Способы измерения времени

Цель: определить самый точный способ измерения времени

Рабочая среда:
build_measure.sh – скрипт собирает исполняемый файл для взятия измерений
clean.sh – скрипт очищает рабочую среду

Таблицы:

gettimeofday

Величина задержки, мс	Результат измерений, мс	Количество повторов	RSE, %	Абсолютная погрешность, мс	Относительная ошибка, %
1000	1009,33	3	0,09	9,33	0,93
100	109,67	3	0,80	9,67	9,67
50	60,75	4	4,27	10,75	21,50
10	16,57	42	4,98	6,57	65,71

clock_gettime

Величина задержки, мс	Результат измерений, мс	Количество повторов	RSE, %	Абсолютная погрешность, мс	Относительная ошибка, %
1000	1004,33	3	0,32	4,33	0,43
100	106,00	3	4,25	6,00	6,00
50	61,00	3	3,28	11,00	22,00
10	14,00	3	4,12	4,00	40,00

clock

Величина задержки, мс	Результат измерений, мс	Количество повторов	RSE, %	Абсолютная погрешность, мс	Относительная ошибка, %
1000	1009,33	3	0,29	9,33	0,93
100	110,00	3	0,91	10,00	10,00
50	66,25	4	4,94	16,25	32,50
10	16,36	11	4,89	6,36	63,64

rdtsc

Величина задержки, мс	Результат измерений, мс	Количество повторов	RSE, %
1000	2710227147,33	3	0,28
100	297484054,67	3	1,52
50	166662054,00	3	2,01
10	40688033,33	3	1,39

Вывод: Самый точный способ **rdtsc**, т.к.
измерения не зависят от вариаций в частоте процессора и выполняется за один такт процессора без прерываний или задержек. Также таблицы показывают нам, что **rdtsc** возвращает измерение с наименьшим процентом относительной стандартной ошибкой среднего, за наименьшее количество замеров.

Сортировка одномерных массивов

Цель: узнать самый быстрый способ работы с одномерным массивом, с помощью сортировки

Способ сортировки: сортировка пузырьком с флагом
Количество запусков: 200

Способы работы с массивом:

- **address_sort** – обращение к элементу массива с использованием адресной арифметики
- **pointer_sort** – обращение к элементу массива по указателю
- **index_sort** – обращение к элементу массив по индексу

Рабочая среда:
apps – каталог с собранными исполняемыми файлы. Размеры создаваемых массивов передаются в качестве аргумента

measures – каталог, который содержит собранные измерения. Он организован следующим образом: есть каталоги, содержащие различные способы обработки элементов массива. Внутри каждого из этих каталогов находятся подкаталоги, представляющие различные виды массивов. В этих подкаталогах хранятся каталоги с различными уровнями оптимизации. И внутри этих каталогов находятся файлы, названия которых указывают на длины массивов и содержат замеры времени выполнения. Также в каталоге есть файл **runs.txt**, который хранит информацию о количестве проведенных измерений.

progs – каталог с исходными кодами программ

data_work – каталог работы с данными
data_work/data – каталог с замеряемыми данными. Каталог храни в себе: файл **optimizations** с уровнями оптимизаций, каталог **arrs**, который хранит в себе файлы с видами массивов, в которых записаны измеряемые размеры массивов
data_work/calced_data – каталог с измеренными данными. Содержит в себе текстовые файлы, которые хранят в себе размеры измеряемых массивов, среднее время, минимально и максимальное время, верхний, нижний и средний квартили. Данные распределены в текстовых файлах распределены по виду массива, уровню оптимизации и способу обращения к элементам массива
data_work/build_apps.sh – скрипт собирает исполняемые файлы и помещает их в каталог **./apps**. Скрипт собирает исполняемые файлы по разным видам обращения к элементам матрицы, уровням оптимизации, и видам массивов. Также скрипт определяет макрос максимальной длины массива при компиляции, которая

передается в скрипт как аргумент (при отсутствии аргумента стандартное значение максимальной длины массива равно 10005) и определяет макрос генерации массива (RAND – генерация случайного массива, SORT – генерация отсортированного массива)
data_work/update_data.sh – скрипт добавление новых замеров в каталог **./measures**. Скрипт создает нужные каталоги при их отсутствии и записывает в них определенное количество замеров, которые передаются в скрипт как аргумент(при отсутствии аргумента стандартное количество замеров равно 10)
data_work/clean_data.sh – скрипт удаления измерений из каталога **./measures**
data_work/make_preproc.py – программа считает среднее значение измерений времени, минимальное и максимальное значения времени, верхний, средний и нижний квартили для каждой измеряемой длины массива, вида массива, уровня оптимизации и способа обращения к элементам массива. Программа записывает полученные измерения в файлы расположенные в каталоге **./data_work/calced_data**
data_work/make_postproc.sh – скрипт отрисовывает графики по скриптам описанным в каталоге **./data_work/graph_draw**. Графики строятся по данным, которые находятся в каталоге **./data_work/calced_data**
data_work/go.sh – скрипт запускает всю систему работы с данными
data_work/graph_draw – каталог хранит в себе все скрипты для заданных графиков
* **data_work/table_create.py** – программа чертит таблицы с относительной стандартной ошибкой среднего для каждой длины массива. Таблицы чертятся для каждого вида массива, уровня оптимизации и способа обращения к элементам массива.

