СОДЕРЖАНИЕ

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. Лабораторная работа проводится с использованием программы для сбора и анализа производительности PCLAB.

Для достижения данной цели необходимо выполнить следующие задачи:

- ознакомиться с теоретическим материалом, касающимся особенностей функционирования подсистемы памяти современных конвейерных суперскалярных ЭВ;
- изучить возможности программы PCLAB;
- изучить средства идентификации микропроцессоров;
- провести исследования времени выполнения тестовых программ;
- сделать выводы о архитектурных особенностях используемых ЭВМ.

Задание 1

Задание: Ознакомиться с возможностями программы PCLAB в Разделе 2 методических указаний. Запустить программу PCLAB 1.0. Изучить идентификационную информацию на вкладке «Идентификация процессора».

Описание PCLAB: Программа PCLAB предназначена для исследования производительности x86 совместимых ЭВМ с IA32 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.

Задание 2

Задание: На основании идентификационной информации о микропроцессоре ЭВМ, используемой при проведении лабораторной работы, определить следующие параметры: размер линейки кэш-памяти верхнего уровня и объем физической памяти. Результаты занести в отчет.

Размер линейки кэша: 64 байт; Объем физической памяти: 4 Гб.

Задание 3

Задание: Ознакомиться с описанием эксперимента «Исследование расслоения динамической памяти» на вкладке «Описание эксперимента». Провести эксперимент. По результатам эксперимента определить: количе-

ство банков динамической памяти; размер одной страницы динамической памяти; количество страниц в динамической памяти. Сделать выводы о использованном способе наращивания динамической памяти. Результаты занести в отчет.

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

Исходные данные:

- Размер линейки кэша: 64 байт;
- Объем физической памяти: 4 Гбайт.

В таблице 1.1 приведены настраиваемые параметры.

Таблица 1.1 — Настраиваемые параметры

Эксперимент	№	Значение	Описание
1	1	32 Кбайт	Максимальное расстояния между читаемыми блоками
	2	128 байт	Шаг увеличения расстояния между читаемыми 4-х байтовыми ячейками
	3	1 Мбайт	Размер массива
2	1	32 Кбайт	Максимальное расстояния между читаемыми блоками
	2	64 байт	Шаг увеличения расстояния между читаемыми 4-х байтовыми ячейками
	3	1 Мбайт	Размер массива

Ниже, на рисунках 1.1 - 1.3 приведены зависимости времени обращения к памяти от расстояния между читаемыми блоками данных.

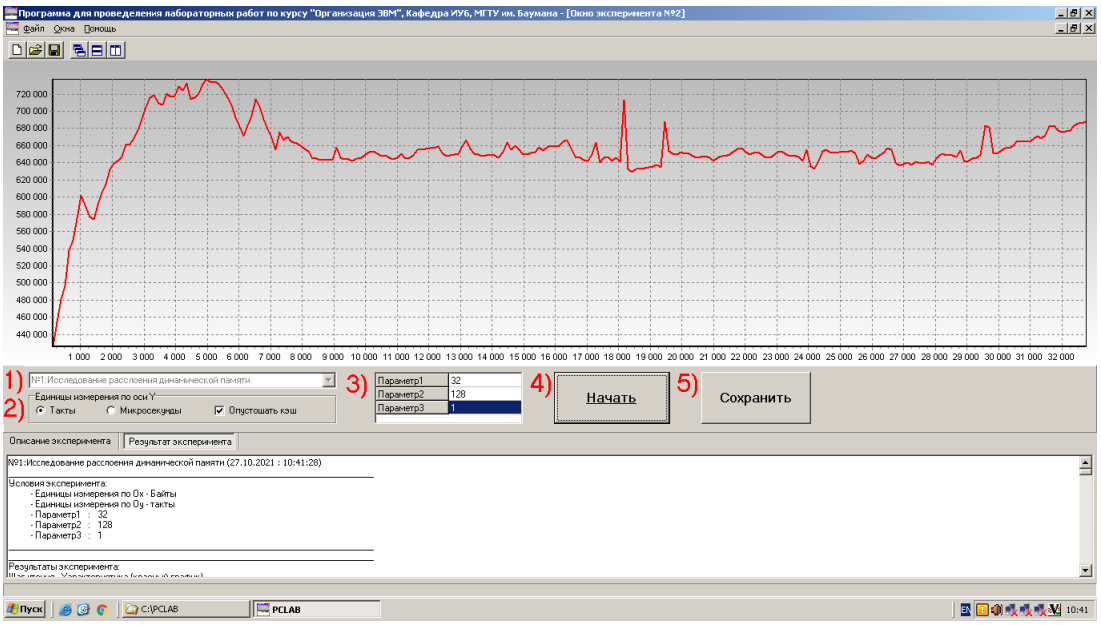

Рисунок 1.1 — Результаты исследования расслоения динамической памяти (часть 1)

