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

РАБОТА ПРОВЕРЕНА

Рецензент
Доцент кафедры ТУиО
ФГБОУ ВО «ЧелГУ», к.ф.-м.н.

_____ С.А. Никитина

«___»_____ 2024 г.

ДОПУСТИТЬ К ЗАЩИТЕ

Заведующий кафедрой, д.ф.-м.н., профессор

_____ Л.Б. Соколинский

«___»_____ 2024 г.

Применение метода проектирования Q-эффективных программ к методу стохастического градиентного спуска для обучения нейронных сетей

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА
ЮУрГУ – 09.04.04.2024.308-1497.ВКР

Научный руководитель,
доцент кафедры СП, к.ф.-м.н.
_____ В.Н. Алеева

Автор работы,
студент группы КЭ-229
_____ А.С. Сапожников

Ученый секретарь
(нормоконтролер)
_____ И.Д. Володченко
«___»_____ 2024 г.

Челябинск, 2024 г.

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

УТВЕРЖДАЮ
Зав. кафедрой СП
_____ Л.Б. Соколинский
29.01.2024 г.

ЗАДАНИЕ
на выполнение выпускной квалификационной работы магистранта
студенту группы КЭ-229
Сапожникову Андрею Сергеевичу,
обучающемуся по направлению
09.04.04 «Программная инженерия»
(магистерская программа «Искусственный интеллект и инженерия данных»)

1. **Тема работы** (утверждена приказом ректора от 22.04.2024 г. № 764-13/12)
Применение метода проектирования Q-эффективных программ к методу стохастического градиентного спуска для обучения нейронных сетей.
2. **Срок сдачи студентом законченной работы:** 20.05.2024 г.
3. **Исходные данные к работе**

3.1. Алеева В.Н. Архитектура вычислительных систем: курс лекций. – Челябинск: Издательский центр ЮУрГУ, 2023. – 150 с.

3.2. Aleeva V.N., Aleev R.Zh. Investigation and Implementation of Parallelism Resources of Numerical Algorithms. // ACM Transactions on Parallel Computing. 2023. – 1–64 pp.

3.3. Антонов А.С. Параллельное программирование с использованием технологии OpenMP: Учебное пособие. – М.: Изд-во МГУ, 2009. – 77 с.

3.4. Антонов А.С. Параллельное программирование с использованием технологии MPI: Учебное пособие. – М.: Изд-во МГУ, 2004. – 71 с.

4. **Перечень подлежащих разработке вопросов**

4.1. Анализ предметной области.

4.2. Применение метода проектирования Q-эффективных программ к методу стохастического градиентного спуска и методу обратного распространения ошибки.

4.3. Тестирование разработанных Q-эффективных программ.

4.4. Экспериментальное исследование разработанных Q-эффективных программ.

5. Дата выдачи задания: 29.01.2024 г.

Научный руководитель,
доцент кафедры СП, к.ф.-м.н. В.Н. Алеева

Задание принял к исполнению А.С. Сапожников

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Актуальность

Технологии искусственного интеллекта развиваются стремительными темпами. Однако все эти технологии требуют больших вычислительных мощностей, что тормозит дальнейшие исследования. Одним из способов решения этой проблемы является использование параллельных вычислений. Так можно, например, ускорить обучение нейронных сетей.

Объектом исследований был выбран метод проектирования Q-эффективных программ, а предметом исследований – метод стохастического градиентного спуска, применяемый для оптимизации целевой функции нейронной сети.

Актуальность данной темы обоснована недостаточной изученностью темы применения концепции Q-детерминанта в области нейронных сетей.

Постановка задачи

Целью выпускной квалификационной работы является применение метода проектирования Q-эффективных программ к методу стохастического градиентного спуска для нейронных сетей. Для достижения данной цели необходимо выполнить следующие задачи.

1. Провести анализ предметной области.
2. Применить метод проектирования Q-эффективных программ к методу стохастического градиентного спуска и методу обратного распространения ошибки.
3. Протестировать разработанные Q-эффективные программы.
4. Провести экспериментальное исследование разработанных Q-эффективных программ.

