

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

ОТЧЕТ
по лабораторной работе №1
по дисциплине «Параллельные алгоритмы»
Тема: Использование функций обмена данными «точка-точка» в
библиотеке MPI.

Студент гр. 9383

Лапина А.А.

Преподаватель

Татаринов Ю.С.

Санкт-Петербург

2021

Цель работы.

Написать программу для поиска глобального максимума, собрав при этом локальные максимумы.

Формулировка задания.

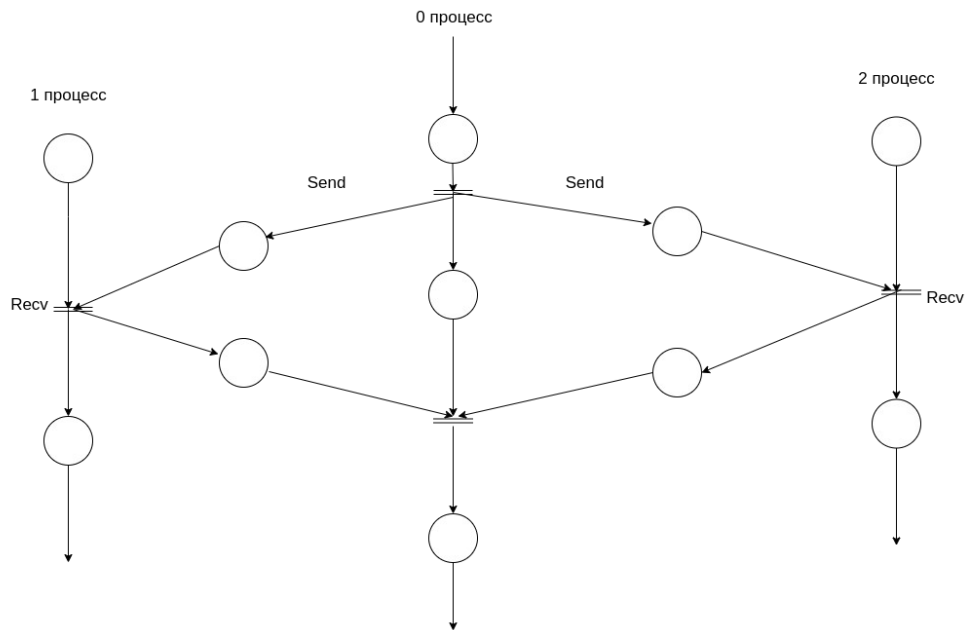
Вариант 4.

Процесс 0 генерирует массив и раздает его другим процессам для поиска максимума. Собрав локальные максимумы, ищет общий.

Краткое описание выбранного алгоритма решения задачи.

Создаем структуру для хранения локального и глобального максимума. Для генерации случайных чисел в интервале от 0 до 100 необходимо прописать `srand(time(NULL))` (Генерировать числа можно не только до ста, для этого достаточно убрать `%100` при использовании функции `rand()`, число 100 выбрано для удобства тестирования программы). Переменная `Size` задает размер массива. Для анализа работы программы измерим время перед ее выполнением, результат запишем в `Start`. Процесс 0 генерирует массив и отправляет его другим процессам с помощью `MPI_Send`, другие процессы принимают созданный массив и затем, происходит поиск локального максимума, после чего с помощью `MPI_Reduce` ищем глобальный максимум (используется `MPI_MAXLOC`). Измеряем время завершения работы программы, результат запишем в `Finish`.

Формальное описание алгоритма с использованием аппарата Сетей Петри для 3 процессов:



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"

struct{

    int value;

    int proc;

}local_max,global_max;
```

```

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

    srand(time(NULL));

    int i, ProcRank, ProcNum, RecvRank;

    int Size = 10;

    MPI_Init(&argc, &argv);

    MPI_Status Status;

    MPI_Comm_rank(MPI_COMM_WORLD, &ProcRank);

    MPI_Comm_size(MPI_COMM_WORLD, &ProcNum);

    int array[Size];

    int max_value;

    double Start = MPI_Wtime();

    if (ProcRank == 0){

        for (i = 0; i < Size; i++){

            array[i] = rand() % 100;

            printf("%d ", array[i]);

            if (i == Size - 1)

                printf("\n");

