ETH 263-2210-00L Computer Architecture, Fall 2021
HW 5: Interconnects, Multiprocessors & Exam Practice

Instructor: Prof. Onur Mutlu
TAs: Juan Gómez Luna, Mohammed Alser, Jisung Park, Lois Orosa Nogueira, Gagandeep Singh, Haiyu Mao, Behzad Salami, Nour Almadhoun Alserr, Mohammad Sadr, Hasan Hassan, Can Firtina Geraldo Francisco De Oliveira Junior, Abdullah Giray Yaglikci, Rahul Bera, Konstantinos Kanellopoulos, Nika Mansouri Ghiasi, Rakesh Nadig, João Dinis Ferreira, Haocong Luo, Roknoddin Azizibarzoki

Given: Sunday, Dec 5, 2021
Due: **Wednesday, Dec 22, 2021**

- **Handin - Critical Paper Reviews (1).** You need to submit your reviews to `https://safari.ethz.ch/review/architecture21/`. Please, check your inbox, you should have received an email with the password you should use to login. If you did not receive any email, contact comparch@lists.inf.ethz.ch. In the first page after login, you should click in "Computer Architecture Home", and then go to "any submitted paper" to see the list of papers.
- **Handin - Questions (2-10).** You should upload your answers to the Moodle Platform (`https://moodle-app2.let.ethz.ch/mod/assign/view.php?id=678325`) as a single PDF file.
- If you have any questions regarding this homework, please ask them the Moodle forum (`https://moodle-app2.let.ethz.ch/mod/moodleoverflow/view.php?id=675990`).
- Please note that the handin questions have a hard deadline. However, you can submit your paper reviews till the end of the semester.

1. Critical Paper Reviews [1,000 points]

You will do at least 4 readings for this homework, of which 3 are tagged as **REQUIRED**. You may access them by *scanning or clicking on the QR codes below.*

Required 1 Required 2 Required 3

Write an approximately one-page critical review for the readings (i.e., papers from #1 to #3 **and** *at least* 1 of the remaining 17 papers, from #4 to #20). If you review a paper other than the 4 mandatory papers, you will receive 250 BONUS points on top of 1,000 points you may get from paper reviews (i.e., each additional submission is worth 250 BONUS points with a possibility to get up to 4000 bonus points).

Please read the guideline slides for reviewing papers and watch Prof. Mutlu's guideline video on how to do a critical paper review. We also provide you with sample reviews which you can access using the QR code. A review with bullet point style is more appreciated. Try not to use very long sentences and paragraphs. Keep your writing and sentences simple. Make your points bullet by bullet, as much as possible. **We will give out extra credit that is worth 0.5% of your total course grade for each good review.**

Guideline Slides Guideline Video Sample Reviews

1. **(REQUIRED)** S. Koppula, L. Orosa, A. G. Yaglikci, R. Azizi, T. Shahroodi, K. Kanellopoulos, and O. Mutlu, *"EDEN: Enabling Energy-Efficient, High-Performance Deep Neural Network Inference Using Approximate DRAM,"* in *MICRO*, 2019. https://people.inf.ethz.ch/omutlu/pub/EDEN-efficient-DNN-inference-with-approximate-memory_micro19.pdf
2. **(REQUIRED)** T. Moscibroda and O. Mutlu. *"A Case for Bufferless Routing in On-Chip Networks"* in *ISCA* 2009 https://people.inf.ethz.ch/omutlu/pub/bless_isca09.pdf
3. **(REQUIRED)** R. Das, O. Mutlu, T. Moscibroda, and C.R. Das. *"Aergia: Exploiting Packet Latency Slack in On-Chip Networks"* in *ISCA* 2010 https://people.inf.ethz.ch/omutlu/pub/aergia_ieee_micro_top_picks11.pdf
4. M.J. Flynn, *"Very high-speed computing systems,"* in *Proceedings of the IEEE* 1966 https://safari.ethz.ch/architecture/fall2021/lib/exe/fetch.php?media=flynn_1966.pdf
5. M.D. Hill, N.P. Jouppi, G.S. Sohi, *"Multiprocessors and Multicomputers,"* in *pp. 551-560, Readings in Computer* Architecture https://safari.ethz.ch/architecture/fall2021/lib/exe/fetch.php?media=multiprocessors-multicomputers.pdf
6. M.D. Hill, N.P. Jouppi, G.S. Sohi, *"Dataflow and Multithreading,"* in *pp. 309-314, Readings in Computer* Architecture https://safari.ethz.ch/architecture/fall2021/lib/exe/fetch.php?media=hill_dataflow_multithreading.pdf
7. B. Smith, *"A pipelined, shared resource MIMD computer,"* in *ICPP* 1978 https://safari.ethz.ch/architecture/fall2021/lib/exe/fetch.php?media=pipelined1978smith.pdf
8. K. Gharachorloo, D. Lenoski, J. Laudon, P. Gibbons, A. Gupta, and J. Hennessy *"Memory Consistency and Event Ordering in Scalable Shared-Memory Multiprocessors"* in *ISCA* 1990 https://safari.ethz.ch/architecture/fall2021/lib/exe/fetch.php?media=memory_consistency_and_event_ordering_in_scalable_shared-memory_multiprocessors.pdf
9. M.M.K. Martin, M.D. Hill, and D.A. Wood. *"Token Coherence: Decoupling Performance and Correctness"* in *ISCA* 2003 https://safari.ethz.ch/architecture/fall2021/lib/exe/fetch.php?media=token_coherence_decoupling_performance_and_correctness.pdf
10. R. Das, O. Mutlu, T. Moscibroda, and C. R. Das. *"Application-Aware Prioritization Mechanisms for On-Chip Networks"* in *MICRO* 2009 https://people.inf.ethz.ch/omutlu/pub/app-aware-noc_micro09.pdf
11. J.H. Patel. *"Processor-Memory Interconnections for Multiprocessors"* in *ISCA* 1979 https://safari.ethz.ch/architecture/fall2021/lib/exe/fetch.php?media=p168-patel.pdf
12. C.L. Seitz. *"The Cosmic Cube"* in *CACM* 1985 https://safari.ethz.ch/architecture/fall2021/lib/exe/fetch.php?media=p22-seitz.pdf
13. C.J. Glass and L.M. Ni. *"The Turn Model for Adaptive Routing"* in *ISCA* 1992 https://safari.ethz.ch/architecture/fall2021/lib/exe/fetch.php?media=p278-glass.pdf
14. M. Fattah, A. Airola, R. Ausavarungnirun, N. Mirzaei, P. Liljeberg, J. Plosila, S. Mohammadi, T. Pahikkala, O. Mutlu, and H. Tenhunen. *"A Low-Overhead, Fully-Distributed, Guaranteed-Delivery Routing Algorithm for Faulty Network-on-Chips"* in *NOCS* 2015 https://people.inf.ethz.ch/omutlu/pub/maze-routing_nocs15.pdf
15. P. Baran. *"On Distributed Communications Networks"* in *IEEE Trans. Comm.,* 1964 https://safari.ethz.ch/architecture/fall2021/lib/exe/fetch.php?media=p2626.pdf
16. C. Fallin, C. Craik, and O. Mutlu. *"CHIPPER: A Low-Complexity Bufferless Deflection Router"* in *HPCA* 2011 https://people.inf.ethz.ch/omutlu/pub/chipper_hpca11.pdf
17. B. Grot, J. Hestness, S.W. Keckler, and O. Mutlu. *"Kilo-NOC: A Heterogeneous Network-on-Chip Architecture for Scalability and Service Guarantees"* in *ISCA* 2011 https://people.inf.ethz.ch/omutlu/pub/kilonoc_isca11.pdf
18. J. Laudon and D. Lenoski. *"The SGI Origin: A ccNUMA Highly Scalable Server"* in *ISCA* 1997 https://safari.ethz.ch/architecture/fall2021/lib/exe/fetch.php?media=the_sgi_origin_a_ccnuma_highly_scalable_server.pdf
19. W.J. Dally and B. Towles. *"Route Packets, Not Wires: On-Chip Interconnection Networks"* in *DAC* 2001 https://safari.ethz.ch/architecture/fall2021/lib/exe/fetch.php?media=onchip_dac01.pdf
20. R. Ausavarungnirun, C. Fallin, X. Yu, K. Chang, G. Nazario, R. Das, G. Loh, and O. Mutlu, *"A case for hierarchical rings with deflection routing: An energy-efficient on-chip communication substrate"* in *PARCO* 2016 https://safari.ethz.ch/architecture/fall2021/lib/exe/fetch.php?media=hierarchical-rings-ausavarungnirun.pdf

2. Interconnects: Warm-Up [150 points]

2.1. Topologies

Suppose you would like to connect 625 processors, and you are considering three different topologies: bus, point-to-point network, and mesh. Describe one disadvantage of each:

(i) A Single Bus.

