МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ

по лабораторной работе №2

по дисциплине «Параллельные алгоритмы»

Тема: Использование аргументов-джокеров.

Студент гр. 9383 _____ Лапина А.А.

Преподаватель _____ Татаринов Ю.С.

Санкт-Петербург

2021

Цель работы.

Написать программу нахождения результата, используя раздачу и сборку массива.

Формулировка задания.

Вариант 4.

Раздача и сборка массива. Процесс 0 генерирует целочисленный массив и раздает его по частям в остальные процессы; порядок раздачи определяется случайным образом, размер каждого следующего передаваемого фрагмента в 2 раза меньше предыдущего. Процессы-получатели выполняют обработку массива и возвращают результат в процесс 0. Процесс 0 должен собрать массив результатов обработки с сохранением последовательности элементов.

Краткое описание выбранного алгоритма решения задачи.

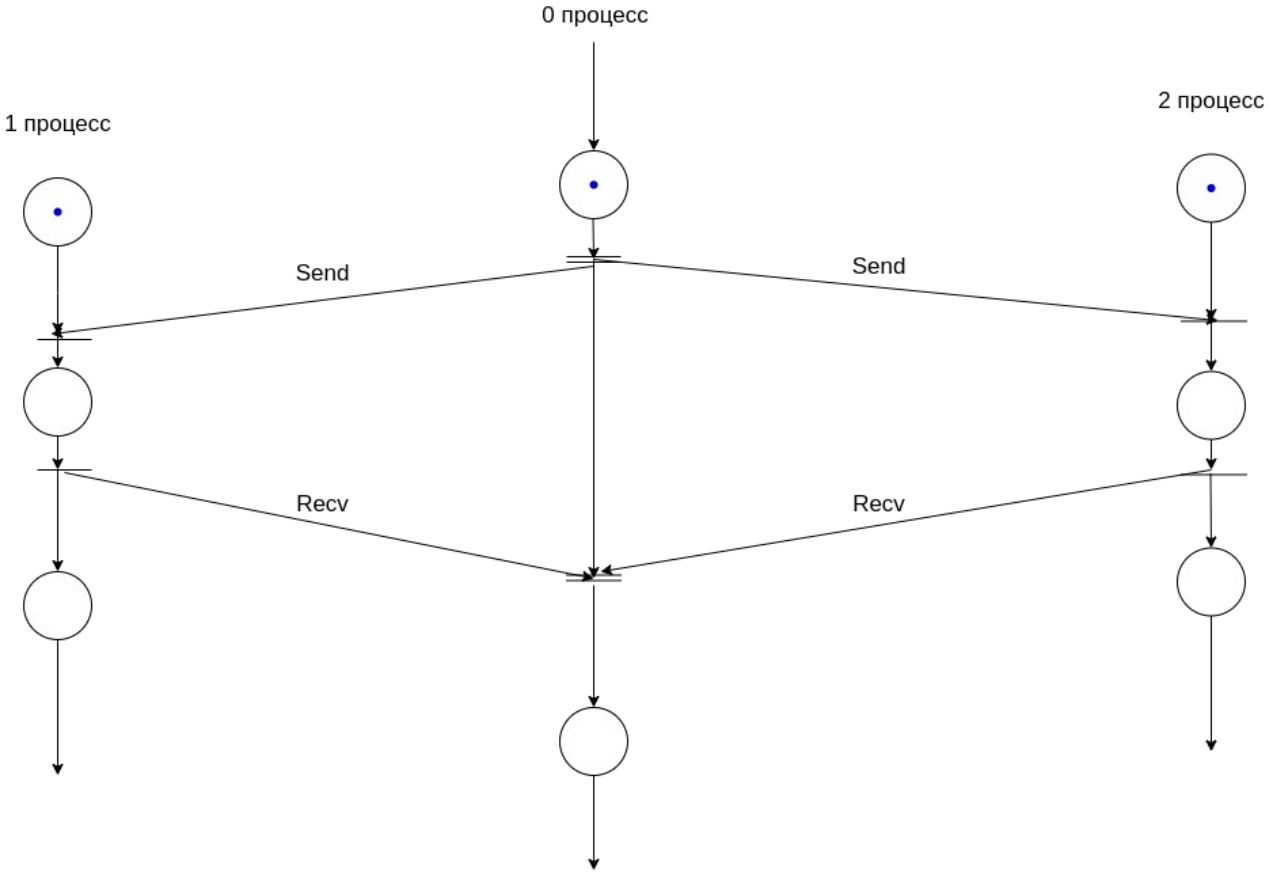
Для решения данной задачи выберем для обработки массива операцию сложения. (Ставим цель — узнать сумму элементов массива). Создаем структуру для хранения локальной и глобальной суммы. При запуске программы необходимо подать на вход параметр koef — коэффициент, который будет отвечать за размер массива (koef >=1). В переменной Size будем расчитывать размер массива, зависимый от коэффициента и количества процессов, так как в задании требуется, чтобы «размер каждого следующего передаваемого фрагмента в 2 раза меньше предыдущего». Так как порядок раздачи от 0 процесса другим определяется случайным образом, то создадим массив OrderArr, где определим случайным образом порядок раздачи (Сначала запишем номера по порядку и перемешаем). Затем создаем основной массив array, сумму элементов которого хотим вычислить, заполняем его рандомными числами от 0 до 100 (для удобства проверки выбрано число 100). С помощью NewSize будем определять размер отправляемого массива, каждый раз размер будет уменьшаться в 2 раза (по заданию). С помощью MPI_Send передаем

