Full Name:

Andrew Id:

15-418/618
Exam 2
Answer Sheet

Problem 1: Multiple Choice (15 points)

Problems that state "list all that apply" may have any number of correct answers (including zero). If none are correct, list "none." The other problems have unique answers.

A. Which of the following memory consistency policies allow the following behavior? Assume x and y start with value zero. (List all that apply.)

```
Thread 0        Thread 1
--------        --------
x <- 1          y <- 1
print(y)        print(x)

Output:
0
0
```

(a) Non-coherent shared memory.

(b) Coherent shared memory.

(c) Total-store-order (TSO) consistency.

(d) Sequential consistency.

Answer:

B. What is the key optimization enabled by total-store-order (TSO) consistency vs. sequential consistency (SC)? (List all that apply.)

(a) MESI coherence.

(b) Prefetching.

(c) Atomic memory operations.

(d) Store buffers.

Answer:

C. Which of the following statements are true about implementations of MPI (list all that apply)

(a) MPI assumes the program is running on a machine with a cache-coherent shared address space.

(b) The MPI library allocates the needed space for all buffers.

(c) Synchronous communication creates a synchronization barrier for the sender and receiver.

(d) A sender can modify the send buffer as soon as the call to `MPI_Send` returns.

Answer:

D. Which of the following statements are true about implementations of OpenMP (list all that apply)

(a) OpenMP assumes the program is running on a machine with a cache-coherent shared address space.

(b) OpenMP can only exploit parallelism by splitting up the index ranges in `for` loops.

(c) OpenMP implements all atomic operations via locks.

(d) OpenMP reduction assumes the reduction operation is associative.

Answer:

E. Assume that in a computer system, all operations continually abort and retry. Which one of the following could describe this scenario?

(a) Deadlock.

(b) Livelock.

(c) Starvation.

(d) None of the above.

Answer:

F. To implement a lock-free single reader, single writer unbounded queue, how many additional pointers does the program need beyond those needed for a sequential implementation?

(a) 0

(b) 1

(c) 2

(d) 3

Answer:

G. Which of the following statements is/are true? List all that apply.

(a) Test & Test and Set locks guarantee fairness of locks, as opposed to Test and Set locks

(b) It is better to spinlock than block a thread if scheduling overhead is larger than expected wait time

(c) You can replace all uses of locks with atomic transaction regions

(d) Pessimistic conflict detection detects conflicts earlier than optimistic detection

Answer:

H. Which of the following are domain-specific languages? List all that apply.

(a) Ruby.

(b) Liszt.

(c) Halide.

(d) C++.

Answer:

I. You are tasked with implementing a hardware transactional memory system in a new multicore processor with a local cache for each processor. Which of the following pieces of information are necessary for you to proceed with your implementation? (List all that apply)

(a) The cache coherence protocol

(b) The clock rate of each core

(c) The number of cores

(d) The versioning policy

Answer:

J. When training neural networks on a cluster and using a parameter server, what can we always expect?

(a) Parameters are always fresh

(b) The neural network is running in some model-parallel manner

(c) Training will always converge

(d) Updates to the server happen asynchronously

Answer:

K. Which of the following statements is/are true? (list all that apply)

(a) ASICs have longer development workflows than FPGAs

(b) An ASIC circuit is more reconfigurable than a FPGA circuit

(c) ASICs developed for a given task are faster than FPGAs at that task

(d) ASICs developed for a given task use more power than FPGAs for the same task

Answer:

L.
```
lock:     ts  R3, memory[address]
          bnz R3, lock
unlock :  st  memory[address], #0
```

Assume the above assembly code follows a RISC ISA, with **ts** representing the test and set instruction. Assume that four threads running on four different processors are trying to acquire the lock. Which of the following is a/are plausible scenarios due to given lock mechanism?