Very high bus contention (with 625 processors)

(ii) A Point-to-Point Network.

Expensive, many individual wires (between every pair of nodes)

(iii) A 25x25 Mesh.

High complexity (e.g., routing logic)

Which one would you choose? Why?

A 25x25 Mesh: it is performance-scalable to 625 processors and not cost-prohibitive.

2.2. Trade-Offs

Company X is designing a multi-core processor with 100 cores. Their initial design connects the cores with a single shared bus. Give two reasons as to why this is not a good idea. ($<$ 20 words each.)

Limited bandwidth $\rightarrow$ high contention.

Cannot operate at high frequency due to electrical loading.

An engineer suggests using a crossbar to connect each core to every other, instead. Give two reasons as to why this is a good idea. ($<$ 20 words each.)

High bandwidth $\rightarrow$ low contention.

Can operate at high frequency.

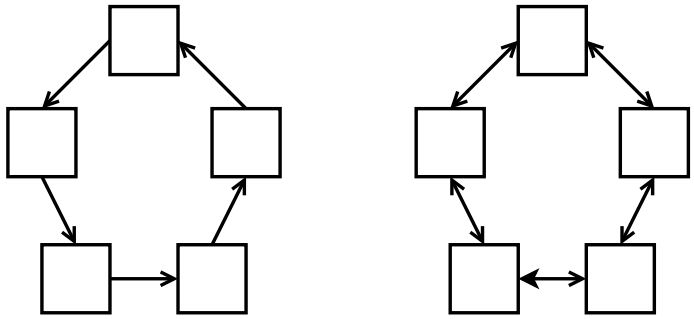
Give two reasons as to why this is not a good idea. ($<$ 20 words each.)

Much more hardware than a single bus: $O(n^2)$ vs. $O(1)$

Power; hard to scale to a larger number of cores.

2.3. Latencies

The following diagrams show two different ring topologies of size n.



a) Uni-directional Ring b) Bi-directional Ring

Throughout this question, assume that:

- n **is an odd number.**
- Packets can move from one node to an adjacent node in 1 cycle.
- The routing mechanism uses the shortest path from the source to the destination.
- The traffic pattern is uniform (i.e., every node has an equal probability of sending a packet to every other node).
- There is no contention (i.e., a packet can always move toward its destination through the shortest path at every cycle).

(a) What is the average latency of a uni-directional ring of size n? Show your work.

$\text{Avg}(1+2+3+\cdots+n-1) = \frac{(n)(n-1)}{2}\frac{1}{n-1} = \frac{n}{2}$

(b) What is the average latency of a bi-directional ring of size n? Show your work.

$2 \times \text{Avg}(1+2+3+\cdots+\frac{n-1}{2}) = 2 \times \frac{(1+\frac{n-1}{2})\frac{n-1}{2}}{2(n-1)} = \frac{n+1}{4}$

3. Interconnects (I) [200 points]

A computer architect would like to design an interconnect for a server processor with 64 cores. Each core has it's private cache and part of the shared physical memory of a shared memory multiprocessor.

The architect chooses to design the interconnect as a 2D mesh. The link width is $2B$ of payload each direction. The interconnect routers can work at frequencies of $100MHz$ (at 0.6V) or $200MHz$ (at 0.8V). The architect chooses to operate the interconnect at $100MHz$ to save energy.

Please answer the following questions. Show your work.

(a) The architect observe that multiple packets are sent from different cores in parallel which cause contention. In case that two packets are trying to use the same link at the same time, what three options does she have to handle packets contention?

Buffer one
Drop one
Misroute one (deflection)

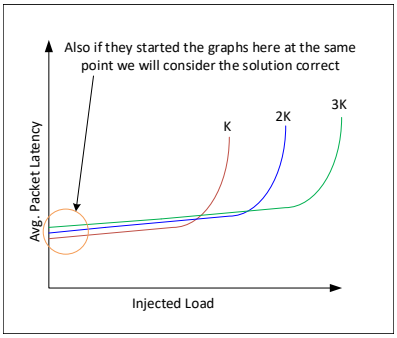
(b) Calculate
i) The diameter of the topology
ii) The bisection bandwidth (in number of links) of the topology
iii) The bisection bandwidth in MB/s (one direction) .

i) diameter $= 2(\sqrt{N} - 1) = 14$.
ii) bisection bandwidth (links) $= \sqrt{N} = 8$
iii) bisection bandwidth (MB/s) $= 2B \times 8 \times 100MHz = 1600MB/s$

(c) The architect observed that some cores have significantly high average packet latency that the others.
i) Explain why this happens and what topology change we can do to keep the latency uniform for all cores.
ii) What is the diameter and bisection bandwidth (in number of links) for the new proposed topology?

i) Nodes at the sides and corners have higher latency as the topology is not symmetric. We can change the 2D mesh to 2D torus.
ii) diameter $= \sqrt{N} = 8$. bisection bandwidth $= 2\sqrt{N} = 16$

(d) The architect examines three types of routers that can buffer K, $2K$ and $3K$ input packets.
i) Draw the curve of average packet latency as a function of the injected load.
ii) On the same drawing, add the curve when buffers are increased to $2K$ packets.
iii) On the same drawing, add the curve when buffers are increased to $3K$ packets.



(e) The architect observe that the router with the $2K$ buffers meets the average packet latency requirements of the quality of service (QoS) for 99.5% of the workloads. However, 0.5% of the workloads have high packet latency and require the routers with bigger buffers (i.e., $3K$) in order to keep the packaet latency within QoS requirement.
Assume that the 0.5% can be easily distinguished by a system global monitoring logic. Propose a power management technique to reduce the packet latency without changing the router design (e.g., routing algorithm, buffers, etc.) or topology?

We can apply dynamic voltage and frequency scaling (i.e., at 99.5% of the time work at $100MHz$ at at the 0.5% scale the frequency to $200MHz$. This cut the packet latency (in seconds) by half and doubles the bisection bandwidth (in MB/s).

4. Interconnects (II) [200 points]

Assume you need to design an interconnect with 256 nodes, where each node consists of a core, a cache, and part of the shared physical memory of a shared memory multiprocessor. The network will support all types of communication (for the purposes of this question, the type of communication does not matter). You are told that the "load" on this network varies over time from very low to very high. Besides that, communication happens in a uniform traffic pattern, where every node has an equal probability of sending a packet to every other node. The packet size ranges from 16 bytes to 256 bytes.

Given this information, pick the best design choices for this network. Your goal is to maximize performance and energy efficiency, and minimize area and design complexity, all at the same time, if possible (the Holy Grail, yes!).

Which of the following would you choose? Circle one in each sub-question and explain.

(a) Choose your network topology:

Crossbar Multistage Logarithmic **[2D Mesh]**

Pick one above, and explain

Area: A 256x256 Crossbar would be too costly $O(N^2) = O(256^2) = 65536$. On the other hand, even though both Multistage Logarithmic and 2D Mesh are blocking, the 2D mesh is 8x less costly ($O(N) = O(256)$) than the Multistage Logarithmic ($O(N \times \log_2(N)) = O(256 \times \log_2(256)) = 256 \times 8$).

Latency: The crossbar has the lowest average latency between the topologyes ($O(1)$). The average latency of for the 2D Mesh is twice as the Multistage Logarithmic. The average latency for the former one is $O(\sqrt{256}) = 16$, while for the later one it is $O(\log_2(256)) = 8$.

Therefore, considering both area and latency, the 2D Mesh has the best trade-off.

(b) Choose your switching strategy in each router:

Circuit switching **[Packet switching]** Does not matter

Pick one above, and explain

Since the communication traffic is uniform, and the network topology is blocking, packet switching will make better use of the switches by dynamically sharing the links.

(c) Choose the type of your routing algorithm:

Deterministic routing Oblivious routing [Adaptive routing] Does not matter

Pick one above, and explain

Once the load changes over time from low to high, an adaptive routing can take decisions based on the number of congested or faulty channel.

(d) Choose your flow control method:

Bufferless Store and Forward Cut Through [Wormhole] Virtual Cut Through

Pick one above, and explain

Since the packet size varies, all of the packet-based flow control methods (Store and Forward, Cut Through, and Virtual Cut Through) would require more buffer space than the router might require (at least the size of the biggest packet). Meanwhile, a Bufferless flow control method would increase router complexity, and produce congestions under high loads.

