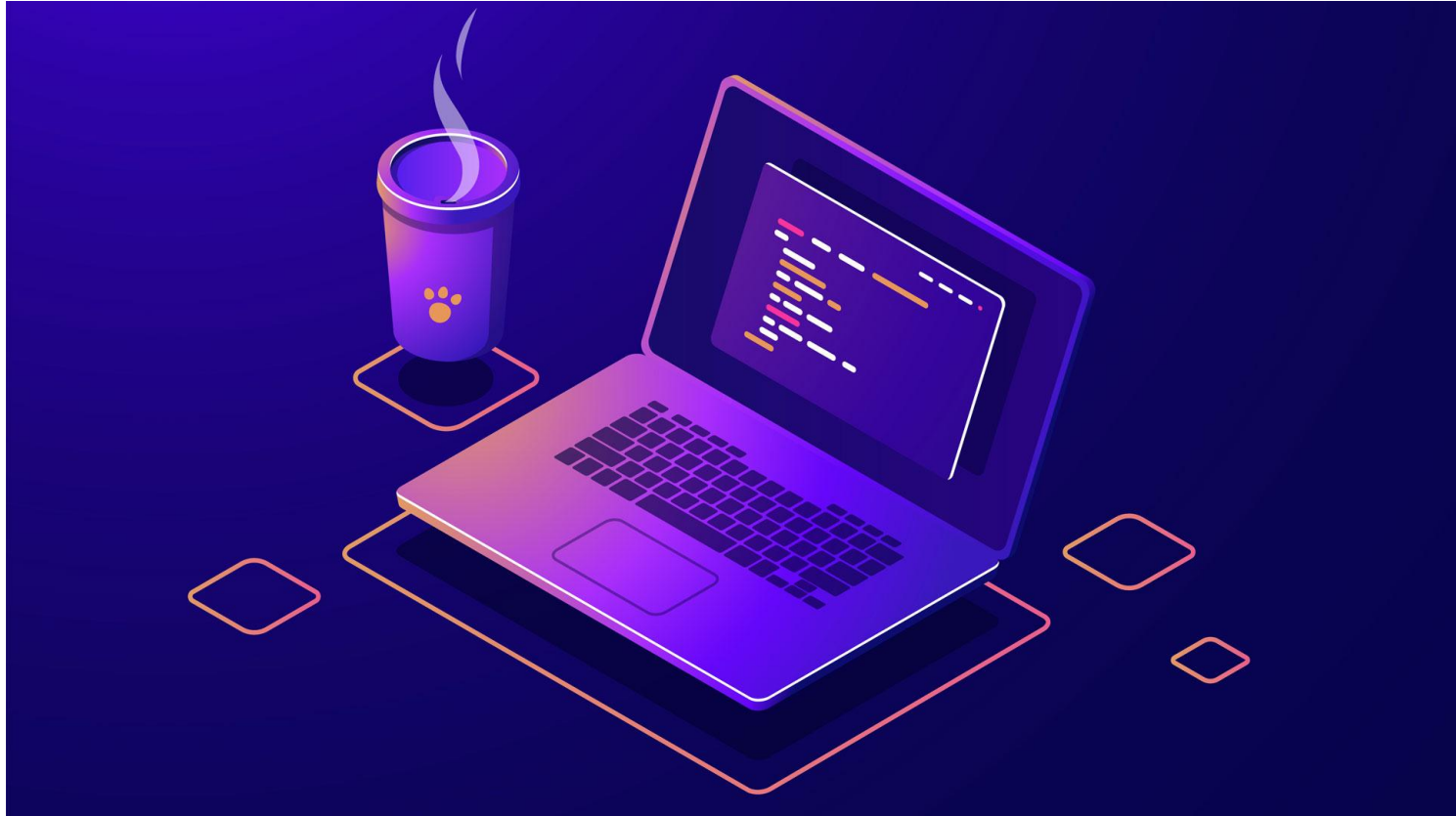