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 «Компьютерные системы и сети (ИУ6)»

НАПРАВЛЕНИЕ ПОДГОТОВКИ «09.03.04 Программная инженерия»

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №3
по курсу «Архитектура ЭВМ»

«Организация памяти суперскалярных ЭВМ»

Студент: ИУ7-53Б _____ М. Д. Маслова
(группа) (подпись, дата) (И. О. Фамилия)

Преподаватель: _____ Е. Н. Дубровин
(подпись, дата) (И. О. Фамилия)

Москва, 2021

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. Работа проводится с использованием программы для сбора и анализа производительности PCLAB.

Для достижения поставленной цели необходимо выполнить следующие **задачи**:

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

1 Основные теоретические сведения

В данном разделе представлено описание программы **PCLAB**, ипользуемой при исследовании производительности в данной лабораторной работе.

1.1 Программа PCLAB

Программа **PCLAB** предназначена для исследования производительности x86 совместимых ЭВМ cIA32 архитектурой, работающих под управлением операционной системы Windows (версий 95 и старше). Исследование организации ЭВМ заключается в проведении ряда экспериментов, направленных на построение зависимостей времениобработки 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. Для измерения времени работы циклов в **PCLAB** используется следующая методика:

– длительность обработки участка профилируемой программы характеризуется изменением величины счетчика тактов процессора, произошедшим за время его работы;

– для предотвращения влияния соседних участков кода на результаты измерений, передначалом замера и после его окончания необходимо выдать команду упорядоченноговыполнения CPUID, препятствующую переупорядочивание потока команд на конвейере процессора;

– замеры количества тактов процессора необходимо повторить несколько раз;

– взаимное влияние последовательных повторов экспериментального участка программыисключается благодаря очищению кэш-памяти и буферов процессора;

– часть граничных результатов отбрасывается (как наибольших, так и наименьших).

2 Практическая часть

В данном разделе представлен ход выполнения лабораторной работы.

2.1 Задание 1

На листинге 2.1 представлены характеристики процессора компьютера в лабороторной аудитории, которые получены в программе **PCLAB**.

Листинг 2.1 – Идентификация процессора

```
 eax in    eax      ebx      ecx      edx
00000000 0000000d 756e6547 6c65746e 49656e69
00000001 0001067a 01020800 0400e3bd bfebfbff
00000002 05b0b101 005657f0 00000000 2cb43078
00000003 00000000 00000000 00000000 00000000
00000004 00000000 00000000 00000000 00000000
00000005 00000040 00000040 00000003 00022220
00000006 00000001 00000002 00000003 00000000
00000007 00000000 00000000 00000000 00000000
00000008 00000400 00000000 00000000 00000000
00000009 00000000 00000000 00000000 00000000
0000000a 07280202 00000000 00000000 00000503
0000000b 00000000 00000000 00000000 00000000
0000000c 00000000 00000000 00000000 00000000
0000000d 00000000 00000000 00000000 00000000
80000000 80000008 00000000 00000000 00000000
80000001 00000000 00000000 00000001 20000000
80000002 65746e49 2952286c 6c654320 6e6f7265
80000003 20295228 20555043 20202020 45202020
80000004 30303333 20402020 30352e32 007a4847
80000005 00000000 00000000 00000000 00000000
80000006 00000000 00000000 04004040 00000000
80000007 00000000 00000000 00000000 00000000
80000008 00003024 00000000 00000000 00000000

Undocument layers
80860000 00000000 00000000 00000000 00000000
80860001 00000000 00000000 00000000 00000000
80860002 00000000 00000000 00000000 00000000
80860003 00000000 00000000 00000000 00000000
80860004 00000000 00000000 00000000 00000000
80860005 00000000 00000000 00000000 00000000
80860006 00000000 00000000 00000000 00000000
80860007 00000000 00000000 00000000 00000000
```

