EECS 570 End Term Exam - Solutions
Winter 2025

Name: _____ Uniqname: _____

Sign the honor code:

I have neither given nor received aid on this exam nor observed anyone else doing so.

_____

Scores:

#	Question	Points
1	Short Answers	/ 25
2	Systolic Array	/ 15
3	On-Chip Network Topology	/ 20
4	GPU Architecture	/ 15
5	GPU Programming	/ 9
6	L2 Cache Coherence	/ 16
Total		/ 100

NOTES:

- **Calculators are allowed, but no PDAs, portables, cell phones, etc.**
- **Don't spend too much time on any one problem.**
- **You have 120 minutes for the exam.**
- **There are 12 pages in the exam (including this one). Please ensure you have all pages.**
- **Be sure to show work and explain what you've done when asked to do so.**

1. Short Answers [25 points]

a) State one advantage and one disadvantage of adaptive routing compared to deterministic routing. [3 points]

Advantage: Avoid congested or faulty areas

Disadvantage: Additional hardware/logic, can lead to deadlock/livelock if not designed properly

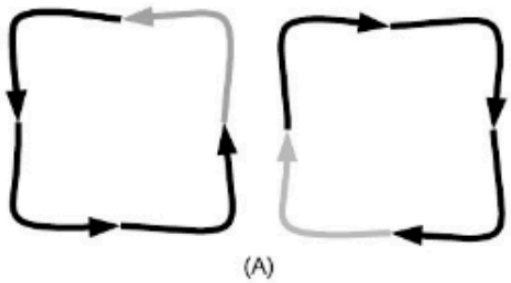
b) State two uses for virtual channels. [3 points]

Prevent Deadlocks
Reduce head-of-line blocking, thus increasing network bandwidth

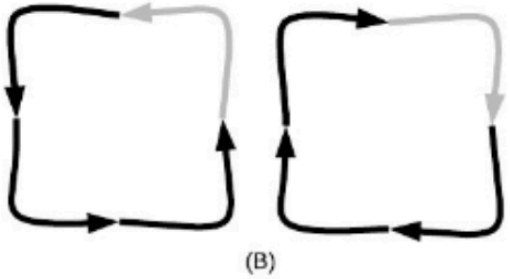
c) Consider a 16-node 2D torus interconnect with minimal (shortest-path) routing. Assume a uniform random traffic pattern, where each node sends packets to all nodes, including itself, with equal probability. What is the average number of hops a packet traverses, counting 0 hops for messages sent to itself? [4 points]

0 Hops: 1/16
1 Hop: 4/16
2 Hops: 6/16
3 Hops: 4/16
4 Hops: 1/16
Avg: (0*(1) + 4*(1) + 6*(2) + 4*(3) + 1*(4))/16 = 2 Hops

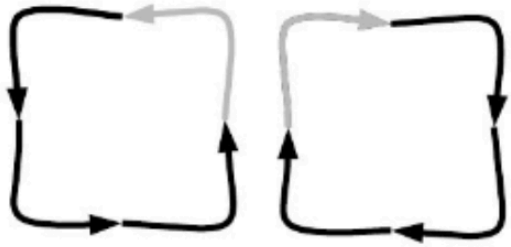
c) The following four figures show various routing turn models, in which the grayed-out turns are forbidden. Classify each of the following turn models(s) as deadlock or deadlock-free (circle the right answer) [8 points]



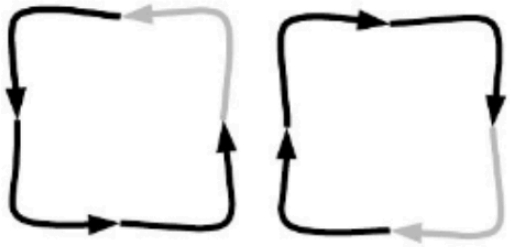
(A)



(B)



(C)



(D)

A: Deadlock / Deadlock-free

B: Deadlock / Deadlock-free

C: Deadlock / Deadlock-free

D: Deadlock / Deadlock-free

d) Briefly explain a non-predictive mechanism that GPUs use to hide memory latency. [4 points]

GPUs achieve memory latency hiding via massive fine-grained multithreading. When a warp (a group of 32 threads) encounters a long-latency memory operation, the GPU's warp scheduler simply switches to another warp that is ready to execute keeping the processing units busy. The register values for all threads remain in the register file, with no expensive OS context switching. This latency hiding is critical to the GPU's performance model because global memory access can take hundreds of cycles, and have simpler control logic with no branch prediction.

e) Explain one solution to reduce the number of pipeline stages in a router microarchitecture [3 points]

Lookahead routing, Speculation/Bypassing.

