МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

НОВОСИБИРСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Факультет информационных технологий

Кафедра параллельных вычислений

ОТЧЕТ

О ВЫПОЛНЕНИИ ЛАБОРАТОРНОЙ РАБОТЫ

“Параллельная реализация решения системы линейных алгебраических уравнений с помощью MPI”

студента 2 курса, 20212 группы

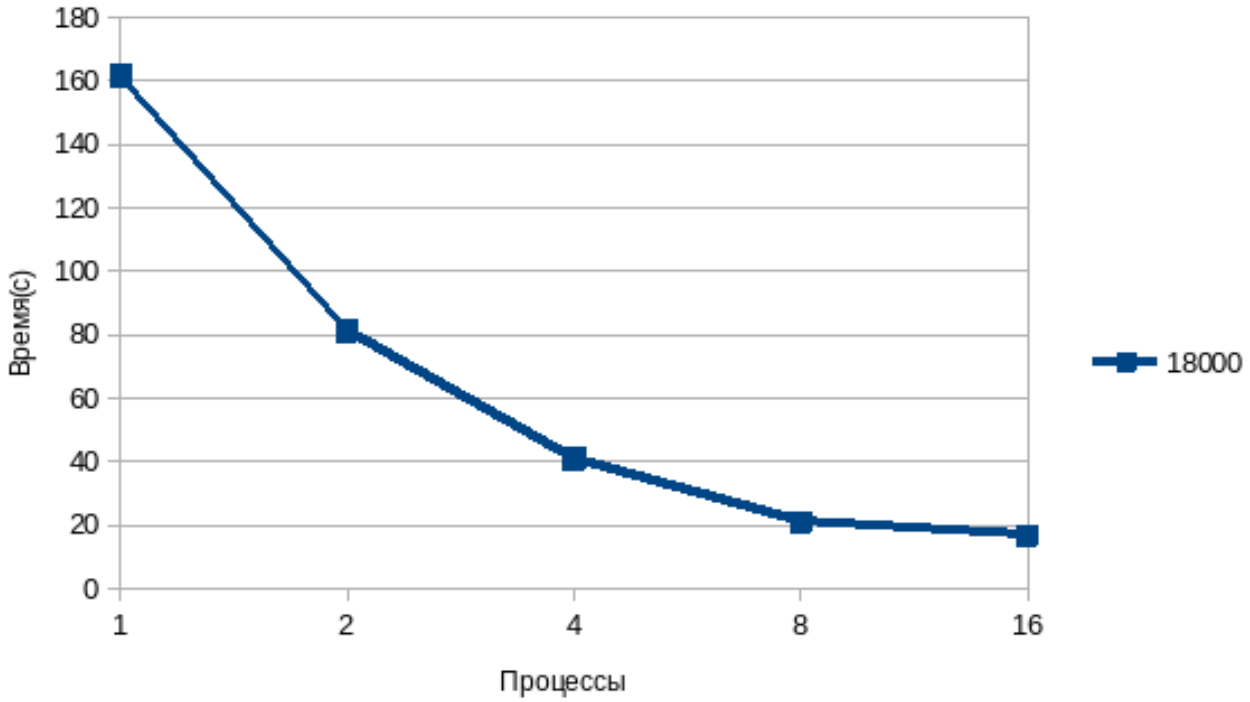
Курбатова Максима Андреевича

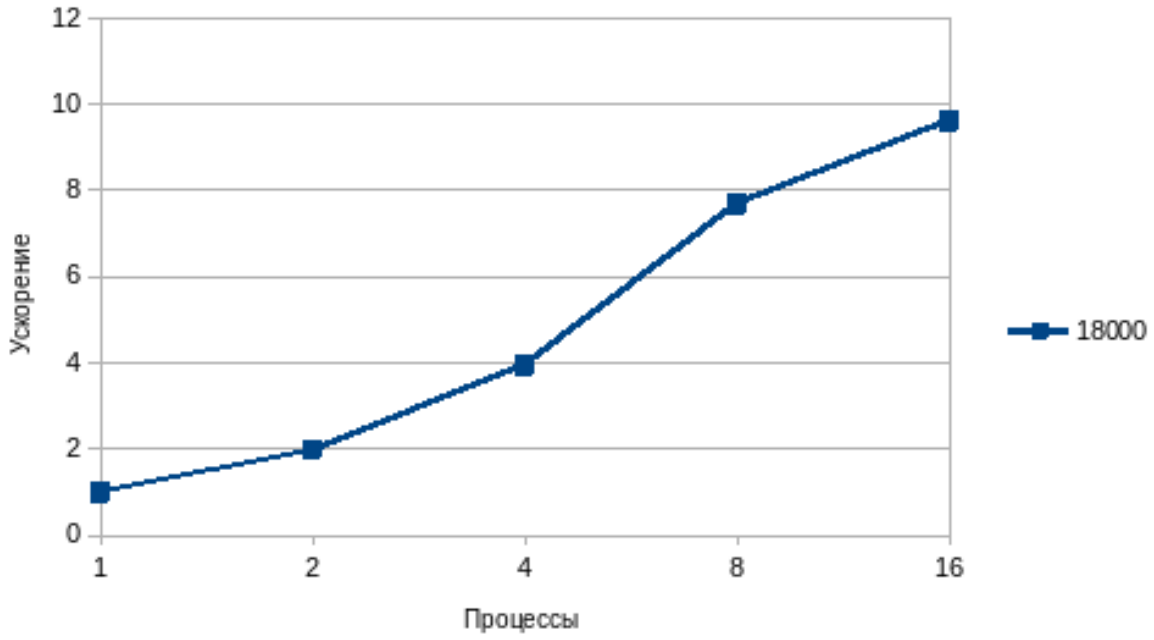
Направление 09.03.01 – «Информатика и вычислительная техника»

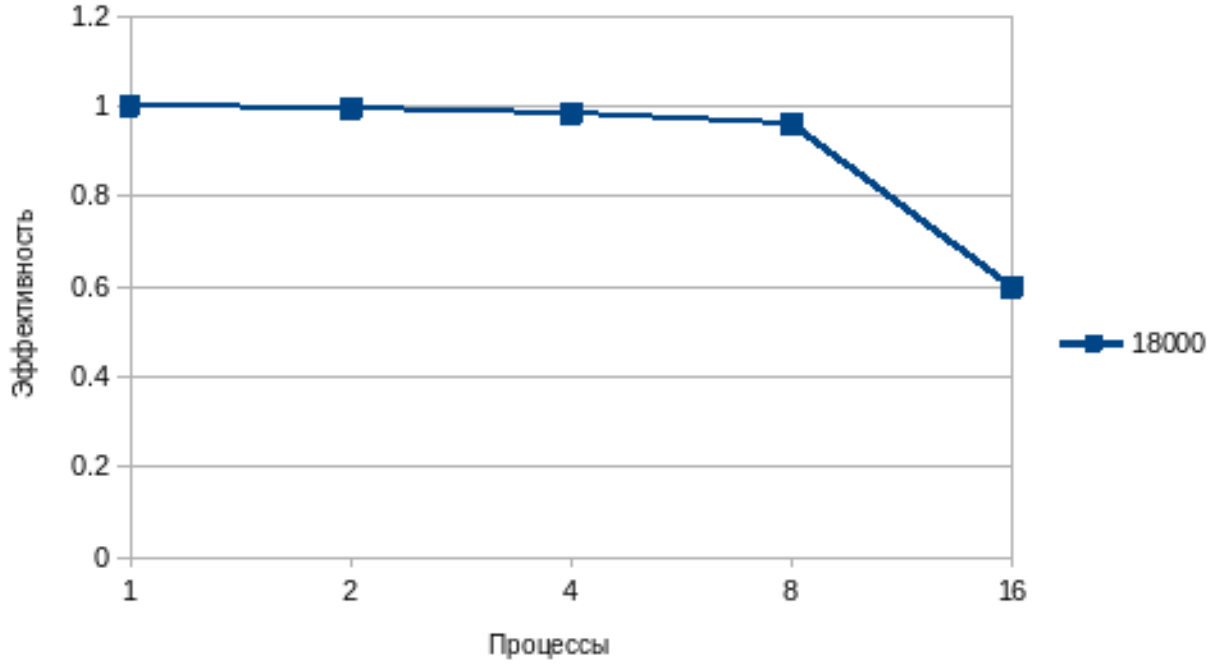
Преподаватель: Кудинов А.Ю.