Листинг 2.1 (продолжение)

```
c0000000 00000000 00000000 00000000 00000000
c0000001 00000000 00000000 00000000 00000000
8ffffffe 00000000 00000000 00000000 00000000
8fffffff 00000000 00000000 00000000 00000000

_____________________________________________
Vendor ID: "GenuineIntel"; CPUID level 13

Дополнительные функции Intel:
Верисия 0001067a:
Type 0 - Original OEM
Family 6 - Pentium Pro
Model 7 - Pentium III/Pentium III Xeon - external L2 cache
Stepping 10
Reserved 4

Extended brand string: "Intel(R) Celeron(R) CPU        E3300  @ 2.50GHz"
CLFLUSH instruction cache line size: 8
Initial APIC ID: 1
Hyper threading siblings: 2

Feature flags bfebfbff:
0   FPU         Присутствует Математический сопроцессор
1   VME         Поддержка расширенных возможностей обработки прерываний в режиме
     виртуального i8086
2   DE          Поддержка отладки
3   PSE         Поддержка страниц размером 4 MB
4   TSC         Счетчик меток реального времени
5   MSR         Поддержка команд rdmsr и wrmsr
6   PAE         Поддержка физического адреса более 32 бит
7   MCE         Поддержка исключений 18 - об аппаратных ошибках
8   CX8         Поддержка инструкции cmpxchg8b
9   APIC        Микропроцессор содержит программно доступный контроллер прерыван
    ий
11  SEP         Поддержка инструкций быстрых системных вызовов sysenter и
    sysexit
12  MTRR        Поддержка регистра mtrr_cap (относится к MSR-регистрам)
13  PGE         Поддержка глобальных страниц
14  MCA         Поддержка архитектуры машинного контроля
15  CMOV        Поддержка инструкций условной пересылки cmov, fcmovcc, fcomi
16  PAT         Процессор поддерживает таблицу атрибутов страницы
17  PSE-36      Процессор поддерживает 4 MB страницы, которые способны адресоват
    ь физическую память до 64 GB
19  CLFLSH      Поддержка инструкции CLFLUSH
21  DS          Поддержка записи отладочной информации
22  ACPI        Управление охлаждением процессора с помощью пустых циклов в зави
    симости от температуры
```

Листинг 2.1 (продолжение)

```
23  MMX         Поддержка MMX
24  FXSR        Поддержка инструкций FXSAVE и FXRSTOR
25  SSE         Поддержка SSE
26  SSE2        Поддержка SSE2
27  SS          Управление конфликтующими типами памяти
28  HTT         Поддержка Hyper-Threading
29  TM          Поддержка автоматического мониторинга температуры
31  SBF         Сигнал Останова при FERR

TLB and cache info:
b1: unknown TLB/cache descriptor
b0: дескриптор TLB-команд, 4K страницы, acc. 4-направ., 128 элементов
05: unknown TLB/cache descriptor
f0: unknown TLB/cache descriptor
57: unknown TLB/cache descriptor
56: unknown TLB/cache descriptor
78: unknown TLB/cache descriptor
30: L1 кэш-команд, 32 KB, acc. 8-направ., длина строки 64 байта
b4: unknown TLB/cache descriptor
2c: L1 кэш-данных, 32 KB, acc. 8-направ., длина строки 64 байта
Processor serial: 0001-067A-BFEB-FBFF-0400-E3BD
```

2.2 Задание 2

На основании идентификационнй информации о микропроцессоре ЭВМ, представленной в задании 1 были определены следующие параметры:

– размер линейка кэш-памяти верхнего уровня (L1): 64 байта;

– объем физической памяти: 64 КБ

2.3 Задание 3

Название эксперимента – исследования расслоения динамической памяти.

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

2.3.1 Исходные данные

1. Размер линейки кэш-памяти верхнего уровня.
2. Объем физической памяти.

2.3.2 Результаты эксперимента

1. Количество банков динамической памяти.
2. Размер одной страницы динамической памяти.
3. Количество страниц в динамической памяти.

2.3.3 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.

2.3.4 Суть эксперимента

Для определения способа трансляции физического адреса приф ормировании 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. Для проведения эксперимента необходимо задать изменяемые параметры.

2.3.5 Проведение эксперимента

Изменяемые параметры:

1. Максимальное расстояние между читаемыми блоками = 6.
2. Шаг увеличения расстояния между читаемыми 4-х байтовыми ячейками = 8.
3. Количество страниц в динамической памяти = 2.

