Exercises

Solutions to the "starred" exercises appear in Appendix B.

6.1 [15] <6.1> 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 are available. This means that

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

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

6.2 [10] <6.1, 6.2> The Transaction Processing Council (TPC) has several different benchmarks. Visit their Web site at *www.tpc.org* and look at the top 10 performers in each benchmark class. Determine whether each of the top 10 configurations is a multiprocessor, and if so, what type (e.g., SMP, NUMA, cluster). Does the ordering look different if price-performance is used as the metric?

6.3 [10] <6.1, 6.2> The Top 500 list categorizes the fastest scientific machines in the world according to their performance on the Linpack benchmark. Visit their Web site at *www.top500.org* and look at the top 100 performers (there are many repeats of a particular vendor product, since individual supercomputer sites rather than a product are counted). Determine how many different supercomputer products occur among the top 100 configurations and what type (e. g., SMP, NUMA, cluster) each different supercomputer is. Try to obtain cost information and see how the data change when cost-performance is considered.

✪ 6.4 [15] <6.3> In small bus-based multiprocessors, write-through caches are sometimes used. One reason is that a write-through cache has a slightly simpler coherence protocol. Show how the basic snooping cache coherence protocol of Figure 6.12 on page 559 can be changed for a write-through cache. From the viewpoint of an implementor, what is the major hardware functionality that is not needed with a write-through cache compared with a write-back cache?

6.5 [20] <6.3> Add a clean exclusive state to the basic snooping cache coherence protocol (Figure 6.12 on page 559). Show the protocol in the format of Figure 6.12.

6.6 [15] <6.3> One proposed solution for the problem of false sharing is to add a valid bit per word (or even for each byte). This would allow the protocol to invalidate a word without removing the entire block, letting a cache 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 6.12) if this capability is included? Remember to consider all possible protocol actions.

6.7 [12/10/15] <6.3> The performance differences for write invalidate and write update schemes can arise from both bandwidth consumption and latency. Assume a memory system with 64-byte cache blocks. Ignore the effects of contention.

a. [12] <6.3> Write two parallel code sequences to illustrate the bandwidth differences between invalidate and update schemes. One sequence should make update look much better and the other should make invalidate look much better.

b. [10] <6.3> Write a parallel code sequence to illustrate the latency advantage of an update scheme versus an invalidate scheme.

c. [15] <6.3> Show, by example, that when contention is included, the latency of update may actually be worse. Assume a bus-based multiprocessor with 50-cycle memory and snoop transactions.

6.8 [15/10] <6.2, 6.4> 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. For four benchmarks—online transaction processing, decision support systems, LU, and FFT—system B gives a speedup of 1.5, 2.5, 2.9, and 2.4, respectively, over system A.

a. [15] <6.4> Following the convention of Figure 6.13, 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. [10] <6.2, 6.4> Based on the discussion of the behavior of OLTP workloads in Section 6.4, what is the important difference between the OLTP workload and the other benchmarks that limits benefit from a more aggressive processor design?

6.9 [15] <6.4> 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?

✪ 6.10 [15] <6.5> 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 reasons why coherence protocols are so hard to design and verify.)

6.11 [20] <6.5> As we discussed earlier, the 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 6.5 to use such replacement hints.

6.12 [12/10/12/15] <6.4, 6.6> One possible approach to achieving the scalability of distributed shared memory and the cost-effectiveness of a bus design is to combine the two approaches, using a set of processors with memories attached directly to the processors and interconnected with a bus. The argument in favor of such a design is that the use of local memories and a coherence scheme with limited broadcast results in a reduction in bus traffic, allowing the bus to be used for a larger number of processors.

Assume the following characteristics for a machine with 64-byte cache blocks:

Total data miss rate (assume instruction miss rate is negligible)	2%
% misses to private data (used only by this processor)	60%
% misses to shared data that is unowned in a remote cache/memory	20%
% misses to shared data that is dirty in a remote cache	80%

Assume a local memory miss takes 100 ns until processor restart. For remote misses that must use the bus, use the E6000 (Wildfire) restart memory miss times from the top portion of Figure 6.50.

a. [12] <6.4, 6.6> Find the average miss time for this design.

b. [10] <6.4, 6.6> Assume a 1 GHz clock rate, a CPI of 1.0 when the cache hit rate is 100%, and that a load or store is issued every other clock cycle. If the processor is stalled during a cache miss until processor restart, what is the effective CPI? What is the effective rate at which loads or stores are issued from the processor?

