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:

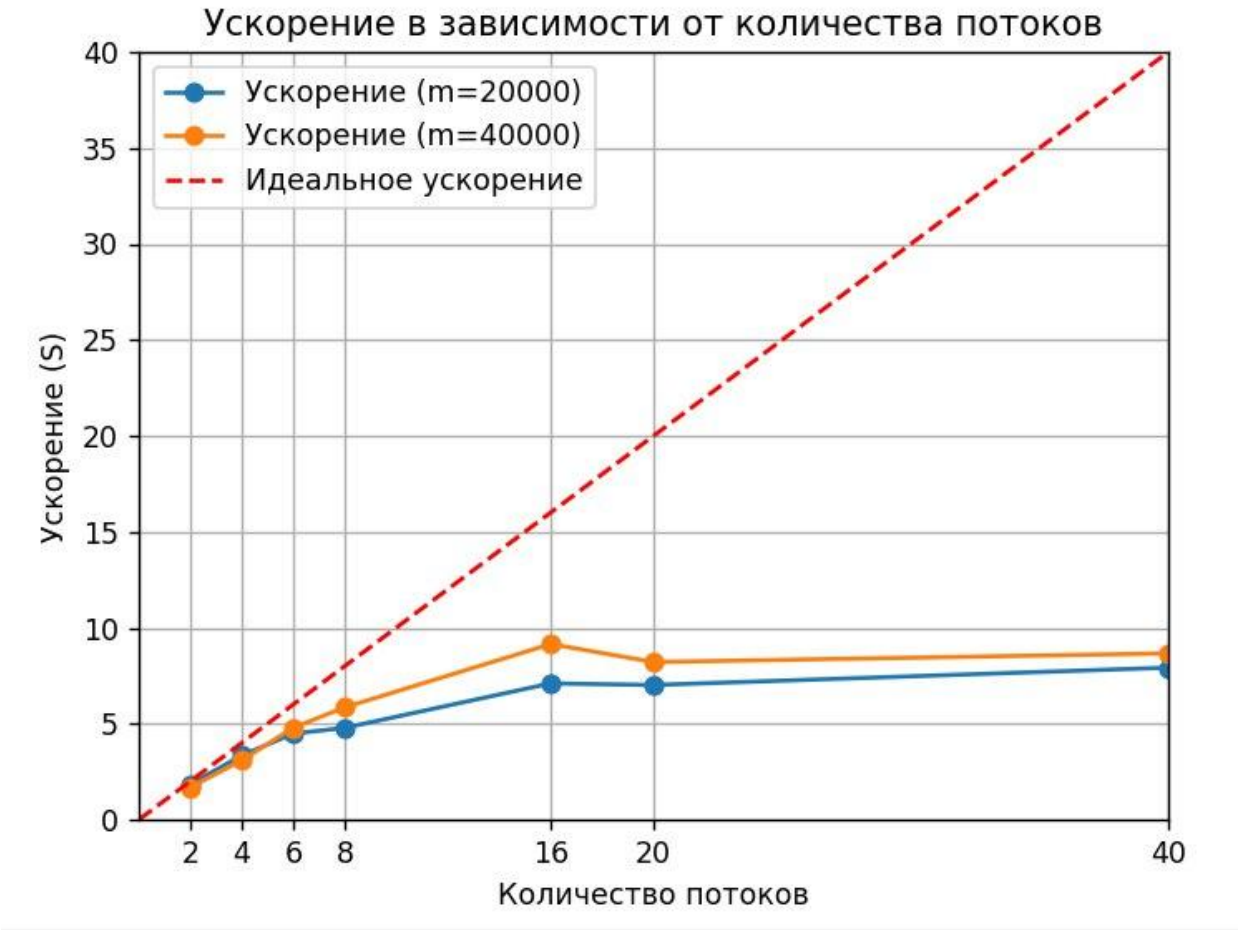
ProLiant XL270d Gen10

NUMA node:

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

OS: Ubuntu 22.04.5 LTS

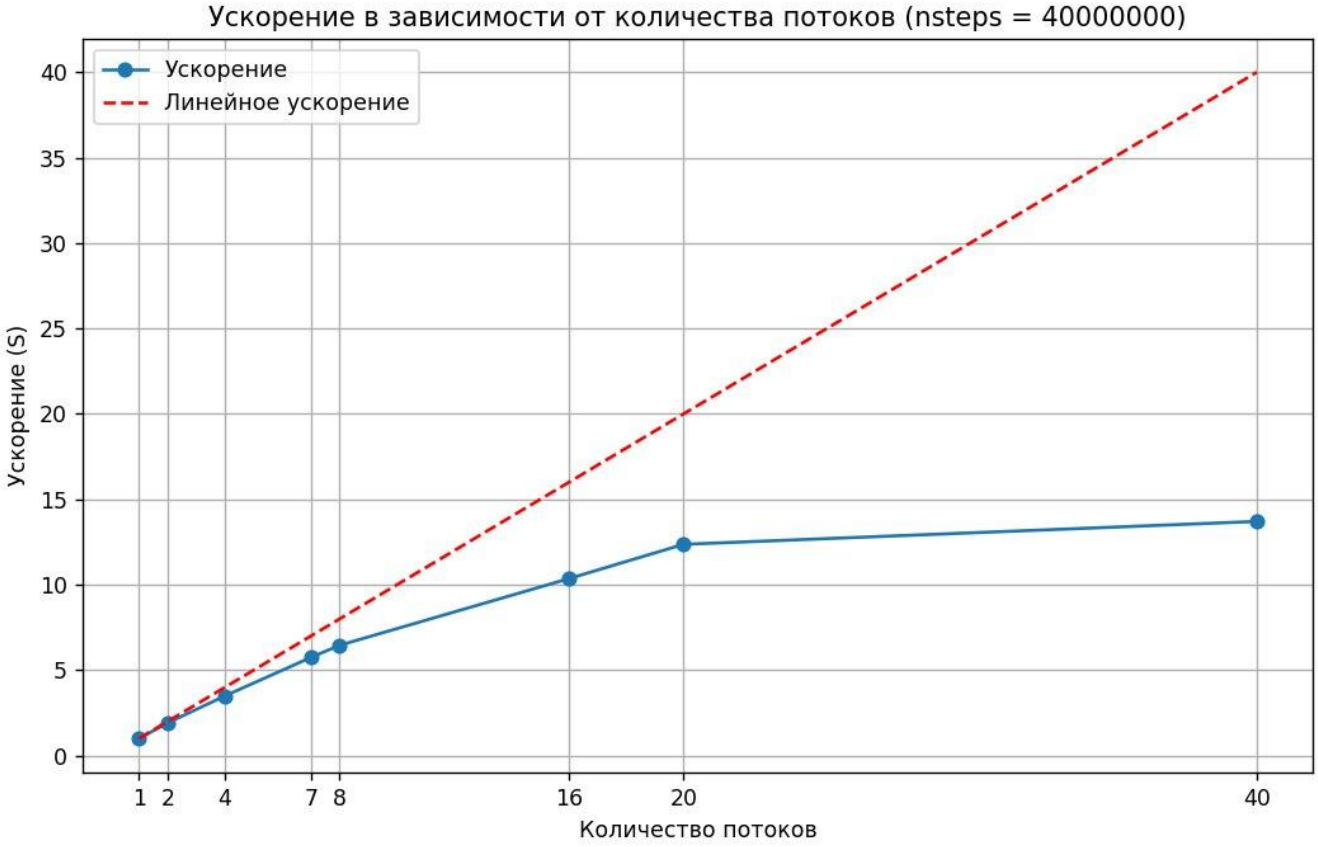
M=N	Количество потоков														
	2			4		6		8		16		20		40	
	T1	T2	S2	T4	S4	T6	S6	T8	S8	T16	S16	T20	S20	T40	S40
20000	4.214	2.285	1.844	1.259	3.346	0.939	4.484	0.880	4.783	0.593	7.100	0.700	7.017	0.531	7.922
40000	13.459	8.015	1.679	4.354	3.091	2.809	4.791	2.300	5.851	1.470	9.150	1.638	8.213	1.553	8.665



Распараллеливание до 16ого потока приносит стабильный окололинейный результат в обоих случаях. Однако после этого происходит некоторый спад и небольшие изменения что может быть связано с накладными расходами на управление потоками и конкуренцией за ресурсы

итог, данную параллельную программу можно адекватно масштабировать и она будет показывать хорошие показатели производительности относительно последовательной программы.

