**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ  
 РОССИЙСКОЙ ФЕДЕРАЦИИ**

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**

**НОВОСИБИРСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ**

**Факультет информационных технологий**

**Кафедра параллельных вычислений**

**ОТЧЕТ**

**О ВЫПОЛНЕНИИ ПРАКТИЧЕСКОЙ РАБОТЫ**

«Два вектора»

студента 2 курса, группы 22204

***Гаан Артем Игоревич***

Направление 09.03.01 – «Информатика и вычислительная техника»

Преподаватель:

Власенко Андрей Юрьевич

Новосибирск 2024

**СОДЕРЖАНИЕ**

[**ЦЕЛЬ 3**](#_heading=h.30j0zll)

[**ЗАДАНИЕ 3**](#_heading=h.3znysh7)

[**ОПИСАНИЕ РАБОТЫ 4**](#_heading=h.tyjcwt)

[**ЗАКЛЮЧЕНИЕ 5**](#_heading=h.1ksv4uv)

[**Приложение 1. Листинг последовательной программы. 6**](#_heading=h.2jxsxqh)

[**Приложение 2. Листинг параллельной программы с коммуникацией типа точка-точка. 8**](#_heading=h.nrj85wlp1wbp)

## ЦЕЛЬ

Сравнить время выполнения трех программ:

1. последовательная программа
2. параллельная, использующая коммуникации типа точка-точка
3. параллельная, использующая коллективные коммуникации

## ЗАДАНИЕ

1. Написать 3 программы, каждая из которых рассчитывает число s по двум данным векторам a и b равной длины N в соответствии со следующим двойным циклом:

for (i = 0; i < N; i++)

for(j = 0; j < N; j++)

s += a[i] \* b[j];

a) последовательная программа

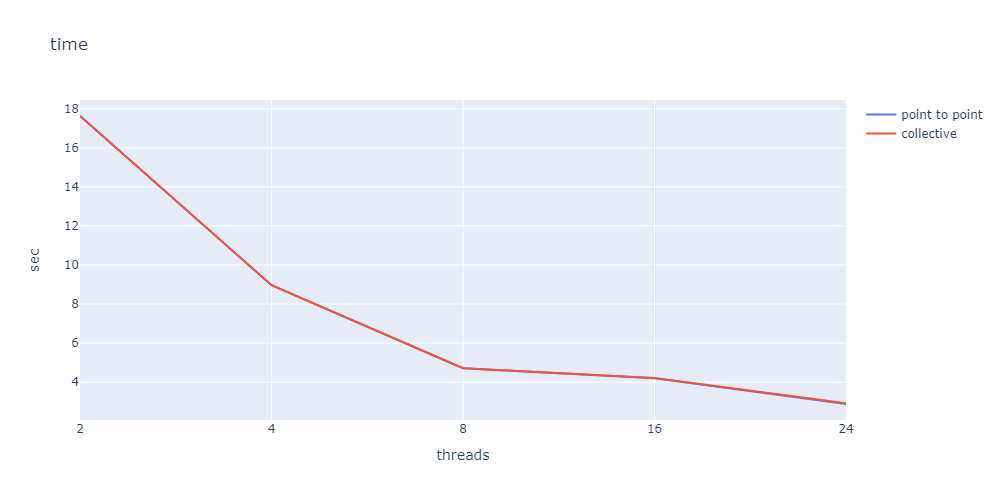
b) параллельная, использующая коммуникации типа точка-точка (MPI\_Send, MPI\_Recv)

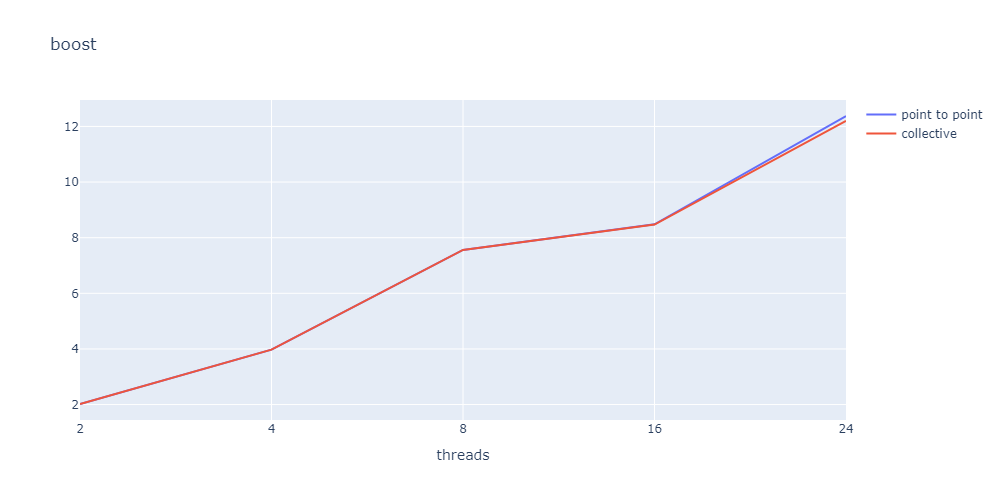
c) параллельная, использующая коллективные коммуникации (MPI\_Scatter, MPI\_Reduce, MPI\_Bcast)

