ЗАДАНИЕ №4.2

Вариант 1. Умножение квадратных матриц.

Цель: Исследование направлено на сравнение времени работы программы, реализующей умножение квадратных матриц, в зависимости от способа умножения (с предварительным транспонированием и без него).

Исследование:

Для проведения исследования было реализовано 3 программы (умножение матриц без транспонирования, умножение матриц с предварительным транспонированием обеих и умножение матриц с предварительным транспонированием одной из них), каждую из которых я компилировал трижды (первый раз с O0-уровнем оптимизации, второй раз с O3-уровнем оптимизации и третий раз с Os-уровнем оптимизации). Матрицы генерировались непосредственно в самих программах. Также были написаны следующие скрипты:

1. Скрипт *build_apps.sh* компилирует каждую из программ трижды (сначала с уровнем оптимизации O0, потом с O3, потом с Os). На выходе получается 9 исполняемых файлов, расположенных в каталоге *./apps*, которые передаются скрипту *create_data.sh*

2. Скрипт *create_data.sh* запускает эти 9 исполняемых файлов (в качестве аргументов скрипт принимает шаг исследования STEP, максимальный размер матриц MAXSIZE и количество замеров для умножения каждым способом EXP) и записывает результаты в 9 текстовых файлов, располагающихся в каталоге *./measure*, предварительно очищая его от прошлых измерений (как мне кажется, нет необходимости хранить прошлые замеры, потому что условия в которых проходит эксперимент могут поменяться (например, производительность процессора упала из-за открытия дополнительных окон), соответственно могут и измениться новые результаты)

3. Скрипт *make_preproc.sh* получает на вход файлы, обработанные предыдущим скриптом, и запускает питоновский скрипт *make_preproc.py,* который обрабатывает всю информацию из файлов (находит максимум, минимум, среднее, RSE и т.д) и записывает эту информацию в одноименные файлы, расположенные в каталоге *measure/reses.* По информации, записанной в этих файлах, я строил все таблицы.

4. Скрипт *make_postproc.sh* получает на входе эти самые обработанные файлы с информацией и запускает питоновский скрипт *make_postproc.py,* который в свою очередь по обработанной информации строит все графики с помощью matplotlib`a

5. Скрипт *research.sh* принимает три аргумента шаг исследования STEP, максимальный размер массива MAXSIZE и количество замеров EXP и по очереди запускает все предыдущие скрипты, передавая им при необходимости аргументы STEP, MAXSIZE и EXP.

Результаты:

Результаты получены посредством выполнения команды:

./research.sh 20 200 2000

Далее в таблицах T — среднее время работы сортировки, RSE — относительная стандартная ошибка среднего, Кол-во — количество замеров

1. Уровень оптимизации O0

	С транспонирование обеих			С транспонированием одной			Без транспонирования		
Размер	T, мс	Кол-во	RSE,%	T, мс	Кол-во	RSE,%	T, мс	Кол-во	RSE,%
1	0	2000	3.37	0	2000	2.19	0	2000	3.03
21	0.02	2000	0.65	0.02	2000	0.37	0.02	2000	0.43
41	0.13	2000	0.16	0.14	2000	0.13	0.14	2000	0.15
61	0.43	2000	0.10	0.45	2000	0.09	0.45	2000	0.10
81	1.18	2000	0.49	1.29	2000	0.45	1.24	2000	0.43
101	2.03	2000	0.32	2.24	2000	0.42	2.28	2000	0.38

121	3.77	2000	0.49	3.74	2000	0.34	3.93	2000	0.41
141	5.67	2000	0.43	6.02	2000	0.45	6.29	2000	0.44
161	8.52	2000	0.48	8.69	2000	0.39	9.04	2000	0.40
181	11.85	2000	0.44	12.45	2000	0.41	13.10	2000	0.43
201	16.55	2000	0.67	17.29	2000	0.60	17.84	2000	0.59

