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 №2

по курсу «Архитектура ЭВМ»

на тему: *«Организация памяти конвейерных суперскалярных электронных вычислительных машин»*

Студент группы ИУ7-51Б _____ (Подпись, дата) Савинова М. Г. (Фамилия И.О.)

Преподаватель _____ (Подпись, дата) Ибрагимов С. В. (Фамилия И.О.)

Москва — 2023 г.

Содержание

1 Эксперимент 1: Исследование расслоения динамической памяти

1.1 Цель эксперимента

Определить способ трансляции физического адреса, используемый при обращении к динамической памяти.

1.2 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.

1.3 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.

1.4 Условия эксперимента

1) Единицы измерения по Ox - Байты;

2) Единицы измерения по Oy - Такты;

3) Максимальное расстояние между читаемыми данными: **32**;

4) Шаг увеличения расстояния между читаемыми 4х байтовыми ячейками: **64**;

5) Размер массива: **8**;

1.5 Результаты эксперимента

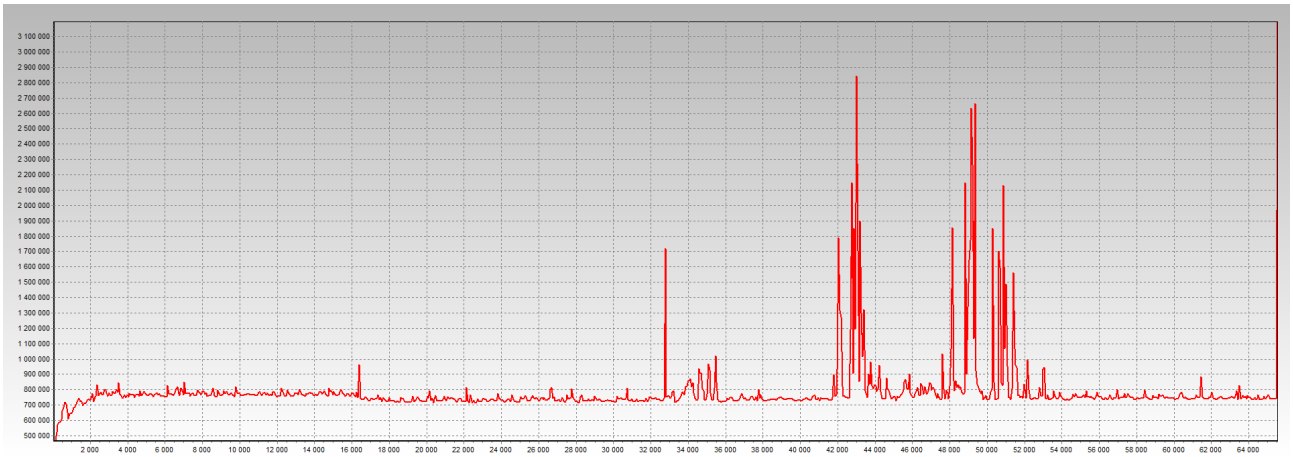

Рисунок 1.1 – Эксперимент 1: Исследование расслоения динамической памяти

1.6 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.

2 Эксперимент 2: Сравнение эффективности ссылочных и векторных структур

2.1 Цель эксперимента

Оценить влияние зависимости команд по данным на эффективность вычислений.

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 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.

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.

2.4 Условия эксперимента

1) Единицы измерения по Ox - Килобайты;

2) Единицы измерения по Oy - Такты;

3) Количество элементов в списке: **1**;

4) Максимальная фрагментация списка: **256**;

5) Шаг изменения фрагментации: **4**;

