
# CPE745-PARALLEL COMPUTING

# An Overview of Target and SIMD Directives and Clauses (open MP)

**Group 1:**

**Nada Aladdin Abdelfatah 158613**

**Batool Mohammad Kayyam 158612**

**With:**

**Dr. Fady Ghanim**

## Introduction to OpenMP:

**OpenMP** is a widely adopted application programming interface that facilitates parallel programming in shared-memory computing environments. By allowing developers to write parallel programs using a combination of compiler directives and runtime library routines, **OpenMP** provides a simple and portable approach for exploiting parallelism in programs, which allows for efficient utilization of multicore processors and shared-memory systems.

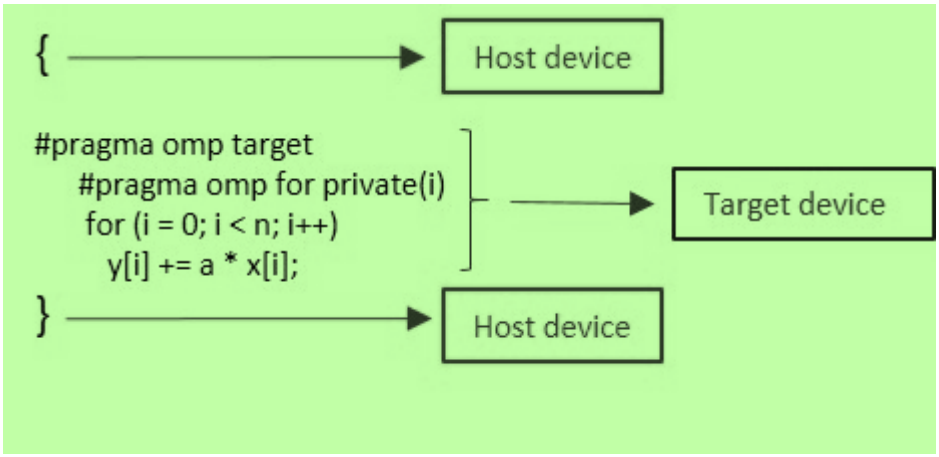
In this report, we will discuss different **OpenMP** directives related to target offloading, including **OpenMP** Target, which enables the execution of specific code blocks on target devices while the rest of the code runs on the host CPU. We will also examine the various clauses that can be used with the **OpenMP Target**. Furthermore, we will talk about some directives such as **OMP Target Teams**, **OMP Target Teams Distribute**, **OMP Target Teams Distribute Parallel**, and **OMP Target Teams Loop**, which extend parallelism beyond a single target device and enable the distribution of work among multiple devices in a team. These directives give programmers the freedom to take advantage of parallelism in heterogeneous systems. Also, we will talk about **OMP Declare Target** directive. This directive allows us to declare variables and functions in an explicit way, which leads to allocating them in the memory of the target device.

Finally, we will take an overview of **OMP SIMD** single instruction multiple data and **OMP Parallel For SIMD**, which allows the vectorization in execute Loops. In addition to talk about some of important clauses used with **SIMD** directives and the **OMP Declare SIMD** directives.

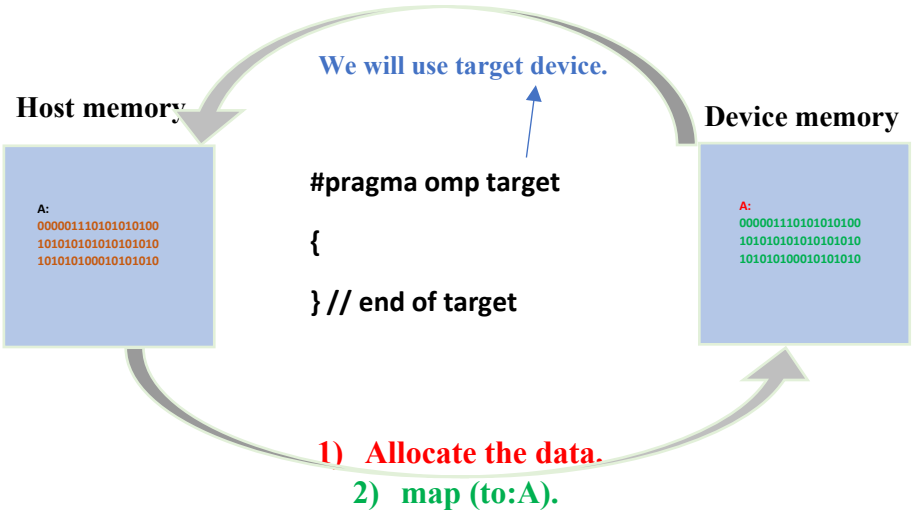
## OpenMP Target

The directive OpenMP Target enables the offloading of code and data to the target device. It allows for the execution of a specified block of code on a target device, such as a GPU or FPGA, while the other parts of the code continue to execute on the host CPU.

