# 15-418/618: Spring 2022 Exam #2

- You will prepare your answers using the answer spaces provided in this exam. Write your answers in the spaces provided; you may print and scan the exam or add your answers digitally.
- You will be given 48 hours to complete the exam, i.e., until 11:59pm EST on Friday 4/15. Submit your exam to Gradescope.
- If there is some reason you do not believe you will be able to make this deadline, you must inform us by Monday 4/11 and get an explicit waiver from us.
- You may access any course materials but, unless explicitly told otherwise, *do not access any other materials.*
- All requests for clarifications must be made via *private* Piazza posts, and all clarifications will be responded to via a *public* FAQ within the first 24 hrs. Do *not* ask for clarifications in public.
- The exam will be designed so that someone with complete preparation can complete it in two hours. We are allowing 48 hours only to provide more flexibility, and to account for the fact that students may be operating in different time zones.

| | |
|---|---|
| Problem 1 | 26 |
| Problem 2 | 12 |
| Problem 3 | 16 |
| Problem 4 | 16 |
| Problem 5 | 18 |
| Problem 6 | 12 |
| **Total** | **100 points** |

Sign the following pledge:

I do hereby swear that, in generating my answers to this exam, I made use of only course resources or others with explicit permission. I did not have any communication of any form with anyone other than the course instructors and teaching assistants about the contents of this exam.

____________________________________
Your signature

# Problem 1: Multiple choice [2pts each]

Read each of the following questions & answers carefully. **Exactly one answer is correct.**

## A. Directory-based coherence

Which is true about a directory-based cache coherent NUMA system with a full-map directory?

A. This type of cache coherence is better than snooping-based cache coherence when global memory has long latency.
B. In a read miss, the requesting cache controller avoids broadcasting by looking up its home node in the directory.
C. Limited pointers are better than full-map directories when a parallel program involves many processors holding the same line in memory.
D. If we keep the cache size fixed, increasing the block size can reduce the directory size.

## B. Performance measurements & tuning

Which of the following statements is **not** a use case of the tool *Perf*?

A. Checking that function invariants (e.g., pre- and post-conditions) are maintained.
B. Finding which syscalls are taking the most time.
C. Counting cache misses in the program.
D. Profiling assembly-level code of some functions.

## C. Memory consistency

Which of the following is **not** true about Total Store Ordering (TSO)?

A. TSO is motivated by the widespread use of store buffers in processor designs.
B. TSO allows loads to be reordered with stores to different addresses.
C. TSO allows loads to be reordered with stores to the same address.
D. TSO can improve performance over sequential consistency.

## D. Interconnection networks

Which of the following is true about interconnection networks?

A. For a crossbar interconnect of 36 network nodes, there are 36 switches.
B. Communication cost is higher starting from the center of a 2D mesh than at the edges.
C. Wormhole flow control eliminates the head of line blocking problem.
D. Circuit-switched routing tends to perform poorly on short messages.

## E. Synchronization

Which of the following statements is correct about synchronization?

A. Busy-waiting is preferable to blocking when no other task requires the processor's resources.
B. In test-and-test-and-set lock, the processor waiting to acquire a lock will send BusRdX in the first test phase.
C. Test-and-set lock with exponential backoff can guarantee fairness.
D. If we don't pad the array-based lock, the interconnect traffic pre-release will be linear to the number of acquiring threads.

## F. Transactional memory

Which of the following statements is correct about transactional memory?

A. With the protection of transactional memory, multiple threads accessing the same critical region will not happen.
B. If a long memory transaction is doomed to conflict with another transaction, optimistic detection of conflict is generally preferable to pessimistic detection.
C. In hardware transactional memory, a cache line's annotation bits are cleared only when a transaction commits.
D. If a memory transaction only contains read operations, it will always commit.

## G. Fine-grain synchronization and lock-free programming

Which of the following statements are always true:

A. Data structures that have locks on insertions are blocking data structures.
B. Comparing the address of a data structure’s node during a CAS operation will ensure that the node has not changed since it was read.
C. Lock-free designs eliminate contention.
D. There exists a locking scheme that is correct for every data structure.

