Midterm II
SOLUTIONS
December 1, 2003
CS252 Graduate Computer Architecture

<table>
<thead>
<tr>
<th>Problem</th>
<th>Possible</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>25</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>25</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>30</td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>100</td>
<td></td>
</tr>
</tbody>
</table>
[ This page left for $\pi$ ]

3.141592653589793238462643383279502884197169399375105820974944
Problem #1: Checking the bits

The error correction coding process can be viewed as a transformation from one space of bits (the unencoded data) to another (the coded data).

1a) For **linear block codes**, there are two matrices, the **Generator** (G) and **Parity Check** (H). What are they, how big are they (in terms parameters n and k), and how are they used?

The **Generator**, G, is a $k \times n$ matrix used to produce code words from data words: Let $d$ be a $k$-bit data vector and $C$ be the resulting $n$-bit code word. Then $C = G \cdot d$. **H** is an $n \times (n-k)$ matrix used to check the parity of a code word and produce a syndrome indicating where errors are located. Let $S$ be an $(n-k)$-bit vector, then $S = H \cdot C$.

1b) Define the minimum Hamming distance, $d_{\text{min}}$, of an error correction code (make sure that this definition makes sense for codes like Reed-Solomon of RAID-5 that have more than one bit per symbol):

$d_{\text{min}}$ is the minimum number of positions (symbols) of difference between two valid code word. If the symbols are bits, this is the number of different bit positions, but can be number of different disk blocks for something like RAID-5.

1c) Name a constraint on the columns of the parity check matrix (H) which would indicate that a code had minimum distance $d_{\text{min}}$.

If a code with parity check matrix $H$ has minimum distance $d_{\text{min}}$, then every combination of $d_{\text{min}}-1$ columns of $H$ must be independent. This means that no combination of $d_{\text{min}}-1$ columns can be combined to yield zero. Similarly, no combination of $d_{\text{min}}-2$ columns, $d_{\text{min}}-3$ columns, etc. This also means that every column is unique and non-zero.

1d) For a code with $d_{\text{min}}$ minimum distance, what is the formula for the maximum number of errors, $E_{\text{detect}}$, that can be reliably detected? Why might you be able to detect more than the $E_{\text{detect}}$ errors?

$$E_{\text{detect}} = d_{\text{min}} - 1.$$ 

You might be able to detect more if the code words are not evenly distributed. However, more simply, you can almost always detect a number of errors that is not evenly divisible by $d_{\text{min}}$. 


1e) For a code with $d_{\text{min}}$ minimum distance, what is the formula for the maximum number of random errors, $E_{\text{correct}}$, that can be corrected reliably?

$$E_{\text{detect}} = \left\lfloor \frac{d_{\text{min}} - 1}{2} \right\rfloor$$

1f) How does your answer to (1e) change if you have knowledge of which symbols are bad (i.e. which symbols have been erased)? How might erased symbols be detected?

*If you know which symbols are bad, then you can often correct up to $d_{\text{min}} - 1$ errors. Erased symbols might be detected by some other means such as checksums over individual symbols (especially makes sense for large symbols such as disk blocks) or even another intra-symbol-level error code.*

1g) What is a systematic error-correction code? Why is this desirable?

*Systematic codes are ones in which the original data appears directly within the code words; i.e. one can extract the original data directly from the code words without any mathematical operations. This is desirable because a receiver of information can immediately start computing on the data in parallel with the syndrome computation (to see if the data is correct).*

1h) Suppose we start with a non-systematic, linear code described by generator $G$ with distance $d_{\text{min}}$. How can we produce a systematic code with generator $G'$ from $G$ that still has distance $d_{\text{min}}$? (hint: what happens if you subtract two code words from the original code? Now think of this operation on $G$).

*The rank of $G$ must be $k$ in order to provide a valid code (since there need to be as many code words as there are data vectors). Consequently, by doing Gaussian elimination on the columns (addition and subtraction of multiples of the columns from one another), plus a possible swapping of rows, you can arrange to have a $k \times k$ identity matrix at the top of the resulting matrix $G'$. Note that Gaussian elimination on the columns is valid because sums and differences of code words are still code words (in a linear code). Swapping columns is valid because it just rearranges the order of the bits in the code word.*
1i) Suppose you are willing to use 5 parity bits. What is the maximum number of data bits ($k$) that you can protect with a Hamming code and still get a distance 4 code? What are the matrices $G$ and $H$ for this code? Prove that your code has distance 4.