2.5 Результаты эксперимента

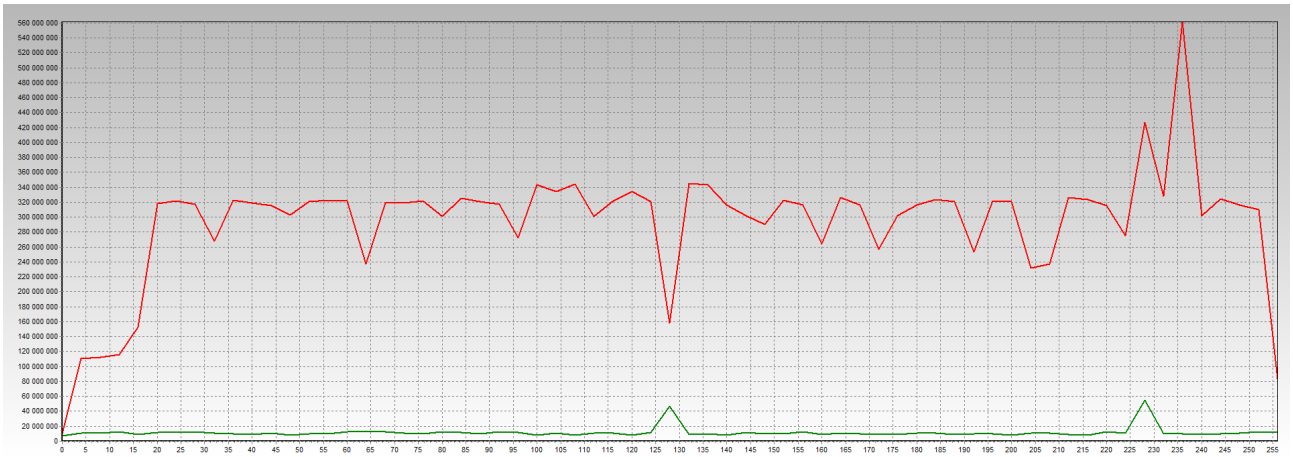

Рисунок 2.1 – Эксперимент 2: Сравнение эффективности ссылочных и векторных структур

```
248 - 10937692 - 316217373
252 - 12233445 - 310516140
256 - 11605975 - 83265571

Список обрабатывался в 25,443032 раз дольше.
```

Рисунок 2.2 – Эксперимент 2: Результаты

Как видно из рисунка 2.2: массив обрабатывается примерно в 20-30 раз быстрее списка.

2.6 Вывод

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

3 Эксперимент 3: Исследование эффективности программной предвыборки

3.1 Цель эксперимента

Выявить способы ускорения вычислений благодаря применению предвыборки данных.

3.2 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 открытия 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.

3.3 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 пред-

выборки, обеспечивающий своевременную загрузку информации в TLB данных.

3.4 Условия эксперимента

1) Единицы измерения по Ox - Байты;
2) Единицы измерения по Oy - Такты;
3) Шаг увеличения расстояния между читаемыми данными: **256**;
4) Размер массива: **512**;

3.5 Результаты эксперимента

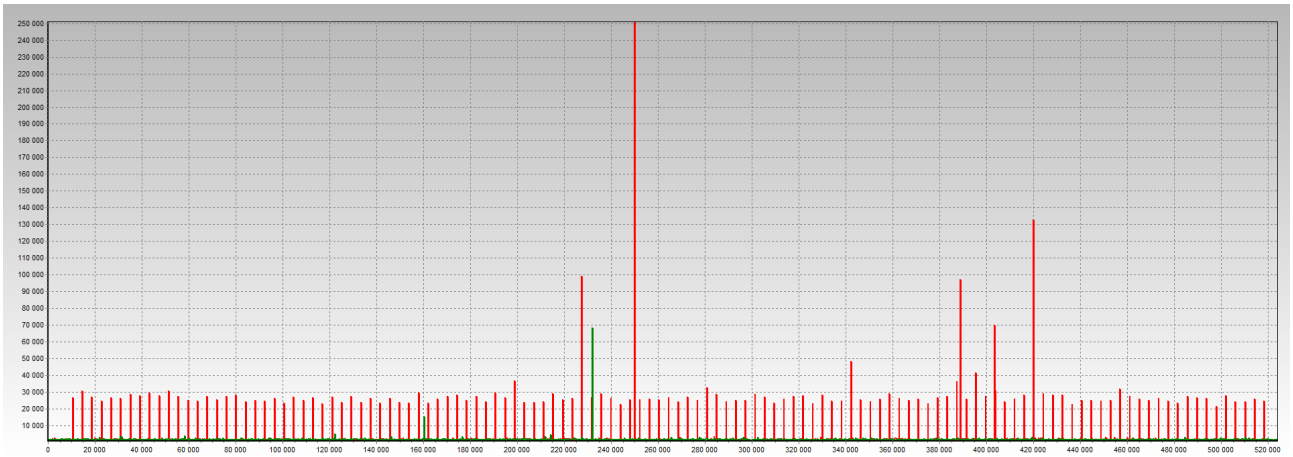

Рисунок 3.1 – Эксперимент 3: Исследование эффективности программной предвыборки

```
523520 - 1570
523776 - 1699
524032 - 1601

Обработка без загрузки таблицы страниц в TLB производилась в 1,9481833 раз дольше.
```

Рисунок 3.2 – Эксперимент 3: Результаты

Как видно на рисунке 3.2, обработка без загрузки таблицы страниц в TLB производилась в 1,9481833 раз дольше.

3.6 Вывод

Используя предвыборку можно ускорить время работы программы почти в 2 раза (как результат представлен 2.4) за счет заблаговременной загрузки страниц в память.

