Белгородский государственный технологический университет им. В. Г. Шухова

Кафедра программного обеспечения вычислительной техники  
и автоматизированных систем

## Лабораторная работа №5 по теме: «Решение практических задач с применением технологии MPI»

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

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

Белгород  
2015

**Цель работы:** получить практический навык использования технологии MPI при решении прикладных задач.

# 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

Решить задачу Коши для системы обыкновенных дифференциальных уравнений первого порядка методом Рунге-Кутты.

## Алгоритм решения

Пусть дана задача Коши для системы дифференциальных уравнений первого порядка:

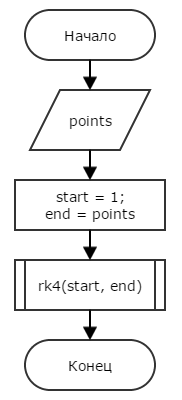
где — независимая переменная, — производная -ой неизвестной функции, — значение независимой переменной в начальный момент времени, — значение -ой неизвестной функции в начальный момент времени.

Зададимся некоторым ненулевым шагом интегрирования . Тогда приближённое значение -ой неизвестной функции в точке можно вычислить по формуле:

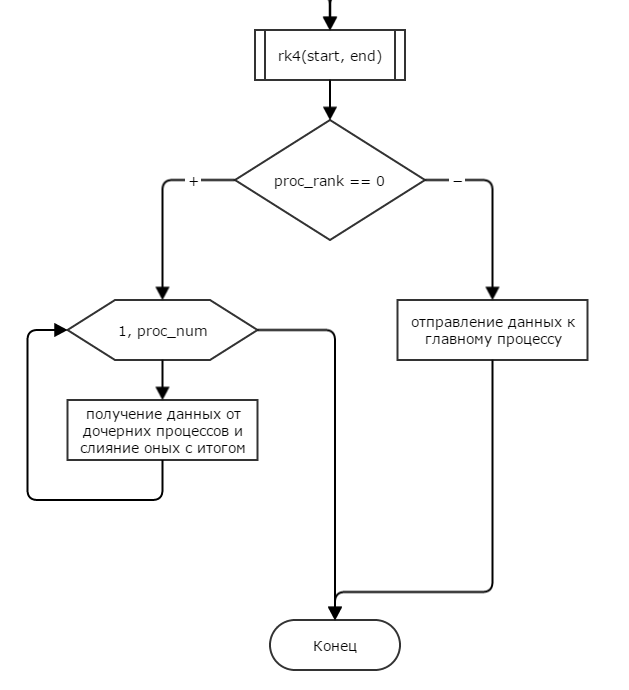
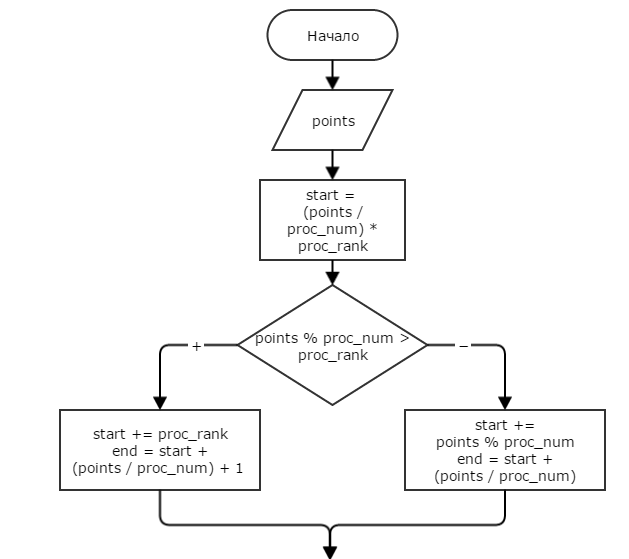
где:

Поскольку при программировании алгоритма Рунге-Кутты программируется также работа с матрицами, то для распараллеливания можно разным вычислителям отправлять на вычисление разные части матрицы, после чего «склеивать» результаты.

