Описание сервера:

Наименование сервака - ProLiant XL270d Gen10
Архитектура - x86_64
Количество ядер CPU(s) - 80
Модель процессора - Intel(R) Xeon(R) Gold 6248 CPU @ 2.50GHz
Работает на частоте - 2.50GHz
Семейство процессора - 6
Потоков в ядре - 2
Ядер в узле - 20
Число узлов - 2
Max MHz - 3900
Min MHz - 1000

NUMA-узлы:
Число - 2
NUMA node0 - ядра 0-19 и 40-59
NUMA node0 - размер 385636 MB
NUMA node1 - ядра 20-39 и 60-79
NUMA node1 - размер 387008 MB

Операционка:
Название - Ubuntu
Версия - 22.04.5 LTS (Jammy Jellyfish)
ID=ubuntu
ID_LIKE=debian
UBUNTU_CODENAME=jammy

Задание 1

График:

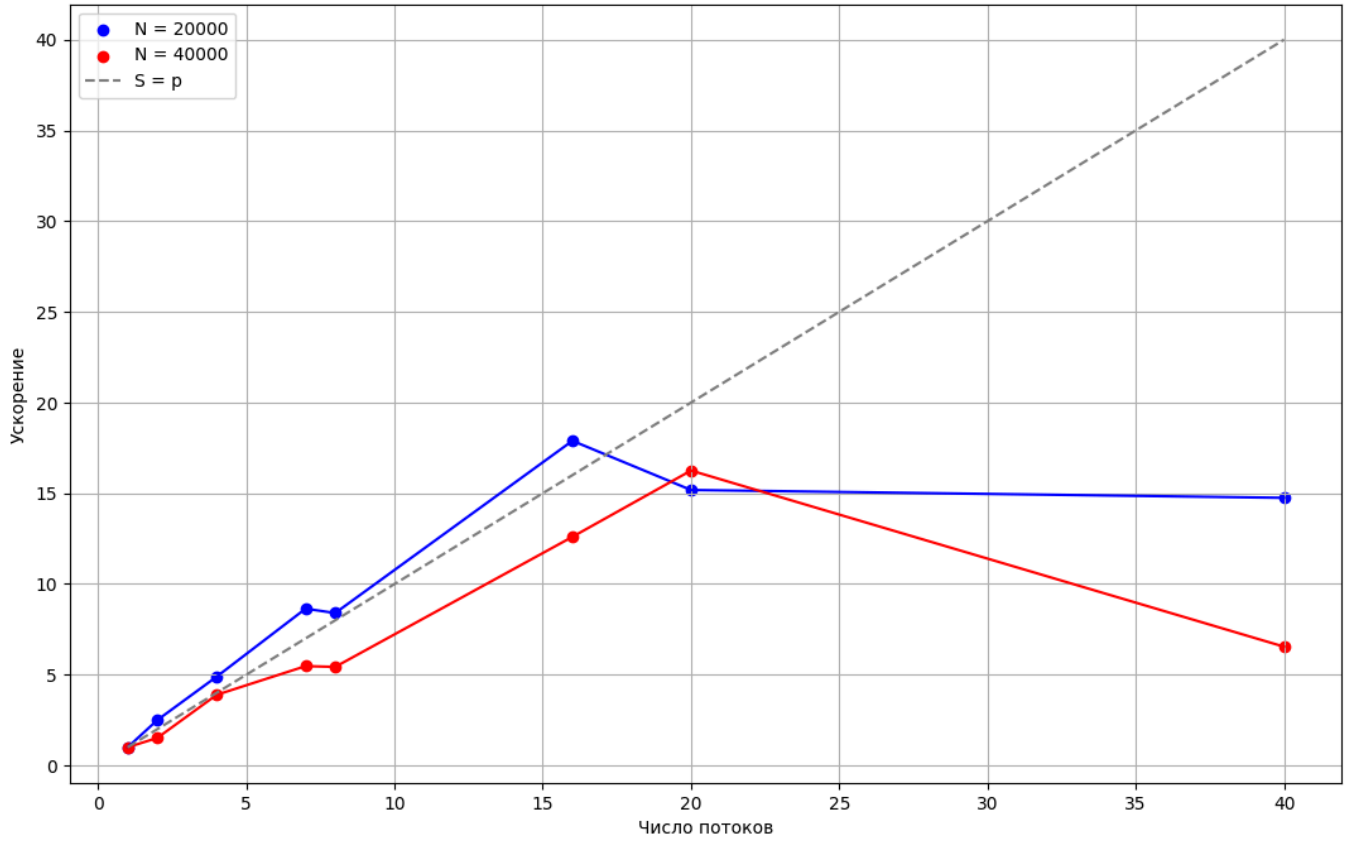


Таблица:

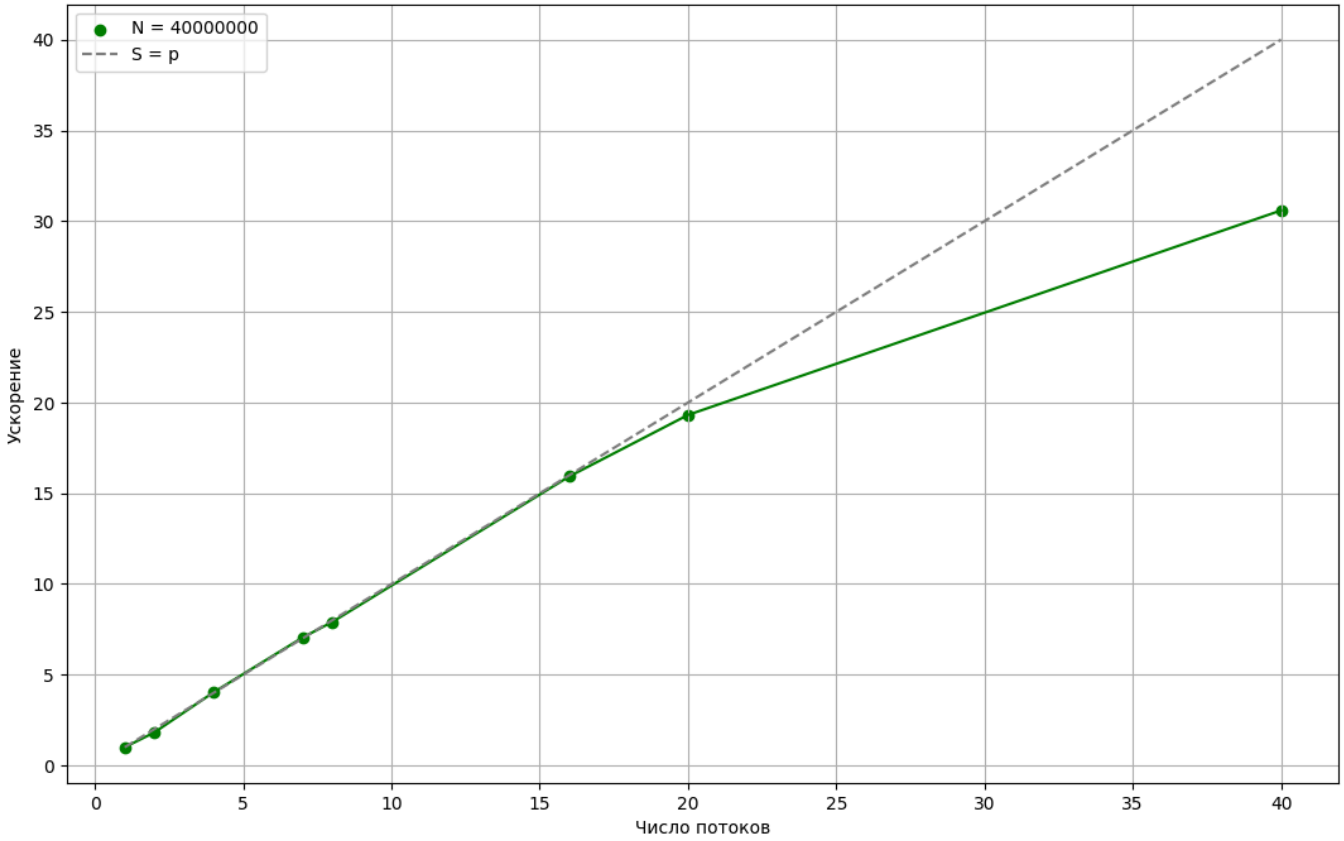
M = N	Количество потоков						
	2			4		7	
	Time (1)	Time (2)	Acceleration (2)	Time (4)	Acceleration (4)	Time (7)	Acceleration (7)
20 000	1.327 s.	0.531 s.	2.5	0.271 s.	4.89	0.154 s.	8.64
40 000	4.258 s.	2.809 s.	1.5	1.094 s.	3.89	0.778 s.	5.47

8		16		20		40	
Time (8)	Acceleration (8)	Time (16)	Acceleration (16)	Time (20)	Acceleration (20)	Time (40)	Acceleration (40)
0.158 s.	8.4	0.074 s.	17.9	0.087 s.	15.2	0.09 s.	14.75
0.784 s.	5.43	0.338 s.	12.6	0.262 s.	16.25	0.651 s.	6.54

Вывод: Ускорение находится около линейных значений приблизительно до пересечения границы в 20 потоков, после чего начинает стагнировать/падать, из чего следует заключение, что выделять число потоков, превышающее число ядер в узле, смысла нету.

