Dado el siguiente código secuencial en C que encuentra en un vector DB la primera posición en la que aparece una determinada clave key:

```
int main() {
    double key = 1.25;
    double * DB = (double *) malloc(sizeof(double) * DBsize);
    initialize(DB, &DBsize); // initialize elements in DB
    unsigned long position = DBsize;
    for (unsigned long i = 0; (i < DBsize) && (position == DBsize); i++)
        if (DB[i] == key) position = i;
}
```

Y la siguiente solución incompleta para la paralelización del bucle for:

```
#pragma omp parallel
{
 unsigned long i, num_elems, lower;
 for (i = lower; (i < (lower + num_elems)) && (i < position); i++) {
     #pragma omp critical
      if ((DB[i] == key) && ( i < position)) position = i;
 }
}
```

```
#pragma omp parallel
{
 unsigned long i, BS, lower;
 for (i = lower; (i < (lower + BS)) && (i < position); i++) {
     #pragma omp critical
      if ((DB[i] == key) && (i<position)) position = i;
 }
}
```

Se pide:

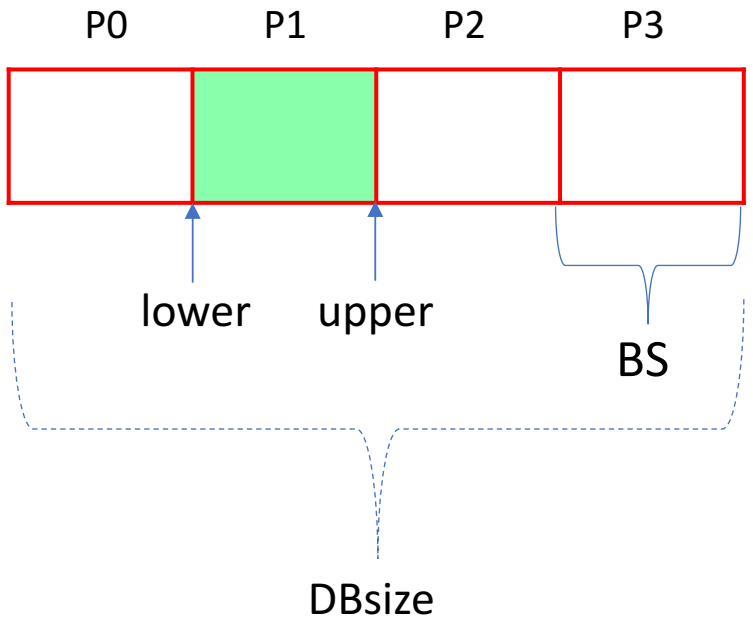
(a) Completad la solución incompleta anterior (con las sentencias y declaraciones de datos necesarias) para que cumpla con las siguientes condiciones, sin preocuparse por posibles problemas de rendimiento: 1) el reparto de iteraciones a procesadores obedezca a una descomposición de datos geométrica tipo BLOCK (es decir, a cada procesador se le asocian `DBsize/P` elementos consecutivos, siendo `P` el número de procesadores); 2) `P` no tiene por que dividir de forma entera `DBsize`, en cuyo caso se deberá maximizar el balanceo de carga; 3) la solución debe permitir que un procesador finalice su ejecución tan pronto encuentre `key` o detecte que no contribuirá a la solución final; y 4) no puede utilizarse el `#pragma omp for` para realizar el reparto de iteraciones.

(b) Modificad el código anterior para que se mejore el rendimiento de forma substancial, reduciendo al mínimo la sequencialización que introduce la sincronización actual.