4 Эксперимент 4: Исследование способов эффективного чтения оперативной памяти

4.1 Цель эксперимента

Исследовать возможности ускорения вычислений благодаря использованию структур данных, оптимизирующих механизм чтения оперативной памяти.

4.2 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 обработки 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.

4.3 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.

4.4 Условия эксперимента

1) Единицы измерения по Ox - Колличество параллельных потоков;
2) Единицы измерения по Oy - Такты;
3) Размер массива: **4**;
4) Количество потоков данных: **64**;

4.5 Результаты эксперимента

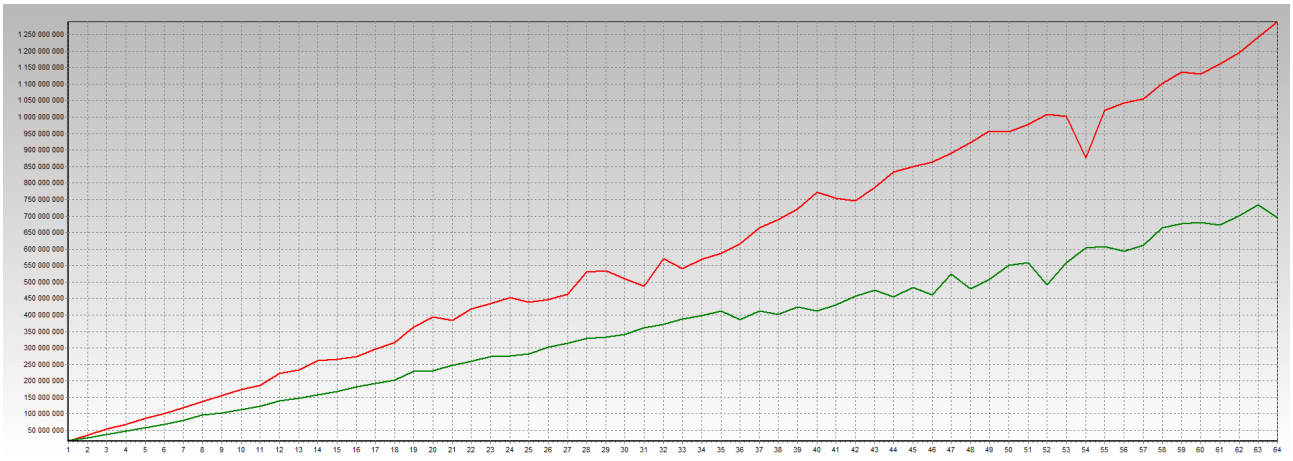

Рисунок 4.1 – Эксперимент 4: Исследование способов эффективного чтения оперативной памяти

63 - 733837574
64 - 694346610

Неоптимизированная структура обрабатывалась в 1,6671335 раз дольше.

Рисунок 4.2 – Эксперимент 4: Результаты

Как видно на рисунке 4.2, неоптимизированная структура обрабатывалась в 1,6671335 раз дольше.

4.6 Вывод

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

5 Эксперимент 5: Исследование конфликтов в кеш-памяти

5.1 Цель эксперимента

Исследовать влияния конфликтов кэш-памяти на эффективность вычислений.

5.2 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.

5.3 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 востребован-

ных данных на некоторый шаг, достаточный для выбора другого набора. Этот шаг соответствует размеру линейки.

5.4 Условия эксперимента

1) Единицы измерения по Ox - Смещение от начала блока;
2) Единицы измерения по Oy - Такты;
3) Размер банка кэш-памяти: **256**;
4) Размер линейки кэш-памяти: **128**;
5) Количество читаемых линеек: **512**;

