Ejercicios

2. Given the following C code in which two tasks have been identified using the *Tareador* API:

```
#define MAX 8
// initialization
for (outer = 0; outer < MAX; outer++) {
   tareador_start_task("for-initialize");
   for (inner = 0; inner < MAX; inner++)
      matrix[outer][inner] = inner;
   tareador_end_task("for-initialize");
   }

// computation
for (outer = 0; outer < MAX; outer++) {
   tareador_start_task("for-compute");
   for (inner = 0; inner <= outer; inner++)
      matrix[outer][inner] = matrix[outer][inner] + foo(outer,inner);
   tareador_end_task("for-compute");
   }
```

Assuming that: 1) in the initialization loop the execution of each iteration of the internal loop lasts 10 cycles; 2) in the computation loop the execution of each iteration of the internal loop lasts 100 cycles; and 3) the execution of the `foo` function does not cause any kind of dependence. **We ask**:

(a) Draw the Task Dependence Graph (TDG), indicating for each node its cost in terms of execution time.

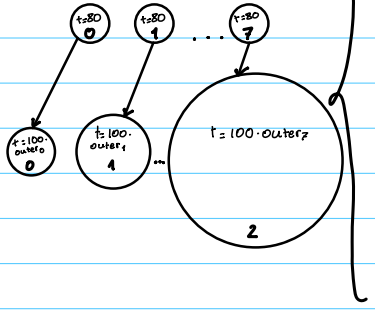
(b) Calculate the values for T_1 and T_∞ as well as the potential *Parallelism*.

(c) Calculate which is the best value for the "speed-up" on 4 processors (S_4), indicating which would be the proper task mapping (assignment) to processors to achieve it.

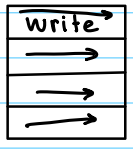
a)

1) Ver tareas:

2) TDG:



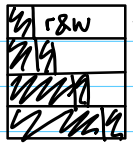
Matrix ini



10 · MAX

$10 \cdot MAX^2$ ciclos

Matrix compu



100 · outer

$100 \cdot outer_0 + \dots + 100 \cdot outer_7$

$= 100 \cdot 3600$

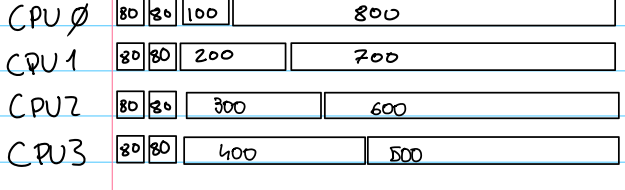
b) T_1, T_∞, Parallelism

$T_1 = 80 \cdot 8 + 3600 = 640 + 3600 = 4240$ cic.

$T_\infty = 80 + 100 \cdot 8 = 880$ ciclos

$$Par = \frac{T_1}{T_\infty} = \frac{4240}{880} = 4{,}8$$

c) $S_4 = \frac{T_1}{T_4}$



$T_4 = 1060$

$$S_4 = \frac{4240}{1060} = 4$$

P11

11. Given the following C code with tasks identified using the *Tareador* API:

```
#define N 4
int m[N][N];

// loop 1
for (int i=0; i<N; i++) {
    tareador_start_task ("loop1");
    for (int k=0; k<=i; k++) {
        m[i][k] = comp1(i,k); // no access to m inside function comp1
    }
    tareador_end_task ("loop1");
}

// loop 2
for (int i=0; i<N; i++) {
    tareador_start_task ("loop2");
    for (int k=i+1; k<N; k++) {
        m[i][k] = comp2(m[k][i]); // no access to m inside function comp2
    }
    tareador_end_task ("loop2");
}

// print all the elements of m
tareador_start_task("print");
print_results(m);
tareador_end_task("print");
```

Assuming that: 1) the execution of the each invocation of functions `comp1` and `comp2` functions takes 10 time units; and 2) the execution of function `print_results` takes 20 time units; **we ask:**

(a) Indicate which positions of matrix `m` are read and/or written by each task generated in `loop 1` and `loop 2` and in task `print_results`. You can fill in the table below to answer this question.

print_results

task id	Read	Written
t9		

(b) Draw the Task Dependence Graph (TDG), indicating for each node its cost in terms of execution time (in time units). Use the task identifiers that we provided in the previous table (i.e. *t*1 ... *t*9).

