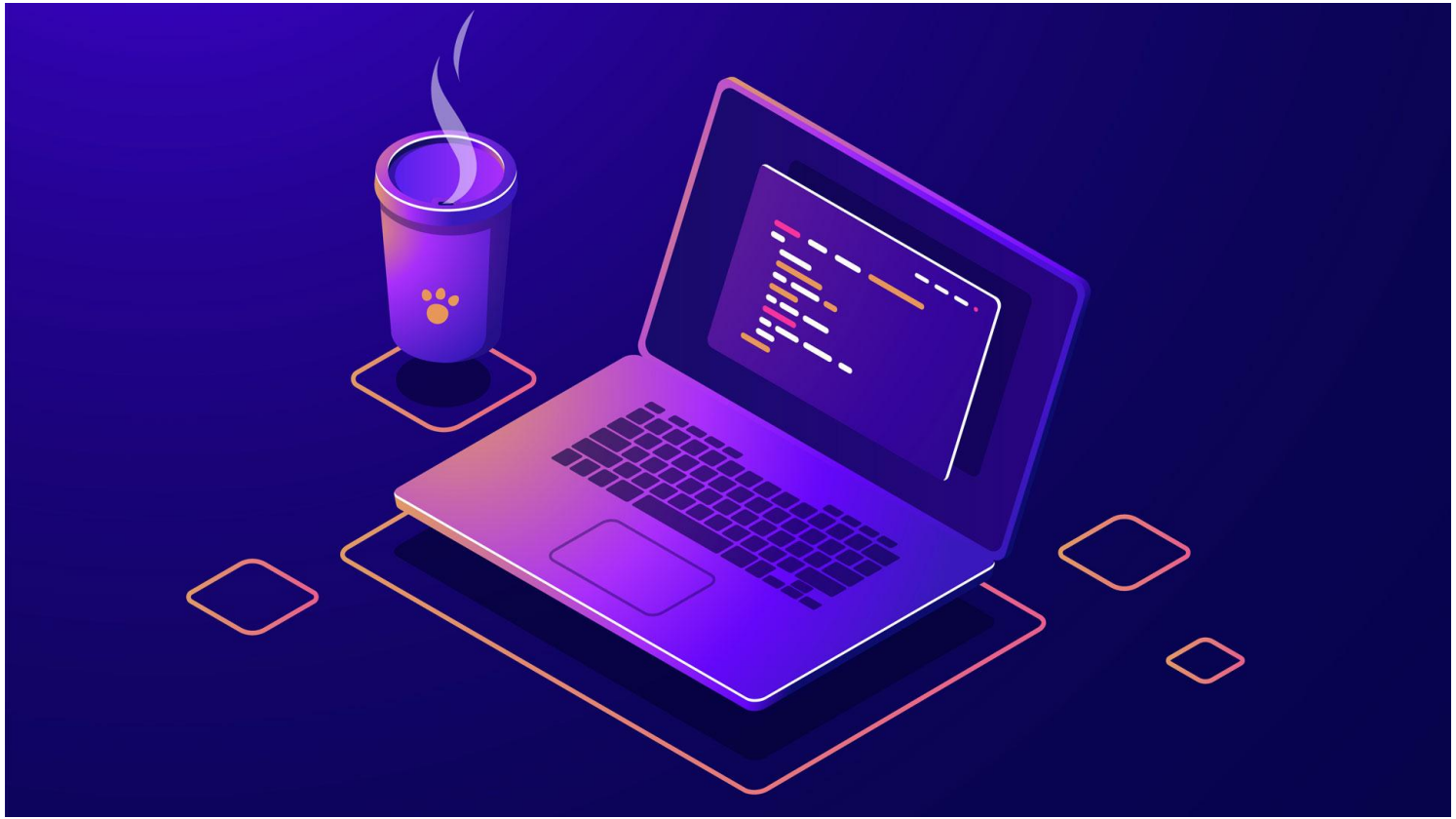
Recursive Task Decomposition in OpenMP



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

Excessive task generation may not be necessary (i.e. cause excessive overhead): need mechanisms to control number of tasks and/or their granularity

- In iterative task decomposition strategies one can control task granularity by setting the number of iterations executed by each task
- In recursive `task` decomposition strategies one can control task granularity by controlling recursion levels where tasks are generated (**cut-off control**)
  - after certain number of recursive calls (static control)
  - when the size of the vector is too small (static control)
  - when there are sufficient tasks pending to be executed (dynamic control)