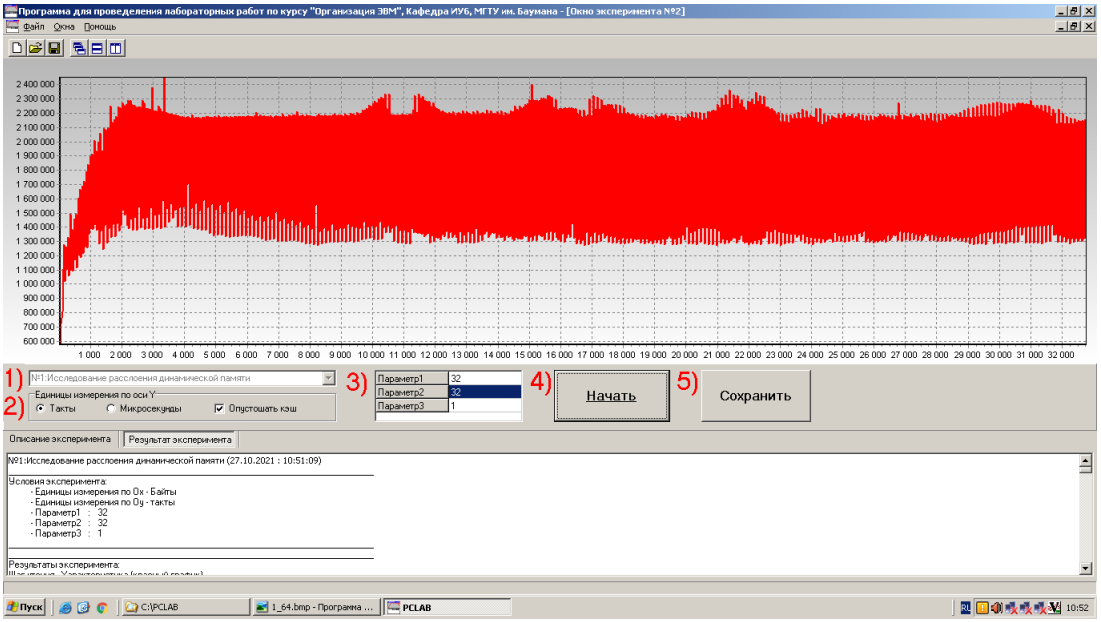

Рисунок 1.2 — Результаты исследования расслоения динамической памяти (часть 2)