(c) Compute the values for T_1, T_∞ and the potential *Parallelism*.

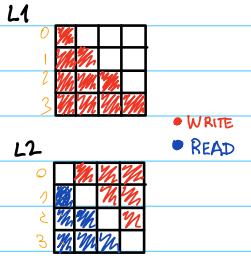
(d) Let's consider that each task creation has an associated overhead of 2 time units. Taking this overhead into account, and assuming that tasks are created in the order in which they are found in the sequential execution, compute the new values for T_1 and T_∞. Clearly identify to which tasks the overhead accounts for.

(e) Assuming a distributed memory machine with a matrix distribution by rows on four processors and a message passing model where the transfer cost of a message of B elements is $t_{comm} = t_s + B \times t_w$ being t_s y t_w the start–up time and transfer time of one element, respectively; write the expression that determines the execution time T_4 of the program (taking into account computation time and data sharing overheads only) for the following data and task assignment to processors:

Processor	P0	P1	P2	P3
Row distribution	m[0][0..3]	m[1][0..3]	m[2][0..3]	m[3][0..3]
Task assignment	*t*1, *t*5, *t*9	*t*2, *t*6	*t*3, *t*7	*t*4, *t*8

a) Comp1 & Comp2 → 10 u.t

Print → 20 u.t



loop 1

task id	iteration (i, k)	Read	Written
t1	(0, 0)		✗
t1	(0, 1)	—	
t1	(0, 2)	—	
t1	(0, 3)	—	
t2	(1, 0)		✗
t2	(1, 1)		✗
t2	(1, 2)	—	
t2	(1, 3)	—	
t3	(2, 0)		✗
t3	(2, 1)		✗
t3	(2, 2)		✗
t3	(2, 3)	—	
t4	(3, 0)		✗
t4	(3, 1)		✗
t4	(3, 2)		✗
t4	(3, 3)		✗

loop2

task id	iteration (i, k)	Read	Written
t5	(0, 0)	—	
t5	(0, 1)	✗	✗
t5	(0, 2)	✗	✗
t5	(0, 3)	✗	✗
t6	(1, 0)	—	
t6	(1, 1)	—	
t6	(1, 2)	✗	✗
t6	(1, 3)	✗	✗
t7	(2, 0)	—	
t7	(2, 1)	—	
t7	(2, 2)	—	
t7	(2, 3)	✗	✗
t8	(3, 0)	—	
t8	(3, 1)	—	
t8	(3, 2)	—	
t8	(3, 3)	—	

print_results

task id	Read	Written
t9	✗	

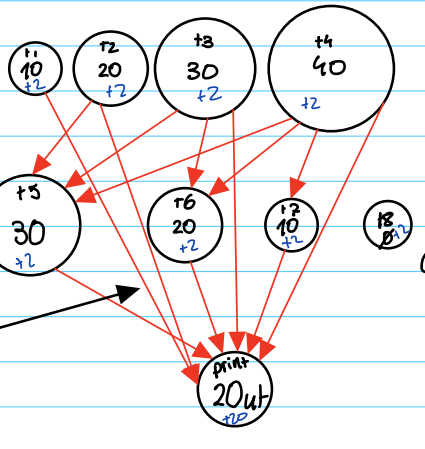
Trabajos es

42 + 32 + 22

o

40+8 + 30+8 + 20+8 ?

b)



Las tareas del L2 no diagonales → necesitamos L1

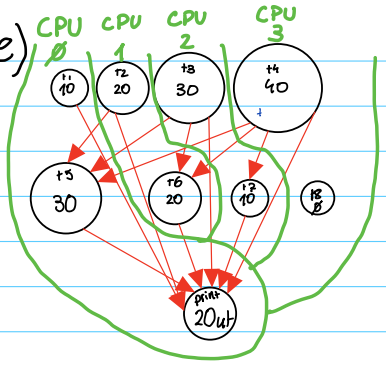
c) $T_1 = 100 + 60 + 20 = 180$

$T_\infty = 40 + 30 + 20 = 90$

$Par = \frac{180}{90} = 2$

d) En T_1 no hay overheads pq no hay creación de tareas

e)



