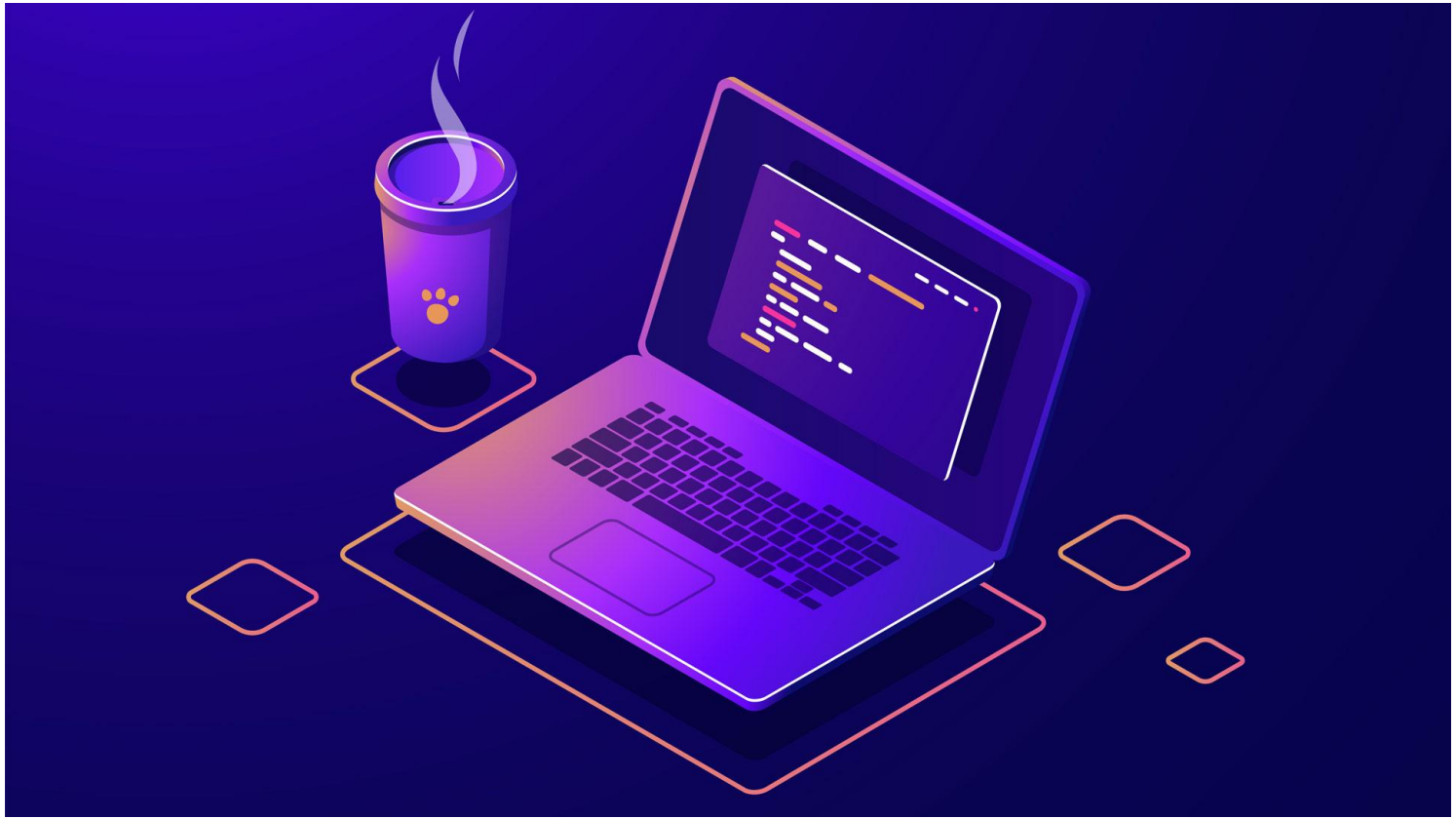
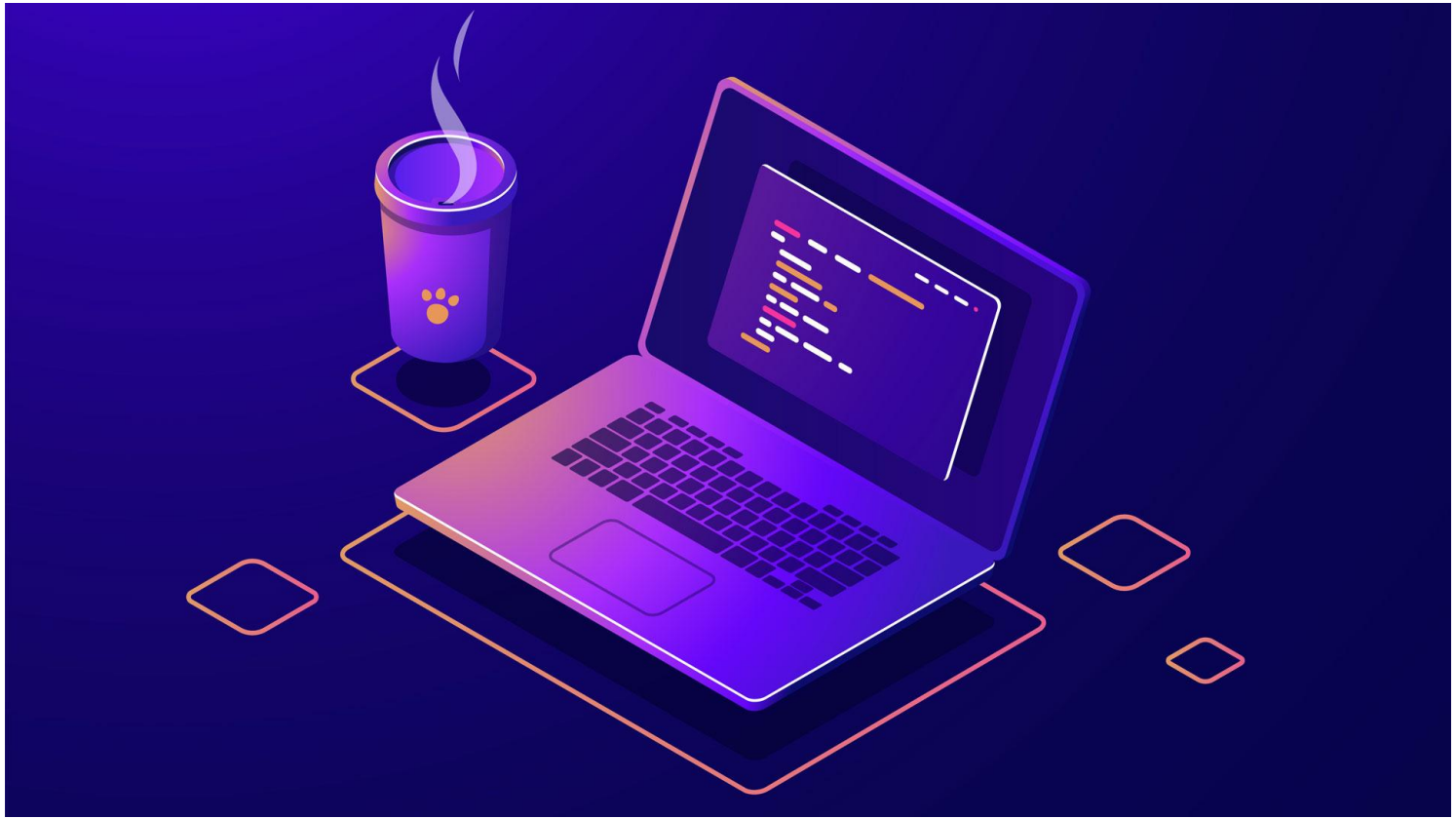
Reduce overhead and serialization