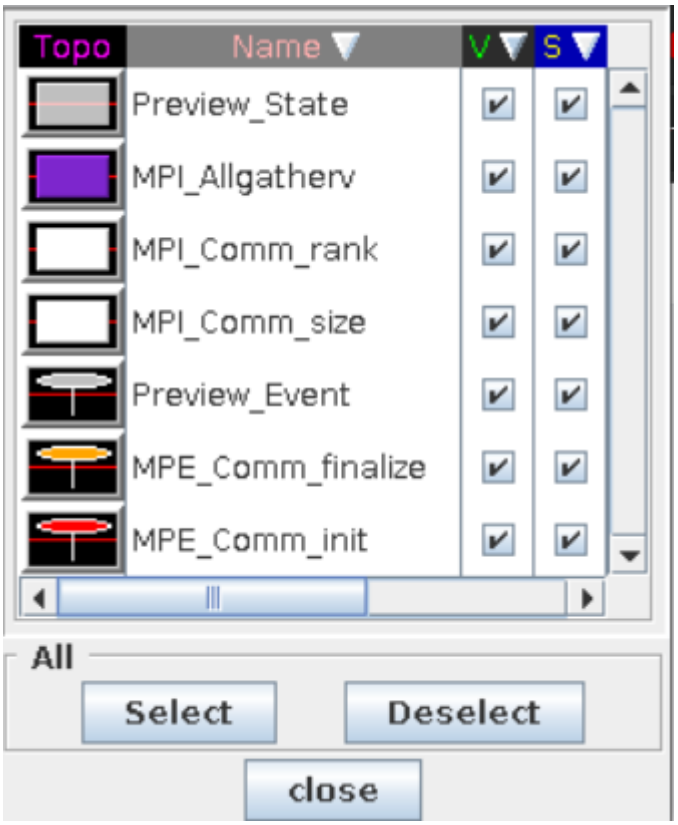
Новосибирск 2022

1 Вариант(Метод итерации)






1.2
1
0.8
0.6
0.4
0.2
0
Эффективность
1
2
4
8
16
Процессы
18000


Topo
Name
V
S
Preview_State
MPI_Allgatherv
MPI_Comm_rank
MPI_Comm_size
Preview_Event
MPE_Comm_finalize
MPE_Comm_init
All
Select
Deselect
close


Lowest / Max. Depth
Zoom Level
Global Min Time
View Init Time
Zoom Focus Time
View Final Time
Global Max Time
Time Per Pixel
TimeLines
SLOG-2
@ world_rank
Time (seconds)

