**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»(ИУ7) НАПРАВЛЕНИЕ ПОДГОТОВКИ 09.03.04 «Программная инженерия»

## О Т Ч Е Т

## по лабораторной работе № 3

**Название** Организация памяти конвейерных суперскалярных электронных вычислительных машин ‘

**Дисциплина** Архитектура элекронно-вычислительных машин

|  |  |  |
| --- | --- | --- |
| Студент: |  | Чыонг Н.В.У. |
| Преподаватель: | подпись, дата | Фамилия, И.О.  Попов А. Ю. |
|  | подпись, дата | Фамилия, И. О. |

Москва — 2022 г.

**Оглавление**

[Цель работы](#_bookmark0) 2

[Основные теоретические сведения](#_bookmark1) 3

[Эксперимент №1: Исследования расслоения динамической па-](#_bookmark2) [мяти](#_bookmark2) 4

[Эксперимент №2: Сравнение эффективности ссылочных и век-](#_bookmark6) [торных структур](#_bookmark6) 8

[Эксперимент №3: Исследование эффективности программной](#_bookmark9) [предвыборки](#_bookmark9) 11

[Эксперимент №4: Исследование способов эффективного чте-](#_bookmark12) [ния оперативной памяти](#_bookmark12) 14

[Эксперимент №5: Исследование конфликтов в кэш-памяти](#_bookmark14) 17

[Эксперимент №6: Сравнение алгоритмов сортировки](#_bookmark17) 20

[Контрольные вопросы](#_bookmark20) 23

[Заключение](#_bookmark21) 26

# Цель работы

Основной целью работы является освоение принципов эффективного использования подсистемы памяти современных универсальных ЭВМ, обес- печивающей хранение и своевременную выдачу команд и данных в цен- тральное процессорное устройство. Работа проводится с использованием программы для сбора и анализа производительности PCLAB.

# Основные теоретические сведения

Программа PCLAB предназначена для исследования производительно- сти x86 совместимых ЭВМ c IA32 архитектурой, работающих под управле- нием операционной системы Windows (версий 95 и старше). Исследование организации ЭВМ заключается в проведении ряда экспериментов, направ- ленных на построение зависимостей времени обработки критических участ- ков кода от изменяемых параметров. Набор реализуемых программой экс- периментов позволяет исследовать особенности построения современных подсистем памяти ЭВМ и процессорных устройств, выявить конструктив- ные 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 исследова- нии конвейерных суперскалярных процессорных устройств, таких как 32-х разрядные процессоры фирмы Intel или AMD, способных выполнять пере- упорядоченную обработку последовательности команд программы, требу- ется использовать специальные средства измерения временных интервалов и запрещения переупорядочивания микрокоманд.

# Эксперимент №1: Исследования рассло- ения динамической памяти

### Цель эксперимента

Определение способа трансляции физического адреса, используемого при обращении к динамической памяти.

### Описание проблемы

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

### Исходные данные

Размер линейки кэш-памяти верхнего уровня; объем физическойь па- мяти.

### Результаты эксперимента

На рисунке [1](#_bookmark3) предаставлен график, полученный в результате экспери- мента с исходными параметрами:

* Максимальное расстояния между читаемыми блоками (К) = 128;
* Шаг увеличения расстояния между читаемыми 4-х байтовыми ячей- ками (Б) = 128;
* Размер массива (М) = 1.



Рисунок 1 – Эксперимет №1, случай 1

На рисунке [2](#_bookmark4) предаставлен график, полученный в результате экспери- мента с исходными параметрами:

* Максимальное расстояния между читаемыми блоками (К) = 128;
* Шаг увеличения расстояния между читаемыми 4-х байтовыми ячей- ками (Б) = 32;
* Размер массива (М) = 1.

