Работа 3. Синхронизация часов

В заданном примере ‘mpiclocksync.c’ реализован метод одновременного запуска во всех процессах заданной функции. В начале своей работы процессы вычисляют смещение показаний своих часов относительно часов процесса 0 и устанавливают глобальное время (sync_clock_linear).

Задание:

1. Запустить и измерить время выполнения функции MPI_Allreduce (MPI_SUM, MPI_DOUBLE)
2. Реализовать модификацию метода для измерения времени MPI_Allreduce по результатам 10 запусков (каждый запуск по запланированному глобальному времени)

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include <mpi.h>

double measure_clock_offset_adaptive(MPI_Comm comm, int root, int peer)
{
    int rank, commsize, rttmin_notchanged = 0;
    double starttime, stoptime, peertime, rtt, rttmin = 1E12,
           invalidtime = -1.0, offset;

    MPI_Comm_rank(comm, &rank);
    MPI_Comm_size(comm, &commsize);

    offset = 0.0;
    for (;;) {
        if (rank != root) {
            /* Peer process */
            starttime = MPI_Wtime();
            MPI_Send(&starttime, 1, MPI_DOUBLE, root, 0, comm);
            MPI_Recv(&peertime, 1, MPI_DOUBLE, root, 0, comm, MPI_STATUS_IGNORE);
            stoptime = MPI_Wtime();
            rtt = stoptime - starttime;

            if (rtt < rttmin) {
                rttmin = rtt;
                rttmin_notchanged = 0;
                offset = peertime - rtt / 2.0 - starttime;
            } else {
                if (++rttmin_notchanged == 100) {
                    MPI_Send(&invalidtime, 1, MPI_DOUBLE, root, 0, comm);
                    break;
                }
            }
        } else {
            /* Root process */
            MPI_Recv(&starttime, 1, MPI_DOUBLE, peer, 0, comm, MPI_STATUS_IGNORE);
            peertime = MPI_Wtime();
            if (starttime < 0.0)
                break;
            MPI_Send(&peertime, 1, MPI_DOUBLE, peer, 0, comm);
        }
    } /* for */
    return offset;
}

double sync_clock_linear(MPI_Comm comm, int root)
{
    int rank, commsize;
    MPI_Comm_rank(comm, &rank);
    MPI_Comm_size(comm, &commsize);

    if (commsize < 2)
        return 0.0;

    double local_offset = 0.0;
    for (int i = 1; i < commsize; i++) {
        MPI_Barrier(comm);
        if (rank == root || rank == i) {
            local_offset = measure_clock_offset_adaptive(comm, root, i);
        }
    }
    return local_offset;
}

double measure_bcast_double(MPI_Comm comm)
{
    double buf, totaltime = 0.0, optime, maxtime = 0.0;
    int i, nreps = 3;

    /* Warmup call */
    MPI_Bcast(&buf, 1, MPI_DOUBLE, 0, comm);
    /* Measures (upper bound) */
    for (i = 0; i < nreps; i++) {
        MPI_Barrier(comm);
        optime = MPI_Wtime();
        MPI_Bcast(&buf, 1, MPI_DOUBLE, 0, comm);
        optime = MPI_Wtime() - optime;
        MPI_Reduce(&optime, &maxtime, 1, MPI_DOUBLE, MPI_MAX, 0, comm);
```

```
        totaltime = totaltime > maxtime ? totaltime : maxtime;
    }
    return totaltime;
}

int main(int argc, char **argv)
{
    int rank, commsize;
    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &commsize);

    // Synchronize clocks
    double clock_offset = sync_clock_linear(MPI_COMM_WORLD, 0);

    // Measure bcast time
    double bcast_time = measure_bcast_double(MPI_COMM_WORLD);
    MPI_Barrier(MPI_COMM_WORLD);

    int is_late = 1;
    double barrier_time, tlocal = 0.0;

    // Broadcast barrier time
    double delta = 10.0;
    if (rank == 0)
        barrier_time = MPI_Wtime() + bcast_time * delta;
    MPI_Bcast(&barrier_time, 1, MPI_DOUBLE, 0, MPI_COMM_WORLD);

    // Translate global time to local
    int is_local_late = 0;
    barrier_time -= clock_offset;

    // Wait for barrier time
    volatile double t = MPI_Wtime();
    if (t > barrier_time) {
        is_local_late = 1;
        fprintf(stderr, "Error: rank %d was late\n", rank);
    } else {
        while ((t = MPI_Wtime()) < barrier_time) {
                /* Wait */;
        }

        // START: Allreduce
        //double local_time = MPI_Wtime();
        //double global_time = local_time + clock_offset;
        //printf("process %3d: local time %.6f, global time %.6f (offset %.6f)\n",
        //       rank, local_time, global_time, clock_offset);

    }

    MPI_Finalize();
    return 0;
}
```