To have a distance-4 code, we need to produce an $H$ matrix in which (1) all the columns are non-zero, (2) pairs of columns of $H$ do not sum to zero (meaning that every column is distinct), and (3) triples of columns of $H$ do not sum to zero; the easiest way to get this last constraint is to make sure that all columns are odd-weight (have an odd number of ones). With $(n-k)=5$ parity bits, the number of odd-weight column vectors is $2^{5-1}=16$. This means that $n \leq 16$. Thus, $k=16-5=11$ data bits max. The matrices are as below (subscripts indicate matrix sizes):

Let $P_{11 \times 5} = \begin{bmatrix} 1 & 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 \end{bmatrix}$

Let $I_{n \times n} = n \times n$ identity

Then $H_{16 \times 5} = [P_{11 \times 5} \ I_{5 \times 5}]$ and $G_{1 \times 16} = [I_{11 \times 11} \ P_{11 \times 5}]$

Proof is simple: $H$ adheres to the 3 conditions above: every row is non-zero and unique (giving a distance-3 code). In addition, every column is odd, so combinations of 3 columns will result in a non-zero code. Consequently, given a code word with errors: $C' = C + \varepsilon$ the syndrome $S = H \cdot C'$ will be non-zero for all error vectors $\varepsilon$ with 3 or fewer bits set. This means that we have a distance 4 code: if we had two code words $C_1$ and $C_2$ that differed by less than 4 bits, then $0 = S = H \cdot (C_1 - C_2) \neq 0$ by the properties of $H$; this is a contradiction.

1j) Explain the function that a circuit that detected and corrected errors in the above code would have to perform. You should tell how to (1) get an “Error detected” signal, $E_{\text{detect}}$, (2) get a double-error signal, $E_{\text{double}}$, and (3) a bad bit identifier, $E_{\text{bad}}$. Note that $E_{\text{detect}}$ and $E_{\text{double}}$ are Boolean (true/false) signals while $0 \leq E_{\text{bad}} < n$ is a number that defines which bit is bad.

This is straightforward. The circuit would take the input code word $C'$ and compute the syndrome: $S = H \cdot C'$. Let $W_S$ be the weight (number of ones) in the resulting (n-k)-bit syndrome. Then, $E_{\text{detect}} \iff W_S \neq 0$. $E_{\text{double}} \iff W_S$ even. Finally, you can think of $E_{\text{bad}}(S)$ as a function from $S$ to a 4-bit value. Let $H_x$ be the $x$'th column of $H$. Then:

$E_{\text{detect}} = x$ if $S = H_x$ for some $x$ and is undefined otherwise. This function is well-defined since every column of $H$ is unique and could be generated as a combinational circuit. For example, if $S = (1 1 1 1 1)$, then $E_{\text{detect}} = 0$, etc. (we could certainly number the bits from the right side of $H$ and call this $E_{\text{detect}} = 15$).
Problem #2: Queuing the DRAM

Suppose that we have a processor connected through a general network to a single DRAM “server”. Suppose that the computation on each of these processors have the following average behavior, and that each of the individual processors are completely uncorrelated (so that the combined behavior appears memoryless):

Instruction Mix: 20% loads, 10% stores, 20% floating-point, 20% integer, 30% branches
Base CPI without cache misses: 1.6.

Now, assume that each processor has a single-level of cache, with separate I and D caches. Assume a write-back cache, 8-word cache lines, a 99.5% hit rate on instructions, a 90% hit rate on loads and a 80% hit rate on stores. Cache hits happen at 1 per cycle with no stalls. Also, assume a write-allocate policy, and that 12.5% of the cache lines in the data cache are dirty and must be written back to memory on a miss. Assume no self-modifying code.

Assume that the network is one word (32-bits) wide and that it operates at 1 word/processor cycle bandwidth. When a cache line is loaded into the cache from the network or sent from the cache back into the network, the processor is tied up during the time that the data is actually transferred.

A one-way network trip to or from the DRAM takes 10 processor cycles. Assume that the DRAM can handle only one request at a time and that it has a service time of 6 processor cycles for the first word + 2 processor cycle for each additional word.

