15-418/618, Fall 2017

Take-Home Quiz 1

Due: Noon, Friday, September 22nd

Instructions:

- The idea behind these take-home quizzes is to give you some practice answering questions in the style of the exams. Since the exams are CLOSED BOOK, CLOSED NOTES, you should also practice doing this quizzes the same way.
- Here is how we suggest that you do this quiz:
  1. Start by reviewing the lecture notes. (This quiz will focus on Lectures 1 through 8.)
  2. Then close up your books and notes and take the quiz. We suggest that you allocate roughly 10 minutes per question.
  3. Review the lecture notes afterward to see how you did.
- After you finish writing up your quiz, be sure to **submit it via Gradescope** so that you can receive credit. Remember that these quizzes count as part of your participation grade (not your exam grade).

SIMD Vectorization

Problem 1:

Recall that even in the presence of conditional branches resulting from `if-then-else` statements inside inner loops, SIMD vector instructions can still potentially achieve parallel speedups by masking off the ALUs that should not be executing. Assuming that P is the probability the conditional test is *true* (and therefore the `then` part should be executed), and that L_t and L_e are the lengths of the `then` and `else` parts, respectively, describe a scenario where we would expect to see the following ALU utilizations. (Be sure to explain your answer in detail; answers without explanations will not receive much credit on the exam!)

A. Utilization = 1/2

B. Utilization = 1/8

Hardware Multithreading

Problem 2:

Recall that hardware-supported multi-threading is one technique that can be used to tolerate the latencies of expensive cache misses. To take advantage of hardware-supported multi-threading, the programmer must specify additional concurrent threads in the application (beyond the number that will actually run simultaneously). Why is this the case? (Explain your answer in terms of how hardware-supported multi-threading works, and why it has this requirement.)

Comparing Programming Models: Shared Address Space vs. Message Passing

Problem 3:

A. Recall that we looked at examples of the Grid Solver kernel that were written using both the *Shared Address Space* and *Message Passing* programming models. Describe two major differences between how the source code is written under these two programming models.

B. Recall that we said in class that it is relatively common that you will need to write parallel software that combines both the *Shared Address Space* and *Message Passing* programming models within the same application. Explain why this makes sense.

Problem 4:

In this problem, we will compare static versus dynamic scheduling.

A. Under what circumstances would you expect *static* scheduling to significantly outperform dynamic scheduling? Please explain your answer (and be as specific as possible).

B. Under what circumstances would you expect *dynamic* scheduling to significantly outperform static scheduling? Please explain your answer (and be as specific as possible).

C. With dynamic schedule, what are the relative advantages and disadvantages of increasing the grain size?

Problem 5:

A. Describe a specific example of how a programmer can reduce the amount of *inherent communication* to be performed by a parallel program.

B. Describe two examples of how *artifactual communication* can arise in a parallel program (thereby potentially hurting performance).