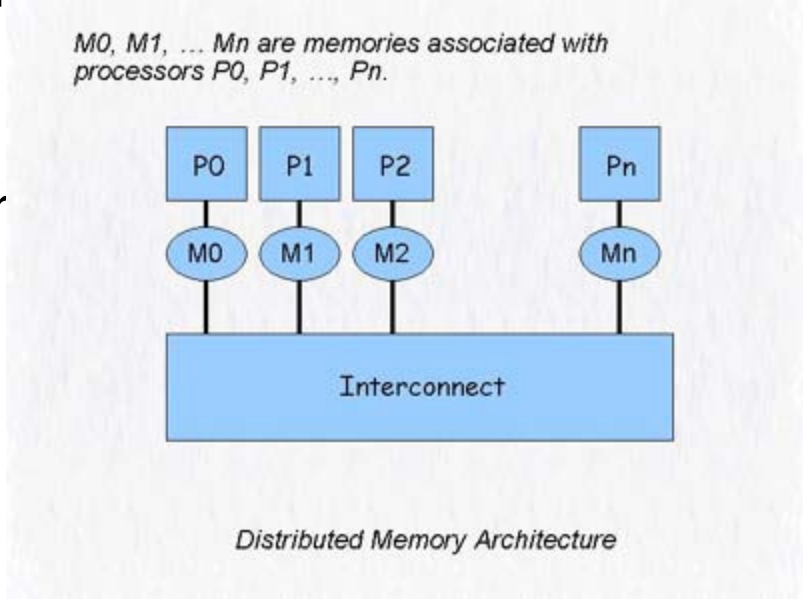
Introduction to OpenMP

Based on NCSA course materials

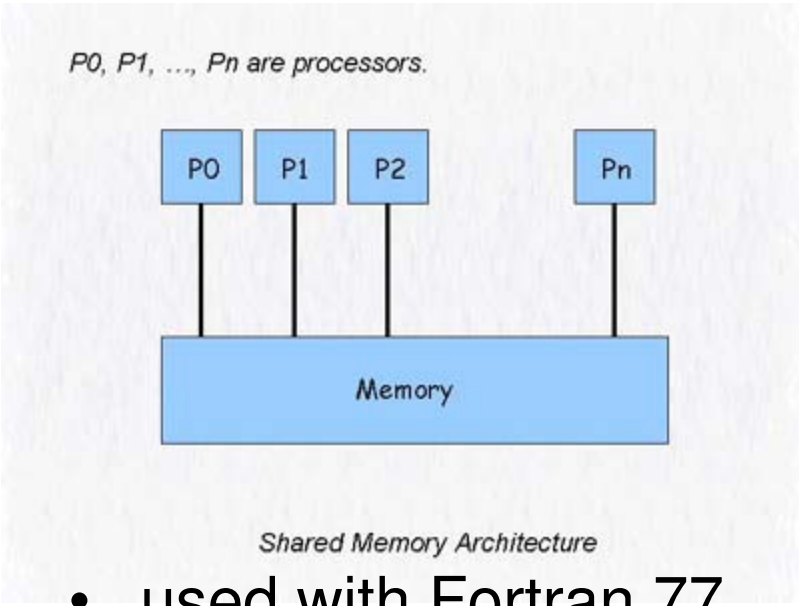
Distributed memory - MPI

- In high performance computing, there are tools that assist programmers with multi-threaded parallel processing on distributed-memory and shared-memory multiproccessor platforms.
- On distributed-memory multiprocessor platforms, each processor has its own memory whose content is not readily available to other processors. Sharing of information among processors is customarily facilitated by message passing using routines from standard message passing libraries such as *MPI*.



Shared memory – Open MP

- On shared-memory multiprocessors, memory among processors can be shared.
- Message passing libraries such as MPI can be, and are, used for the parallel processing tasks.
- However, a directive-based *OpenMP* Application Program Interface (API) has been developed specifically for shared-memory parallel processing.
- OpenMP has broad support from many major computer hardware and software manufacturers.
- Has emerged as the standard for shared-memory parallel computing.



Shared Memory Architecture

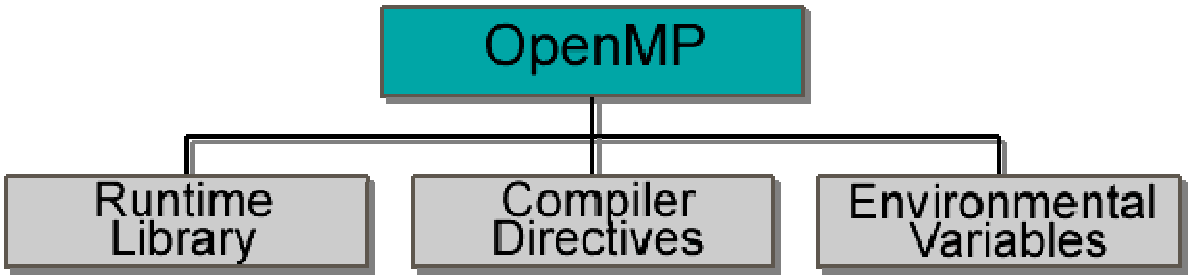
- used with Fortran 77, Fortran 90, C or C++
- for a cluster of single-processor and shared-memory multiprocessor computers, it is possible to use both paradigms in the same application

MLP=MPI+OpenMP

- MPI is used to connect all machines within the cluster to form one virtual machine,
- OpenMP is used to exploit the shared-memory parallelism on individual shared-memory machines within the cluster.
- This approach is commonly referred to as Multi-Level Parallel Programming (MLP).

What is OpenMP

- OpenMP is comprised of three complementary components:
  - a set of *directives* used by the programmer to communicate with the compiler on parallelism.
  - a *runtime library* which enables the setting and querying of parallel parameters such as number of participating *threads* and the thread number.
  - a limited number of *environment variables* that can be used to define runtime system parallel parameters such as the number of threads.



Example

- Code segments that consume substantial CPU cycles frequently involve do loops (or *FOR* loops in C). For loops that are parallelizable, OpenMP provides a rather simple directive to instruct the compiler to parallelize the loop immediately following the directive.

```
call omp_set_num_threads(nthread) !requests
   "nthread" threads
!$OMP PARALLEL DO
   DO i=1,N
         DO j=1,M
                  . . .
         END DO
   END DO
!$OMP END PARALLEL DO
```