Белгородский государственный технологический университет им. В. Г. Шухова

Кафедра программного обеспечения вычислительной техники  
и автоматизированных систем

## Лабораторная работа №3 по теме: «Знакомство с технологией MPI»

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

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

Белгород  
2015

**Цель работы:** ознакомиться с технологией MPI; научиться компилировать и запускать MPI программы; получить навык работы с простейшими средствами передачи сообщений между процессами.

# 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

1. Ознакомиться с технологией MPI.
2. Откомпилировать поставляемую с реализацией MPI тестовую программу и запустить её на различном числе процессоров. Написать к программе поясняющие комментарии.
3. Протестировать программу на 4, 8, 16, 32 и 40 узнал при различной величине входных данных. Составить таблицы коэффициента ускорения, построить графики, сделать выводы.

## Описание алгоритма

Рассматриваемая программа на языке С, вычисляет значение числа Пи с некоторой наперед заданной точностью (данная программа входит в реализацию MPICH в качестве примера). Для вычисления приближенного значения числа π в данном примере используется соотношение:

где n – некоторое положительно число (количество элементов ряда). Чем больше n, тем выше точность приближения.

Параллелизм программы основан на том, что i-й процесс из k запущенных вычисляет сумму каждого k-го слагаемого ряда, начиная с i-го (всего необходимо вычислить n слагаемых). После того как каждый процесс закончит вычисления, промежуточные результаты складываются.

## Исходный код

1. #include <stdio.h>
2. #include <math.h>
3. /\* Подключение библиотеки MPI \*/
4. #include "mpi.h"
6. **double** f(**double**);
8. /\* Функция для промежуточных вычислений \*/
9. **double** f(**double** a) {
10. **return** (4.0 / (1.0 + a\*a));
11. }
13. **int** main(**int** argc, **char** \*argv[]) {
14. /\* Объявление переменных \*/
15. **int** done = 0, n, myid, numprocs, i;
16. **double** PI25DT = 3.141592653589793238462643;
17. **double** mypi, pi, h, sum, x;
18. **double** startwtime = 0.0, endwtime;
19. **int**  namelen;
20. **char** processor\_name[MPI\_MAX\_PROCESSOR\_NAME];
22. /\* Инициализация и установка параметров MPI \*/
23. MPI\_Init(&argc, &argv);
24. MPI\_Comm\_size(MPI\_COMM\_WORLD, &numprocs);
25. MPI\_Comm\_rank(MPI\_COMM\_WORLD, &myid);
26. MPI\_Get\_processor\_name(processor\_name, &namelen);
28. **while** (!done) {
29. **if** (myid == 0) {
30. fprintf(stdout, "Enter the number of intervals: (0 quits) ");
31. fflush(stdout);
33. **if** (scanf("%d", &n) != 1) {
34. fprintf(stdout, "No number entered; quitting\n");
35. n = 0;
36. }
38. /\* Запуск секундомера \*/
39. startwtime = MPI\_Wtime();
40. }
42. /\* Рассылка интервалов всем процессорам, в том числе и самому себе \*/
43. MPI\_Bcast(&n, 1, MPI\_INT, 0, MPI\_COMM\_WORLD);
45. **if** (n == 0) {
46. done = 1;
47. } **else** {
48. h   = 1.0 / (**double**) n;
49. sum = 0.0;
51. **for** (i = myid + 1; i <= n; i += numprocs) {
52. x = h \* ((**double**)i - 0.5);
53. sum += f(x);
54. }
56. mypi = h \* sum;
58. /\* Операция редукции (все посчитанные значения в одно) \*/
59. MPI\_Reduce(&mypi, &pi, 1, MPI\_DOUBLE, MPI\_SUM, 0, MPI\_COMM\_WORLD);
61. **if** (myid == 0) {
62. printf("pi is approximately %.16f, Error is %.16f\n",
63. pi, fabs(pi - PI25DT));
65. endwtime = MPI\_Wtime();
66. printf("wall clock time = %f\n", endwtime-startwtime);
67. fflush(stdout);
68. }
69. }
70. }
72. /\* Завершение работы MPI \*/
73. MPI\_Finalize();
75. **return** 0;
76. }

## Тестирование программы

На 4 узлах:

|  |  |  |
| --- | --- | --- |
| **N** | **π** | **T** |
| 100 | 3,1416009869231249 | 0,000860 |
| 10 000 | 3,1415926544231239 | 0,000782 |
| 1 000 000 | 3,1415926535899033 | 0,003152 |
| 100 000 000 | 3,1415926535902168 | 0,310328 |
| 10 000 000 000 | 3,1415926535899734 | 2,361767 |

На 8 узлах:

|  |  |  |
| --- | --- | --- |
| **N** | **π** | **T** |
| 100 | 3,1416009869231249 | 0,001139 |
| 10 000 | 3,1415926544231247 | 0,001110 |
| 1 000 000 | 3,1415926535898899 | 0,002057 |
| 100 000 000 | 3,1415926535896137 | 0,083598 |
| 10 000 000 000 | 3,1415926535898202 | 1,170985 |

На 16 узлах:

|  |  |  |
| --- | --- | --- |
| **N** | **π** | **T** |
| 100 | 3,1416009869231249 | 0,000959 |
| 10 000 | 3,1415926544231239 | 0,001018 |
| 1 000 000 | 3,1415926535899033 | 0,001391 |
| 100 000 000 | 3,1415926535902168 | 0,042516 |
| 10 000 000 000 | 3,1415926535899734 | 0,585830 |

На 32 узлах:

|  |  |  |
| --- | --- | --- |
| **N** | **π** | **T** |
| 100 | 3,1416009869231249 | 0,000905 |
| 10 000 | 3,1415926544231247 | 0,000773 |
| 1 000 000 | 3,1415926535898899 | 0,000991 |
| 100 000 000 | 3,1415926535896137 | 0,028457 |
| 10 000 000 000 | 3,1415926535898202 | 0,305208 |

На 40 узлах:

|  |  |  |
| --- | --- | --- |
| **N** | **π** | **T** |
| 100 | 3,1416009869231249 | 0,000827 |
| 10 000 | 3,1415926544231239 | 0,000840 |
| 1 000 000 | 3,1415926535899033 | 0,001047 |
| 100 000 000 | 3,1415926535902168 | 0,017874 |
| 10 000 000 000 | 3,1415926535899734 | 0,249704 |

Рассчитаем коэффициент ускорения для каждой группы итераций, по формуле:

где — время итераций на i узлах.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **N** | **K­4** | **K8** | **K16** | **K32** | **K40** |
| 100 | 1 | 0,755048 | 0,896767 | 0,950276 | 1,039903 |
| 10 000 | 1 | 0,704505 | 0,768173 | 1,011643 | 0,930952 |
| 1 000 000 | 1 | 1,532329 | 2,265996 | 3,180626 | 3,010506 |
| 100 000 000 | 1 | 3,712146 | 7,299087 | 10,905155 | 17,361978 |
| 10 000 000 000 | 1 | 2,016906 | 4,031489 | 7,738221 | 9,458267 |

## Вывод

В ходе выполнения лабораторной работы были получены базовые знания о технологии MPI и получен навык работы с простейшими средствами передачи сообщений между процессами. Была произведена компиляция, запуск и тестирование программы-примера, написанной с использованием технологии MPI. Программа решает прикладную задачу — приближенное нахождение числа π. Была измерена производительность программы на различном кол-ве узлов. Измерения подтвердили, что при высокой точности вычислений программа работает быстрее на большем кол-ве узлов.