МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное
образовательное учреждение высшего образования
«Южно-Уральский государственный университет
(национальный исследовательский университет)»

Высшая школа электроники и компьютерных наук
Кафедра «Электронные вычислительные машины»

РАБОТА ПРОВЕРЕНА
Рецензент,
к.т.н., доцент
_____Б.В. Винников
«___»_____ 2019 г.

ДОПУСТИТЬ К ЗАЩИТЕ
Заведующий кафедрой ЭВМ
_____ Г.И. Радченко
«___»_____ 2019 г.

Исследование вычислительных свойств ассоциативного решающего поля на примере задач сортировки массивов

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА К ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЕ

Руководитель работы,
к.т.н., доцент каф. ЭВМ
_____И.Л. Кафтанников
«___»_____2019 г.

Автор работы,
студент группы КЭ-222
_____А.Д. Прасолова
«___»_____2019 г.

Нормоконтролёр,
ст. преп. каф. ЭВМ
_____ С.В. Сяськов
«___»_____2019 г.

Челябинск-2019

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное
образовательное учреждение высшего образования
«Южно-Уральский государственный университет
(национальный исследовательский университет)»
Высшая школа электроники и компьютерных наук
Кафедра «Электронные вычислительные машины»

УТВЕРЖДАЮ
Заведующий кафедрой ЭВМ
_____ Г.И. Радченко
«___»_____2019 г.

ЗАДАНИЕ

на выпускную квалификационную работу бакалавра

студенту группы КЭ-222
Прасоловой Анастасии Дмитриевны
обучающемуся по направлению
09.04.01 «Информатика и вычислительная техника»

1. **Тема работы: «**Исследование вычислительных свойств ассоциативного решающего поля на примере задач сортировки массивов» утверждена приказом по университету от 25 апреля 2019 г. №899

2. **Срок сдачи студентом законченной работы:** 1 июня 2019 г.

3. **Исходные данные к работе:** Патентные базы данных, научные статьи, диссертации, книги.

- Qing, G., AC-DIMM: Associative Computing with STT-MRAM / G. Qing, G. Xiaochen, R. Patel, E. Ipek, G. Friedman // ISCA’13Tel-Aviv. – 2013. – P. 189–200.

- Yavits, L. Resistive associative processor/ L. Yavits, S. Kvatinsky, A. Morad, R. Ginosar // IEEE Computer Architecture Letters. – 2015. – P. 148-151.
- Патентная база данных – Федеральная служба по интеллектуальной собственности.
- Кнут, Д. Искусство программирования в 4 томах, том 3. Сортировка и поиск / Дональд Кнут; пер. с анг. В. Т. Тертышный, И. В. Красиков. М.: Вильямс, 2001. – 720 с.

4. **Перечень подлежащих разработке вопросов:**

- анализ классических методов повышения производительности вычислительных систем;
- функциональный анализ ассоциативной памяти;
- исследование вычислительных свойств АРП при выполнении операции сортировки;
- разработка нового алгоритма сортировки и анализ его производительности.

5. **Дата выдачи задания:** 1 декабря 2018 г.

Руководитель работы _____/*И.Л. Кафтанников*/

Студент _____/А.*Д. Прасолова*/

КАЛЕНДАРНЫЙ ПЛАН

Этап	Срок сдачи	Подпись руководителя
Поиск и анализ научно-исследовательских работ по теме ВКР	01.03.2019	
Составление библиографического списка	01.04.2019	
Обоснование актуальности выбранной темы, определение целей и задач ВКР	01.05.2019	
Написание пояснительной записки ВКР	15.05.2019	
Компоновка текста работы и сдача на нормоконтроль	24.05.2019	
Подготовка презентации и доклада	30.05.2019	

Руководитель работы _____/ *И.Л. Кафтанников* /

Студент _____/ А.*Д. Прасолова* /

Аннотация

А.Д. Прасолова. Исследование вычислительных свойств ассоциативного решающего поля на примере задач сортировки массивов. – Челябинск: ФГАОУ ВО «ЮУрГУ (НИУ)», ВШЭКН; 2019, 68 с., 25 ил., библиогр. список – 23 наим.

Ассоциативные решающие поля (АРП) обладают большим потенциалом повышения производительности обработки информации. Но проведено мало исследований вычислительных свойств данного вида памяти. Этим обуславливается актуальность данной работы.

В рамках выпускной квалификационной работы проводятся теоретические исследования особенностей работы ассоциативного решающего поля, существующих разработок АРП, возможности реализации в АРП различных известных алгоритмов сортировки. Разрабатывается алгоритм, повышающий алгоритмическую производительность с учетом структуры и свойств АРП. Проводится оценка производительности алгоритма.

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Внедрение информационных технологий в различные сферы жизни человека, автоматизация производства, совершенствование устройств контроля, технологических процессов приводят к увеличению объема информации. Современные устройства не всегда обеспечивают требуемое быстродействие.

Для увеличения производительности вычислительных устройств в настоящее время используется распараллеливание обработки информации. Одним из методов распараллеливания является применение вычислительных устройств на базе SIMD архитектур. Такими архитектурами являются графические процессоры и ассоциативная память (АП). В настоящей работе осуществляется анализ вычислительных свойств АП.

Отличительной особенностью ассоциативного запоминающего поля (АРП) заключается в том, что в АРП можно получить хранящиеся в них данные без предварительного указания адресов соответствующих ячеек. Для поиска информации используется некоторая совокупность свойств или описаний объекта. Также АП позволяет получить параллельный доступ ко всем ячейкам памяти.

На сегодняшний день появилось такое технологическое решение для изготовления АП, как магниторезистивная память (MRAM). MRAM хранит информацию в виде различного сопротивления элементов хранения. Это обеспечивает уменьшение энергопотребления, увеличения срока службы и быстродействия по сравнению с другими видами памяти.

Ассоциативная память (АП) обладает большим потенциалом повышения производительности обработки информации. Но проведено мало исследований по внедрению и возможностям использования данного вида памяти. Этим обуславливается актуальность данной работы.

Вычислительные свойства АП изучены еще недостаточно, поэтому в работе осуществляется анализ одной из наиболее распространенных задач

обработки информации – сортировки, как одного из наиболее ресурсозатратных преобразований.

Цель и задачи исследования

Цель научно-исследовательской работы – провести теоретические исследования особенностей работы ассоциативного решающего поля, возможности реализации в АРП различных известных алгоритмов сортировки и разработать алгоритм, повышающий алгоритмическую производительность с учетом структуры и свойств АРП.

Для достижения поставленной цели необходимо решить следующие задачи:

- исследование и выявление особенностей функционирования ассоциативного решающего поля;
- анализ существующих разработок АРП;
- анализ и исследование отображения существующих алгоритмов преобразований на АРП;
- разработка нового алгоритма сортировки, учитывая особенности архитектуры и функционирования АП.

1. АНАЛИЗ КЛАССИЧЕСКИХ МЕТОДОВ ПОВЫШЕНИЯ ПРОИЗВОДИТЕЛЬНОСТИ ВЫЧИСЛИТЕЛЬНЫХ СИСТЕМ

1.1. СУЩЕСТВУЮЩИЕ ОГРАНИЧЕНИЯ ПОВЫШЕНИЯ БЫСТРОДЕЙСТВИЯ

Внедрение информационных технологий в различные сферы жизни человека, автоматизация производства, совершенствование устройств контроля, технологических процессов приводят к увеличению объема информации. Современные системы не всегда обеспечивают требуемое быстродействие. Существуют следующие ограничения:

- узкое место архитектуры фон Неймана;
- проблема увеличения производительности процессоров.

1.1.1. УЗКОЕ МЕСТО АРХИТЕКТУРЫ ФОН НЕЙМАНА

Джон фон Нейман в 1946 году предложил организацию ЭВМ, которая опирается на следующие принципы.

- Принцип двоичного кодирования.
- Принцип однородности памяти. Данные и программы хранятся в одной памяти.
- Принцип адресуемости памяти. Память состоит из пронумерованных ячеек. Процессор в любой момент времени, зная адрес, имеет доступ к любой ячейке памяти.
- Принцип конвейерной организации выполнения команд. Программа состоит из определенной последовательности команд, которые выполняются процессором в определенном порядке.

В устройствах с классической архитектурой фон Неймана определенный класс задач становится трудно решить из-за больших объемов исходных данных. Совместное использование шины для памяти данных и памяти программ порождает узкое место архитектуры фон Неймана, а именно ограничение пропускной способности между процессором и

памятью. Вследствие того, что память для программ и для данных не может быть доступна одновременно, пропускная способность канала «процессор-память» и скорость доступа к памяти достаточно сильно ограничивают скорость работы процессора. Скорость работы процессора и объём памяти развивались гораздо быстрее, чем пропускная способность шины между ними. Поэтому узкое место стало большой проблемой, серьёзность которой возрастает с каждым новым поколением процессоров. Термин «узкое место архитектуры фон Неймана» ввёл Джон Бэкус в 1977 в своей лекции «Можно ли освободить программирование от стиля фон Неймана?», которую он прочитал при вручении ему Премии Тьюринга [1].

1.1.2. ПРОБЛЕМА УВЕЛИЧЕНИЯ ПРОИЗВОДИТЕЛЬНОСТИ ПРОЦЕССОРА

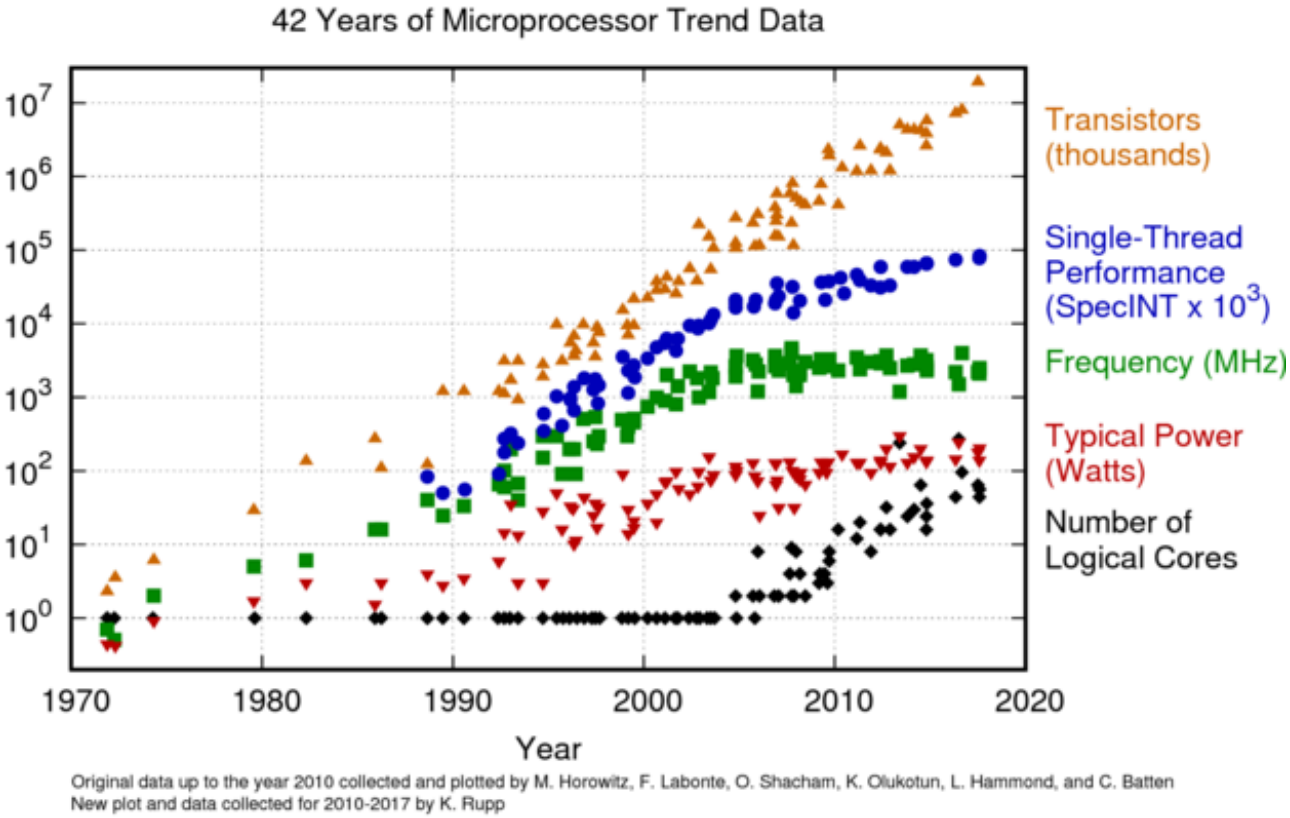

Рисунок 1.1 – Тенденции развития микропроцессоров

График, представленный на рисунке 1.1, был построен на основе данных, собранных М. Горовиц, Ф. Лабонте, О. Шахама, К. Олукотуна, Л. Хаммонд и К. Баттен. Информация после 2010 года была дополнена до 2017

года Карлом Раппом. График отражает тенденции развития микропроцессоров [2].

До 2000-ых годов тактовая частота процессора и потребляемая мощность росли согласно представленным на рисунке 1.1 зависимостям. Закон Мура выполнялся на протяжении многих лет благодаря явлению, сформулированному в законе Деннарда. Согласно этому закону, уменьшая размеры транзисторов на кристалле, можно пропорционально снижать подаваемое на затвор напряжение, но при этом переключающие свойства транзистора сохраняются, а скорость переключения повышается. Таким образом, уменьшая размеры транзистора и повышая тактовую частоту процессора, можно повышать его производительность.

На этом основывается заключение, определившее будущее развитие технологий на много десятилетий вперед: для увеличения производительности необходимо повысить частоту, количество транзисторов на кристалле, а также снижать энергопотребление.

Однако если взглянуть на дополнения графика, сделанные К. Раппом, станет понятно, что прогноз показывает стабилизацию производительности, несмотря на увеличение числа транзисторов на кристалле.

Первый процессор Intel 8086 с частотой 8MHz потреблял менее 2W. Pentium 3, работающий на частоте 1GHz, потреблял уже 33W. То есть энергопотребление возросло в 17 раз, а тактовая частота за то же время увеличилась в 125 раз.

Также из графика видно, что, во-первых, производительность однопоточной системы несколько увеличилась. Это увеличение было достигнуто благодаря управлению питанием и динамической настройке тактовой частоты. Во-вторых, увеличение количества ядер пропорционально количеству транзисторов.

В феврале 2003 года Мур опубликовал статью «No Exponential is Forever: But “Forever” Can Be Delayed!» [3], в которой признал, что экспоненциальный рост физических величин в течение длительного периода

времени невозможен, и постоянно достигаются те или иные пределы. Только лишь эволюция технологий изготовления транзисторов позволяла продлить действие закона еще на некоторое время.

Главная причина ограничения роста транзисторов состоит в том, что при уменьшении размеров транзисторов токи утечки создают все больше проблем: они приводят к нагреванию микросхемы, что, в свою очередь, ведет к тепловому разгону процессора и выходу его из строя.

Появление определенных пределов энергопотребления сопутствовали появлению такого ограничения, известного как Utilization Wall. Согласно этому ограничению с новым технологическим процессом часть площади кристалла задействованной в активной работе (где будут переключаться транзисторы) убывает экспоненциально. Эта площадь кристалла измеряется долями процентов. Оставшаяся большая доля кристалла, которая не была задействована в работе, и получила название «тёмный» кремний («Dark» Silicon) (см. рисунок 1.2).