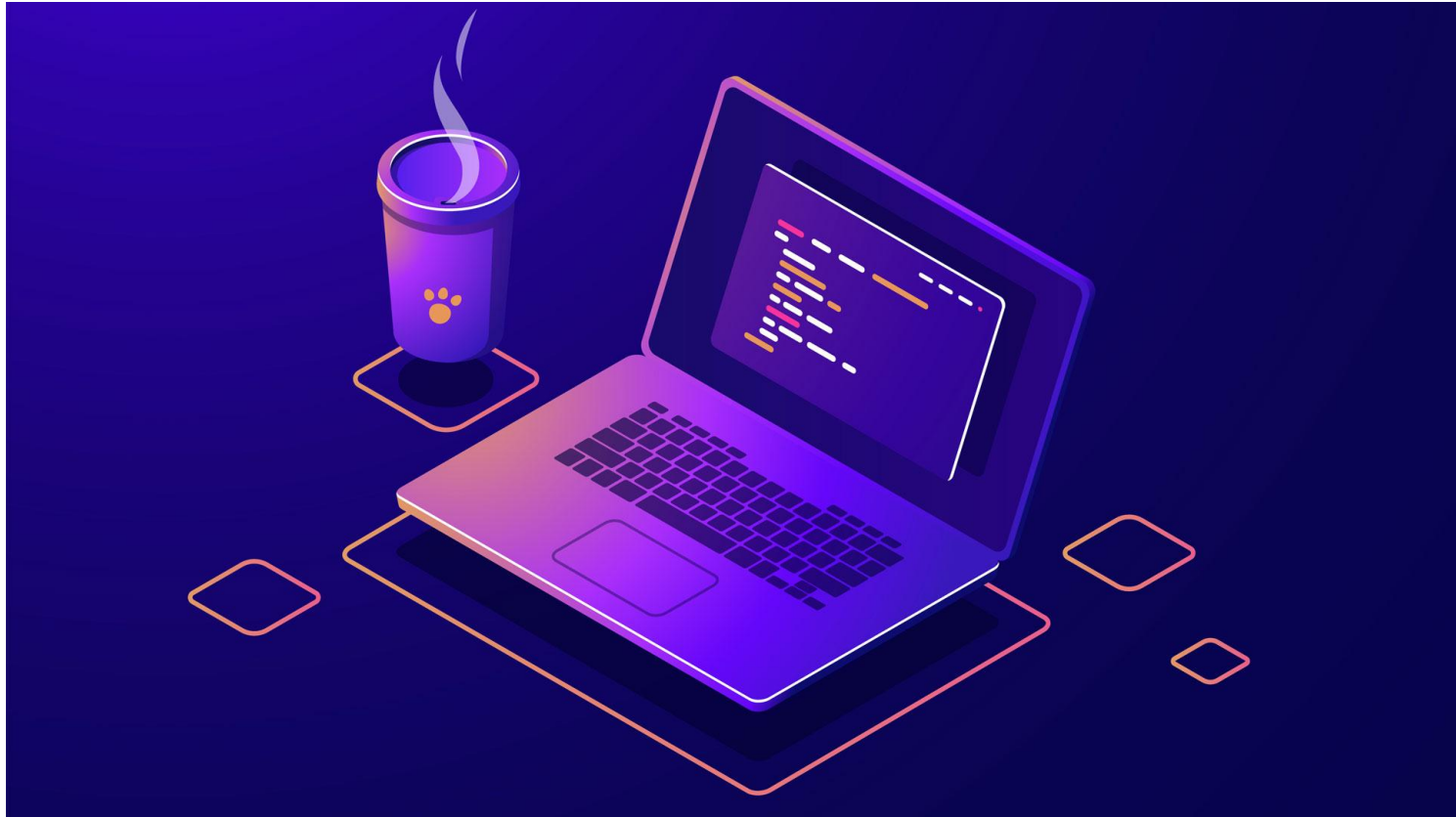# Depth Recursion Control

