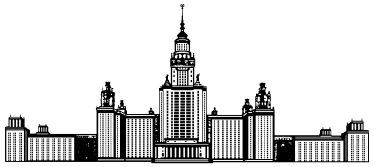
Московский государственный университет имени М. В. Ломоносова



Факультет Вычислительной Математики и Кибернетики

Кафедра Суперкомпьютеров и Квантовой Информатики

КУРСОВАЯ РАБОТА

«Исследование методов параллельного обучения нейронных сетей»

Выполнил:

студент 3 курса 323 группы

Толстобров Кирилл Александрович

Научный руководитель:

к.ф.-м.н., доцент

Попова Нина Николаевна

Москва, 2022

Содержание

1 Введение **3**

2 Постановка задачи **4**

3 Обзор существующих методов параллельного обучения нейронных сетей **5**

3.1 Последовательное обучение нейронных сетей 5

3.2 Сверточные нейронные сети 7

3.3 Параллельные методы обучения 9

3.3.1 Уровень синхронизации 12

3.3.2 Уровень архитектуры 13

3.3.3 Уровень сжатия данных 16

3.4 Выводы 16

4 Методы реализации примеров параллельных алгоритмов обучения нейросетей **17**

4.1 Local-SGD 18

4.2 Gossip 18

4.3 Sparsification 19

5 Экспериментальное исследование реализованных параллельных алгоритмов **21**

5.1 Эксперименты на Polus 23

5.2 Эксперименты на ЛОМОНОСОВ-2 23

5.3 Выводы 27

6 Заключение **29**

6.1 Направления дальнейших исследований 29

Список литературы **31**

7 Приложение **32**

1 Введение

Нейронная сеть - это частный случай методов машинного обучения, вдохновленный процессами, происходящими в мозге живого организма. Фундаментальной единицей нейронной сети является математическое представление нейрона - клетки нервной системы, предназначенной для приема, обработки, хранения и передачи информации. Нейрон получает от дендритов - своих разветвленных отростков - электрические сигналы, а в результате их обработки формирует выходной сигнал, который передается другим нейронам.

Одноименное математическое представление нейрона преобразует вектор входных сигналов в выходной сигнал. Для этого он вычисляет скалярное произведение вектора входных сигналов и своего вектора весов, к полученному значению прибавляет величину, называемую смещением, после чего к результату применяет некоторую функцию активации.

Нейроны объединяют в слои, из которых и состоит нейронная сеть. Каждый нейрон слоя получает вектор сигналов от предыдущего слоя, а выходной сигнал отправляет на следующий слой. Нейронная сеть принимает на входной слой некоторый набор признаков, а на выходном слое возвращает искомые значения (например, класс, к которому сеть относит входной объект).

Нейронные сети используются для решения практических задач в ряде областей: от классификации изображений и распознавания речи до автономного вождения и медицинской диагностики.

Чтобы нейронная сеть работала правильно, ее необходимо обучить. Обучение - это процесс настройки весов сети (здесь и далее к весам также будем относить смещения). В процессе обучения нейронная сеть выявляет сложные зависимости между входными и выходными данными и учиться выполнять обобщение.

Обучение нейронной сети - это длительный процесс. При решении сложных задач обучение может занимать от нескольких часов до нескольких дней или даже недель.

Кроме того, для хранения весов модели и обучающего датасета может потребоваться большое количество оперативной памяти.

К счастью, структура нейронных сетей позволяет использовать для их обучения распределенные вычисления. Параллельная обработка данных значительно ускоряет процесс обучения, а распределение модели или обучающего датасета между несколькими вычислительными узлами позволяет решить проблему нехватки памяти.

К настоящему времени было предложено большое количество методов параллельного обучения нейронных сетей. Каждый метод имеет как определенные преимущества, так и недостатки. При оценке и сравнении эффективности данных методов необходимо учитывать ряд параметров, таких как скорость сходимости, время, затраченное на синхронизацию, длительность коммуникаций и т. д. В данной работе будет произведен обзор существующих методов параллельного обучения нейронных сетей, а затем будут предоставлены результаты экспериментального исследования и сравнения наиболее интересных из них.

2 Постановка задачи

Целью работы является исследование методов параллельного обучения нейронных сетей. Для достижения поставленной цели необходимо решить следующие задачи:

1. Провести обзор существующих методов параллельного обучения нейронных сетей.
2. Выбрать и реализовать подмножество рассмотренных методов.
3. Провести экспериментальное исследование выбранных методов, сделать выводы об их эффективности.

3 Обзор существующих методов параллельного обучения нейронных сетей

В данном разделе будет произведен обзор существующих методов параллельного обучения нейронных сетей. Для начала кратко опишем принципы последовательного обучения нейронных сетей, а также рассмотрим архитектуру сверточных нейронных сетей, нацеленную на эффективную обработку изображений.

3.1 Последовательное обучение нейронных сетей

Стандартный алгоритм обучения нейронных сетей состоит из двух шагов: прямого распространения и обратного распространения ошибки.

В прямом распространении сеть получает входные данные, обрабатывает их и вычисляет целевые значения. Например, в задаче классификации изображений входными данными являются значения цветовых каналов каждого пикселя картинки, а целевое значение - класс, к которому принадлежит объект на изображении. Прямое распространение начинается с подачи входных данных на первый слой сети. После этого данные передаются на второй слой, каждый нейрон которого вычисляет значение некоторой функции активации от увеличенного на смещение скалярного произведения полученного вектора значений и своего вектора весов. Примером функции активации является ReLu:

$$\varphi(x) = max(0, x)$$

Таким образом, каждый нейрон вычисляет выходной сигнал следующим образом:

$$h_j = \varphi(\sum_{i=1}^{n} w_{ji}x_i + b_j)$$

Вычисленные каждым нейроном второго слоя значения формируют вектор, который передается на следующий слой, где процесс повторяется. После последовательного прохождения через все слои на выходе сети формируется результат - предсказанное целевое значение.

Задачей обучения нейронной сети является минимизация функции ошибки:

$$E(w) = \sum_{i=1}^{l} L(a(x_i, w), y_i)$$

где x_i - элемент обучающей выборки, l - размер этой выборки, w - множество всех весов сети, $a(x_i, w)$ - предсказанное сетью значение, соответствующее элементу выборки, y_i - реальное целевое значение, соответствующее элементу выборки, $L(a, y)$ - некоторая заданная функция потерь.

Обычно для минимизации применяют метод градиентного спуска. Это пошаговый алгоритм, на каждой итерации которого вектор w изменяется в направлении наибольшего убывания функции $E(w)$. Для этого вычисляется градиент указанной функции $\bigtriangledown E(w)$, после чего происходит обновление весов:

$$w = w - \eta \bigtriangledown E(w)$$

где η - положительный параметр, называемый темпом обучения.

Вычисление градиента происходит на этапе обратного распространения ошибки. Этот этап состоит в распространении сигналов ошибки от выходов сети к её входам, в направлении обратном прямому распространению сигналов в обычном режиме работы. Для этого последовательно вычисляются производные функции ошибки по весам каждого слоя: от последнего к первому.

Производные вычисляются по следующему правилу:

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial z_j} \frac{\partial z_j}{\partial w_{ij}} = x_i \frac{\partial E}{\partial z_j}$$

Здесь w_{ij} - i-ый вес j-го нейрона текущего слоя, x_i - i-ый элемент входного вектора этого слоя, $z_j = \sum_i w_{ij} x_i$ - скалярное произведение, вычисляемое на j-ом нейроне. Выражение $\frac{\partial E}{\partial z_j}$ называют локальным градиентом, оно вычисляется похожим образом, и зависит от локального градиента следующего слоя и весов текущего слоя.

Видно, что указанная выше функция ошибки зависит от всех элементов обучающей выборки. Это означает, что чтобы определить новое приближение вектора весов

необходимо вычислить градиент от каждого элемента выборки. Такой вариант градиентного спуска называется Batch Gradient Descent.

Другой вариацией является стохастический градиентный спуск (Stochastic Gradient Descent - SGD). В этом методе значение градиента аппроксимируются градиентом функции ошибки, вычисленном только на одном случайно выбранном элементе обучения. Таким образом параметры модели изменяются после обработки каждого объекта обучения.

Третьим, наиболее гибким вариантом, является Mini-Batch Gradient Descent. В данном методе обучающая выборка разбивается на мини-батчи фиксированного размера. Расчет функции ошибки и градиентов производится в результате обработки одного мини-батча. Процесс обработки мини-батча называют итерацией, а один проход через весь обучающий датасет - эпохой. Этот метод имеет большую частоту обновления весов, чем в Batch Gradient Descent, что обеспечивает более устойчивую сходимость. При этом пакетные обновления делают вычисления более быстрыми, чем в стохастическом градиентном спуске.

3.2 Сверточные нейронные сети

Сверточная нейронная сеть - специальная архитектура нейронных сетей, нацеленная на эффективное распознавание изображений. Основными видами слоев в сверточной нейронной сети являются сверточные слои, пулинговые слои, а также уже рассмотренные полносвязные слои.

