ping_pong_mpi.c

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

int main(int argc, char** argv) {
    int rank, size;
    double start_time, end_time;

    // Inicializa o ambiente MPI
    MPI_Init(&argc, &argv);

    // Obtém o rank (ID do processo) e o tamanho do comunicador
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);

    // O programa deve ser executado com exatamente 2 processos
    if (size != 2) {
        if (rank == 0) {
            fprintf(stderr, "Este programa requer exatamente 2 processos para ser executado.\n");
        }
        MPI_Finalize();
        return 1;
    }

    // Vetor de tamanhos de mensagem a serem testados, incluindo 4 MB
    int message_sizes[] = {8, 128, 512, 1024, 4096, 16384, 65536, 131072, 262144, 524288, 1048576, 4194304};
    int num_sizes = sizeof(message_sizes) / sizeof(int);

    // Número de trocas para cada tamanho de mensagem
    int num_iterations = 1000;

    for (int i = 0; i < num_sizes; i++) {
        int current_size = message_sizes[i];

        // Aloca o buffer para a mensagem
        char *send_buffer = (char*) malloc(current_size);
        char *recv_buffer = (char*) malloc(current_size);

        // O processo 0 inicia a comunicação
        if (rank == 0) {
            start_time = MPI_Wtime();

            for (int j = 0; j < num_iterations; j++) {
                // Envia a mensagem para o processo 1
                MPI_Send(send_buffer, current_size, MPI_CHAR, 1, 0, MPI_COMM_WORLD);
                // Recebe a mensagem de volta do processo 1
                MPI_Recv(recv_buffer, current_size, MPI_CHAR, 1, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
```

```
48              }
49
50              end_time = MPI_Wtime();
51              double total_time = end_time - start_time;
52              double average_time_ms = (total_time / (2 * num_iterations)) * 1000;
53
54              // Exibe os resultados
55              printf("Tamanho da Mensagem: %d bytes, Tempo Médio por troca: %f
    ms\n", current_size, average_time_ms);
56          }
57
58          // O processo 1 responde à comunicação
59          else if (rank == 1) {
60              for (int j = 0; j < num_iterations; j++) {
61                  // Recebe a mensagem do processo 0
62                  MPI_Recv(recv_buffer, current_size, MPI_CHAR, 0, 0,
    MPI_COMM_WORLD, MPI_STATUS_IGNORE);
63                  // Envia a mesma mensagem de volta
64                  MPI_Send(send_buffer, current_size, MPI_CHAR, 0, 0,
    MPI_COMM_WORLD);
65              }
66          }
67
68          free(send_buffer);
69          free(recv_buffer);
70
71      }
72
73      // Finaliza o ambiente MPI
74      MPI_Finalize();
75
76      return 0;
77  }
```