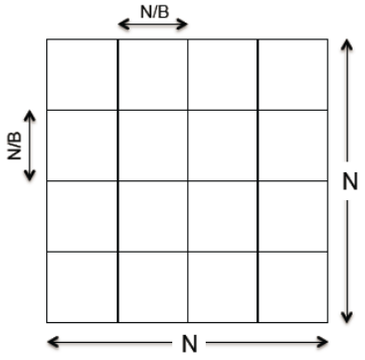
Problema 9 (d and e only)

Given the following code computing matrix u, in which the k loop iterates from 1 to i:

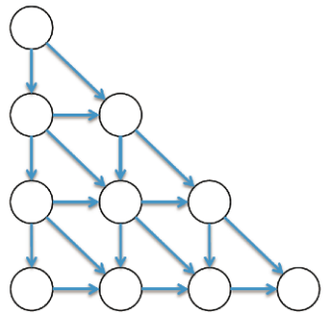
```
for (i = 1; i < N-1; i++)
    for (k = 1; k <= i; k++) {
        tmp = 0.3 * u[i+1][k+1] + 0.7 * u[i-1][k-1];
        u[i][k] = (tmp * tmp) / 4;
    }
```

and the following tasks definition: a task is in charge of computin a block of N/B rows by N/B columns

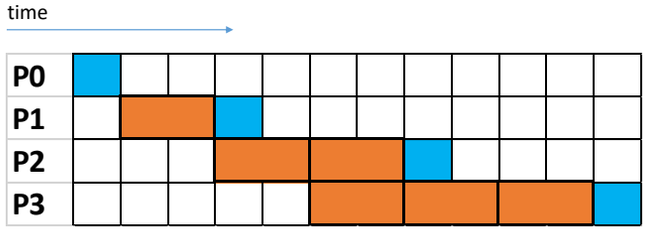


(d) If these tasks are mapped to a machine with $P = B$ processors, so that each row of tasks is mapped to a different processor, and assuming that N is very large compared to P ($N >> P$). We ask to draw a temporal diagram with the parallel execution and obtain the expression for T_p as a function of N.

El primer que hem de dibuixar és el TDG, a partir dels accessos que es fan en el codi:



on tenim les tasques que calculen els blocs diagonals i les tasques que calculen els blocs no diagonals. El diagrama temporal de l'execució seria:



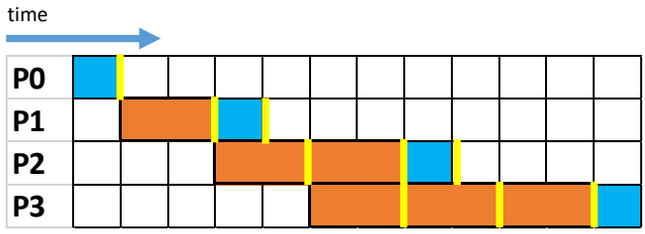
en el que els quadrats blaus representen l'execució d'un bloc diagonal i els rectangles taronja l'execució d'un bloc no diagonal.

Quina seria l'expressió per T_p?

$$T_p = 2 \cdot t_d + (P-2) \cdot t_{nd} + (B-1) \cdot t_{nd} = 2 \cdot t_d + (2 \cdot P - 3) \cdot t_{nd}$$

(e) If the cost for each synchronization between two processors takes t_{sync} time units, obtain the synchronization overhead that would be added to the previous expression for T_p.