5. Interconnects (III) [200 points]

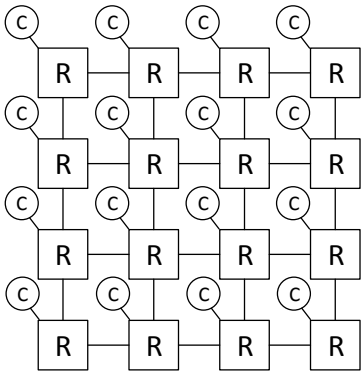
Suppose you would like to connect 2^N processors, and you are considering four different topologies:

- $\sqrt{2^N} \times \sqrt{2^N}$ 2D mesh
- $\sqrt{2^{N-2}} \times \sqrt{2^{N-2}}$ 2D concentrated mesh (Cmesh), where each router serves four processors
- $\sqrt{2^N} \times \sqrt{2^N}$ 2D torus
- Hypercube

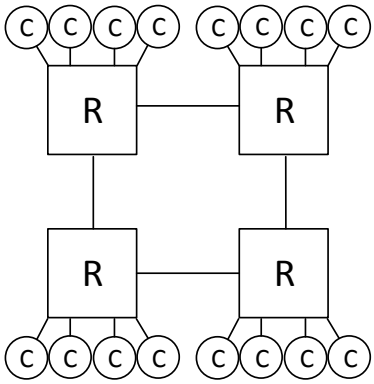
Please answer the following questions. Show your work.

(a) For N = 4, please draw how each network looks like. You can use ... (three dots) to avoid repeated patterns. *Please clearly label each of the network you draw.*

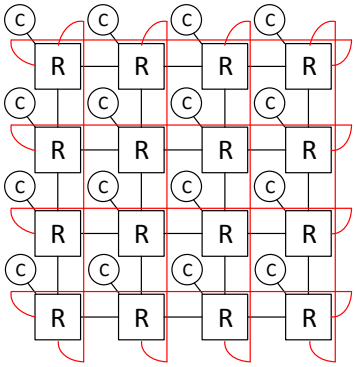
2D Mesh:



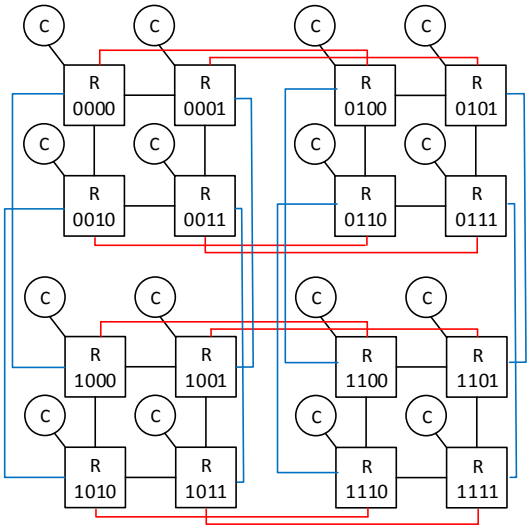
CMesh:



2D Torus:



Hypercube:



For the remaining questions, *assume N = 8.*

(b) For N = 8, calculate the number of network links for each network. (Hint: a single network link is bi-directional)

2D mesh: $2 \times (\sqrt{2^N} - 1)(\sqrt{2^N})$ links $\rightarrow 2 \times 15 \times 16 = 480$ links
Cmesh: $2 \times (\sqrt{2^{N-2}} - 1)(\sqrt{2^{N-2}})$ links $\rightarrow 2 \times 7 \times 8 = 112$ links
2D torus: $2 \times (\sqrt{2^N})(\sqrt{2^N})$ links $\rightarrow 2 \times 16 \times 16 = 512$ links
Hypercube: $2^N \times N/2$ links $\rightarrow 256 \times 8/2 = 1024$ links

(c) For N = 8, calculate the number of input/output ports including the injection/ejection ports for *each router* in these topologies (Hint: give answer to all types of routers that exist in an irregular network).

2D mesh: (4+1) inputs/outputs, (3+1) inputs/outputs, and (2+1) inputs/outputs
Cmesh: (4+4) inputs/outputs, (3+4) inputs/outputs, and (2+4) inputs/outputs
2D torus: (4+1) inputs/outputs
Hypercube: N+1 inputs/outputs $\rightarrow$ 9 inputs/outputs

(d) Assume a network link can be faulty. For each topology, what is the minimum possible number of faulty links that are needed to make at least one processor unreachable from any other processor?

2D mesh: 2 links
Cmesh: 2 links
2D torus: 4 links
Hypercube: $Nlinks \rightarrow 8$ links

6. Building Multicore Processors [150 points]

You are hired by Amdahl's Nano Devices (AND) to design their newest multicore processor. Ggl, one of AND's largest customers, has found that the following program can predict people's happiness.

```
for (i = 12; i < 2985984; i++) {
  past = A[i-12];
  current = A[i];
  past *= 0.37;
  current *= 0.63;
  A[i] = past + current;
}
```

`A` is a large array of 4-byte floating point numbers, gathered by Ggl over the years by harvesting people's private messages. Your job is to create a processor that runs this program as fast as possible.

Assume the following:

- You have magically fast DRAM that allows infinitely many cores to access data in parallel. We will relax this strong assumption in parts (d), (e), (f).
- Each floating point instruction (addition and multiplication) takes 10 cycles.
- Each memory read and write takes 10 cycles.
- No caches are used.
- Integer operations and branches are fast enough that they can be ignored.

(a) Assuming infinitely many cores, what is the maximum steady state speedup you can achieve for this program? Please show all your computations.

The cycle counts are extra information that are not necessary.
Instead, the loop body is sequential, while 12 consecutive iterations are parallel. Notice that the 13th iteration is dependent on the 1st iteration.
Therefore $\frac{1}{12}$ of the program is serial. Using Amdahl's law, this solves to a maximum speedup of 12.

(b) What is the minimum number of cores you need to achieve this speedup?

12 cores are needed.

(c) Briefly describe how you would assign work to each core to achieve this speedup.

Assign iteration `i` to processor `i%12`.

It turns out magic DRAM does not exist except in Macondo[1]. As a result, you have to use cheap, slow, low-bandwidth DRAM. To compensate for this, you decide to use a private L1 cache for each processor. The new specifications for the DRAM and the L1 cache are:

- DRAM is shared by all processors. DRAM may only process one request (read or write) at a time.
- DRAM takes 100 cycles to process any request.
- DRAM prioritizes accesses of smaller addresses and write requests. (Assume no virtual memory)
- The cache is direct-mapped. Each cache block is 16 bytes.
- It takes 10 cycles to access the cache. Therefore, a cache hit is processed in 10 cycles and a cache miss is processed in 110 cycles.

All other latencies remain the same as specified earlier. Answer parts (d), (e), (f) assuming this new system.

(d) Can you still achieve the same steady state speedup as before? Please explain.

No. Notice that the serial bottleneck is now caused by the DRAM. In steady state, only one memory access is performed for every four iterations. So four iterations take 100 + 200 = 300 cycles. 12 iterations requires 900 cycles.
In order to achieve 12x speedup, each core needs to complete each iteration in 75 cycles. However, fetching from memory alone requires 100 cycles. Therefore, it is not possible.

By just saying that it is now slower than before is not enough, since the single core baseline is also slower.

(e) What is the minimum number of cores your processor needs to provide the maximum speedup?

3 cores are needed to take the maximum advantage of each cache hit, and a maximum speed up of almost 3 can be achieved.
Core 1 calculates iterations i+0, i+1, i+2, i+3, core 2 calculates iteration i+4, i+5, i+6, i+7, and core 3 calculates iteration i+8, i+9, i+10, i+11.

(f) Briefly describe how you would assign work to each core to achieve this speedup.

3 cores are needed to take the maximum advantage of each cache hit, and a maximum speed up of almost 3 can be achieved.
Core 1 calculates iterations i+0, i+1, i+2, i+3, core 2 calculates iteration i+4, i+5, i+6, i+7, and core 3 calculates iteration i+8, i+9, i+10, i+11.

[1] An imaginary town featured in *One Hundred Years of Solitude* by the late Colombian author Gabriel García Márquez (1927-2014).

