



Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н. Э. Баумана
(национальный исследовательский университет)
(МГТУ им. Н. Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

Отчет по лабораторной работе №3 по курсу "Архитектура ЭВМ"

Тема Организация памяти суперскалярных ЭВМ

Студент Цветков И.А.

Группа ИУ7-53Б

Оценка (баллы) _____

Преподаватель Дубровин Е.Н.

Содержание

1 Выполнение лабораторной работы	4
1.1 Программа PCLAB	4
1.2 Характеристики процессора	5
2 Эксперименты	8
2.1 Исследования расслоения динамической памяти	8
2.1.1 Исходные данные	8
2.1.2 Результаты эксперимента	8
2.1.3 Описание проблемы	8
2.1.4 Суть эксперимента	9
2.1.5 Проведение эксперимента	9
2.1.6 Вывод	11
2.2 Сравнение эффективности ссылочных и векторных структур . .	12
2.2.1 Результаты эксперимента	12
2.2.2 Описание проблемы	12
2.2.3 Суть эксперимента	12
2.2.4 Проведение эксперимента	13
2.2.5 Вывод	16
2.3 Исследование эффективности программной предвыборки	16
2.3.1 Исходные данные	16
2.3.2 Результаты эксперимента	16
2.3.3 Описание проблемы	16
2.3.4 Суть эксперимента	17
2.3.5 Проведение эксперимента	17
2.3.6 Вывод	20
2.4 Исследование способов эффективного чтения оперативной па- мяти	20
2.4.1 Исходные данные	20
2.4.2 Результаты эксперимента	20
2.4.3 Описание проблемы	20
2.4.4 Суть эксперимента	21

2.4.5	Проведение эксперимента	21
2.4.6	Вывод	24
2.5	Исследование конфликтов в кэш-памяти	24
2.5.1	Исходные данные	24
2.5.2	Результаты эксперимента	24
2.5.3	Описание проблемы	24
2.5.4	Суть эксперимента	25
2.5.5	Проведение эксперимента	25
2.5.6	Вывод	27
2.6	Сравнение алгоритмов сортировки	27
2.6.1	Исходные данные	27
2.6.2	Результаты эксперимента	27
2.6.3	Описание проблемы	28
2.6.4	Суть эксперимента	28
2.6.5	Проведение эксперимента	28
2.6.6	Вывод	30
Заключение		31

Введение

Целью данной работы является освоение принципов эффективного использования подсистемы памяти современных универсальных ЭВМ, обеспечивающей хранение и своевременную выдачу команд и данных в центральное процессорное устройство. Работа проводится с использованием программы для сбора и анализа производительности PCLAB.

В ходе работы необходимо ознакомиться с теоретическим материалом, касающимся особенностей функционирования подсистемы памяти современных конвейерных и суперскалярных ЭВМ, изучить возможности программы PCLAB, изучить средства идентификации микропроцессоров, провести исследования времени выполнения тестовых программ, сделать выводы о архитектурных особенностях используемых ЭВМ

1 Выполнение лабораторной работы

При исследовании производительности будет использована **PCLAB**.

1.1 Программа PCLAB

Программа **PCLAB** предназначена для исследования производительности x86 совместимых ЭВМ сIA32 архитектурой, работающих под управлением операционной системы Windows (версий 95 и старше). Исследование организации ЭВМ заключается в проведении ряда экспериментов, направленных на построение зависимостей времени обработки критических участков кода от изменяемых параметров. Набор реализуемых программой экспериментов позволяет исследовать особенности построения современных подсистем памяти ЭВМ и процессорных устройств, выявить конструктивные параметры конкретных моделей ЭВМ.

Процесс сбора и анализа экспериментальных данных в **PCLAB** основан на процедуре профилировки критического кода, т.е. в измерении времени его обработки центральным процессорным устройством. При исследовании конвейерных суперскалярных процессорных устройств, таких как 32-х разрядные процессоры фирмы Intel или AMD, способных выполнять переупорядоченную обработку последовательности команд программы, требуется использовать специальные средства измерения временных интервалов из-за прещения переупорядочивания микрокоманд. Для измерения времени работы циклов в **PCLAB** используется следующая методика:

- длительность обработки участка профилируемой программы характеризуется изменением величины счетчика тактов процессора, произошедшими за время его работы;
- для предотвращения влияния соседних участков кода на результаты измерений, перед началом замера и после его окончания необходимо выдать команду упорядоченного выполнения CPUID, препятствующую переупорядочивание потока команд на конвейер процессора;
- замеры количества тактов процессора необходимо повторить несколько раз;

- взаимное влияние последовательных повторов экспериментального участка программы исключается благодаря очищению кэш-памяти и буферов процессора;
- часть граничных результатов отбрасывается (как наибольших, так и наименьших). По оставшимся результатам замеров определяется средняя величина. На рисунке 8 показан внешний вид программы PCLAB 1.0.

1.2 Характеристики процессора

На рисунке 1.1 представлены характеристики процессора компьютера в лабораторной аудитории, которые получены в программе **PCLAB**.

