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-MB 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) 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 $myPID$ 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. (20 points) [Hennessy & Patterson 7.2] In this exercise, assume that $T_{base} = 10$ and $T_{loop} = 15$. Also assume that the store latency is always included in the running time.

Consider this vector code run on an 80-Mhz version of DLXV for a fixed vector length of 64. Ignore all strip-mining overhead, but assume that the store latency must be included in the time to perform the loop. The entire sequence produces 64 results.

```assembly
LV V1,Ra
MULTV V2,V1,V3
ADDV V4,V1,V3
SV Rb,V2
SV Rc,V4
```
(a) Assuming no chaining and a single memory pipeline, how many clock cycles per result (including both stores as one result) does this vector sequence require?

(b) If the vector sequence is chained, how many clock cycles per result does this sequence require?

(c) Suppose DLXV had three memory pipelines and chaining. If there were no bank conflicts in the accesses for the above loop, how many clock cycles are required per result for this sequence?

**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. (20 points)** In many cases with data-intensive algorithms, the data in the system needs to go through phases. Typical phases are the producing phase, the transforming phase, and the consumption phase. Suppose we have an MP system where the phases are distributed to different processors. Also, suppose we are looking at one particular set of data, and each processor could concurrently be dealing with many more data in different phases.

Assume that all the variables are shared and initialized as follows:

```plaintext
int A = 0;
int B = 0;
boolean produced = false;
boolean transformed = false;
boolean consumed = true;
```

The following program is given for three processors, one for each phase.

<table>
<thead>
<tr>
<th>Producer</th>
<th>Transformer</th>
<th>Consumer</th>
</tr>
</thead>
</table>
| 1. while (! consumed);  
2. A = 1;  
3. B = 1;  
4. produced = true; | 1. while (! produced);  
2. A = A + B;  
3. B = 0;  
4. transformed = true; | 1. while (!transformed);  
2. print(A);  
3. print(B);  
4. consumed = true; |

(a) Show the output for one pass of these processes under the following memory models:

(i) SC – Sequential Consistency
(ii) PRAM – Pipelined RAM
(iii) PC – Processor Consistency

If there is a difference between PRAM and PC please explain why.

(b) Make necessary adjustments to the programs for each processor if RC (Release Consistency) is used. Make necessary assumptions and introduce variables (if needed).

(c) What would be the outcome if the RC model used in (b) were—

(i) ERC (Eager Release Consistency)
(ii) LRC (Lazy Release Consistency)?
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:

- $R_{x1}(i) \equiv (i + 1) \mod N$, where $N = n^2$
- $R_{xn}(i) \equiv (i + n) \mod N$

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.