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

ОТЧЕТ

по лабораторной работе №4
по дисциплине «Параллельные алгоритмы»
Тема: Группы процессов и коммутаторы

Студент гр. 9381 _____ Колованов Р.А.

Преподаватель _____ Татаринов Ю.С.

Санкт-Петербург
2021

Цель работы.

Написание программы, использующую группы процессов и коммутаторы библиотеки MPI.

Формулировка задания.

Вариант 11.

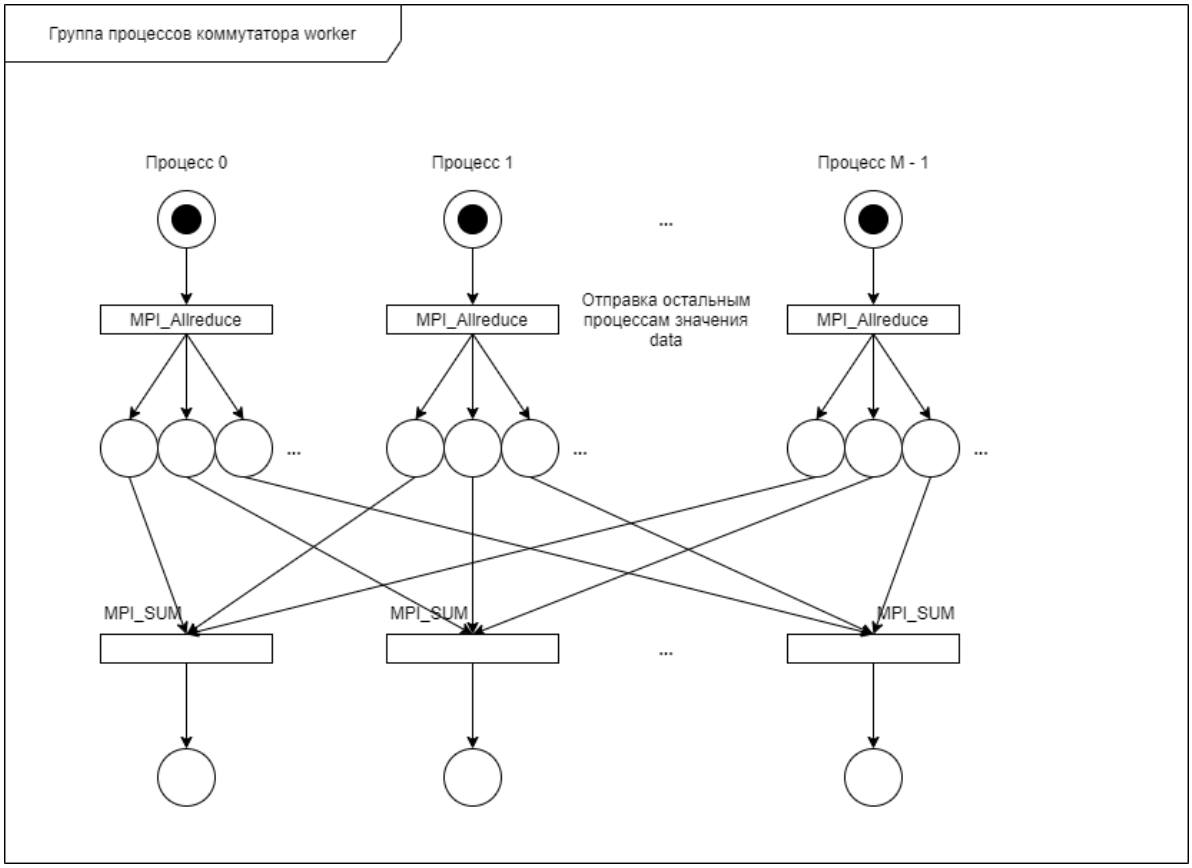
В каждом процессе дано целое число N, которое может принимать два значения: 0 и 1 (имеется хотя бы один процесс с N = 1). Кроме того, в каждом процессе с N = 1 дано вещественное число A. Используя функцию MPI_Comm_split и одну коллективную операцию редукции, найти сумму всех исходных чисел A и вывести ее во всех процессах с N = 1.

Указание: при вызове функции MPI_Comm_split в процессах, которые не требуется включать в новый коммуникатор, в качестве параметра color следует указывать константу MPI_UNDEFINED.

Краткое описание алгоритма.

Для начала в каждом процессе генерируется число isWorker, которое может быть равно 0 или 1. При этом у 0 процесса isWorker равен 1. Для каждого процесса задано вещественное число data, равное 2.5. Далее при помощи MPI_Comm_split происходит перемещение процессов с isWorker = 1 из коммутатора MPI_COMM_WORLD в новый коммутатор workers. После чего для процессов коммутатора workers производится коллективная операция MPI_Allreduce, при помощи которой вычисляется сумма исходных чисел data во всех процессах с isWorker = 1 и выводится на экран.