На рисунке 2.1 представлен график, полученный в результате эксперимента.

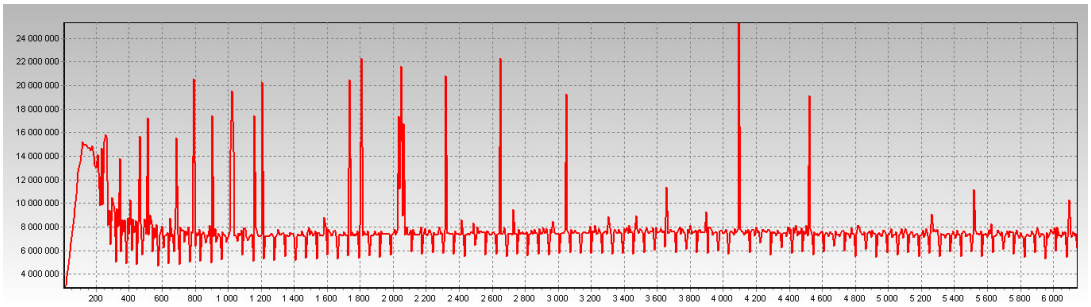

Рисунок 2.1 – Результат эксперимента 1 (байты – такты)

Получим значения следующих величин:

$$B = T_1/M, \tag{2.1}$$

где B – количество банков; T_1 – минимальный шаг чтения динамической памяти, при котором происходит постоянное обращение к одному и тому же банку; M – объем данных, являющийся минимальной порцией обмена кэш-памяти верхнего уровня с оперативной памятью и соответствует размеру линейки кэш-памяти верхнего уровня

А также

$$S = T_2/B, \tag{2.2}$$

где B – количество банков; T_2 – соответствует расстоянию (в байтах) между началом двух последовательных страниц одного банка; S – количество секторов.

В результате эксперимента было получено, что $T_1 = 128$ и $T_2 = 4096$, тогда:

$$B = T_1/M = 128/64 = 2; \tag{2.3}$$

$$S = T_2/B = 4096/2 = 2048. \tag{2.4}$$

2.3.6 Вывод

Необходимо учитывать расслоение памяти при обработке данных, потому что оперативная память неоднородна, поэтому для обращение к последовательно расположенным данным может потребоваться разное количество времени.

2.4 Задание 4

Название эксперимента – сравнение эффективности ссылочных и векторных структур.

Цель эксперимента – оценка влияния зависимости команд по данным на эффективность вычислений.

2.4.1 Результаты эксперимента

Отношение времени работы алгоритма, использующего зависимые данные, ко времени обработки аналогичного алгоритма обработки независимых данных.

2.4.2 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.

2.4.3 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 междуэлементами списка.

2.4.4 Проведение эксперимента

Изменяемые параметры:

1. Количество элементов в списке = 1.
2. Максимальная фрагментации списка = 256.
3. Шаг увеличения фрагментации = 4.

На графике, который представлен на рисунке 2.2:

– **красная линия** – количество тактов работы алгоритма с использованием списка;

– **зеленая линия** – количество тактов работы алгоритма с использованием массива.

При этом на оси абсцисс указана фрагментация списка.

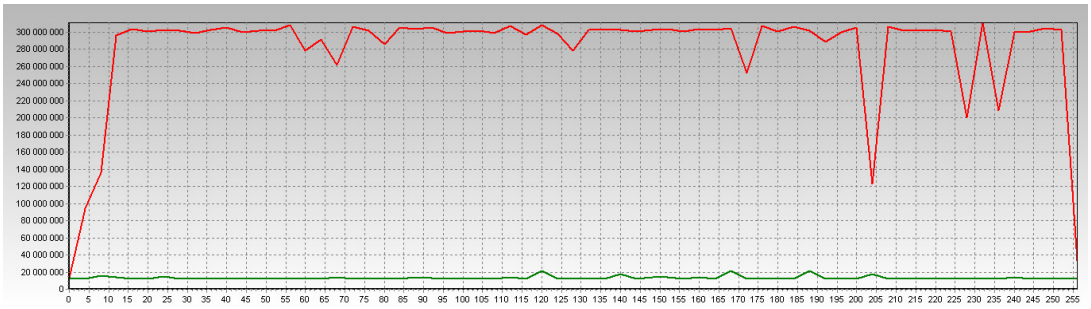

Рисунок 2.2 – График эксперимента 2 (килобайты – такты)

```
№2:Сравнение эффективности ссылочных и векторых структур данных (23.10.2021 : 12:26:16)
Результаты эксперимента:
...
_____________________________________________________________
Список обрабатывался в 21,11339 раз дольше.
_____________________________________________________________
```