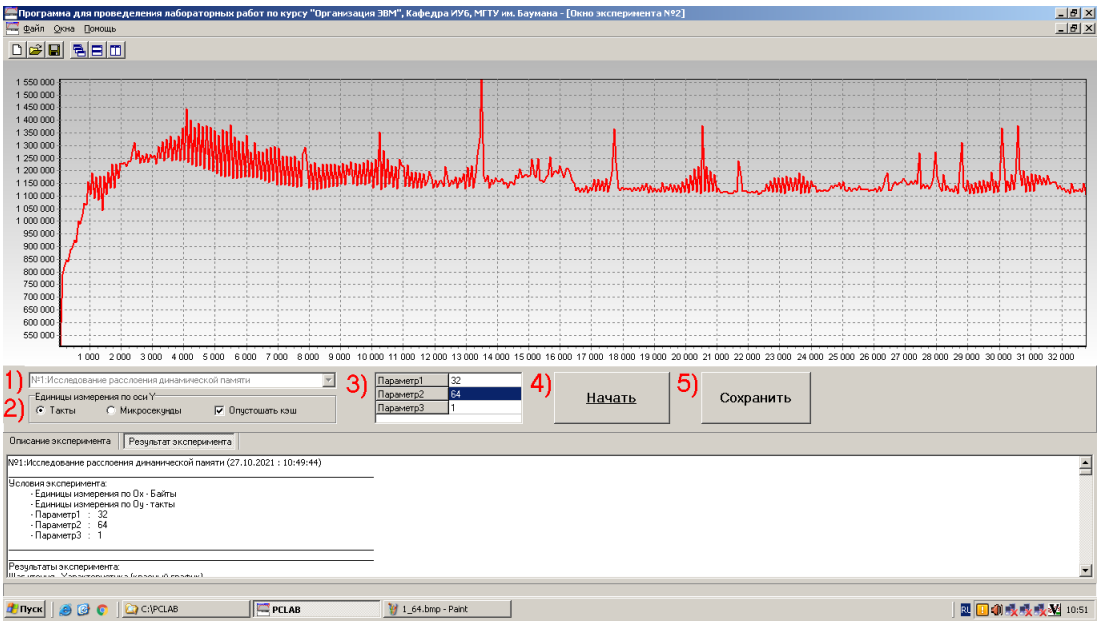

Рисунок 1.3 — Результаты исследования расслоения динамической памяти (часть 3)

Ниже, на рисунке 1.4 приведена блочно-циклическая схема расслоения памяти, помогающая более ясно представить суть эксперимента.

Рисунок 1.4 — Блочно-циклическая схема расслоения памяти

Результаты эксперимента:

- количество банков динамической памяти: $B = T1/P = 1024/128 = 8$;
- размер одной страницы динамической памяти: $PS = T2/B = 4096/8 = 512\ byte$;
- количество страниц в динамической памяти: $C = V/(PS*B*P) = \frac{2^{32}}{2^{19}} = 8192\ byte$.

Примечание:

- B — количество банок памяти;
- P — объем данных, являющийся минимальной порцией обмена кэш-памяти верхнего уровня с оперативной памятью;
- PS — размер страницы DRAM памяти;
- T1 — размер блока в одной банке памяти;
- T2 — размер страницы одной банки памяти;
- V — объем физического пространства ОП;
- C — количество страниц в ОП.

Выводы:

1. Память расслоена, доступ к ней с разным временем доступа, в зависимости от размера запрашиваемого блока.

2. Эксперимательно были получены значения количества банков ОП, размер страница банка, размер страницы DRAM.

3. Данные следует размещать так, чтобы они укладывали в одну страницу.

4. Данные следует выравнивать по размеру линейки кэша.

5. Данные следует обрабатывать так, чтобы минимизировать количество последовательных обращений в одну банку ОП.

Задание 4

Задание: Ознакомиться с описанием эксперимента «Сравнение эффективности ссылочных и векторных структур данных». Провести эксперимент. По результатам эксперимента определить: отношение времени работы алгоритма, использующего зависимые данные, ко времени обработки аналогичного алгоритма обработки независимых данных. Сделать выводы об эффективности ссылочных и векторных структур данных и способах ее повышения. Результаты занести в отчет.

Цель эксперимента: оценить влияние зависимости команд по данным на эффективность вычислений.

В таблице 1.2 приведены настраиваемые параметры.

Таблица 1.2 — Настраиваемые параметры

№	Значение	Описание
1	1 Мбайт	Количество элементов в списке
2	32 Кбайт	Максимальная фрагментации списка
3	1 Кбайт	Шаг увеличения фрагментации

Ниже, на рисунке 1.5 приведена зависимость времени выполнения поиска минимального значения для массива и односвязного списка.

