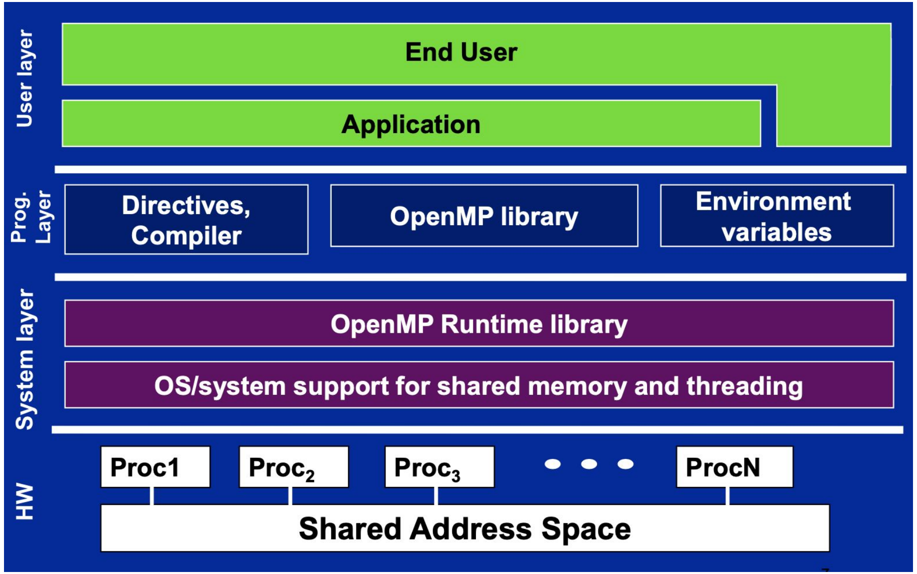
OMP Programming

Computhon2021-1

By: Amro Alabsi Aljundi

Introduction - 1

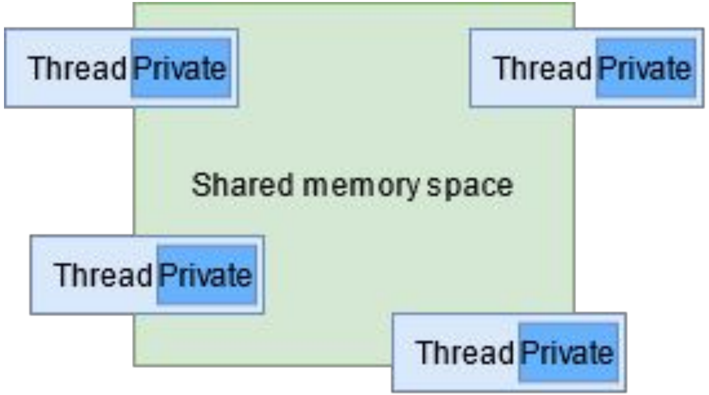
- It is a **parallel programming model** that is very frequently used today.
    - Alternatives to C++ threads or Pthreads.
- **Easy** API composed of **compiler directive** and **library functions.**



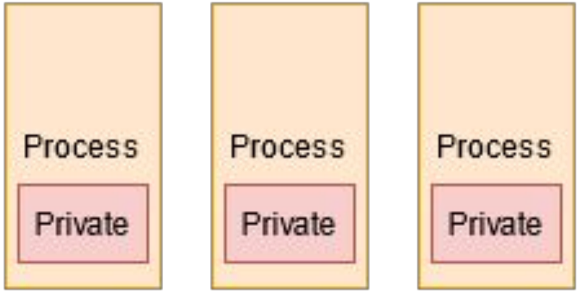
Source: YouTube - Introduction to OpenMP - Tim Mattson (Intel)

Introduction - 2

- Based on the **shared memory multiprocessing (SMP)** where threads share an address space



