1. One possible explanation could be that the L1 block size on this machine is 16 bytes and that both A and B share the same cache block. In this case, every time a Thread A updates the value of A, it becomes the owner of the cache block and invalidates it in the L1 cache of the other thread (since they run on different cores), and every time Thread B updates the value of B the same thing occurs. Every time a thread wants to write to a block that is invalid, it results in a cache miss at which point the caches need to be synchronized and the latest version of the block needs to be brought into the L1 cache of the requesting thread. This overhead can cause a serious loss in performance.

There are many ways in which we could address this. One way to fix it would be to decrease the cache line size to 8 bytes to ensure that A and B are allocated to different cache blocks and hence will not result in an invalidation. Or we have a guarantee that the compiler allocates variables in order then we could introduce a dummy variable C in between A and B and not use it. Another way would be to add locks in the code such that every time a thread acquires a lock it updates its variable a billion times before releasing it.

2. (a) SMT outperforms CMP:

1. A single thread with high Instruction Level Parallelism will have higher performance on an SMT than a CMP because the SMT will have double the resources compared to CMP and only one thread will be active in CMP (in one core).
2. Another case is running 2 threads where one is memory bound and another thread is computational. On an SMT machine, both threads can be assigned at the same time where one thread executes using the resources when another (memory-bound) thread is waiting.
3. Two threads that complement each other and result in positive interference (shares cache, branch prediction)

(b) CMP outperforms SMT:

1. Two threads that results in thrashing because of too many conflict misses on the data cache.
2. Two threads with high ILP that compete for renaming registers, ROB entries.

3. a. A physically-indexed, virtually-tagged cache
Every virtual address should be translated by the TLB before indexing the cache or the tag array.
Hit Time = 3 + max(3.5 + 2, 4) + 1.5 = 10 ns
b. A virtually-indexed, virtually-tagged cache
For a virtual cache, the translation of physical address is required only when there is a cache miss. The TLB access is not on the critical path of cache access.
Hit Time = max(3.5 + 2, 4) + 1.5 = 7ns

c. A virtually-indexed, physically-tagged cache
The address translation can be performed in parallel with tag and data lookup of cache. After the results are out, the cache can compare them immediately to determine if the access is a hit or miss.
Hit Time = max(max(3, 3.5) + 2, 4) + 1.5 = 7ns

d. A physically-indexed, physically-tagged cache
For a physical cache, every virtual address should be translated by the TLB before indexing the cache or the tag array.
Hit Time = 3 + max(3.5 + 2, 4) + 1.5 = 10ns

4.
a) Instruction Level Parallelism:
It can be exploited using Out Of Order processors and the limitation is finding instructions to execute out of order (no true dependencies) and have enough renaming registers.

b) Data Level Parallelism:
Data Level Parallelism can be exploited by using vector or SIMD architectures. The limitation is parallelizing normal code to run well on these machines is hard and this parallelism is applicable to only specific workloads.

c) Thread Level Parallelism:
This can be exploited using multiprocessors like CMP and the limitation could be finding parallelism in code, overhead of cache coherence and difficulty in programming due to issues with consistency/concurrency.

5.
a. CPI = 1 + Wrong prediction penalty
   = 1 + ( ⅕ * (1-0.8) * 3 )
   = 1 + ( 0.2 *0.2 * 3)
   = 1.12

Cycles per instruction = 1.12
Cycles for ‘n’ instructions = 1.12 * n

b. 1.12 = 1 + ( ⅕ * (1-x) * 19 )
   0.12 = 3.8 (1-x)
The branch prediction rate of the Pentium 4 should be 96.84% for the same performance as MIPS.

6.
   a) Consider a branch with the prediction as follows:
      T T T T NT NT

      This will have a prediction accuracy of 1/3

      If 2 such branches are interleaved as follows: (Read B1, B2, B1, B2 and so on)

      B1: T T T T NT NT
      B2: T T T T NT NT

      In this case, the prediction accuracy is 2/3

      Thus, there is positive interference.

   b) Consider a branch with prediction as follows:
      T T T T T T

      This will have prediction accuracy of 100% in steady state.

      If it is combined with another branch as follows:

      B1 : T T T T T T
      B2 : NT NT NT NT NT NT

      It will have only 50% accuracy.
      Thus, there is negative interference.

   c) Considering 2 random branches, the chances of them being correlated is very less. Correlation usually happens only when branches are at close proximity. When 2 branches are not correlated to each other, the outcomes of each looks like noise to the other. Introducing noise in branch predictor will end up in confusing the predictor causing negative interference.