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

Студент: _____ Ву Х. Д.
подпись, дата Фамилия, И.О.

Преподаватель: _____ Ибрагимов С. В.
подпись, дата Фамилия, И. О.

Москва — 2023 г.

Оглавление

Цель работы

Основной целью 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.

Основные теоретические сведения

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. 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 и запрещения переупорядочивания микрокоманд.

Эксперимент №1: Исследования расслоения динамической памяти

Цель эксперимента

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

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.

Исходные данные

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

Результаты эксперимента

На рисунке 1 предаставлен график, полученный в результате эксперимента с исходными параметрами:

- Максимальное расстояния между читаемыми блоками (K) = 32;
- Шаг увеличения расстояния между читаемыми 4-х байтовыми ячейками (Б) = 64;
- Размер массива (M) = 2.

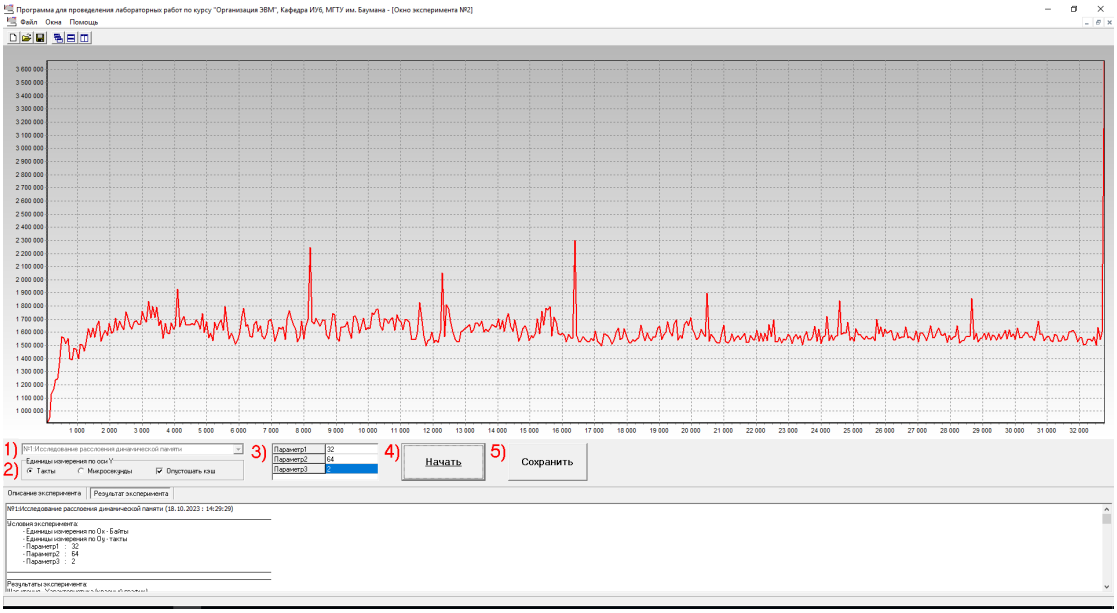

Рисунок 1 – Эксперимет №1, результат

Вывод

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

Эксперимент №2: Сравнение эффективности ссылочных и векторных структур

Цель эксперимента

Оценка влияния зависимости команд по данным на эффективность вычислений.

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 операндами 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.

Результаты эксперимента

На рисунке 2 предаставлен график, полученный в результате эксперимента с исходными параметрами:

- Количество элементов в списке (M) = 4;
- Максимальная фрагментации списка (K) = 128;
- Шаг увеличения фрагментации (K) = 4.