Заниженность значений в случае с $N = 40000$ списываю на загруженность сервера

Задание 2



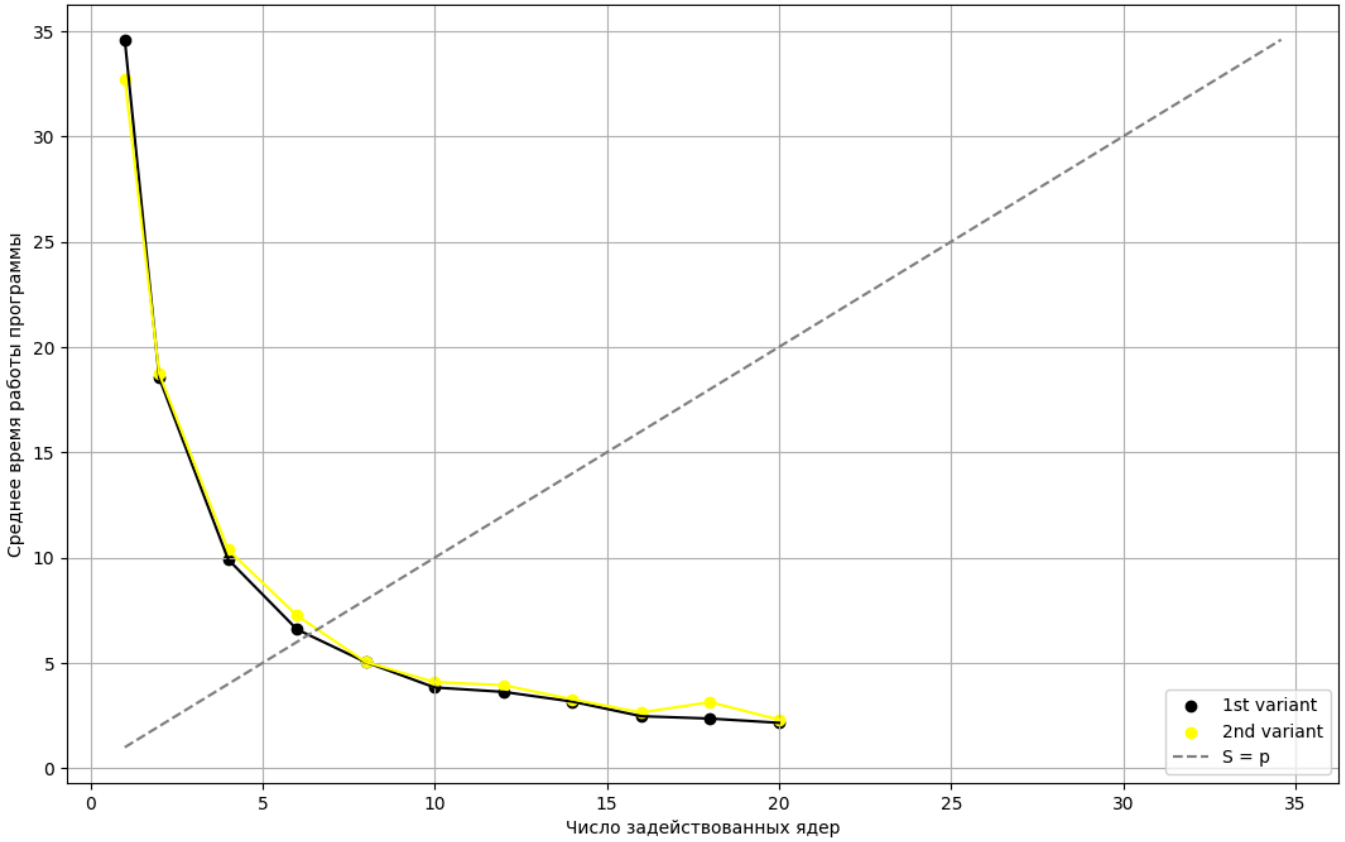
Вывод: Не отличается от того, который был сделать в первом задании, однако в данном случае то ли программа правильнее распараллелена, то ли сервак пустой был, но в любом случае значения ускорения практически идеально соответствуют линейным значениям.

