Parallel Programming on CPUs
Master HPC-IA, Mines Paris

Dimitrios Tsitsos

December 30, 2022

Introduction

The objective of this project was to understand how parallel programming in CPUs work using OpenMP, TBB, MPI and see in practise how a sequential code can be improved when parallelizing. The project of this course was splitted in two parts.

- The first part was to parallelize a matrix-vector multiplication with dense and sparse matrices.
- The second part was to parallelize our project "life-of-boids" with OMP and TBB.

Benchmark report for Dense/Sparse Matrices

In the tables below we can see the timetables of Sparse and Dense matrices.

	np	nx	Sequential Time	MPI Time	Acceleration
0	1	50	0.006353	0.032869	0.193271
1	1	100	0.097614	0.506982	0.192539
2	1	150	0.487082	2.585590	0.188384
3	1	200	1.563480	8.081150	0.193472
4	2	50	0.006510	0.041299	0.157622
5	2	100	0.098354	0.667362	0.147377
6	2	150	0.502209	3.384030	0.148405
7	2	200	1.558850	10.563100	0.147576
8	4	50	0.006973	0.038067	0.183188
9	4	100	0.106717	0.701871	0.152047
10	4	150	0.532777	3.516980	0.151487
11	4	200	1.668450	11.196200	0.149018
12	8	50	0.006927	0.133201	0.052001
13	8	100	0.104992	2.246250	0.046741
14	8	150	0.531111	11.493400	0.046210
15	8	200	1.654650	36.343900	0.045527
16	12	50	0.006876	0.172615	0.039837
17	12	100	0.104269	2.769180	0.037653
18	12	150	0.543153	14.288600	0.038013
19	12	200	1.657470	44.882600	0.036929
20	16	50	0.006983	0.218228	0.032000
21	16	100	0.104263	3.063140	0.034038
22	16	150	0.532302	15.540100	0.034253
23	16	200	1.703750	49.258300	0.034588
24	24	50	0.006881	0.264412	0.026026
25	24	100	0.106698	3.278950	0.032540
26	24	150	0.537072	16.885600	0.031807
27	24	200	1.656840	53.461700	0.030991

Figure 1: Timetabe of Dense matrix.

	np	nx	Sequential Time	MPI Time	Acceleration
0	1	50	0.000017	0.000049	0.341926
1	1	100	0.000063	0.000201	0.315040
2	1	150	0.000141	0.000486	0.290524
3	1	200	0.000252	0.001341	0.187702
4	2	50	0.000017	0.000177	0.095207
5	2	100	0.000063	0.000458	0.138041
6	2	150	0.000146	0.000932	0.156544
7	2	200	0.000261	0.002237	0.116790
8	4	50	0.000017	0.000288	0.060596
9	4	100	0.000070	0.000691	0.100873
10	4	150	0.000156	0.001219	0.128107
11	4	200	0.000261	0.003940	0.066179
12	8	50	0.000016	0.184537	0.000088
13	8	100	0.000070	0.002309	0.030197
14	8	150	0.000156	0.007354	0.021156
15	8	200	0.000269	0.013150	0.020441
16	12	50	0.000018	0.011724	0.001525
17	12	100	0.000070	0.029854	0.002339
18	12	150	0.000156	0.009401	0.016545
19	12	200	0.000281	0.017854	0.015762
20	16	50	0.000017	0.275892	0.000061
21	16	100	0.000070	0.041514	0.001682
22	16	150	0.000150	0.053395	0.002817
23	16	200	0.000269	0.046359	0.005795
24	24	50	0.000017	0.080612	0.000209
25	24	100	0.000070	0.081505	0.000856
26	24	150	0.000151	0.102072	0.001478
27	24	200	0.000290	0.089008	0.003254

Figure 2: Timetabe of Sparse matrix.

At the next plots we can see the evolution of the calculation time as a function of matrix sizes, for different numbers of processors.

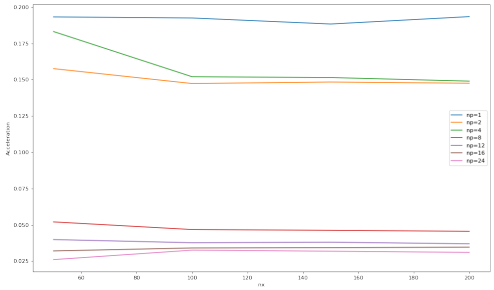

Figure 3: Acceleration vs number of rows for each number of threads (Dense).

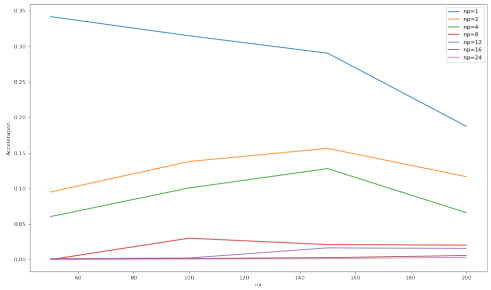

Figure 4: Acceleration vs number of rows for each number of threads (Sparse).

