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

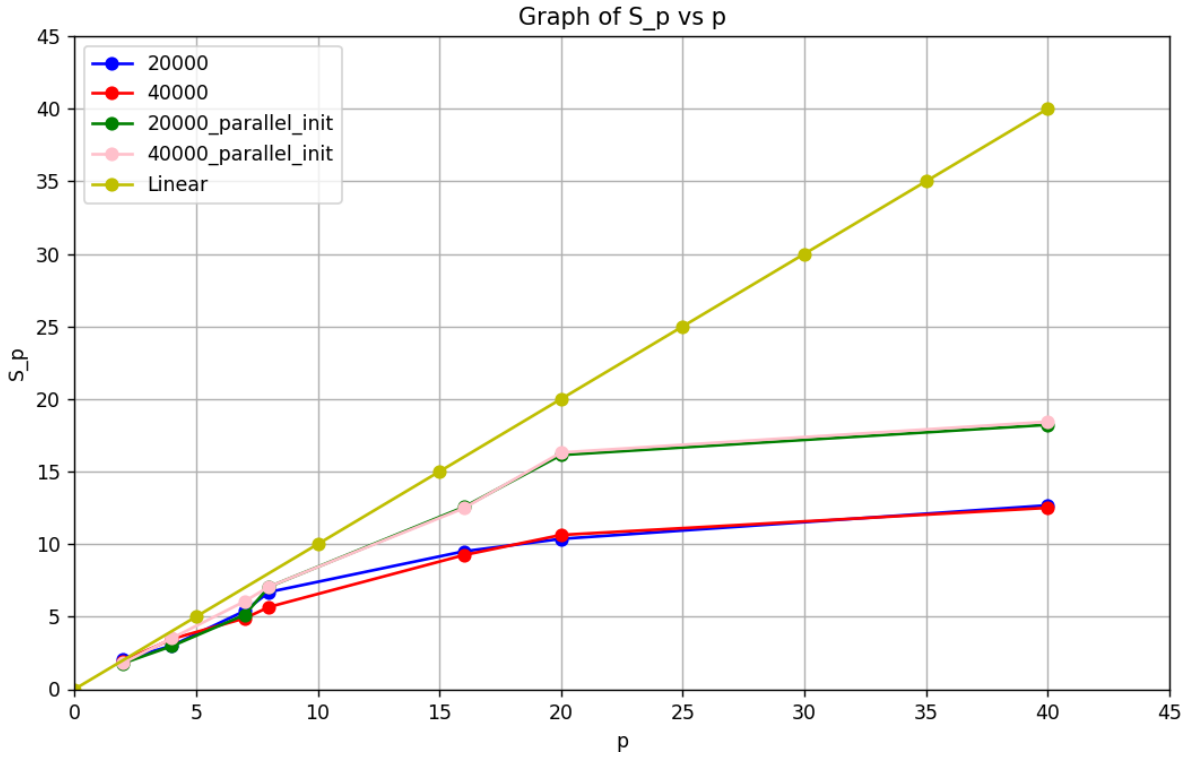
node 0 size: 385636 MB

node 1 size: 387008 MB

OS: Ubuntu 22.04.5 LTS

M=N	Количество потоков								
	2			4		7		8	
	T1	T2	S2	T4	S4	T7	S7	T8	S8
20000	1.14	0.56	2.03	0.38	3	0.21	5.4	0.15	6.70
40000	4.25	2.19	1.94	1.23	3.46	0.87	4.89	0.75	5.67
Parallel Init 40000	4.24	2.35	1.81	1.21	3.51	0.7	6.06	0.6	7.07
Parallel Init 20000	1.13	0.63	1.79	0.38	2.97	0.22	5.13	0.16	7.06

