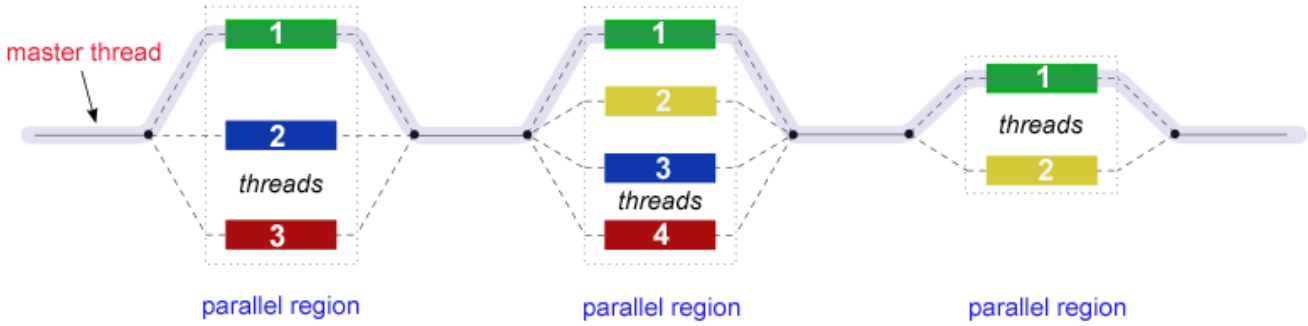
OpenMP basics

OpenMP programs accomplish parallelism exclusively through the use of **threads**. A thread of execution is the smallest unit of processing that can be scheduled by an operating system.Threads exist within the resources of a single `process`. Without the process, they cease to exist. Typically, the **number of threads** match the **number of machine processors/cores**. However, the actual use of threads is up to the application.



OpenMP fork- join (Image credit: llnl.gov OpenMP tutorial)

OpenMP is an explicit (not automatic) programming model, offering the programmer full control over parallelization. It uses the **fork-join** model of parallel execution in addition to **thread based**, **compiler directive** program flows. All OpenMP programs begin as a single process (master thread). The master thread executes sequentially until the first parallel region construct is encountered. the master thread then **creates (fork)** a team of **parallel threads**. When the team threads complete the statements in the parallel region construct, they **synchronize (join) and terminate**, leaving only the master thread.

Three primary OpenMP components are:

- Compiler directives
- Runtime library routines

- Environment Variables

The OpenMP **directives** appear as comments in your source code and are ignored by compilers unless you tell them otherwise. Directives have the following syntax: `sentinel directive-name [clause, ...]`. See an example, below

```
#pragma omp parallel default(shared) private(beta,pi)
```

OpenMP provides several **environment variables** for controlling the execution of parallel code at run-time. For example to set `8` threads while running your code, you would do the following:

```
export OMP_NUM_THREADS=8
```