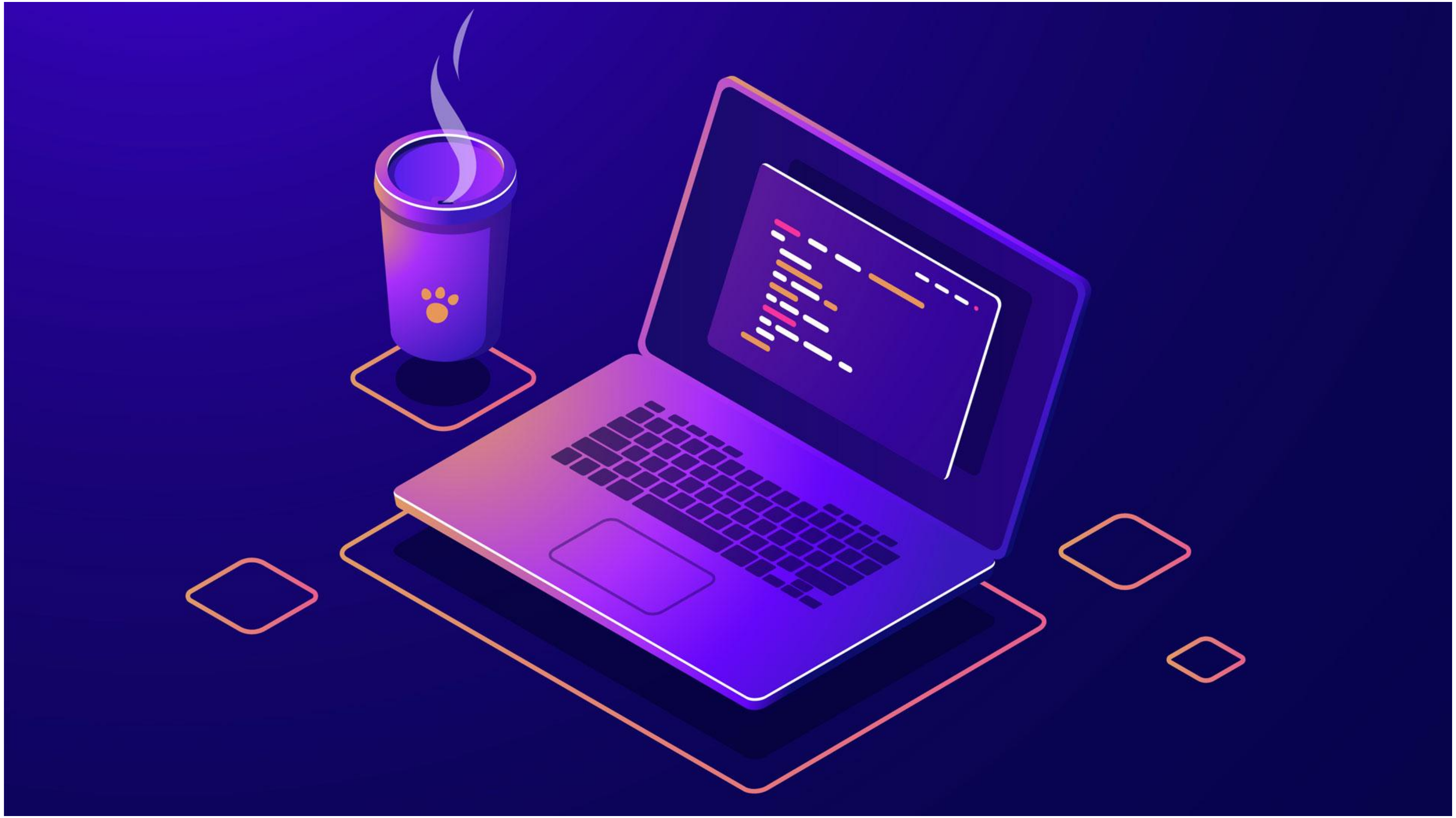
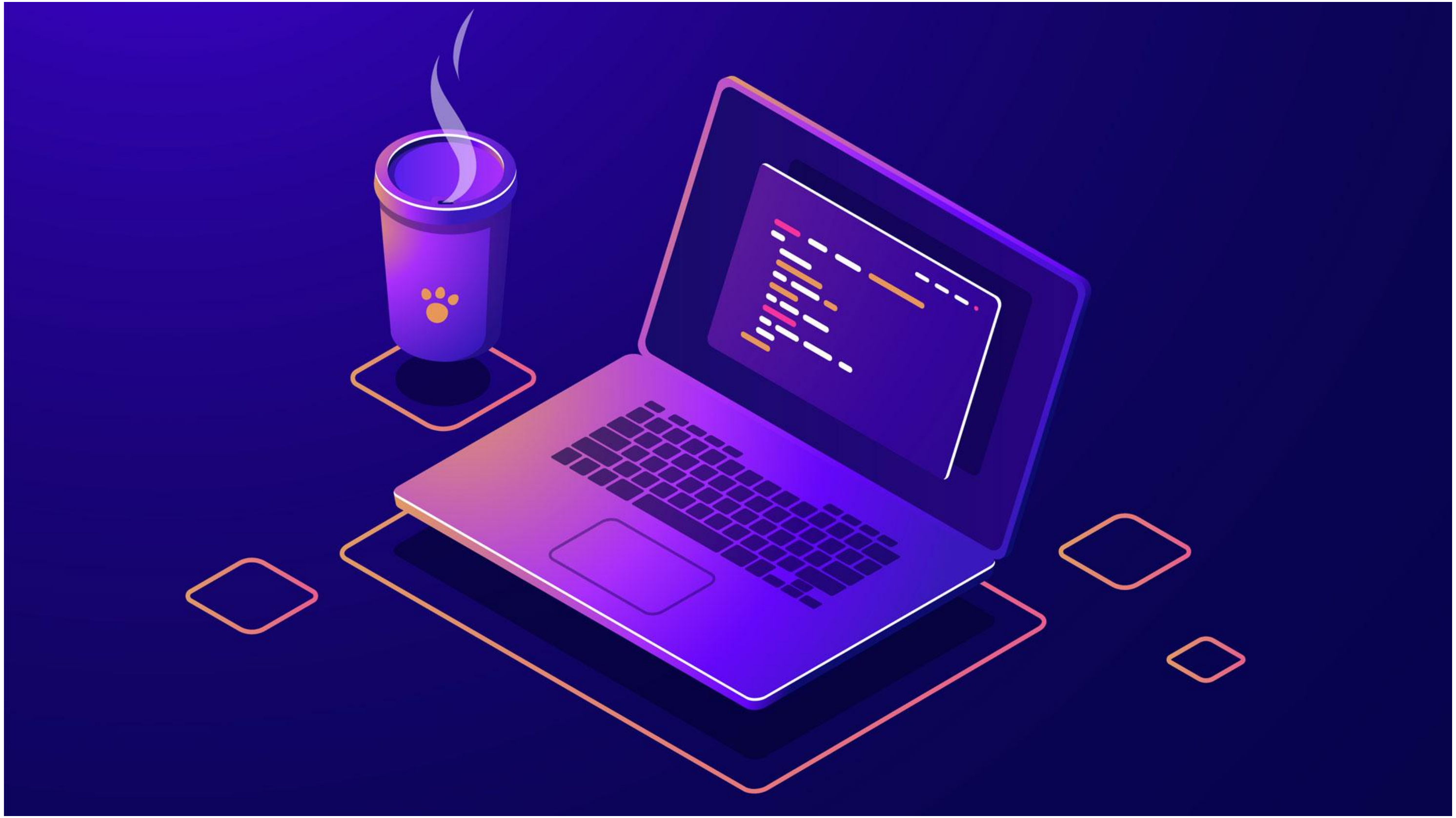
OpenMP Problems