Cada tasca, diagonal o no diagonal, ha d'avisar a les tasques que depenen d'ella, amb un overhead de t_{sync}. En el diagrama temporal són les franges grogues que hem afegit:



Quina seria la nova expressió per T_p en la que incloem aquest overhead?

$$T_p = 2 \cdot t_d + (2 \cdot P - 3) \cdot t_{nd} + T_{ovh}$$
$$T_{ovh} = (P - 1) \cdot t_{sync} + (B - 1) \cdot t_{sync} = 2 \cdot (P-1) \cdot t_{sync}$$

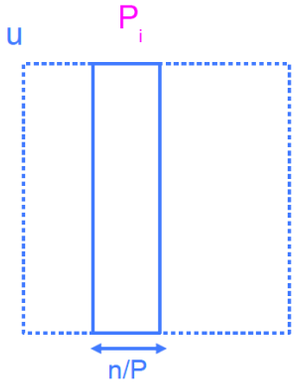
Problema 14

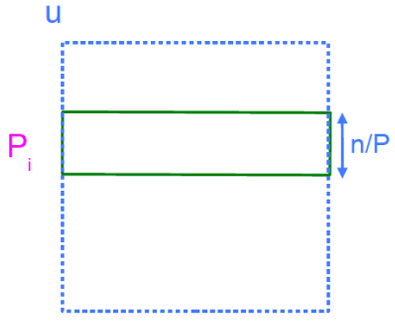
We want to find the expression that determines the parallel execution time in p processors (T_p) for the following loop:

```
for (i=1; i<n; i++) {
    for (k=0; k<n-1; k++) {
        u[i][k] = 0.8*u[i-1][k] + 0.5*u[i][k+1] - 0.2*u[i][k];
    }
}
```

using the data sharing model explained in class based on the distributed memory architecture with message passing: the access time to remote data is determined by $t_{comm} = t_s + m \times t_w$, being t_s and t_w the "start-up" and sending time of an element, respectively, and being m the size of the message. The execution time of the iteration of the body of the most internal loop is t_c.

Two different data distributions are considered: *Column distribution* (the matrix u is distributed so that each processor has n/p consecutive columns) and *Row distribution* (the matrix u is distributed so that each processor has n/p consecutive rows). For each data distribution **we ask** to 1) define the most appropriate definition of a task; and 2) complete the following table with the different contributions to T_p.

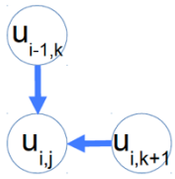




Quines iteracions dels bucles hauria d'executar cada processador per tal de minimitzar els accessos remots? (o en altres paraules, maximitzar els accessos locals, amb cost zero)?

a) Column distribution: un processador hauria d'executar aquelles iteracions de k que escriuen (i llegeixen) elements de u que estan en la seva memòria local. Totes les iteracions del bucle i.
b) Row distribution: un processador hauria d'executar aquelles iteracions de i que escriuen (i llegeixen) elements de u que estan en la seva memòria local. Totes les iteracions del bucle k.

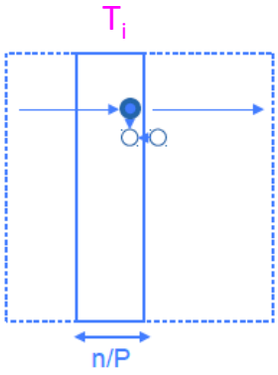
Si ho fem així, quins accessos dels que es fan en el bucle provocarien dependències de dades?



Només l'accés a u[i-1][k] provoca una dependència Read-After-Write. L'altre accés a u[i][k+1] no provoca dependència Read-After-Write.

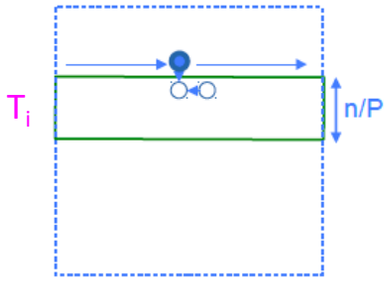
Comencem per **l'accés a l'element u[i-1][k]:**

Column Distribution:



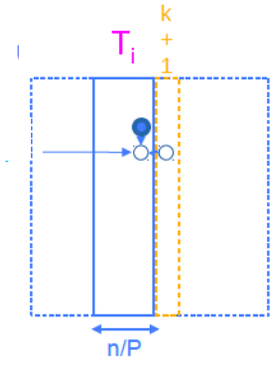
NO provoca dependència
NO provoca accés remot

Row Distribution:



SI provoca dependència
SI provoca accés remot, després de calcular

En els dos casos, **l'accés a l'element u[i][k+1]** **NO** provoca dependència de dades, ja que l'element de la dreta es calcula a posteriori. Però **SI** provocarà accessos remots en el cas de *Column Distribution*, i **NO** en el cas de *Row Distribution*.