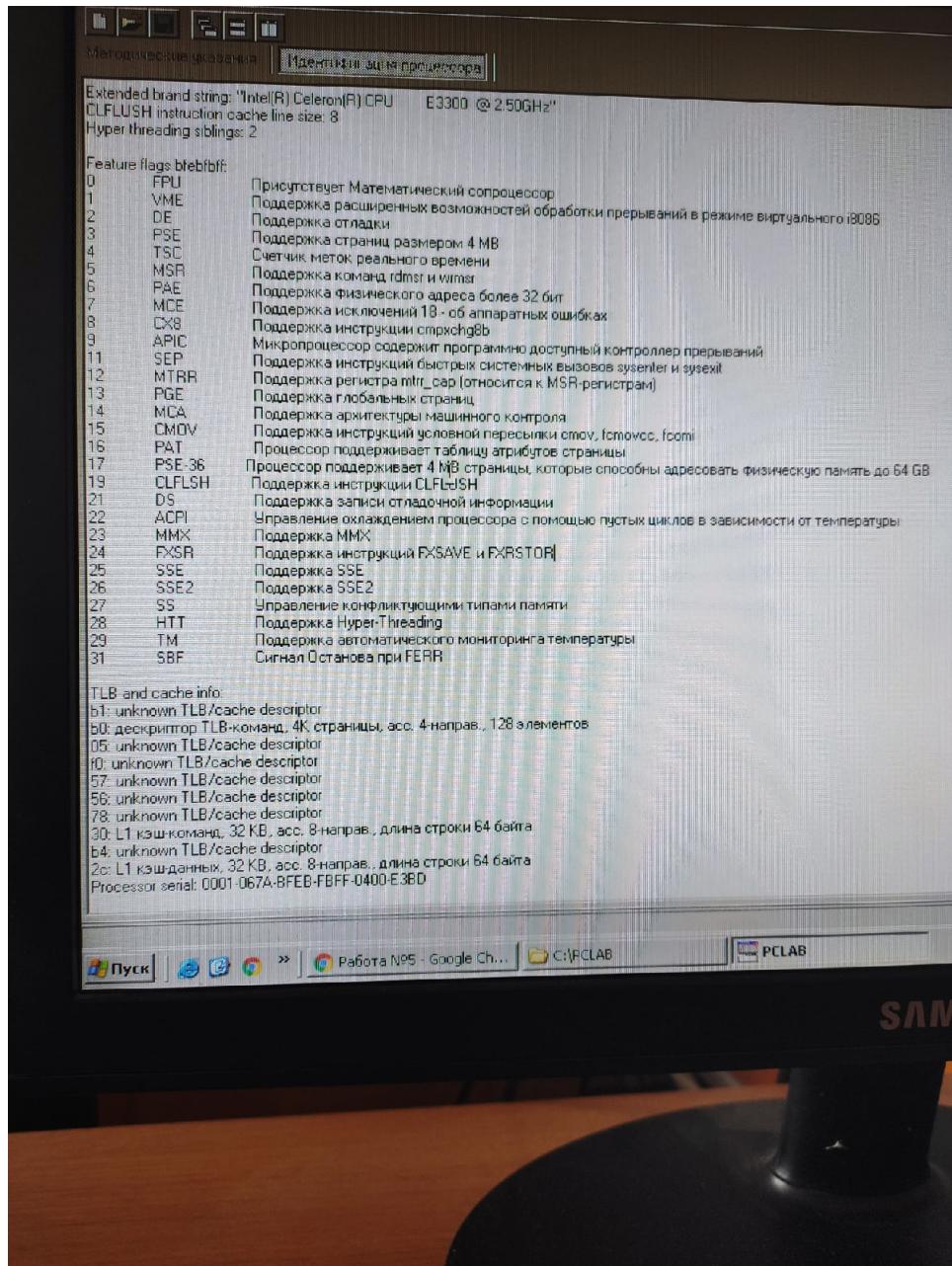


Рисунок 1.1 – Характеристики процессора

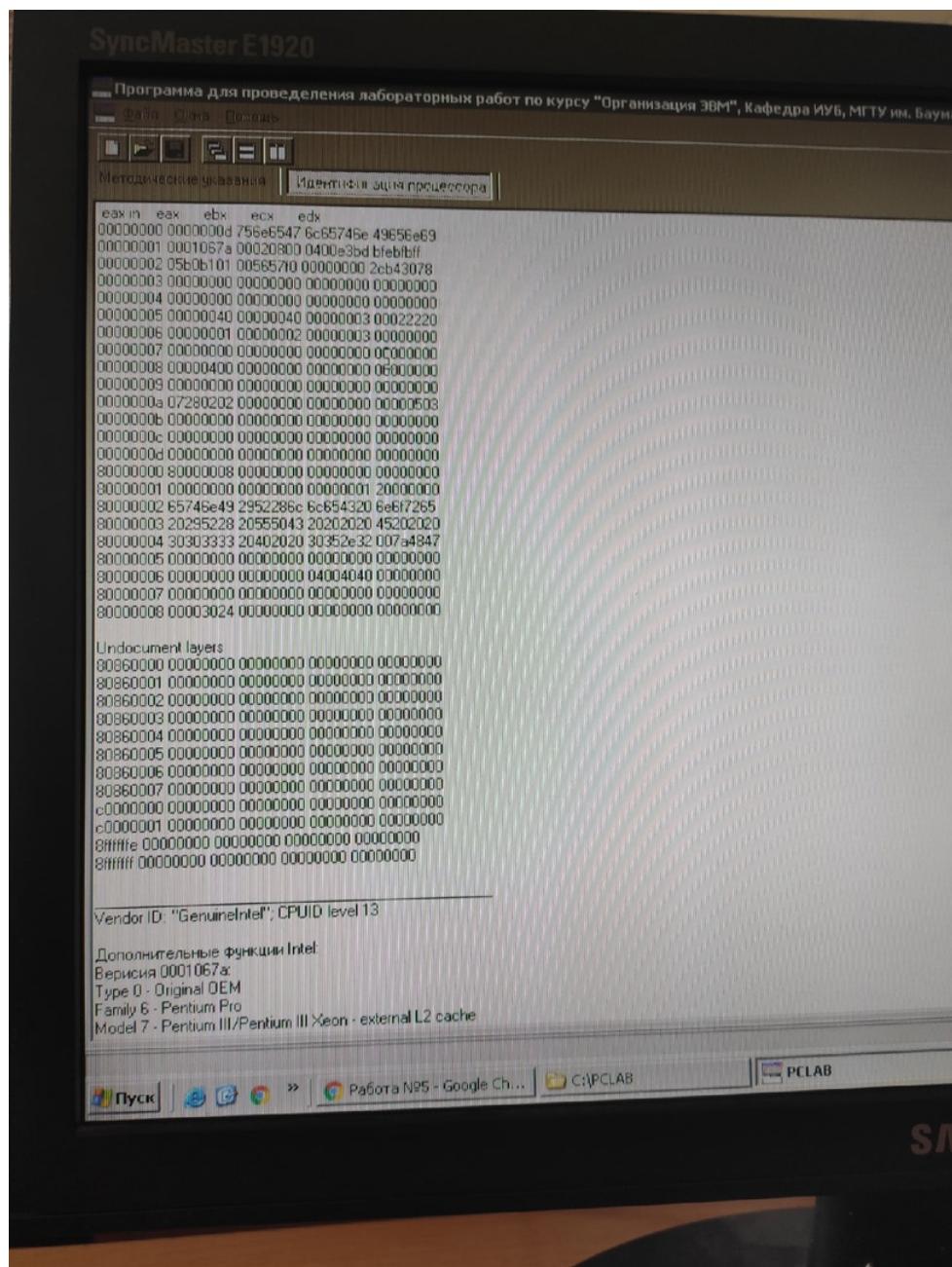


Рисунок 1.2 – Идентификация процессора

2 Эксперименты

Для исследования производительности был проведен ряд экспериментов, которые представлены ниже.

2.1 Исследования расслоения динамической памяти

Цель эксперимента – определение способа трансляции физического адреса, используемого при обращении к динамической памяти.

2.1.1 Исходные данные

1. Размер линейки кэш-памяти верхнего уровня.
2. Объем физической памяти.

2.1.2 Результаты эксперимента

1. Количество банков динамической памяти.
2. Размер одной страницы динамической памяти.
3. Количество страниц в динамической памяти.

2.1.3 Описание проблемы

В связи с конструктивной неоднородностью оперативной памяти, обращение к последовательно расположенным данным требует различного времени. В связи с этим, для создания эффективных программ необходимо учитывать расслоение памяти, характеризуемое способом трансляции физического адреса.

2.1.4 Суть эксперимента

Для определения способа трансляции физического адреса при формировании сигналов выборки банка, выборки строки и столбца запоминающего массива применяется процедура замера времени обращения к динамической памяти по последовательным адресам с изменяющимся шагом чтения. Для сравнения времен используется обращение к одинаковому количеству различных ячеек, отстоящих друг от друга на определенный шаг. Результат эксперимента представляется зависимостью времени (или количества тактов процессора), потраченного на чтение ячеек, от шага чтения. Для проведения эксперимента необходимо задать изменяемые параметры.

2.1.5 Проведение эксперимента

Изменяемые параметры.

1. Максимальное расстояние между читаемыми блоками = 7.
2. Шаг увеличения расстояния между читаемыми 4-х байтовыми ячейками = 10.
3. Количество страниц в динамической памяти = 4.