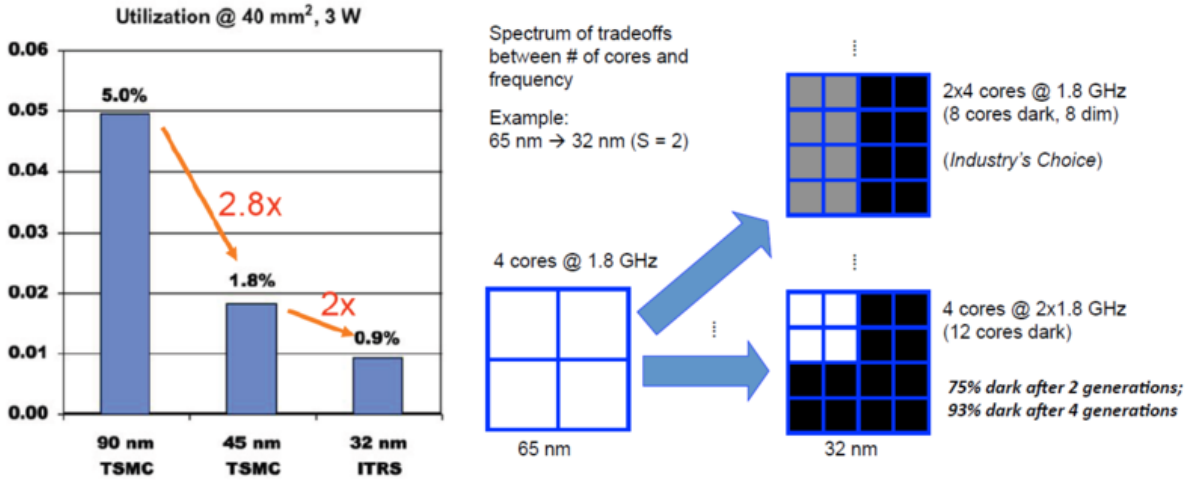

Рисунок 1.2 – Utilization Wall

Таким образом, увеличение количества транзисторов на кристалле упирается в определенные пределы выделяемой мощности процессора на единицу площади. При превышении этих пределов процессоры будут разрушаться от перегревания. И преодолеть эти пределы невозможно без использования нетрадиционных, громоздких и дорогостоящих систем

охлаждения. В результате с 2006 года частота микропроцессоров массового производства не растет выше 4ГГц.

1.2. ВЫЧИСЛИТЕЛЬНЫЕ СТРУКТУРЫ

Big Data — это различные подходы, инструменты и методы обработки структурированных и неструктурированных данных, чтобы в дальнейшем использовать их для конкретных задач. Для Big Data выделяют традиционные определяющие характеристики, которые называются «Три V»:

- Volume – величина физического объёма.
- Velocity – скорость прироста и необходимости быстрой обработки данных для получения результатов.
- Variety – возможность одновременно обрабатывать различные типы данных.

Возникновение феномена Big Data связано с появлением новых технологических возможностей для анализа огромного количества данных, например Machine Learning, Artificial Intelligence, Data mining.

Для задач, требующих вычислений над большим объемом данных, существуют суперкомпьютерные системы. Они могут быть основаны на многоядерных процессорах (Intel, AMD, IBM), векторных процессорах (NEC, Cray), графических процессорах (NVidia, AMD).

1.2.1.МНОГОЯДЕРНЫЕ ПРОЦЕССОРЫ

Многоядерные процессоры – это интегральная схема, которая использует два или более отдельных ядра на одном процессорном кристалле. Каждое ядро имеет собственный кэш. Высокая производительность многоядерных процессоров по сравнению с одноядерными заключается в разбиении выполнения потоков (различных задач) на несколько ядер. Проведя исследования в области разработок многоядерных устройств, можно заключить, что топология микропроцессора и его устройство не влияют на тот факт, что энергопотребление сильно ограничивает увеличение

количества ядер. Учитывая низкий прирост производительности, добавление новых ядер не даст желаемого преимущества для обоснования необходимости и окупаемости дальнейшего совершенствования технологического процесса.

На рисунке 1.3 представлены различные варианты исполнения многоядерной архитектуры. На рисунке 1.3(б) изображены два независимых ядра. Такое решение дает не только экономию места, но и экономию энергии, так как часть компонентов является общей для обоих ядер. На рисунках (в) и (г) ядра имеют общую кэш-память, но во втором случае ядра разделены на потоки.

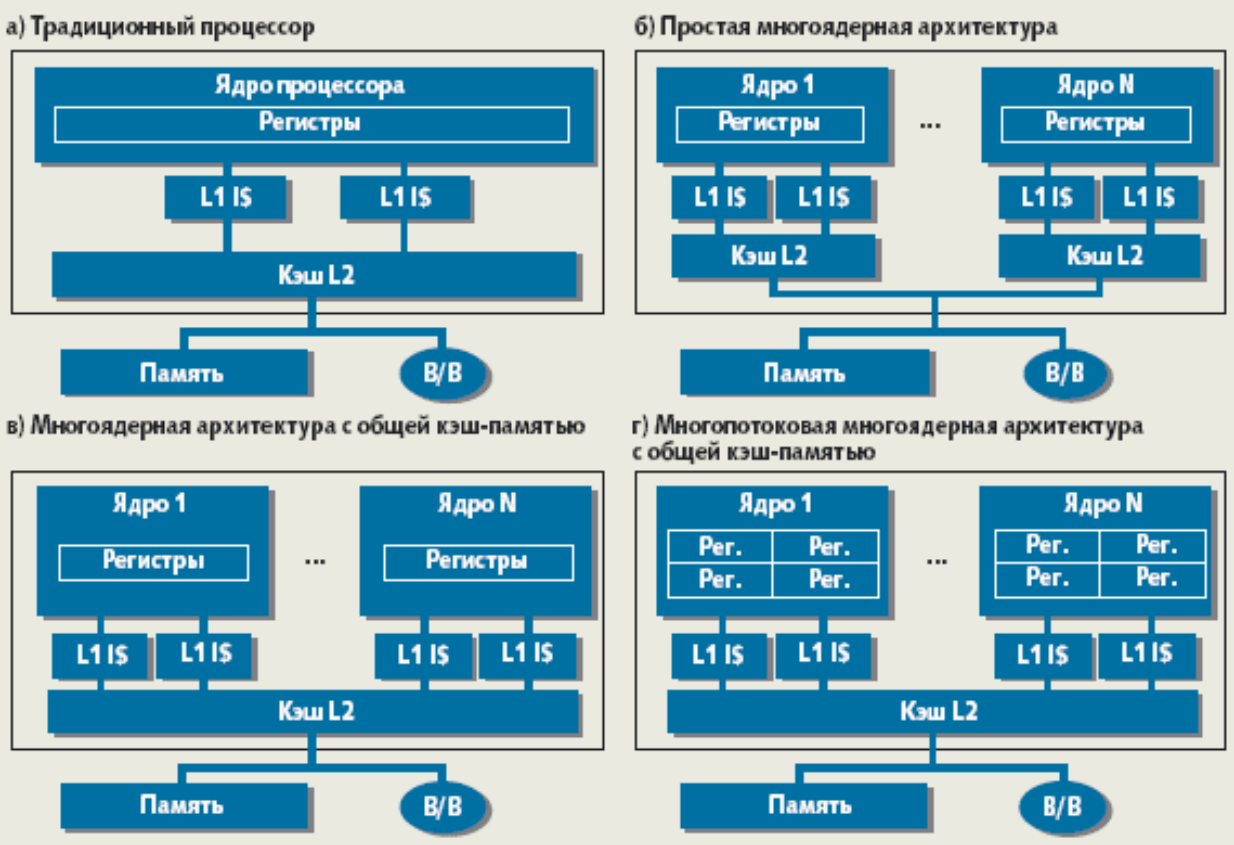

Рисунок 1.3 – Многоядерный процессор

1.2.2.ГРАФИЧЕСКИЕ ПРОЦЕССОРЫ

Графический процессор (Graphics Processing Unit, GPU) – микропроцессор, специализируется на обработке графической информации. Он отвечает за задачи по формированию трехмерного (3D) изображения и позволяет разгрузить центральный микропроцессор (CPU) от выполнения

операций, связанных с расчетами геометрических трансформаций, моделей освещения и т. д. Различия в устройстве процессоров представлено на рисунке 1.4.

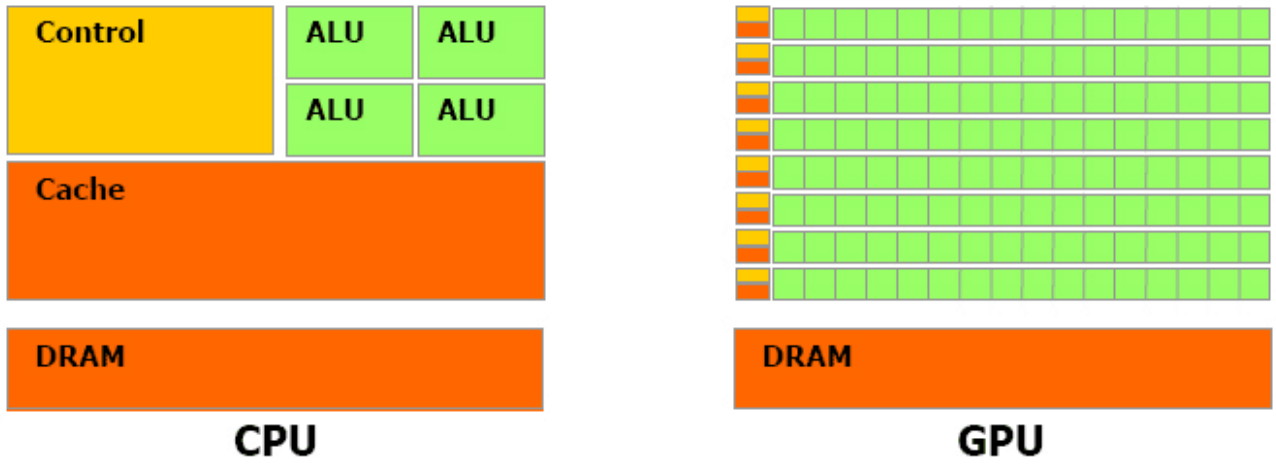

Рисунок 1.4 – Различия CPU и GPU

Современные модели графических процессоров могут полноценно применяться для общих вычислений. GPGPU (general purpose computing for graphics processing units) – техника использования графического процессора для выполнения вычислений в приложениях, которые обычно производит центральный процессор. Это стало возможным за счет добавления программируемых шейдерных блоков и более высокой арифметической точности растровых конвейеров. Это позволяет разработчикам программного обеспечения использовать потоковые процессоры для не-графических данных.

Хотя графический процессор имеет множество преимуществ, таких как большая вычислительная мощность, большая пропускная способность памяти и низкое энергопотребление в отношении высокой вычислительной мощности, существуют некоторые ограничения. Эти ограничения затрудняют оптимизацию производительности. Поэтому разработка кода с использованием графического процессора может занять больше времени и потребовать более сложной работы. Кроме того, код GPU работает параллельно. Но в некоторых случаях нет алгоритмов, которые могут быть

вписаны в GPU, поэтому необходимо разработать новые параллельные алгоритмы.

1.2.3.ВЕКТОРНЫЕ ПРОЦЕССОРЫ

Векторный процессор – это процессор, в котором операндами команд выступают упорядоченные массивы данных – векторы.

Пусть , и – это массивы, имеющие одинаковую размерность и одинаковую длину (формула 1).

,

, (1)

,

где – элементы массивов и , ,

– элементы массива , .

Имеется оператор, описанный формулой 2.

(2)

Векторный процессор за цикл выполнения одной команды произведет сложение соответствующих элементов массивов и и присвоит вычисленные значения соответствующим элементам массива . Это возможно благодаря тому, что все операнды хранятся в векторном регистре. Обычному последовательному процессору пришлось бы несколько раз выполнять операцию сложения элементов двух массивов. Векторный процессор выполняет лишь одну команду.

Многоядерные, графические и векторные процессоры, несмотря на определенное распараллеливание, используют известные принципы преобразования информации, основанные на конвейерной обработке. Вследствие этого имеются ограничения, связанные с тем, что алгоритм при программировании отображается на жестко зафиксированный набор команд вычислительной системы.

Также несовершенство суперкомпьютерных систем выражается в ограниченной пропускной способности шин передачи данных, неравномерной нагрузке между отдельными вычислительными узлами, проблеме параллельных вычислений – закон Амдала.

«В случае, когда задача разделяется на несколько частей, суммарное время её выполнения на параллельной системе не может быть меньше времени выполнения самого длинного последовательного фрагмента», то есть производительность любой параллельной программы ограничена последовательной частью кода, не поддающейся распараллеливанию.

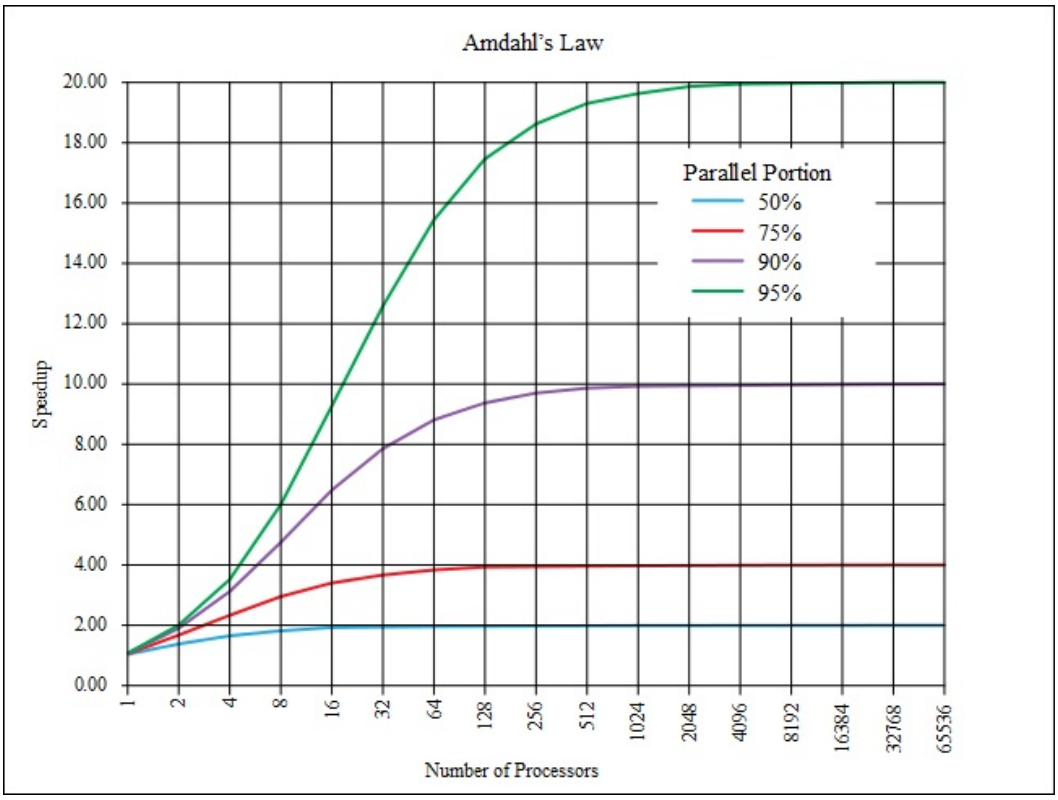

Рисунок 1.5 – Закон Амдала

Пусть необходимо решить задачу с некоторым объемом вычислений. Предположим, что алгоритм задачи такой, что какая-то часть от общего объёма вычислений может быть получена только последовательными расчётами, а, соответственно, остальная часть алгоритма может быть распараллелена (то есть время вычисления будет обратно пропорционально числу задействованных узлов). Тогда ускорение, полученное на

вычислительной системе, состоящей из процессоров, по сравнению с однопроцессорным решением не будет превышать величины (формула 3).

(3)

где – доля последовательных вычислений, %,

(– часть распараллеленных вычислений, %,

– количество вычислительных узлов, шт.

Зависимость ускорения от и представлена в таблице 1.

Таблица 1 – Зависимость ускорения от и

	10	100	1000
0%	10,000	100,000	1000,000
10%	5,263	9,174	9,910
25%	3,077	3,883	3,988
40%	2,174	2,463	2,496

Из формулы (3) и таблицы 1 видно, что алгоритм, вообще не содержащий последовательных вычислений , позволяет получить линейный прирост производительности с ростом количества вычислителей в системе. Если доля последовательных вычислений в алгоритме равна 10%, то увеличение числа процессоров до 10 дает ускорение в 5,263 раз, а увеличение числа процессоров до 1000 даст ускорение в 9,910 раз.

Можно сделать вывод, что при доле последовательных вычислений общий прирост производительности не может превысить . Так, если половина кода — последовательная, то общий прирост никогда не превысит двух.