## H. MPI, OpenMP, Cilk implementation

It’s better to choose conservative asynchronous message passing over optimistic asynchronous message passing when:

(1) Most messages are short (2) Most messages are large bulk data transfer
(3) Receive buffers are large (4) Receive buffers are small and might overflow

A. (1)(3)
B. (1)(4)
C. (2)(3)
D. (2)(4)

## I. Heterogeneous Parallelism

Suppose there is an asymmetric set of total $n$ processing resources. Assume $r$ is the resources dedicated to the 'high-performance' core, and all other cores have resource = 1. Compared with a system with $n$ total resources and $r\ (>1)$ resources for each core, which of the following $perf(r)$ would let the asymmetric system benefit from heterogeneity?

A. $perf(r) = r$
B. $perf(r) = log(r)$
C. Both of the above
D. Neither of the above

## J. Domain-specific languages

Which of the following is true about the Halide DSL?

A. Halide is implemented as a standalone programming language.
B. Halide allows the user to separately describe the algorithm and its execution schedule.
C. Halide can only be used to write image processing algorithms.
D. Halide allows the programmer to use arbitrary control flow such as mutual recursion.

## K. Domain-specific frameworks

Which of the following statements is true:

A. Domain-specific frameworks require specialized hardware.
B. Domain-specific frameworks require custom programming languages.
C. Domain-specific frameworks increase performance portability on different platforms.
D. Domain-specific frameworks are difficult to use but increase performance.

## L. DNNs (1/2)

Which of the following statements is **not** correct about DNNs?

A. Model parameters are updated after each mini-batch.
B. Training has less memory footprint than inference.
C. We can use pruning to compress the model for inference.
D. Convolution kernels can be accelerated with matrix multiplication techniques.

## M. DNNs (2/2)

Which of the following statements is correct about model training?

A. Data-parallel training reduces the memory footprint to store parameters in each device.
B. Model-parallel training reduces the memory footprint to store parameters in each device.
C. In the parameter-server data-parallel setting, the communication cost per device increases as we increase the number of worker devices in data-parallel training.
D. A normal allreduce data-parallel setting is not susceptible to stale worker problem.

## Problem 2: Memory Consistency [12 pts]

Consider the following code segments. `X`, `Y`, `Z`, and `flag` are shared variables located in **main memory**. `reg1`, `reg2`, `reg3`, and `reg4` are local **registers**. All variables are initialized to 0.

| P1 | P2 |
|---|---|
| `X = 1`<br>`Y = 1`<br>`while (flag != 0) {}`<br>`reg1 = X`<br>`reg2 = Y`<br>`flag = 1` | `X = 2`<br>`while (flag != 1) {}`<br>`Y = 2`<br>`reg4 = Y`<br>`reg3 = X` |

*Note:* Only 1 load/store instruction is required for each line of the code except the while loop.

### Problem 2.1 A simple multiprocessor machine

Suppose the above code runs on a sequentially consistent machine:

- Each thread's memory operations execute in program order.
- Main memory only has 1 port and can serve one processor at a time.

**[4 pts]** What are the possible values that the 4 reads on X, Y can return? Mark an 'X' in the table below for each register if it can take the corresponding value in the column. (Hint: There can be multiple possibilities for each register.)

| | **Value == 1?** | **Value == 2?** |
|---|---|---|
| **reg1** | | |
| **reg2** | | |
| **reg3** | | |
| **reg4** | | |

### Problem 2.2 FIFO Write buffer

Building on the system in problem 2.1, a write buffer is implemented for each processor.

- A store instruction will be considered complete once the processor puts the store request in the write buffer.
- The write buffer can handle multiple outstanding store requests.

- All loads execute in program order.
- All stores are executed in program order (i.e., the store buffer is FIFO).
- Each cycle, the processor can issue the oldest store request in the write buffer or an outstanding load request (if there is one) to the main memory.
- All stores will eventually appear at memory.

**[4 pts]** What are the possible values that the 4 reads on X, Y can return? Mark an ‘X’ in the table below for each register if it can take the corresponding value in the column. (Hint: There can be multiple possibilities for each register.)

| | **Value == 1?** | **Value == 2?** |
|---|---|---|
| **reg1** | | |
| **reg2** | | |
| **reg3** | | |
| **reg4** | | |

## Problem 2.3 Out-of-order execution

Building on the system in problem 2.2, the processor further implements out-of-order execution, i.e., it can issue any executable instruction regardless of program order.

