

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

**ОТЧЕТ**  
**по лабораторной работе №1**  
**по дисциплине «ПА»**  
**Тема: ИСПОЛЬЗОВАНИЕ ФУНКЦИЙ ОБМЕНА ДАННЫМИ «ТОЧКА**  
**- ТОЧКА» В БИБЛИОТЕКЕ MPI**

Студент гр. 8304

\_\_\_\_\_

Сергеев А.Д.

Преподаватель

\_\_\_\_\_

Татаринов Ю.С.

Санкт-Петербург

2020

### **Задание.**

Процесс 0 генерирует массив и раздает его другим процессам для вычисления локальных сумм, после чего вычисляет общую сумму.

### **Описание алгоритма.**

Процесс с рангом 0 создаёт массив чисел с плавающей точкой, делит его на равные части между остальными процессами и рассылает им указатели на эти части указав в поле тэг свой ранг. После этого он создаёт массив локальных сумм и ожидает окончания работы остальных процессов для его заполнения, при чем, как только очередной процесс завершается, вычисленная им сумма заносится в элемент массива, номер которого равен рангу этого процесса. Сумма элементов массива локальных сумм и будет общей суммой элементов массива.

Каждый из процессов с рангом, отличным от нуля, создаёт массив, длина которого равна частному количества элементов массива, созданного нулевым процессом, и количеству ненулевых процессов. После этого он ожидает сообщение от нулевого процесса, заполняющего массив, вычисляет сумму его элементов и посылает сумму нулевому процессу, указывая в поле тэг свой ранг.

### **Листинг программы.**

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <mpi.h>

#define array_size 10000000
#define array_upper_limit 1000.0

void populate_array(double* arr, unsigned int size) {
    srand(time(NULL));
    printf("Array (length %d): [ ", size);
    for (unsigned int i = 0; i < size; i++) {
        arr[i] = (array_upper_limit / RAND_MAX) * rand();
        //printf("%lf ", arr[i]);
    }
    printf("]\n");
}
```

```

double sum_array(const double* arr, unsigned int size) {
    if (size == 0) return -1;
    double max = arr[0];
    for (unsigned int i = 0; i < size; i++) if (arr[i] > max) max = arr[i];
    return max;
}

int main(int argc, char* argv[]) {
    int ProcNum, ProcRank;
    MPI_Status Status;

    MPI_Init(&argc, &argv);
    MPI_Comm_size(MPI_COMM_WORLD, &ProcNum);
    MPI_Comm_rank(MPI_COMM_WORLD, &ProcRank);

    int work_nodes = ProcNum - 1;
    int node_quote = array_size / work_nodes;

    if (ProcRank == 0) {
        double start_time = MPI_Wtime();
        printf ("Root process started at %lf\n", start_time);

        double* array = (double*) malloc(array_size * sizeof(double));
        populate_array(array, array_size);

        for (int i = 1; i < ProcNum; i++)
            MPI_Send(array + (i - 1) * node_quote, node_quote, MPI_DOUBLE, i,
                ProcRank, MPI_COMM_WORLD);

        double* local_sums = (double*) malloc(work_nodes * sizeof(double));
        printf ("Local maxes received (one of each %d elements): [ ",
            node_quote);
        for (int i = 1; i < ProcNum; i++) {
            MPI_Recv(&(local_sums[i - 1]), 1, MPI_DOUBLE, MPI_ANY_SOURCE, i,
                MPI_COMM_WORLD, &Status);
            printf("%lf ", local_sums[i - 1]);
        }
        printf("]\n");

        free(array);
        double max = sum_array(local_sums, ProcNum);
        free(local_sums);
        printf("Global array max: %lf\n", max);

        double end_time = MPI_Wtime();
        printf("Root process ended at %lf\n", end_time);
        printf("All operations took %lf\n", end_time - start_time);

    } else {
        double* arr = (double*) malloc(node_quote * sizeof(double));

```

```

MPI_Recv(arr, node_quote, MPI_DOUBLE, MPI_ANY_SOURCE, 0, MPI_COMM_WORLD,
&Status);
double local_sum = sum_array(arr, node_quote);
MPI_Send(&local_sum, 1, MPI_DOUBLE, 0, ProcRank, MPI_COMM_WORLD);
free(arr);
}

MPI_Finalize();
return 0;
}

```

## Результаты работы.