2a) What is the maximum rate at which the DRAM can service requests? (in accesses per processor cycle)? Do not account for the network yet, just the DRAM “server”:

We know from the above that cache lines are 8 words and that the DRAM takes 6 cycles for the first word and 2 cycles for each additional word. Because this is a write-back cache, the DRAM only services cache-line sized jobs. So, \( T_{\text{service}} = 6 + (8 \times 1) \times 2 = 20 \) cycles. So, maximum rate is 1 request every 20 cycles, or 0.05 requests/cycle.

2b) Assume no queuing in the network or DRAM (only one request outstanding at a time). The network and cache are pipelined for fills, however. Also, assume a single processor with a blocking cache. Assume that the time for sending data into the network or taking data from the network must be accounted for (i.e. when filling or replacing data in the cache). What is the penalty that a processor suffers when requesting data from memory?

Since there is no queuing, we need the unloaded time to send a request to the dram, for the dram to process this request, and for data to make it back to the cache. Since the network and cache are pipelined, we can send words into the network as soon as they come out of the DRAM. Thus, the time for the last word to be in the cache is:

\[ \text{Penalty} = 10 + 20 + 10 = 40 \text{ cycles}. \]
2c) How long does it take to flush a dirty cache line from the processor cache into the network?

8 cycles: network bandwidth of 1 word/cycle with an 8-words cache line.

2d) Taking (2b) and (2c) into account, what is the CPI for the processor with a blocking cache? 

*Hint: Don’t forget instruction misses and dirty flushes from cache back to memory.*

The trick here is to note that the time to handle a cache fill from memory in a blocking cache is suffered by the processor for both read and write misses (in a write-allocate cache, a write miss first pulls in the complete cache from memory like a read-miss). Further, every so often a cache miss will evict a dirty cache line (12.5% of the time), causing an additional delay of 8 cycles (2c) to get the data into the network on its way to the DRAM. So, we can compute several DRAM penalties as follows:

- Cache\text{fill} = 40 \text{ (from 2b)}
- Cache\text{evict} = 8 \text{ (from 2c)}
- Penalty\text{inst} = 40 \text{ Instruction cache never dirty (no self-modifying code)}
- Penalty\text{read} = Penalty\text{write} = 40 + 0.125 \times 8 \text{ Dirty 12.5\% of the time}

CPI = CPI\text{BASE} + (Miss\text{inst} \times Penalty\text{inst}) + (0.2 \times Miss\text{read} \times Penalty\text{read}) + (0.1 \times Miss\text{write} \times Penalty\text{write})

= 1.6 + (0.005 \times 40) + (0.2 \times 0.1 + 0.1 \times 0.2) \times (40 + 0.125 \times 8)

= 1.6 + 0.2 + 1.64 = 3.44

2e) What is the RATE (in accesses per processor cycle) at which requests are made to the DRAM?

We want (Requests/Instruction)/(cycles/instruction) = Requests/cycle = \lambda

Requests/Instruction = number of requests that make it into the network. Note that this includes cache fill requests (from reads and writes) as well as flushes of dirty lines. To be explicit, a read-miss which flushes a dirty cache line will cause 2 requests: one to read the new line and one to flush the dirty line. Consequently, every cache miss in the data cache causes 1.125 requests on average.

Requests/Instruction = 0.005 \text{ (instructions)} + (0.2 \times 0.1 + 0.1 \times 0.2) \times 1.125 \text{ (reads/writes)}

= 0.05 \text{ Requests/Instruction}

So, \lambda = 0.05/3.44 = 0.0145 \text{ Requests/cycle}

2f) What is the DRAM utilization? Can this ever be greater than one without queuing? Why or why not?

Utilization = \xi = \lambda\times T_{\text{service}} \times 0.145 \times 20 = 0.290

This can never be greater than one with a blocking cache because we wait for each request to finish before launching another request. Thus, the DRAM is always idle during the network communication times; Hence, the DRAM is not 100\% utilized.
2g) Next, assume that we have queuing of requests at the DRAM. Assume that we have a non-blocking cache that can issue many requests (unbounded) without stalling. Assume that the processor can make forward progress with an outstanding instruction miss (weird assumption, but makes math easier). Finally, assume a fully pipelined network, but not fully pipelined DRAM. Cache fills and replacements still must happen at network bandwidth, but now we can overlap the network time.

Question: What is the rate of requests into the network now? What is the DRAM utilization? (Hint: now, the processor is only stalled for the time that data is transferred to/from the network, not during the network latency).

Since we haven’t changed the program or cache-miss behavior of the cache, the Requests/instruction doesn’t change (still 0.05 from 2e).

