Universidad Autónoma de Madrid

Escuela Politécnica Superior

UNIVERSIDAD AUTÓNOMA DE MADRID

MASTER'S DEGREE IN RESEARCH AND INNOVATION IN COMPUTATIONAL INTELLIGENCE AND INTERACTIVE SYSTEMS (I^2-ICSI)
Numerical and Data-Intensive Computing (Course 2022/23)

MÁSTER UNIVERSITARIO EN INGENIERÍA INFORMÁTICA
Computación a Gran Escala (Curso 2022/23)

Laboratory 2: OPENMP (Parallel loops)

Deadline: ***November 3***

Compulsory assignment (5 points)

Proceed carefully through the following steps, completing the lab report (homework) as requested. The lab report must be a full technical document consisting of a front page, index and sections, which is to be delivered as a PDF file through the Moodle portal by the given deadline date.

1. Download associated material (openmp2.tar.gz) from Moodle's course page into personal working directory.
2. Uncompress and untar associated material:

   ```
   gunzip openmp2.tar.gz
   tar -xvf openmp2.tar
   ```

3. Go to directory openmp2/task1
4. Edit and understand example "task1.c".
5. Compile the program and execute it, writing down the wall time. The latter is the minimum sequential time (Ts) of the loop for static workload.
6. Set the number of OpenMP threads to 2 and insert a parallel loop directive with static scheduling right above the sequential loop:

   ```
   #pragma omp parallel for schedule( static )
   for (i=0; i<ITERATIONS; i++) Static_Workload( i );
   ```

7. Compile the program and execute it, making sure both logical processors belong to different cores. Write down the wall time, which is the parallel time (Tp) of the algorithm for both static workload and static scheduling. Compute speedup and efficiency for two cores.
8. Try different chunk sizes (i.e.: 1, 2, 4, 8, 16, …), writing down the corresponding parallel times. Determine the fastest alternative for static scheduling (with or without chunk size). Example for chunk size equal to 1 (horizontal mapping):

   ```
   #pragma omp parallel for schedule( static, 1 )
   for (i=0; i<ITERATIONS; i++) Static_Workload( i );
   ```

9. Change static scheduling for dynamic scheduling. Compile the program and execute it, making sure both logical processors belong to different cores. Write down the wall time, which is the parallel time (Tp) of the algorithm for both static workload and dynamic scheduling. Compute speedup and efficiency for two cores. Example:

```
#pragma omp parallel for schedule( dynamic )
for (i=0; i<ITERATIONS; i++) Static_Workload( i );
```

10. Try different chunk sizes (i.e.: 1, 2, 4, 8, 16, …), writing down the corresponding parallel times. Determine the fastest alternative for dynamic scheduling (with or without chunk size) and its speedup and efficiency for two cores. Example for chunk size equal to 1:

```
#pragma omp parallel for schedule( dynamic, 1 )
for (i=0; i<ITERATIONS; i++) Static_Workload( i );
```

11. Write down the results and conclusions for static workload to lab report (homework).
12. Set the number of OpenMP threads to 1, comment the parallel loop directive and change the static workload for dynamic workload:

```
// #pragma omp parallel for schedule( dynamic, 1 )
for (i=0; i<ITERATIONS; i++) Dynamic_Workload( i );
```

13. Compile the program and execute it, writing down the wall time. The latter is the minimum sequential time (Ts) of the loop for dynamic workload.
14. Set the number of OpenMP threads to 2, uncomment the parallel loop directive and set static scheduling:

```
#pragma omp parallel for schedule( static )
for (i=0; i<ITERATIONS; i++) Dynamic_Workload( i );
```

15. Compile the program and execute it, making sure both logical processors belong to different cores. Write down the wall time, which is the parallel time (Tp) of the algorithm for both dynamic workload and static scheduling. Compute speedup and efficiency for two cores.
16. Try different chunk sizes (i.e.: 1, 2, 4, 8, 16, …), writing down the corresponding parallel times. Determine the fastest alternative for static scheduling (with or without chunk size) and its speedup and efficiency for two cores. Example for chunk size equal to 1:

```
#pragma omp parallel for schedule( static, 1 )
for (i=0; i<ITERATIONS; i++) Dynamic_Workload( i );
```