2. Analyze a 3×3 Systolic Array [15 points]

You are given a 3×3 systolic array that can multiply two 3×3 matrices, A and B.

$$C_{i,j} = \sum_{k=0}^{2} a_{i,k} \cdot b_{k,j}$$

Dataflow:

Each element in the input matrix moves from one PE to another as follows:

- Each row of A flows **horizontally** across the array (from left to right).
- Each column of B flows **vertically** down the array (from top to bottom).

Processing Elements (PEs):

Every cycle,

- Each PE **multiplies** the two new input values it receives from its neighbours.
- Each PE **accumulates** the product into a local register that holds the partial sum.
- The **partial sum** stays **inside** the same PE (output stationary)

A PE does not compute in a cycle when it doesn't receive valid inputs.

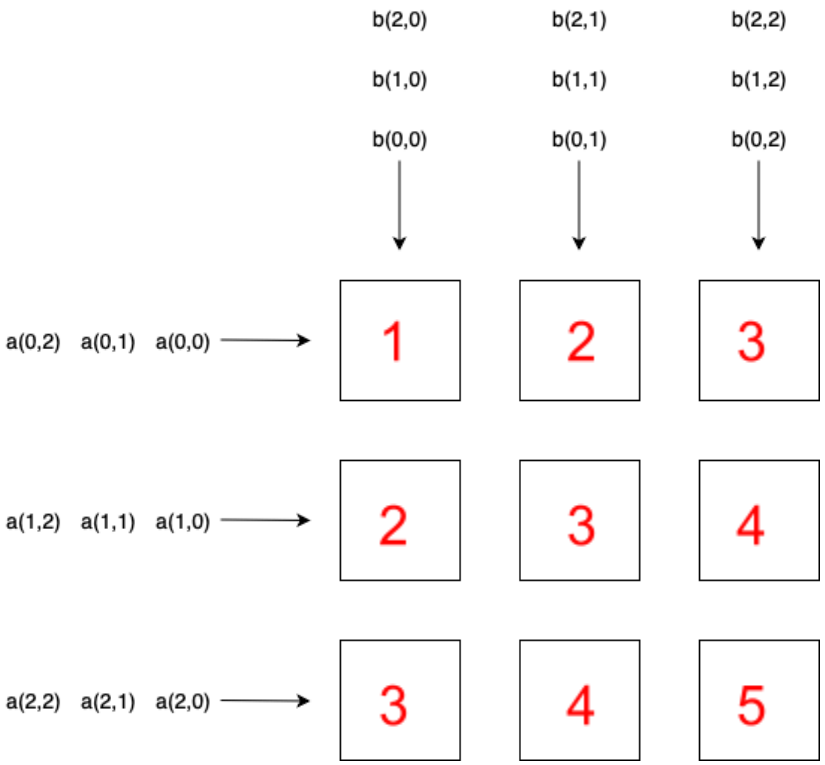
Completion:

Once all the input matrix data has passed through the systolic array, PE(i,j) contains the final sum for C(i,j).

Assume:

- At time t = 0, the systolic array is completely empty.
- Example: At t=1, PE(0,0) computes A(0,0) * B(0,0), and adds it to its partial sum.

(a) Determine the cycle in which PE (i,j) will compute the first valid product and accumulate it to its partial sum. Fill the following grid with those cycle numbers. Briefly explain. [8 points]



Row i of A and Col i of B need to have the start offset = i cycles so that the PEs get the correct inputs for partial products. Since each hop is 1 cycle, we get the numbers mentioned above.

Some of the answers mention the cycle numbers when the PE finishes computing C(i,j). We've given full credits if the reasoning matches the answer in this case.

