Министерство науки и высшего образования Российской Федерации
Муромский институт (филиал)
федерального государственного бюджетного образовательного учреждения высшего образования
«Владимирский государственный университет
имени Александра Григорьевича и Николая Григорьевича Столетовых»
(МИ ВлГУ)

Факультет Информационных технологий

Кафедра Информационные системы

ОТЧЕТ

по преддипломной практике
(наименование практики)

Ми ВЛГУ
(место прохождения практики)

Руководитель

_____ _____
(оценка) (подпись) (от предприятия)

Члены комиссии к.т.н, доц. каф. ИС Щаников С.А.
(подпись) (от института)

Студент ИС-117
(группа)

_____ _____ Минеев Р.Р.
(подпись) (Ф.И.О.) (фамилия, инициалы)

_____ _____
(подпись) (Ф.И.О.)

_____
(подпись) (дата)

Муром 2021

БЛАНК ЗАДАНИЯ

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Имитационное моделирование — ключевая технология в работе с ИИ. Оно особенно полезно при создании системы на основе обучения с подкреплением, для которой сложно или невозможно получить входные данные из реального мира.

За последние 5 лет в машинном обучении развивается новый интересный подход - in-memory computing. Применить такой подход стало возможно за счет разработки нового вида оперативной памяти ReRAM, то есть Resitive RAM - памяти, построенной на переменных резисторах. Для оперативной памяти преимущества такой системы заключаются в том, что по скорости она не уступает транзисторной ОЗУ, но при выключении питания вся информация сохраняется. Для машинного обучения применение ReRAM заключается в том, что самую частую операцию взвешивания входного вектора можно выполнять не в цифровом виде на процессоре, как это делается сейчас, а в аналоговом. Каждая ячейка ReRAM выступает в роли делителя напряжения, который изменяет входное напряжение на определенный коэффициент. Она может выполнять умножение весовых коэффициентов в аналоговом виде, а операция умножения является самой частой в машинном обучении. Проблема в таком подходе заключается в том, что аналоговый принцип обработки информации в ReRAM вносит погрешности из-за неизбежного влияния шумов и помех [1]. Этим обусловлена актуальность темы.

Цель, поставленная на преддипломную практику, – произвести анализ технического задания.

Задачи, поставленные на преддипломную практику:

- проанализировать необходимость тестирования нейронных сетей;
- произвести обзор методов параллельных вычислений;
- обосновать выбор средств реализации разрабатываемой программы;
- предъявить требования к разрабатываемой программе.

1 Необходимость тестирования нейронных сетей

Создание искусственных нейронных сетей было вдохновлено биологическими аналогами – нейронными сетями живых организмов, которые отлично справляются с решением сложно формализуемых и не формализуемых математически задач, таких как распознавание, классификация, кластеризация, обобщение и т.д. Уоррен Мак-Каллок, который был нейрофизиологом, и Уолтер Питс, который был математиком, предложили собственную модель, описывающую процесс функционирования биологических нейронных сетей в упрощенном виде. Френк Розенблат реализовал эту модель аппаратно в виде первого в мире нейрокомпьютера Марк 1 [2]. Для хранения инструкций использовалась перфорированная лента, а для работы с данными — электромеханические регистры. Это позволяло одновременно пересылать и обрабатывать команды и данные, благодаря чему значительно повышалось общее быстродействие компьютера. Однако такой многообещающий старт развития технологии, в которой компьютер не программируется, а обучается выполнению интеллектуальных задач, был внезапно прерван, после опубликования книги “Персептроны” Марвина Минского в 1970-х годах [3].

Развитие нейрокомпьютеров с тех пор было приостановлено, а теория машинного обучения совершенствовалась лишь в области математического аппарата. В это время активное развитие наблюдалось в области программируемой цифровой техники, результаты, которого мы видим в настоящее время: мобильная электроника, персональные компьютеры, специализированные компьютеры для медицины, военной промышленности и т.п. Вся эта техника функционирует на 2 основных принципах:

- архитектура фон Неймана;
- транзистор, как основной элемент аппаратной реализации.

Совместное использование шины для памяти программ и памяти данных приводит к узкому месту архитектуры фон Неймана, а именно ограничению пропускной способности между процессором и памятью по сравнению с объемом

памяти [4]. Из-за того, что память программ и память данных не могут быть доступны в одно и то же время, пропускная способность канала «процессор-память» и скорость работы памяти существенно ограничивают скорость работы процессора — гораздо сильнее, чем если бы программы и данные хранились в разных местах [5] (рис.1).