1.2.4.НЕЙРОМОРФНЫЙ ПРОЦЕССОР TRUENORTH

На семинаре «IBM Innovation Day» были представлены Не-Неймановские вычислители. Нейроморфный процессор TrueNorth является

первым процессором, который не использует архитектуру фон Неймана. В отличие от традиционных вычислительных архитектур, логика IBM TrueNorth изначально проектировалась под исполнение рекуррентных нейронных сетей, которые могут применяться для классификации различной информации, в том числе, изображений, речи и видео. Также исследовательский центр IBM вместо традиционной модели «Compute-centric Model» предлагает «Data-centric Model».

Старая модель представляет собой архитектуру, в которой центральный процессор (CPU) является центром системы обработки, а данные пересылаются из памяти в CPU и обратно (рисунок 1.6). Скорость обработки зависит от пропускной способности шин данных. CPU должно обеспечивать обширную систему команд для решения различных задач. Поэтому узконаправленные задачи могут обрабатываться с меньшей скоростью, чем на специализированном вычислителе.

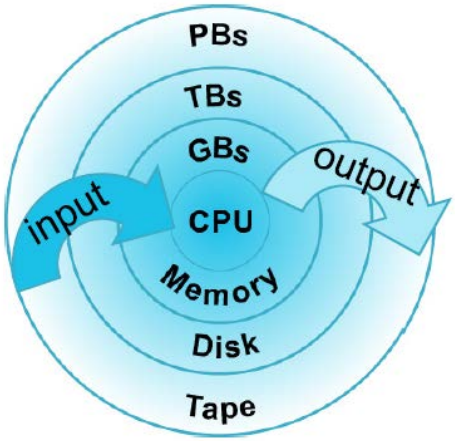

Рисунок 1.6 – Модель с вычислителем в центре системы

Новая модель предлагает расположенное в центре хранилище данных, которое окружают специализированные CPU (рисунок 1.7). Они предназначены для узких задач и могут быть взаимозаменяемы. Данные обрабатываются непосредственно в вычислителе, что исключает пересылку данных. В такой модели можно использовать специализированные структуры памяти, представляющие собой SIMD архитектуру, например, ассоциативную память.

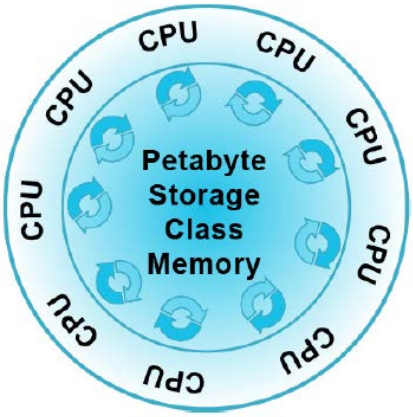

Рисунок 1.7 – Модель с хранилищем данных в центре системы

По классификации Флинна SIMD (single instruction, multiple data – одиночный поток команд, множественный поток данных) – принцип компьютерных вычислений, позволяющий обеспечить параллелизм на уровне данных. SIMD-компьютеры состоят из одного управляющего модуля, называемого контроллером, и нескольких модулей обработки данных, называемых процессорными элементами. На сегодняшний день существуют различные реализации SIMD структур (MRAM, STT-MRAM, ReRAM [5, 6, 7]), вычислительные возможности таких структур имеют теоретическое обоснование, например, в работах Маркова.

1.2.5.FPGA

FPGA расшифровывается как field-programmable gate array (программируемые пользователем вентильные матрицы, ППВМ). Это полупроводниковое устройство, конфигурация которого может быть запрограммирована изготовителем или пользователем. Логика работы микросхемы описывается с помощью языка проектирования (например, VHDL). ППВМ является архитектурной разновидностью программируемых логических интегральных схем (ПЛИС).

На рынке есть два лидера в производстве FPGA-чипов: Intel и Xilinx.. Intel пришла на рынок совсем недавно — в 2015 году, поглотив компанию Altera, которая была основана в то же время, что и Xilinx. Технологии Altera

и Xilinx во многом схожи, как и среды разработки. Первый свой чип FPGA XC2064 основатели Xilinx изобрели в 1985 году. Эта микросхема положила начало новой технологии и новому рынку. Через 20 лет Росс Фримен был занесён в Национальный Зал славы изобретателей США благодаря этому изобретению.

Концепция программируемых вентильных матриц, логических блоков и логических вентилей запатентована Дэвидом Пейджем и Луверном Петерсоном в 1985 году. В 1990-х годах произошёл резкий подъем интереса к ППВМ, вследствие чего увеличился объём производства. В самом начале 1990-х годов ППВМ использовались только в области телекоммуникаций и сетей связи. С усложнением структуры этих устройств они начали применяться в потребительской электронике, в автомобильной промышленности и других отраслях.

Пик популярности ППВМ пришелся на 1997 год, когда Адриан Томпсон объединил генетические алгоритмы и технологию FPGA для создания такого устройства, которое было способно отличить звуковые тоны частотой 1 КГц и 10 КГц.

Логика FPGA может быть перепроектирована практически в любой момент в процессе их использования. В интегральных схемах специального назначения (application-specific integrated circuit, ASIC) используются аналогичные по строению логические матрицы, но они конфигурируются один раз в процессе производства, в то время как FPGA могут постоянно перепрограммироваться и менять топологию соединений в процессе использования. Однако такая гибкость требует значительного увеличения количества транзисторов микросхемы. Матрицы FPGA состоят из конфигурируемых логических блоков, которые похожи на переключатели с множеством входов и одним выходом (логические вентили или gates). В цифровых схемах такие переключатели реализуют базовые двоичные операции AND, OR, NAND, NOR и XOR. Для большинства современных микропроцессоров функции логических блоков фиксированы и не могут

изменяться. Но в FPGA функции блоков и конфигурация соединений между ними могут меняться с помощью посылаемых схеме специальных сигналов.

FPGA включают в себя три программируемых элемента (рисунок 1.8):

- логические блоки (ЛБ или Logic Block);
- блоки ввода-вывода (БВВ или input/output pin);
- внутренние связи (Interconnect resources).

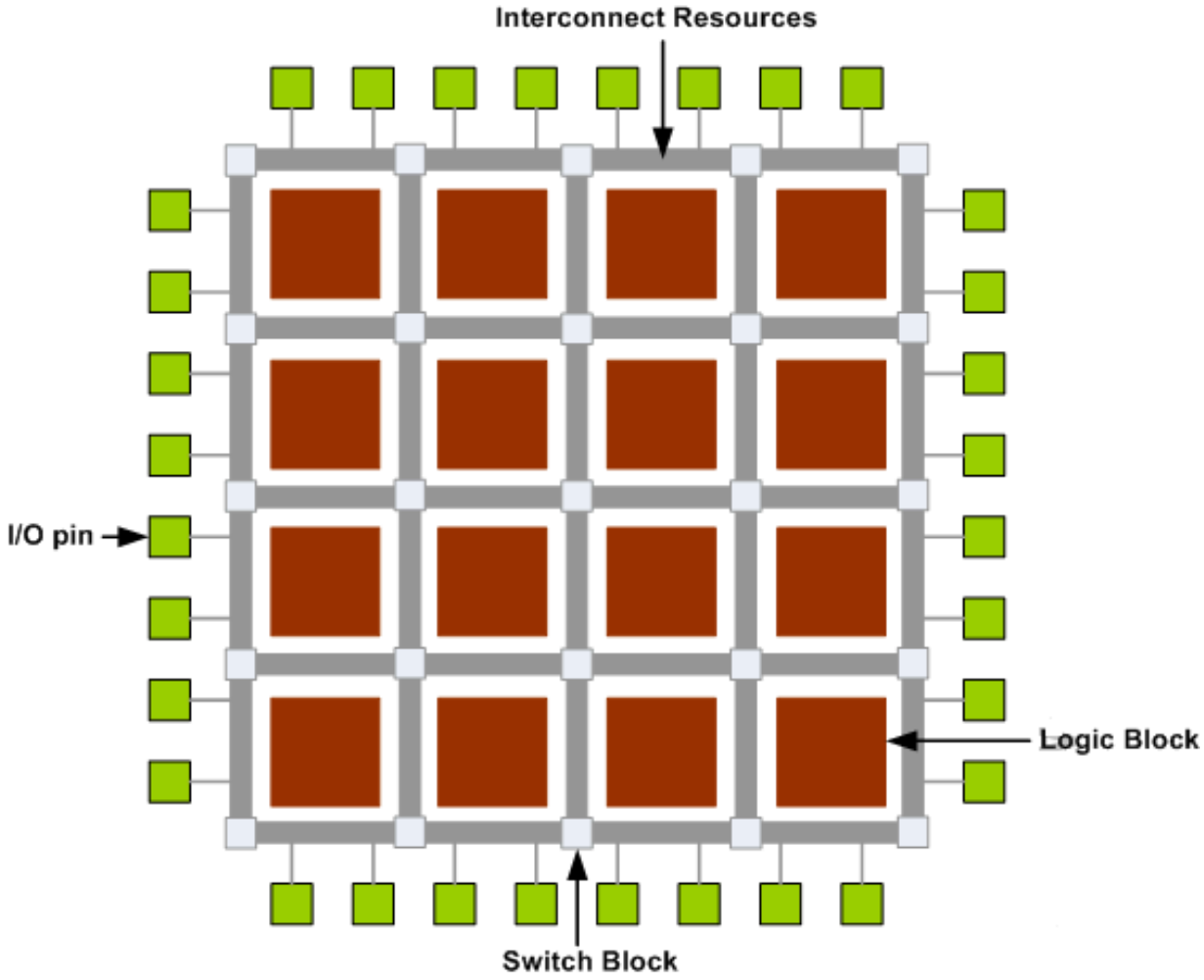

Рисунок 1.8 – Упрощенная структура микросхемы

ЛБ являются функциональными элементами для построения логики пользователя. БВВ обеспечивают связь между контактами корпуса и внутренними сигнальными линиями. Программируемые ресурсы внутренних связей обеспечивают управление путями соединения входов и выходов логических блоков и блоков ввода-вывода на соответствующие сети. Все каналы трассировки имеют одинаковую ширину (одинаковое количество

проводников). Большинство блоков ввода-вывода вписываются либо в одну строку (по высоте), либо в один столбец (по ширине) массива вентилей.

Логический блок классической FPGA состоит из блока, задающего булеву функцию от нескольких аргументов (она называется таблицей соответствия — Look Up Table, LUT) и триггера (flip-flop, FF) на четыре входа, входа синхронизации (clock) и одного выхода. В последние годы производители начали переходить на таблицы истинности с большим числом входов, что позволяет использовать меньшее число логических блоков для типичных приложений, также в новых поколениях используется по два триггера на логический блок (рисунок 1.10).

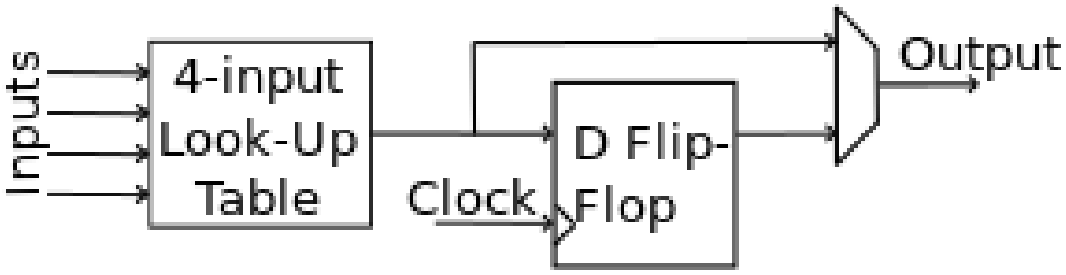

Рисунок 1.9 – Типичный логический блок

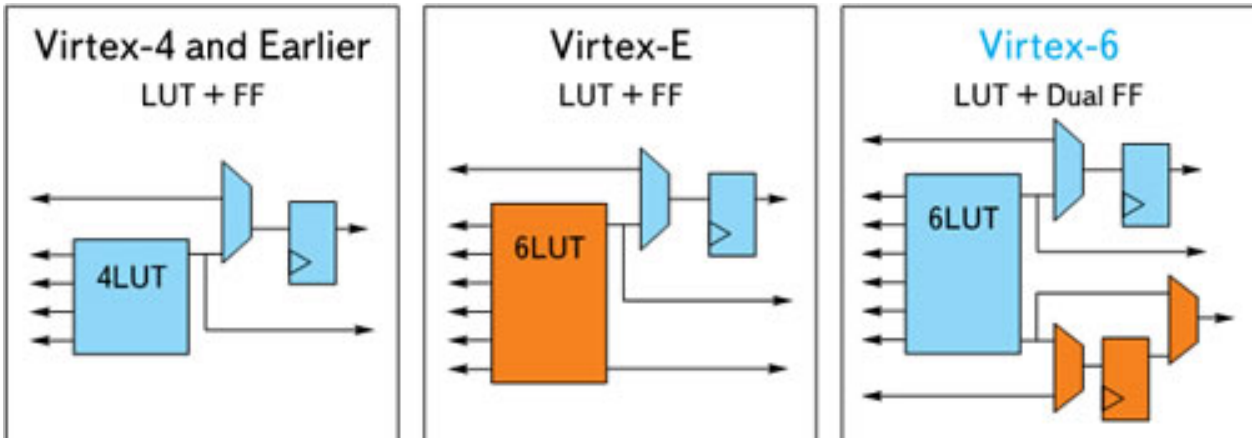

Рисунок 1.10 – Эволюция архитектуры логической ячейки в различных поколениях FPGA Virtex

Выход логического блока только один – регистровая или нерегистровая выходная таблица истинности. Поскольку сигналы синхронизации в коммерческих матрицах FPGA трассируются специальными трассировочными цепями, управление этими сигналами делается отдельно.

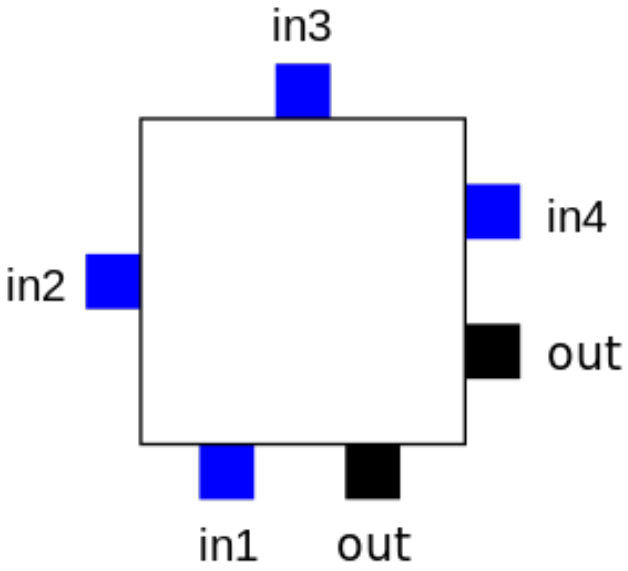

Рисунок 1.11 – Расположение контактов логического блока

Входы расположены на разных сторонах логического блока; выходной контакт может трассироваться в двух каналах: либо справа от блока, либо снизу. Выходные контакты каждого логического блока могут соединяться с трассировочными сегментами в смежных каналах. Аналогично, контактная площадка блока ввода-вывода (pad) может соединяться с трассировочным элементом в любом смежном канале. Например, верхняя контактная площадка кристалла может соединяться с любым из W проводников (где W – ширина канала) в горизонтальном канале, расположенном непосредственно под ним.

Как правило, трассировка FPGA несегментирована, то есть каждый сегмент проводника соединяет только один логический блок с переключательным блоком. Из-за обхода программируемых переключателей в переключательном блоке трассировка получается более длинной. Для увеличения скорости внутрисистемных соединений, в некоторых архитектурах FPGA между логическими блоками используются более длинные трассировочные соединения.

В месте пересечения вертикальных и горизонтальных каналов создаются переключательные блоки. При такой архитектуре для каждого проводника, входящего в переключательный блок, существуют три

программируемых переключателя, которые позволяют ему подключаться к трём другим проводникам в смежных сегментах канала. Модель или топология выключателей, используемая в этой архитектуре, является планарной или доменной топологией переключательных блоков. В этой топологии проводник трассы номер 1 подключается только к проводнику трассы номер 1 в смежных каналах, проводник трассы номер 2 подключается только к проводникам трассы номер 2 и так далее.