$T_4 = T_{com} + T_{comp} = t_s + B \cdot t_w + T_{comp}$

CPU0: t1 | CPU 1 2 3 t5 | 1 2 3 t9

CPU1: t2 | 2 3 t6

CPU2: t3 | 3 t7

CPU3: t4

$T_4 = T_{comp} + T_{mov} =$

$T_{comp} = T_4 + T_5 + T_9$

$T_{mov} = 3(t_s + 1 \cdot t_w) + 3(t_s + 4 \cdot t_w)$

12. For each of these two loops:

```
1) for (i=1; i<n; i++)
     for (j=1; j<n; j++) {
      B[i][j] = A[i][j-1]+A[i-1][j]+A[i][j];
     }
```

```
2) for (i=1; i<n; i++)
     for (j=1; j<n; j++) {
      A[i][j] = A[i][j-1]+A[i-1][j]+A[i][j];
     }
```

answer the following questions:

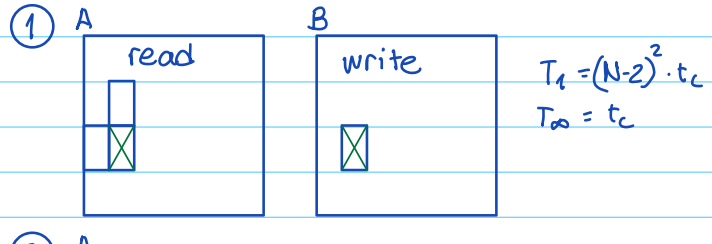
(a) Assuming that we define a task as the body of the innermost loop and that its execution time is t_c, calculate T_1 and T_∞.

(b) Consider that we are using a distributed memory machine with P processors and that matrices A and B are distributed by columns, each processor storing in its local memory the elements in a block of n rows by n/P consecutive columns.

- Determine, for each code, which is the most suitable parallelisation strategy for the data distribution indicated above.
- Calculate T_P (including both the computation time and data sharing overhead) for each code and the parallelisation strategy you proposed, assuming that the cost for a message of B elements is $t_s + B \times t_w$.

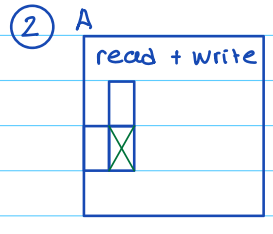
P12

a)



$T_1 = (N-2)^2 \cdot t_c$

$T_\infty = t_c$

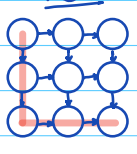


$T_1 = (N-2)^2 \cdot t_c$

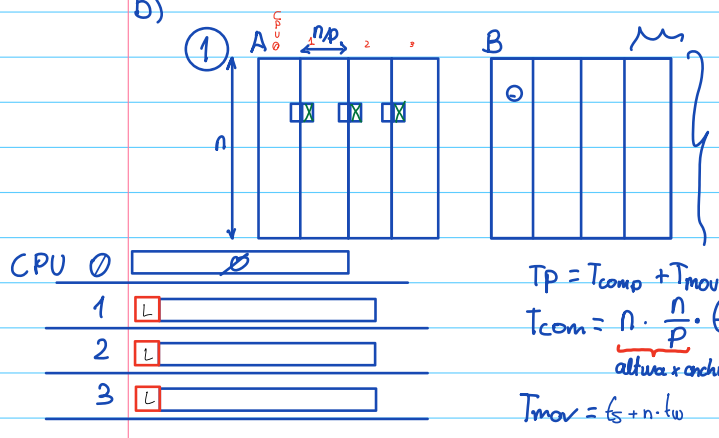
$T_\infty = ((N-1)+(N-1)-1)\,t_c$

⚠ El -1 (y -2) viene de que i,j empiezan en 1

TDG



b)



