CS-350 - Fundamentals of Computing Systems
Homework Assignment #1

Due on September 21, 2022 at 11:59 pm

Prof. Renato Mancuso

Renato Mancuso

Problem 1

A cloud gaming server (think Stadia, but that people actually play) allows players to offload the actual computation necessary to play a video-game. A single interaction between a user and the systems consists of the following steps: (1) A user submits control inputs over a network interface (NI); (2) the CPU interprets the inputs; (3) the GPU renders the next video frame; and (4) finally the NI transmits back the next rendered video frame to the players. Assume that each *interaction* is a self-contained process. Further assume that every time a process is completed by the system, another one is ready to be executed in the dispatch queue. Each process is identical in terms of resource demand and follows this precise timeline. First, the NI takes 3 ms to obtain the user inputs (during this time, the NI resource is busy); next, the CPU takes 5 ms to interpret the user inputs and launch the GPU kernel (during this time, the CPU resource is busy); next, the GPU takes 7 ms to render the current frame (during this time, the GPU resource is busy); finally the NI takes another 6 ms to transmit the rendered frame back to the user. The latter step concludes the single-interaction process.
Initially, the system is designed such that it can handle at most one process at a time. In other words, no processing over the next set of user inputs occurs until the result for the previous set of inputs has been sent to the user. Also, your server processes requests always in the same order, i.e., if a resource is available, it goes to the request that has started processing earlier. Answer the following.

a) What are the steady-state utilizations of the NI, CPU, and GPU?

b) What is the overall throughput of the system? Show your work.

c) What is the bottleneck of the system? Motivate your answer.

In your next design iteration, you enhance your system to be able to have exactly two processes active at the time (i.e., MPL = 2). This is done by starting two processes (e.g., from two different users), and as soon as either of these processes completes, a new process is started, and so on. Now, assume that the system has been running for a long time (i.e., it has reached steady state). Answer the following.

d) What are the new utilizations of the NI, CPU, and GPU?

e) What is the overall new throughput of the system?

Having experimented with MPL=2, you decide to generalize and wonder about the following:

f) What is the maximum MPL beyond which no further improvements are to be expected from the system?

g) Assuming no constraints on the MPL, what is the capacity of the system?

Problem 2

The Gidiyor-Gitti company provides zero-contact through-the-window catapult-aided food delivery. Each time a new delivery order is placed, the system needs to compute (1) wind speed at the destination address for catapult calibration and (2) traffic conditions for the driver to pick up the food to be catapulted. These pieces of information are processed in a computing cluster. On average, each order triggers computation that is completed in 5.5 minutes. 40% of that time is spent on pure wind estimation at the CPU, while 25% is spent on I/O. As you are trying to shave some time off the current order-to-catapult time, you narrow down two possible optimizations. A first option (A) consists in upgrading the CPUs. The upgrade will cost $10,000. The new processors will take 20% less time to process each instruction (e.g., instruction cycle goes down from 10 nanoseconds to 8 nanoseconds). The second option (B) is to use a new array of high-performance I/O devices that brings down the latency of each I/O operation from 12 microseconds to 10 microseconds (which costs $35,000). Answer the following questions:

a) What speedup do you expect if you pick option A?

b) What speedup do you expect if you pick option B?

c) On a per-dollar basis, which of the two improvements is better?

d) What is the speedup that you expect if you pick both improvements?

e) What is the theoretical limit for the speedup that can be achieved by making the CPUs infinitely fast?

f) What is the theoretical limit for the speedup that can be achieved by making the I/O devices infinitely fast?

Problem 3

A scientific large-scale application that performs weather pattern prediction is deployed on a 100-core machine—i.e., a machine with 100 CPUs. To save power, some CPUs can be turned off by the system. In which case, the application is restricted to run only on the CPUs that are powered on (online). In a single run, the application performs the following sequence of operations. (1) it initializes its state which takes 5 seconds and cannot be parallelized; (2) it launches one prediction heuristic per each square-foot of covered area. These are kept independent from each other and thus can be executed in parallel, with each taking 1 second to complete. The total area covered by the weather prediction is 100 square feet in size. (3) It serializes the result obtained from each local prediction into a global prediction. This step takes 15 seconds and cannot be parallelized. Answer the following and motivate your answers.

a) What is the speed-up of the entire application when it operates on a single CPU, compared to the case where it operates on all the available CPUs?

b) How many CPUs we would need to keep online to ensure that we are able to achieve a weather prediction throughput of 2 predictions per minute?

c) What is the weather prediction capacity of this systems?

d) Does Amdahl's Law provide a good approximation to compute the speed-up of the system when 30 CPUs are turned on, compared to the case when only 1 CPU is online?

e) Assume now that you have the option to turbo-boost only one CPU. Turbo-boosting one CPU makes that CPU twice as fast. When a CPU is turbo-boosted only a total of 50 CPUs can be kept online, while the other 50 must be powered off. Is turbo-boosting beneficial from a standpoint of throughput maximization?

