PARALLEL SYSTEMS EXERCISE-2A

LABORATORY 2024-25 (15%)

Write a program in **OpenMP** that sorts a sequence of N integers *A[0…N-1] in parallel,* according to the recursive sorting algorithm **'multisort'** listed on slides **#49-50** of the OpenMP.ppt file .

[Consult among others and rely for your implementation on the code of the file ***OpenMP_msort.c*** *(indicative implementation of the mergesort sorting algorithm in OpenMP) posted on Eclass.]*

Measure the performance of your implementation for large values of 'N' and for different numbers of threads, and compare them against each other. acceleration achieved in each case.

Deliverables: code, annotation/documentation, sample runs/results.

PARALLEL SYSTEMS EXERCISE-2B

LABORATORY 2024-25 (25%)

1. Write a program in **CUDA** that (given as input 'A', 'N'), finds for a two-dimensional array of integers *A(NxN),* the average value of its elements (m), and its maximum *(amax)* element. Then check if its maximum element is greater than *N-times* its average value *(amax > N*m),* and accordingly:

i. If this is true, create a new matrix *B(NxN)* (which it prints to the screen at the end) where:

$$Bij = (m - Aij) / amax$$

For the above table *B,* it is also requested to calculate and print on the screen, the minimum element value of *(amin).*

ii. If this is not the case, create a new matrix *C(ÿxN)* (which it will print on the screen at the end) where:

$$Cij = (Aij + Ai(j+1) + Ai(j-1)) / 3$$

Also consider that if *j+1=N* then *Ai(j+1)=Ai0,* while if j-1=-1 then *Ai(j-1)=Ai(ÿ-1).*

2. Write a program in **CUDA** that, given a two-dimensional array, integers *A(NxN)* calculates the covariance log of *A* as follows:

i. For each column of table A, it calculates the average of the column's elements.

ii. From each element of a column of table *A* subtracts the average of the corresponding column calculated in the previous step.

iii. Multiplies the matrix *A* of the previous step by its inverse. Since the resulting matrix is symmetric, it is sufficient to calculate either the upper or lower triangular part of the result.

Other Requirements: Measure the performance of your implementation for different values of 'N', different total number of threads (e.g. N, N^2 etc.), and different number of blocks and threads/block. Also consider the possibility of using shared memory in each subquery. Finally, where critical region protection / reduction is required, consider the possibility of using both *atomic commands* and the possibility of applying *a binary tree algorithm.*

Deliverables: Code, annotation/documentation, sample runs/results.

Submission Deadline: **12/1/2025** (pp. you are invited to work *on only one of your choice – or if you wish, both – of the two parts of the Exercise – see the relevant Announcement posted on Eclass for more details).* The Exercises you will submit will be examined orally at the end of the semester.