

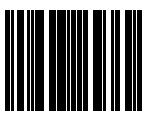
Министерство науки и высшего образования Российской Федерации
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО
ITMO University

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА
GRADUATION THESIS

Разработка архитектуры распределенной системы вычислений свёрточных
нейронных сетей

Обучающийся / Student Рождественский Никита Сергеевич
Факультет/институт/кластер/ Faculty/Institute/Cluster факультет программной инженерии и компьютерной техники
Группа/Group Р34102
Направление подготовки/ Subject area 09.03.04 Программная инженерия
Образовательная программа / Educational program Системное и прикладное программное обеспечение 2019
Язык реализации ОП / Language of the educational program Русский
Статус ОП / Status of educational program
Квалификация/ Degree level Бакалавр
Руководитель ВКР/ Thesis supervisor Быковский Сергей Вячеславович, кандидат технических наук, Университет ИТМО, факультет программной инженерии и компьютерной техники, доцент (квалификационная категория "ординарный доцент")

Обучающийся/Student

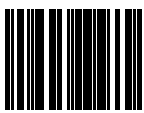
Документ подписан	
Рождественский Никита Сергеевич	
25.05.2023	

(эл. подпись/ signature)

Рождественский
Никита
Сергеевич

(Фамилия И.О./ name
and surname)

Руководитель ВКР/
Thesis supervisor

Документ подписан	
Быковский Сергей Вячеславович	
24.05.2023	

(эл. подпись/ signature)

Быковский
Сергей
Вячеславович

(Фамилия И.О./ name
and surname)

**Министерство науки и высшего образования Российской Федерации
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО
ITMO University**

**ЗАДАНИЕ НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ /
OBJECTIVES FOR A GRADUATION THESIS**

Обучающийся / Student Рождественский Никита Сергеевич

Факультет/институт/кластер/ Faculty/Institute/Cluster факультет программной инженерии и компьютерной техники

Группа/Group P34102

Направление подготовки/ Subject area 09.03.04 Программная инженерия

Образовательная программа / Educational program Системное и прикладное программное обеспечение 2019

Язык реализации ОП / Language of the educational program Русский

Статус ОП / Status of educational program

Квалификация/ Degree level Бакалавр

Тема ВКР/ Thesis topic Разработка архитектуры распределенной системы вычислений свёрточных нейронных сетей

Руководитель ВКР/ Thesis supervisor Быковский Сергей Вячеславович, кандидат технических наук, Университет ИТМО, факультет программной инженерии и компьютерной техники, доцент (квалификационная категория "ординарный доцент")

Основные вопросы, подлежащие разработке / Key issues to be analyzed

Техническое задание:

1. Выполнить сравнительный анализ имеющихся способов распределенного вычисления свёрточных нейронных сетей. Составить сравнительную таблицу. Предоставить список используемых источников.
2. Выполнить сравнительный анализ существующих видов архитектур для вычисления свёрточных нейронных сетей на кластере. Предоставить схематические изображения существующих архитектур. Составить сравнительную таблицу интерфейсов ввода-вывода, которые используют данные архитектуры.
3. Разработать 3 различных архитектуры для вычисления свёрточных нейронных сетей. Предоставить схематические изображения полученных архитектур. Сравнить теоретические возможности разработанных архитектур и предоставить результаты в виде таблицы. Выбрать лучшую архитектуру и обосновать выбор.
4. Разработать имитационную модель, позволяющую проанализировать выбранную архитектуру и выявить ее недостатки. Предоставить исходные коды алгоритмов имитации. Составить таблицу с результатами тестирования.

Исходные данные к работе:

1. Ben-Nun T., Hoeffler T. Demystifying parallel and distributed deep learning: An in-depth concurrency analysis //ACM Computing Surveys (CSUR). – 2019. – Т. 52. – №. 4. – С. 1-43.
2. Wang T. et al. FPDeep: Scalable acceleration of CNN training on deeply-pipelined FPGA clusters //IEEE Transactions on Computers. – 2020. – Т. 69. – №. 8. – С. 1143-1158.

3. Blott M. et al. Scaling neural network performance through customized hardware architectures on reconfigurable logic //2017 IEEE International Conference on Computer Design (ICCD). – IEEE, 2017. – С. 419-422.
4. Хайдарова Р. Р. и др. Модель распределенной сверточной нейронной сети на кластере компьютеров с ограниченными вычислительными ресурсами //Научно-технический вестник информационных технологий, механики и оптики. – 2020. – Т. 20. – №. 5. – С. 739-746.
5. Zhang C. et al. Energy-efficient CNN implementation on a deeply pipelined FPGA cluster //Proceedings of the 2016 International Symposium on Low Power Electronics and Design. – 2016. – С. 326-331.

Цель и задачи работы:

Цель: увеличение эффективности вычисления свёрточных нейронных сетей путем распределения расчетов на кластер микрокомпьютеров.

Поставленные задачи:

1. Анализ способов распределенного вычисления СНС.
2. Обзор существующих видов архитектур для решения задач распределенных вычислений СНС.
3. Разработка архитектуры для вычислений СНС с различными конечными вычислителями.
4. Создание имитационной модели и ее тестирование.

Перечень подлежащих разработке вопросов:

1. Зависимость загрузки сетей кластера от увеличения количества конечных вычислителей.
2. Оценка энергозатрат для вычисления свёрточной нейронной сети на кластере с различными конечными вычислителями.
3. Сравнение затраченных времени и ресурсов на кластере и персональном компьютере.

Рекомендуемые материалы и пособия для выполнения работы:

1. Dunder A., Jin J., Culurciello E. Convolutional clustering for unsupervised learning //arXiv preprint arXiv:1511.06241. – 2015.
2. Zhai S. et al. Design of convolutional neural network based on fpga //Journal of Physics: Conference Series. – IOP Publishing, 2019. – Т. 1168. – №. 6. – С. 062016.
3. Gratadour D. et al. Prototyping AO RTC using emerging high performance computing technologies with the Green Flash project //Adaptive Optics Systems VI. – SPIE, 2018. – Т. 10703. – С. 404-418.

Форма представления материалов ВКР / Format(s) of thesis materials:

пояснительная записка, презентация

Дата выдачи задания / Assignment issued on: 06.02.2023

Срок представления готовой ВКР / Deadline for final edition of the thesis 31.05.2023

Характеристика темы ВКР / Description of thesis subject (topic)

Тема в области фундаментальных исследований / Subject of fundamental research: нет / not

Тема в области прикладных исследований / Subject of applied research: да / yes

СОГЛАСОВАНО / AGREED:

--	--

Руководитель ВКР/
Thesis supervisor

Документ подписан
Быковский Сергей Вячеславович
02.05.2023

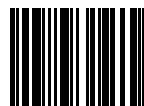


Быковский
Сергей
Вячеславович

(эл. подпись)

Задание принял к
исполнению/ Objectives
assumed BY

Документ подписан
Рождественский Никита Сергеевич
03.05.2023



Рождественский
Никита
Сергеевич

(эл. подпись)

Руководитель ОП/ Head
of educational program

(эл. подпись)

**Министерство науки и высшего образования Российской Федерации
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО
ITMO University**

**АННОТАЦИЯ
ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЫ
SUMMARY OF A GRADUATION THESIS**

Обучающийся / Student Рождественский Никита Сергеевич
Факультет/институт/кластер/ Faculty/Institute/Cluster факультет программной инженерии и компьютерной техники
Группа/Group P34102
Направление подготовки/ Subject area 09.03.04 Программная инженерия
Образовательная программа / Educational program Системное и прикладное программное обеспечение 2019
Язык реализации ОП / Language of the educational program Русский
Статус ОП / Status of educational program
Квалификация/ Degree level Бакалавр
Тема ВКР/ Thesis topic Разработка архитектуры распределенной системы вычислений свёрточных нейронных сетей
Руководитель ВКР/ Thesis supervisor Быковский Сергей Вячеславович, кандидат технических наук, Университет ИТМО, факультет программной инженерии и компьютерной техники, доцент (квалификационная категория "ординарный доцент")

**ХАРАКТЕРИСТИКА ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЫ
DESCRIPTION OF THE GRADUATION THESIS**

Цель исследования / Research goal

Увеличение эффективности вычисления свёрточных нейронных сетей путем распределения расчетов на кластер микрокомпьютеров.

Задачи, решаемые в ВКР / Research tasks

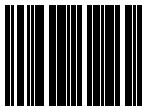
1) Способы распределенного вычисления СНС. 2) Обзор существующих видов архитектур для решения задач распределенных вычислений СНС. 3) Разработка архитектуры для вычислений СНС с различными конечными вычислителями. 4) Создание имитационной модели и ее тестирование.

Краткая характеристика полученных результатов / Short summary of results/findings

Наиболее выделяется 3 способа распределенного вычисления СНС: параллелизм данных, параллелизм слоев, параллелизм моделей. Наиболее эффективным является совмещение этих 3 парадигм. Наиболее распространён расчет СНС последовательным методом на кластере ПЛИС. Разработанная архитектура имеет вид звезды на верхнем уровне и может иметь различные конечные вычислители в рамках вычислительного блока. Была разработана система приложений, включающая в себя 5 программных компонента: HTTP клиент, HTTP сервер, локальный сервер, клиент вычислительного блока, программа для обучения нейронной сети. Так же, данная система была протестирована на комплексе аппаратных вычислителей. Данный программно-аппаратный комплекс хорошо показал

себя на обработке пула изображений размером 1000 с периодом задержек 0-10мс. Общее время вычисления 1000 изображений на кластере всего на 17,2% больше, чем на стационарном процессоре. Энергии же, для работы кластера, потребовалось в 7 раз меньше. Так же, кластер эффективен при запросах реже 1с. Процессор простаивает более 83% времени и потребляет в среднем около 7Вт, кластер же простаивает около 33% времени при потреблении около 1,5Вт.