Формальное описание алгоритма.



Листинг программы.

Листинг 1. Код программы.

```
#include <iostream>
#include <mpi.h>


int main(int argc, char** argv) {
	int processNumber, processRank;
	MPI_Comm workers;

	MPI_Init(&argc, &argv);
	MPI_Comm_size(MPI_COMM_WORLD, &processNumber);
	MPI_Comm_rank(MPI_COMM_WORLD, &processRank);

	srand(time(nullptr) + static_cast<time_t>(processRank) * 1000);

	int isWorker = (processRank != 0)? rand() % 2 : 1;
	double data = 2.5;
	double sum = 0.0;

	MPI_Comm_split(MPI_COMM_WORLD, (isWorker)? isWorker : MPI_UNDEFINED,
processRank, &workers);

	if (isWorker) {
		double startTime = MPI_Wtime();
		MPI_Allreduce(&data, &sum, 1, MPI_DOUBLE, MPI_SUM, workers);
		double elapsedTime = MPI_Wtime() - startTime;
```

```
            std::cout << "Process #" << processRank << ": sum = " << sum <<
"\n";

            double maxTime;
            MPI_Reduce(&elapsedTime,  &maxTime,  1,  MPI_DOUBLE,  MPI_MAX,  0,
workers);

            if (processRank == 0) {
                  std::cout << "Elapsed time: " << maxTime << " seconds\n";
            }
      }

      MPI_Finalize();

      return 0;
}
```

Результаты работы программы на различном количестве процессов.

№ п/п	Количество процессоров	Результаты работы программы
1.	1	Process #0: sum = 2.5 Elapsed time: 8.09999e-06 seconds
2.	2	Process #0: sum = 5 Process #1: sum = 5 Elapsed time: 2.01e-05 seconds
3.	4	Process #0: sum = 5 Process #3: sum = 5 Elapsed time: 0.0001449 seconds
5.	8	Process #6: sum = 10 Process #5: sum = 10 Process #1: sum = 10 Process #0: sum = 10 Elapsed time: 0.0003756 seconds
6.	12	Process #6: sum = 17.5 Process #10: sum = 17.5 Process #1: sum = 17.5 Process #11: sum = 17.5 Process #7: sum = 17.5 Process #2: sum = 17.5 Process #0: sum = 17.5 Elapsed time: 0.000539 seconds
7.	16	Process #1: sum = 22.5 Process #9: sum = 22.5 Process #10: sum = 22.5 Process #4: sum = 22.5 Process #14: sum = 22.5 Process #0: sum = 22.5 Process #15: sum = 22.5 Process #6: sum = 22.5

		Process #5: sum = 22.5 Elapsed time: 0.0007302 seconds
8.	32	Process #21: sum = 40 Process #27: sum = 40 Process #17: sum = 40 Process #26: sum = 40 Process #4: sum = 40 Process #3: sum = 40 Process #13: sum = 40 Process #0: sum = 40 Process #31: sum = 40 Process #8: sum = 40 Process #7: sum = 40 Process #2: sum = 40 Process #22: sum = 40 Process #12: sum = 40 Process #23: sum = 40 Process #18: sum = 40 Elapsed time: 0.0050646 seconds
9.	64	Process #5: sum = 80 Process #39: sum = 80 Process #57: sum = 80 Process #16: sum = 80 Process #29: sum = 80 Process #43: sum = 80 Process #54: sum = 80 Process #10: sum = 80 Process #15: sum = 80 Process #11: sum = 80 Process #1: sum = 80 Process #20: sum = 80 Process #59: sum = 80 Process #40: sum = 80 Process #30: sum = 80 Process #44: sum = 80 Process #34: sum = 80 Process #35: sum = 80 Process #58: sum = 80 Process #2: sum = 80 Process #48: sum = 80 Process #24: sum = 80 Process #63: sum = 80 Process #25: sum = 80 Process #21: sum = 80 Process #62: sum = 80

		Process #49: sum = 80 Process #38: sum = 80 Process #6: sum = 80 Process #19: sum = 80 Process #0: sum = 80 Process #53: sum = 80 Elapsed time: 0.0249439 seconds

График зависимости времени выполнения программы от числа процессов.