So, it is the CPI that changes. The only difference from our computation in 2d is that the penalties now only include the network bandwidth time in and out of the cache (which stalls the processor). So, we have:

\[ \text{Penalty}_{\text{inst}} = 8 \text{ and } \text{Penalty}_{\text{data}} = 8 \times 1.125 = 9 \]

Following the computation in 2d, we now have:

\[ \text{CPI} = 1.6 + 0.005 \times 8 + (0.2 \times 0.1 + 0.1 \times 0.2) \times 9 = 1.6 + 0.04 + 0.36 = 2.0 \]

\[ \lambda = \frac{\text{Requests/Instruction}}{(\text{Cycles/Instruction})} = 0.05/2.0 = 0.025 \]

Thus, Utilization = \( \xi = 0.025 \times 20 = 0.5 \)

2h) Now, assume that the processor requests are exponentially distributed. The DRAM service time is deterministic (i.e. not exponentially distributed). What is the average time that a request spends at the DRAM, including any queuing time?

Since DRAM is deterministic, we need to set \( C = 0 \) (no standard deviation of service time)

\[ T_{\text{DRAM}} = T_{\text{queue}} + T_{\text{service}} = \frac{(1 + C)}{2} \left( \frac{\xi}{1 - \xi} \right) T_{\text{service}} + T_{\text{service}} \]

\[ = \frac{1}{2} \left( \frac{0.5}{1 - 0.5} \right) 20 + 20 = 30 \]

2i) What is the average number of cache misses for load instructions that the processor will have outstanding at any one time (hint: remember Little’s law and the average time to service a request as seen by the processor)?

In general, Little’s law can be written: \( L_{\text{sys}} = \lambda \cdot T_{\text{sys}} \) for any system. Here, we think about the complete queuing system of network+DRAM. All requests take the same time through this system: \( T_{\text{sys}} = T_{\text{network}} + T_{\text{DRAM}} = (10 + 10) + 30 = 50 \). Here, \( \lambda \) is the rate at which loads cause cache misses. So, \( \lambda = (0.2 \times 0.1)/\text{CPI} = 0.02/2.0 = 0.01 \) load requests/cycle.

So: Average number of read misses outstanding: \( 0.01 \times 50 = 0.5 \).
Problem #3: Producing a Disk Subsystem

Suppose that we build a disk subsystem to handle a high rate of I/O by coupling many disks together. Properties of this system are as follows:

- Uses 10GB disks that rotate at 10,000 RPM, have a data transfer rate of 10 MBytes/s (for each disk), and have a 8 ms seek time, 4 KByte block size
- Has a SCSI interface with a 2ms controller command time.
- Has an average I/O size of 32 KByte
- Is limited only by the disks
- Has a total of 20 disks

Each disk can handle only one request at a time, but each disk in the system can be handling a different request. The data is not striped (all I/O for each request has to go to one disk).

3a) What is the average service time to retrieve a single disk block from a random location on a single disk, assuming no queuing time (i.e. the unloaded request time)?

Keep in mind that, for data, KB=2^{10} and MB=2^{20}.

\[
T_{service} = T_{controller} + T_{seek} + T_{rotational} + T_{transfer} = 2\text{ms} + 8\text{ms} + T_{rotational} + T_{transfer}
\]

\[
T_{rotational} = \frac{1}{2} \text{time for 1 rotation} = \frac{1}{2} \times (60 \text{ s/min}) \times (1/10,000 \text{ min/revolution}) = 3\text{ms}
\]

\[
T_{transfer} = 2^{12} \text{bytes} / (10 \times 2^{20} \text{bytes/s}) = (2^{-8}/10) \times 1000\text{ms} = 0.390625\text{ms}
\]

\[
T_{service} = 2 + 8 + 3 + 0.39 = 13.39\text{ms}
\]

3b) Assuming that the OS makes a series of single block requests (waiting for one to complete before requesting the next one), what is the bandwidth that can be achieved?

We can get 4192 bytes every 13.39 ms = 4192/0.01339 bytes/second = 313 Kbytes/sec

Note that bandwidth is often measured with K=1000, MB=1000000. (But not always – it’s a weird world)

3c) What is the service time for a 32 KByte sequential request? What is the bandwidth that can be achieved on consecutive requests now?

Only difference from 3a is \(T_{transfer}\) piece which is now = 8 times as much = 8 \times 0.39 = 3.125 ms.