другим процессам информацию о длине передаваемого массива, а затем и весь массив.

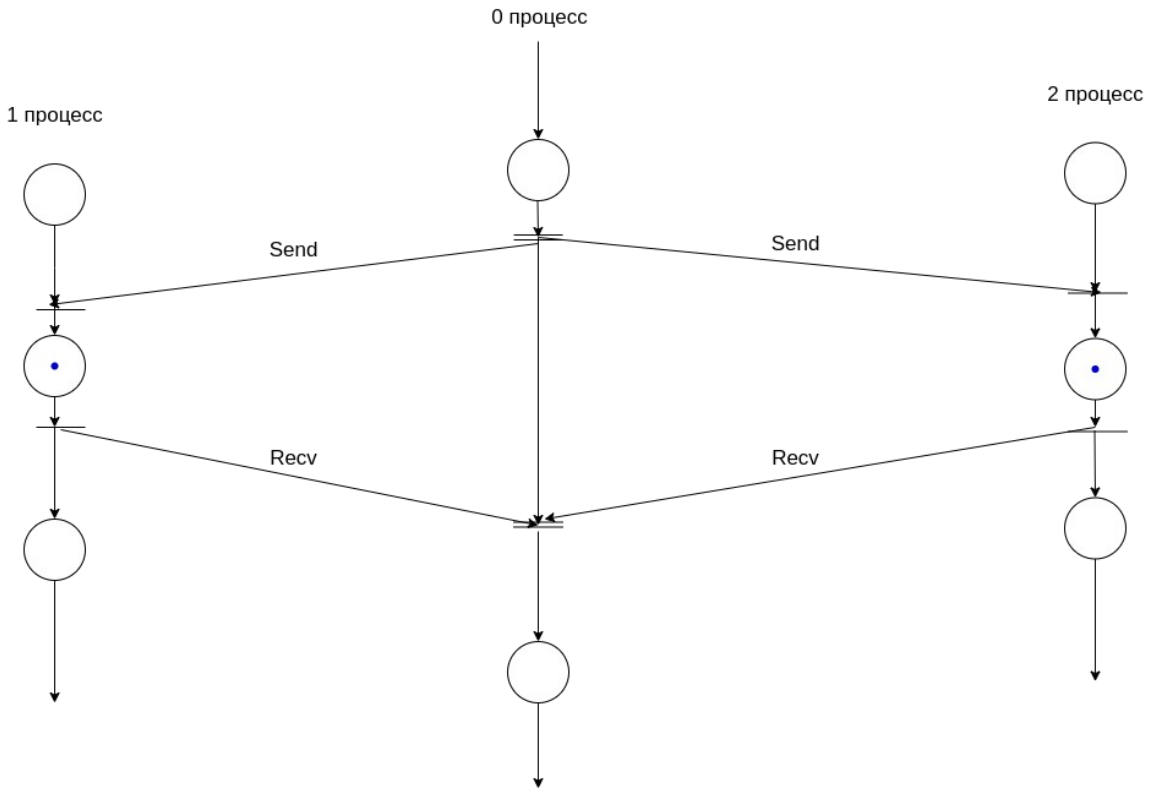
Ненулевые процессы получают данную информацию с помощью MPI_Recv, после этого каждый процесс считает сумму своего подмассива и записывает ее в local_sum. После из ненулевых процессов отправляем результат локальной суммы в нулевой процесс, где собираем локальные суммы и ищем общую. Считаем время работы программы, выводим результат.

