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»

НАПРАВЛЕНИЕ ПОДГОТОВКИ «09.03.04 Программная инженерия»

ОТЧЕТ
по лабораторной работе №2

Название: Организация памяти конвейерных суперскалярных ЭВМ

Дисциплина: Архитектура ЭВМ

Студент ИУ7-53Б _____ Н. Д. Лысцев
Группа | Подпись, дата | И. О. Фамилия

Преподаватель _____ А. Ю. Попов
Подпись, дата | И. О. Фамилия

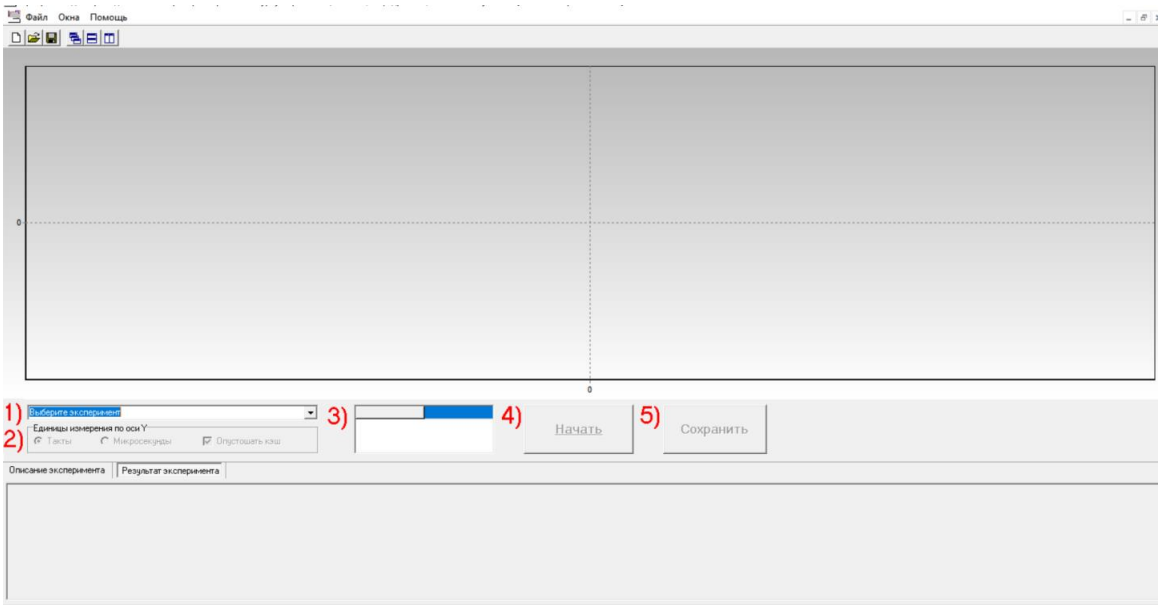
Москва, 2023 г.

Цель работы

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

Задание 1

В этом задании прошло ознакомление с возможностями программы PCLAB. Изучена информация вкладки "Идентификация процесса". Процесс сбора и анализа экспериментальных данных в PCLAB основан на процедуре профилировки критического кода (в измерении времени его обработки центральным процессорным устройством). При запуске программы доступен



следующий интерфейс (рис. 1):

Рисунок 1 – Окно эксперимента

В верхней части экрана можно будет увидеть зависимость, полученную путем проведения эксперимента. Составляющие интерфейса:

1. название эксперимента;
2. выбор единиц измерения эксперимента;
3. параметры эксперимента;
4. старт проведения эксперимента;
5. сохранение результатов эксперимента.

Задание 2

Благодаря информации с вкладки «Идентификация процесса» были установлены параметры система:

- размер линейки кэш-памяти верхнего уровня - 128 б;
- объем физической памяти - 2 Гб.

Задание 3

Результаты эксперимента на рис. 2.