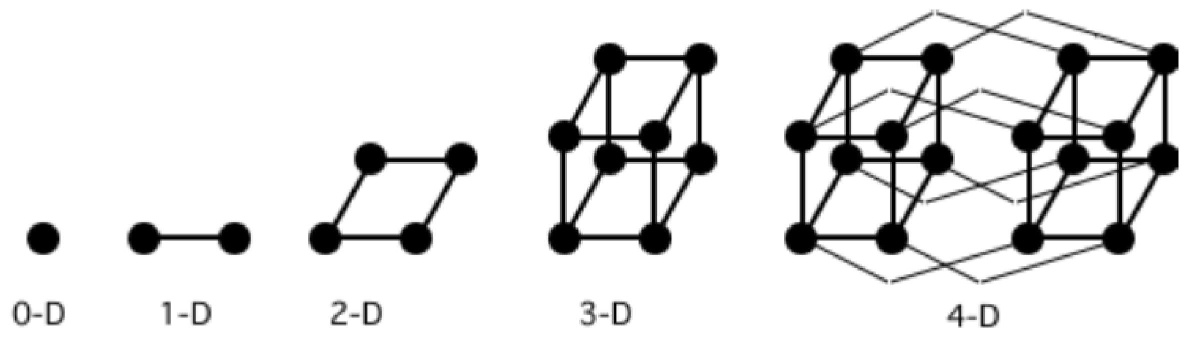
**[4 pts]** What are the possible values that the 4 reads on X, Y can return? Mark an ‘X’ in the table below for each register if it can take the corresponding value in the column. (Hint: There can be multiple possibilities for each register.)

| | **Value == 1?** | **Value == 2?** |
|---|---|---|
| **reg1** | | |
| **reg2** | | |
| **reg3** | | |
| **reg4** | | |

# Problem 3: Interconnection network [16pts]

## Problem 3.1 Hypercube interconnection

Consider the following diagrams of hypercube interconnection topology when dimension *m = 0, 1, 2, 3, 4, …*



The construction of the m dimension hypercube when *m > 0* is as follows:

a. Replicate the *(m-1)* dimension hypercube.

b. Link every corresponding/symmetric node pair from the two replicas.

And the case when *m = 0* is defined as just one single node.

### 3.1.1 Number of Nodes *N*

**[2pts]** Explicitly express *N(m)* in terms of *m*.

### 3.1.2 Bisection Bandwidth *B*

**[2pts]** Explicitly express *B(m)* in terms of m. (Assuming that *m > 0*)

## 3.1.3 Cost *C* (measured as total number of links)

**[2pts]** Explicitly express *C(m)* in terms of *C(m – 1)* and *N(m)*. (Assuming that $m > 0$)

## 3.1.4 Latency *T* (for any node pair, *T* is measured as number of links on the shortest path between the two nodes)

**[4pts]** Explain why the network diameter *Tmax(m)* = *m*. (hint: try induction)

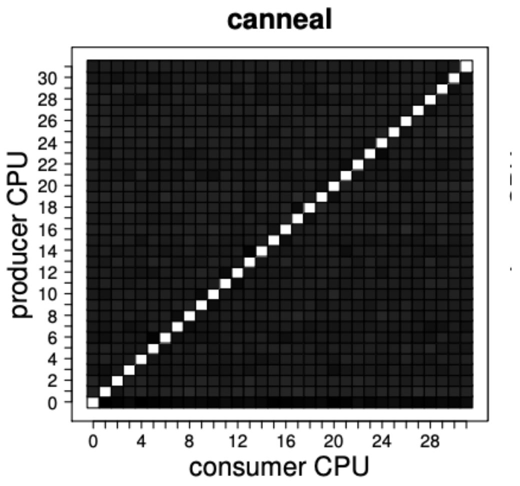
## Problem 3.2 Best interconnection for a communication pattern

The following plots show the communication pattern between pairs of processors. A darker square means the pair of processors communicate more frequently, while a lighter square means there is less communication.

For each plot, there is one interconnection network most suitable for the pattern among the candidates provided. You should also explain your choice from **both** latency **and** cost aspects.