(c) Proponed e implementad una descomposición de datos geométrica alternativa que reduzca el tiempo de ejecución requerido, en media, para encontrar la primera posición en la que aparece la clave `key`.

a) Input BLOCK Geometric Data Decomposition:

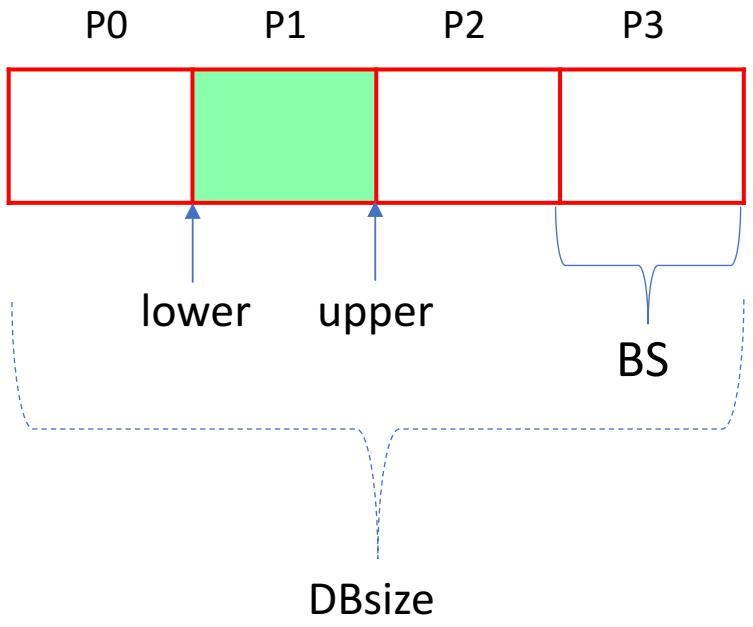
- Block size = DBsize / P
- (DBsize % P) can be > 0
- Execution at each processor must end as soon as possible
- You cannot use *omp for*




```
#pragma omp parallel
{
 int myid = omp_get_thread_num ();
 int P = omp_get_num_threads ();
 …
 unsigned long lower =
 unsigned long upper =
 …
 unsigned long i;
 for (i = lower; ( … ); i++) {
     #pragma omp critical
      if (DB[i] == key && i < position)
              position = i;
 }
}
```

a) Input BLOCK Geometric Data Decomposition:

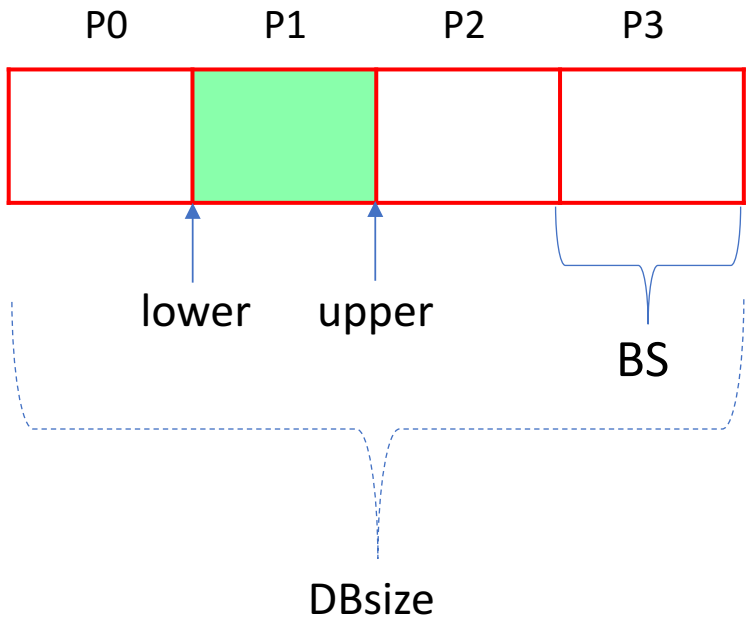
- **Block size = DBsize / P**
- (DBsize % P) can be > 0
- Execution at each processor must end as soon as possible
- You cannot use *omp for*




```
#pragma omp parallel
{
 int myid = omp_get_thread_num ();
 int P = omp_get_num_threads ();
 unsigned long BS = DBsize / P;
 unsigned long lower = myid * BS;
 unsigned long upper = lower + BS;
 …
 unsigned long i;
 for (i = lower; (i < upper ); i++) {
     #pragma omp critical
      if (DB[i] == key && i < position)
              position = i;
 }
}
```

a) Input BLOCK Geometric Data Decomposition:

- Block size = DBsize / P
- (DBsize % P) can be > 0
- **Execution at each processor must end as soon as possible**
- You cannot use *omp for*

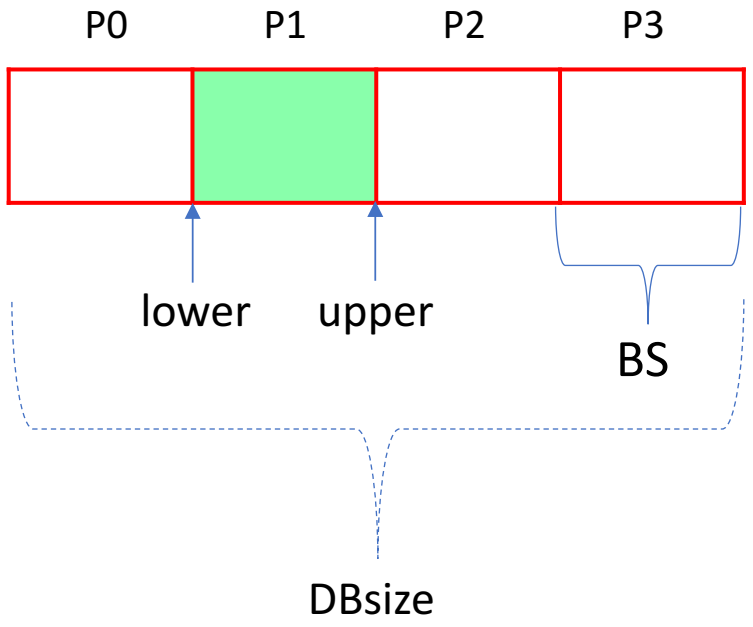



```
#pragma omp parallel
{
 int myid = omp_get_thread_num ();
 int P = omp_get_num_threads ();
 unsigned long BS = DBsize / P;
 unsigned long lower = myid * BS;
 unsigned long upper = lower + BS;
 …
```

```
 unsigned long i;
 for (i = lower; (i < upper && i < position); i++) {
     #pragma omp critical
      if (DB[i] == key && i < position)
              position = i;
 }
}
```

a) Input BLOCK Geometric Data Decomposition:

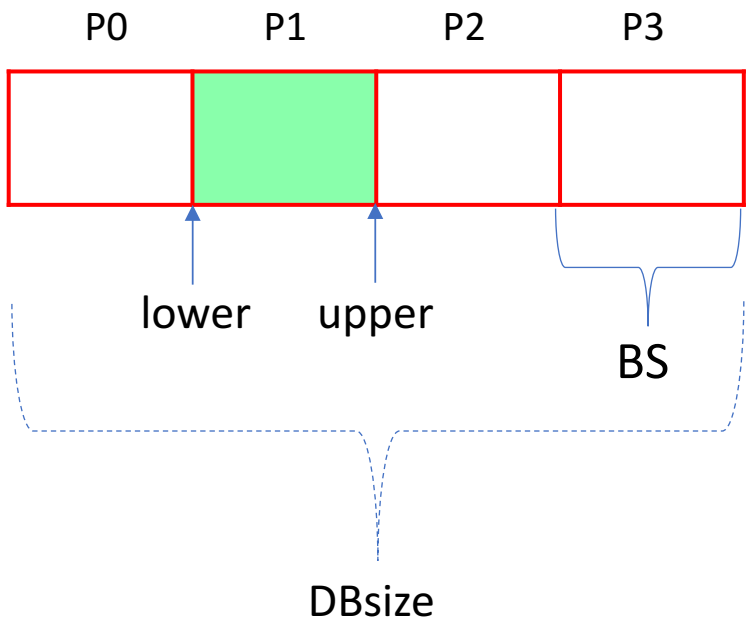
- Block size = DBsize / P
- **(DBsize % P) can be > 0**
- Execution at each processor must end as soon as possible
- You cannot use *omp for*




```
#pragma omp parallel
{
 int myid = omp_get_thread_num ();
 int P = omp_get_num_threads ();
 unsigned long BS = DBsize / P;
 unsigned long lower = myid * BS;
 unsigned long upper = lower + BS;
 unsigned long mod = DBsize % P;
 if (mod > 0)
        …
 }



 unsigned long i;
 for (i = lower; (i < upper && i < position); i++) {
     #pragma omp critical
      if (DB[i] == key && i < position)
              position = i;
 }
}
```