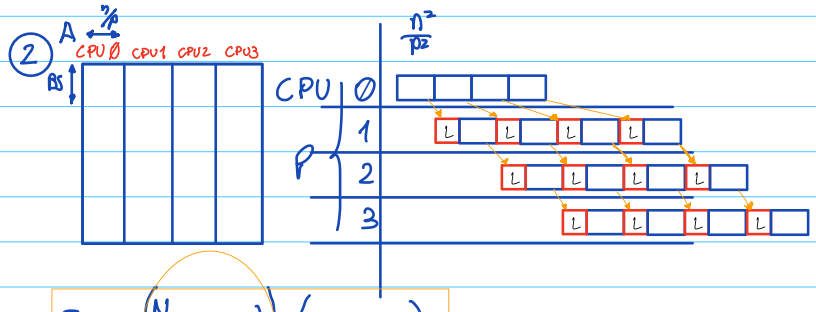
1) Coarse-grained
2) Elemento
3) Bloque

$T_p = T_{comp} + T_{mov}$

$T_{com} = n \cdot \frac{n}{p} \cdot t_c$

altura x anchura columna

$T_{mov} = t_s + n \cdot t_w$



$T_{mov} = \left(\frac{N}{BS} + (P-2)\right) \cdot (t_s + BS \cdot t_w)$

$T_{comp} = \left(\frac{N}{BS} + (P-1)\right) \cdot (BS \cdot \frac{N}{p} \cdot t_c)$

→ ¿por qué?

P3

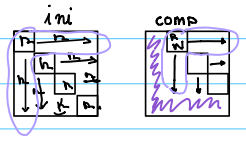
3. Given the following C code with tasks identified using the *Tareador* API:

```
#define N 4
int m[N][N];

// initialization
for (int i=0; i<N; i++) {
    tareador_start_task ("for_initialize");
    for (int k=i; k<N; k++) {
        if (k == i) modify_d(&m[i][i], i, i);
        else {
            modify_nd (&m[i][k], i, k);
            modify_nd (&m[k][i], k, i);
        }
    }
    tareador_end_task ("for-initialize");
}

// computation
for (int i=0; i<N; i++) {
    tareador_start_task ("for_compute");
    for (int k=i+1; k<N; k++) {
        int tmp = m[i][k];
        m[i][k] = m[k][i];
        m[k][i] = tmp;
    }
    tareador_end_task ("for-compute");
}

// print results
tareador_star_task ("output");
print_results(m);
tareador_end_task ("output");
```

3.-

5ut

100ut



b) $T_1 = \sum T_{total} = \frac{230}{145} = 1,58$ = Parallelism

$T_{\infty} = 40+15+100 = 145$

$T_{par} = 55$

$\Psi = \frac{T_{par}}{T_1} = \frac{55}{230} = 0,24$

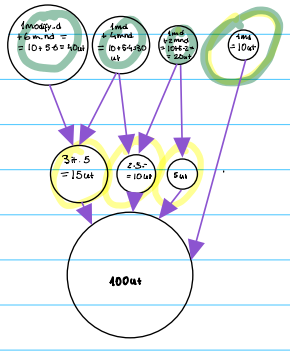
modify_d → 10 ut

modify_nd → 5ut

PASO 1: ver tareas

PASO 2: TDG

PASO 3: Mem distrib.



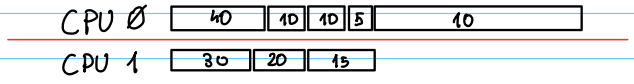
40 + 10 + 10 + 5 + 10

30 + 20 + 15

$T_2 = T_{sec} + T_{par}/P = (1-\Psi)\cdot T_1 + \frac{55}{2} = 0,76 \cdot 230 + \frac{55}{2} = 225,3$

$S_2 = \frac{T_1}{T_2} = \frac{230}{225,3} = 1,02$

c)



P5

5. The following figure shows an incomplete time diagram for the execution of a parallel application on 4 processors:

The figure has a set of rectangles, each rectangle represents the execution of a task with its associated cost in time units. In the timeline there are two regions (1 and 2) with 4 parallel tasks each. The execution cost for tasks in $region_1$ is unknown (x time units each); the cost for each task in $region_2$ is 4 time units. The computation starts with a sequential task (with cost 5), then all tasks in $region_1$ running in parallel, followed by another sequential task (with cost 2), then all tasks in $region_2$ running in parallel followed by a final sequential task (with cost 1).