### 3.2.1 "Canneal" communication

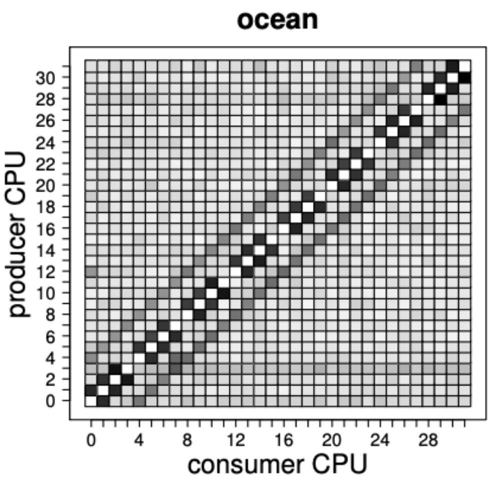
**[3pts]** Based on the communication pattern shown below, which network topology is most suitable: unidirectional ring, crossbar or H-tree?



Explain your choice in terms of latency and cost.

## 3.2.2 “Ocean” communication

**[3pts]** Based on the communication pattern shown below, which network topology is most suitable: unidirectional ring, crossbar or H-tree?



Explain your choice in terms of latency and cost.

## Problem 4: Synchronization and lock-free programming [16 pts]

In this problem, we will build on the lock-free linked list presented in class. For your reference, here is the code snippet, as presented in class, to insert a new node in a lock-free linked list:

```
struct Node {
  int value;
  Node* next;
};

struct List {
  Node* head;
};

// insert new node after specified node
void insert_after(List* list, Node* after, int value) {
  Node* n = new Node;
  n->value = value;

  // assume case of insert into empty list handled
  // here to keep code simple

  Node* prev = list->head;

  while (prev->next) {
    if (prev == after) {
      while (1) {
         Node* old_next = prev->next;
         n->next = old_next;
         if (compare_and_swap(&prev->next, old_next, n) == old_next)
           return;
      }
    }
    prev = prev->next;
  }
}
```

## Problem 4.1: Simple deletion in lock-free linked list

Imagine that we implement a simple version of delete in this lock-free linked list as seen below:

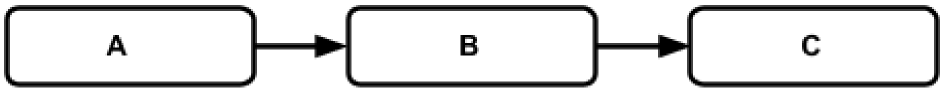
```
void delete(List* list, Node* n){

  // assume delete in a zero or one element list is handled

  Node* prev = list->head;

  while (prev->next) {
    if (prev->next == n) {
      while (1) {
         if (compare_and_swap(&prev->next, n, n->next) == n->next) {
           delete n;
           return;
        }
      }
    }
    prev = prev->next;
  }
}
```

**[4 pts]** Unfortunately, this linked list implementation isn't correct. Imagine we have the following list:



What are two function calls that if performed concurrently will cause this implementation to produce an incorrect linked list? Explain why this causes incorrect behavior, what the resulting linked list will contain, and what it should contain.

## Problem 4.2: Fixing deletion in lock-free linked list

To remedy this problem, we need to revisit our `Node` implementation. Imagine instead we add a `marker` to each node to represent whether we are currently trying to delete it.

```
struct Node {
  int value;
  union {
    Node* next;
    bool marker;
  }
};
```

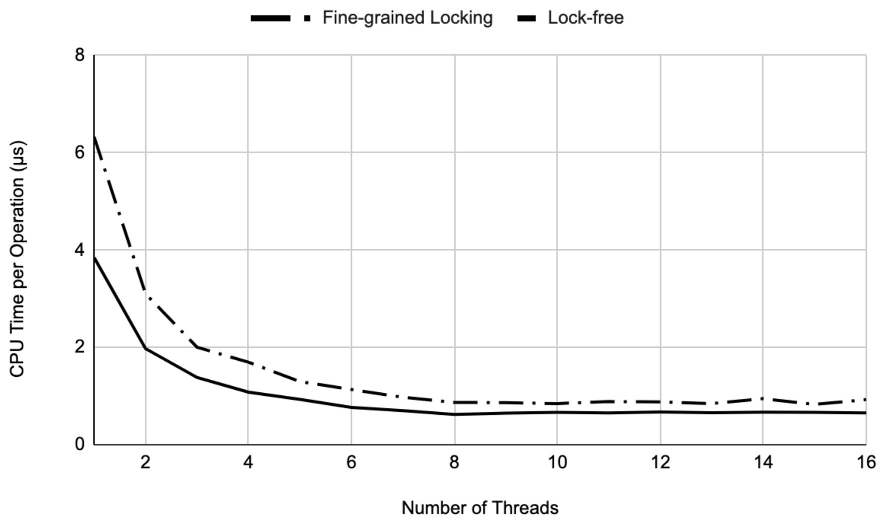
Now instead of doing one CAS operation to perform a delete, we first set the marker field to 1 in the deleted node during the first CAS operation. Then, we perform a second CAS operation to remove the node (as we did in the original delete code). We also change `insert_after` to skip nodes where the marker field is set to 1.

