Introduction to MPI Programming

Schrodinger ZHU Yifan
<i@zhuyi.fan>

September 29, 2021

Topology

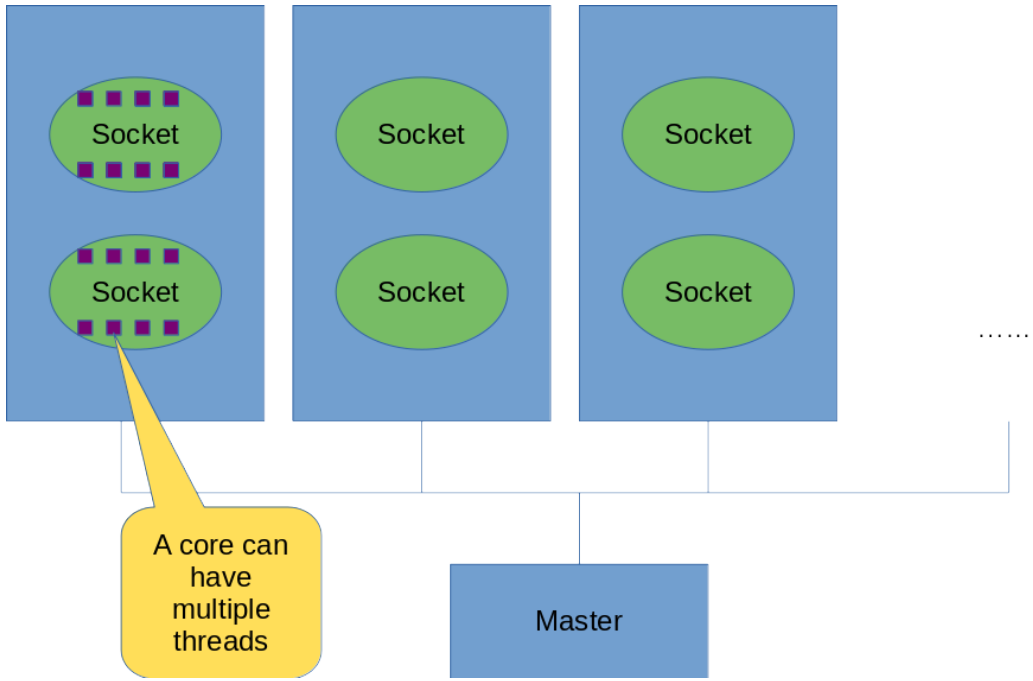

Figure: Cluster Topology

Blocking Point-wise Communication

```
MPI_Send(
    void* data,
    int count,
    MPI_Datatype datatype,
    int destination,
    int tag,
    MPI_Comm communicator);
```

Blocking Point-wise Communication

```
MPI_Recv(
    void* data,
    int count,
    MPI_Datatype datatype,
    int source,
    int tag,
    MPI_Comm communicator,
    MPI_Status* status)
```

Blocking Point-wise Communication

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

Figure: Tags for MPI Supported Types

Blocking Point-wise Communication

```
if (0 == rank) {
    data = 1999'12'08;
    for (int i = 1; i < size; ++i) {
        data = data + i;
        std::cout << "sending " << data << " to " << i <<
↪   std::endl;
        MPI_Send(&data, 1, MPI_INT, i, 0, MPI_COMM_WORLD);
    }
} else {
    MPI_Recv(&data, 1, MPI_INT, 0, 0, MPI_COMM_WORLD,
↪   MPI_STATUS_IGNORE);
    std::cout << "received " << data << " at " << rank <<
↪   std::endl;
}
```

Blocking Point-wise Communication

```
❯ mpirun -np 6 main
sending 19991209 to 1
sending 19991211 to 2
sending 19991214 to 3
sending 19991218 to 4
received 19991209 at 1
received 19991211 at 2
received 19991214 at 3
received 19991218 at 4
received 19991223 at 5
sending 19991223 to 5
```

Figure: Output

Blocking Point-wise Communication

```
int MPI_Sendrecv(
    const void *sendbuf,
    int sendcount,
    MPI_Datatype sendtype,
    int dest,
    int sendtag,
    void *recvbuf,
    int recvcount,
    MPI_Datatype recvtype,
    int source,
    int recvtag,
    MPI_Comm comm,
    MPI_Status *status);
```

Blocking Point-wise Communication

```cpp
#include <mpi.h>
#include <vector>

int main(int argc, char **argv) {
    int rank;
    int size;
    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    std::vector<int> send(size), recv(size);
    for (auto i = 0; i < size; ++i) {
        send[i] = rank + i;
    }
    auto target = (rank + 1) % size;
    auto source = ((rank - 1) % size + size) % size;
    MPI_Sendrecv(send.data(), size, MPI_INT, target, 0, recv.data(), size, MPI_INT, source, 0, MPI_COMM_WORLD,
                 MPI_STATUS_IGNORE);
    std::cout << "received ";
    for (auto i = 0; i < size; ++i) {
        std::cout << recv[i] << " ";
    }
    std::cout << "from " << source << " at " << rank << std::endl;
    MPI_Finalize();
}
```

Figure: Code

Blocking Point-wise Communication

```
csc4005-assignment-1/cmake-build-debug on  master [!] via △ v3.21.3
❯ mpirun -np 6 main
received 5 6 7 8 9 10 from 5 at 0
received 0 1 2 3 4 5 from 0 at 1
received 1 2 3 4 5 6 from 1 at 2
received 2 3 4 5 6 7 from 2 at 3
received 3 4 5 6 7 8 from 3 at 4
received 4 5 6 7 8 9 from 4 at 5
```