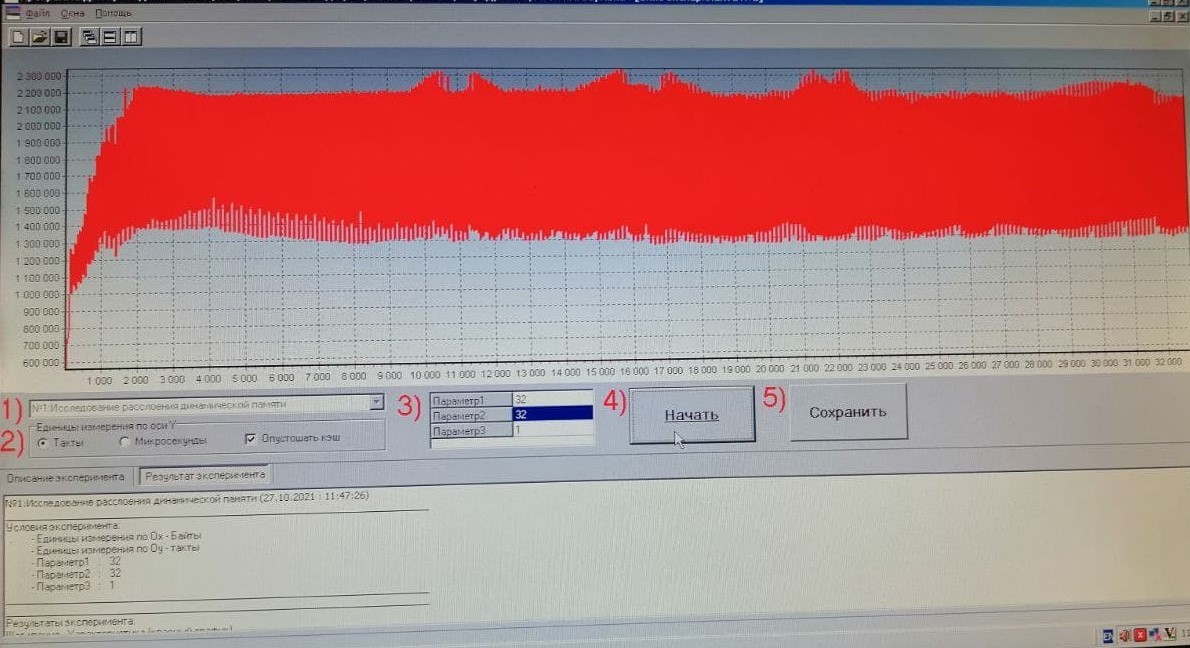


Рисунок 2 – Эксперимет №1, случай 2

На рисунке [3](#_bookmark5) предаставлен график, полученный в результате экспери- мента с исходными параметрами:

* Максимальное расстояния между читаемыми блоками (К) = 128;
* Шаг увеличения расстояния между читаемыми 4-х байтовыми ячей- ками (Б) = 64;
* Размер массива (М) = 1.

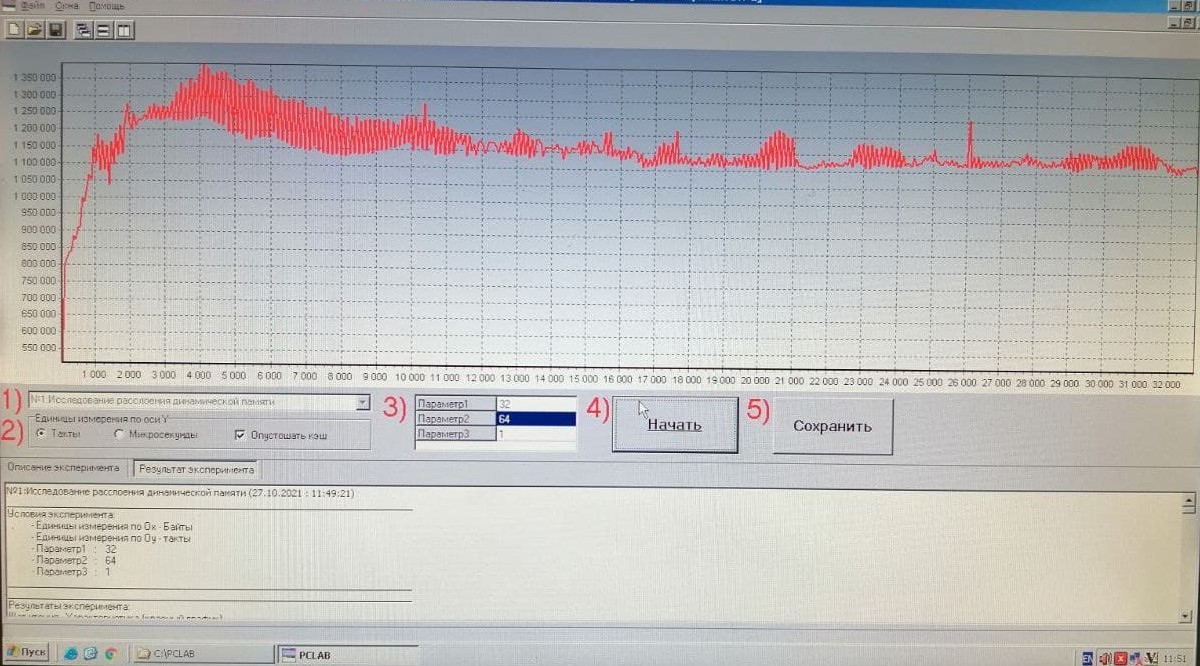


Рисунок 3 – Эксперимет №1, случай 3

Первый пик, который мы видим на рисунке [1](#_bookmark3) – адресное расстояние между страницей в одном банке и следующей страницей в том же банке. Это адресное расстояние равно 1024.

При просмотре результатов эксперимента № 1 случая 2, где параметр "Шаг увеличения расстояния между читаемыми 4-х байтовыми ячейка- ми"равен 64, можно заметить, что первый запрос на 64б ждет нормальное количество времени, а вот вторые ждут меньше, третьи опять нормальное количество времени и т.д. То есть можно сказать, что размер одной страни- цы у нас равен 128б, так как когда мы делаем запрос данных с какой-либо страницы, она загружается полностью, и тогда следующий запрос данных к этой странице уже будет длиться меньше по количеству времени.