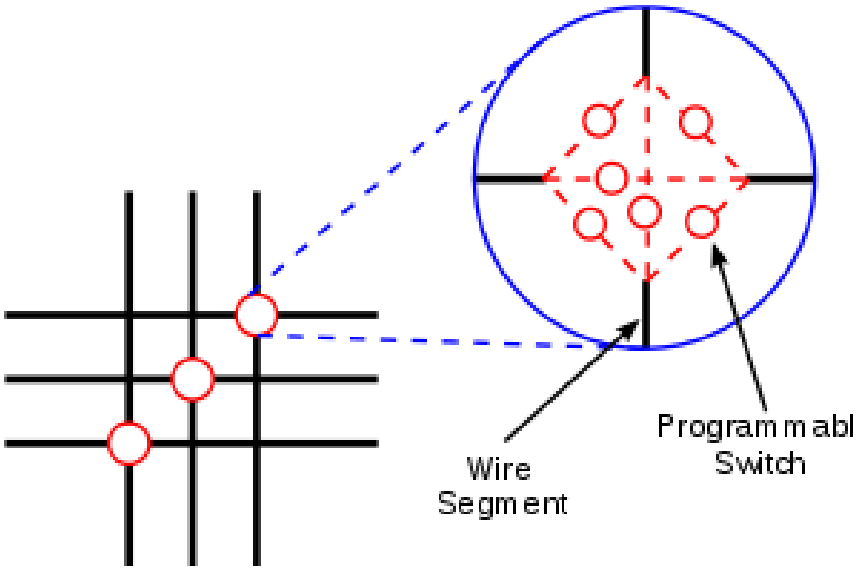

Рисунок 1.12 – Топология переключательного блока

Количество логических блоков в современных кристаллах FPGA может быть различным и зависит от типа и емкости кристалла. У Xilinx есть кристаллы с количеством ЛБ в пределах примерно от четырех тысяч до трех миллионов.

Современные семейства FPGA расширяют описанные выше возможности и включают встроенную функциональность высокого уровня. Благодаря наличию этих общих функции на кристалле кремния можно уменьшить площадь самого кристалла, к тому же при такой схеме эти функции будут работать быстрее, чем при их реализации на основе примитивов. Примерами таких функций являются блоки цифровой обработки сигналов, мультиплексоры, быстрая логика ввода-вывода и встроенная память встроенные процессоры.

FPGA также широко применяются для систем проверки пригодности, в том числе в докремниевой и послекремниевой проверке пригодности, а также при разработке программ для встраиваемых систем. Это позволяет компаниям-производителям интегральных схем проверять работоспособность своих устройств до изготовления их на заводе, сокращая время выхода изделия на рынок.

VHDL (VHSIC (Very high speed integrated circuits) Hardware Description Language) – язык описания аппаратуры интегральных схем. Язык проектирования VHDL является базовым языком при разработке аппаратуры современных вычислительных систем.

Язык был разработан в 1983 г. по заказу Министерства обороны США с целью формального описания логических схем для всех этапов разработки электронных систем, от модулей микросхем до крупных вычислительных систем.

Первоначально язык предназначался для моделирования, но позднее из него было выделено синтезируемое подмножество. Написание модели на синтезируемом подмножестве позволяет автоматический синтез схемы функционально эквивалентной исходной модели. Средствами языка VHDL возможно проектирование на различных уровнях абстракции (поведенческом или алгоритмическом, регистровых передач, структурном), в соответствии с техническим заданием и предпочтениями разработчика. В данный язык заложена возможность иерархического проектирования, максимально реализующая себя в экстремально больших проектах с участием большой группы разработчиков. Представляется возможным выделить следующие три составные части языка: алгоритмическую (основанную на языках Ada и Pascal) – придает языку VHDL свойства языков программирования; проблемно-ориентированную – представляет VHDL как язык описания аппаратуры; и объектно-ориентированную, интенсивно развиваемую в последнее время.

Стандартами закреплены многие его усовершенствования, так например, начиная со стандарта VHDL-2000, язык приобретает основы объектно-ориентированной парадигмы. Стандарт VHDL-93 является последним, полностью поддерживаемым средствами САПР стандартом.

VHDL создан как средство описания цифровых систем, однако существует подмножество языка – VHDL AMS (Analog Mixed Signal), позволяющее описывать как чисто аналоговые, так и смешанные, цифро-аналоговые схемы.

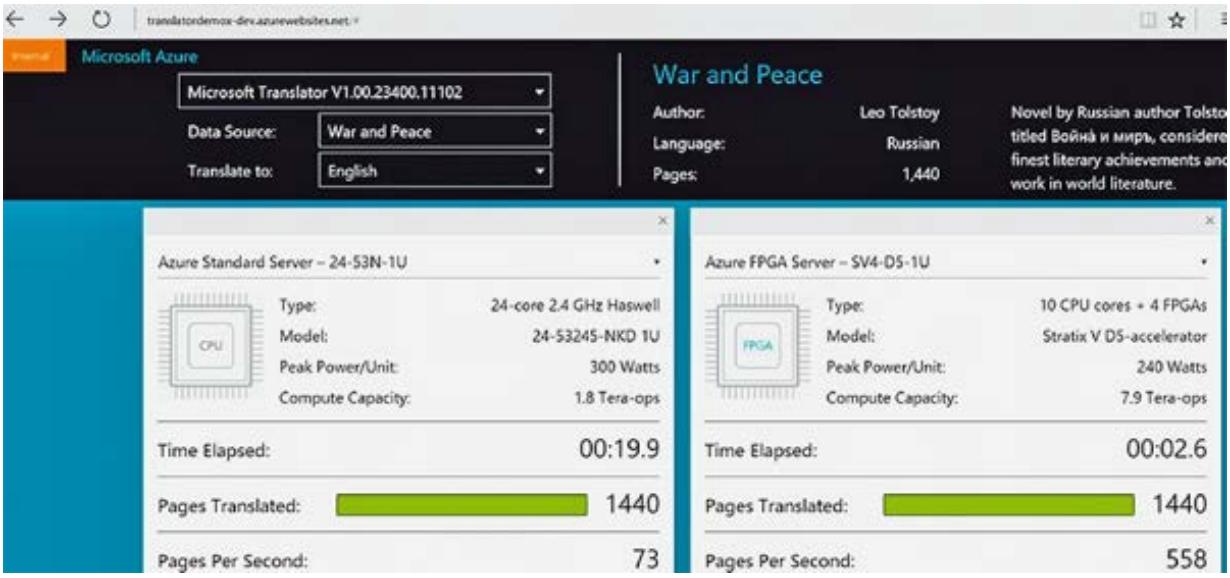

Рисунок 1.13 – Структура стандартного сервера и FPGA сервера

Специалисты Microsoft на конференции Ignite продемонстрировали работу гипермасштабного облака, с помощью которого 5 млн статей «Википедии» (около 3 млрд слов) были переведены с английского языка на испанский меньше чем за 0,1 с. Это стало возможным благодаря разработанным в корпорации программируемым логическим матрицам FPGA, которые использовались в вычислительных узлах облака вместе с обычными процессорами. Преимущество FPGA заключается в том, что такие чипы можно оптимизировать для конкретной нагрузки и менять их конфигурацию по мере совершенствования алгоритмов. В Microsoft провели еще одну демонстрацию, переведя «Войну и мир» с русского на английский сперва с помощью четырехъядерного процессора большой мощности, а затем

с использованием четырех плат, оснащенных FPGA-ускорителями. Во втором случае было израсходовано в 5 раз меньше энергии. Как сообщил глава Microsoft Сатья Наделла, новая вычислительная инфраструктура создана для осуществления проектов корпорации в области искусственного интеллекта, в том числе по развитию помощника Cortana и платформы ботов.

Инфраструктура построена на базе ускорителей Stratix V D5 компании Altera, которую корпорация Intel купила в 2015 году за 16,7 млрд долл.

1.2.6.ИИ-ПРОЦЕССОРЫ ASCEND

Huawei анонсировала первые в мире процессоры, оптимизированные под устройства на основе искусственного интеллекта. Эти процессоры технологически реализуют те же идеи, что когда-то закладывались при разработке отечественной платформы «Эльбрус» — центральный процессор в сочетании со вспомогательными чипами для оптимального решения узкоспециализированных задач, благодаря принципу разделения функциональных задач. В структуре был не только центральный суперскалярный процессор, но и добавлялись различные специализированные сопроцессоры, которые выполняли, например, разложение в ряды Фурье или какие-то другие преобразования, разгружая центральный процессор. И в Huawei пошли по аналогичному пути, перекладывая решение отдельных задач на специализированные чипы. С точки зрения разработчика это дает большие возможности для более эффективной оптимизации нагрузки.

В Huawei очень внимательно следят за технологиями в области искусственного интеллекта и могут констатировать, что их развитие сейчас тормозит именно нехватка специализированных процессорных мощностей. Из-за этого факта для решения задач ИИ, когда требуется, например, моментальное распознавание объектов автопилотами на транспорте, приходится упрощать, оптимизировать математические модели. [4]

1.2.7. ВЫВОДЫ

В 1 главе данной работы рассмотрено современное состояние вычислительных систем, реализованных на основе архитектуре фон Неймана. В то же время есть достаточно много предложений по применению структур не фон Неймановского типа (например, графические и векторные процессоры), которые могут быть использованы для существенного увеличения производительности при решении задач определенного класса.

Использование энергонезависимой памяти (рисунок 1.14) позволит исключить несколько уровней памяти сегодняшней иерархии, что позволит уйти от перемещения данных, ускорит выполнение вычислительных задач и снизит энергозатраты.

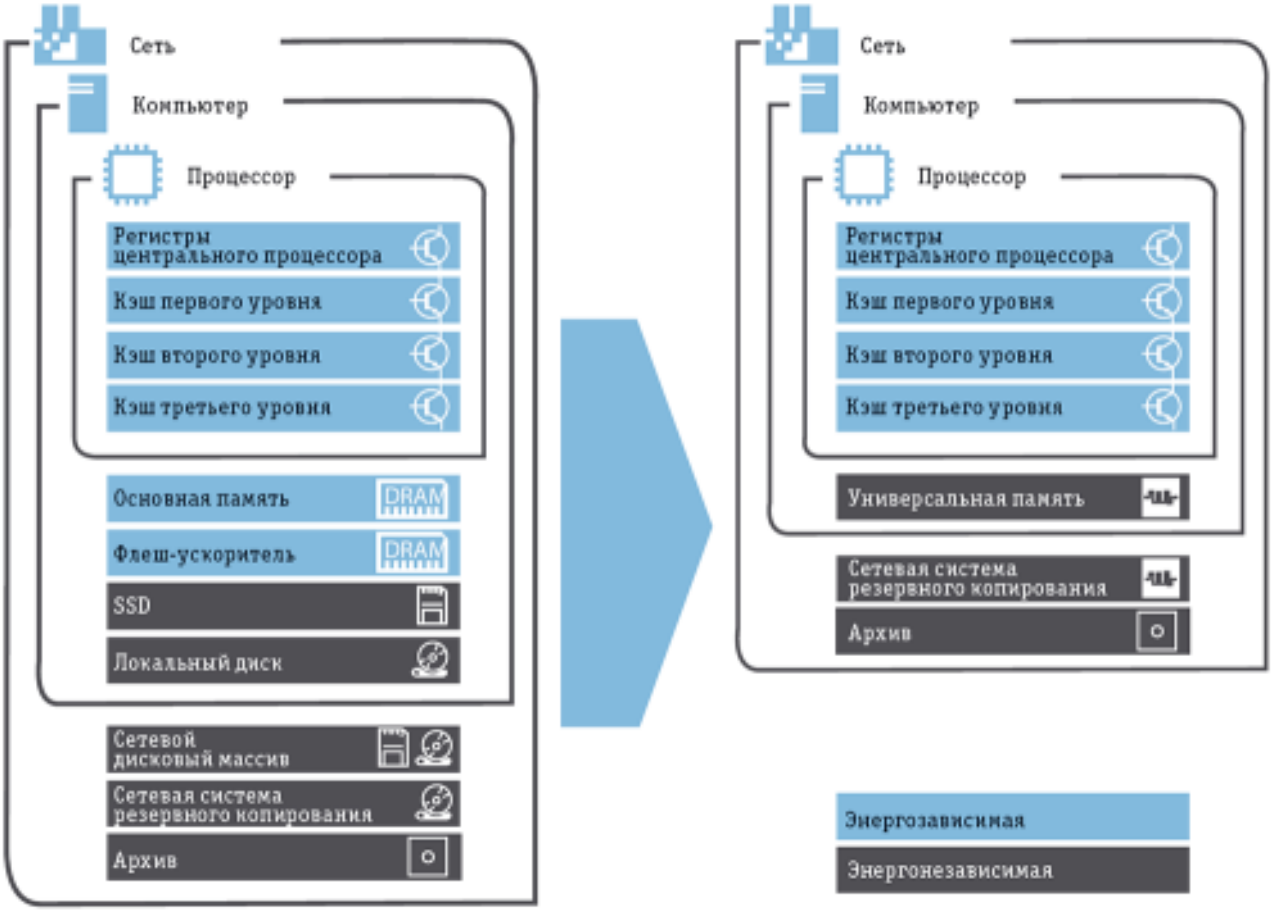

Рисунок 1.14 – Иерархия памяти

Также перспективным решением является использование FPGA. Далее описаны результаты сравнительного анализа CPU, GPU, FPGA и ASIC. CPU

универсален, на нем можно запустить любой алгоритм, он наиболее гибок, и использовать его легче всего благодаря огромному количеству языков программирования и сред разработки.

При этом из-за универсальности и последовательного выполнения инструкций CPU снижается производительность и повышается энергопотребление схемы. Происходит это потому, что на каждую полезную арифметическую операцию CPU совершает много дополнительных операций, связанных с чтением инструкций, перемещением данных между регистрами и кэшем.

Интегральные схемы специального назначения. На этой платформе требуемый алгоритм реализуется аппаратно за счет прямого соединения транзисторов, все операции связаны только с выполнением алгоритма и нет никакой возможности изменить его. Отсюда максимальная производительность и наименьшее энергопотребление платформы. Но недостаток заключается в невозможности перепрограммирования ASIC.

Изначально микросхемы GPU были разработаны для обработки графики, но сейчас используются и для вычислений общего назначения. Если алгоритм можно распараллелить, то на GPU получится добиться значительного ускорения по сравнению с CPU. С другой стороны, последовательные алгоритмы будут реализовываться хуже, поэтому платформа оказывается менее гибкой, чем CPU.

Наконец, FPGA. Эта платформа сочетает эффективность ASIC с возможностью менять программу. ПЛИС не универсальны, но существует класс алгоритмов и задач, которые на них будут показывать лучшую производительность, чем на CPU и даже GPU.

Решающее же преимущество FPGA – это способность обрабатывать данные в темпе их поступления с минимальной задержкой реакции. Таким образом, FPGA используются там, где нужна высокая производительность обработки данных, наименьшее время реакции, а также низкое энергопотребление. Но недостатком такого решения является

дополнительные временные затраты на изучение языка VHDL и программирование самой структуры FPGA.

Далее будет рассмотрена еще одно устройство с SIMD архитектурой – ассоциативная память, которая позволяет выполнять вычисления, не перемещая данные в обрабатывающий блок и обратно, также производить обработку не отдельного слова, а разрядного слайса всего массива данных.

2. ФУНКЦИОНАЛЬНЫЙ АНАЛИЗ АССОЦИАТИВНОЙ ПАМЯТИ

Существует множество задач, требующих в меньшей степени выполнение арифметических вычислений и в большей степени логических. От самого простого – поиск экстремума для заданного набора значений, до сложных комплексных задач, требующих декомпозиции. Быстродействие играет большую роль, когда речь идет о больших объемах информации. Рассмотренные в подразделе 1.2 системы не могут обеспечить требуемую скорость обработки. Поэтому необходимо пересмотреть концепцию работы этих систем.

Рассмотрим, каким образом отображается постановка решения задачи на машины с архитектурой фон Неймана.