Таблицы:

address_sort_O0

	sorted			random		
размер массива	t, мкс	кол-во повторов	RSE, %	t, мкс	кол-во повторов	RSE, %
500	19236,56	200	1,56	18151,82	200	1,40
1000	35372,92	200	1,08	40500,67	200	1,44
1500	50434,79	200	0,99	52269,95	200	1,26
2000	63988,14	200	0,97	65867,34	200	1,00
2500	73442,13	200	0,79	88717,41	200	1,76
3000	84270,62	200	0,76	112308,35	200	1,39
3500	94301,94	200	0,77	106866,40	200	1,06
4000	101278,84	200	0,80	120416,27	200	1,13
4500	98267,19	200	0,47	139565,84	200	1,06
5000	103972,57	200	0,57	146822,23	200	1,12
5500	130684,93	200	0,87	149624,77	200	1,06
6000	113839,87	200	0,89	162590,21	200	0,84
6500	120890,85	200	1,10	164367,18	200	0,86
7000	117932,84	200	1,03	164862,65	200	0,64
7500	116443,22	200	1,03	172421,92	200	0,65

8000	110311,90	200	0,83	165350,02	200	0,53
8500	121325,36	200	1,27	171538,99	200	0,42
9000	102826,75	200	0,83	177167,83	200	0,48
9500	84780,12	200	0,58	197315,42	200	0,74
10000	76677,76	200	0,53	200908,36	200	0,77

address_sort_O2

	sorted			random		
размер массива	t, мкс	кол-во повторов	RSE, %	t, мкс	кол-во повторов	RSE, %
500	4368,17	200	1,94	4718,85	200	0,95
1000	9523,25	200	1,40	12456,98	200	2,52
1500	13498,04	200	1,28	16823,47	200	1,21
2000	16668,44	200	1,15	23327,42	200	1,22
2500	19048,56	200	0,95	29843,59	200	1,13
3000	22124,58	200	0,79	36864,90	200	0,89
3500	27074,40	200	1,39	47773,47	200	1,02
4000	27651,53	200	1,08	55236,76	200	0,84
4500	29969,47	200	0,98	61347,26	200	0,57
5000	33047,93	200	1,05	73096,15	200	0,58
5500	32919,07	200	0,91	81930,79	200	0,50
6000	34167,36	200	0,94	94518,87	200	0,47
6500	36435,71	200	1,13	107643,29	200	0,57
7000	37132,92	200	1,03	131728,12	200	0,74
7500	37894,56	200	1,08	156760,23	200	0,65
8000	38817,88	200	1,07	185195,81	200	0,97
8500	39775,52	200	0,96	199963,55	200	1,08
9000	38000,92	200	0,85	221059,61	200	0,76
9500	35268,57	200	0,91	259667,23	200	0,61
10000	35785,86	200	0,93	230362,11	200	0,70

pointer_sort_O0

	sorted			random		
размер массива	t, мкс	кол-во повторов	RSE, %	t, мкс	кол-во повторов	RSE, %
500	12700,94	200	1,66	11617,44	200	1,16
1000	22308,40	200	0,78	27481,44	200	1,40
1500	40871,67	200	1,31	34378,14	200	1,12
2000	44331,15	200	1,32	42150,50	200	0,89
2500	50414,79	200	1,06	56043,21	200	1,14
3000	57835,42	200	1,16	60263,49	200	0,47
3500	65050,19	200	1,13	68486,83	200	0,85
4000	71178,17	200	1,23	79688,40	200	0,94

4500	70331,06	200	0,98	89312,05	200	1,10
5000	70500,07	200	0,22	85420,59	200	0,18
5500	74943,97	200	0,39	94970,74	200	0,64
6000	86171,05	200	1,22	106445,43	200	1,17
6500	81035,59	200	0,80	113548,88	200	0,95
7000	77839,61	200	0,43	121614,13	200	1,45
7500	87539,15	200	1,36	123507,60	200	1,09
8000	87849,62	200	1,25	118934,28	200	0,44
8500	74126,86	200	0,43	124740,10	200	0,61
9000	72544,31	200	0,47	123666,32	200	0,61
9500	74673,96	200	0,93	122862,99	200	0,20
10000	66086,73	200	0,75	132694,27	200	0,79

pointer_sort_O2

	sorted			random		
размер массива	t, мкс	кол-во повторов	RSE, %	t, мкс	кол-во повторов	RSE, %
500	2917,54	200	0,67	3393,30	200	0,48
1000	5522,56	200	0,79	7033,47	200	0,44
1500	8026,40	200	0,46	11329,68	200	1,01
2000	12019,47	200	1,48	14259,17	200	0,67
2500	14858,68	200	1,44	18370,24	200	0,69
3000	15656,58	200	1,24	21280,74	200	0,78
3500	17516,72	200	1,06	24508,64	200	0,56
4000	21159,58	200	1,53	28562,83	200	0,56
4500	19495,04	200	0,73	30992,06	200	0,28
5000	21353,98	200	0,93	39219,82	200	1,76
5500	22859,35	200	0,99	39305,23	200	0,86
6000	26742,60	200	1,11	43103,72	200	0,60
6500	26118,56	200	1,36	45162,34	200	0,43
7000	25909,15	200	1,25	53164,33	200	1,10
7500	25151,83	200	1,20	59355,30	200	1,05
8000	26329,79	200	2,03	62502,58	200	1,07
8500	26425,72	200	1,51	70168,56	200	1,29
9000	24590,40	200	1,31	74003,52	200	0,90
9500	22184,81	200	0,45	83397,52	200	1,12
10000	23789,49	200	1,30	83509,90	200	0,76

index_sort_O0

	sorted			random		
размер массива	t, мкс	кол-во повторов	RSE, %	t, мкс	кол-во повторов	RSE, %
500	20079,60	200	1,15	19001,22	200	1,51