Тогда можно сказать, что если "размер страницы- "размер динамиче- ской памяти"/ "количество страниц то имея результаты приведенные вы- ше, количество страниц = 1024*/*128 = 8.

### Вывод

Оперативная память неоднородна, и для обращения к последовательно расположенными данным может потребоваться различное количество вре- мени. Поэтому, при создании программ необходимо учитывать расслоение памяти при обработке данных.

# Эксперимент №2: Сравнение эффектив- ности ссылочных и векторных структур

### Цель эксперимента

Оценка влияния зависимости команд по данным на эффективность вы- числений.

### Описание проблемы

Обработка зависимых данных происходит в тех случаях, когда резуль- тат работы одной команды используется в качестве адреса операнда дру- гой. При программировании на языках высокого уровня такими операнда- ми являются указатели, активно используемые при обработке ссылочных структур данных: списков, деревьев, графов. Обработка данных структур процессорами с длинными конвейерами команд приводит к заметному уве- личению времени работы алгоритмов: адрес загружаемого операнда стано- вится известным только после обработки предыдущей команды. В проти- воположность этому, обработка векторных структур, таких как массивы, позволяет эффективно использовать аппаратные возможности ЭВМ.

### Результаты эксперимента

На рисунке [4](#_bookmark7) предаставлен график, полученный в результате экспери- мента с исходными параметрами:

* Количество элементов в списке (М) = 1;
* Максимальная фрагментации списка (К) = 32;
* Шаг увеличения фрагментации (К) = 1.

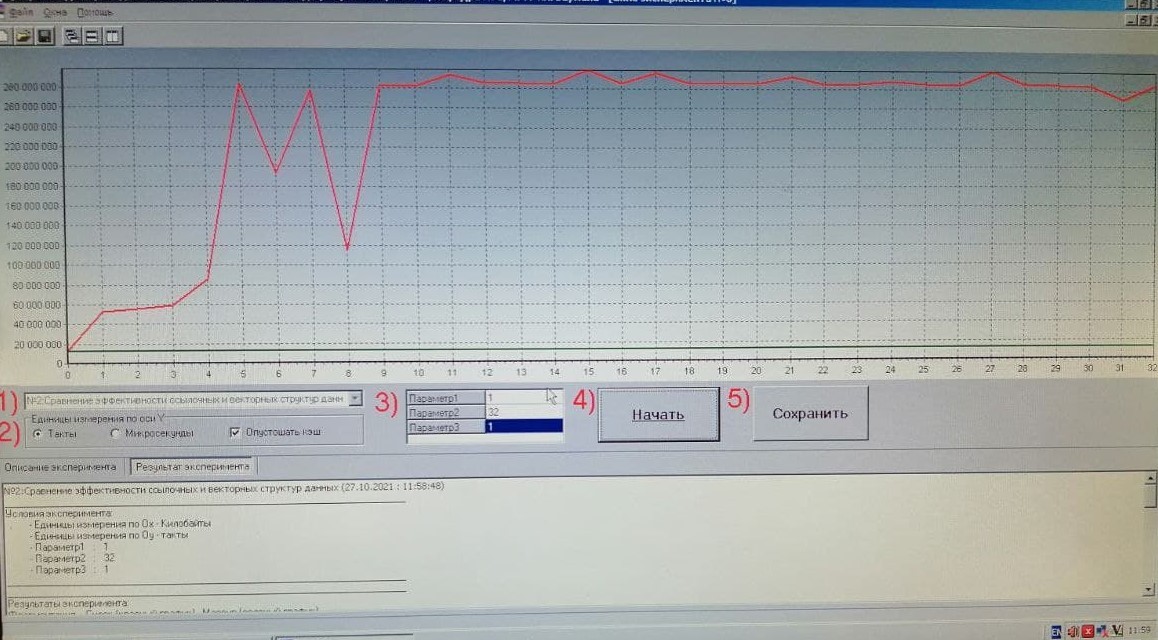


Рисунок 4 – Эксперимет №2

Результат сравнения времени (как вывод программы) представлен на рисунке [5.](#_bookmark8) Как видно на рисунке, список обрабатывается почти в 20 раз медленнее.