Обучающийся/Student

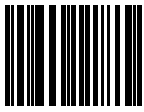
Документ подписан	
Рождественский Никита Сергеевич	
25.05.2023	

(эл. подпись/ signature)

Рождественский
Никита
Сергеевич

(Фамилия И.О./ name
and surname)

Руководитель ВКР/
Thesis supervisor

Документ подписан	
Быковский Сергей Вячеславович	
24.05.2023	

(эл. подпись/ signature)

Быковский
Сергей
Вячеславович

(Фамилия И.О./ name
and surname)

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ.....	8
1. ОБЗОР СУЩЕСТВУЮЩИХ АРХИТЕКТУР ДЛЯ РАСПРЕДЕЛЕННОГО ВЫЧИСЛЕНИЯ СВЕРТОЧНОЙ НЕЙРОННОЙ СЕТИ.....	10
1.1 Обзор классической архитектуры сверточной нейронной сети.....	10
1.2 Обзор способов параллелизма сверточных нейронных сетей.....	11
1.3 Обзор существующих архитектур и построенных на них систем.	13
2. РАЗРАБОТКА АРХИТЕКТУРЫ ДЛЯ ВЫЧИСЛЕНИЙ СНС С РАЗЛИЧНЫМИ КОНЕЧНЫМИ ВЫЧИСЛИТЕЛЯМИ	17
2.1 Выбор архитектуры верхнего уровня	17
2.2 Обзор вычислительных блоков.....	18
2.3 Минусы предложенной архитектуры.....	20
2.4 Пример применения разработанной архитектуры.....	21
3. СОЗДАНИЕ ИМИТАЦИОННОЙ МОДЕЛИ И ЕЕ ТЕСТИРОВАНИЕ.	23
3.1 Обзор используемых технологий и аппаратуры.	23
3.2 Особенности реализации сервера.....	25
3.3 Особенности реализации клиента.	27
3.4 Выполнение симуляций.....	30
3. ЗАКЛЮЧЕНИЕ	42
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ	43

ВВЕДЕНИЕ

Актуальность. За последний год нейронные сети перешли на совершенно новый уровень, относительно прошлого десятилетия. Современные инструменты с интегрированными нейронными сетями пользуются все большим спросом, начиная от графических редакторов, и доходящих до юридических помощников. При этом, потребляемые ими мощности тоже растут и для работы лучших из нейронных сетей уже требуются серьёзные сервера с большими объёмами памяти и мощными вычислителями, способными быстро пропустить сквозь себя большие объёмы данных.

Ввиду физических ограничений существующих архитектур и специфичности некоторых вычислений нейронных сетей, а в частности СНС, энергозатраты и эффективность существующих решений сравнительно низка.

Одним из возможных решений данной проблемы – распределение вычислений на несколько вычислителей, что не всегда возможно, ввиду разнообразия архитектур нейронных сетей и моделями их вычислений. Наиболее хорошо из семейства нейронных сетей распределению вычислений подвержены СНС, или части нейронных сетей и программных комплексов, которые содержат в себе СНС.

Степень теоретической разработанности темы. Идею кластера энергоэффективных вычислителей для нейронных сетей реализовывали не единожды [4], но все они были моногамными – состояли из одинаковых конечных вычислителей, пусть в некоторых случаях они выполняли разные роли, но гибкость таких систем сильно ограничена и некоторые задачи могут выполняться неэффективно, процессоры на отдельных вычислителях могут простаивать относительно других, в одних вычислителях может не хватать ОЗУ, при переизбытке ее в других частях системы и тд.

Для каждой такой разработки (кластера) создается отдельная новая архитектура со своими интерфейсами передачи данных и изменение конечных вычислителей без изменения архитектуры почти невозможно.

Разработка архитектуры, в которой появляется возможность менять конечные вычислители, подстраивая их характеристики под необходимости конкретного процесса в вычислении СНС, и ставиться задачей данной работы.

Имея готовую архитектуру, разработчики смогут более гибко использовать ее при подходе к решению своих конкретных задач, что может плодотворно сказаться на экономической составляющей проекта и его масштабируемости.

Цель и задачи исследования. Целью работы является увеличение эффективности вычисления свёрточных нейронных сетей путем распределения расчетов на кластер микрокомпьютеров.

Поставленные задачи:

- 1) Способы распределенного вычисления СНС.
- 2) Обзор существующих видов архитектур для решения задач распределенных вычислений СНС.
- 3) Разработка архитектуры для вычислений СНС с различными конечными вычислителями.
- 4) Создание имитационной модели и ее тестирование.

1. ОБЗОР СУЩЕСТВУЮЩИХ АРХИТЕКТУР ДЛЯ РАСПРЕДЕЛЕННОГО ВЫЧИСЛЕНИЯ СВЕРТОЧНОЙ НЕЙРОННОЙ СЕТИ

1.1 Обзор классической архитектуры сверточной нейронной сети

Для сравнения методов перемещения вычислений между вычислителями в кластере посмотрим на структуру СНС и попробуем предположить, какие вычисления можно разнести на разные вычислители без существенных временных задержек. На рисунке 1 показан стандартный вид архитектур свёрточных нейронных сетей.

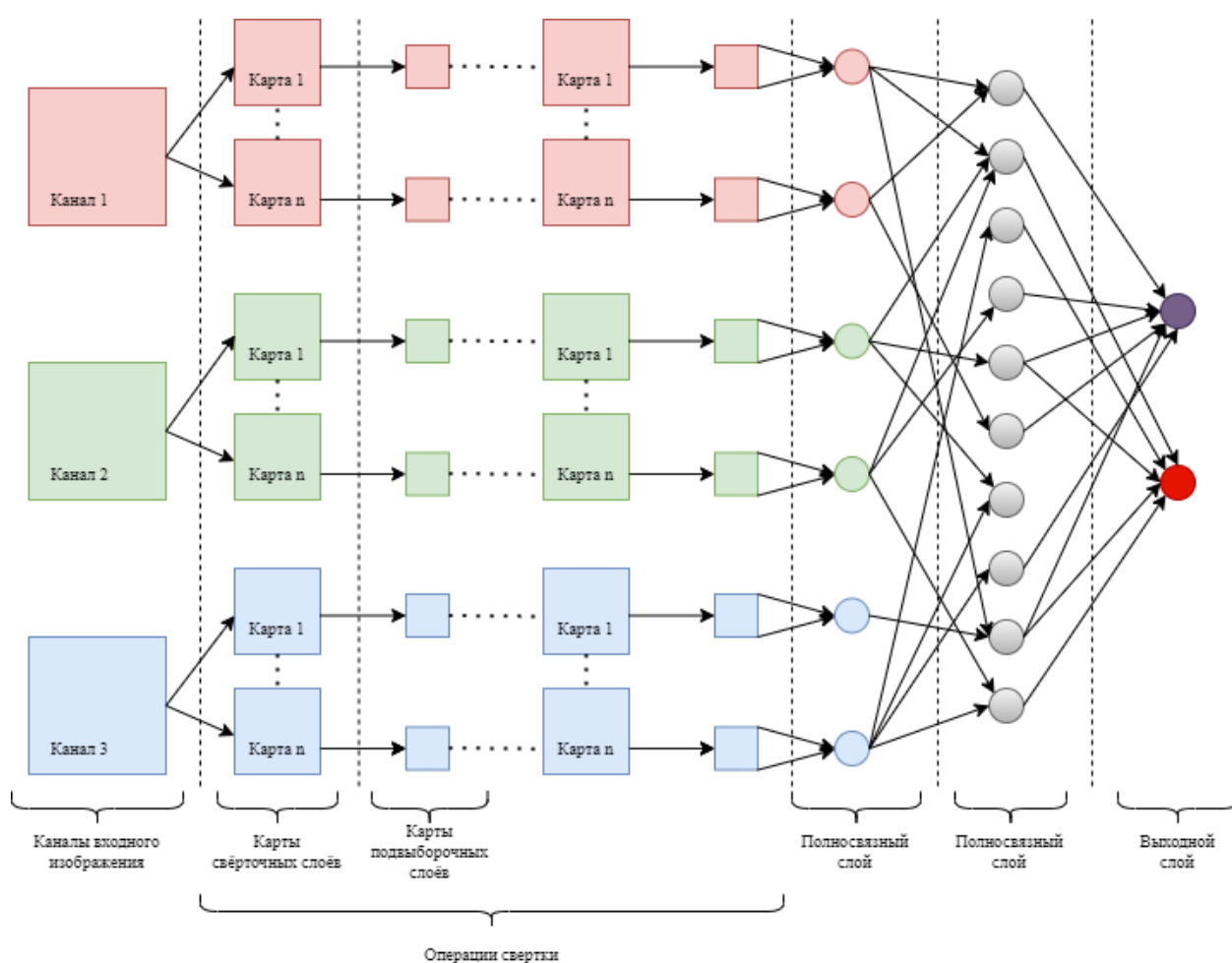


Рисунок 1. Общий вид архитектуры СНС [3],[7].

На рисунке видно, что сама концепция СНС хорошо подходит для распараллеливания, начиная от параллельной обработки разных каналов изображения до разбора карт слоев на части и вычисления их на разных

кристаллах. Так же, методы вычисления СНС постоянно улучшают, что снижает требования к вычислителям [6].

1.2 Обзор способов параллелизма сверточных нейронных сетей

На текущий момент наиболее известны 3 способа распараллеливания, хорошо описанные в [1]. Схематичное изображение показано на рисунке 2.

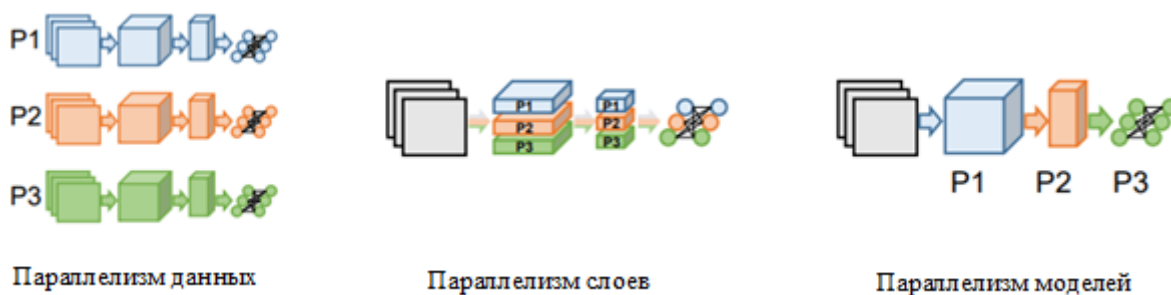


Рисунок 2. Схематичное изображение видов параллелизма СНС.