Формальное описание алгоритма с использованием аппарата Сетей Петри для 3 процессов (число 3 выбрано для наглядности):

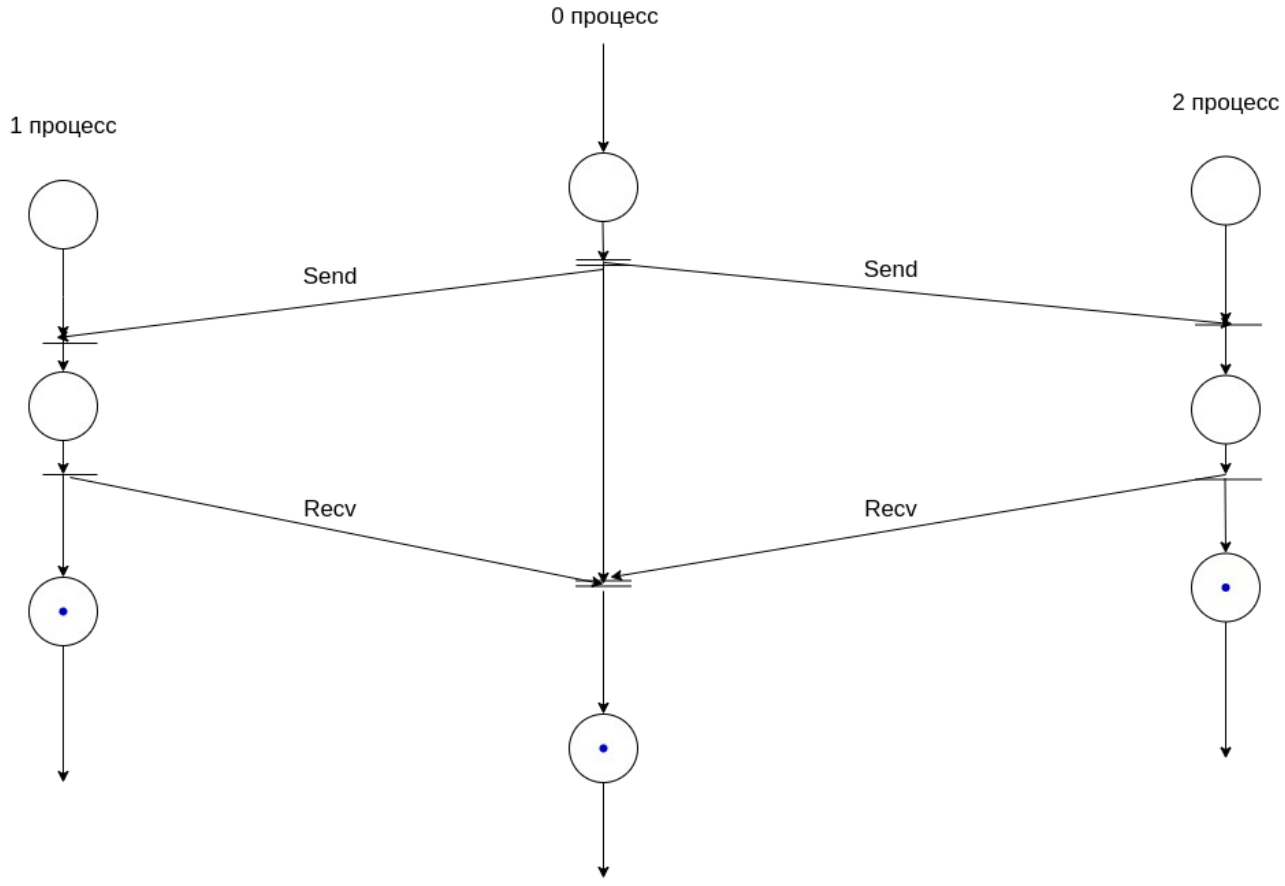
Шаг 1 — формируем массив в процессе 0 и рассылаем его части:



Шаг 1 — обработка массива по частям в ненулевых процессах:



Шаг 2 — отправка результата обработки массива не нулевыми процессами и сборка результата в процессе 0:



Для количества процессов равного n схема выглядит аналогично, только изменяется количество процессов. Создана схема для 3 процессов для наглядности.

Листинг программы.

```
  #include <stdio.h>

#include "stdlib.h"

#include <mpi.h>

#include "time.h"

#include <math.h>


struct{

    int value;

    int proc;

}local_sum,global_sum;


void main( int argc, char *argv[] ) {

    srand(time(NULL));

    int koef = atoi(argv[1]);

    int i, ProcRank, ProcNum, RecvRank;

    MPI_Init(&argc, &argv);

    MPI_Status Status;

    MPI_Comm_rank(MPI_COMM_WORLD, &ProcRank);

    MPI_Comm_size(MPI_COMM_WORLD, &ProcNum);

    i = 1;
```

```
int Size = 0;
for(i = 1; i < ProcNum; i++){
    Size = Size + (1<<i);
}
Size = Size * koef; //размер массива с данными
int array[Size];
int NewSize;
double Start = MPI_Wtime();
if (ProcRank == 0){
        int OrderArr[ProcNum]; //массив для определения процесса для отправки данных 0 процессом
    for(i = 0; i < ProcNum; i++){
        OrderArr[i] = i;
    }
    //для отправки рандомному процессу
    for(int i = 1; i<ProcNum; i++){
        int ind1 = 1 + (rand() % (ProcNum-1));
        int ind2 = 1 + (rand() % (ProcNum-1));
        int temp =  OrderArr[ind1];
        OrderArr[ind1] = OrderArr[ind2];
        OrderArr[ind2] = temp;
    }
    //заполняем массив рандомными числами
    for (i = 0; i < Size; i++){
```

```
            array[i] = rand() % 100;
            printf("%d  ", array[i]);
            if (i == Size - 1)
                printf("\n");
        }
        NewSize = Size/2 + 1;
        int j_NewArr = 0;
        for (i = 1; i < ProcNum; i++){
            int SendArr[NewSize];
            for(int j = 0; j < NewSize; j++){
                SendArr[j] = array[j_NewArr];
                j_NewArr++;
            }
            //передаем другим процессам количество элементов в подмассиве
            MPI_Send(&NewSize, 1, MPI_INT, OrderArr[i], 0, MPI_COMM_WORLD);
            //передаем другим процессам массив по частям
            MPI_Send(&SendArr, Size, MPI_INT, OrderArr[i], 0, MPI_COMM_WORLD);
            NewSize = NewSize/2;
            //принимаем результат работы от процессов
                        MPI_Recv(&local_sum.value,  1,  MPI_INT,  OrderArr[i],  0,
MPI_COMM_WORLD, &Status);
            //обрабатываем полученный результат
            global_sum.value = global_sum.value + local_sum.value;
            double Finish = MPI_Wtime();
```

```
            double Time = Finish-Start;
            if (i == ProcNum - 1){
                printf("Sum = %d\n", global_sum.value);
                printf("Time: %f\n", Time);
            }
        }
    }
    else{
        //принимаем массив от 0 процесса
        MPI_Recv(&NewSize, 1, MPI_INT, 0, 0, MPI_COMM_WORLD, &Status);
        int RecvArr[NewSize];
        MPI_Recv(&RecvArr, Size, MPI_INT, 0, 0, MPI_COMM_WORLD, &Status);
        //ищем сумму подмассива
        local_sum.value = 0;
        for(i=0; i<NewSize; i++){
            local_sum.value = local_sum.value + RecvArr[i];
        }
        //отправляем обработку в нулевой процесс
        MPI_Send(&local_sum.value, 1, MPI_INT, 0, 0, MPI_COMM_WORLD);
    }
    MPI_Finalize();
}
```