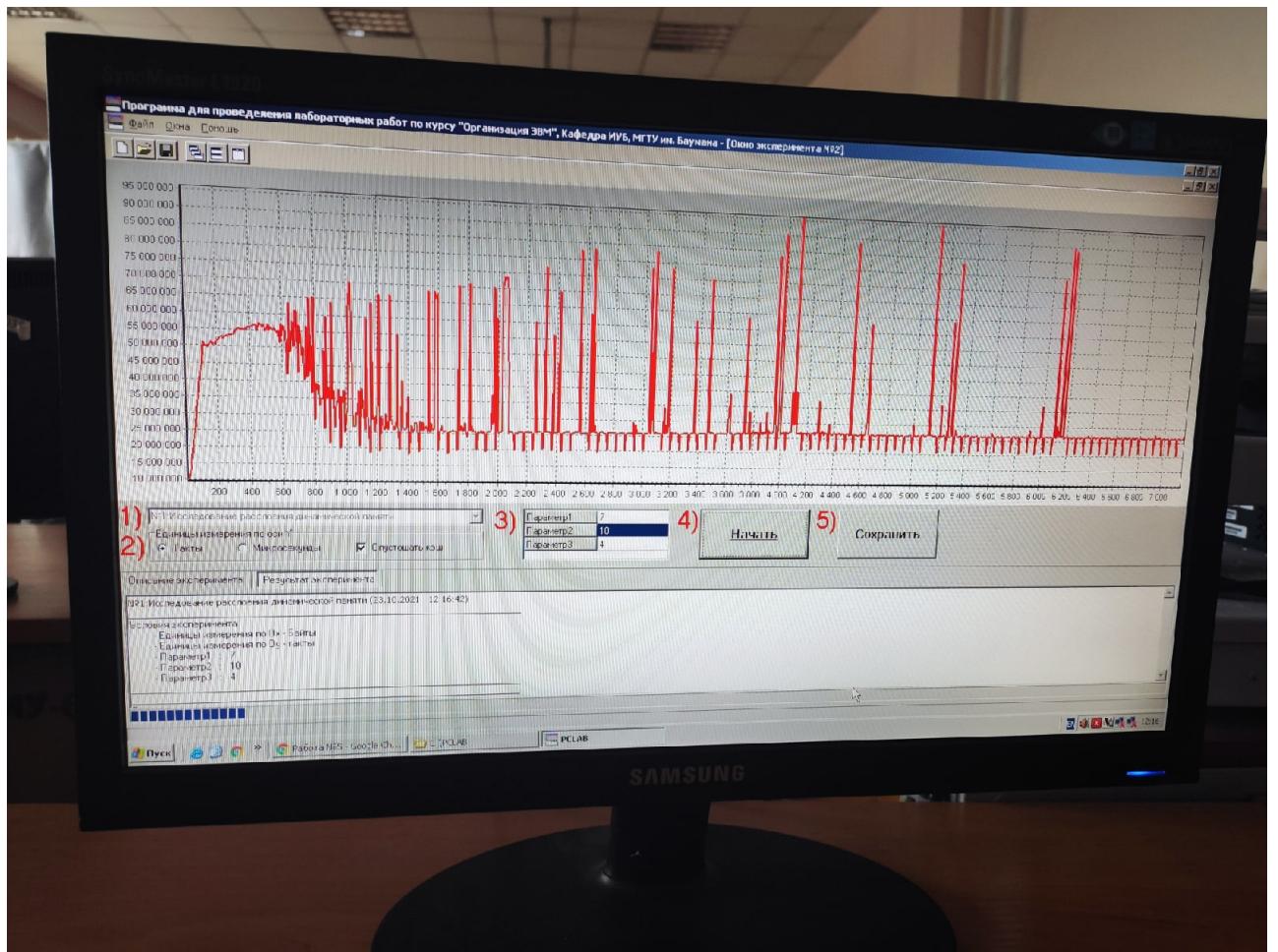


Рисунок 2.1 – Результат выполнения эксперимента в PCLAB

Также получим значения следующих величин:

$$B = T_1/M, \quad (2.1)$$

где B – количество банков; T_1 – минимальный шаг чтения динамической памяти, при котором происходит постоянное обращение к одному и тому же банку; M – объем данных, являющийся минимальной порцией обмена кэш-памяти верхнего уровня с оперативной памятью и соответствует размеру линейки кэш-памяти верхнего уровня

А также

$$S = T_2/B, \quad (2.2)$$

где B – количество банков; T_2 – соответствует расстоянию (в байтах) между началом двух последовательных страниц одного банка; S – количество секторов.

В результате эксперимента было получено, что $T_1 = 128$ и $T_2 = 4096$, тогда:

$$B = T_1/M = 128/64 = 2; \quad (2.3)$$

$$S = T_2/B = 4096/2 = 2048. \quad (2.4)$$

2.1.6 Вывод

Необходимо учитывать расслоение памяти при обработке данных, потому что оперативная память неоднородна, поэтому для обращение к последовательно расположенным данным может потребоваться разное количество времени.

2.2 Сравнение эффективности ссылочных и векторных структур

Цель эксперимента – оценка влияния зависимости команд по данным на эффективность вычислений.

2.2.1 Результаты эксперимента

Отношение времени работы алгоритма, использующего зависимые данные, ко времени обработки аналогичного алгоритма обработки независимых данных.

2.2.2 Описание проблемы

Обработка зависимых данных происходит в тех случаях, когда результат работы одной команды используется в качестве адреса операнда другой. При программировании на языках высокого уровня такими operandами являются указатели, активно используемые при обработке ссылочных структур данных: списков, деревьев, графов. Обработка данных структур процессорами с длинными конвейерами команд приводит к заметному увеличению времени работы алгоритмов: адрес загружаемого операнда становится известным только после обработки предыдущей команды. В противоположность этому, обработка векторных структур, таких как массивы, позволяет эффективно использовать аппаратные возможности ЭВМ.

2.2.3 Суть эксперимента

Для сравнения эффективности векторных и списковых структур в эксперименте применяется профилировка кода двух алгоритмов поиска минимального значения. Первый алгоритм использует для хранения данных список, в то время как во втором применяется массив. Очевидно, что время работы

алгоритма поиска минимального значения в списке зависит от его фрагментации, т.е. от среднего расстояния между элементами списка.

2.2.4 Проведение эксперимента

Изменяемые параметры.

1. Количество элементов в списке = 4.
2. Максимальная фрагментации списка = 256.
3. Шаг увеличения фрагментации = 4.

На графике, который представлен на рисунке 2.2:

- **красная линия** – количество тактов работы алгоритма с использованием списка;
- **зеленая линия** – количество тактов работы алгоритма с использованием массива.

При этом на оси абсцисс указана фрагментация списка.