So, \(T_{service} = 16.125\text{ms}\)  Bandwidth = 32768/0.016125 = 2.032 Mbytes/sec
3d) Given the average I/O size of 32KB again, what is the maximum number of I/Os per second (IOPS) for the whole disk system?

One disk can handle \( \frac{1}{0.016125} = 62.02 \) requests/second

Total system IOPS = \( 62.02 \times 20 = 1240.4 \) requests/second

Suppose that we decide to improve the system by using new, better disks. For the same total price as the original disks, you can get 11 disks that have 19GB each, rotate at 12000 RPM, transfer at 12MB/s and have a 6ms seek time.

3e) What is the average unloaded service time for random 32KByte requests in the new system?

\[
T_{\text{service}} = T_{\text{controller}} + T_{\text{seek}} + T_{\text{rotational}} + T_{\text{transfer}} = 2\text{ms} + 6\text{ms} + T_{\text{rotational}} + T_{\text{transfer}}
\]

\[
T_{\text{rotational}} = \frac{1}{2} \text{ time for 1 rotation} = \frac{1}{2} \times (60 \text{ s/min}) \times (1/12,000 \text{ min/revolution}) = 2.5\text{ms}
\]

\[
T_{\text{transfer}} = \frac{2^{15} \text{ bytes}}{(12 \times 2^{20} \text{ bytes/s})} = (2^{-5}/12) \times 1000\text{ms} = 2.60\text{ms}
\]

\[
T_{\text{service}} = 2 + 6 + 2.5 + 2.60 = 13.10\text{ms}
\]

3f) What is the maximum number of IOPS in the new system?

\[
IOPS = 11 \times (1/0.01310) = 839.7 \text{ requests/second}
\]
Treat the entire system as a M/M/m queue (that is, a system with m servers rather than one), where each disk is a server. All requests are in a single queue. Assume that both systems receive an average of 800 I/O requests per second. Assume that any disk can service any request.

3g) What is the mean response time of the old system? The new one? You might find the following equation for an M/M/m queue useful:

\[
\text{Server Utilization (} \zeta \text{)} = \frac{\lambda}{1 - m} = \lambda \times \frac{\text{Time}_{\text{server}}}{m}
\]

\[
\text{Time}_{\text{queue}} = \text{Time}_{\text{server}} \times \left[ \frac{\zeta}{m(1 - \zeta)} \right]
\]

The solution to the problem is a simple matter of plugging in the right values:

\[
\zeta_{\text{old}} = 800 \times 0.016125/20 = 0.645
\]

\[
\zeta_{\text{new}} = 800 \times 0.01310/11 = 0.953
\]

\[
\text{Time}_{\text{oldsystem}} = \text{Time}_{\text{oldserver}} \times \left[ \frac{\zeta}{m(1 - \zeta)} \right] + \text{Time}_{\text{oldserver}}
\]

\[
= 16.125\text{ms} \times \left[ 1 + \frac{0.645}{20(1 - 0.645)} \right] = 17.59\text{ms}
\]

\[
\text{Time}_{\text{newsystem}} = \text{Time}_{\text{newserver}} \times \left[ \frac{\zeta}{m(1 - \zeta)} \right] + \text{Time}_{\text{newserver}}
\]

\[
= 13.10\text{ms} \times \left[ 1 + \frac{0.953}{11(1 - 0.953)} \right] = 37.25\text{ms}
\]

3h) Which system has a lower average response time? Why?

*First system has much lower response time. Even though the disks are slower, there are almost twice as many disk heads. Consequently, the old system can handle many more aggregate IOPS, greatly reducing the queuing time.*
Question #4: Grab Bag

4a) What are the 3 C’s of cache-miss classification? What is the 4th C of multiprocessing?

“Compulsory” misses, “Capacity” misses, and “Conflict” misses. The 4th C is for “coherence” misses.

4b) What is the difference between access time and cycle time in DRAM? Why should there be two such numbers?

“Access time” is the time to retrieve information from an idle DRAM. “Cycle time” is the minimum rate at which successive access can be launched. (actually, these numbers usually refer to random accesses – if you access successive items in a DRAM row, you can do so faster than the cycle time). There are two such numbers because the DRAM needs to recover from each access before starting the next (“recover” means pre-charging the bit lines and resetting the sense-amps). Thinking of the recovery time as being after the access finishes leads to two numbers with Access Time < Cycle Time.

4c) Explain why DRAM must be refreshed. What does the memory system have to do in order to refresh DRAM?