17. Change static scheduling for dynamic scheduling. Compile the program and execute it, making sure both logical processors belong to different cores. Write down the wall time, which is the parallel time (Tp) of the algorithm for both dynamic workload and dynamic scheduling. Compute speedup and efficiency for two cores. Example:

```
#pragma omp parallel for schedule( dynamic )
for (i=0; i<ITERATIONS; i++) Dynamic_Workload( i );
```

18. Try different chunk sizes (i.e.: 1, 2, 4, 8, 16, …), writing down the corresponding parallel times. Determine the fastest alternative for dynamic scheduling (with or without chunk size) and its speedup and efficiency for two cores. Example for chunk size equal to 1:

```
#pragma omp parallel for schedule( dynamic, 1 )
for (i=0; i<ITERATIONS; i++) Dynamic_Workload( i );
```

19. Change dynamic scheduling for guided scheduling. Compile the program and execute it, making sure both logical processors belong to different cores. Write down the wall time, which is the parallel time (Tp) of the algorithm for both dynamic workload and guided scheduling. Compute speedup and efficiency for two cores. Example:

```
#pragma omp parallel for schedule( guided )
for (i=0; i<ITERATIONS; i++) Dynamic_Workload( i );
```

20. Try different chunk sizes (i.e.: 1, 2, 4, 8, 16, …), writing down the corresponding parallel times. Determine the fastest alternative for guided scheduling (with or without chunk size) and its speedup and efficiency for two cores. Example for chunk size equal to 1:

```
#pragma omp parallel for schedule( guided, 1 )
for (i=0; i<ITERATIONS; i++) Dynamic_Workload( i );
```

21. Write down the results and conclusions for dynamic workload to lab report (homework).

22. Go to directory openmp2/task2
23. Edit and understand example "task2.c".
24. Compile the program and execute it, writing down the wall time. The latter is the minimum sequential time (Ts) for naïve matrix multiplication.
25. Set the number of OpenMP threads to 2. Parallelize function `Mul1Par`, determining if the maximum speedup is achieved after parallelizing the outer loop (i.e., coarse granularity), the middle loop (i.e., medium granularity) or the inner loop (i.e., fine granularity). Determine the best scheduling policy. Compute the best speedup and efficiency for two cores.
26. Write down the results and conclusions for naïve matrix multiplication to lab report, **including the source code** of the parallelized function (homework).

Optional assignment A (1 point)

27. Change function `Mul1Par` for the optimized version (function `Mul2`) developed in the first laboratory (profilers – task2). Set the number of OpenMP threads to 1, compile the program and execute it, writing down the wall time. The latter is the minimum sequential time (Ts) for optimized matrix multiplication.
28. Set the number of OpenMP threads to 2. Parallelize function `Mul2` optimally (generating function `Mul2Par`). Compute the best speedup and efficiency for two cores.
29. Write down the results and conclusions for the optimized matrix multiplication to lab report, **including the source code** of the parallelized function (homework).

Optional assignment B (1.5 points)

30. Go to directory openmp2/task3 (Fractal generation).
31. Compile the program and execute it, writing down the wall time. The latter is the minimum sequential time (Ts) for the original application.
32. Set the number of OpenMP threads to 2. Parallelize the given application optimally. Compute the best speedup and efficiency for two cores.
33. Write down the results and conclusions to lab report, **including the source code** of the function/s that has/have been parallelized (homework).

Optional assignment C (2.5 points)

34. Go to directory openmp2/task4 (image processing application profiled in laboratory 1 at profilers – task3).

35. Compile the program and execute it, writing down the wall time. The latter is the minimum sequential time (Ts) for the original application.
36. Taking into account the bottleneck functions identified in the first laboratory (profilers – task3), parallelize the given application optimally. Compute the best speedup and efficiency for two cores.
37. Write down the results and conclusions to lab report, **including the source code** of the function/s that has/have been parallelized (homework).