

Unit 3: Task decomposition strategies

Video lesson 4 – Linear, iterative and recursive decompositions

Eduard Ayguadé, Josep Ramon Herrero,
Daniel Jiménez and Gladys Utrera

Barcelona Supercomputing Center
Universitat Politècnica de Catalunya

Identifying tasks in your sequential program (patterns)

- **Linear** task decomposition
 - A task is a "code block" or a procedure invocation
- **Iterative** task decomposition
 - Tasks found in iterative constructs, such as loops (countable or uncountable)
- **Recursive** task decomposition
 - Tasks found in divide-and-conquer problems and other recursive problems

Example 1: linear task decomposition

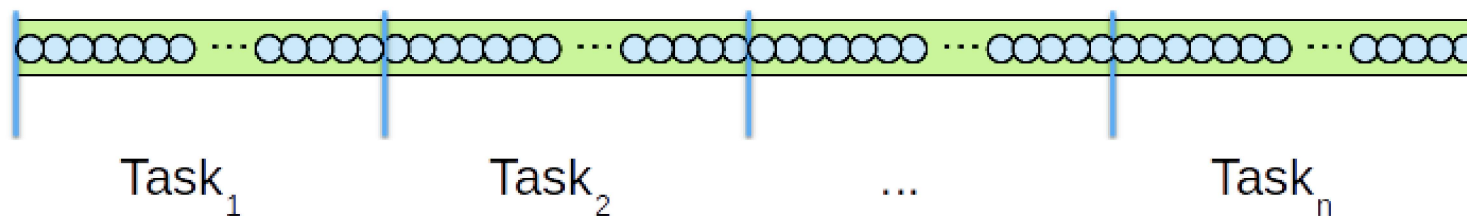
A task is a "code block" or a procedure invocation

```
int main() {  
    ...  
    tareador_start_task("init_A");  
    initialize(A, N);  
    tareador_end_task("init_A");  
  
    tareador_start_task("init_B");  
    initialize(B, N);  
    tareador_end_task("init_B");  
  
    tareador_start_task("dot_product");  
    dot_product (N, A, B, &result);  
    tareador_end_task("dot_product");  
    ...  
}
```

Example 2: iterative task decomposition

A task is a chunk of iterations of a loop, as for example, in the sum of two vectors

```
void vector_add(int *A, int *B, int *C, int n) {  
    for (int i=0; i< n; i++) C[i] = A[i] + B[i];  
}  
  
void main() {  
    ....  
    vector_add(a, b, c, N);  
    ...  
}
```



Example 2: sum of two vectors (cont.)

Single loop iteration:

```
void vector_add(int *A, int *B, int *C, int n) {  
    for (int i=0; i< n; ii++)  
        tareador_start_task("singleit");  
        C[i] = A[i] + B[i];  
        tareador_end_task("singleit");  
}
```

Chunk of loop iterations:

```
#define BS 16  
void vector_add(int *A, int *B, int *C, int n) {  
    for (int ii=0; ii< n; ii+=BS) {  
        tareador_start_task("chunkit");  
        for (i = ii; i < min(ii+BS, n), i++)  
            C[i] = A[i] + B[i];  
        tareador_end_task("chunkit");  
    }  
}
```

Example3: Non countable loops - list traversal example

List of elements, traversed using an uncountable (while) loop

```
int main() {
    struct node *p;

    p = init_list(n);
    ...

    while (p != NULL) {
        tareador_start_task("computeNode");
        process_work(p);
        tareador_end_task("computeNode");
        p = p->next;
    }
    ...
}
```