The syntax of the target construct in C/C++:
#pragma omp target [clause[ [,] clause] ... ] new-line
structured-block




4) **Map from: To get the data back**




1) **Allocate the data.**
2) **map (to:A).**
3) **Computation and update the data.**

## Clauses Used with Target

As we know we use target directive to offload a part of the code to the target device. In this section we will talk about some clauses that are used with target directive.

**Map** is a clause used in openMP codes to explain and select the exact data flow between target device (g.e., GPU) and the host (g.e., CPU). We can choose the exact data we want to move to the target device by using **map** clause. This will ensure that all data we will need int the target device is available and return the data after execution.
Mapping is done by using three data transfer methods (to, from, and tofrom), these methods take the variables as parameters.

- **to**: Copy the data from the host (CPU) to the target device (GPU), so the target device now can manage the data.
- **from**: Copy the data from the target device (GPU) to the host (CPU) back after making the computations.
- **tofrom**: Copy the data in bidirectional way from and to the target device.

*Example: #pragma omp target map (from: array[0:N])*

**If** is a clause which can be used with target directive to offload the part of code to the target device depending on the condition. If the condition is true, the part of code will execute in the target device.

*How it is used? if([ target :] scalar-expression)*

**Device** is a clause used with target directive that allows us to choose the target device to offload the code to, by passing integer value represents the device id.

*Example:*

```
#pragma omp target device(device_id)
{
   // Code executed on the specified target device
}
```

**Private** is a clause that is used to declare a variable in private way in each target device. So, each thread in each target device has its own copy.

*Example:*

```
#pragma omp target private(shared_var)
  {
      shared_var = omp_get_thread_num();
// Each thread on the target device modifies its private copy
  }
```

**Firstprivate** is a clause used to declare a variable in private way in each target device. Each thread in each target device has its own copy. Also, it initialized the variable from the shared variable.

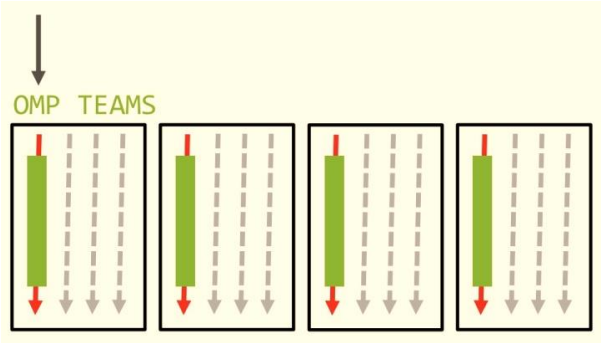
```
#pragma omp target firstprivate(shared_var, private_var)
  {
      private_var = shared_var + omp_get_thread_num();
// Each thread on the target device modifies its private copy
  }
```

## OMP Target Teams

The **OMP Target Teams directive** expands the OpenMP API's parallelism capabilities to include target devices other than just one. It enables programmers to divide work among multiple target devices, assembling teams of devices to run programs simultaneously. This directive is particularly useful when dealing with heterogeneous systems that have multiple accelerators or coprocessors.

Syntax:

**#pragma omp target teams distribute [clause[ [,] clause] ... ] new-line**



**num_teams:** specifies the number of teams to create, where each team consists of one or more threads

**thread_limit:** specifies the maximum number of threads per team.

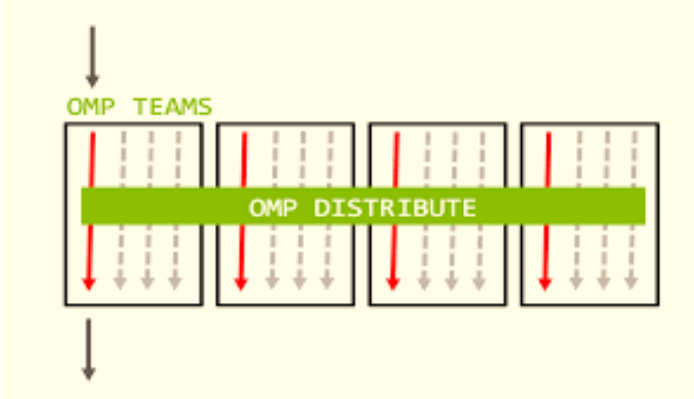
## OMP target teams distribute

syntax:

```
#pragma omp target teams distribute [clause[ [,] clause] ... ] new-line
        loop-nest
```

This directive is used as an extension of target teams directive. Which performs the distribution over the teams. That will distribute all loop iterations across the team of threads on the target device.
Example:

```
#pragma omp target teams distribute map(to:a[0:N][0:M],b[0:N][0:M])
map(from:c[0:N][0:M])
        for (int i = 0; i < N; i++) {
                #pragma omp distribute parallel for
                    for (int j = 0; j < M; j++) {
                       c[i][j] = a[i][j] + b[i][j];
                     }
        }
```




Ability to execute multiple independent blocks of code simultaneously.

