Lab 11 of SP19-BL-ENGR-E599-30563

Lab 11 will introduce the concept and hands-on of distributed programs on HPC cluster.

Goal

- Understand the workflow of MPI on multiple nodes

Deliverables

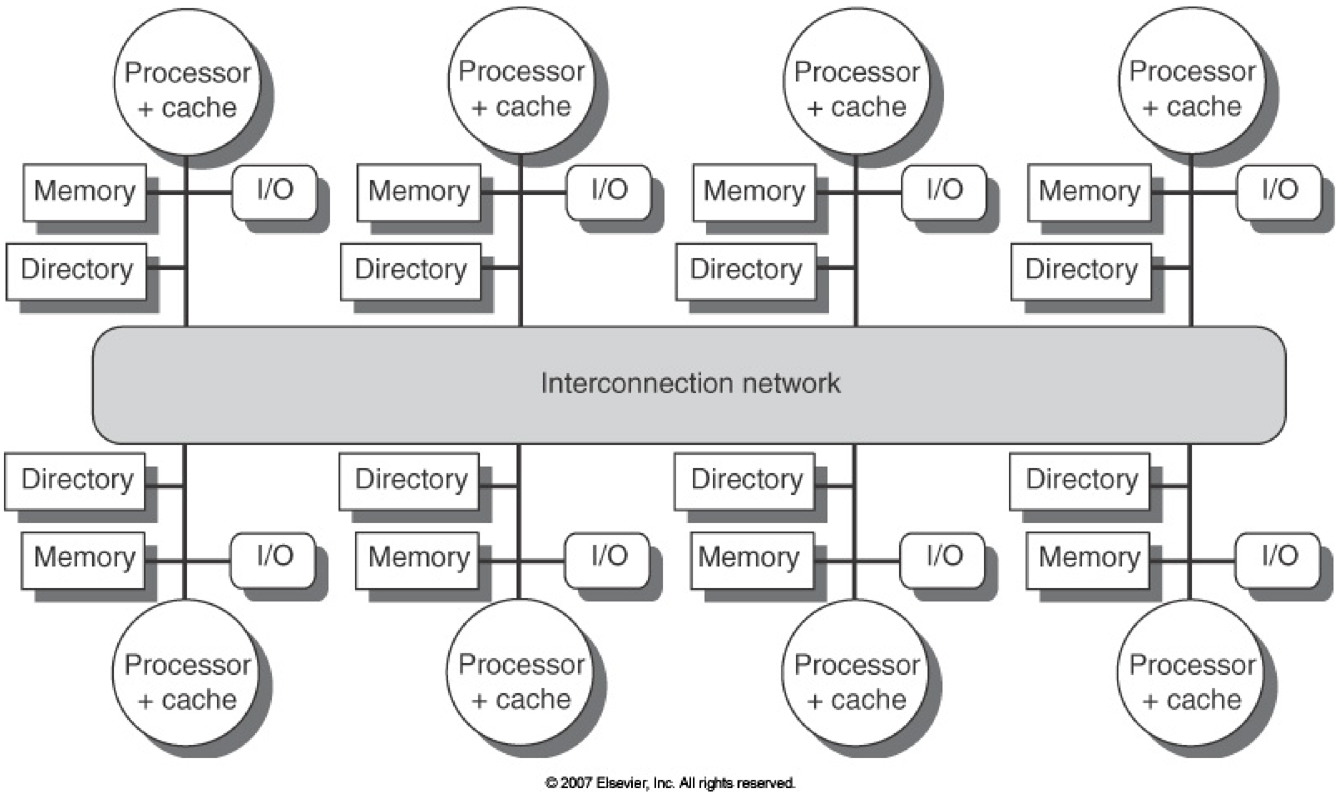
- run the MPI sample with 2,4,8 processes, and submit the execution time in a table

Evaluation

Lab participation: credit for 1 point based upon a successful completion of the lab tasks

HPC/DC Cluster and Distributed System

Both of HPC and DC (Data center) clusters are made of connected machine nodes. Since each node (machine) has its own memory resource, the total memory address is distributed from the perspective of the whole system.



The Architecture of a Distributed System

To synchronize the data within memory (i.e., communication)