Task Generation Control

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

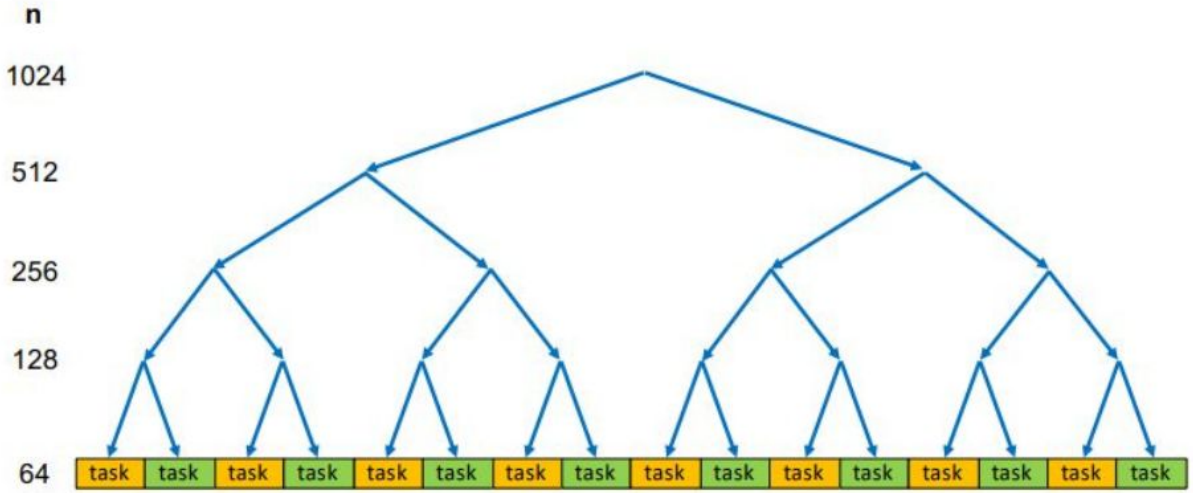
Recursively divide the problem into smaller sub-problems

```
#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);
}
```

Task Generation Control

Recursive task decomposition: leaf strategy (1)

A task corresponds with each invocation of `dot_product` once the recursive invocations stop



- Sequential generation of tasks

Task Generation Control

Recursive task decomposition: leaf strategy (2)

```
#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
        #pragma omp task
        dot_product(A, B, n);
}

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

Task Generation Control

Recursive task decomposition: leaf strategy (2)

```
#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
        #pragma omp task
        dot_product(A, B, n);
}

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

Also…

We also need to use the atomic directive.

`atomic` : ensures that a specific storage location is accessed atomically, rather than exposing it to the possibility of multiple, simultaneous reading and writing threads that may result in indeterminate values.

Task Generation Control

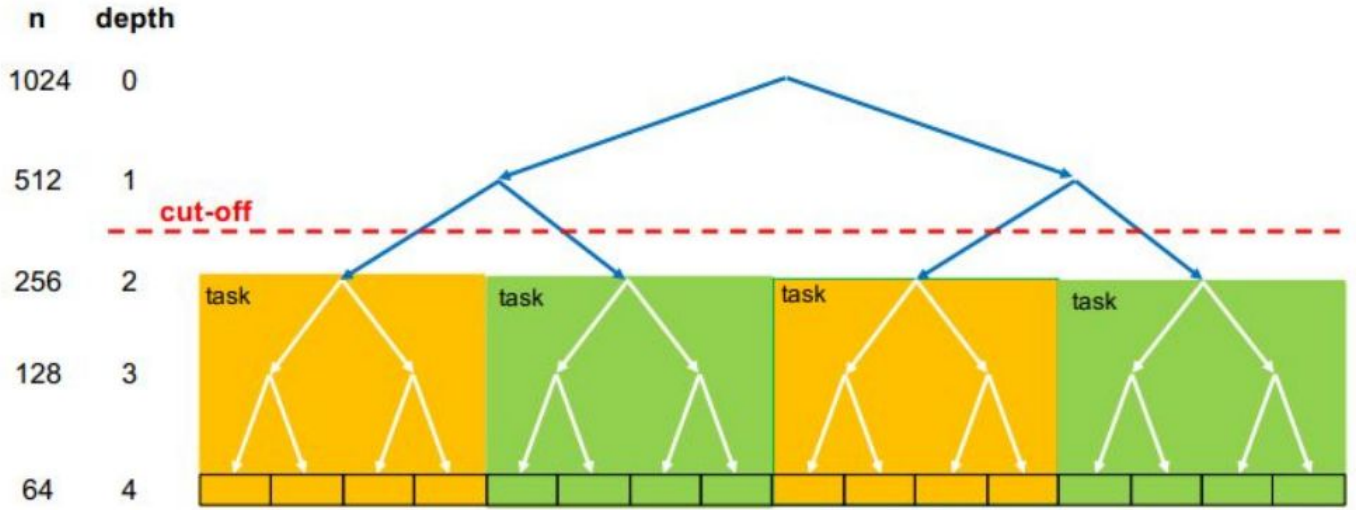
Recursive task decomposition: leaf strategy (3)

```
#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++)
       #pragma omp atomic
       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
       #pragma omp task
       dot_product(A, B, n);
}
```

Task Generation Control

How to control task granularity in leaf strategy (1)

Leaf parallelization with **depth recursion control**



Task Generation Control

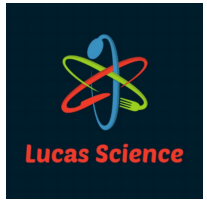
How to control task granularity in leaf strategy (2)

Leaf strategy with **depth recursion control**

```
#define CUTOFF 2
...
void rec_dot_product(int *A, int *B, int n, int depth) {
    if (n>MIN_SIZE) {
        int n2 = n / 2;
        if (depth == CUTOFF)
            #pragma omp task
            {
                rec_dot_product(A, B, n2, depth+1);
                rec_dot_product(A+n2, B+n2, n-n2, depth+1);
            }
        else {
            rec_dot_product(A, B, n2, depth+1);
            rec_dot_product(A+n2, B+n2, n-n2, depth+1);
        }
    }
    else    // if recursion finished, need to check if task has been generated
        if (depth <= CUTOFF)
            #pragma omp task
            dot_product(A, B, n);
        else
            dot_product(A, B, n);
}
...
```

Instructor Social Media

Youtube: Lucas Science



Instagram: lucaasbazilio

Twitter: lucasebazilio