(b) Compute the latency of one matrix multiplication operation. Explain. [3 points]

7 cycles

(c) Compute the throughput of the systolic array in terms of MAC (multiply-accumulate) per cycle. [4 points]

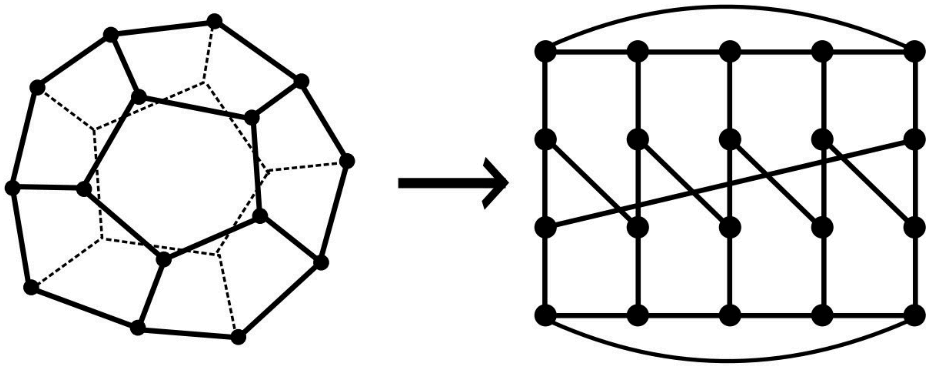
Average throughput for one Matrix Multiplication : 27/7 MAC/cycle

Average throughput assuming inputs keep streaming in: 9 MAC/cycle

We've given credits for both answers.

3. On-Chip Network Topology [20 points]

Several topologies have been investigated for on-chip networks. One interesting proposal calls for designing NoCs based on a dodecahedron.



A dodecahedron is a polyhedron (three-dimensional polygon) with twelve faces, wherein each face forms a regular pentagon. The left figure shows a 3D dodecahedron as seen from above, dotted lines indicating edges on the hidden faces. A router can be placed at each of the 20 vertices of the shape. The right figure illustrates an equivalent planar topology–how the topology can be laid out in a regular grid on a 2D plane. This is how an actual chip using such a topology would be connected.

Compare the 20-node dodecahedral NoC against a 20-node 5×4 mesh network.

(a) List two advantages that the dodecahedral NoC has over the mesh network. [4 points]

symmetry, path diversity, less congestion, load balance, lower avg/max hops, lower latency, fewer links, lower/uniform router degree

(b) Calculate the longest paths (in terms of hops) for both topologies and explain how this impacts each design. [4 points]

Dodec diameter: 5 hops
Mesh diameter: 7 hops

better routing performance, lower latency, lower power

(c) Design a deadlock-free routing algorithm for the dodecahedral topology. Assume you have only one message class. You can use at most two virtual channels.

Hint: Reason using following directions for packets in 3 dimensions: top, bottom, clockwise, counter-clockwise. [12 points]

Algorithm:

Visualize Dodec as three rings (of sizes 5, 10, 5) with 'vertical' through links.
Dimension-order routing: going along ring first then via 'vertical' links, or vice-versa.
VC assignment: different VCs for up/down and clockwise/counter-clockwise directions.

Argue why your routing algorithm is deadlock-free.

Acyclic paths with dimension-order routing + dateline or escape VC within rings.

4. GPU [15 points]

(a) Why do GPUs have thousands of small cores instead of a few powerful cores like Intel CPUs? [3 points]

GPUs are designed to be throughput oriented. This means that the aggregate work done across all cores is prioritized over the single-thread latency of any single core.

(b) State one performance metric where GPU underperforms a CPU [2 points]

Latency.

(c) Describe briefly how GPUs enable fast warp-level switching [3 points]

GPUs have *large registers* that store the architectural register states of all "live" warps scheduled on a GPU core. This enables the hardware to context switch from one warp to another, without having to save (old thread's) and restore (new thread') register states from memory through a software OS routine.

(d) Specify **two main** reasons why a similar functionality (fast thread context switch) is less beneficial for CPUs, so much so that none of the commercially available CPU hardware have that support. [3 points]

Some answered by stating the reasons for why context switches in CPUs are expensive. Or pointed out that they are rare, and so fast context switch is not useful. But the question is, what if you had a fast context switch like in GPU that allowed you to hide memory latency by switching between threads (which would increase CS frequency)? Why would that be less beneficial? The key insight here is that CPUs at their core (pun intended) are latency-oriented rather than throughput-oriented like GPUs. There were several correct answers that stemmed from this observation. Such answers include:

Typical CPU workloads consist of fewer threads with complex control flow. The limited thread pool available in most CPU applications wouldn't justify the hardware investment, as there simply aren't enough threads to keep the switching mechanism busy.