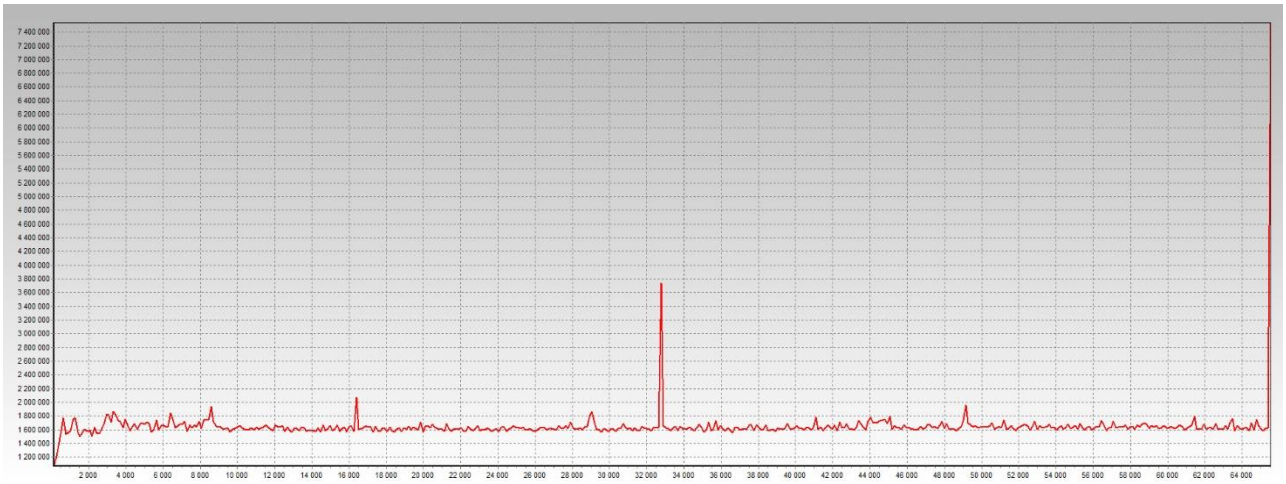


Рисунок 2 – Исследование расслоения динамической памяти

До точки Т2 = 2048 б наблюдается рост графика, связанный с тем, что при росте шага чтения требуется открытие большего числа страниц. При этом в одном банке памяти может находиться одна и та же открытая страница. Точка Т2 - расстояние между началами дух последовательных страница памяти одного банка. Тогда размер страницы памяти можно найти, разделив шаг Т2 на число банков.

Точка Т1 = 128 б представляет собой точку локального максимума. Эта точка соответствует размеру одного набора. При этом постоянно происходит обращение к одному и тому же банку, то есть не происходит чередование банков памяти. По ее значению этого шага легко можно определить число банков памяти. Для этого размер набора надо разделить на размер линейки Б = Т1 / П = 128 б / 128 б = 1.

Размер одной страницы памяти РС = Т2 / Б = 2048 б / 1 = 2048 б. Число страниц физической оперативной памяти С = О / (РС * Б * П) = 2147482648 / (2048 * 1 * 128) = 8192 шт.

Поскольку память состоит из определенного количества банков, наилучшим способом обращения к ней будет обращение к последовательным элементам соседних банков, худшим - обращение к следующей строке одного и того же банка.

Задание 4

Результаты эксперимента на рис. 3.