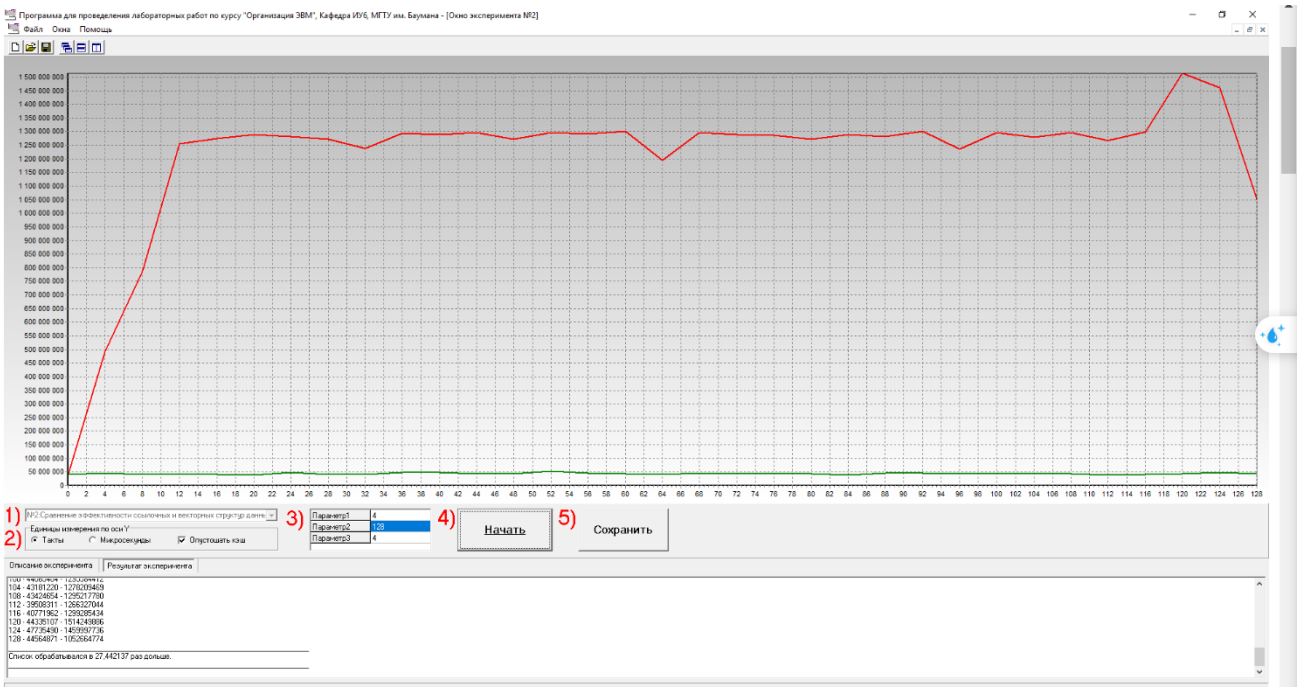

Рисунок 2 – Эксперимет №2

Вывод

Вывод из полученных результатов можно сделать следующий: использовать структуры данных надо с учетом технологического фактора определенной задачи. Если решаемая задача продполагает возможность использования массива, то надо использовать его, особенно если использование списка не дает существенной разницы (особенно выиграша во времени).

Эксперимент №3: Исследование эффективности программной предвыборки

Цель эксперимента

Выявление способов ускорения вычислений благодаря применению предвыборки данных.

Исходные данные

Степень ассоциативности и размер TLB данных.

Описание проблемы

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

Результаты эксперимента

На рисунке 3 предаставлен график, полученный в результате эксперимента с исходными параметрами:

- Шаг увеличения расстояния между читаемыми данными (Б) = 512;
- Размер массива (K) = 64.

