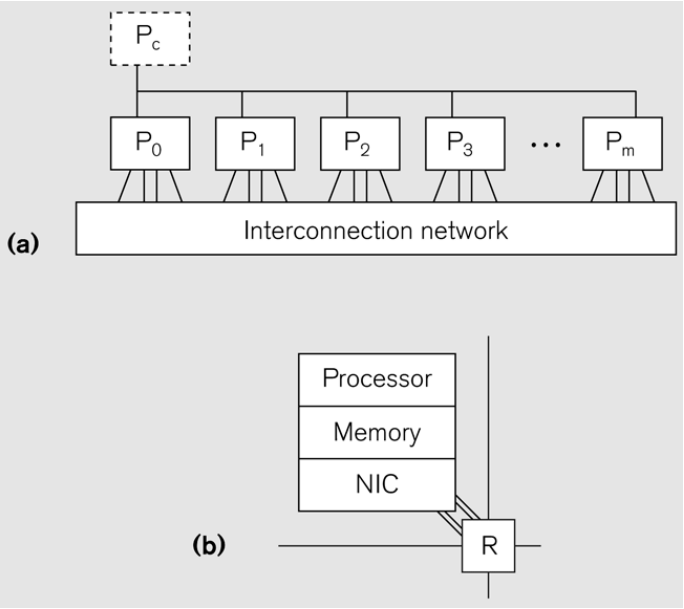
CTA (Candidate Type Architecture) Model - accounts for communication costs by explicitly separating memory references into inexpensive local memory references and expensive non-local memory references.



P_i - P processors - std. sequential (RAM model) computers connected by an interconnection network

P_c - controller processor to perform initialization, synchronization, eurekas, etc.

interconnection - communication network with unspecified topology

non-local memory access time, $\lambda >> 1$, can be between 2-5 orders of magnitude larger than local memory

a low node degree implies that a processor cannot have more than 1 or 2 network transfers in flight at once

Locality Rule: Fast parallel programs tend to maximize the number of local memory references and minimize the number of non-local references.

1. If we apply the CTA Model to counts 3s example, why would the Locality rule have identified the problem of Try 2 design (single global `count` variable)?

2. A problem that takes *T* time on a single processor would ideally take how long on *P* processors?

3. Why might ideal performance not be achieved in a parallel program?

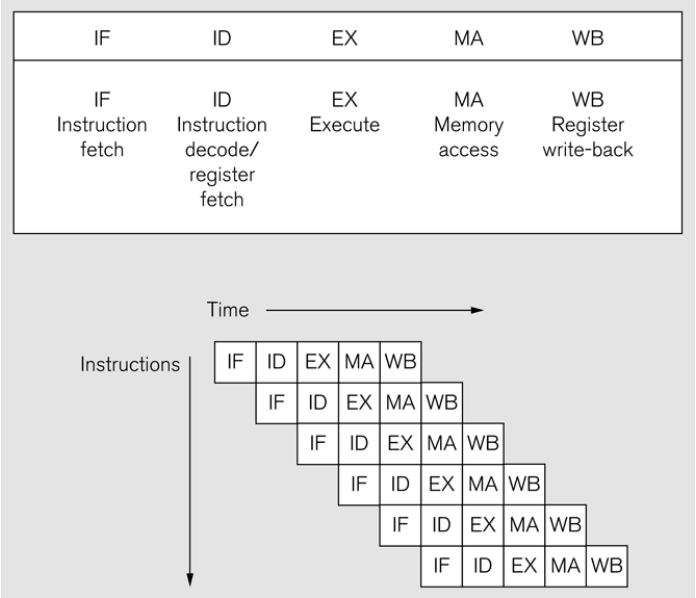
4. There are two common abstractions describing parallelism: threads and processes. What are the differences between the two with respect to?

a) communication with other threads/processes

b) overhead costs for creating and destroying

Two aspects to better speed/performance:
- *latency* - the amount of time it take to complete a given unit of work
- *throughput* - the amount of work that can be completed per unit of time

Consider a simplified processor pipeline:



a) If the unit of work is an instruction, how is the latency effected by pipelining?

b) If the unit of work is an instruction, how is the throughput effected by pipelining?

c) If the unit of work is the whole program, how are latency and throughput effected by pipelining?

d) How does the operating system (OS) scheduler, exploit parallelism to hide latency when an executing process performs an I/O operation?

5. There are 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

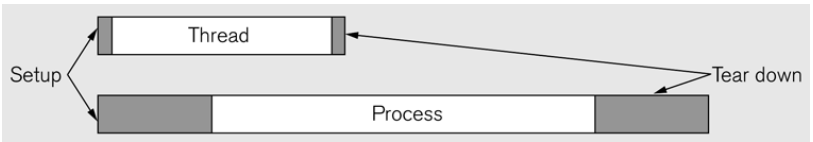
Identify the follow special cases as one of the above.

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

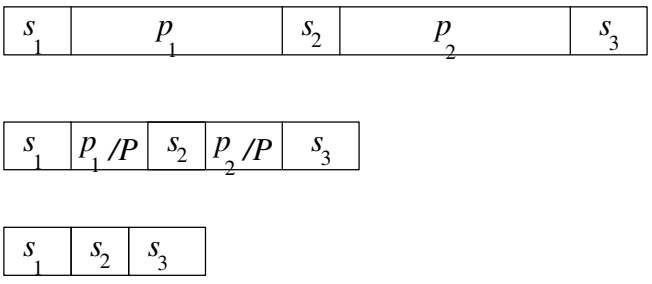
Mechanism	Components of Communication Cost
Shared Memory	Transmission delay, coherency operations, mutual exclusion, contention
1-sided	Transmission delay, mutual exclusion, contention
Message Passing	Transmission delay, data marshalling, message formation, demarshalling, contention

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

f) Sequential computation performed redundantly across all processors

6. 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*.

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



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 =

Consideration for avoiding parallel performance loss:
- dependence - the ordering relationship between two computations which describe limits to parallelism
- granularity - frequency of interaction among the threads/processes (fine vs. course)
- locality - temporal locality (memory references are clustered in time) and spatial locality (memory references are clustered by address

A data dependence is an ordering on a pair of memory operations that must be perserved to maintain correctness.
Three types of data dependences:
- flow (true) dependence: read-after-write (RAW)
- anti-dependence: write-after-read (WAR)
- output dependence: write-after-write (WAW)

a) Identify the dependences in the following code segment:

```
1       sum = a + 1;
2       first_term = sum * scale1;
3       sum = b + 1;
4       second_term = sum * scale2;
```

b) How can we rename (use extra variables) to remove anti- and output dependences?

7. *Course grain granularity* refers to threads/processes that only infrequently depend on data or events in other threads/processes. *Fine grain granularity* refers to threads/processes that interact frequently with other threads/processes.

a) How does granularity impact communication or synchronization overheads?

b) Which type of parallel computer systems can best support fine-grain parallelism?

8. Algorithms that operate on blocks of data exploit temporal or spatial locality of reference?

9. Why does locality in a parallel program minimizes dependences among threads/processes and thereby reducing overhead and contention?