Tree Strategy with recursion control

Task Generation Control

- Iterative   Task   Decompositions

- Recursive Task Decompositions

Task Generation Control

- Iterative Task Decompositions

- Recursive Task Decompositions

Task Generation Control

- Iterative Task Decompositions

- **Recursive Task Decompositions**

Task Generation Control

Recursive task decomposition: divide–and–conquer (1)

Recursively divide the problem into smaller sub-problems

```c
#define N 1024
#define MIN_SIZE 64
int result = 0;

void dot_product(int *A, int *B, int n) {
    for (int i=0; i< n; i++)
        result += A[i] * B[i];
}

void rec_dot_product(int *A, int *B, int n) {
    if (n>MIN_SIZE) {
        int n2 = n / 2;
        rec_dot_product(A, B, n2);
        rec_dot_product(A+n2, B+n2, n-n2);
    }
    else
        dot_product(A, B, n);
}

void main() {
    rec_dot_product(a, b, N);
}
```
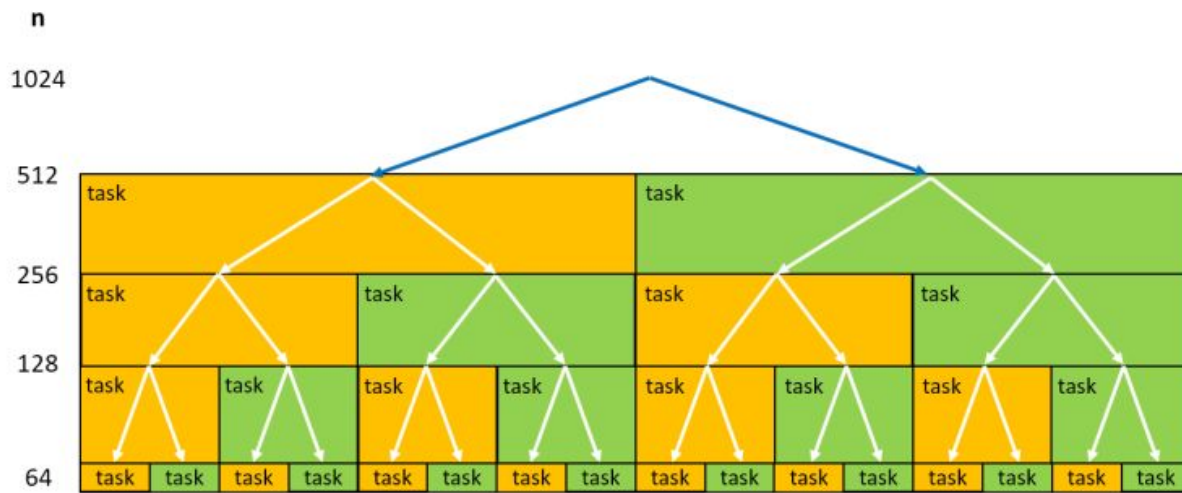
Lucas Bazilio - Udemy

Task Generation Control

Recursive task decomposition: tree strategy (1)

A task corresponds with each invocation of `rec_dot_product`

► Parallel generation of tasks

► Granularity: some tasks simply generate new tasks

Task Generation Control

Recursive task decomposition: different sequential code ...

```c
int dot_product(int *A, int *B, int n) {
    int tmp = 0;
    for (int i=0; i< n; i++) tmp += A[i] * B[i];
    return(tmp);
}

int rec_dot_product(int *A, int *B, int n) {
    int tmp1, tmp2 = 0;
    if (n>MIN_SIZE) {
        int n2 = n / 2;
        tmp1 = rec_dot_product(A, B, n2);
        tmp2 = rec_dot_product(A+n2, B+n2, n-n2);
    } else  tmp1 = dot_product(A, B, n);
    return(tmp1+tmp2);
}

void main() {
    result = rec_dot_product(a, b, N);
}
```

Task Generation Control

Recursive task decomposition: tree strategy (2)

```c
int dot_product(int *A, int *B, int n) {
    int tmp = 0;
    for (int i=0; i< n; i++) tmp += A[i] * B[i];
    return(tmp);
}

int rec_dot_product(int *A, int *B, int n) {
    int tmp1, tmp2 = 0;
    if (n>MIN_SIZE) {
        int n2 = n / 2;
        #pragma omp task shared(tmp1) // firstprivate(A, B, n, n2) by default
        tmp1 = rec_dot_product(A, B, n2);
        #pragma omp task shared(tmp2) // firstprivate(A, B, n, n2) by default
        tmp2 = rec_dot_product(A+n2, B+n2, n-n2);
        #pragma omp taskwait
    } else tmp1 = dot_product(A, B, n);
    return(tmp1+tmp2);
}

void main() {
    #pragma omp parallel
    #pragma omp single
    result = rec_dot_product(a, b, N);
}
```

Shared Clause

shared: Specifies that one or more variables should be shared among all threads.

taskwait: Specifies a wait on the completion of child tasks of the current task.

