Hybrid MPI - OpenMP

Dr. Dimitris Dellis

GRNET

Athens, 21 Nov. 2018

About Hybrid MPI-OpenMP

- MPI
  - Each process usually keeps portion of data.
  - Need to communicate, send/accumulate data.
  - Can work across nodes.
  - Overheads of Network and temporary communication buffers.
  - In some cases temporary buffers are bigger than main program data.
- OpenMP
  - One process keeps all data in memory.
  - Only in parallel regions use extra memory.
  - Limited on a single node.

- No network overhead
- Start/end parallel regions overhead.

- Hybrid
  - Both.
  - Discuss Pros and cons.

From MPI to Hybrid

- We'll use the sum of squares MPI example

```
#include <stdio.h>
#include "mpi.h"
int main(int argc, char *argv[]) {
        int my_rank, p, i, res, finres, start, end, num, N, source,
            target, tag1 = 5, tag2 = 6;
        MPI_Status status;
        MPI_Init(&argc, &argv);
        MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
        MPI_Comm_size(MPI_COMM_WORLD, &p);
    double starttime,endtime;
        if (my_rank == 0)  {
                printf("Enter last number: ");
                scanf("%d", &N);
        starttime=MPI_Wtime();
                for (target = 1; target < p; target++) {
                        MPI_Send(&N, 1, MPI_INT, target, tag1, MPI_COMM_WORLD);
                }
         } else {
```

```
        MPI_Recv(&N, 1, MPI_INT, 0, tag1, MPI_COMM_WORLD, &status);
}
res   = 0;
num   = N / p;
start = (my_rank * num) + 1;
end   = start + num;
for (i = start; i < end; i++) {
        res += (i * i);
}
if (my_rank != 0)  {
        MPI_Send(&res, 1, MPI_INT, 0, tag2, MPI_COMM_WORLD);
} else {
        finres = res;
        printf("\nResult of process %d: %d\n", my_rank, res);
        for (source = 1; source < p; source++)  {
                MPI_Recv(&res, 1, MPI_INT, source, tag2, MPI_COMM_WORLD, &status);
                finres += res;
                printf("\nResult of process %d: %d\n", source, res);
        }
        printf("\n\n\nFinal result: %d\n", finres);
        printf("\n\n\nFinal result: %d\n", finres);
endtime=MPI_Wtime();
printf("Run took : %lf [sec]\n",endtime-starttime);
}
```

```
        MPI_Finalize();
        return(0);
}
```

From MPI to Hybrid

- Discuss possiblities for Hybrid MPI - OpenMP

From MPI to Hybrid

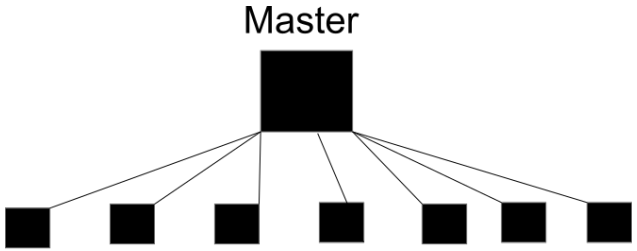
```
#include <stdio.h>
#include "mpi.h"
#include <omp.h>
int main(int argc, char *argv[]) {
      int my_rank, p, i, res, finres, start, end, num, N, source,
          target, tag1 = 5, tag2 = 6;
      MPI_Status status;
      MPI_Init(&argc, &argv);
      MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
      MPI_Comm_size(MPI_COMM_WORLD, &p);
        double starttime,endtime;
      if (my_rank == 0)  {
            printf("Enter last number: ");
            scanf("%d", &N);
                starttime=MPI_Wtime();
            for (target = 1; target < p; target++) {
                  MPI_Send(&N, 1, MPI_INT, target, tag1, MPI_COMM_WORLD);
            }
       } else {
            MPI_Recv(&N, 1, MPI_INT, 0, tag1, MPI_COMM_WORLD, &status);
      }
```

```
  res    = 0;
  num    = N / p;
  start = (my_rank * num) + 1;
  end    = start + num;
#pragma omp parallel for reduction(+:res)
  for (i = start; i < end; i++) {
        res += (i * i);
  }
  if (my_rank != 0)  {
        MPI_Send(&res, 1, MPI_INT, 0, tag2, MPI_COMM_WORLD);
  } else {
        finres = res;
        printf("\nResult of process %d: %d\n", my_rank, res);
        for (source = 1; source < p; source++)  {
              MPI_Recv(&res, 1, MPI_INT, source, tag2, MPI_COMM_WORLD, &status);
              finres += res;
              printf("\nResult of process %d: %d\n", source, res);
        }
    endtime=MPI_Wtime();
        printf("\n\n\nFinal result: %d\n", finres);
    printf("Run took : %lf [sec]\n",endtime-starttime);
  }
  MPI_Finalize();
  return(0);
```

```
}
```

