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:

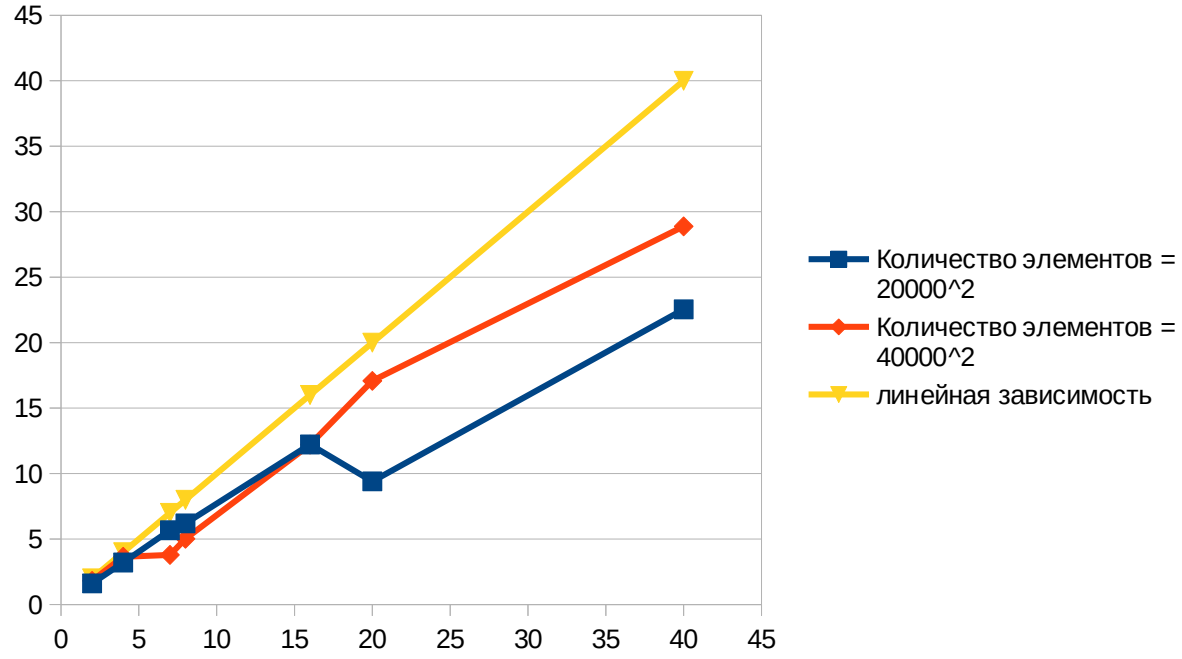
ProLiant XL270d Gen10

NUMA node:

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

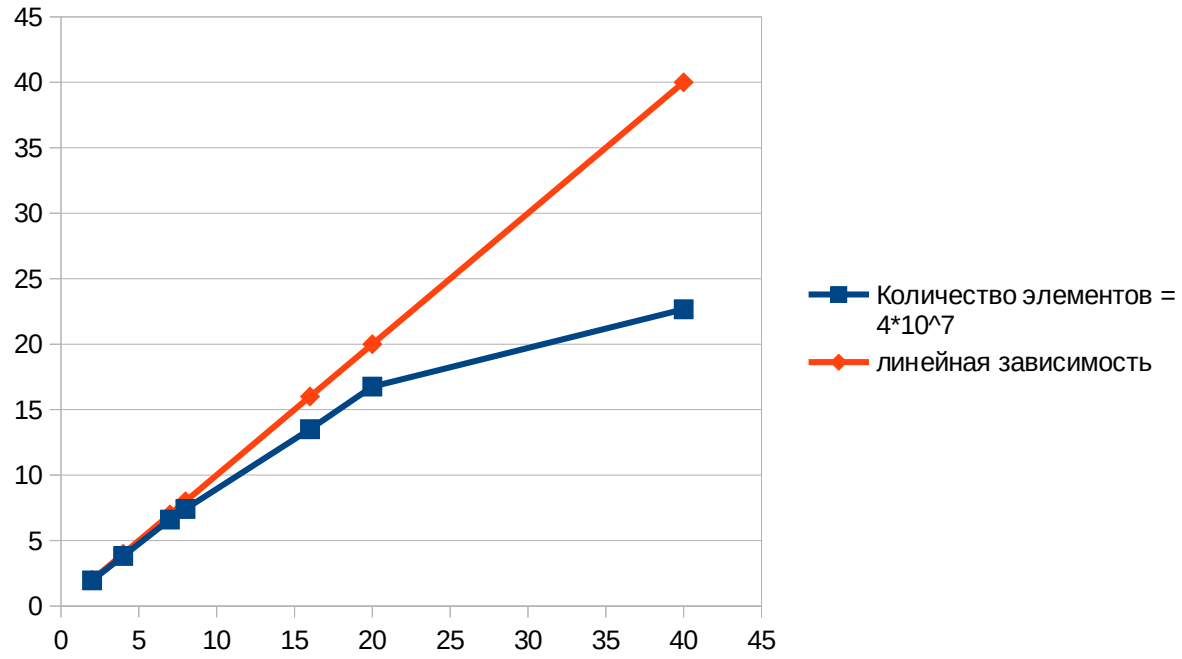
OS: Ubuntu 22.04.5 LTS

M=N	Количество потоков														
	2			4		7		8		16		20		40	
	T1	T2	S2	T4	S4	T7	S7	T8	S8	T16	S16	T20	S20	T40	S40
20000	1,080	0,670	1,611	0,336	3,206	0,190	5,670	0,174	6,196	0,088	12,225	0,114	9,408	0,047	22,54
40000	4,780	2,638	1,811	1,309	3,649	1,261	3,789	0,952	5,018	0,391	12,210	0,279	17,080	0,165	28,88



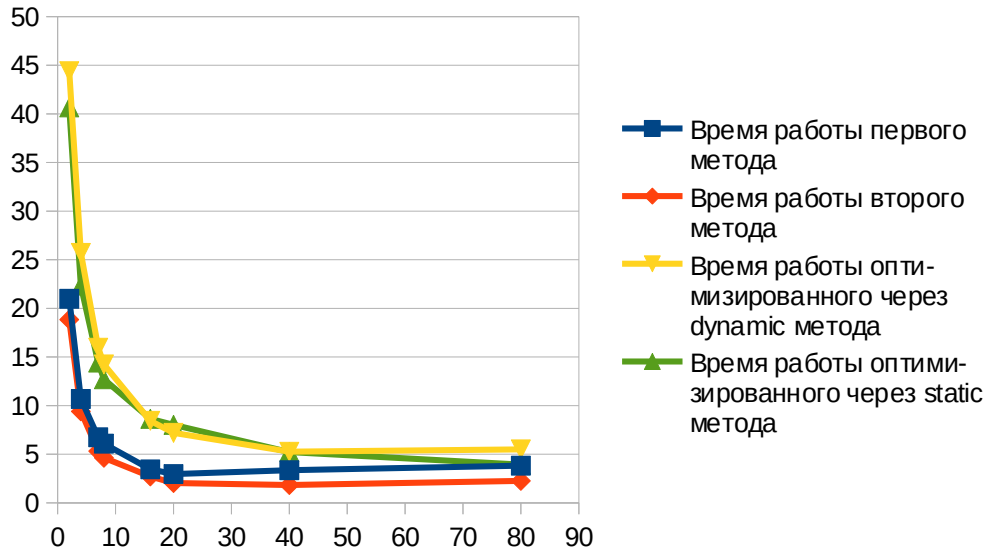
Распараллеливание до 16 потоков приносит стабильный окололинейный результат в обоих случаях. При распараллеливании больше чем на 16 потоков график эффективности от распараллеливания на матрице 20000x20000 растет не так быстро(на 20 потоках в принципе проседает), в отличие от матрицы 40000x40000, у которой график эффективности от распараллеливания продолжает расти и быть окололинейным.

