Full Name:

Andrew Id:

15-418/618 Spring 2019
Exercise 3

Assigned:	Mon., Feb. 18
Due:	Fri., Feb. 22, 11:59 pm

Overview

This exercise is designed to help you better understand the lecture material and be prepared for the style of questions you will get on the exams. The questions are designed to have simple answers. Any explanation you provide can be brief—at most 3 sentences. You should work on this on your own, since that's how things will be when you take an exam.

You will submit an electronic version of this assignment to Gradescope as a PDF file. For those of you familiar with the LATEX text formatter, you can download the template, configuation, and figure files at:

http://www.cs.cmu.edu/~418/exercises/config-ex3.tex
http://www.cs.cmu.edu/~418/exercises/ex3.tex
http://www.cs.cmu.edu/~418/exercises/figs/cubes.pdf
http://www.cs.cmu.edu/~418/exercises/figs/msi.pdf

Instructions for how to use this template are included as comments in the file. Otherwise, you can use this PDF document as your starting point. You can either: 1) electronically modify the PDF, or 2) print it out, write your answers by hand, and scan it. In any case, we expect your solution to follow the formatting of this document.

Problem 1: Problem Scaling

This problem is a three-dimensional extension of the grid problem described in the lecture on workload-driven performance evaluation:

http://www.cs.cmu.edu/~418/lectures/10_perfeval.pdf.

Consider an iterative solver that operates on a three-dimensional grid of size $N \times N \times N$. It requires N iterations to reach convergence. We refer to the parameter N as the *problem size*. Increasing N by a factor of 2 increases the number of grid elements $8\times$ and the number of iterations $2\times$.

The state of the system is represented as a three-dimensional array of values `state`. The function `update` computes a new value of the state based on the current state. For each element `state[x][y][z]`, `update` computes the new value as a function of its current value and that of its six neighbors:

Self: `state[x][y][z]`

West: `state[x-1][y][z]`

East: `state[x+1][y][z]`

North: `state[x][y-1][z]`

South: `state[x][y+1][z]`

Down: `state[x][y][z-1]`

Up: `state[x][y][z+1]`

(You don't need to know the exact function.)

The overall operation can then be described in a pseudo-C notation as:

```
// For each iteration
for (iter = 0; iter < N; iter++) {
    // Main computation
    for all x, y, z in 0..N
        nstate[x][y][z] = update(state, x, y, z);
    for all x, y, z in 0..N
        state[x][y][z] = nstate[x][y][z]
}
```

In mapping the computation onto P processors, the state array is split into P cubic blocks, each containing N^3/P array elements. Each iteration (labeled "Main computation") involves having each processor 1) communicate the boundary values with its (up to six) neighbors, and 2) computing the update function for all N^3/P of its assigned elements. The program uses M gigabytes of memory per processor and runs in total time T.

1. Consider the following two scaling possibilities, yielding a problem of size N' running on P' processors, as illustrated in Figure 1.

   (a) Double the value of N, while holding P constant: $N' = 2N$, $P' = P$

   (b) Hold N constant, while increasing P by 8×: $N' = N$, $P' = 8P$

   Fill in the table below showing how the amounts of computation and communication for a single processor, performing a single iteration would scale, relative to the original problem. (For example, 2× would indicate growth by a factor of two.)
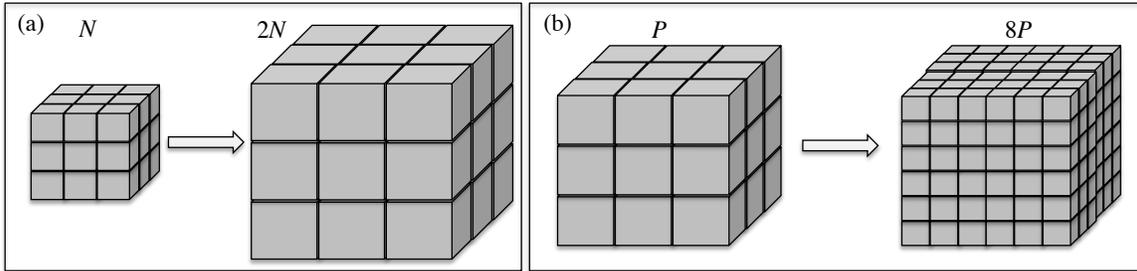
Figure 1: Scaling a problem by doubling the grid size N (a), or by increasing the number of processors $8\times$ (b).

