1. For a 64 processor system, compare the interconnection network for each of the following topologies. (We normalize the bandwidth of a single link to "1").

	Bus	Ring	Torus	6-d Hypercube	Fully Connected
Total # of Switches	-				
Links per Switch	-				
Total # of links	1				
Network Bandwidth	1				
Bisection Bandwidth	1				

2. The above table focuses on the overall characteristics of different interconnection networks. If we focus on a single data transmission of *n* bytes between two processors (the *source* and *destination*), then transmission time is effected by:

- *latency* (*l*) - the time that elapses between the source's beginning to transmit the data and the destination 's receiving the first byte of data.
- bandwidth (*b*) - the rate at at which the destination receives data after it has started to receive the first byte (i.e., *b* B/sec.)

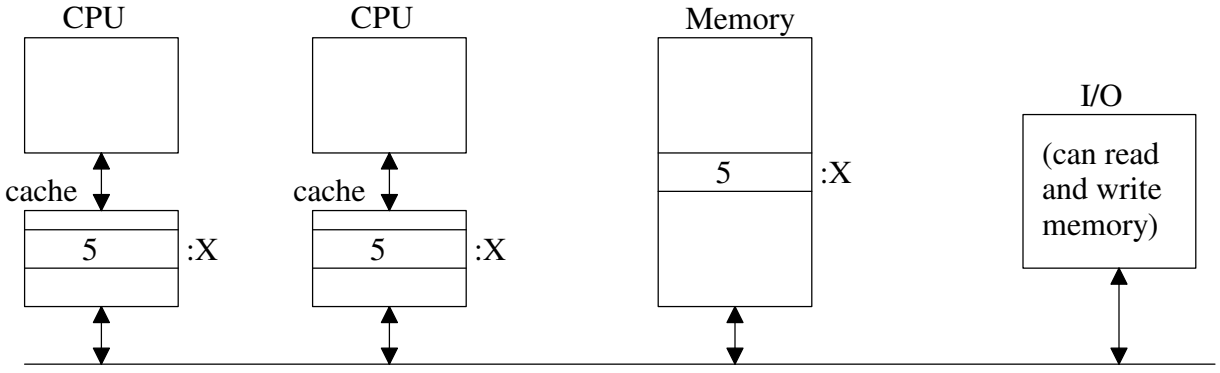
a) What is the formula for transmitting an *n* bytes message between a *source* and *destination* with a bandwidth of *b* B/second?

message transmission time =

b) What components in the above table effect the latency?

c) What components in the above table effect the bandwidth?

Cache Coherency Solution - bus watching with write through / Snoopy caches - caches eavesdrop on the bus for other caches write requests. If the cache contains a block written by another cache, it take some action such as invalidating it's cache copy.



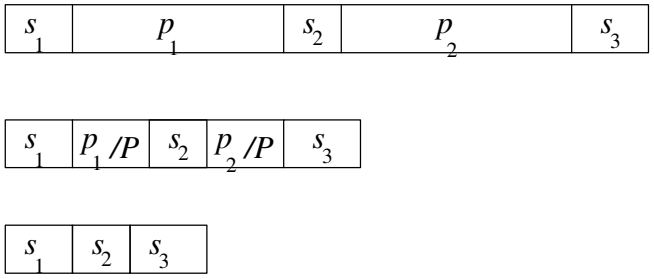
The MESI protocol is a common write-back cache-coherency protocol. Each cache line is marked as: Modified, Exclusive, Shared or Invalid.

	Modified	**E**xclusive	**S**hared	**I**nvalid
This cache line valid?	Yes	Yes	Yes	No
The memory copy is ...	out of date	valid	valid	-
Copies exist in other caches?	No	No	Maybe	Maybe
A write to this line ...	does not go to the bus	does not go to the bus	goes to the bus and update cache	goes directly to bus

3. How can distributed shared memory machines do cache coherency?

Amdahl's law expresses the limitations of parallelization due to the existence of non-parallelizable computations. If $1/S$ of the computation is inherently sequential, then the maximum speedup performance improvement (speedup) is limited to a factor of S.

$$speedup = \frac{sequential\ execution\ time}{parallel\ execution\ time} = \frac{T_S}{T_P}$$



4. If $1/S$ is the fraction of sequential program that is non-parallelizable, what is the formula for the T_P assuming linear speedup of the parallelizable portion of the sequential program?