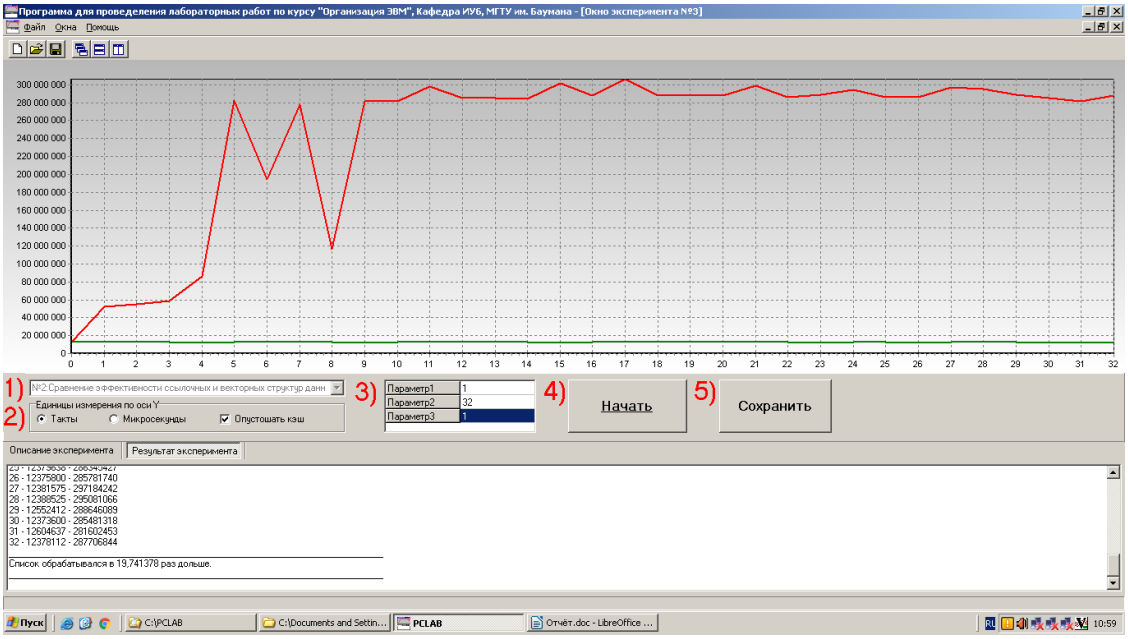

Рисунок 1.5 — Сравнение эффективности ссылочных и векторных структур данных

Из полученного графика видна проблема сематического разрыва. Следует использовать структуры данных с учётом технологического фактора. Для машины «лучше» использовать массив т.к. она плохо работает со списками.

Результаты эксперимента. Односвязный список обрабатывался в **19,741378** раз дольше.

Выводы:

1. Связанные данные следует организовывать так, чтобы при работе программы они были как можно ближе друг к другу расположены в ОП.

2. Использование структур данных, помогающих ясней представить и быстрее решить задачу, может приводить к значительному снижению производительности системы (Таким образом, в эксперименте наблюдали следствие сематического разрыва)

Задание 5

Задание: Для ЭВМ, используемой при проведении лабораторной работы определить следующие параметры: степень ассоциативности и размер TLB данных. Ознакомиться с описанием и провести эксперимент «Исследование эффективности программной предвыборки». По результатам эксперимента определить: отношение времени последовательной обработки блока данных ко времени обработки блока с применением предвыборки; время и количество тактов первого обращения к странице данных. Сделать выводы об эффективности предвыборки и способах ее повышения. Результаты занести в отчет.

Цель эксперимента: выявить способы ускорения вычислений благодаря применению предвыборки данных.

Исходные данные:

- степень ассоциативности TLB данных: 4 ячейки;
- размер TLB данных: 128 групп.

В таблице 1.3 приведены настраиваемые параметры.

Таблица 1.3 — Настраиваемые параметры

№	Значение	Описание
1	512 байт	Шаг увеличения расстояния между читаемыми данными
2	128 Кбайт	Размер массива

Ниже, на рисунке 1.6 приведены зависимости времени обращения к памяти от расстояния между читаемыми блоками данных. Красный график показывает время или количество тактов работы алгоритма без предвыборки. Зеленый график показывает время или количество тактов работы алгоритма с использованием предвыборки.