**[4 pts]** Why does this implementation prevent the problem you described in part 1?

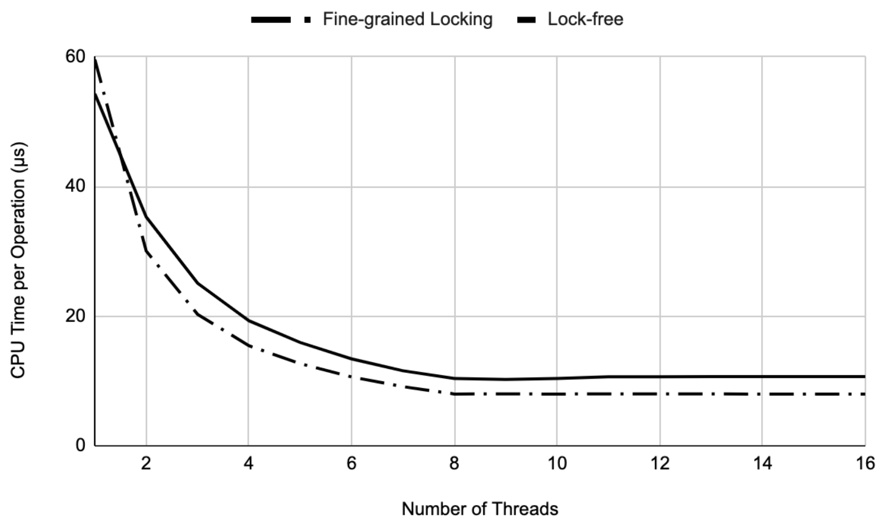
## Problem 4.3: Comparing locking and lock-free linked lists

We have implemented a lock-free linked list using the ideas discussed in Part 2 and a fine-grained locking linked list. Instead of locking and unlocking nodes along the way when the insert/delete operation tries to locate the target position, our improved version advances the pointer as close to the target position as possible without grabbing any lock and then tries to lock the previous node of the target position. As a result, our fine-grained locking linked list is an improved version compared to the "Padded List hand-over-hand" locking algorithm discussed in the lecture. The following graphs show performance benchmarks for the two linked list implementations on a 8-core GHC machine.

The first experiment initialized each linked list with integers ranging from 1 to $2^{12}$. It then spawned N threads, each of which performed 12.5% insert, 12.5% delete, 75% find operations within the same range. Below is the graph of the performance of these two implementations (measured by CPU time per operation) versus number of threads.



The second experiment initialized each linked list with integers ranging from 1 to $2^{15}$. It then spawned N threads, each of which performed 50% insert and 50% delete operations within the same range. The performance of these two implementations versus the number of threads is also shown below.



**[4 pts]** Why is fine-grained locking worse in the first experiment, but better in the second experiment?

**[4 pts]** Imagine you were a software engineer and your manager told you to implement a concurrent linked list implementation. When would you implement a coarse-grained locking linked list, a fine-grained locking linked list, or a lock-free linked list? Name one circumstance in which you would implement each and why.

## Problem 5: Heterogeneous Computing [18 pts]

Banana Inc. is one of the world's premiere consumer electronics companies, and has asked you to design some high performance processors using ideas from heterogeneous computing.

For this problem, we will perform the task of photorealistic image synthesis. Consider the following pseudocode that takes a scene description and, in sequence: 1) ray-traces a photorealistic image of the scene, 2) passes the raw image through a denoising neural network to improve image quality, and 3) encodes the outputted image array into a compressed format and writes the result to disk:

```
void renderScene(const std::string &sceneName, const std::string &outputPath) {
    scene = loadSceneFromDisk(sceneName);
    imageRaw = raytrace(scene);
    imageClean = denoisingNet(imageRaw);
    imageEncoded = imageEncode(imageClean);
    saveImageToDisk(imageEncoded, outputPath);
}
```