7. Asymmetric Multicore (I) [150 points]

A microprocessor manufacturer asks you to design a multicore processor for modern workloads. You should optimize it assuming a workload with 80% of its work in the parallel portion while the reset is serial. You have two design configurations: 1) Small cores (SC) and 2) Large + Small cores (LC) that can fit into the processor's die area:

- *SC*: A design that contains 8 small cores, which share the same die. You have the choice to build a multicore processor that has 8 small cores that share the same die. Seven of these small cores operate at a *baseline* fixed throughput. The other eighth small core is an overclockable core, that is it can operate at either: 1) *baseline* throughput, or 2) *overclocked* with 2× the baseline throughput.
- *LC*: You also have another design choice where you can build a multicore processor that has 1 large core and 4 small cores that all share the same die. The four small cores operate at the baseline throughput. The large core is 4× faster than a small core.

You are given that the dynamic power (i.e., when the core is active) and the static power (i.e., when the core is idle) of baseline/overclocked small core are 1/8 Watts and 0.5/2 Watts, respectively. The dynamic power and the static power of a large core are 2 Watts and 1 Watt, respectively.

	# Cores	Throughput	Active Power (W)	Idle Power (W)
SC	7 small cores	Baseline	1	0.5
	1 overclockable small core	2 x Baseline	8	2
LC	4 small cores	Baseline	1	0.5
	1 Large core	4 x Baseline	2	1

The SC processor executes the parallel portion on the small cores including the overclockable core using the baseline throughput, and the serial portion *only* on the overclockable core (using baseline or overclocked options). The LC processor executes the parallel portion on the small cores, and the serial portion *only* on the large core.

Please answer the following questions.

(a) Which of the two design configurations results in the highest performance when considering that the overclockable small core runs at (i) the baseline throughput, and (ii) the overclocked throughput? Show your work.

The overclockable small core configuration when running the overclockable core at the higher throughput is the best options.

Explanation:
The best case speedup for the first configuration when running using the baseline throughput can be calculated as:
$Speedup_1 = \frac{1}{\frac{0.2}{1}+\frac{0.8}{8}} = \frac{10}{3} = 3.33.$

The best case speedup for the first configuration when running using the overclocked throughput can be calculated as:
$Speedup_2 = \frac{1}{\frac{0.2}{2}+\frac{0.8}{8}} = \frac{10}{2} = 5.$

The best case speedup for the second configuration can be calculated as:
$Speedup_3 = \frac{1}{\frac{0.2}{4}+\frac{0.8}{4}} = \frac{1}{0.25} = 4.$

Without loss of generality, we assume that the baseline execution time is 1, hence total execution time of each option can be calculated using:
$t_{total} = t_{serial} + t_{parallel} = \frac{1}{Speedup}$ seconds.

(b) The energy consumption should also be a metric of reference in your design. Which of the two design configurations results in the lowest energy consumption. Show your work.

The Large Core configuration has the lowest energy consumption.

Explanation:
We can calculate the energy consumption as:
$E_{total} = E_{serial} + E_{parallel} = P_{serial} \times t_{serial} + P_{parallel} \times t_{parallel}$

The energy consumption for the first configuration when running using the baseline throughput can be calculated as:

$E_{total1} = (1 + 7 \times 0.5) \times 0.2 + (1 + 1 \times 7) \times 0.1 = 1.7$
The energy consumption for the first configuration when running using the overclocked throughput can be calculated as:
$E_{total2} = (8 + 7 \times 0.5) \times 0.1 + (1 + 1 \times 7) \times 0.1 = 1.95$
The energy consumption for the second configuration can be calculated as:
$E_{total3} = (2 + 4 \times 0.5) \times 0.05 + (1 + 1 \times 4) \times 0.2 = 1.2$

(c) You are asked to use another key metric of reference that is called *Energy-Delay Product (EDP)*. We define EDP as the product of the energy consumption times the execution time. Which of the two design configurations has the least EDP? Show your work.

The Large Core configuration has the lowest EDP.

Explanation:
We can calculate the Energy-Delay Product (EDP) as:
$EDP = Energy \times Execution\_Time$

The EDP for the first configuration when running using the baseline throughput can be calculated as:
$EDP_1 = 1.7 \times 0.3 = 0.51$
The EDP for the first configuration when running using the overclocked throughput can be calculated as:
$EDP_2 = 1.95 \times 0.2 = 0.39$
The EDP for the second configuration can be calculated as:
$EDP_3 = 1.2 \times 0.25 = 0.3$

8. Asymmetric Multicore (II) [200 points]

A microprocessor manufacturer asks you to design an asymmetric multicore processor for modern workloads. Your design contains one large core and several small cores, which share the same die. Assume the total die area is A units. The table below describes the area, performance, and power specifications for each core type.

Type of Core	Area (mm^2)	Performance	Dynamic Power (W)	Static Power (W)
Large	S	$\sqrt{S}$	S	$\frac{1}{4} \times S$
Small	1	1	1	0.5

The serial portion of a workload executes only on the large core, while the parallel portion executes on both large and small cores. On this multiprocessor, we will execute a workload where a fraction P of its work, and $1 - P$ of its work is serial. You will fit as many small cores as possible, after placing the large core. Consider the following two configurations:

- Configuration X: A = 32, S = 4.
- Configuration Y: A = 32, S = 16.

Please answer the following questions. Show your work. Express your equations and solve them.

(a) What is the speedup of the workload on configuration X, compared to the execution time on a single-core processor of area 1 (i.e., one of the small cores)?

$Speedup_x = \frac{1}{\frac{1-P}{2} + \frac{P}{30}}$

Explanation:

$Speedup_x = \frac{1}{\frac{1-P}{\sqrt{4}} + \frac{P}{\sqrt{4}+(32-4)\times 1}}$

$Speedup_x = \frac{1}{\frac{1-P}{2} + \frac{P}{2+28}}$

$Speedup_x = \frac{1}{\frac{1-P}{2} + \frac{P}{30}}$

(b) What is the speedup of the workload on configuration Y, compared to the execution time on a single-core processor of area 1 (i.e., one of the small cores)?

$Speedup_y = \frac{1}{\frac{1-P}{4} + \frac{P}{20}}$

Explanation:

$Speedup_x = \frac{1}{\frac{1-P}{\sqrt{16}} + \frac{P}{\sqrt{16}+(32-16)\times 1}}$

$Speedup_x = \frac{1}{\frac{1-P}{4} + \frac{P}{4+16}}$

$Speedup_x = \frac{1}{\frac{1-P}{4} + \frac{P}{20}}$

(c) For what range of P does the workload run faster on Y than on X? Show your work.

$P < \frac{60}{64} \Rightarrow P < 0.94$

Explanation:
$Speedup_y > Speedup_x$
$\Rightarrow {Speedup_y}^{-1} < {Speedup_x}^{-1}$
$\Rightarrow \frac{1}{\frac{1-P}{4}+\frac{P}{20}} < \frac{1}{\frac{1-P}{2}+\frac{P}{30}}$
$\Rightarrow 24 \times P - \frac{8 \times P}{3} < 20$
$\Rightarrow \frac{64 \times P}{3} < 20$
$\Rightarrow P < \frac{60}{64}$

(d) For what range of P does the workload consumes less energy when running on Y than on X? Show your work.

$P < \frac{75}{94} \Rightarrow P < 0.80$

Explanation:
We can calculate the energy consumption as:
$E_{total} = E_{large} + E_{small}$
$E_{large} = (P_{large\_dynamic} + P_{large\_static}) \times t_{serial} + (P_{large\_dynamic} + P_{large\_static}) \times t_{parallel}$
$E_{small} = (P_{small\_dynamic} + P_{small\_static}) \times t_{parallel} + (P_{small\_static}) \times t_{serial}$

Thus:
$Energy_y = 20 \times \frac{1-P}{4} + 20 \times \frac{P}{20} + 16 \times [1.5 \times \frac{P}{20} + 0.5 \times \frac{1-P}{4}]$
$Energy_x = 5 \times \frac{1-P}{2} + 5 \times \frac{P}{30} + 28 \times [1.5 \times \frac{P}{30} + 0.5 \times \frac{1-P}{2}]$
$Energy_y < Energy_x$
$P < \frac{75}{94}$

8.1. Dual-Core Execution

Assume that two large cores can operate in a "dual-core execution" (DCE) manner to achieve the single-thread performance of an even "larger" core that is **N**× faster than the largest core on the chip. The "dual-core execution" occurs only during the serial portion of the workload. During the parallel portion, the two large cores separate from each other (on-the-fly) and operate as two independent cores. The serial portion executes only on the "dual-core", and the parallel portion executes on all the cores. The table below describes the area and performance specifications for each core for the DCE design.

Type of Core	**Area** (mm^2)	**Performance**
$Large_1$	S_1	$\sqrt{S_1}$
$Large_2$	S_2	$\sqrt{S_2}$
DCE	$S_1 + S_2$	$\mathbf{N} \times \sqrt{Max(S_1, S_2)}$
Small	1	1

Consider the following two configurations:

- Configuration DCE: A = 32, $S_1 = 9$, $S_2 = 4$.