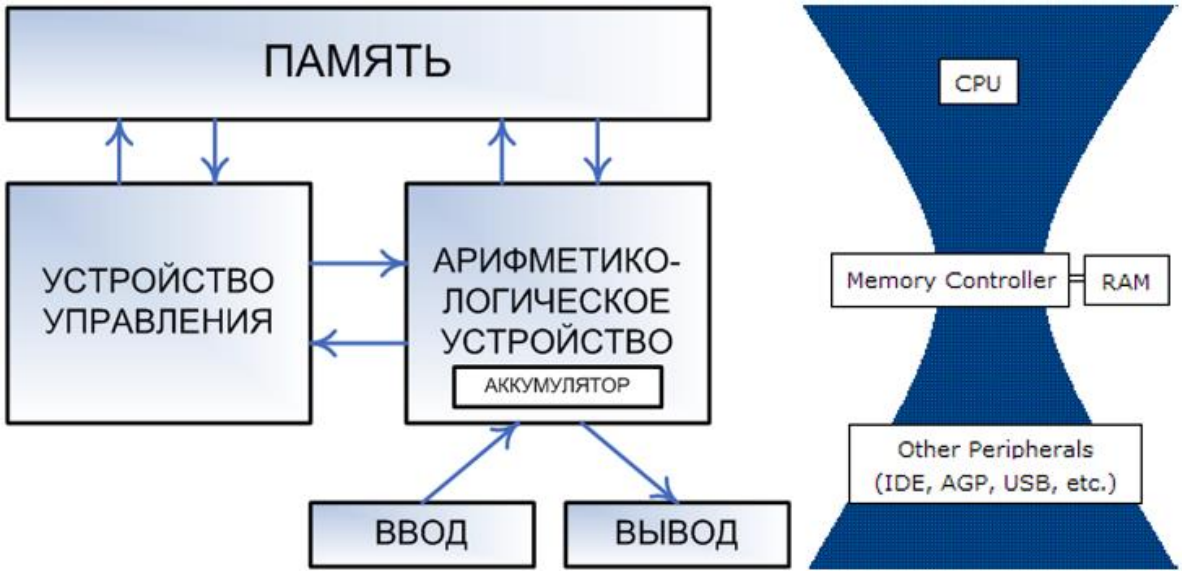

Рисунок 1 – Архитектура Фон Неймана и ее проблема

Применение транзисторов так же имеет проблемы связанные с масштабированием и перегревом. Проблема масштабирования хорошо описывается законом Мура. Закон Мура – эмпирическое наблюдение, изначально сделанное Гордоном Муром, согласно которому (в современной формулировке) количество транзисторов, размещаемых на кристалле интегральной схемы, удваивается каждые 24 месяца. В 2007 году Мур заявил, что закон, очевидно, скоро перестанет действовать из-за атомарной природы вещества и ограничения скорости света. Это означает, что в ближайшие годы, для поддержания имеющихся темпов уменьшения размеров транзисторов в процессорах, их размер должен стать меньше размера атома, что физически не возможно. Экспоненциальный рост физических величин в течение длительного времени невозможен, и постоянно достигаются те или иные пределы. Лишь эволюция транзисторов и технологий их изготовления продлевает действие закона.

Другая, вытекающая из данного закона проблема – нарастающее энергопотребление и соответственно нагрев цифровых процессоров. За 50 лет энергопотребление выросло на 2 порядка, что влечет за собой не только экономические проблемы, вызванные высокими затратами на обслуживание электросетей, но и экологические, вызванные увеличенным расходом теплоносителей в процессоре (рис.3).