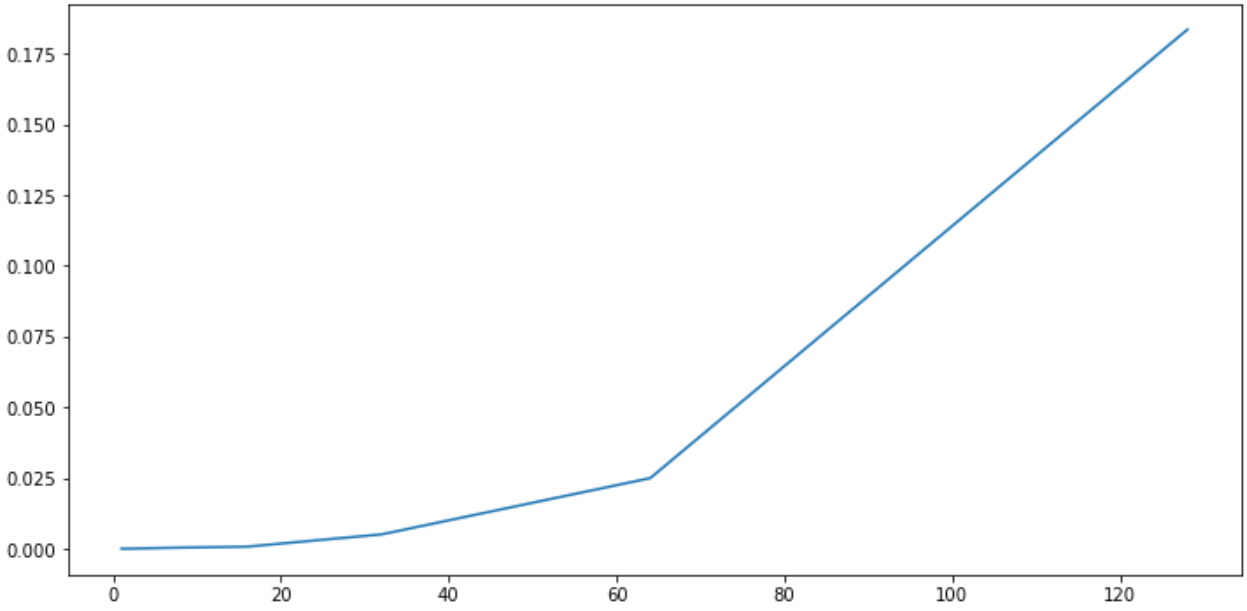
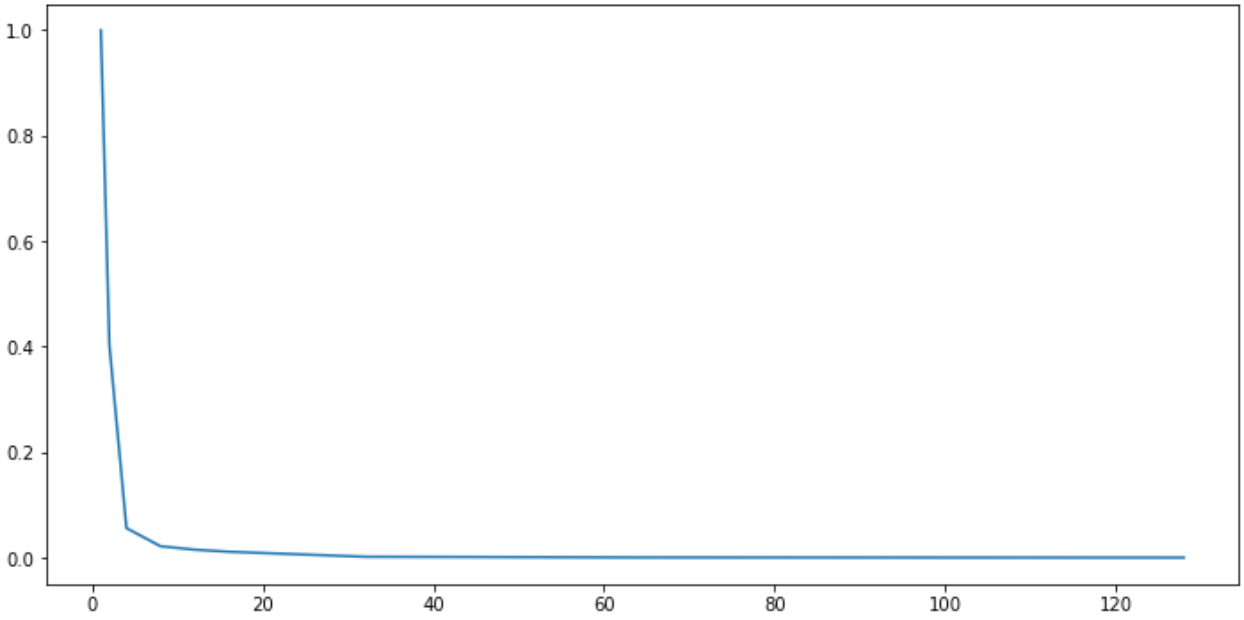


График ускорения.



Выводы по работе.

Была написана программа, осуществляющая суммирование вещественных чисел, получаемых от каждого процесса, относящегося к коммутатору workers, и сохранения результата суммы в каждом процессе, относящегося к коммутатору workers. Было выполнено измерение времени работы с разным количеством процессов. С ростом числа процессов будет расти и количество передаваемых вещественных чисел (количество элементов суммы), и количество считаемых сумм. Отсюда при увеличении количества процессов можно предположить, что время работы программы увеличивается со скоростью квадратичной функции.