1000	41048,69	200	0,93	36156,75	200	1,02
1500	59792,07	200	0,43	48823,88	200	0,91
2000	73548,74	200	0,75	65870,98	200	0,86
2500	90907,36	200	0,74	91818,65	200	1,04
3000	102517,35	200	0,61	96340,07	200	1,01
3500	101517,55	200	0,98	112707,44	200	1,20
4000	88156,14	200	0,78	108469,51	200	0,50
4500	108165,59	200	1,20	158317,73	200	1,05
5000	98525,77	200	0,43	155318,11	200	1,26
5500	104084,24	200	0,58	160192,08	200	1,28
6000	104487,17	200	0,73	148336,99	200	0,81
6500	104073,57	200	0,62	163888,28	200	0,68
7000	107560,27	200	0,97	168319,23	200	0,70
7500	101804,12	200	0,67	169479,58	200	0,70
8000	96355,56	200	0,34	172995,30	200	0,70
8500	91367,98	200	0,32	176109,92	200	0,59
9000	83917,90	200	0,24	188803,54	200	0,71
9500	77007,91	200	0,22	194890,27	200	0,88
10000	75758,70	200	1,10	209674,17	200	1,16

index_sort_O2

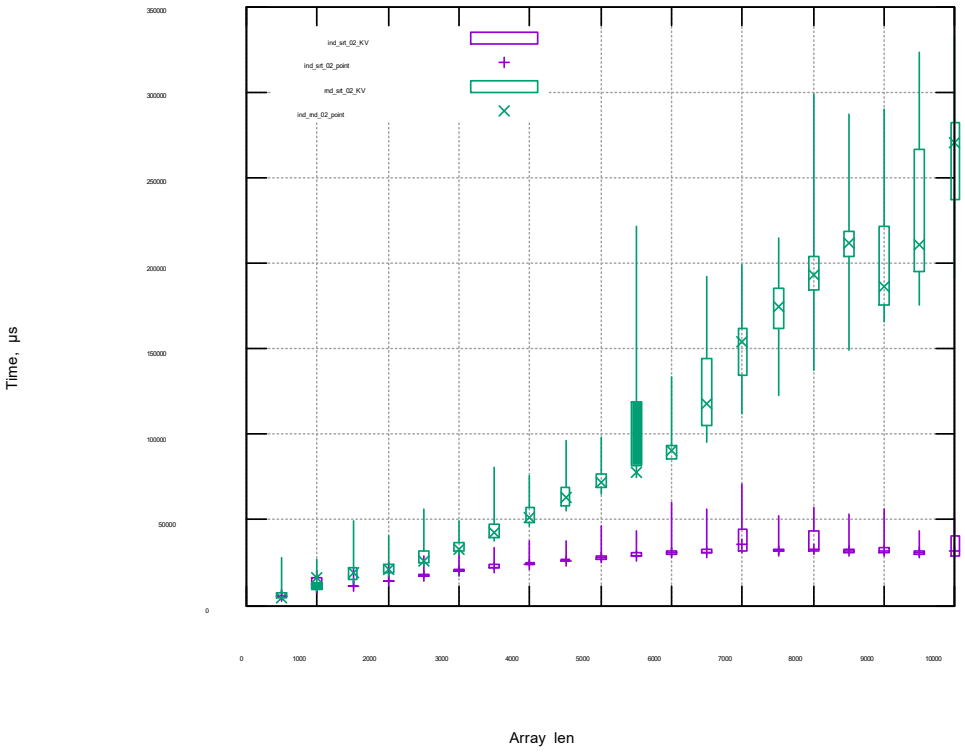
	sorted			random		
размер массива	t, мкс	кол-во повторов	RSE, %	t, мкс	кол-во повторов	RSE, %
500	5496,01	200	1,28	6011,16	200	2,66
1000	10153,51	200	1,17	12161,34	200	1,88
1500	11480,42	200	0,68	19659,67	200	1,91
2000	14607,66	200	0,50	22209,06	200	1,27
2500	18247,76	200	0,67	28618,63	200	1,26
3000	20916,42	200	0,65	34651,61	200	0,86
3500	23564,03	200	0,69	44915,07	200	1,05
4000	25349,29	200	0,58	54009,68	200	0,85
4500	26964,17	200	0,53	64617,46	200	0,81
5000	28986,35	200	0,79	73470,18	200	0,60
5500	30199,97	200	0,48	93491,00	200	1,47
6000	31923,69	200	0,90	96262,39	200	0,60
6500	32641,80	200	0,69	116465,40	200	1,16
7000	38916,10	200	1,42	148963,56	200	0,85
7500	33198,82	200	0,62	171269,88	200	0,80
8000	36455,79	200	1,20	191848,26	200	0,75
8500	32528,31	200	0,53	206697,50	200	0,84
9000	33536,65	200	0,93	198045,60	200	1,00
9500	31352,93	200	0,51	227512,52	200	1,20