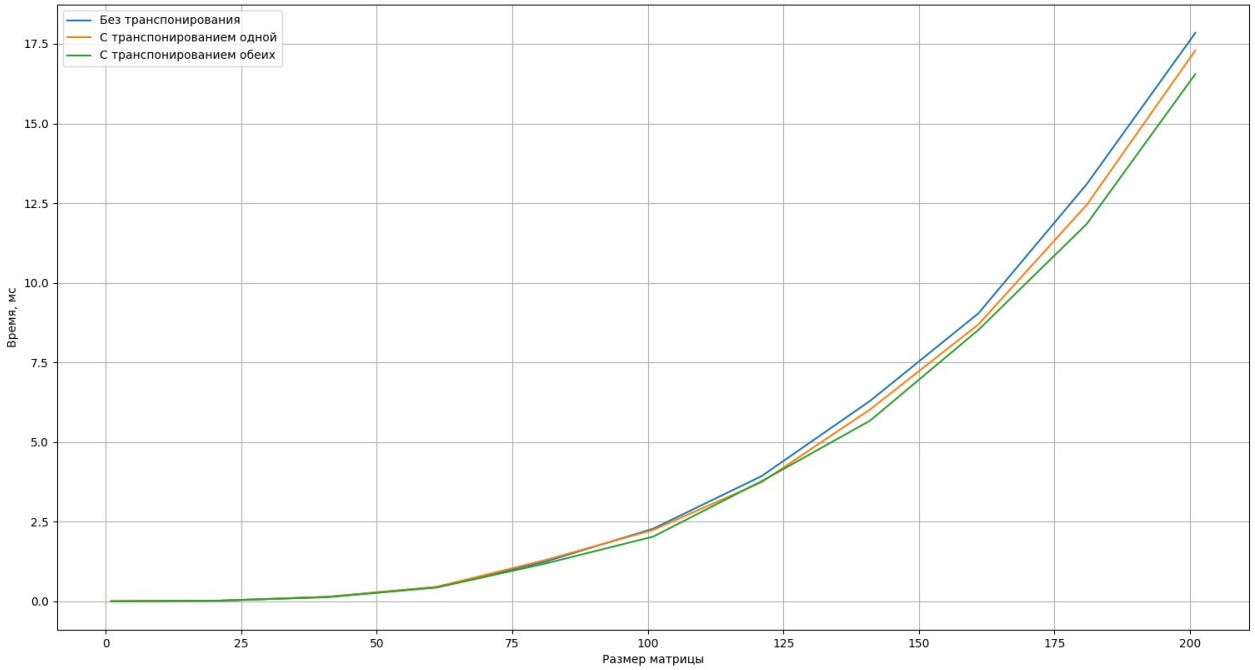

График с уровнем оптимизации O0

2. Уровень оптимизации O3

	С транспонирование обеих			С транспонированием одной			Без транспонирования		
Размер	T, мс	Кол-во	RSE,%	T, мс	Кол-во	RSE,%	T, мс	Кол-во	RSE,%
1	0	2000	4.42	0	2000	1.84	0	2000	2.93
21	0	2000	1.22	0	2000	0.62	0	2000	0.95
41	0.02	2000	0.85	0.02	2000	0.21	0.02	2000	0.29
61	0.06	2000	0.67	0.06	2000	0.38	0.06	2000	0.12
81	0.14	2000	0.57	0.13	2000	0.18	0.15	2000	0.09
101	0.27	2000	0.45	0.25	2000	0.16	0.30	2000	0.06
121	0.44	2000	0.29	0.49	2000	0.50	0.54	2000	0.07
141	0.90	2000	0.54	0.83	2000	0.61	1.04	2000	0.44

161	1.03	2000	0.34	1.01	2000	0.27	1.34	2000	0.28
181	1.60	2000	0.51	1.63	2000	0.51	1.99	2000	0.43
201	1.88	2000	0.09	1.91	2000	0.33	2.53	2000	0.35