Существует множество задач . Для решения отдельной задачи существует множество алгоритмов .

(4)

где – алгоритм для решения *i*-й задачи.

Каждый алгоритм представляет собой набор пар преобразований и проверок условий.

, (5)

где – преобразование,

– проверка условия.

При корректном описании алгоритма решение задачи может быть запрограммировано. Пары преобразования и проверки условия из формулы 5 трансформируются в операторы языка высокого уровня. Операторы языка высокого уровня отображаются на множество команд конкретного процессора.

,

(6)

,

где – множество команд конкретного процессора, шт.

Например, для процессоров Intel 80386 инструкций.

Алгоритм в процессе программирования привязывается к определенному набору команд в рамках конкретной системы команд процессора. И все преобразования, которые требуются в алгоритме, реализуются средствами этой системы команд, что может быть не всегда оптимально при решении задач определенного класса.

Структура машинных инструкций микропроцессов серии Intel 80x86 в общем случае представлена на рисунке 2.1.

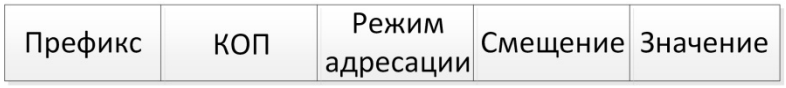

Рисунок 2.1 – Структура машинных инструкций микропроцессов серии Intel 80x86

Префикс – особый однобайтовый код, уточняющий поведение команды. В начале каждой команды может располагаться до четырех префиксов. Если нет ни одного префикса, то команда ведет себя "по умолчанию".

КОП – поле команды, определяющее операцию, которая должна быть выполнена по данной команде. Может занимать от одного до двух байт.

Режим адресации – определяет форму адресации и специфицирует используемые регистры.

Смещение – значение эффективного адреса операнда.

Выполнение команд можно представить следующими этапами:

- чтение команды в регистр команд из ячейки памяти с адресом, указанным в счетчике команд;
- декодирование кода операции;
- формирование адресов операндов;
- чтение операндов;

- исполнение операции в АЛУ;
- вывод результата.

Между этими этапами есть вспомогательные операции, которые связаны с обменом данных между памятью и вычислителем. Только операции декодирования команды и ее выполнения можно отнести к этапам непосредственной трансформации команды. Все остальные этапы являются обслуживающими.

Затраты на время передачи данных между вычислителем и памятью, ограниченная система команд, конвейерный принцип работы традиционных процессоров, избыток обслуживающих операций не позволяют в полной мере обеспечить необходимый уровень быстродействия.

В главе 1 настоящей работы показано, что существующие классические решения проблемы быстродействия обладают рядом несовершенств. Исходя из этого, появляется задача по исследованию такой архитектуры, как ассоциативная память.

Отличительная особенность ассоциативных запоминающих устройств заключается в том, что они способны выдавать хранящуюся в них информацию без предварительного указания адресов тех ячеек, в которых находятся данные. Для поиска информации используется некоторая совокупность свойств или описаний объекта. Действия в АП представляют собой функцию от значения данных .

Ассоциативная память может быть организована как программным, так и аппаратным способом. При программной реализации понятие АП используется в основном как модель взаимодействия процессора с хранилищем данных. Например, в реляционных базах данных (БД) для ускорения поиска требуемой информации широко используются ключевые поля, входящие в состав каждой записи БД. Для быстрого поиска по ключам используют специальные индексные файлы, построенные, например, по принципу двоичных деревьев. Адресной информацией в таком случае

является не номер записи, а содержимое полей. Например, поля код товара, или фамилия человека. Индексные файлы позволяют укорить процедуру поиска.

При аппаратной реализации АП используется принцип обработки по вертикальным слайсам. Вертикальный слайс – это вертикальный срез всех одноименных разрядов массива.

На рисунке 2.2 представлена традиционная обработка. Память представляет собой массив из разрядных элементов. Для нахождения нужного элемента в массиве ячеек слова просматриваются по адресам. Такая структура справедлива для устройств с архитектурой фон Неймана. Время обработки пропорционально количеству слов в массиве.

, (7)

где – время обработки;

– коэффициент пропорциональности;

– количество элементов в массиве.

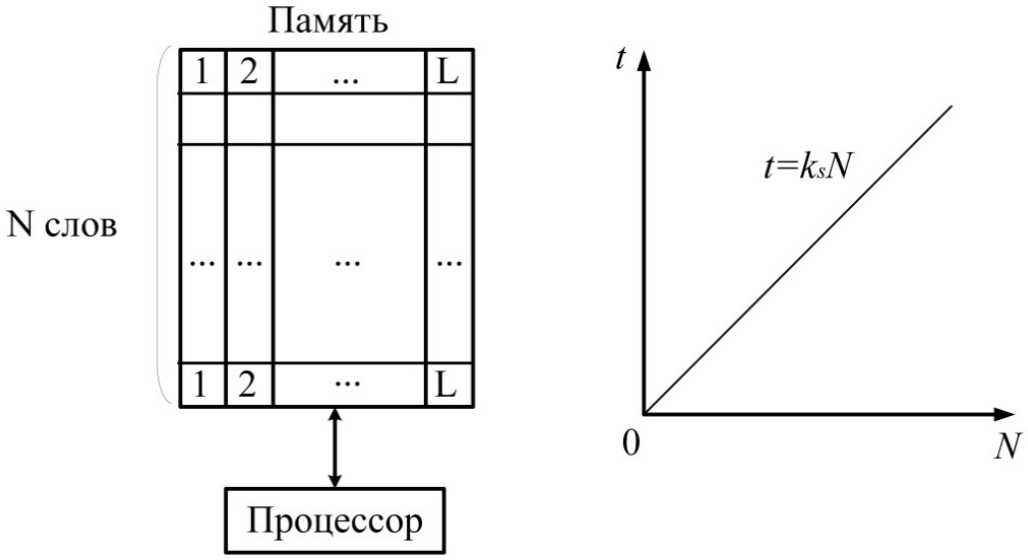

Рисунок 2.2 – Традиционная обработка данных

Зачастую сравнение всех разрядов массива не оптимально для большинства преобразований. В таких случаях можно использовать послайсную обработку (рисунок 2.3), в которой все слова просматриваются

одновременно. Время обработки пропорционально разрядности элементов в массиве.

, (8)

где – время обработки;

– коэффициент пропорциональности;

– разрядность элементов в массиве.

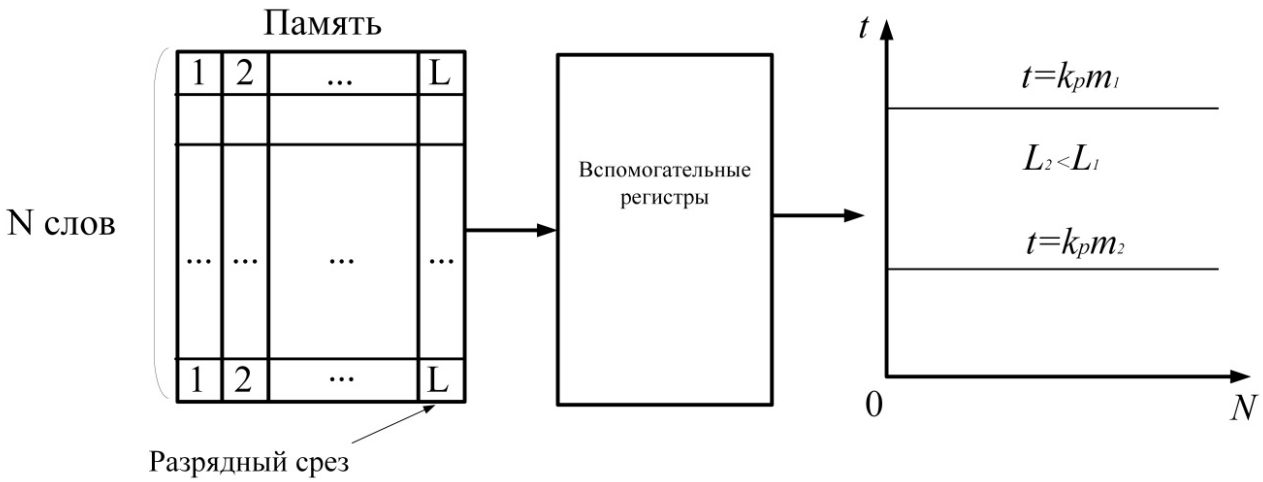

Рисунок 2.3 – Послайсная обработка данных в АП

Представление ассоциативных решающих полей (АРП)

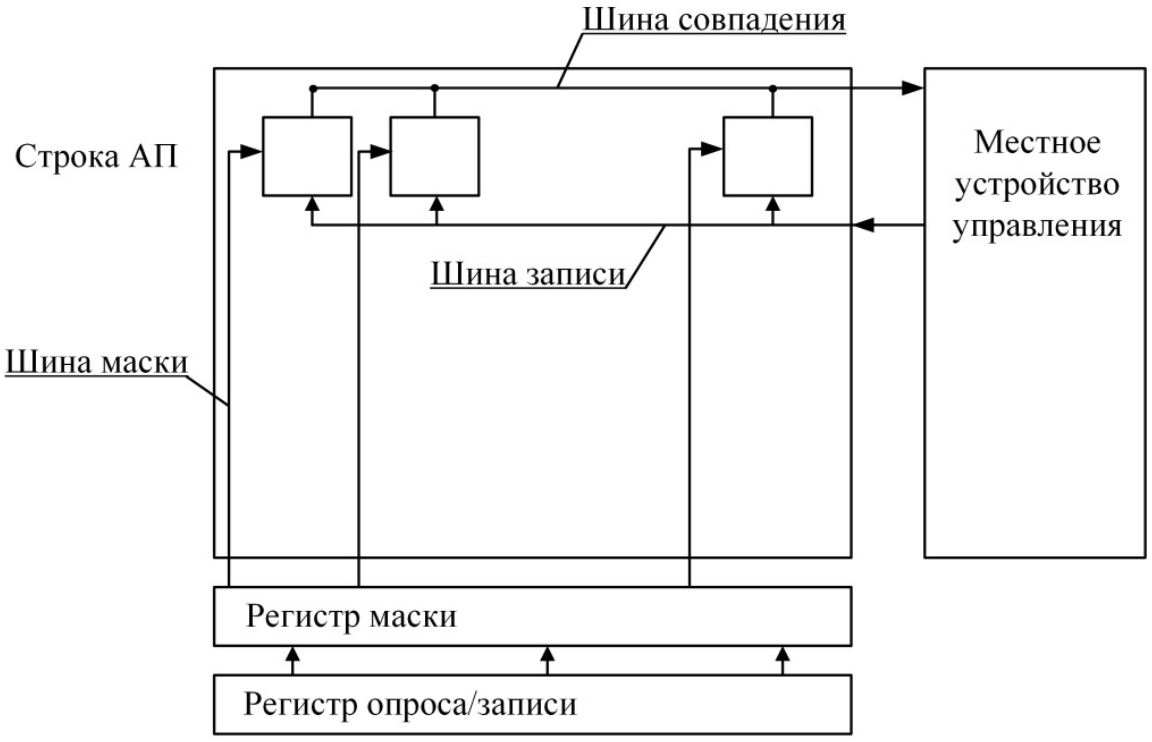

Рисунок 2.4 – Структура АРП

В настоящее время, появилось большое количество новых твердотельных элементов хранения информации (Non-Volatile Memory, NVM). Память на этих элементах, с одной стороны, начинает заменять HDD, превосходя последние по времени доступа. С другой, в обозримом будущем, превзойдет их по объемам и снижению стоимости хранения информации. Появление таких элементов и структур на них позволяет вернуться к идеям распределенных систем информационных преобразований непосредственно в памяти.

2.1. РАСПРЕДЕЛЕННЫЕ СИСТЕМЫ

MRAM (Magnetoresistive Random-Access Memory) – вид памяти, основанный на магнитных запоминающих элементах. Каждая ячейка хранения построена на базе магнитного туннельного перехода (MTJ – Magnetic Tunnel Junction). Сама MTJ-ячейка представляет собой структуру из нескольких слоев: слоя с фиксированной поляризацией, тонкого диэлектрического туннельного барьера и свободного магнитного слоя. Когда на такой MTJ-элемент подается смещение, электроны поляризуются магнитным слоем и пересекают диэлектрический барьер за счет процесса, известного как туннелирование. Направление ориентации намагниченности в слоях может либо совпадать, либо быть противоположным друг другу. Таким образом, при совпадении ориентации намагниченности, вследствие эффекта туннельного магнитосопротивления, электрическое сопротивление уменьшается, и это принимается за логический «0», а при противоположном направлении намагниченности в слоях сопротивление возрастает, и это интерпретируется как «1». Такие изменения сопротивления, вызванные изменением магнитного состояния устройства, известны как магниторезистивный эффект. Чтобы узнать, какое значение записано в ячейку, необходимо приложить напряжение к транзистору и зафиксировать уровень тока через ячейку.

В отличие от большинства других технологий изготовления памяти в MRAM данные хранятся как магнитные состояния вместо электрических

зарядов и считываются путем измерения сопротивления ячейки, без влияния на ее магнитное состояние. Использование магнитных состояний для хранения информации имеет два главных преимущества. Во-первых, магнитная поляризация не имеет утечек со временем, как это происходит с зарядом, таким образом, информация на запоминающей ячейке остается даже после отключения напряжения питания. Во-вторых, переключение магнитной поляризации между двумя состояниями не влияет на движение электронов или атомов, и, таким образом, нет механизма износа запоминающих ячеек.

Также есть недостатки технологии MRAM – для магнитного поля требуется довольно много энергии, что нежелательно, особенно в случае применения таких микросхем в мобильных устройствах, где к энергоэффективности предъявляются особые требования.

Другой недостаток – индуцируемое магнитное поле при уменьшении размера ячеек начинает воздействовать на соседние ячейки, повышая риск искажения информации в них.

Чтобы получить высокую плотность памяти, ячейки MRAM, показанные на рисунке 2.5, упорядочены в матрицу, в которой каждая шина записи перекрывает сотни или тысячи ячеек памяти, как показано на рисунке 2.6. Во время выполнения операции записи импульс тока проходит через цифровую и битовую шины, изменяя значения только тех ячеек памяти, которые находятся на пересечении этих шин. Во время выполнения операции чтения изолирующий транзистор целевого элемента хранения данных открывается, создавая ток смещения MTJ, и полученный ток сравнивается с опорным током, чтобы определить состояние с высоким сопротивлением или с низким.

