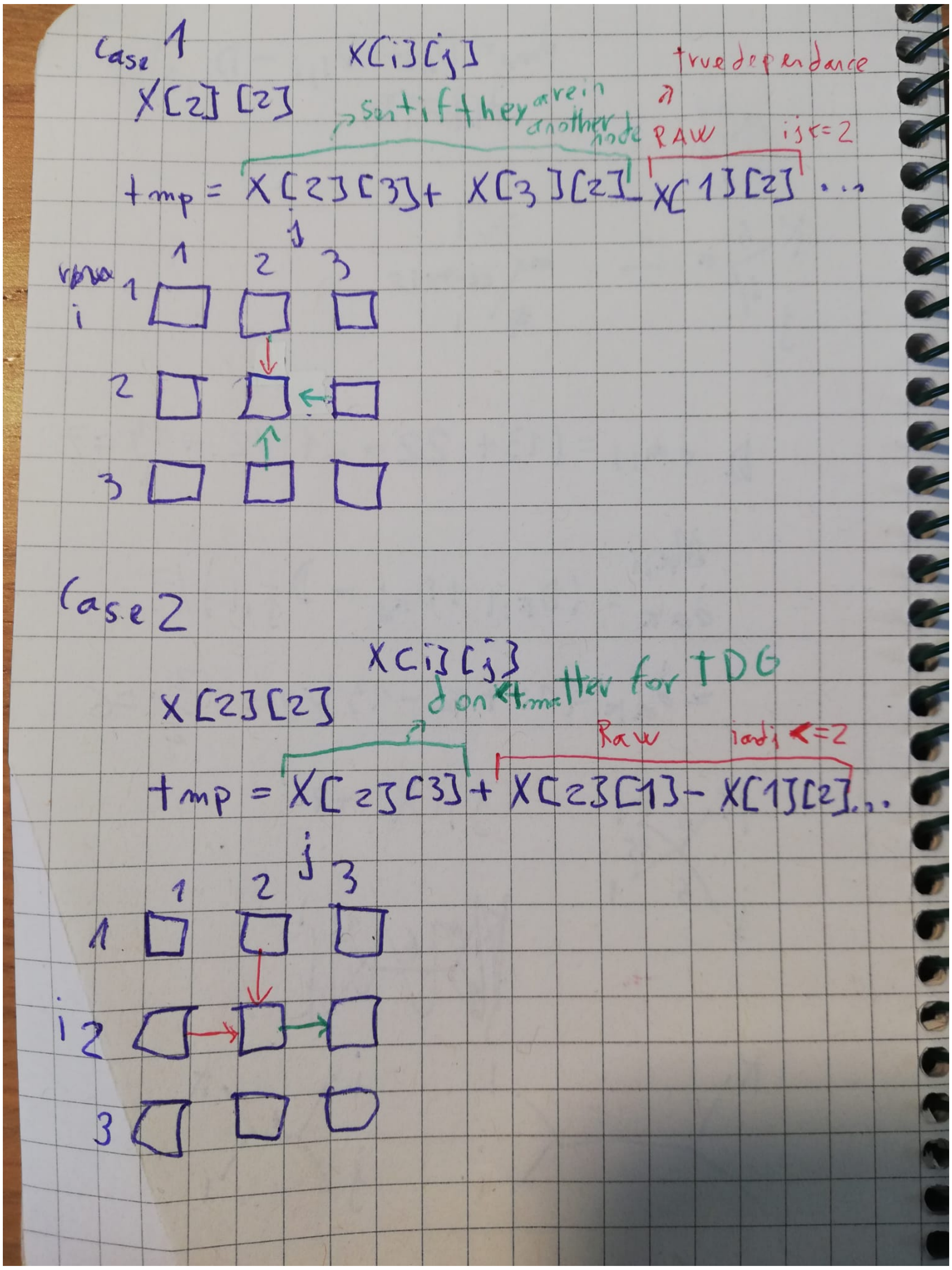
Started on	Friday, 26 April 2024, 11:42 AM
State	Finished
Completed on	Friday, 26 April 2024, 12:16 PM
Time taken	34 mins 23 secs
Grade	**3.00** out of 3.00 (**100**%)



Red part is the one that is used for TDG

Question 1

Correct

Mark 3.00 out of 3.00

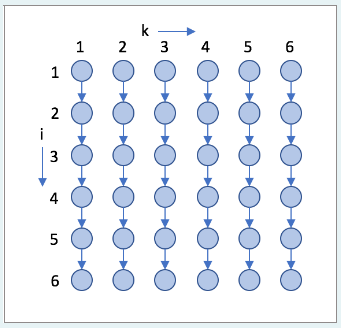
Let's assume a distribution of the elements of matrix X to P processors by rows (n ÷ P consecutive rows per processor). For example, the figure below illustrates the matrix distribution for P = 4 and n = 8:

