Complete MPI Programs

MPI Basics

```
#include <mpi.h>
#include <stdio.h>

int main(int argc, char** argv) {
    MPI_Init(&argc, &argv);

    int world_rank;
    MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);

    int world_size;
    MPI_Comm_size(MPI_COMM_WORLD, &world_size);

    printf("Hello from process %d out of %d\n", world_rank, world_size);

    MPI_Finalize();
    return 0;
}
```

MPI Communication

```
#include <mpi.h>
#include <stdio.h>

int main(int argc, char** argv) {
    MPI_Init(&argc, &argv);

    int world_rank;
    MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);

    if (world_rank == 0) {
        int data = 100;
        MPI_Send(&data, 1, MPI_INT, 1, 0, MPI_COMM_WORLD);
        printf("Process 0 sent data %d to Process 1\n", data);
    } else if (world_rank == 1) {
        int received_data;
        MPI_Recv(&received_data, 1, MPI_INT, 0, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
        printf("Process 1 received data %d from Process 0\n", received_data);
    }

    MPI_Finalize();
    return 0;
}
```

MPI Synchronization

```
#include <mpi.h>
#include <stdio.h>

int main(int argc, char** argv) {
    MPI_Init(&argc, &argv);

    int world_rank;
    MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);

    printf("Process %d before synchronization\n", world_rank);
    MPI_Barrier(MPI_COMM_WORLD);
    printf("Process %d after synchronization\n", world_rank);

    MPI_Finalize();
    return 0;
}
```

MPI Data Movement

```
#include <mpi.h>
#include <stdio.h>

int main(int argc, char** argv) {
    MPI_Init(&argc, &argv);

    int world_rank;
    MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);

    int data;
    if (world_rank == 0) {
        data = 42;
        printf("Process 0 broadcasting data: %d\n", data);
    }

    MPI_Bcast(&data, 1, MPI_INT, 0, MPI_COMM_WORLD);
    printf("Process %d received data: %d\n", world_rank, data);

    MPI_Finalize();
    return 0;
}
```

MPI Collective Computation

```
#include <mpi.h>
#include <stdio.h>

int main(int argc, char** argv) {
    MPI_Init(&argc, &argv);

    int world_rank;
    MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);
    int world_size;
    MPI_Comm_size(MPI_COMM_WORLD, &world_size);

    int local_value = world_rank + 1;
    int sum;

    MPI_Reduce(&local_value, &sum, 1, MPI_INT, MPI_SUM, 0, MPI_COMM_WORLD);

    if (world_rank == 0) {
        printf("Sum of all ranks: %d\n", sum);
    }

    MPI_Finalize();
    return 0;
}
```