EECS 570 Final Exam — Solution
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>Total</td>
<td></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]

See Tardis ‘15

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 does not 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.

Lease coherence:
- each requestor is ‘leased’ a cache block for a fixed duration after which it must self-invalidate.
- multiple read leases can be granted at a time, but only one write lease at any time.
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]

Single writer and multiple sharers, no need to wait for invalidations.
Migratory sharing pattern with delay between reads and writes.
 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]

- Dodeca diameter: 5 hops  
  - Better routing performance, lower latency, lower power
- Mesh diameter: 7 hops

See Dodec ‘14
(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.**
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]

Compiler inserts a fence between any pair of store-load, the only memory ordering relaxed by TSO hardware compared to SC.

Compiler should be made SC-preserving -- that is, its compiler optimizations should not violate SC at the language-level.

(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]

Guarantee:

SC-preserving compiler guarantees that its optimizations preserve SC. If we run its output binary on SC hardware, we can guarantee SC-C++.

SC-on-All compiler from (a) is SC-preserving. In addition, it guarantees SC-C++ on weaker hardware (TSO) by inserting additional fences.

Performance:

SC-on-All incurs higher overhead, as it has to obey all the constraints of SC-preserving compiler. Plus, it adds overhead (fences) to ensure SC on weaker TSO hardware.
(c) What hardware guarantee do you need to support SC-C++ language standard using an SC-preserving compiler? [2 points]

SC hardware.

(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:

Use RacerX to detect data-races, and only insert fences between a store-load pair, if either the store or the load is racy. This reduces the number of fences that need to be executed.

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

Eliminates memory ordering overhead for memory accesses that are proven to be data-race-free.

Sources of remaining overhead in SC-on-All compared to a DRF0-compliant compiler:

RacerX can report false positives (false data races). A given program may also have data-races. These accesses need fences.

(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]
Compiler transforms a given program by making every synchronization-free-region (SFR) as a transaction. This can be done by starting a transaction at thread start, and ending a transaction at the thread end. Any synchronization operation in the middle is compiled as follows:

```c
sync_op() → transaction_end;
    sync_op();
    transaction_begin;
```

At runtime, any transaction conflict indicates a potential data-race. Since we are not interested in enforcing transactional semantics, we **don’t need to rollack** to recover from these conflicts.

This solution may report false (conflicts) data-races due to false sharing. It may also miss data-races if two racing transactions are not executed concurrently (i.e., don’t overlap in time):

```c
x_begin
    x++
x_end

y_begin
    y++
y_end
```

**(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]

If we use the solution from (g), and require that TM semantics are enforced, and then we get SC-C++. Because, every memory access is part of some transaction, and TM system guarantees that the transactions are serializable and preserves program order.

This solution may not work for some C++ programs with data-races that cause cyclic dependency between SFRs (transactions). Here is an example. Assume \( X = Y = 0 \) initially.

```c
lock(L)
    Y = 1;
    while(X == 0);
unlock(L)
lock(M)
    X = 1;
    while (Y == 0);
unlock(M)
```

If we turn these SFRs into transactions, the system would deadlock, as there is no valid execution that guarantees serializable order between these two transactions. However, there is a valid SC behavior for this program that is deadlock-free. One could solve this problem by dynamically recompiling the program by removing transactions around conflicting region, and inserting a fence between every pair of memory accesses within that region.

Alternatively, compiler can place every memory access within a new transaction. This will work, but very inefficient. It is similar to making all memory accesses to be atomic/volatile accesses, but adds unnecessary TM start/commit overhead.
4. **Multiple Network-On-Chip** [25 points]  

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]**

Link width of a subnet in FourNet: 32-bits/4 = **8 bits**.

Flit size:

> In an on-chip network, a flit is typically the same size as a phit (link width = 8 bits). Refer Jerger’s synthesis lecture. Thus, flit size in FourNet is **8 bits -- 1/4th of OneNet flit.**

Packet size in terms of flits:

> Packet size doesn’t change. So a packet in FourNet has **4x more flits** than OneNet.
(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]

No. We are given that 4 VCs are sufficient for deadlock freedom in OneNet. We can establish 1-1 mapping between a VC in OneNet to a subnet in FourNet, and assign all the message classes mapped to a VC in OneNet to its corresponding subnet in FourNet. Such an assignment will guarantee deadlock freedom.

(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:

FourNet’s input VC buffer is 4x smaller than OneNet’s. Because, there are 4x number of routers in FourNet, and the number of VC per router is same as OneNet.

Crossbar Switch:

Link width of FourNet’s subnet is 4x smaller (part (a)). A crossbar switch’s size in a router is quadratic w.r.t link width. Therefore, FourNet’s crossbar switch is 16x smaller than OneNet’s.

Switch allocator:

Number of input VCs and output links are the same in both designs. So, roughly the same.

VC allocator:

Number of VCs and output links are the same in both designs. So, roughly the same.
(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:

It is given that the aggregate buffer area is the same. So, each FourNet router has \( \frac{1}{4} \) th the buffer size of a OneNet router. However, FourNet router has only one buffer. OneNet router has 4 buffers -- one for each VC. Thus, the input buffer size is the same in both designs.

Crossbar Switch:

**Same answer as 4(c).** Link width of FourNet's subnet is 4x smaller (part (a)). A crossbar switch’s size in a router is quadratic w.r.t link width. Therefore, FourNet's crossbar switch is 16x smaller than OneNet's.

Switch allocator:

OneNet has 4 VCs. So, it needs a separable switch allocator with two stages. First set of arbiters select a winner input VC for each input link. These winners then compete in the second stage, where each arbiter select at most one winner per output port.

**Since FourNet has only one VC, there is no need for the first set of arbiters.**

VC allocator:

Unlike OneNet, since FourNet has no VCs, it does not need a 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]

FourNet with 4 VCs. **VCs help reduce head-of-line blocking**, even in multiple networks. But throughput benefits of VCs in FourNet may not be as significant as OneNet.
(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. [4 points]

There are many metrics to predict congestion. Some of them are:

Buffer occupancy of either the local router, or adjacent routers. Buffer occupancy could be measured in many ways: current status, average over the last N cycles, max. over the last N cycles, etc.

Measure the number of packets injected by the local node over the last N cycles.

Measure the number of packets transmitted by a router over the last N cycles.

(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? [4 points]

FourNet is better suited for power-gating, because it has higher path diversity than OneNet. Even when a router in a subnet is put to sleep, other subnets retain the connectivity/path between two nodes. Under low network load, 3 subnets (3/4th of the network routers) can be power-gated without losing connectivity. However, in OneNet, even under low network load, intermittent traffic between nodes may keep most of the routers active.
5. Cache Coherence Verification [10 points] See Fractal Coherence ‘10

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.

<table>
<thead>
<tr>
<th>Child 1</th>
<th>Child 2</th>
<th>Interface</th>
</tr>
</thead>
<tbody>
<tr>
<td>M</td>
<td>I</td>
<td>M</td>
</tr>
<tr>
<td>S</td>
<td>I/S</td>
<td>S</td>
</tr>
<tr>
<td>I</td>
<td>I</td>
<td>I</td>
</tr>
</tbody>
</table>
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)?

Composite state of the interface
M: identity of owner
S: list of sharers

(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.

The minimal system has two components: single basic node □, and Level_1 node □
(i) Structural Induction:
assume n node system composed only using components of given structure
replace any one basic node with the Level_1 interface node to compose n+1 node system

(ii) If either component is absent, the structural induction breaks.