Сверточный слой принимает на вход набор изображений в виде матриц, применяет к ним операцию свертки и на выходе образует новый набор матриц. Свертка - это операция над входной матрицей A и матрицей B, называемой ядром, результатом которой является матрица C. Каждый элемент C вычисляется как скалярное произведение B на некоторую подматрицу A такого же размера, при этом ядро «движется» по матрице A с определенным шагом, в каждом положении считается скалярное про-

изведение B на ту часть A, на которую ядро наложено, а результат помещается в соответствующую позицию C.

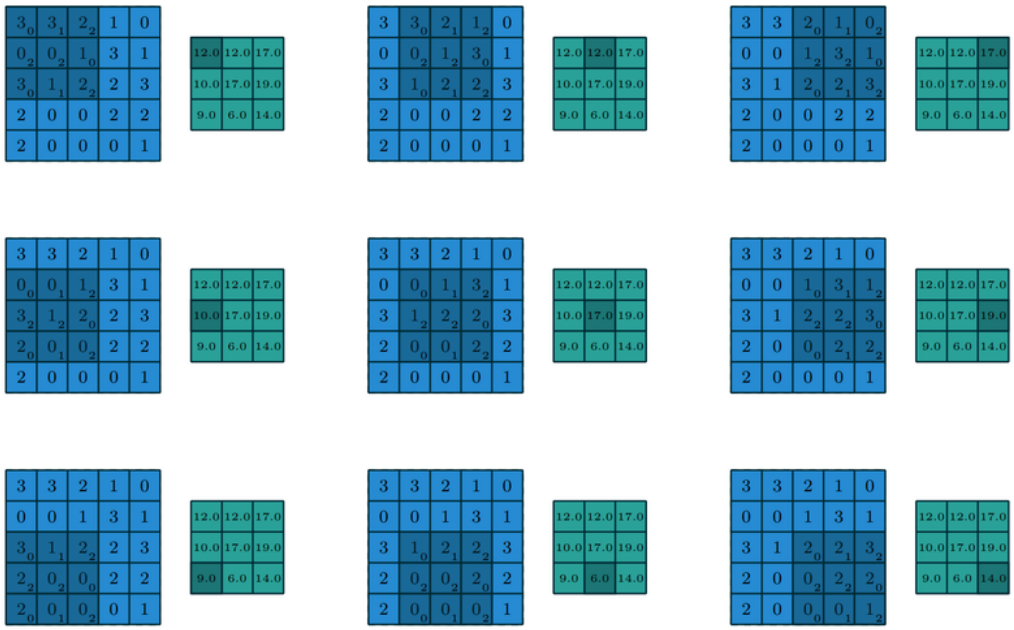


Рис. 1: Пример свертки двух матриц

Пулинговый слой используется для уменьшения размера изображения. Входное изображение делится на блоки определенного размера, к каждому блоку применяется некоторая функция, результат которой записывается в соответствующую позицию матрицы на выходе. Примером используемой функции является функция максимума, возвращающая максимальный элемент блока.

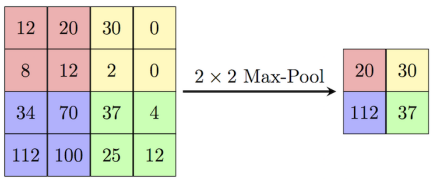


Рис. 2: Пример операции пулинга

3.3 Параллельные методы обучения

Перейдем к параллельным методам обучения. Существует три основные стратегии разделения нейронной сети для распределения обучения: параллелизм данных, параллелизм модели и параллелизм слоев.

При параллелизме данных между вычислительными устройствами распределяется обучающая выборка. Как уже было сказано, в градиентном спуске с мини-батчами выборка разделяется на $N = M/B$ частей, где M - размер всей выборки, B - выбранный размер мини-батча. В стратегии параллелизма данных каждое из P вычислительных устройств получает $n = N/P$ мини-батчей. Каждое устройство обрабатывает свой набор данных, при этом вычисленные на разных узлах градиенты периодически усредняются, чтобы сохранить согласованность модели. Алгоритм данной стратегии можно описать так:

1. Инициализация весов модели нейронной сети на каждом вычислительном устройстве и их синхронизация
2. Равномерное разделение обучающей выборки по устройствам в случайном порядке
3. Локальная обработка одного мини-батча каждым устройством, вычисление локальных градиентов
4. Усреднение градиентов между всеми устройствами
5. Обновление весов модели с помощью усредненных градиентов
6. Если остались необработанные мини-батчи, переход к пункту 3
7. Начало следующей эпохи, переход к пункту 2

Следующая стратегия - параллелизм модели - заключается в распределении между вычислительными устройствами нейронов сети. Каждый слой сети делится на равное количество частей, если слой содержит K нейронов, то каждому устройству

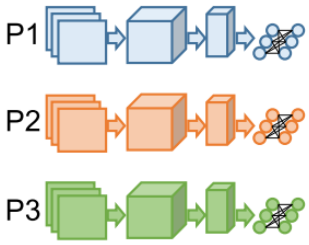


Рис. 3: Параллелизм данных.

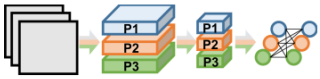


Рис. 4: Параллелизм модели

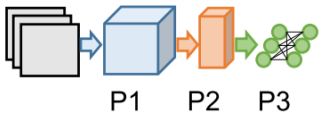


Рис. 5: Параллелизм слоев

отходит K/P из них. Нейроны, находящиеся на устройстве, получают на вход вектор значений, преобразуют его в выходной сигнал, который передают на следующий слой. При этом выходной сигнал с каждого нейрона необходимо отправить на каждое другое вычислительное устройство, чтобы все нейроны следующего слоя получили полный набор сигналов. Данная стратегия не проигрывает параллелизму данных в скорости обработки изображений и не теряет скорость сходимости модели. Однако количество коммуникаций между устройствами сильно увеличивается, что добавляет значительные временные затраты. Также каждое устройство в этой стратегии вынуждено хранить весь набор обучающих данных, что увеличивает затраты памяти.

Последняя стратегия - параллелизм слоев. Она заключается в распределении слоев нейронной сети между вычислительными устройствами. При этом вычисления происходят по принципу конвейера: устройство обрабатывает входные данные, отправляет их на следующий слой на другом устройстве и начинает обработку следующего набора данных. В данной стратегии нет необходимости в хранении полной модели на каждом узле, однако ускорение обработки изображений становится меньше, чем в параллелизме данных и параллелизме модели. Это связано с конвейерной природой данной стратегии, а также с тем, что время обработки данных на разных слоях может сильно отличаться, что приводит к длительному простаиванию некоторых узлов в ожидании данных с предшествующего слоя.

Как показано в [4], при параллельном обучении на большом количестве вычислительных устройств появляются значительные временные затраты на коммуникации - обмен данными и синхронизацию между устройствами. На рис. 6 представлены результаты исследований из [4], в которых рассматривалась длительность коммуникаций при использовании гибридной схемы параллелизма данных и модели. Здесь P_c обозначает количество групп параллелизма данных, а P_r - количество вычислительных устройств в одной группе, функционирующих в ее пределах по стратегии параллелизма модели. Видно, что даже при оптимальном выборе конфигурации параллелизма на большом количестве узлов коммуникационные временные затраты существенны, а иногда они даже превышают вычислительные затраты.

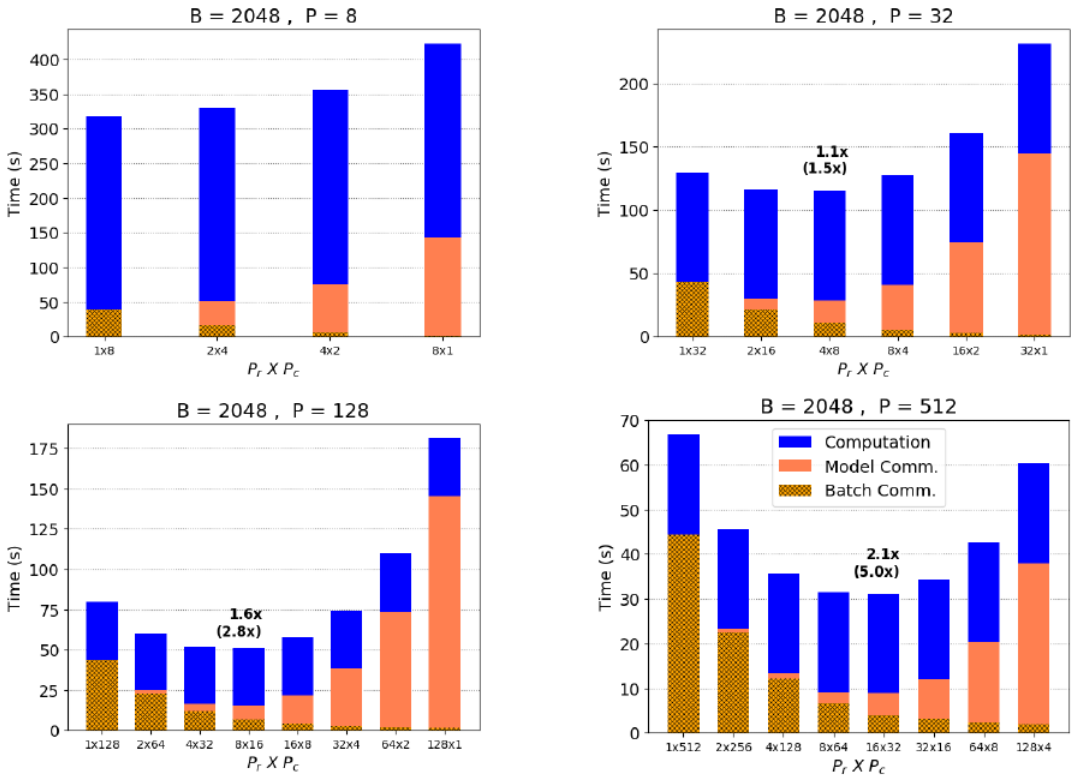


Рис. 6:

Чтобы справиться с данной проблемой, к настоящему времени был предложен ряд методов параллельного обучения в рамках стратегии параллелизма данных. Ос-

новной целью данных методов является снижение коммуникационных затрат, при этом стараются сохранить приемлемую скорость сходимости модели (то есть ускорение коммуникаций не должно сильно снижать точность модели к концу обучения). По таксономии, предложенной в [1], их можно разделить на три уровня: уровень синхронизации, уровень архитектуры и уровень сжатия данных.

3.3.1 Уровень синхронизации

На этом уровне определяется, когда должна происходить отправка градиентов для усреднения. Базовый подход называется массово-синхронным (Bulk Synchronous Parallel) или просто синхронным и подразумевает, что вычислительные устройства обмениваются градиентами после каждой итерации обучения.

Более гибким методом является Local-SGD. В этом подходе обмен градиентами не производится, однако веса моделей усредняются после прохождения каждым вычислительным узлом I итераций (при $I = 1$ получаем массово-синхронный метод). В таком случае, количество коммуникаций уменьшается в I раз, что значительно снижает коммуникационные затраты. Однако при больших значениях I страдает сходимость модели, так как веса на разных узлах могут стремиться к разным локальным минимумам функции ошибки, из-за чего после их усреднения будут получаться довольно хаотичные результаты. В [6] дается теоретическое доказательство хорошей сходимости этого метода при следующих ограничениях на I:

$$I \leq \frac{T^{1/4}}{N^{3/4}}$$

где T - общее количество итераций в обучении, N - количество вычислительных узлов.

Синхронность указанных методов является существенным недостатком в случае, если используемая вычислительная система состоит из узлов с разной производительностью. В таком случае, более быстрые узлы по окончании требуемого количества итераций будут вынуждены простаивать в ожидании более медленных узлов. Следовательно, производительность системы не будет использоваться на максимум.

Чтобы решить данную проблему был предложен асинхронный метод усреднения весов. Для его реализации необходимо наличие одного узла, на котором будут храниться актуальные значения весов модели. Остальные устройства по окончании обработки одного мини-батча не дожидаясь других отправляют полученные градиенты на этот узел, он, в свою очередь, обновляет модель с помощью полученных градиентов. Перед началом новой итерации обучения каждое устройство получает от данного узла актуальные значения весов. Данный метод позволяет эффективно использовать производительность системы, однако появляется проблема устаревания градиентов: так как обновление модели на каждом устройстве производится только в начале итерации обучения, большую часть времени они будут функционировать с неактуальными значениями весов.

Последний метод - частично-синхронный (Stale-Synchronous) - является компромиссом между синхронным и асинхронным обучением. Вычислительные устройства в данном методе работают асинхронно, однако могут провести лишь определенное количество итераций, после чего начнут ждать окончания текущей итерации остальных узлов, чтобы совершить синхронизацию модели. Перед началом синхронизации быстрые узлы успеют обработать несколько мини-батчей, а более медленные - меньшее количество. В таком случае, неоднородная по производительности система будет использоваться относительно эффективно, при этом устаревание градиентов будет не таким значительным, как в полностью асинхронном подходе.

3.3.2 Уровень архитектуры

На данном уровне решается вопрос выбора топологии вычислительной сети. Стандартный метод, используемый для распределенного обучения во многих популярных нейросетевых фреймворках, заключается в использовании сервера параметров. В данном методе один из вычислительных узлов назначается сервером параметров и является хранилищем всех весов модели. Обучение начинается с того, что сервер параметров рассылает остальным вычислительным узлам инициализированные значения весов. После обработки своих мини-батчей рабочие узлы отсылают вы-

численные локальные градиенты на сервер параметров. Получив градиенты со всех рабочих, сервер обновляет с их помощью веса модели, после чего актуальные значения весов отправляет обратно рабочим для продолжения обучения. Такой метод также называют централизованным. Его существенным недостатком является узкое место, возникающее при коммуникациях, так как все рабочие приблизительно в одно время начинают отправлять градиенты на сервер. Чтобы частично справиться с этой проблемой, можно использовать несколько серверов параметров, каждый из которых будет хранить лишь определенную часть модели. В таком случае объем принимаемых сервером градиентов уменьшится, что ускорит коммуникации.

Более эффективным решением является децентрализованный метод. В нем усреднение весов производится с помощью операции Allreduce. Это означает, что по окончании итерации каждый вычислительный узел отправляет вычисленные градиенты на каждый другой узел. Данный подход стал более эффективным в связи с появлением эффективных реализаций операции Allreduce. Например, в [2] был предложен Ring-Allreduce. Принцип работы данной реализации представлен на рис. 7. Операция в указанном подходе реализуется за $2(N-1)$ шагов, где N - количество узлов. При этом на каждом шаге происходят обмены данными только с соседями, что позволяет избежать узкого места при коммуникациях, которое присутствует в централизованном методе. Предполагается, что данная реализация операции Allreduce является оптимальной при больших размерах передаваемого буфера.

Последним на уровне архитектуры является метод gossip. Он так же является децентрализованным, однако, чтобы еще сильнее сократить коммуникационные затраты, обмен данными предлагается производить не со всеми узлами, а только с определенной их частью. Выбор получателей на каждом узле происходит с помощью матрицы весов K, которая определяет, как градиенты, полученные на других устройствах, будут влиять на локальную модель. Если в i-ой строчке и j-ом столбце матрицы находиться число 0.1, значит вычислительное устройство с номером i в конце итерации получает градиент от устройства с номером j, умножает его на 0.1 и прибавляет к своим весам модели. Если в указанной позиции матрицы стоит 0, то

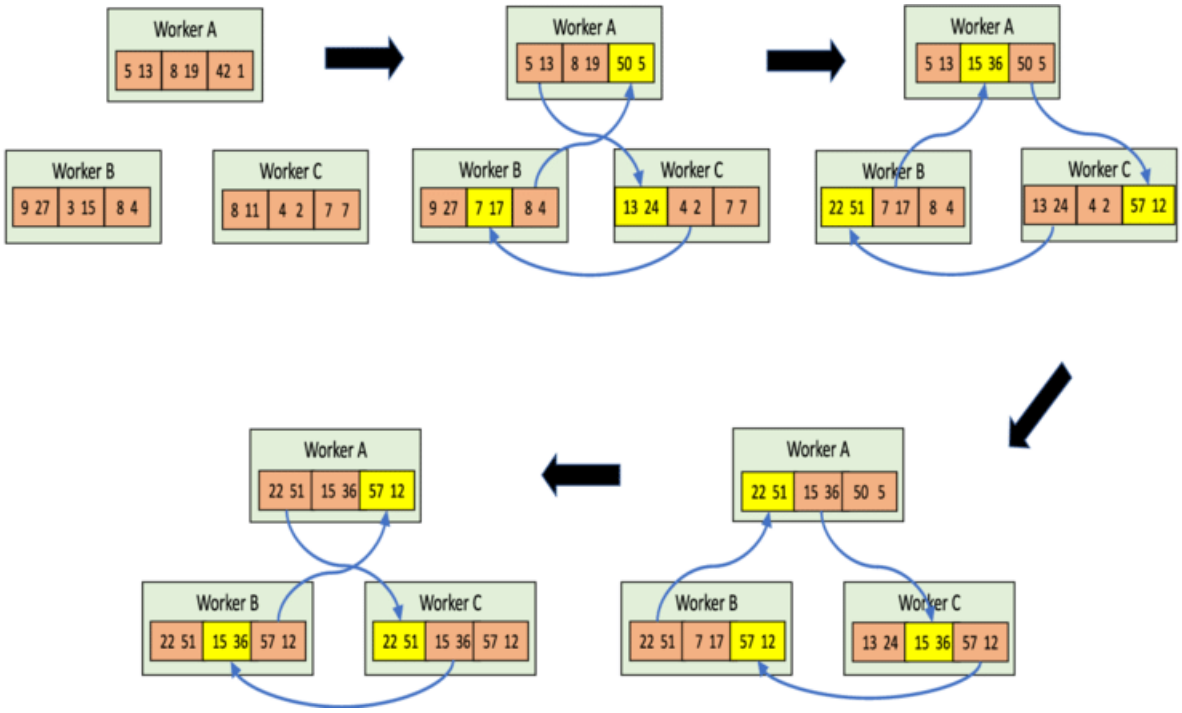


Рис. 7: Ring-Allreduce

данные устройства не обмениваются градиентами. В самом простом и популярном случае происходит усреднение градиентов между соседями. То есть каждый узел получает градиенты от двух соседних узлов, усредняет их и свои градиенты, после чего обновляет веса модели. В таком случае матрица весов топологии имеет следующий вид:

$$\mathrm{K}_{m,n} = \begin{pmatrix} 1/3 & 1/3 & 0 & 0 & \cdots & 0 & 1/3 \\ 1/3 & 1/3 & 1/3 & 0 & \cdots & 0 & 0 \\ 0 & 1/3 & 1/3 & 1/3 & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \ddots & \cdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & 1/3 & 1/3 \\ 1/3 & 0 & 0 & 0 & \cdots & 1/3 & 1/3 \end{pmatrix}$$

Стоит понимать, что при использовании данного метода модели, хранящиеся на разных узлах, перестают быть одинаковыми, что негативно сказывается на скорости сходимости.

3.3.3 Уровень сжатия данных

На данном уровне производится сжатие данных перед их отправкой. Уменьшение объема отправляемых буферов позволяет ускорить коммуникации. Методы данного уровня можно разделить на две группы: квантование и спарсификация.

В методах квантования каждый передаваемый градиент относят к одной из групп, которые представляют определенный набор значений. Фактически, по некоторой стратегии производится округление градиентов до некоторых величин, которые затем кодируются и отправляются на другой вычислительный узел. Округление данных позволяет сократить набор возможных значений, благодаря чему их можно закодировать с использованием меньшего количества бит. Так, например, в [5] удалось сократить количество используемых для передачи одного значения бит с 32 до 4, сохранив при этом хорошую сходимость.

Методы спарсификации используют тот факт, что многие градиенты, вычисляемые в течение обучения, довольно невелики, и их можно не отправлять, принимая равными нулю. При этом сходимость модели страдает незначительно. Обычно в таких методах на вычислительном узле перед отправкой градиентов происходит их сравнение с некоторым пороговым значением. Если некоторое изменение веса меньше этого порогового значения, оно не отправляется. При этом пороговое значение может быть статическим или адаптивным. Во втором случае оно подстраивается таким образом, чтобы отсеивать определенный процент отправляемых значений. Помимо самих градиентов в этом методе также необходимо отправлять информацию о расположении передаваемых градиентов, то есть индексы матрицы весов, соответствующие им.

3.4 Выводы

Была рассмотрена большая часть существующих на сегодняшний день методов параллельного обучения нейронных сетей. Стоит заметить, что методы трех пере-

численных уровней можно использовать комбинированно, что, в некоторых случаях, может дать лучший результат, чем при их раздельном использовании.

В [1] можно посмотреть, какие из перечисленных методов и их комбинаций уже хорошо исследованы, а какие еще предстоит изучить. Можно заметить, что довольно мало исследований было посвящено методу gossip на уровне архитектуры, методу спарсификации на уровне сжатия данных, а также их комбинации. В данной работе будут реализованы и экспериментально исследованы два указанных метода и метод Local-SGD. Эти методы были выбраны в связи с тем, что на сегодняшний день они плохо изучены, а также потому, что их можно использовать комбинированно.

4 Методы реализации примеров параллельных алгоритмов обучения нейросетей

Для проведения экспериментального исследования указанных методов параллельного обучения было необходимо построить их программную реализацию, а также программную реализацию нейронной сети, в рамках которой они будут работать. Для этого был выбран язык программирования C++, так как он отличается высокой скоростью выполнения программ.

Было решено отказаться от использования существующих нейросетевых библиотек, так как большая их часть обладает высокоуровневым интерфейсом, затрудняющим внедрение собственной реализации методов параллельного обучения. Это также позволяет избежать существующие в этих библиотеках оптимизации, которые могут неожиданным образом влиять на эффективность исследуемых методов.

Обучение нейронной сети было решено проводить на CPU. Для организации параллельной работы и реализации коммуникаций между вычислительными устройствами был использован программный интерфейс MPI. MPI является наиболее распространённым стандартом интерфейса обмена данными в параллельном програм-

мировании. Он позволяет писать переносимые, хорошо масштабируемые и эффективные параллельные программы.

4.1 Local-SGD

Для реализации Local-SGD необходимо установить значение периода синхронизаций, а также определить способ проведения коммуникаций. В случае применения Local-SGD без добавления других исследуемых методов, для проведения коммуникаций используется операция MPI Allreduce.

Algorithm 1 Local-SGD (на i-ом устройстве)

1: Инициализировать веса модели $w_{0,i} = w_0$, установить темп обучения $\eta > 0$, количество итераций K и период синхронизаций $p > 0$
2: **Цикл от** $k = 1$ **до** K **выполнять**
3: Вычислить градиент функции ошибки по одному мини-батчу $\bigtriangledown E_i(w_{k-1,i})$
4: **Если** $k \bmod p = 0$ **тогда**
5: Вычислить средние значения весов между устройствами $\bar{w} = \frac{1}{P}\sum_{j=0}^{P-1} w_{k-1,j}$ (Суммирование ведется операцией Allreduce)
6: Обновить локальные веса: $w_{k,i} = \bar{w} - \eta \bigtriangledown E_i(w_{k-1,i})$
7: **иначе**
8: Обновить локальные веса: $w_{k,i} = w_{k-1,i} - \eta \bigtriangledown E_i(w_{k-1,i})$
9: **Конец Если**
10: **Конец Цикл**

4.2 Gossip

Для реализации была выбрана вариация метода gossip, основанная на распространении градиента, предложенная в [3]. В данной вариации на каждом шаге партнеры для выполнения усреднения градиента меняются. Сначала происходит обмен с двумя соседями, то есть с процессами, ранг которых отличается от собственного ранга

устройства на 1. На следующей итерации обучения обмен будет происходить между процессами, ранг которых различается на 2, на третьей итерации - на 3 и т.д. Это незначительно замедляет коммуникации, однако сильно улучшает сходимость метода.

Algorithm 2 Gossip (на i-ом устройстве)

1: Инициализировать веса модели $w_{0,i} = w_0$, установить темп обучения $\eta > 0$, количество итераций K и расстояние до партнеров по обмену $s = 1$
2: **Цикл от** $k = 1$ **до** K **выполнять**
3: Вычислить градиент функции ошибки по одному мини-батчу $\bigtriangledown E_i(w_{k-1,i})$
4: Вычислить партнеров по обмену: $r = (i + s) \bmod P$, $l = (i - s + P) \bmod P$
5: Получить градиенты от соседей (Обмен ведется операцией Sendrecv)
6: Обновить локальные веса: $w_{k,i} = w_{k-1,i} - \frac{1}{3}\eta(\bigtriangledown E_l(w_{k-1,l}) + \bigtriangledown E_i(w_{k-1,i}) + \bigtriangledown E_r(w_{k-1,r}))$
7: $s = s + 1$
8: **Если** $s > \frac{P}{2}$ **тогда**
9: $s = 1$
10: **Конец Если**
11: **Конец Цикл**

4.3 Sparsification

При разработке стало понятно, что реализовать метод спарсификации при использовании стандартной децентрализованной топологии - крайне нетривиальная задача. Это связано с тем, что каждый вычислительный узел имеет свой набор и количество близких к нулю градиентов. Это означает, что при коммуникации невозможно использовать операцию Allreduce, на каждом вычислительном узле придется выделять память под индексы и изменения весов, полученные от каждого другого устройства, а время на обработку полученных значений многократно увеличится,

ведь придется отдельно обрабатывать данные, пришедшие с каждого узла. Однако использование спарсификации совместно с топологией gossip позволяет избежать всех этих проблем. Ведь на каждом вычислительном узле необходимо получить и обработать данные лишь от 2 партнеров по коммуникации. По этой причине спарсификация была реализована только в комбинации с топологией gossip.

