hello.cpp

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

int main( int argc, char ** argv ) {

    MPI_Init( NULL, NULL );
    int rank;
    MPI_Comm_rank( MPI_COMM_WORLD, &rank );

    while ( 1 )
        std::cout << "Hello, world! says rank " << rank << std::endl;
    MPI_Finalize();
    return 0;
}
```

sum.cpp

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

int main() {
    MPI_Init( NULL, NULL );
    int rank, nprocs;
    MPI_Comm_rank( MPI_COMM_WORLD, &rank );
    MPI_Comm_size( MPI_COMM_WORLD, &nprocs );

    int start = rank * 1024 / nprocs;
    int end = ( rank + 1 ) * 1024 / nprocs;

    printf( "I am rank %d out of %d processes; I will be summing from %d to %d\n",
            rank, nprocs, start, end
          );

    MPI_Barrier( MPI_COMM_WORLD );
    int sum = 0;
    for ( int i = start; i < end; i++ ) sum += i;
    printf( "rank %d partial sum = %d\n", rank, sum );

    int total_sum = 2;
    MPI_Reduce( &sum, &total_sum, 1, MPI_INT, MPI_SUM, 0, MPI_COMM_WORLD );

    MPI_Barrier( MPI_COMM_WORLD );

    printf( "Total sum is %d on rank %d\n", total_sum, rank );

    MPI_Finalize();
    return 0;
}
```

pi.cpp

```
#include<mpi.h>
#include<cstdio>
#include<cmath>
#include<cstdlib>
#include<ctime>

int main() {
    MPI_Init( NULL, NULL );

    int N = 1024 * 1024 * 128;
    int rank, nprocs;
    MPI_Comm_rank( MPI_COMM_WORLD, &rank );
    MPI_Comm_size( MPI_COMM_WORLD, &nprocs );
    srand( time( 0 ) + rank );

    int in = 0;
    double t0 = MPI_Wtime();
    for ( int i = 0; i < N; i++ ) {
        double x = rand() / double( RAND_MAX );
        double y = rand() / double( RAND_MAX );
        if ( x * x + y * y < 1.0 ) ++in;
    }
    MPI_Barrier( MPI_COMM_WORLD );
    double t1 = MPI_Wtime();

    int total_in;
    MPI_Reduce( &in, &total_in, 1, MPI_INT, MPI_SUM, 0, MPI_COMM_WORLD );

    if ( rank == 0 ) {
        double pi = total_in / double( N ) / double( nprocs ) * 4.0;
        printf( "pi = %lf with %d samples, error = %.5le, used %.3lf seconds\n",
                pi, N, std::fabs( pi - M_PI ), t1 - t0 );
    }

    MPI_Finalize();
    return 0;
}
```

heat.cpp

```
#include<cstdio>
#include<mpi.h>
#include<ctime>
#include<cmath>
#include<vector>
#include<fstream>
#include<cstring>
#include<cstdlib>

int main() {
    MPI_Init( NULL, NULL );

    int rank, nprocs;
    MPI_Comm_rank( MPI_COMM_WORLD, &rank );
```

```
    MPI_Comm_size( MPI_COMM_WORLD, &nprocs );

    int NT = 1024 * 1024 * 2, N = NT / nprocs, M = 400;
    double h = 1, k = 0.01;
    std::vector<double> u( N ), v( N );

    // IC
    int start = rank * NT / nprocs;
    double t0 = MPI_Wtime();
    for ( int i = 0; i < N; i++ ) {
        u[i] = ( i + start )  >= ( NT / 2 ) ? 1 : 0;
    }

    // solve
    for ( int m = 0; m < M; m++ ) {
        //printf( "step %d\n", m );
        // internal points
        for ( int j = 1; j < N - 1; j++ ) {
            v[j] = u[j] + k / ( h * h ) * ( u[j - 1] + u[j + 1] - 2 * u[j] );
        }
        // PBC
        double left, right;
        MPI_Status status;
        MPI_Request req1, req2;
        MPI_Isend( &u[0],   1, MPI_DOUBLE, ( rank - 1 + nprocs ) % nprocs, 0,
MPI_COMM_WORLD, &req1 );
        MPI_Isend( &u[N - 1], 1, MPI_DOUBLE, ( rank + 1 + nprocs ) % nprocs, 0,
MPI_COMM_WORLD, &req2 );
        MPI_Recv( &left,   1, MPI_DOUBLE, ( rank - 1 + nprocs ) % nprocs, 0,
MPI_COMM_WORLD, &status );
        MPI_Recv( &right,  1, MPI_DOUBLE, ( rank + 1 + nprocs ) % nprocs, 0,
MPI_COMM_WORLD, &status );

        v[0] = u[0] + k / ( h * h ) * ( left + u[1] - 2 * u[0] );
        v[N - 1] = u[N - 1] + k / ( h * h ) * ( u[N - 2] + right - 2 * u[N - 1] );

        MPI_Wait( &req1, &status );
        MPI_Wait( &req2, &status );

        for ( int j = 0; j < N; j++ ) u[j] = v[j];
        // dump
        if ( m % 100 == 0 ) {
            MPI_Barrier( MPI_COMM_WORLD );
            if ( rank == 0 ) {
                printf( "Done step %d\n", m );
                char fn[256];
                sprintf( fn, "step-%04d.txt", m );
                std::ofstream fout( fn );
                std::vector<double> others_result( N );
                printf( "dumping my own result\n" );
                for ( int j = 0; j < N; j++ ) fout << u[j] << std::endl;
                for ( int r = 1; r < nprocs; r++ ) {
                    printf( "receiving result from rank %d\n", r );
                    MPI_Status status;
                    MPI_Recv( others_result.data(), N, MPI_DOUBLE, r, 0,
MPI_COMM_WORLD, &status );
                    printf( "result from rank %d received, dumping\n", r );
```

```
                    for ( int j = 0; j < N; j++ ) fout << others_result[j] <<
std::endl;
                }
            } else {
                printf( "rank %d sending own result to rank 0\n", rank );
                MPI_Send( u.data(), N, MPI_DOUBLE, 0, 0, MPI_COMM_WORLD );
            }
        }
    }
    double t1 = MPI_Wtime();

    if ( rank == 0 ) printf( "Wall time: %lf\n", t1 - t0 );

    return 0;
}
```