Результаты работы программы

1) для 4 процессоров и длине сообщения koef = 1;

```
white-lilac@white-lilac:~/Документы/ParAlg$ mpirun -np 4 ./l2_.x 1
25  43  36  61  40  55  69  34  10  22  67  10  45  49
Sum = 566
Time: 0.000125
```

2) для 4 процессоров и длине сообщения koef = 10;

```
white-lilac@white-lilac:~/Документы/ParAlg$ mpirun -np 4 ./l2_.x 10
23  68  6  80  15  71  14  17  63  82  33  77  75  27  77  41  49  60  77  42  29
90  98  11  8  12  16  18  0  46  37  23  14  43  4  82  15  70  51  30  4  84  8
32  64  37  73  65  97  50  7  78  93  57  89  53  22  57  24  22  4  61  97  70  5
7  53  4  24  24  56  54  80  40  14  12  56  52  38  73  1  40  81  80  33  90  21
  39  12  79  63  86  35  24  84  5  81  89  62  5  65  18  12  46  10  78  58  67
 82  48  40  36  89  73  16  22  16  89  61  80  20  76  67  55  1  3  13  34  44
75  92  10  45  56  56  55  34  66  74  69  15
Sum = 5813
Time: 0.000156
```

3) для 4 процессоров и длине сообщения koef = 100;

```
36  2  9  52  28  69  31  99  43  5  53  3  88  46  71  75  3  87  14  87  65  55
81  53  97  32  41  20  79  49  76  15  3  86  20  84  7  51  83  51  9  37  54  97
  35  25  24  90  64  38  78  29  45  11  82  94  95  75  15  74  76  91  90  80  2
9  62  16  37  13  99  88  22  88  42  72  75  19  48  66  35  87  96  64  84  7  9
8  79  55  26  94  29  2  37  71  34  67
Sum = 60243
Time: 0.000774
```

(На скриншоте изображены не все элементы массива из-за их объема)

4) для 4 процессоров и длине сообщения koef = 500;

```
 94  33  28  81  80  19  56  65  82  9  38  39  51  24  57  95  95  45  87  10  53
 28  15  63  27  55  13  17  4  63  69  50  49  49  31  29  21  39  47  3  49  85
95  52  62  52  48  57  50  35  19  3  15  35  18  42  42  83  60  98  47  81  1  4
8  82  32  29  55  24  76  11  73  14  6  25  76  58  25  33
Sum = 298788
Time: 0.002957
```

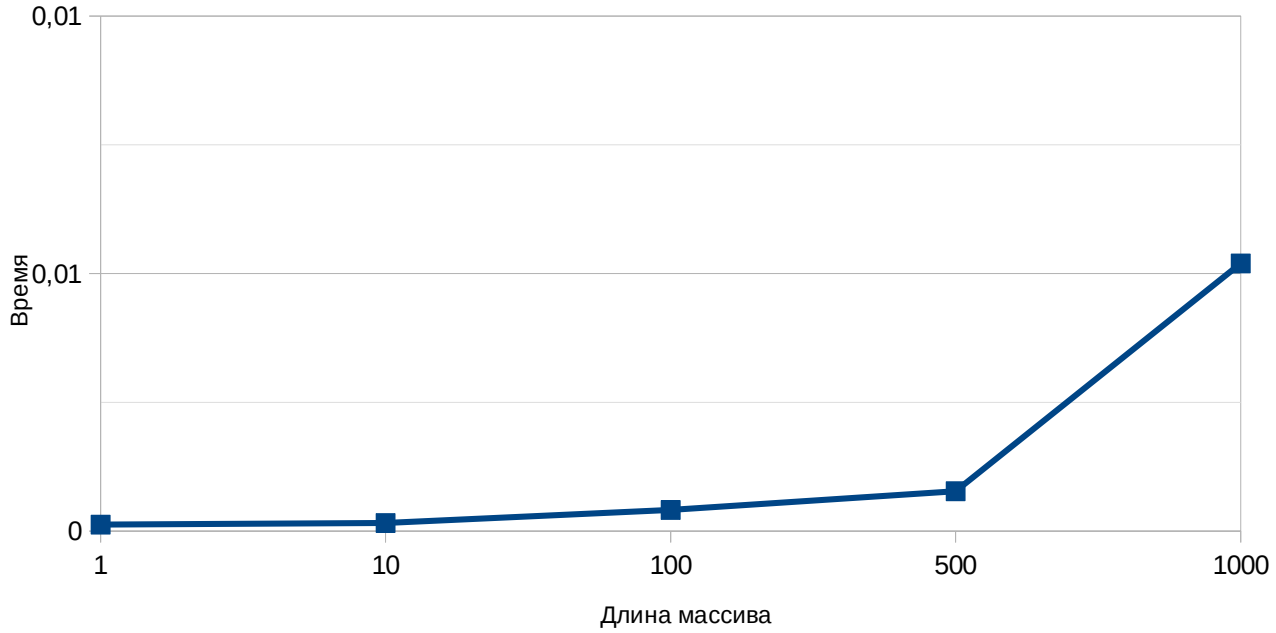
(На скриншоте изображены не все элементы массива из-за их объема)