- 1) Параллелизм данных: в СНС конфигурации каждого уровня (размер ядра, размер пула и размер шага) сильно различаются, что требует различных конфигураций оборудования для достижения оптимальной производительности. Таким образом, ПЛИС необходимо перенастраивать между уровнями, что приводит к значительным накладным расходам. Кроме того, поскольку каждая ПЛИС выполняет все уровни в последовательном порядке, каждый уровень запускается только после завершения предыдущего уровня. Для всех промежуточных функций веса должны быть сохранены на хосте и загружены с него после завершения слоя, что приводит к интенсивному трафику внешней памяти;
- 2) Параллелизм слоев: сопоставляет слои СНС с отдельными узлами и конвейерами вычислений. Он использовался как в графических процессорах, так и в фреймворках FPGA. Во втором случае каждый слой назначается отдельному процессору. Поскольку каждый уровень сопоставляется с определенным процессором, рабочие нагрузки между

устройствами не распределяются. Рабочая нагрузка несбалансированная, и существует интенсивная связь с внешней памятью. Таким образом, в то время как параллелизм слоев смягчает некоторые проблемы, связанные с загрузкой ПЛИС и частой реконфигурацией, он страдает от других проблем: балансировка нагрузки и зависание, когда одни узлы ждут завершения других;

- 3) Параллелизм моделей: эта стратегия разделяет работу в соответствии с нейронами в каждом слое. В этом случае образец мини-пакета копируется на все процессоры, а разные части СНС вычисляются на разных процессорах, что позволяет экономить память (поскольку вся сеть не хранится в одном месте), но требует дополнительной связи после каждого слоя.

Для наглядности сведем аналитическое сравнение этих способов в условных единицах из [1] в таблицу 1.

Таблица 1. Сравнение архитектур параллелизма при применении к ПЛИС.

	Способ	Параллелизм	Память	Нагрузка на сеть
1	Параллелизм данных	2	2	4
2	Параллелизм слоев	1	3	2
3	Параллелизм моделей	3	1	1
4	Смешанный	4	4	3

1.3 Обзор существующих архитектур и построенных на них систем.

В открытых источниках существует несколько вариантов архитектур для распределенного вычисления СНС. В большей части из них конечным вычислителем является ПЛИС.

На рисунке 3 можно увидеть изображение типичной архитектуры построения кластера для вычислений СНС на ПЛИС. В данном случае, на каждом ПЛИС организовано вычисления целого количества слоев (1 или больше). После вычисления очередного слоя вся нужная информация передается на следующий ПЛИС по высокоскоростной линии данных (в данном случае - линии SATA). Управляет конфигурацией и распределением мощностей центральный ПК, через интерфейсы JTAG и SPI. Главной проблемой данной архитектуры является ограниченный размер слоя, который можно будет уместить на 1 ПЛИС. Другим узким местом является пропускная способность SATA.

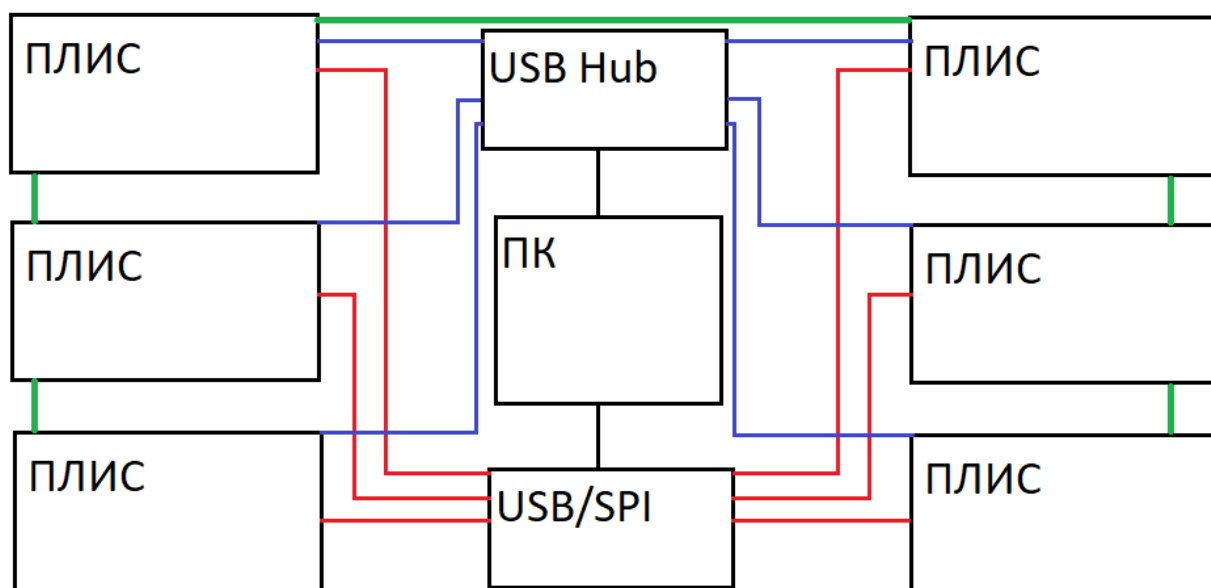


Рисунок 3. Архитектура, использующая ПЛИС как конечные вычислители. Красными линиями изображены линии SPI, зелеными - SATA, синими – JTAG [8].

Модификацией предыдущей архитектуры является изображенная на рисунке 4. В данной архитектуре появляется возможность вычислять один слой СНС на нескольких, подряд идущих ПЛИС. Данное решение позволяет решить проблемы роста до определенного этапа, но и повышает требования к пропускной способности сети, соединяющей разные ПЛИС. Поэтому основной шиной данных здесь является FMC-НРС.

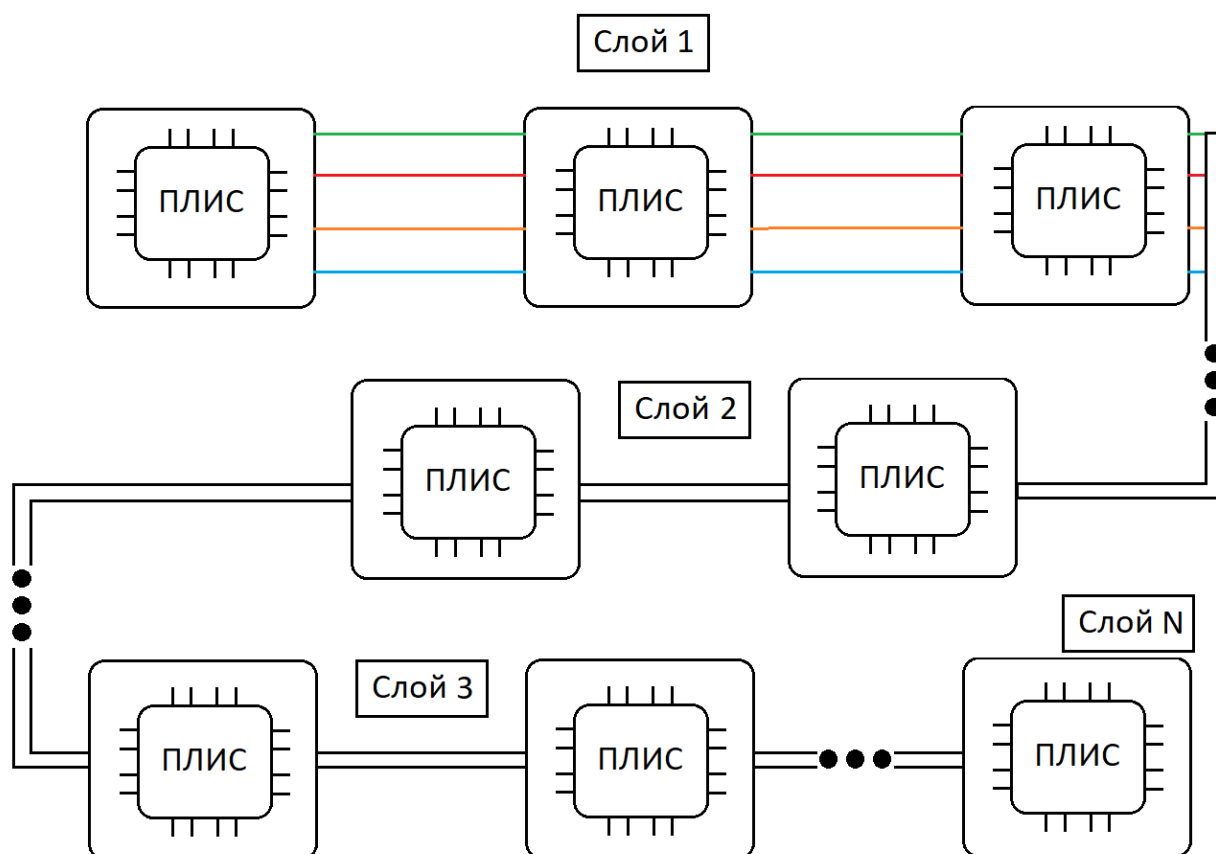


Рисунок 4. Архитектура с вычислением 1 слоя на последовательных ПЛИС [2].

Существуют и другие модификации этой “Последовательной” архитектуры, использующие другие интерфейсы передачи данных, или модифицированную и оптимизированную микроархитектуру, но главный принцип остается тот же.