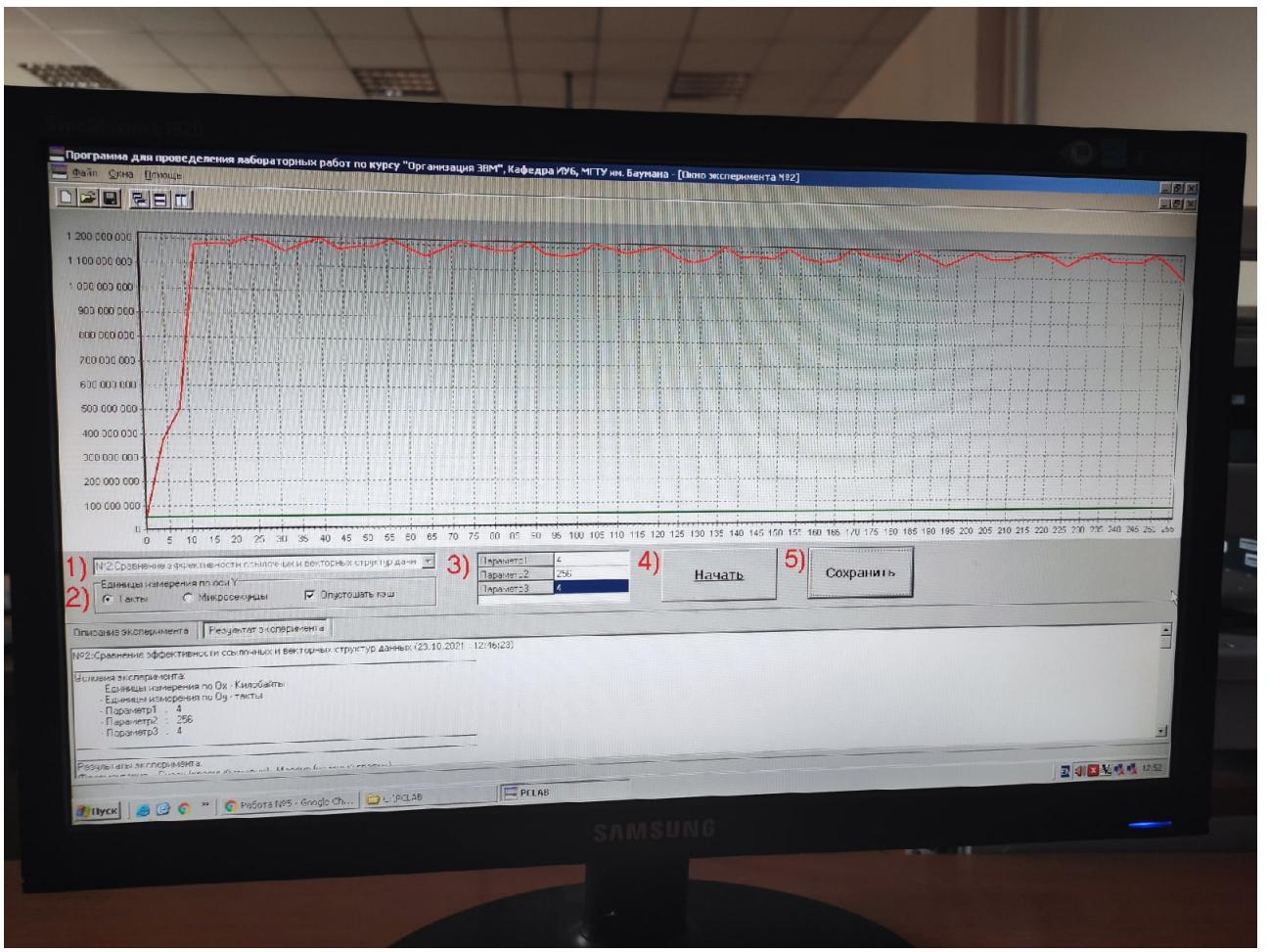


Рисунок 2.2 – Результат выполнения эксперимента в PCLAB

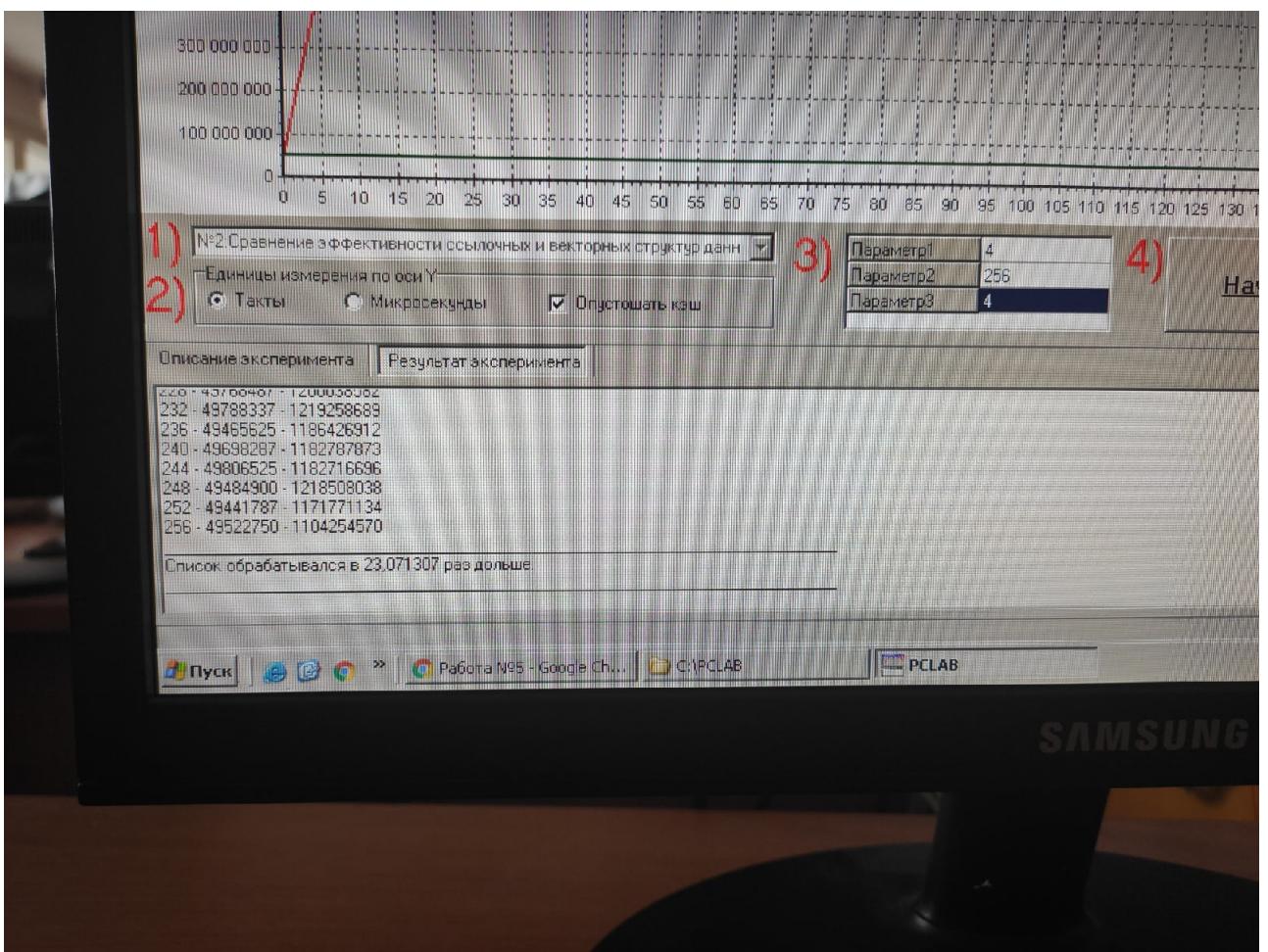


Рисунок 2.3 – Результат сравнения времени работы

Также на рисунке 2.3 представлен результат сравнения, на котором видно, что список обрабатывается в 23,071307 раз дольше массива.

2.2.5 Вывод

Необходимо учитывать технологический фактор решаемой задачи, от которой и зависит то, какая структура данных подходит больше. В данном случае приоритетным является использование массива.

2.3 Исследование эффективности программной предвыборки

Цель эксперимента – оценка влияния зависимости команд по данным на эффективность вычислений.

2.3.1 Исходные данные

1. Степень ассоциативности.
2. Размер TLB данных.

2.3.2 Результаты эксперимента

Отношение времени последовательной обработки блока данных ко времени обработки блока с применением предвыборки; время и количество тактов первого обращения к странице данных.

2.3.3 Описание проблемы

Обработка больших массивов информации сопряжена с открытием большого количества физических страниц памяти. При первом обращении к стра-

нице памяти наблюдается увеличенное время доступа к данным. Это связано с необходимостью преобразования логического адреса в физический адрес памяти, а также с открытием страницы динамической памяти и сохранения данных в кэш-памяти. Преобразование выполняется на основе информации о использованных ранее страницах, содержащейся в TLB буфере процессора. Первое обращение к странице при отсутствии информации в TLB вызывает двойное обращение к оперативной памяти: сначала за информацией из таблицы страниц, а далее за востребованными данными. Предвыборка заключается в заблаговременном проведении всех указанных действий благодаря дополнительному запросу небольшого количества данных из оперативной памяти.

2.3.4 Суть эксперимента

Эксперимент основан на замере времени двух вариантов подпрограмм последовательного чтения страниц оперативной памяти. В первом варианте выполняется последовательное чтение без дополнительной оптимизации, что приводит к дополнительным двойным обращениям. Во втором варианте перед циклом чтения страниц используется дополнительный цикл предвыборки, обеспечивающий своевременную загрузку информации в TLB данных.

