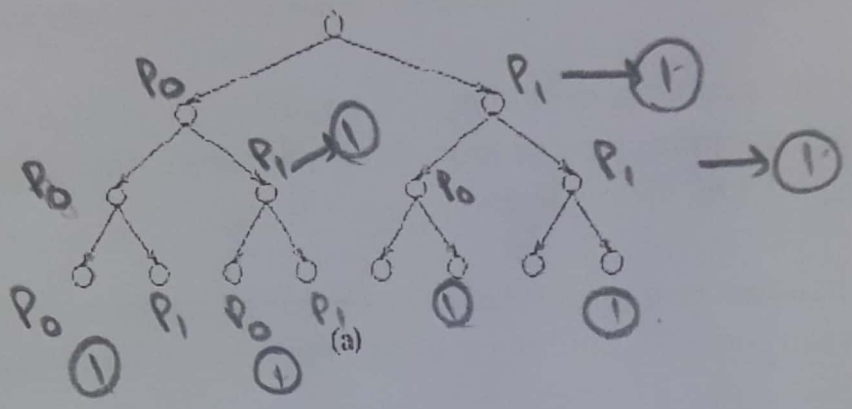
15. Read carefully and answer the questions:

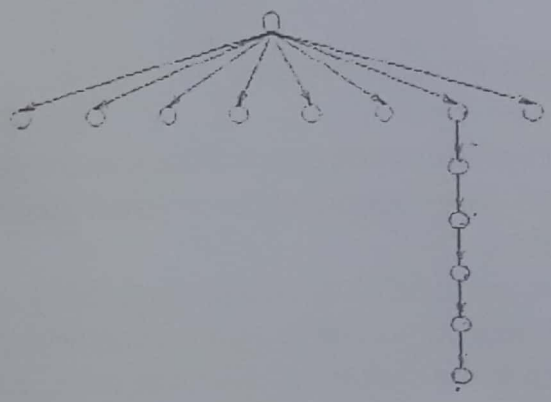
As we have learned, tasks are programmer-defined units of computation into which the main computation is subdivided by means of decomposition and an abstraction used to express dependencies among tasks and their relative order of execution is known as a ***task dependency graph.*** Each circle in the figure below represents a task.

The number and size of tasks into which a problem is decomposed determines the ***granularity*** of the decomposition.

For the two task graphs given in Figure below, determine the following:

1. Maximum degree of concurrency.
2. Critical Path length or span or depth of the computation.
3. Maximum achievable speedup over one process (processor) assuming that an arbitrarily large number of processes (processors) is available.
4. The minimum number of processes (processors) needed to obtain the maximum possible speedup.
5. The maximum achievable speedup if the number of processes (processors) is limited to (a) 2, and (b) 8.





(b)

'LEDGE

hereby Pledge that I shall answer the questions at the examination to the best of my ability and that I shall not resort
ny unfair means of any nature.

Supervisor's Signature : ..... Student's Signature: Nishi

Course : CS301 Name : Nishi Doshi

Date : 13/3/19 Student Identity Number : 201601408

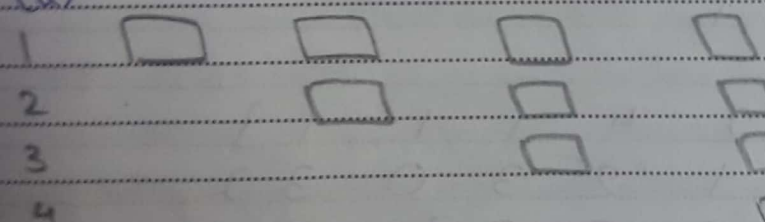
Number of Supplementaries : 1 Programme : B. Tech (ICT + CS) 2016

Questions	1	2	3	4	5	6	7	8	9	10	TOT
Marks Obtained											
Questions	11	12	13	14	15	16	17	18	19	20	
Marks Obtained											

Examiner's Signature

Start here

(a)



From the pipeline of depth 'm', one can see that a 1 operation gets completed after time 'm' and then subsequently for every clock cycle, one operation completion is there.