Your task is to select the best chip design for a processor that will be used specifically for this task. Chips are made using four hardware components:

- **CPU**: CPUs can perform all kinds of computation, and are best suited for tasks that are difficult to parallelize or not performed commonly enough to justify hardware specialization (i.e. `loadSceneFromDisk`, `saveImageToDisk`).
- **GPU**: GPUs are best suited for graphics and data-parallel applications (i.e. `raytrace`, `denoisingNet`, and `imageEncode`).
- **Media Engine**: A component that is optimized to only perform image encoding and decoding (i.e. `imageEncode`).
- **Neural Engine**: A component that is optimized to only perform neural network inference (i.e. `denoisingNet`).

To specify a chip design, you must describe the number of cores in each hardware component. Each added core takes up some chip area. **The maximum allowed chip area is 13 mm$^2$.**

For each hardware component, the table on the next page describes the area per core and the speed per core on all functions in `renderScene`, measured in Work Units per second (WU/s).

**Not all tasks are parallelizable**, as shown in the table below. For tasks that are parallelizable, we assume that they achieve **perfect speedup across cores**, but can **only run on one type of hardware** at a time (i.e., a task cannot simultaneously run on CPU and GPU at once). We also assume that the five tasks in the pseudocode above must run one after another.

|  | Parallelizable? | CPU | GPU | Media Engine | Neural Engine |
|---|---|---|---|---|---|
| Area per core |  | $1mm^2$ | $2mm^2$ | $4mm^2$ | $4mm^2$ |
| Speed: `loadSceneFromDisk()` | No | 1 WU/s | -- | -- | -- |
| Speed: `raytrace()` | Yes | 0.1 WU/s | 4 WU/s | -- | -- |
| Speed: `denoisingNet()` | Yes | 0.1 WU/s | 4 WU/s | -- | 32 WU/s |
| Speed: `imageEncode()` | Yes | 0.1 WU/s | 2 WU/s | 24 WU/s | -- |
| Speed: `saveImageToDisk()` | No | 1 WU/s | -- | -- | -- |

Finally, we also assume that **it takes 1s to transfer data** between any pair of hardware components. A solution that computes `loadSceneFromDisk` on CPU, `rayTrace / denoisingNet / imageEncode` on GPU, and `saveImageToDisk` on CPU would spend 2s in total on data transfer: 1s from CPU → GPU at the start, and 1s from GPU → CPU at the end.

Here are some additional examples to help illustrate the setup:

- On a chip with 8x CPU cores, running 32 WU in `loadSceneFromDisk` and no other tasks, your program takes 32s to compute as the task cannot be parallelized.
- On a chip with 2x Neural Engine cores, running 128 WU in `denoisingNet` and no other tasks, your program takes 2s to compute plus any data transfer time.

## Problem 5.1. Optimal Runtime Using All Components

**[3 pts]** Write an expression for the runtime of `renderScene` as a function of the following terms. For this subpart only, you should assume that each hardware component has at least one core (i.e. $N_C$, $N_G$, $N_M$, $N_N$ >= 1), and that you always use the hardware component with fastest speed per core for each task (i.e., always use GPU cores for `raytrace`, Neural Engine cores for `denoisingNet`, and Media Engine cores for `videoEncode`).

- $W_L$: Number of work units in `loadSceneFromDisk()`
- $W_R$: Number of work units in `raytrace()`
- $W_D$: Number of work units in `denoisingNet()`
- $W_V$: Number of work units in `videoEncode()`
- $W_S$: Number of work units in `saveVideoToDisk()`
- $N_C$: Number of CPU cores
- $N_G$: Number of GPU cores
- $N_M$: Number of Media Engine cores
- $N_N$: Number of Neural Engine cores

## Problem 5.2. To Specialize or Not To Specialize

Design a chip that achieves the best possible runtime on a task with the following workload:

| | |
|---|---|
| ● $W_L$ = 1 WU | (`loadSceneFromDisk`) |
| ● $W_R$ = 64 WU | (`raytrace`) |
| ● $W_D$ = 32 WU | (`denoiseNet`) |
| ● $W_V$ = 24 WU | (`imageEncode`) |
| ● $W_S$ = 1 WU | (`saveImageToDisk`) |

**[6 pts]** Describe your optimal chip design, including how many cores each component has and the chip's total runtime on this workload. Intuitively, why does this design perform better than some of the other designs that you considered? If you'd like, feel free to write a short program or spreadsheet for your computations. Remember that the maximum allowed chip area is $13mm^2$.