10000	35032,79	200	1,25	260436,97	200	0,93

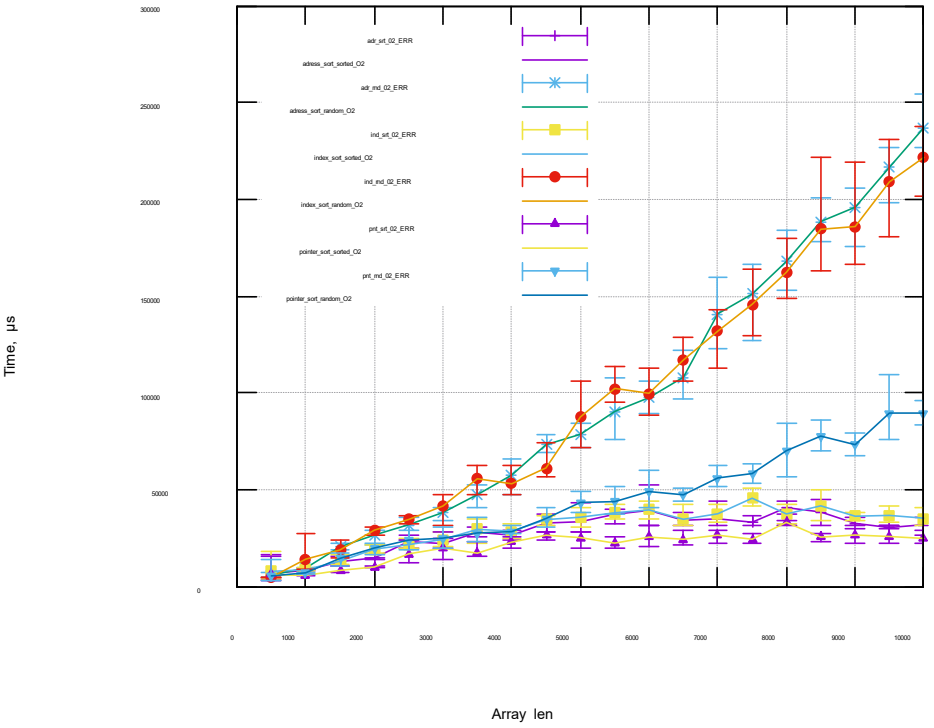
Графики:

Графики зависимости времени работы от размеров массива

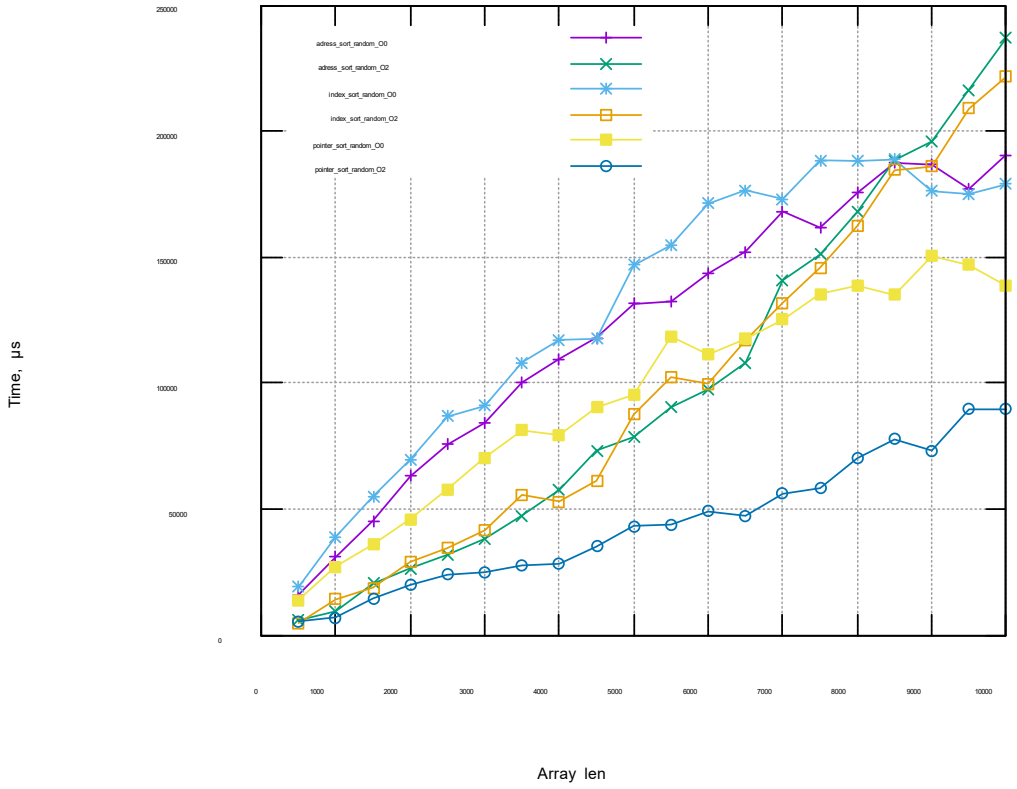
- **kvartil.svg** – графики с усами любых массивов с оптимизацией O2 и способом обработки элементов массива с помощью индексации