Example 4: "Divide-and-conquer" task decomposition

Sum of two vectors by recursively dividing the problem into smaller sub-problems

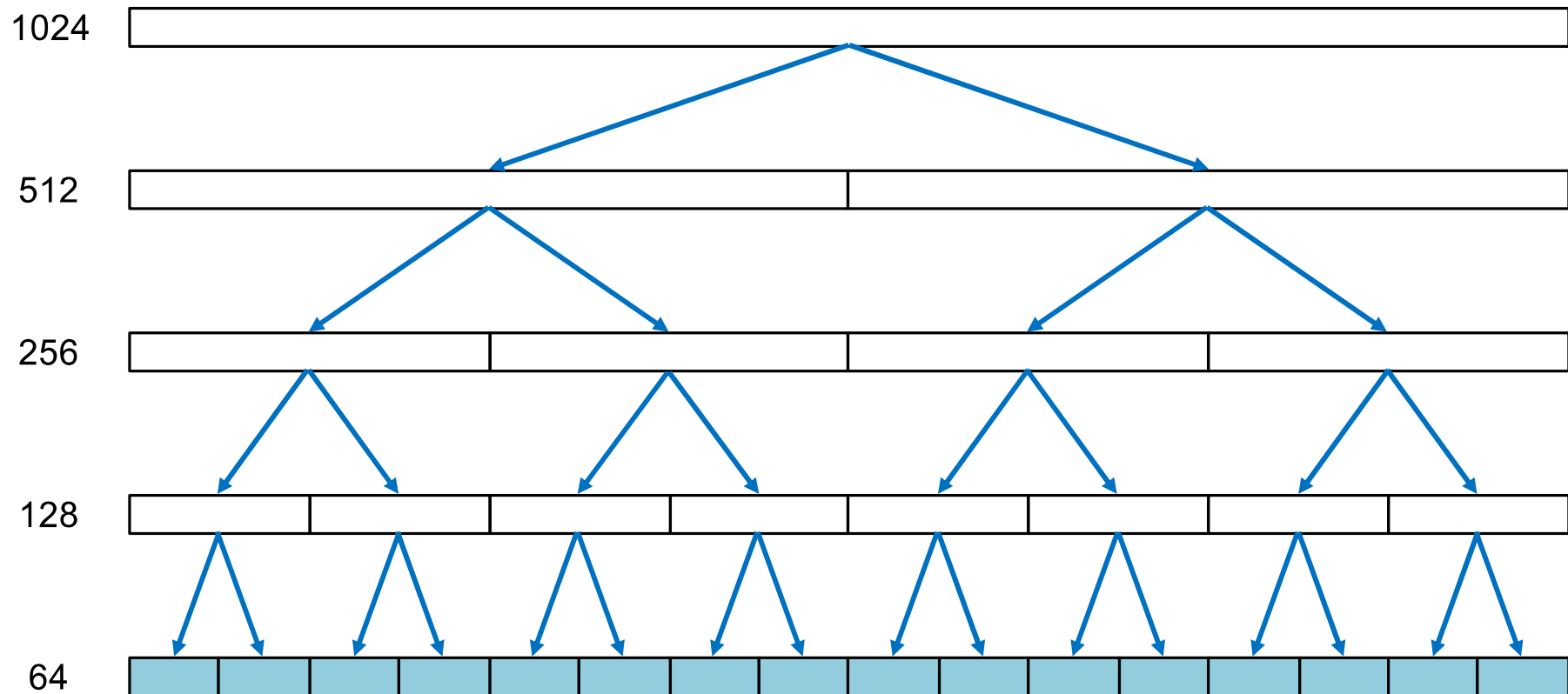
```
#define N 1024
#define MIN_SIZE 64