Figure: Output

Probing

```
MPI_Get_count(
    MPI_Status* status,
    MPI_Datatype datatype,
    int* count);
MPI_Probe(
    int source,
    int tag,
    MPI_Comm comm,
    MPI_Status* status);
```

Probing

```
const int MAX_NUMBERS = 100;
int numbers[MAX_NUMBERS];
int number_amount;
if (world_rank == 0) {
    // Pick a random amount of integers to send to process one
    srand(time(NULL));
    number_amount = (rand() / (float)RAND_MAX) * MAX_NUMBERS;

    // Send the amount of integers to process one
    MPI_Send(numbers, number_amount, MPI_INT, 1, 0, MPI_COMM_WORLD);
    printf("0 sent %d numbers to 1\n", number_amount);
} else if (world_rank == 1) {
    MPI_Status status;
    // Receive at most MAX_NUMBERS from process zero
    MPI_Recv(numbers, MAX_NUMBERS, MPI_INT, 0, 0, MPI_COMM_WORLD,
             &status);

    // After receiving the message, check the status to determine
    // how many numbers were actually received
    MPI_Get_count(&status, MPI_INT, &number_amount);

    // Print off the amount of numbers, and also print additional
    // information in the status object
    printf("1 received %d numbers from 0. Message source = %d, "
           "tag = %d\n",
           number_amount, status.MPI_SOURCE, status.MPI_TAG);
}
```

Probing

```c
int number_amount;
if (world_rank == 0) {
    const int MAX_NUMBERS = 100;
    int numbers[MAX_NUMBERS];
    // Pick a random amount of integers to send to process one
    srand(time(NULL));
    number_amount = (rand() / (float)RAND_MAX) * MAX_NUMBERS;

    // Send the random amount of integers to process one
    MPI_Send(numbers, number_amount, MPI_INT, 1, 0, MPI_COMM_WORLD);
    printf("0 sent %d numbers to 1\n", number_amount);
} else if (world_rank == 1) {
    MPI_Status status;
    // Probe for an incoming message from process zero
    MPI_Probe(0, 0, MPI_COMM_WORLD, &status);

    // When probe returns, the status object has the size and other
    // attributes of the incoming message. Get the message size
    MPI_Get_count(&status, MPI_INT, &number_amount);

    // Allocate a buffer to hold the incoming numbers
    int* number_buf = (int*)malloc(sizeof(int) * number_amount);

    // Now receive the message with the allocated buffer
    MPI_Recv(number_buf, number_amount, MPI_INT, 0, 0,
             MPI_COMM_WORLD, MPI_STATUS_IGNORE);
    printf("1 dynamically received %d numbers from 0.\n",
```

Immediate Point-wise Communication

```
int MPI_Isend(
    const void *buf,
    int count,
    MPI_Datatype datatype,
    int dest,
    int tag,
    MPI_Comm comm,
    MPI_Request *request);
```

Immediate Point-wise Communication

```
int MPI_Irecv(
    void *buf,
    int count,
    MPI_Datatype datatype,
    int source,
    int tag,
    MPI_Comm comm,
    MPI_Request *request);
```

Immediate Point-wise Communication

```
int MPI_Wait(
    MPI_Request *request,
    MPI_Status *status);
int MPI_Test(
    MPI_Request *request,
    int *flag,
    MPI_Status *status);
```

Immediate Point-wise Communication

```
MPI_Request send_req, recv_req;
MPI_Isend(send.data(), size, MPI_INT, target, 0,
↪  MPI_COMM_WORLD, &send_req);
MPI_Irecv(recv.data(), size, MPI_INT, source, 0,
↪  MPI_COMM_WORLD, &recv_req);
int sflag = 0, rflag = 0;
do {
    MPI_Test(&send_req, &sflag, MPI_STATUS_IGNORE);
    MPI_Test(&recv_req, &rflag, MPI_STATUS_IGNORE);
} while (!sflag || !rflag);
```

Broadcast

```
MPI_Bcast(
    void* data,
    int count,
    MPI_Datatype datatype,
    int root,
    MPI_Comm communicator);
```

Broadcast

```
int main(int argc, char **argv) {
    int data;
    int rank;
    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    if (0 == rank) {
        std::cin >> data;
    }
    MPI_Bcast(&data, 1, MPI_INT, 0, MPI_COMM_WORLD);
    std::cout << data << std::endl;
    MPI_Finalize();
}
```

Barrier

```
MPI_Barrier(MPI_COMM_WORLD);
```

Barrier

```
{

↪   std::this_thread::sleep_for(std::chrono::milliseconds{100}
↪   * rank);
    std::cout << "hi (no bar)" << std::endl;
}
MPI_Barrier(MPI_COMM_WORLD);
{

↪   std::this_thread::sleep_for(std::chrono::milliseconds{100}
↪   * rank);
    MPI_Barrier(MPI_COMM_WORLD);
    std::cout << "hi (bar)" << std::endl;
}
```

More

- Scatter and Gather: https://mpitutorial.com/tutorials/mpi-scatter-gather-and-allgather/
- Reduce: https://mpitutorial.com/tutorials/mpi-reduce-and-allreduce/
- Group Division: https://mpitutorial.com/tutorials/introduction-to-groups-and-communicators/

Stacktrace

```c
#include <mpi.h>
void wrong() {
    int data = 1;
    int rank;
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    if (0 == rank) {
        MPI_Recv(&data, 1, MPI_INT, 3, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
    }
}
int main(int argc, char **argv) {
    MPI_Init(&argc, &argv);
    wrong();
    MPI_Finalize();
}
```

Figure: Wrong Code

Stacktrace

```
mpirun --timeout 5 --get-stack-traces ./main
```

Figure: Stacktrace

Parallel Debug

See descriptions on BB.