a) Input BLOCK Geometric Data Decomposition:

- Block size = DBsize / P
- **(DBsize % P) can be > 0**
- Execution at each processor must end as soon as possible
- You cannot use *omp for*



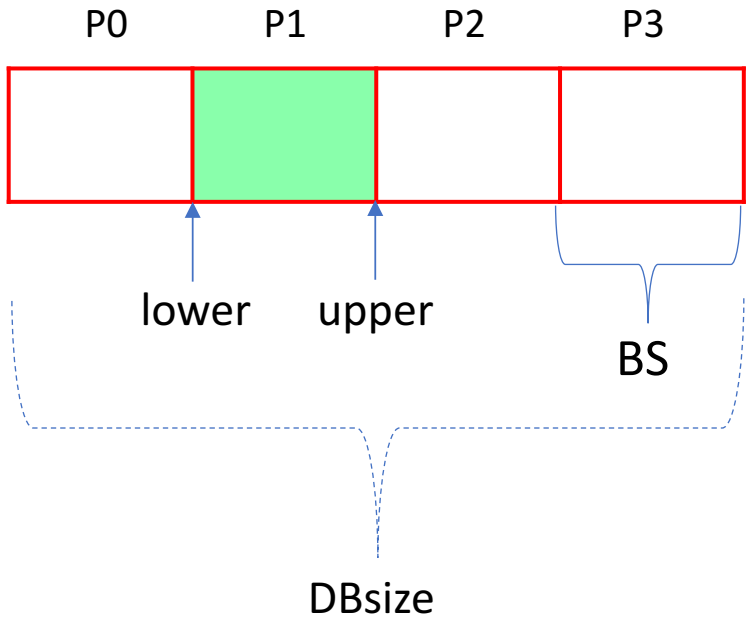

```
#pragma omp parallel
{
 int myid = omp_get_thread_num ();
 int P = omp_get_num_threads ();
 unsigned long BS = DBsize / P;
 unsigned long lower = myid * BS;
 unsigned long upper = lower + BS;
 unsigned long mod = DBsize % P;
 if (mod > 0)
        …
 }
```

EJEMPLO:

		Total iterations = 15							
		Total number threads = 4							
	myid	iters							
	0	0	1	2					
	1	3	4	5					
	2	6	7	8					
	3	9	10	11	12	13	14		mod=3
	myid	iters							
id < mod	0	0	1	2	3				
	1	3	4	5	6	7			
	2	6	7	8	9	10	11		
id >= mod	3	9	10	11	12	13	14	x	

