Introduction to OpenACC `parallel` & `loop` Constructs

Stefano Markidis

Three Key-Points

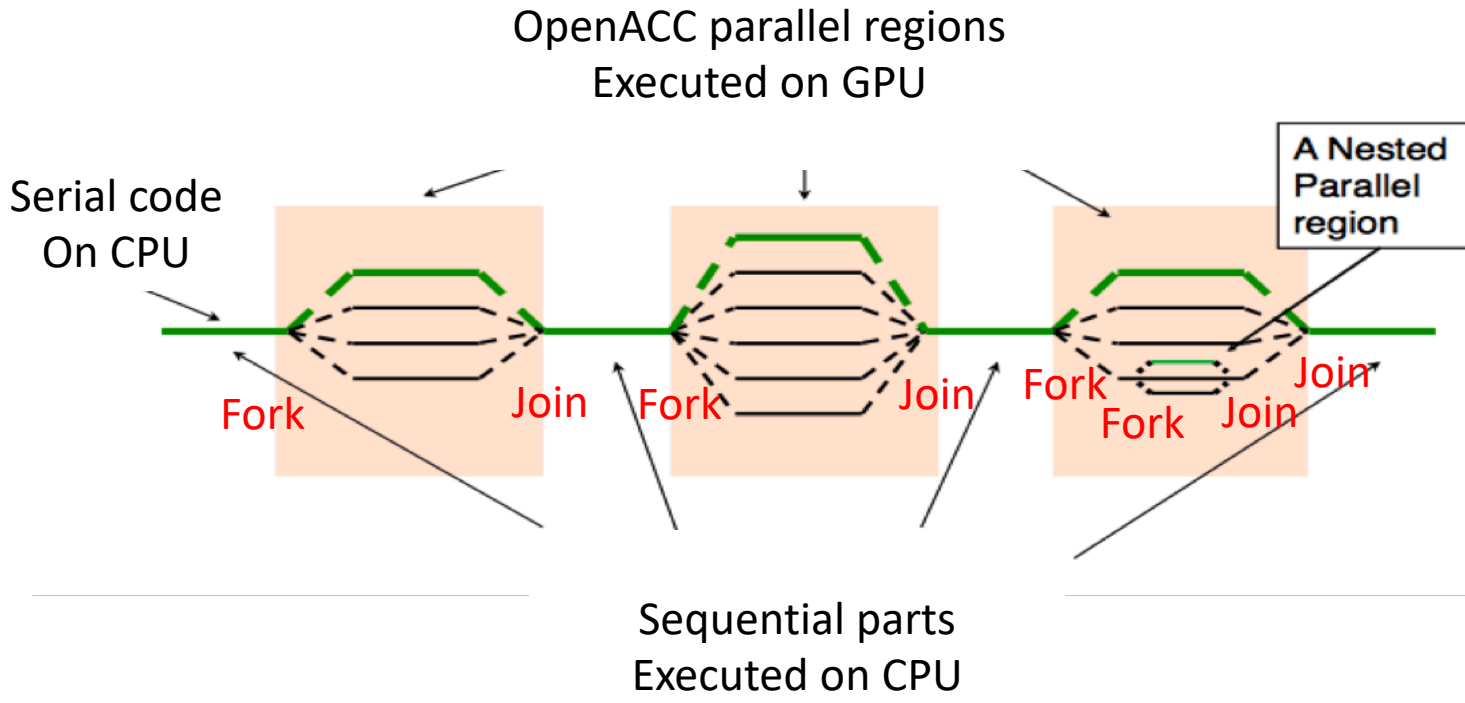
1. The `parallel` construct allows us to specify which code should be executed on the GPU
2. OpenACC uses the fork-join model for parallelism
3. The loop construct allows to use work-sharing in for loops

`parallel` Construct

- The `parallel` construct is a construct to specify which part of the program is to be executed on the accelerator.
- When a program encounters a parallel construct, the execution of the code within the structured block of the construct (also called a parallel region) is moved to the accelerator.