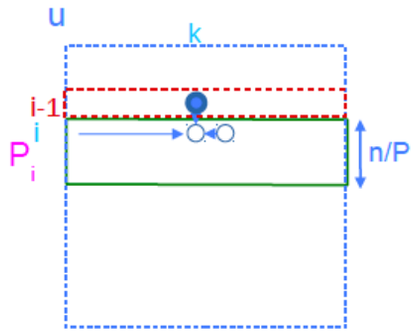


Un processador necessita la primera columna sencera del processador de la dreta (tots menys el processador P-1).

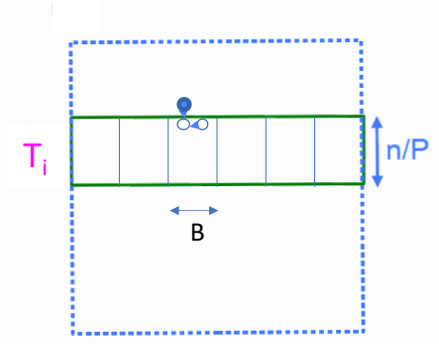
El diagrama temporal d'execució seria: (R representa l'accés remot a la columna del processador de la dreta, amb cost de $t_s+N \cdot t_w$, i Task l'execució de n/Pxn elements del bloc columna)

P0	R	Task
P1	R	Task
P2	R	Task
P3		Task

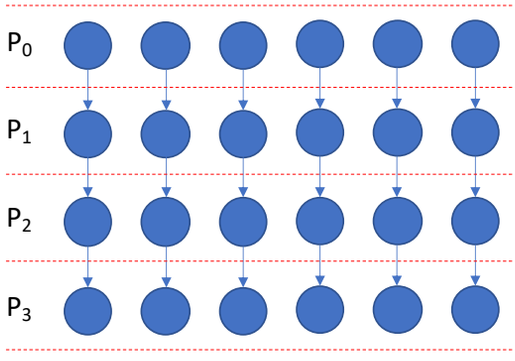
En el cas de *Row Distribution*, l'accés remot és provocat per la dependència i implica transferir tota la darrera fila del processador anterior. Ja veieu que no està traient cap paral·lelisme del problema. **Quina hauria de ser la definició de tasca llavors? Anem a una granularitat més fina.**



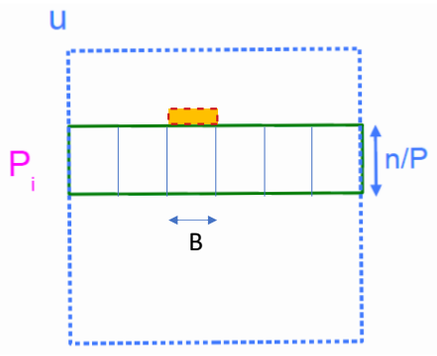
Per la *Row Distribution*, si definim una tasca com un bloc de n/P files per B columnes, que guanyaríem? Doncs que ara la dependència només es amb una tasca de n/PxB elements



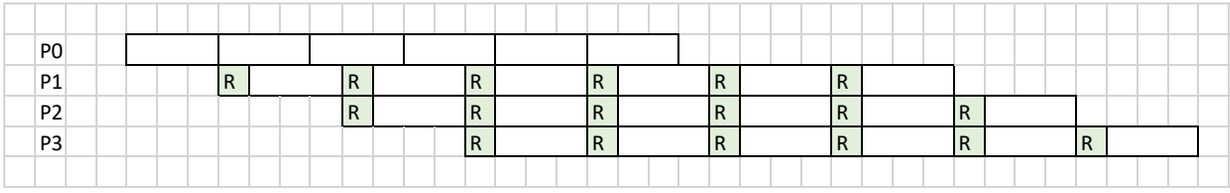
Amb aquesta definició de tasca queda un TDG (nomes amb les Read-after-Write dependences) com el mostrat a continuació: (mostrant també l'assignació de tasques a processadors)