Algorithm 3 Gossip + Sparsification (на i-ом устройстве)

1: Инициализировать веса модели $w_{0,i} = w_0$, установить темп обучения $\eta > 0$, количество итераций K, расстояние до партнеров по обмену $s = 1$ и порог сжатия t

2: **Цикл от** $k = 1$ **до** K **выполнять**

3: Вычислить градиент функции ошибки по одному мини-батчу $\bigtriangledown E_i(w_{k-1,i})$

4: Найти компоненты градиента, превышающие порог t, а также их индексы

5: Вычислить партнеров по обмену: $r = (i + s) \bmod P$, $l = (i - s + P) \bmod P$

6: Получить от соседей компоненты их градиентов, превышающие порог, а также соответствующие индексы (Обмен ведется операцией Sendrecv)

7: Обновить локальные веса: $w_{k,i} = w_{k-1,i} - \frac{1}{3}\eta(\bigtriangledown E_l(w_{k-1,l}) + \bigtriangledown E_i(w_{k-1,i}) + \bigtriangledown E_r(w_{k-1,r}))$. (Обновляются только веса, соответствующие индексам компонент градиентов, превышающих порог)

8: $s = s + 1$

9: **Если** $s > \frac{P}{2}$ **тогда**

10: $s = 1$

11: **Конец Если**

12: **Конец Цикл**

5 Экспериментальное исследование реализованных параллельных алгоритмов

В ходе работы было произведено 3 серии вычислительных экспериментов. Эксперименты заключались в обучении нейронной сети с использованием параллельных методов. Измерялись следующие величины:

- Точность сети после каждой эпохи обучения
- Общее время обучения
- Время, затраченное на коммуникации
- Время, затраченное на синхронизацию весов/градиентов (является суммой времени коммуникаций и времени синхронизации работы процессов)

В ходе экспериментов обучение нейронной сети производилось на датасетах CIFAR-10 И CIFAR-100. Каждый из этих наборов данных содержит по 60000 цветных изображений размера 32×32. Изображения в CIFAR-10 разделены на 10 классов по 6000 изображений в каждом, а в CIFAR-100 - на 100 классов по 600 изображений. В каждом наборе данных 50000 изображений - тренировочные, а остальные 10000 - тестовые.

Выбранная для обучения на CIFAR-10 модель нейронной сети представлена на Рис. 8. Данная модель была выбрана в связи с ее простотой и относительно небольшими размерами. Для обучения на CIFAR-100 была выбрана такая же модель, с разницей в последнем полносвязном слое, размер которого 100, а не 10.

Эксперименты проводились с зафиксированной последовательностью псевдослучайных чисел (т.е. функции srand() и ее аналогам в качестве аргумента передавалась константа). Это было сделано для того, чтобы полностью убрать из экспериментов факторы недетерминизма, мешающие объективно оценить исследуемые методы.

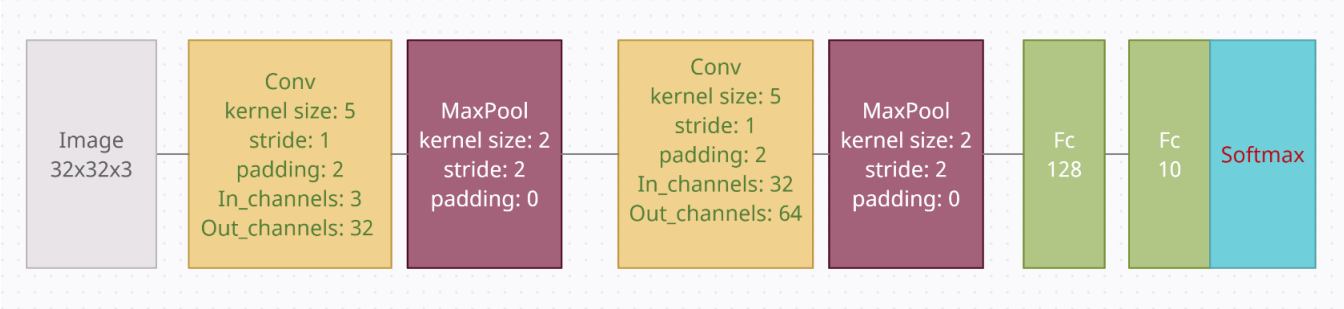


Рис. 8: Используемая модель нейронной сети

Обучение производилось в 10 конфигурациях, отличающихся используемыми параллельными методами (запись Local-SGD(p) означает, что используется период синхронизации, равный p. В случае p=1 это равносильно использованию массово-синхронного метода):

1. Local-SGD(1) + Allreduce топология
2. Local-SGD(1) + gossip топология
3. Local-SGD(1) + gossip топология + спарсификация
4. Local-SGD(4) + Allreduce топология
5. Local-SGD(4) + gossip топология
6. Local-SGD(4) + gossip топология + спарсификация
7. Local-SGD(16) + Allreduce топология
8. Local-SGD(16) + gossip топология
9. Local-SGD(16) + gossip топология + спарсификация
10. Local-SGD(32) + Allreduce топология

Пороговое значение в методе спарсификации задавалось равным 1, так как в ходе предварительных экспериментов оно оказалось оптимальным, сокращая количество

передаваемых градиентов на 95%, при этом несильно уменьшая точность модели после обучения.

5.1 Эксперименты на Polus

Первые две серии экспериментов проходили на вычислительном комплексе IBM Polus. Polus состоит из 5 вычислительных узлов (на момент проведения работы было доступно 3 из них), каждый из которых содержит 2 десятиядерных процессора IBM POWER8.

В ходе первой серии экспериментов нейронная сеть обучалась на наборе данных CIFAR-10. Обучение производилось в течение 50 эпох, размер мини-батчей и темп обучения были равны соответственно 100 и 0.0001. При обучении использовалось 16, 32 и 60 ядер. Результаты экспериментов на 60 ядрах видны на рисунках 9 и 10. Остальные результаты можно посмотреть в приложении.

В ходе второй серии экспериментов нейронная сеть обучалась на наборе данных CIFAR-100. Параметры обучения были такими же, как и в первой серии. Временные затраты в ходе этих экспериментов оказались практически такими же, как и в первой серии. Результаты измерения точности модели при обучении на 60 ядрах видны на рисунке 11.

5.2 Эксперименты на ЛОМОНОСОВ-2

Последняя серия экспериментов проводилась на суперкомпьютере ЛОМОНОСОВ-2. Каждый вычислительный узел ЛОМОНОСОВ-2 содержит 14-ядерный процессор Intel Haswell-EP E5-2697v3, 2.6 GHz.

В этой серии экспериментов нейронная сеть обучалась на наборе данных CIFAR-10. Для обучения использовалось 32, 60 и 210 ядер (размещенные на 3, 5 и 15 вычислительных узлах соответственно). Параметры обучения были такими же, как и в первых двух сериях.