2.3.5 Проведение эксперимента

Изменяемые параметры.

1. Шаг увеличения расстояния между читаемыми данными = 256.
2. Размер массива = 256.

На графике, который представлен на рисунке 2.4:

- **красная линия** – количество тактов работы алгоритма без использования предвыборки;
- **зеленая линия** – количество тактов работы алгоритма с использованием предвыборки.

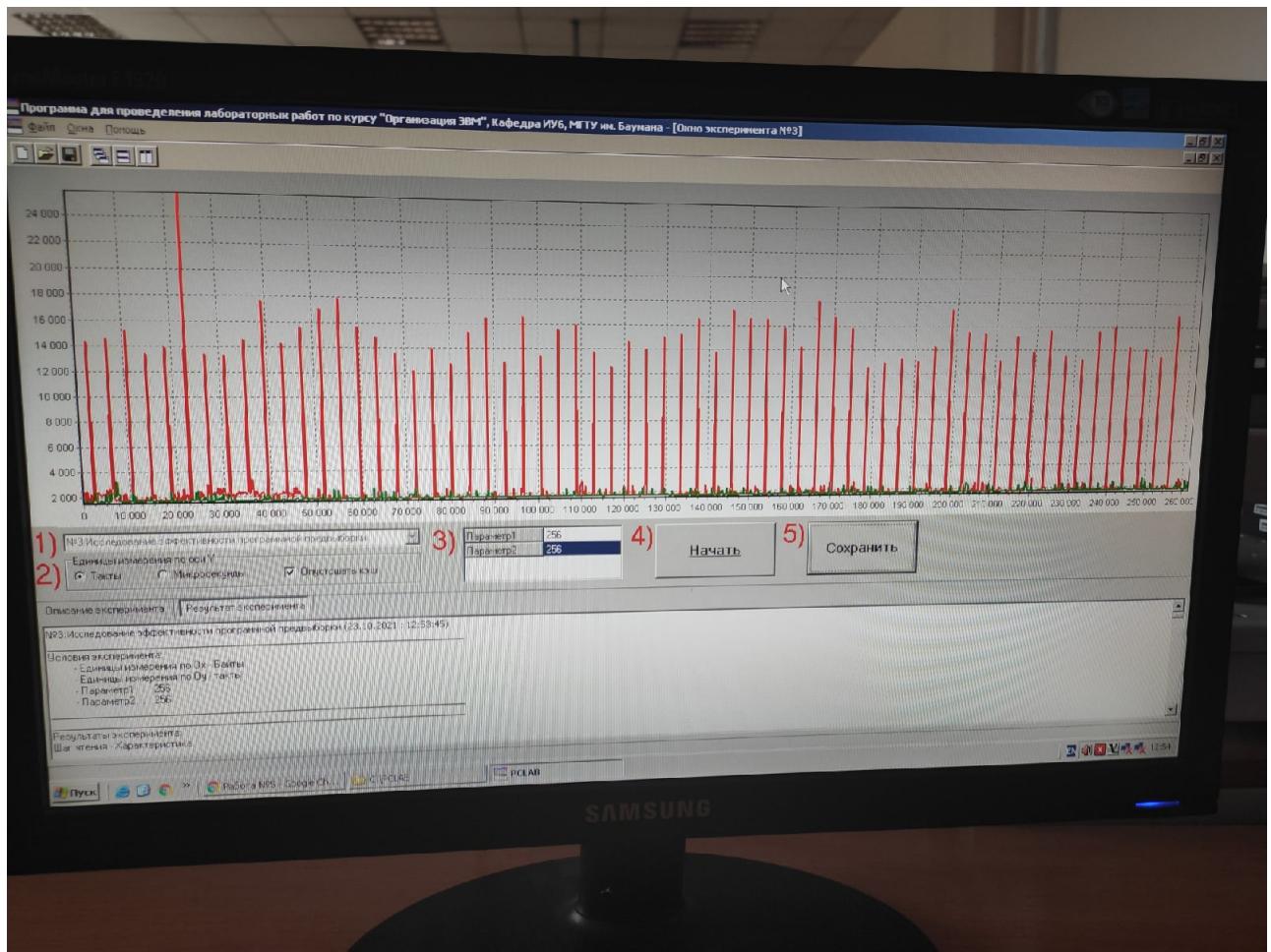


Рисунок 2.4 – Результат выполнения эксперимента в PCLAB

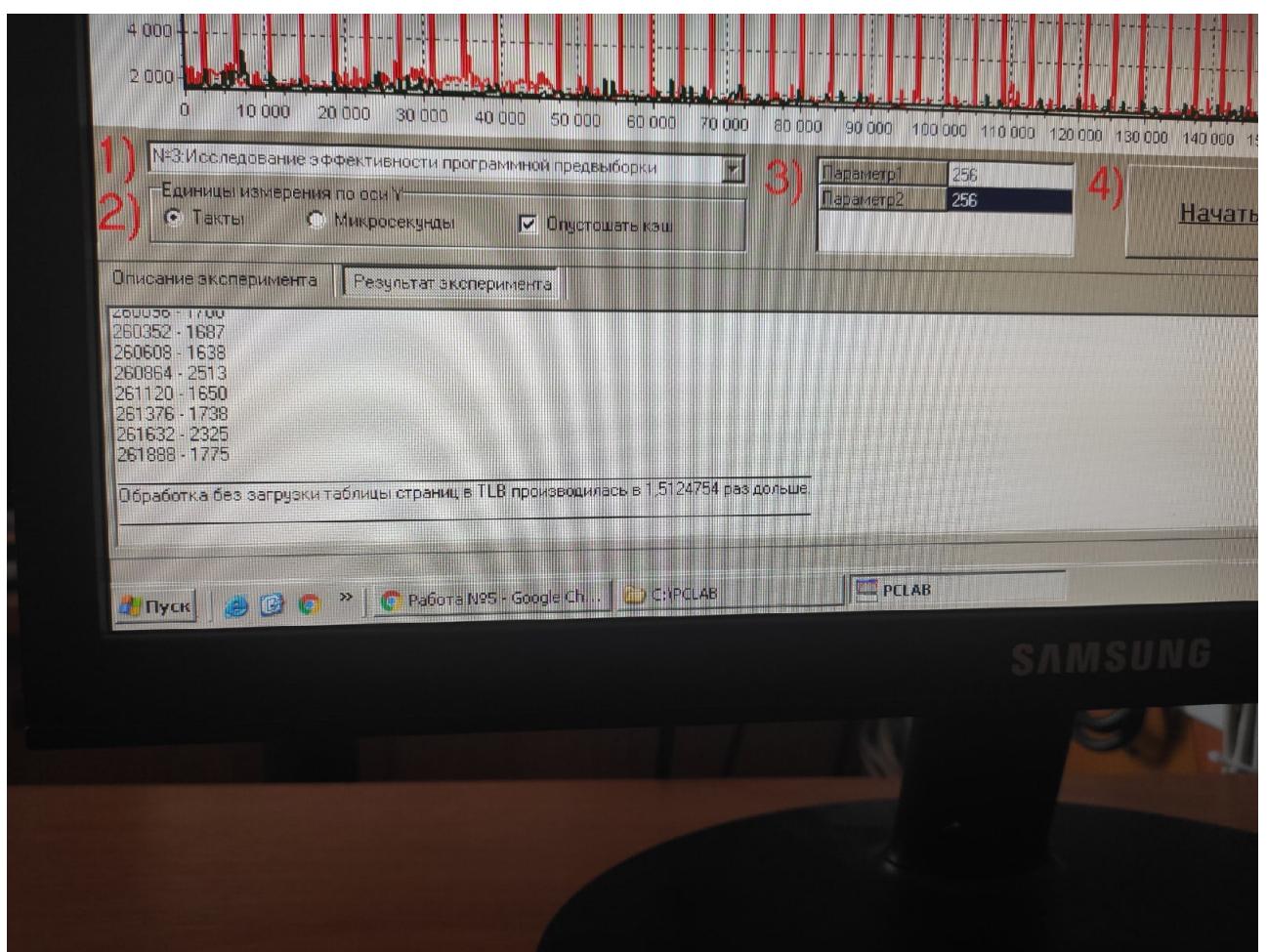


Рисунок 2.5 – Результат сравнения времени работы

Также на рисунке 2.5 представлен результат сравнения, на котором видно, что обработка без загрузки страниц в TLB в 1.5124754 раза дольше.