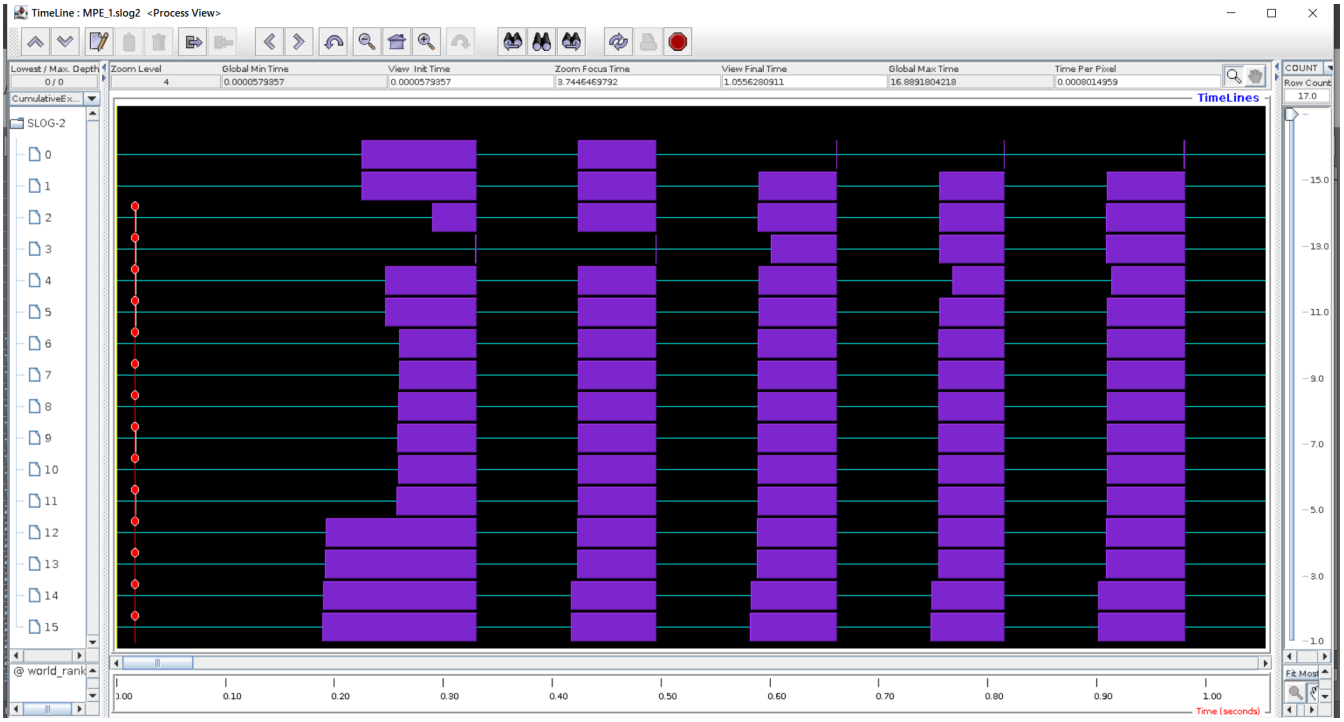
