CPU info:

```
Architecture:            x86_64
CPU(s):                  80
Vendor ID:               GenuineIntel
Model name:              Intel(R) Xeon(R) Gold 6248 CPU @ 2.50GHz
Thread(s) per core:      2
Core(s) per socket:      20
Socket(s):               2
CPU max MHz:             3900.0000
CPU min MHz:             1000.0000
```

Caches (sum of all):

```
L1d:                     1.3 MiB (40 instances)
L1i:                     1.3 MiB (40 instances)
L2:                      40 MiB (40 instances)
L3:                      55 MiB (2 instances)
```

Наименование сервера:

ProLiant XL270d Gen10

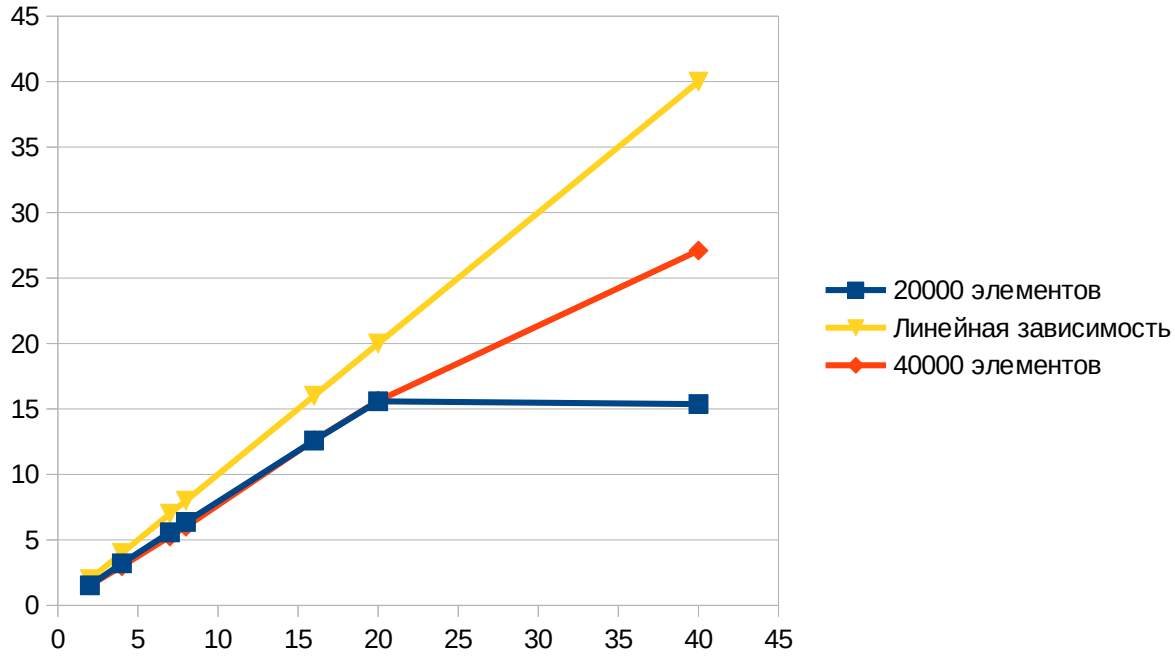
NUMA node:

```
2 nodes
node 0 size: 385636 MB
node 1 size: 387008 MB
```