Задание 3

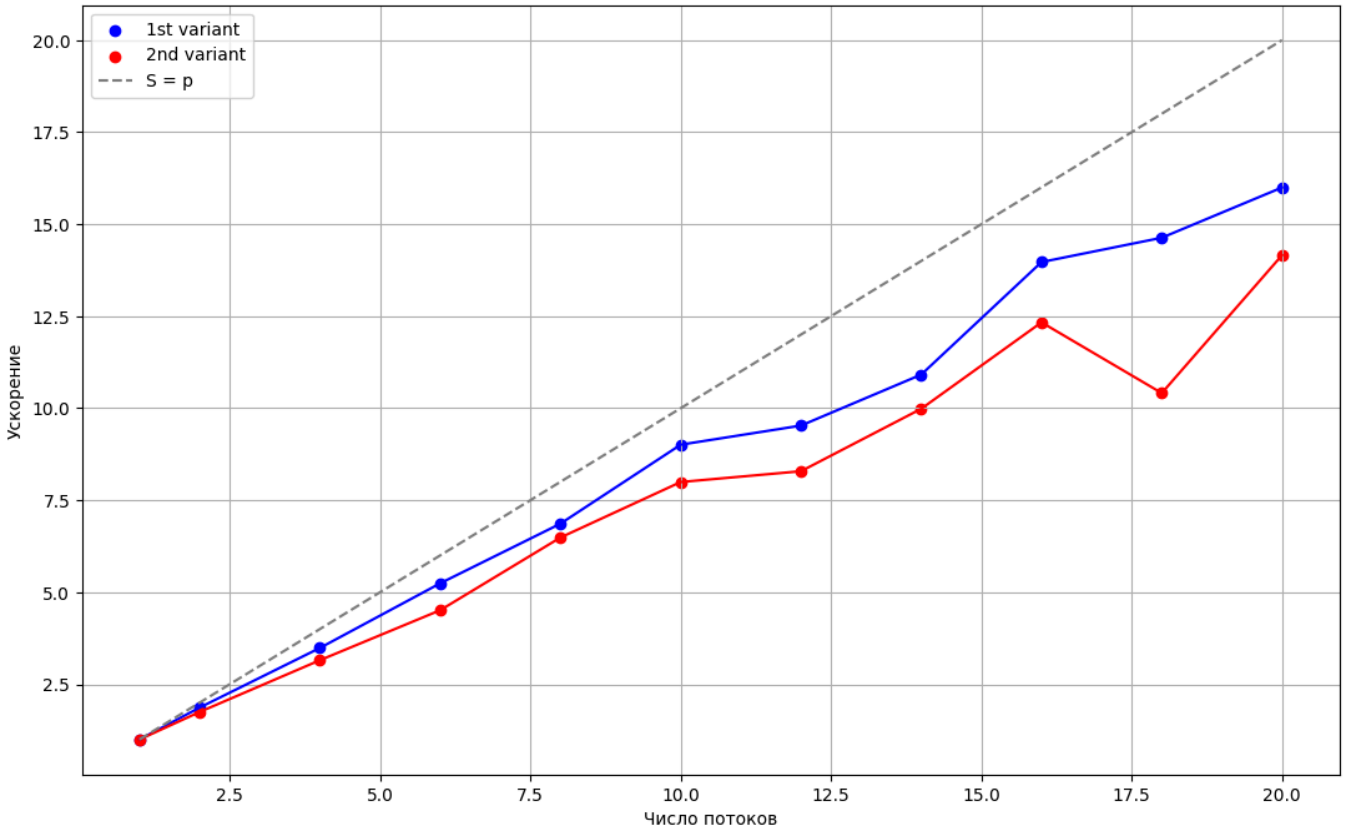
Время работы программ:

(позднее выяснил, что при построении графиков перепутал подпись в легенде, т.е. на самом деле 1st variant - это вариант с одной параллельной секцией, а 2nd variant - с

