

□ Information

Written Exam in IN3030/IN4330

2024 Spring Semester

Duration: 4 hours

Date: May 27th, 2024

General information:

- IN4330 students are given the same questions as IN3030 students but are expected to give better and more in-depth answers.
- Your answer should reflect your own independent work and should be a result of your own learning and work effort.
- Concerning Answers that involve Java Programming: the prime concern is what the program does; it is a secondary concern that the program is entirely syntactically correct, so good pseudo-code where boring or less relevant Java details are missing is acceptable: Spend your time on getting the algorithms right rather than the syntax. Do not worry about importing libraries.

Support material:

All written materials. All lecture materials from 2024 and 2022 are available as pdfs in Inspira at the bottom of the page.

The answer must be written with an academic standard.

You can answer in English, Norwegian (bokmål or nynorsk), Danish or Swedish. Or a mixture of these languages.

1 Paralellization of Odd Even Sort

Odd-Even sort

Given the sorting algorithm below, describe how you would parallelize it, so that you achieve a speedup for large arrays (the example below is merely for an array of size four).

IMPORTANT: your parallel algorithm MUST be LOYAL to the original algorithm. As an example of a NON-LOYAL solution is one where each of k threads does an Odd-Even sort of $1/k$ of the array followed by a merge sort of the k sorted parts. This is not acceptable as it substantially modifies the original algorithm: if we use the same technique for the sequential version, we will vastly improve the original sequential algorithm. Thus you are to parallelize the algorithm merely by reordering the original sorting steps and then running some of them in parallel.

```
// Java Program to implement
// Odd-Even / Brick Sort
import java.io.*;

class OddEvenC {
    public static void oddEvenSort(int arr[], int n)
    {
        boolean isSorted = false; // Initially array is unsorted

        while (!isSorted) {
            isSorted = true;
            int temp = 0;

            // Perform Bubble sort on odd indexed element
            for (int i = 1; i <= n - 2; i = i + 2) {
                if (arr[i] > arr[i + 1]) {
                    temp = arr[i];
                    arr[i] = arr[i + 1];
                    arr[i + 1] = temp;
                    isSorted = false;
                }
            }

            // Perform Bubble sort on even indexed element
            for (int i = 0; i <= n - 2; i = i + 2) {
                if (arr[i] > arr[i + 1]) {
                    temp = arr[i];
                    arr[i] = arr[i + 1];
                    arr[i + 1] = temp;
                    isSorted = false;
                }
            }
        }

        return;
    }
}
```

```
public static void main(String[] args)
{
    int arr[] = { 34, 2, 10, -9 };
    int n = arr.length;

    oddEvenSort(arr, n);
    for (int i = 0; i < n; i++)
        System.out.print(arr[i] + " ");

    System.out.println(" ");
}
```

Write your suggestion here:

Maximum marks: 20

2 Implementation of parallel Odd-Even Sort

Implement your suggested parallel version of Odd-Even Sort from the previous question. Make sure that your suggestion is LOYAL to the original algorithm.

Write the code for your parallel version of Odd-Even sort

1	
---	--

Maximum marks: 12

3 Speed of Light in Vacuum

What is the speed of light in vacuum? Choose the precise answer to get two points (an approximate answer that is within 1 percent of the correct value will be given one point).

Choose ONE alternative:

- ☐ 300 000 m/s
- ☐ 299 792 458 m/s
- ☐ 299 458 792 m/s
- ☐ 299 458 792 km/s
- ☐ 300 000 000 km/s
- ☐ 300 000 km/s

Maximum marks: 2

4 Effects of the speed of light

The slow speed of light:

Select one alternative:

- ☐ Moore's law has for many years meant that CPU speeds have been exponentially increasing thus making the latency caused by the slow speed of light insignificant.
- ☐ Is not important because the latency that it causes is significant only for long distance data transmission, say between cities.
- ☐ Means that when loading a value from main memory, a CPU core may have to wait many cycles because of the latency caused by the slow speed of light.
- ☐ Can be significantly increased by using fiber optic cables.
- ☐ The latency it causes can be overcome by increasing the clock cycle on CPUs. For example, doubling the CPU speed will reduce the latency by a factor of two.

Maximum marks: 2

5 Hiding Latency

The latency caused by the slow speed of light can be partially hidden from the programmer by:

Select one alternative:

- ☐ Taking advantage of caching.
- ☐ Increasing the voltage of each core.
- ☐ Using special high speed chips for the main memory and placing them separately far away from the processor chip as to avoid overheating the processor chip.
- ☐ Replacing copper connections by WiFi based connections.

Maximum marks: 2

6 One foot - how long is it?

How long does it take light in vacuum to travel 1 foot? Assume that one foot is about 30 cm.

Select one alternative:

- ☐ About 3 1/2 nanoseconds.
- ☐ About 1 nanosecond.
- ☐ About 1/2 nanosecond.
- ☐ About 1/3 microsecond.
- ☐ About 333 nanoseconds.
- ☐ About 1/3 nanoseconds.
- ☐ About 1 microsecond.

Maximum marks: 2

7 PairUp Synchronization

PairUpC

This question is about a new synchronization mechanism that you are to implement.

Write a Java class, **PairUpC**, that has a single method:

`pairUp()`

that does the following:

if no one is waiting, the calling thread waits.

if there already is a thread waiting, the calling thread unblocks the waiting thread and then both threads continue.