## Блок-схема итерационного алгоритма



## Блок-схема параллельного алгоритма



## Исходный код

Функция для численного решения системы дифференциальных уравнений методом Рунге-Кутты:

1. /\*
2. count -- количество уравнений в системе
3. \*/
4. **template** < **typename** T, **int** count >
6. /\*
7. syst -- система уравнений:
8. x -- параметр системы;
9. ys -- массив производных;
10. rightParts -- массив правых частей.
11. initConds -- массив начальных значений;
12. first -- начальная точка отрезка;
13. last -- конечная точка отрезка;
14. xOutPoints -- результат вычислений параметра;
15. ysOutMatrix -- результат вычисления производных;
16. pointsNumb -- количество точек для вывода;
17. start -- точка начала вычисления;
18. end -- точка конца вычисления;
19. stepsFact -- необходимая точность.
20. \*/
21. **int** rk4(**void** (\*syst)(T x, T\* ys, T\* rightParts),
22. T\* initConds, T first, T last,
23. T\* xOutPoints, T\*\* ysOutMatrix,
24. **int** pointsNumb, **int** start, **int** end,
25. **int** stepsFact = 1) {
27. **if** ((!count) || (pointsNumb <= 1) || (!stepsFact)) {
28. // -1: неверные данные.
29. **return** -1;
30. }
32. **if** (last < first) {
33. // -2: конечная точка не может быть меньше начальной.
34. **return** -2;
35. }
37. // количество шагов алгоритма
38. **const** **int** steps = (pointsNumb - 1) \* stepsFact;
39. // шаг интегрирования
40. **const** T h = (last - first) / T(steps);
42. // временные массивы для вычисления
43. T varYs[count], tmpYs[count];
44. T c1[count], c2[count], c3[count], c4[count];
46. **int** i, j, i\_out;
48. T x = xOutPoints[0] = first;
50. **for** (j = 0; j < count; ++j) {
51. ysOutMatrix[j][0] = varYs[j] = initConds[j];
52. }
54. **if** (start == 0) {
55. start = 1;
56. }
58. **for** (i\_out = start; i\_out < end; ++i\_out) {
60. **for** (i = 0; i < stepsFact; ++i) {
62. syst(x, varYs, c1);
64. **for** (j = 0; j < count; ++j) {
65. tmpYs[j] = varYs[j] + c1[j] \* 0.5 \* h;
66. }
68. x +=  0.5 \* h;
69. syst(x, tmpYs, c2);
71. **for** (j = 0; j < count; ++j) {
72. tmpYs[j] = varYs[j] + c2[j] \* 0.5 \* h;
73. }
75. syst(x, tmpYs, c3);
77. **for** (j = 0; j < count; ++j) {
78. tmpYs[j] = varYs[j] + c3[j] \* h;
79. }
81. x +=  0.5 \* h;
82. syst(x, tmpYs, c4);
84. **for** (j = 0; j < count; ++j) {
85. varYs[j] += (c1[j] + (c2[j] + c3[j]) + (c2[j] + c3[j]) + c4[j]) \* h / 6.0;
86. }
88. }
90. xOutPoints[i\_out] = x;
92. **for** (j = 0; j < count; ++j) {
93. ysOutMatrix[j][i\_out] = varYs[j];
94. }
96. }
98. **return** 0;
99. }

Основной модуль программы:

1. #include "rk4.h"
2. #include "mpi.h"
3. #include <iostream>
4. #include <math.h>
6. **using** std::cout;
7. **using** std::cin;
8. **using** std::endl;
10. // подсчёт начала и конца вычислений
11. **void** getStartEnd(**int** points, **int** proc\_num, **int** proc\_rank, **int** &start, **int** &end) {
12. start = (points / proc\_num) \* proc\_rank;
13. **if** (points % proc\_num > proc\_rank) {
14. start += proc\_rank;
15. end = start + (points / proc\_num) + 1;
16. } **else** {
17. start += points % proc\_num;
18. end = start + (points / proc\_num);
19. }
20. }
22. // система, которую нужно решить
23. **void** s(**double** x, **double** ys[], **double** rp[]) {
24. // dy0/dx = y1  
     rp[0] = ys[1];
25. // dy1/dx = -1/2 \* y1 - 2 \* y0 + 3 \* sin(x)    
     rp[1] = -1/2 \* ys[1] - 2 \* ys[0] + 3 \* sin(x);
26. }
28. // количество уравнений в системе
29. **const** **int** num = 2;
30. // массив начальных значений
31. **double** iC[num] = {1.0, 2.0};
32. // границы отрезка вычислений
33. **double** a = 0.0, b = 1.0;
35. **int** main(**int** argc, **char** \*argv[]) {
36. **int** i, j, k, l, points, eps, erc, start, end;
37. **int** proc\_num, proc\_rank;
38. **char** c;
39. **double** time;
40. **double** \*xOut;
41. **double** \*ysOut[num];
42. MPI\_Status status;
44. // инициализируем MPI
45. MPI\_Init(&argc, &argv);
46. // proc\_num -- общее число процессоров
47. MPI\_Comm\_size(MPI\_COMM\_WORLD, &proc\_num);
48. // proc\_rank -- ранг текущего процессора
49. MPI\_Comm\_rank(MPI\_COMM\_WORLD, &proc\_rank);
51. **if** (!proc\_rank) {
52. time = MPI\_Wtime();
53. }
55. **if** (argc < 2) {
56. cout << "Need more arguments (steps)" << endl;
57. exit(-1);
58. } **else** {
59. points = atoi(argv[1]) + 1;
60. // eps = atoi(argv[2]);
61. eps = 100000;
62. }
64. xOut = **new** **double** [points];
65. **for**(i = 0; i < num; ++i) {
66. ysOut[i] = **new** **double** [points];
67. }
69. // нахождение отрезка матрицы, который должен считать этот процесс
70. getStartEnd(points, proc\_num, proc\_rank, start, end);
72. // вычисление СДУ методом Рунге-Кутты
73. erc = rk4<**double**, num>(s, iC, a, b, xOut, ysOut, points, start, end, eps);
75. /\*
76. \*  если текущий процесс главный,
77. \*  то 'сливаем' все отрезки матриц в итоговую
78. \*/
79. **if** (!proc\_rank) {
80. **for** (**int** i = 1; i < proc\_num; i++) {
81. **double** \*tmpXOut;
82. **double** \*tmpYsOut[num];
84. tmpXOut = **new** **double** [points];
85. **for**(j = 0; j < num; ++j) {
86. tmpYsOut[j] = **new** **double** [points];
87. }
89. // получаем матрицу значений параметров
90. //MPI\_Recv(buffer, count, type, from, tag, communicator, status)
91. MPI\_Recv(tmpXOut, points, MPI\_DOUBLE, MPI\_ANY\_SOURCE, 1, MPI\_COMM\_WORLD,  
     &status);
92. // вычисляем, какой её отрезок необходим
93. getStartEnd(points, proc\_num, status.MPI\_SOURCE, start, end);
94. // вставляем его в общий результат
95. **for** (**int** i = start; i < end; ++i) {
96. xOut[i] = tmpXOut[i];
97. }
99. // получаем матрицу значений производных
100. MPI\_Recv(\*tmpYsOut, points \* num, MPI\_DOUBLE, MPI\_ANY\_SOURCE, 2,  
      MPI\_COMM\_WORLD, &status);
101. // вычисляем, какой её отрезок необходим
102. getStartEnd(points, proc\_num, status.MPI\_SOURCE, start, end);
103. // вставляем его в общий результат
104. **for** (**int** i = start; i < end; ++i) {
105. xOut[i] = tmpXOut[i];
107. **for** (j = 0; j < num; ++j) {
108. ysOut[j][i] = tmpYsOut[j][i];
109. }
110. }
112. // чистим память
113. **delete** [] tmpXOut;
114. **for** (j = 0; j < num; ++j) {
115. **delete**[] tmpYsOut[j];
116. }
117. }
118. } **else** {
119. // отправляем подсчитанные матрицы главному процессу
120. // MPI\_Send(buffer, count, type, to, tag, communicator)
121. MPI\_Send(xOut, points, MPI\_DOUBLE, 0, 1, MPI\_COMM\_WORLD);
122. MPI\_Send(\*ysOut, points \* num, MPI\_DOUBLE, 0, 2, MPI\_COMM\_WORLD);
123. }
125. // чистим память
126. **delete** [] xOut;
127. **for** (i = 0; i < num; ++i) {
128. **delete**[] ysOut[i];
129. }
131. **if** (!proc\_rank) {
132. time = MPI\_Wtime() - time;
133. // выводим время работы
134. cout << "Time: " << time << endl;
135. }
137. // завершаем работу MPI
138. MPI\_Finalize();
140. **return** 0;
141. }

## Тестирование программы

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **N­ит**  **Nузлов** | **10** | **100** | **1000** | **10 000** |
| **4** |  |  |  |  |
| **8** |  |  |  |  |
| **16** |  |  |  |  |
| **32** |  |  |  |  |
| **40** |  |  |  |  |