Problem 1. (25 points) [CS&G 6.1] Consider two machines \( M_1 \) and \( M_2 \). \( M_1 \) is a four-processor shared-\( L_1 \)-cache machine, whereas \( M_2 \) is a four-processor bus-based snooping-cache machine. \( M_1 \) has a single shared 1-BM two-way set-associative cache with 64-byte blocks, whereas each processor in \( M_2 \) has a 256-KB direct-mapped cache with 64-byte blocks. \( M_2 \) uses the Illinois MESI coherence protocol. Consider the following piece of code:

```c
double A[1024, 1024]; /* row-major; 8-byte elements */
double C[4096];
double B[1024, 1024];

for (i=0; i<1024; i+=1) /* loop-1 */
  for (j=myPID; j<1024; j+=numPEs)
for (i=0; i<1024; i+=numPEs) /* loop-2 */
  for (j=0; j<1024; j+=1)
    { A[i, j] = (B[i+1, j] + B[i-1, j] + B[i, j+1] + B[i, j-1]) / 4.0; }
```

(a) Assume that the array \( A \) starts at hexadecimal address 0x0 (i.e., hexadecimal address 0), array \( C \) at 0x800000 and array \( B \) at 0x808000. All caches are initially empty. Each processor executes the preceding code, and \( \text{myPID} \) varies from 0 to 3 for the four processors. Compute misses for \( M_1 \) separately for the two loop nests. Do the same for \( M_2 \), stating any assumptions that you make.

On either machine, the arrays are too large to fit in cache, so both loops behave as if the cache is empty. The cache is large enough to hold the working set of three rows of the source matrix and one row of the target matrix. Given the presence of \( C \), row \( A[i] \) conflicts with \( B[i-4,*] \) on \( M_2 \). (They map to the same set on \( M_1 \).) Thus, cold misses on \( A \) will know out old rows of \( B \). A cache block holds 8 elements, however, work is assigned in cyclic fashion is each row (or distributed in cyclic column). Each block holds 8 elements, so a given processor computes two elements per block.

(b) Briefly comment on how your answer to part (a) would change if array \( C \) were not present. State any other assumptions that you make.

(c) What can be learned about advantages and disadvantages of shared-cache architecture from these exercises?

Problem 2. (25 points) [CS&G 7.4, 7.7] In order to solve the following problems, you need to carefully study Example 7.1 (p. 460 of Culler, Singh, & Gupta) first.

(a) Reconsider Example 7.1 where the number of hops for an \( n \)-node configuration is \( \sqrt{n} \). How does the average transfer time increase with the number of nodes? What about \( \frac{3}{2} \sqrt{n} \)?
(b) Reconsider Example 7.1 where the network is a simple ring. The average distance between two nodes on a ring of \( n \) nodes is \( \frac{n}{2} \). How does the average transfer time increase with the number of nodes? Assuming each link can be occupied by at most one transfer at a time, how many such transfers can take place simultaneously?

**Problem 3.** (10 points) [CS&G 8.10] A pattern of sharing that might be detected dynamically is a producer-consumer pattern, in which one processor repeatedly writes (produces) a variable and another processor repeatedly reads (consumes) it. Is the standard MESI invalidation-based protocol well suited to this? Why or why not? What enhancements or protocol might be better, and what are the savings in latency or traffic? How would you dynamically detect and employ the changes?

**Problem 4.** (15 points) (a) Like the perfect shuffle, the butterfly permutation by itself does not give a completely connected network. Show how a butterfly interconnection partitions 4- and 8-node networks.

(b) Since the perfect shuffle did not provide a complete connection, the exchange was added to the network, as discussed on page 10 of Lecture 23. By looking at the bit-representations of the source and destination nodes, what “property” does the perfect shuffle permutation have which prevents complete connection (as an obvious observation)? Does this property hold for the butterfly permutation as well, and if so, how? Will the introduction of an exchange permutation into the butterfly interconnection network provide a complete connection, as it did with the perfect shuffle? If so, prove it. If not, would it work for any special cases (e.g., networks of a particular size \( N \)?

(c) Would adding only the super-butterfly permutations be a way to provide a complete connection with a maximum path length of \( O(\log N) \)?

**Problem 5.** (25 points) This problem deals with a sample interconnection network, similar to the mesh network on pp. 5–6 of Lecture 23. Consider a unidirectional mesh interconnection network of size \( n \times n \). The routing functions are:

\[
R_{+1}(i) \equiv (i + 1) \mod N, \quad \text{where } N = n^2 \\
R_{+n}(i) \equiv (i + n) \mod N.
\]

For example, if \( n = 3 \), the network would be as shown at the right.

(a) What is the greatest distance between processors if \( n = 3 \)?

(b) What is the average distance between any two processors?

(c) Repeat parts (a) and (b) with \( n = 4 \).

(d) For networks of different sizes, give a general formula for the maximum and average distances between two processors in terms of \( n \).

(e) List some advantage(s) and disadvantage(s) of this interconnection network vs. a standard mesh interconnection network.