Hybrid MPI/OpenMP applications

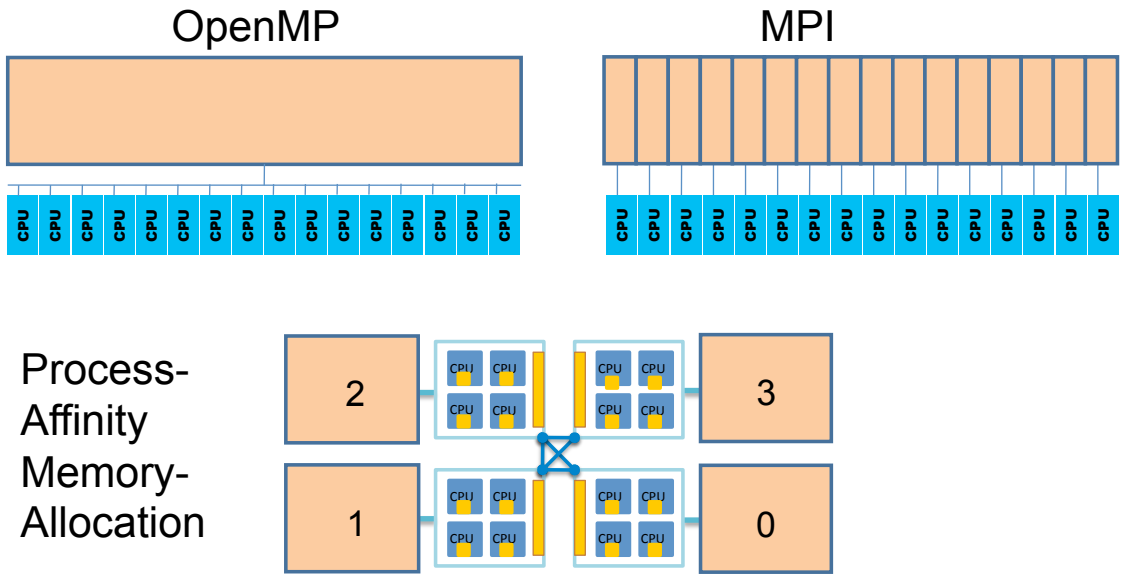
24-Nov-2015

Hybrid MPI/OpenMP

- Why hybrid programming?
  - Current hardware is made of multiple NUMA nodes
  - Eliminates domain decomposition at node level
    - Automatic data coherence within a node (no explicit data send/ receive)
  - Take advantage of faster data movement /synchronization within a node
  - Only profitable if on-node aggregation of MPI processes is faster as a single shared-memory algorithm
    - Faster implementation using shared memory

Hybrid MPI/OpenMP

- Node views

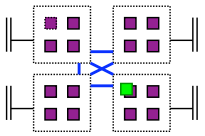


- NUMA systems
  - Where threads/processes and memory allocations go?

Hybrid MPI/OpenMP

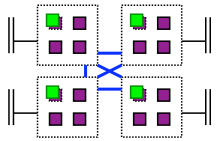
- MPI/thread modes
  - Node level:
    - Single MPI process per node (e.g., machine)
    - Threads share the node memory

1 MPI Tasks
16 Threads/Task



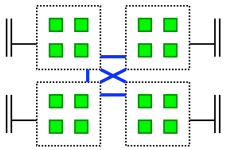
  - Socket level
    - Single MPI process per socket (e.g., processor)
    - Threads share the socket memory

4 MPI Tasks
4Threads/Task



  - Core level
    - One MPI process per core (not a hybrid approach)

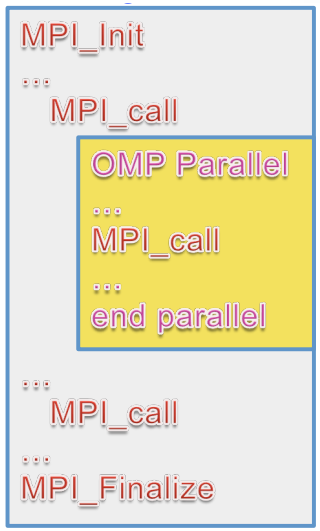
16 MPI Tasks



Hybrid MPI/OpenMP

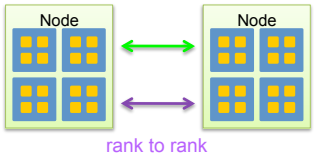
- Selecting a mode
  - *mpirun* can bind processes at **node/socket/core** level
    - Enables a process to use multiple cores
  - Set the number of threads in OpenMP (e.g., set OMP_NUM_THREADS)