1. Замерить время работы последовательной программы и параллельных на 2, 4, 8, 16, 24 процессах. Рекомендуется провести несколько замеров для каждого варианта запуска и выбрать минимальное время.
2. Построить графики времени, ускорения и эффективности.

## ОПИСАНИЕ РАБОТЫ

1. Было написано 3 программы в соответствии с заданием.
2. Было замерено время работы каждой программы с N = 99984. (было выбрано такое N, чтобы время работы последовательной программы было больше 30 секунд и было кратно числу потоков). Замер времени производился несколько раз для получения наиболее точного результата.
3. Были построены графики зависимости времени, ускорения и эффективности от количества процессов.

* Ускорение: Sp = T1 / Tp, где T1 - время работы ПОСЛЕДОВАТЕЛЬНОЙ программы. Tp - время работы параллельной программы на p процессах/потоках.
* Эффективность: Ep = Sp / p \* 100%.
* T1 = 35.620917.



## 

## 

## 

## 

## 

## 

## 

## 

## 

## 

## 

## 

## 

## 

## 

## 

## 

## 

## 

## 

## 

## 

## ЗАКЛЮЧЕНИЕ

В ходе выполнения данной работы была исследована зависимость времени, ускорения и эффективности от количества процессов. По полученным данным был построен график. Можно сделать вывод, что использование параллельного программирования помогает ускорить программу, но увеличение числа потоков не дает большую эффективность.

## Приложение 1. Листинг последовательной программы.

#include "stdio.h"

#include "stdlib.h"

#include "mpi.h"

#define N 99984

void fill\_vectors(int\* a, int\* b, const int size) {

if(a != NULL && b != NULL) {

for(size\_t i = 0; i < size; ++i) {

a[i] = i;

b[i] = i;

}

}

}

long long int find\_s(const int\* a, const int\* b, const int size) {

long long int s = 0;

if(a != NULL && b != NULL) {

for(size\_t i = 0; i < size; ++i) {

for(size\_t j = 0; j < N; ++j) {

s += a[i]\*b[j];

}

}

}

return s;

}

int main(int argc, char\*\* argv) {

MPI\_Init(&argc, &argv);

int\* a = malloc(sizeof(int) \* N);

int\* b = malloc(sizeof(int) \* N);

double start = MPI\_Wtime();

fill\_vectors(a, b, N);

const long long int s = find\_s(a, b, N);

double end = MPI\_Wtime();

printf("Time taken: %lf sec.\n", end - start);

printf("s is: %lld\n", s);

free(a);

free(b);

MPI\_Finalize();

return 0;