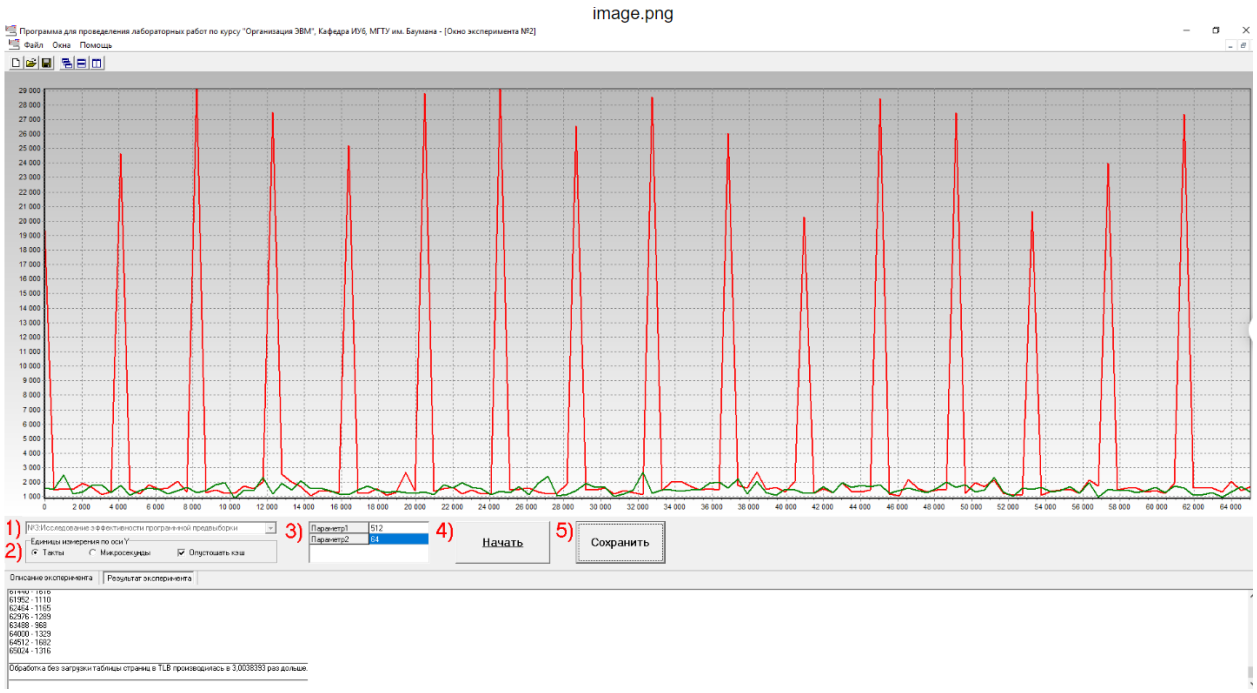

Рисунок 3 – Эксперимет №3

Результат сравнения времени (как вывод программы) представлен на рисунке 4. Как видно на рисунке, обработка без загрузки таблицы страниц в TLB производилась в 3,0038393 раз дольше.

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

Сокращение времени работы алгоритма, который использует предвыборку, происходит в том случае, когда информация о востребованных страницах помещается в TLB.

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

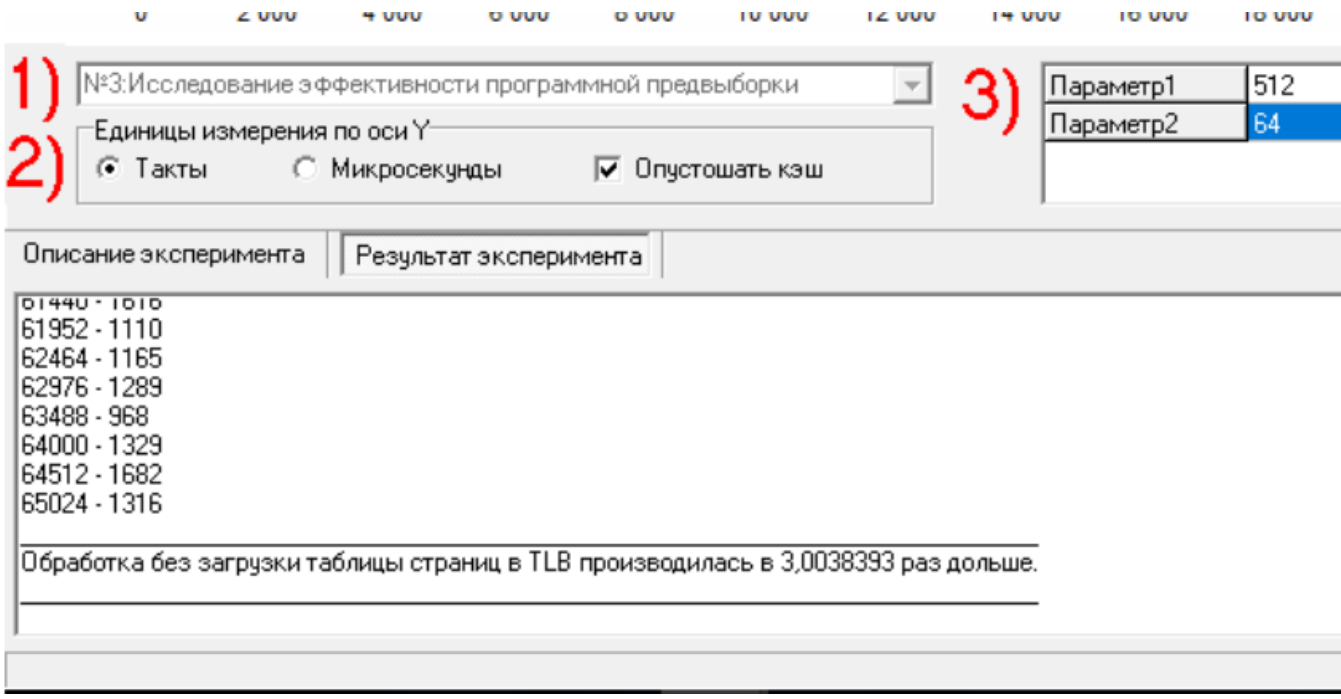

Рисунок 4 – Эксперимет №3, результат

Вывод

Используя предвыборку можно ускорить время работы программы почти в 3 раза за счет заблаговременной загрузки страниц в память.

