Parallel and Distributed Computing

CS3006

Lecture 18

Hybrid Programming (MPI + OpenMP)

15th May 2024

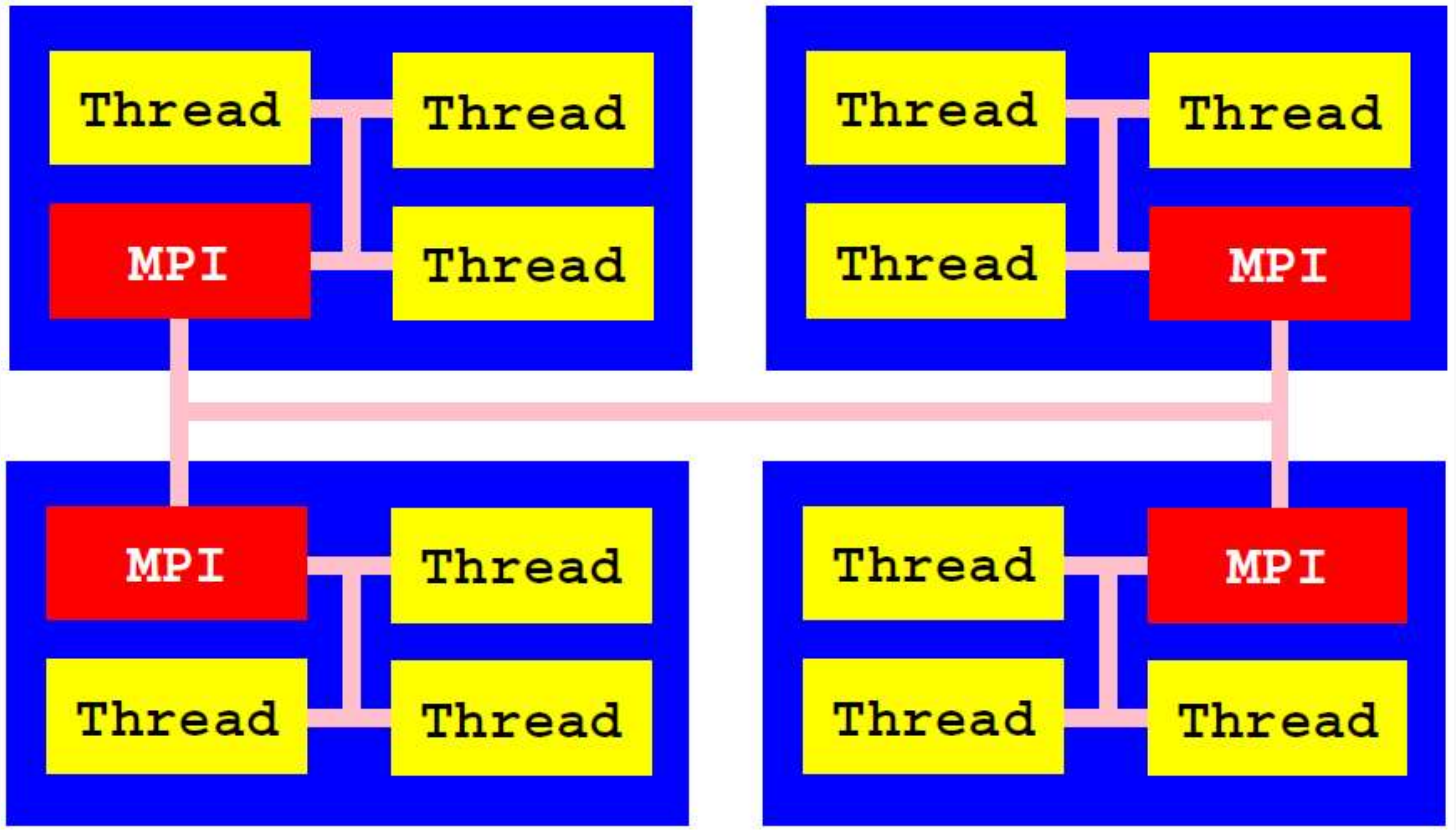
Dr. Rana Asif Rehman

Hybrid Programming with MPI and OpenMP

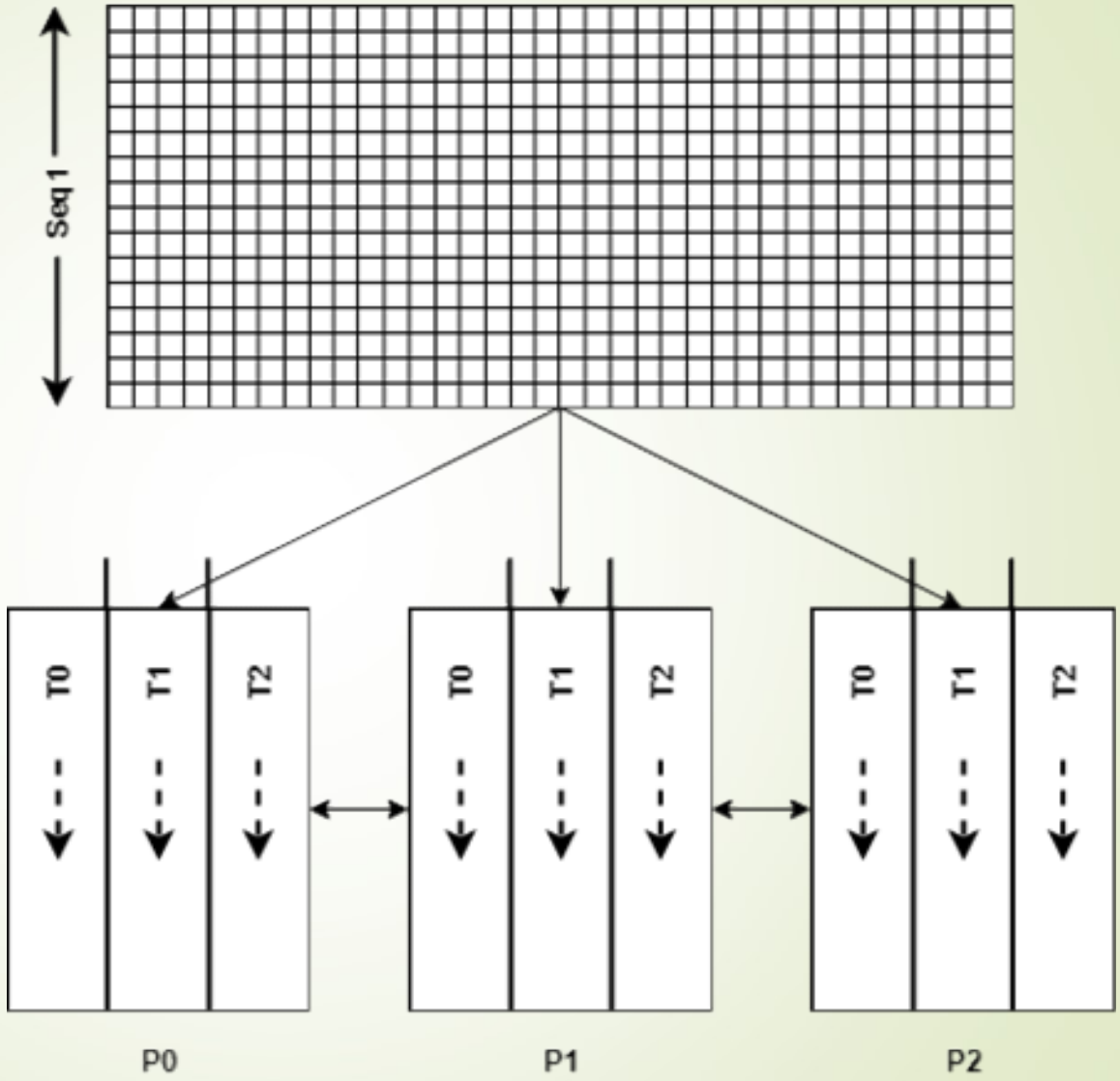
- Consider your cluster of workstations consists of different multicore processors.
- Launching a single MPI process for each of the workstations, may not optimally use the available resources.

- We can combine **MPI and OpenMP** to increase utilization of the resources by:
  - Using single MPI process per workstation
  - And generating different number of threads in each workstation.
  - Each workstation can have different number of threads, according to their infrastructures.

Hybrid Programming with MPI and OpenMP



Single MPI process per workstation and multiple threads



Hybrid Programming with MPI and OpenMP [Use cases]

- The simplest and safe way to combine MPI with OpenMP is to **never use the MPI calls inside the OpenMP parallel regions**.
- MPI level communication must only be done by the master thread.

```
main(int argc, char **argv) {
...
MPI_Init(&argc, &argv);
... // master thread only --> MPI calls here
#pragma omp parallel
{
.     .. // team of threads --> no MPI calls here
}
... // master thread only --> MPI calls here
MPI_Finalize();
}
```

Hybrid Programming with MPI and OpenMP [Helloworld.c program]

```
MPI_Init(&argc, &argv);
    int P , name_len;
    MPI_Comm_size(MPI_COMM_WORLD, &P);
    int process_id;
    MPI_Comm_rank(MPI_COMM_WORLD, &process_id);
    char processor_name[MPI_MAX_PROCESSOR_NAME];
    MPI_Get_processor_name(processor_name, &name_len);
omp_set_dynamic(0);
    omp_set_num_threads(omp_get_num_procs());
    int thread_id;
#pragma omp parallel private(thread_id)
{
 thread_id = omp_get_thread_num();
printf("From Process#%d thread_id %d out of %d mapped on
processor:%s  Hello Hybrid\n", process_id, thread_id,
omp_get_num_threads(), processor_name);
}
MPI_Finalize();
```