Your solution **MUST ONLY** use Java semaphores for synchronization.

You may use the following semaphore methods: `acquire()`, `release()` **ONLY**.

Write the class below:

Write your PairUpC Java Class here:

1	
---	--

Maximum marks: 18

8 Test Program for PairUpC

Write a Java test program that demonstrates the representative unit testing for the implemented your **PairUpC** class. Ensure that the output of the test explains the steps of the execution.

Explain the test that you chose and why it gives good test coverage.

Submit the program, its output and any comments that you might have.

Write your Java test program here:

1	
---	--

Maximum marks: 8

9 Amdahl's Law

If a program has an initial sequential part taking 20% of the execution time, what is the maximum achievable speedup according to Amdahl's Law regardless of how many cores that are available?

Select one alternative:

- ☐ 2.5
- ☐ 8/10
- ☐ 4
- ☐ 0.2
- ☐ 5
- ☐ 3
- ☐ 8

Maximum marks: 2

10 Max Speedup

If all parts of a program are easily parallelizable, what is the maximum achievable speedup on an 4 core machine? Assume that the effective number of cores is 4, i.e., including any performance boosting techniques such as hyperthreading.

Select one alternative:

- ☐ 2
- ☐ 16
- ☐ 4
- ☐ 8

Maximum marks: 2

11 Max Speedup with Four Cores

On a machine that *effectively* has four cores (assume no hyperthreading), is it possible to achieve a speedup of MORE than four when parallelizing a fully optimized sequential program?

Select one alternative

- ☐ If a speedup of MORE than four is possible, then it is also possible to rewrite the parallel version into a sequential program that is faster than the original sequential program thus breaking the assumption that the original sequential program was fully optimized.
- ☐ A speedup of more than four can be achieved by having, for example, eight threads, which will allow a speedup of up to eight.
- ☐ Yes, by adding more main memory to the system.
- ☐ No, with four cores speedups cannot exceed two because of Amdahl's law.

Maximum marks: 4

12 Cache friendliness of QuickSort

Consider the following sequential version of Quicksort programmed in Java. Comment on its cache-friendliness:

```
public void quickSort(int arr[], int begin, int end) {
    if (begin < end) {
        int partitionIndex = partition(arr, begin, end);
        quickSort(arr, begin, partitionIndex-1);
        quickSort(arr, partitionIndex+1, end);
    }
}

private int partition(int arr[], int begin, int end) {
    int pivot = arr[end];
    int i = (begin-1);
    for (int j = begin; j < end; j++) {
        if (arr[j] <= pivot) {
            i++;
            int swapTemp = arr[i];
            arr[i] = arr[j];
            arr[j] = swapTemp;
        }
    }
    int swapTemp = arr[i+1];
    arr[i+1] = arr[end];
    arr[end] = swapTemp;
    return i+1;
}
```

Fill in your comments here

Maximum marks: 14

13 Java Threads Termination

Java Threads Termination

In a Java program that contains more than one thread, when does the program normally (ignoring exceptions and other error conditions) finish execution?

Select an alternative:

- ☐ When all the threads created in the main() method have terminated AND any thread created by any other thread in the program has terminated AND the main() method also has terminated.
- ☐ When all the threads created in the main() method have terminated AND the main() method also has terminated.
- ☐ When the main() method terminates.
- ☐ When all the threads created in the main() method have terminated.
- ☐ When all the threads created by the program have terminated AND the main() method has called join on ALL the threads.

Maximum marks: 3

14 Java Threads Execution on Multicore

Java Threads Execution on Multicore

Which of the following statements are true for Java Threads that are created in the same program? (More than one statement might be true, so select all alternatives that you think are true statements. An correct answer gives one point; an incorrect answer -1 point; a blank answer 0 point.)

Select one or more alternatives:

- ☐ Java threads can execute in parallel independently of one another on multi-core machines.
- ☐ Java threads appear to execute in parallel independently of one another even on single-core machines because the Java Virtual Machine runs each thread for a short time then switches to another thread, if there is one that wants to run.
- ☐ If there are more threads in a program than available cores, the program cannot be run.
- ☐ All Java threads execute at the same speed because each thread is assigned to its own core .
- ☐ Each core runs one Java thread. When that thread terminates, another Java thread is selected and run until it terminates - and so on.
- ☐ Java threads execute in parallel: Every core runs a thread until it either terminates or has used up a time slice, whereafter another thread (possibly the same thread) is run on the core - and so on. If no threads are available to run, the core waits for a runnable thread to show up.
- ☐ Java threads always execute one at a time. All the others must wait.

Maximum marks: 4

15 Safety of Java

Safety of Java

When using multiple Java threads, it is, in general, safe to do the following (choose the answers that are true; an extra point when you have chosen exactly the set of correct answers - note that there can be between 0 and 4 answers that are true):

Select one or more alternatives:

- ☐ It is OK for two threads to both read and write a shared integer variable as long as they make sure to synchronize their access to the variable.
- ☐ When multiple threads update a shared integer then it is always safe, if the integer is an instance of the Java class AtomicInteger.
- ☐ Two Java threads can safely write to adjacent array elements in an integer array.
- ☐ Two threads can READ from the same memory location, e.g, the same element in an array without synchronization problems as long as no other thread writes to the memory location.

Maximum marks: 5