Reduction Clause



Reduction Clause

Reduction is a very common pattern where all threads accumulate values into a single variable

```
reduction(operator:list)
```

- Valid operators are: +,-,*,|,||,&,&&,^,min, max
- The compiler creates a `private` copy of each variable in list that is properly initialized to the identity value
- At the end of the region, the compiler ensures that the `shared` variable is properly (and safely) updated with the partial values of each thread, using the specified operator

Example: Computation of Pi

```
void main ()
{
    int i, id;
    double x, pi, sum;

    step = 1.0/(double) num_steps;
    omp_set_num_threads(NUM_THREADS);
    #pragma omp parallel private(x, i, id) reduction(+:sum)
    {
        id = omp_get_thread_num();
        for (i=id+1; i<=num_steps; i=i+NUM_THREADS) {
            x = (i-0.5)*step;
            sum = sum + 4.0/(1.0+x*x);
        }
    }
    pi = sum * step;
}
```

Specifying reduction operations in explicit tasks generated with either `task`:

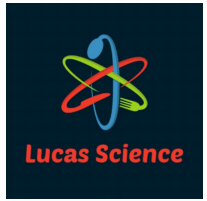
```
#pragma omp parallel
#pragma omp single
{
    #pragma omp taskgroup task_reduction(+: sum)
    for (i=0; i< SIZE; i++)
        #pragma omp task firstprivate(i) in_reduction(+: sum)
        sum += X[i];
}
```

or `taskloop`:

```
#pragma omp parallel
#pragma omp single
{
    // implicit taskgroup in taskloop construct
    #pragma omp taskloop reduction(+: sum)
    for (i=0; i< SIZE; i++)
        sum += X[i];
}
```

Instructor Social Media

Youtube: Lucas Science



Instagram: lucaasbazilio

Twitter: lucasebazilio