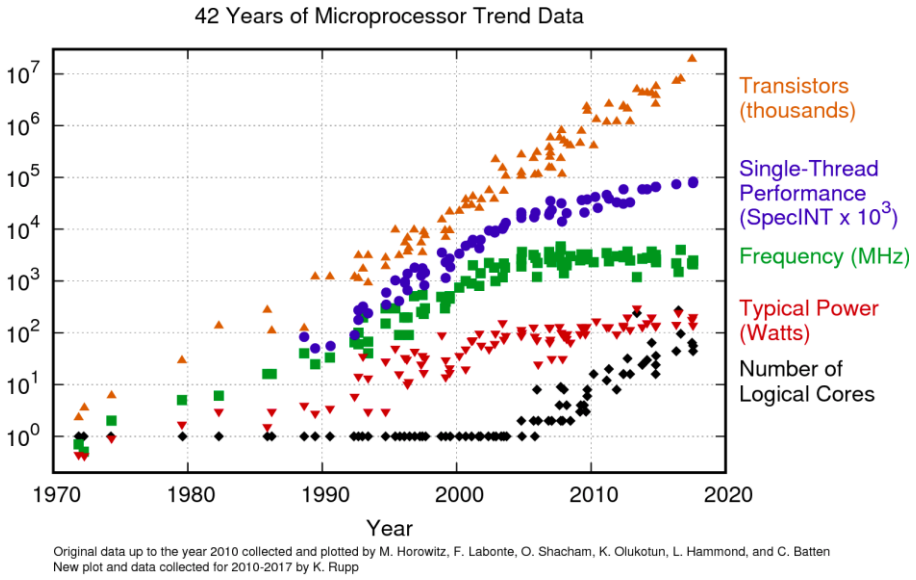

Рисунок 3 – Тренд роста энергопотребления

Альтернативой решения рассмотренных проблем является переход от рассмотренного выше подхода к применению новых парадигм вычисления и новых электронных компонентов. Для этого необходимо сменить архитектуру вычислений и материалы электронных компонентов.

Как было сказано в выше, искусственные нейронные сети были вдохновлены их биологическими аналогами, и по своей природе они являются параллельными средствами обработки информации. Такой параллелизм можно достигнуть за счет применения большого количества простых вычислительных узлов в целом выполняющих одну большую сложную задачу.

Рассматриваемые материалы – это переменные резисторы на основе тонких пленок различных диэлектриков. Из них делают новые виды оперативной памяти – ReRAM (рис.4). ReRAM очень подходит для аппаратной реализации нейронных

сетей, так как отдельная ячейка памяти может выполнять умножение на весовой коэффициент по закону Ома, что значительно увеличит быстродействие и снизит энергопотребление интеллектуальных систем.

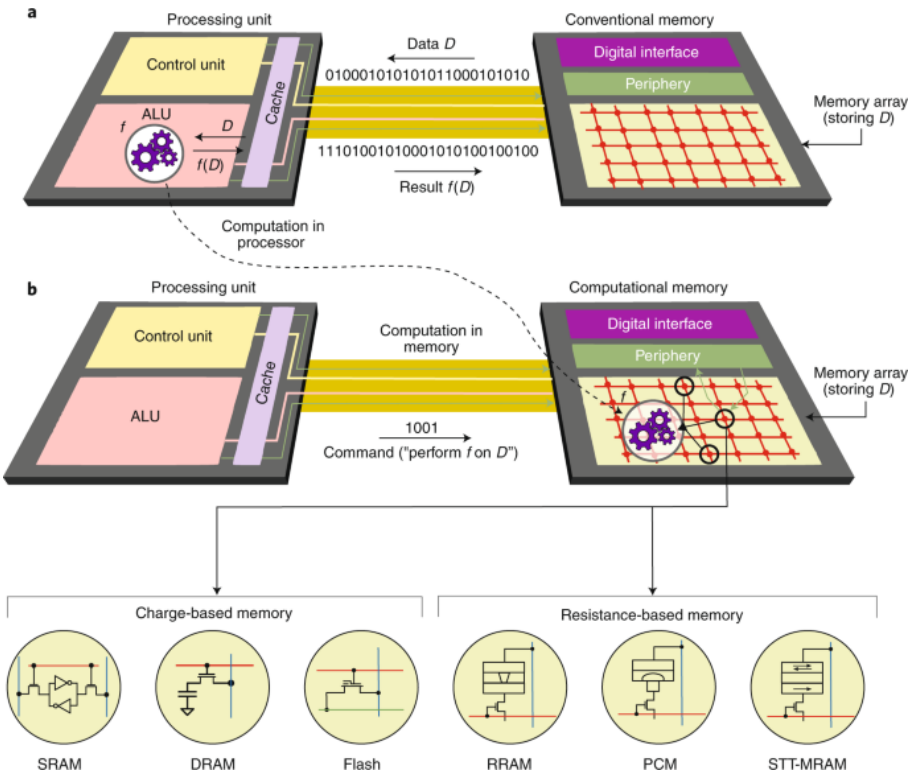

Рисунок 4 – Организация вычислений в ReRAM

Однако достигнутое на этапе компьютерного проектирования номинальное качество работы ИНС снижается в реальных условиях эксплуатации при реализации на ReRAM во многих случаях до полной потери работоспособности из-за аналогового принципа обработки информации. Причиной этого является неизбежное влияние внутренних и внешних физических и информационных дестабилизирующих факторов, а также производственных и эксплуатационных погрешностей значений параметров элементов и платформы их реализации.

