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

ОТЧЕТ

по лабораторной работе №2

по дисциплине «ПА»

Тема: ИСПОЛЬЗОВАНИЕ АРГУМЕНТОВ - ДЖОКЕРОВ.

Студент гр. 8304 _____ Сергеев А.Д.

Преподаватель _____ Татаринов Ю.С.

Санкт-Петербург

2020

Задание.

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.

Описание алгоритма.

Процесс 0 генерирует целочисленный массив, число элементов которого равно двум в степени, равной количеству ненулевых процессов минус один. После этого каждому процессу с рангом, не равным нулю, в случайном порядке нулевой процессор посылает два сообщения: в первом — число элементов, которые этому процессу необходимо будет обработать, а во втором - вторую половину ещё необработанной части массива. Вместе с этим в специальном массиве сохраняется соответствие ранга массива количеству элементов, которые он должен вернуть. Затем процесс 0 ожидает возвращения обработанных частей массива и записывает в массив полученные части в изначальном порядке.

Процесс, ранг которого не равен нулю, ожидает сообщения, в котором будет указана длина обрабатываемой им части массива, а затем создаёт целочисленный массив такой длины. После этого он ожидает ещё одного сообщения от нулевого процесса, содержащего данные для заполнения этого массива. После получения этого сообщения и обработки массива, процесс возвращает обработанную часть в нулевой процесс.

Листинг программы.

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

```
#define premier 1000000933
#define array_upper_limit 1000

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

void mod_array(unsigned int* arr, unsigned int size) {
for (unsigned int i = 0; i < size; i++) arr[i]--;
}

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;

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

unsigned int array_size = (1u « (unsigned int) work_nodes) - 1u;
printf("Calculated array size : %d\n", array_size);

unsigned int* array = (unsigned int*) malloc(array_size * sizeof(unsigned
int));
populate_array(array, array_size);

unsigned int* local_sizes = (unsigned int*) malloc(work_nodes *
sizeof(unsigned int));
unsigned int passing_elements = array_size / 2 + 1;
unsigned int adv = premier % work_nodes;
for (int i = 1; i < ProcNum; i++) {
MPI_Send(&passing_elements, 1, MPI_UNSIGNED, i, ProcRank,
MPI_COMM_WORLD);
MPI_Send(array + passing_elements - 1, passing_elements, MPI_UNSIGNED, i,
ProcRank, MPI_COMM_WORLD);
local_sizes[i - 1] = passing_elements;
passing_elements /= 2;
adv = (adv + premier) % work_nodes;
}
```

```
printf("]\n");
for (int i = 1; i < ProcNum; i++)
MPI_Recv(&(array[local_sizes[i - 1] - 1]), local_sizes[i - 1],
MPI_UNSIGNED, MPI_ANY_SOURCE, i, MPI_COMM_WORLD, &Status);

free(local_sizes);

printf("Array (length %d): [ ", array_size);
for (unsigned int i = 0; i < array_size; i++) printf("%d ", array[i]);
printf("]\n");

free(array);

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 {
unsigned int local_size;
MPI_Recv(&local_size, 1, MPI_UNSIGNED, MPI_ANY_SOURCE, 0, MPI_COMM_WORLD,
&Status);
unsigned int* arr = (unsigned int*) malloc(local_size * sizeof(unsigned
int));
MPI_Recv(arr, local_size, MPI_UNSIGNED, MPI_ANY_SOURCE, 0,
MPI_COMM_WORLD, &Status);
mod_array(arr, local_size);
MPI_Send(arr, local_size, MPI_UNSIGNED, 0, ProcRank, MPI_COMM_WORLD);
free(arr);
}

MPI_Finalize();
return 0;
}
```

Результаты работы.

```
Root process started at 1602745379.877186
Calculated array size : 1
Array (length 1): [ 834 ]
Array (length 1): [ 833 ]
Root process ended at 1602745379.877240
All operations took 0.000055
```

Рисунок 1 — Результат работы программы для 2 узлов

```
Root process started at 1602745426.244155
Calculated array size : 7
Array (length 7): [ 250 563 796 555 883 194 664 ]
Array (length 7): [ 249 562 795 554 882 193 663 ]
Root process ended at 1602745426.244314
All operations took 0.000159
```

Рисунок 2 — Результат работы программы для 4 узлов

```
Root process started at 1602745450.904813
Calculated array size : 63
Array (length 63): [ 20 795 211 589 166 182 256 357 5 383 5 640 572 590 604 502 299 681 630 649 144 166 805 224 972 653 934 373 586 641 67 606 788 630 548 954 165
 804 663 170 540 20 810 464 611 767 966 262 448 596 912 944 115 717 168 87 723 103 460 389 96 879 268 ]
Array (length 63): [ 19 794 210 588 165 181 255 356 4 382 4 639 571 589 603 501 298 680 629 648 143 165 804 223 971 652 933 372 585 640 66 605 787 629 547 953 164
 803 662 169 539 19 809 463 610 766 965 261 447 595 911 943 114 716 167 86 722 102 459 308 95 878 267 ]
Root process ended at 1602745450.917019
All operations took 0.012206
```