c. [12] <6.4, 6.6> Assume a split-transaction bus with a request and acknowledge for all bus transactions. If a bus request or acknowledge requires 16 bytes of bus bandwidth and a data transfer requires a total of 80 bytes, what is the bandwidth demand on the shared bus per processor?

d. [15] <6.4, 6.6> Split-transaction buses are quite complex in practice, and the only accurate way to assess their bandwidth limitations is often by measurement. Using the results from part (b) and the assumptions from part (c) about bandwidth requirements, determine how many processors could share a bus with the bandwidth characteristics of the E6000 (Wildfire) bus as shown in the top portion of Figure 6.51. Assume that the processors should not consume more than 80% of the unowned or exclusive data bandwidth.

✪ 6.13 [12/10/12] <6.4, 6.6, 6.11> In this exercise, we compare the design of the Sun Wildfire to the Sun Origin on the basis of memory latency and bandwidth. For this exercise, use the latency measurements to processor restart from Figure 6.50 and the bandwidth measurements in Figure 6.51. Assume a 64-processor design with 16 processors per Wildfire node. Assume that remote accesses are uniformly distributed. (Remember that this affects Wildfire and the Origin differently!).

a. [12] <6.6, 6.11> For the remote access time outside of an Origin or Wildfire node, use the average remote latency for less than 128 nodes from the second section of Figure 6.50. Assuming that 80% of the remote misses are to dirty lines, find the average remote memory access time for Wildfire and Origin.

b. [10] <6.4, 6.6, 6.11> What should the fraction of dirty remote requests be to minimize the difference in remote access time between Wildfire and the Origin? What does this tell you about the design of the cache system within a node?

c. [12] <6.4, 6.11> Compare the local memory bandwidth of the Origin and E6000 nodes using the data in the top of Figure 6.51. Assuming accesses are either unowned or dirty, for what fraction of dirty accesses will the bandwidth of the two designs be the same?

6.14 [15/15/20] <6.6> A key design question in DSM multiprocessors is how big to make the nodes in the design. In Origin, each node contains two processors, while in Wildfire a node contains more processors, with 16 to 20 being a likely target. This choice represents a complex design trade-off: When each node is larger, more accesses use the first level of interconnect, which is usually faster, but larger nodes typically impose a higher burden on accesses that must go remote. Throughout this exercise assume a total of 128 processors, and assume the goal is to minimize the remote access time. A spreadsheet will make this exercise much easier!

Assume the following local and remote access times:

Intranode access time with n processors per node ($n \geq 2$)	$300 + 20\,(n - 1)$ ns
Average remote access time	Local memory access time $+ 100 + 100 \times \sqrt{\frac{128}{n}}$

a. [15] <6.6> Assuming a uniform distribution of remote accesses, find the optimum node size for this multiprocessor.

b. [15] <6.6> A one-processor node could have a considerably lower local memory access time. Does there exist a one-processor node design that is faster than the optimal node size from part (a)? If so, what must the local memory access time be? If not, how would you change this design if you wanted to use a one-processor node design?

c. [20] <6.6> Now consider optimizing node size for best average memory access time. Suppose the memory accesses are not uniformly distributed, but heavily biased toward a nearest neighbor structure. The probability of a nearest neighbor being in the same node is given by the following:

Processors per node	Fraction within node
2–3	25%
4–7	50%
8–15	62.5%
16–31	75%
31–63	87.5%
≥ 64	100%

If this is the only improvement gained from a nearest neighbor allocation, find the optimal node size, assuming that 60% of the remote accesses are nearest neighbor.

6.15 [20] <6.6> In reality, a nearest neighbor access pattern also conveys advantages by reducing the time to access a remote node. Assume that if the nearest neighbor is not in the same node, then the access time is Local memory access time + 200. Now find the optimal node size assuming 60% of the remote accesses are nearest neighbor.