```

/bin/bash /home/milty/Documents/current/sem5/PA-lab0/build.sh 51 ./lab1.c
milty@mirrmere:~/Documents/current/sem5/PA-lab0$ /bin/bash /home/milty/Documents/current/sem5/PA-lab0/build.sh 51 ./lab1.c
Root process started at 1602740672.209395
Array (length 100): [ 216.337278 635.137322 966.439932 47.742480 948.543339 767.826284 449.995995 793.777950 934.027501 450.673355 572.797662 630.779304 94.472318
893.323129 219.716828 478.823285 474.551742 670.149138 575.750907 684.628025 146.165267 59.158342 186.908554 249.227271 392.700397 473.938264 135.248938 317.5082
89 796.818831 972.490780 390.404341 13.156109 607.628102 356.844273 60.898589 556.171441 124.670476 510.894583 349.949390 58.697977 969.567939 922.747052 689.4772
81 64.040257 816.070180 909.194109 542.863542 290.621922 579.343247 118.614449 975.249948 725.508514 177.772791 162.158501 974.735785 570.473187 636.096765 109.98
4723 887.981477 432.915595 82.475502 278.385818 446.071704 690.103604 635.230091 506.970293 246.275045 759.900567 17.864876 596.224435 818.598545 987.432815 518.9
71487 508.075826 51.473071 335.041667 417.269935 594.336614 625.663589 996.613182 712.951063 600.913536 722.121696 890.723854 763.072037 696.857481 461.197041 399
.168801 806.842203 349.178517 832.084397 889.317705 627.564335 278.156100 579.421309 262.794426 785.126394 825.696354 22.694993 802.991270 ]
Local maxes received (one of each 2 elements): [ 635.137322 966.439932 948.543339 793.777950 934.027501 630.779304 893.323129 478.823285 670.149138 684.628025 146
.165267 249.227271 473.938264 317.508289 972.490780 390.404341 607.628102 556.171441 510.894583 349.949390 969.567939 689.477281 909.194109 542.863542 579.343247
975.249948 177.772791 974.735785 636.096765 887.981477 278.385818 690.103604 635.230091 759.900567 596.224435 987.432815 518.971487 335.041667 594.336614 996.6131
82 712.951063 890.723854 763.072037 461.197041 806.842203 889.317705 627.564335 579.421309 825.696354 802.991270 ]
Global array max: 996.613182
Root process ended at 1602740672.339236
All operations took 0.129841

```

Рисунок 1 — Результат запуска программы для 51 процесса

```

Root process started at 1602740760.549416
Array (length 100): [ 754.260567 986.864691 99.259561 217.750221 644.397331 200.241739 220.289154 757.533989 317.631812 910.413518 908.057931 878.778605 910.67746
9 452.401529 482.811457 428.296061 460.521086 428.500550 84.751703 933.425517 517.851824 841.076650 544.134045 412.871514 110.411067 165.329937 100.914699 537.856
891 454.457166 97.843665 119.087866 208.717732 84.708355 218.347428 426.467954 729.105686 418.589166 646.757108 486.639675 736.220979 557.170625 394.697006 614.99
9583 467.848093 847.099135 97.811040 896.144154 307.620220 526.311590 980.895937 241.045736 44.162614 821.972587 785.179782 457.034128 932.383654 950.509719 557.9
48827 478.240544 404.966884 655.792492 589.328410 613.684617 740.500847 807.675837 40.152571 469.606532 226.265004 686.909678 956.246207 962.485983 244.080302 350
.943813 577.405565 711.928395 198.042947 675.296605 688.072550 505.663167 201.608195 588.968407 746.708904 245.770808 410.941073 531.888685 702.804936 343.324727
482.398404 260.753763 813.565271 887.365289 916.546255 402.893681 501.049905 657.047181 210.569518 541.202476 126.653633 436.834521 228.112154 ]
Local maxes received (one of each 10 elements): [ 986.864691 933.425517 841.076650 736.220979 980.895937 950.509719 956.246207 962.485983 813.565271 916.546255 ]
Global array max: 986.864691
Root process ended at 1602740760.599274
All operations took 0.049858

```

Рисунок 2 — Результат запуска программы для 11 процессов

