Отчет

1. OpenMP

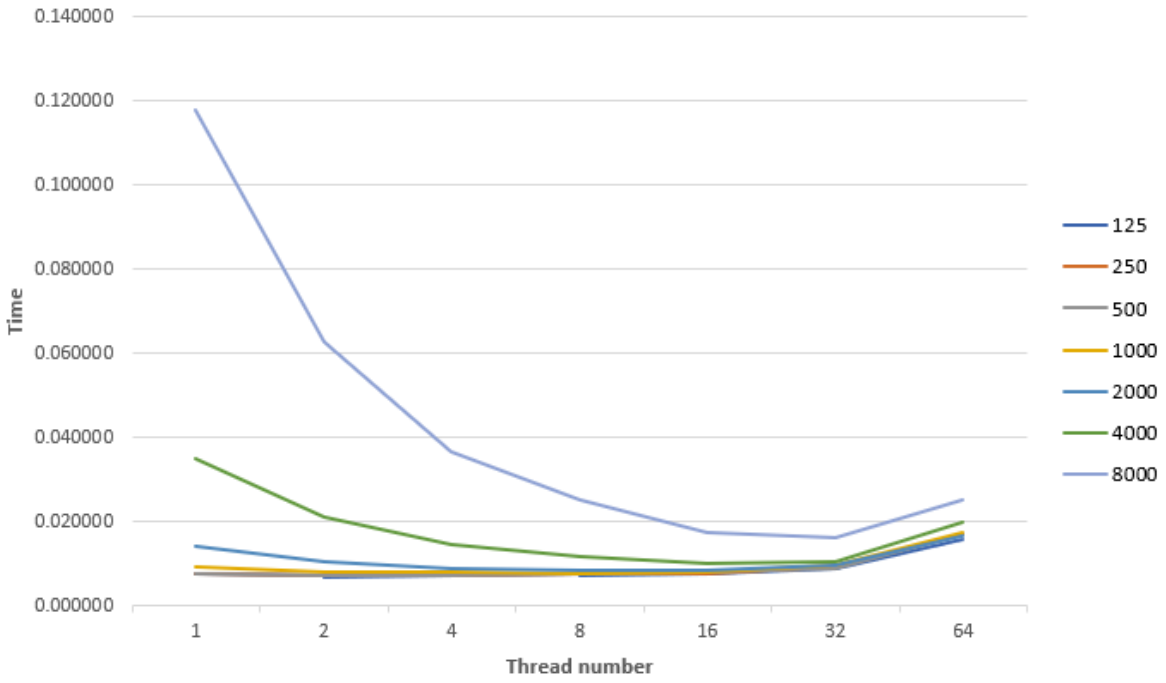
1.1. Source

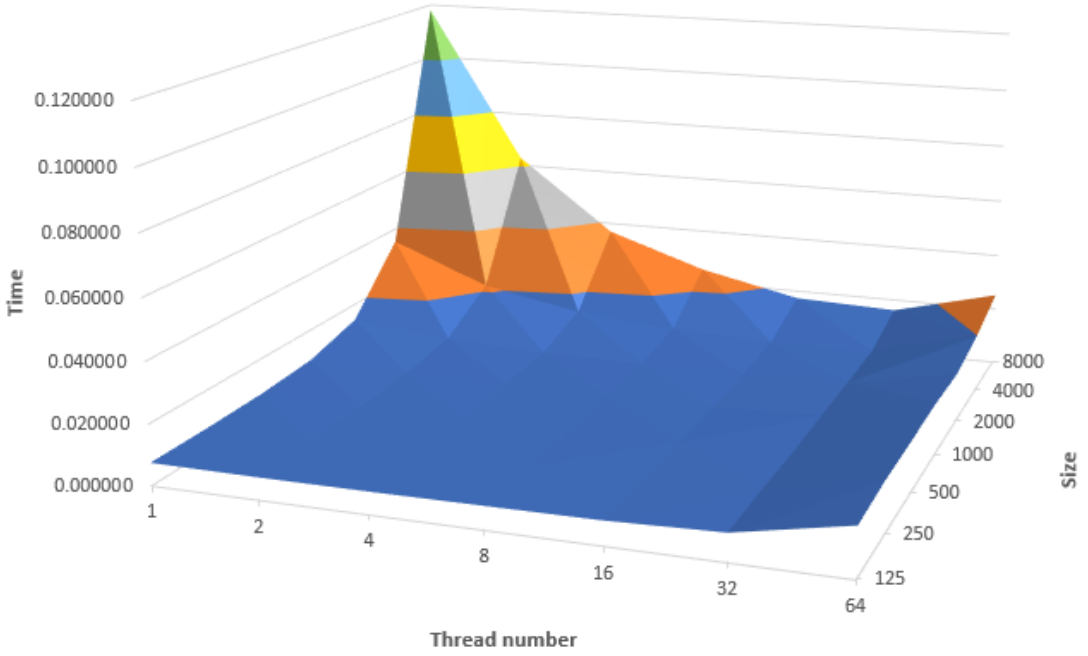
```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <omp.h>

int main(int argc, char* argv[]) {
    int i, j;
    int N = atoi(argv[1]);

    double matrix[N*N];
    memset(matrix, 1, N*N);

    double vector[N];
    memset(vector, 1, N);

    double result[N];
    memset(result, 0, N);

    double start = omp_get_wtime();

    #pragma omp parallel
    {
        double result_private[N];
        memset(result_private, 0, N);
        int i, j;
        #pragma omp for
        for(i = 0; i < N; i++) {
            for(j = 0; j < N; j++) {
                result_private[i] += matrix[i * N + j] * vector[j];
            }
        }
        #pragma omp critical
        {
            for(i = 0; i < N; i++) result[i] += result_private[i];
        }
    }
    double end = omp_get_wtime();

    printf("%f\n", end - start);
    return 0;
}
```

