```

#include "main.h"
#include <sstream>

#define MPI_TASK_ID_TAG 1
#define MPI_RESULT_TAG 2
#define RESULT_VECTOR_SIZE 8

using namespace std;
using namespace std::chrono;

int segment_count;
int NOHTING_TO_DO = -666;
int WORK_DONE = 666;


int main(int argc, char **argv) {
#ifdef TIME_MEASURING
  cout << "Time measuring ON" << endl;
  high_resolution_clock::time_point full_start, full_end, alg_start, alg_end;
  full_start = high_resolution_clock::now();
#endif
  int mpi_node_id, mpi_process_count;
  vector<SegmentData> segments, computed_segments;

  std::atexit(exit_handler);

    CmdDriver parser(argc, argv);
  SqliteDriver sql(parser.getParam("sql"));
  FileDriver file(parser.getParam("bounds"));

  try {
    segments = sql.load(segments);
    file.load();
  }
  catch (string& msg){
    cerr << "Load file problem!\n" << msg << endl;
    return(EXIT_FAILURE);
  }
  segment_count = segments.size();

  MPI_Init(&argc, &argv);
  MPI_Comm_rank(MPI_COMM_WORLD, &mpi_node_id);
  MPI_Comm_size(MPI_COMM_WORLD, &mpi_process_count);


  if (mpi_node_id == MASTER_ID) {
    EquationBounds::printEquationBounds();
    cout << "Master start (" << mpi_node_id << ")" << endl;
    master(mpi_process_count, segments);
  }
  else {
    cout << "Slave start (" << mpi_node_id << ")" << endl;
#ifdef TIME_MEASURING
  alg_start = high_resolution_clock::now();
#endif
    slave(mpi_node_id, segments);
#ifdef TIME_MEASURING
  alg_end = high_resolution_clock::now();
#endif
    cout << endl << "================== RESULT from " << mpi_node_id << " ==================" << endl;
    int size = segments.size();
    for(int i = 0; i < size; i++){
```

```cpp
      if(segments[i].getResult().getMetric() != DBL_MAX){
        computed_segments.push_back(segments[i]);
        // cout << mpi_node_id << " Segment " << segments[i].getId() << ": " << segments[i].getResult() << endl;
      }
    }
    size = computed_segments.size();
    for(int i = 0; i < size; i++){
      cout << mpi_node_id << " Segment " << computed_segments[i].getId() << ": " << computed_segments[i].getResult() << endl;
    }
    cout << endl << "====================================" << endl << endl;
    try {
      sql.save(computed_segments);
    }
    catch (string& msg){
      cerr << mpi_node_id << ": Save SQLite problem!\n" << msg << endl;
    }
  }

  MPI_Finalize();

#ifdef TIME_MEASURING
  full_end = high_resolution_clock::now();
  auto full_duration = duration_cast<microseconds>(full_end - full_start).count();
  auto alg_duration = duration_cast<microseconds>(alg_end - alg_start).count();

  cout << "#" << mpi_node_id << "Time execute for Nelder-Mead algorithm: " << alg_duration << "us => " << alg_duration / 1000.0 << "ms => " << alg_duration / 1000000.0 << "s" <<  endl;
  cout << "#" << mpi_node_id << "Time execute for the entire program: " << full_duration << "us => " << alg_duration / 1000.0 << "ms => " << alg_duration / 1000000.0 << "s" << endl;
#endif

  return(EXIT_SUCCESS);
}

void master(int mpi_process_count, vector<SegmentData> &segments){
  string msg_head = "#0: ";
  cout << msg_head << "I'm MPI master!" << endl;
  MPI_Status status;
  int remaining_segments_count = segment_count;
  int received_responses = 0;

  int current_segment_index = 0;
  //kazdemu procesu poslu index segmentu, ktery ma zpracovat
  for (int i = 1; i < mpi_process_count; i++, current_segment_index++, remaining_segments_count--) {
    MPI_Send(&current_segment_index, 1, MPI_INT, i, MPI_TASK_ID_TAG, MPI_COMM_WORLD);
    // cout << msg_head << "Send work on INDEX " << current_segment_index << " with ID " << segments[current_segment_index].getId() << " to " << i << endl;
  }

  int slave_id = 0;
  int result = 0;
  while (remaining_segments_count > 0) {
    MPI_Recv(&result, 1, MPI_INT, MPI_ANY_SOURCE, MPI_RESULT_TAG, MPI_COMM_WORLD, &status);
    slave_id = status.MPI_SOURCE;
    received_responses++;

    MPI_Send(&current_segment_index, 1, MPI_INT, slave_id, MPI_TASK_ID_TAG, MPI_COMM_WORLD);

    current_segment_index++;
    remaining_segments_count--;
  }

  while((segment_count - received_responses) > 0) {
    MPI_Recv(&result, 1, MPI_INT, MPI_ANY_SOURCE, MPI_RESULT_TAG, MPI_COMM_WORLD, &status);
    received_responses++;
    slave_id = status.MPI_SOURCE;
    MPI_Send(&NOHTING_TO_DO, 1, MPI_INT, slave_id, MPI_TASK_ID_TAG, MPI_COMM_WORLD);
  }
}

void slave(int my_id, vector<SegmentData> &segments){
  ostringstream oss_msg_head;
  oss_msg_head << "#" << my_id << ": ";
  string msg_head = oss_msg_head.str();

  cout << "I'm MPI slave #" << my_id << "..." << endl;
  MPI_Status status;
  int working_segment = -1;
  vector<double> result(RESULT_VECTOR_SIZE);

  do {
    // cout << msg_head << "Waiting for some job..." << endl;
    MPI_Recv(&working_segment, 1, MPI_INT, MASTER_ID, MPI_TASK_ID_TAG, MPI_COMM_WORLD, &status);

    if (working_segment != NOHTING_TO_DO) {
      // cout << msg_head << "I do work on segment " << segments[working_segment].getId() << endl;
      // cout << segments[working_segment] << endl;
      NealderMead alg(segments[working_segment]);
      // cout << msg_head << "My result is " << segments[working_segment].getResult() << endl;
      // cout << msg_head << "Best simplex params " << alg.getSimplex()[0] << endl;

      MPI_Send(&WORK_DONE, 1, MPI_INT, MASTER_ID, MPI_RESULT_TAG, MPI_COMM_WORLD);
      // cout << msg_head << "Send result vector" << endl;
    }
    // else {
    //  cout << msg_head << "Nothing to do..." << endl;
    // }
  } while (working_segment >= 0);
  // cout << msg_head << "I'm done with my job!" << endl;
}


void exit_handler(){
  if (DEBUG){
    std::cout << "Press Enter to exit..." << std::endl;
    std::cin.get();
  }
}
```