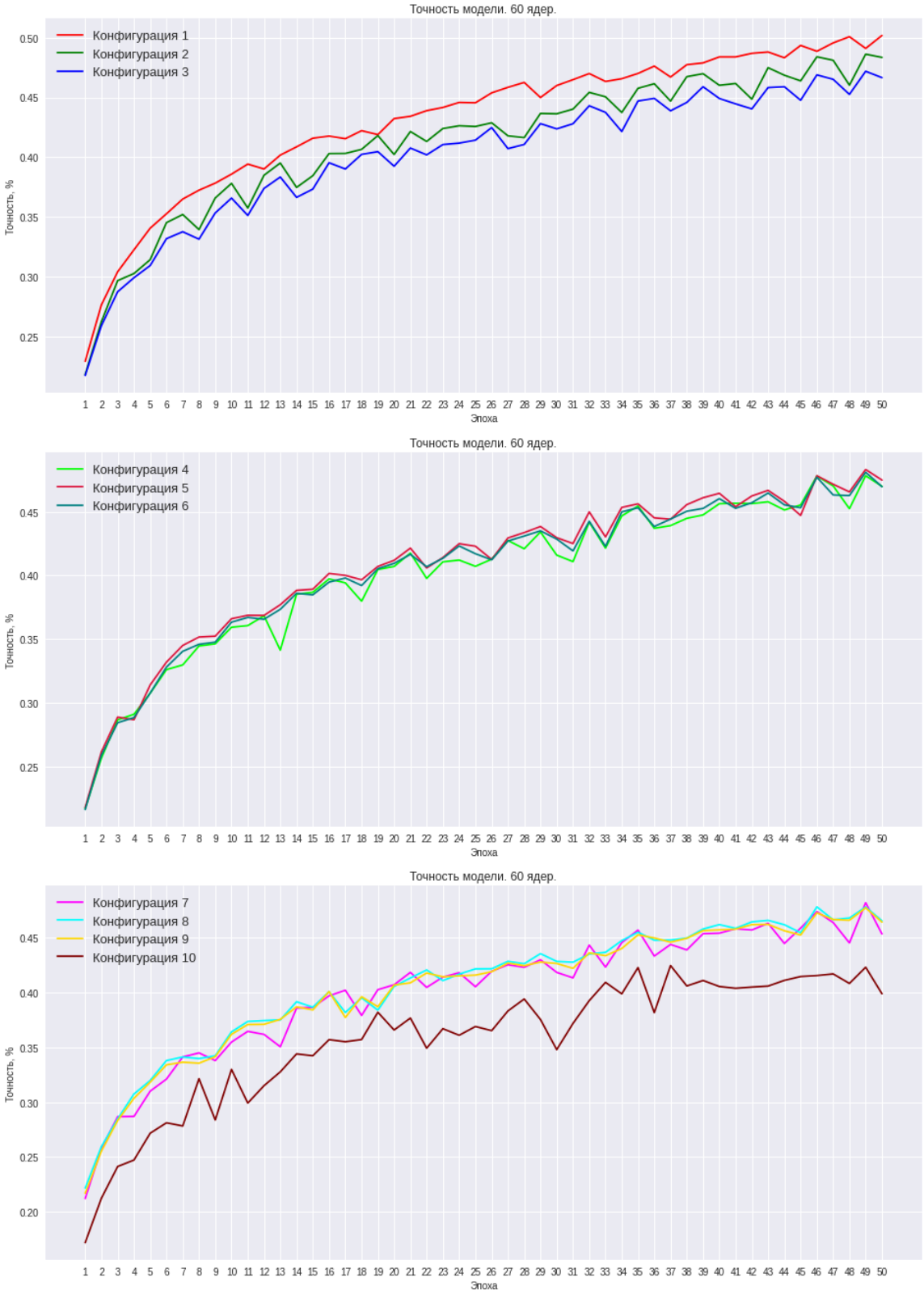


Рис. 9: CIFAR-10. Точность модели в разных конфигурациях при обучении на 60 ядрах

Рис. 10: Временные затраты при обучении на 60 ядрах в разных конфигурациях. IBM Polus

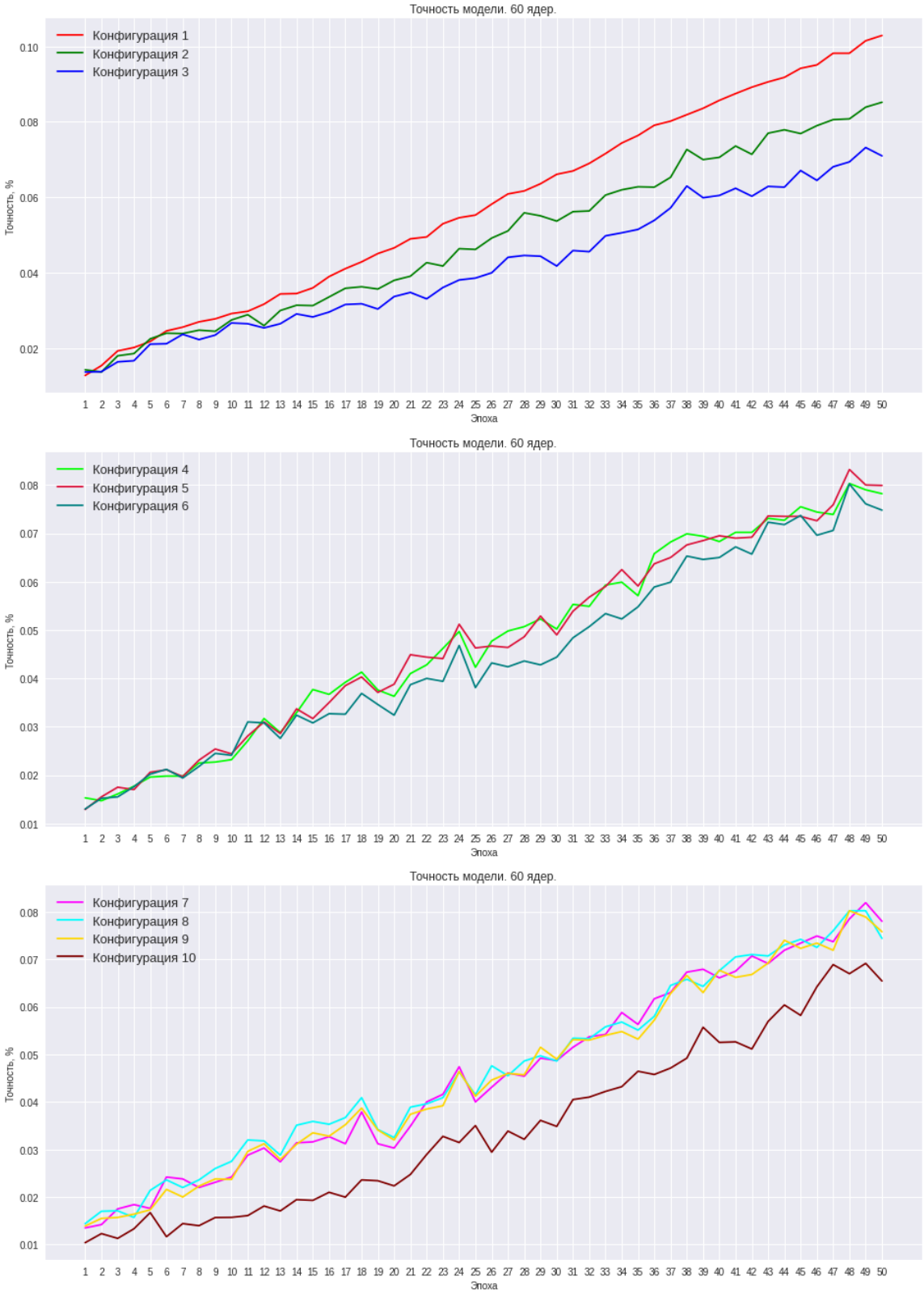


Рис. 11: CIFAR-100. Точность модели в разных конфигурациях при обучении на 60 ядрах

В связи с фиксацией при вычислениях последовательности псевдослучайных чисел, результаты измерения точности совпадают с полученными в ходе первой серии экспериментов. Результаты измерения временных затрат при обучении на 60 ядрах приведены на рисунке 12. Остальные результаты находятся в приложении.

5.3 Выводы

На основании результатов экспериментов можно сделать ряд выводов об эффективности исследуемых методов параллельного обучения.

Local-SGD показывает значительное ускорение коммуникаций, зависящее от выбранного периода синхронизаций. Использование данного метода снижает точность модели на 2-4%, однако при выборе большого периода синхронизаций точность падает значительно.

Метод gossip позволяет ускорить коммуникации в 2-5 раз, при этом точность модели падает приблизительно на 2%, если не использованы другие методы.

Добавление к gossip спарсификации ускоряет коммуникации еще в 2-10 раз (в некоторых экспериментах ускорение доходило до 20). При использовании периода синхронизации $p = 1$, спарсификация дополнительно ухудшает точность модели на 1-2%.

Однако при использовании Local-SGD с периодом синхронизации $p > 1$ добавление gossip и спарсификации практически не ухудшает точность модели. В то же время ускорение коммуникаций при применении комбинации методов достигает наиболее высоких значений.

Наиболее эффективной оказалась конфигурация 9. Она ускоряет коммуникации в среднем в 40 раз, при этом точность модели падает на 1-3%.

При обучении нейронной сети на вычислительном комплексе Polus использование исследуемых методов не дает существенного ускорения. Это связано с тем, что при выполнении вычислений на Polus время, затраченное на коммуникации, невелико в сравнении с общим временем обучения.

Рис. 12: Временные затраты при обучении на 60 ядрах в разных конфигурациях. ЛОМОНОСОВ-2

Однако применение исследуемых методов на ЛОМОНОСОВ-2 позволяет ускорить обучение сети в 2-3 раза. Исходя из этого можно сделать вывод о высокой эффективности данных параллельных методов при проведении обучения на системах с медленными коммуникациями.

6 Заключение

В данной работе были рассмотрены параллельные методы обучения нейронных сетей. Были исследованы методы Local-SGD, gossip и спарсификация и сделан вывод о высокой эффективности данных методов при проведении обучения на системах с медленными коммуникациями. В то же время на системах с быстрыми коммуникациями данные методы могут оказаться неэффективными. Это говорит о важности правильного подбора методов параллельного обучения в зависимости от архитектуры используемой вычислительной системы.

Также было показано, что исследованные методы следует использовать совместно, т. к. это сильно ускоряет обучение нейронной сети, при этом незначительно ухудшая ее точность.

6.1 Направления дальнейших исследований

Проведенные вычислительные эксперименты показывают, что значительную долю в параллельном обучении занимает синхронизация вычислительных устройств. Даже в однородных системах, состоящих из равных по производительности узлов, время, затраченное на синхронизацию, может превышать время, затраченное на вычисления. Это говорит об актуальности и важности направления исследований, посвященному асинхронным методам параллельного обучения.

Другим важным направлением является исследование реализаций параллельных алгоритмов с использованием иных технологий параллельного программирования. Например, можно добиться повышения эффективности алгоритмов, используя гибридный подход MPI+OpenMP. Необходимо также исследовать эффективность ме-

тодов при проведении обучения на графических ускорителях с использованием технологии CUDA.

Список литературы

[1] Communication-Efficient Distributed Deep Learning: A Comprehensive Survey / Z. Tang, S. Shi, X. Chu et al. — 2020.