Even though fast context switching would immediately help existing applications by speeding up the work typically handled by the OS (and potentially even more than that as we can image over time programs would start to take advantage of this support more often) implementing it would require significant trade-offs. With die area being extremely limited, this would likely require smaller caches, simplified branch predictors, or reduced out-of-order execution capabilities. For latency-sensitive CPU workloads, these trade-offs would likely degrade overall performance more than the benefits gained from fast switching.

(e) A GPU has a memory bandwidth of 900 GB/s and can perform 20 TFLOPS (20×10^12 floating-point operations per second). Calculate the arithmetic intensity threshold (in FLOPS/byte), above which an application is compute-bound. [4 points]

Threshold = Compute Capability / Memory Bandwidth
= 20 × 10^12 FLOPS / (900 × 10^9 B/s)
= 20,000 GFLOPS / 900 GB/s
= 22.22 FLOPS/byte

5. CUDA Programming [9 points]

(a) Consider the following CUDA kernel that processes an array of integers:

```cuda
__global__ void conditionalVectorAddition(int idx, int* data, int* result) {
    int idx =      // assume idx is correctly
                   // initialized using blockDim, blockIdx, threadIdx
                   //  such that thread_i will have idx = i

    if (idx < data[idx]) {
        result += data[idx];
    }
}
```

Assume this kernel launches a single block with a size of 8 threads (smaller than actual warps) and the following values in the `data` array:

[5, 3, 8, -6, 11, 9, -1, 14]

a) Calculate the SIMD utilization efficiency for this warp. Briefly explain. (SIMD utilization is the percentage of threads that are actively doing useful work during execution) [4 points]

There are 6 threads {0, 1, 2, 4, 5, 7} where the condition(idx < data[idx]) is true and 2 threads {3, 6} where it's false. When executing the conditional branch, only 6 out of 8 threads are doing useful work while the other results are predicated away. Therefore, the SIMD utilization efficiency is 6/8 = 75%.

Some students answered 6/32=18.75% assuming that the SIMD width was 32 which is typical for actual GPUs. The instructions were intended to mean both the block size launched AND the SIMD width were 8, but such answers were accepted as correct.

(b) Consider a CUDA kernel that processes a matrix of dimensions width × height, where each thread computes a single element of the matrix.

Write an expression for the corresponding row and column indices (i, j) in the matrix that this thread should process. Show your work. [5 points]

Notes:
- You may refer to the CUDA variables: blockDim, blockIdx, threadIdx
- The formula should ensure each thread processes a unique matrix element

i = blockIdx.x * blockDim.x + threadIdx.x
j = blockIdx.x * blockDim.x + threadIdx.x

6. L2 Cache Coherence in a 4-Core System [16 points]

Consider a 4-core system with a 3-level inclusive memory hierarchy, consisting of L1, L2, and Main Memory. Each core has a private L1 cache, while L2 caches are shared as follows:

- Core 0 and Core 1 share one L2 cache
- Core 2 and Core 3 share another L2 cache

The system employs a bus-based snooping MSI protocol at each cache level to maintain coherence. There are two buses in the system:

- L1 Bus: Connects the L1 caches within the same L2 domain.
- L2 Bus: Connects the L2 caches and interfaces with main memory.

To differentiate bus transactions occurring at each level, we use suffixes:

`_L1` for L1 bus transactions;
`_L2` for L2 bus transactions.

For example

`BusRd_L1` represents a Bus Read (BusRd) request on the L1 bus
`BusRd_L2` represents a Bus Read (BusRd) request on the L2 bus.

List of Possible Bus Actions

L1 Bus Transactions

FwdBusRd_L1

- Forwarded request due to BusRd on L2. If an L1 cache line is in the Modified state, it transitions to Shared and provides the data on the bus.