        }

        for (i = 1; i < ProcNum; i++){

            MPI_Send(&array, Size, MPI_INT, i, 0, MPI_COMM_WORLD);

        }

    }

    else{

```

```

MPI_Recv(&array, Size, MPI_INT, 0, 0, MPI_COMM_WORLD, &Status);

max_value=array[0];

for(i=0; i<Size; i++){

    if(array[i]>max_value)

        max_value=array[i];

}

local_max.value = max_value;

}

MPI_Reduce(&local_max, &global_max, 1, MPI_2INT, MPI_MAXLOC, ProcNum -
1, MPI_COMM_WORLD );

double Finish = MPI_Wtime();

double Time = Finish-Start;

if (ProcRank == ProcNum - 1){

    printf("Global maximum = %d\n", global_max.value);

    printf("Time: %f\n", Time);

}

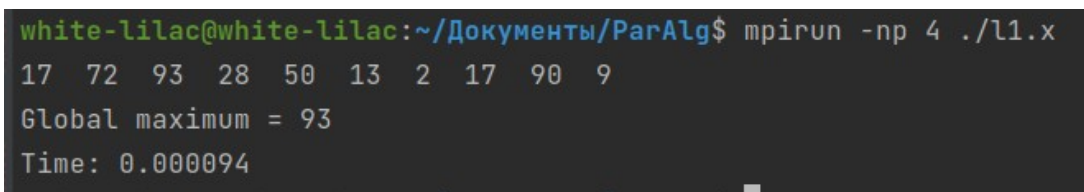
MPI_Finalize();

}

```

Результаты работы программы

1) для 4 процессоров и длине сообщения Size = 10;



```

white-lilac@white-lilac:~/Документы/ParAlg$ mpirun -np 4 ./l1.x
17 72 93 28 50 13 2 17 90 9
Global maximum = 93
Time: 0.000094

```

2) для 4 процессоров и длине сообщения Size = 100;

```
white-lilac@white-lilac:~/Документы/ParAlg$ mpirun -np 4 ./l1.x
71 71 37 56 30 35 30 27 60 52 2 11 71 95 65 48 30 3 91 45 38 0 47 71 60 3
9 21 45 24 65 90 47 36 80 56 18 67 86 45 79 90 0 90 61 95 8 61 25 11 52 2
2 49 5 69 72 65 61 93 62 85 58 5 85 46 37 41 64 4 27 62 84 69 62 26 30 9
34 91 86 45 95 9 46 52 30 70 69 91 64 32 29 74 89 14 73 26 7 89 82 86
Global maximum = 95
Time: 0.000260
```

3) для 4 процессоров и длине сообщения Size = 1000;

```
17 70 72 11 11 11 1 47 48 37 87 37 17 34 84 37 70 13 31 83 13 7 38 45 3
32 26 73 78 91 89 97 33 81 60 44 45 13 46 91 52 35 0 24 41 36 35 38 4 86
53 17 45 9 14 51 94 40 76 72 83 65 21 16 99 33 13 96 46 59 87 51 46 40 27
87 76 62 77 80 1 31 97 98 92 11 1 86 3 77 58 86 95 31 55 46 64 68 42 63
79 81 14 77 21 93 16 98 7 94 30 60 77 80 59 69 43 60 56 47 90 66 85 85 50
40 83 66 60 25 29 91 6 95 68 28 88 85 78 48 31 60 8 8 92 67 77 36 80 85
35
Global maximum = 99
Time: 0.000411
```

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

4) для 4 процессоров и длине сообщения Size = 10000;