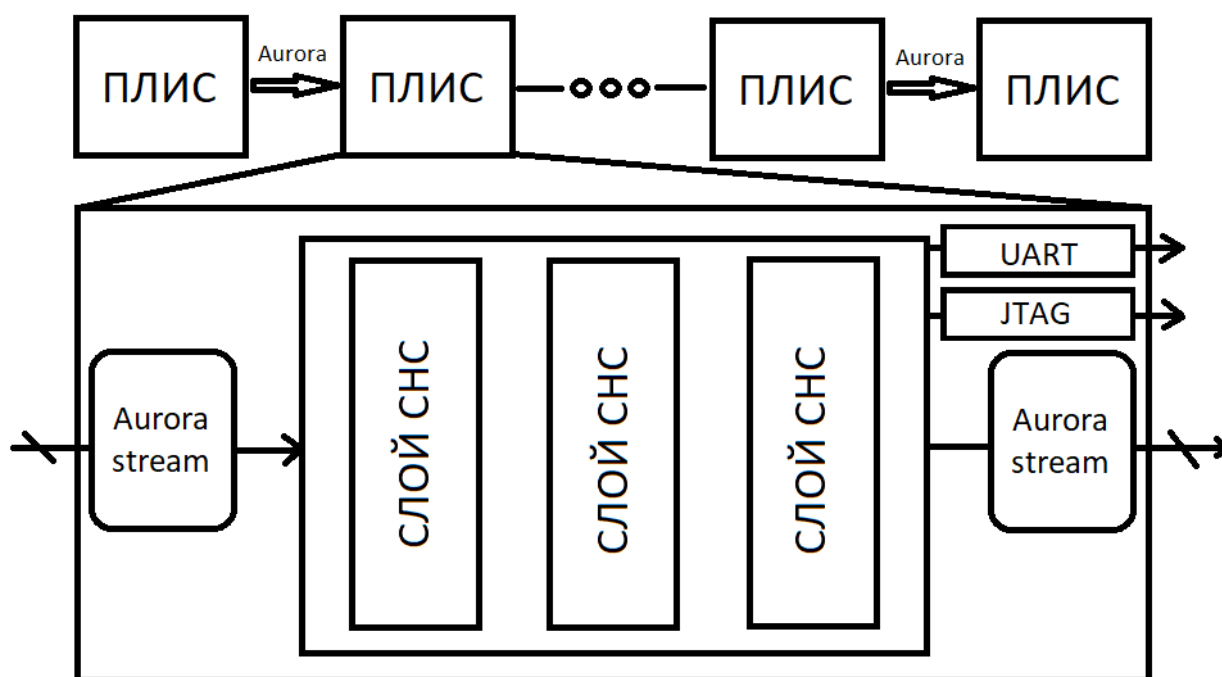


Рисунок 5. Последовательная архитектура, построенная на проприетарном протоколе Xilinx Aurora [5].

Если свести данные о интерфейсах передачи, использующихся в подобных последовательных кластерах, получим следующую таблицу (таблица 2)

Таблица 2. Сравнение существующих интерфейсов передачи для кластеров с ПЛИС.

Интерфейс	Количество линий	Максимальная скорость передачи по линии	Необходимые требования для использования
SATA v3	1	До 6 Гбит/с	Плата расширения
PCIe v3	4-8-16	До 2 Гбайт/с	Поддержка платой ПЛИС
FMC	160-400	До 10 Гбит/с	Поддержка платой ПЛИС
Fast Ethernet	1	До 100 Мбит/с	Плата расширения
Gigabit Ethernet	1	До 1000 Мбит/с	Плата расширения
Aurora 64/66b protocol	1	До 100 Гбит/с	Поддержка платой ПЛИС

Можно заметить, что для объединения компьютеров в кластеры в основном используются стандартные интерфейсы передачи, не связанные с нейросетями или ПЛИС. С одной стороны это упрощает архитектуру и ее использование, с другой же, неспециализированный интерфейс может повышать накладные расходы на нормализацию данных и их преобразование в пакетный формат. В энергоэффективных системах, например, такие расходы, зачастую, становятся неприемлемыми.

2. РАЗРАБОТКА АРХИТЕКТУРЫ ДЛЯ ВЫЧИСЛЕНИЙ СНС С РАЗЛИЧНЫМИ КОНЕЧНЫМИ ВЫЧИСЛИТЕЛЯМИ

2.1 Выбор архитектуры верхнего уровня

Рассматривая наиболее распространённые методы создания кластеров, которые я показал ранее, наиболее разумной мне показалась идея воспользоваться диалектикой Гегеля и попытаться объединить обе эти концепции.

Как уже говорилось, главная проблема последовательного метода вычисления СНС на ПЛИС – это плохая масштабируемость и низкая степень параллелизма. Зачастую, параллельно вычислять можно только разные каналы цветов изображения (обычно их 3), и до тех пор, пока свёрточные слои не переходят в полносвязные. После этого количество связей между разными слоями (особенно в больших сетях) переходит порог эффективности расчета сети на кластере. Нагрузка на сеть становится большой и задержки передачи сигналов между нейронами существенно замедляют процесс вычисления и эффект ускорения вычислений исчезает, а следовательно, пропадает смысл распределенного вычисления.

С другой стороны, используя последовательный метод с ПЛИС, мы можем существенно ускорить вычисления каких-либо приоритетных целей, при относительно небольших энергетических затратах. Например, у нас есть некая система реального времени и ей требуется обработанный результат с камер видеонаблюдения раз в 1 секунду. Мы можем рассчитать точное количество времени, за которое кластер последовательных ПЛИС обработает один кадр и сделать приоритетную очередь на эту часть сети.

Поэтому, я предлагаю сделать кластер топологии звезда, в котором вместо конечного вычислителя на вершине будут вычислительные блоки с разным функционалом и набором конечных вычислителей соответственно.

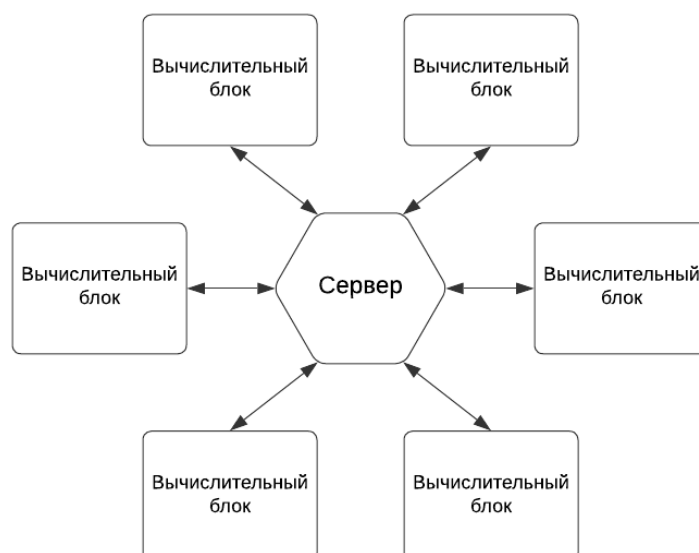


Рисунок 6. Разработанная архитектура верхнего уровня.

На рисунке представлено схематичное изображение архитектуры верхнего уровня. Центральным узлом является сервер, который управляет распределением поступающих на него задач.

В приоритетные обязанности сервера входит:

- Выдача приоритетов обработки.
- Создание очередей.
- Распределение задач в очереди на подключенные вычислительные блоки.
- Отправка задач в вычислительные блоки.
- Прием ответов от вычислительных блоков.

2.2 Обзор вычислительных блоков

Так, как в вычислительных блоках предполагается использовать разные конечные вычислители, то и единой архитектуры вычислительного блока составить невозможно. Но каждый вычислительный блок должен содержать некоторые элементы для возможности встраивания основную архитектуру.

В перечень этих элементов входит:

- Компьютер клиент, который обменивается информацией с сервером
- Набор конечных вычислителей, которые занимаются вычисление подготовленных данных от компьютера клиента.

В общем можно выделить 2 группы вычислительных блоков в зависимости от конечного вычислителя:

- Содержащие ПЛИС
- Использующие стандартные процессоры

На следующих рисунках предоставлены примеры вычислительных блоков

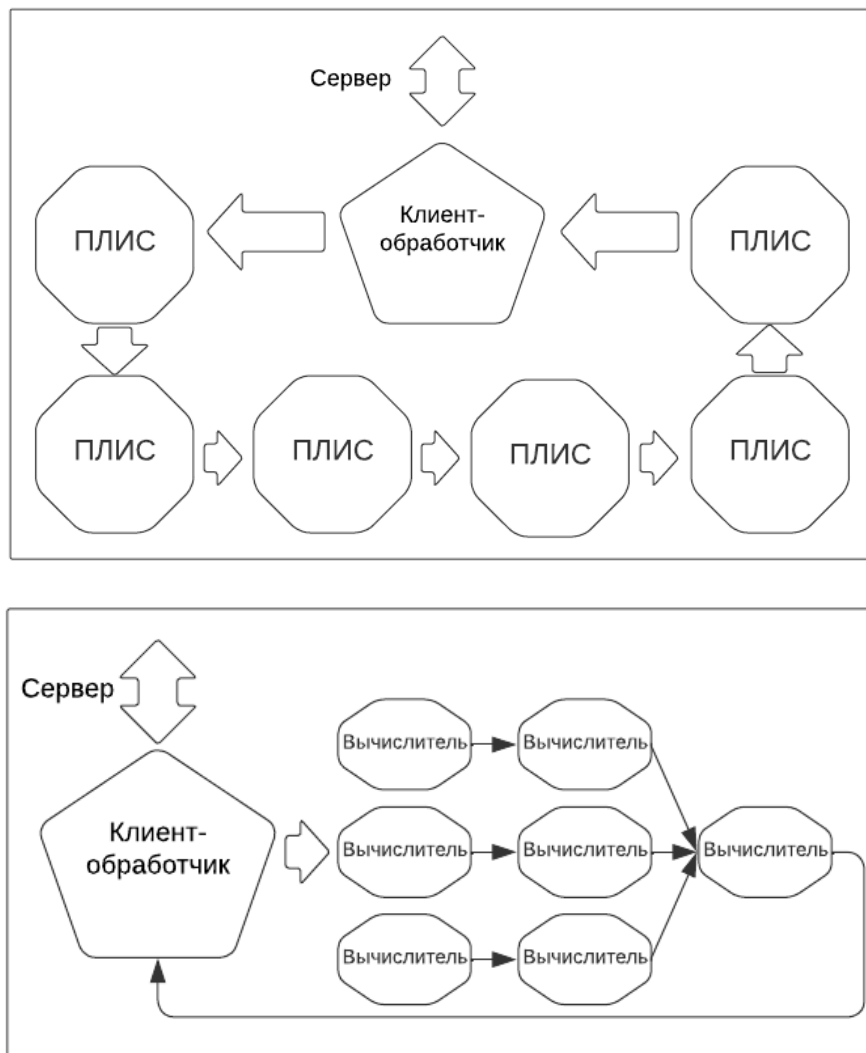


Рисунок 7. Примеры архитектур вычислительных блоков.