5) для 4 процессоров и длине сообщения koef = 1000;

```
 68  28   4   6  35  39  97  88  25  75  40  90  57  64  94  12  62  71  97  61  58
45  42  38  52  58   2  48  75  28  82  95  56  38  53  44  77   3  84   2  78  24  44
  87  40  90   0  54  61  49  16  72  94  10  10  98  68  64  98  95  92  80  91   0
 70  44  44  99  47  29  53  77   5  97  65  46  39  17   0   1  66  68  25  12  30   3
5  10  99  99   8  94  43  40  37  43  62  82  40  61  81  21  14  59  26  63  24   7
2   2  93  25  55  59  93  80  71  24  67  33  75  18  93  21  61  33  59  57  95   9
3  97   8  74  18  22  85  44  85  61  69  87  54
Sum = 608772
Time: 0.005196
```

(На скриншоте изображены не все элементы массива из-за их объема)

График зависимости времени выполнения программы от числа процессов равному 4 для разных длин массива.



При увеличении длины сообщения для одинакового количества процессов (равного 4) время выполнения программы возрастает. Из-за масштаба кажется, что при увеличении массива время сильно возрастает, но это происходит из-за неравномерно распределенных длин сообщений в процессе построения графика (так как взят размер массива, увеличивающийся в большое количество раз).

Результаты работы программы

1) для 2 процессоров и длине сообщения koef = 1;

```
white-lilac@white-lilac:~/Документы/ParAlg$ mpirun -np 2 ./l2_.x 1
33  12
Sum = 45
Time: 0.000041
```

2) для 4 процессоров и длине сообщения koef = 1;

```
white-lilac@white-lilac:~/Документы/ParAlg$ mpirun -np 4 ./l2_.x 1
23  72  18  63  49  16  22  43  15  2  88  95  76  49
Sum = 631
Time: 0.000091
```

3) для 6 процессоров и длине сообщения koef = 1;

```
white-lilac@white-lilac:~/Документы/ParAlg$ mpirun -np 6 ./l2_.x 1
29  13  79  26  90  79  83  71  58  20  73  50  42  85  70  98  99  80  35  34  98
 42  36  4  91  66  59  42  49  55  28  79  20  59  5  63  39  40  34  97  61  59
99  3  97  21  53  96  54  89  82  4  31  70  9  74  37  20  68  38  27  96
Sum = 3408
Time: 0.000126
```

4) для 8 процессоров и длине сообщения koef = 1;

```
 15  17  22  52  19  21  82  97  12  19  62  61  97  35  44  31  95  12  95  33  7
 95  65  5  25  67  17  45  88  33  61  4  3  35  56  22  8  38  71  20  10  34  33
  7  21  29  39  17  94  86  50  1  81  67  6  7  35  76  4  75  61  17  79  64  52
  36  87  12  26  10  85  88  96  18  96  18  0  87  35  94  73  37  47  6  4  53  6
 5  91  29  70  67  43  87  46  7  92  34  46  4  13  57  89  1  53
 Sum = 11481
 Time: 0.000471
```