OS: Ubuntu 22.04.5 LTS

Количество потоков

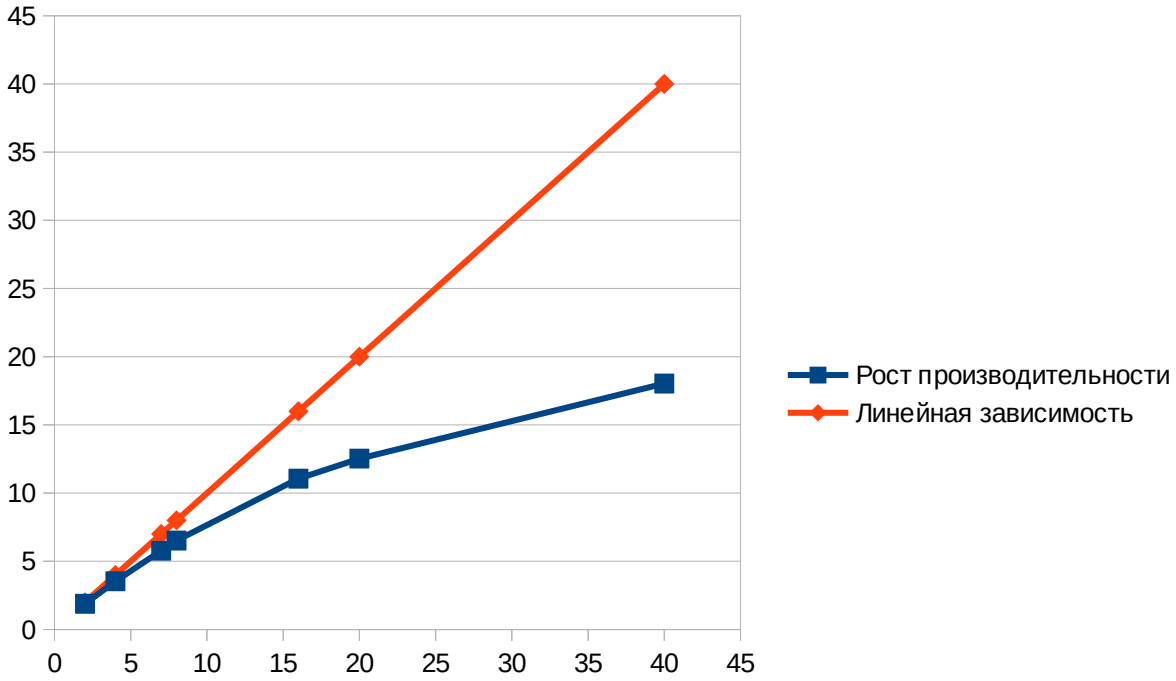
M=N	2			4		7		8		16		20		40	
	T1	T2	S2	T4	S4	T7	S7	T8	S8	T16	S16	T20	S20	T40	S40
20000	1.295	0.681	1.534	0.325	3.215	0.187	5.588	0.164	6.371	0.083	12.59	0.067	15.59	0.068	15.376
40000	5.151	2.735	1.526	1.394	2.994	0.793	5.263	0.694	6.014	0.331	12.61	0.267	15.632	0.154	27.103



Распараллеливание на 2 — 20 потоков приносит окололинейный результат. При распараллеливании больше чем на 20 потоков эффективность от распараллеливания на матрице 20000x20000 снижается, в отличие от матрицы 40000x40000.