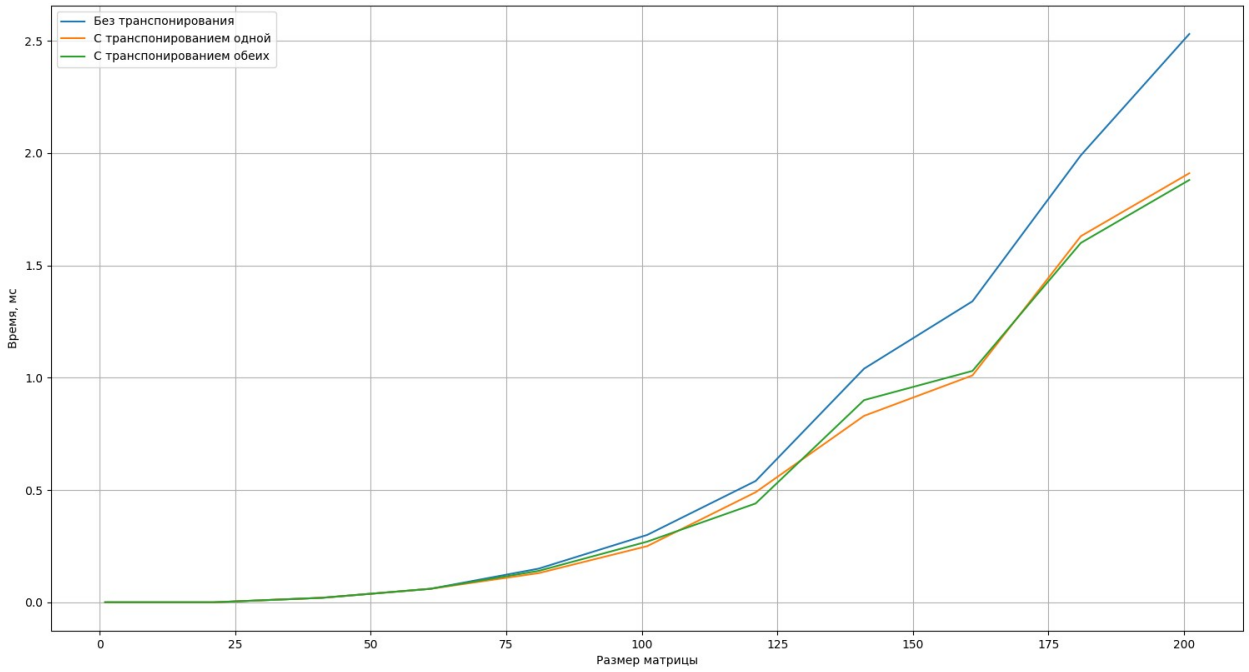

График с уровнем оптимизации O3

2. Уровень оптимизации Os

	С транспонирование обеих			С транспонированием одной			Без транспонирования		
Размер	T, мс	Кол-во	RSE,%	T, мс	Кол-во	RSE,%	T, мс	Кол-во	RSE,%
1	0	2000	0.54	0	2000	3.09	0	2000	2.18
21	0.01	2000	0.31	0.01	2000	0.61	0.01	2000	0.56
41	0.06	2000	0.12	0.04	2000	0.18	0.04	2000	0.17
61	0.18	2000	0.10	0.13	2000	0.11	0.12	2000	0.15
81	0.35	2000	0.42	0.34	2000	0.12	0.31	2000	0.13
101	0.62	2000	0.05	0.62	2000	0.05	0.63	2000	0.37
121	1.08	2000	0.33	1.23	2000	0.45	1.19	2000	0.47
141	1.74	2000	0.39	1.60	2000	0.06	1.74	2000	0.45
161	2.51	2000	0.41	2.54	2000	0.38	2.55	2000	0.44
181	3.64	2000	0.44	3.69	2000	0.42	3.49	2000	0.38