```
2 54 87 73 10 68 80 37 59 25 87 76 59 85 56 40 12 16 84 38 86 58 83 29 4
2 70 96 12 38 1 79 92 55 66 17 17 87 97 6 46 23 94 22 82 31 30 22 43 46 5
8 33 84 69 16 65 63 38 61 76 28 62 55 20 69 74 37 39 61 35 45 59 10 91 33
44 22 63 66 17 61 77 2 46 46 70 11 9 8 73 85 88 87 93 8 9 19 98 48
Global maximum = 99
Time: 0.006247
```

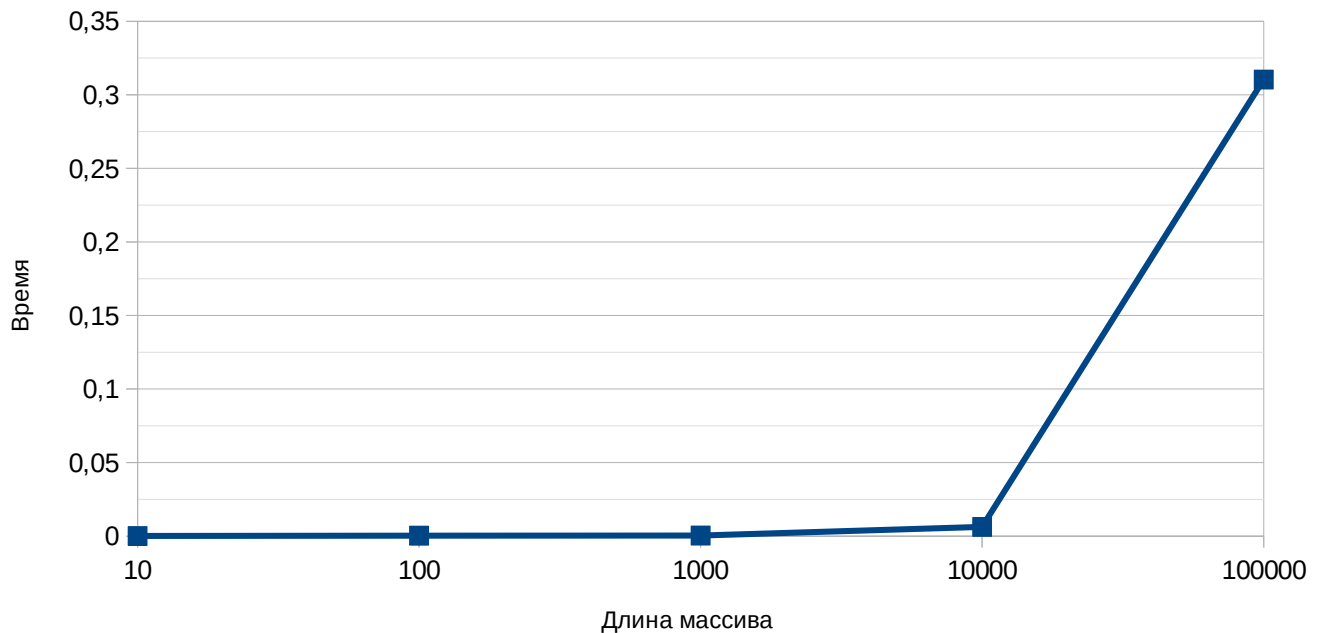
(На скриншоте изображены не все элементы массива из-за большого объема).

5) для 4 процессоров и длине сообщения Size = 100000;

```
48 85 53 4 34 72 80 97 46 70 48 59 33 46 4 5 96 63 34 1 25 60 89 52 29
58 26 8 84 55 95 84 92 0 89 78 72 21 76 70 91 76 82 25 75 38 82 23 1 68 2
5 27 29 14 31 58 24 10 18 9 17 65 93 62 65 34 40 38 55 68 60 47 97 42 24
72 80 58 95 34 26 20 13 7 87 44 66 11 6
Global maximum = 99
Time: 0.310337
```

(На скриншоте изображены не все элементы массива из-за большого 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 10 раз).

Результаты работы программы

1) для 2 процессоров и длине сообщения Size = 30;

```
white-lilac@white-lilac:~/Документы/ParAlg$ mpirun -np 2 ./l1.x
7 54 57 66 52 62 84 1 35 5 28 54 48 65 40 83 71
46 86 63 24 25 0 23 29 64 54 72 9 32
Global maximum = 86
Time: 0.000148
```

2) для 4 процессоров и длине сообщения Size = 30;