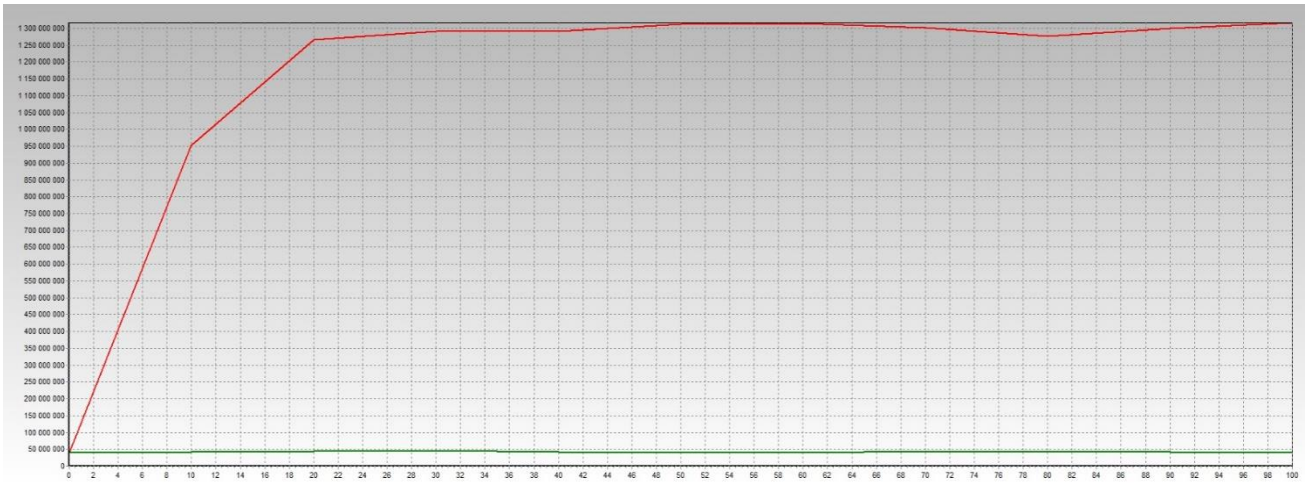


Рисунок 3 – Сравнение эффективности ссылочных и векторных струк-тур данных

По результатам эксперимента можно сказать, что список (зависимые данные) обрабатываются в среднем в 19.3 раз дольше, чем массив (независимые данные). Можно сделать вывод, что по возможности лучше использовать массив, а не список. При обработке списков адрес следующегоэлемента станет известен лишь после обращения к текущему, таким образом предвыборка в кэш не производится. При работе с массивами же предвыборка производится, и обработка предзагруженных в кэш и последовательно расположенных в памяти происходит гораздо быстрее.

Задание 5

Результаты эксперимента представлены на рис. 4.

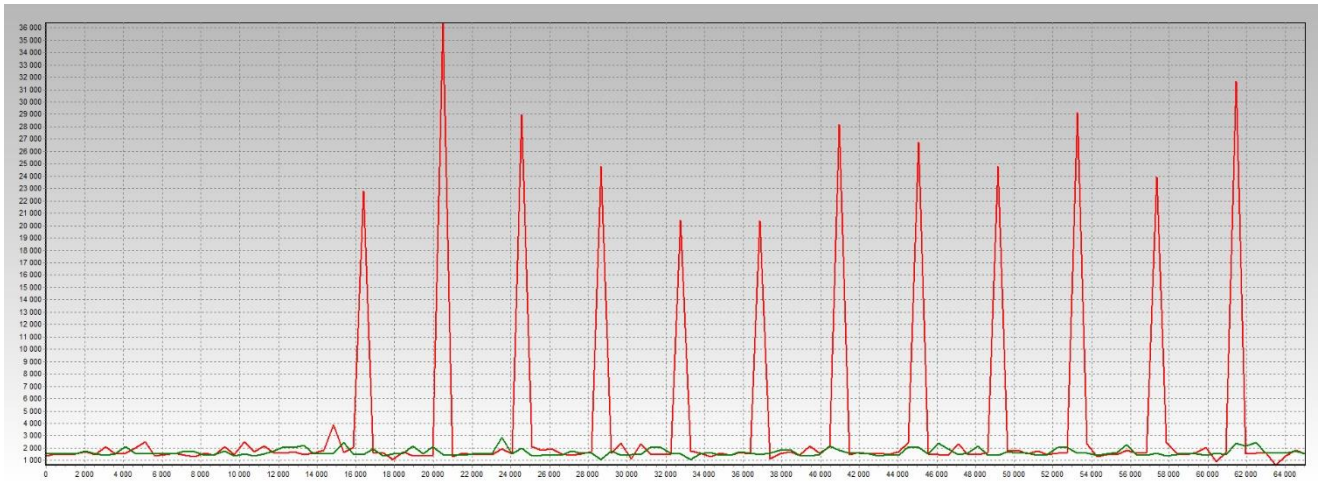


Рисунок 4 – Исследование эффективности программной выборки

Обработка без загрузки таблицы страниц в TLB производилась в 1.4842 раз дольше. При обработке больших объемов данных происходит открытие большого числа страниц физической памяти. Возникает необходимость преобразования логических адресов в физические на основании хранимых в TLB процессора данных. Если в TLB процессора отсутствует необходимая информация, то приходится обращаться к оперативной памяти дважды, что приводит к потере производительности. Пики на красном графике соответствуют ситуации, когда информация о искомой странице в TLB отсутствовала. Использование предвыборки позволяет предварительно загрузить данные в TLB, что в ряде случаев повышает производительность.

Задание 6

Результаты эксперимента представлены на рис. 5.