Following transactions are the same as taught in lectures:

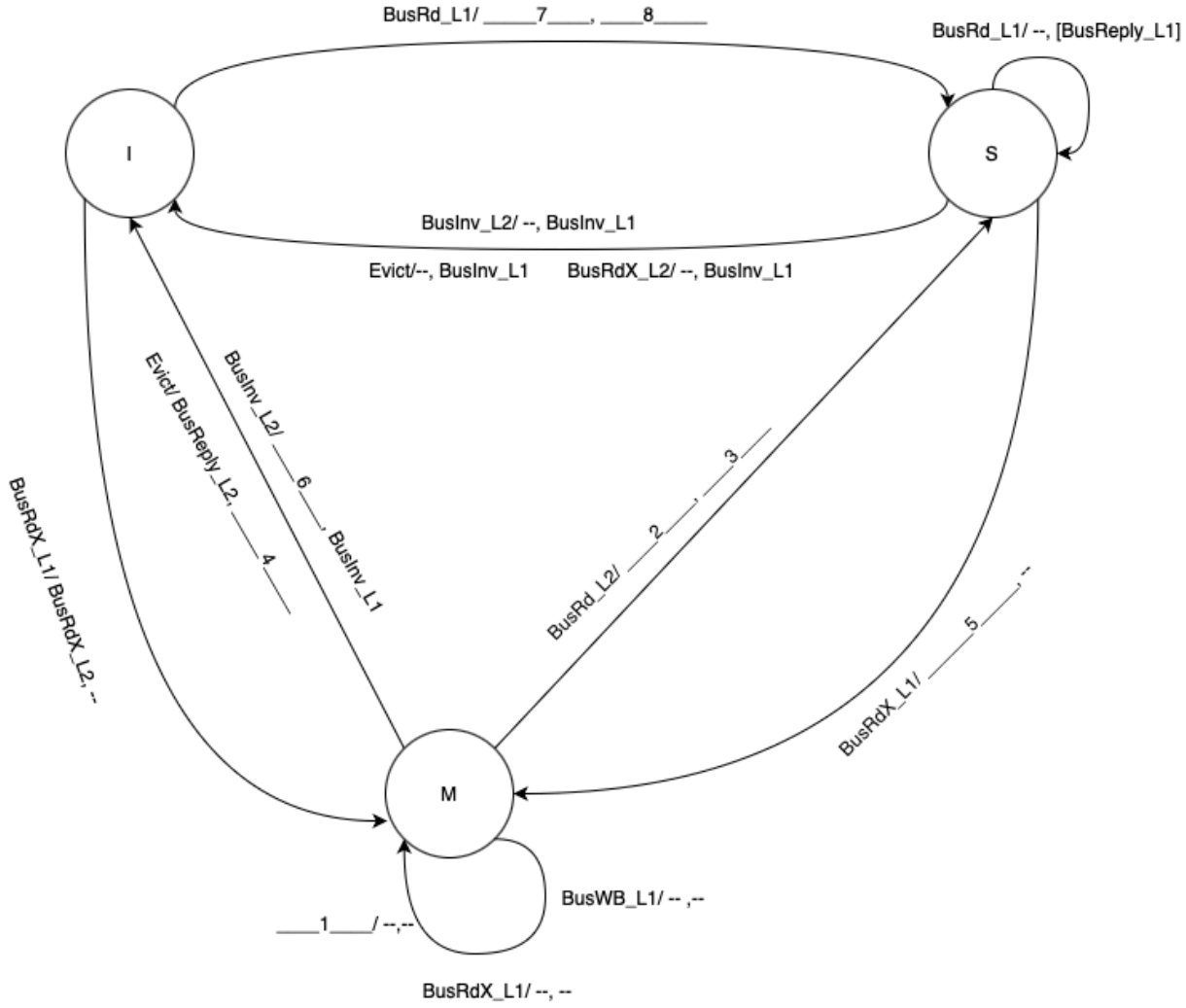
BusRd_L1, BusRdX_L1, BusWB_L1, BusInv_L1, BusReply_L1

L2 Bus Transactions

Following transactions are the same as taught in lectures

BusRd_L2, BusRdX_L2, BusWB_L2, BusInv_L2, BusReply_L2

Fill in the blanks for the state transition diagram of an **inclusive** L2 cache. Consider each transition to be atomic. Each transition in the state diagram is labeled as: ***Action / L2 bus transaction, L1 bus transaction***. '--' is used if no bus transaction occurs.



1: BusRd_L1

2: BusReply_L2

3. FwdBusRd_L1

4: BusInv_L1

5: BusInv_L2

6: BusReply_L2

7: BusRd_L2

8: BusReply_L1