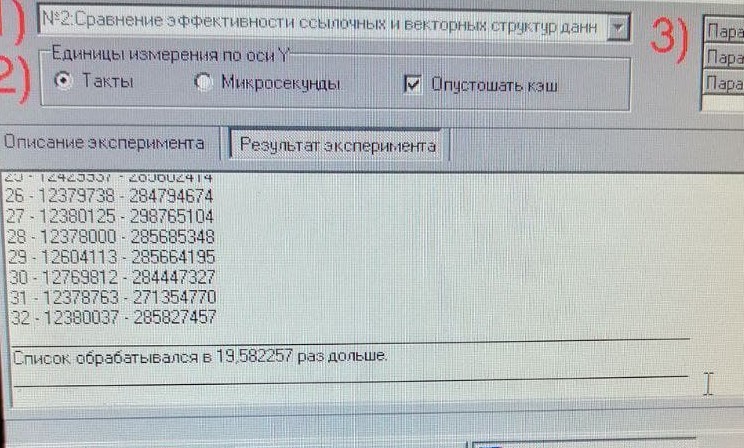


Рисунок 5 – Эксперимет №2, результат

### Вывод

Вывод из полученных результатов можно сделать следующий: исполь- зовать структуры данных надо с учетом технологического фактора опреде-

ленной задачи. Если решаемая задача продполагает возможность исполь- зования массива, то надо использовать его, особенно если использование списка не дает существенной разницы (особенно выиграша во времени).

# Эксперимент №3: Исследование эффек- тивности программной предвыборки

### Цель эксперимента

Выявление способов ускорения вычислений благодаря применению пред- выборки данных.

### Исходные данные

Cтепень ассоциативности и размер TLB данных.

### Описание проблемы

Обработка больших массивов информации сопряжена с открытием боль- шого количества физических страниц памяти. При первом обращении к странице памяти наблюдается увеличенное время доступа к данным. Это связано с необходимостью преобразования логического адреса в физиче- ский адрес памяти, а также c 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 обра- щение 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 допол- нительному запросу небольшого количества данных из оперативной памя- ти.

### Результаты эксперимента

На рисунке [6](#_bookmark10) предаставлен график, полученный в результате экспери- мента с исходными параметрами:

* Шаг увеличения расстояния между читаемыми данными (Б) = 512;
* Размер массива (К) = 128.

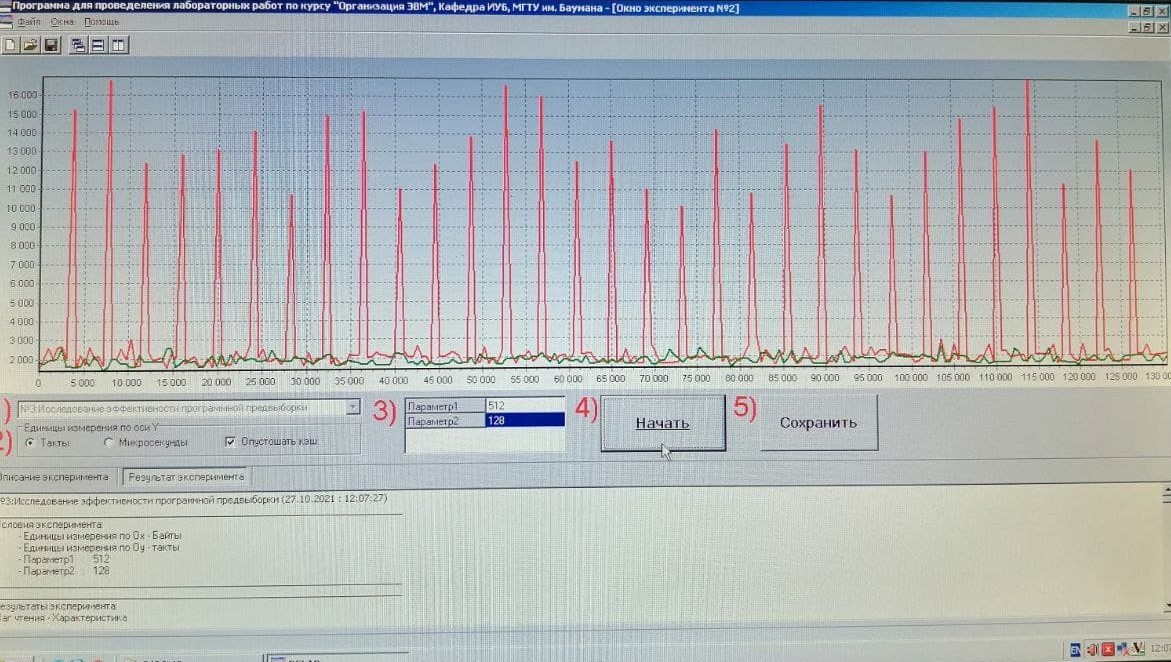


Рисунок 6 – Эксперимет №3

Результат сравнения времени (как вывод программы) представлен на рисунке [7.](#_bookmark11) Как видно на рисунке, обработка без загрузки таблицы страниц в TLB производилась в 3,7639087 раз дольше.