Hybrid Programming with MPI and OpenMP [Helloworld.c program]

Important: At header files include both mpi.h and omp.h

Compiling the program:

mpicc -fopenmp ./helloworld.c -o ./a.out

Executing the program:

mpiexec -np 2 ./a.out

```
husnain8721@Haier-PC:/mnt/c/hybrid$ mpiexec -np 2 ./a.out
From Process#0 thread_id 0 out of 4 mapped on processor:Haier-PC  Hello Hybrid
From Process#0 thread_id 1 out of 4 mapped on processor:Haier-PC  Hello Hybrid
From Process#1 thread_id 3 out of 4 mapped on processor:Haier-PC  Hello Hybrid
From Process#1 thread_id 1 out of 4 mapped on processor:Haier-PC  Hello Hybrid
From Process#1 thread_id 2 out of 4 mapped on processor:Haier-PC  Hello Hybrid
From Process#0 thread_id 3 out of 4 mapped on processor:Haier-PC  Hello Hybrid
From Process#1 thread_id 0 out of 4 mapped on processor:Haier-PC  Hello Hybrid
From Process#0 thread_id 2 out of 4 mapped on processor:Haier-PC  Hello Hybrid
```

Hybrid Programming with MPI and OpenMP

- Till now we have only seen solutions that don't require inter-workstation (OR MPI level) communication in the parallel region

- What if we try to use MPI calls within openmp parallel regions?
  - Default calls for MPI-1 primitive routines are not thread safe
  - May result in strange outputs

- So, solution?
  - Use MPI_Init_thread() with **MPI_THREAD_MULTIPLE** as support level instead of simple MPI_init() call for starting the program

Hybrid Programming with MPI and OpenMP


```
int MPI_Init_thread(int *argc, char ***argv, int required, int *provided)
```

- MPI_Init_thread() initializes the MPI execution environment (similar to MPI_Init())
- defines the support level for multithreading:
  - **required** is the aimed support level
  - **provided** is the support level provided by the MPI implementation

Available Support levels:

- MPI_THREAD_SINGLE – only one thread will execute (the same as initializing the environment with MPI_Init())
- MPI_THREAD_FUNNELED – only the master thread can make MPI calls
- MPI_THREAD_SERIALIZED – all threads can make MPI calls, but only one thread at a time can be in such state
- MPI_THREAD_MULTIPLE – all threads can make simultaneous MPI calls without any constraints

Hybrid Programming with MPI and OpenMP

MPI_THREAD_FUNNELED

- With support level MPI_THREAD_FUNNELED only the master thread canmake MPI calls.
- One way to ensure this is to protect the MPI calls with the **omp master** directive.

```
#pragma omp parallel
{
    ...
    #pragma omp barrier // explicit barrier at entrance
    #pragma omp master // only the master thread makes the MPI call
    mpi_call();
    #pragma omp barrier // explicit barrier at exit
    ...
}
```

Hybrid Programming with MPI and OpenMP

MPI_THREAD_SERIALIZED

- Only one thread at a time will make calls to the MPI library, but all threads are eligible to make such calls as long as they do not do so at the same time

```
#pragma omp parallel private(tid) num_threads(2)
{
      tid = omp_get_thread_num();
      if(tid== 0){  mpi_call(); }
   #pragma omp barrier // barrier so that other threads don't enter next mpi call
      if(tid== 1){  mpi_call(); }
      #pragma omp barrier // explicit barrier
     ...
}
```

Hybrid Programming with MPI and OpenMP

MPI_THREAD_MULTIPLE

- All the threads can make simultaneous MPI calls without any constraints
- No need of any barrier synchronizations among the threads
- MPI library is then responsible for thread safety within that library, and for any libraries that it in turn uses

```
#pragma omp parallel private(tid) num_threads(2)
{
      ….
       tid = omp_get_thread_num();
      mpi_call();  //all the threads will call this mpi routine parallely
     ...
}
```

Hybrid Programming with MPI and OpenMP

MPI_THREAD_MULTIPLE

- How to identify the thread number that is involved in the communication?
- You can pass **threadID** as message **tag.** The receiving process then can identify each message from the source uniquely.

```
#pragma omp parallel private(tid) num_threads(2)
{
…
      tid= omp_get_thread_num();
      If (my_rank == 0) // MPI process 0 sends 2 messages
            MPI_Send(a, 1, MPI_INT, 1, tid, MPI_COMM_WORLD);
      else if (my_rank == 1) // MPI process 1 receives  2 messages
      MPI_Recv(b, 1, MPI_INT, 0, tid, MPI_COMM_WORLD, &status);
…
}
```

Hybrid Programming with MPI and OpenMP

hybrid_multiple_support_sendRecv.c

- Program uses MPI_THREAD_MULTIPLE as level of support
- The program basically implements a naïve hybrid one-to-all broadcast of a buffer containing 4 integers.
- Source process uses 4 threads, where each thread is responsible of sending one integer to all the other processes
- While receiving processes also use 4 threads, each which receives a single integer from the source process and copies it into correct position in the receive buffer.