Эксперимент №4: Исследование способов эффективного 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, размер буфера чтения.

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.

Результаты эксперимента

На рисунке 5 предаставлен график, полученный в результате эксперимента с исходными параметрами:

- Размер массива (M) = 4;
- Количество потоков данных = 64.

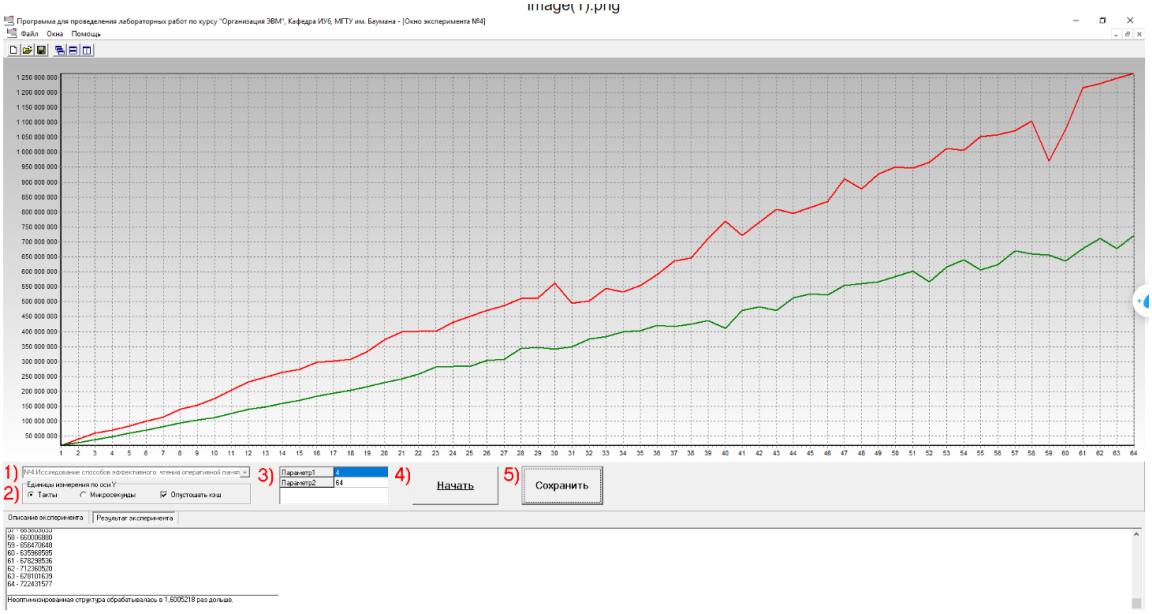

Рисунок 5 – Эксперимет №4

Результат сравнения времени (как вывод программы) представлен на рисунке 6. Как видно на рисунке, неоптимизированная структура обрабатывалась в 1,6 раз дольше.