Hence, initial m-1 ideal time where no operation is completed and then N time units where on every time unit 1 operation is completed equals total amount of time = N + m - 1

Speedup = $\frac{\text{Without pipelining time required}}{\text{With pipelining time required}}$

$= \frac{Nm}{N+m-1}$

When $N >> m$; Speedup $\cong \frac{Nm}{N} \cong m$

b) Throughput $= \frac{N}{N+m-1}$

when $N >> m$; throughput $\cong \frac{N}{N} \cong 1$

2. CSR

3	0	1	0
0	0	0	0
0	2	4	1
1	0	0	1

Data array [3 1 2 4 1 1 1 1]

Column array [0 2 1 2 3 0 3]

Row array [0 2 2 5 7]

∴ Number of rows in Matrix = 5 - 1 = 4

→ Represents non-zero data present in matrix

→ Represents the column where that data element is stored.

→ Represents the start of the positioned row.

a) Structured block

- A structured block is a sectional block which has one point of entry and one point of exit.
- Branching is not allowed inside a structured block. (The reason why branching is not allowed is that it may result the thread not returning and hence will violate the definition of structured block.)
- Thus, main characteristic of a structured block is that it has one point of entry and one point of exit.

3. b) The profiling tool gprof will be used to find the bottleneck of the algorithm before parallelizing it.

3. c) We want to ensure constant efficiency

∴ ~~W = K T_o~~

We know that $W \propto T_o$ } The rate at which problem size show change is directly proportional to T_o

Hence $W' = K T_o$

$T_o = p^{3/2} + p^{3/4} W^{3/4}$

As T_o is a complex function; we can break it into seperate terms and analyze

∴ $T_o = p^{3/2}$

$W = K T_o$

∴ $W = K p^{3/2}$

↓

Here we see problem size changes as $p^{3/2}$

$T_o = p^{3/4} W^{3/4}$

$W = K T_o$

∴ $W = K p^{3/4} W^{3/4}$

∴ $W^{1/4} = K p^{3/4}$

∴ $W = K p^3$

↓

Here we see problem size changes as p^3

↓

Hence, this is a dominating factor.

$p^3 > p^{3/2}$ (As $p \geq 1$)

Thus, the problem size should grow at p^3 rate.

4. Speedup to be achieved = 4 times
Parallel time execution = 0.8 t
Total time for execution = t
Serial time execution = 0.2 t

Serial fraction of the code $s = \frac{0.2t}{t} = 0.2$

Speedup $\Psi \leq p + (1-p)s$
where p is number of processors
s is serial fraction for parallel code

$\therefore \quad 4 \leq p + (1-p)(0.2)$
$4 \leq p + 0.2 - 0.2p$
$\therefore \quad 4 \leq 0.8p + 0.2$
$\therefore \quad 3.8 \leq 0.8p$
$\therefore p \geq 4.75$

As p should be integer; 5 processors should be used.

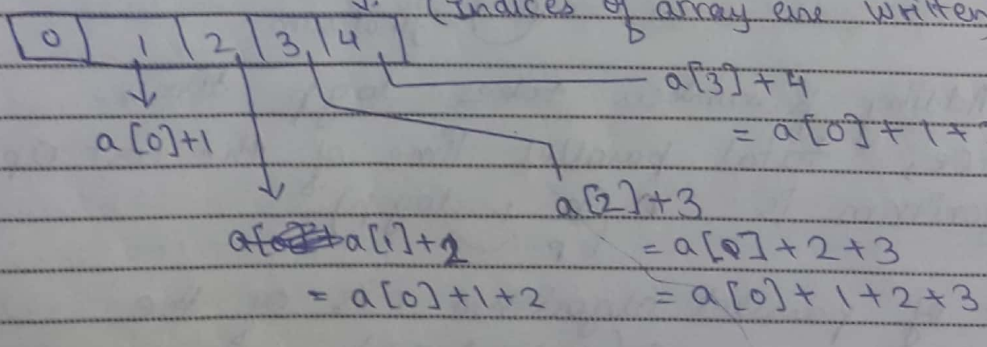
The reason for using Gustafson's Barsis law and not Amdalh's law is that we have knowledge of execution time of parallelized code.

