COMP262-01 LAB10, CH17-18

Parallel Programming by simulating the use of a Multicore Processor (for ex. Intel® Core™ i5-4670R Processor)

DESCRIPTION:
We will 'simulate' a multicore system operation by modifying a program with implicit parallelism.
Assume that a Computer system has a Multicore Processor with four cores and a shared memory.
This assignment consists of 'adapting' the provided **'SUMST.java'** program, which currently runs as a **SISD**, that is to say, as one continuous SINGLE 'thread' of execution on one single processor/core.
The provided shell **(SUMST.java)**, is already coded with the structure that is needed for the implementation of MULTIPLE 'threads' of execution simulationg multiple cores.
You need to scale the code to the requested number of threads/cores and adapt the code so as to run as a **SIMD**, with multiple 'threads/cores' executing in parallel.

REQUIREMENTS:
The DATA STRUCTURE used in this program consists of an array of integer values, which are totally independent of each other. The current code populates the entries in the array (the integer values), in sequence, one value at the time.
You need to expand (scale) the structure of the program to execute 4 'threads', each running the same code, but operating on different segments of the data structure. We are implementing the 'divide and conquer' approach, so that the whole data structure is partitioned/segmented in equal parts and operated on by each of the threads, in parallel.
The array consists of 40 entries, which should be apportioned amongst the 4 'threads', with each having a subset of 10 entries each, to work on.

TO DO: Modify the program to run 4 'threads' each processing a subset of 10 entries each.

When completed please submit a **modified/new SUMMT.java** source code via the file submission mechanism.

We are providing an example of the desired output, (**SUMMT-output.pdf)**, that your completed program must emulate. PLEASE be aware that the results of each run will be different as far as the order of the output lines, since it all depends on the OS thread dispatching algorithm/code!