MPI

John H. Osorio Ríos

Abstract (1/1)

- Real HPC Applications make use of clusters.
- You need combine CUDA+OpenMP+MPI, or their counterparts.

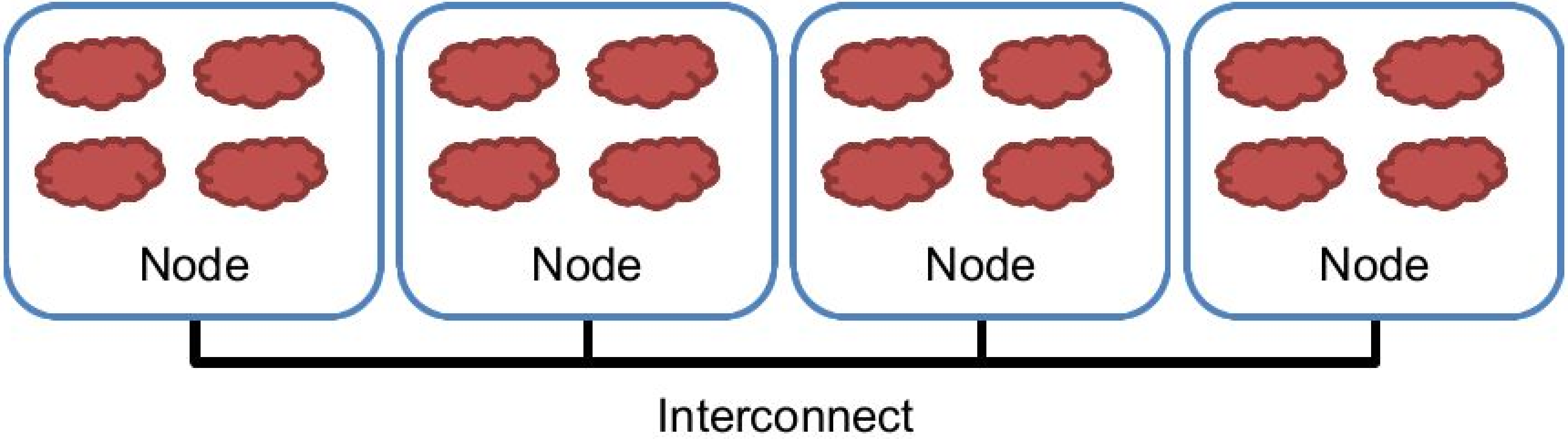
Background (1/3)

- Before 2009 there was practically no top supercomputer using GPUs
- Many of the top supercomputers use both CPUs and GPUs in each node
- Look at Green 500 and TOP 500 list

Background (2/3)

- The dominating programming interface for cluster today is MPI
- MPI permits communication between processes running in differents nodes
- MPI assumes a distributed memory model.

Background (3/3)



MPI Basics (1/3)

- MPI programs are based on the SPMD parallel execution model. All MPI processes execute the same program.

MPI Basics (2/3)

- int MPI_Init (int*argc, char***argv)
  - Initialize MPI
- int MPI_Comm_rank (MPI_Comm comm, int *rank)
  - Rank of the calling process in group of comm
- int MPI_Comm_size (MPI_Comm comm, int *size)
  - Number of processes in the group of comm
- int MPI_Comm_abort (MPI_Comm comm)
  - Terminate MPI comminication connection with an error flag
- int MPI_Finalize ( )
  - Ending an MPI application, close all resources

MPI Basics (3/3)

```c
#include "mpi.h“

int main(int argc, char *argv[]) {
    int pad = 0, dimx = 480+pad, dimy = 480, dimz = 400, nreps = 100;
    int pid=-1, np=-1;

    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &pid);
    MPI_Comm_size(MPI_COMM_WORLD, &np);

    if(np< 3) {
        if(0 == pid) printf(“Needed 3 or more processes.\n");
        MPI_Abort( MPI_COMM_WORLD, 1 ); return 1;
    }
    if(pid < np - 1)
        compute_process(dimx, dimy, dimz/ (np - 1), nreps);
    else
        data_server( dimx,dimy,dimz);

    MPI_Finalize();
    return 0;
}
```

MPI Point to Point Communication (1/4)

- int MPI_Send (void *buf, int count, MPI_Datatype datatype, int dest, int tag, MPI_Comm comm)
  - Buf: starting address of send buffer (pointer)
  - Count: Number of elements in send buffer (nonnegative integer)
  - Datatype: Datatype of each send buffer element (MPI_Datatype)
  - Dest: Rank of destination (integer)
  - Tag: Message tag (integer)
  - Comm: Communicator (handle)

MPI Point to Point Communication (2/4)

- int MPI_Recv (void *buf, int count, MPI_Datatype datatype, int source, int tag, MPI_Comm comm, MPI_Status *status)
  - buf: starting address of receive buffer (pointer)
  - Count: Maximum number of elements in receive buffer (integer)
  - Datatype: Datatype of each receive buffer element (MPI_Datatype)
  - Source: Rank of source (integer)
  - Tag: Message tag (integer)
  - Comm: Communicator (handle)
  - Status: Status object (Status)

MPI Point to Point Communication (3/4)

MPI datatype	C equivalent
MPI_SHORT	short int
MPI_INT	int
MPI_LONG	long int
MPI_LONG_LONG	long long int
MPI_UNSIGNED_CHAR	unsigned char
MPI_UNSIGNED_SHORT	unsigned short int
MPI_UNSIGNED	unsigned int
MPI_UNSIGNED_LONG	unsigned long int
MPI_UNSIGNED_LONG_LONG	unsigned long long int
MPI_FLOAT	float
MPI_DOUBLE	double
MPI_LONG_DOUBLE	long double
MPI_BYTE	char

MPI Point to Point Communication (4/4)

```c
// Find out rank, size
int world_rank;
MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);
int world_size;
MPI_Comm_size(MPI_COMM_WORLD, &world_size);

int number;
if (world_rank == 0) {
    number = -1;
    MPI_Send(&number, 1, MPI_INT, 1, 0, MPI_COMM_WORLD);
} else if (world_rank == 1) {
    MPI_Recv(&number, 1, MPI_INT, 0, 0, MPI_COMM_WORLD,
             MPI_STATUS_IGNORE);
    printf("Process 1 received number %d from process 0\n",
           number);
}
```

Bibliography (1/1)

- http://mpitutorial.com/beginner-mpi-tutorial/
- Programming Massively Parallel Processors. Chapter 19

THANKS

john@sirius.utp.edu.co