Problem 2 - Point C



Problem 2

The following sequential code in C finds all positions in vector `DBin` in which a set of keys (contained in vector keys) appear. Positions where keys appear are stored in a new vector `DBout` (the order in `DBout` of the positions found is irrelevant).

```
int main() {
    double keys[nkeys], DBin[DBsize], DBout[nkeys][DBsize];
    unsigned int i, k, counter[nkeys];

    getkeys(keys, nkeys);                   // get keys
    init_DBin(DBin, DBsize);                // initialize elements in DBin
    clear_DBout(DBout, nkeys, DBsize);  // initialize elements in DBout
    clear_counter(counter, nkeys);      // initialize counter to zero

    for (i = 0; i < DBsize; i++)
        for (k = 0; k < nkeys; k++)
            if (DBin[i] == keys[k]) DBout[k][counter[k]++] = i;
}
```

Problem 2

```
#define DBsize 1048576
#define nkeys 16 // the number of processors can be larger than the number of keys

 int main() {
     double keys[nkeys], DBin[DBsize], DBout[nkeys][DBsize];
     unsigned int i, k, counter[nkeys];

     getkeys(keys, nkeys);                     // get keys
     init_DBin(DBin, DBsize);                  // initialize elements in DBin
     clear_DBout(DBout, nkeys, DBsize);        // initialize elements in DBout
     clear_counter(counter, nkeys);            // initialize counter to zero

     for (i = 0; i < DBsize; i++)
         for (k = 0; k < nkeys; k++)
             if (DBin[i] == keys[k]) DBout[k][counter[k]++] = i;
 }
```

Problem 2

(a) Write a first *OpenMP* parallelisation that implements an **iterative task decomposition strategy** of the outermost loop `i`, making use of the `taskloop` directive, in which you minimise the serialisation introduced by the synchronisation that you may introduce. **Note:** you are not allowed to change the structure of the two loops.