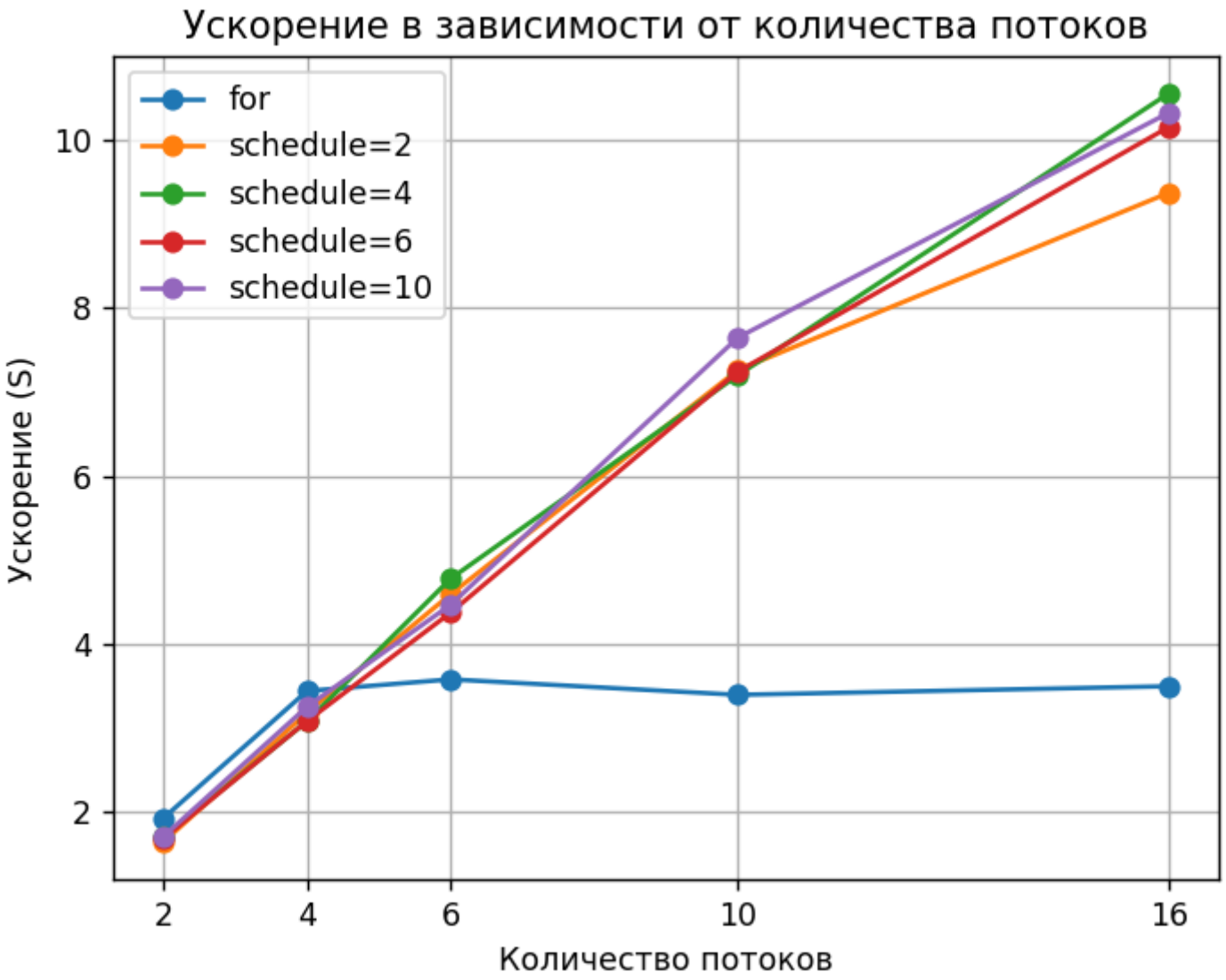
steps	Количество потоков														
	2			4		7		8		16		20		40	
	T1	T2	S2	T4	S4	T7	S7	T8	S8	T16	S16	T20	S20	T40	S40
4 * 10^7	0.455	0.239	1.900	0.130	3.488	0.079	5.709	0.071	6.363	0.043	10.477	0.036	12.304	0.033	13.70



Распараллеливание приносит стабильный окололинейный результат до 20 потоков включительно. При 40 потоках прирост не настолько сильный, но он все равно ощутим.

Соответственно, параллельную программу можно адекватно масштабировать и она будет показывать хорошие показатели производительности относительно последовательной программы.

Ver.	Количество потоков									
	2		4		6		10		16	
	T2	S2	T4	S4	T6	S6	T10	S10	T16	S16
for	29.072	1.930	16.300	3.442	15.66	3.580	16.51	3.396	16.03	3.498
shedule=2	22.863	1.646	11.825	3.183	8.196	4.593	5.186	7.260	4.018	9.368
shedule=4	23.302	1.705	12.896	3.081	8.314	4.778	5.520	7.198	3.763	10.55
shedule=6	23.622	1.687	12.921	3.084	9.110	4.374	5.502	7.242	3.925	10.15
shedule=10	23.270	1.712	12.243	3.254	8.916	4.468	5.210	7.646	3.860	10.32



Итак, самый эффективный алгоритм — второй(вся программа в одном #pragma omp parallel). Самый эффективный schedule 4.