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

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

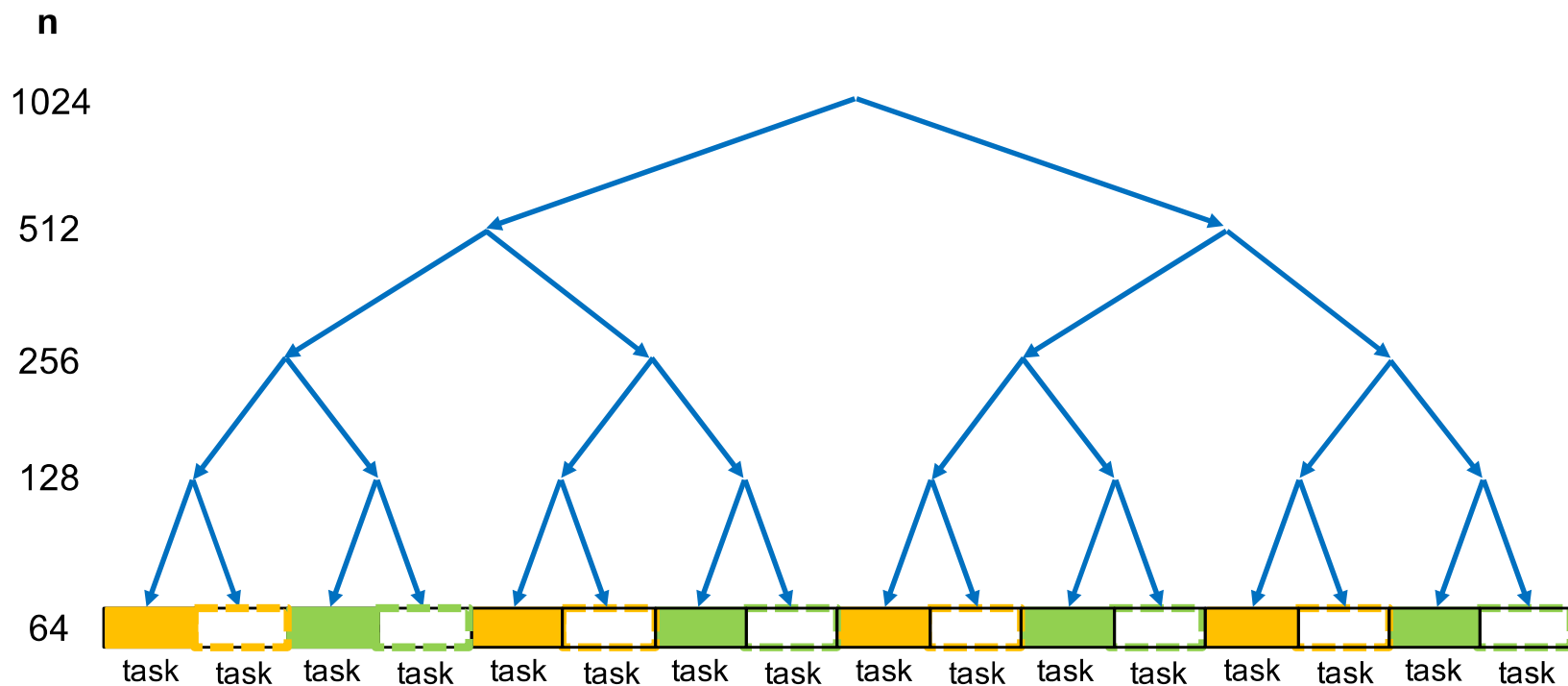
void main() {
    ....
    rec_vector_add(a, b, c, N);
    ...
}
```