## Problem 5.3. Upsampling the Output

Your boss at Banana has asked you to increase your output resolution from 720p to 6K so that it fits on the latest displays. Since raytracing is expensive, you decide to keep the same output resolution for `raytrace` and add some upsampling layers to `denoiseNet`, which changes the workload above to the following:

| | |
|---|---|
| • $W_L$ = 1 WU | (`loadSceneFromDisk`) |
| • $W_R$ = 64 WU | (`raytrace`) |
| • $W_D$ = 128 WU | (`denoiseNet` + upsampling) |
| • $W_V$ = 288 WU | (`imageEncode`) |
| • $W_S$ = 12 WU | (`saveImageToDisk`) |

**[6 pts]** Describe your optimal chip design for this modified workload, including how many cores each component has and your chip's total runtime. Intuitively, why is your previous chip design from Part 5.2 less well-suited for this modified workload?

## Problem 5.4. Analysis and Takeaways

**[3 pts]** Based on your results above, discuss two factors that engineers should consider when designing a high-performance heterogeneous processor with specialized compute units. How does each factor that you listed affect the processor's performance?

# Problem 6: Deep Neural Networks Parallelism [12 pts]

## Problem 6.1 Parameter Server

Given a data parallel setting with the following parameter server design:

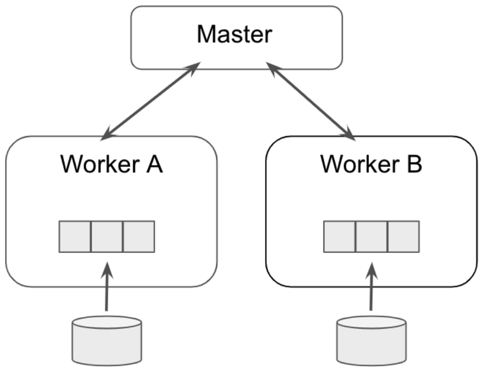


Figure: Illustration of Parameter Server Strategy

**[4 pts]** What is the right order of the following actions (*Hint:* Some actions might be wrong; Some steps may not be included in the process)

(a) The parameter server broadcasts the updated parameters to all the worker nodes
(b) The parameter server sends initial parameters to all the worker nodes
(c) The worker nodes send partitioned training data to the parameter server
(d) The parameter server computes the new parameters with collected gradients
(e) The worker nodes compute the local gradients independently
(f) The worker nodes send the local gradients to the parameter server

**[2 pts]** Write an expression for the total amount of data being sent over the network for one epoch, according to the steps described in the previous question. (*Hint:* assume the training data is already pre-distributed among the worker nodes evenly)
Useful notation:
$N$ – the number of worker nodes
$M$ – the total size of the parameters/gradient
$D$ – the total size of training data

## Problem 6.2 AllReduce vs. RingReduce

Now, consider two different reduce-update strategies: the Naive AllReduce and the Ring AllReduce introduced as below. What are the total communication costs for the two topologies, respectively? (*Hint:* Use Big O notation.)

Useful notation:
$N$ – the number of worker nodes
$M$ – the total size of the parameters/gradient
$D$ – the total size of training data

### 6.2.1 Naive AllReduce

Each worker sends their **local gradients** to others, then each worker uses aggregated gradients to update the local parameter.