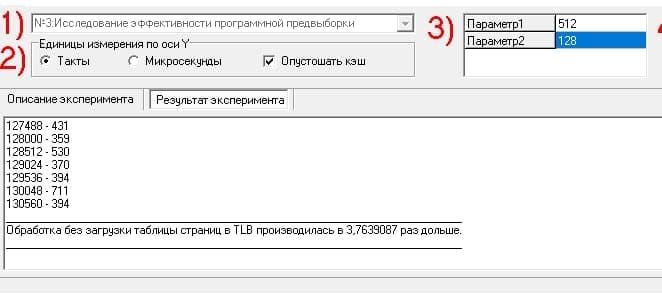


Рисунок 7 – Эксперимет №3, результат

Красный график - чтение страниц последовательно из оперативной па- мяти. Зеленый график - чтение страниц, используя дополнительный цикл предвыборки, обеспечивающий своевременную загрузку информации в TLB данных.

Сокращение времени работы алгоритма, который использует предвы- борку, происходит в том случае, когда информация о востребованных стра- ницах помещается в TLB.

Пики на красном графике происходят из-за того, что процессу необо- димо преобразовать физический адрес в логический.

### Вывод

Используя предвыборку можно ускорить время работы программы по- чти в 4 раза (как результат представлен 3.7) за счет заблаговременной загрузки страниц в память.

# Эксперимент №4: Исследование спосо- бов эффективного чтения оперативной памяти

### Цель эксперимента

Исследование возможности ускорения вычислений благодаря использо- ванию структур данных, оптимизирующих механизм чтения оперативной памяти.

### Исходные данные

Адресное расстояние между банками памяти, размер буфера чтения.

### Описание проблемы

При обработке информации, находящейся в нескольких страницах и банках оперативной памяти возникают задержки, связанные с необходи- мостью открытия и закрытия страниц DRAM памяти. При программиро- вании на языках высокого уровня такая ситуация наблюдается при интен- сивной обработке нескольких массивов данных или обработке многомерных массивов. При этом процессоры, в которых реализованы механизмы аппа- ратной предвыборки, часто не могут организовать эффективную загрузку данных. Кроме этого, объемы запрошенных данных оказываются замет- но меньше размера пакета, передаваемого из оперативной памяти. Таким образом, эффективная обработка нескольких векторных структур данных без их дополнительной оптимизации не использует в должной степени воз- можности аппаратных ресурсов.

### Результаты эксперимента

На рисунке [6](#_bookmark10) предаставлен график, полученный в результате экспери- мента с исходными параметрами:

* Размер массива (М) = 2;
* Количество потоков данных = 64.

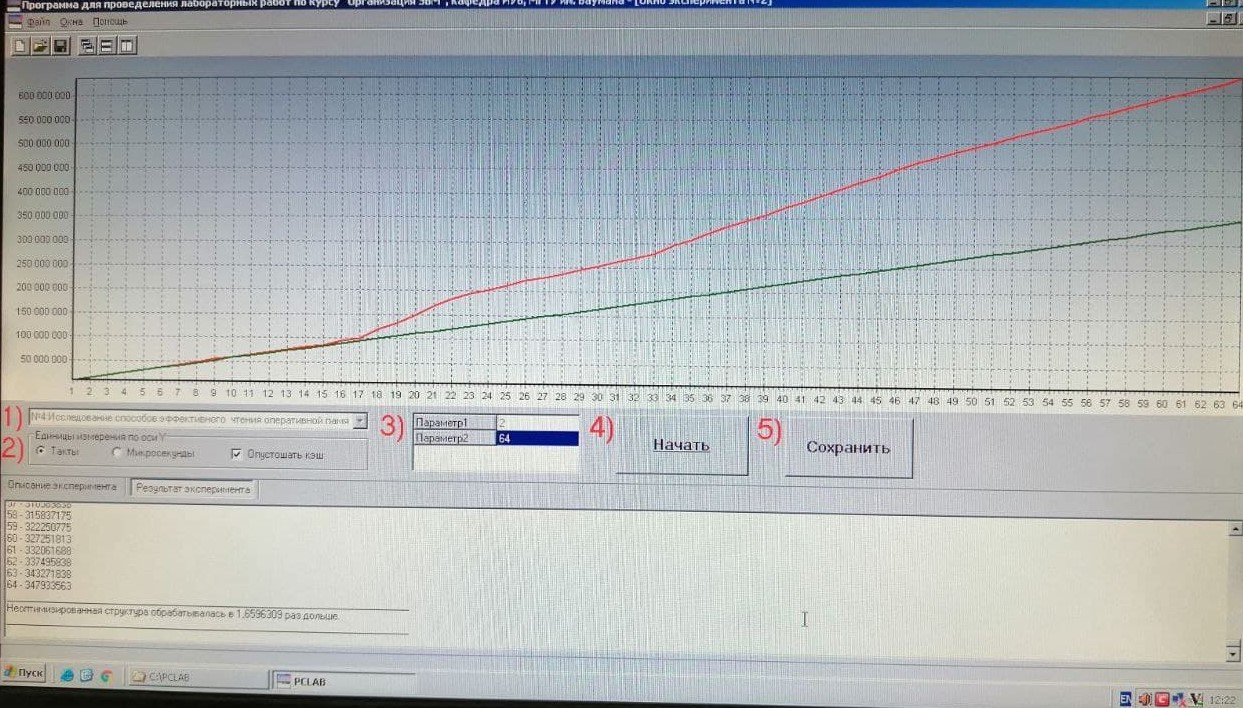


Рисунок 8 – Эксперимет №4

Результат сравнения времени (как вывод программы) представлен на рисунке [9.](#_bookmark13) Как видно на рисунке, неоптимизированная структура обраба- тывалась в 1,6596309 раз дольше.