В обязанности клиента входит:

- Прием задачи от сервера
- Подготовка данных к вычислению*
- Отправка данных на подконтрольные конечные вычислители
- Получение ответа и отправка его на сервер

* В зависимости от конечного исполнителя подготовка данных может сильно отличаться. Для последовательного кластера ПЛИС необходимо преобразовать изображение в массив нормированных данных. Для кластера микрокомпьютеров – разделить изображения на каналы и тд.

2.3 Минусы предложенной архитектуры

Перед тем, как переходить к конкретике, я попытался определить основные минусы данной теоритической модели, которые уже можно видеть на данном этапе проектирования.

Главным минусом является количество компьютеров, необходимых для управления. В каждом вычислительном блоке будет присутствовать минимум 1 компьютер, который будет выполнять сугубо распределительную роль и не будет заниматься конкретно вычислением СНС.

Эта проблема хорошо заметна на малых мощностях, когда в каждом вычислительном блоке сравнительно немного конечных вычислителей и каждый управляющий компьютер является заметным энергопотребителем общей системы. Соответственно, чем больше вычислительный блок и мощнее конечные вычислители, тем меньше влияния управляющий компьютер оказывает на общее потребление системы.

Сразу появляется вопрос, почему бы не использовать ресурсы управляющего компьютера для одного из этапов вычисления СНС? Для малого размера вычислительного блока, где управляющий компьютер может простаивать, мы можем позволить использовать его ресурсы для вычисления

СНС, хоть это и будет нарушать принцип единой ответственности. Но предполагается, что для управляющего компьютера будет выбираться сравнительно слабый вычислитель, главным требованием для которого будет служить наличие сетевого интерфейса для общения с сервером. Операции преобразования изображений, даже высокого разрешения, требуют небольших мощностей, сравнительно с расчетом СНС.

2.4 Пример применения разработанной архитектуры

Минимальная реализация разработанной архитектуры представляется мне следующим образом. Сервер состоит из 1 компьютера, на котором работают 2 приложения:

- HTTP сервер – принимает запросы извне, сохраняет данные из запроса в хранилище и передает запрос на локальный сервер.
- Локальный сервер – обрабатывает запросы от HTTP сервера, создает очереди выполнения на основе приоритетов и распределяет эти запросы по имеющимся вычислительным блокам.

Локальный и HTTP сервера могут являться частями одного приложения, но задачи, которые они исполняют не должны перемешиваться друг с другом. На рисунке 8 представлен пример реализации данной архитектуры на кластере в минимально возможном виде. Запросы поступают из глобальной сети и, проходя через маршрутизатор, попадают на веб сервер. Веб сервер сохраняет данные на локальное хранилище, составляет запрос и отправляет его на локальный сервер. Локальный сервер ждет свободного исполнителя в виде клиента вычислительного блока и отправляет ему данные на вычисление через тот же маршрутизатор. Таким образом вычислительные блоки могут быть

логически отделены от внешней сети без дополнительного оборудования путем использования файрволов на маршрутизаторе.

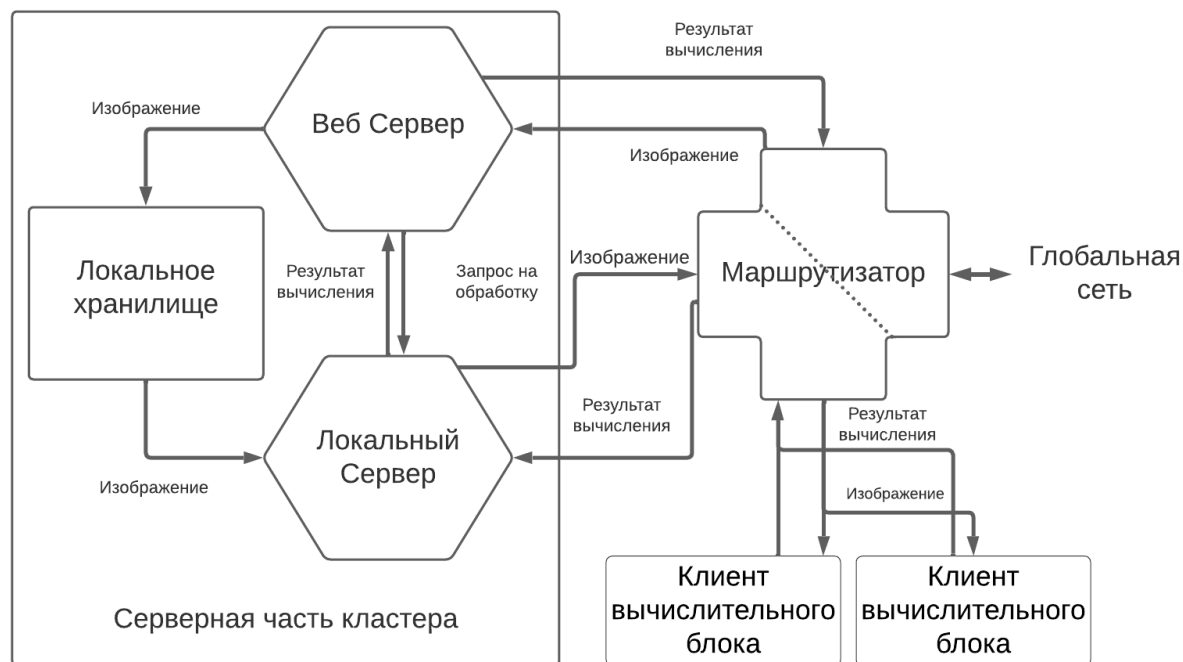


Рисунок 8. Пример реализации разработанной архитектуры.

Данный пример не использует полные возможности архитектуры, но довольно хорошо показывает возможности к ее масштабированию. Для минимальной его реализации в железе нет необходимости в дорогостоящем оборудовании или программном обеспечении. Сравнительно легко можно реализовать горячую замену вычислительных блоков, или расширить их количество.

3. СОЗДАНИЕ ИМИТАЦИОННОЙ МОДЕЛИ И ЕЕ ТЕСТИРОВАНИЕ.

3.1 Обзор используемых технологий и аппаратуры.

В качестве архитектуры имитационной модели я использовал версию кластера из рисунка 8. Единственное упрощение – в качестве вычислительного блока использовался только 1 микрокомпьютер – клиент.

В качестве HTTP сервера я решил использовать SpringBoot-2.0[9]. Его легко можно будет запустить на множестве устройств, он сравнительно прост, имеет большой функционал и может быть легко масштабирован.

Локальный сервер интегрирован в HTTP сервер посредством Spring Beans. Он выполняется в отдельном потоке. Для общения с клиентом локальный сервер использует Linux TCP Socket. Каждый сеанс общения посредством сокетов выполняется в отдельном потоке, все необходимые операции синхронизированы. Изображения пересылаются на клиент посредством Linux команды SCP

Клиент представляет собой однопоточное приложение и использует Python-3 и TensorFlow-2.12.0[10].

Характеристики компьютера, на котором запущено серверное приложение:

- Процессор - AMD Ryzen 5 3600.
- ОЗУ - 32Гб DDR4 3200МГц.
- Накопитель - SSD M.2 Kingston NV1 3D NAND
- Сеть - RJ45 2.5 Гбит/с

Микрокомпьютер, на котором запускается клиент – ROCK PI S[11].

- Процессор - Rockchip RK3308 1,3ГГц Quad A35 64bit
- ОЗУ - 512MB DDR3
- Накопитель - 32Gb microSDHC.

- Сеть - RJ45 100 Мбит



Рисунок 9. Микрокомпьютер Rock Pi S.

Коммутатор -TP-link LS108G, 8 портов 10/100/1000 Мбит/с



Рисунок 10. Коммутатор -TP-link LS108G.

3.2 Особенности реализации сервера.

Задачи на сервер приходят через POST запрос “/addRequest”. Каждый запрос состоит из следующих параметров:

- “username” - имя пользователя.
- “priority” - приоритет запроса.
- “file” - изображение для распознавания.
- “dataType” - формат изображения.

Метод, обрабатывающий запрос, выполняет следующую последовательность действий:

1. Генерирует уникальный в данной сессии идентификатор.
2. Создает директорию пользователя, если это первый запрос от него.
3. Сохраняет переданное изображение под именем сгенерированного идентификатора.
4. Создает объект Request с параметрами запроса.
5. Передает этот объект в локальный сервер.
6. Возвращает сообщение с идентификатором, по которому можно будет получить ответ.

Данный сервер является однопоточным и обрабатывает по одному запросу за раз. В его задачи входит только общение с клиентами и передача данных локальному серверу, в рамках имитационной модели ему не требуется несколько потоков для обработки клиентов.