В связи с вышесказанным, актуальными являются исследования, направленные на разработку автоматизированных инженерных методов проектирования высоконадежных ИНС, позволяющих приблизить технические параметры и характеристики ИНС к их потенциально достижимым значениям.

2 Обзор методов организации параллельных вычислений

Необходимость разделять вычислительные задачи и выполнять их одновременно (параллельно) возникла задолго до появления первых вычислительных машин.

Существует 3 метода распараллеливания расчетов:

- распараллеливание по задачам (такое распараллеливание актуально для сетевых серверов и других вычислительных систем, выполняющих одновременно несколько функций либо обслуживающих многих пользователей);

- распараллеливание по инструкциям (аппаратно реализовано в современных центральных процессорах общего назначения, поскольку оно эффективно при исполнении программ, интенсивно обменивающихся разнородной информацией с другими программами и с пользователем компьютера);

- распараллеливание по данным [6].

Потоковая обработка данных особенно эффективна для алгоритмов, обладающих следующими свойствами, характерными для задач физического и математического моделирования:

- большая плотность вычислений — велико число арифметических операций, приходящихся на одну операцию ввода-вывода (например, обращение к памяти);

- локальность данных по времени - каждый элемент загружается и обрабатывается за время, малое по отношению к общему времени обработки, после чего он больше не нужен.

Ядра современных центральных процессоров являются суперскалярными, поддерживая векторную обработку (расширения SSE и 3DNow!), сами же CPU обычно содержат несколько ядер. Однако графические процессоры включают в себя тысячи параллельных «вычислителей». Кроме того, при поточно-параллельных расчетах графические процессоры имеют преимущество благодаря следующим перечисленным особенностям архитектуры:

- память GPU (Graphics Processing Unit) оптимизирована на максимальную пропускную способность (а не на скорость произвольного доступа, как у CPU), что

ускоряет загрузку потока данных;

- большая часть транзисторов графического процессора предназначена для вычислений, а не для управления исполнением программы;

- при запросах к памяти, за счет конвейерной обработки данных, не происходит приостановки вычислений.

Однако обработка ветвлений (исполнение операций условного перехода) на GPU менее эффективна, поскольку каждый управляющий блок обслуживает не один, а несколько вычислителей.

Таким образом, производительность одного GPU при хорошо распараллеливаемых вычислениях аналогична кластеру из сотен обычных вычислительных машин, причем графические процессоры сейчас поддерживают практически все операции, используемые в алгоритмах общего назначения:

- математические операции и функции вещественного аргумента;

- организацию циклов;

- операции условного перехода (которые исполняются сравнительно медленно, поскольку в составе GPU блоков управления меньше, чем вычислительных блоков) [7].

В различных источниках информации можно найти много разных определений процессов и потоков. Такой разброс определений обусловлен, в первую очередь, эволюцией операционных систем, которая приводила к изменению понятий о процессах и потоках, во-вторых, различием точек зрения, с которых рассматриваются эти понятия.

С точки зрения пользователя, процесс — экземпляр программы во время выполнения, а потоки — ветви кода, выполняющиеся «параллельно», то есть без предписанного порядка во времени.

С точки зрения операционной системы, процесс — это абстракция, реализованная на уровне операционной системы. Простейшей операционной системе не требуется создание новых процессов, поскольку внутри них работает одна-единственная программа, запускаемая во время включения устройства. В более сложных системах необходимо создавать новые процессы для возможности

функционирования нескольких программ.

Процесс — это просто контейнер, в котором находятся ресурсы программы:

- адресное пространство;
- потоки;
- открытые файлы;
- дочерние процессы и т.д.

Также, с точки зрения операционной системы, поток — это абстракция, реализованная на уровне операционной системы. Поток был придуман для контроля выполнения кода программы. Это контейнер, в котором находятся счетчик команд, регистры и стек.

Поток легче, чем процесс, и создание потока стоит дешевле. Потоки используют адресное пространство процесса, которому они принадлежат, поэтому потоки внутри одного процесса могут обмениваться данными и взаимодействовать с другими потоками.

Поддержка множества потоков внутри одного процесса в рамках работы над разрабатываемой программой необходима. В случае, когда одна программа выполняет множество задач, поддержка потоков в одном процессе позволяет:

- разделить ответственность за разные задачи между разными потоками;
- повысить быстродействие.