5. (i) Yes, parallelization is possible.
The loop shown has no dependency carried on it.

```
do i=1,n
   a[2i] = b[i] + c[i]   ⟶ This line access all even elements of array.
   d[i] = a[2i+1]        ⟶ This line access all odd elements of array.
enddo
```

Hence, no dependency between two statements.
Thus, parallelization is possible.

(ii) No, parallelization of the loop is not possible.
Reason, being every element of array is dependent on the previous element.
Thus, till the time, previous element in array is not computed; the current value cannot be computed.



(iii) Yes, the loop can be parallelized.
As we can see; every element a[i] can be written using general formula

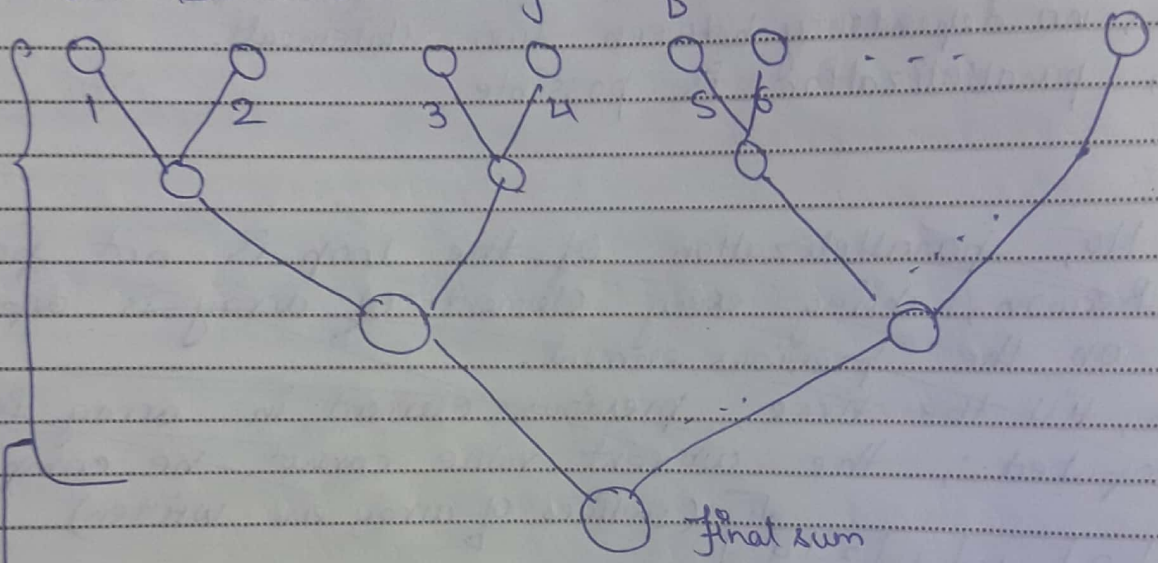
$$a[i] = a[0] + \frac{(i)(i+1)}{2} \qquad \left(\because \sum n = \frac{n(n+1)}{2}\right)$$

$a[0] = 0$; Hence $a[i] = \frac{i(i+1)}{2}$

Thus, parallelizing the loop is easier as no element of array is dependent on any other. The value is calculated by a unique formula derived above.

6. a) Parallel runtime of cost optimal algorithm for adding n numbers using p processing elements.

There are p processing elements.
Divide all the numbers into p processors. Each
Every processor calculates sum for n/p elements.
At the end we have p numbers who need
to be summed to get final result.



→ Adding p numbers takes log p time.
Hence, total parallel time of the cost optimal algorithm is $O\left(\frac{n}{p} + \log p\right)$

Cost of parallel algorithm is of the
$O(n + p \log p)$

b) (i) Efficiency decreases as problem size processors increases for a fixed problem size.

(ii) Latency

(iii) Temporal locality

(iv) (i) RAM

7. 2 channels 0.5 GHz DRAM clock frequency
64 bit DDRAM

Hence; Throughput = (Clock frequency) × (No. of channels)
× (Size of DDRAM) × 2 (Double Data Rate) ← for DDRAM

= $0.5 \times 10^9 \times 2 \times 64 \times 2$
= 128×10^9 bits/sec
= 16 GBytes/sec.