(b) Write a second *OpenMP* parallelisation that also implements an **iterative task decomposition strategy**, but this time applied to the innermost loop `k`, again making use of the `taskloop` directive, in which you maximise the parallelism that can be exploited. **Notes:** 1) `taskloop` has an implicit `taskgroup` synchronisation that you can omit with the `nogroup` clause; 2) observe that the number of keys is not large when compared to the possible number of processors to use; and 3) you are not allowed to change the structure of the two loops.

(c) Finally, write a third *OpenMP* parallelisation that implements a `task`–based **recursive divide–and–conquer decomposition strategy**, with the following requirements: 1) the recursion splits the input vector `DBin` in two almost identical halves, with a base case that corresponds to checking a single element of `DBin`; 2) uses a **cut–off strategy based on the size of the input vector**, so that tasks are only generated while that size is larger than `CUT_SIZE`; 3) only uses *OpenMP* pragmas and clauses for the implementation of the cut–off strategy; and 4) you have to use the synchronisation mechanism, if needed, that maximises the parallelism in the program.

Mutual Exclusion

Mutual exclusion: mechanism to ensure that only one task at a time executes the code within a region.

Mutual Exclusion

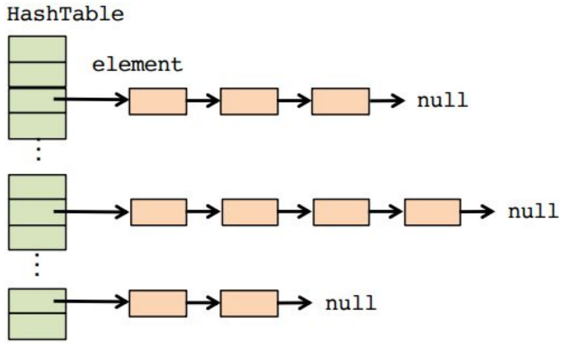
Associate a lock variable with each slot in the hash table, protecting the chain of elements in an slot

```
omp_lock_t hash_lock[SIZE_HASH];

#pragma omp parallel
#pragma omp single
{
for (i = 0; i < SIZE_HASH; i++) omp_init_lock(&hash_lock[i]);

#pragma omp taskloop
for (i = 0; i < SIZE_TABLE; i++) {
     int index = hash_function (dataTable[i], SIZE_HASH);
     omp_set_lock (&hash_lock[index]);
     insert_element (dataTable[i], index, HashTable);
     omp_unset_lock (&hash_lock[index]);
}

for (i = 0; i < SIZE_HASH; i++) omp_destroy_lock(&hash_lock[i]);
}
```



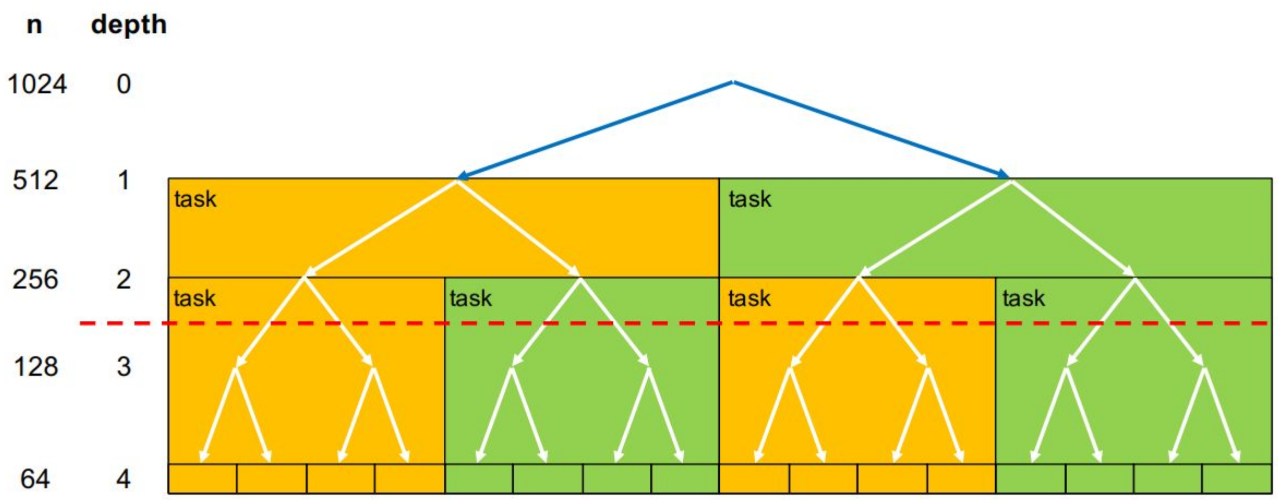
Threads may be inserting elements into the hash table in parallel, as long as these elements hash to different slots

Recursive Task Decomposition

In recursive task decomposition strategies one can control task granularity by controlling recursion levels where tasks are generated (cut-off control).

Recursive Task Decomposition

Tree strategy with **depth recursion control**



Instructor Social Media

Youtube: Lucas Science

Instagram: lucaasbazilio

Twitter: lucasebazilio