HPC: Collection of Exercises (with solutions)

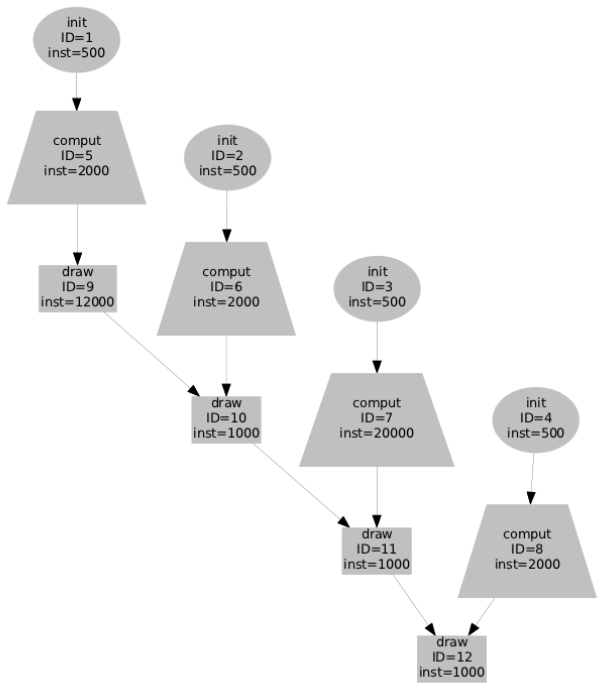
José Ramón Herrero and Miquel Àngel Senar

BDBI (6th term)

1

Understanding parallelism

1. Given the following Task Dependence Graph (TDG), in which each node represents a task with a name, an instance number *ID* and the cost of executing it in terms of number of instructions *inst*. Edges represent dependences between pairs of tasks.



   We ask:

   (a) Which is the value for the T_1, T_∞ and parallelism metrics for this TDG?

   Solution: $T_1 = 43000$; $T_\infty = 22500$ (definido por los nodos con ID=3, 7, 11 y 12. El paralelismo es el cociente $T_1/T_\infty = 1.91$

   (b) Which is the minimum number of processors P_{min} that are necessary to achieve the potential parallelism computed in the previous question?

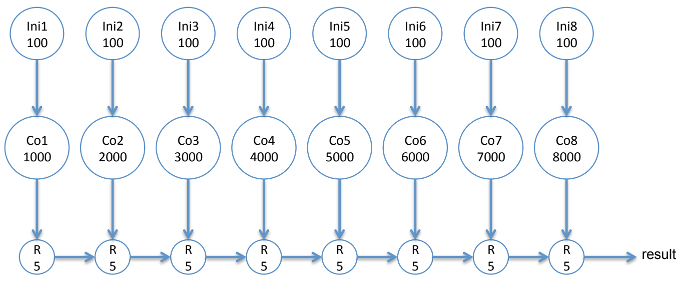
   Solution: El número mínimo de procesadores es 2. Uno de ellos ejecutaria el camino crítico (ID=3, 7, 11 y 12, con coste 22500) mientras que el otro ejecutaria el resto (ID=1, 2, 4, 5, 6, 8, 9 y 10, con coste 20500).

2. We want to execute the tasks of the TDG in the previous problem on 4 processors, each processor executing a task sequence `init-compute-draw` (for example, tasks 1, 5 and 9 on processor 0, tasks 2, 6 and 10 on processor 1, ...).

(a) Which is the speed-up that will be achieved when the tasks in this TDG are ideally executed on 4 processors?

(b) Assume that we are able to balance the work among processors, which means that each node 1-4 weights 500, each node 5-8 weights 6500, and each node 9-12 weights 3750 instructions. Which is the speed-up that would be achieved with 4 processors, assuming the same task assignment as before?