[2] *Gibiansky A.* Bringing HPC techniques to deep learning. . http://research.baidu.com/bringing-hpc-techniques-deep-learning. — 2017.

[3] GossipGraD: Scalable Deep Learning using Gossip Communication based Asynchronous Gradient Descent / J. Daily, A. Vishnu, C. Siegel et al. — 2018.

[4] Integrated model, batch, and domain parallelism in training neural networks / A. Gholami, A. Azad, P. Jin et al. — 2018.

[5] QSGD: Communication-Efficient SGD via Gradient Quantization and Encoding / D. Alistarh, D. Grubic, J. Li et al. — 2017.

[6] *Yu H., Yang S., Zhu S.* Parallel Restarted SGD with Faster Convergence and Less Communication: Demystifying Why Model AveragingWorks for Deep Learning. — 2018.

7 Приложение

В данном разделе, начиная со следующей страницы, приведены результаты исследований:

- Точность сети, обученной на CIFAR-10 при использовании 16 ядер
- Точность сети, обученной на CIFAR-10 при использовании 32 ядер
- Точность сети, обученной на CIFAR-100 при использовании 16 ядер
- Точность сети, обученной на CIFAR-100 при использовании 32 ядер
- Время коммуникаций, синхронизаций и общее время обучения на 16 ядрах на IBM Polus
- Время коммуникаций, синхронизаций и общее время обучения на 32 ядрах на IBM Polus
- Время коммуникаций, синхронизаций и общее время обучения на 32 ядрах на ЛОМОНОСОВ-2
- Время коммуникаций, синхронизаций и общее время обучения на 210 ядрах на ЛОМОНОСОВ-2

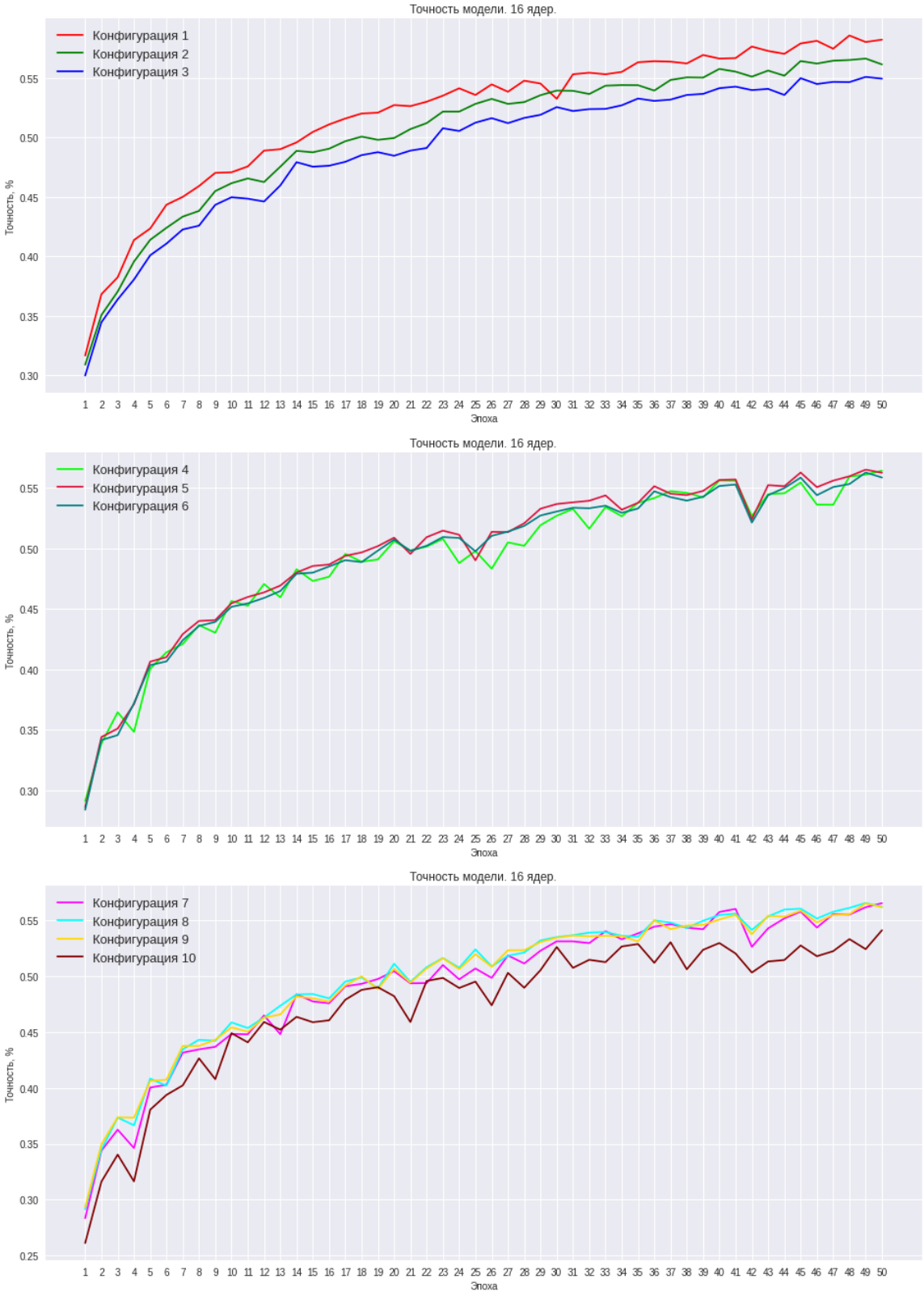


Рис. 13: CIFAR-10. Точность модели в разных конфигурациях при обучении на 16 ядрах

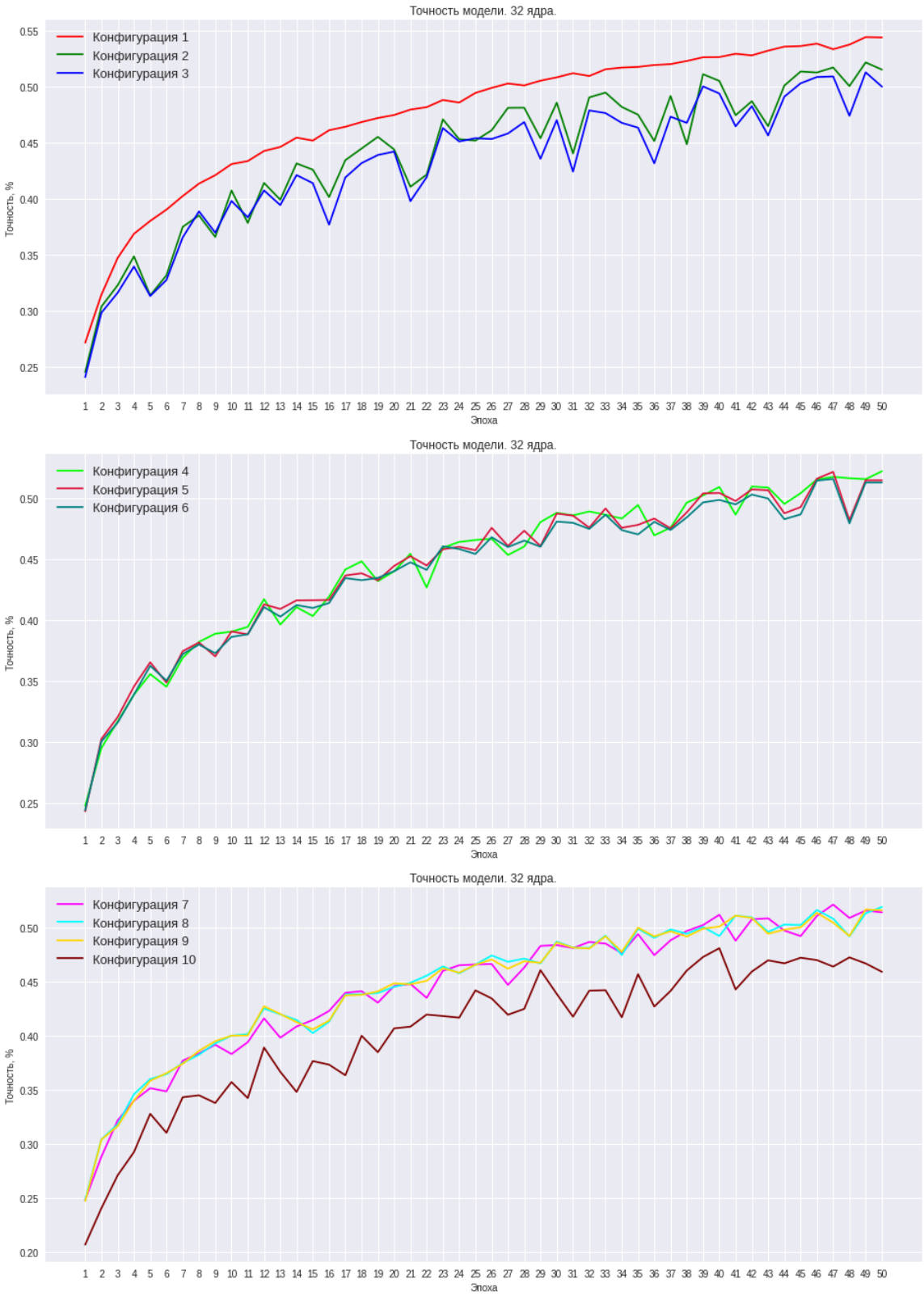


Рис. 14: CIFAR-10. Точность модели в разных конфигурациях при обучении на 32 ядрах

