Numerical Scientific Computing

Jakob Lund Jensen
jjens19@student.aau.dk

November 29, 2024

Mini Project Description

1 Title

Mini-project step 1: naive, numpy, numba and multi-core

2 Moodle description

For this hand-in, you are expected to deliver your work on the Mandelbrot mini-project, including the following:

Code as runnable Python files (.py or .ipynb):

1. naive (loops),
2. numpy (vectorized),
3. Numba-optimized version(s),
4. Parallel version using multiprocessing

Worksheet (either as supplementary PDF or in Jupyter Notebook with code) that includes a comparison of performance (in terms of execution time) for each of the algorithms.

Furthermore, for the multiprocessing version, you should analyze:

1. what is the optimal chunk size for different number of processes (P) (in terms of execution time)
2. compare execution time and speed-up for different number of processes (P)

3 Execution time comparison

I have decided to include results from both my desktop computer and my laptop because the Numpy implementation gave a strange result on my desktop. I have tried running the Numpy solution multiple times and still get around 1670s, while only getting around 330s on my laptop. I cannot explain this, especially since the Naïve and Numba implementations both run faster on my desktop. This comparison shows for multiprocessing results both with and without the usage of Numba. The notation used here is (amount of processes, the number of chunks). Since my laptop can at most run 8 processes, results have not been gathered for these categories.
All results are in seconds and have been rounded to two decimals.

Algorithm	Naïve	Numpy	Numba	MP(12,20)	MP(12,20) Numba	MP(10,1)	MP(10,1) Numba
Desktop PC	174,88	1670,19	14,33	55,89	37,55	53,95	36,34
Laptop PC	200,04	329,79	16,25	-	-	-	-

The Numpy implementation was quite poor, leading to worse execution speeds than the Naïve implementation. However, I cannot explain why the Numpy implementation runs exceptionally poor on my desktop. In theory, the Numpy solution should be quicker than the Naïve implementation.

4 Comparison of different amounts of processes and chunk sizes

The table shows all data using 1-12 processes and 1-5000 chunks that are evenly divisible by 5000. All results in the table have been rounded to the nearest second. The number of processes is shown horizontally, and the number of chunks is shown vertically. The green cell indicates the overall fastest combination, the blue cells indicate the fastest cells for each amount of processes, while the red cell indicates the overall slowest combination. The colored cells are taking the decimals into account.

	1	2	3	4	5	6	7	8	9	10	11	12
1	185	97	69	58	54	53	52	52	53	51	52	54
2	190	97	72	58	54	53	52	52	52	52	51	52
4	186	96	69	58	54	53	52	52	52	52	55	52
5	187	96	69	63	54	53	52	52	52	52	57	52
8	187	98	70	60	55	54	53	52	52	55	57	52
10	188	96	70	60	55	54	54	53	52	53	62	52
20	189	96	70	60	55	54	54	52	57	53	59	51
25	187	97	70	60	56	55	59	52	53	53	59	52
40	188	97	71	61	59	55	55	52	53	53	60	52
50	192	97	71	61	56	56	55	52	54	54	60	52
100	189	98	72	62	58	56	56	54	55	55	59	53
125	187	98	73	62	58	57	56	55	55	55	55	53
200	194	101	75	65	59	59	58	56	56	57	57	58
250	193	102	78	64	60	60	59	57	58	57	58	56
500	190	103	83	72	64	64	64	66	63	66	63	61
625	193	104	83	70	69	73	68	65	66	66	66	64
1000	195	120	91	87	71	72	72	71	71	72	72	70
1250	196	112	105	80	79	80	79	79	79	80	79	79
2500	204	124	124	123	124	125	124	124	124	124	125	124
5000	216	212	212	212	215	212	214	212	212	213	212	212

Green cell: 51,073861100012
Red cell: 215,543883099977

From here we can determine the optimal chunk sizes for each amount of processes:

Number of processes	1	2	3	4	5	6	7	8	9	10	11	12
Optimal chunk size (nx5000)	5000	1000	1000	5000	2500	2500	5000	5000	2500	5000	2500	250

Figure 1 shows the execution speed for the different amounts of processes using different amounts of chunks. A 2nd degree polynomial tendency line is fitted to the scatters.

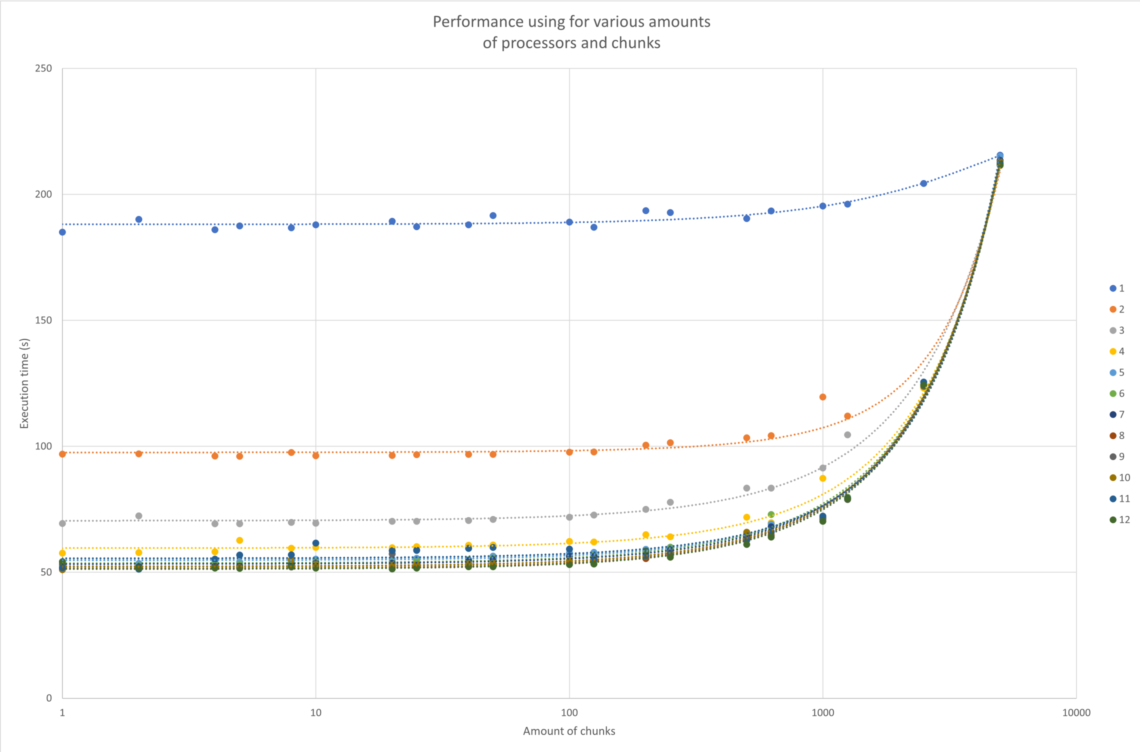

Figure 1: Logarithmic graph showing execution speed for different combinations of processes and chunks

The theoretical speedup is given by the following formula:

$$S_p = \frac{T_1}{T_p}$$

Where S_p is the speedup, T_1 is the time using a single process, and T_p is the time using p processes.

Processes	1	2	3	4	5	6	7	8	9	10	11	12
Speedup	0,95	1,82	2,53	3,03	3,27	3,32	3,38	3,39	3,39	3,42	3,41	3,40

This table shows the speedup for the various amounts of processes using their best-performing amount of chunks. The speedup has been rounded to 2 decimal points.