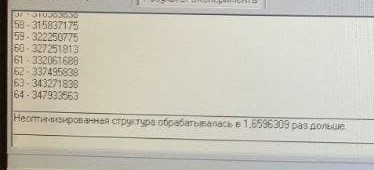


Рисунок 9 – Эксперимет №4, результат

Красный график показывает время или количество тактов работы ал- горитма, использующего неоптимизированную структуру.

Зеленый график показывает время (или количество тактов) работы ал- горитма с использованием оптимизированной структуры.

Оптимизация заключается в том, что струтура данных, ускоряющая обработку современным процессорам, пытается максимально исключить несвоевременную передачу данных, то есть передавть только лишь вос- требованную для вычислений инфорацию. Поэтому снижается количество открытий и закрытий страниц DRAM-памяти и обеспечивается параллель- ная обработка данных, а также выполнение операций загрузки и выгрузки.

### Вывод

Можно сделать вывод, что для ускорения работы алгоритмов, необхо- димо правильно упорядочить данные.

# Эксперимент №5: Исследование конфлик- тов в кэш-памяти

### Цель эксперимента

Исследование влияния конфликтов кэш-памяти на эффективность вы- числений.

### Исходные данные

Размер банка кэш-памяти данных первого и второго уровня, степень ассоциативности кэш-памяти первого и второго уровня, размер линейки кэшпамяти первого и второго уровня.

### Описание проблемы

Наборно-ассоциативная кэш-память состоит из линеек данных, органи- зованных в несколько независимых банков. Выбор банка для каждой пор- ции 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 физи- ческого 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 на- блюдается постоянное вытеснение данных из кэш-памяти, причем больший ее объем остается незадействованным.

### Результаты эксперимента

На рисунке [10](#_bookmark15) предаставлен график, полученный в результате экспери- мента с исходными параметрами:

* Размер банка кэш-памяти (К) = 128;
* Размер линейки кэш-памяти (б) = 128;
* Количество читаемых линеек = 32.



Рисунок 10 – Эксперимет №5

Результат сравнения времени (как вывод программы) представлен на рисунке [11.](#_bookmark16) Как видно на рисунке, чтение с конфликтами банков произво- дилось в 9,1083954 раз дольше.

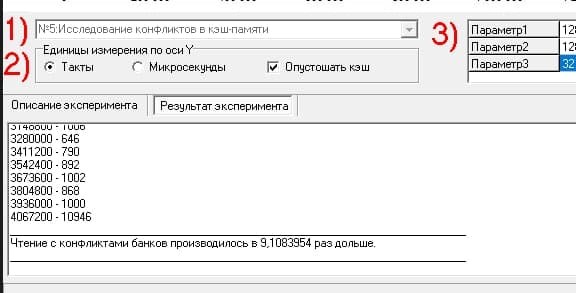


Рисунок 11 – Эксперимет №5, результат

Красный график показывает время или количество тактов работы про- цедуры, читающей данные с конфликтами в кэш-памяти.

Зеленый график показывает время или количество тактов работы про- цедуры, не вызывающей конфликтов в кэш-памяти. Ось абсцисс отражает смещение читаемой ячейки от начала блока данных.

Красный график соответствует алгоритму, который построен таким об- разом, что чтение данных выполняеся с шагом, кратным размеру банка. Именно это и порождает постоянные конфликты в кэш-памяти.

Зеленый график соответсвует алгоритму, который оптимизируется раз- мещение данных в кэш с помощью задания смещения востребованных дан- ных на шаг, достаточный для выбора другого набора. (Шаг соответствует размеру линейки).

### Вывод

Можно сделать вывод, что использование кэш-памяти работа процессо- ра ускоряется почти в 10 раз.

# Эксперимент №6: Сравнение алгорит- мов сортировки

### Цель эксперимента

Исследование способов эффективного использования памяти и выявле- ние наиболее эффективных алгоритмов сортировки, применимых в вычис- лительных системах.

### Исходные данные

Количество процессоров вычислительной системы, размер пакета, ко- личество элементов в массиве, разрядность элементов массива.