3. Given the following task dependence graph:

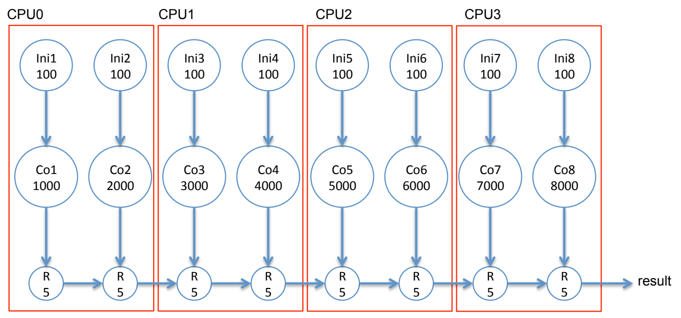


in which each node `IniX` runs for 100 time units, each node `CoY` runs for `Y*1000` time units and each node `R` runs for 5 time units. **We ask:**

(a) Calculate the values for T_1, T_∞ and the potential *Parallelism*.

Solution: $T_1 = 8 \times 100 + 1000 + 2000 + 3000 + 4000 + 5000 + 6000 + 7000 + 8000 + 8 \times 5 = 36840$
$T_\infty = 100 + 8000 + 5 = 8105$, determinado por el camino crítico `Ini8--Co8-R`
$Parallelism = T_1 \div T_\infty = 36840 \div 8105 = 4.54$

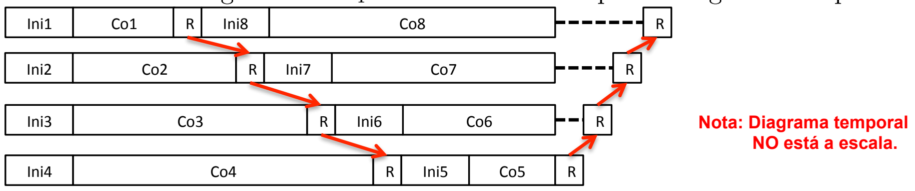
(b) Calculate T_4 and the "speed–up" S_4 in an architecture with 4 processors if tasks are mapped (assigned) to processors as shown below:



Solution: $T_4 = 2 \times 100 + 7000 + 8000 + 2 \times 5 = 15210$, determinado por CPU3, el resto se solapa totalmente.
$S_4 = T_1 \div T_4 = 36840 \div 15210 = 2.42$

(c) Determine the assignment of tasks to processors that would yield the best "speed-up" on 4 processors. Calculate such S_4.

Solution: En este grafo tenemos un problema de balanceo de carga, es decir, que la CPU3 tiene mucha más carga que el resto. Una asignación más balanceada sería `1--8`, `2--7`, `3--6` y `4--5`. Con esta asignacón el T_4 viene determinado por el diagrama temporal siguiente:



siendo $T_4 = 2 \times 100 + 9000 + 5 + 4 \times 5 = 9225$ y el speed–up $S_4 = 36840 \div 9225 = 3.99$

4. Given the following C code in which two types of tasks have been identified using an API to define tasks within *start_task* and *end_task*:

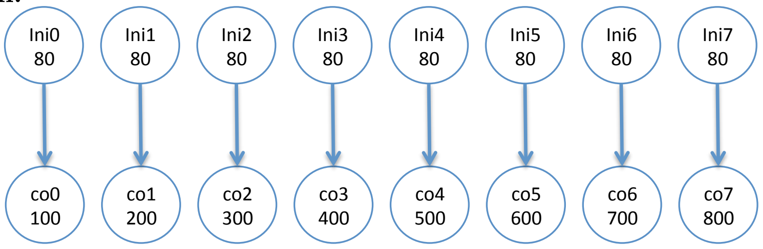
```
#define MAX 8
// initialization
for (outer = 0; outer < MAX; outer++) {
   start_task("for-initialize");
   for (inner = 0; inner < MAX; inner++)
      matrix[outer][inner] = inner;
   end_task("for-initialize");
   }

// computation
for (outer = 0; outer < MAX; outer++) {
   start_task("for-compute");
   for (inner = 0; inner <= outer; inner++)
      matrix[outer][inner] = matrix[outer][inner] + foo(outer,inner);
   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.

Solution:

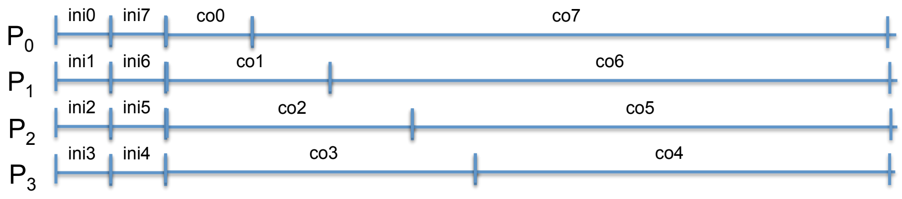


(b) Calculate the values for T_1 and T_∞ as well as the potential parallelism.

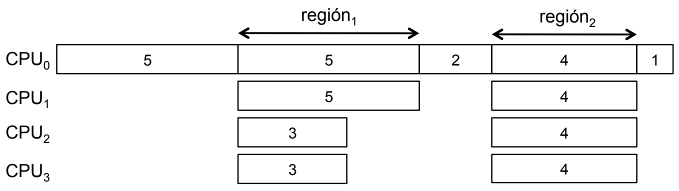
Solution: $T_1 = 4240$, $T_\infty = 880$, $parallelism = 4,81$

(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.

Solution: $S_4 = 4240/1060 = 4$, obtained with the following scheduling:



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



The application has two parallel regions and three sequential parts. In this diagram the numbers in the boxes represent the execution times of the different application bursts. **We ask**:

(a) Calculate the "speed-up" S_4 that is achieved in the execution with 4 processors.

Solution: The speed-up with 4 processors is the quotient between T_1 (5+16+2+16+1) and T_4 (5+5+2+4+1), i.e., $S_4 = 40/17 = 2.35$.

(b) Calculate the parallel fraction (ϕ) for the application that is represented in the time diagram above. Which is the "speed-up" that could be achieved when using infinite processors ($S_{p\to\infty}$, assuming that the two parallel regions can be ideally decomposed into the ∞)?

Solution:

The parallel fraction is computed as the ratio of the execution time coming from the part of the code that can potentially be executed in parallel with respect to the total sequential execution time, i.e., $\phi = (16+16)/40 = 0.8$. With this value of phi, the speed-up with infinite processors is $S_{p\to\infty} = 1/(1-\phi) = 5$.

(c) Imagine it is possible to remove the load imbalance that exists in "regiónl" at the expenses of adding an "overhead" that is proportional to the number of processors ($0.05 \times p$). Which would be the expression for S_p in this case?.

Solution: Knowing that $S_p = T_1/T_p$, we must find the expression for T_p. Thus, T_1=40 and $T_p = (5+2+1) + (16+16)/p + (0.05 \times p)$.

6. Given the following code in C:

```
int it_dot_product(int *X, int *Y, int n) {
   int i, sum=0;
   for (i=0; i<n; i++) sum += X[i]*Y[i];
   return sum;
}

int rec_dot_product(int *X, int *Y, int n) {
   int ndiv4 = n/4, sum1, sum2, sum3, sum4;

   if (n<=4) return it_dot_product(X,Y,n);
   sum1 = rec_dot_product(X, Y, ndiv4);
   sum2 = rec_dot_product(X+ndiv4, Y+ndiv4, ndiv4);
   sum3 = rec_dot_product(X+2*ndiv4, Y+2*ndiv4, ndiv4);
   sum4 = rec_dot_product(X+3*ndiv4, Y+3*ndiv4, n-3*ndiv4);
   return sum1+sum2+sum3+sum4;
}

void main() {
   int sum, X[N], Y[N];
   ...
   sum = rec_dot_product(X,Y,N);
   ...
}
```

Assume that there is not a parallelization strategy already defined for this application and that its sequential execution time is $T_{seq} = 6$ time units. Answer the following questions:

(a) Which should be the parallel fraction (ϕ) in order to achieve $S_{p\to\infty} = 100$?

Solution: We have to look for the ϕ that provides a $S_{p\to\infty} = 100$. Since $S_{p\to\infty} = 100 = \frac{1}{1-\phi}$ then we obtain $\phi = \frac{99}{100}$.

(b) Which should be the parallel fraction (ϕ) if we are looking for an $S_{p\to\infty} = 100$ but we introduce a constant overhead of 0.01 time units due to task creation?

Solution: This should be more than 99%, in particular $\frac{595\times100}{600}$%. We have to look for the ϕ that provides a $S_{p\to\infty} = 100$, considering that overhead.

$S_{p\to\infty} = 100 = \frac{T_{seq}}{T_{seq}(1-\phi)+0.01}$

$\phi = \frac{595}{600}$

2

Performance Model

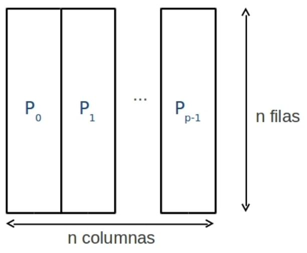
1. For each of these two loops:

```
1) for (i=1; i<n; i++)                    2) for (i=1; i<n; i++)
    for (j=1; j<n; j++) {                      for (j=1; j<n; j++) {
     B[i][j] = A[i][j-1]+A[i-1][j]+A[i][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) Assuming that we are using a distributed memory machine and that we are doing a distribution of the matrices A and B by columns, as shown below:



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

Solution:

(a) i. $T_1 = (n-1) \times (n-1) \times t_c \approx n^2 \times t_c$
$T_\infty = t_c$

ii. $T_1 = (n-1) \times (n-1) \times t_c \approx n^2 \times t_c$
$T_\infty = (n-1) + (n-2) \times t_c = (2 \times n - 3) \times t_c \approx 2 \times n \times t_c$

(b) i. Computations can proceed fully in parallel after some initial comunication, namely sending the last column of matrix A of processor P_i to the processor P_{i+1} (except for processor P_{P-1} that does not send its last column to any other processor). Then:
$T_p = (n-1) \times (n-1)/P \times t_c + 1 \times (t_s + (n-1) \times t_w) \approx n^2/P \times t_c + (t_s + n \times t_w)$

ii. Requires blocking to avoid sequentialization. Let us define a block of B rows within the set of columns given to a processor. Then, assuming n is large we take $n \approx n-1$, and:
$T_p \approx (n/B + P - 1) \times (B \times n/P) \times t_c + (n/B + P - 2) \times (t_s + B \times t_w)$

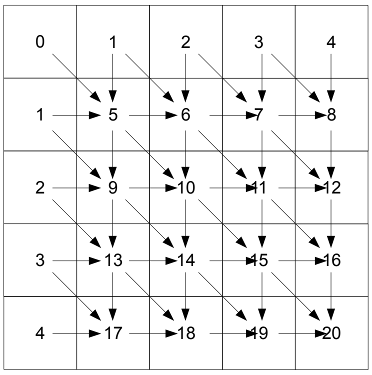
2. Assume a distributed memory machine and a message passing model where the message cost is $t_{comm} = t_s + m \times t_w$, being t_s the "start-up" time and t_w the transfer time of a word. Answer the following questions related to the following code:

```
void smith-waterman(int h[N+1][N+1], char a[N], char b[N], int sim[20][20]) {
    int i,j;
    int diag, down, right;

    for (i=0;i<=N;i++) {
      h[0][i]=0;
      h[i][0]=0;
    }
    for (i=1;i<=N;i++)
      for (j=1;j<=N;j++) {
        diag    = h[i-1][j-1] + sim[a[i-1]][b[j-1]];
        down    = h[i-1][j] + 4;
        right   = h[i][j-1] + 4;
        h[i][j] =  MAX4(diag,down,right,0);
      }
}
```

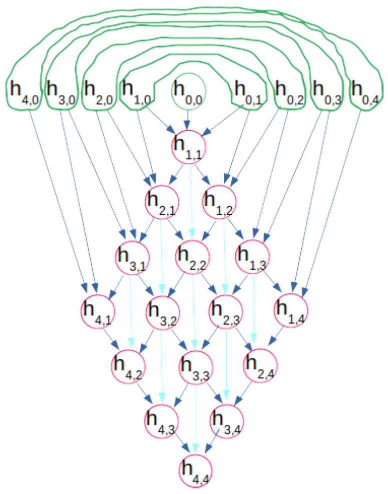
(a) Draw matrix `h` and the data dependences between the computation of its elements (i.e. for each element indicate which elements of the same matrix need to be computed before).

Solution: The figure below shows the case for $N = 4$. The numbers identify the tasks. The first row and column share the same numbering since each task in the initialization loop sets a value in the first row and a value in the first column.



