EECS 570 Final Exam
Winter 2020

Name: _______________________________        Uniqname: ___________________

Sign the honor code:

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

______________________________

Scores:

<table>
<thead>
<tr>
<th>#</th>
<th>Question</th>
<th>Points</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Directory Coherence</td>
<td>/ 20</td>
</tr>
<tr>
<td>2</td>
<td>On-Chip Network Topology</td>
<td>/ 20</td>
</tr>
<tr>
<td>3</td>
<td>Memory Consistency</td>
<td>/ 25</td>
</tr>
<tr>
<td>4</td>
<td>Multiple Network-On-Chip</td>
<td>/ 25</td>
</tr>
<tr>
<td>5</td>
<td>Cache Coherence Verification</td>
<td>/ 10</td>
</tr>
<tr>
<td></td>
<td><strong>Total</strong></td>
<td>/ 100</td>
</tr>
</tbody>
</table>

NOTES:

- Open book, open notes. You can refer to lectures and papers in the reading list.
- You cannot use Internet, refer papers not included in the reading list, or discuss problems/solutions with anyone.
- Don’t spend too much time on any one problem.
- You have 24 hours for the exam.
- There are 15 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) **Directory Coherence**  

[20 points]

In a directory-based cache coherence protocol, invalidation and forward requests incur significant latency and bandwidth. Design a new directory coherence protocol, TCoherence, that has the following features:

- **Write doesn't require any invalidation messages**, i.e. no messages sent to sharers when a node wants to write.
- **Read/write requests are not forwarded** by the directory to a node with read/write permission.

To design TCoherence, you should take advantage of a globally synchronized clock.

Globally synchronized clock: Assume each node in a multiprocessor system has access to a precise clock. It is accurate to a single processor clock cycle. The clocks across all the nodes are synchronized, and so they all provide the same count at any given instant.

Similar to a conventional directory protocol, TCoherence should enable read-sharing (allows multiple nodes to read a cache block concurrently). It should have a modified (M) state that gives write permission to a single node. It should also allow nodes to exploit temporal locality.

(a) Explain your overall solution using figures, if necessary. Be explicit about how you are taking advantage of synchronized clock.  

[16 points]

➢ Answer questions i-iii to explain your solution in depth.
i) What state, if any, is maintained at the directory? How does this improve storage requirements of a conventional directory with full sharer lists?

ii) How does a node complete a read request? How does your scheme allow multiple sharers?

iii) How does a node complete a write request? How does your scheme guarantee exclusive access while a node has write permission?
(b) Describe a memory access pattern for which TCoherence will be better than invalidation-based coherence protocol. How? [4 points]
2) **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 $5 \times 4$ mesh network.

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

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

Argue why your routing algorithm is deadlock-free.
3. Memory Consistency [25 points]

(a) You are asked to architect SC-C++ language, a C++ variant that guarantees SC to the programmers. But you only have x86 (TSO variant) hardware to run your programs on. How would you change an existing C++ compiler (DRF0 compliant) to support the new standard? Assume your compiler cannot do any additional dataflow analysis. We will refer to this compiler as SC-on-All. [4 points]

(b) What is the difference between an SC-preserving compiler (Marino et al. PLDI 2011) and the SC-on-All compiler you designed for question (a)? State the difference in terms of compiler guarantees. Which compiler do you expect to provide higher performance, and why? [4 points]
(c) What hardware guarantee do you need to support SC-C++ language standard using an SC-preserving compiler? [2 points]

(d) DrMagic has developed a powerful new compiler analysis tool called RacerX that can identify all data-races in a program (no false negatives), but may also report a few false positives (memory accesses that can never race are reported as racy). How would you use RacerX to improve the efficiency of SC-on-All compiler (the one that guarantees SC on x86 for C++ programs) from question (a)? State why this new solution could be more efficient, and point out sources of remaining SC overhead compared to the original DRF0-compliant compiler for C++. [6 points]

Improve SC-on-All using RacerX:

Sources of efficiency compared to naive SC-on-All compiler from (a):

Sources of remaining overhead in SC-on-All compared to a DRF0-compliant compiler:
(e) Assume that you are given a compiler-hardware system that supports transactional memory (TM). Explain how you would use TM to detect data-races at runtime (while executing a program). Give an example to illustrate a data-race that manifests at runtime, but may be missed by your TM based data-race detector. Do you benefit from TM’s rollback feature? [5 points]

(f) Can you use TM to support SC-C++? Explain a solution that would work for most programs. Are there programs that your solution may not work, and if so, provide a simple code example. [4 points]
4. **Multiple Network-On-Chip**  

Consider a 3x3 mesh NoC. Its router uses a conventional 5 stage router pipeline, and supports 4 virtual channels (VCs). Each link is 32 bit wide. We will call this as **OneNet**.