We can see that for both dense and sparse matrices the less np we have, the more acceleration we get.

At the next plots we can see the evolution of the calculation time as a function of the number of processors, for different sizes of matrices.

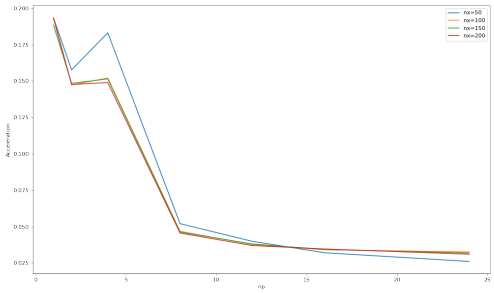

Figure 5: Acceleration vs number of threads for each number of rows.

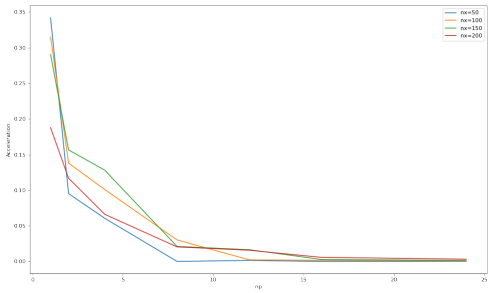

Figure 6: Acceleration vs number of threads for each number of rows.

We can see here that the speedup decreased as the np becomes bigger.

Conclusion

As we can see from the timetable and the plots that the Acceleration is not good as the sequential time is usually less than the MPI time. One solution that could improve our results was to use the MPIScutter and MPIGather but the speedup would remain bad.

Benchmark report for Life of Boids

In the table below we can see the timetable of Life of Boids for OMP and TBB.

	Threads	Mode	Count
0	1	OMP	1770.910
1	1	TBB	1794.370
2	2	OMP	914.480
3	2	TBB	938.579
4	4	OMP	497.881
5	4	TBB	516.218
6	8	OMP	636.666
7	8	TBB	524.677
8	12	OMP	631.739
9	12	TBB	553.282
10	16	OMP	595.284
11	16	TBB	626.429
12	24	OMP	615.089
13	24	TBB	718.967

Figure 7: Timetabe of Life of Boids.

In the following plot we see the evolution of execution time for 1000 birds with TBB OpenMP.

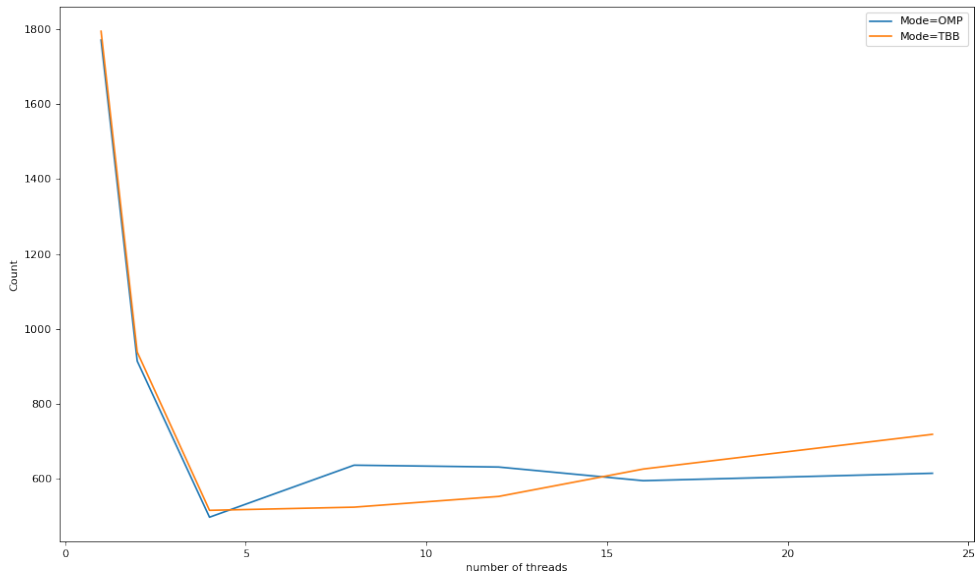

Figure 8: Timetabe of Life of Boids.

Conclusion

Compared to the Matrices the Life-of-Boids gives very good speedup, both with OMP and TBB. Compred to these two tho, the one who gives the best speedup is the TBB (based on the timetable).

As a result, we have seen in the scope of this project how parallel programming is used and how we can improve an already existing project (Life of Boids) parallelizing it. The difference is significant qnd we can reach even better result if we paralellize bigger part of the code!