}

## Приложение 2. Листинг параллельной программы с коммуникацией типа точка-точка.

1. #include "stdio.h"
2. #include "stdlib.h"
3. #include "mpi.h"
4. #define N 99984
5. void fill\_vectors(int\* a, int\* b, const int size) {
6. if(a != NULL && b != NULL) {
7. for(size\_t i = 0; i < size; ++i) {
8. a[i] = i;
9. b[i] = i;
10. }
11. }
12. }
13. long long int find\_s(const int\* a, const int\* b, const int size) {
14. long long int s = 0;
15. if(a != NULL && b != NULL) {
16. for(size\_t i = 0; i < size; ++i) {
17. for(size\_t j = 0; j < N; ++j) {
18. s += a[i]\*b[j];
19. }
20. }
21. }
22. return s;
23. }
24. int main(int argc, char\*\* argv) {
25. int size, rank;
26. MPI\_Init(&argc, &argv);
27. MPI\_Comm\_size(MPI\_COMM\_WORLD, &size);
28. MPI\_Comm\_rank(MPI\_COMM\_WORLD, &rank);
29. const int size\_of\_block = N / size;
30. int\* a = malloc(sizeof(int) \* N);
31. int\* b = malloc(sizeof(int) \* N);
32. long long int s, tmp;
33. double start, end;
34. if(rank == 0) {
35. start = MPI\_Wtime();
36. fill\_vectors(a, b, N);
37. for(size\_t i = 1; i < size; ++i) {
38. MPI\_Send(a + size\_of\_block\*i, size\_of\_block, MPI\_INT, i, 1, MPI\_COMM\_WORLD);
39. MPI\_Send(b, N, MPI\_INT, i, 1, MPI\_COMM\_WORLD);
40. }
41. s = find\_s(a, b, size\_of\_block);
42. for(size\_t i = 1; i < size; ++i) {
43. MPI\_Recv(&tmp, 1, MPI\_LONG\_LONG\_INT, MPI\_ANY\_SOURCE, 2, MPI\_COMM\_WORLD, MPI\_STATUS\_IGNORE);
44. s += tmp;
45. }
46. } else {
47. MPI\_Recv(a + size\_of\_block\*rank, size\_of\_block, MPI\_INT, 0, 1, MPI\_COMM\_WORLD, MPI\_STATUS\_IGNORE);
48. MPI\_Recv(b, N, MPI\_INT, 0, 1, MPI\_COMM\_WORLD, MPI\_STATUS\_IGNORE);
49. tmp = find\_s(a + size\_of\_block\*rank, b, size\_of\_block);
50. MPI\_Send(&tmp, 1, MPI\_LONG\_LONG\_INT, 0, 2, MPI\_COMM\_WORLD);
51. }
52. if(rank == 0) {
53. end = MPI\_Wtime();
54. printf("Time taken: %lf sec.\n", end - start);
55. printf("s is: %lld\n", s);
56. }
57. free(a);
58. free(b);
59. MPI\_Finalize();
60. return 0;
61. }

## Приложение 2. Листинг параллельной программы с коллективными коммуникациями.

1. #include "stdio.h"
2. #include "stdlib.h"
3. #include "mpi.h"
4. #define N 99984
5. void fill\_vectors(int\* a, int\* b, const int size) {
6. if(a != NULL && b != NULL) {
7. for(size\_t i = 0; i < size; ++i) {
8. a[i] = i;
9. b[i] = i;
10. }
11. }
12. }
13. long long int find\_s(const int\* a, const int\* b, const int size) {
14. long long int s = 0;
15. if(a != NULL && b != NULL) {
16. for(size\_t i = 0; i < size; ++i) {
17. for(size\_t j = 0; j < N; ++j) {
18. s += a[i]\*b[j];
19. }
20. }
21. }
22. return s;
23. }
24. int main(int argc, char\*\* argv) {
25. int size, rank;
26. MPI\_Init(&argc, &argv);
27. MPI\_Comm\_size(MPI\_COMM\_WORLD, &size);
28. MPI\_Comm\_rank(MPI\_COMM\_WORLD, &rank);
29. const int size\_of\_block = N / size;
30. int\* a;
31. int\* b = malloc(sizeof(int) \* N);
32. long long int s, tmp;
33. double start, end;
34. if(rank == 0) {
35. a = malloc(sizeof(int) \* N);
36. start = MPI\_Wtime();
37. fill\_vectors(a, b, N);
38. s = 0;
39. }
40. int\* local = malloc(sizeof(int) \* size\_of\_block);
41. MPI\_Scatter(a, size\_of\_block, MPI\_INT, local, size\_of\_block, MPI\_INT, 0, MPI\_COMM\_WORLD);
42. MPI\_Bcast(b, N, MPI\_INT, 0, MPI\_COMM\_WORLD);
43. tmp = find\_s(local, b, size\_of\_block);
44. MPI\_Reduce(&tmp, &s, 1, MPI\_LONG\_LONG\_INT, MPI\_SUM, 0, MPI\_COMM\_WORLD);
45. free(local);
46. if(rank == 0) {
47. end = MPI\_Wtime();
48. printf("Time taken: %lf sec.\n", end - start);
49. printf("s is: %lld\n", s);
50. free(a);
51. }
52. free(b);
53. MPI\_Finalize();
54. return 0;
55. }