МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ

ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«ВЯТСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

ФАКУЛЬТЕТ КОМПЬЮТЕРНЫХ И ФИЗИКО-МАТЕМАТИЧЕСКИХ НАУК

КАФЕДРА ПРИКЛАДНОЙ МАТЕМАТИКИ И ИНФОРМАТИКИ

Отчёт по лабораторной работе №7 по дисциплине «Параллельное программирование»

**Коллективные обмены в MPI**

|  |  |
| --- | --- |
| Выполнил: студент группы ФИб-4302-51-00 | / Д.А. Савин / |
| Проверил: ст. преподаватель каф. ПмиИ | / В.А. Бызов / |

Киров 2021

Задание 1

Написать параллельную программу, вычисляющую ln 2 по формуле  (*N* ≤ 5∙109).   
Реализовать а) с использованием парных операций, б) с использованием коллективных операций.

Замерить среднее время выполнения программы для *n* = 108, 5∙108, 109 (на каждое – не менее 3 запусков) для 1, 2, 4 и 8 процессов. Вычислить среднее ускорение для 2, 4 и 8 процессов. Рассчитать эффективность параллельного алгоритма. Построить графики ускорения и эффективности.

Сравнить результаты использования парных и коллективных операций.

Код:

|  |
| --- |
| #include <iostream>  #include <math.h>  #include <mpi.h>  using namespace std;  int main(int argc, char\* argv[])  {  int proc\_rank, proc\_num;  long long N;  double sum = 0;  MPI\_Status st;  MPI\_Init(&argc, &argv);  MPI\_Comm\_rank(MPI\_COMM\_WORLD, &proc\_rank);  MPI\_Comm\_size(MPI\_COMM\_WORLD, &proc\_num);  double start, end;  if (proc\_rank == 0)  {  // вводим n и рассылаем всем процессам  std::cout << "input N: ";  N = pow(10, 8);  cout << N << endl;  //std::cin >> N;  start = MPI\_Wtime();  for (int i = 1; i < proc\_num; ++i)  MPI\_Send(&N, 1, MPI\_LONG\_LONG, i, 0, MPI\_COMM\_WORLD);  }  else  {  MPI\_Recv(&N, 1, MPI\_LONG\_LONG, 0, MPI\_ANY\_TAG, MPI\_COMM\_WORLD, &st);  }  // считаем значение  for (long long k = proc\_rank + 1; k <= N; k += proc\_num)  sum += (((k - 1) & 1) == 1 ? -1.0 : 1.0) / k;  // отправляем главному процессу  if (proc\_rank == 0)  {  double tmpsum;  // собираем значения других процессов и вычисляем результат  for (int i = 1; i < proc\_num; ++i)  {  MPI\_Recv(&tmpsum, 1, MPI\_LONG\_LONG, MPI\_ANY\_SOURCE, MPI\_ANY\_TAG, MPI\_COMM\_WORLD, &st);  sum += tmpsum;  }  end = MPI\_Wtime();  std::cout << "time to sum = " << end - start << "s\n";  printf("Result: %.10f", sum);  }  else  {  MPI\_Send(&sum, 1, MPI\_LONG\_LONG, 0, 0, MPI\_COMM\_WORLD);  }  MPI\_Finalize();  return 0;  } |

Изображение выглядит как текст

Автоматически созданное описание

|  |
| --- |
| #include <iostream>  #include <math.h>  #include <mpi.h>  #include <chrono>  int main(int argc, char\* argv[])  {  int proc\_rank, proc\_num;  long long N;  double sum = 0;  MPI\_Status st;  MPI\_Init(&argc, &argv);  MPI\_Comm\_rank(MPI\_COMM\_WORLD, &proc\_rank);  MPI\_Comm\_size(MPI\_COMM\_WORLD, &proc\_num);  double start, end;  if (proc\_rank == 0)  {  // вводим n  std::cout << "input N: ";  std::cin >> N;  start = MPI\_Wtime();  }  // рассылаем всем N  MPI\_Bcast(&N, 1, MPI\_LONG\_LONG, 0, MPI\_COMM\_WORLD);  // считаем сумму  double tmpsum = 0;  for (long long k = proc\_rank + 1; k <= N; k += proc\_num)  tmpsum += (((k - 1) & 1) == 1 ? -1.0 : 1.0) / k;  // собираем значения в сумме и выводим результаты  MPI\_Reduce(&tmpsum, &sum, 1, MPI\_DOUBLE, MPI\_SUM, 0, MPI\_COMM\_WORLD);  if (proc\_rank == 0)  {  end = MPI\_Wtime();  std::cout << "time to sum = " << end - start << std::endl;  printf("Result: %.10f", sum);  }  MPI\_Finalize();  return 0;  } |