6.16 [20/15/30] <6.5> 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: Processor count × Memory blocks. If memory is grown linearly with processor count, then 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 32–128 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 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. [20] <6.5> One method to obtain a scalable directory protocol is to organize the multiprocessor as a logical hierarchy with the processors at the leaves of the hierarchy and directories positioned at the root of each subtree. The directory at each subtree root records which descendents cache which memory blocks, as well as which memory blocks with a home in that subtree are cached outside of 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 incorporate both the number of nodes at each level of the hierarchy as well as the total number of nodes.

b. [15] <6.5> Assume that each level of the hierarchy in part (a) has a lookup cost of 50 cycles plus a cost to access the data or cache of 50 cycles, when the point is reached. We want to compute the AMAT (average memory access time—see Chapter 5) for a 64-processor multiprocessor with four-node subtrees. Use the data from the Ocean benchmark run on 64 processors (Figure 6.31) and assume that all noncoherence misses occur within a subtree node and that coherence misses are uniformly distributed across the multiprocessor. Find the AMAT for this multiprocessor. What does this say about hierarchies?

c. [30] <6.5> An alternative approach to implementing directory schemes is to implement bit vectors that are not dense. There are two such 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 of 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 of 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 two schemes. Assume a perfect cache for nonshared references, so as to focus on coherency behavior. Determine the number of extraneous invalidations as the directory cache size is increased.

6.17 [10] <6.3–6.6> Some systems do not use multiprocessing for performance. Instead they run the same program in lockstep on multiple processors. What potential benefit is possible on such multiprocessors?

✪ 6.18 [15] <6.7> Some multiprocessors have implemented a special broadcast coherence protocol just for locks, sometimes even using a different bus. Evaluate the performance of the spin lock in the example on page 596 assuming a write broadcast protocol.

6.19 [15] <6.7> Implement the barrier in Figure 6.39 on page 598, using queuing locks. Compare the performance to the spin lock barrier.

✪ 6.20 [15] <6.7> Implement the barrier in Figure 6.39 on page 598, using fetch-and-increment. Compare the performance to the spin lock barrier.

6.21 [15] <6.7> Implement the barrier in Figure 6.42 on page 602, so that barrier release is also done with a combining tree.

6.22 [15] <6.7> 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 snoopy write invalidate protocol.

✪ 6.23 [15] <6.7> Find the time for n processes to synchronize using a standard barrier. Assume that the time for a single process to update the count and release the lock is c.

6.24 [15] <6.7> Find the time for n processes to synchronize using a combining tree barrier. Assume that the time for a single process to update the count and release the lock is c.

6.25 [25] <6.7> Implement a software version of the queuing lock for a bus-based system. Using the model in the example on page 596, how long does it take for 20 processors to acquire and release the lock? You need only count bus cycles.

6.26 [15/15/15] <6.4, 6.5, 6.7> Prefetching is a technique that allows the "consumer" of data to request the data to its cache *before* it needs them. With multiprocessors, we can think of the "dual" of this technique where the "producer" of the data "evicts" the data from its cache *after* it is done with them. An extension of such "postflushes" could be to send the data to the next processor that needs the data, in cases where that can be determined.

a. [15] <6.4, 6.7> When is prefetching likely to be applicable? When is producer-initiated communication likely to be beneficial? Would producer-initiated communication be applicable in the context of the queuing locks and tree barriers discussed in Section 6.7?

b. [15] <6.7> Assume a shared-memory multiprocessor system that takes 100 cycles for a memory access and 300 cycles for a cache-to-cache transfer. A program running on this machine spends 60% of its time stalled on memory accesses and 20% of its time stalled on synchronization. Of these memory stalls, 20% are due to producer-consumer data access patterns where the writer of data can identify the processor that will read the value next. In these cases, producer-initiated communication can directly transfer data to the cache of the next processor needing the data. This will reduce the latency of these memory accesses from 300 cycles for a cache-to-cache transfer to 1 cycle for a cache hit. Another 30% of the memory stalls are due to migratory data patterns where data move from one processor to another, but the migration path is unclear to the source processor. In this case, a producer-initiated communication primitive, such as "postflush," can reduce the latency of the memory access from 300 cycles to 100 cycles. Further assume that all the synchronization is due to tree barriers and that the tree barrier overhead can be reduced by 40% with producer-initiated communication. Assuming no other overheads, what is the reduction in execution time with producer-initiated communication?

c. [15] <6.5> What memory system and program code changes are required for implementing producer-initiated communication?