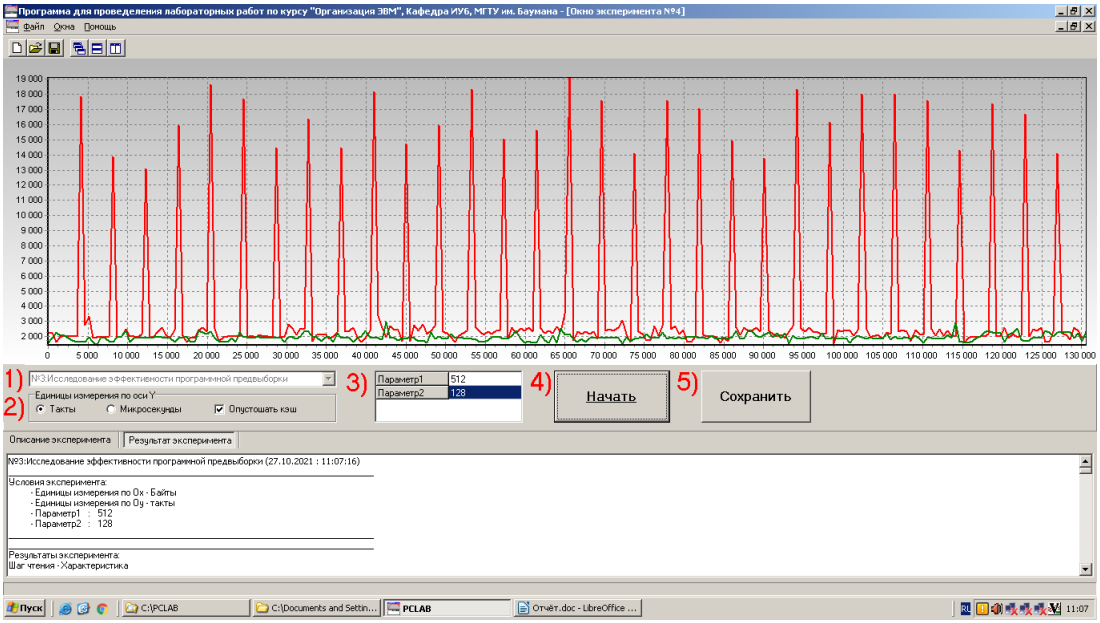

Рисунок 1.6 — Исследование эффективности программной предвыборки

Время обращения к первому элементу в таблицы в 20 раз больше т.к. мы не знаем где это страница находится в памяти, мы имеем только логический адрес, а нужен физический. Поэтому логично использовать предвыборку. В данном примере ускорение почти в 2 раза.

Результаты эксперимента:

- обработка без загрузки таблицы страниц в TLB производилась в **1,8456621** раз дольше;

- время первого обращения к странице данных: **15‘000** тактов.

Вывод: Для исключения задержек, связанных с получением физического адреса начала страницы, имеет смысл предварительно загрузить страницы в TLB перед работой с большими массивами данных.

Задание 6

Задание: Ознакомиться с описанием и провести эксперимент «Исследование способов эффективного чтения оперативной памяти». По результатам эксперимента определить: отношение времени обработки блока памяти неоптимизированной структуры ко времени обработки блока структуры, обеспечивающей эффективную загрузку и параллельную обработку данных. Сделать выводы о способах повышения эффективности чтения оперативной памяти.

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

Исходные данные:

- адресное расстояние между банками памяти: 128 байт;
- размер буфера чтения: 4 КБайт.

В таблице 1.4 приведены настраиваемые параметры.

Таблица 1.4 — Настраиваемые параметры

№	Значение	Описание
1	1 Мбайт	Размер массива
2	128 ед.	Количество потоков данных

Ниже, на рисунке 1.7 приведены зависимости времени чтения данных от количества одновременно обрабатываемых массивов для неоптимизированной структуры (красный график) и структуры, обеспечивающей эффективную загрузку и параллельную обработку данных (зеленый график).

