Ejemplo 1

Versión 1: Tareas implícitas

```
void doComputation (int * vector, int n)
{
     #pragma omp parallel
     {
          int whoAmI = omp_get_thread_num();
          int howmany = omp_get_num_threads();
          int size = n/howmany;
          for(int i = whoAmI*size; i < (whoAmI+1)*size; ++i)
          {
               compute(&vector[i]);
          }
     }
}
```

Versión 2: Tareas explícitas usando “task”

```
void doComputation (int * vector, int n)
{
     for(int i = 0; i < n; ++i)
          #pragma omp task
          compute(&vector[i]);
}
…
void main(){
     #pragma omp parallel
     #pragma omp single
     partition (vector, N);
}
```

Versión 3: Tareas explícitas usando “task” con control de granularidad

```
void doComputation(int * vector, int n)
{
     int BS = …;
     for(int ii = 0; ii<n; ii+=BS)
          #pragma omp task
          for(int i = ii; i < min((ii+BS), n); ++i)
               compute(&vector[i]);
}
```

Versión 4: Tareas explícitas con taskloop

```
void doComputation(int *vector, int n){
     int BS = …;
     #pragma omp taskloop grainsize(BS)
     for (int i = 0; i < n; i++){
          compute (&vector[i]);
}
```

Ejemplo 2:

```
void doComputation(int * vector, int n) {
    for(int i = 0; i < n; ++i)
        result += compute(&vector[i]);
}
```

Versión 1: Atomic

```
void doComputation(int * vector, int n) {
    #pragma omp taskloop
    for (int i = 0; i < n;  i++)
        #pragma omp atomic
        result += compute(&vector[i]);
}
```

Versión 2: Reduction

```
void doComputation (int * vector, int n) {
    #pragma omp taskloop reduction(+: result)
    for (int i = 0; i < n; i++)
        result += compute (&vector[i]);
}
```

Versión 3: Critical

```
void doComputation(int * vector, int n) {
    #pragma omp taskloop
    for (int i = 0; i < n;  i++)
        #pragma omp critical
        if (vector[i] < result)
            result += compute(&vector[i]);
}
```

Versión 4: Critical optimizado

```
void doComputation(int * vector, int n) {
    #pragma omp taskloop
    for (int i = 0; i < n;  i++)
        #pragma omp critical
        if (vector[i] < result)
            result += compute(&vector[i]);
}
```

Versión 5: Reduction II

```
void doComputation (int * vector, int n) {
    #pragma omp taskloop reduction(min: result)
    for (int i = 0; i < n; i++)
        if (vector[i] < result)
            result = vector[i];
}
```

Versión 6: “named”critical

```
void doComputation(int * vector, int n) {
    #pragma omp taskloop
    for (int i = 0; i < n;  i++)
        if (vector[i]%2)
        #pragma omp critical (parallel)
        compute(result.even);
        else
        #pragma omp critical (senar)
            compute(result.odd);
}
```

Ejemplo 3:

```
#define SIZE_TABLE 1048576
typedef struct {
    int data;
    element * next;
} element;

element * HashTable[SIZE_TABLE];

void doComputation(int * vector, int n) {
    for (i = 0; i < n; i++) {
        int index = hash_function(vector[i]);
        insert_element(vector[i], index);
    }
}
```

Versión 1: Critical

```
void doComputation(int * vector, int n) {
     #pragma omp taskloop
     for (i = 0; i < n; i++){
        int index = hash_function(vector[i]);
        #pragma omp critical
        insert_element(vector[i], index);
    }
}
```

Versión 2: Con locks

```
omp_lock_t hash_lock[SIZE_TABLE];
void doComputation(int * vector; int n)
{
     #pragma omp taskloop
     for (i=0; i < n; i++)
     {
          int index = hash_func(vector[i]);
          omp_set_lock(&hash_lock[index]);
          insert_element(vector[i], index);
          omp_unset_lock(&hash_lock[index]);

     }
}

void main()
{
     for(i = 0; i < HASH_TABLE_SIZE; ++i)
          omp_init_lock(&hash_lock[i]);

     #pragma omp parallel
     #pragma omp single
     partition(vector, N);
     for(i = 0; i < SIZE_TABLE; ++i)
          omp_destroy_lock(&hash_lock[i]);

}
```