Основная нагрузка кладется на локальный сервер. Он отвечает за:

- Организацию приоритетных очередей
- Общение с клиентами
- Распределение задач по вычислительным блокам
- Передачи файлов клиентам

Локальный сервер инициализируется вместе с HTTP сервером и работает в отдельном потоке. Он создает объект `ServerSocket`, на котором происходят попытки подключения от клиентов. Алгоритм подключения и передачи задачи выглядит следующим способом:

1. `ServerSocket` принимает запрос на подключение от клиента и создает `clientSocket`, что является объявлением вычислительного блока, что он готов к обработке следующего изображения.
2. Локальный сервер создает и запускает объект `ClientHandler` в отдельном потоке при помощи `ThreadPoolExecutor`. Таким образом создается пул потоков, которые можно будет переиспользовать для следующих запросов, что повышает производительность приложения.

Далее, общение с клиентом происходит в отдельном потоке объектом `ClientHandler` следующим образом:

1. `ClientHandler` пытается получить от локального сервера свободную задачу из очереди задач.
2. Запросы повторяются до появления свободной задачи через время таймаута.
3. После получения задачи `ClientHandler` пытается копировать изображение посредством команды `SCP` на клиент вычислительного блока.
4. При успешном копировании `ClientHandler` отправляет имя файла через `TCP` сокет клиенту вычислительного блока и ждет ответа с результатом обработки.
5. `ClientHandler` создает объект `Answer` и кладет его в `HashMap` по ключу `Id`.
6. Сохраняет различные счетчики в лог файл.
7. Цикл повторяется с 1 го пункта.

Задача по хранению запросов и организации операций по взаимодействию с ними лежит на классе `QueuesController`. Он инициализируется вместе с локальным сервером и создает список приоритетных очередей в соответствии с конфигурацией. Приоритетные списки используют механизм FIFO и не являются блокирующими, но операции по взятию и вставке синхронизированы так, что блокируют взаимодействие с очередями, пока их использует какой-либо поток.

Для очередей определены 2 основные операции:

- `getNextRequest` - проходит по списку очередей от высшего приоритета к низшему и пытается взять первый объект `Request` из очереди. Если ни в одной из очередей нет объектов, метод возвращает `null`.
- `addRequest` - вставляет запрос в конец очереди по приоритету.

Объекты `queuesController` и `localServer` являются Spring `@Component` и создаются в единственном экземпляре на все запросы.

3.3 Особенности реализации клиента.

В рамках симуляции работы вычислительного блока, клиент запускает вычисления сверточной нейронной сети на той же машине, на которой работает сам (в данном случае это микрокомпьютер Rock Pi S). Сама СНС обучена на компьютере на процессоре AMD Ryzen 5 3600. Структура модели СНС показана на рисунке 13 и имеет 8 слоев, из которых 3 полносвязных. Модуль обучена на базе данных MNIST[12](70000 изображений) обучалась 40 эпох и имела `batch_size` размером 64. Зависимость точности классификации от эпохи обучения показана на рисунке 12. Пример изображения из датасета представлен на рисунке 11. Обученная модель сохраняется в

формате SavedModel (создается директория с файлами весов, конфигураций и метаданных модели).

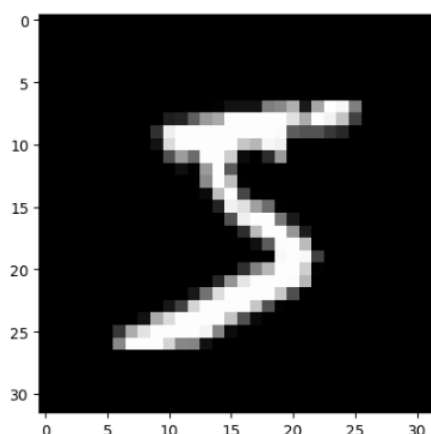


Рисунок 11. Пример изображения из датасета [12].

Клиент является однопоточным python3 приложением и работает следующим образом:

1. Загружает обученную модель посредством `tensorflow.keras.models.load_model`
2. Подключается к локальному серверу по локальному адресу с помощью linux сокетов.
3. Ждет пока на локальном сервере появиться задача и сервер отдаст ее на выполнение.
4. Получает задачу (в данном случае имя файла), открывает изображение из хранилища.
5. Подготавливает изображение к распознаванию
6. Распознает изображение
7. Отправляет результат на сервер при помощи сокет.
8. Удаляет изображение из хранилища (память сравнительно ограничена).
9. Цикл повторяется с 3 пункта

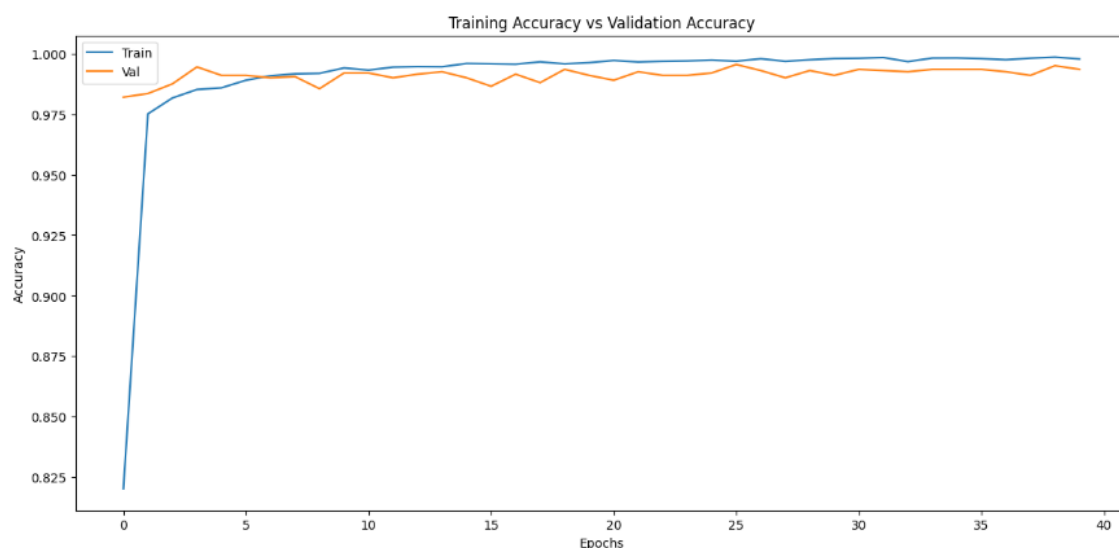


Рисунок 12. Зависимость точности вычисления от количества эпох.

Layer (type)	Output Shape	Param #
resizing (Resizing)	(None, 224, 224, 3)	0
conv2d (Conv2D)	(None, 56, 56, 96)	34944
lambda (Lambda)	(None, 56, 56, 96)	0
activation (Activation)	(None, 56, 56, 96)	0
max_pooling2d (MaxPooling2D)	(None, 27, 27, 96)	0
conv2d_1 (Conv2D)	(None, 7, 7, 256)	614656
lambda_1 (Lambda)	(None, 7, 7, 256)	0
activation_1 (Activation)	(None, 7, 7, 256)	0
max_pooling2d_1 (MaxPooling2D)	(None, 3, 3, 256)	0
conv2d_2 (Conv2D)	(None, 1, 1, 384)	885120
activation_2 (Activation)	(None, 1, 1, 384)	0
conv2d_3 (Conv2D)	(None, 1, 1, 384)	1327488
activation_3 (Activation)	(None, 1, 1, 384)	0
conv2d_4 (Conv2D)	(None, 1, 1, 256)	884992
activation_4 (Activation)	(None, 1, 1, 256)	0
flatten (Flatten)	(None, 256)	0
dense (Dense)	(None, 1024)	263168
dropout (Dropout)	(None, 1024)	0
dense_1 (Dense)	(None, 10)	10250
Total params: 4,020,618		
Trainable params: 4,020,618		

Рисунок 13. Структура модели СНС.

3.4 Выполнение симуляций.

Для тестирования Я создал разные конфигурации кластера. Полный кластер можно увидеть на рисунке 14. Уменьшение конечных вычислителей выполнялось отключения нужных плат от сети питания.



Рисунок 14. Собранный кластер, в работе.

Так же, Я разработал HTTP клиент для генерации запросов с настраиваемыми параметрами нагрузок.

Среди отслеживаемых параметров находятся:

- $T(p)$ - среднее время от отправки запроса с изображением до получения результата вычисления (измеряется на клиенте) (мс).
- $T(k)$ - среднее время копирования файла на клиент вычислительного блока(мс).
- $T(o)$ - среднее время ожидания клиентом свободной задачи(мс).
- $T(v)$ - среднее время выполнения задачи на клиенте вычислительного блока с пересылкой(мс).
- W - среднее потребление энергии (Вт).
- S - средняя скорость передачи данных (Мбит/с).

- ОЗУ - максимальный Объем памяти, потребляемым приложением (Мбайт).
- С - Загрузка CPU (%).
- Nrand(0-X) – случайное нормально распределенное число на отрезке [0:X]

Конфигурация-1:

- 1 конечный исполнитель - ПК,
- Процессор - AMD Ryzen 5 3600.
- ОЗУ - 32Гб DDR4 3200МГц.
- 1000 Запросов
- Среднее время вычисления на клиенте ~ 17мс

Таблица 3. Результаты тестирования конфигурации 1 при 1 HTTP клиенте.

Среднее время между запросами (мс)	T(п) (мс)	T(к) (мс)	T(о) (мс)	T(в) (мс)	W (Вт)	S (Кбит/с)	ОЗУ (Мбайт)	С (%)
0	39,7	1.1	1.32	37.3	15-20	0	633	15.6
10	40.2	1.22	10.11	39,4	15-20	0	620	13.8
100	56.1	1.7	102,1	52.7	10-15	0	604	7.1
1000	57.7	1.7	1002.3	54.2	5-10	0	604	1.33
Nrand (0-1000)	56.1	1.7	492.5	52.9	5-15	0	633	4.2

В данном случае HTTP клиент, сервер и клиент вычислительно блока находятся на одной машине, поэтому сеть не нагружена. Так же, потребление процессора измерялось утилитой powertop, поэтому колеблется в определенных рамках и довольно неточно.