2.3.6 Вывод

При использовании заблаговременной загрузки страниц можно получить ускорение работы программы.

2.4 Исследование способов эффективного чтения оперативной памяти

Цель эксперимента – исследование возможности ускорения вычислений благодаря использованию структур данных, оптимизирующих механизм чтения оперативной памяти.

2.4.1 Исходные данные

1. Адресное расстояние между банками памяти.
2. Размер буфера чтения.

2.4.2 Результаты эксперимента

Отношение времени обработки блока памяти неоптимизированной структуры ко времени обработки блока структуры, обеспечивающей эффективную загрузку и параллельную обработку данных

2.4.3 Описание проблемы

При обработке информации, находящейся в нескольких страницах и банках оперативной памяти возникают задержки, связанные с необходимостью

открытия и закрытия страниц DRAM памяти. При программировании на языках высокого уровня такая ситуация наблюдается при интенсивной обработке нескольких массивов данных или обработке многомерных массивов. При этом процессоры, в которых реализованы механизмы аппаратной предвыборки, часто не могут организовать эффективную загрузку данных. Кроме этого, объемы запрошенных данных оказываются заметно меньше размера пакета, передаваемого из оперативной памяти. Таким образом, эффективная обработка нескольких векторных структур данных без их дополнительной оптимизации не использует в должной степени возможности аппаратных ресурсов. Для создания структур данных, оптимизирующих их обработку современными процессорами, требуется максимально исключить несвоевременную передачу данных, т.е. передавать в каждом пакете только востребованную для вычислений информацию. В результате такой оптимизации снижается количество кэш-промахов, сокращается количество открытий и закрытий страниц DRAM-памяти, обеспечивается параллельная обработка данных и выполнение операций загрузки и выгрузки.

2.4.4 Суть эксперимента

Для сравнения производительности алгоритмов, использующих оптимизированные и неоптимизированные структуры данных используется профилировка кода двух подпрограмм, каждая из которых должна выполнить обработку нескольких блоков оперативной памяти. В алгоритмах обрабатываются двойные слова данных (4 байта), что существенно меньше размера пакета (32 – 128 байт). Неоптимизированный вариант структуры данных представляет собой несколько массивов в оперативной памяти, в то время как оптимизированная структура состоит из чередующихся данных каждого массива.

2.4.5 Проведение эксперимента

Изменяемые параметры.

1. Размер массива = 2.

2. Количество потоков данных = 64.

На графике, который представлен на рисунке 2.6:

- **красная линия** – количество тактов работы алгоритма, который использует неоптимизированную структуру;
- **зеленая линия** – количество тактов работы алгоритма с использованием оптимизированной структуры.