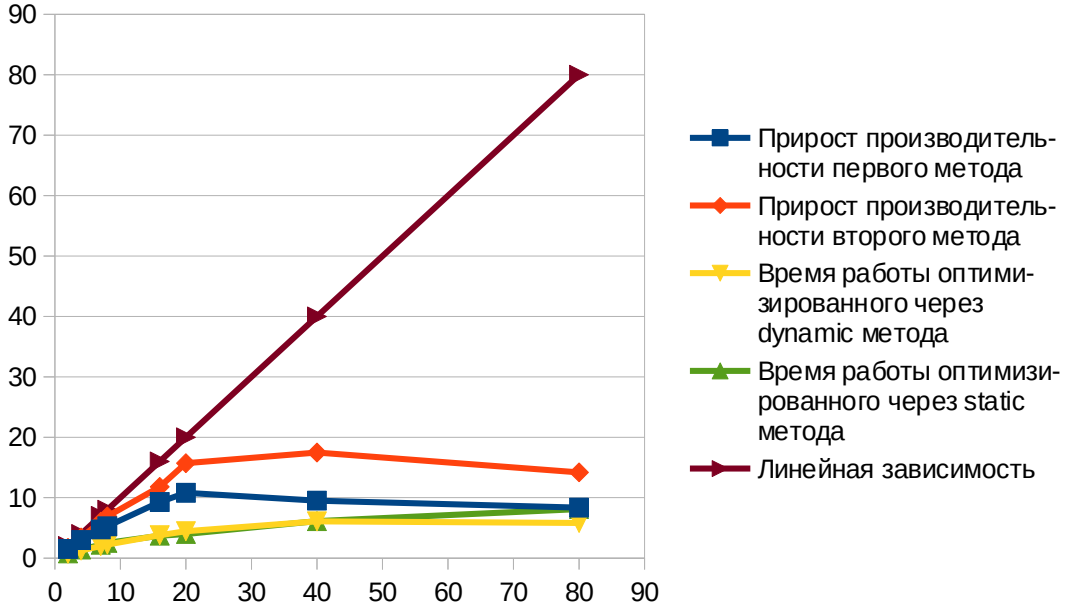
steps	Количество потоков														
	2			4		7		8		16		20		40	
	T1	T2	S2	T4	S4	T7	S7	T8	S8	T16	S16	T20	S20	T40	S40
4 * 10^7	0,476	0,243	1,960	0,124	3,826	0,072	6,607	0,064	7,416	0,035	13,506	0,028	16,763	0,021	22,66



Распараллеливание до 20 потоков приносит стабильный окололинейный результат. При распараллеливании больше чем на 20 потоков график эффективности от распараллеливания растет не так быстро.

Ver.	Количество потоков															
	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	32,02	20,98	1,525	10,68	2,997	6,744	4,748	6,091	5,257	3,448	9,285	2,964	10,80	3,365	9,514	3,826	8,369
2	32,02	18,85	1,698	9,405	3,405	5,324	6,014	4,636	6,907	2,712	11,80	2,038	15,71	1,832	17,48	2,255	14,20
dynamic	32,02	44,38	0,721	25,71	1,245	15,96	2,00	14,24	2,247	8,436	3,795	7,207	4,443	5,261	6,086	5,498	5,824
static	32,02	40,67	0,787	22,33	1,434	14,43	2,219	12,71	2,518	8,641	3,705	8,022	3,992	5,223	6,130	3,941	8,124





Итак, самый эффективный алгоритм — второй(вся программа в одном #pragma omp parallel).
Распараллеливание больше чем на 40 потоков не имеет смысла.
Очень странно, что «оптимизированные» алгоритмы (через schedule) не оптимизировали достаточно хорошо, чтобы догнать второй алгоритм.

Таким образом, самый эффективный вариант в данном случае — поместить весь цикл в одну #pragma omp parallel.