8. a) ~~N = 2^12~~

16KB L1 data cache

$\frac{128}{4}$ = 32 integers in 1 cache blocks.

No. of cache blocks in L1 = $\frac{16 \times 2^{10}}{128} = 128$

No. of cache blocks in L2 = $\frac{2^{20}}{128} = 8192$

a) $N = 2^{12}$

We can store $32 \times 128 = 2^5 2^7 = 2^{12}$ integers in L1 cache.

At a time 32 integers will be loaded in L1 cache.

∴ No. of Misses ~~cycles~~ = $\frac{2^{12}}{32} = \frac{2^{12}}{2^5} = 2^7$

Miss cycle = $2^7 \times 10$
= 1280 miss cycles

As the miss are found only when first loop runs; afterwards all data stored in L1 cache and all hits. ⇒ No more miss cycles.

b) $N = 2^{17}$

No. of integers in L2 cache = 2^{19} Integers.

Initially L1 cache gets filled with 2^{12} Integers with miss cycles of 1280.

The remaining integers get loaded and stored in L2 cache. But their miss ~~cycles~~ required 110 ~~cycles~~ cycles.

$\therefore \frac{2^{17} - 2^{12}}{32} = \frac{2^{12}(2^5 - 1)}{32}$

$= 2^7 (2^5 - 1)$

$= 3968$ misses are there.

Hence, $3968 \times 110 = 436480$ miss cycles there.

After 1st iteration $436480 + 1280$

$= 437760$ miss cycles aquired.

→ for 2nd and consequent every iteration 2^{12} iterations will aquire no miss

But remaining stored in L2 cache will aquire miss

$\therefore 10 \times (N-1) \times (2^{17} - 2^{12})$ additional misses cycle experienced.

$\therefore$ Total miss cycles $= 437760 + 1.664 \times 10^{11}$

$\cong 166.429 \times 10^9$ miss cycles.

c) $N = 2^{20}$

Here, we observe that Number of integers Increases from the capacity of both L1 and L2 cache.

$\therefore$ For 1st iteration; $437760 + (2^{20} - 2^{18} - 2^{12})(10+100)$ miss cycles

$= 86494720$

$= 8.649 \times 10^7$

After this we observe that from 2nd iteration till end; we have a miss for every integers in L2 cache and miss for every integers not in L1 cache and L2 cache.

Thus $(N-1)\left[(2^{18} \times 10) + (2^{20} - 2^{18} - 2^{12})(110)\right]$

(2^{18}: L1 cache miss there; $2^{20} - 2^{18} - 2^{12}$: L1 and L2 cache miss there)

$\therefore (2^{20}-1)\left[(2^{18} \times 10) + (8605696 0)\right]$

$\therefore 8.8678 \times 10^7 \times (2^{20}-1)$

$2^{20} - 1 \cong 2^{20} \Rightarrow 2^{20} \times 10^7 \times 8.8678$ miss cycles

Thus, total miss cycles

$= 8.649 \times 10^7 + 10^7 \times 8.8678 \times 2^{20}$

$= 10^7 (8.649 + 8.8678 \times 2^{20})$ miss cycl

9. 16 KB → 256 cache blocks available

4 MB → 65536 cache blocks available.

N = 512

L1 cache can fit 2^{12} integers

L2 cache can fit 2^{20} integers

Tiling → 4 blocks of A, B, C, D need to be stored in cache.

Consider their block sizes be B

$\therefore 4B^2 < 2^{12}$

$\therefore B^2 < 2^{10}$

$B \leq 32$

Thus 32 x 32 blocks sizes should be the tiling size

T = (32 x 32) elements

= 4096 = 4 KB

This will give us better performance as every time; at amount of data will be loaded from L1 cache and 32 x 32 matrix output can be generated with no misses. The underlying assumption is that entire block gets loaded in the cache (T x T).

10. (a)
1. Runtime library routine ✓
2. Runtime library routine ✓
3. Directives ✓
4. Directives ✓
5. Environment variables ✓

b) (ii), (iv) (2nd and 4th) ✓

11.
```
for (j=0; j<M; j++)                    1.
    for (i=0; i<N; i++)                2.
        A[i] += B[j][i] + C[i]         3.
```

Initially we used to get a miss for execution of 3rd statement as B[j][j] wasn't present in cache. [Row major considered].