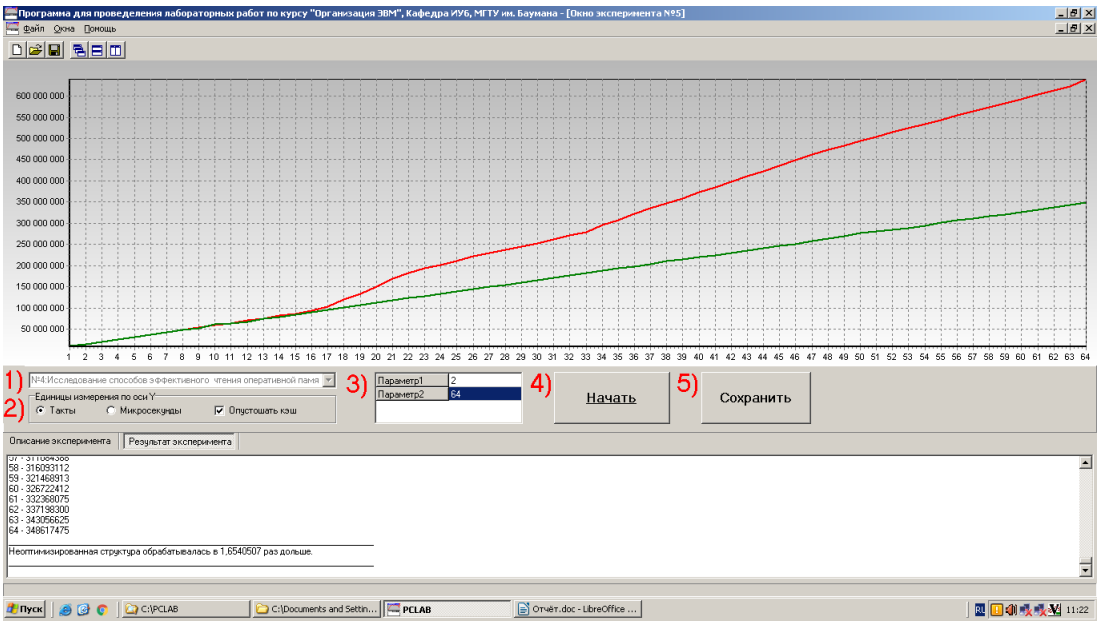

Рисунок 1.7 — Исследование способов эффективного чтения оперативной памяти

Результаты эксперимента:

• отношение времени обработки блока памяти неоптимизированной структуры ко времени обработки блока структуры, обеспечивающей эффективную загрузку и параллельную обработку данных: **1.65**.

Вывод:

• упорядочив данные определённым образом, можно ускорить приложение (мы группируем данные, которые используем вместе.);

• следует переупорядочивать данные, выравнивая их по размеру кэш-линии, тем самым исключая несвоевременную передачу данных;

• следует размещать данные как можно ближе друг к другу: стараться, по возможности, не обращаться к диспетчеру кучи за памятью, помнить и использовать на приктике особенности выравнивания данных (например в C-структурах);

Задание 7

Задание: Для ЭВМ, используемой при проведении лабораторной работы определить следующие параметры: размер банка кэш-памяти

данных первого и второго уровня, степень ассоциативности кэш-памяти первого и второго уровня, размер линейки кэш-памяти первого и второго уровня. Ознакомиться с описанием и провести эксперимент «Исследование конфликтов в кэш-памяти». По результатам эксперимента определить: отношение времени обработки массива с конфликтами в кэш-памяти ко времени обработки массива без конфликтов. Сделать выводы о способах устранения конфликтов в кэш-памяти.

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

Исходные данные:

- размер банка кэш-памяти данных первого и второго уровня: 32 КБайт;
- степень ассоциативности кэш-памяти первого и второго уровня: 8 ячеек;
- размер линейки кэш памяти первого и второго уровня: 64 байт.

В таблице 1.5 приведены настраиваемые параметры.

Таблица 1.5 — Настраиваемые параметры

№	Значение	Описание
1	128 Кбайт	Размер банка кэш-памяти
2	128 байт	Размер линейки кэш-памяти
3	32 ед.	Количество читаемых линеек

Ниже, на рисунке 1.8 приведены зависимости времени обращения к памяти от расстояния между читаемыми блоками данных.