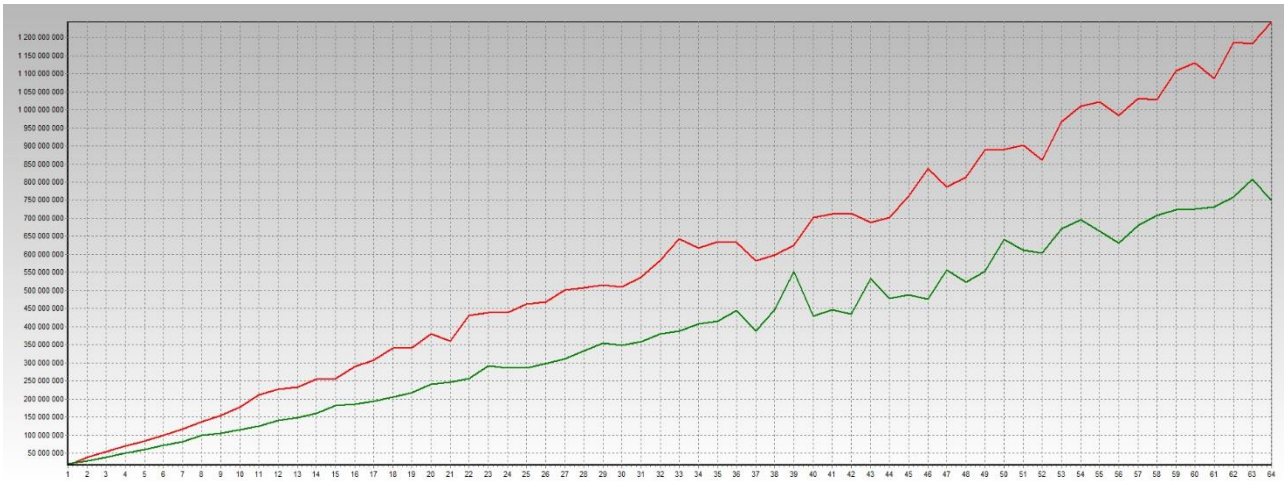


Рисунок 5 – Исследование способов эффективного чтения оперативной памяти

Неоптимизированная структура обрабатывалась в 1.1438 раз дольше. При выборке из оперативной памяти происходит получение одной линейки. При этом объеме востребованных данных при программировании на языках высокого уровня зачастую меньше размера линейки. Оптимизация структур данных позволяет передавать в одном пакете только востребованные данные. Это позволяет снизить количество кэш-промахов и повышает производительность системы до 2-х раз.

Задание 7

Результаты эксперимента представлены на рис. 6.

