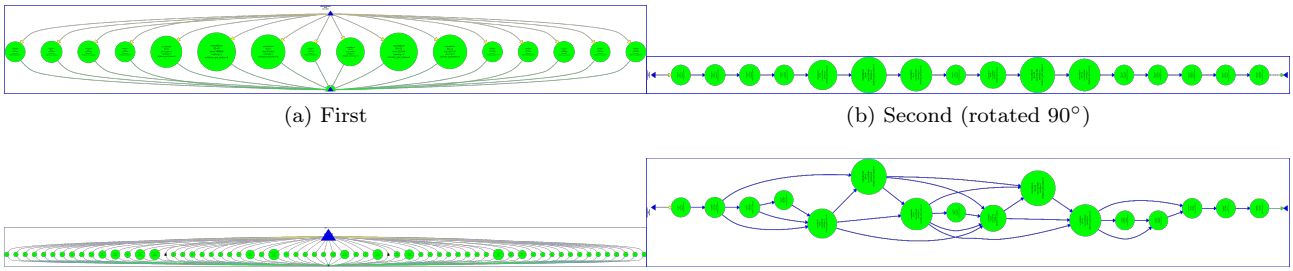
PAR – Final Exam Laboratory – Course 2023/2024-Q2

Student name: ..... **Subgroup:** .....

June 11th, 2024

Problem 1: Lab 3 (3.0 points) Given the following `Tareador` views:



(a) First

(b) Second (rotated 90°)

(c) Third

(d) Fourth (rotated 90°)

We ask you to:

1. (1 point) Indicate which parallel strategy corresponds to each figure. Briefly reason your answer.

```
- No comments are given because it is part of the lab sessions.
first: Coarse grain Iterative (original)
second: Coarse grain Iterative (original, executed using -d)
third: Leaf recursive
fourth: Coarse grain Iterative (original, executed using -h)
```

2. (1 point) Identify the data structures that provoke the data dependencies shown in the second and fourth figures, and how did you run the program in both cases with `run-tareador.sh`.

```
- No comments about the dependences shown are given because it is
  part of the lab sessions.
- Run: ./run-tareador.sh mandelbrot -h (fourth) o -d (second)
```

3. (1 point) Assuming the data dependencies of the previous question:

   (a) Can we make them disappear from `Tareador` views? If so, explain how.

```
% Yes, we can. Using tareador_disable_object(&variable)
```

   (b) How did you deal with those dependencies in Lab 4? Specify which OpenMP directives you used in order to preserve the correctness of your code.

```
- No comments are given because it is part of the lab sessions.
```

Problem 2: Lab 4 (4.5 points) Consider the following table with the speedups obtained from the execution of `submit-omp-strong.sh` script for each of the task decomposition implementations:

Number of threads	Speedup S1	Speedup S2	Speedup S3	Speedup S4
1	0.9	0.9	0.9	0.9
4	1.3	3.8	3.7	3.3
8	1.2	7.1	6.6	4.3
12	1.2	10.3	9.3	5.0
16	1.2	13.7	11.8	5.1
20	1.2	16.8	14.1	5.1

We ask you to answer the following questions:

1. (1 point) Identify the task decomposition that corresponds to each strategy ($S1 - S4$ in the table) based on the approximate speedups shown.

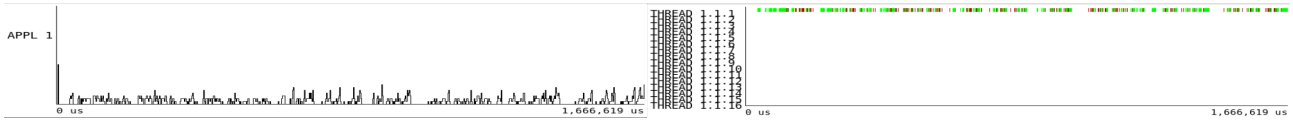
```
S1: Leaf Recursive
S2: Fine Grain Iterative
S3: Tree Recursive
S4: Original Coarse grain Iterative
```

2. (0.5 points) Briefly explain why the speedup of all task decompositions is less than 1 when using one thread.

   This is due to the parallel region overheads, synchronizations overheads, added to the fact we are using only one thread, which is sequential.