Topologies

- Up to now, communication use master-slaves (or farm)



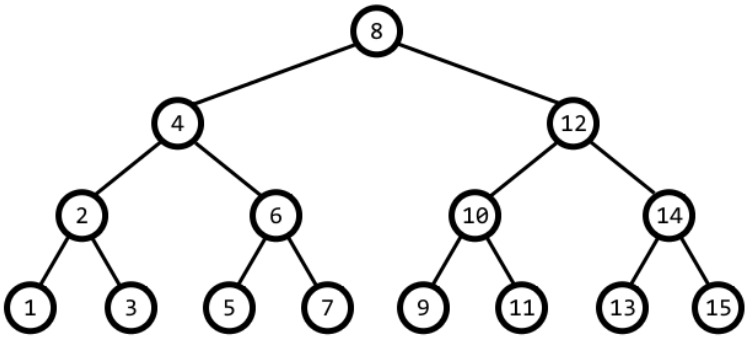
method

- Typical Source example

```
if ( me == 0 ) {
     MPI_Send(..........);
   }else{
     MPI_Recv(..........);
}
```

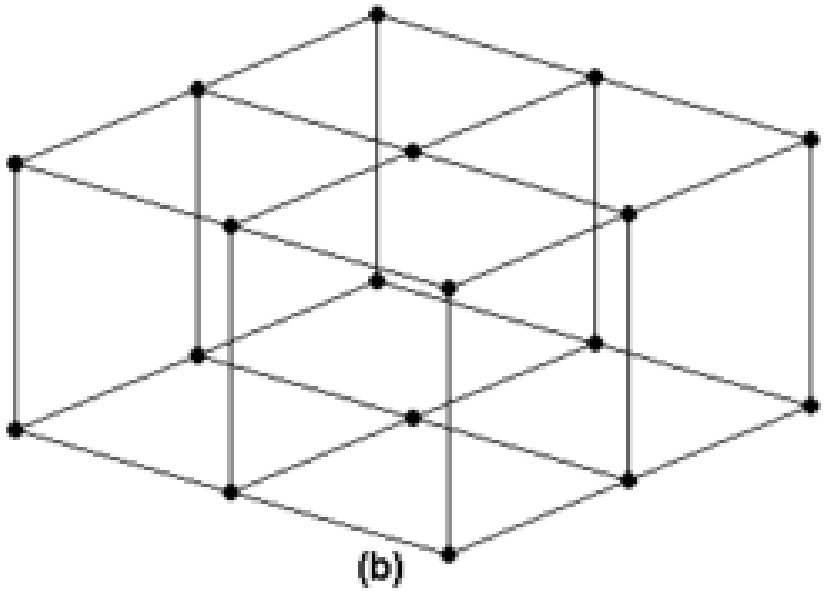
- What if slave 1 takes more time than the rest ?
- Master waits slave 1 to complete and then tries next slaves
- If N procs in communicator, N-1 pairs of Send/Recv.
- Other implementations ?

Topologies: Binary Tree



- Binary Tree of order N includes $2^N - 1$ tasks - N is the number of levels.
- Rend/Recv runs partially in parallel.
- Collection of data is completed in less steps.

Topologies: 2D/3D Grid



- Collection of data in parallel.

Topologies: Other

- Ring
- HyperCube
- 2-6 D Torus
- Many collective operations functions use some high level topology to collect/broadcast data.

Topologies

- Usually topology is Problem driven, commonly 2 or 3D Grid
- Number of tasks can not be any arbitrary number.
- Spatial Decomposition is performed based on number of Tasks.
- With some numbers of Task may lead to very bad decompoositions.
- Example : KNL with 68 cores with 3D decomposition can be only 1x4x17 or 2x2x17

How Hybrid can help in performance

- Supposing that code runs with similar speed if it is pure MPI or hybrid MPI-OpenMP.
- Decreasing the number of Tasks giving more threads per task, may lead to better spatial decompositions.
- Increasing the number of Tasks in an MPI application
  - Decreases the computation per task
  - Increase the communication cost
- If an application reaches the maximum scalability (i.e. number of tasks) using hybrid may extent the scaling of application by increase of computation and decrease communication.
- When computation density is not similar across tasks, using hybrid may improve dead time of tasks with low computational density.

Questions ?