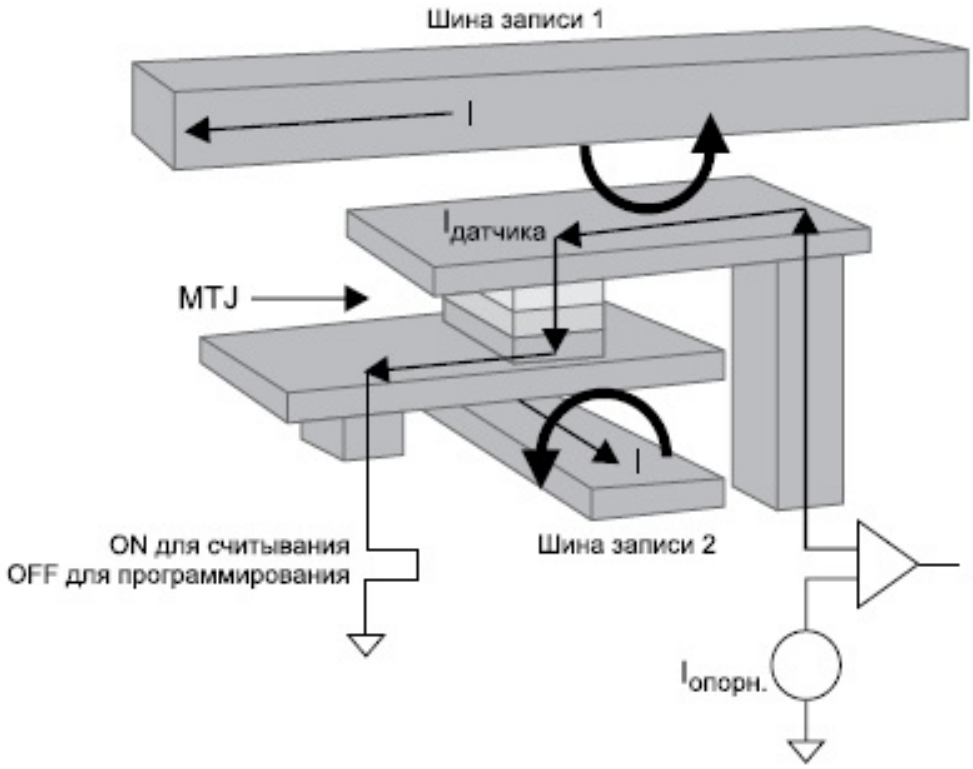

Рисунок 2.5 – Ячейка памяти MRAM

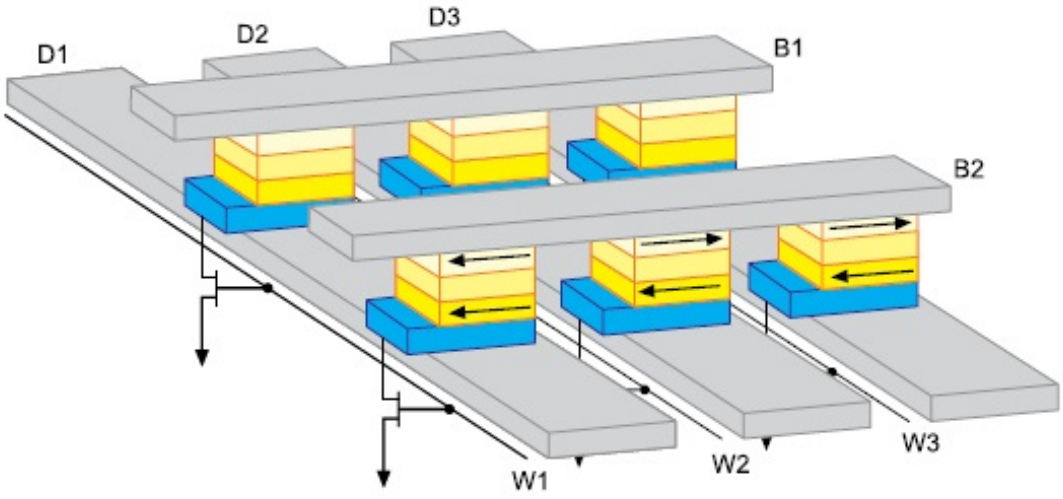

Рисунок 2.6 – Массив ячеек MRAM с шинами записи

Патентно-информационный поиск позволил провести анализ существующих отечественных и зарубежных разработок ассоциативной памяти. Среди отечественных разработок не найдены инновационные решения. Но среди зарубежных исследований можно остановиться на нескольких интересных решениях.

2.1.1.МАГНИТОРЕЗИСТИВНАЯ ПАМЯТЬ С ДВОЙНЫМ ПЕРЕХОДОМ

Зарубежная компания CrocusTechnology является одним из ключевых разработчиков MRAM-памяти в мире. В мае 2011 года РОСНАНО и CrocusTechnology договорились о создании в России производства памяти MRAM. Для защиты производственных мощностей были выкуплены патенты у NXP Semiconductors на технологии магниторезистивной памяти. Купленные у нидерландского производителя полупроводников NXP патенты описывают полный спектр технологий создания памяти MRAM – от проектирования и производства до внедрения в конкретные устройства.

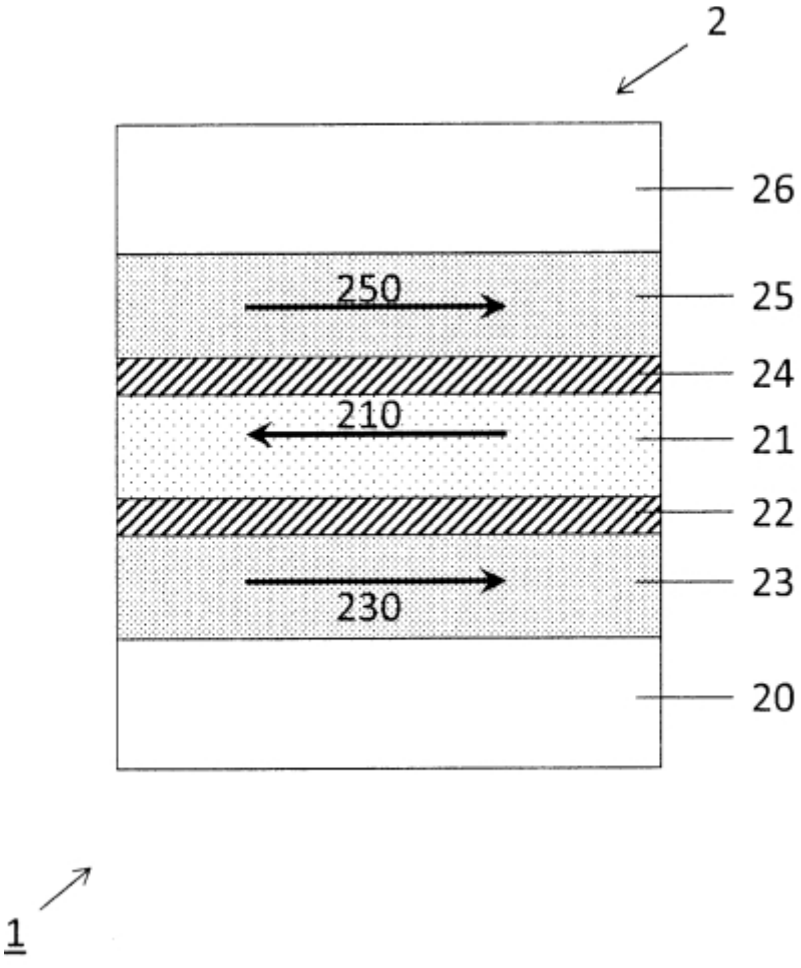

Рисунок 2.7 – Ячейка магнитной оперативной памяти

На рисунке 2.7 изображена ячейка магнитной оперативной памяти включающей в себя первый слой хранения, второй слой хранения и слой считывания.

Сущность изобретения заключается в том, что ячейка магнитной оперативной памяти (MRAM) включает в себя первый туннельный барьерный слой, заключенный между слоем мягкого ферромагнетика, имеющим свободную намагниченность и первым слоем твердого ферромагнетика, имеющим первую намагниченность хранения; второй туннельный барьерный слой, заключенный между слоем мягкого ферромагнетика и вторым слоем твердого ферромагнетика, имеющим вторую намагниченность хранения; причем первая намагниченность хранения является свободно ориентируемой при первом высоком заранее определенном температурном пороге, и вторая намагниченность хранения является свободно ориентируемой при втором заранее определенном высоком температурном пороге; при этом первый высокий заранее определенный температурный порог выше второго заранее определенного высокого температурного порога. Технический результат: обеспечение возможности использования ячейки MRAM в качестве троичной ассоциативной памяти (TCAM) при уменьшенном размере ячейки [5].

2.1.2.МАГНИТОРЕЗИСТИВНАЯ ПАМЯТЬ С ПЕРЕНОСОМ СПИНОВОГО МОМЕНТА

От недостатков магниторезистивной памяти позволяет избавиться технология STT-MRAM – модификация описанной выше MRAM, при которой изменение поляризации намагниченного слоя выполняется не за счет формирования магнитного поля, а при помощи переноса момента импульса электрона (spin) с заданным направлением поляризации. Вращающий момент этих электронов, попадающих в изменяемый ферромагнитный слой, передается намагниченности и ориентирует ее в заданном направлении.

Отсюда и название этого варианта технологии – STT-MRAM (spin-torque-transfer MRAM).

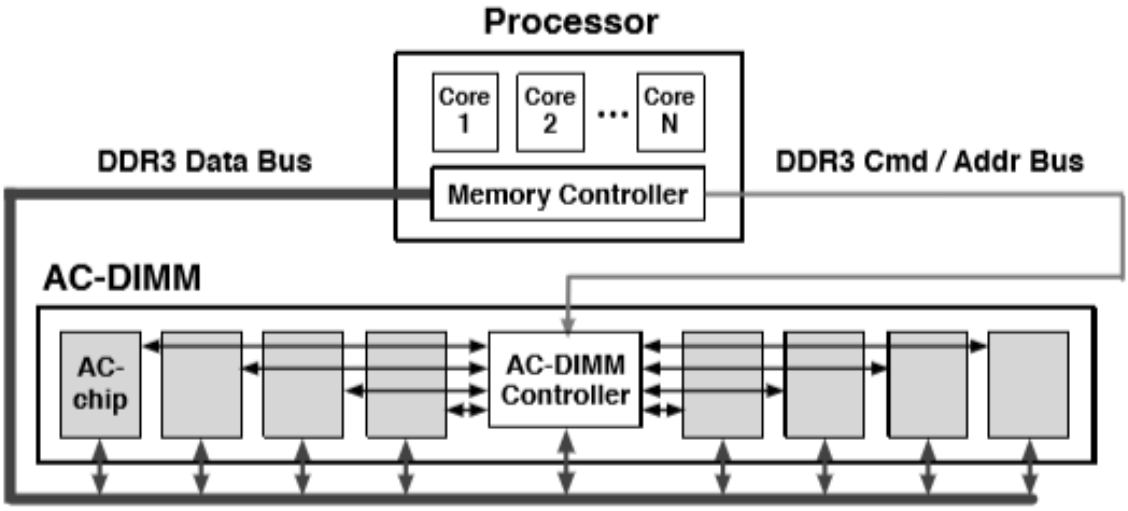

Рисунок 2.8 – вычислительная система с AC-DIMM

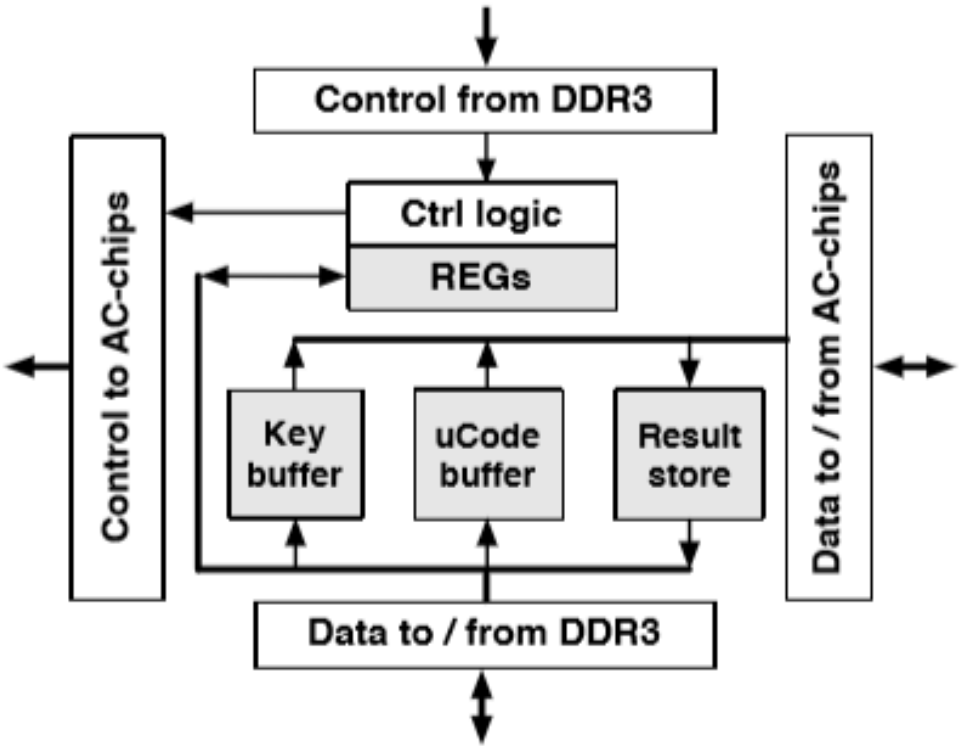

Рисунок 2.9 – контроллер на базе DIMM

AC-DIMM (Associative Computing with STT-MRAM) устраняет ограниченную гибкость резистивного TCAM ускорителя за счет комбинирования двух возможностей: ассоциативный поиск и вычисления в памяти. Улучшение за счет расширения TCAM систем с наборами

интеграторов, программируемых пользовательских микроконтроллеров, которые оперируют непосредственно на результатах поиска, и за счет архитектурного решения: пара ключ-значение могут быть расположены в той же строке TCAM. Предлагаются новые двоичные последовательности массивов TCAM, которые позволяют реализовать систему, используя STT-MRAM. AC-DIMM достигает 4.2x ускорения и 6.5x сокращения энергии по сравнению с системой, основанной на RAM [6].

2.1.3.РЕЗИСТИВНЫЙ АССОЦИАТИВНЫЙ ПРОЦЕССОР

Резистивный ассоциативный процессор (Resistive Associative Processor, ReAP) – массивно-параллельный ускоритель SIMD матриц. Ассоциативный процессор содержит массив адресно-ориентированной резистивной памяти (CAM) и периферийные схемы, позволяет хранить и обрабатывать данные в одном и том же месте.

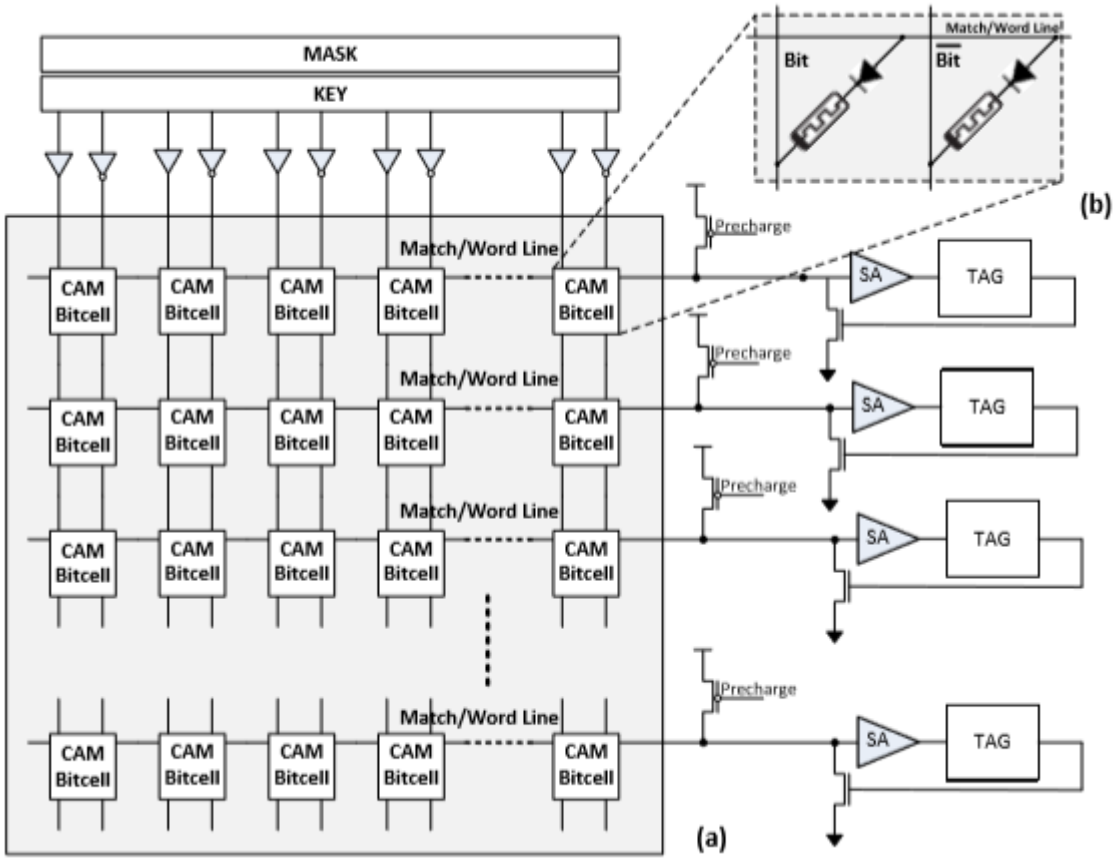

Рисунок 2.10 – Резистивный ассоциативный процессор

Время выполнения векторной операции в AP не зависит от размера вектора, что обеспечивает эффективную параллельную обработку очень больших векторов. Обеспечивает лучшую энергоэффективность, чем обычные параллельные ускорители. Технология резистивной памяти потенциально позволяет масштабировать точку доступа от нескольких миллионов до нескольких сотен миллионов процессорных блоков на одном кремниевом кристалле.