Изображение выглядит как текст

Автоматически созданное описание

Парные операции:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Размер, n | Число процессов | | | |
| 1 | 2 | 4 | 8 |
| 10 8 | 0,254 | 0,13 | 0,0669 | 0,034 |
| 5 \* 10 8 | 1,281 | 0,651 | 0,328 | 0,166 |
| 10 9 | 2,550 | 1,293 | 0,655 | 0,33 |

Таблица 1 – Время работы в секундах

|  |  |  |  |
| --- | --- | --- | --- |
| Размер, n | Число процессов | | |
| 2 | 4 | 8 |
| 10 8 | 1,95384615 | 3,90769 | 7,47059 |
| 5 \* 10 8 | 1,96439628 | 3,90462 | 7,64458 |
| 10 9 | 1,97209302 | 3,91385 | 7,70909 |

Таблица 2 – Ускорение работы

|  |  |  |  |
| --- | --- | --- | --- |
| Размер, n | Число процессов | | |
| 2 | 4 | 8 |
| 10 8 | 0,962 | 0,776 | 0,462 |
| 5 \* 10 8 | 0,982 | 0,776 | 0,484 |
| 10 9 | 0,993 | 0,723 | 0,492 |
| Таблица 3. Эффективность для парных операций. | | | |

Коллективные операции:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Размер, n | Число процессов | | | |
| 1 | 2 | 4 | 8 |
| 10 8 | 0,254 | 0,13 | 0,0669 | 0,034 |
| 5 \* 10 8 | 1,281 | 0,651 | 0,328 | 0,166 |
| 10 9 | 2,55 | 1,293 | 0,655 | 0,33 |

Таблица 4 – Время работы при коллективных операциях

|  |  |  |  |
| --- | --- | --- | --- |
| Размер, n | Число процессов | | |
| 2 | 4 | 8 |
| 10 8 | 1,95384615 | 3,79671 | 7,47059 |
| 5 \* 10 8 | 1,96774194 | 3,90549 | 7,71687 |
| 10 9 | 1,97215777 | 3,89313 | 7,72727 |

Таблица 5 – Ускорение работы

|  |  |  |  |
| --- | --- | --- | --- |
| Размер, n | Число процессов | | |
| 2 | 4 | 8 |
| 10 8 | 0,962 | 0,776 | 0,462 |
| 5 \* 10 8 | 0,982 | 0,776 | 0,484 |
| 10 9 | 0,993 | 0,723 | 0,492 |
| Таблица 6. Эффективность для парных операций. | | | |

Как видно из таблиц и графиков, разница между парными операциями и коллективными на уровне погрешности из-за того, что в ОС запущены прочие процессы.

Задание 2

Написать параллельную программу, проверяющую, является ли заданный массив размера *N* (*N* ≤ 5∙108) упорядоченным. Замерить среднее время выполнения программы для *N* = 108, 5∙108 (на каждое – не менее 3 запусков) для 1, 2, 4 и 8 процессов. Вычислить среднее ускорение для 2, 4 и 8 процессов. Рассчитать эффективность параллельного алгоритма. Построить графики ускорения и эффективности.

