**Exercise 1 (Start and stop MPI)**

1)

./hello

**Hello World!**

mpirun -np 1 hello

**Hello World!**

mpirun -np 4 hello

**Hello World!**

**Hello World!**

**Hello World!**

**Hello World!**

**2)**

**Total processes: 4**

**Hello from process: 0**

**Hello from process: 3**

**Hello from process: 2**

**Hello from process: 1**

    MPI\_Init(&argc, &argv); // Initialize MPI environment

    MPI\_Comm\_rank(MPI\_COMM\_WORLD, &rank); // Get the rank of this process

   if (rank == 0)

   {

      MPI\_Comm\_size(MPI\_COMM\_WORLD, &size); // Get the total number of processes

      printf("Total processes: %d\n", size);

   }

    printf("Hello from process: %d \n", rank);

**Exercise 2 (Deadlock test)**

**Processor 0 got 1.000000 from processor 1**

**Processor 1 got 0.000000 from processor 0**

  if (rank == 0) {

      MPI\_Isend(&a, 1, MPI\_DOUBLE, 1, 111, MPI\_COMM\_WORLD, &request);

      MPI\_Recv(&b, 1, MPI\_DOUBLE, 1, 222, MPI\_COMM\_WORLD, &status);

      MPI\_Wait(&request, &status); // Wait for the send operation to complete

      printf("Processor 0 got %f from processor 1\n", b);

  } else {

      MPI\_Isend(&a, 1, MPI\_DOUBLE, 0, 222, MPI\_COMM\_WORLD, &request);

      MPI\_Recv(&b, 1, MPI\_DOUBLE, 0, 111, MPI\_COMM\_WORLD, &status);

      MPI\_Wait(&request, &status); // Wait for the send operation to complete

      printf("Processor 1 got %f from processor 0\n", b);

  }

**Exercise 3 (Point-to-Point communication)**

  /\* Exchange variable a in a circular fashion using non-blocking communication\*/

  int tag = 111; // Use a single tag for both send and receive operations

  MPI\_Isend(&a, 1, MPI\_DOUBLE, (rank+1)%size, tag, MPI\_COMM\_WORLD,&request\_send);

  MPI\_Irecv(&b,1,MPI\_DOUBLE,(rank+size-1)%size,tag,MPI\_COMM\_WORLD,&request\_recv);

  MPI\_Wait(&request\_send, &status); // Wait for the send operation to complete

  MPI\_Wait(&request\_recv, &status); // Wait for the receive operation to complete

  printf("Processor %d got %f from processor %d\n", rank, b, (rank+size-1)%size);

Processor 5 got 104.000000 from processor 4

Processor 6 got 105.000000 from processor 5

Processor 4 got 103.000000 from processor 3

Processor 7 got 106.000000 from processor 6

Processor 0 got 107.000000 from processor 7

Processor 1 got 100.000000 from processor 0

Processor 2 got 101.000000 from processor 1

Processor 3 got 102.000000 from processor 2

**Exercise 4 (Ping-pong)**

**Exercise 5 (Communication ’one-to-all’)**

**En bild som visar text

Automatiskt genererad beskrivning**

Processor 1 got 999.999000

Processor 2 got 999.999000

Processor 3 got 999.999000

Processor 4 got 999.999000

Processor 5 got 999.999000

Processor 6 got 999.999000

Processor 7 got 999.999000

Processor 8 got 999.999000

Processor 9 got 999.999000

Processor 10 got 999.999000

Processor 11 got 999.999000

Processor 12 got 999.999000

Processor 13 got 999.999000

Processor 14 got 999.999000

Processor 15 got 999.999000

**Exercise 6 (Computing the value of π in parallel)**

  t\_begin = MPI\_Wtime(); // Start measuring time

  for (i = istart; i <= istop; i++) { /\* The local loop \*/

    double tmp = dx\*(0.5 + i);

    sum += dx\*4.0/(1.0 + tmp\*tmp);

  }

  t\_end = MPI\_Wtime(); // Stop measuring time

  if (rank == 0) {

    double globsum = sum;

    for (i = 1; i < size; i++) { /\* Collect the partial sums \*/

      MPI\_Status status;

      MPI\_Recv(&sum, 1, MPI\_DOUBLE, i, i, MPI\_COMM\_WORLD, &status);

      globsum += sum;

    }

    printf("PI is approx. %.16f\n",  globsum);

    printf("Elapsed time : %1.2f\n", t\_end-t\_begin);

  }

  else

  { /\* Send my partial sum to the processor with its

        rank equal to zero \*/

    MPI\_Send(&sum, 1, MPI\_DOUBLE, 0, rank, MPI\_COMM\_WORLD);

  }

PI is approx. 3.1415926535893619

Elapsed time : 0.40

**Exercise 7 (Computing the value of π using a Monte Carlo method)**