Из таблицы видно, что время вычисления для HTTP клиента в 2-3 раза выше, чем для клиента вычислительного блока, в связи с накладными операциями общения и передачи файлов. Тестовое изображение подготавливается к обработке на клиенте и сжимается до квадрата со стороной 32 пикселя, что тоже занимает вычислительные мощности клиента.

Так же видно, что существует ненулевое время простоя, значит HTTP клиент не может полностью нагрузить вычислительный блок, и он простаивает без задач.

Таблица 4. Результаты тестирования конфигурации 1 при 4 HTTP клиентах.

Среднее время между запросами (мс)	T(п) (мс)	T(к) (мс)	T(о) (мс)	T(в) (мс)	W (Вт)	S (Кбит/с)	ОЗУ (Мбайт)	C (%)
0	201,6	1,4	0	49,7	20-30	0	780	16
10	169,7	1,2	0	44,6	20-25	0	774	17
100	65,4	1,2	0	40,5	15-20	0	720	15
500	53,4	1,1	99,9	43,6	5-10	0	620	6.5
1000	56.024	1,2	218,5	44,1	5-10	0	630	4
Nrand (0-1000)	61.6	1,5	100,8	47,3	20-30	0	700	7.5

4 клиента уже могут нагрузить вычислительный блок, чтобы он работал без простаивания задач. Потребления процессора на низком времени между запросами заметно выросло. Но также выросло время ожидания для клиентов. Оно приходит к предыдущим измерениям только на времени

между запросами 100мс и выше. В основном это связано с тем, что клиент вычислительного блока – однопоточное приложение и не может загрузить все ядра процессора. Это так же видно по низкой загрузке процессора. В целом, вычислять изображения малого размера на нейронной в однопоточном приложении на процессоре – несостоятельная идея ввиду большие накладные расходы вычислительных мощностей.

Конфигурация-2:

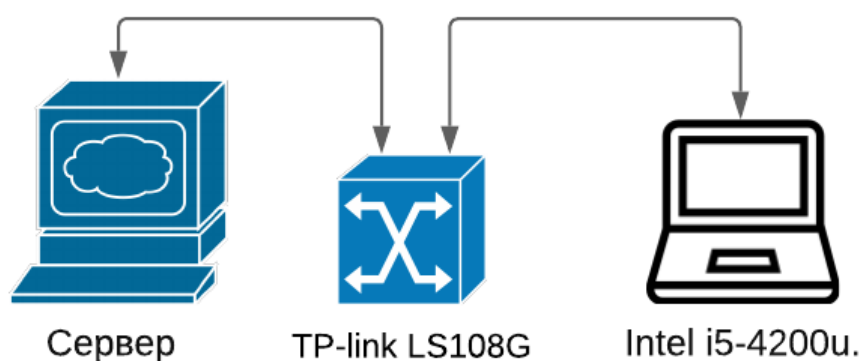


Рисунок 15. Схематическое изображение конфигурации 2.

- 1 конечный исполнитель - ПК,
- Процессор - Intel i5-4200u.
- ОЗУ - 8Гб DDR3 1333МГц.
- 1000 Запросов
- среднее время вычисления на клиенте ~ 45мс

Таблица 5. Результаты тестирования конфигурации 2 при 1 HTTP клиенте.

Среднее время между запросами (мс)	T(п) (мс)	T(к) (мс)	T(о) (мс)	T(в) (мс)	W (Вт)	S (Кбит/с)	ОЗУ (Мбайт)	C (%)
0	253.09	184,74	2,07	69,1	5	30	557	8
100	385.75	248,01	101,26	149,8	3.8	26	528	6

Так же, я решил запустить клиент вычислительного блока на сравнительно маломощном ноутбуке. Время на пересылку изображения и общения с ним сильно перевешивает время самого вычисления (почти в 5 раз), то есть процессор простаивает 80% времени. При этом потребления сети довольно мало.

Конфигурация-3:

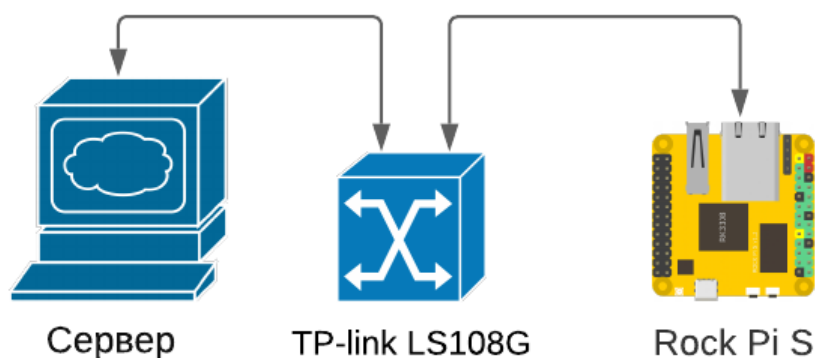


Рисунок 16. Схематическое изображение конфигурации 3.

- 1 конечный исполнитель - ROCK PI S.
- Процессор - Rockchip RK3308 1,3ГГц Quad A35 64bit.
- ОЗУ - 512MB DDR3.
- 100 Запросов.
- среднее время вычисления на клиенте ~ 320мс
- В простое система потребляет 0.15Вт±0.05Вт

Таблица 6. Результаты тестирования конфигурации 3 при 1 HTTP клиенте.

Среднее время между запросами (мс)	T(п) (мс)	T(к) (мс)	T(о) (мс)	T(в) (мс)	W (Вт)	S (Кбит/с)	ОЗУ (Мбайт)	C (%)
0	1374.06	440,44	1,8	826,9	0.24	16	350	26

10	1258.4	450,1	11,94	834	0.25	16	335	23
100	1236.05	492	97,4	832,04	0.23	14	330	18.75
500	1184.7	475,6	490,1	823,5	0.18	11	333	19
1000	1298.15	505,95	952,5	857,9	0.15	7	330	8.9
Nrand (0-1000)	1299.85	522,55	533,75	839,8	0.22	13	334	15

Под постоянными нагрузками время простоя почти равно 0, поэтому я не вижу смысла тестировать на 4 HTTP клиентах. Время выполнения для клиента так же примерно в 4 раза больше, чем для вычислительного блока. В целом логично, что система из 1 энергоэффективного исполнителя пока что нежизнеспособна.

Конфигурация-4:

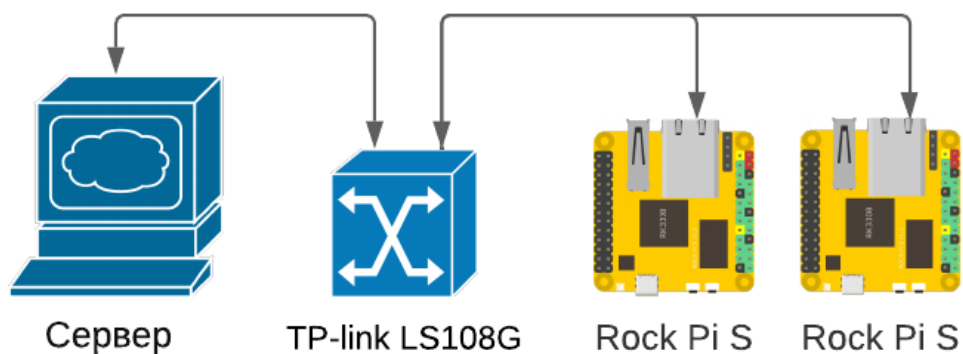


Рисунок 17. Схематическое изображение конфигурации 4.

- 2 конечных исполнителя - ROCK PI S.
- Процессор - Rockchip RK3308 1,3ГГц Quad A35 64bit.
- ОЗУ - 512MB DDR3.
- 200 Запросов.
- среднее время вычисления на клиенте ~ 320мс

- В простое система потребляет меньше $0.4\text{Вт} \pm 0.1\text{Вт}$

Таблица 7. Результаты тестирования конфигурации 4 при 1 HTTP клиенте

Среднее время между запросами (мс)	T(п) (мс)	T(к) (мс)	T(o) (мс)	T(в) (мс)	W (Вт)	S (Кбит/с)	ОЗУ (Мбайт)	С (%)
0	1157.5	388,32	1692,3	812,4	0.75	18	334	15.75
100	1352.0	433,6	697,5	962,07	0.71	14	336	13.24
500	1165.8	379,9	1941,1	792,3	0.63	12	340	8
1000	1284.9	423,02	2513,8	812,8	0.54	8	337	5
Nrand (0-1000)	1288.4	429,8	2550	814,8	0.59	10	338	10.74

При использовании 2х вычислительных блоков и 1 HTTP клиенте появляется сравнительно большой простой системы, причем заметно больше среднего времени между 2мя запросами. Это происходит так же из-за сравнительно больших накладных расходов на передачу файлов и сообщений. При этом среднее потребление энергии заметно возрастает по сравнению с 1 вычислительным блоком.

Таблица 8. Результаты тестирования конфигурации 4 при 4 HTTP клиентах.

Среднее время между запросами (мс)	T(п) (мс)	T(к) (мс)	T(o) (мс)	T(в) (мс)	W (Вт)	S (Кбит/с)	ОЗУ (Мбайт)	С (%)
0	2357.2	419,9	0	914,7	1.1	45	335	28

100	2259.7	413,8	0	815,3	1.05	43	337	24.75
500	1865.8	407,3	0	821,3	0.92	36	328	21.5
1000	1475.65	426,8	0	810,7	0,8	25	333	15
2000	1262.0	458,3	368,7	812,8	0.63	16	337	12.75
3000	1213.3	420,1	717	815,1	0.5	10	334	9.8
5000	1435.85	487,3	859,8	829	0.7	18	339	16.5

Такая конфигурация вычислительных блоков и HTTP клиентов уже неплохо себя показывает. Простой системы начитается только при времени задержек более 2 секунд, при этом, оптимальное отношение загрузки кластера, потреблении энергии и времени между запросами находится в районе 1900-2100мс.