Код:

|  |
| --- |
| #include <iostream>  #include <math.h>  #include <mpi.h>  #include <string>  #include <algorithm>  using namespace std;  const int answerTag = 0;  const int feedbackTag = 1;  // 1 - по возрастанию, -1 - по убыванию, 0 - массив равен, INT\_MAX - разные  int checkArray(int\* arr, int size)  {  int result = 0;  for (int i = 1; i < size; ++i)  {  int temp = 0;  if (arr[i - 1] < arr[i])  temp = 1;  else if (arr[i - 1] > arr[i])  temp = -1;  if (result == 0)  result = temp;  else if (result != temp && temp != 0)  {  result = INT\_MAX;  break;  }  }  return result;  }  int main(int argc, char\* argv[])  {  int proc\_rank, proc\_num;  MPI\_Status st;  MPI\_Init(&argc, &argv);  MPI\_Comm\_rank(MPI\_COMM\_WORLD, &proc\_rank);  MPI\_Comm\_size(MPI\_COMM\_WORLD, &proc\_num);  double start, end;  int n;  int\* arr = NULL;  if (proc\_rank == 0)  {  std::cout << "Input N:";  n = pow(10, 8);  cout << n << endl;  //std::cin >> n;  std::cout << "Choose:\n1. Random\n2. Ascending\n3. Descending" << std::endl;  int plan;  std::cin >> plan;  if (plan < 1 || plan > 3)  return 0;  // заполняем массив так, как нам надо  arr = new int[n];  for (int i = 0; i < n; ++i)  {  if (plan == 1)  arr[i] = rand();  else if (plan == 2)  arr[i] = i;  else arr[i] = n - i;  }  start = MPI\_Wtime(); // записываем время начала рассчета  for (int i = 1; i < proc\_num; ++i) // отправляем размер всем  MPI\_Send(&n, 1, MPI\_INT, i, answerTag, MPI\_COMM\_WORLD);  }  else  {  // принимаем размер n  MPI\_Recv(&n, 1, MPI\_INT, 0, answerTag, MPI\_COMM\_WORLD, &st);  }  int\* sendcounts = new int[proc\_num]; // кол-во элементов  int\* displs = new int[proc\_num]; // смещение  int remainder = n; // остаток  for (int i = 0; i < proc\_num; ++i)  {  displs[i] = i > 0 ? displs[i - 1] + sendcounts[i - 1] - 1 : 0; // смещение = смещение старого значения + кол-во элементов у пред. элемента - 1  sendcounts[i] = ceil(remainder / (double)(proc\_num - i)) + (i != proc\_num - 1 ? 1 : 0); // кол-во элементов = остаток / кол-во оставшихся процессов + (если процесс не последний, то добавляем границу следующего)  remainder = remainder - (sendcounts[i] - 1); // остаток = остаток - (кол-во элементов - 1)  }  // буфер для приёма  int\* buff = new int[sendcounts[proc\_rank]];  // распределяем между процессами  MPI\_Scatterv(arr, sendcounts, displs, MPI\_INT, buff, sendcounts[proc\_rank], MPI\_INT, 0, MPI\_COMM\_WORLD);  // получаем результат  int res = checkArray(buff, sendcounts[proc\_rank]);  // если главный процесс и упорядочено, то получаем значения других процессов и смотрим результат  if (proc\_rank == 0)  {  if (res != INT\_MAX)  {  for (int i = 1; i < proc\_num; ++i)  {  int temp;  MPI\_Recv(&temp, 1, MPI\_INT, MPI\_ANY\_SOURCE, feedbackTag, MPI\_COMM\_WORLD, &st);  if (res == 0)  res = temp;  else if (temp != res && temp != 0) // если значения не совпадают, значит массив не упорядочен  {  res = INT\_MAX;  break;  }  }  }  end = MPI\_Wtime();  std::cout << "Time: " << end - start << std::endl;  std::cout << "Res: " << (res != INT\_MAX ? (res == 1 ? "Ascending" : "Descending") : "FALSE") << std::endl;  }  else  {  // отправляем результат главному процессу  MPI\_Send(&res, 1, MPI\_INT, 0, feedbackTag, MPI\_COMM\_WORLD);  }  // чистим память  delete[] sendcounts;  delete[] displs;  delete[] arr;  delete[] buff;  MPI\_Finalize();  return 0;  } |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Размер, n | Число процессов | | | |
| 1 | 2 | 4 | 8 |
| 10 8 | 0,286 | 0,180 | 0,133 | 0,115 |
| 5 \* 10 8 | 1,443 | 0,909 | 0,666 | 0,576 |
| Таблица 7 – Время работы алгоритма, сек. | | | | |

|  |  |  |  |
| --- | --- | --- | --- |
| Размер, n | Число процессов | | |
| 2 | 4 | 8 |
| 10 8 | 1,58888889 | 2,1504 | 2,487 |
| 5 \* 10 8 | 1,58745875 | 2,1667 | 2,5052 |