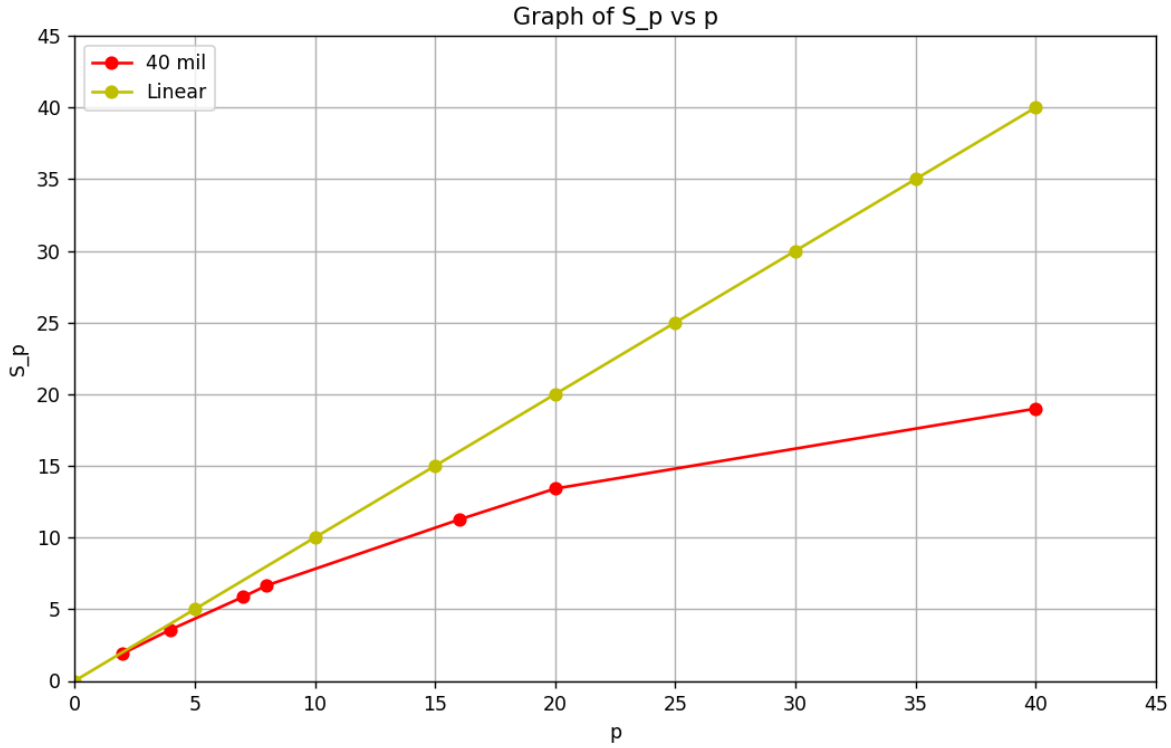
M=N	Количество потоков					
	16		20		40	
	T16	S16	T20	S20	T40	S40
20000	0.12	9.5	0.11	10.36	0.09	12.67
40000	0.46	9.24	0.40	10.63	0.34	12.5
Parallel Init 40000	0.34	12.47	0.26	16.31	0.23	18.43
Parallel Init 20000	0.09	12.55	0.07	16.14	0.062	18.22



На данном графике можно увидеть, что при небольшом количестве потоков (до 8 включительно) практически нет разницы между распараллеливанием только умножения матриц и распараллеливанием также и инициализации этих матриц, ускорение близко к линейному. Далее же распараллеливание инициализации начинает преобладать и дает ускорение примерно в полтора раза большее, чем без нее. В целом можно сказать, что если распараллеливать инициализацию, то ускорение будут окололинейно и данная программа будет показывать хороший результат относительно последовательной.

Steps	Количество потоков								
	2			4		7		8	
	T1	T2	S2	T4	S4	T7	S7	T8	S8
4E7	0.46	0.24	1.91	0.13	3.58	0.08	5.85	0.07	6.66