- **errs.svg** – кусочно-линейные графики любых массивов и всех способов обработки элементов массива при оптимизации O2



- **random.svg** – кусочно-линейные графики случайных массивов всех уровней оптимизации и способов обработки элементов массива



- **sorted.svg** – кусочно-линейные графики отсортированных массивов всех уровней оптимизации и способов обработки элементов массива

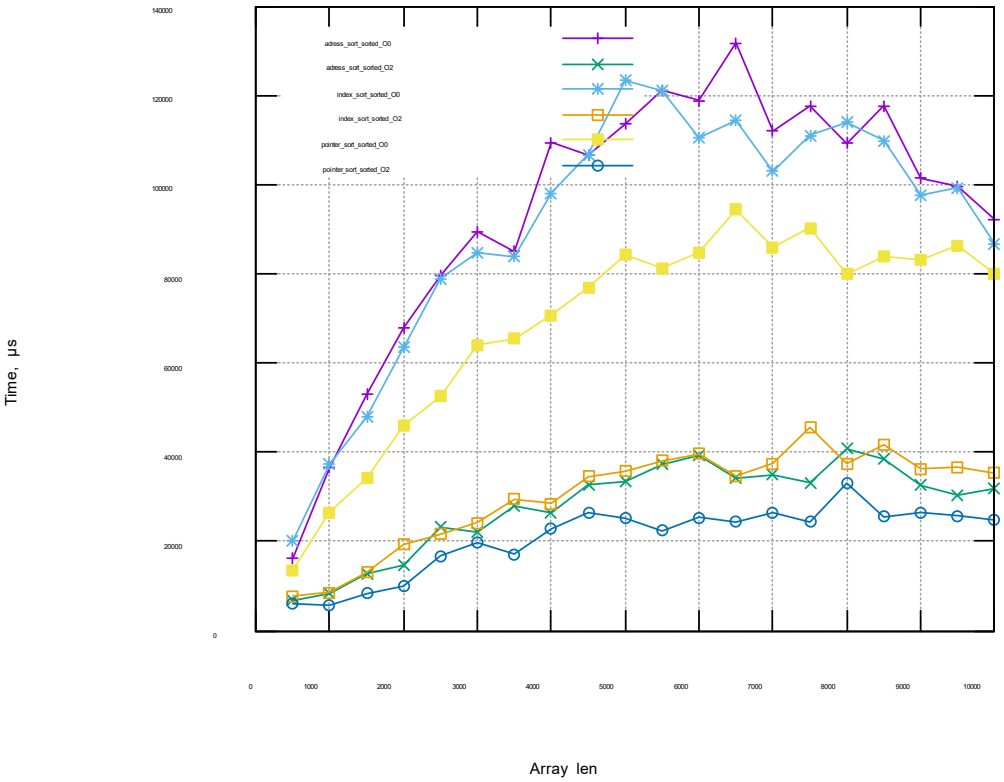


Таблица с величиной ln:

pointer_sort_random_O2

размер массива	t, мкс	кол-во повторов	RSE, %	$\frac{\ln(t_{i+1}) - \ln(t_i)}{\ln(n_{i+1}) - \ln(n_i)}$
500	3393,30	200	0,48	1,05
1000	7033,47	200	0,44	1,18
1500	11329,68	200	1,01	0,80
2000	14259,17	200	0,67	1,14
2500	18370,24	200	0,69	0,81
3000	21280,74	200	0,78	0,92
3500	24508,64	200	0,56	1,15
4000	28562,83	200	0,56	0,69
4500	30992,06	200	0,28	2,23
5000	39219,82	200	1,76	0,02
5500	39305,23	200	0,86	1,06
6000	43103,72	200	0,60	0,58
6500	45162,34	200	0,43	2,20
7000	53164,33	200	1,10	1,60
7500	59355,30	200	1,05	0,80
8000	62502,58	200	1,07	1,91
8500	70168,56	200	1,29	0,93
9000	74003,52	200	0,90	2,21
9500	83397,52	200	1,12	0,03

Вывод: Графики показывают, что работа указателями является самым быстрым способом. Такой способ обращения к элементам массива является самым быстрым т.к. не тратит времени на операции чтения индексов

Сложение квадратных матриц

Цель: узнать самый быстрый способ работы с двумерными массивами

Количество запусков: 2100

Способы работы с матрицами:

- **snrestrict** – матрица передается в функцию без использования **restrict** и со стандартным обходом элементов
- **cnrestrict** – матрица передается в функцию без использования **restrict** и с обходом элементов по столбцам
- **restrict** – матрица передается в функцию с использованием **restrict** и со стандартным обходом элементов