$T_P =$

5. What are the four major categories of performance loss that prevents *linear speedup* (i.e., P processors speeding up a computation by a factor of P)?

- Overhead - additional costs in the parallel solution not in the sequential computation
- Non-parallelizable computation
- Idle processors
- Contention for resources

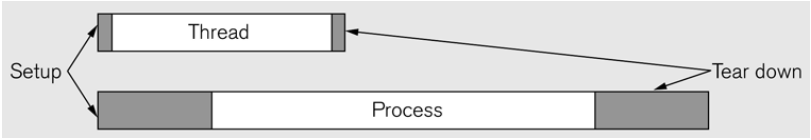
Identify the follow special cases as one of the above.

a) theads doing extra computation to determine which part of the parallel computation they need to perform

b) parallel computation is unevenly distributed to processors so some finish before others

c) a spin lock in which a waiting thread repeated checks for the availability of a lock on a shared variable

d) thread/process setup and teardown time



e) Communication costs from the communication mechanism

Mechanism	Components of Communication Cost
Shared Memory	Transmission delay, coherency operations, mutual exclusion, contention
1-sided	Transmission delay, mutual exclusion, contention
Message Passing	Transmission delay, data marshalling, message formation, demarshalling, contention

(get and put operations)
(send and recv operations)

f) Sequential computation performed redundantly across all processors

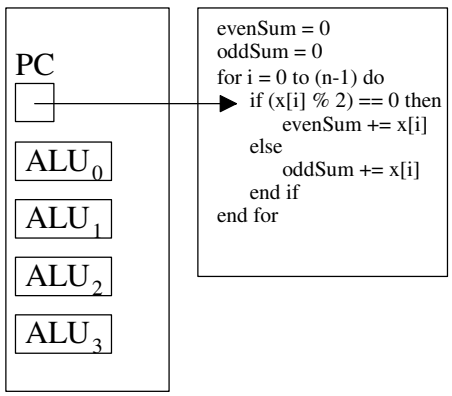
Flynn's Taxonomy

SISD
(single-instruction, single-data)
sequential uniprocessor computer

MISD
???

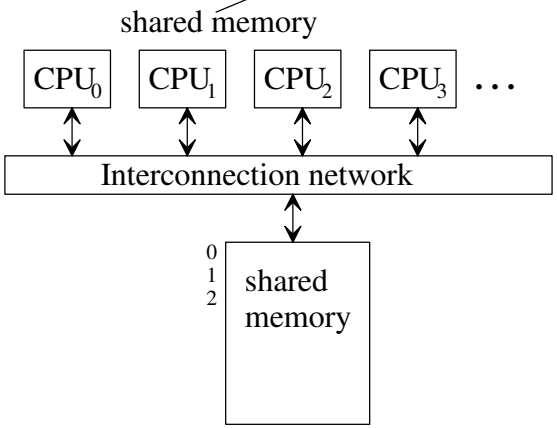
MIMD (multiple-instruction, multiple-data)
CPU's execute different programs (or different points in the same program -- SPMD)

SIMD
(single-instruction, multiple-data)

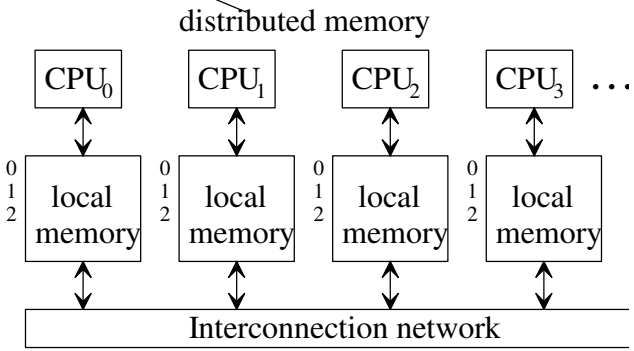


Single program, but multiple ALUs executing on different data value or not (e.g., ALU_0 on x[0], ALU_1 on x[1], etc...) executing if the condition is not satisfied.

(GPU programming is SIMD-like)



Programming model: (e.g. pthreads)
* start single process that "forks" threads
* each thread carries out a task
* threads communicate and synchronize through shared variable in memory



Programming model: (e.g., MPI)
* start multiple processes on each CPU
* each process carries out a task
* processes communicate and synchronize by send messages through the interconnection network