You are asked to design and analyze a multiple network-on-chip, FourNet. FourNet has four 3x3 mesh networks (subnets). It has four times the number of routers as OneNet. However, **FourNet’s aggregated router buffer area and link wires are same as OneNet** -- these resources are equally divided among the subnets. For example, if OneNet’s router has a 4 KB input buffer, then a router in FourNet’s subnet has 1 KB input buffer. A node can inject and receive a packet into any of the four subnets. Once a packet is injected into a subnet, all of its flits travel in the same subnet till they reach their destination.

(a) **OneNet** has 32-bit links. What should be the link width of a subnet in FourNet? How would that affect the size of flits and packet size in terms of number of flits compared to OneNet?  

[3 points]
(b) Assume OneNet used 4 VCs to ensure protocol-level deadlock freedom by assigning dependent message classes to different VCs. Do you need VCs in FourNet for deadlock-freedom? Explain. [3 points]

(c) Assume four VCs in each subnet. Roughly compare the size/area of OneNet and FourNet routers’ buffers, switches, switch and VC allocators. Explain your rationale. [4 points]

Size of each input VC buffer:

Crossbar Switch:

Switch allocator:

VC allocator:
(d) If you were to support only one VC per subnet, roughly compare the size/area of OneNet and FourNet routers’ buffers, switches, switch and VC allocators. Explain your rationale. [4 points]

Size of each input VC Buffer:

Crossbar Switch:

Switch allocator:

VC allocator:

(e) Consider these configurations: FourNet with 4 VCs vs FourNet with 1 VC. Which one do you expect to provide higher throughput, and why? [3 points]

(f) Assume that a node can inject a packet into any of the four subnets. If a router in FourNet can sense congestion in a subnet, then the source node can inject a packet into the subnet...
that is least congested. Describe two different heuristics that a router can use to predict congestion in a subnet. Assume that a router can communicate only with its neighbors to measure congestion.

[g] If a component in a processor has been idle, it can be switched off (power-gated) to save energy. However, power-gating can result in performance loss, as you may need to wait 10s of cycles to wake-up a component when you need it.

Which of the two configurations, OneNet or FourNet, is better suited for power-gating, and why?
5. **Cache Coherence Verification**  [10 points]

A major challenge in designing cache coherence protocols is with scaling their verification for modern many-core systems. The number of system states explodes as we increase the number of processors. An even more fundamental problem is that adding more nodes exposes *new behavior* which is otherwise not exercised in a smaller system. As a result, a protocol is often verified only for its instances upto a certain $N$ nodes.

**Design for scalable verifiability:**

One idea is to view the system as a *logical* tree structure as follows:

- **Basic node** is a leaf in the tree (grey squares). It is a set of cores with their caches and memories.
- **Interface nodes** are the internal nodes (yellow elliptical) that logically compose the behavior of its child nodes.

A top interface at the root node exposes the behavior of the entire sub-system to the external system. Multiple basic nodes joined by a top or an internal interface forms a Level_1 node. Recursively, two or more Level_n-1 nodes joined by a top or an internal interface compose a Level_n node, $n$ being the height of the tree. Assume a binary tree.

The key idea is to identify a *minimal system* structure that can be composed into *any* larger system by expanding the basic node(s) into another instance of a component of the minimal system. If this minimal system is verified to be cache coherent *and* the *interface preserves self-equivalence*, then the system is formally verified to be cache coherent with arbitrary number of nodes.

(a) **Design the interface behavior for self-equivalence** for MSI coherence protocol by answering the following.

For a cache block, how should the *logical* coherence protocol state (M/S/I) for an interface node be assigned based on the coherence state of its children? It should capture the equivalent coherence states of all its child nodes.  

**[5 points]**

**Hint:** What should be the *logical* protocol state of a Level_1 interface node if its left child node is Modified and its right child node is Invalid? Think through other possible combinations.
For a cache block, what additional information (e.g. owner, sharers) must be stored in the interface node for each coherence state (M/S/I)?

(b) For a system using binary trees, reason why the following structure is the **minimal system**. You should argue (i) how it is possible to compose an arbitrarily large system using only components of this structure, and (ii) why that is not possible using any smaller structure.  [5 points]

Hint: How would you compose a system with 6 nodes? Drawing this may help your reasoning.