5.5 Результаты эксперимента

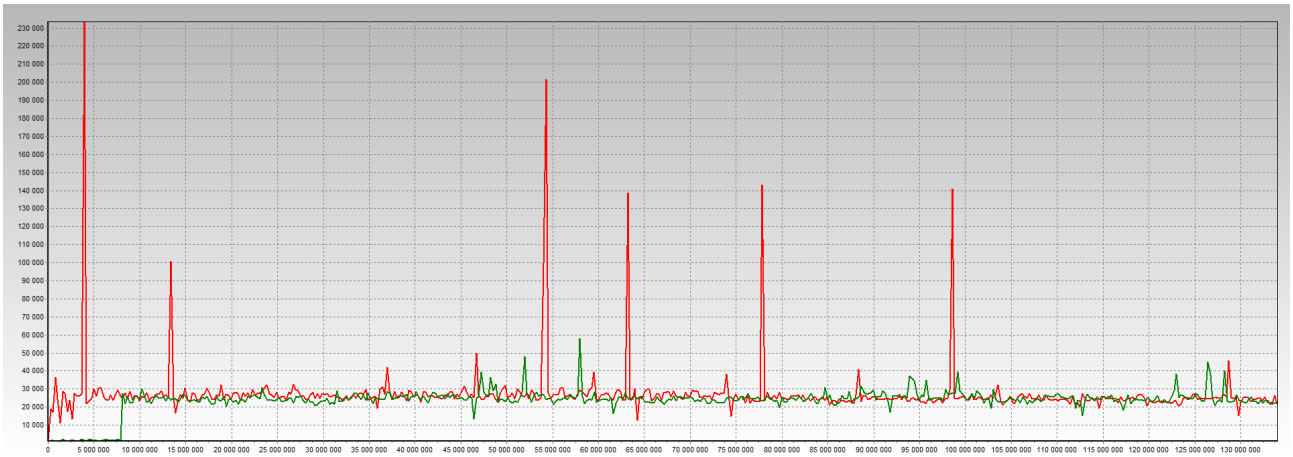

Рисунок 5.1 – Эксперимент 5: Исследование конфликтов в кеш-памяти

133758720 - 22275
134020992 - 22419

Чтение с конфликтами банков производилось в 1,157578 раз дольше.

Рисунок 5.2 – Эксперимент 5: Результаты

Как видно на рисунке 5.2, чтение с конфликтами банков производилось в 1,157578 раз дольше.

5.6 Вывод

Можно сделать вывод, что при использовании кэш-памяти работа процессора ускоряется.

6 Эксперимент 6: Сравнение алгоритмов сортировки

6.1 Цель эксперимента

Исследовать способы эффективного использования памяти и выявление наиболее эффективных алгоритмов сортировки, применимых в вычислительных системах. 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).

6.2 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.

6.3 Условия эксперимента

1) Единицы измерения по Ox - Размер массива;
2) Единицы измерения по Oy - Такты;
3) Количество 64-х разрядных элементов массивов: **4**;
4) Шаг увеличения размера массива: **1024**;

6.4 Результаты эксперимента

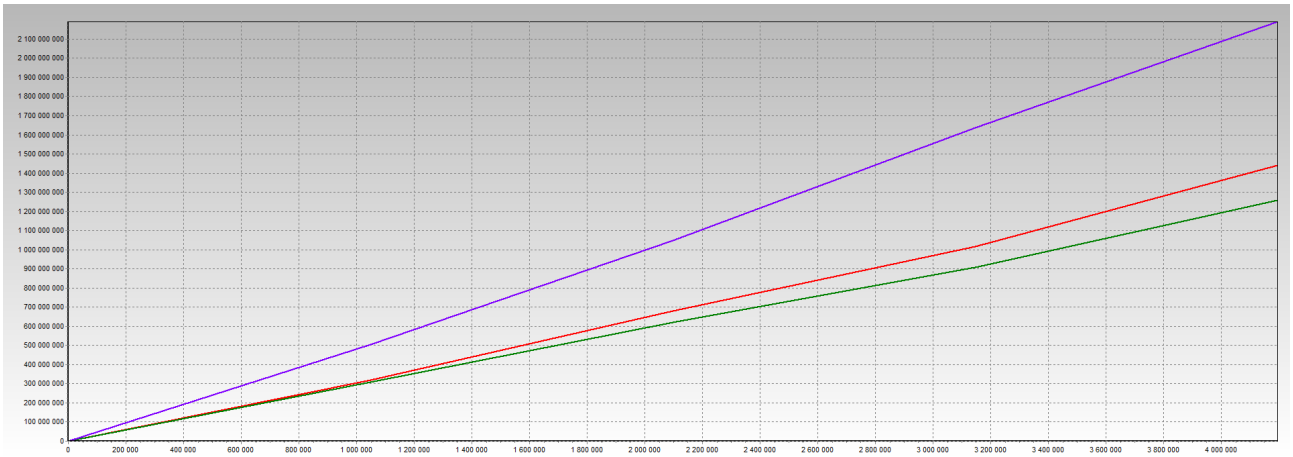

Рисунок 6.1 – Эксперимент 5: Сравнение алгоритмов сортировки

```
QuickSort работал в 1,5571845 раз дольше Radix-Counting Sort.
QuickSort работал в 1,7368255 раз дольше Radix-Counting Sort, оптимизированного под 8-процессорную ЭВМ.
```

Рисунок 6.2 – Эксперимент 6: Результаты

Как видно на рисунке 6.2, QuickSort работал в 1,5571845 раз дольше Radix-Counting Sort, и QuickSort работал в 1,7368255 раз дольше Radix-Counting Sort, оптимизированного под 8-процессорную ЭВМ.

6.5 Вывод

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

более быстрой работы.

7 Заключение

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.