Белгородский государственный технологический университет им. В. Г. Шухова

Кафедра программного обеспечения вычислительной техники  
и автоматизированных систем

## Лабораторная работа №4 по теме: «Средства MPI для обмена сообщениями»

**Выполнил:**  
студент группы ПВ-31  
Адаменко И. И.

**Проверил:**к. т. н., доцент  
Михелёв В. М.

Белгород  
2015

**Цель работы:** ознакомиться со средствами технологии MPI для передачи сообщений между процессами и получить практический навык их использования.

# Теоретическая часть

MPI — программный интерфейс (API) для передачи информации, который позволяет обмениваться сообщениями между экземплярами программы, выполняющими одну задачу (которые могут быть запущенными на различных компьютерах).

MPI является наиболее распространённым стандартом интерфейса обмена данными в параллельном программировании, существуют его реализации для большого числа компьютерных платформ. В настоящее время существует большое количество бесплатных и коммерческих реализаций MPI.

Практически все реализации MPI представляют собой внешнюю подключаемую библиотеку. В связи с этим, при компилировании MPI программ, компилятору необходимо дополнительно указывать заголовочные и библиотечные файлы.

В поставку MPI, как правило, включаются две версии lib-файлов — отладочная и обыкновенная. В то время как обыкновенная служит для сборки финальных версий программ, оптимизированных на исполнение, отладочные версии позволяют собирать программы с дополнительной информацией, необходимой для отладки. В реализации MPI от Intel, к примеру, о том, что lib-файл является отладочным, говорит присутствие буквы d в имени файла (impi.lib — обыкновенный файл, impid.lib — отладочный вариант).

Базовым механизмом связи между MPI процессами является передача и приём сообщений. Сообщение несёт в себе передаваемые данные и информацию, позволяющую принимающей стороне осуществлять их выборочный приём:

* отправитель — ранг (номер в группе) отправителя сообщения;
* получатель — ранг получателя;
* признак — может использоваться для разделения различных видов сообщений;
* коммуникатор — код группы процессов.

Операции приёма и передачи могут быть блокирующимися и неблокирующимися. Для неблокирующихся операций определены функции проверки готовности и ожидания выполнения операции.

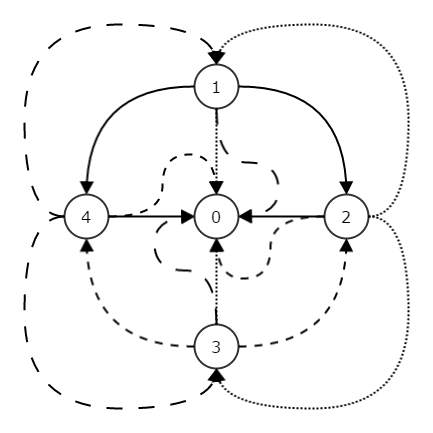
# Практическая часть

## Задание для варианта №1

Топология «звезда». Дочерние процессы пересылают пакеты данных центральному процессору следующим образом: пакет рассылается каждым дочерним процессом двум соседям, откуда они пересылаются центральному процессу.

Оценить время рассогласованности получения пары пакетов центральным процессором от дочерних в зависимости от количества дочерних процессов.

## Топология обмена сообщениями между процессами



На графе выше представлен пример передачи сообщений при наличии четырёх дочерних процессов.

То есть, каждый дочерний процесс вычисляет номера дочерних процессов «слева» и «справа» от него, передаёт сообщение им, а они, в свою очередь, передают сообщение центральному процессору.