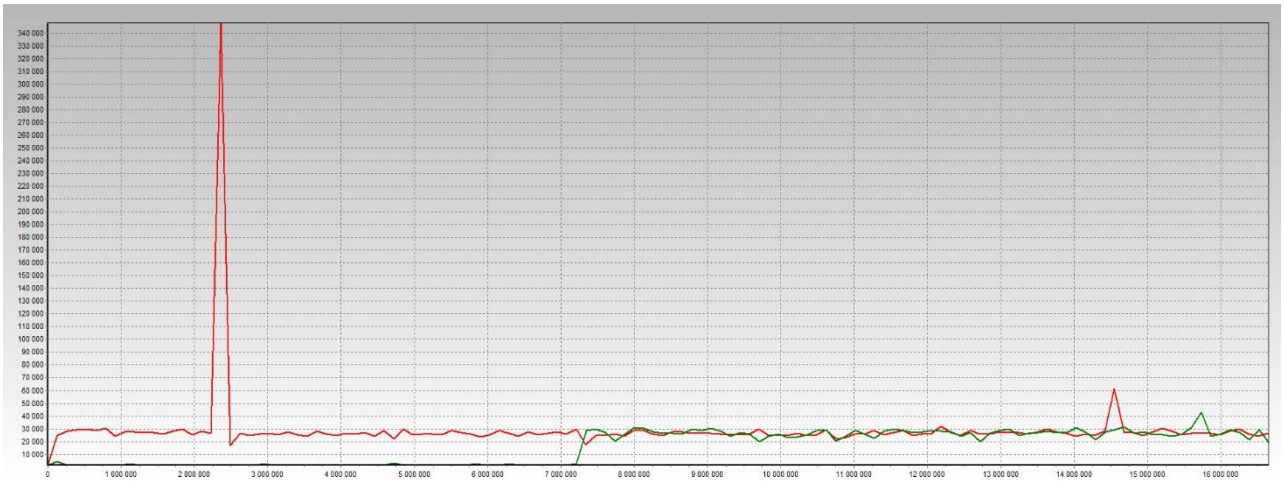


Рисунок 6 – Исследование конфликтов в кэш-памяти

Чтение с конфликтами банков производилось в 9.7563 раз дольше. Конфликты в кэш-памяти возникают в случае чтения данных с шагом, кратным размеру банка памяти. В этом случае задействованным будет только один банк памяти и степень ассоциативности памяти "снижается" до 1. При этом наблюдается постоянное вытеснение данных из кэш-памяти и больший её объём остается незадействованным.

Задание 8

Результаты эксперимента представлены на рис. 7.

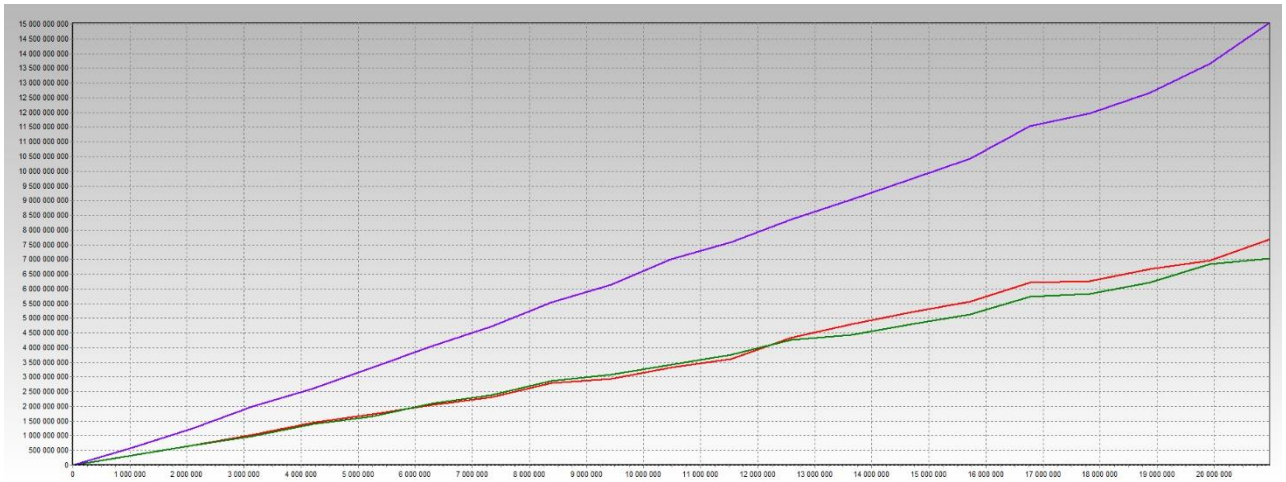


Рисунок 7 – Исследование алгоритмов сортировки

QuickSort работал в 2,9042 раз дольше Radix-Counting Sort. QuickSort работал в 3,5779 раз дольше Radix-Counting Sort, оптимизированного под 8-процессорную ЭВМ. За счет разной логики работы различные алгоритмы сортировки по-разному используют физическую память, отчего сильно разнится время доступа к ней и время работы соответственно. Так, QuickSort использует попарные сравнения, отчего увеличивается число обращений к массиву данных, следовательно, и к памяти, в связи, с чем увеличивается общее время выполнения сортировки. В Radix-Counting Sort сортировка происходит по-другому, отчего этой подобной проблемы не возникает, что снижает вычислительную сложность. Распараллеливание повышает производительность и без того быстрого алгоритма.

Вывод

В ходе лабораторной работы были освоены принципы эффективного использования подсистемы памяти современных универсальных ЭВМ, обеспечивающей хранение и своевременную выдачу команд и данных в центральное процессорное устройство. С помощью PCLAB был проведен ряд экспериментов на сравнение скорости обращения к блокам данных. Производительность обращения к памяти можно увеличить за счет использования предвыборки, упорядочивания расположения данных, дабы избежать не предугадываемых обращения к памяти, а также использование структур данных, позволяющих осуществлять предзагрузку данных в кэш-память, параллельную обработку данных.