### Описание проблемы

Существует несколько десятков алгоритмов сортировки. Их можно клас- сифицировать по таким критериям, как: назначение (внутренняя и внеш- няя сортировки), вычислительная сложность (алгоритмы с вычислитель- ными сложностями *𝑂*(2)*, 𝑂*(*\*𝑔*())*, 𝑂*()*, 𝑂*(*/𝑔*())), емкостная слож- ность (алгоритмы, требующие и не требующие дополнительного масси- ва), возможность распараллеливания (не распараллеливаемые, ограничен- но распараллеливаемые, полностью распараллеливаемые), принцип опре- деления порядка (алгоритмы, использующие парные сравнения и не ис- пользующие парные сравнения).

#### Radix Sort

Логика данной сортировки проста. Допустим, у нас есть массив из 10 чисел.

Сначала идет сортировка их по первому (старшему) разряду. Сортиров- ка в таком случае выполняется с помощью сортировки подсчетом (count sort). Сложность — *𝑂*().

В итоге получается 10 «корзин» — в которых старший разряд 0, 1, 2 и т.д.

Далее в каждой из корзин запускаем ту же процедуру, но только рас- сматриваем уже не старший разряд, а следующий за ним, и т.д.

Такие действия выполняется до последнего разряда.

### Результаты эксперимента

На рисунке [12](#_bookmark18) предаставлен график, полученный в результате экспери- мента с исходными параметрами:

* Количество 64-х разрядных элементов массивов (М) = 1;
* Шаг увеличения размера массива (К) = 32.

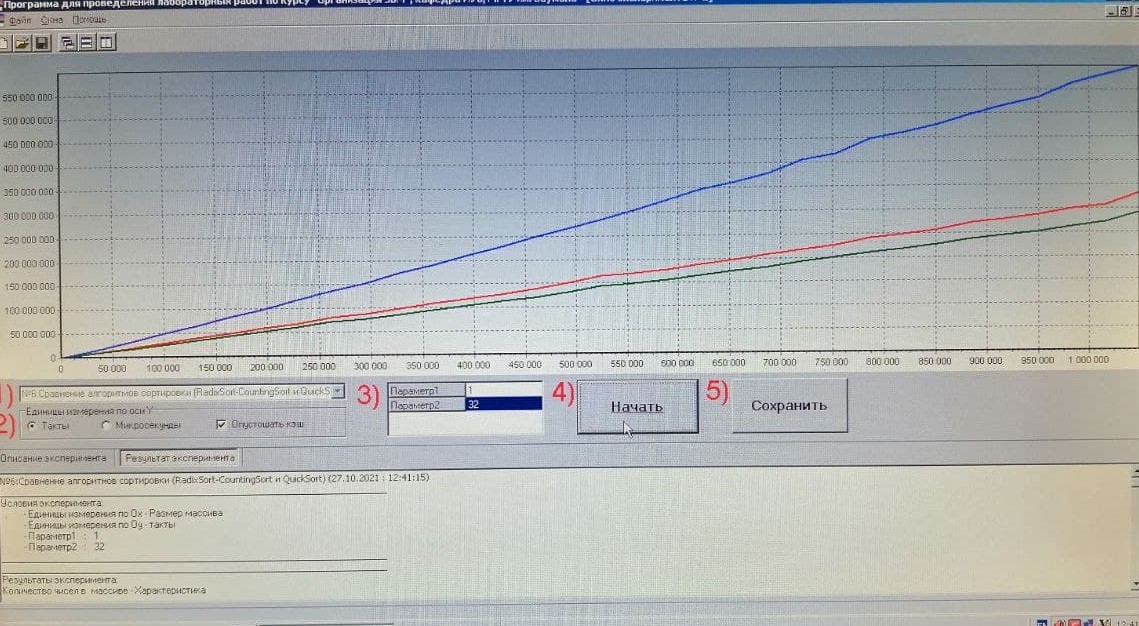


Рисунок 12 – Эксперимет №6

Результат сравнения времени (как вывод программы) представлен на рисунке [13.](#_bookmark19) Как видно на рисунке, QuickSort работал в 2,6696797 раз доль- ше Radix-Counting Sort, и QuickSort работал в 4,0447133 раз дольше Radix- Counting Sort, оптимизированного под 8-процессорную ЭВМ.

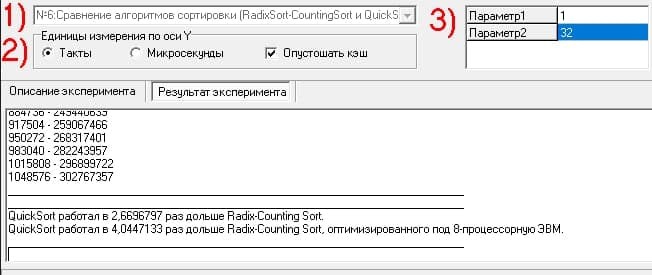


Рисунок 13 – Эксперимет №5, результат

Фиолетовый график показывает время или количество тактов работы алгоритма QuickSort.

Красный график показывает время или количество тактов работы неопти- мизированного алгоритма Radix-Counting.

Зеленый график показывает время или количество тактов работы оп- тимизированного под 8-процессорную вычислительную систему алгоритма Radix-Counting.