(a) Starvation

(b) Deadlock

(c) Above lock doesn't ensure mutual exclusion

(d) All of the above

Answer:

M. Which of the following performance evaluation tools has the least overhead?

(a) valgrind

(b) gprof

(c) top

(d) pin

Answer:

N. What is the worst case latency of an n-node mesh network?

(a) $O(\sqrt{n})$

(b) $O(\log n)$

(c) $O(n)$

(d) $O(n \log n)$

Answer:

O. Which of the following interconnect networks is non-blocking?

(a) Mesh

(b) Fat Tree

(c) Tree

(d) Multi-stage Logarithmic

Answer:

Problem 2: Locks and memory consistency (18 points)

2A: Lock properties (12 points)

Provide brief answers to the following questions:

(1) The C keyword `volatile` indicates to the compiler that changes to a value may occur between different accesses, even if it does not appear to be modified. Why must the field `serving` in `ticket_lock_t` be declared as volatile to guarantee correct compilation?

Answer:

(2) Does this lock guarantee fairness (i.e., any request will eventually be granted)? Explain.

Answer:

(3) Why can the incrementing of `next->serving` in function `ticket_lock_unlock` be done with ordinary addition rather than atomic fetch-and-add?

Answer:

(4) We saw in Exercise 4 that we could improve the performance of a lock based on atomic compare-and-swap by inserting a random delay in the busy-waiting loop, with the amount of the delay following an exponential distribution. In similar experiments for a ticket lock, this scheme leads to worse performance. Explain why that could be the case.

Answer:

(5) Suppose the definition of type `ticket_value_t` were changed from `unsigned long` to `unsigned char`. Would this impose any additional limits on the values of `niters` or `nthreads` in the main program? Explain.

Answer:

(6) Suppose you run the main program on an 8-core processor with `nthreads` set to 8. The machine uses a bus-based cache coherence system following an MSI protocol. Assume at some point in the main program execution that Thread 0 is in its critical section and all other threads are in their busy-wait loops, attempting to acquire the lock, with Thread 1 having the smallest ticket value and Thread 7 having the largest. For all of the caches, describe which field(s) of the lock would be held in each cache and what their state(s) would be.

Answer:

(7) Describe all of the bus traffic and cache transitions that would occur due to the following sequence. Describe only the traffic relevant to the lock. Assume there are no conflict misses. For each of the actions listed, describe any resulting actions that would occur by other threads or caches until a steady state is reached.

(a) Thread 0 exits its critical section and releases the lock, and Thread 1 then acquires the lock and enters its critical section.
Answer:

(b) Thread 0 attempts to acquire the lock and re-enters the busy-wait loop.
Answer:

2B: Memory consistency (6 points)

For each of the six possible fence locations either: 1) justify why it is required by describing a scenario in which incorrect behavior could arise without it, or 2) justify why it is not required.

Possible fence #1

Answer:

Possible fence #2

Answer:

Possible fence #3

Answer:

Possible fence #4

Answer:

Possible fence #5

Answer:

Possible fence #6

Answer:

Problem 3: Managing Concurrency (10 points)

Provide brief answers to the following questions:

A. Your friend writes three unit tests that each execute a pair of operations concurrently on the list shown above.

- Test 1: `insert_from_head(1)`, `delete_from_head(9)`
- Test 2: `insert_from_head(8)`, `delete_from_head(2)`
- Test 3: `insert_from_head(8)`, `insert_from_tail(1)`

The first two unit tests complete without error, but the third test goes badly and it does not terminate with the right answer. Describe what behavior is observed and why the problem occurs. (All unit tests start with the list in the state shown above.)

Answer:

B. Imagine you removed all locks and implemented `insert_from_head`, `delete_from_head`, and `insert_from_tail` by placing the entire body of these functions in an atomic block for execution on a system **supporting lazy optimistic transactional memory with aggressive conflict resolution**. Does this fix the correctness problem described in part A? Why or why not?