(e) For what range of N does the workload run faster on DCE than on X? Assume $P = 0.8$. Show your work.

$N > 0.71$

Explanation:

$Speedup_{DCE} = \frac{1}{\frac{1-P}{N\times(\sqrt{9})} + \frac{P}{3+2+19}}$

$Speedup_x = \frac{1}{\frac{1-P}{2} + \frac{P}{2+28}}$

$\Rightarrow \frac{1-P}{3\times N} + \frac{P}{24} > \frac{1-P}{2} + \frac{P}{30}$

$\Rightarrow \frac{3\times N}{1-P} > \frac{2\times 30\times 24}{30\times 24\times(1-P)+2\times 24\times P-2\times 30\times P}$

$\Rightarrow N > \frac{480-480\times P}{720-732\times P}$

$\Rightarrow N > \frac{480-480\times 0.8}{720-732\times 0.8}$

$\Rightarrow N > 0.71$

(f) For what range of P does the workload run faster on DCE than on X? Assume $N = 1.5$. Show your work.

$P < \frac{100}{103} \Rightarrow P < 0.97$

Explanation:

$\frac{1-P}{0.5\times(3+2)} + \frac{P}{3+2+19} < \frac{1-P}{2} + \frac{P}{2+28}$

$\Rightarrow P < \frac{12}{13}$

(g) Suppose you are executing workloads where a fraction P of its work is *infinitely* parallelizable. Which configuration would you choose? DCE or Y? Why?

Y, since it would provide better speedup for larger values of P.

9. Cache Coherence [200 points]

We have a system with 4 processors {P0, P1, P2, P3} that can access memory at byte granularity. Each processor has a private data cache with the following characteristics:

- Capacity of 256 bytes.
- Direct-mapped.
- Write-back.
- Block size of 64 bytes.

Each processor has also a dedicated private cache for instructions. The characteristics of the instruction caches are not necessary to solve this question. All data caches are connected to and actively snoop a global bus, and cache coherence is maintained using the MESI protocol, as we discussed in class. Note that on a write to a cache block in the S state, the block transitions directly to the M state. The range of accessible memory addresses is from `0x00000` *to* `0xffffff`.

The semantics of the instructions used in this question are as follows:

Opcode	Operands	Description
ld	rx,[ry]	rx ← Mem[ry]
st	rx,[ry]	rx → Mem[ry]
addi	rx,#VAL	rx ← rx + VAL
subi	rx,#VAL	rx ← rx - VAL
j	TARGET	jump to TARGET
bneq	rx,ry,TARGET	if([rx]!=[ry]) jump to TARGET

This is the initial state of the data caches in all processors:

Initial Tag Store States

Cache for P0		
Set	*Tag*	*MESI state*
0	`0x100`	`M`
1	`0x010`	`S`
2	`0x100`	`I`
3	`0x222`	`E`

Cache for P1		
Set	*Tag*	*MESI state*
0	`0x100`	`I`
1	`0x333`	`E`
2	`0x100`	`E`
3	`0x333`	`S`

Cache for P2		
Set	*Tag*	*MESI state*
0	`0x101`	`E`
1	`0x010`	`S`
2	`0x010`	`S`
3	`0x222`	`I`

Cache for P3		
Set	*Tag*	*MESI state*
0	`0x102`	`M`
1	`0x010`	`S`
2	`0x010`	`S`
3	`0x333`	`S`

This is the final state of the data caches in all processors (the shadowed sets in the figure represent the sets that change compared to the initial state):

Final Tag Store States

Cache for P0		
Set	*Tag*	*MESI state*
0	**0x102**	**S**
1	**0x102**	**E**
2	**0x102**	**E**
3	**0x333**	**I**

Cache for P1		
Set	*Tag*	*MESI state*
0	**0x102**	**S**
1	**0x333**	**I**
2	0x100	E
3	**0x333**	**M**

Cache for P2		
Set	*Tag*	*MESI state*
0	0x101	E
1	**0x333**	**M**
2	0x010	S
3	**0x222**	**E**

Cache for P3		
Set	*Tag*	*MESI state*
0	**0x102**	**S**
1	**0x333**	**I**
2	0x010	S
3	**0x333**	**I**

(a) Make the following assumptions:

- Each processor executes the instructions in a *sequentially consistent* manner.
- You can make use of five registers: r0, r1, r2, r3, and r4.
- The ordering between two instructions from different processors might be ambiguous when there is no synchronization. If the order between two memory requests from different processors is ambiguous, and if the ordering is important for the final result, indicate the ordering between the two instructions in your solution.
- The initial values of all registers are the same in all processors, and they contain the following values:

r0=0x22200 r1=0x10200 r2=0x3338f r3=0x00000 r4=0x102C0

What are the minimum sequences of instructions in each processor that lead to the caches final state? Fill in the blanks. Write one instruction per line. Show your work as needed.

	P0
0	`addi r2,#0x40`
1	`ld r3,[r2]`
2	`LP:ld r3,[r1]`
3	`addi r1,#0x40`
4	`bneq r1,r4,LP`
5	
6	

	P1
0	`ld r3,[r1]`
1	`addi r2,#0x40`
2	`st r3,[r2]%(after P0: line 1)`
3	
4	
5	
6	

	P2
0	`addi r0,#0xC0`
1	`subi r2,#0x40`
2	`ld r3,[r0]`
3	`st r3,[r2]`%(after P3: line 1)
4	
5	
6	

	P3
0	`subi r2,#0x40`
1	`ld r3,[r2]`
2	
3	
4	
5	
6	

NOTE: This may not be the only solution.

(b) Now assume all four processors execute the following code (This part is independent of part (a)):

P0, P1, P2, and P3	
1	`LOOP: st r0, [r1]`
2	`addi r1, #0x40`
3	`bneq r1, r4, LOOP`

The final state of the tag store is the following:

Final Tag Store States

Cache for P0		
Set	*Tag*	*MESI state*
0	0x102	M
1	0x102	M
2	0x102	M
3	0x100	M

Cache for P1		
Set	*Tag*	*MESI state*
0	0x102	I
1	0x102	I
2	0x102	I
3	0x102	M

Cache for P2		
Set	*Tag*	*MESI state*
0	0x106	M
1	0x106	M
2	0x104	M
3	0x104	M

Cache for P3		
Set	*Tag*	*MESI state*
0	0x106	I
1	0x106	I
2	0x106	M
3	0x106	M

Make the following assumptions about the initial state of the caches:

- The tag is the same in all sets of a processor (but the tags might be different among processors).
- The MESI state of all cache lines in all processors is the same.

What are the initial state values of the registers r0, r1, and r4 in all four processors? Show your work.

Solution:
P0: r0 = X, r1 = 0x10200, r4 = 0x102C0
P1: r0 = X, r1 = 0x102C0, r4 = 0x10300
P2: r0 = X, r1 = 0x10600, r4 = 0x10680
P3: r0 = X, r1 = 0x10680, r4 = 0x106C0 (or 0x10700)

NOTE: This may not be the only solution.

Explanation:

- The code executes store instructions in a loop.
- The address of the cache line is contained in r1.
- In each iteration we load the next cache line, i.e., we 1) increase the set while maintaining the same tag, or 2) increase the tag if the current address maps to set 3.
- We know from P0 (we can make a similar observation from P2) that we can use the code to modify sets 0 and 1, or sets 2 and 3, but not all at the same time.
- Because all sets are in Modified state, we can conclude that the initial state was Modified, and therefore, all the MESI states were initially in Modified state.
- P1 has some cache lines in Invalid state. For these lines to be Invalid, P0 has to invalidate them. Therefore, we know that the code for the first loop has to store cache lines with the tag 0x102 in sets 0, 1, and 2. Taking this into consideration, the only code option for P1 requires to store a cache line with tag 0x102 in set 3. A similar reasoning can be applied to processor 2 and 3.

What is the initial state of all caches? Fill in the blanks below.

Initial Tag Store States

Cache for P0		
Set	*Tag*	*MESI state*
0	0x100	M
1	0x100	M
2	0x100	M
3	0x100	M

Cache for P1		
Set	*Tag*	*MESI state*
0	0x102	M
1	0x102	M
2	0x102	M
3	0x102	M

Cache for P2		
Set	*Tag*	*MESI state*
0	0x104	M
1	0x104	M
2	0x104	M
3	0x104	M

Cache for P3		
Set	*Tag*	*MESI state*
0	0x106	M
1	0x106	M
2	0x106	M
3	0x106	M

10. Memory Consistency [150 points]

A programmer writes the following two C code segments. She wants to run them concurrently on a multicore processor, called SC, using two different threads, each of which will run on a different core. The processor implements *sequential consistency*, as we discussed in the lecture.