Cada tasca, a què hauria d'accedir del processador anterior? Un tros de fila de B elements, tal com es mostra a continuació:



El camí crític el defineixen les primeres tasques en cada processador que formen una cadena de dependències, mes la resta de tasques que executa l'últim processador (P_3 en el dibuix) una darrera l'altre. I en aquest camí tenim tants remote accesses, de B elements cadascun, com fletxes entren en els nodes que tenim en el camí crític.



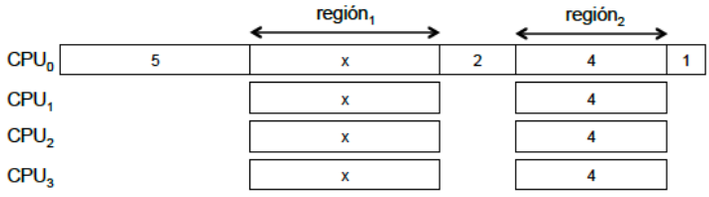
En total (P-1)+(B-1) accessos R en el camí crític. Cada R afegint un overhead de $t_s+B\cdot t_w$. I cada tasca amb temps $(n/P\cdot B)\cdot t_c$.

Per tant, la taula que es demana en el problema seria la següent:

		Distribución por columnas	Distribución por filas
Definicion de tarea		$N \div p$ iteraciones bucle k	$(N \div p) \times B$ iteraciones de los bucles i y k, respectivamente
Accesos remotos iniciales	Número total de mensajes	$p-1$	–
	Tamaño de cada mensaje	$\approx N$	–
	Contribución a T_p	$t_s + N \times t_w$	0
Parallel computation	Numero total de tareas	p	$p \times (N \div B)$
	Tamano de cada tarea	$(N \div p) \times N$	$(N \div p) \times B$
	Contribución a T_p	$(N^2 \div p) \times t_c$	$((N \div B)+p-1) \times ((N \div p) \times B) \times t_c$
Accesos remotos durante cálculo paralelo	Número total de mensajes	–	$(p-1) \times (N \div B)$
	Tamaño de cada mensaje	–	$\approx B$
	Contribución a T_p	0	$((N \div B)+p-2) \times (t_s + B \times t_w)$

Problema 5 Tema 2

Given the following incomplete time diagram for the execution of a parallel application on 4 processors:



Numbers inside the boxes represent the execution time for the different execution bursts, being this value (x) unknown for the bursts in $region_1$. Knowing that a "speed-up" of 9 could be achieved when the application makes use of infinite processors ($S_\infty = 9$, assuming that the parallel regions can be decomposed into infinity ∞), **we ask:**

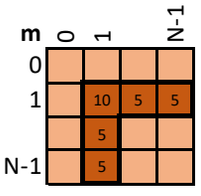
(a) What is the parallel fraction (ϕ) for the application represented in the time diagram above?
Solución: From $S_\infty = 1/(1-\phi)$ we can compute $\phi = 8/9$

(b) Which is the "speedup" that is achieved in the execution with 4 processors (S_4)?
Solución: Once we have ϕ we can compute $S_4 = 1/((1-\phi)+(\phi/4)) = 3$

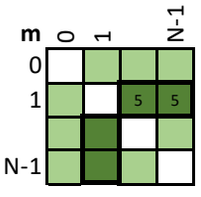
(c) Which is the value x in $region_1$?
Solución: From ϕ we can also compute the value of x since $\phi = (16+4x)/(24+4x)$. After some maths we get $x = 12$.