- Program Model
  - Start with MPI initialization
    - Create OMP parallel regions within MPI processes
    - Serial regions are the master thread on each process
    - MPI rank is known to all threads
    - *Call MPI library in serial or in parallel regions*
  - Finalize MPI

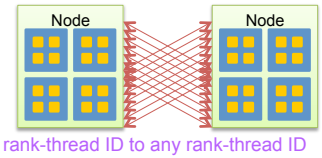


Hybrid MPI/OpenMP

- MPI/thread message modes
  - Single thread
    - MPI from OpenMP serial region to serial region



  - Multiple-thread
    - MPI from OpenMP parallel region to parallel region
    - **Requires a thread-safe MPI implementation**



- Thread-safe MPI
  - Use MPI_Init_thread to select/determine MPI's thread level of support
  - Thread safety is controlled by "provided" types:
    - single, funneled, serialized and multiple
      - Single means there is no multi-threading
      - Funneled means only the master thread calls MPI
      - Serialized means multiple threads can call MPI, but only 1 call can be in progress at a time (serialized).
      - Multiple means MPI is thread safe.

Hybrid MPI/OpenMP

- MPI/thread message modes

Single

```
#include <mpi.h>
int main(int argc, char **argv){
 int rank, size, ierr, i;

 ierr= MPI_Init(&argc,&argv[]);
 ierr= MPI_Comm_rank (...,&rank);
 ierr= MPI_Comm_size (...,&size);
//Setup shared mem, compute & Comm

#pragma omp parallel for
   for(i=0; i<n; i++){
     <work>
   }
// compute & communicate

 ierr= MPI_Finalize();
```

Funneling through master

```
#include <mpi.h>
int main(int argc, char **argv){
 int rank, size, ierr, i;

#pragma omp parallel
{
  #pragma omp barrier
  #pragma omp master
  {
   ierr=MPI_<Whatever>(...)
  }

  #pragma omp barrier

}
}
```

Serialize through single

```
#include <mpi.h>
int main(int argc, char **argv){
int rank, size, ierr, i;
mpi_init_thread(MPI_THREAD_SERIALIZED,
                                iprovided)
#pragma omp parallel
{
  #pragma omp barrier
  #pragma omp single
  {
   ierr=MPI_<Whatever>(...)
  }

  //pragma omp barrier

}
}
```

- Thread-rank communication in multiple-thread MPI

```
...
 nthreads=OMP_GET_NUM_THREADS()
 ithread   =OMP_GET_THREAD_NUM()
 call pwork(ithread, irank, nthreads, nranks...)
 if(irank == 0) then
   call mpi_send(ithread,1,MPI_INTEGER, 1,  ithread, MPI_COMM_WORLD, ierr)
 else
   call mpi_recv(          j,1,MPI_INTEGER, 0,  ithread, MPI_COMM_WORLD, istatus,ierr)
```

Communicate between ranks.

Threads use tags to differentiate.

Hybrid MPI/OpenMP

- Lab: hybrid prime filter
  - Pipeline of MPI processes
  - Each MPI process provides a local farm (parallel for *workshare*)

```
if (myrank==0) {

      PrimeServer *ps1 = new PrimeServer();
      ps1->minitFilter(1,SMAXP/3,SMAXP);

      int *ar = new int[pack/2];
      for(int i=0; i<10; i++) {
          generate(i*pack, (i+1)*pack, ar);
          ps1->mprocess(ar,pack/2);            <= OpenMP code
          ... // send ar to pid 1
     }

} else if (myrank==1) {

      PrimeServer *ps2 = new PrimeServer();
      ps2->minitFilter(SMAXP/3+1,2*SMAXP/3,SMAXP);

      int *ar = new int[pack/2];
      for(int i=0; i<10; i++) {
         ... // receive pack ar from pid 0
          ps2->mprocess(ar,pack/2);            <= OpenMP code
         ... // send ar to pid 2
      }

} else {

}
```