Hybrid Programming with MPI and OpenMP

hybrid_multiple_support_sendRecv.c OUTPUT

- For 2 MPI processes and sendbuff=[2,4,6,8]

process_id:0-->thread#0 is sending 2 to process 1
process_id:0-->thread#1 is sending 4 to process 1
process_id:0-->thread#2 is sending 6 to process 1
process_id:0-->thread#3 is sending 8 to process 1
process-id:1 Received 2 from thread#0 of source
process-id:1 Received 4 from thread#1 of source
process-id:1 Received 6 from thread#2 of source
process-id:1 Received 8 from thread#3 of source
recvbuff=2 4 6 8

Hybrid Programming with MPI and OpenMP

hybrid_multiple_support_sendRecv.c OUTPUT

- For 4 MPI processes **[Sorce Process output]**

```
process_id:0-->thread#0 is sending 2 to process 1
process_id:0-->thread#0 is sending 2 to process 2
process_id:0-->thread#0 is sending 2 to process 3

process_id:0-->thread#1 is sending 4 to process 1
process_id:0-->thread#1 is sending 4 to process 2
process_id:0-->thread#1 is sending 4 to process 3

process_id:0-->thread#2 is sending 6 to process 1
process_id:0-->thread#2 is sending 6 to process 2
process_id:0-->thread#2 is sending 6 to process 3

process_id:0-->thread#3 is sending 8 to process 1
process_id:0-->thread#3 is sending 8 to process 2
process_id:0-->thread#3 is sending 8 to process 3
```

Hybrid Programming with MPI and OpenMP

hybrid_multiple_support_sendRecv.c OUTPUT

- For 4 MPI processes **[receiving processes output]**

```
process-id:1 Received 2 from thread#0 of source
process-id:1 Received 4 from thread#1 of source
process-id:1 Received 6 from thread#2 of source
process-id:1 Received 8 from thread#3 of source

process-id:2 Received 2 from thread#0 of source
process-id:2 Received 4 from thread#1 of source
process-id:2 Received 6 from thread#2 of source
process-id:2 Received 8 from thread#3 of source

process-id:3 Received 2 from thread#0 of source
process-id:3 Received 6 from thread#2 of source
process-id:3 Received 4 from thread#1 of source
process-id:3 Received 8 from thread#3 of source
recvbuff=2      4       6       8
```

Hybrid Programming with MPI and OpenMP

hybrid_multiple_support_sendRecv.c OUTPUT

- Actual output snippet

```
usnain8721@Haier-PC:/mnt/c/hybrid$ mpiexec -np 4 ./a
rocess_id:0-->thread#3 is sending 8 to process 1
process_id:0-->thread#3 is sending 8 to process 2
process_id:0-->thread#3 is sending 8 to process 3
process_id:0-->thread#0 is sending 2 to process 1
process_id:0-->thread#0 is sending 2 to process 2
process_id:0-->thread#0 is sending 2 to process 3
process_id:0-->thread#1 is sending 4 to process 1
process_id:0-->thread#1 is sending 4 to process 2
process_id:0-->thread#1 is sending 4 to process 3
process-id:1 Received 8 from thread#3 of source
rocess-id:2 Received 8 from thread#3 of source
rocess-id:2 Received 2 from thread#0 of source
rocess-id:2 Received 4 from thread#1 of source
rocess-id:3 Received 4 from thread#1 of source
rocess_id:0-->thread#2 is sending 6 to process 1
process_id:0-->thread#2 is sending 6 to process 2
process_id:0-->thread#2 is sending 6 to process 3
process-id:1 Received 2 from thread#0 of source
rocess-id:1 Received 4 from thread#1 of source
rocess-id:1 Received 6 from thread#2 of source
rocess-id:2 Received 6 from thread#2 of source
rocess-id:3 Received 2 from thread#0 of source
rocess-id:3 Received 8 from thread#3 of source
rocess-id:3 Received 6 from thread#2 of source
ecvbuff=2       4       6       8
```

Questions

References

1. Kumar, V., Grama, A., Gupta, A., & Karypis, G. (2017). *Introduction to parallel computing*. Redwood City, CA: Benjamin/Cummings.