	Thread T0		**Thread T1**
Instr. T0.0	`X[0] = 2;`	Instr. T1.0	`X[0] = 1;`
Instr. T0.1	`flag[0] = 1;`	Instr. T1.1	`X[0] += 2;`
Instr. T0.2	`a = X[0]*2;`	Instr. T1.2	`while(flag[0] == 1);`
Instr. T0.3	`b = Y[0]-1;`	Instr. T1.3	`a = flag[0];`
Instr. T0.4	`c = X[0];`	Instr. T1.4	`X[0] = 2;`
		Instr. T1.5	`Y[0] = 10;`

`X` and `flag` have been allocated in main memory. Thread 0 and Thread 1 have their private processor registers to store the values of `a` , `b`, and `c`. A read or write to any of these variables generates a single memory request. The initial values of all memory locations and variables are 1. Assume each line of the C code segment of a thread is a *single* instruction.

(a) Do you find something that could be wrong in the C code segments? Explain your answer.

Thread 1 will never finish.

Explanation:
The while loop in instruction T1.2 is an infinite loop, because the value of `flag[0]` is 1 since the beginning of the program.

(b) What could be possible final values of `X[0]` in the SC processor, after executing both C code segments? Explain your answer. Provide all possible values.

2, 3, or 4.

Explanation:
The sequential consistency model ensures that the operations of each individual thread are executed in the order specified by its program. Across threads, the ordering is enforced by the use of `flag[0]`. Thread 1 will remain in instruction T1.2 until `flag[0]` has a value that is not 1. However, thread 1 will never finish execution. There are *at least* three possible sequentially-consistent orderings that lead to *at most* three different values of `X` at the end:
Ordering 1: T1.0 $\rightarrow$ T1.1 $\rightarrow$ T0.0, Final value: `X[0]` = 2.
Ordering 2: T0.0 $\rightarrow$ T1.0 $\rightarrow$ T1.1, Final value: `X[0]` = 3.
Ordering 3: T1.0 $\rightarrow$ T0.0 $\rightarrow$ T1.1, Final value: `X[0]` = 4.

(c) What could be possible final values of `a` in the SC processor, after executing both C code segments? Explain your answer. Provide all possible values.

2, 4, 6, or 8.

Explanation:
The value of `a` is twice the value of `X[0]`:
Ordering 1: T1.0 → T1.1 → T0.0 → T0.2, Final value: `X[0]` = 4.
Ordering 2: T0.0 → T1.0 → T1.1 → T0.2, Final value: `X[0]` = 6.
Ordering 3: T1.0 → T0.0 → T1.1 → T0.2, Final value: `X[0]` = 8.

Ordering 4: T0.0 → T0.1 → T1.0 → T0.2, Final value: `X[0]` = 2.

(d) What could be possible final values of `b` in the SC processor, after both threads finish execution? Explain your answer. Provide all possible values.

0.

Explanation:
Because the value of `b` depends only on the value of `Y[0]` (instruction T0.3). The initial value of `Y[0]` is 1. Instruction T1.4 will not be executed as T1 enters an infinite loop after executing instruction T1.2.

(e) With the aim of achieving higher performance, the programmer tests her code on a new multicore processor, called NC, that does not implement memory consistency. Thus, there is *no* guarantee on the ordering of instructions as seen by different cores.
What is the final value of `X[0]` in the NC processor, after executing both threads? Explain your answer.

1, 2, 3, or 4.

Explanation:
Since there is no guarantee of a strict order of memory operations, as seen by different cores, instruction T1.1 could complete before or after instruction T1.0, from the perspective of the core that executes thread 0. If instruction T1.1 completes before instruction T1.0, from the perspective of the core that executes T0, instruction T0.0 could complete before or after instruction T1.0. Thus, there are at least five possible weakly-consistent orderings that lead to different values of `X[0]` at the end:
Ordering 1: T0.0 $\rightarrow$ T1.1 $\rightarrow$ T1.0, Final value: `X[0]` = 1.
Ordering 2: T1.4 $\rightarrow$ T0.0 $\rightarrow$ T1.1 $\rightarrow$ T1.0, Final value: `X[0]` = 1.
Ordering 3: T1.0 $\rightarrow$ T1.1 $\rightarrow$ T0.1, Final value: `X[0]` = 2.
Ordering 4: T0.0 $\rightarrow$ T1.0 $\rightarrow$ T1.1, Final value: `X[0]` = 3.
Ordering 5: T1.0 $\rightarrow$ T0.0 $\rightarrow$ T1.1, Final value: `X[0]` = 4.

11. RowHammer [200 points]

11.1. RowHammer Properties

Determine whether each of following statements is true or false.

(a) Cells in a DRAM with a smaller technology node are more vulnerable to RowHammer.

1. True 2. False

(b) Cells which have shorter retention times are especially vulnerable to RowHammer.

1. True 2. False

(c) The vulnerability of cells in a victim row to RowHammer depends on the data stored in the victim row.

1. True 2. False

(d) The vulnerability of cells in a victim row to RowHammer depends on the data stored in the aggressor row.

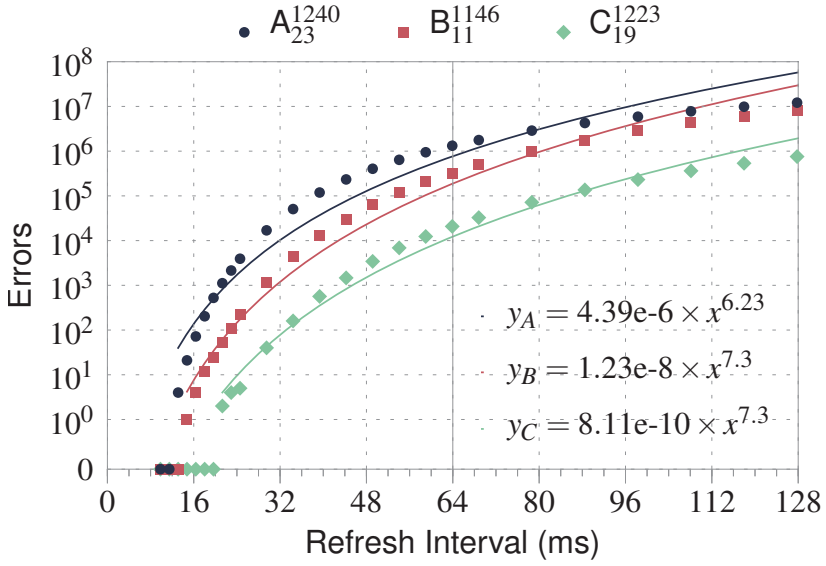
1. True 2. False

(e) RowHammer-induced errors are mostly repeatable.

1. True 2. False

11.2. RowHammer Mitigations

One research group rigorously investigated the relationship between the DRAM refresh rate and RowHammer-induced errors using real DRAM modules from three major memory manufacturers: A, B, and C. The following figure shows the characterization results of the *most RowHammer-vulnerable* modules of each DRAM manufacturer A, B, and C (A_{23}, B_{11}, and C_{19}, respectively). As shown in the figure below, we can achieve an *effectively-zero* probability of RowHammer-induced errors when we reduce the refresh interval to 8 ms. This is because the probability of RowHammer-induced errors becomes less than that of a random DRAM circuit fault.



As a leading computer architect of a company, you are asked to design an efficient RowHammer mitigation technique based on the characterization results.

(a) A junior engineer suggests to simply reduce the refresh interval from the default 64 ms to 8 ms. How will the bank utilization U and the DRAM energy consumption E of all refresh operations be changed over the current system?

$E' = 8 \times E$
$U' = 8 \times U$

(b) A DRAM manufacturer releases a higher capacity version of the same DRAM module (4x the total capacity). After rigorously reverse-engineering, you determine that the manufacturer doubles both the (1) number of rows in a bank, and (2) number of banks in a module without changing any other aspects (e.g., row size, bus rate, and timing parameters). Your company considers upgrading the current system with the higher-capacity DRAM module and asks your opinion. In the current system, the bank utilization of refresh operations is 0.05 when the refresh interval is 64 ms. Would reducing the refresh interval to 8 ms still be viable (in terms of RowHammer protection capability and performance overheads) in a module with 2x the number of rows per bank and 2x the number of banks per module? How about in a module with 4x the number of rows per bank and 4x the number of banks per module?