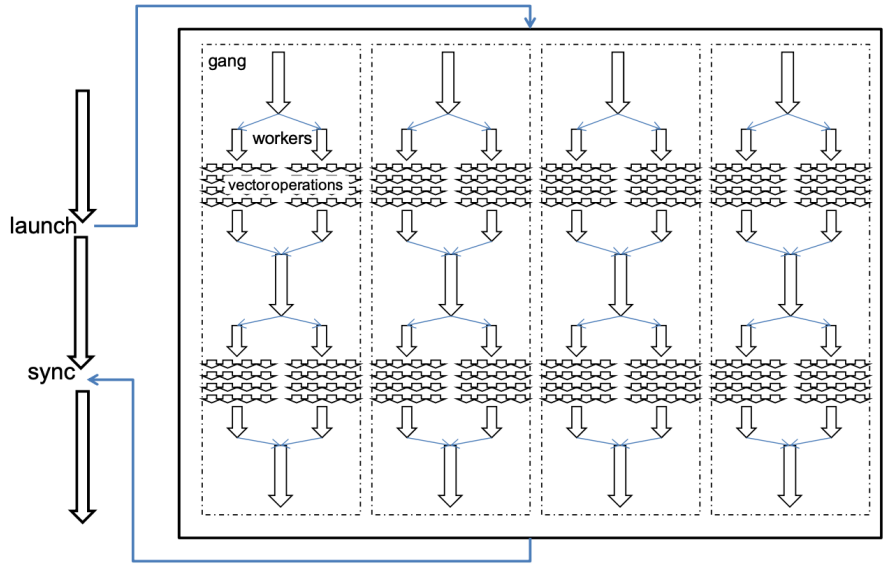
OpenACC Fork-Join Parallelism

- OpenACC uses the fork-join parallelism model
- OpenACC does not assume any synchronization capability on the accelerator, except for thread forking and joining.
- Once work is distributed among the threads within an execution unit, the threads execute in parallel.



How Many GPU Threads? Gangs of Workers

- Gangs of workers on the accelerator are created to execute the parallel region
  - Initially only one worker (a gang lead) within each gang will execute the parallel region
  - Other workers might be deployed when there is more parallel work at an inner level
- The `number of gangs` can be specified by the `num_gangs` clause, and the number of workers within each gang can be specified by the `num_workers` clause.




```
#pragma acc parallel copyout(a) num_gangs(1024) num_workers(32){
    a = 23;
}
```

A total of 1,024 x 32 = 32,768 workers are created. The a = 23 statement will be executed in parallel and redundantly by 1,024 gang leads.

What is Wrong with This?

```
#pragma acc parallel num_gangs(1024) {
    for (int i=0; i<2048; i++) {
        ...
    }
}
```

- The loop will be executed on the accelerator **OK**
- However, we won't get any speedup because all 2,048 iterations will be executed sequentially and redundantly by the 1,024 gang lead **KO**
  - We need work-sharing = each iteration is taken care by one GPU thread

`loop` Construct for Work-Sharing

To get speedup, we need to distribute the 2,048 iterations among the gangs. Need to use the `loop` construct:

```
#pragma acc parallel{
  #pragma acc loop
    for (int i=0; i<2048; i++) {
      ...
    }
}
```


```
#pragma acc parallel loop {
  for (int i=0; i<2048; i++) {
      ...
  }
}
```

Scheduling Work-Sharing

- There are three levels of parallelism expressable in OpenACC:
  - `gang`
  - `worker`
  - `vector`
- They can be placed after loop to guide the compiler.
- This won't make much difference and we are best to let the compiler decide

```
#pragma acc parallel loop gang
  for (i =0; i < 100;i++) {
    a[ i] = b [i ];
  }
```

To Summarize

- The parallel construct allows us to specify which code should be executed on the GPU
- OpenACC uses the fork-join model for parallelism
- The loop construct allows to use work-sharing in for loops