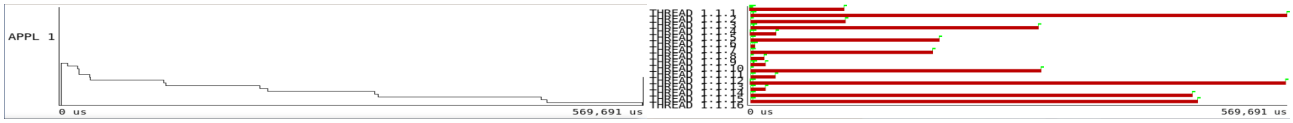
3. (1.5 points) Identify the task decomposition that corresponds to each pair of `Paraver` views for:

   (a) (0.5 points) an execution trace with 16 threads using the "instantaneous parallelism" and the "explicit tasks function created" hints. Briefly reason your answer.



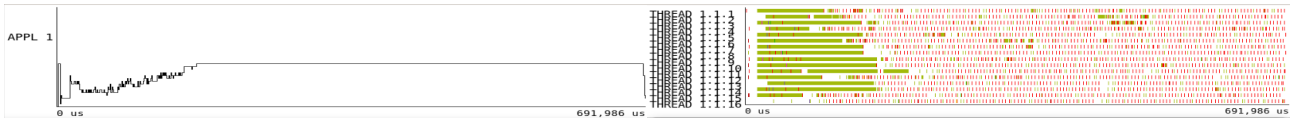
   Leaf recursive task decomposition.
   No comments are given because it is part of the lab sessions.

   (b) (0.5 points) an execution trace with 16 threads using the "instantaneous parallelism" and the "explicit tasks function executed" hints. Briefly reason your answer.



   Coarse grain (original) iterative task decomposition.
   No comments are given because it is part of the lab sessions.

   (c) (0.5 points) an execution trace with 16 threads using the "instantaneous parallelism" and the "explicit tasks function created" hints. Briefly reason your answer.



   Tree recursive task decomposition
   No comments are given because it is part of the lab sessions.

4. (1.5 points) Propose an OpenMP version of this portion of code in order to exploit the parallelism between the two for loops while maintaining the correctness of the subsequent code:

```
...
for (int px = x; px < x + TILE; px++) {
  M[y][px] = pixel_dwell(COLS, ROWS, CminR, CminI, CmaxR, CmaxI, px, y,...);
  M[y+TILE-1][px] = pixel_dwell(COLS, ROWS, CminR, CminI, CmaxR, CmaxI, px, y+TILE-1,...);
  equal = equal && (M[y][x] == M[y][px]);
  equal = equal && (M[y][x] == M[y + TILE - 1][px]);
}

for (int py = y; py < y + TILE; py++) {
  M[py][x] = pixel_dwell(COLS, ROWS, CminR, CminI, CmaxR, CmaxI, x, py,...);
  M[py][x+TILE-1] = pixel_dwell(COLS, ROWS, CminR, CminI, CmaxR, CmaxI, x+TILE-1,py,...);
  equal = equal && (M[y][x] == M[py][x]);
  equal = equal && (M[y][x] == M[py][x + TILE - 1]);
}

if (equal && M[y][x]==maxiter)
 { ...  }
else { ... }
```

No code is given because it is part of the lab sessions.

Problem 3: Lab 5 (2.5 points) Complete the following table with the information regarding the data decompositions done in the laboratory. You should fill the table with the data decompositions (second column) that have the characteristic of the first column, and briefly reason why in the last column. Note: S_p stands for the speedup using p threads.

Characteristic	Data Decomposition(s)	Reason Why
Load balancing for implicit tasks less than 50%		
Load balancing for implicit tasks is 100%, S_{20} is very high		
Load balancing for implicit tasks is 100%, but $S_{20} \leq S_{16}$		
It has the lowest number of L2 misses for 20 threads		
It has the highest number of L2 misses for 20 threads		
IPC is very low due to false sharing coherence protocol		
If `maxiter` is set to 1 it may have the same speedup as the best strategy		
It is difficult to affect its/their load balancing although we increase `maxiter`		

No comments are given in some parts because it is reported in the lab sessions.

Possible Solution:

Characteristic	Data Decomposition(s)	Reason Why
Load balancing for implicit tasks less than 50%	block by columns	
Load balancing for implicit tasks is 100%, S_{20} is very high	cyclic by rows	
Load balancing for implicit tasks is 100%, but $S_{20} \leq S_{16}$	cyclic by columns	
It has the lowest number of L2 misses for 20 threads	cyclic by rows	
It has the highest number of L2 misses for 20 threads	cyclic by columns	
IPC is very low due to false sharing coherence protocol	cyclic by columns	
If `maxiter` is set to 1 it may have the same speedup as the best strategy	block by columns	if the mandelbrot to be computed has not unbalance, it would not have unbalance
It is difficult to affect its/their load balancing although we increase `maxiter`	cyclic by rows or columns	they are very well balanced (1 row/1 column)