```

Root process started at 1602740798.718863
Array (length 100): [ 878.225101 688.532942 107.890279 388.712392 643.100370 600.958199 947.546254 975.066298 437.440578 638.462384 775.731428 44.040567 673.51877
3 929.440090 679.285156 910.685190 873.596032 988.787744 448.613070 98.492187 288.691222 929.235633 270.071240 539.923824 368.987315 902.580693 600.292293 7.98585
3 499.557079 556.292229 576.996479 377.782100 244.825170 684.886758 766.494572 807.925540 285.844956 714.040026 862.991838 723.285534 352.503210 638.723266 767.32
6181 26.021982 568.163356 446.611256 936.707173 441.759387 435.399000 385.320242 540.251574 724.090223 314.555875 810.322814 264.014046 683.543190 712.983506 864.
306339 691.529043 212.460586 420.598568 268.525521 590.242765 665.423738 953.412279 356.737337 553.349278 239.257235 70.778162 416.341115 962.542769 423.281372 55
.064380 729.868869 449.303354 623.227736 176.480125 386.010527 64.987122 611.879125 771.330769 605.238696 335.969347 85.886643 415.561510 599.983393 769.429833 12
8.465016 464.289733 460.958875 340.925601 884.888301 729.484397 931.168367 550.312039 682.896676 287.905703 103.661316 922.153911 358.683866 ]
Local maxes received (one of each 50 elements): [ 988.787744 962.542769 ]
Global array max: 988.787744
Root process ended at 1602740798.721502
All operations took 0.002639

```

Рисунок 3 — Результат запуска программы для 3 процессов

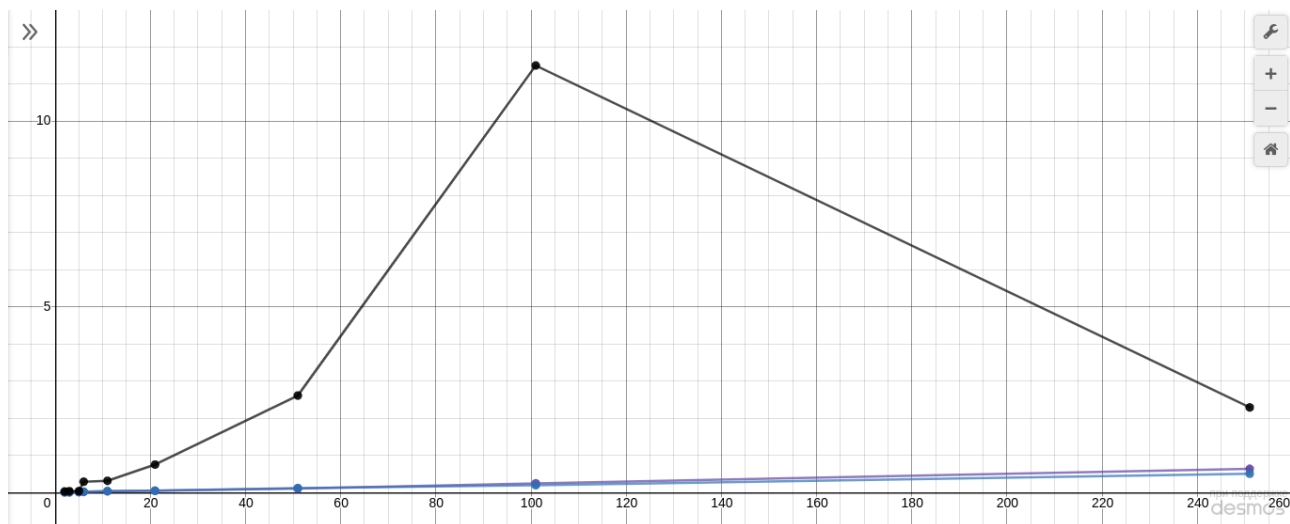


Рисунок 4 — Графики зависимости времени работы от количества процессов

Синий график — массив из 100 элементов.

Фиолетовый график — массив из 10000 элементов.

Черный график — массив из 1000000 элементов.

### Выводы.

Была написана, в которой использовались параллельные вычисления MPI. Различия во времени выполнения программы для разного количества процессов обусловлены тем, что задача поиска суммы элементов массива сама по себе выполняется гораздо быстрее, чем передача сообщений между узлами. Разница во времени выполнения программы между случаями с одинаковым количеством процессов, но разными размерами массива всё же демонстрируют зависимость времени выполнения от количества элементов массива. Пик, наблюдаемый при выполнении на 100 процессах, по моему мнению, появляется из-за различных вариантов оптимизации передачи сообщений используемых для разных количеств процессов.