201	4.77	2000	0.56	4.85	2000	0.55	4.88	2000	0.61

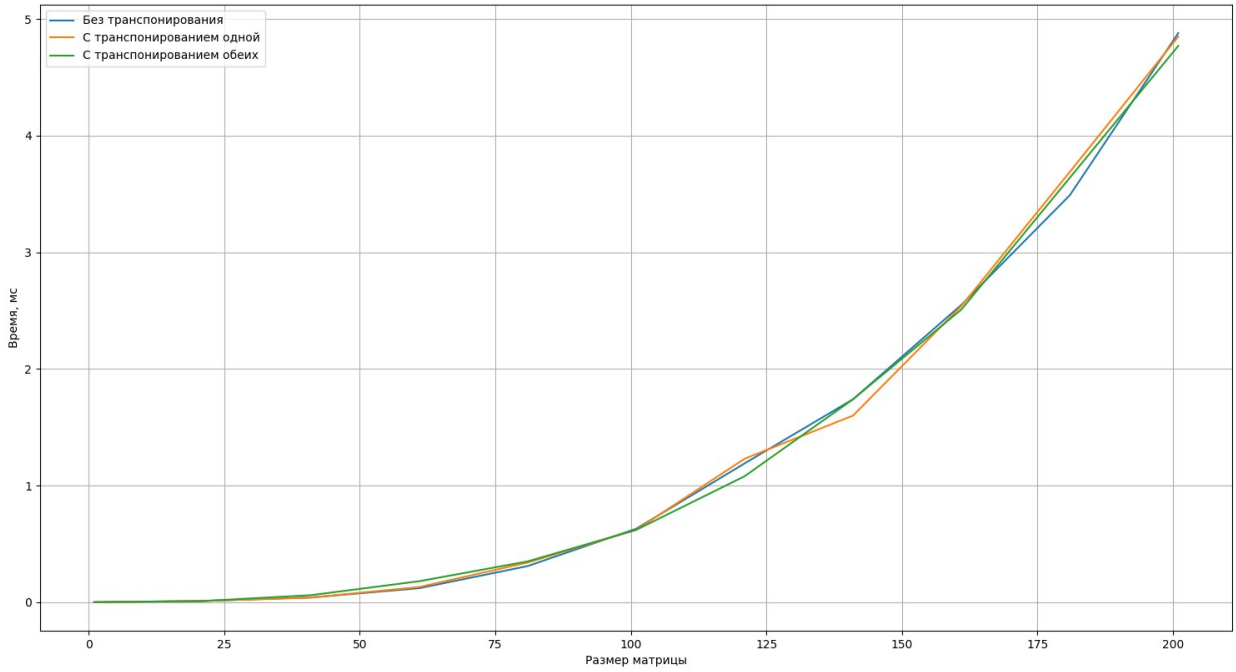

График с уровнем оптимизации Os

Уровень оптимизации O3

	С транспонирование обеих			С транспонированием одной			Без транспонирования		
Размер	T, мс	LOG	RSE,%	T, мс	LOG	RSE,%	T, мс	LOG	RSE,%
1	0	0.61	4.42	0	0.61	1.84	0	0.51	2.93
21	0	2.42	1.22	0	2.32	0.62	0	2.75	0.95
41	0.02	2.80	0.85	0.02	2.81	0.21	0.02	3.08	0.29
61	0.06	2.96	0.67	0.06	3.00	0.38	0.06	3.25	0.12
81	0.14	2.85	0.57	0.13	2.97	0.18	0.15	3.08	0.09
101	0.27	2.73	0.45	0.25	3.78	0.16	0.30	3.23	0.06
121	0.44	4.72	0.29	0.49	3.36	0.50	0.54	4.26	0.07
141	0.90	1.02	0.54	0.83	1.51	0.61	1.04	1.91	0.44
161	1.03	3.71	0.34	1.01	4.09	0.27	1.34	3.38	0.28
181	1.60	1.53	0.51	1.63	1.51	0.51	1.99	2.27	0.43
201	1.88	-	0.09	1.91	-	0.33	2.53	-	0.35

Общий вывод: Как можно заметить, реализации с транспонированием быстрее реализации без транспонирования. Причем самым быстрым является реализация с транспонированием обеих матриц, потом идет реализация с транспонированием одной, а потом уже реализация без транспонирования. Казалось бы, почему? Ведь при реализации без транспонирования меньше всего операций, а при реализации с транспонированием обеих — больше всего, но на деле меньше операций, как оказалось, дольше работает. Весь фокус в том, что в Си двумерный массив представляется в кэше построчно. При реализации без транспонирования компилятору приходится перемещать память туда-сюда: сначала загрузить в кэш одну строку матрицы, потом другую строку матрицы. При реализации с транспонированием компилятор не перезагружает строку в кэш. Доступ к элементам осуществляется в рамках одной строки, соответственно это менее ресурсозатратно для компилятора. Отсюда и более быстрая работа с предварительным транспонированием.