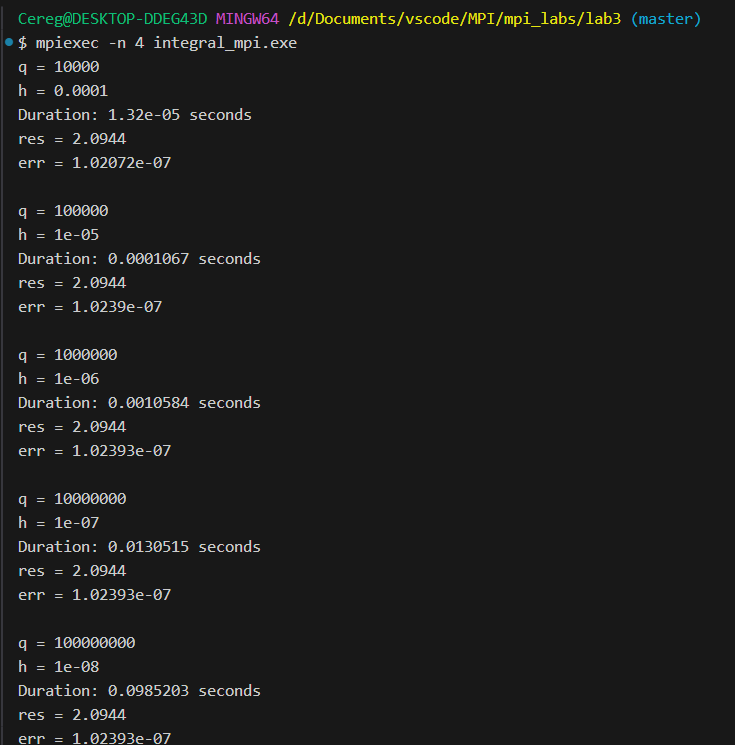
3. «Решение интеграла»

В работе необходимо вычислить случайный интеграл средней сложности на диапазоне от (например) 1 до 1000 с шагом 1 (от 1 до 2, от 2 до 3 и т.д.) для проверки алгоритма вычисления. В дальнейшем шаг вычислений уменьшить в 100-1000 раз и запустить в последовательном и параллельном режимах работы аналогично предыдущим лабораторным работам.



1. /\*
2. \* Программа вычисляет определенный интеграл функции 4/sqrt(4 - x²) на интервале [0, 1]
3. \* методом прямоугольников с использованием параллельных вычислений через MPI.
4. \* Сравнивает результат с эталонным значением и выводит погрешность.
5. \*/
6. #**include** <iostream>
7. #**include** <math.h>
8. #**include** <ctime>
9. #**include** <mpi.h> // Для работы с MPI
10. #**include** <chrono> // Для точного замера времени
11. #**define** NUM 5 // Количество тестовых случаев
12. const double refer = 2.094395; // Эталонное значение интеграла
13. /\*\*
14. \* Вычисляет интеграл методом средних прямоугольников с распараллеливанием через MPI
15. \*
16. \* @param q Количество интервалов разбиения
17. \* @param h Шаг интегрирования
18. \* @return Приближенное значение интеграла
19. \*/
20. double **integral**(int q, double h)
21. {
22. int rank, size;
23. MPI\_Comm\_rank(MPI\_COMM\_WORLD, &rank); // Получаем ранг текущего процесса
24. MPI\_Comm\_size(MPI\_COMM\_WORLD, &size); // Получаем общее количество процессов
25. double local\_sum = 0.0; // Локальная сумма процесса
26. double global\_sum = 0.0; // Глобальная сумма после редукции
27. // Распределение работы между процессами
28. int chunk = q / size; // Количество интервалов на процесс
29. int start = rank \* chunk; // Начальный индекс для процесса
30. int end = (rank == size - 1) ? q : start + chunk; // Корректировка для последнего процесса
31. **auto** t1 = std::chrono::high\_resolution\_clock::now(); // Старт замера времени
32. // Локальные вычисления интеграла для своего блока интервалов
33. **for** (int i = start; i < end; i++)
34. {
35. double x = h \* i + h / 2; // Середина интервала
36. local\_sum += (4.0 / sqrt(4.0 - x \* x)) \* h; // Формула метода прямоугольников
37. }
38. **auto** t2 = std::chrono::high\_resolution\_clock::now(); // Окончание замера
39. std::chrono::duration<double> duration = t2 - t1;
40. // Сбор и суммирование результатов на процессе 0 (MPI\_Reduce)
41. MPI\_Reduce(&local\_sum, &global\_sum, 1, MPI\_DOUBLE, MPI\_SUM, 0, MPI\_COMM\_WORLD);
42. // Вывод времени вычислений только на корневом процессе
43. **if** (rank == 0)
44. {
45. std::cout << "Duration: " << duration.count() << " seconds" << std::endl;
46. }
47. **return** global\_sum;
48. }
49. int **main**(int argc, char \*\*argv)
50. {
51. MPI\_Init(&argc, &argv); // Инициализация MPI
52. int rank;
53. MPI\_Comm\_rank(MPI\_COMM\_WORLD, &rank); // Получаем ранг процесса
54. const double a = 0.0, b = 1.0; // Границы интегрирования
55. int q[NUM] = {10000, 100000, 1000000, 10000000, 100000000}; // Наборы разбиений
56. double h[NUM]; // Массив шагов интегрирования
57. // Вычисление для разных значений разбиения
58. **for** (int i = 0; i < NUM; i++)
59. {
60. h[i] = (b - a) / static\_cast<double>(q[i]); // Расчет шага
61. // Вывод параметров только на корневом процессе
62. **if** (rank == 0)
63. {
64. std::cout << "q = " << q[i] << std::endl;
65. std::cout << "h = " << h[i] << std::endl;
66. }
67. double res = integral(q[i], h[i]); // Вычисление интеграла
68. // Вывод результатов и погрешности только на корневом процессе
69. **if** (rank == 0)
70. {
71. std::cout << "res = " << res << std::endl;
72. std::cout << "err = " << fabs(res - refer) << "\n\n";
73. }
74. }
75. MPI\_Finalize(); // Завершение работы с MPI
76. **return** 0;
77. }