Структура и содержание работы

Работа состоит из введения, четырех глав, заключения, списка литературы и двух приложений. Объем работы составляет 43 страницы, объем списка литературы – 20 источников.

В первой главе рассматривается концепция Q-детерминанта и метод проектирования Q-эффективных программ, также процесс обучения нейронной сети. Рассматривается метод стохастического градиентного спуска и метод обратного распространения ошибки.

Вторая глава посвящена разработке, архитектуры нейронной сети, Q-эффективных программ, реализующих метод стохастического градиентного спуска и метод обратного распространения ошибки для систем с общей и распределенной памятью.

В третьей главе описан порядок запуска Q-эффективных программ и приведены примеры запуска Q-эффективных программ.

Четвертая глава содержит результаты вычислительных экспериментов Q-эффективных программ для систем с общей и распределенной памятью. Выполняется построение графиков зависимостей времени выполнения, ускорения, эффективности и от количества ядер и размера архитектуры нейронной сети.

В заключении подводятся итоги данного исследования.

В приложении содержатся результаты вычислительных экспериментов Q-эффективной программы для системы с общей памятью и распределенной памятью

1. АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

1.1. Исследование и реализация ресурса параллелизма численных алгоритмов

История развития параллельных вычислительных систем (ВС) насчитывает десятки лет, однако проблема эффективной реализации на них алгоритмов остается низкой. При ее решении следует исходить из того, что первичным является алгоритм, а архитектура ВС должна быть приспособлена к структуре алгоритма. Предлагаемый подход, с одной стороны, может быть использован для повышения эффективности выполнения алгоритмов на реальных ВС, с другой стороны, используемое в нем описание структуры алгоритмов выявляет требования к архитектуре ВС, согласующейся со структурой алгоритмов.

В 1985 году В.Н. Алеевой было разработано представление алгоритма в форме Q-детерминанта, которое положено в основу концепции Q-детерминанта [1,2]. Концепция Q-детерминанта позволяет найти эффективную реализацию алгоритма и исследовать его ресурс параллелизма.

Концепция Q-детерминанта

Концепция Q-детерминанта позволяет представить любой численный алгоритм в форме Q-детерминанта, найти реализацию алгоритма, полностью использующую его ресурс параллелизма, проверить возможность выполнения алгоритма на параллельной вычислительной системе и оценить характеристики ресурса параллелизма алгоритма: высоту и ширину.

Рассмотрим алгоритмическую проблему (формула 1):

$$\vec{y} = F(N, B), \qquad (1)$$

где N – множество параметров размерности проблемы;

B – множество входных данных, $\vec{y} = (y_1, \cdots, y_m)$ – множество выходных данных;

$y_i \notin$ B для каждого $i \in \{1, \cdots, m\}$, целое число m является либо константой, либо значением вычислимой параметрической функции от N ($N \neq \emptyset$).

Пусть α – численный алгоритм для решения алгоритмической проблемы $\vec{y} = F(N, B)$ и $M = \{1, \cdots, m\}$. Предположим, что алгоритм α состоит в том, чтобы для каждого $i \in M$ нужно найти y_i, как значение Q-терма f_i.

По Q-термом следует понимать выражение w над сочетанием конечного или счетного множества переменных B и конечного множества операций Q.

Тогда множество Q-термов $\{f_i | i \in M\}$ называется Q-детерминантом алгоритма α. Также система уравнений $y_i = f_i$ для всех $i \in M$ называется представлением алгоритма α в форме Q-детерминанта.

При этом множество M может состоять из нескольких подмножеств:

1) безусловные Q-термы;
2) условные Q-термы;
3) условно бесконечные Q-термы.

Одно или несколько подмножеств могут быть пустыми.

Метод проектирования Q-эффективных программ

Q-эффективная реализация алгоритма основана на следующих утверждениях:

1) для любого численного алгоритма можно построить Q-детерминант;
2) Q-детерминант позволяет описать Q-эффективную реализацию алгоритма;
3) если Q-эффективная реализация алгоритма является выполнимой, то можно разработать программу для ее выполнения.