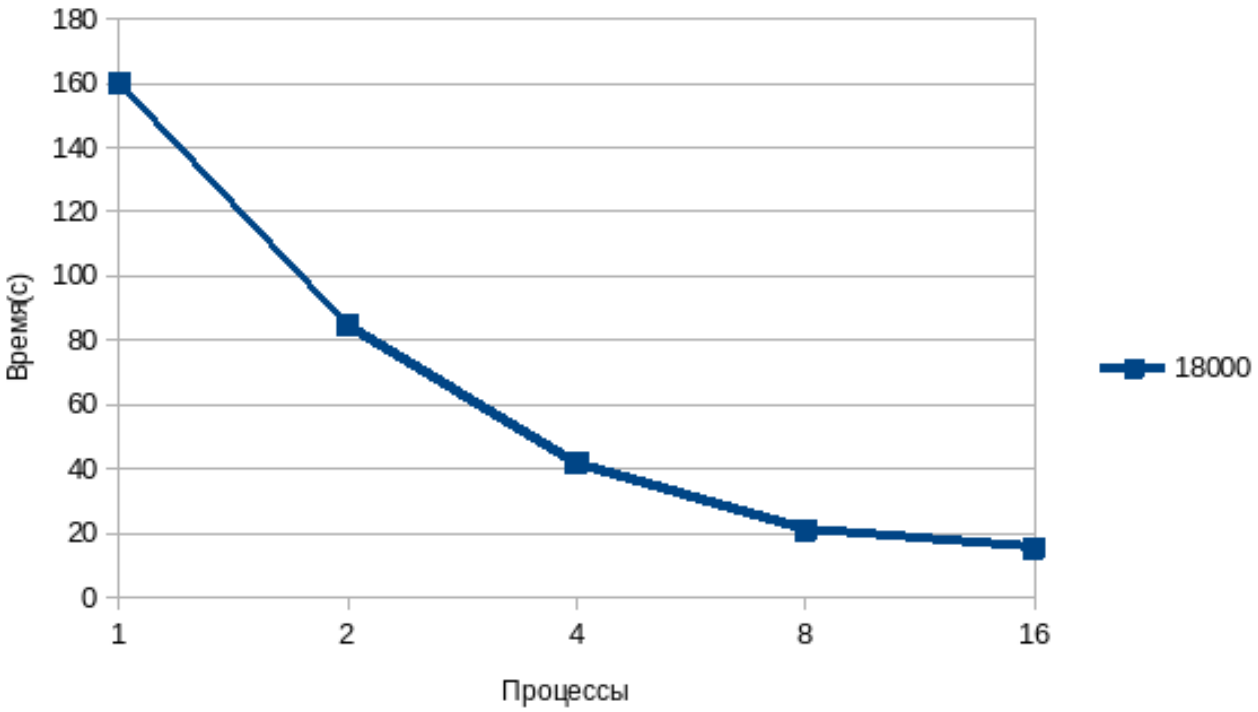
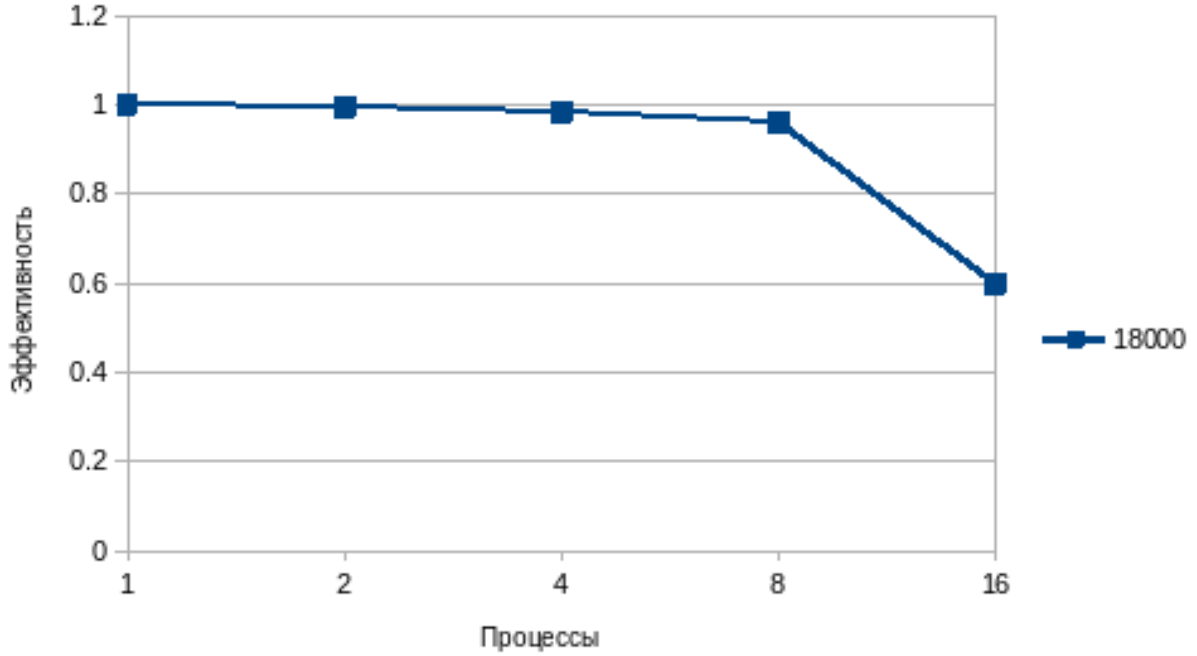