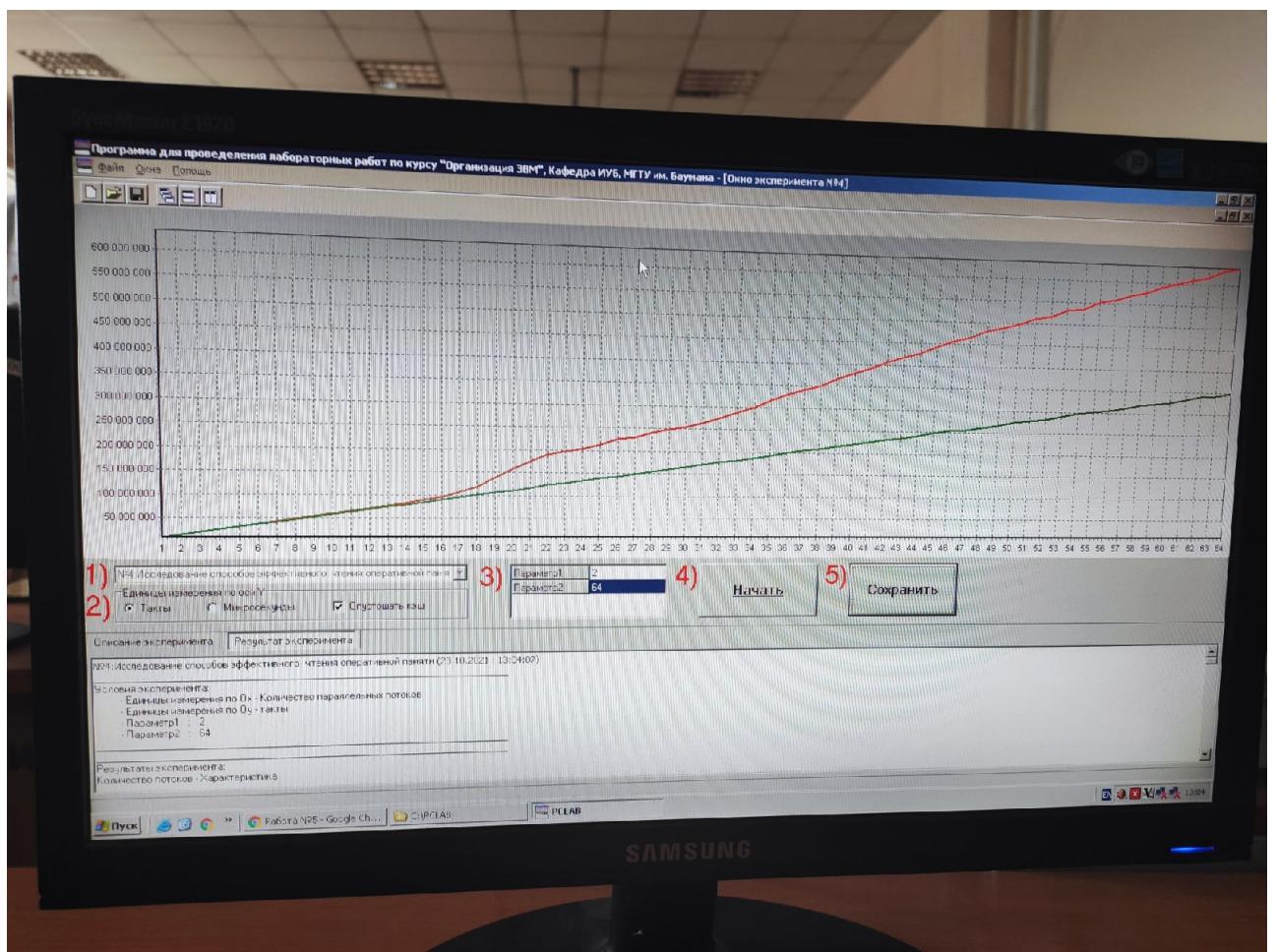


Рисунок 2.6 – Результат выполнения эксперимента в PCLAB

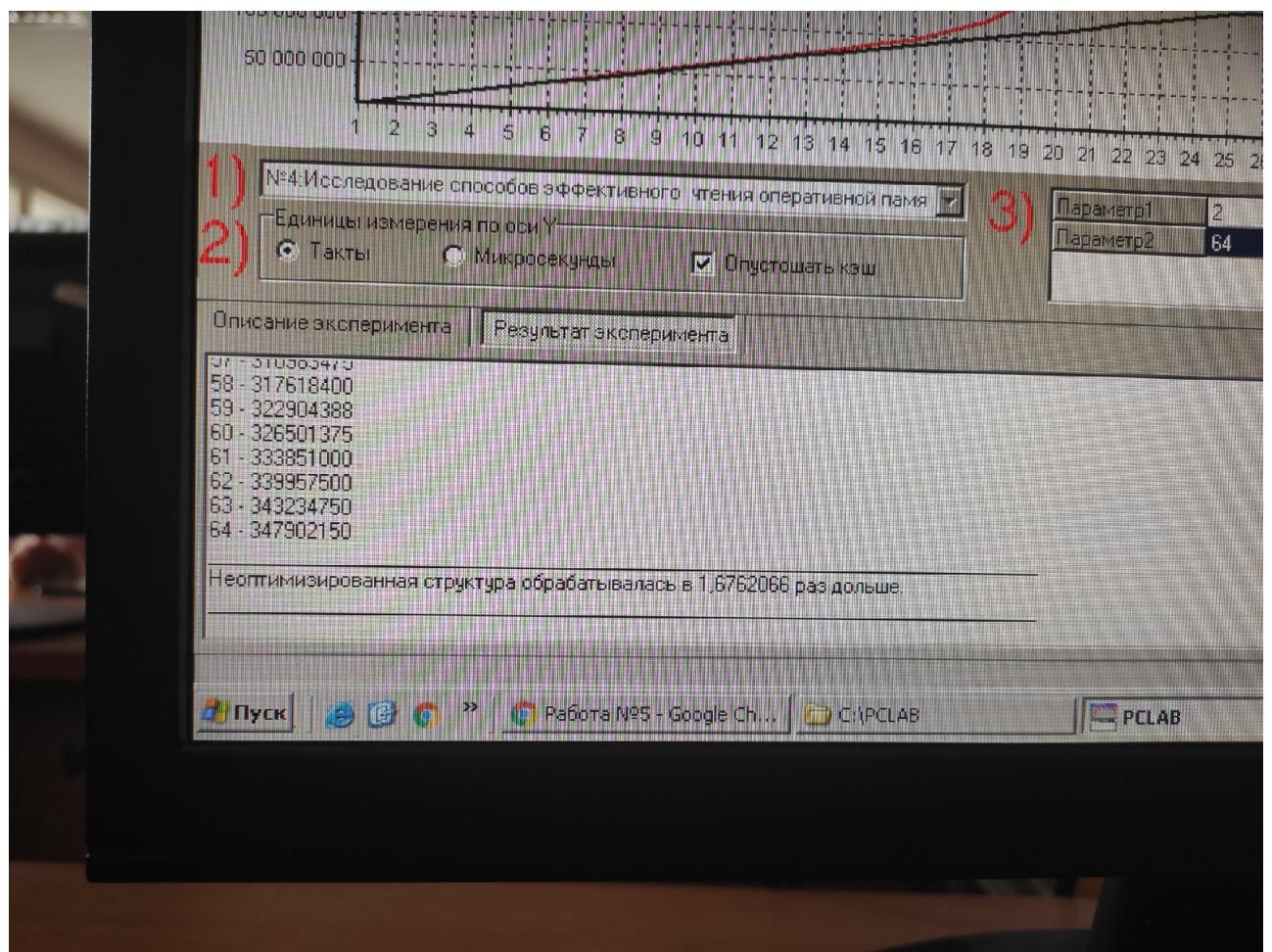


Рисунок 2.7 – Результат сравнения времени работы

Также на рисунке 2.7 представлен результат сравнения, на котором видно, что неоптимизированная структура обрабатывается в 1.6762066 раза дольше.

2.4.6 Вывод

Чем лучше упорядочены данные, тем быстрее работает алгоритм.

2.5 Исследование конфликтов в кэш-памяти

Цель эксперимента – исследование влияния конфликтов кэш-памяти на эффективность вычислений.

2.5.1 Исходные данные

1. Размер банка кэш-памяти данных первого и второго уровня.
2. Степень ассоциативности кэш-памяти первого и второго уровн.
3. Размер линейки кэш-памяти первого и второго уровня.

2.5.2 Результаты эксперимента

Отношение времени обработки массива с конфликтами в кэш-памяти к времени обработки массива без конфликтов.

2.5.3 Описание проблемы

Наборно-ассоциативная кэш-память состоит из линеек данных, организованных в несколько независимых банков. Выбор банка для каждой порции кэшируемых данных выполняется по ассоциативному принципу, т.е. из условия улучшения представительности выборки, в то время как целевая линейка

в каждом из банков жестко определяется по младшей части физического адреса. Совокупность таких линеек всех банков принято называть набором. Таким образом, попытка читать данные из оперативной памяти с шагом, кратным размеру банка, приводит к их помещению в один и тот же набор. Если же количество запросов превосходит степень ассоциативности кэш-памяти, т.е. количество банков или количество линеек в наборе, то наблюдается постоянное вытеснение данных из кэш-памяти, причем больший ее объем остается незадействованным.

2.5.4 Суть эксперимента

Для определения степени влияния конфликтов в кэш-памяти на эффективность вычислений используется профилировка двух процедур чтения и обработки данных. Первая процедура построена таким образом, что чтение данных выполняется с шагом, кратным размеру банка. Это порождает постоянные конфликты в кэш-памяти. Вторая процедура оптимизирует размещение данных в кэш с помощью задания смещения востребованных данных на некоторый шаг, достаточный для выбора другого набора. Этот шаг соответствует размеру линейки.

2.5.5 Проведение эксперимента

Изменяемые параметры.

1. Размер банка кэш-памяти = 256.
2. Размер линейки кэш-памяти = 32.
3. Количество читаемых линеек = 64.

На графике, который представлен на рисунке 2.8:

- **красная линия** – количество тактов работы функции, которая читает данные с конфликтами в кэш-памяти;

- **зеленая линия** – количество тактов работы функции, которая не вызывает конфликтов в кэш-памяти.

При этом на оси абсцисс отражено смещение читаемой ячейки от начала блока данных.

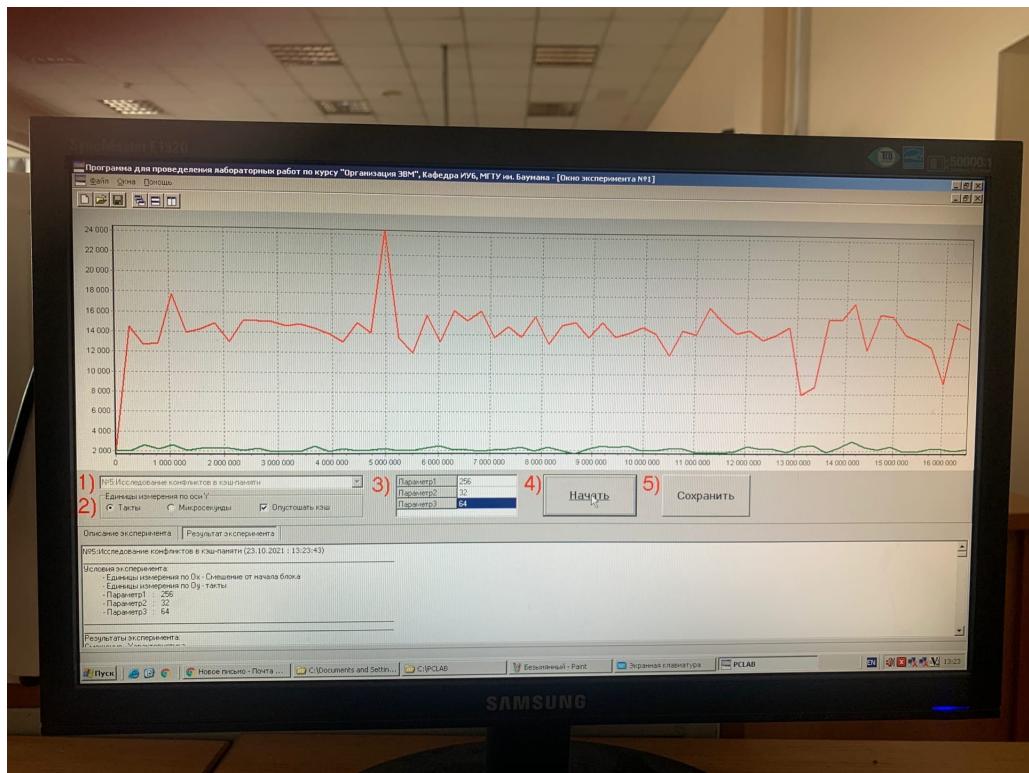


Рисунок 2.8 – Результат выполнения эксперимента в PCLAB

В результате работы сравнения было определено, что с конфликтами данных производится в 6.2300479 раза дольше.