Рисунок 2.3 – Результат эксперимента 2

На рисунке 2.3 представлен результат сравнения, на котором видно, что список обрабатывается в 21,11339 раз дольше массива.

2.4.5 Вывод

Необходимо учитывать технологический фактор решаемой задачи, от которой и зависит то, какая структура данных подходит больше. В данном случае приорететным является использование массива.

2.5 Задание 5

Название эксперимента – исследование эффективности программной предвыборки.

Цель эксперимента – оценка влияния зависимости команд по данным на эффективность вычислений.

2.5.1 Исходные данные

1. Степень ассоциативности.
2. Размер TLB данных.

2.5.2 Результаты эксперимента

Отношение времени последовательной обработки блока данных ко времени обработки блока с применением предвыборки; время и количество тактов первого обращения к странице данных.

2.5.3 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.

2.5.4 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 предвыборки, обеспечивающий своевременную загрузку информации в TLB данных.

2.5.5 Проведение эксперимента

Изменяемые параметры:

1. Шаг увеличения расстояния между читаемыми данными = 64.
2. Размер массива = 128.

На графике, который представлен на рисунке 2.4:

– **красная линия** – количество тактов работы алгоритма без использования предвыборки;

– **зеленая линия** – количество тактов работы алгоритма с использованием предвыборки.

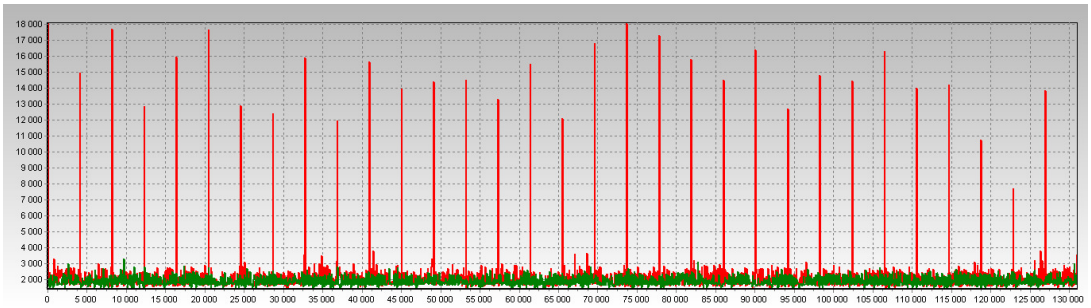

Рисунок 2.4 – График эксперимента 3 (байты – такты)

```
№3:Исследование эффективности программной предвыборки (23.10.2021 : 12:55:57)
Результаты эксперимента:
...
_____________________________________________________________________
Обработка без загрузки таблицы страниц в TLB производилась в 1,1642163 раз дольше.
_____________________________________________________________________
```

Рисунок 2.5 – Результат эксперимента 3

На рисунке 2.5 представлен результат сравнения, на котором видно, что обработка без загрузки страниц в TLB в 1,1642163 раза дольше.

2.5.6 Вывод

При использовании заблаговременной загрузки страниц можно получить ускорение работы программы.

2.6 Задание 6

Название эксперимента – исследование способов эффективного чтения оперативной памяти.

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

2.6.1 Исходные данные

1. Адресное расстояние между банками памяти.
2. Размер буфера чтения.

2.6.2 Результаты эксперимента

Отношение времени обработки блока памяти неоптимизированной структуры ко времени обработки блока структуры, обеспечивающей эффективную загрузку и параллельную обработку данных.

2.6.3 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.