Конфигурация-5:

- 4 конечных исполнителя - ROCK PI S.
- Процессор - Rockchip RK3308 1,3ГГц Quad A35 64bit.
- ОЗУ - 512MB DDR3.
- 200 Запросов.
- среднее время вычисления на клиенте ~ 320мс
- В простое система потребляет меньше 0.8Вт±0.2Вт

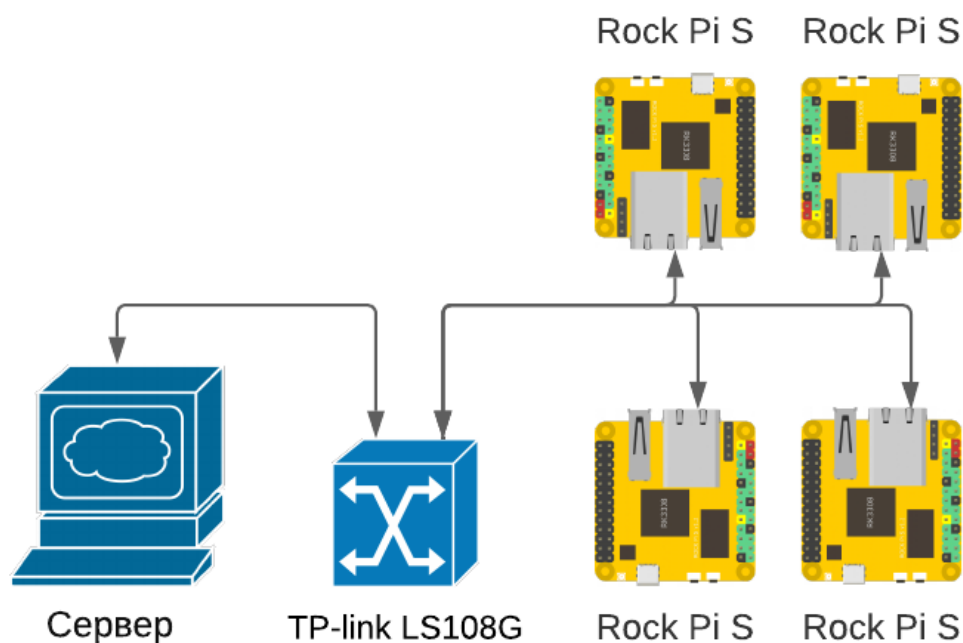


Рисунок 18. Схематическое изображение конфигурации 5.

Таблица 9. Результаты тестирования конфигурации 5 при 1 HTTP клиенте.

Среднее время между запросами (мс)	T(п) (мс)	T(к) (мс)	T(о) (мс)	T(в) (мс)	W (Вт)	S (Кбит/с)	ОЗУ (Мбайт)	C (%)
0	1353.69	424	3119.1	818.5	1.21	12.9	334	6.375
100	1355.11	446.1	3257.8	823.8	1.23	11.3	336	5.01
500	1387.94	456.6	5538	827.5	1.19	8.53	340	3.11
Nrand (0-1000)	1379.65	474.2	4441.9	873.7	1.22	11.1	337	3.01

Так же, как и на конфигурации с 2 вычислительными блоками, 1 HTTP клиент не может должным образом нагрузить кластер, поэтому данная система так же не состоявшаяся.

Таблица 10. Результаты тестирования конфигурации 2 при 4 НТТР клиентах.

Среднее время между запросами (мс)	T(п) (мс)	T(к) (мс)	T(о) (мс)	T(в) (мс)	W (Вт)	S (Кбит/с)	ОЗУ (Мбайт)	C (%)
0	1559.3	432.6	0	823.4	1.7	49	331	14.75
500	1194.6	429.1	157.32	816.1	1.57	43	328	12.21
1000	1214.0	424.1	407.4	821.5	1.53	27	336	11
3000	1210.3	432.4	2053.3	829.3	1.35	25	329	5.3
5000	1219.4	445.3	4613.7	829.3	1.2	14	331	4.2
Nrand (0-5000)	1233.8	485	1952	842.1	1.34	26	340	6.3

4 вычислительных блока уже могут себя хорошо показывать на сравнительно малых паузах между запросами. Оптимальные потребление энергии и времени вычисления находятся в границе паузы 400-600мс. Но все же, при непостоянных нагрузках кластер простаивает довольно много времени.

Конфигурация-6:

Сравнения времени вычисления 1000 запросов. Запросы уже лежит на сервере. НТТР клиент ждет только ответов.

Таблица 11. Результаты тестирования конфигурации 6.

Среднее время между запросами (мс)	T(к) (мс)	T(в) (мс)	T(общее) (с)	W (Вт)	S (Кбит/с)
------------------------------------	--------------	--------------	-----------------	-----------	---------------

AMD Ryzen 5 3600	198	45,4	249	~15	0
ROCK PC (1)	426.3	850.1	1303.6	0.39	18
ROCK PC (2)	418.9	829.4	616.9	1.02	33
ROCK PC (3)	438.29	818.9	414.9	1.56	56
ROCK PC (4)	431.29	819.7	301.3	2.1	68
Intel i5-4200u.	212,2	105,2	311	~9	56

В данном сравнении хорошо прослеживается преимущество кластера из 4х энергоэффективных микрокомпьютеров над высокоскоростным Ryzen 5 3600. Время расчета 1го изображения более чем в 18 раз превышает время расчета 1го изображения на 1м ROCK PC, но общее время вычисления 1000 изображений на кластере из 4х ROCK PC вполне сопоставимо с 1м Ryzen 5 3600, а энергопотребление меньше более чем в 7 раз.

3.5 Выводы.

В ходе проведения симуляций были выяснены следующие достоинства и недостатки системы.

- Она не совсем подходит для вычисления больших объёмов малых изображений. Затраты времени по передачи изображений на вычислители сравнимы с временем вычислений изображения даже на наименее мощных исполнителях. Из симуляций видно, как плохо загружены процессоры (не превышает 20%). Как одно из решений

данной проблемы можно рассматривать копирование задач группами по несколько, что может сильно повысить загрузку процессора и уменьшить время простоя.

- При большом количестве задач с малым периодом ожидания кластер оказывается эффективней производительного процессора: имеет гораздо более низкие потребления при сравнимом времени расчета.
- При малом объёме задач, широко разнесенных по времени, кластер тоже становится эффективнее ввиду особо малом потреблении энергии во время простоя (около 0.8Вт).
- Rock рі s довольно плохо справляется с задачами конечного вычислителя ввиду малого объёма памяти (512Мб) при размерах даже простой модели (~300-400МБ).

3. ЗАКЛЮЧЕНИЕ

В рамках данной выпускной квалификационной работы было выполнено следующие этапы:

- Поиск и сравнение способов распределенного вычисления СНС.
Наиболее выделяется 3 вида: параллелизм данных, параллелизм слоев, параллелизм моделей. Наиболее эффективным является совмещение этих 3 парадигм. Наиболее распространён расчет СНС последовательным методом на кластере ПЛИС.
- Разработка архитектуры для вычислений СНС с различными конечными вычислителями. Конечная архитектура имеет вид звезды на верхнем уровне и может иметь различные конечные вычислители в рамках вычислительного блока.
- Создание имитационной модели и ее тестирование. Была разработана система приложений, включающая в себя 5 программных компонента: HTTP клиент, HTTP сервер, локальный сервер, клиент вычислительного блока, программа для обучения нейронной сети. Так же, данная система была протестирована на комплексе аппаратных вычислителей. Данный программно-аппаратный комплекс хорошо показал себя на обработке пула изображений размером 1000 с периодом задержек 0-10мс. Общее время вычисления 1000 изображений на кластере всего на 17,2% больше, чем на стационарном процессоре. Энергии же, для работы кластера, потребовалось в 7 раз меньше. Так же, кластер эффективен при запросах реже 1с. Процессор простаивает более 83% времени и потребляет в среднем около 7Вт, кластер же простаивает около 33% времени при потреблении около 1,5Вт.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Ben-Nun T., Hoefler T. Demystifying parallel and distributed deep learning: An in-depth concurrency analysis //ACM Computing Surveys (CSUR). – 2019. – Т. 52. – №. 4. – С. 1-43.
2. Wang T. et al. FPDeep: Scalable acceleration of CNN training on deeply-pipelined FPGA clusters //IEEE Transactions on Computers. – 2020. – Т. 69. – №. 8. – С. 1143-1158.
3. Blott M. et al. Scaling neural network performance through customized hardware architectures on reconfigurable logic //2017 IEEE International Conference on Computer Design (ICCD). – IEEE, 2017. – С. 419-422.
4. Хайдарова Р. Р. и др. Модель распределенной сверточной нейронной сети на кластере компьютеров с ограниченными вычислительными ресурсами //Научно-технический вестник информационных технологий, механики и оптики. – 2020. – Т. 20. – №. 5. – С. 739-746.
5. Zhang C. et al. Energy-efficient CNN implementation on a deeply pipelined FPGA cluster //Proceedings of the 2016 International Symposium on Low Power Electronics and Design. – 2016. – С. 326-331.
6. Dundar A., Jin J., Culurciello E. Convolutional clustering for unsupervised learning //arXiv preprint arXiv:1511.06241. – 2015.
7. Zhai S. et al. Design of convolutional neural network based on fpga //Journal of Physics: Conference Series. – IOP Publishing, 2019. – Т. 1168. – №. 6. – С. 062016.
8. Gratadour D. et al. Prototyping AO RTC using emerging high performance computing technologies with the Green Flash project //Adaptive Optics Systems VI. – SPIE, 2018. – Т. 10703. – С. 404-418.
9. Spring Boot // Spring URL: <https://spring.io/projects/spring-boot> (дата обращения: 12.05.2023).
10. TensorFlow 2.12.0 // Github URL: <https://github.com/tensorflow/tensorflow/releases/tag/v2.12.0> (дата обращения: 12.05.2023).
11. ROCK Pi S // Radxa URL: <https://wiki.radxa.com/RockpiS> (дата обращения: 12.05.2023).
12. Mnist // Tensorflow URL: <https://www.tensorflow.org/datasets/catalog/mnist?hl=en> (дата обращения: 12.05.2023).