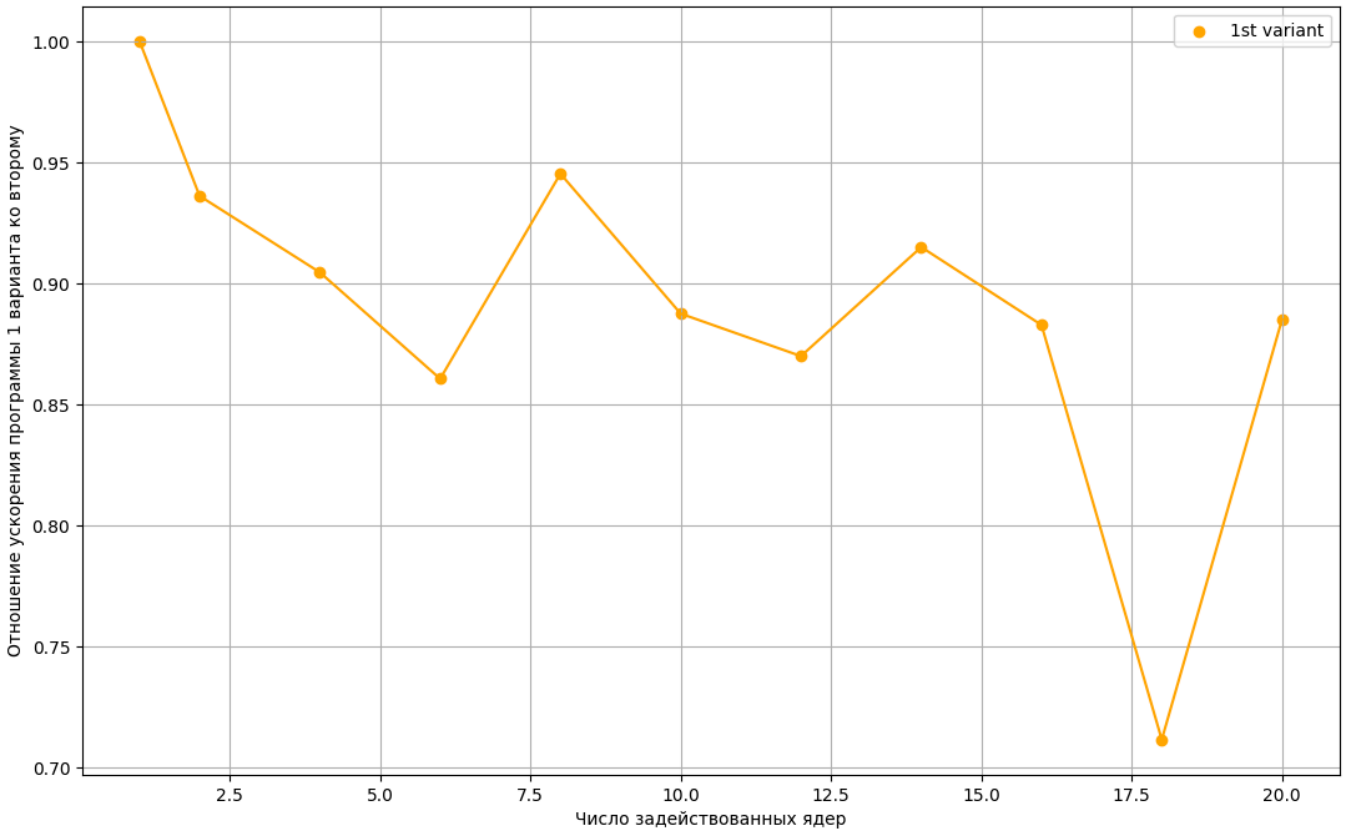
отдельной секцией на каждый цикл)



Ускорение работы программ:



Эффективность ускорения вторым методом относительно первого:



Вывод: Итак, оба варианта программы показывают +- одинаковый результат, однако первый вариант (т.е. с одной параллельной секцией) работает всё таки немного, но быстрее. Полагаю, что это можно списать на не очень правильное ~~мягко говоря~~ распараллеливание с моей стороны, поэтому если программу будет писать пряморукий человек, то время выполнения не изменится (хотя, быть может, я не прав, так как из-за того, что секция для распараллеливания одна, потоки не будут ждать друг-друга после завершения работы циклов, а значит первый вариант, по-идее, как раз и должен работать чуть быстрее).

Пункт 3 задания 3

Тут я юзал версию программы с одной параллельной секцией на 20 параллельных потоках.

Shchedule с типом dynamic отрабатывал невероятно долго (я так и не дождался результатов), думаю, что это происходит из-за того, что каждая итерация выполняет одинаковую по нагрузке "работу", и в этом случае static куда более оптимизирован. Результаты со static следующие:

N = 1000

Для 1 чанка время выполнения составило 8.92 секунды

Для 10 чанков время выполнения составило 4.08 секунд

Для 100 чанков время выполнения составило 1.95 секунд

Вывод: в условиях данной задачи использование static - оптимальный вариант.