### Вывод

Можно сделать вывод о том, что существует сортировка, работающая быстрее чем QuickSort, при этом, даже ее можно еще оптимизировать для более быстрой работы.

# Контрольные вопросы

#### Назовите причины расслоения оперативной памяти.

Скорость обработки данных в процессоре в несколько раз превышает скорость доступа к информации, которая размещена в оперативной памя- ти. Необходимо, чтобы итоговая скорость выполнения команды процессо- ром как можно меньше зависела от скорости доступа к коду команды и к используемым в ней операндам из памяти.

Расслоение ОЗУ — один из аппаратных путей решения проблемы дисба- ланса в скорости доступа к данным, размещенным в оперативной памяти, и производительностью процессора.

Физически ОЗУ представимо в виде объединения k устройств, способ- ных хранить одинаковое количество информации и способных взаимодей- ствовать с процессором независимо друг от друга. При этом адресное про- странство организовано таким образом, что подряд идущие адреса, или ячейки памяти, находятся в соседних устройствах (блоках) оперативной памяти. Тем самым расслоение памяти в идеале увеличивает скорость до- ступа в k раз, плюс буфер команд позволяет сократить обращения к ОЗУ. Поэтому можно сказать, что ОЗУ без расслоения памяти — один кон- троллер на все банки, но ОЗУ с расслоением памяти — каждый банк об-

служивает отдельный контроллер.

#### Как в современных процессорах реализована аппаратная пред- выборка.

Аппаратная предвыборка происходит неявно, без участия программи- ста или компилятора. Кэш-контроллер анализирует, по каким адресам и в каком порядке программа обращается к оперативной памяти, и пыта- ется предугадать, какие данные вскоре могут понадобиться программе, и осуществляет их автоматическую предвыборку в кэш-память.

Разумеется, по последовательности обращений программы в оператив- ную память в общем случае невозможно предсказать, какие данные пона- добятся впоследствии.

Например, обработка массива: если кэш-контроллер обнаруживает, что

оперативная память последовательно опрашивается с некоторым фиксиро- ванным шагом, то он делает предположение, что это проиходит обработка массива, и начинает загружать следующие блоки данных заранее.

Аппаратная предвыборка данных в кэш-память не застрахована от оши- бок. Если кэш-контроллер распознал, что это не обход массива, то данные, которые он загрузит в кэш-память, могут и не понадобиться. Однако хуже ошибки может быть то, что они могут «вытеснить» какие-либо полезные данные, находившиеся в кэш-памяти, и их придется загружать еще раз.

#### Какая информация храниться в TLB.

TLB (Translation-Lookaside Buffer) представляет собой память с ассоци- ативной выборкой, которая содержит 20-тиразрядные базовые адреса 32- х страниц, то есть, старшие 20 разрядов физического адреса страницы. Каждый из базовых адресов имеет свой признак (тег). В качестве тега ис- пользуются старшие 20 разрядов линейного адреса, то есть поля TABLE и PAGE.

Случай, когда базовый адрес страницы находится в TLB, называется КЭШ-попаданием.

Помимо тега для каждого базового адреса страницы в TLB хранится дополнительная информация, позволяющая определить, какую страницу можно заменить на вновь вводимую. Поскольку TLB хранит адреса только 32 страниц объема 4 кб каждая, то микропроцессор может непосредственно формировать физические адреса для 128 Кб памяти.

#### Какой тип ассоциативной памяти используется в кэш-памяти второго уровня современных ЭВМ и почему.

В современных компьютерах применяют кэш-память второго уровня, которая находится между процессором и ОП и еще больше повышает про- изводительность ЭВМ.

Кэш-память 2-го уровня, как правило, унифицирована, т. е. может со- держать как команды, так и данные.

Если она встроена в ядро ЦП, то говорят о S-cache (Secondary Cache, вторичный кэш), в противном случае – о B-cache (Backup Cache, резервный

кэш). В современных серверных ЦП объем S-cache составляет от одного до нескольких мегабайт, a B-cache – до 64 Мбайт.

#### Приведите пример программной предвыборки.

При программной предвыборке программист или компилятор явно встав- ляет в программу команды предвыборки данных по тому или иному адресу в оперативной памяти.

Программные предвыборки потребляют ресурсы в процессоре, и ис- пользование слишком многих предвыборок может ограничить их эффек- тивность.

Примеры таких предвыборок – предвыборку данных в цикле для неза- висимости от информации находящейся вне цикла и предвыборку в основ- ных блоках, которые часто исполняются, но которые редко используют ее не зависимо от целей предвыборки.

**Вывод**

Основаны основные принципы эффективного использования подсисте- мы памяти современных универсальных ЭВМ, обеспечивающей хранение и своевременную выдачу команд и данных в центральное процессорное устройство.

Работа была проведена с использованием программы для сбора и ана- лиза производительности PCLAB.

Поставленная цель достигнута.