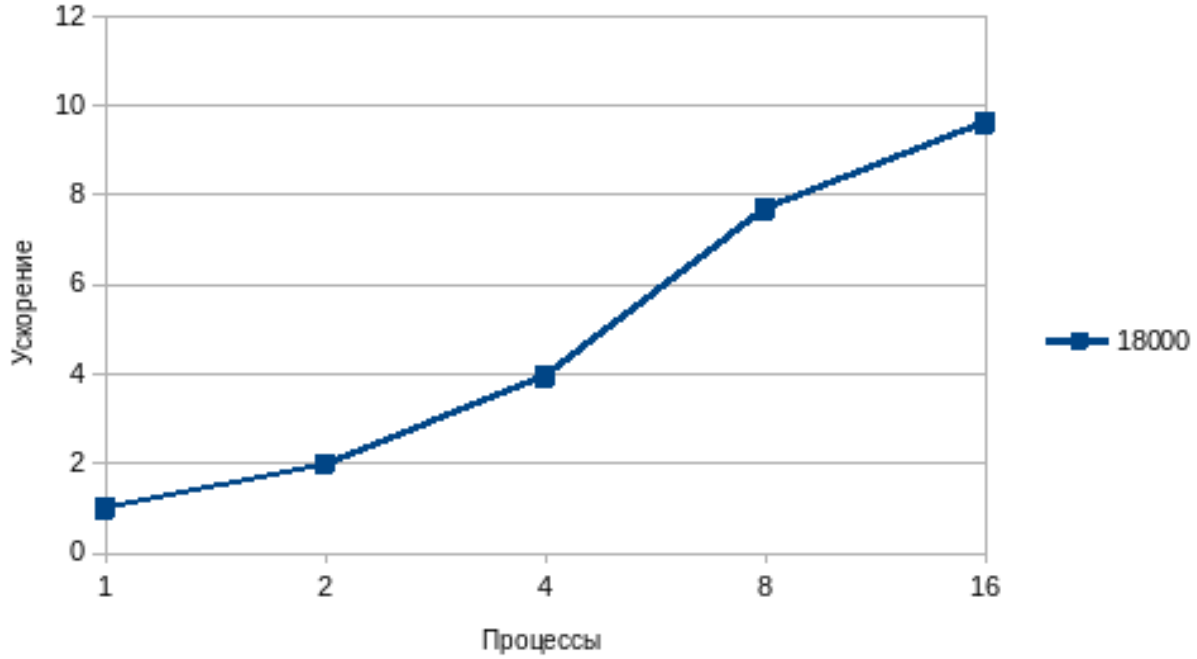