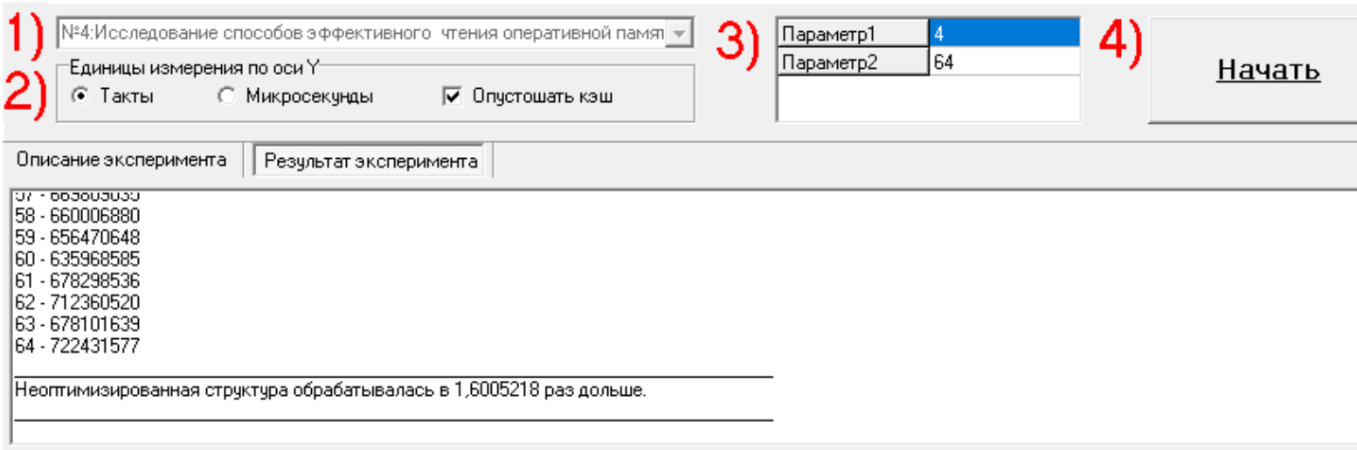

Рисунок 6 – Эксперимет №4, результат

Красный график показывает время или количество тактов работы алгоритма, использующего неоптимизированную структуру.

Зеленый график показывает время (или количество тактов) работы алгоритма 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 востребованную для вычислений инфорацию. Поэтому снижается количество открытий и закрытий страниц DRAM-памяти и обеспечивается параллельная обработка данных, а также выполнение операций загрузки и выгрузки.

Вывод

Можно сделать вывод, что для ускорения работы алгоритмов, необходимо правильно упорядочить данные.

Эксперимент №5: Исследование 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, 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, 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.

Результаты эксперимента

На рисунке 7 предаставлен график, полученный в результате эксперимента с исходными параметрами:

- Размер банка кэш-памяти (K) = 4;
- Размер линейки кэш-памяти (б) = 64;
- Количество читаемых линеек = 256.

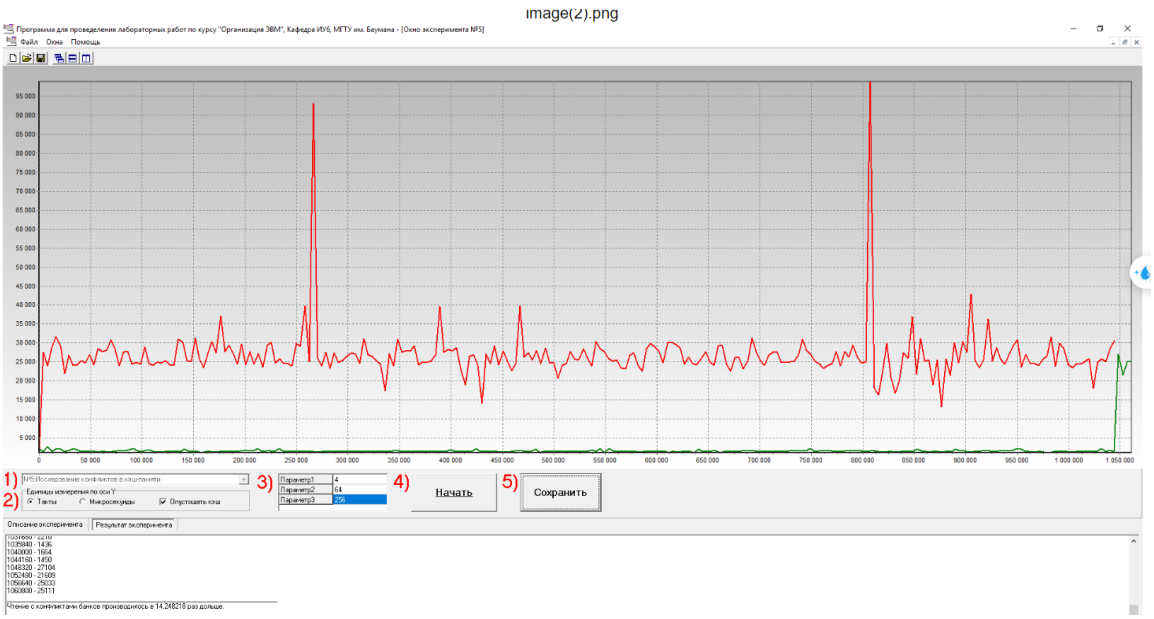

Рисунок 7 – Эксперимет №5

Результат сравнения времени (как вывод программы) представлен на рисунке 8. Как видно на рисунке, чтение с конфликтами банков производилось в 14,24 раз дольше.