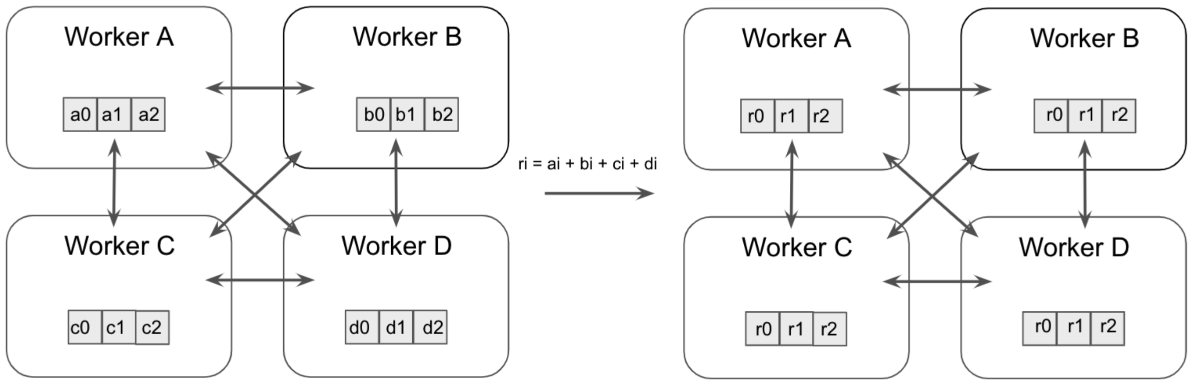


Figure: Illustration of AllReduce Strategy

**[3 pts]** 6.2.1 What is the total amount of data being sent over the network for one epoch? Explain your answer.

## 6.2.2 RingReduce

**Step1.** Each worker sends one slice of (M/N) gradient to the next worker, and repeats until each worker has the aggregated version of M/N gradient .

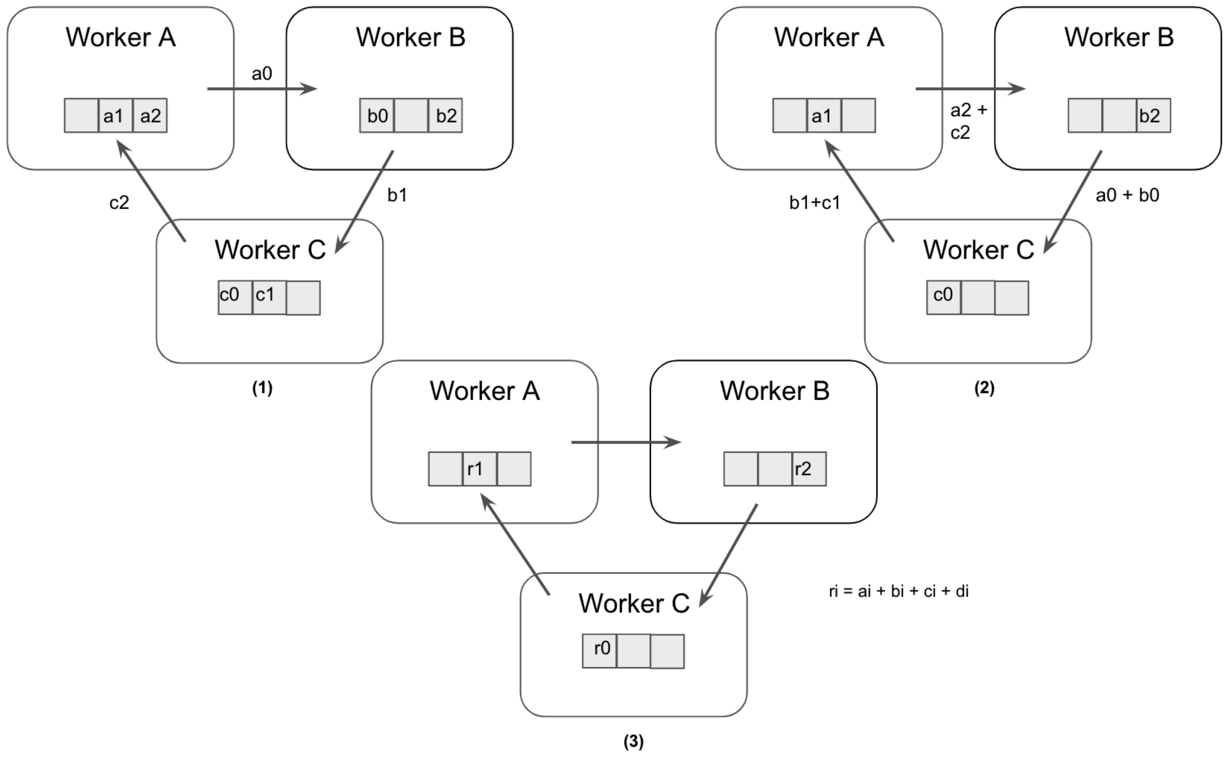


Figure: Illustration of Step 1 in RingReduce Strategy

**Step 2.** Each worker sends one slice of aggregated gradient to the next worker, and repeats until each worker has the complete version of M gradient. Then each worker uses the gradient to update their local parameter.

**[3 pts]** 6.2.2 What is the total amount of data being sent over the network for one epoch? Explain your answer.