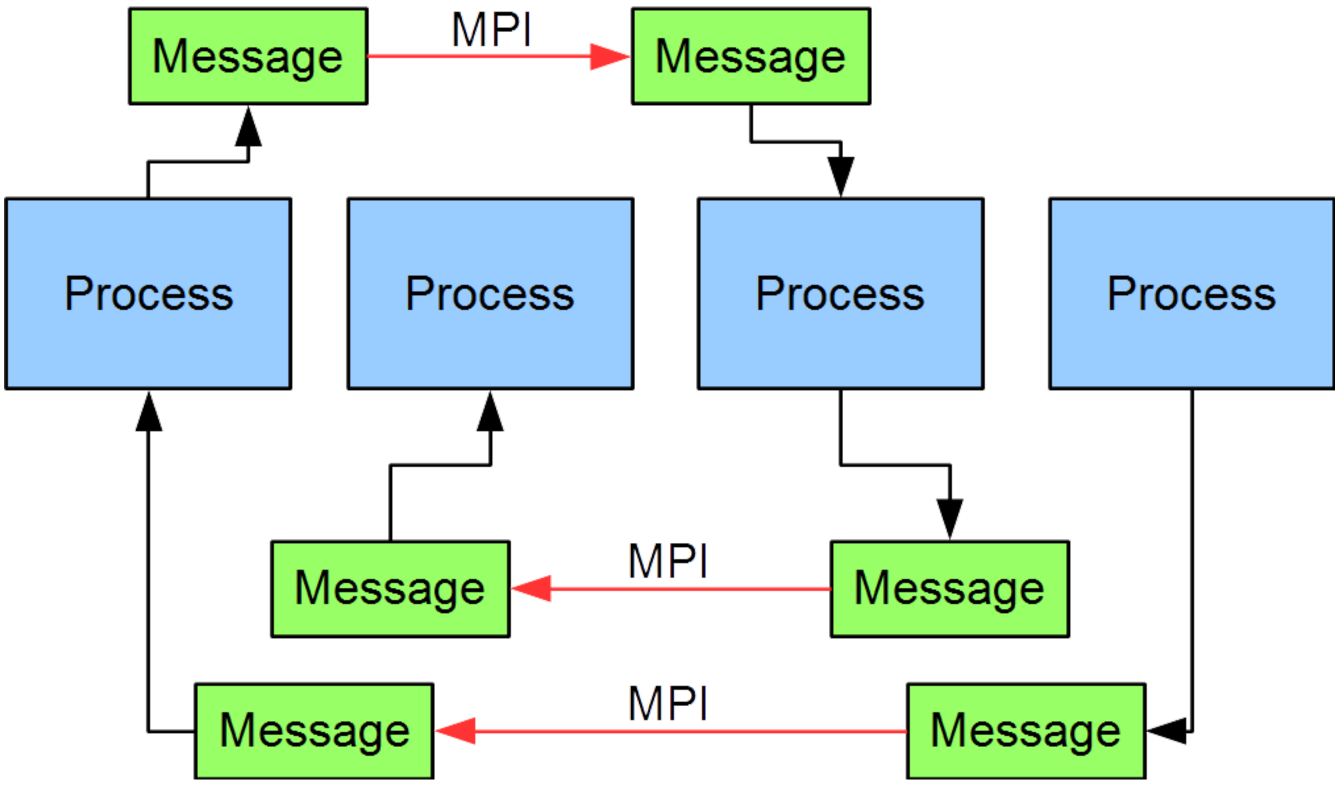
- Shared-memory system (single node)
    - Use thread-programming mode (all threads access the same memory address table and exchange data)

    - Use process-programming mode (MPI, Harp, Spark) to explicitly pass message data.
- Distributed system (HPC cluster, DC cluster)
    - Use process-programming mode among distinct nodes
    - Use thread-programming mode within the same node.

SPMD and MPI

SPMD (single program, multiple data), is a programming model to split the tasks and run them simultaneously on different processors. Here, each task is handled by a process instead of a thread, and therefore it requires explicit message (data) exchange for parallel programs that have data synchronization points. MPI (Message Passing Interface) is the standard interface of communicating data across processes in the HPC community.

In MPI program, every process executes the same codes (single program), however, you could use if else branch on the rank id to assign different tasks to them. `rank 0` is considered as the `root` process.



Message passing among different MPI processes

Running a Simple MPI program on Multiple Nodes

We first demonstrate the idea of running a program on distributed cluster system. We will run a program to calculate the mean value of a large number of integers across multiple nodes. Here, we will use the OpenMPI library.

,copy the lab files to your home directory

```
cp -r /share/jproject/lc37/distributed-tutorial ~/
cd ~/distributed-tutorial/MPI_sample
```

Initialize MPI

The main.cpp file calculate the mean value of an integer array. To use MPI, you shall include the mpi.h header and initialize the MPI

```
MPI_Init(&argc, &argv);

MPI_Comm_size(MPI_COMM_WORLD, &nprocs);
MPI_Comm_rank(MPI_COMM_WORLD, &rank_id);
```

Secondly, fetch the total MPI process number and the rank id for the current process. Thirdly, create the local array on each MPI rank, and start the iterations to compute the mean value

```
int array_len_local = array_len/nprocs;
int* obj_array = (int*)malloc(array_len_local*sizeof(int));
create_array(array_len_local, obj_array);
```

Given a fixed total array size, using more MPI processes will lead to less local computation workload on each process.

Local Computation

The local computation returns the sum value of the sub-arrays.

```
float compute_sum(int len, int* array_input)
{
    float sum = 0;
    for(int i=0;i<len;i++)
        sum += array_input[i];

    return sum;
}
```

MPI Allreduce

Invoke a MPI allreduce function to synchronize the local sum value

```
// start MPI_allreduce
MPI_Allreduce(&sum_local, &sum_global, 1, MPI_FLOAT, MPI_SUM, MPI_COMM_WORLD);
```

Each process has got the same sum_global as the total sum of the array

Finally, compute the mean value on the root rank process

```
if (rank_id == 0)
    mean += (sum_global/array_len);
```

Compile and Run the MPI program

To compile

```
make
```

To run the program, first create a hosts file to specify your nodes name. Open the hosts file

```
vim ./hosts
```

add the name of nodes assigned to you

```
j-xx1
j-xx2
```

We then use the `run.sh` script to launch the MPI program. It contains commands to configure and launch MPI jobs

```
#!/bin/bash

MPI_INSTALL=/share/jproject/lc37/openmpi_build/bin

${MPI_INSTALL}/mpirun -machinefile ./hosts -n $1 --mca btl openib,self --map-by node --bind-to none ./mpi_mean
```

Parameters are:

- -machinefile: specifies the nodes to use (in hosts file)
- -n : the number of MPI process to launch
- -mca btl openib,self: use InfiniBand if possible
- --map-by node: mapping the MPI process to nodes
- --bind-to none: not to bind the process

Execute the script and set up the first argument as the number of MPI processes

```
./run.sh 2
```