## OMP Target Teams Distribute Parallel

Syntax:

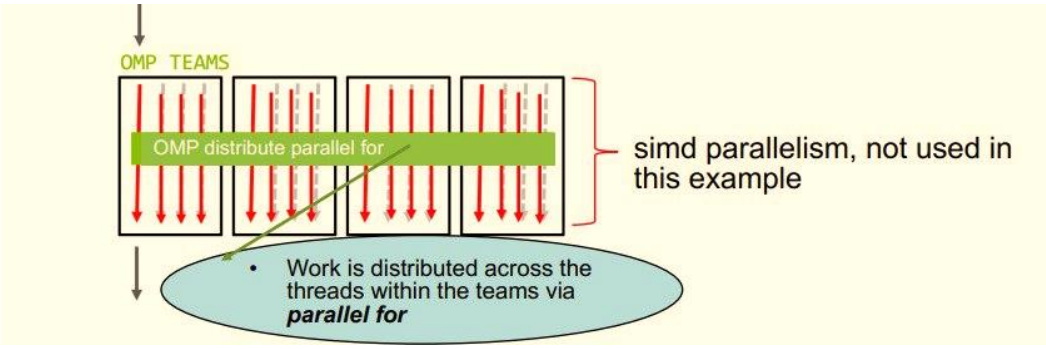
#pragma omp target teams distribute parallel for [clause[ [,] clause] ... ] new-line
loop-nest

Now, in this directive we will be able to distribute the loop iterations inside each teams of threads. By distribute the work over each thread to apply the parallelization inside each team.
Then, each thread will do a subset of the loop iterations.

Example:

```
#pragma omp target teams distribute parallel map(to:x[0:sz]) map(tofrom:y[0:sz])
    for (int i = 0; i < sz; i++) {
      y[i] = a * x[i] + y[i];

    }
```




Ability to execute multiple independent operations within a single block of the code simultaneously.

## OMP Target Teams Loop

Syntax:

*#pragma omp target teams loop [clause[ [,] clause] ... ] new-line*

This clause will apply parallelism over several teams of threads. So, each team will execute a certain number of loop iterations.

Example:

```
#pragma omp target teams loop
for (int i = 0; i < 10; i++) {
    // Loop body
}
```

## OMP Declare Target

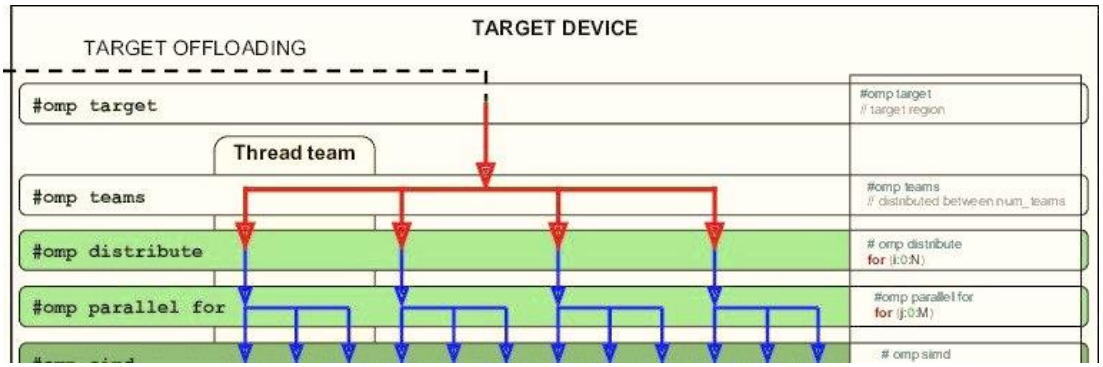
```
#pragma omp declare target new-line
   declaration-definition-seq
```

This clause will provide us with a way to declare functions to use in target device by calling it directly. So, just after declaration done on the function, the target device can use it while execution.