Knowing that an ideal speed-up of 9 could be achieved if the application could make use of infinite processors ($S_{p\to\infty} = 9$), and assuming that the two parallel regions can be decomposed ideally, with as many tasks as processors with the appropriate fraction of the original cost, **we ask:**

(a) What is the parallel fraction (ϕ) for the application represented in the time diagram above?

(b) Which is the "speedup" that is achieved in the execution with 4 processors (S_4)?

(c) Which is the value x in $region_1$?

a) $S_{p\to\infty} = \frac{1}{1-\varphi} = 9 \quad 9 - 9\varphi = 1$

$8 = 9\varphi \quad \varphi = \frac{8}{9} = 0{,}89$

b) $S_4 = \frac{T_1}{T_4} = \frac{T_1}{T_{sec} + T_{par}} = \frac{T_1}{(1-\varphi)\cdot T_1 + \varphi \cdot \frac{T_1}{p}}$

$= \frac{1}{1-\varphi+\frac{\varphi}{p}} = \frac{1}{0{,}12 + \frac{\varphi}{4}} = 3$

c) $S_4 = 3 = \frac{T_{par}}{T_1} = \frac{4x+16}{8+4x+16} \rightarrow$

PREGUNTA ⟶ ¿Por qué este cálculo?

P10

10. Given the following two loops in a C program instrumented with *Tareador*:

```
#define N 16
#define BS 4
void main() {
char stringMessage[16];

tareador_ON();
for (int ii = 0; ii < N; ii=ii+BS) {
    sprintf(stringMessage,"loop1(%d)",ii);
    tareador_start_task(stringMessage);
    for (int i = ii;  i< ii+BS; i++)                    // BS perfectly divides N
        for (int j = 0; j < N; j++)
            b[i][j] = foo(a[i][j]);
    tareador_end_task(stringMessage);
    }
for (int ii = 0; ii < N; ii=ii+BS) {
    sprintf(stringMessage,"loop2(%d)",ii);
    tareador_start_task(stringMessage);
    for (int i = max(1,ii); i < min(ii+BS, N-1); i++) // min y max to ensure the access
        for (int j = 0; j < N; j++)                   // within the range of matrix b
            c[i][j] = goo(b[i][j], b[i-1][j], b[i+1][j]);
    tareador_end_task(stringMessage);
    }
tareador_OFF();
}
```

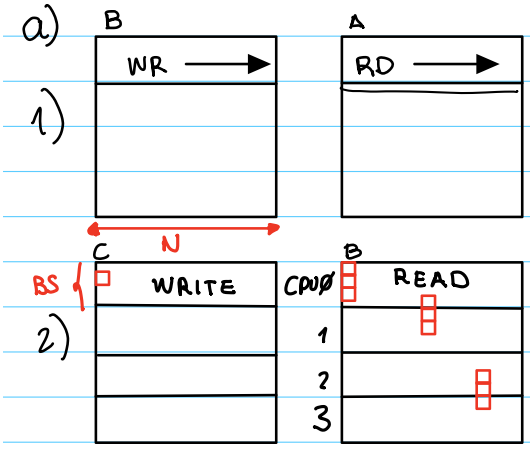
We ask:

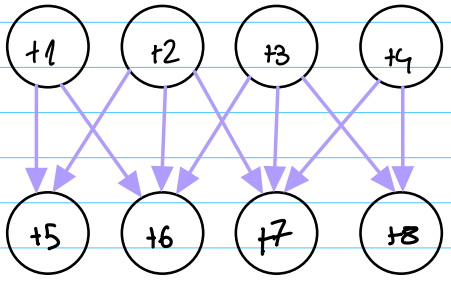
(a) Draw the Task Dependence Graph (TDG) that would be generated, assuming functions `foo` and `goo` only access to their arguments and do not update any other global variable.

