

Assignment 3

ECSE 420: Parallel Computing

Due: April 12, 2024

Submission instructions: Students are to work in groups of two on the assignment. Each group should submit a pdf file that contains the following information: students' names, students' IDs, instructions on how to run each file and the associated question solved. Students are also expected to submit a .zip file containing their source code. This code must compile and run without error. The code must be well formatted, commented, and follow the [Google Java Style Guide](#). For more information, see the ECSE420AssignmentSubmissionInstructions.pdf in the Assignment section on [myCourses](#).

Questions

1. (18 marks) Chapter 7, Memory access, Anderson Lock

The following benchmark is designed to measure the average time for reading array A containing L elements from memory, when the array elements are s words apart:

```
for stride  $s$  from 4 Bytes (1 word) to  $L/2$  by 2x ( $s = 1, 2, 4, \dots, L/2$ )
time the following loop (repeat many times and average)
  for  $i$  from 0 to  $L-1$ 
    load  $A[i \cdot s]$  from memory (4 Bytes)
```

So, the benchmark reads L words from memory into cache, where L is a constant; however, these L words are not consecutive memory locations (they are " s " words apart).

The results obtained from the benchmark are shown in Fig.1:

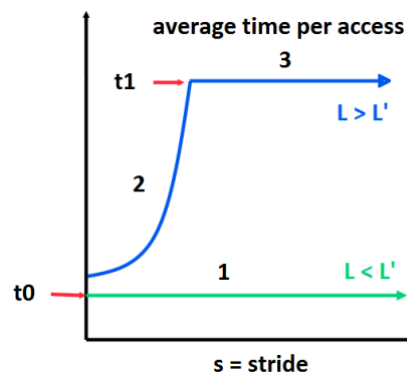


Figure1. Average time for reading each array element.

Assuming only one level of cache exists, the cache line size is 4 words, and a single processor is running the benchmark, answer the following questions:

- 1.1 In Fig.1, when the total number of elements L of the array is smaller than L' , the average time per access remains constant. What does the value of L' represent? What does time t_0 show?
- 1.2 When L is larger than L' , what does time t_1 indicate in Fig.1?
- 1.3 For each part of the graph (i.e., parts 1, 2 and 3), justify its behaviour.
- 1.4 We know a phenomenon called **false sharing** could cause unnecessary invalidations in ALock (Anderson Lock), and one way to avoid false sharing is to **pad** array elements so that



distinct elements are mapped to distinct cache lines. Considering the results from Fig.1, how could the padding technique used in ALock degrade the overall performance of the lock?

2. **(18 marks)** Chapter 9, examining the fine-grained synchronization algorithm.
 - 2.1. Provide the code for the *contains()* method missing from the fine-grained synchronization (hand-over-hand locking) algorithm described in chapter 9.
 - 2.2. Write a test to verify the *contains()* method and explain why your implementation is correct.
3. **(12 marks)** Chapter 10, designing a bounded lock-based queue
 - 3.1. Design a bounded lock-based queue implementation using an array instead of a linked list. Allow parallelism by using two separate locks for the head and the tail.
 - 3.2. Try to transform your algorithm to be lock-free. Where do you run into difficulty?
4. **(32 marks)** Chapter 16, matrix vector multiplication
 - 4.1. Implement a sequential matrix vector multiplication.
 - 4.2. Give an efficient and highly parallel multithreaded algorithm for multiplying an $n \times n$ matrix A by a length- n vector x that achieves *work* $\Theta(n^2)$ and *critical path* $\Theta(\log n)$.
[Hint: If you use a cached thread pool with the Future interface, to achieve the $\Theta(\log n)$ critical path, you can follow the algorithm for matrix multiplication in Chapter 16. Otherwise, depending on how you divide-up the problem, you could obtain a critical path different than $\Theta(\log n)$. This is acceptable, just be sure to show your work and justify in your report the critical path of your implementation.]
 - 4.3. Write a test program that measures the execution time for multiplying a 4,000 by 4,000 matrix with a corresponding 4000 wide vector using the parallel method and sequential method, respectively. Discuss the execution time of the two methods and compute the speedup. Be sure to indicate how many threads are being used.
 - 4.4. Analyze and discuss the *work* and *critical-path* length of your implementation and give the *parallelism*.

Total: 80 marks