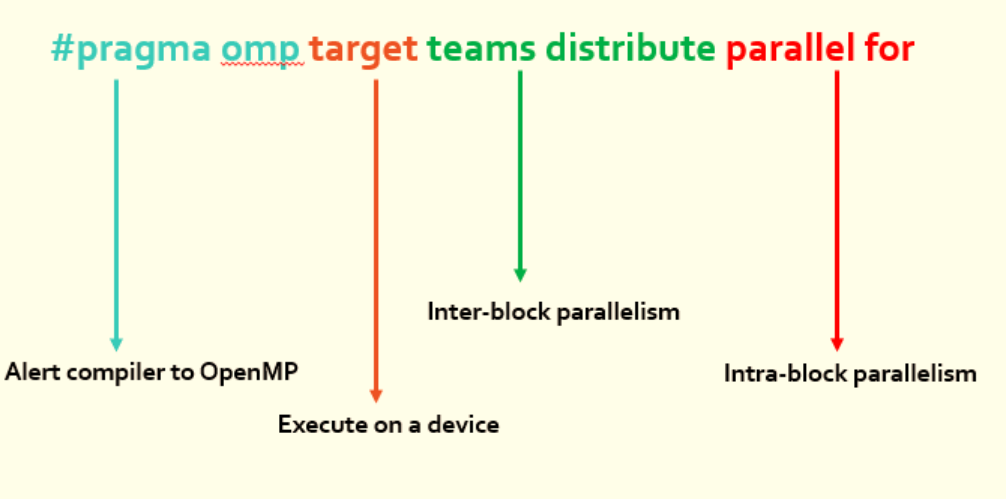
Note: If we just declare the function as shown in the example without calling it under target directive, that will be useless.

Example:

```
#pragma omp declare target
  int FirstFunction(int a);
  int SecondFunction(int a);
#pragma omp end declare target
  ...
#pragma omp target {
  ...
  x = FirstFunction(a) * SecondFunction(a);
  ...
}
```

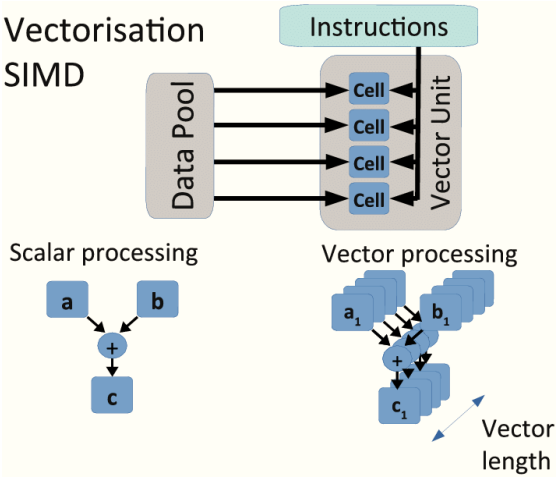


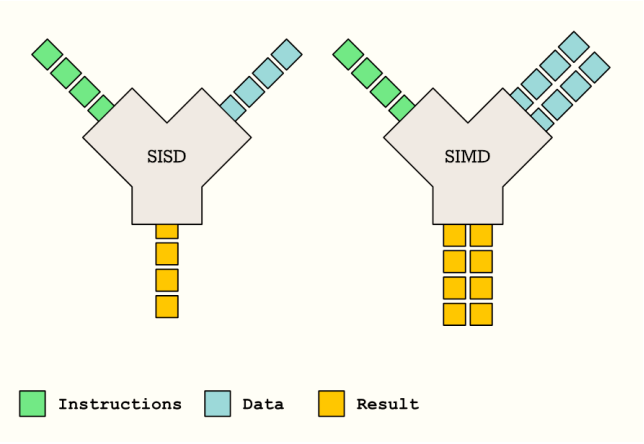

This picture will summarize what we mentioned above.




## OpenMP SIMD

Now we will talk about another directive related to OpenMP which is SIMD. SIMD which stands for single Instruction Multiple Data, is a different parallel processing that performs a way to execute a single instruction on multiple data elements simultaneously.




Syntax:

*#pragma omp simd loop [clause[ [,] clause] ... ] new-line*

*for loops*

By using the **"#pragma omp simd"** directive and appropriate clauses, developers can vectorize loops.