Problem 4

Coding Assignment: In this problem you will develop code to generate random numbers that follow a particular distribution. Common programming languages provide you with a (pseudo) random number generator that yields a uniformly distributed random variable.
This question is about using such capability to develop a generator for arbitrarily distributed random variables.

a) Write a Java function with the following prototype: `double getExp(double lambda)`, that returns a random value that is distributed according to an exponential distribution with a mean of $T = \frac{1}{\lambda}$ (*Hint: An approach for doing this is provided in the lecture notes in Chapter 6, see Box 6.2.1 and Box 6.2.2*). The value of λ will be passed as a parameter to your code. The function should be implemented in its own java class named "Exp" in a file named `Exp.java`. **Please be mindful of the capitalization.**

   Apart from implementing the function itself, the class should also include a `main(...)` function. The `main(...)` function should accept 2 parameters from the calling environment: (1) a double holding the value of λ, and (2) an integer N for number of random samples to generate and print.

   It is responsibility of the `main(...)` function to internally invoke the implemented `getExp(...)` function exactly N times, and to print its result. At each invocation, nothing else should be printed in the console, except the value returned by `getExp(...)` in decimal format.

   If you are not familiar with how to pass parameters (a.k.a. command-line arguments) from the calling environment to a Java program, take a look at this: `https://www.journaldev.com/12552/public-static-void-main-string-args-java-main-method`.

b) Write a Java class called Event (in its own `Event.java` file) that represents a generic timed event with at least three fields: (1) a generic type, e.g. A or B; and (2) the timestamp at which the event occurs; and a unique ID of the form [type][number], for instance A0, A1, ... for A-type events. Next, implement a sorted timeline of events. The class should be called Timeline (in its own `Timeline.java` file) and should contain a `void addToTimeline(Event evtToAdd)` function that accepts two pieces of information that adds the event in input into the timeline in a chronologically sorted manner. Next, create another function, say `Event popNext()`, that removes the oldest event from the timeline and returns the removed event to the caller. Finally, implement a `LoadGenerator` class in the file `LoadGenerator.java`. The load generator must expose one main function called `void releaseRequest(Event evt)`. When called, this function looks at the timestamp of the event passed as a parameter and generates a new event to be inserted in the Timeline. The event type is fixed for each generator and passed in input (via the constructor). The time between the current timestamp and the new timestamp is exponentially distributed with some parameter λ that is also passed in input (via the constructor).

   Please be mindful of the capitalization on the name of the Java files you will submit.

   Apart from implementing the functions described above, the LoadGenerator class should also include a `main(...)` function. The `main(...)` function should accept 3 parameters from the calling environment (in the following order):

   (a) λ_A average arrival rate of events of type A in events/ms;

   (b) λ_B average arrival rate of events of type B in events/ms;

   (c) T length of simulation time in milliseconds.

   It is responsibility of the `main(...)` function to perform the following: (1) instantiate a single Timeline object; (2) generate and add to the Timeline exactly one type-A event with timestamp 0; (3) same as 2 for a type-B event; (4) instantiate TWO LoadGenerator objects, to generate type-A and type-B events, respectively, and pass the corresponding parameters λ_A and λ_B; and (5) scan through the Timeline,

Problem 4 continued on next page...

removing (`popEvent`) the next element in chronological order and invoking the `releaseRequest(...)` function on the corresponding LoadGenerator. Doing this will generate the next event of the given type and the loop will continue. (6) Terminate the loop when the timestamp on the current event is greater than the provided simulation length T.

Whenever the `releaseRequest(Event evt)` is invoked, also print in output the information about the event being processed (not the new one being created!) following the format [ID]: ¡timestamp¿.

For instance, the first type-A event should be named A0 and should have arrival time t_{A0}, i.e. timestamp, equal to 0. The next event A1 will have timestamp $t_{A1} = t_{A0} + r_1$, where r_1 is the inter-arrival between A0 and A1 and it is randomly generated using the `getExp(...)` function developed in Part 1. For event A2 will hold that $t_{A2} = t_{A1} + r_2$ and so on.

Your overall printout should look something like this:

```
A0: 0
B0: 0
A1: <timestamp t_A1>
A2: <timestamp t_A2>
A3: <timestamp t_A3>
B1: <timestamp t_B1>
A4: <timestamp t_A4>
A5: <timestamp t_A5>
B2: <timestamp t_B2>
...
```

where `<timestamp t_Xi>` is the simulated time in milliseconds at which the i-th event of type X occurred printed in decimal format and without the surrounding "<" and ">" characters. Suggestion: No need to generate all the events first and then print. You can also generate and print as you go. This is also called the **trace** of the simulation.

Be particularly careful to follow the format described above. CodeBuddy will be very sensitive to output formatting issues.

Submission Instructions: in order to submit this homework, please follow the instructions below for exercises and code.
The solutions for Problem 1-3 should be provided in PDF format, placed inside a single PDF file named `hw1`.pdf and submitted via Gradescope. Follow the instructions on the class syllabus if you have not received an invitation to join the Gradescope page for this class. You can perform a partial submission before the deadline, and a second late submission before the late submission deadline.

The solution for Problem 4 should be provided in the form of Java source code. To submit your code, place all the `.java` files inside a compressed folder named `hw1`.zip. Make sure they compile and run correctly according to the provided instructions. The first round of grading will be done by running your code. Use CodeBuddy to submit the entire `hw1`.zip archive at `https://cs-people.bu.edu/rmancuso/courses/cs350-fa22/scores.php?hw=hw1`. You can submit your homework multiple times until the deadline. Only your most recently updated version will be graded. You will be given instructions on Piazza on how to interpret the feedback on the correctness of your code before the deadline.