Резистивные запоминающие устройства хранят информацию, модулируя сопротивление наноразмерных запоминающих элементов (мемристоров). Мемристоры являются двухполюсными устройствами, в которых сопротивление устройства изменяется в зависимости от электрического тока или напряжения. Сопротивление мемристора ограничено минимальным сопротивлением (состояние низкого сопротивления, логическая «1») и максимальным сопротивлением (состояние высокого сопротивления, логический «0»).

В [7] сравнивается производительность и энергопотребление резистивных ассоциативных процессоров с показателями традиционной CMOS AP и обычного SIMD-ускорителя (GPU). Авторы пришли к выводу, что, хотя высокая плотность мощности и ограниченная долговечность мемристоров ограничивают потенциал ReAP, он обеспечивает гораздо лучшую масштабируемость и более высокую производительность по сравнению с CMOS AP и обычными ускорителями SIMD. Дальнейший прогресс в разработке новых материалов для мемристоров обеспечит непрерывную масштабируемость, улучшенную энергоэффективность и более высокую долговечность для ReAP.

2.2. СТРУКТУРА И ПРИНЦИП РАБОТЫ АССОЦИАТИВНЫХ РЕШАЮЩИХ ПОЛЕЙ

Ассоциативные решающие поля в общем случае представляют собой структуру из массива данных, регистра опроса/записи, регистра маски и регистра совпадений.

Массив данных хранит подлежащую обработке информацию. Строки каждой таблицы нумеруются сверху вниз, а столбцы – слева направо. Каждая единица данных хранится в отдельной строке и обрабатывается отдельным процессорным элементом. Массив данных определяется следующим выражением.

,

где – элемент массива;

номер блока;

– индекс строки;

– высота АРП;

– индекс столбца;

– ширина АРП (длина строки).

Служебный регистр опроса/записи предназначен для хранения искомого слова или слова для записи в массив данных (формула 9).

, (9)

где – разряд регистра опроса.

В регистре маски задается маска поиска.

, (10)

где – элемент регистра маски.

Регистр совпадений хранит результат опроса по массиву данных на наличие совпадений с регистром опроса/записи по заданной маске (формула 11).

, (11)

где – разряд регистра совпадений;

Регистр совпадений входит в блок местного устройства управления и выполняет 3 функции:

- обнуление массива;
- признак записи в строку (флаг разрешения записи);
- признак совпадения.

Рассмотрим особенности работы и устройства АРП на примере подзадачи поиска экстремума в массиве – сравнения элементов массива с константой.

Каждая строка массива представляет собой двоичный код из разрядов. В регистр опроса заносится искомый код. В регистре маски единичные значения ставятся в тех разрядах, для которых в соответствующих разрядах регистра опроса/записи будет производиться поиск по массиву данных. Таким образом, значение искомого разряда определяется выражением (12).

, (12)

где – разряд регистра опроса/записи;

– разряд регистра маски;

– индекс столбца.

Примем указание на ячейку обращением к её содержимому. Тогда функция опроса ячейки будет определяться выражением (13).

, (13)

где – элемент массива.

Если для каждой ячейки одной строки тождество будет истинным, тогда найден код, равный значению в регистре опроса. Результат опроса определяется выражением (14).

(14)

Результат функции опроса для каждой строки заносится в соответствующий разряд регистра совпадений . Чтобы проверить результат поиска на наличие совпадений, необходимо использовать функцию совпадений (формула 15).

(15)

2.3. ЗАГРУЗКА ДАННЫХ В АССОЦИАТИВНУЮ ПАМЯТЬ

Загрузка данных в ассоциативную память осуществляется двумя способами: по словам, соответственно расположению слов в оперативной памяти, и по слайсам. В первом случае слова загружаются последовательно с помощью регистра опроса/записи. Во втором, загрузка одного разряда происходит из ортогонального буфера параллельно для всех слов с последующим сдвигом разряда.

2.3.1.ПОСЛЕДОВАТЕЛЬНАЯ ЗАГРУЗКА ДАННЫХ

Структура АРП для последовательной загрузки данных представлена на рисунке 2.4. Адрес строки для загрузки определяется с помощью дешифратора, либо единицей в регистре совпадений, которая сдвигается по мере заполнения массива.

В регистр опроса/записи загружается первое значение. Условие записи в массив описывается выражением (16).

, (16)

где – разряд регистра маски;

– индекс слайса;

– разряд регистра совпадения;

– строка, куда будет производиться запись.

После загрузки данных в регистре совпадения сдвигается единица, тем самым меня строку для загрузки, в регистр опроса/записи загружается новое значение.

2.3.2.ПОСЛАЙСНАЯ ЗАГРУЗКА ДАННЫХ

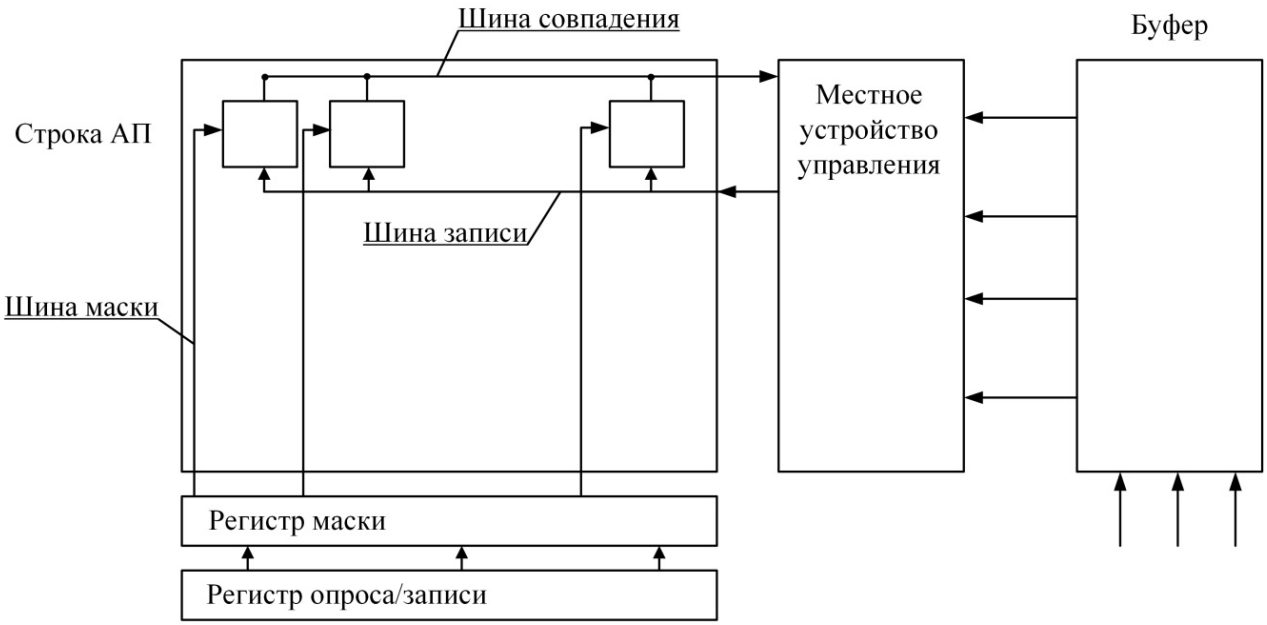

Рисунок 2.11 – Структура при послайсной загрузке данных

Данные загружаются в ортогональный буфер из оперативной памяти. Массив предварительно обнуляется. является разрешающим сигналом записи. Если условие записи, описываемое выражением (17), выполняется, то производится запись соответствующего значения в разряд всех разрешенных строк накопителя. После записи меняется разряд в массиве данных посредством сдвига маски, и сдвигаются строки в буфере на 1 разряд.

, (17)

где – разряд регистра маски;

– номер разряда, для которого будет производиться запись;

– разряд регистра опроса/записи;

– первый разряд *n*-ой строки буфера данных.

2.4 АНАЛИЗ ВОЗМОЖНОСТИ ВЫПОЛНЕНИЯ В АРП ФРАГМЕНТА ПРОГРАММЫ

Представлено описание подпрограммы pop(uint_t _Tst)

```
inline iint_t
  pop( uint_t _Tst )
    {//get amount of 1-bits in a word
    register uint_t
      _Val =  _Tst -((_Tst >> 0x01) & 0x55555555);
      _Val = (_Val & 0x33333333)    +
                ((_Val >> 0x02) & 0x33333333);
      _Val = (_Val + (_Val >> 0x04))& 0x0f0f0f0f;
      _Val =  _Val + (_Val >> 0x08);
    //quick result
    return    (_Val + (_Val >> 0x10))& 0x0000003f;
    }
```

В представленном выше фрагменте выполняются операции сдвига (>>), поразрядной конъюнкции (&), сложения и вычитания над 32-разрядными кодами. Всего 15 операций, из них 5 арифметических, 5 сдвигов, 5 логических. Если каждая операция выполняется за 1 такт, то для массива N переменных потребуется количество тактов .

Оценим количество тактов, если выполнить эти операции на ассоциативных решающих полях.

Операции сложения и вычитания, как показано ранее, без ускорения требуют тактов. Для 32 разрядов:

тактов.

Операция сдвига потребует незначительных затрат, поскольку ее можно выполнить на том же массиве, сдвигая не массив, а маску операции

относительно массива. Если регистры данных и маски дополнить кэш-памятью констант (или вместо регистров поставить кэш данных и констант), то можно выполнять сдвиг маски и установку данных за 1 такт.

Также реализуется в АРП операция поразрядной конъюнкции с константой. Поскольку , можно просто записать нулевые значения одновременно в соответствующие разрядные слайсы (столбцы) сразу всего массива, то есть потребуется 1 такт.

Если в АРП повторить последовательность операции вычислений фрагмента, то число тактов будет равняться

.

Но это при условии, что все данные уже загружены в АРП. Если учесть загрузку данных до выполнения функции pop() и выгрузку после ее вычисления, то .

В таблице 2 представлена зависимость количества тактов от различного количества элементов массива *N* и условиях загрузки.

Таблица 2 – Зависимость тактов от количества элементов массива

N	Такты обычного процессора	Такты АП без учета загрузки	Такты АП с учетом загрузки	Выигрыш без/с учетом загрузки в n раз
100	1 500	1 585	1 785	0,95/0,84
1000	15 000	1 585	3585	9,50/4,20
10 000	150 000	1 585	21585	950,00/6,95
100 000	1 500 000	1 585	201585	9500,00/7,44

Итоговая таблица показывает, что для максимизации эффективности применения АРП желательно минимизировать число промежуточных обменов массивов в АРП и соответственно модернизировать имеющиеся

алгоритмы. Если функция pop() вызывается другой функцией, то все данные уже находятся в АП и потом используются без перезагрузки, поэтому в данном случае можно говорить о выигрыше на порядки [ ].

2.3.3.ВЫВОДЫ

Во второй главе настоящей работы представлен обзор технических устройств, позволяющих реализовать АРП – это магниторезистивная память и ее вариации. Произведен функциональный анализ свойств ассоциативных решающих полей. Формализованы такие операции, как сравнение элементов массива с константой, загрузка данных в АРП.

3.ИССЛЕДОВАНИЕ ВЫЧИСЛИТЕЛЬНЫХ СВОЙСТВ АРП ПРИ ВЫПОЛНЕНИИ ОПЕРАЦИИ СОРТИРОВКИ

В разделе 2.2 настоящей работы была описана архитектура АРП. Далее, опираясь на представленное описание, необходимо провести исследование вычислительных свойств ассоциативной памяти. Для этой задачи будет рассматриваться одно из наиболее ресурсозатратных преобразований – сортировка.

3.1. ПОНЯТИЕ «СОРТИРОВКА»

Слово «сортировка» определяется как перестановка предметов, при которой они располагаются в порядке возрастания или убывания. Пусть имеется набор значений и соответствующий набор адресов , сформированный случайным образом (формула 19).

Тогда массив значений задается следующим выражением (формула 18).

, (18)

где – значения, сформированные случайным образом;

– адреса.

(19)

Определение адресов описано в выражении (20).

. (20)

Тогда сортировка в общем случае будет определять местоположение значений. Функция упорядочивания пар ключ-значение сравнивает два значения и располагает их в нужной последовательности.

(21)

Алгоритмы сортировки применяются для решения различных задач.

а. *Задача группирования* заключается в том, чтобы отобрать все элементы с одинаковыми значениями некоторого признака. Пусть имеется *N* элементов, расположенных случайным образом, причем значения некоторых

элементов равны. Необходимо переупорядочить массив так, чтобы равные по значению элементы занимали соседние позиции в массиве. Один из способов решения данной задачи – сортировка массива, т.е. расположение элементов в порядке не убывания.

б. *Поиск общих элементов в двух или более массивах*. Если два или более массивов отсортировать в одном порядке, то можно найти в них все общие элементы за один последовательный просмотр всех массивов без повторов. Сортировка позволяет использовать последовательный доступ к большим массивам в качестве приемлемой замены прямой адресации.

в. *Поиск информации по значениям ключей*. Сортировка используется и при поиске, с ее помощью можно сделать результаты обработки данных более удобными для восприятия человеком.

Существует большое множество различных алгоритмов сортировки. Но они различны по быстродействию и эффективности использования ресурсов. Все зависит от механизма упорядочивания элементов, способов выбора опорного элемента сравнения, требований к дополнительной памяти. Для анализа эффективности этих алгоритмов в условиях ассоциативной памяти, были рассмотрены следующие виды сортировок:

- обменная сортировка – сортировка пузырьком;
- сортировка выбором;
- сортировка вставками.

Для оценки работы алгоритма сортировки используется понятие «временная сложность». Сложность алгоритмов обычно оценивают по времени выполнения или по используемой памяти. В обоих случаях сложность зависит от размеров входных данных: массив из 100 элементов будет обработан быстрее, чем аналогичный из 1000. При этом точное время зависит от процессора, типа данных, языка программирования и множества других параметров.

Вычислительная сложность алгоритма – это оценка объёма работы в зависимости от объёма входных данных. Для чёткого описания сложности алгоритма используется асимптотическая сложность – примерная оценка времени работы алгоритма. Обозначается как .

3.1.1.МЕТОД ПУЗЫРЬКА

Семейство обменных сортировок относится к методам обмена или транспозиций, подразумевающих систематический обмен местами между элементами пар, в которых нарушается упорядоченность, до тех пор, пока таких пар не останется.

Суть алгоритма пузырьковой сортировки заключается в том, чтобы при сравнении ключей адреса поменять местами, если соответствующие ключи расположены не в том порядке, который задается условиями задачи. Затем продолжать то же самое с , и т.д. Во время выполнения этих операций записи с наибольшими ключами будут двигаться выше по массиву. Описанная последовательность операций завершится тем, что запись с наибольшим ключом займет положение . При многократном выполнении данного процесса соответствующие записи переместятся в позиции , и т.д. В итоге все записи будут упорядочены.

1. Присвоить INDEX . INDEX – индекс самой первой записи, о которой еще не известно, заняла ли она свою окончательную позицию. В начале сортировки еще ничего не известно о порядке размещения элементов массива.

2. Присвоить . Выполнить п.3 при Затем перейти к п.4. (Если то перейти к 4).

3. Если , то поменять местами и установить .

4. Если , завершить выполнение сортировки. В противном случае присвоить и перейти к п.2.

Время работы алгоритма сортировки методом пузырька сильно зависит от числа проходов, числа обменов и числа сравнений.

В представленном алгоритме сортировки возможна минимизация времени обработки массива. Последовательность записей можно просматривать попеременно в обоих направлениях. При таком проходе среднее число сравнений несколько сокращается. В [8] представлено интересное наблюдение: если j – такой индекс, что и не меняются местами на двух последовательных проходах в противоположных направлениях, то записи и должны занимать свои окончательные позиции и их можно исключить из последующих сравнений. Например, просматривая перестановку 4 3 2 1 8 6 9 7 5 слева направо, получаем 3 2 1 4 6 8 7 5 9; записи и не поменялись местами. При просмотре последней перестановки справа налево все еще меньше записи (новой записи). Следовательно, можно сразу же сделать вывод о том, что записи и могут и не участвовать ни в одном из последующих сравнений.