P_0	[0,0]	[0,1]	[0,2]	[0,3]	[0,4]	[0,5]	[0,6]	[0,7]
	[1,0]	[1,1]	[1,2]	[1,3]	[1,4]	[1,5]	[1,6]	[1,7]
P_1	[2,0]	[2,1]	[2,2]	[2,3]	[2,4]	[2,5]	[2,6]	[2,7]
	[3,0]	[3,1]	[3,2]	[3,3]	[3,4]	[3,5]	[3,6]	[3,7]
P_2	[4,0]	[4,1]	[4,2]	[4,3]	[4,4]	[4,5]	[4,6]	[4,7]
	[5,0]	[5,1]	[5,2]	[5,3]	[5,4]	[5,5]	[5,6]	[5,7]
P_3	[6,0]	[6,1]	[6,2]	[6,3]	[6,4]	[6,5]	[6,6]	[6,7]
	[7,0]	[7,1]	[7,2]	[7,3]	[7,4]	[7,5]	[7,6]	[7,7]

We ask you to: indicate which Task Dependence Graph (TDG) would be the one generated for each of the two codes assuming we want to generate a task for each iteration of the inner loop.
Hint: When analyzing dependencies for creating a Task Dependence Graph (TDG) we need to check, for the original sequential code, which values were modified before the value being updated in the current iteration. Those are the ones that need to be *Read After Written (RAW)* and cause a *True dependence*. So, please check carefully how loop iterations are sequentially executed one after the other, and focus on which memory locations they access for reading and writing.

Code (1):

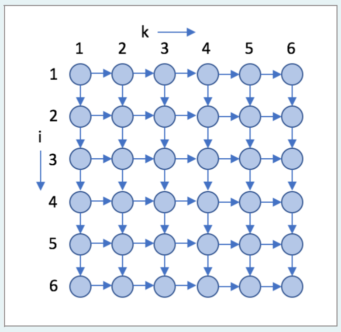
```
int X[n][n], COEFF[n];
...
for (int i = 1; i < n-1; i++) {
    for (int k = 1; k < n-1; k++) {
        start_task ("loop");
        int tmp = X[i][k+1] + X[i+1][k] - X[i-1]
[k] - COEFF[k];
        X[i][k] = tmp/4;
        end_task ("loop");
    }
}
```



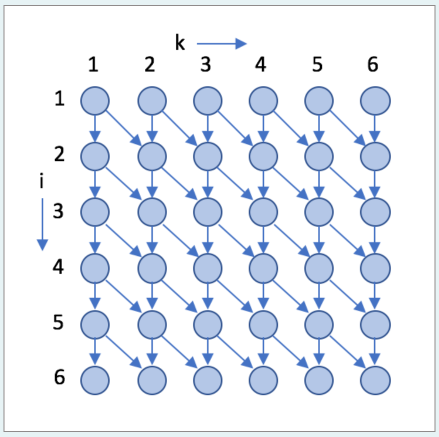
✓

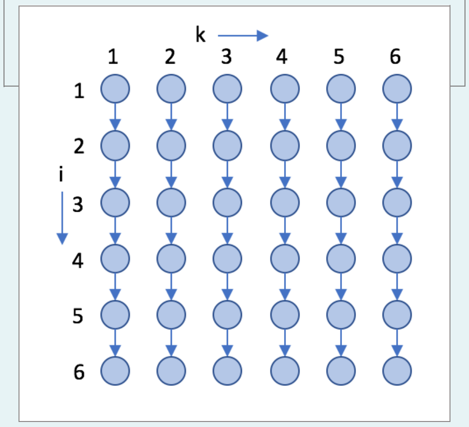
Code (2):

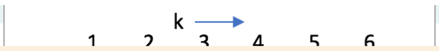
```
int X[n][n], COEFF[n];

...
for (int i = 1; i < n-1; i++) {
    for (int k = 1; k < n-1; k++) {
        start_task ("loop");
        int tmp = X[i][k+1] + X[i][k-1] - X[i-1]
[k] - COEFF[k];
        X[i][k] = tmp/4;
        end_task ("loop");
    }
}
```



✓







Your answer is correct.

When analyzing dependencies for creating a Task Dependence Graph (TDG) we need to check, for the original sequential code, which values were modified before the value being updated in the current iteration. Those are the ones that need to be *Read After Written (RAW)* and cause a *True dependence.*

For instance, for Code (1):

in the original sequential code before any parallelization, neither X[i][k+1] nor X[i+1][k] have been modified yet by the time X[i][k] is updated. Therefore, when going parallel, the task updating X[i][k] does not need to wait for the tasks which are going to update them. However, it has to wait for the task which updates X[i][k-1] because in the original sequential order it is modified before X[i][k].

To avoid creating an incorrect parallel code always check which was the original order in the sequential execution and determine the dependencies that need to be fulfilled.

(*) There are other kind of dependencies, that will be explored later in the course:

- *Write After Read (WAR)* also known as (a.k.a.) *Antidependence*

  and

- *Write After Write (WAW), a.k.a. Output Dependence.)*

Note that the distribution of the elements of matrix X to P processors by rows is not relevant for the creation of the TDG which depends solely on the calls to the *start_task* and *end_task* API and the data dependences it detects during a sequential execution of the code.