Now by interchanging loops, cache miss is obtained when entire cache block is utilized that is N/B where B is size of cache block times per outer loop iteration.

∴ $M\left(\frac{N+1}{B}\right)$ ← for newer version → Cache miss.

for older version ~~NM~~ → Cache miss.

12. Function 2 → 40% ⇒ 0.4t } → Before optimization
Function 1 → 60% ⇒ 0.6t

t' = Execution time for 1st function before optimization
t'' = Execution time for 2nd function before optimization

t = Execution time before optimization (full code)

Formatted

After optimization;

$$\therefore t = \frac{t'}{4} + \frac{t''}{2} + (1 - c - 0.4)t$$ ← Not optimized. Left fraction.

$t'' = 0.4t$

$t' = ct$ Where $c = ?$

$$\therefore 0.65 = \frac{c}{4} + \frac{0.4}{2} + (1 - c - 0.4)$$

$$c - \frac{c}{4} = 0.2 + 1 - 0.4 - 0.65$$

$$\frac{3c}{4} = \frac{1}{5}$$

$$\therefore c = \frac{1}{5}$$

$$\therefore c = 0.2$$

Hence function 1 comprises 20% of the execution time before optimization.

13. a) (i) $T_o = pT_p - T_s$

(ii) n processing elements $T_s = \Theta(n)$ $T_p = \log n$

Speedup = $\frac{n}{\log n} = \frac{T_s}{T_p}$ Efficiency = $\frac{n}{n \log n} = \frac{1}{\log n} = \frac{T_s}{pT_p}$

cost = $pT_p = n \log n$

(b) ~~MPI_Init()~~

~~if (my_rank MPI_my_rank) = 0~~

~~MPI_Send(&x, sizeof(x),~~

(b) MPI_init()

int rank = MPI_my_rank()

if (rank = 0)

int x

MPI_Send(&x, 1, integer, 1, MPI_comm)

else if (rank = 1)

int y

MPI_receive(&y, 256, integer, 0, MPI_comm)

c) MPI_finalize()

14. a) False (i)

(b) True (ii)

(c) True (iii)

(d) True (iv)

b) do i = 1, n-1

b(i) = a(i) + s * a(i+1)

end do

Code Balance = $\frac{\text{Data traffic}}{\text{flops}} = \frac{3}{2} = 1.5$

We can backtrack the loop

```
t = s * a(n-1)
do i = n-2, 1, -1
    b(i) = a(i) + t
    t = s * a(i)
end do
```

Here ~~computational intensity~~

code balance = $\frac{2}{2} = 1$

[Data traffic is calculated for accessed elements]

Hence, by backtracking, we can ~~improve~~ decrease the code balance (that is computational intensity improvement).

b) The code balance cannot be improved for the given code

Code Balance = $\frac{1+1+1}{2}$ (b(i): 1, s: 1, a(i): 1)

for +, *

a(i+1) is present when loading (a(i))

$= \frac{3}{2} = 1.5$

The cache utilization for memory access is already done. Further the computations cannot be altered as well.

Hence, code balance cannot be improved.

15. Assuming every task has work/time complexity equal to m.

∴ 1. for (a) Max. concurrency = 8 ✓
for (b) Max. concurrency = 8 ✓

2. Critical path Length for a) 4 ✓
for b) 6 ✓

3. Max. Speedup for a) $\frac{n}{\log n}$ where n = No of nodes.

$= \frac{16}{\log 15} \simeq \frac{16}{4.9} \simeq 3.839$ ✓

for b) $\frac{15}{6} \simeq 2.17$ ✓

4. Processors required minimum
a) 8 ✓
b) 8

5. a) 2 processors. ⇒ $\frac{15}{7} \simeq 2.14$ for a) ✓

$\frac{13}{9} \simeq 1.44$ for b) ✓

b) 8 processors ⇒ a) $15/\log 15 \simeq 3.83$ ✓
b) $13/6 \simeq 2.17$ ✓