6.27 [20/30] <6.2–6.7> Both researchers and industry designers have explored the idea of having the capability to explicitly transfer data between memories. The argument in favor of such facilities is that the programmer can achieve better overlap of computation and communication by explicitly moving data when they are available. The first part of this exercise explores the potential on paper; the second explores the use of such facilities on real multiprocessors.

a. [20] <6.2–6.7> Assume that cache misses stall the processor, and that block transfer occurs into the local memory of a DSM node. Assume that remote misses cost 100 cycles and that local misses cost 40 cycles. Assume that each DMA transfer has an overhead of 10 cycles. Assuming that all the coherence traffic can be replaced with DMA into main memory followed by a cache miss, find the potential improvement for Ocean running on 64 processors (Figure 6.31).

b. [30] <6.2–6.7> Find a multiprocessor that implements both shared memory (coherent or incoherent) and a simple DMA facility. Implement a blocked matrix multiply using only shared memory and using the DMA facilities with shared memory. Is the latter faster? How much? What factors make the use of a block data transfer facility attractive?

6.28 [10/12/10/12] <6.8> As discussed in Section 6.8, 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. [10] <6.8> At the end of the code segment, what is the value you would expect for C?

b. [12] <6.8> 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. [10] <6.8> If you wanted to make the system sequentially consistent, what are the key constraints you need to impose?

d. [12] <6.8> Assume 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 (e.g., the SPARC instruction set), 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?

6.29 [10/10/10] <6.9> The following results are seen from a simulation study of five floating-point benchmarks and two integer benchmarks from the SPEC92 suite. The branch misprediction rate nearly doubles from 5% to 9.1% going from 1 thread to 8 threads in an SMT processor. However, the wrong-path instructions fetched (on a misprediction) drops from 24% on a single-threaded processor to 7% on an 8-thread processor.

a. [10] <6.9> What causes the increase in branch misprediction rate?

b. [10] <6.9> Why is there a decrease in the number of wrong-path instructions fetched even though there is an increase in branch misprediction rates? (*Hint:* This is related to the scheduling of threads.)

c. [10] <6.9> Based on these numbers, what conclusions can you draw about the conflicts caused due to contention and interference on various resources in a multithreaded processor?

✪ 6.30 [15] <6.9> On page 612 one of the design challenges listed for SMT processors is ensuring that cache conflicts generated by the simultaneous execution of multiple threads do not cause significant performance degradation (often referred to as destructive cache interference). However, a simulation study of the online transaction-processing workload, OLTP, on an 8-threaded SMT processor shows a decrease in instruction cache miss rate from 14% to 9%. When is such a decrease, called constructive cache interference, likely to happen?

6.31 [15/15/10] <6.9> One of the important design decisions with an SMT processor is the heuristic to identify the "preferred thread." The following problem illustrates some of the challenges with this.

An eight-way SMT processor is running a multithreaded version of a parallel program. The processor uses the heuristic of giving preference to a thread that has the fewest instructions in the decode, rename, and instruction queues.

a. [15] <6.9> What are the advantages with this heuristic?

b. [15] <6.9> Craft a scenario where this heuristic may lead to a particular thread not being scheduled. (This is called starvation.)

c. [10] <6.9> What other heuristics can you think of to schedule the preferred thread?

6.32 [25] <6.10> 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 LRU replacement, and both caches have the same block size.

6.33 [15] <6.10> The key differences, with respect to a hardware shared-memory system, of a shared virtual memory (SVM) system are (1) the longer latency of remote memory accesses through the OS handler and over the LAN, and (2) the larger units of coherence (pages instead of cache blocks). How are these two key differences likely to affect the performance and operation of the system? What are ways to address each negative effect?

✪ 6.34 [10/15] <6.10> Consider a multiprocessor on a chip with four CPU cores. Each core has a 64 KB first-level cache, and the chip includes a 1 MB second-level cache.

a. [10] <6.10> If the system implements multilevel inclusion between the L1 and L2 caches, what is the upper bound on the capacity of the L2 that is wasted with duplicate data?

b. [15] <6.10> To avoid the potential waste of second-level cache capacity due to multilevel inclusion, the system designer may decide to forgo maintaining the inclusion property. In this case, blocks that are replaced in an L1 cache cause a write back to the L2 cache. What optimization discussed in Chapter 5 is this similar to? What other changes would you require to minimize contention for the L1 cache?

6.35 [20] <6.5, 6.11> As we saw in "Putting It All Together" and in "Fallacies and Pitfalls," data distribution can be important when an application has a nontrivial private data miss rate caused by capacity misses. This problem can be attacked with compiler technology (distributing the data in blocks) or through architectural support, as we saw in the description of CMR on Wildfire.

