Alexandria University

Faculty of Engineering

Computer and Communications Department

CC423: Advanced Computer Architecture

Sheet 4: Thread-Level Parallelism

---

1. Assume that we have a function for an application of the form F (i , p), which gives the fraction of time that exactly i processors are usable given that a total of p processors is available. That means that

$$\sum_{i=1}^{p} F(i, p) = 1$$

   Assume that when i processors are in use, the applications run i times faster. Rewrite Amdahl's law so it gives the speedup as a function of p for some application.

2. In this exercise, we examine the effect of the interconnection network topology on the clock cycles per instruction (CPI) of programs running on a 64-processor distributed-memory multiprocessor. The processor clock rate is 3.3 GHz and the base CPI of an application with all references hitting in the cache is 0.5. Assume that 0.2% of the instructions involve a remote communication reference. The cost of a remote communication reference is (100 + 10h) ns, where h is the number of communication network hops that a remote reference has to make to the remote processor memory and back. Assume that all communication links are bidirectional.

   a. Calculate the worst-case remote communication cost when the 64 processors are arranged as a ring, as an 8X8 processor grid, or as a hypercube. (Hint: The longest communication path on a 2^n hypercube has n links.)
   b. Compare the base CPI of the application with no remote communication to the CPI achieved with each of the three topologies in part (a).
   c. How much faster is the application with no remote communication compared to its performance with remote communication on each of the three topologies in part (a).

3. Show how the basic snooping protocol of Figure 1 can be changed for a write-through cache. What is the major hardware functionality that is not needed with a write-back cache?

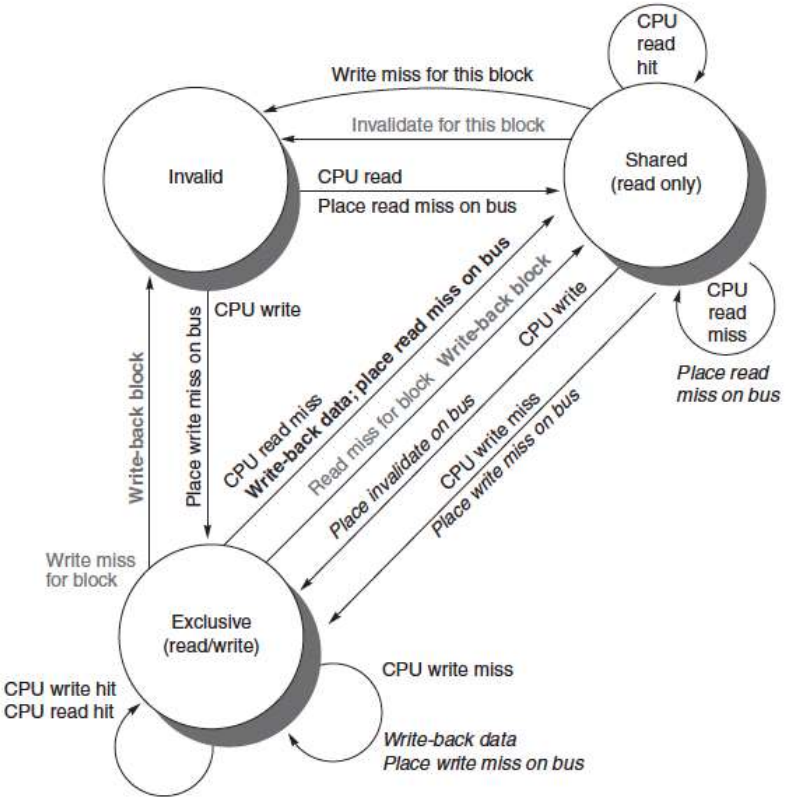

Figure 1

4. Add a clean exclusive state to the basic snooping cache coherence protocol Figure 1. Show the protocol in the format of Figure 1.

5. One proposed solution for the problem of false sharing is to add a valid bit per word. This would allow the protocol to invalidate a word without removing the entire block, letting a processor keep a portion of a block in its cache while another processor writes a different portion of the block. What extra complications are introduced into the basic snooping cache coherence protocol Figure 1 if this capability is included? Remember to consider all possible protocol actions.

6. This exercise studies the impact of aggressive techniques to exploit instruction-level parallelism in the processor when used in the design of shared-memory multiprocessor systems. Consider two systems identical except for the processor. System A uses a processor with a simple single-issue in-order pipeline, while system B uses a processor with four-way issue, out-of-order execution, and a reorder buffer with 64 entries.

   a. Following the convention of Figure 2, let us divide the execution time into instruction execution, cache access, memory access, and other stalls. How would you expect each of these components to differ between system A and system B?
   b. Based on the discussion of the behavior of the On-Line Transaction Processing (OLTP) workload in Section 5.3, what is the important difference between the OLTP workload and other benchmarks that limits benefit from a more aggressive processor design?

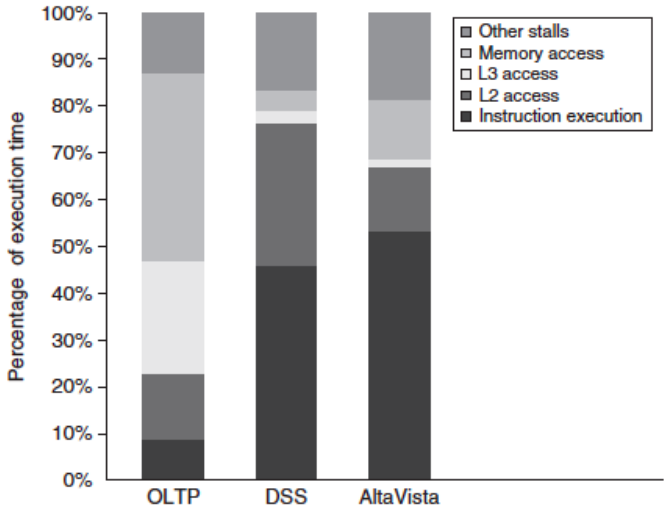

Figure 2

7. How would you change the code of an application to avoid false sharing? What might be done by a compiler and what might require programmer directives?