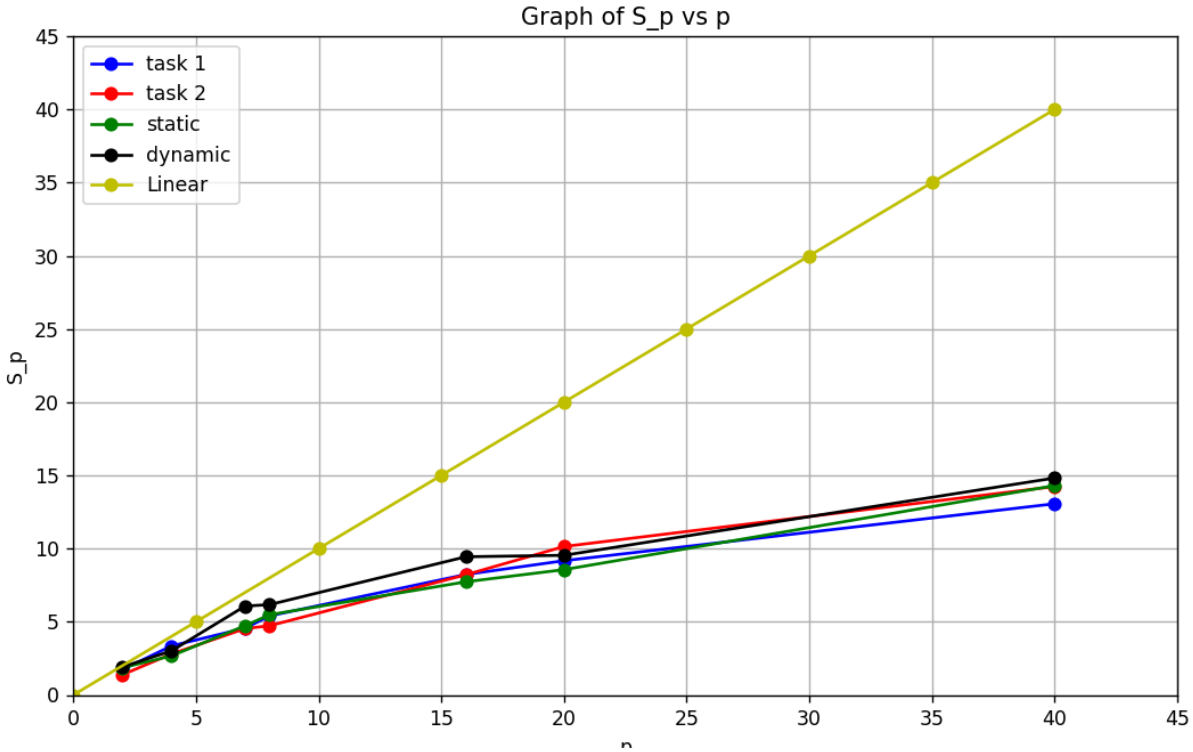
Steps	Количество потоков					
	16		20		40	
	T16	S16	T20	S20	T40	S40
4E7	0.04	11.25	0.03	13.41	0.02	18.99



Распараллеливание дает хороший окололинейный результат вплоть до 20 потоков. Далее прирост не такой яркий, но все еще ощутимый. В целом можно сказать, что данная программа будет показывать хороший результат относительно последовательной.

Способ	Количество потоков								
	2			4		7		8	
	T1	T2	S2	T4	S4	T7	S7	T8	S8
1 способ	36.33	20.99	1.73	10.85	3.35	7.93	4.56	6.73	5.4
2 способ	38.44	27.87	1.38	13.72	2.8	8.48	4.53	8.12	4.73
static	39.79	21.45	1.86	14.88	2.67	8.41	4.73	7.26	5.48
dynamic	37.95	19.92	1.91	12.62	3.01	6.25	6.07	6.15	6.17

Способ	Количество потоков					
	16		20		40	
	T16	S16	T20	S20	T40	S40
1 способ	4.41	8.23	3.96	9.17	2.78	13.06
2 способ	4.68	8.21	3.79	10.14	2.7	14.24
static	5.15	7.73	4.65	8.56	2.78	14.3
dynamic	4.02	9.44	3.98	9.54	2.56	14.82



Как можно заметить, если сравнивать 1 и 2 вариант программы, то быстрее будет работать 2(создается одна параллельная секция #pragma omp parallel, охватывающая весь итерационный алгоритм). Если провести исследование на определение оптимального schedule, то лучшим окажется schedule dynamic (за счет динамического распределения вычислительных ресурсов), тогда как schedule static практически не будет отличаться от варианта программы без schedule.