TimeLine : MPE_1.slog2 <Process View>
Lowest / Max. Depth
Zoom Level
Global Min Time
View Init Time
Zoom Focus Time
View Final Time
Global Max Time
Time Per Pixel
TimeLines
SLOG-2
@ world_rank
Time (seconds)

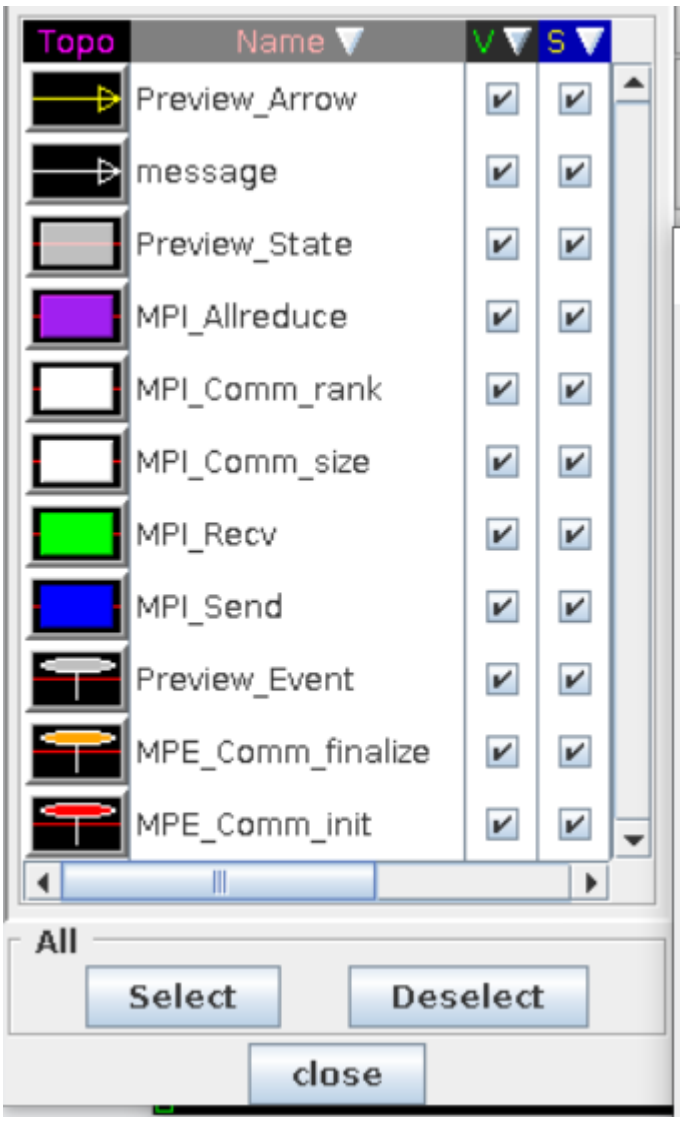
рис 1

2 Вариант(Метод итерации)




1.2
1
0.8
0.6
0.4
0.2
0
Эффективность
1
2
4
8
16
Процессы
18000


12
10
8
6
4
2
0
Ускорение
1
2
4
8
16
Процессы
18000


Topo
Name
V
S
Preview_Arrow
message
Preview_State
MPI_Allreduce
MPI_Comm_rank
MPI_Comm_size
MPI_Recv
MPI_Send
Preview_Event
MPE_Comm_finalize
MPE_Comm_init
All
Select
Deselect
close

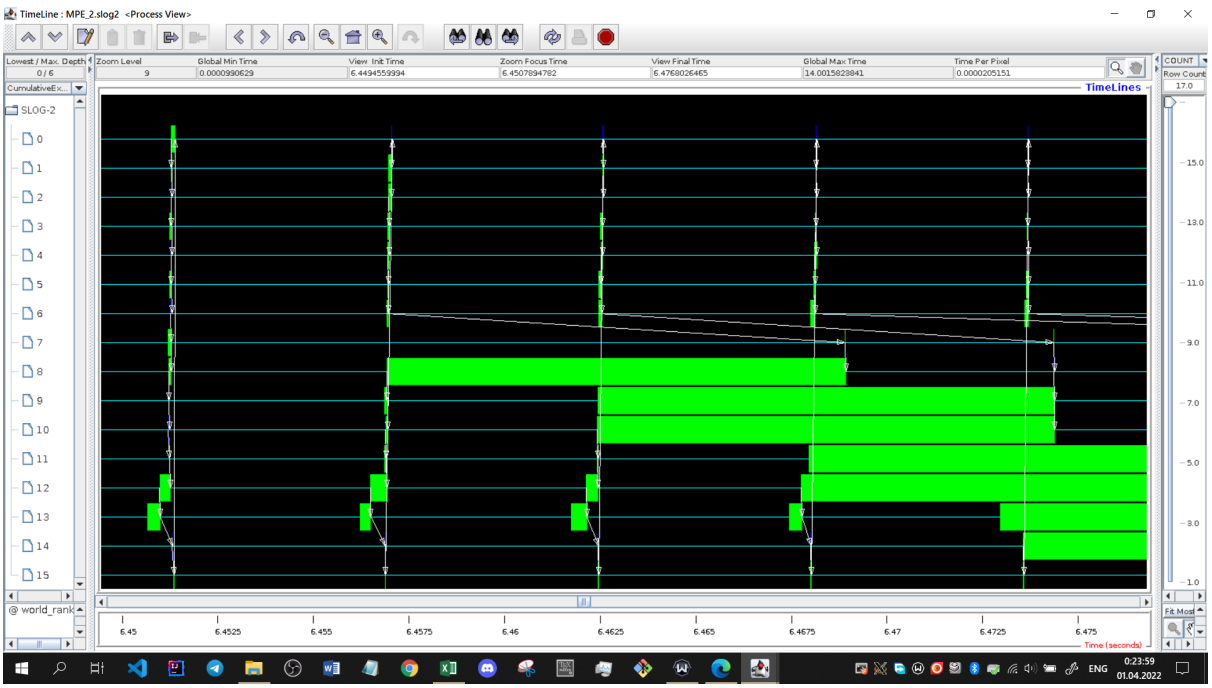

TimeLine : MPE_2.slog2 <Process View>
TimeLines
SLOG-2
Time (seconds)