Для создания структур данных, оптимизирующих их обработку современными процессорами, требуется максимально исключить несвоевременную передачу данных, т.е. передавать в каждом пакете только востребованную для вычислений информацию. В результате такой оптимизации снижается количество кэш-промахов, сокращается количество открытий и закрытий страниц DRAM-памяти, обеспечивается параллельная обработка данных и выполнение операций загрузки и выгрузки.

2.6.4 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.

2.6.5 Проведение эксперимента

Изменяемые параметры:

1. Размер массива = 2.
2. Количество потоков данных = 64.

На графике, который представлен на рисунке 2.6:

– **красная линия** – количество тактов работы алгоритма, который использует неоптимизированную структуру;

– **зеленая линия** – количество тактов работы алгоритма с использованием оптимизированной структуры.

На рисунке 2.7 представлен результат сравнения, на котором видно, что неоптимизированная структура обрабатывается в 1.6532999 раза дольше.

2.6.6 Вывод

Чем лучше упорядочены данные, тем быстрее работает алгоритм.

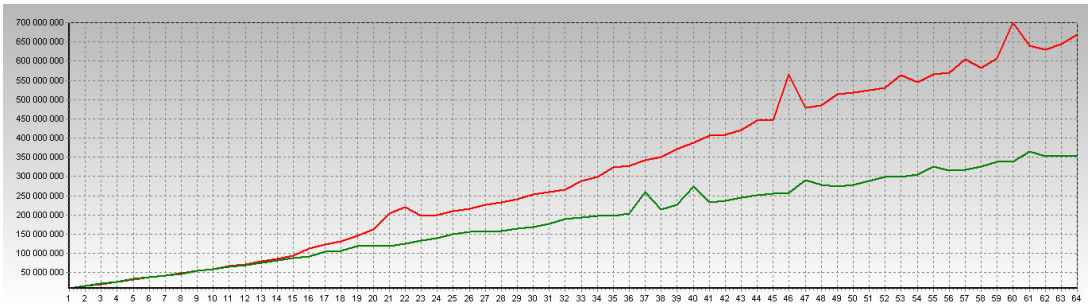

Рисунок 2.6 – График эксперимента 4 (количество параллельных потоков – такты)

```
№4:Исследование способов эффективного  чтения оперативной памяти (23.10.2021 : 13:09:37)
Результаты эксперимента:
...
_____________________________________________________________
Неоптимизированная структура обрабатывалась в 1,6532999 раз дольше.
_____________________________________________________________
```

Рисунок 2.7 – Результат эксперимента 4

2.7 Задание 7

Название эксперимента – исследование конфликтов в кэш-памяти.

Цель эксперимента – исследование влияния конфликтов кэш-памяти наэффективность вычислений.

2.7.1 Исходные данные

1. Размер банка кэш-памяти данных первого и второго уровня.
2. Степень ассоциативности кэш-памяти первого и второго уровн.
3. Размер линейки кэш-памяти первого и второго уровня.

2.7.2 Результаты эксперимента

Отношение времени обработки массива с конфликтами вкэш-памяти ко времени обработки массива без конфликтов.

2.7.3 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.

2.7.4 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.

2.7.5 Проведение эксперимента

Изменяемые параметры:

1. Размер банка кэш-памяти = 256.
2. Размер линейки кэш-памяти = 8.
3. Количество читаемых линеек = 64.

На графике, который представлен на рисунке 2.8:

– **красная линия** – количество тактов работы функции, которая читает данные с конфилктами в кэш-памяти;

– **зеленая линия** – количество тактов работы функции, которая не вызывает конфликтов в кэш-памяти.

При этом на оси абсцисс отражено смещение читаемой ячейки от начала блока данных.

В результате работы сравнения было определено, что с конфликтами данных производится в 6.7020765 раза дольше.