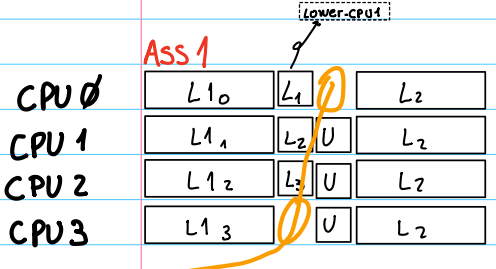
(b) Write the expression that determines the execution time T_4, clearly indicating the contribution of the computation time T_4^{comp} and data sharing overhead T_4^{mov}, for the two following assignments of tasks to processors:

Task	Assignment 1	Assignment 2
loop1(0)	0	0
loop1(4)	1	1
loop1(8)	2	2
loop1(12)	3	3
loop2(0)	0	0
loop2(4)	1	0
loop2(8)	2	0
loop2(12)	3	0

You can assume: 1) a distributed-memory architecture with 4 processors; 2) matrices `a`, `b` and `c` are initially distributed by rows (N/BS consecutive rows per processor); 3) once the second loop is finished, you don't need the return matrices to their original distribution; 4) data sharing model with $t_{comm} = t_s + m \times t_w$, being t_s y t_w the start–up time and transfer time of one element, respectively; and 5) the execution time for a single iteration of the innermost loop body takes t_c.





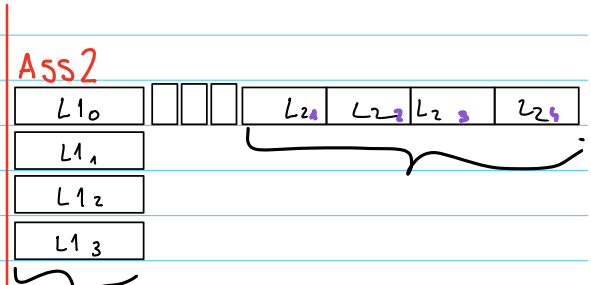


WHY NOT?

$T_4 = T_4^{comp} + T_4^{mov}$

$T_4^{comp} = 2 \cdot (BS \cdot N \cdot t_c)$

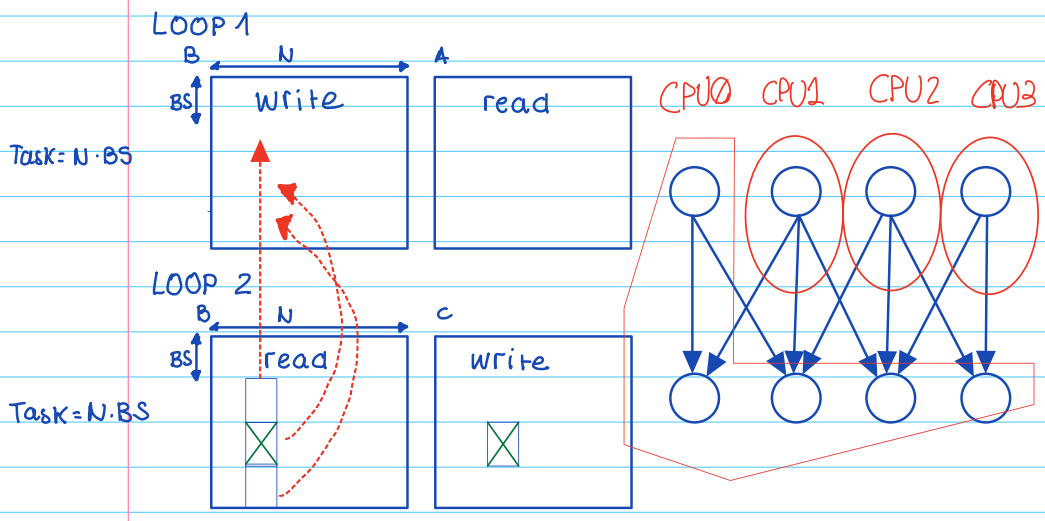
$T_4^{mov} = 2(t_s + N \cdot t_w)$



$T_4 = T_4^{comp} + T_4^{mov}$

$T_4^{comp} = (BS \cdot N \cdot t_c) \cdot 5$ → WHY NOT?

$T_4^{mov} = 3 \cdot (t_s + N \cdot BS \cdot t_w)$


LOOP 1
B
N
A
BS
write
read
Task: N·BS
LOOP 2
B
N
C
BS
read
write
Task=N.BS
CPU0
CPU1
CPU2
CPU3