DRAM stores information as charges on capacitors. These charges leak into the surrounding Silicon; consequently, the values stored on these capacitors must be refreshed periodically to retain their values. Typically, the memory system performs a refresh by reading each DRAM row in succession; the act of reading a row will refresh all bits in that row.

4d) How does utilizing multiple DRAM banks help the overall throughput of a memory system? Why should a prime number of banks be desirable?

The use of multiple DRAM banks permits some of the banks to be in the process of recovering while others are accessing. Consequently, a system with multiple DRAM banks can process more overall requests/unit time. A prime number of banks can ensure that the load of requests for each memory bank is roughly balanced for a wide variety of different memory strides (particularly useful for Vector processors).
4e) What is a victim cache, what does it help, and why is it a better engineering alternative than other options?

A victim cache is a small, fully-associative cache placed below a normal cache. When items are evicted from the primary cache, they are placed into the victim cache before being discarded entirely. Items in the victim cache can be retrieved more quickly than from the DRAM. Victim caches help reduce the penalty of thrashing and thus increase the effective associativity of the primary cache. A large direct-mapped cache + victim cache may be a faster and cheaper alternative than a higher-associativity cache with similar performance.

4f) How can parallelism (such as in a vector processor) be used to reduce total energy consumed by a computation? Why doesn’t a superscalar processor get this advantage?

The vector instruction set exposes a lot of parallelism. To reduce energy for applications that have fixed throughput requirements (such as multimedia applications), we can (1) increase the computational resources by some factor, \( n \), then (2) decrease the clock rate by the same factor, and (3) reduce the voltage as much as possible while still running at the new clock rate. The first two maintain the throughput of the system and will balance out in energy (since you still need to do the same number of total operations to compute something, so that energy is balanced). Because the system is running more slowly, this enables (3), which will reduce the energy per operation.

A superscalar processor doesn’t typically get this advantage because the overhead of scheduling (reorder buffer, reservation stations, renaming, commitment, etc), burns more energy than you save from the parallelism.

4g) What is the AMAT performance equation for a system with 2-levels of cache and DRAM?

\[
AMAT = Time_{Hit_{L1}} + MissRate_{L1} \times (Time_{Hit_{L2}} + MissRate_{L2} \times MissPenalty_{L2}).
\]

4h) List three different ways in which the compiler can reduce cache misses in a program.

Here are several:
1. Rearrange code to avoid thrashing in the instruction cache
2. Merging separate arrays together to increase spatial locality
3. Splitting loops into more nests to Block cache operations (eg: matrix multiplication)
4. Rearranging loops to get stride-1 accesses in the inner loop (more spatial locality)
5. Inserting prefetch operations into a program to avoid misses
4i) Explain the difference between coherence and consistency:

“Coherence” is about the behavior of a single location as seen by multiple processors. “Consistency” is about the order of updates between different memory locations as seen by multiple processors. A multiprocessor is coherent if (1) every update is eventually seen by every processor and (2) these updates are seen in the same order. There are many types of consistency, one of which is “sequential consistency” as defined in 4j below.

4j) What does it mean for a multiprocessor to be sequentially consistent? Why is this a desirable requirement for a multiprocessor?

A multiprocessor is sequentially consistent if for every actual run of a shared-memory program there is a sequential interleaving of the program instructions on a uniprocessor that would produce the same result, i.e., which (1) preserves the program order of every processor and (2) preserves the dataflow of communication between processors.

Sequential consistency is desirable because it is “intuitive”, i.e., is the behavior that people tend to expect when thinking about a parallel execution. It is also the behavior that one would get from a multithreaded execution on a single processor.

4k) Name the three conditions sufficient for sequential consistency that were presented in class and draw a diagram illustrating these conditions:

1. All processors issue memory requests in order.
2. If a processor has issued a store, it will wait for the store to complete before it issues another request.
3. If a store is in flight, a processor will wait for the store to complete (be visible to all processors) before allowing a load to that same address to see the new value.

![Diagram]

4l) What is release consistency, and why was it proposed as an alternative to sequential consistency?

Release consistency is a consistency mechanism in which programmers must protect shared memory regions with locks: they first “acquire” the lock, make their shared reads and writes, then “release” the lock. This was proposed as an alternative to sequential consistency because it allows the “ordinary accesses” (non-lock accesses) to be reordered and placed into write buffers.
[This is another blank page for you to use]