SMP



Message Passing Interface

API Overview

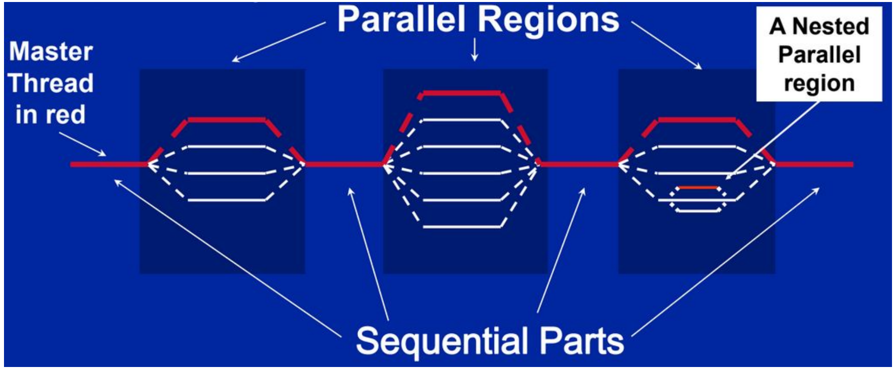
- The API is mostly composed of **compiler directives.**
- These directives act on **structured blocks** (code within `{ }`) that has a single entry and a single exit location

Example:

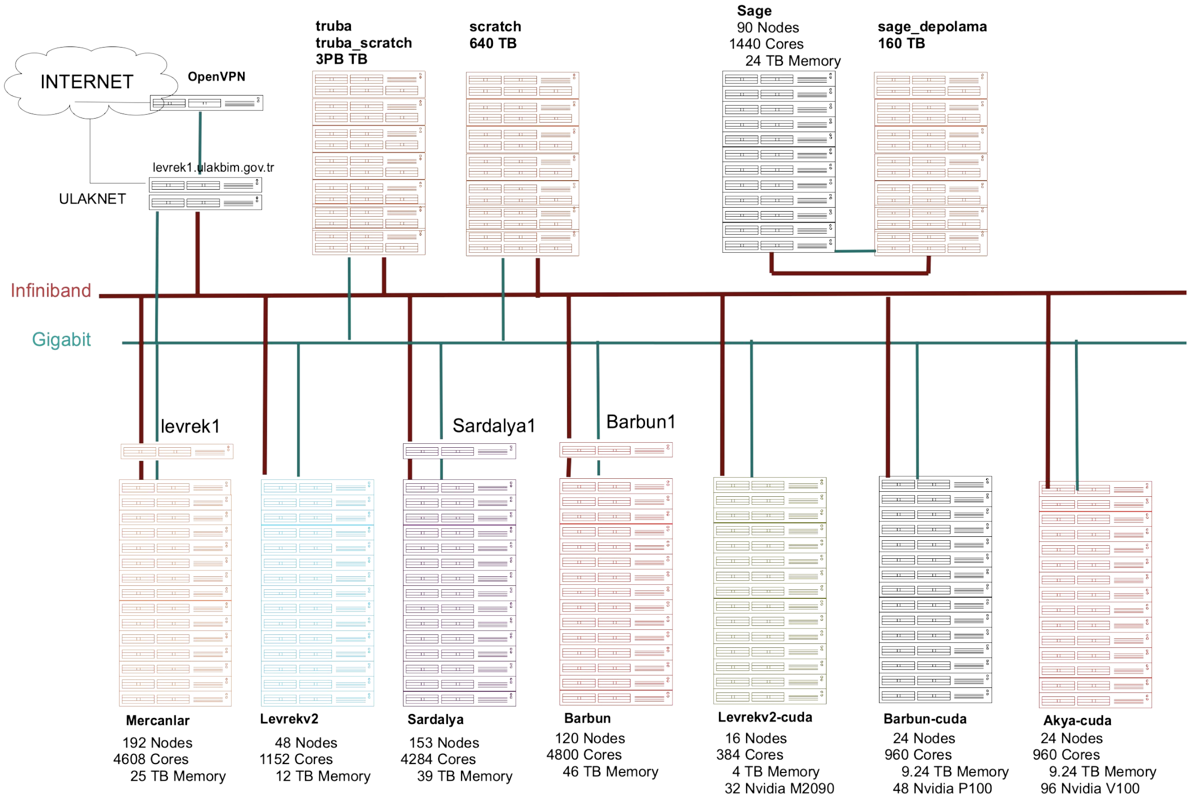
```
#pramga omp parallel num_threads(16)
{
  // entry
  ...
  ...
  // exit
}
```

Execution model: fork-join

- Execution starts with a single **master thread.**
- It will **fork a team of threads** that run in parallel.
- Team of threads will **join** the master thread.