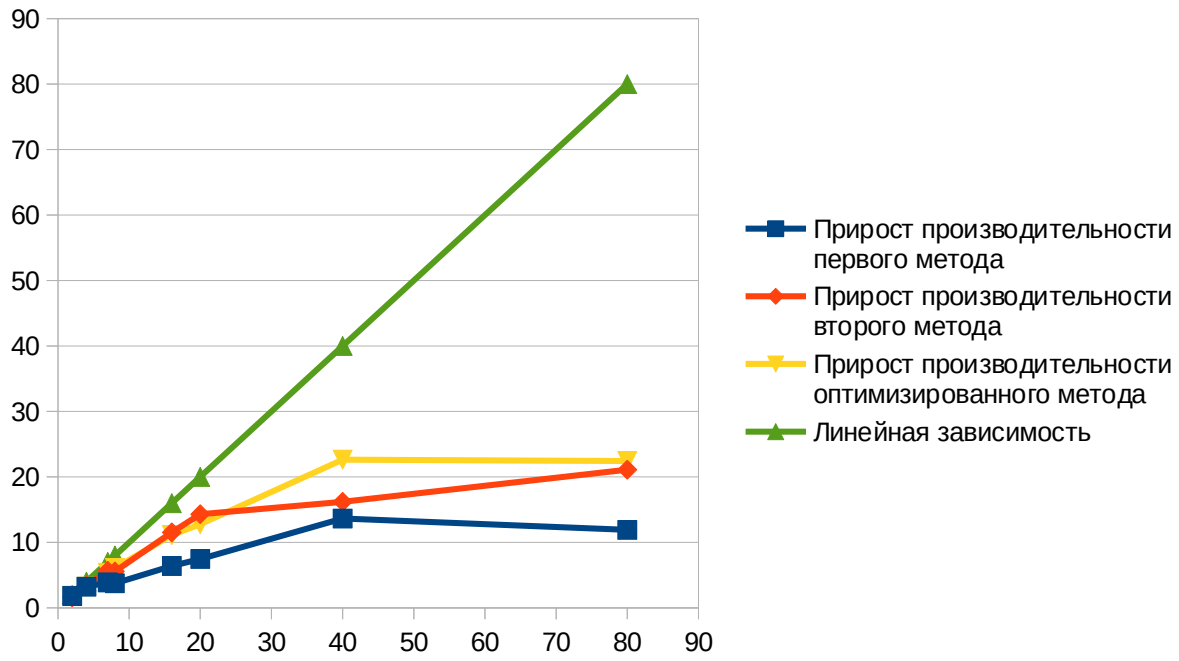
Количество потоков

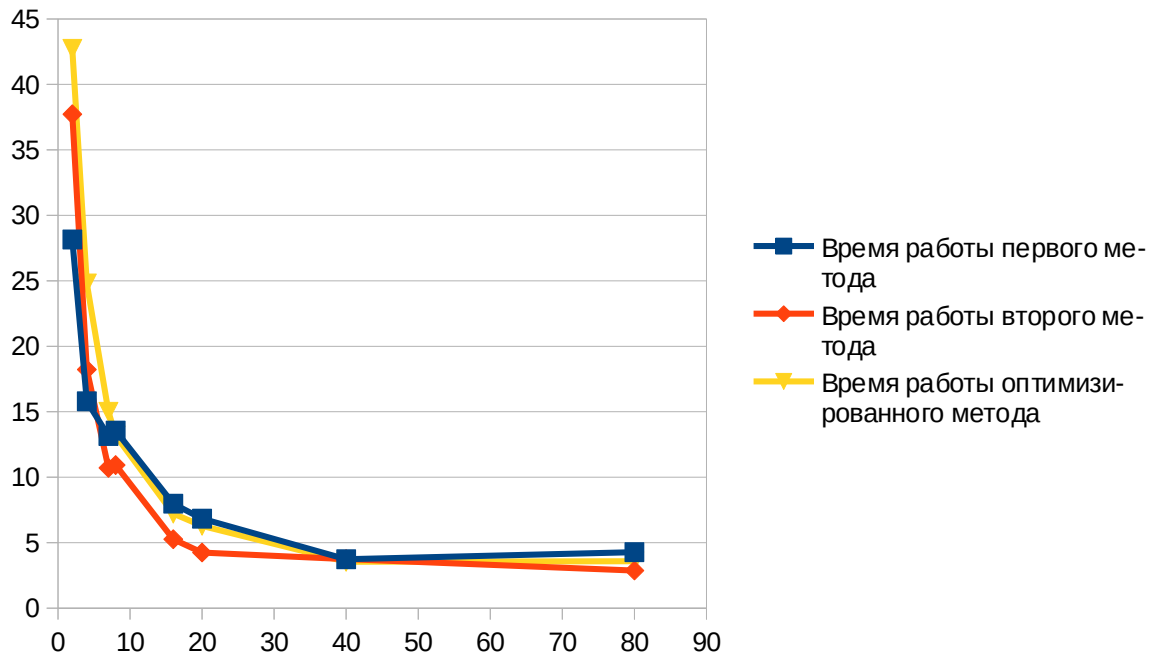
steps	2			4		7		8		16		20		40	
	T1	T2	S2	T4	S4	T7	S7	T8	S8	T16	S16	T20	S20	T40	S40
4 * 10^7	0.468	0.249	1.89	0.133	3.54	0.082	5.77	0.073	6.52	0.042	11.06	0.037	12.53	0.026	18.03



Распараллеливание на 2 — 16 потоков особо эффективно (практически линейно),
на 20 — 40 всё ещё эффективно, но прирост не настолько ощутим относительно 2 — 16.

	Количество потоков															
Ver.	2			4		7		8		16		20		40		80	
	T1	T2	S2	T4	S4	T7	S7	T8	S8	T16	S16	T20	S20	T40	S40	T80	S80
1	51.09	28.16	1.81	15.80	3.23	13.15	3.88	13.56	3.76	7.98	6.40	6.84	7.46	3.74	13.64	4.28	11.91
2	60.90	37.72	1.61	18.22	3.34	10.71	5.68	10.93	5.56	5.27	11.53	4.25	14.31	3.75	16.20	2.88	21.10
optimized	80.48	42.71	1.88	24.79	3.24	15.00	5.36	13.18	6.10	7.22	11.13	6.29	12.77	3.55	22.63	3.58	22.43





Весьма неоднозначные выводы получаются. С одной стороны, прирост производительности самый большой на оптимизированном через schedule алгоритме (optimized), затем идет второй алгоритм, и наконец первый.

Но если посмотреть на время выполнения, то побеждает второй алгоритм, затем идет optimized, и наконец первый (и то, смотря при каком количестве потоков). Но я не уверен что время в данном случае — адекватная мера, так как я пока я делал замеры времени первого и второго алгоритмов, сервер видимо каким-либо образом загрузился, и optimized алгоритм работал медленнее, чем оба предыдущих. Подождав некоторое время, я повторил эксперимент над optimized, и он начал работать с той скоростью, которая указана в данный момент в таблице. Поэтому возможно, что адекватнее будет обращать внимание именно на график прироста производительности, так как он относителен. Если следовать этой логике, то правильнее будет использовать оптимизированный через schedule алгоритм (ну или второй алгоритм, если выбирать из первых двух).