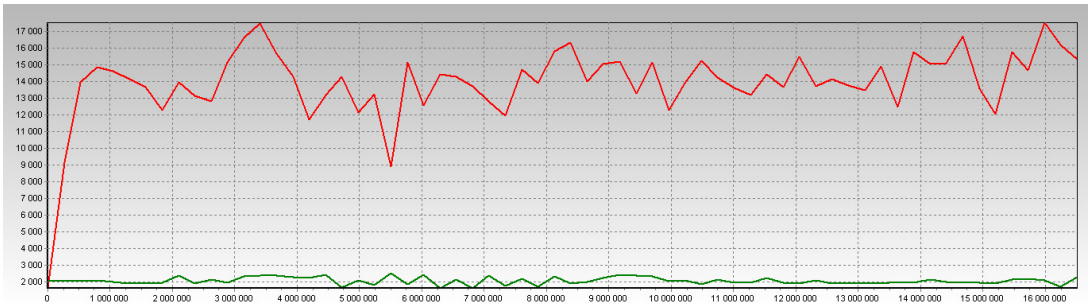

Рисунок 2.8 – График эксперимента 5 (смещение от начала блока – такты)

```
№5:Исследование конфликтов в кэш-памяти (23.10.2021 : 13:30:21)
Результаты эксперимента:
...
_____________________________________________________________
Чтение с конфликтами банков производилось в 6,7020765 раз дольше.
_____________________________________________________________
```

Рисунок 2.9 – Результат эксперимента 5

2.7.6 Вывод

Использование кэш-памяти позволяет значительно ускорить работу процессора.

2.8 Задание 8

Название эксперимента – сравнение алгоритмов сортировки. **Цель эксперимента** – сследование способов эффективного использования памяти ивыявление наиболее эффективных алгоритмов сортировки, применимых в вычислительных системах.

2.8.1 Исходные данные

1. Количество процессоров вычислительной системы.
2. Размер пакета.
3. Количество элементов в массиве.
4. Разрядность элементов массива.

2.8.2 Результаты эксперимента

Отношение времени сортировки массива алгоритмом *QuickSort* ко времени сортировки алгоритмом *Radix-Counting Sort* и ко времени сортировки

Radix-Counting Sort, оптимизированной под 8-процессорную вычислительную систему.

2.8.3 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 $O(n^2)$, $O(n \log n)$, $O(n)$, $O(\frac{n}{\log n})$), емкостная сложность (алгоритмы, требующие и нетребующие дополнительного массива), возможность распараллеливания (нераспараллеливаемые, ограниченно распараллеливаемые, полностью распараллеливаемые), принцип определения порядка (алгоритмы, использующие парные сравнения и неиспользующие парные сравнения).

2.8.4 Суть эксперимента

Эксперимент основан на замере времени трех вариантов алгоритмов сортировки (*Quick Sort*, *Radix-Counting Sort*, оптимизированный *Radix-Counting Sort*).

2.8.5 Проведение эксперимента

Изменяемые параметры:

1. Количество 64-х разрядных элементов массивов = 20.
2. Шаг увеличения размера массива = 1024.

На графике, который представлен на рисунке 2.10:

– **фиолетовая линия** – количество тактов работы алгоритма QuickSort;

– **красная линия** – количество тактов работы алгоритма неоптимизированного алгоритма Radix-Counting;

– **зеленая линия** – количество тактов работы оптимизированного под 8-процессорную вычислительную систему алгоритма Radix-Counting.

При этом на оси абсцисс отражено количество 64-разрядных элементов сортируемых массивов, а на оси ординат – количество тактов.

Также на рисунке 2.11 представлен результат сравнения, на котором видно, что *QuickSort* работал в 2,0717642 раза дольше, чем *Radix-Counting Sort*,