(b) Assuming the following definitions for tasks: 1) each iteration of the first initializing to zero loop (cost of an iteration $t_{zero} = 1$ time unit) and 2) each iteration of the most internal loop in the nested loop (cost of an iteration $t_c = 100$ time units):

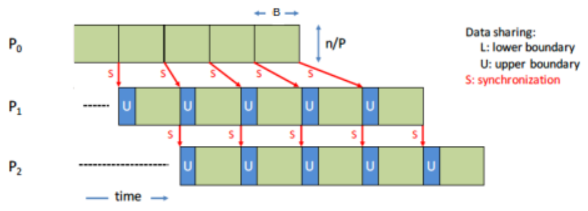
- Draw the task dependence graph (TDG) between tasks for $N = 4$.
Solution:



- Obtain the expressions for T_1 and T_∞ as a function of N. How many processors P_{min} are needed to guarantee that T_∞ can be reached?
  Solution:
  $T_1 = (N + 1) \times t_{zero} + (N \times N) \times t_c$
  $T_\infty = t_{zero} + (2 \times N - 1) \times t_c$
  To get T_∞ we should have at least N processors

(c) Assuming that the vector `a`, the vector `b`, and the matrix `sim` have been replicated in P processors, and the matrix `h` is distributed by rows between these P processors ($\frac{N}{P}$ consecutive rows per processors). Which would be the most appropriate definition for the tasks for the computation loop in the program, with the objective of reducing T_p? Obtain the expression for T_p for that loop. You can assume that $N >> P$.

Solution: The best task definition involves blocking: each task will work on a block of $(N/P) \times (B)$ iterationes of loops i and j, respectively. It is necessary to apply blocking to loop j in order to avoid a sequential execution.



Thus, T_p can be approximated with:
$T_p = (N/B + P - 1) \times (N/P \times B) \times t_c + (N/B + P - 2) \times (t_s + B \times t_w)$