Таблица 8 – Ускорение работы

|  |  |  |  |
| --- | --- | --- | --- |
| Размер, n | Число процессов | | |
| 2 | 4 | 8 |
| 10 8 | 0,814 | 0,535 | 0,301 |
| 5 \* 10 8 | 0,848 | 0,598 | 0,329 |
| Таблица 9 – Эффективность распараллеливания. | | | |

Задание 3

Написать параллельную программу, вычисляющую произведение квадратной матрицы размера N x N на вектор размера N (N <= 10000). Замерить среднее время выполнения программы для N = 3000, 5000 и 10000 (на каждое – не менее 3 запусков) для 1, 4 и 16 процессов. Вычислить среднее ускорение для 4 и 16 процессов. Рассчитать эффективность параллельного алгоритма. Построить графики ускорения и эффективности.

|  |
| --- |
| #include <iostream>  #include <cmath>  #include <mpi.h>  using namespace std;  int proc\_rank, proc\_num;  int\* Create1(int n)  {  int\* x = new int[n \* n];  for (int i = 0; i < n; i++)  {  for (int j = 0; j < n; j++)  {  x[i \* n + j] = 1;  }  }  return x;  }  int\* Create2(int n)  {  int\* x = new int[n];  for (int i = 0; i < n; i++)  {  x[i] = 1;  }  return x;  }  int main(int argc, char\* argv[])  {  const int buf\_size = 1;  int n = 3000;  int\* x = new int[1], \* y = new int[n], \* z;  double start, end, diff;  MPI\_Init(&argc, &argv);  MPI\_Comm\_rank(MPI\_COMM\_WORLD, &proc\_rank);  MPI\_Comm\_size(MPI\_COMM\_WORLD, &proc\_num);  if (proc\_rank == 0)  {  delete[] x;  delete[] y;  x = Create1(n);  y = Create2(n);  start = MPI\_Wtime();  }  int\* countsx = new int[proc\_num];  int\* countsrx1 = new int[proc\_num];  double k = (double)n / proc\_num;  int k1 = (int)k;  if (n % proc\_num != 0)  k1 = n / proc\_num + 1;  for (int i = 0; i < proc\_num; i++)  {  countsx[i] = min((int)((n - k1 \* i) \* n), (int)(k1 \* n));  countsrx1[i] = i \* n \* k1;  }  int\* countsy = new int[proc\_num];  int\* countsry1 = new int[proc\_num];  for (int i = 0; i < proc\_num; i++)  {  countsy[i] = min((int)((n - k1 \* i)), (int)(k1));  countsry1[i] = i \* k1;  }  int\* x1 = new int[countsx[proc\_rank]];  MPI\_Scatterv(x, countsx, countsrx1, MPI\_INTEGER, x1, countsx[proc\_rank], MPI\_INTEGER, 0, MPI\_COMM\_WORLD);  MPI\_Bcast(y, n, MPI\_INTEGER, 0, MPI\_COMM\_WORLD);  int\* sum = new int[countsy[proc\_rank]];  for (int i = 0; i < countsy[proc\_rank]; i++)  {  sum[i] = 0;  for (int j = 0; j < n; j++) {  sum[i] += x1[i \* n + j] \* y[j];  }  }  z = new int[n];  MPI\_Gatherv(sum, countsy[proc\_rank], MPI\_INTEGER, z, countsy, countsry1, MPI\_INTEGER, 0, MPI\_COMM\_WORLD);  if (proc\_rank == 0)  {  end = MPI\_Wtime();  diff = end - start;  cout << diff << endl;  }  MPI\_Finalize();  delete[] x;  delete[] x1;  delete[] y;  delete[] z;  delete[] sum;  } |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Размер, n | Число процессов | | | |
| 1 | 2 | 4 | 8 |
| 3000 | 0,280 | 0,170 | 0,133 | 0,125 |
| 5000 | 0,076 | 0,046 | 0,033 | 0,0303 |
| 10000 | 0,315 | 0,187 | 0,130 | 0,111 |
| Таблица 10 – Время работы алгоритма, сек. | | | | |
| Размер, n | Число процессов | | |
| 2 | 4 | 8 |
| 3000 | 1,64705882 | 2,1053 | 2,24 |
| 5000 | 1,65217391 | 2,303 | 2,5083 |
| 10000 | 1,68449198 | 2,4231 | 2,8378 |