1.2. Results (size - row and column number, boost = 1 thread time / n threads time)

threads\size	125	250	500	1000	2000	4000	8000
1	0.007545	0.007518	0.007867	0.009205	0.014249	0.035198	0.117973
2	0.007350	0.007347	0.007438	0.008170	0.010753	0.021285	0.062858
4	0.007381	0.007360	0.007490	0.007926	0.009106	0.014669	0.036765
8	0.007599	0.007564	0.007564	0.007887	0.008668	0.011852	0.02511
16	0.007838	0.007832	0.008077	0.007995	0.008438	0.010191	0.017319
32	0.009010	0.009330	0.009020	0.009600	0.009890	0.010522	0.016116
64	0.015957	0.017161	0.017215	0.017514	0.016669	0.019975	0.025222
boost							
1	1	1	1	1	1	1	1
2	1.026531	1.023275	1.057677	1.126683	1.325119	1.653653	1.876818
4	1.022219	1.021467	1.050334	1.161368	1.564792	2.399482	3.20884
8	0.992894	0.993919	1.040058	1.16711	1.643862	2.969794	4.698248
16	0.962618	0.959908	0.974	1.151345	1.68867	3.453832	6.811767
32	0.837403	0.805788	0.872173	0.958854	1.440748	3.345182	7.320241
64	0.472833	0.438086	0.456985	0.52558	0.85482	1.762103	4.677385

1.3. Diagrams





1.4. Conclusion

На входных данных большего размера (матрицы 1000*1000 и более) хорошо видно, что при увеличении количества нитей пропорционально падает время выполнения, однако при количестве нитей 32 и более заметен рост времени выполнения

2. MPI

2.1. Algoritm

В данном задании использовалась ленточная схема

2.2. Source

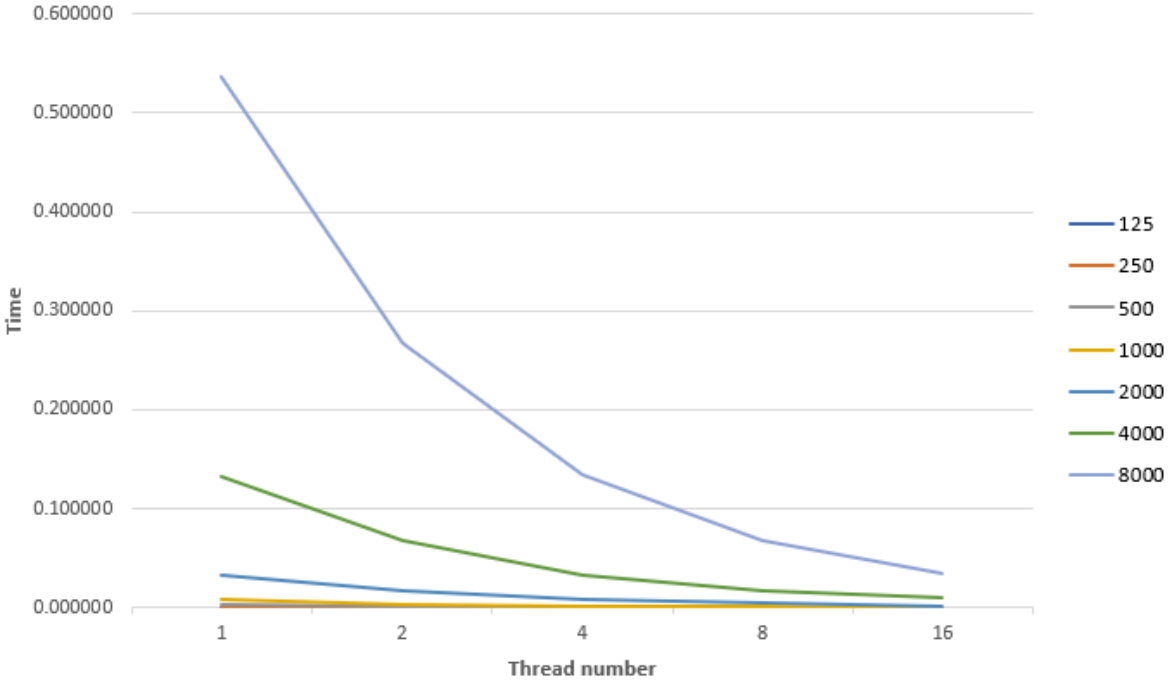
```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <mpi.h>

int main(int argc, char* argv[]) {
    // Initialize MPI
    MPI_Init(&argc, &argv);

    int N = atoi(argv[1]);

    double *matrix[N];
    int i, j;
    for(i = 0; i < N; i++) {
        matrix[i] = (double*)calloc(N, sizeof(double));
        memset(matrix[i], 1, N);
    }

    double vector[N];
    memset(vector, 1, N);

    double result[N];
    memset(result, 0, N);
```