Answer:

C. Consider two transactions simultaneously performing `insert_from_head(3)` and `delete_from_head(9)`. Assume both transactions start at the same time on different cores and the transaction for `insert_from_head(3)` proceeds to commit while the `delete_from_head(9)` transaction has just iterated to the node with value 6. Must either of the two transactions abort in this situation? Why? **(Remember this is an lazy optimistic transactional memory system!)**

Answer:

D. Must either transaction abort if the transaction for `delete_from_head(9)` proceeds to commit before the transaction for `insert_from_head(3)` does? Why? **Please assume that at the time of the attempted commit, `insert_from_head(3)` has iterated to node 2, but has not begun to modify the list.**

Answer:

E. Must either transaction abort if the situation in part D is changed so that `delete_from_head(9)` attempts to commit first, but by this time `insert_from_head(3)` has made updates to the list (although not yet initiated its commit)? Why?

Answer:

Problem 4: Homogeneous vs. Heterogeneous Parallelism (20 points)

This problem concerns the costs and benefits of heterogeneous parallelism.

We will begin by considering multicore CPU processors, comparing the performance of symmetric (homogeneous) and asymmetric (heterogeneous) designs.

Imagine you are a hardware architect designing a new multicore CPU. Suppose that the CPU can take at most N resources (e.g., resources could be area of at most 200 mm^2 or power of at most 100 W).

You can design CPU cores with the following characteristics:

- **Out-of-order ("OOO"):** The OOO core achieves sequential speedup (vs. some baseline core) as a function of resources r

$$\text{speedup}_O(r) = \sqrt{r+1} - 1 \tag{1}$$

- **In-order ("IO"):** The IO core achieves sequential speedup (vs. some baseline core) as a function of resources r:

$$\text{speedup}_I(r) = 1 - \frac{1}{2r+1} \tag{2}$$

For this problem, all speedups are normalized to a core with sequential performance $= 1$. Feel free to give approximate numerical answers (e.g., within 0.1 of optimal), and we recommend that you write a simple program or use spreadsheet software to get numerical answers.

4A: Homogeneous multicore (7 points): Suppose an application spends a fraction f of its execution running in parallel with ideal speedup, and $1 - f$ executing sequentially.

(1) What is the speedup **in the sequential region** of the program for a single core of size r? (Give an expression involving N, f, and r.)

OOO Core. *Answer*:

IO Core. *Answer*:

(2) What is the speedup **in the parallel region** of the program for a homogeneous multicore? (Give an expression involving N, f, and r.)

OOO Core. *Answer*:

IO Core. *Answer*:

(3) What is the **end-to-end speedup** for a homogeneous multicore? (Give an expression involving N, f, and r.)

OOO Core. *Answer*:

IO Core. *Answer*:

(4) Suppose that $f = 0.9$ and $N = 16$. What speedup is achieved by the **best** homogeneous OOO and IO designs? Which performs better? How many cores does the best design have?

Answer:

(5) Now suppose that $f = 0.8$ and $N = 16$. What speedup is achieved by the **best** homogeneous OOO and IO designs? Which performs better? How many cores does the best design have?

Answer:

(6) Compare your last two answers for $f = 0.9$ to $f = 0.8$. What do they say about how applications affect the best architecture? About the resilience of different architectural choices to different applications?

Answer:

4B: Heterogeneous multicore (8 points): Now suppose that you are able to combine different kinds of cores in your multicore design. Let's consider the performance of a heterogeneous multicore CPU using a mix of OOO and IO cores, written as a function of f, N, r_O (out-of-order core size) and r_I (in-order core size).

Note: The optimal heterogeneous design will consist of a single OOO core plus many IO cores. The sequential region runs on the OOO core.

(1) What is the speedup **in the sequential region** of the program? (Give an expression involving N, f, r_O, and r_I.)

Answer:

(2) What is the speedup **in the parallel region** of the program for a heterogeneous multicore? (Give an expression involving N, f, r_O, and r_I.)

Answer:

(3) What is the **end-to-end speedup** for a heterogeneous multicore? (Give an expression involving N, f, r_O, and r_I.)

Answer:

(4) Suppose that $f = 0.9$ and $N = 16$ and, for now, fix in-order cores at size $r_I = 1$. What speedup does the **best** heterogeneous design achieve?