	Computation	Communication
(a): $N' = 2N$, $P' = P$		
(b): $N' = N$, $P' = 8P$		

2. Now assume both N and P can vary. Based on these specific cases, give formulas for how the computation, communication, and arithmetic intensity would scale as functions of N and P. Again, quantify these values with respect to the computation and communication each processor would perform during a single iteration. (By way of reference, the formulas for the two-dimensional version were given in slide 7 of the lecture.)

Computation	Communication	Arithmetic Intensity

3. Suppose we have a machine with $8P$ processors, each identical to the original P processors. We consider three scaling possibilities, yielding a new problem of size N' requiring total time T' and M' gigabytes per processor:

Problem scaling: $N' = N$. Solve the same problem faster. Goal is to minimize T'

Memory scaling: $M' = M$. Make full use of the increased total memory. Goal is to maximize N'.

Time scaling: $T' = T$. Solve a bigger problem in the same amount of time. Goal is to maximize N'.

Fill in the following table with formulas indicating the problem size N', the per-processor memory requirement M', the ideal total time T', and the change in arithmetic intensity. (By way of reference, this information for the two-dimensional version was given in the slide 24 of the lecture.)

Scaling Type	N'	M'	T'	Arith. Intensity
Problem	N			
Memory		M		
Time			T	

Problem 2: Cache coherency

Your friend suggests modifying the MSI coherence protocol so that PrRd / BusRd behavior on the I-to-S transition is changed to PrRd / BusRdX, as is shown below:
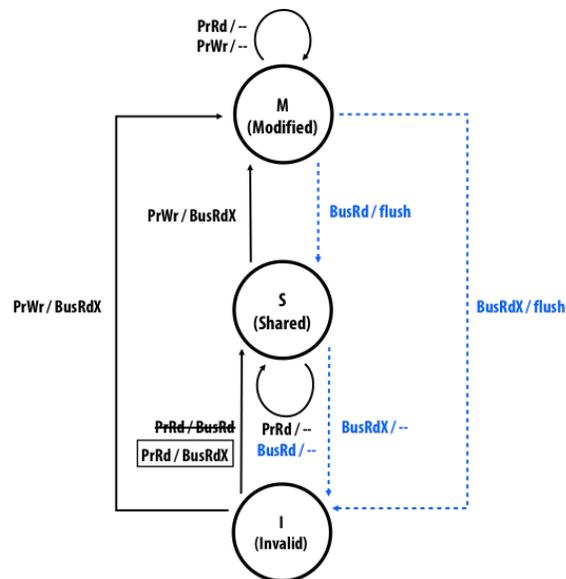
A. Is the memory system still coherent? What impact does this change have on the system?

B. You are hired by Intel to design an invalidation based cache-coherent processor with a large number of cores. The main application for this processor will be to train a chess AI by playing games against itself for experiments. It is critical that it doesn't play too many games during training to maintain the validity of the experiment, so we store how many games are played in a shared counter:

```
int num_games = 100000;
int games_played = 0;

// This code is run on each core
while (true) {
  // Atomically get value of count prior to increment, and
  // write incremented value to games_played
  int val = atomic_add(&games_played, 1);

  if (val > num_games) break;
  play_game()
}
```

Unsure on how to design the cache coherence of this processor, you check with one of the senior designers. She suggests that you should use a bus-based, snooping coherence implementation, rather than a directory-based protocol, because the broadcasting it performs would be more efficient in this case. Do you agree or disagree? Why or why not?

C. You are now tasked with finding the best performing chess AI. You plan to run the following code on the processor:

```
int num_experiments = 10000;
int best_score = 0;

#pragma omp parallel for
for (int i = 0; i < num_experiments; i++) {
  int score = perform_experiment(i);
  if (score > best_score) {
    // Atomically retrieve best score, compute max,
    // and write result to best_score
    atomic_max(&best_score, score);
  }
}
```

Trying to improve the performance of this code, you remove the `if` statement prior to the `atomic_max`, noticing that it is redundant. Suprisingly, the performance of the program is drastically reduced. Why did this happen?

D. You are hired as a TA for 15-418/618. In an attempt to speed up exam grading, you decide to multithread the grading program on a cache coherent system as shown below

```
int scores[NUM_STUDENTS];

#pragma omp parallel for schedule(dynamic)
for (int student = 0; student < NUM_STUDENTS; student++) {
    for (int problem = 0; problem < NUM_EXAM_PROBLEMS; problem++) {
        // performs a write to scores[student]
        grade_and_update(student, problem, &scores[student]);
    }
}
```

Dismayed by the speed of this program, you analyze the runtime of this program and find that there is a large amount of bus communication during the execution.

What kind of communication is happening in this program? What performance problem is this a result of?