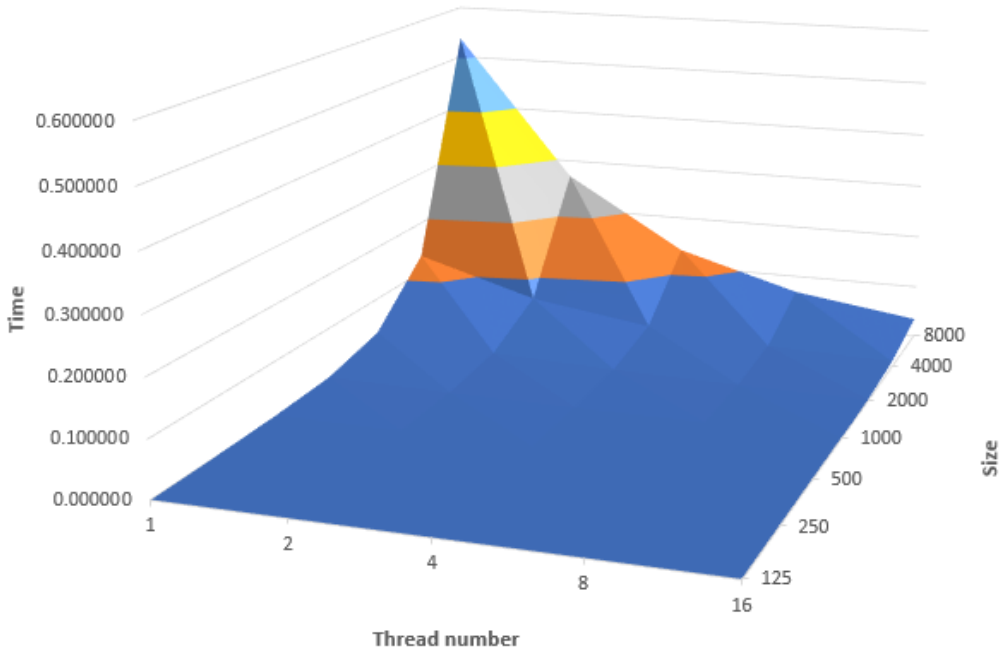
```
    // Get the number of processes and the rank of the current process
    int num_procs, rank;
    MPI_Comm_size(MPI_COMM_WORLD, &num_procs);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);

    // The matrix and vector are divided into chunks and distributed among the processes
    int chunk_size = N / num_procs;
    int start = rank * chunk_size;
    int end = start + chunk_size;

    double start_time = MPI_Wtime();

    for(i = start; i < end; i++) {
        for(j = 0; j < N; j++) {
            result[i] += matrix[i][j] * vector[j];
        }
    }

    if(rank == 0) {
        double end_time = MPI_Wtime();
        printf("%f\n", end_time - start_time);
    }

    // Finalize MP
    MPI_Finalize();

    return 0;
}
```

2.3. Results (size - row and column number, boost = 1 thread time / n threads time)

processes\siz	125	250	500	1000	2000	4000	8000
1	0.000164	0.000520	0.002582	0.008300	0.033394	0.132933	0.537199
2	0.000065	0.000262	0.001034	0.004130	0.016675	0.067077	0.267979
4	0.000033	0.000131	0.000526	0.002072	0.008484	0.033563	0.133516
8	0.000017	0.000066	0.000261	0.001050	0.004216	0.017200	0.068147
16	0.000010	0.000031	0.000129	0.000515	0.002135	0.010651	0.034884
boost							
1	1	1	1	1	1	1	1
2	2.523077	1.984733	2.497099	2.009685	2.002639	1.981797	2.004631
4	4.969697	3.969466	4.908745	4.005792	3.936115	3.960701	4.02348
8	9.647059	7.878788	9.89272	7.904762	7.920778	7.728663	7.882944
16	16.4	16.77419	20.0155	16.1165	15.64122	12.4808	15.39958

2.4. Diagrams





2.5. Conclusion

Результаты, в целом, аналогичны OpenMP, однако общее время исполнения для MPI всё-таки меньше. Также увеличение времени исполнения при возрастании числа процессов не так заметно, т.к. как здесь используется меньшие их количесвто (не

более 16, а не 64 как для нитей) в силу невозможности запуска программы на таком количестве процессов.