3. We want to find the expression that determines the parallel execution time on p processors (T_p) for the following code:

```
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.

		Column distribution	Row distribution
Task definition			
Initial remote accesses	Total number of messages		
	Size of each message		
	Contribution to T_p		
Parallel computation	Total number of tasks		
	Size of each task		
	Contribution to T_p		
Remote accesses during parallel execution	Total number of messages		
	Size of each message		
	Contribution to T_p		

Solution:

Using a column distribution the best task definition would assign n/p consecutive iterations of loop k to each processor (and all iterations of loop i). In this way: 1) the dependency caused by the access `u[i-1][k]` is internal to the execution of each processor and does not require communication; 2) the access `u[i][k+1]` does not cause a dependency but requires an initial communication in order to receive the whole 1^{st} column in the processor to the right (all processes but the rightmost one).

Using a row distribution the best task definition would work on a block of n/p x B consecutive iterations of loops i and k, respectively. In this way:, 1) in spite of the dependency caused by the access `u[i-1][k]`, *blocking* allows for the overlap of the execution of blocks (made of B columns) in several processors, impliying the communication of B elements in the last row within the block); 2) the access `u[i][k+1]` does not imply communication since all data in row `i` are in the same processor.

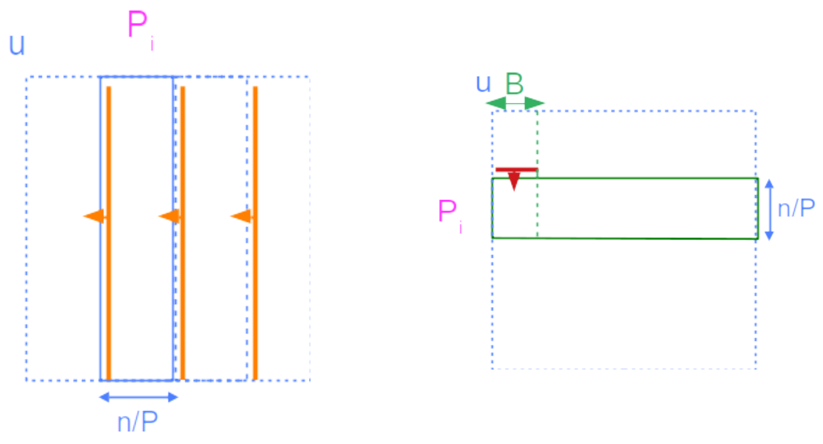


Figure 2.1: *
Task definition and communications for Column (left) and row (right) distributions

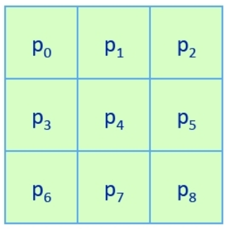
		Column distribution	Row distribution
Task definition		n iterations of loop i and $n \div p$ iterations of loop k	$(n \div p) \times B$ iterations of loops i and k
Initial remote accesses	Total number of messages	$p - 1$	–
	Size of each message	$\approx n$	–
	Contribution to T_p	$t_s + n \times t_w$	0
Parallel computation	Total number of tasks	p	$p \times (n \div B)$
	Size of each task	$(n \div p) \times n$	$(n \div p) \times B$
	Contribución a T_p	$(n^2 \div p) \times t_c$	$((n \div B) + p - 1) \times ((n \div p) \times B) \times t_c$
Remote accesses during parallel execution	Total number of messages	–	$(p - 1) \times (n \div B)$
	Size of each message	–	$\approx B$
	Contribution to T_p	0	$((n \div B) + p - 2) \times (t_s + B \times t_w)$

4. Assume a distributed memory architecture with message passing, where each message of m elements incurs an overhead of $t_{comm} = t_s + m \times t_w$.

   (a) Given the following loop:

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

and a 2D block distribution for matrices u and f, as shown for the following case with 9 processors (3^2):



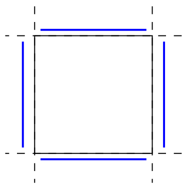
Assume 1) a task is defined as a block of $N \div P$ by $N \div P$ consecutive iterations of the `i` and `k` loop, respectively; 2) t_c the execution time of the innermost loop body; 3) P^2 the number of processors; and 4) n the size of the matrix (n rows by n columns), being n large compared to P. With this 2D block distribution, each processor will execute one task. Answer the following questions:

i. Indicate the data that each processor should receive from other processors, and when it has to receive such data.

ii. Each processor could start its computation when it has all the necessary data in its local memory but we will consider that the computation starts once all the communications in each step are completed. Obtain the model of execution time (computation and communication).

Solution: In all questions: Since n is large we consider $n-2 \approx n$

i. Data received: Each processor will need to receive the boundary elements from its neighbour processors. The boundary consists of n/P elements in the first/last column/—row in the block. Since the source matrix is not modified, all the boundaries can be communicated initially.



ii. $T_{P^2} = \underbrace{4 \times (t_s + \frac{n}{P} \times t_w)}_{T_{Initial\_Comm}} + \underbrace{\frac{n^2}{P^2} \times t_c}_{T_{Comp}}$

(b) Continuing with the same assumptions above and same task definition, consider next the following loop:

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

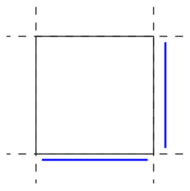
i. Indicate the data that each processor should receive from other processors, and when it has to receive such data.

ii. Each processor could start its computation when it has all the necessary data in its local memory but we will consider that the computation starts once all the communications in each step are completed. Obtain the model of execution time (computation and communication).

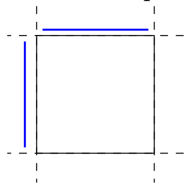
Solution:

i. Each processor will need to receive the boundary elements from its neighbour processors. The boundary consists of n/P elements in the first/last column/row in the block. Since the source matrix is modified, we need to distinguish the boundaries that can be communicated initially from those that need to be sent only once they have been modified, respecting data dependencies.

Initially:
the boundaries received from the right and the bottom can be communicated.



During the execution of the loops:
the boundaries received from the left and the top can be communicated after the corresponding blocks have been updated.



ii. $T_{P^2} = \underbrace{2 \times (t_s + \frac{n}{P} \times t_w)}_{T_{Initial\_Comm}} + \underbrace{(2P-1) \times \frac{n^2}{P^2} \times t_c}_{T_{Comp}} + \underbrace{(2P-2) \times 2 \times (t_s + \frac{n}{P} \times t_w)}_{T_{Other\_Comm}}$