8. Assume a directory-based cache coherence protocol. The directory currently has information that indicates that processor P1 has the data in "exclusive" mode. If the directory now gets a request for the same cache block from processor P1, what could this mean? What should the directory controller do? (Such cases are called race conditions and are the reason why coherence protocols are so difficult to design and verify.)

9. A directory controller can send invalidates for lines that have been replaced by the local cache controller. To avoid such messages and to keep the directory consistent, replacement hints are used. Such messages tell the controller that a block has been replaced. Modify the directory coherence protocol of Section 5.4 to use such replacement hints.

10. One downside of a straightforward implementation of directories using fully populated bit vectors is that the total size of the directory information scales as the product (i.e., processor count x memory blocks). If memory is grown linearly with processor count, the total size of the directory grows quadratically in the processor count. In practice, because the directory needs only 1 bit per memory block (which is typically 32 to 128 bytes), this problem is not serious for small to moderate processor counts. For example, assuming a 128-byte block, the amount of directory storage compared to main memory is the processor count/1024, or about 10% additional storage with 100 processors. This problem can be avoided by observing that we only need to keep an amount of information that is proportional to the cache size of each processor. We explore some solutions in these exercises.

    a. One method to obtain a scalable directory protocol is to organize the multiprocessor as a logical hierarchy with the processors as leaves of the hierarchy and directories positioned at the root of each subtree. The directory at each subtree records which descendants cache which memory blocks, as well as which memory blocks with a home in that subtree are cached outside the subtree. Compute the amount of storage needed to record the processor information for the directories, assuming that each directory is fully associative. Your answer should also incorporate both the number of nodes at each level of the hierarchy as well as the total number of nodes.

    b. An alternative approach to implementing directory schemes is to implement bit vectors that are not dense. There are two strategies; one reduces the number of bit vectors needed, and the other reduces the number of bits per vector. Using traces, you can compare these schemes. First, implement the directory as a four-way set associative cache storing full bit vectors, but only for the blocks that are cached outside the home node. If a directory cache miss occurs, choose a directory entry and invalidate the entry. Second, implement the

directory so that every entry has 8 bits. If a block is cached in only one node outside its home, this field contains the node number. If the block is cached in more than one node outside its home, this field is a bit vector, with each bit indicating a group of eight processors, at least one of which caches the block. Using traces of 64-processor execution, simulate the behavior of these schemes. Assume a perfect cache for non-shared references so as to focus on coherency behavior. Determine the number of extraneous invalidations as the directory cache size in increased.

11. Implement the classical test-and-set instruction using the load-linked/ store-conditional instruction pair.

12. One performance optimization commonly used is to pad synchronization variables to not have any other useful data in the same cache line as the synchronization variable. Construct a pathological example when not doing this can hurt performance. Assume a snooping write invalidate protocol.

13. One possible implementation of the load-linked/store-conditional pair for multicore processors is to constrain these instructions to using uncached memory operations. A monitor unit intercepts all reads and writes from any core to the memory. It keeps track of the source of the load-linked instructions and whether any intervening stores occur between the load-linked and its corresponding store-conditional instruction. The monitor can prevent any failing store conditional from writing any data and can use the interconnect signals to inform the processor that this store failed. Design such a monitor for a memory system supporting a four-core symmetric multiprocessor (SMP). Take into account that, generally, read and write requests can have different data sizes (4, 8, 16, 32 bytes). Any memory location can be the target of a load-linked/ store-conditional pair, and the memory monitor should assume that load-linked/ store-conditional references to any location can, possibly, be interleaved with regular accesses to the same location. The monitor complexity should be independent of the memory size.

14. As discussed in Section 5.6, the memory consistency model provides a specification of how the memory system will appear to the programmer. Consider the following code segment, where the initial values are

A=flag=C=0.

P1	P2
A= 2000	while (flag ==1){;}
flag=1	C=A

a. At the end of the code segment, what is the value you would expect for C?

b. A system with a general-purpose interconnection network, a directory-based cache coherence protocol, and support for nonblocking loads generates a result where C is 0. Describe a scenario where this result is possible.

c. If you wanted to make the system sequentially consistent, what are the key constraints you would need to impose?

Assume that a processor supports a relaxed memory consistency model. A relaxed consistency model requires synchronization to be explicitly identified. Assume that the processor supports a "barrier" instruction, which ensures that all memory operations preceding the barrier instruction complete before any memory operations following the barrier are allowed to begin. Where would you include barrier instructions in the above code segment to ensure that you get the "intuitive results" of sequential consistency?

15. Prove that in a two-level cache hierarchy, where L1 is closer to the processor, inclusion is maintained with no extra action if L2 has at least as much associativity as L1, both caches use line replaceable unit (LRU) replacement, and both caches have the same block sizes.

16. When trying to perform detailed performance evaluation of a multiprocessor system, system designers use one of three tools: analytical models, trace-driven simulation, and execution-driven simulation. Analytical models use mathematical expressions to model the behavior of programs. Trace-driven simulations run the applications on a real machine and generate a trace, typically of memory operations. These

traces can be replayed through a cache simulator or a simulator with a simple processor model to predict the performance of the system when various parameters are changed. Execution-driven simulators simulate the entire execution maintaining an equivalent structure for the processor state and so on. What are the accuracy and speed trade-offs between these approaches?