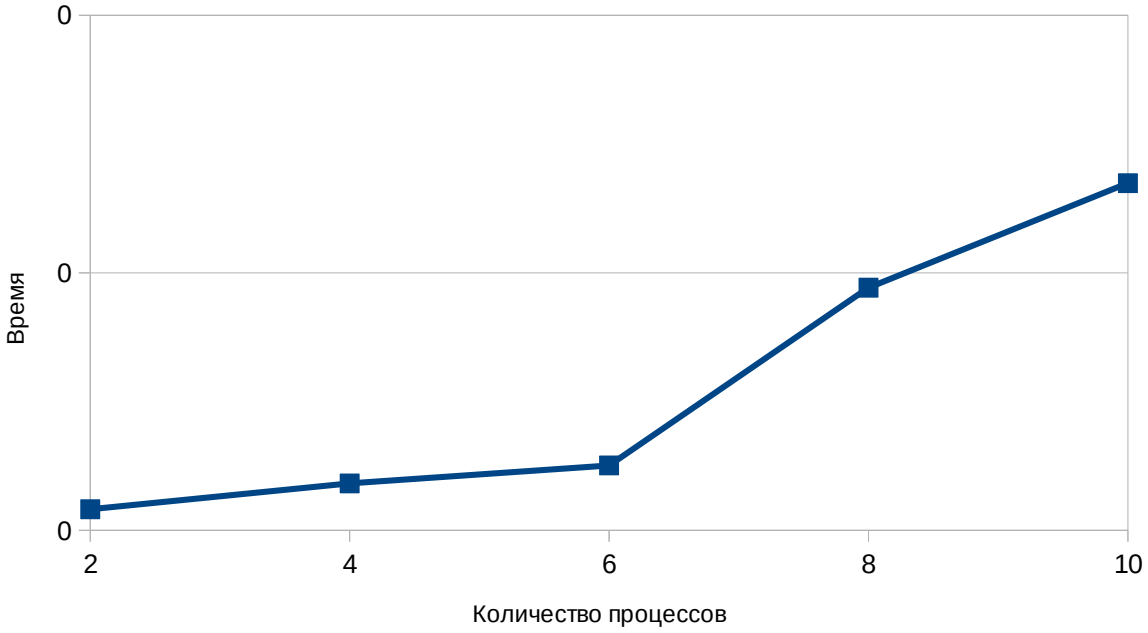
(На скриншоте изображены не все элементы массива из-за их объема)

5) для 10 процессоров и длине сообщения koef = 1;

```
 91  96  54  95  68  17  40  41  95  25  66  48  89  70  7  60  7  62  66  29  41
43  76  99  29  44  68  56  60  73  5  52  21  11  99  90  28  92  31  75  17  49
23  6  19  83  66  26  45  32  7  38  28  83  37  9  79  58  17  40  83  22  44  4
 85  43  94  13  87  25  41  5  74  64  63  45  47  82  23  45  66  30  83  94  66
 73  55  97  83  73  37  66  47  33  70  33  29  65  98  16  42  39  21  69  4  37
 14  3  19
Sum = 49165
Time: 0.000674
```

(На скриншоте изображены не все элементы массива из-за их объема)

График ускорения (эффективности):



При увеличении количества процессов время незначительно меняется (на тысячные миллисекунд), это происходит из-за того, что нулевой процесс отправляет сгенерированный массив, а остальные процессы его принимают, а затем отправляют назад. На это тратится дополнительное время, а также это происходит из-за того, что размер массива зависит от количества процессов, так как по условию необходимо, чтобы каждый процесс принимал подмассив в 2 раза меньше предыдущего, следовательно с увеличением числа процессов увеличивается и исходный массив, и поэтому график возрастает.

Вывод.

Была написана программа нахождения результата, используя раздачу и сборку массива. а также проанализирована работа написанной многопроцессорной программы.