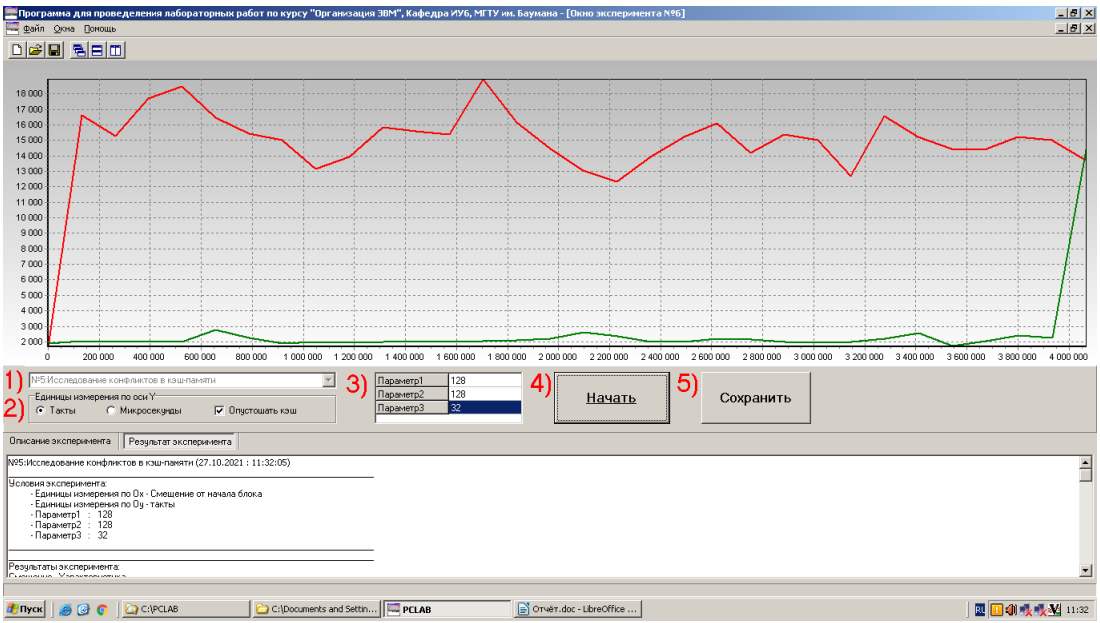

Рисунок 1.8 — Исследование конфликтов в кэш-памяти

Результаты эксперимента: отношение времени обработки массива с конфликтами в кэш-памяти ко времени обработки массива без конфликтов = **5,9547263**.

Ниже, на рисунке 1.9 приведена кэш-память с четырехканальным частично-ассоциативным отображением, помогающая понять суть эксперимента.

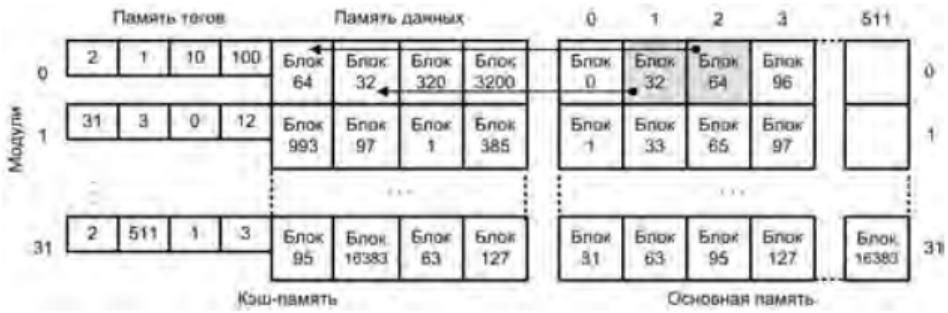

Рисунок 1.9 — Кэш-память с четырехканальным частично-ассоциативным отображением

Вывод:

- кэш память ускоряет процессор примерно в 6 раз;
- данные следует выравнивать по адресам, кратным размеру линейки кэша;

• данные следует обрабатывать так, чтобы уменьшить количество последовательных обращений к блокам памяти, соответствующих одному набору (модулю) (рисунок 1.9).

Задание 8

Задание: Ознакомиться с описанием и провести эксперимент «Исследование алгоритмов сортировки». По результатам эксперимента определить: отношение времени сортировки массивов алгоритмом QuickSort ко времени сортировки алгоритмом Counting Radix, а также ко времени сортировки Counting-Radix алгоритмом, оптимизированным под 8-процессорную вычислительную систему. Сделать выводы о наиболее эффективном алгоритме сортировки.

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

В таблице 1.6 приведены настраиваемые параметры.

Таблица 1.6 — Настраиваемые параметры

№	Значение	Описание
1	1 Мбайт	Количество 64-х разрядных элементов массивов
2	8 Кбайт	Шаг увеличения размера массива

Ниже, на рисунке 1.10 приведены зависимости времени сортировки (алгоритмы Quick Sort, Radix-Counting Sort, Оптимизированный Radix-Counting Sort) от размера исходного массива.