Task Generation Control

Recursive task decomposition: tree strategy (2)

```c
int dot_product(int *A, int *B, int n) {
   int tmp = 0;
   for (int i=0; i< n; i++) tmp += A[i] * B[i];
   return(tmp);
}

int rec_dot_product(int *A, int *B, int n) {
   int tmp1, tmp2 = 0;
   if (n>MIN_SIZE) {
       int n2 = n / 2;
       #pragma omp task shared(tmp1) // firstprivate(A, B, n, n2) by default
       tmp1 = rec_dot_product(A, B, n2);
       #pragma omp task shared(tmp2) // firstprivate(A, B, n, n2) by default
       tmp2 = rec_dot_product(A+n2, B+n2, n-n2);
       #pragma omp taskwait
   } else tmp1 = dot_product(A, B, n);
   return(tmp1+tmp2);
}

void main() {
   #pragma omp parallel
   #pragma omp single
   result = rec_dot_product(a, b, N);
}
```
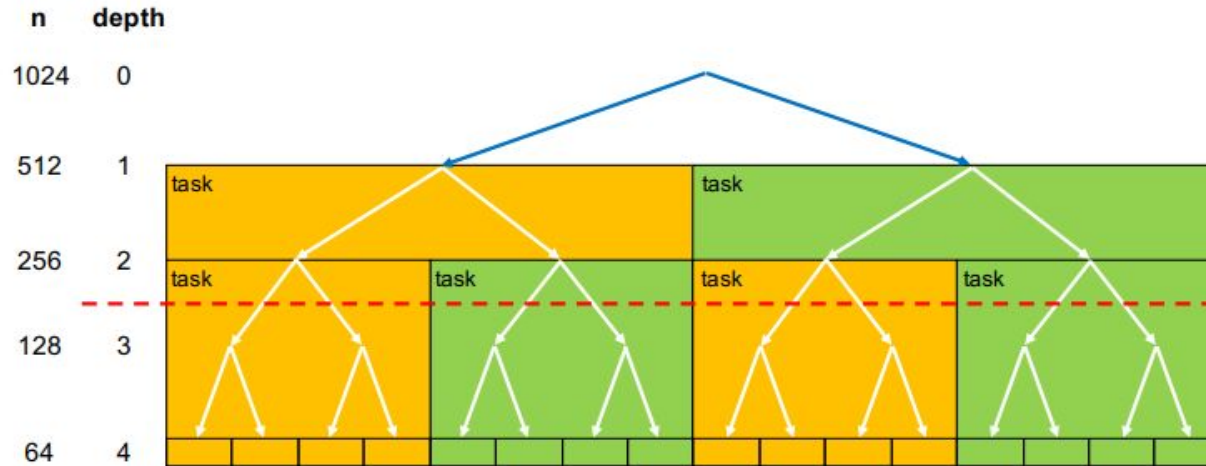
Task Generation Control

How to control task granularity in tree strategy (1)

Tree strategy with **depth recursion control**

Task Generation Control

How to control task granularity in tree strategy (2)

Tree strategy with **depth recursion control**

```c
#define N 1024
#define MIN_SIZE 64
#define CUTOFF 3

int rec_dot_product(int *A, int *B, int n, int depth) {
    int tmp1, tmp2 = 0;
    if (n>MIN_SIZE) {
        int n2 = n / 2;
        if (depth < CUTOFF) {
            #pragma omp task shared(tmp1)
            tmp1 = rec_dot_product(A, B, n2, depth+1);
            #pragma omp task shared(tmp2)
            tmp2 = rec_dot_product(A+n2, B+n2, n-n2, depth+1);
            #pragma omp taskwait
        } else {
            tmp1 = rec_dot_product(A, B, n2, depth+1);
            tmp2 = rec_dot_product(A+n2, B+n2, n-n2, depth+1);
        }
    }
    else tmp = dot_product(A, B, n);
    return(tmp1+tmp2);
}
```

Task Generation Control

OpenMP support for cut-off

► `final` clause: If the expression of a `final` clause evaluates to *true* the generated task and **all of its descendent tasks** will be final. The execution of a final task is sequentially **included** in the generating task (but the task is still generated)

► `omp_in_final()` intrinsic function: it returns true when executed in a final task region; otherwise, it returns false.

Task Generation Control

OpenMP support for cut-off: tree strategy

Making use of omp_in_final:

```
#define MIN_SIZE 64
#define CUTOFF 3
...
int rec_dot_product(int *A, int *B, int n, int depth) {
    int tmp1, tmp2 = 0;
    if (n>MIN_SIZE) {
        int n2 = n / 2;
        if (!omp_in_final()) {
            #pragma omp task shared(tmp1) final(depth >= CUTOFF)
            tmp1 = rec_dot_product(A, B, n2, depth+1);
            #pragma omp task shared(tmp2) final(depth >= CUTOFF)
            tmp2 = rec_dot_product(A+n2, B+n2, n-n2, depth+1);
            #pragma omp taskwait
        } else {
            tmp1 = rec_dot_product(A, B, n2, depth+1);
            tmp2 = rec_dot_product(A+n2, B+n2, n-n2, depth+1);
        }
    }
    else tmp1 = dot_product(A, B, n);
    return(tmp1+tmp2);
}
...
```

Instructor Social Media

Youtube: Lucas Science

Instagram: lucaasbazilio

Twitter: lucasebazilio