Answer:

(5) What is the **best** overall speedup when the **in-order core size** r_I **varies** from 0.1, 0.5, 1.0, 2.0, to 4.0?

Answer:

(6) Qualitatively, what does this say about the optimal heterogeneous architecture? (Does it remind you of any designs you've used this semester?)

Answer:

4C: Scaling (5 points): Finally, let's use the model you've already developed to see how things change with different resource budgets. For the rest of this problem, fix $f = 0.9$ and $r_I = 0.1$ in the heterogeneous design.

(1) What are the best **heterogeneous** and **homogeneous** designs when $N = 4$?

Answer:

(2) What are the best **heterogeneous** and **homogeneous** designs when $N = 64$?

Answer:

(3) What does this say about the benefits of heterogeneity as resource budgets increase over time?

Answer:

Problem 5: Memory Performance during Convolution Computation (18 points)

5A: Sequential performance (8 points)

A. Consider the following code:

```
#define LEN 5932
float data[LEN];
float sum = 0.0;
for (int i = 0; i < LEN; i++)
   sum += data[i];
```

Calculate the CPIS for this code assuming array `data` is initially in the following memory levels. Remember: report these numbers to the first decimal place.

L1 cache. *Answer*:

L2 cache. *Answer*:

Main memory. *Answer*:

B. Determine the following regarding function `convolve`:

(1) The total number of input data elements accessed during the computation, and the highest memory level in which this would fit (where L1 $>$ L2 $>$ Memory.)

Answer:

(2) The number of input data elements accessed for one value of outer loop index `i`, and the highest memory level in which this would fit.

Answer:

(3) The number of times each input data element is accessed for one value of outer loop index `i`.

Answer:

C. Determine the following regarding function `convolve`:

(1) The total number of weight values accessed during the computation, and the highest memory level in which this would fit.
Answer:

(2) The number of weight values accessed for one value of outer loop index `i`, and the highest memory level in which this would fit.
Answer:

(3) The number of times each weight value is accessed for one value of outer loop index `i`.
Answer:

D. Determine the single-core performance of function `convolve`:

(1) Describe the access pattern for the input data within the innermost two loops, in terms of which levels of memory the data would reside in.
Answer:

(2) What would be the resulting average number of cycles per load operation for the input data?
Answer:

(3) Describe the access pattern for the weights within the innermost two loops, in terms of which levels of memory the weights would reside in.
Answer:

(4) What would be the resulting average number of cycles per load operation for the weights?
Answer:

(5) What is the CPIS for the function?
Answer:

5B: Parallel performance (10 points)

A. Determine the multicore performance of function `convolve_par` when running with T threads for $T \leq 8$.

(1) For a given thread, describe the access pattern for the input data within the innermost two loops, in terms of which levels of memory the data would reside in.

Answer:

(2) For a given thread, what would be the resulting average number of cycles per load operation for the input data?

Answer:

(3) For all threads, describe the access pattern for the weights within the innermost two loops, in terms of which levels of memory the weights would reside in.

Answer:

(4) For a given thread, what would be the resulting average number of cycles per load operation for the weights?

Answer:

(5) What is the CPIS per thread for the function?

Answer:

(6) What is the speedup, relative to sequential execution, as a function of T?

Answer:

B. Describe how you could restructure the order in which elements are accessed within the inner loop of the code to give the program superlinear speedup. You can assume that $T \in \{1, 2, 4, 8\}$. You may wish to make use of the values `thread_count` and `thread_id` that are computed within the parallel block. You should write code for the restructured loop.

Answer:

C. What would be the resulting CPIS per thread for $T \in \{2, 4, 8\}$? Explain.

Answer:

D. What would be the speedup, relative to the sequential code for $T \in \{2, 4, 8\}$?

Answer:

Pledge

On the answer sheet, you must (electronically) sign the following pledge:

> I do hereby swear that, in generating my answers to this exam, I made use of only the resources explicitly listed in the exam document. 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.
>
> Signed