рис 2

Анализ + вопросы с семинара:

1) Почему на 16 процессах падает эффективность(помимо объяснения про землекопов)?

Ответ:

Идеальные условия: Судя по профилированию, функция MPI_Allgatherv работает в среднем примерно 0.09 секунд. Далее я написал простенькую программу, чтобы понять, сколько нужно времени на умножение матрицы 18000x18000 / n на вектор, длиной 18000.

```cpp
#include<iostream>
#define SIZE 18000*18000 / 16
#define WIDTH 18000

double scalar_mul(double* vec_1, double* vec_2, int N) {
    double res = 0.0;
    for(int i = 0; i < N; i++) {
        res += vec_1[i] * vec_2[i];
    }
    return res;
}

int main(int argc, char* argv[]) {
    double* arr = new double[SIZE];
    double* vector = new double[WIDTH];

    struct timespec start, end;
    clock_gettime(CLOCK_MONOTONIC_RAW, &start);

    for(int i = 0; i < SIZE; i+=WIDTH) {
        scalar_mul(arr + i, vector, WIDTH);
    }

    clock_gettime(CLOCK_MONOTONIC_RAW, &end);
    printf("Time taken: %lf sec.\n", end.tv_sec-start.tv_sec + 0.000000001*(end.tv_nsec-start.tv_nsec));
    return 0;
}
```

16 процессов:

```
maxim@Maxwell:~/Desktop/OPP/lab2$ ./a.out
Time taken: 0.123170 sec.
```

8 процессов:

```
maxim@Maxwell:~/Desktop/OPP/lab2$ ./a.out
Time taken: 0.218999 sec.
```

8 процессов:

1) У каждого процесса 18000 * 18000 / 8 = 2225 строк размером 18000.
2) За одну итерацию 0.22 + 0.09 секунд

16 процессов:

1) У каждого процесса 18000 * 18000 / 16 = 1125 строк размером 18000.
2) За одну итерацию 0.12 + 0.09 секунд

Текущие условия: На самом деле эффективность не должны была упасть!!! Но, так как, если мы запускаем 16 процессов, то у нас процессы запускаются на двух конечных устройствах, что увеличивает время работы MPI_Allgatherv + из-за большего числа процессов она и так будет работать дольше. И увеличивается оно более чем в два раза, отсюда получается, что время за одну итерацию увеличивается, а значит эффективность падает.

2) Почему же MPI_Allgatherv работает разное время?

На самом деле они и работают примерно одинаково, просто внутри MPI_Allgatherv стоит MPI_Barrier. А значит все процессы вынуждены простаивать, пока MPI_Allgatherv отработает на всех процессах.

3) Почему на рис 2 есть более длинные зеленые полосы(MPI_Recv)?
   Опять же, все было хорошо на процессах 0-7 и 8-15, но время функции MPI_Recv увеличилась именно при общении 7 и 8 процессов, это связано, что ядра на разных конечных устройствах, вследствии чего, образовывается задержка циклического пересыла даллее. Те 8 процесс еще сам не принял данные, а 9 процесс уже ждет от него эти данные. Хорошая иллюстрация зависимости по данным.

Вывод:

Второй вариант реализации не уступает первому и по памяти, и по времени. Как минимум, это может быть из-за того, что я использовал свой циклический буффер и выделял дополнительную память для хранения пересланных значений вне таймера, в то время, как внутри MPI_Allgatherv происходит выделения памяти, которое входит в общее время, измеренное таймером.