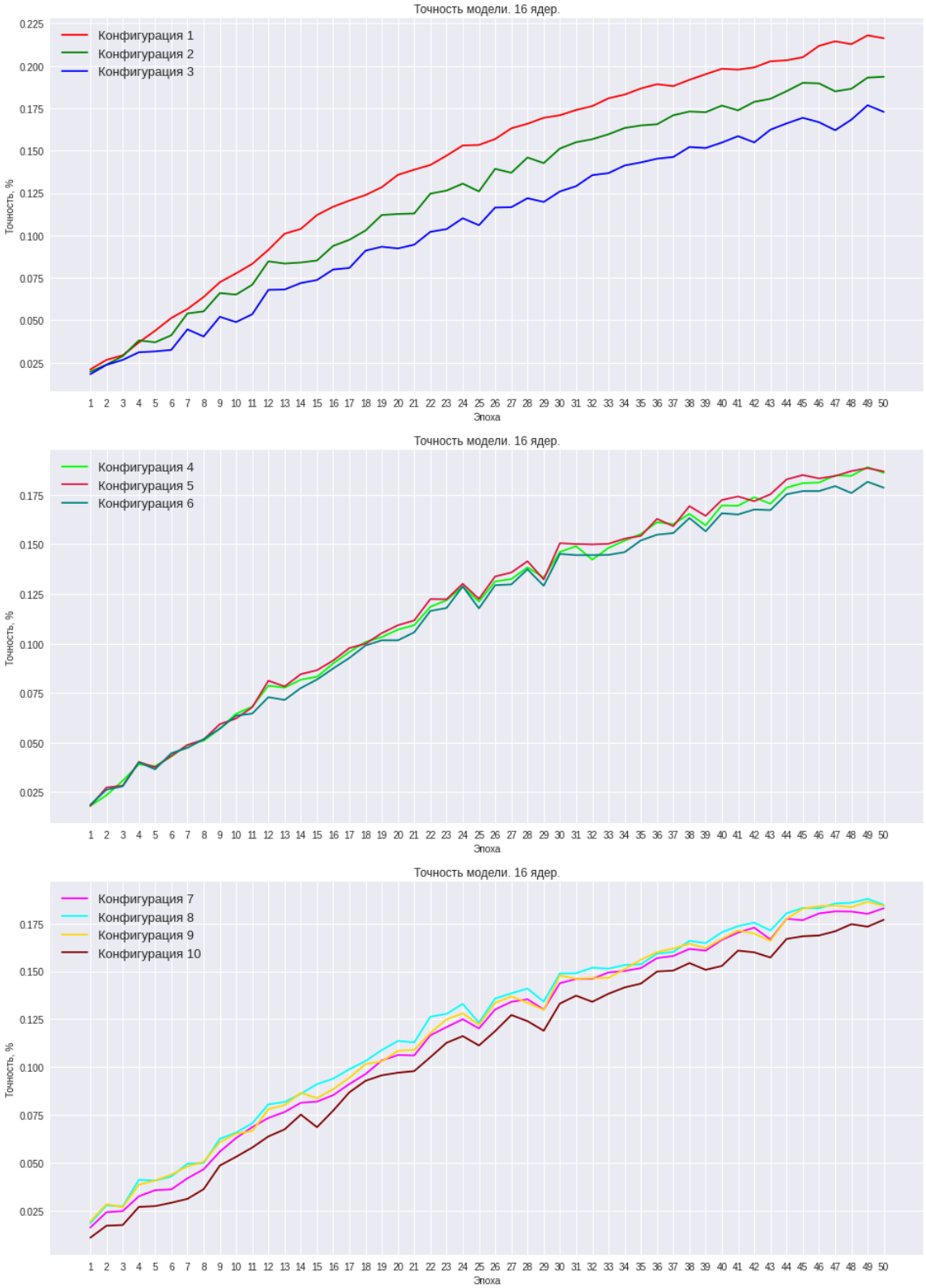


Рис. 15: CIFAR-100. Точность модели в разных конфигурациях при обучении на 16 ядрах

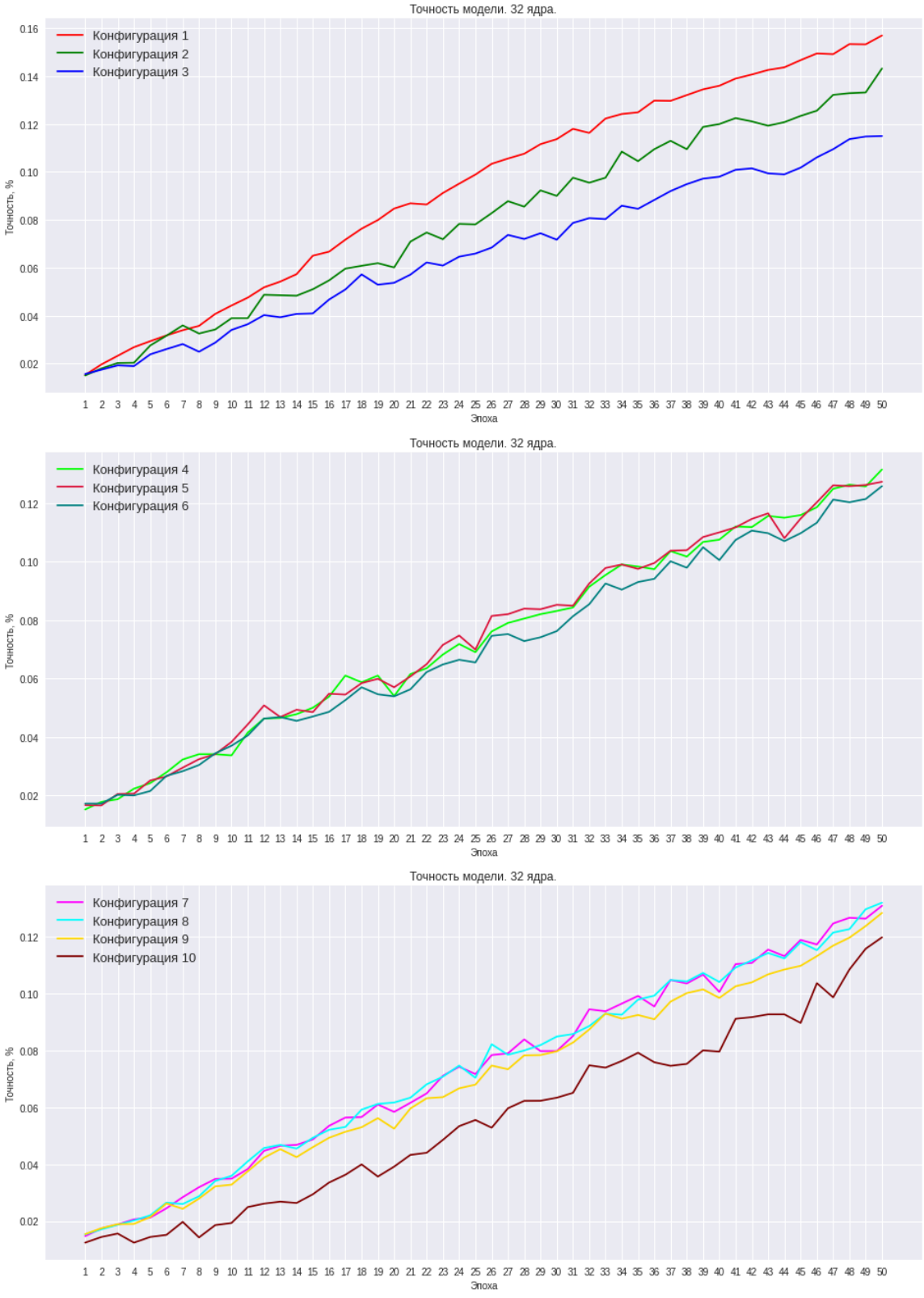


Рис. 16: CIFAR-100. Точность модели в разных конфигурациях при обучении на 32 ядрах

Рис. 17: Временные затраты при обучении на 16 ядрах в разных конфигурациях. IBM Polus

Рис. 18: Временные затраты при обучении на 32 ядрах в разных конфигурациях. IBM Polus

Рис. 19: Временные затраты при обучении на 32 ядрах в разных конфигурациях. ЛОМОНОСОВ-2

Рис. 20: Временные затраты при обучении на 210 ядрах в разных конфигурациях. ЛОМОНОСОВ-2