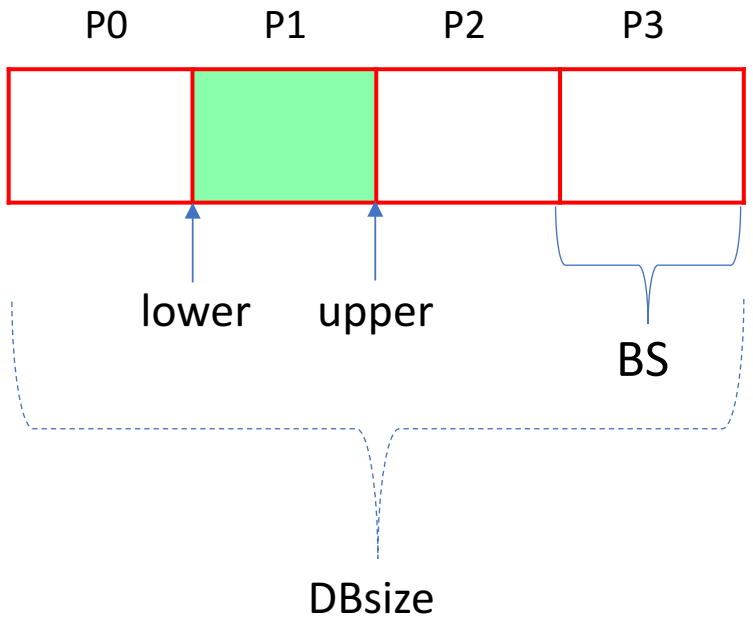
a) Input BLOCK Geometric Data Decomposition:

- Block size = DBsize / P
- **(DBsize % P) can be > 0**
- Execution at each processor must end as soon as possible
- You cannot use *omp for*



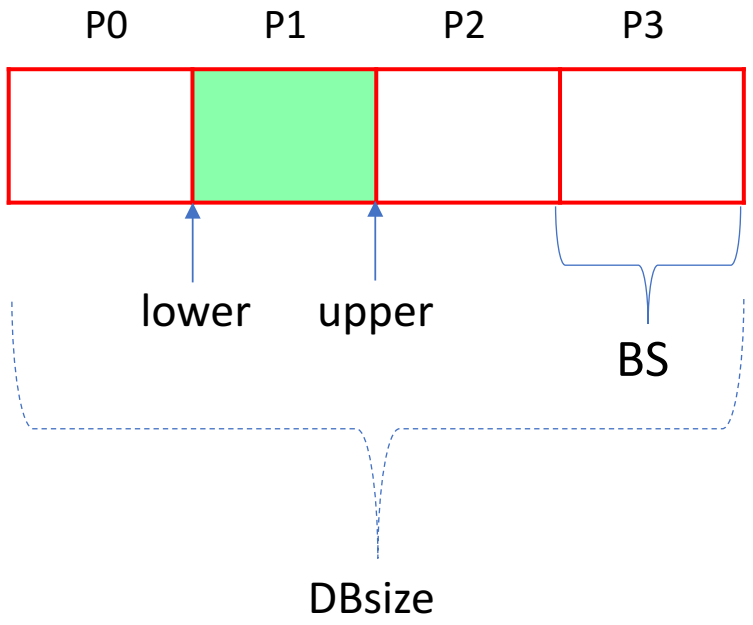

```
#pragma omp parallel
{
 int myid = omp_get_thread_num ();
 int P = omp_get_num_threads ();
 unsigned long BS = DBsize / P;
 unsigned long lower = myid * BS;
 unsigned long upper = lower + BS;
 unsigned long mod = DBsize % P;
 if (mod > 0)
       if (myid < mod) {
              lower += myid; upper += myid + 1;
       } else {
              lower += mod; upper += mod;
       }
 }
 unsigned long i;
 for (i = lower; (i < upper && i < position); i++) {
     #pragma omp critical
      if (DB[i] == key && i < position)
             position = i;
 }
}
```

b) Optimize the code by reducing the serialization within the loop.