Problema 3 Tema 2

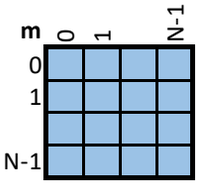
```
#define N 4
int m[N][N];

// 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");
```



task for_initialize (i=1, cost=30)

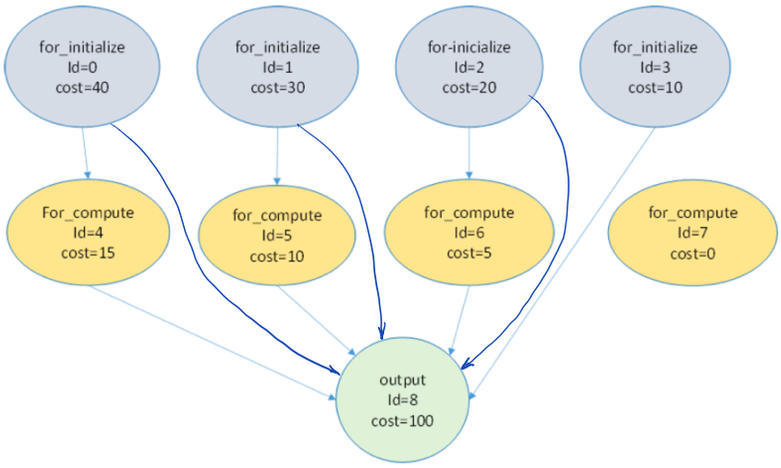


task for_compute (i=1, cost=10)



task print_results (cost=100)

(a) Draw the task dependence graph (TDG), indicating for each node its cost in terms of execution time (in time units).



Solution:

(b) Compute the values for T_1, T_∞, the parallel fraction (phi) as well as the potential parallelism.

Solution:

$T_1 = (40+30+20+10)+(15+10+5)+100 = 230$

$T_\infty = 40+15+100 = 155$

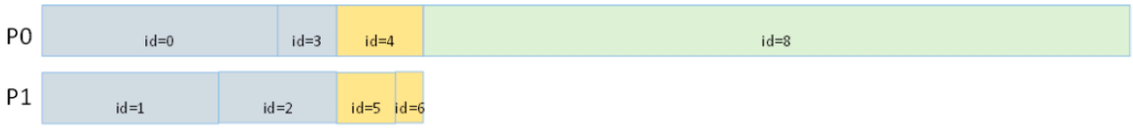
$phi = ((40+30+20+10)+(15+10+5))/230 = 130/230 = 0.57$

$Parallelism = 230/155 = 1.48$

Which would be the value for P_{min} for this TDG? 2? 3? 4? The value for P_{min} IS NOT 2, as you will see when asnwering the next question (S_2 < Parallelism). P_{min} IS NOT 4, as one could guess by using as many processors as màximum width of the graph, in this case it is not. It is Pmin is 3: one processor to execute the critical path (nodes 0-4-8), another to execute the next critical path (nodes 1-5) and another to execute the rest (2-3-6-7).

(c) Indicate which would be the most appropriate task assignment on two processors in order to obtain the best possible "speed up". Calculate T_2 and S_2.

Solution:



$T_2 = 40+10+15+100 = 165$

$S_2 = 230 \ / \ 165 = 1.39$

Observe that S_2 < Parallelism. The execution is well balanced since each processor executes half of the parallel fraction in the code.

Altres problemes pendents de fer

Problema 2 Tema 1

P (app1)	app1		app2	
	1200,0	P (app2)	2000,0	
0		8	250,0	
1	1200,0	7	285,7	1200,0
2	600,0	6	333,3	600,0
3	400,0	5	400,0	400,0
4	300,0	4	500,0	500,0
5	240,0	3	666,7	666,7
6	200,0	2	1000,0	1000,0
7	171,4	1	2000,0	2000,0
8	150,0	0		

Una execució no multiplexada també donaria el mateix temps d'execució, fent primer una aplicació amb 8 processadors i després l'altre també amb 8 processadors; però aquesta solució no compliria amb la condició especificada en l'enunciat del problema.