Метод проектирования Q-эффективных программ использует расширенную модель концепции Q-детерминанта и состоит из этапов [3–5].

1. Построение Q-детерминанта алгоритма.
2. Описание Q-эффективной реализации алгоритма.
3. Разработка параллельной программы для выполнимой Q-эффективной реализации алгоритма.

На первых двух этапах метода используется базовая модель концепции Q-детерминанта, а на третьем этапе – расширенная модель. Последняя отличается от базовой тем, что учитывает особенности выполнения алгоритмов на реальных вычислительных системах. Новая расширенная модель концепции Q-детерминанта получается добавлением моделей параллельных вычислений: PRAM для общей памяти и BSP для распределенной памяти.

Программа, полученная с помощью данного метода, была названа Q-эффективной, а процесс ее разработки Q-эффективным программированием. Так как Q-эффективная программа выполняет Q-эффективную реализацию алгоритма, то она полностью использует ресурс параллелизма алгоритма. Таким образом, Q-эффективная программа имеет самый высокий параллелизм среди программ, реализующих алгоритм.

Для разработки Q-эффективной программы для общей памяти достаточно описания Q-эффективной реализации алгоритма. При разработке Q-эффективной программы для распределенной памяти следует учитывать, что данные должны обладать свойством локальности, иначе могут возникнуть многочисленные промахи кэша при их считывании из памяти, что приведет к снижению быстродействия.

Обеспечить локальность данных позволяет распределение вычислений между вычислительными узлами вычислительной системы, основанное на описании Q-эффективной реализации алгоритма. Для распределения вычислений между вычислительными узлами применяется принцип «master-slave». При этом используется один вычислительный узел «master» и несколько вычислительных узлов «slave». Узел «master» обозначается буквой M, а множество узлов «slave» – буквой S. При разработке Q-эффективных программ используется язык программирования C++, технология OpenMP для ВС с общей памятью, технологии MPI и OpenMP для ВС с распределенной памятью.

1.2. Нейронные сети прямого распространения

Нейронная сеть – это математическая модель, построенная по принципу организации биологических нейронных сетей. Нейронные сети являются универсальным инструментом и способны решать разного рода задачи: распознавание образов и классификация, прогнозирование, кластеризация.

В зависимости от задачи применяют разные типы архитектур нейронных сетей. Под архитектурой понимается общая структура сети: сколько в ней должно быть блоков (по-другому, слоев) и как эти блоки связаны между собой [7]. Нейронная сеть прямого распространения является одной из первых появившихся архитектур. Пример этой архитектуры представлен на рисунке 1.

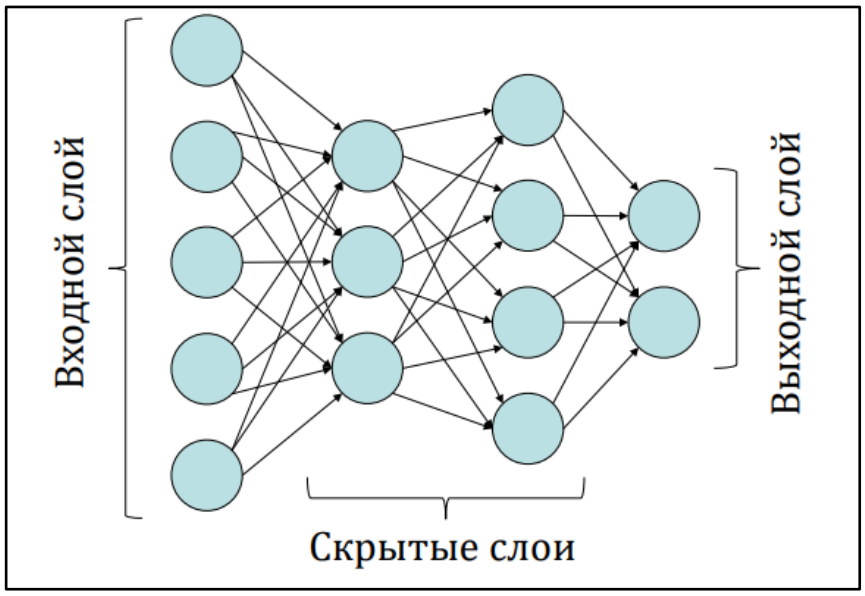

Рисунок 1 – Пример архитектуры нейронной сети прямого распространения [20]