3.1.2.СОРТИРОВКА ПУТЕМ ВСТАВОК

Идея вставки элемента в уже отсортированную часть массива заключается в следующем. Пусть и записи уже размещены так, что . Ключи сравниваются по очереди с , , … до тех пор, пока не обнаружится, что запись следует вставить между и . Тогда записи передвинутся на одну позицию вверх. Новая запись в позицию . После завершения алгоритма все ключи будут упорядочены: .

1. Выполнить шаги от 2 до 5 при и после этого завершить выполнение процедуры.

2. Присвоить . (При выполнении следующих этапов будет предпринята попытка вставить запись *P* в нужное место, сравнивая .)

3. Если , то перейти к п.5. (Нашлось искомое место для записи *P*.)

4. Присвоить . Если то вернуться к п.3. (Если то ключ K – наименьший из рассмотренных ранее, а значит, запись *P* должна переместиться на первую позицию.)

5. Присвоить .

3.1.3.СОРТИРОВКА ПОСРЕДСТВОМ ВЫБОРА

Обобщенный вариант алгоритма сортировки посредством выбора сводится к следующим пунктам.

а. Поиск наименьшего ключа, соответствующая запись пересылается в область вывода, и ключ заменяется значением ∞ (которое по предположению больше любого реального ключа).

б. Повторить пункт (а). В этот раз будет выбран ключ, наименьший из оставшихся.

в. Повторять пункт (а) до тех пор, пока все N записей не будут выбраны.

На шаге (а) требуется выполнять сравнений каждый раз, когда выбирается очередной элемент. Также необходимо выделить отдельную область памяти для накопления результата. Но для уменьшения затрат на дополнительную память отобранное значение можно записать в соответствующую окончательную позицию, а запись, которая ее занимала, перенести на место отобранной. Тогда эту позицию не нужно будет рассматривать при последующих выборах. На этой идее основан следующий алгоритм.

Записи перекомпоновываются в пределах того же объема памяти. Сортировка основывается на описанном выше алгоритме.

1. Выполнить пункты 2 и 3 при

2. Найти наибольший из ключей пусть это будет , где *i* выбирается как можно большим.

3. Взаимно переставить записи . (Теперь записи занимают свои окончательные позиции.)

Интересно сравнить алгоритм сортировки посредством выбора с сортировкой методом пузырька. Метод пузырька можно рассматривать как алгоритм выбора, поскольку в нем за один раз выбирается более одного элемента. Следовательно, при сортировке методом пузырька выполняется меньше сравнений, чем при простом выборе, и этот алгоритм может показаться более быстрым. Но в действительности метод пузырька работает медленнее метода сортировки посредством выбора. Сортировка методом пузырька проигрывает из-за того, что выполняется слишком много обменов, в то время как при сортировке посредством простого выбора данные пересылаются довольно редко.

Время работы алгоритма сортировки посредством выбора зависит от числа элементов, числа сравнений и числа правосторонних максимумов.

3.1.4.ПРЕДСТАВЛЕНИЕ КЛАССИЧЕСКИХ АЛГОРИТМОВ СОРТИРОВКИ В АРП

В рассматриваемой главе представлены классические алгоритмы сортировки. Было замечено, что данные алгоритмы при отображении на ассоциативную память не отражают всех особенностей ее работы. Так как в их основе заложено последовательное сравнение и преобразование данных. Поэтому для каждого алгоритма был предложен вариант повышения быстродействия за счет упрощения сложности алгоритма.

Сортировка методами пузырька и выбором предполагает достижение упорядоченности за счет перестановки двух элементов после сравнения. Такой принцип неэффективен для ассоциативных решающих полей, т.к. в АРП можно не использовать адресацию, перемещение элементов порождает определенную задержку по времени.

При сортировке посредством вставками найденный элемент вставляется в нужную позицию, т.е. . Соответственно

остальные элементы меняют свои адреса (раздел 3.1.2). Для АРП такие вставки также будут неэффективны, так как необходимо менять адресацию сразу для набора значений.

Скорость работы рассмотренных выше алгоритмов сортировки сильно зависит от объема исходных данных. Так как средняя временная сложность алгоритмов . Ассоциативная память разрешает эту проблему за счет послайсной обработки данных. Например, чтобы найти нужную позицию для значения в массиве данных классическим способом необходимо произвести последовательный опрос по каждому полю в памяти, пока не будет найден больший или меньший элемент в зависимости от поставленных условий. Рассмотрим эту же задачу сравнения элемента с массивом при сортировке по возрастанию в АРП.

Пусть имеется массив данных *G*, загруженный в матрицу АРП. Исходный массив состоит из *N L*-разрядных элементов.

Элемент необходимо расположить в регистре опроса/записи D.

, (22)

где – разряд регистра опроса/записи;

– разряд значения Q, позицию которого необходимо найти.

В регистре маски M в левом разряде выставляется значение 1.

Производится опрос по первому разряду для всех элементов массива. Возможное количество вхождений для всего массива – от 1 до N. Все найденные вхождения, отмечаются в регистре совпадения R. Далее, маска сдвигается на один разряд вправо, и производится новый опрос. Так, пока не будет найден элемент самый близкий по значению к .

Если совпадений по массиву нет, в правом незамаскированном разряде регистра опроса/записи стоит 0, то происходит смена значения разряда на единицу. И производится новый поиск. Если в правом незамаскированном

разряде регистра опроса/записи стоит 1, то – самый максимальный элемент из опрашиваемой части массива.

Таким образом, сравнение одного значения по всему массиву занимает от 1 до L тактов, где L – разрядность поля в АРП.

3.1.5.ВЫВОДЫ

В таблице (2) приведены классические значения оценки временной сложности алгоритмов сортировки и оценки, полученные при изменении алгоритма с учетом особенностей ассоциативного решающего поля (– количество элементов, – разрядность).

Таблица 3 – Сравнение временных оценок сложности

	Классическая временная сложность (в среднем)	Временная сложность для АП
Пузырьковая сортировка		
Сортировка вставками		
Сортировка выбором		
Бисерная сортировка		

Из таблицы (3) видно, что за счет сравнения элемента с массивом не последовательно, а поразрядно, в общем случае уменьшается временная сложность в ─ раз. В ходе анализа отображения алгоритмов было выявлено, что сортировка в ассоциативной памяти сводится к опросам на совпадение. Поэтому можно сформулировать новый алгоритм сортировки для АРП, отвечающий всем структурным и функциональным особенностям данной архитектуры.

4. АЛГОРИТМ СОРТИРОВКИ

Архитектура и принцип работы ассоциативных решающих полей позволяют уйти от сортировки посредством перестановки элементов. Следовательно, задача сортировки в ассоциативной памяти сводится к поиску экстремумов, которые, в зависимости от поставленной задачи, могут помечаться в дополнительных регистрах с присвоенными им адресами, исключаться из дальнейшего рассмотрения или копироваться в другую область памяти.

Рассмотрим задачу сортировки по убыванию.

Пусть имеется исходный неотсортированный массив данных , состоящий из L-разрядных элементов. Загрузка в память может осуществляться построчно (с помощью регистра данных), или послайсно (с использованием ортогонального буфера). После загрузки в матрицу АРП, массив данных располагается в массиве данных .

Для АРП необходимо ввести дополнительное поле для адреса, либо выделить такой же объем памяти для результирующего отсортированного массива.

Введем дополнительный регистр результата Z в устройство управления. Нумерация разрядов регистров опроса/записи , маски и полей данных массива осуществляется слева направо (где), регистра совпадений и регистра результата сверху вниз (где).

Для нахождения максимального значения в массиве данных необходимо использовать следующий алгоритм.

1. Введем переменную .

2. В k-ом разряде регистра опроса/записи D и регистра маски M выставить значение 1, а все остальные, ранее не участвовавшие в опросе, в 0.

, (23)

где – разряд регистра опроса/записи;

– разряд регистра маски;

– разрядность регистра опроса/записи, массива и регистра маски.

3. Опросить все доступные элементы на совпадение ( ).

4. Если в регистре совпадения *R* содержится хотя бы одна единица, значит, в массиве есть элементы, у которых в первом разряде 1. Тогда для этих значений продолжается опрос на следующее вхождение с единицей в следующем разряде.

, (24)

где – разряд регистра совпадения;

– номер разряда;

– разряд регистра результата.

5. Иначе – опрос для всех элементов, участвовавших в предыдущем опросе, на наличие 0 в k-ом разряде.

(25)

Выполнение алгоритма продолжается до тех пор, пока не будет произведен опрос для *L*-го разряда. Для найденного значения в регистре результата *Z* выставляется 1, тем самым, исключая элемент из дальнейшего рассмотрения ( ) (формула 26).

. (26)

Для этого значения также выставляется адрес в регистре адреса, либо элемент добавляется в результирующий массив данных. Далее алгоритм повторяется.

Так найден максимальный элемент для заданного массива данных. Чтобы найти следующее значение, необходимо повторить описанный ранее алгоритм без учета тех элементов, которые уже исключены из рассмотрения

.

Сложность для алгоритмов с последовательными вычислениями в среднем (формула 27).

,

(27)

,

где – количество тактов на опрос разряда на совпадение;

– количество тактов на сравнение;

– количество тактов на определение знака.

Сложность для алгоритмов с параллельными вычислениями (формула 28).

,

(28)

где – количество тактов на обработку 1 слайса;

– количество тактов на опрос разряда на совпадение;

– коэффициент времени на обработку 1 разряда;

– коэффициент времени на обработку всего слайса.

Коэффициент эффективности показывает отношение временной сложности последовательных вычислений к параллельным вычислениям и характеризует оценку выигрыша по времени при использовании параллельной обработки.

– (29)

Значения временных коэффициентов представлено в таблице 4.

Таблица 4 – Значения временных коэффициентов

2	5	10
5	5	25
5	10	50

Изменение коэффициента эффективности в зависимости от количества элементов *n* и временных коэффициентов при разрядности слов массива представлено в таблице 5.

Таблица 5 – Динамика Коэффициента эффективности

n	10	25	50
100	0.625	0.25	0.125
1000	6.25	2.5	1.25
10000	62.5	25	12.5
100000	625	250	125

5. ЗАКЛЮЧЕНИЕ

Сочетание таких свойств, как высокая производительность при групповой обработке массивов информации, достаточная универсальность делает Ассоциативную память перспективным устройством для широкого класса систем обработки информации. С использованием АП появляется возможность решать ранее не решаемые задачи и повысить эффективность функционирования существующих и разрабатываемых систем обработки информации.

Для выполнения работы был проведен поиск и анализ информации по теме ассоциативной памяти. Поиск осуществлялся по базам данных патентов, диссертаций, научных работ. В результате поиска и анализа выяснилось, что существуют реализации ассоциативной памяти на магниторезистивной памяти. Ассоциативная память обладает большим потенциалом повышения производительности обработки информации. Но проведено мало исследований вычислительных свойств ассоциативных решающих полей.

В настоящей работе был проведен функциональный анализ АРП. Исследование вычислительных свойств на примере задачи сортировки. После анализа отображения классических алгоритмов сортировки на АРП был предложен новый алгоритм сортировки, который учитывает все функциональные и структурные особенности АРП.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Бэкус, Дж. Можно ли освободить программирование от стиля фон Неймана? Функциональный стиль и соответствующая алгебра программ. // Лекции лауреатов премии Тьюринга за первые двадцать лет 1966-1985. / под ред. Р. Эшенхерста. – М.: Мир, 1993. – С. 84-158.
2. 42 Years of Microprocessor Trend Data. – https://www.karlrupp.net/2018/02/42-years-of-microprocessor-trend-data/. Дата обращения 05.03.2019.
3. Moore, G.E. No Exponential is Forever: But "Forever" Can Be Delayed! // Digest of Technical Papers of IEEE International Solid-State Circuits Conference. – 2003. – P. 20–23.
4. HUAWEI Создала первые в мире ИИ-процессоры, пойдя по пути разработчиков «Эльбрусов». – https://blog.huawei.ru/technology/huawei. Дата обращения 09.04.2019.
5. Пат. 2572464 Российская Федерация, МПК G11C 11/00 (2006.01). Ячейка магнитной оперативной памяти с двойным переходом для применений троичной ассоциативной памяти / К. Бертран. – № 2012111795/28; заявл. 27.03.2012; опубл. 10.01.2016, Бюл. № 1.
6. Qing, G., AC-DIMM: Associative Computing with STT-MRAM / G. Qing, G. Xiaochen, R. Patel, E. Ipek, G. Friedman // ISCA’13Tel-Aviv. – 2013. – P. 189–200.
7. Yavits, L. Resistive associative processor/ L. Yavits, S. Kvatinsky, A. Morad, R. Ginosar // IEEE Computer Architecture Letters. – 2015. – P. 148-151.
8. Iverson, K. E., A Programming Language // Wiley. – 1962. P. 218-219
9. Марков, А.А. Теория алгорифмов / А. А. Марков, Н. М. Нагорный. – М.: Наука, 1984. – 432 с.
10. Фостер, К. Ассоциативные параллельные процессоры. – М.: Энергоиздат. 1981. – 240с.

11. Evaluation of a non-volatile FPGA based on MRAM technology. Conference Paper, January 2006, DOI: 10.1109/ICICDT.2006.220782. Source: IEEE Xplore.
12. Atsushi Ooka, Shingo Atat, Kazunari Inoue, Masayuki Murata. Design of a high-speed content-centric-networking router using content addressable memory. 2014 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS). DOI: 10.1109/INFCOMW.2014.6849275 Source: IEEE Xplore.
13. Кнут, Д. Искусство программирования в 4 томах, том 3. Сортировка и поиск / Дональд Кнут; пер. с анг. В. Т. Тертышный, И. В. Красиков. М.: Вильямс, 2001. – 720 с.
14. Arsovski, T. Chandler, and A. Sheikholeslami. Aternary content-addressable memory (TCAM) based on 4T static storage and including a current-race sensing scheme. Solid-State Circuits, Journal of. – 2003. – P. 155
15. Кафтанников, И.Л. Модель массовой обработки данных в ассоциативном решающем поле. / И.Л. Кафтанников, Г.А. Никитин // Распараллеливание обработки информации: тезисы докладов и сообщений 5 Всесоюзной школы-семинара: Ч.1.– Львов, 1985. – С. 130–131.
16. Abu Sebastian. Temporal correlation detection using computational phase-change memory // Nature Communications. Springer Nature. – 2017.
17. Кохонен, Т. Ассоциативная память / Т. Кохонен. – М.: Мир, 1985 – 238 с.
18. Eshraghian, K. Memristor MOS content addressable memory (MCAM): Hybrid architecture for future high performance search engines // K. Eshraghian, R. Cho, O. Kavehei, S.-K. Kang, D. Abbott, and S.-M. S. Kang. // Very Large Scale Integration (VLSI) Systems, IEEE Transactions on. – 2011. – P. 1407

19. 00_PlenarPM_11_PerevozchikovA_Part1. – http://2017.nscf.ru/postkremnievye-vychisleniya/postkremnievye-vychisleniya-sbornik-tezisov. Дата обращения 10.04.2019
20. Кабак, И.С. Аппаратная реализация ассоциативной памяти произвольного размера / И.С. Кабак, Н.В. Суханова // Вестник МГТУ «Станкин». Серия «Информатика и моделирование». – 2010. – №1 (9). – С. 135 – 139.
21. Таненбаум Э. Архитектура компьютера / Э. Таненбаум – 6-е изд. – СПб.: Питер, 2013. – 809 с.
22. Однородные микроэлектронные ассоциативные процессоры / И.В. Прангишвили, Г.М. Попова, О.Г. Смородинова, А.А. Чудин. – М.: Сов. Радио, 1973. – 280 с.
23. Борисов В.В. Проблемы интеграции средств ассоциативной обработки в технические информационные системы // Системы компьютерной математики и их приложения / Борисов В.В., Полячков А.В., Чандер А.П., – 2015. – № 16. – С. 67-70.