1. B is faster because A needs gather non-contiguous memory / B can do contiguous vector loads

2. Branch divergence. Waiting for the other half (statistically -> random!) to complete. (maybe it is more precise that the initial values are randomly generated centered at 0.5)

3. A.
Shared memory: data stored in a shared memory address space, accessed directly by each processor
Message passing: data store in separate memory spaces, local data accessed directly, remote data accessed via message passing

B.
Shared memory: communication implicitly carried out via modification to shared variables (i.e. loads and stores)
Message passing: communication explicitly carried out via message passing

C.
Shared memory: synchronization explicitly carried out via synchronizers like locks
Message passing: communication implicitly contained in communication

D.
Shared memory: no replication needed, coherence issue can occur among different caches
Message passing: coherence between different nodes (memory spaces) has to be maintained if replication is present

4. A
(a). Runtime highly predictable for each small task, especially if also the size of each small task is about the same.
(b). Little overhead in scheduling, could have scheduled things at compile time.

B
(a). Runtime of each small task varies a lot and is intrinsically not predictable.
(b). Flexibility to perform load balancing when needed by adjusting workload of each node dynamically.

C
(a). Scheduling has overhead (such as, task queues and synchronizing access to said queues) and usually the amount of overhead is not relevant to the size of each task. If each task is too small in size, the ratio of overhead would be too high.
(b). Large task size makes dynamic scheduling less flexible/effective because it is harder to make the load really balanced.
(ci). Only a small fraction of total time is time spent on executing instructions, indicating that overhead for scheduling is too high. And the time is spent on synchronization
(cii). Too much time is spent on waiting for access/synchronization, indicating that some tasks are taking too long to finish. Early in execution, lots of work is completed
(d).
Advantages:

1.locality of work,
2. Avoids global lock,
3. Less overhead to get task from local queue

Disadvantages:

1. Have to move data around if initial task assignment too skewed, (load balancing);
2. Synchronization is a little bit trickier, with task stealing

5. A

(a). Block is better. No false sharing. And less communication between processors due to perimeter:area.
(bi). Block decomposition performance will be worse. Because the number of total data being invalidated each time would be larger from false sharing
(bii). No big influence on row decomposition, spatial locality improves, so performance may improve.

B

False sharing impacts the sigchld handler that would otherwise be shared.
Add padding to each of the three fields so that they live on different cache lines. In this way when they are used in different access patterns, no unnecessary invalidation need to occur.
(Most students instead like to duplicate the values to be per processor... )

6. A.

Because writes are frequent, so invalidations are frequent. It is not highly probable that we have accumulated a lot of sharers between two writes (invalidations).

B

(a).
Full-bit -> 4 bytes overhead each 32 bytes -> 4/32 or 12.5%
Limited -> 3*(5) bits overhead each 32 bytes -> 15/(8*32) or 5.85%
(b).
Full-bit -> 128 bytes overheat each 32 bytes -> 400%
Limited -> 3*10 bits overhead each 32 bytes -> 30 / (8 * 32) or 11.7%

C

(a). It is clear where to get the line (whether clean or dirty) so no need for broadcasting or any more complex mechanism.
(b). (I had students discuss this in class last week) The most recent reader is the F state, so by an LRU across system, that reader is more likely to still have the line when the next request comes in.

7.

Sequential consistency is referring to all processors agreeing on the global order of visible operations.
A.
(In class, I taught about transition states, such as SM and IM. They are a useful way to describe what is happening here, but not strictly required)

P1's cache controller was in SM (S going to M, requiring upgrade), but is now invalidated as its operation is after P2. When it acquires the bus, it must issue a BusRdX as it transitions from IM -> M.

B (can be multiple valid answers)

- Mapping responses back to requests, requires table
- Queues can block as multiple requests can be outstanding, need separate request and response queues
- Multiple caches requesting exclusive permission, can detect and delay later writers rather than NACKing