Как можно увидеть из рисунка 1, нейронная сеть прямого распространения имеет входной слой, несколько скрытых слоев и выходной слой. Каждый нейрон текущего слоя связан со всеми нейронами предыдущего слоя, поэтому такую сеть еще называют полносвязной. Эти связи выражаются в

виде синаптических весов. Для текущего слоя l все связи будут выражены в виде матрицы синаптических весов $W^{(l)}$ (формула 2):

$$W^{(l)} = \begin{pmatrix} w_{11} & \cdots & w_{1r} \\ \vdots & \ddots & \vdots \\ w_{h1} & \cdots & w_{hr} \end{pmatrix}, \quad (2)$$

где h – количество нейронов в слое $l-1$; r – количество нейронов в слое l.

В математическом смысле каждый полносвязный слой нейронной сети может быть описан формулой (3):

$$A^{(l)} = activation\left(Z^{(l)}\right), \quad (3)$$

где $A^l = \{a_1, \dots, a_r\}$ – вектор выходных сигналов нейронной сети текущего слоя l, $activation$ – функция активации нейрона, $Z^{(l)}$ – активационный потенциал слоя l.

Функция активации нейрона представляет собой математическую модель реакции нейрона на входные данные.

Активационный потенциал $Z^{(l)}$ вычисляется по формуле (4):

$$Z^{(l)} = A^{(l-1)} \times W^{(l)} + B^{(l)}, \quad (4)$$

где $B^{(l)} = \{b_1, \dots, b_r\}$ – вектор смещений слоя l; $W^{(l)}$ – матрица весов слоя l.

Если матрица весов $W^{(l)}$ необходима для передачи сигнала от предыдущего слоя к текущему, то вектор смещений $B^{(l)}$ позволяет смещать значение функции активации влево или вправо, что может иметь решающее значение для успешного обучения.

Под обучением нейронной сети понимается подбор весов W и смещений B таким образом, чтобы увеличить точность работы нейронной сети при выполнении текущей задачи.

Помимо весов и смещений при обучении используется набор данных, который состоит из наборах входных значений $X = \{x_1, \cdots, x_k\}$, где k – размер набора и, если перед нейронной сетью стоит задача классификации, метка класса, принадлежащая этому набору входных значений.

Алгоритм обучения следующий.

1. Нейронам первого слоя присваиваются значения одного набора входных значений X.

2. Далее осуществляется прямое распространение, то есть вычисляются активационные потенциалы и выходные значения всех нейронов всех слоев по формулам (3) и (4).

3. По последнему слою определяется класс, распознанный нейронной сетью. Распознавание осуществляется путем выбора нейрона с максимальным значением выходного сигнала a, среди нейронов последнего слоя L.

4. Далее дается оценка эффективности распознавания с помощью стоимостной функции.

5. В зависимости от значения функции потерь осуществляется подбор новых весов и смещений так, чтобы свести значение стоимостной функции к минимуму.

Процесс нахождения минимума функции называется оптимизацией, а саму функцию в этом случае называют целевой. Для этих целей используют различные методы оптимизации, речь о которых пойдет в следующей главе.

1.3. Выбор метода оптимизации целевой функции нейронной сети

Для оптимизации целевой функции используются различные методы. Приведем некоторые из них [6].

1. Различные реализации метода градиентного спуска.
2. Импульсные методы.
3. Адаптивные методы.

Среди предложенных реализаций стохастический градиентный спуск (СГС), является одним из самых точных. Поэтому к нему будет применен метод проектирования Q-эффективных программ.

