UNIVERSIDAD AUTÓNOMA DE MADRID

MÁSTER UNIVERSITARIO EN INGENIERÍA INFORMÁTICA
Procesamiento y Gestión de Datos Masivos (Curso 2024/25)

Laboratory 3: LOOP PARALLELIZATION WITH OPENMP

Deadline: ***November 22***

Compulsory assignment (7 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 (openmp3.tar.gz) from Moodle's course page into personal working directory.
2. Uncompress and untar associated material:

```
gunzip openmp3.tar.gz
tar -xvf openmp3.tar
```

3. Go to directory openmp3/task1
4. Edit and understand example "task1.c". The sequential loop (function `Loop`) is the one parallelized in unit 2.4.6.
5. Using OpenMP, implement the parallel version of the above sequential loop (function `LoopPar`) based on the pseudocode derived and shown in unit 2.4.6. Be aware that OpenMP parallel loops have implicit barrier synchronization at the end, thus making explicit barriers redundant. Also notice that outer parallel loops in nested loops must explicitly declare as private the loop variables of their corresponding inner loops.
6. Compile the program and execute it, making sure that both loops yield the same result (program must print out "Equal 1").
7. Write down the sequential and parallel wall times for two and four threads, respectively. Compute the associated speedups and efficiencies. Check if the parallel program yields efficiencies above 100%. In that case, the parallel program achieves "superlinear speedup", which is a beneficial side-effect of the net increase in cache resources with respect to the sequential execution due to the use of several cores with their associated local caches.
8. Write down the results and conclusions to lab report, **including the source code** of function `LoopPar` (homework).

9. Go to directory openmp3/task2

10. Edit and understand example "task2.c". The sequential loop (function `Loop`) is the one listed below:

```
for (i=0; i<N; i++)
 for (j=0; j<N; j++)
 {
    B[i+1][j-1] = A[i+1][j] - A[i-1][j];  // S1
    D[i][j] = D[i-1][j-1] * 3;            // S2
    C[i+1][j] = D[i][j] * B[i-1][j] * 2;  // S3
    C[i-1][j+1] = D[i+2][j+1] * 3;        // S4
 }
```

11. Implement the parallel version of the above sequential loop using OpenMP (function `LoopPar`).
12. Compile the program and execute it, making sure that both loops yield the same result (program must print out "Equal 1").
13. Write down the sequential and parallel wall times for two and four threads, respectively. Compute the associated speedups and efficiencies. Check if the parallel program yields superlinear speedup.
14. Write down the results and conclusions to lab report, **including the dependency graph and the source code** of function `LoopPar` (homework).

Optional assignment (3 points)

15. Go to directory openmp3/task3
16. Edit and understand example "task3.c". The sequential loop (function `Loop`) is the one listed below:

```
for (i=0; i<N; i++)
 for (j=0; j<N; j++)
 {
    C[i+1][j+1] = D[i-3][j+1] * 3;        // S1
    D[i-1][j] = D[i][j] * B[i][j-1] - 2;  // S2
    B[i+1][j-1] = A[i+1][j] - A[i-1][j];  // S3
    A[i][j] = D[i-1][j-1] * 3;            // S4
 }
```

17. Implement the parallel version of the above sequential loop using OpenMP (function `LoopPar`).
18. Compile the program and execute it, making sure that both loops yield the same result (program must print out "Equal 1").
19. Write down the sequential and parallel wall times for two and four threads, respectively. Compute the associated speedups and efficiencies. Check if the parallel program yields superlinear speedup.
20. Write down the results and conclusions to lab report, **including the dependency graph and the source code** of function `LoopPar` (homework).