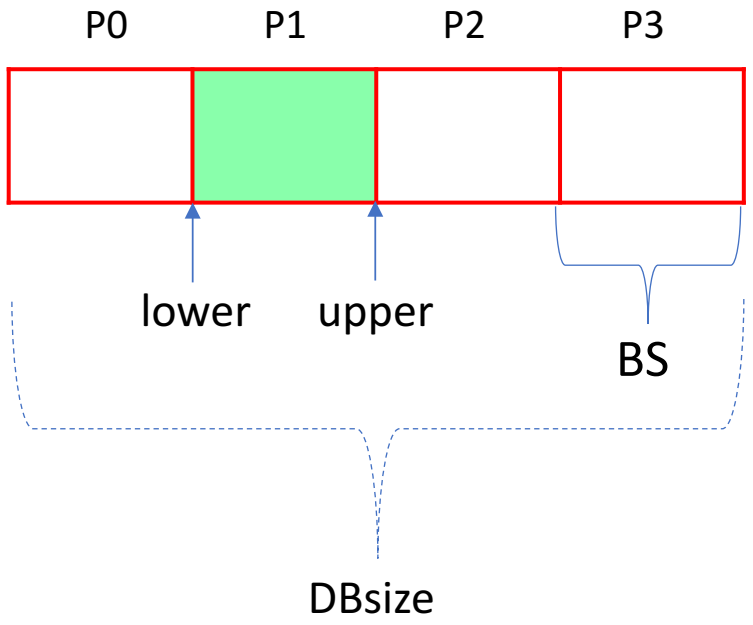

```
#pragma omp parallel
{
 int myid = omp_get_thread_num ();
 int P = omp_get_num_threads ();
 unsigned long BS = DBsize / P;
 unsigned long lower = myid * BS;
 unsigned long upper = lower + BS;
 unsigned long mod = DBsize % P;
 if (mod > 0)
       if (myid < mod) {
              lower += myid; upper += myid + 1;
       } else {
              lower += mod; upper += mod;
       }
 }
 unsigned long i;
 for (i = lower; (i < upper && i < position); i++) {
     #pragma omp critical
      if (DB[i] == key && i < position)
             position = i;
 }
}
```

b) Optimize the code by reducing the serialization within the loop.




```
#pragma omp parallel
{
 int myid = omp_get_thread_num ();
 int P = omp_get_num_threads ();
 unsigned long BS = DBsize / P;
 unsigned long lower = myid * BS;
 unsigned long upper = lower + BS;
 unsigned long mod = DBsize % P;
 if (mod > 0)
       if (myid < mod) {
              lower += myid; upper += myid + 1;
       } else {
              lower += mod; upper += mod;
       }
 }
 unsigned long i;
 for (i = lower; (i < upper && i < position); i++) {
     #pragma omp critical
      if (DB[i] == key && i < position)
             position = i;
 }
}
```

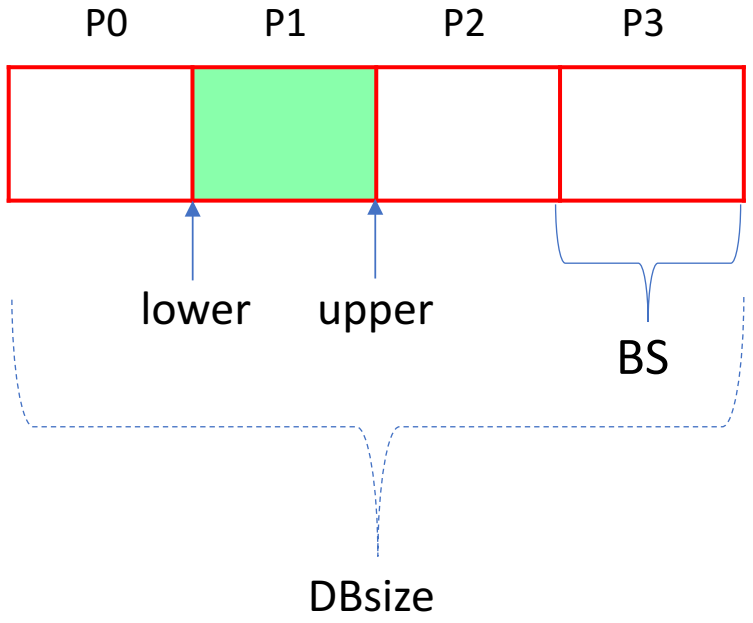
b) Optimize the code by reducing the serialization within the loop.