## OMP Parallel For SIMD

Syntax:

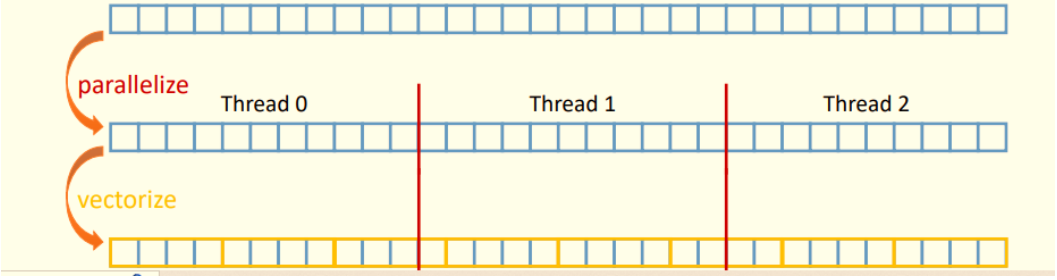
*#pragma omp parallel for simd [clause[[,] clause] ...]*

In this directive we will take advantage of parallelization and vectorization. This will be done by executing multiple loop iterations on different data elements.

Example:

```
#pragma omp parallel for simd
 for (int i = 0; i < n; i++) {
    c[i] = a[i] + b[i];
 }
```

```
void sprod(float *a, float *b, int n) {
  float sum = 0.0f;
#pragma omp for simd reduction(+:sum)
  for (int k=0; k<n; k++)
    sum += a[k] * b[k];
  return sum;
}
```



## Clauses Used with SIMD

This part will include clauses used with SIMD to perform different purposes.

**safelen clause:**

By using this clause, we will safely execute multiple loop iterations. That done by passing an integer number represents the maximum consecutive iterations.

```
#pragma omp simd safelen(4)
for (int i = 0; i < N; i++) {
   // Loop body
   result[i] = array[i] + scalar;
}
```

**Uniform:**

A clause used to indicate that there is a variable (which passed as parameter) will have a constant value across all iterations of the loop.

```
#pragma omp simd uniform(scalar)
  for (int i = 0; i < N; i++) {
     c[i] = a[i] + b[i] + scalar;
  }
```

**collapse:**

A clause specifies how many nested loops should be collapsed into a single loop. In the example, the collapse (2) clause will collapse the two outer loops into a single loop for parallelization.

```
#pragma omp simd collapse(2)
  for (int i = 0; i < N; i++) {
     for (int j = 0; j < N; j++) {
        double sum = 0.0;
        #pragma omp simd
        for (int k = 0; k < N; k++) {
           sum += a[i][k] * b[k][j];
        }
        c[i][j] = sum;
     }
  }
```

| Clause | Syntax | How it works |
|---|---|---|
| simdlen | #pragma omp simd simdlen(value)<br>for (int i = 0; i < n; i++) {<br>// Loop body<br>} | is used to specify the preferred vector length or number of SIMD lanes to be used for vectorization. It allows you to provide a hint to the compiler about the desired SIMD vector length. |
| linear | #pragma omp simd linear(i:1)<br>for (int i = 0; i < n; i++) {<br>y[i] = a * x[i] + b * i;<br>} | is a clause used to indicate that a variable in the loop will have a linear relationship with the loop index. This qualifier provides additional information to the compiler about the data access pattern of the variable |
| aligned | #pragma omp simd aligned(x: 32) aligned(y: 32)<br>for (int i = 0; i < n; i++) {<br>y[i] = x[i] + scalar;<br>} | is used to specify that the arrays x and y should be aligned to a 32-byte boundary. |
| order | #pragma omp simd order(concurrent)<br>for (int i = 0; i < n; i++) {<br>// Loop body<br>} | is used to specify that the loop iterations must be executed in a specific order. This can be useful when the loop contains dependencies between iterations, and the order of execution must be controlled to ensure correct results. |

## OMP Declare SIMD

Syntax:

```
#pragma omp declare simd [clause][[,]clause] ...]
```

This clause will create a version of a function which can apply multiple arguments using Single Instruction Multiple Data (SIMD) instructions from a single invocation from a SIMD loop.
In this example we are inform the compiler that a specific function can be safely executed using SIMD instructions. The compiler can then use this information to generate SIMD instructions for loops that call the function.

```
#pragma omp declare simd
        double myfunc(double x, double y);
....
....
....
#pragma omp simd for (int i = 0; i < N; i++) {
        result[i] = myfunc(array[i], scalar);
}
```

## Conclusion

OpenMP Target is an OpenMP directive used to offload parallel computations to an accelerator device, such as a GPU. We have learned various OpenMP directives that can be used with the Target directive, including target teams, target teams distribute, target teams distribute parallel, and target teams loop. These directives allow developers to target specific parallel constructs.

Additionally, we have learned the declare target and declare simd directives, which allow developers to specify functions that can be offloaded to an accelerator device or executed using SIMD instructions, respectively.

Finally, we have covered the OMP Parallel For SIMD combines thread-level parallelism and SIMD vectorization, offering a comprehensive approach to maximizing parallelism.