1. In 1966 Michael J. Flynn suggested Flynn's Taxonomy that classified computer architectures based on the number of instruction streams and data streams:

	Single Instruction Stream	**Multiple Instruction Stream**
Single Data Stream	SISD	MISD
Multiple Data Stream	SIMD	MIMD

How would we classify each our computer types?

a) single CPU computer

b) chip multiprocessors (e.g., quad-core processor)

c) Symmetric Multi-Processor

d) clusters

e) supercomputers

2. "How can we write programs that perform well on such diverse machines/computers?"
Algorithm Designer's answer: abstract away unimportant details to get a single accurate model of parallel computers to guide parallel program development (PRAM model and CTA model)

PRAM (Parallel Random Access Machine) model:

- unspecified number of instruction execution units connected to a single unbounded shared memory containing both instructions and data
- instruction execution units follow their own threads (i.e., different PCs), but they execute instructions in lock step (all execute an instruction is a cycle -- share a common clock)
- all execution units observe a single sequence of memory state changes (called the *single memory image*). For example, if one instruction changes x[0] at the same time another instruction (executing on a different execution unit) changes x[1], then on the next instruction(s) the values of both memory locations have been updated.
- multiple instructions reading/writing to the same memory location at the same time give different PRAM models. Some possible models: EREW (exclusive-read, exclusive-write), ERCW (exclusive-read, concurrent-write), CREW (concurrent-read, exclusive-write), or CRCW (concurrent-read, concurrent-write)
  typically concurrent-reads are allowed, with concurrent-writes handled many ways (sub-models), some are:
  - memory location only changed if all writes are trying to write the same value,
  - memory location changed to an arbitrary value being written, etc.

a) Is the PRAM model closest to SIMD or MIMD?

b) Is the PRAM model closest to NUMA (non-uniform memory access) or UMA?

3. Four versions of PRAM (EREW, ERCW, CREW, CRCW) model depending on how simultaneous Reads/Writes handled:

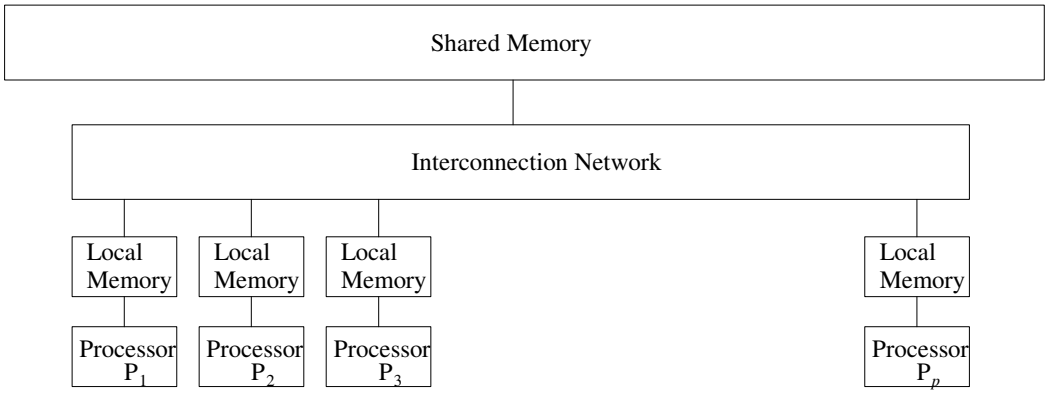
a) Which version of PRAM allows for least amount of concurrent access?

b) Which version allows for the most amount of concurrent access?

c) Which is the closest match to "real" computer hardware?

4. Parallel algorithm extensions:

- data-parallel algorithms with all processors executing the same program simultaneously on different data elements
- assume an infinite number of processors
- each processor is indexed and each processor can determine their index: "p = index of this processor"
- "local" variables are used only by one processor
- only read or write shared memory element to/from local variables, i.e., shared variables cannot be compared (except ones that don't change), incremented, etc.



a) Which assumption(s) are not realistic?

4. Consider an array S of values:

	1	2	3	4	5	6	7	8
S:	12	10	5	15	18	12	4	16

a) Outline the sequential algorithm to find the largest value in S

b) What is the big-oh notation of the sequential algorithm?

c) Complete the comparisons of the tournament algorithm below:

	1	2	3	4	5	6	7	8
S:	12	10	5	15	18	12	4	16

12 15 18 16

d) Why might the tournament algorithm be easier to parallelize than the algorithm in (a)?

5. CREW PRAM algorithm to find the largest "key":

```
keytype parlargest ( int n, keytype S[] ) {
   index step, size
   local index p
   local keytype first, second

  p = index of this processor
   size = 1
   for (step = 1; step <= lg n; step++) {
      if (this processor needs to execute in this step) {
         read S[2*p - 1] into first
         read S[2*p - 1 + size] into second
         write maximum(first, second) into S[2*p -1]
         size = 2 * size
      } // end if
   } // end for
   return S[1]
}
```

a) Trace the above algorithm on S below:

	1	2	3	4	5	6	7	8
S:	12	10	5	15	18	12	4	16

b) What should the condition of the "if" statement be?

c) What would be the θ( ). Let *n* be the size of S?

6. Why are many dynamic programming algorithms "easy" to parallelize? Consider "dynamic programming" binomial coefficient function calculated using the recursive definition for the binomial coefficient:

$C(n,\ k) = C(n-1,\ k-1) + C(n-1,\ k)$ for $1 \le k \le (n-1)$, and

$C(n,\ k) = 1$ for $k = 0$ or $k = n$

								Row #
C(0,0)								0
C(1,0)	C(1,1)							1
C(2,0)	C(2,1)	C(2,2)						2
C(3,0)	C(3,1)	C(3,2)	C(3,3)					3
C(4,0)	C(4,1)	C(4,2)	C(4,3)	C(4,4)				4
C(5,0)	C(5,1)	C(5,2)	C(5,3)	C(5,4)	C(5,5)			5
⋮								⋮
				C(n-1,k-1)	C(n-1,k)			n-1
					+			
C(n,0)	C(n,1)	C(n,2)	. . .		C(n,k)	C(n, n-1)	C(n,n)	n

7. Consider parallel merge sort of S:

	1	2	3	4	5	6	7	8
S:	12	10	5	15	18	12	4	16