Assume that we have two DSM multiprocessors: one with CMR support and one without such support. Both multiprocessors have one processor per node, and remote coherence misses, which are uniformly distributed, take 1 μs. Assume that all capacity misses on the CMR multiprocessor hit in the local memory and require 250 ns. Assume that capacity misses take 200 ns when they are local on the DSM multiprocessor without CMR and 800 ns otherwise. Using the Ocean data for 32 processors (Figure 6.23), find what fraction of the capacity misses on the DSM multiprocessor must be local if the performance of the two multiprocessors is identical.

6.36 [15] <6.13> In contrast to the MXP chip that includes four similar MIPS 32 processors (discussed in Section 6.13), some embedded processors may support multiprocessing between nonuniform (heterogeneous) processor cores. For example, consider a DSP chip that includes a three-way issue master processor, two VLIW processors with support for special-purpose DSP primitives, and an intelligent DMA and memory controller, in addition to several on-chip DRAM banks. (Many commercial DSP processors exist with similar configurations; for example, the TI TMS 320C82.)

Discuss the trade-offs of writing parallel applications for such a heterogeneous system compared to writing parallel applications for the MXP configuration. What are the implications on binary software compatibility for future generations?

6.37 [Discussion] <6> 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 then 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 including maintaining an equivalent structure for the processor state, and so on. What are the accuracy/speed trade-offs between these approaches?

6.38 [Discussion] <6.11> Construct a scenario whereby a truly revolutionary architecture—pick your favorite candidate—will play a significant role. *Significant* is defined as 10% of the computers sold, 10% of the users, 10% of the money spent on computers, or 10% of some other figure of merit.

6.39 [30] <6.3–6.7, 6.11> Using an available shared-memory multiprocessor, see if you can determine the organization and latencies of its memory hierarchy. For each level of the hierarchy, you can look at the total size, block size, and associativity, as well as the latency of each level of the hierarchy. If the multiprocessor uses a nonbus interconnection network, see if you can discover the topology and latency characteristics of the network. Try to make a table like that in Figure 6.50 for the machine. The lmbench (*www.bitmover.com/lmbench/*) and stream (*www.cs.virginia.edu/stream/*) benchmarks may prove useful in this exercise.

6.40 [30] <6.3–6.7, 6.11> Perform Exercise 6.39 but looking at the bandwidth characteristics rather than latency. See if you can prepare a table like that in Figure 6.51. Extend the table by looking at the effect of strided accesses, as well as sequential and unrelated accesses.

6.41 [40] <6.2, 6.10, 6.14> A multiprocessor or cluster is typically marketed using programs that can scale performance linearly with the number of processors. The project here is to port programs written for one multiprocessor to the others and to measure their absolute performance and how it changes as you change the number of processors. What changes need to be made to improve performance of the ported programs on each multiprocessor? What is the ratio of processor performance according to each program?

6.42 [35] <6.2, 6.10, 6.14> Instead of trying to create fair benchmarks, invent programs that make one multiprocessor or cluster look terrible compared with the others, and also programs that always make one look better than the others. It would be an interesting result if you couldn't find a program that made one multiprocessor or cluster look worse than the others. What are the key performance characteristics of each organization?

6.43 [40] <6.2, 6.10, 6.14> Multiprocessors and clusters usually show performance increases as you increase the number of processors, with the ideal being *n* times speedup for *n* processors. The goal of this biased benchmark is to make a program that gets worse performance as you add processors. For example, this means that one processor on the multiprocessor or cluster runs the program fastest, two are slower, four are slower than two, and so on. What are the key performance characteristics for each organization that give inverse linear speedup?

6.44 [50] <6.2, 6.10, 6.14> Networked workstations can be considered multicomputers or clusters, albeit with somewhat slower, though perhaps cheaper, communication relative to computation. Port some cluster benchmarks to a network using remote procedure calls for communication. How well do the benchmarks scale on the network versus the cluster? What are the practical differences between networked workstations and a commercial cluster, such as the IBM-SP series?

6.45 [50] <6.3, 6.4, 6.5, 6.8> Implement parallel versions of two standard algorithms—matrix multiply and mergesort—for a shared-memory architecture that supports a relaxed memory consistency model. You will have to decide on a suitable partitioning of the computation, a suitable data layout across the processors, and implement the necessary synchronization to ensure correctness. Use a publicly available simulator, such as RSIM (*www.ece.rice.edu/~rsim*), to measure the speedups you get for various processor sizes. Experiment with various cache sizes, different latency parameters, and different working set sizes. Experiment with different cache coherence protocols. Vary the parameters to model both UMA and NUMA systems. How does that affect your experiments?