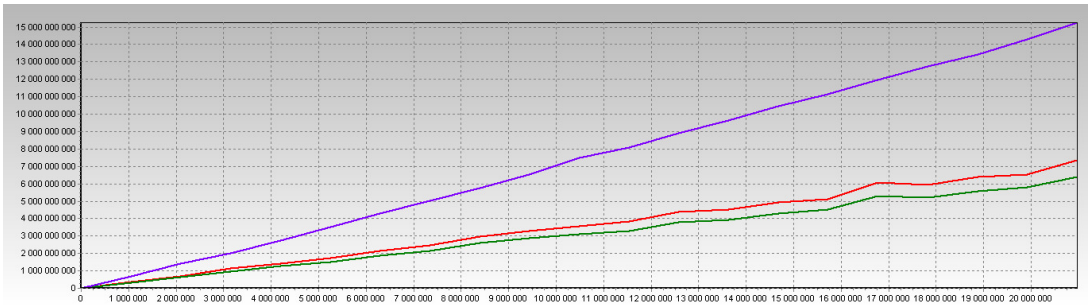

Рисунок 2.10 – График эксперимента 6 (число элементов – такты)

```
№6:Сравнение алгоритмов сортировки (RadixSort-CountingSort и QuickSort) (23.10.2021 : 13:33:14)
Результаты эксперимента:
...
_____________________________________________________________________
QuickSort работал в 2,0717642 раз дольше Radix-Counting Sort.
QuickSort работал в 2,3783462 раз дольше Radix-Counting Sort, оптимизированного под 8-процессорную ЭВМ.
_____________________________________________________________________
```

Рисунок 2.11 – Результат эксперимента 6

и в 2,3783462 раза дольше, чем *Radix-Counting Sort*, оптимизированного под 8-процессорную ЭВМ.

2.8.6 Вывод

Radix-Counting Sort является более быстрой сортировкой, чем QuickSort, при этом даже Radix-Counting Sort можно улучшить, оптимизировав его под 8-процессорную ЭВМ.

3 Контрольные вопросы

В данном разделе представлены ответы на контрольные вопросы.

– **Назовите преимущества и недостатки аппаратных ускорителей на ПЛИС по сравнению с CPU и графическими ускорителями**

Преимущества:

– создание специализированной вычислительной структуры для реализации желаемой функциональности;

– низкая стоимость в сравнению с обычными аппаратными ускорителями;

– большая частота эмуляции;

– компактность.

Недостатки:

– необходимость перекомпиляции проекта и переконфигурации ПЛИС при любом исправлении содержимого проекта;

– наличие специализированного программного обеспечения для разделения модели микросхемы на части для загрузки в отдельные ПЛИС.

– **Назовите основные способы оптимизации циклических конструкций ЯВУ, реализуемых в виде аппаратных ускорителей**

– конвейерная обработка;

– разворачивание циклов;

– потоковая обарботка.

– **Назовите этапы работы программной части ускорителя в хост системе**

Этап 1. Инициализация среды OpenCL.

Этап 2. Создание трех буферов, необходимых для обмена данными с ядром: два буфера для передачи исходных данных и один для вывода результата.

Этап 3. Запуск задачи на исполнение.

Этап 4. Чтение выходного буфера, содержащего результаты работы ядра, после завершения работы всех команд.

– **В чем заключается процесс отладки для вариантов сборки Emulation-SW, Emulation-HW и Hardware?**

  – *Программная эмуляция* (Emulation-SW) — код ядра компилируется для работы на ЦПУ хост-системы. Этот вариант сборки служит для верификации совместного исполнения кода хост-системы и кода ядра, для выявления синтаксических ошибок, выполнения отладки на уровне исходного кода ядра, проверки поведения системы.

  – *Аппаратная эмуляция* (Emulation-HW) — код ядра компилируется в аппаратную модель (RTL), которая запускается в специальном симуляторе на ЦПУ. Этот вариант сборки и запуска занимает больше времени, но обеспечивает подробное и точное представление активности ядра. Данный вариант сборки полезен для тестирования функциональности ускорителя и получения начальных оценок производительности.

  – *Аппаратное обеспечение* (Hardware) - код ядра компилируется в аппаратную модель (RTL), а затем реализуется на FPGA. В результате формируется двоичный файл xclbin, который будет работать на реальной FPGA.

– **Какие инструменты и средства анализа результатов синтеза возможно использовать в Vitis HLS для оптимизации ускорителей?**

  – компилятор Xilinx Vitis v++;

  – Assistant View (получение отчетов о сборке аппартных ядер);

  – внутрисхемный отладчик Vivado (отслеживание любых сигналов ускорителя для анализа событий);

  – сводный отчет Link Summary (Vitis Analyzer, получение диаграмм системы и платформы и др.)

Заключение

В ходе лабораторной работы были изучены

Были выполнены следующие задачи:

–

–

–

–

–

–

–

Таким образом, все поставленные задачи были выполнены, а цель достигнута.