Рабочая среда:
apps – каталог с собранными исполняемыми файлы. Размеры создаваемых массивов передаются в качестве аргумента

measures – каталог, который содержит собранные измерения. Он организован следующим образом: есть каталоги, содержащие различные способы обработки и передачи матриц. В этих каталогах хранятся каталоги с различными уровнями оптимизации. И внутри этих каталогов находятся файлы, названия которых указывают на длины массивов и содержат замеры времени выполнения. Также в каталоге есть файл **runs.txt**, который хранит информацию о количестве проведенных измерений.

progs – каталог с исходными кодами программ

data_work – каталог работы с данными
data_work/data – каталог с замеряемыми данными. Каталог храни в себе: файл **optimizations** с уровнями оптимизаций и файл **matrix_sizes**, который хранит в себе измеряемые размеры матриц
data_work/calced_data – каталог с измеренными данными. Содержит в себе текстовые файлы, которые хранят в себе размеры измеряемых матриц и среднее время. Данные распределены в текстовых файлах по виду передачи и обработки матриц, уровню оптимизации
data_work/build_apps.sh – скрипт собирает исполняемые файлы и помещает их в каталог **./apps**. Скрипт собирает исполняемые файлы по разным видам передачи и

обработки матрицы, уровням оптимизации. Также скрипт определяет макрос максимальной длины массива при компиляции, которая передается в скрипт как аргумент (при отсутствии аргумента стандартное значение максимальной длины массива равно 1000)
data_work/update_data.sh – скрипт добавление новых замеров в каталог **./measures**. Скрипт создает нужные каталоги при их отсутствии и записывает в них определенное количество замеров, которые передаются в скрипт как аргумент(при отсутствии аргумента стандартное количество замеров равно 100)
data_work/clean_data.sh – скрипт удаления измерений из каталога **./measures**
data_work/make_preproc.py – программа считает среднее значение измерений времени для каждого измеряемого размера матрицы, уровня оптимизации и способа передачи и обработки матрицы. Программа записывает полученные измерения в файлы расположенные в каталоге **./data_work/calced_data**
data_work/make_postproc.sh – скрипт отрисовывает графики по скриптам описанным в каталоге **./data_work/graph_draw**. Графики строятся по данным, которые находятся в каталоге **./data_work/calced_data**
data_work/go.sh – скрипт запускает всю систему работы с данными
data_work/graph_draw – каталог хранит в себе все скрипты для заданных графиков
* **data_work/table_create.py** – программа чертит таблицы с относительной стандартной ошибкой среднего для каждой длины массива. Таблицы чертятся для каждого вида массива, уровня оптимизации и способа обращения к элементам массива.

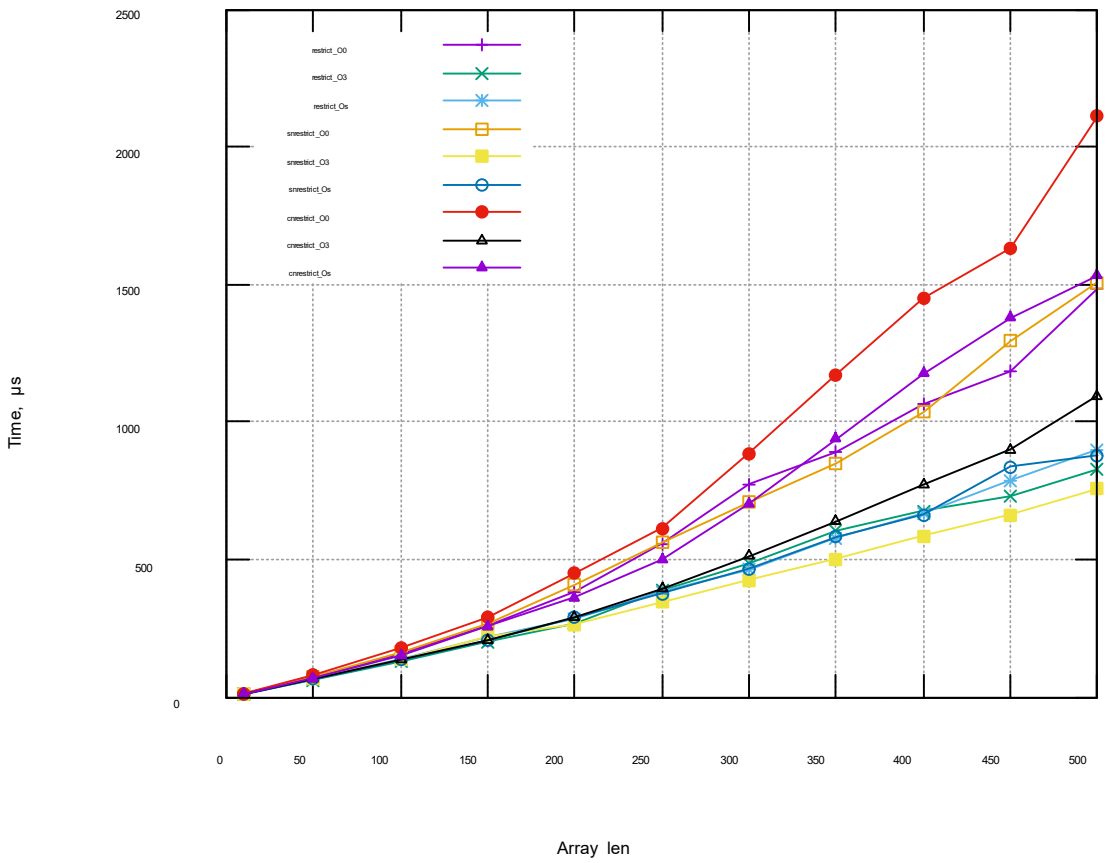
Таблицы:

	restrict_O3			snrestrict_O3			cnrestrict_O3		
length	t_i, мкс	RSE, %	$\frac{\ln(t_{i+1}) - \ln(t_i)}{\ln(n_{i+1}) - \ln(n_i)}$	t_i, мкс	RSE, %	$\frac{\ln(t_{i+1}) - \ln(t_i)}{\ln(n_{i+1}) - \ln(n_i)}$	t_i, мкс	RSE, %	$\frac{\ln(t_{i+1}) - \ln(t_i)}{\ln(n_{i+1}) - \ln(n_i)}$
10	11,12	2,01	1,08	10,95	1,98	1,14	10,64	0,68	1,14
50	62,91	0,62	1,04	68,36	0,65	0,99	66,49	0,52	1,03
100	129,25	0,44	1,10	135,42	0,58	1,20	135,79	0,51	1,03
150	201,55	1,71	0,98	220,21	0,65	0,65	206,36	0,39	1,19
200	267,52	0,37	1,67	265,61	0,69	1,17	290,62	0,64	1,35
250	387,91	0,58	1,23	344,98	0,36	1,16	393,05	0,37	1,44
300	485,67	0,65	1,42	426,57	0,47	1,08	511,02	0,67	1,44
350	604,65	1,72	0,86	503,70	0,29	1,15	637,78	0,47	1,43
400	677,79	0,98	0,63	587,15	0,81	1,05	772,13	0,42	1,30
450	730,01	0,43	1,20	664,20	0,25	1,24	900,37	0,41	1,87
500	828,27	0,47		756,66	1,16		1096,24	0,42	

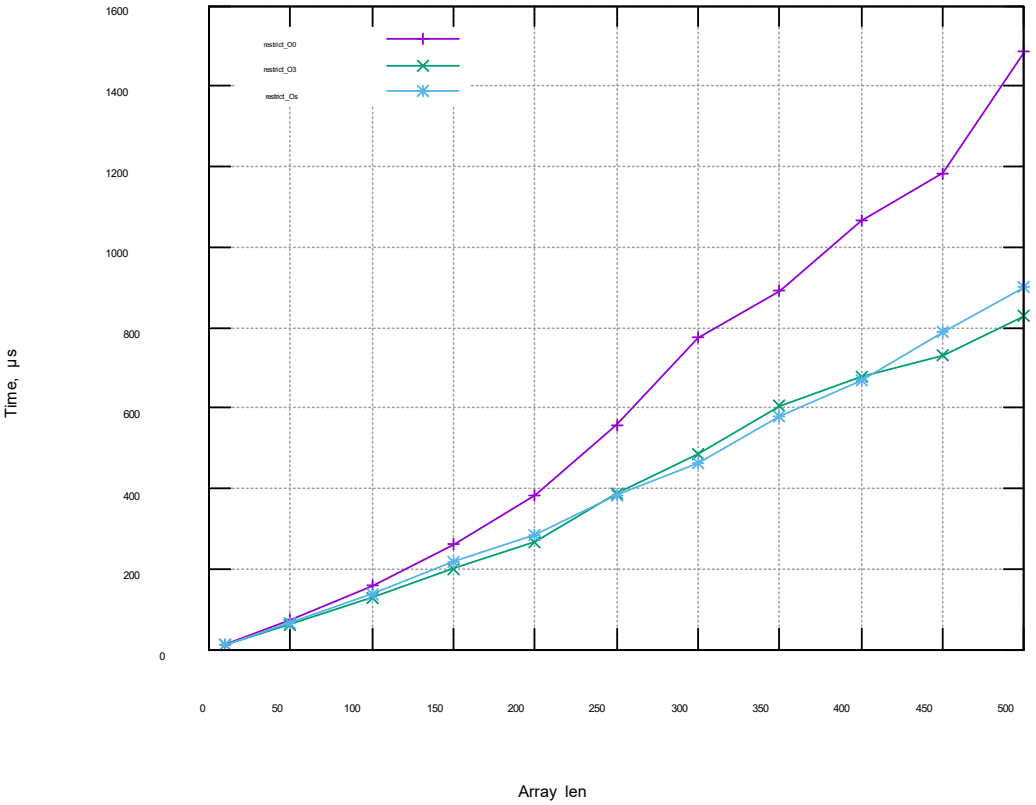
Графики:

Графики зависимости времени работы от размеров квадратных матриц

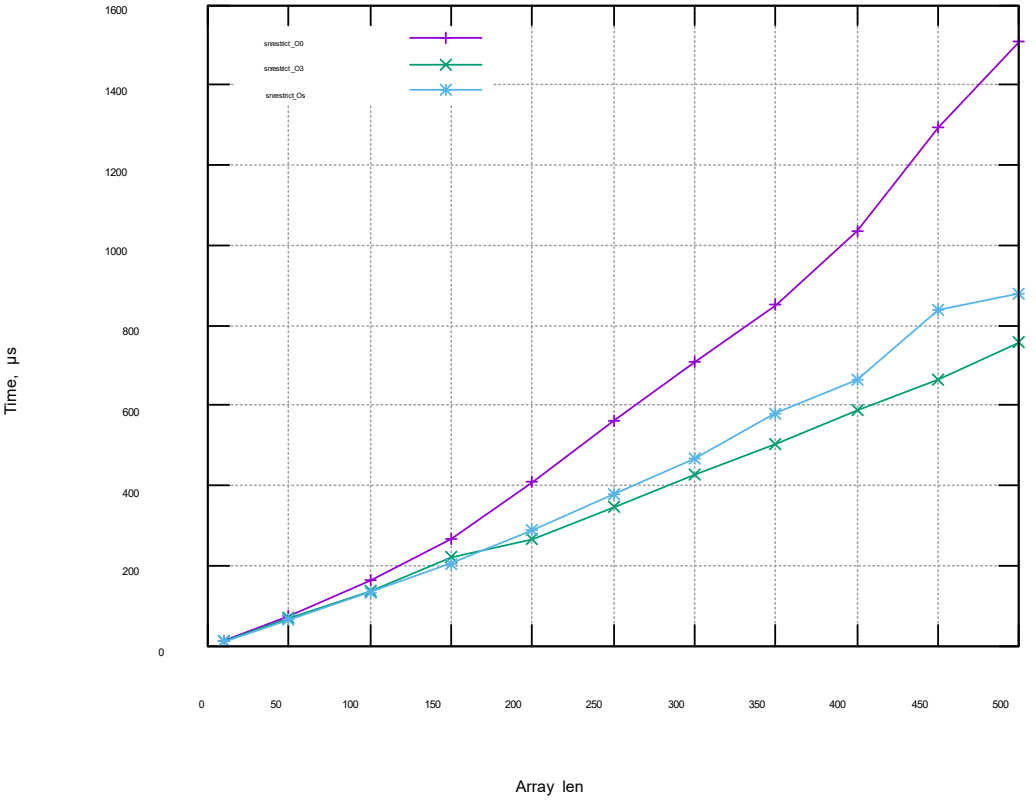
- **all.svg** – кусочно-линейные графики всех уровней оптимизации и способов обработки передачи и обработки матрицы



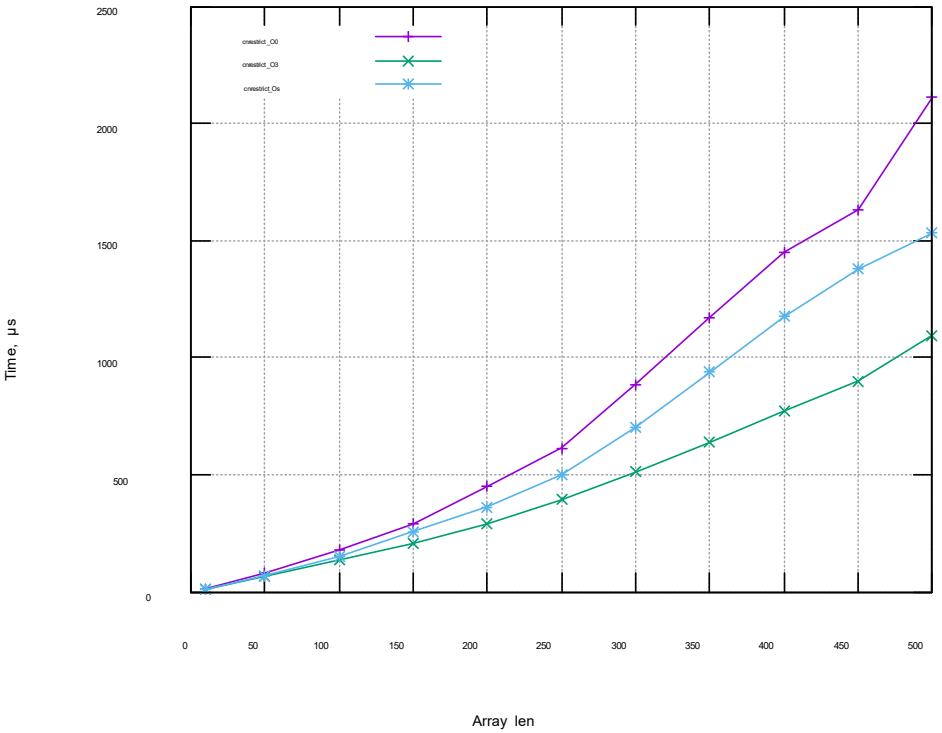
- **restr.svg** – кусочно-линейные графики всех уровней оптимизации при использовании **restrict**



- **snrestr.svg** – кусочно-линейные графики всех уровней оптимизации без использования **restrict** и со стандартным обходом элементов



- **cnrestr.svg** – кусочно-линейные графики всех уровней оптимизации без использования **restrict** и обходом элементов по столбцам



Вывод: Графики показывают, что использование restrict ускоряет работу программы. **restrict** ускоряет код, т.к. он сообщает компилятору, что нет никаких зависимостей между памятью, на которую указывают различные его указатели.