Таблица 11 – Ускорение работы

|  |  |  |  |
| --- | --- | --- | --- |
| Размер, n | Число процессов | | |
| 2 | 4 | 8 |
| 3000 | 0,814 | 0,535 | 0,301 |
| 5000 | 0,848 | 0,598 | 0,329 |
| 10000 | 0,935 | 0,584 | 0,312 |
| Таблица 12 – Эффективность распараллеливания. | | | |

Задание 4

Написать параллельную программу, в которой процессы с чётными номерами вычисляют минимальное значение в массиве из M элементов (M < 5∙108), а процессы с нечётными номерами вычисляют скалярное произведение векторов размера N (N < 5∙108). Числа M и N вводятся пользователем. Продемонстрировать корректность работы программы.

Код:

|  |
| --- |
| #include <iostream>  #include <cmath>  #include <mpi.h>  using namespace std;  int proc\_rank, proc\_num;  int\* Create1(int n)  {  int\* x = new int[n \* n];  for (int i = 0; i < n; i++)  {  for (int j = 0; j < n; j++)  {  x[i \* n + j] = 1;  }  }  return x;  }  int\* Create2(int n)  {  int\* x = new int[n];  for (int i = 0; i < n; i++)  {  x[i] = 1;  }  return x;  }  int main(int argc, char\* argv[])  {  const int buf\_size = 1;  int n = 3000;  int\* x = new int[1], \* y = new int[n], \* z;  double start, end, diff;  MPI\_Init(&argc, &argv);  MPI\_Comm\_rank(MPI\_COMM\_WORLD, &proc\_rank);  MPI\_Comm\_size(MPI\_COMM\_WORLD, &proc\_num);  if (proc\_rank == 0)  {  delete[] x;  delete[] y;  x = Create1(n);  y = Create2(n);  start = MPI\_Wtime();  }  int\* countsx = new int[proc\_num];  int\* countsrx1 = new int[proc\_num];  double k = (double)n / proc\_num;  int k1 = (int)k;  if (n % proc\_num != 0)  k1 = n / proc\_num + 1;  for (int i = 0; i < proc\_num; i++)  {  countsx[i] = min((int)((n - k1 \* i) \* n), (int)(k1 \* n));  countsrx1[i] = i \* n \* k1;  }  int\* countsy = new int[proc\_num];  int\* countsry1 = new int[proc\_num];  for (int i = 0; i < proc\_num; i++)  {  countsy[i] = min((int)((n - k1 \* i)), (int)(k1));  countsry1[i] = i \* k1;  }  int\* x1 = new int[countsx[proc\_rank]];  MPI\_Scatterv(x, countsx, countsrx1, MPI\_INTEGER, x1, countsx[proc\_rank], MPI\_INTEGER, 0, MPI\_COMM\_WORLD);  MPI\_Bcast(y, n, MPI\_INTEGER, 0, MPI\_COMM\_WORLD);  int\* sum = new int[countsy[proc\_rank]];  for (int i = 0; i < countsy[proc\_rank]; i++)  {  sum[i] = 0;  for (int j = 0; j < n; j++) {  sum[i] += x1[i \* n + j] \* y[j];  }  }  z = new int[n];  MPI\_Gatherv(sum, countsy[proc\_rank], MPI\_INTEGER, z, countsy, countsry1, MPI\_INTEGER, 0, MPI\_COMM\_WORLD);  if (proc\_rank == 0)  {  end = MPI\_Wtime();  diff = end - start;  cout << diff << endl;  }  MPI\_Finalize();  delete[] x;  delete[] x1;  delete[] y;  delete[] z;  delete[] sum;  } |

Изображение выглядит как текст, электроника, черный, снимок экрана

Автоматически созданное описание

Вывод

В результате выполнения лабораторной работы были получены навыки работы с коллективных взаимодействием технологии MPI и были реализованы задачи с применением MPI.