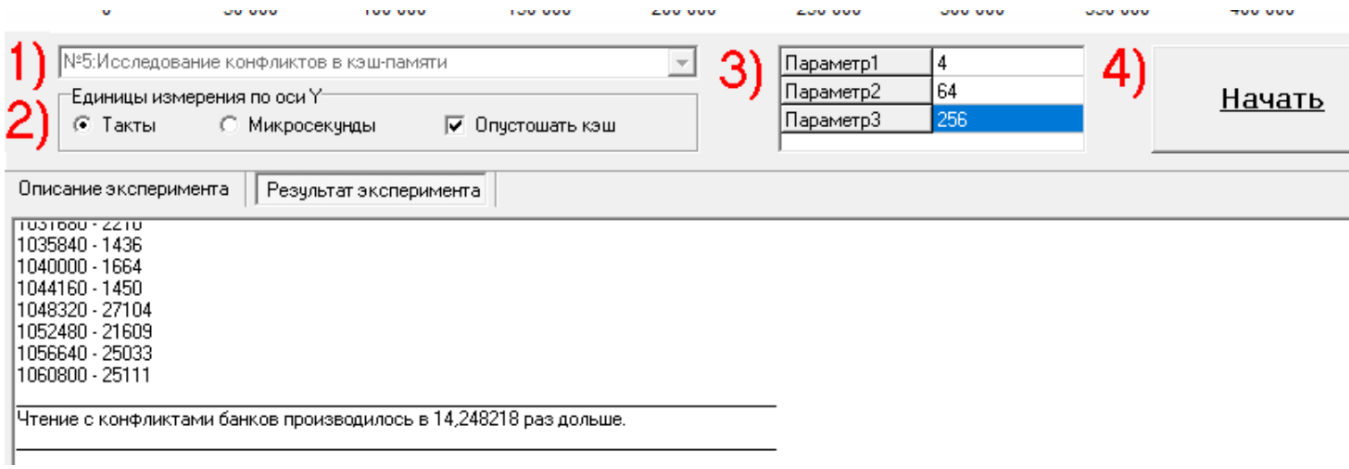

Рисунок 8 – Эксперимет №5, результат

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

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

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

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

Вывод

Можно сделать вывод, что использование кэш-памяти работа процессора ускоряется почти в 14 раз.

Эксперимент №6: Сравнение алгоритмов сортировки

Цель эксперимента

Исследование способов эффективного использования памяти и выявление наиболее эффективных алгоритмов сортировки, применимых в вычислительных системах.

Исходные данные

Количество процессоров вычислительной системы, размер пакета, количество элементов в массиве, разрядность элементов массива.

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 не требующие дополнительного массива), возможность распараллеливания (не распараллеливаемые, ограниченно распараллеливаемые, полностью распараллеливаемые), принцип определения порядка (алгоритмы, использующие парные сравнения и не использующие парные сравнения).

Radix Sort

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

Сначала идет сортировка их по первому (старшему) разряду. Сортировка в таком случае выполняется с помощью сортировки подсчетом (count sort). Сложность — $O(n)$.

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 рассматриваем уже не старший разряд, а следующий за ним, и т.д.

Такие действия выполняется до последнего разряда.

Результаты эксперимента

На рисунке 9 предаставлен график, полученный в результате эксперимента с исходными параметрами:

- Количество 64-х разрядных элементов массивов (M) = 1;
- Шаг увеличения размера массива (K) = 32.