Example 4: "Divide-and-conquer" task decomposition



Two possible decomposition strategies

- **Leaf strategy:** a task corresponds with each invocation of `vector_add` once the recursive invocations stop



Example 4: Leaf task decomposition

```
#define N 1024
#define MIN_SIZE 64

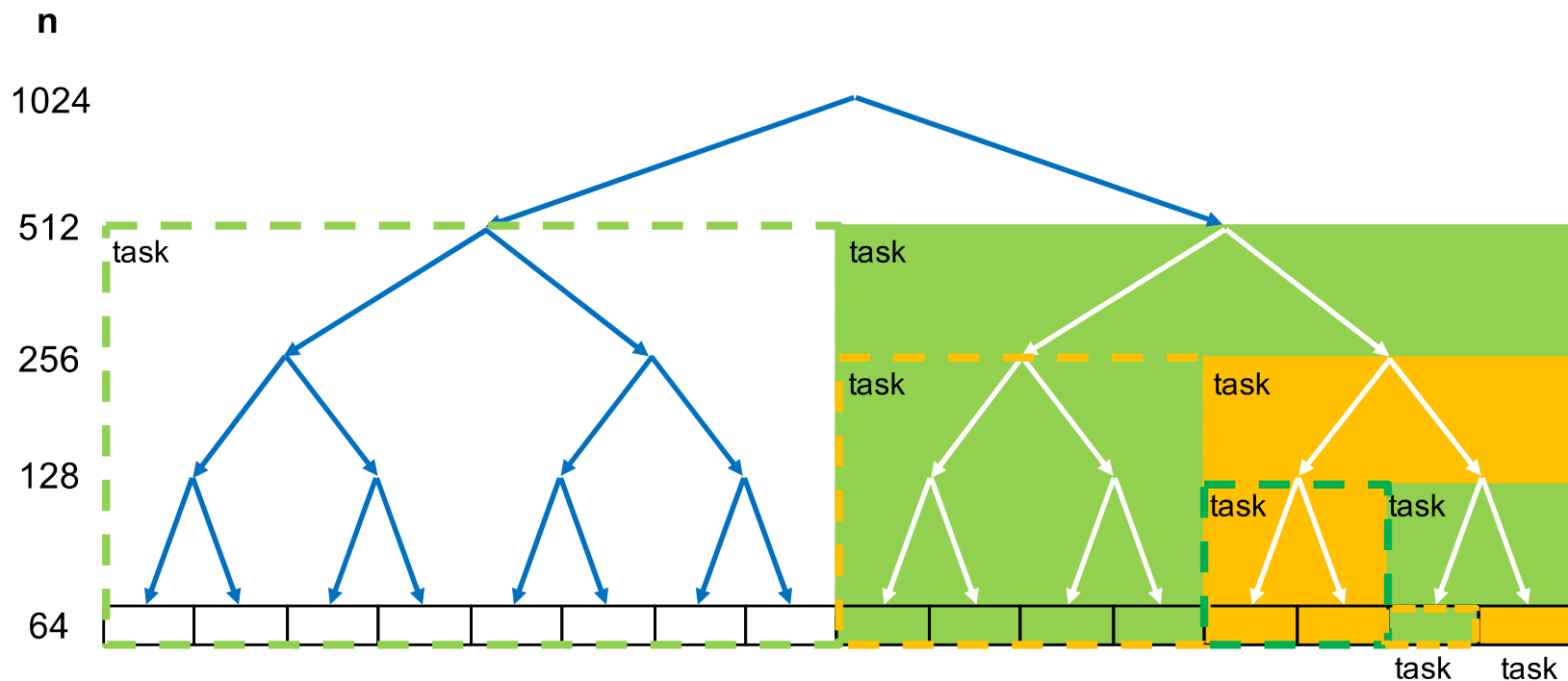
void vector_add(int *A, int *B, int *C, int n) {
    for (int i=0; i< n; i++) C[i] = A[i] + B[i];
}

void rec_vector_add(int *A, int *B, int *C, int n) {
    if (n>MIN_SIZE) {
        int n2 = n / 2;
        rec_vector_add(A, B, C, n2);
        rec_vector_add(A+n2, B+n2, C+n2, n-n2);
    }
    else
    {
        tareador_start_task("leaftask");
        vector_add(A, B, C, n);
        tareador_end_task("leaftask");
    }
}

void main() {
    ....
    rec_vector_add(a, b, c, N);
    ...
}
```

Two possible decomposition strategies (cont.)

- **Tree strategy:** a task corresponds with each invocation of `rec_vector_add` during the *parallel* recursive execution



Example 4: Tree task decomposition

```
#define N 1024
#define MIN_SIZE 64

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

void rec_vector_add(int *A, int *B, int *C, int n) {
    if (n>MIN_SIZE) {
        int n2 = n / 2;
        tareador_start_task("treetask1");
        rec_vector_add(A, B, C, n2);
        tareador_end_task("treetask1");
        tareador_start_task("treetask2");
        rec_vector_add(A+n2, B+n2, C+n2, n-n2);
        tareador_end_task("treetask2");
    }
    else vector_add(A, B, C, n);
}

void main() {
    ....
    rec_vector_add(a, b, c, N);
    ...
}
```

Unit 3: Task decomposition strategies

Video lesson 4 – Linear, iterative and recursive decompositions

Eduard Ayguadé, Josep Ramon Herrero,
Daniel Jiménez and Gladys Utrera

Barcelona Supercomputing Center
Universitat Politècnica de Catalunya