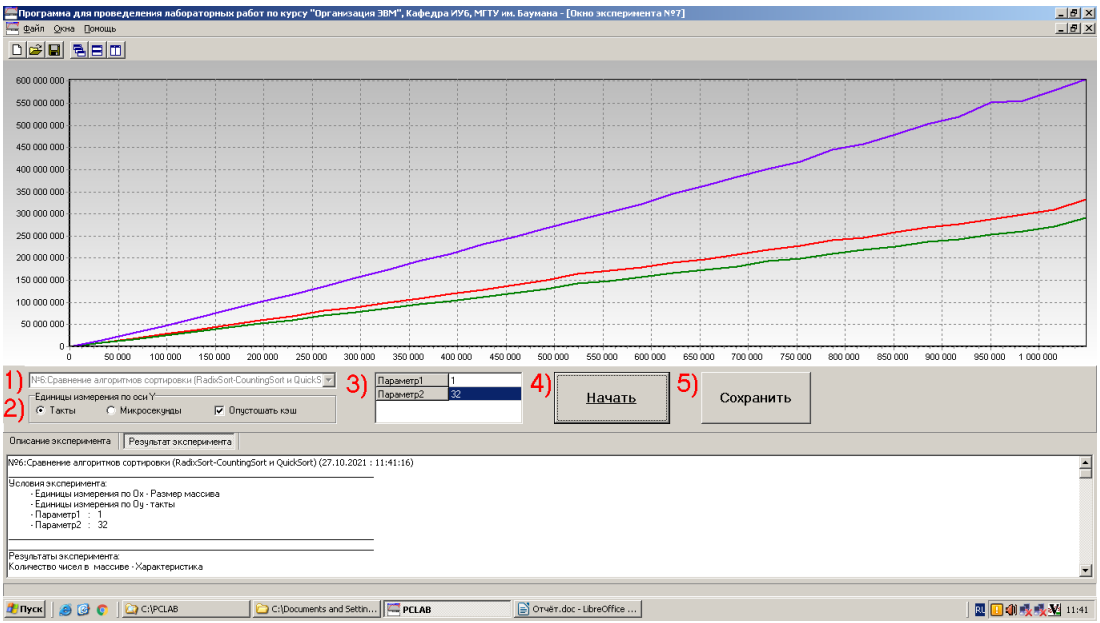

Рисунок 1.10 — Исследование алгоритмов сортировки

Примечание: Фиолетовый график показывает время или количество тактов работы алгоритма QuickSort. Красный график показывает время или количество тактовработы неоптимизированного алгоритма Radix-Counting. Зеленый график показывает время или количество тактов работы оптимизированного под 8-процессорную вычислительную систему алгоритма Radix-Counting.

Результаты эксперимента:

- отношение времени сортировки массива алгоритмом QuickSort ко времени сортировки алгоритмом Radix-Counting Sort: **1.8379134**;

- отношение времени сортировки массива алгоритмом QuickSort ко времени сортировки Radix-Counting Sort, оптимизированной под 8-процессорную вычислительную систему: **2.0982673**.

Выводы:

- существует алгоритм поразрядной сортировки с сложности меньше чем линейной вычислительной сложности O(n/log(n));

- следует выбирать алгоритмы на основе типа входных данных, которые позволяют решить задачу наиболее эффективно.

Ответы на контрольные вопросы

1. Назовите причины расслоения оперативной памяти

Расслоение памяти позволяет повысить пропускную способность оперативной памяти (ОП) за счет компоновки ОП из нескольких банков. При таком построении, процедуры обращений к нескольким банкам памяти можно совместить.

2. Как в современных процессорах реализована аппаратная предвыборка?

Если говорить о суперскалярных процессорах, то суть предвыборки заключается в том, чтобы как можно лучше загрузить вычислительный конвейер: при каждой возможности производится считывание команд из памяти, опережающее ход вычислений. Команды после выборки помещаются в быстродействующем буфере предвыборки. Буфер, например, может быть организован по принципу очереди, тогда команды в него поступают в порядке их выполнения в программе. Высокое быстродействие буфера и наличие в нем значительного количества команд позволяет одновременно загружать все конвейеры процессора.

Также можно привести другой пример аппаратной предвыборки: уменьшение времени получения физического адреса страницы ОП при загрузке его в TLB. Суть предвыборки заключается в загрузке физического адреса не одной страницы, а сразу нескольких (8 или 16) за одно обращение к памяти [это возможно за счет расслоения памяти].

3. Какая информация храниться в TLB?

Если рассматривать машину со страничной организацией виртуальной памяти, то физические адреса начала страниц в физической памяти (фреймов).

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

Частично-ассоциативный, так как сочетает достоинства прямого и полностью ассоциативного способов отображения: экономия памяти

тэгов (по сравнению с полностью ассоциативным кэшем) и уменьшение количества попаданий в один и тот же набор линеек кэша (по сравнению с прямым отображением).

5. Приведите пример программной предвыборки

Ниже в листинге 1.1 приведена часть программы из 3 эксперимента.

Листинг 1.1 — Пример программной предвыборки

```
1  int *p = get_array(arr_size);    // Get huge array from enternal world
2  int tmp;
3  for (int a = 0; a < arr_size; a += SIZEOF_PAGE)
4      tmp = *(int *) (int(p) + a);
5
6  for (int a = 0; a < arr_size; a += 1)
7  {
8      int elem = *(int *) (int(p) + a);
9      ...                          // Array processing
10 }
```