With a 2x increase, it is still viable, but very impractical. Most throughput is consumed by refresh operations ($U = 0.05 \times 8 \times 2 = 0.8$)

With 4x increase, it is impossible to perfectly defend RowHammer attacks by reducing the refresh interval, because we cannot refresh every row within 8 ms ($U = 0.05 \times 8 \times 4 = 1.6 > 1.$) Furthermore, throughput loss would reaches very close to 1.

(c) Due to significant overheads of reducing the refresh interval, your team decides to find other RowHammer mitigation techniques with lower overhead. The junior engineer proposes a counter-based approach as follows:

In this approach, the memory controller maintains a counter for each row R, which increments when an adjacent row is activated, and resets when Row R is activated or refreshed. If the value of a row R's counter exceeds a threshold value, T, the memory controller activates row R and resets its respective counter.

Here are some specifications on the current memory system:

- Interface: DDR3-1333
- $CL = 7$
- $tRCD = 13.5$ ns
- $tRAS = 35$ ns
- $tWR = 15$ ns
- $tRP = 13.5$ ns
- $tRFC = 120$ ns
- Configuration: Per-channel memory controller deals with 2 ranks, each of which has 8 banks. The number of rows per bank is 2^{15}. Each row in one bank is 8 KiB.

Are the given specifications enough to implement the proposed approach? If no, list the specifications additionally required.

No. We should know the exact mapping between row addresses and physical rows.

(d) Suppose that all the necessary specifications for implementing the proposed approach are known. Calculate the maximum value of T that can guarantee the same level of security against RowHammer attacks over when adopting 8-ms refresh interval?

T should be less than or equal to the maximum number of activations within 8 ms.

$$\begin{aligned} T \leq 8 \times 10^{-3}/t_{RC} &= 8 \times 10^{-3}/(t_{RAS} + t_{RP}) \\ &= 8 \times 10^{-3}/48.5 \times 10^{-9} \\ &= 164948.45... \\ \therefore T &= 164948 \end{aligned}$$

(e) Calculate the number of bits required for counters in each memory controller. How does it change when the number of rows per bank and the number of banks per chip are doubled?

The number of bits B of a single counter should meet $2^B > T = 0.164 \times 10^5$.
$2^{17} < T < 2^{18}, \therefore B = 18$
$18\ [\frac{bit}{row}] \times (2^{15})\ [\frac{row}{bank}] \times 16\ [bank] = 9 \times 2^{20}$ bits $= 9$ Mib

It will increase to 36 Mib with 2x rows and 2x banks.

(f) Obviously, we can reduce the size of counters in each memory controller, if we use a smaller value for T. What are its drawbacks?

Performance and energy overheads from unnecessary activations.

(g) You recall *PARA (Probabilistic Adjacent Row Activation)* which is proposed in the first RowHammer paper. Here is how a PARA-enabled memory controller works (for more details, see Section 8.2 in the paper[2]):

Whenever a row is closed, the controller flips a biased coin with a probability p of turning up heads, where $p << 1$. If the coin turns up heads, the controller opens one of its adjacent rows where either of the two adjacent rows are chosen with equal probability ($p/2$). Due to its probabilistic nature, PARA does not guarantee that the adjacent will always be refreshed in time. Hence, PARA cannot prevent disturbance errors with absolute certainty. However, its parameter p can be set so that disturbance errors occur at an extremely low probability — many orders of magnitude lower than the failure rates of other system components (e.g., more than 1% of hard-disk drives fail every year.)

Suppose that the probability of experiencing an error for PARA in 64 ms is 1.9×10^{-22} when $p = 0.001$ at the *worst operating conditions*. How can we estimate the probability of experiencing an error in one year based on that value? Show your work. (If you just put an answer, you will get no points for this problem.)

Let P_I be the probability of experiencing an error for PARA in a time interval I (i.e., $P_{64ms} = 1.9 \times 10^{-22}$). Since we can consider a year as a sequence of independent 64-ms time windows, the number of errors in a year can be modeled as a random variable X that is binomially-distributed with parameters $B(N,\ P_{64ms})$ where N is the number of 64-ms time windows in a year (i.e., $N = (365 \times 24 \times 60 \times 60)/(64 \times 10^{-3}) = 492,750,000$). Therefore, the probability of that at least one error occurs in a year can be calculated as follows:

$$
\begin{aligned}
\therefore P_{1year} &= P(X \geq 1) \\
&= 1 - P(X = 0) \\
&= 1 - (1 - P_{64ms})^N \\
&= 1 - (1 - 1.9 \times 10^{-22})^{492750000} \\
&= 1 - 0.99999999999990637750000000438259 \\
&= 9.362249999999561741... \times 10^{-14}
\end{aligned}
$$

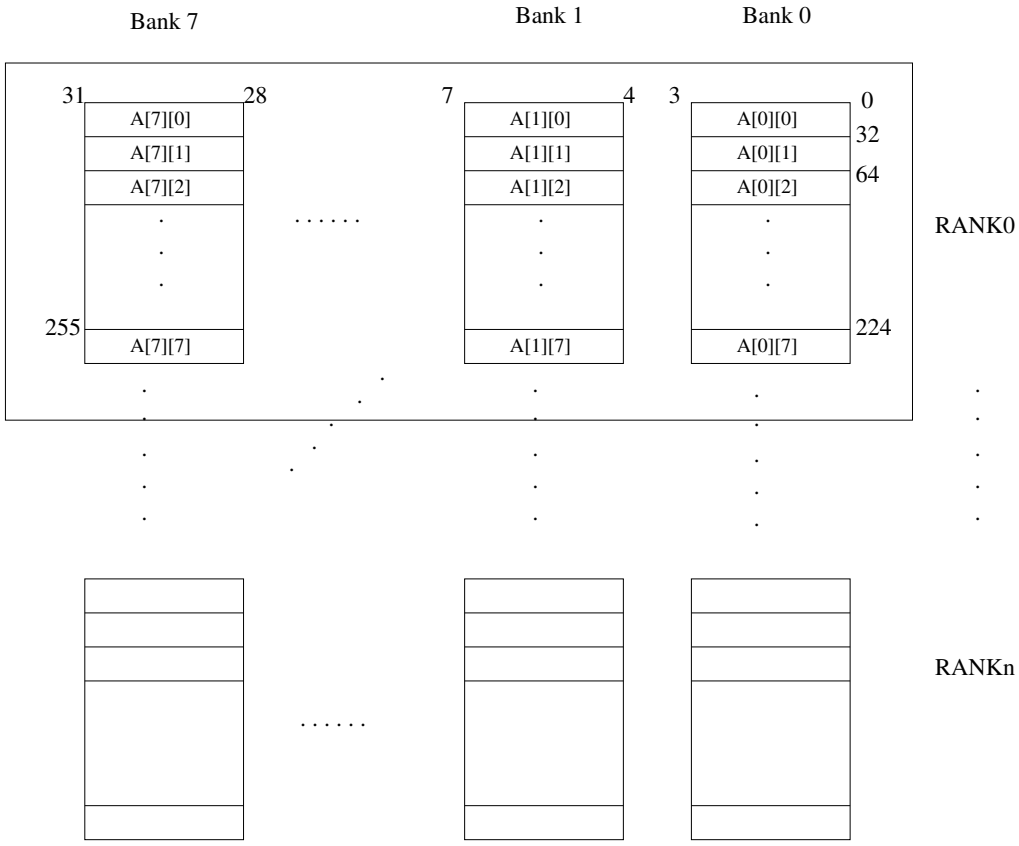
[2]Yoongu Kim, Ross Daly, Jeremie Kim, Chris Fallin, Ji Hye Lee, Donghyuk Lee, Chris Wilkerson, Konrad Lai, and Onur Mutlu, "Flipping Bits in Memory without Accessing Them: an Experimental Study of DRAM Disturbance Errors," in Proceedings of the 41st Annual International Sympoisum on Computer Architecture, 2014. `https://people.inf.ethz.ch/omutlu/pub/dram-row-hammer_isca14.pdf`

12. Main Memory Organization and Interleaving [150 points]

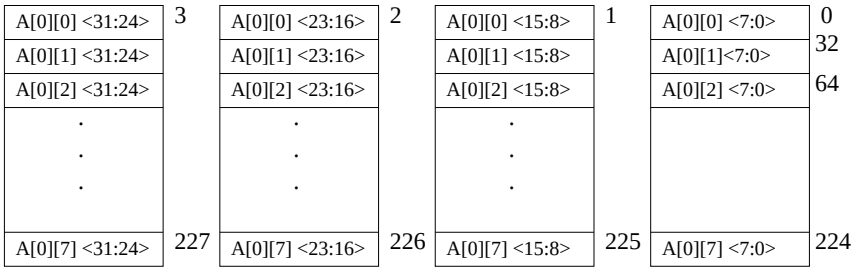
Consider the following piece of code:

```
for(i = 0; i < 8; ++i){
  for(j = 0; j < 8; ++j){
    sum = sum + A[i][j];
  }
}
```

The figure below shows an 8-way interleaved, byte-addressable memory. The total size of the memory is 4KB. The elements of the 2-dimensional array, A, are 4-bytes in length and are stored in the memory in column-major order (i.e., columns of A are stored in consecutive memory locations) as shown. The width of the bus is 32 bits, and each memory access takes 10 cycles.