2.5.6 Вывод

Использование кэш-памяти позволяет значительно ускорить работу процессора.

2.6 Сравнение алгоритмов сортировки

Цель эксперимента – исследование способов эффективного использования памяти и выявление наиболее эффективных алгоритмов сортировки, применяемых в вычислительных системах.

2.6.1 Исходные данные

1. Количество процессоров вычислительной системы.
2. Размер пакета.
3. Количество элементов в массиве.
4. Разрядность элементов массива.

2.6.2 Результаты эксперимента

Отношение времени сортировки массива алгоритмом QuickSort ко времени сортировки алгоритмом Radix-Counting Sort и ко времени сортировки Radix-Counting Sort, оптимизированной под 8-процессорную вычислительную систему.

2.6.3 Описание проблемы

Существует несколько десятков алгоритмов сортировки. Их можно классифицировать по таким критериям, как: назначение (внутренняя и внешняя сортировки), вычислительная сложность (алгоритмы с вычислительными сложностями $O(n^2)$, $O(n * \log(n))$, $O(n)$, $O(n / \log(n))$), емкостная сложность (алгоритмы, требующие и нетребующие дополнительного массива), возможность распараллеливания (нераспараллелиемые, ограниченно распараллелиемые, полностью распараллелиемые), принцип определения порядка (алгоритмы, использующие парные сравнения и неиспользующие парные сравнения).

2.6.4 Суть эксперимента

Эксперимент основан на замере времени трех вариантов алгоритмов сортировки (Quick Sort, Radix-Counting Sort, Оптимизированный Radix-CountingSort)

2.6.5 Проведение эксперимента

Изменяемые параметры.

1. Количество 64-х разрядных элементов массивов = 2.
2. Шаг увеличения размера массива = 128.

На графике, который представлен на рисунке 2.9:

- **фиолетовая линия** – количество тактов работы алгоритма QuickSort;
- **красная линия** – количество тактов работы алгоритма неоптимизированного алгоритма Radix-Counting;
- **зеленая линия** – количество тактов работы оптимизированного под 8-процессорную вычислительную систему алгоритма Radix-Counting.

При этом на оси абсцисс отражено количество 64-разрядных элементов сортируемых массивов, а на оси ординат – количество тактов.

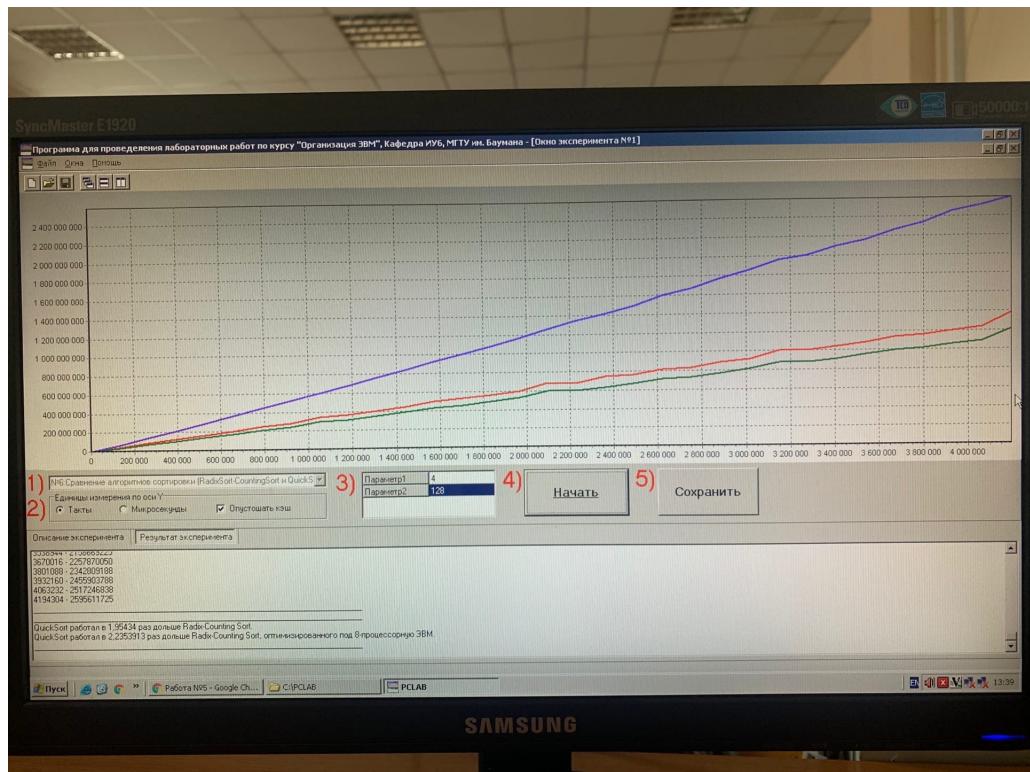


Рисунок 2.9 – Результат выполнения эксперимента в PCLAB

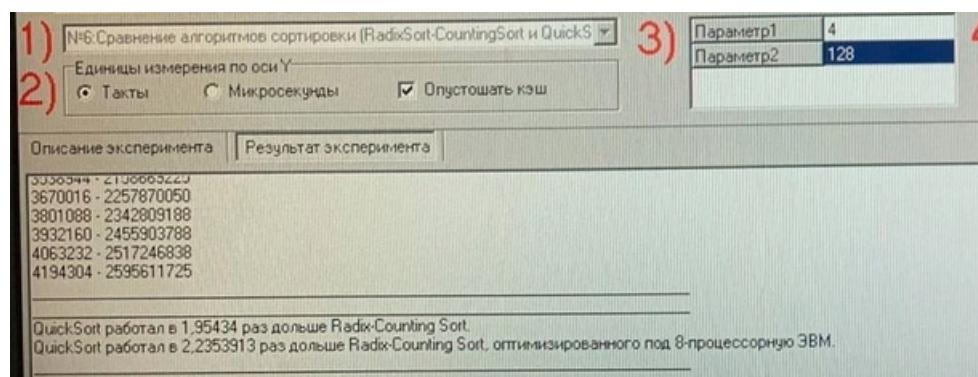


Рисунок 2.10 – Результат сравнения времени работы

Также на рисунке 2.10 представлен результат сравнения, на котором видно, что QuickSort работал в 1.95434 раза дольше, чем Radix-Counting Sort, и в 2.353913 раза дольше, чем Radix-Counting Sort, оптимизированного под 8-процессорную ЭВМ.

2.6.6 Вывод

Radix-Counting Sort является более быстрой сортировкой, чем QuickSort, при этом даже Radix-Counting Sort можно улучшить, оптимизировав его под 8-процессорную ЭВМ.

Заключение

В данной лабораторной работе были рассмотрены различные ситуации для оптимизации того иного алгоритма, которые позволяют в разы ускорить их скорость работы, используя специальные подходы.