Отличие процесса от потока состоит в том, что процесс рассматривается операционной системой, как заявка на все виды ресурсов (память, файлы и пр.), кроме одного — процессорного времени. Поток — это заявка на процессорное время. Процесс — это всего лишь способ сгруппировать взаимосвязанные данные и выделить область локальной памяти, а потоки — это единицы выполнения, которые выполняются на процессоре.

3 Выбор средства

Для разработки программы были выбраны следующие средства:

- язык программирования Python;
- модуль Threading;
- модуль Multiprosessing.

Язык программирования Python был выбран исходя из того, что данный язык часто используется для написания нейросетей.

В случае с разработкой данной системы использование потоков не подойдет, так как Python имеет GIL (Global Interpreter Lock).

GIL – это своеобразная блокировка, позволяющая только одному потоку управлять интерпретатором Python. Это означает, что в любой момент времени будет выполняться только один конкретный поток.

Работа GIL может казаться несущественной для разработчиков, создающих однопоточные программы. Но во многопоточных программах GIL может негативно повлиять на производительность процессоро-зависимых программ.

Модуль Threading впервые был представлен в Python 1.5.2 как продолжение низкоуровневого модуля потоков. Модуль Threading значительно упрощает работу с потоками и позволяет программировать запуск нескольких операций одновременно. Потоки в Python лучше всего работают с операциями ввода/вывода, такими как загрузка ресурсов из интернета или чтение файлов на компьютере.

Модуль Multiprocessing был добавлен в Python версии 2.6 [7]. Изначально он был определен в PEP 371 Джесси Ноллером и Ричардом Одкерком. Модуль Multiprocessing позволяет создавать процессы таким же образом, как при создании потоков при помощи модуля Threading. Использование данного модуля позволяет обойти GIL и воспользоваться возможностью использования нескольких процессоров на компьютере.

4 Требования к разрабатываемой программе

На вход разрабатываемой программе должна поступать модель нейронной сети, а также тестовые значения алгоритма рандомизации данных.

Разрабатываемая программа должна иметь следующие функции:

- функцию загрузки модели из класса программы (а также, описание входных данных тестирования нейронной сети: на чем она обучалась, каким образом загружается тестовая выборка);

- функция определения и вывода временных данных (например, весов нейросетей на всех эпохах обучения);

- функция загрузки полученных данных в исходную модель (таким образом создается новая модель, с новыми данными);

- функция, возвращающая показатели нейросети (точность или расхождение) после проверки модели.

Задачей разрабатываемой программы является повышение скорости расчетов большого количество данных (большого количества тестов, которые будут проводиться для модели нейронной сети). Для этого необходимо равномерно распределить нагрузку по всему процессору.

На выходе пользователь программы должен получать график и возможность сохранения готового ответа, выданного программой. А также необходимо дать пользователю возможность запустить повторное тестирование над той же моделью, если это потребуется, без перезапуска данной программы.

ЗАКЛЮЧЕНИЕ

Во время преддипломной практики был проведен анализ проекта, разрабатываемого в рамках выпускной квалификационной работы.

В рамках отчета по преддипломной практике были рассмотрены следующие задачи:

- проанализирована необходимость тестирования нейронных сетей;
- произведен обзор методов параллельных вычислений;
- обоснован выбор средств реализации разрабатываемого программного продукта;
- предъявлены требования к разрабатываемой программе.

Итогом преддипломной практики является полный анализ технического задания к выпускной квалификационной работе.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1 Перепелкин, Е.Е. Вычисления на графических процессорах (GPU) в задачах математической и теоретической физики; Огни - Москва, 2017. - 750 с.

2 Галушкин, А. И. Нейрокомпьютеры. Учебное пособие / А.И. Галушкин. - М.: Альянс, 2014. - 528 с.

3 Смирнов А. Д. Архитектура вычислительных систем: Учебное пособие для вузов. — М.: Наука, 1990. — 320 с.

4 Таненбаум, Э.С. Современные операционные системы: [пер. с англ.]. Питер, 2011. – 1115 с.

5 Каган, Б. М., Каневский, М. М., Цифровые вычислительные машины и системы, 2-е изд., М., 1973. – 347 с.

6 Вьюненко, Л.Ф. Имитационное моделирование: учебник и практикум для академического бакалавриата / Л. Ф. Вьюненко, М. В. Михайлов,; под ред. Л. Ф. Вьюненко. — М. : Издательство Юрайт, 2017. — 283 с.

7 Multiprocessing — Process-based parallelism [электронный ресурс]. Режим доступа: https://docs.python.org/3/library/multiprocessing.html (дата обращения: 14.04.21).