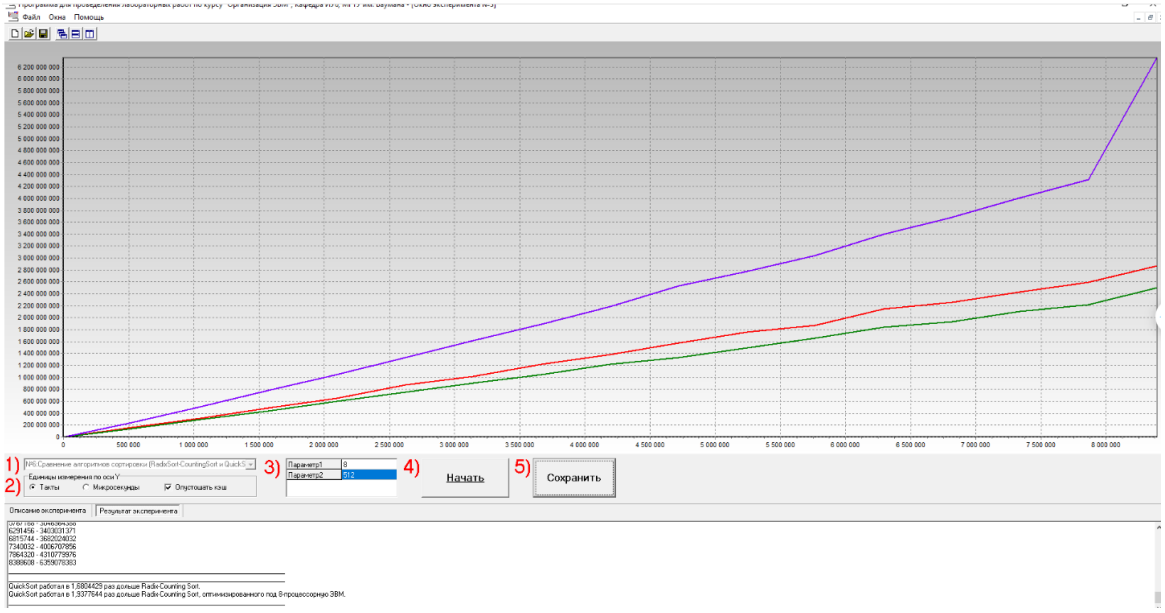

Рисунок 9 – Эксперимет №6

Результат сравнения времени (как вывод программы) представлен на рисунке 10. Как видно на рисунке, QuickSort работал в 1.68 раз дольше Radix-Counting Sort, и QuickSort работал в 1.94 раз дольше Radix-Counting Sort, оптимизированного под 8-процессорную ЭВМ.

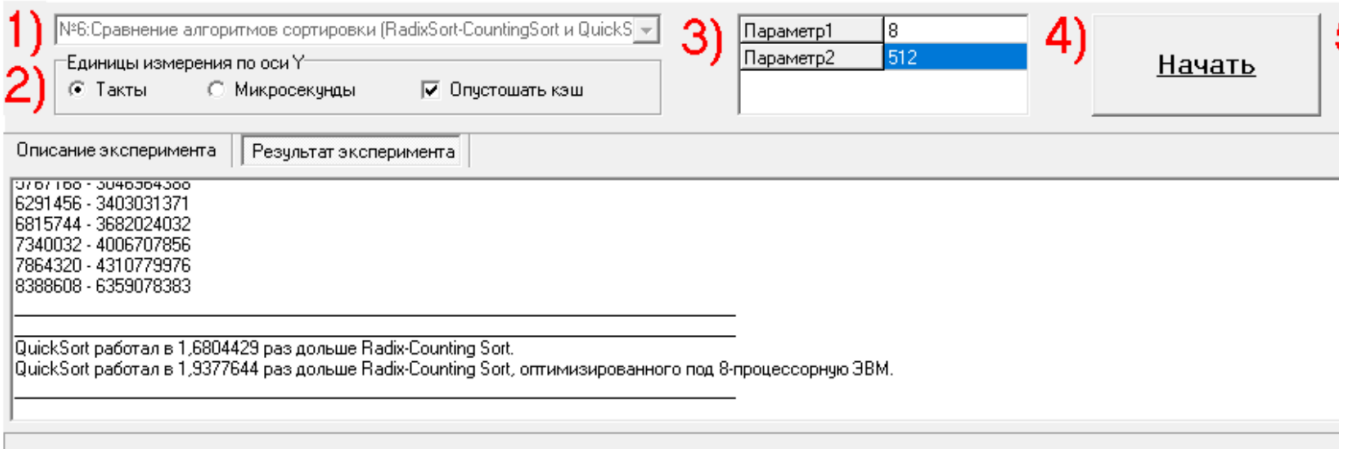

Рисунок 10 – Эксперимет №6, результат

Фиолетовый график показывает время или количество тактов работы алгоритма QuickSort.

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

Зеленый график показывает время или количество тактов работы оптимизированного 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.

Вывод

Основаны основные принципы эффективного использования подсистемы 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 анализа производительности PCLAB.

Поставленная цель достигнута.