```
white-lilac@white-lilac:~/Документы/ParAlg$ mpirun -np 4 ./l1.x
33 14 70 33 3 2 63 5 66 54 76 28 0 64 88 3 72 8
6 25 46 6 26 47 69 43 1 94 72 51 3
Global maximum = 94
Time: 0.000336
```

3) для 6 процессоров и длине сообщения Size = 30;

```
white-lilac@white-lilac:~/Документы/ParAlg$ mpirun -np 6 ./l1.x
25 29 48 85 17 85 86 41 23 65 4 98 6 57 48 15 28
69 5 4 24 22 81 23 53 64 89 47 46 26
Global maximum = 98
Time: 0.000452
```

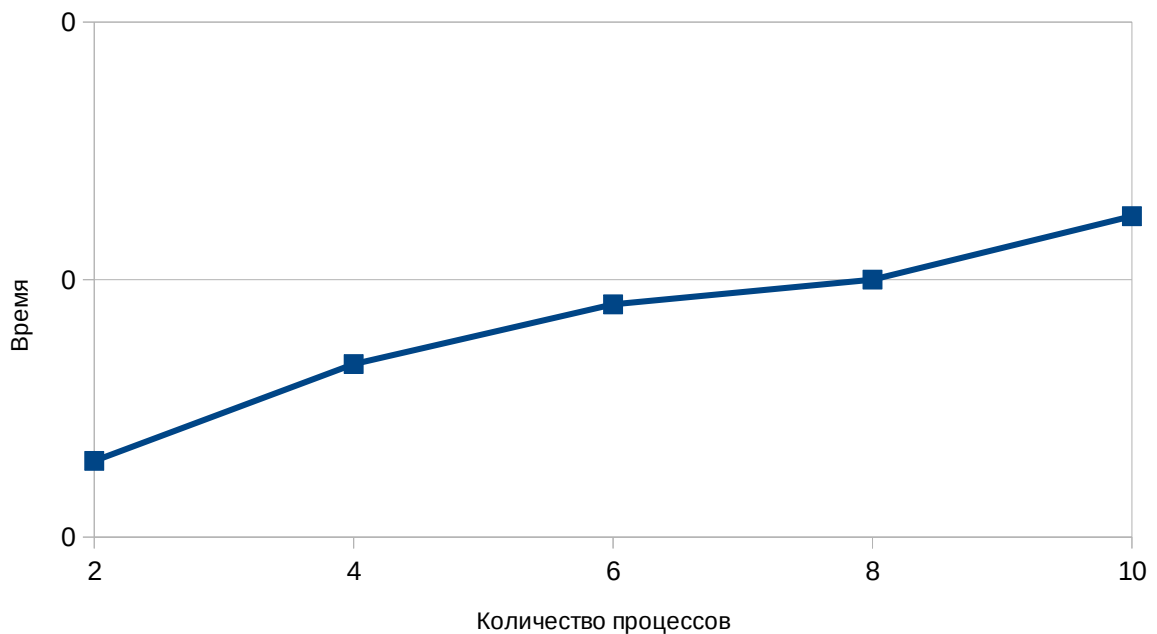
4) для 8 процессоров и длине сообщения Size = 30;

```
white-lilac@white-lilac:~/Документы/ParAlg$ mpirun -np 8 ./l1.x
79 31 7 3 18 80 29 34 88 21 40 75 98 90 23 82 36
71 28 32 17 63 68 69 63 95 72 78 40 50
Global maximum = 98
Time: 0.000500
```

5) для 10 процессоров и длине сообщения Size = 30;

```
white-lilac@white-lilac:~/Документы/ParAlg$ mpirun -np 10 ./l1.x
3 59 1 6 7 82 33 0 16 73 74 84 17 86 94 49 85 7
9 4 57 23 9 39 3 47 97 86 36 22 15
Global maximum = 97
Time: 0.000623
```

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. На это тратится дополнительное время, поэтому график возрастает.

Вывод.

Была написана программа поиска глобального максимума, собрав при этом локальные максимумы, а также проанализированна работа написанной многопроцессорной программы.