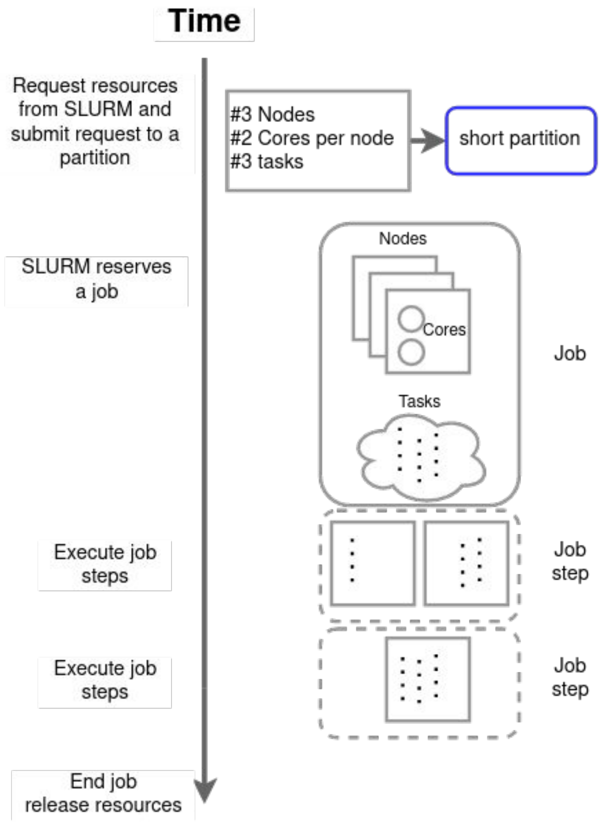
TRUBA Architecture



SLURM Model

1. Request a bunch of resources from SLURM
2. SLURM allocates resources by creating a **job** for a specific **time period.**
    a. Nodes (servers)
    b. Cores
    c. Memory
    d. Tasks (program executions - sort of like processes)
3. Execute work using a job by dispatching **job steps:** program **executions** that use your reserved **tasks**.

Note: the default allocation time period is 2 minutes.



Example 1: creating a parallel region

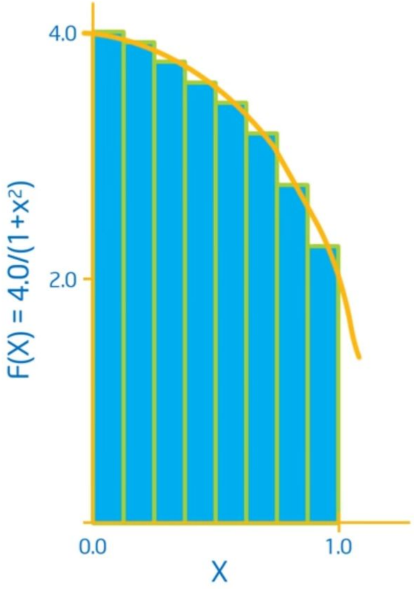
- `#pragma omp parallel` directive
- Setting number of threads
    - Environment variables
    - `omp_set_num_threads()`
    - `num_threads()` clause of `#pragma omp`
- Nested parallelism

Running example: calculating Pi

Calculating the following integral returns the value of Pi

$$\int_0^1 \frac{4.0}{(1+X^2)} \, dx = \pi$$

We can calculate this integral numerically very easily:



Attempt 0: naive parallelization

- Parallelization of the loop
- Function calls
    - omp_num_threads()
    - omp_thread_num()
- Variable scopes

Attempt 1: padding to solve false sharing

- Padding the array solves our issue

Attempt 2: use a lock to control a single shared variable

- We can use a single shared variable and control its access with a lock
- Where we place the lock is very important!!

Attempt 3: use private variables for per-thread sums

- There is no reason to access the shared variable at every iteration.
- Each thread can use a variable in its private memory
- Access to the shared variable can be protected with
  - Lock
  - Critical section
  - Atomic

Attempt 4: use a #pragma omp for

- We can leave the task of carrying out the loop to OMP
- We can control the schedule

Attempt 5: use a #pragma omp for with reduction

- We can also ask OMP to carry out the reduction for us.

Conclusion

- OMP is a parallel programming model for shared-memory systems
- Easy to use API composed of (mainly) compiler directives
- Uses the fork-join execution model

Pi parallelization example:

- Using shared values and false sharing
- Control access to shared variables with locks, critical sections, and atomic
- Worksharing construct: for loop
- Reduction in the for loop