A more detailed picture of the memory chips in Bank 0 of Rank 0 is shown below.



(a) Since the address space of the memory is 4KB, 12 bits are needed to uniquely identify each memory location, i.e., Addr[11:0]. Specify which bits of the address will be used for:

- Byte on bus
  Addr [1 : 0]
- Interleave/Bank bits
  Addr [4 : 2]
- Chip address
  Addr [7 : 5]
- Rank bits
  Addr [11 : 8]

(b) How many cycles are spent accessing memory during the execution of the above code? Compare this with the number of memory access cycles it would take if the memory were not interleaved (i.e., a single 4-byte wide array).

Assumptions: Requests are scheduled in order one after another and any number of requests can be outstanding at a time (limited only by bank conflicts).

As consecutive array elements are stored in the same bank, they have to be accessed serially. However, the memory access in the last iteration of the inner for loop (A[x][7]) and the memory access in the first iteration of the next time through the inner for loop (A[x+1][0]) go to different banks and can be accessed in parallel.
Therefore, number of cycles = $(10 \times 8 \times 8)$ - (9×7) = 577 cycles.

(c) Can any change be made to the current interleaving scheme to optimize the number of cycles spent accessing memory? If yes, which bits of the address will be used to specify the byte on bus, interleaving, etc. (use the same format as in part a)? With the new interleaving scheme, how many cycles are spent accessing memory? Remember that the elements of A will still be stored in column-major order.

Elements along a column are accessed in consecutive cycles. So, if they were mapped to different banks, accesses to them can be interleaved.
This can be done by using the following address mapping:

- Byte on bus
  Addr [1 : 0]
- Interleave/Bank bits
  Addr [7 : 5]
- Chip address
  Addr [4 : 2]
- Rank bits
  Addr [11 : 8]

The first iteration of the outer loop starts at time 0. The first access is issued at cycle 0 and ends at cycle 10. Accesses to consecutive banks are issued every cycle after that until cycle 7 and these accesses complete by cycle 17. The first access of the second iteration of the outer loop can start only in cycle 10, as bank 0 is occupied until cycle 10 with the first access of the previous outer loop iteration. The second iteration's accesses complete at cycle 27. Extending this, the eighth iteration's first access starts at cycle 70 and completes by cycle 87.
Therefore, number of cycles = 87 cycles.

(d) Using the original interleaving scheme, what small changes can be made to the piece of code to optimize the number of cycles spent accessing memory? How many cycles are spent accessing memory using the modified code?

```
for(i = 0; i < 8; ++i){
  for(j = 0; j < 8; ++j){
    sum = sum + A[i][j];
  }
}
```

The inner and outer loop can be reordered as follows:

for(j = 0; j < 8; ++j)
for(i = 0; i < 8; ++i)
sum = sum + A[i][j];

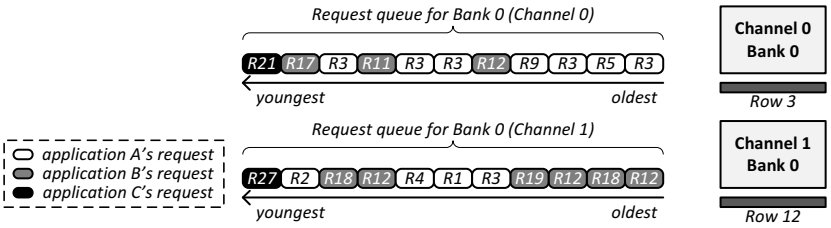
Thus, elements along a row, which are stored in consecutive banks (using the original interleaving scheme), will be accessed in consecutive cycles. The effect this has on the number of cycles is exactly the same as part (c).
Therefore, number of cycles = 87 cycles.

13. Memory Scheduling [200 points]

To serve a memory request, the memory controller issues one or multiple DRAM commands to access data from a bank. There are four different DRAM commands.

- `ACTIVATE`: Loads the row (that needs to be accessed) into the bank's row-buffer. This is called *opening* a row. **(Latency: 15ns)**
- `PRECHARGE`: Restores the contents of the bank's row-buffer back into the row. This is called *closing* a row. **(Latency: 15ns)**
- `READ/WRITE`: Accesses data from the row-buffer. **(Latency: 15ns)**

The following figure shows the snapshot of the memory request buffers (in the memory controller) at t_0. Each request is color-coded to denote the application to which it belongs (assume that all applications are running on separate cores). Additionally, each request is annotated with the address (or index) of the row that the request needs to access (e.g., *R3* means that the request is to the 3^{rd} row). Furthermore, assume that all requests are read requests.



A memory request is considered to be *served* when the `READ` command is complete (i.e., 15ns after the request's `READ` command has been issued). In addition, each application (A, B, or C) is considered to be **stalled** until *all* of its memory requests (across all the request buffers) have been served.

Assume that, initially (at t_0) each bank has the 3^{rd} and the 12^{th} row loaded in the row-buffer, respectively. Furthermore, no additional requests from any of the applications arrive at the memory controller.

13.1. Application-Unaware Scheduling Policies

(a) Using the **FCFS** scheduling policy, what is the **stall time** of each application?

App A: MAX(2H+7M, H+9M) = H+9M = 15+405 = 420ns

App B: MAX(2H+8M, H+8M) = 2H+8M = 30+360 = 390ns

App C: MAX(2H+9M, H+10M) = H+10M = 15+450 = 465ns

(b) Using the **FR-FCFS** scheduling policy, what is the stall time of each application?

App A: MAX(5H+2M, (4H+2M)+4M) = 4H+6M = 60+270 = 330ns

App B: MAX((5H+2M)+3M, 4H+2M) = 5H+5M = 75+225 = 300ns

App C: MAX(((5H+2M)+3M)+M, ((4H+2M)+4M)+M) = 4H+7M = 60 + 315 = 375ns

(c) What property of memory references does the *FR-FCFS* scheduling policy exploit? (Three words or less.)

Row-buffer locality

(d) Briefly describe the scheduling policy that would **maximize** the *request throughput* at any given bank, where request throughput is defined as the number of requests served per unit amount of time. (Less than 10 words.)

FR-FCFS

13.2. Application-Aware Scheduling Policies

Of the three applications, application C is the least memory-intensive (i.e., has the lowest number of outstanding requests). However, it experiences the largest stall time since its requests are served only after the numerous requests from other applications are first served. To ensure the shortest stall time for application C, one can assign its requests with the highest priority, while assigning the same low priority to the other two applications (A and B).

(a) **Scheduling Policy X:** When application C is assigned a high priority and the other two applications are assigned the same low priority, what is the stall time of each application? (Among requests with the same priority, assume that *FR-FCFS* is used to break ties.)

App A: MAX(M+(4H+3M), M+(3H+3M)+4M) = 3H+8M = 45+360 = 405ns

App B: MAX(M+(4H+3M)+3M, M+(3H+3M)) = 4H+7M = 60+315 = 375ns

App C: MAX(M, M) = M = 45ns

Can you design an even better scheduling policy? While application C now experiences low stall time, you notice that the other two applications (A and B) are still delaying each other.

(b) Assign priorities to the other two applications such that you minimize the average stall time across all applications. Specifically, list all **three** applications in the order of highest to lowest priority. (Among requests with the same priority, assume that *FR-FCFS* is used to break ties.)

C > B > A

(c) **Scheduling Policy Y:** Using your new scheduling policy, what is the stall time of each application? (Among requests with the same priority, assume that *FR-FCFS* is used to break ties.)

App A: MAX(M+(3M)+(4H+3M), M+(3H+3M)+4M) = 3H+8M = 45+360 = 405ns

App B: MAX(M+(3M), M+(3H+3M)) = 3H+4M = 45+180 = 225ns

App C: MAX(M, M) = M = 45ns

(d) Order the four scheduling policies (FCFS, FR-FCFS, X, Y) in the order of lowest to highest average stall time.

Y<X<FR-FCFS<FCFS