Рисунок 3 — Результат работы программы для 7 узлов

```
Root process started at 1602745479.745813
Calculated array size : 255
Array (length 255): [ 893 117 79 705 868 819 906 166 432 880 51 947 692 212 212 507 43 610 870 724 302 702 636 109 69 587 701 325 170 445 598 415 914 677 120 134
848 379 300 280 611 704 579 303 268 143 162 311 754 32 387 56 734 375 517 804 314 570 481 484 15 79 251 281 756 372 767 956 751 420 236 714 124 167 18 744 311 180
 55 65 565 794 473 299 521 343 103 836 913 937 672 281 16 924 562 125 648 682 81 399 102 670 465 226 837 483 970 148 16 377 565 581 523 391 880 396 734 336 232 99
9 273 257 280 641 533 195 766 181 877 200 932 979 222 397 557 59 233 879 560 249 256 125 182 131 868 62 527 954 750 112 954 375 721 586 369 254 781 487 435 10 687
 719 989 909 116 898 321 701 777 881 950 385 358 132 516 579 547 396 533 649 508 839 377 581 426 746 835 559 233 622 570 273 341 559 534 809 810 855 511 587 88 81
3 973 799 298 841 378 845 237 911 846 97 103 223 30 529 321 865 88 907 487 658 180 180 570 66 342 380 922 853 319 362 18 292 161 316 486 539 513 75 803 360 525 25
8 935 555 787 257 421 227 164 260 238 696 793 ]
Array (length 255): [ 892 116 78 704 867 818 905 165 431 879 50 946 691 211 211 506 42 609 869 723 301 701 635 108 68 586 700 324 169 444 597 414 913 676 119 133
847 378 299 279 610 703 578 302 267 142 161 310 753 31 386 55 733 374 516 803 313 569 480 483 14 78 250 280 755 371 766 955 750 419 235 713 123 166 17 743 310 179
 54 64 564 793 472 298 520 342 102 835 912 936 671 280 15 923 561 124 647 681 80 398 101 669 464 225 836 482 969 147 15 376 564 580 522 390 879 395 733 335 231 99
8 272 256 279 640 532 194 765 180 876 199 931 978 221 396 556 58 232 878 559 248 255 124 181 130 867 61 526 953 749 111 953 374 720 585 368 253 780 486 434 9 686
718 988 908 115 897 320 700 776 880 949 384 357 131 515 578 546 395 532 648 507 838 376 580 425 745 834 558 232 621 569 272 340 558 533 808 809 854 510 586 87 812
 972 798 297 840 377 844 236 910 845 96 102 222 29 528 320 864 87 906 486 657 179 179 569 65 341 379 921 852 318 361 17 291 160 315 485 538 512 74 802 359 524 257
 934 554 786 256 420 226 163 259 237 695 792 ]
Root process ended at 1602745479.786117
All operations took 0.040303
```

Рисунок 3 — Результат работы программы для 9 узлов

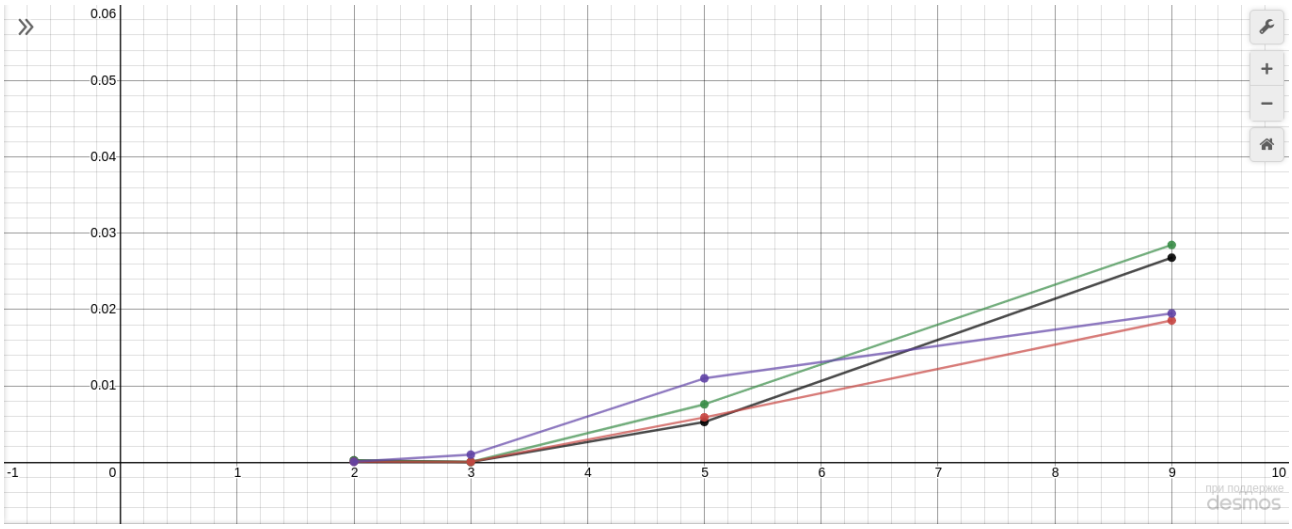

Рисунок 4 — Графики зависимости времени работы от количества процессов

Красный график — количество элементов массива вычислено по описанному алгоритму.

Фиолетовый график — количество элементов массива увеличено вдвое.

Черный график — количество элементов массива увеличено в 32 раза.

Зеленый график — количество элементов массива увеличено в 128 раз.

Выводы.

Создана программа, распределяющая данные между процессами неравномерно, но с сохранением их порядка. Увеличение количества времени, требующегося на обработку массива при увеличении количества процессов связано с тем, что операция обработки массива всё ещё занимает гораздо меньше времени, чем пересылка большого количества данных между процессами. Тем не менее зависимость времени выполнения от количества элементов массива сохраняется, несмотря на то, что размер массива влияет на время работы меньше, чем количество узлов.