Filter the cases where it is known before entering the critical section that the if condition will evaluate to FALSE

```
#pragma omp parallel
{
 int myid = omp_get_thread_num ();
 int P = omp_get_num_threads ();
 unsigned long BS = DBsize / P;
 unsigned long lower = myid * BS;
 unsigned long upper = lower + BS;
 unsigned long mod = DBsize % P;
 if (mod > 0)
       if (myid < mod) {
              lower += myid; upper += myid + 1;
       } else {
              lower += mod; upper += mod;
       }
 }
 unsigned long i;
 for (i = lower; (i < upper && i < position); i++) {
     #pragma omp critical
      if (DB[i] == key && i < position)
              position = i;
 }
}
```

b) Optimize the code by reducing the serialization within the loop.



Filter the cases where it is known before entering the critical section that the if condition will evaluate to FALSE

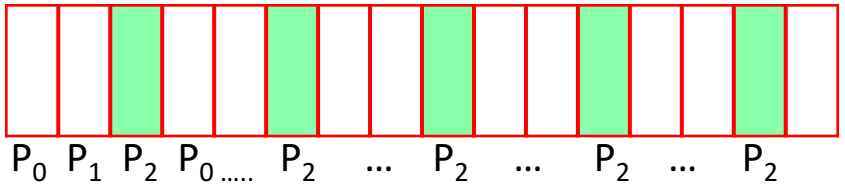
```
#pragma omp parallel
{
 int myid = omp_get_thread_num ();
 int P = omp_get_num_threads ();
 unsigned long BS = DBsize / P;
 unsigned long lower = myid * BS;
 unsigned long upper = lower + BS;
 unsigned long mod = DBsize % P;
 if (mod > 0)
       if (myid < mod) {
              lower += myid; upper += myid + 1;
       } else {
              lower += mod; upper += mod;
       }
 }
 unsigned long i;
 for (i = lower; (i < upper && i < position); i++) {
      if (DB[i] == key)
              #pragma omp critical
              if (i < position)
                     position = i;
 }
}
```

c) Write an alternative Geometric Data Decomposition that minimizes the average execution time.

```
#pragma omp parallel
{
 int myid = omp_get_thread_num ();
 int P = omp_get_num_threads ();
 unsigned long i;

 for (i = … ; (i < …  && i < position); i …) {
      if (DB[i] == key)
            #pragma omp critical
            if (i < position)
                  position = i;
 }
}
```

Find as soon as possible the first position i that verifies DB[i] == key …

c) Write an alternative Geometric Data Decomposition that minimizes the average execution time.



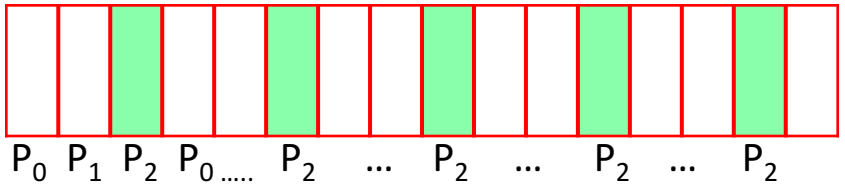

```
#pragma omp parallel
{
 int myid = omp_get_thread_num ();
 int P = omp_get_num_threads ();
 unsigned long i;

 for (i = …; (i < … && i < position); i…) {
      if (DB[i] == key)
            #pragma omp critical
            if (i < position)
                  position = i;
 }
}
```

Find as soon as possible the first position i that verifies DB[i] == key …

Data decomposition so that every thread has the same chance to find it:

Input Cyclic Geometric Data Decomposition

c) Write an alternative Geometric Data Decomposition that minimizes the average execution time.




```
#pragma omp parallel
{
 int myid = omp_get_thread_num ();
 int P = omp_get_num_threads ();
 unsigned long i;

 for (i = myid; (i < DBsize && i < position); i+= P) {
      if (DB[i] == key)
            #pragma omp critical
            if (i < position)
                  position = i;
 }
}
```

Find as soon as possible the first position i that verifies DB[i] == key …

Data decomposition so that every thread has the same chance to find it:

Input Cyclic Geometric Data Decomposition