## Исходный код

1. #include "mpi.h"
2. #include <stdio.h>
4. **int** main(**int** argc, **char** \*argv[])
5. {
6. **int** proc\_num,
7. proc\_rank,
8. recv\_rank;
10. **double** time, time\_min = 1.0/0.0, time\_max = 0, time\_avg = 0;
11. MPI\_Status status;
13. // инициализируем MPI
14. MPI\_Init(&argc, &argv);
15. // proc\_num -- общее число процессоров
16. MPI\_Comm\_size(MPI\_COMM\_WORLD, &proc\_num);
17. // proc\_rank -- ранг текущего процессора
18. MPI\_Comm\_rank(MPI\_COMM\_WORLD, &proc\_rank);
20. /\*  если процессов меньше 3,
21. то нельзя вычленить 2 дочерних из них \*/
22. **if** (proc\_num < 3) {
23. **if** (proc\_rank == 0) {
24. printf("Need more than 2 child processes!\n");
25. }
26. **return** 0;
27. }
29. // массив для хранения результатов
30. auto array = **new** **double**[proc\_num][2]();
32. // proc\_rank == 0 -- главный процесс
33. **if** (proc\_rank == 0) {
35. /\*  принимает информацию от дочерних процессов
36. и записывает время её приёма в массив \*/
37. **for** (**int** i = 1; i < proc\_num \* 2 - 1; i++)
38. {
39. // MPI\_Recv(buffer, count, type, from, tag, communicator, status)
40. MPI\_Recv(&recv\_rank, 1, MPI\_INT, MPI\_ANY\_SOURCE,
41. MPI\_ANY\_TAG, MPI\_COMM\_WORLD, &status);
42. **if** (array[recv\_rank][0] == 0.0) {
43. array[recv\_rank][0] = MPI\_Wtime();
44. } **else** {
45. array[recv\_rank][1] = MPI\_Wtime();
46. }
47. }
49. /\*  для всех записанных результатов
50. вычисляется время рассогласованности,
51. минимальное, максимальное и среднее время \*/
52. **for** (**int** i = 1; i < proc\_num; i++)
53. {
54. time = array[i][1] - array[i][0];
55. time\_avg += time;
57. **if** (time < time\_min) {
58. time\_min = time;
59. }
61. **if** (time > time\_max) {
62. time\_max = time;
63. }
65. printf("Time of process %d: %lf\n",
66. i, time);
67. }
69. time\_avg /= proc\_num - 1;
70. printf("Min: %lf; Max: %lf; Avg: %lf\n",
71. time\_min, time\_max, time\_avg);
73. } **else** {
74. // определение соседних дочерних процессов
75. **int** prev = (proc\_rank == 1) ? proc\_num - 1 : proc\_rank - 1,
76. next = (proc\_rank + 1 == proc\_num) ? 1 : proc\_rank + 1,
77. buf;
79. // отправка информации соседним дочерним процессам
80. // MPI\_Send(buffer, count, type, to, tag, communicator)
81. MPI\_Send(&proc\_rank, 1, MPI\_INT, prev, 0, MPI\_COMM\_WORLD);
82. MPI\_Send(&proc\_rank, 1, MPI\_INT, next, 0, MPI\_COMM\_WORLD);
84. // получаем информацию от соседа 'слева' и отсылаем главному процессу
85. MPI\_Recv(&buf, 1, MPI\_INT, prev, 0, MPI\_COMM\_WORLD, &status);
86. MPI\_Send(&buf, 1, MPI\_INT, 0, 0, MPI\_COMM\_WORLD);
88. // получаем информацию от соседа 'справа' и отсылаем главному процессу
89. MPI\_Recv(&buf, 1, MPI\_INT, next, 0, MPI\_COMM\_WORLD, &status);
90. MPI\_Send(&buf, 1, MPI\_INT, 0, 0, MPI\_COMM\_WORLD);
91. }
93. // завершаем работу MPI
94. MPI\_Finalize();
96. **return** 0;
97. }

## Тестирование программы

|  |  |  |  |
| --- | --- | --- | --- |
| **Nузлов** | **Tmin** | **Tmax** | **Tavg** |
| 4 |  |  |  |
| 8 |  |  |  |
| 16 |  |  |  |
| 32 |  |  |  |
| 40 |  |  |  |