Метод стохастического градиентного спуска [7]

Перед началом обучения обучающая выборка разделяется на несколько мини-пакетов, каждый из которых содержит одинаковое количество наборов входных значений $X = \{X_1, \cdots, X_v\}$ и правильных меток класса $Y = \{y_1, \cdots, y_v\}$, где v – размер мини-пакета.

На каждом шаге обучения (по-другому, эпохе) нейронная сеть проходит по всем наборам входных значений всех мини-пакетов. После каждого пройденного набора входных значений определяется точность нейронной сети с помощью стоимостной функции. Для данной работы воспользуемся функцией, рассчитывающей среднеквадратическую ошибку (формула 5):

$$C = \frac{1}{|v|} \sum_{(x,y)\in \mathrm{v}} \frac{|A^L - y|^2}{2}, \tag{5}$$

где $A^L = \{a_1, \dots, a_r\}$ – вектор выходных сигналов нейронной сети последнего слоя, длина которого равна количеству нейронов m на данном слое l.

Обычно в качестве функций активации [8] используют сигмоиду или линейный выпрямитель. Для данной задачи возьмем сигмоиду, которая вычисляется по формуле (6):

$$\sigma(z) = \frac{1}{1 + e^{-z}}. \tag{6}$$

Далее на основе полученных результатов вычисляются градиенты для всех весов W и смещений B по формулам (7) и (8):

$$W^{*(l)} = W^{(l)} - \varepsilon \cdot \nabla_W C^{(l)}, \tag{7}$$

$$B^{*(l)} = B^{(l)} - \varepsilon \cdot \nabla_B C^{(l)}, \tag{8}$$

где ε – скорость обучения, положительный скаляр, определяющий длину шага, $\nabla_W C^{(l)}$ и $\nabla_B C^{(l)}$ – матрицы частных производных целевой функции C.

Метод обратного распространения ошибки [9]

Данный метод используется для вычисления градиентов $\nabla_W C^{(l)}$ и $\nabla_B C^{(l)}$, применяемых в методе СГС.

После обработки нейронной сети одного набора входных значений X,

вычисляется мера влияния нейронов выходного слоя на величину ошибки δ^L по формуле (9):

$$\delta^L = \frac{\partial C}{\partial A^L} \cdot \sigma'(Z^L). \quad (9)$$

Для среднеквадратической ошибки формула (9) выглядит следующим образом (формула 10):

$$\delta^L = (A^L - y) \cdot \sigma'(Z^L). \quad (10)$$

Далее рассчитывается мера влияния нейронов каждого слоя l от $L-1$ слоя и до первого по формуле (11):

$$\delta^{(l)} = \sigma'\left(Z^{(l)}\right) \cdot (\delta^{(l+1)} \cdot W^{(l+1)\,T}). \quad (11)$$

По полученным значения рассчитываются градиенты весов и смещений для каждого слоя по формулам (12) и (13):

$$\nabla_W C^{(l)} = \delta^{(l)} \cdot A^{(l-1)}, \quad (12)$$

$$\nabla_B C^{(l)} = \delta^{(l)}. \quad (13)$$

1.4. Архитектура суперкомпьютера «Торнадо ЮУрГУ»

Суперкомпьютер «Торнадо ЮУрГУ» представляет собой вычислительный кластер, который состоит из 480 вычислительных узлов и двух серверов (управления и доступа пользователей), объединенных между собой с помощью транспортной сети Infiniband и двух Ethernet сетей (мониторинга и управления заданиями) [10].

Вычислительные узлы имеют сквозную нумерацию вида «nodeXXX» (где X – число от 001 до 480), где каждый узел имеет в своем составе:

1) сопроцессор Intel Xeon Phi, имеющий имя вида «nodeXXX-mic0» (внутри вычислительного узла возможна адресация по «mic0»);

2) сетевой интерфейс Infiniband (имя Ib0 в пределах хоста или nodeXXX-ib0 в пределах кластера);

3) сетевой интерфейс Ethernet (eth0).

На таблице 1 представлены основные технические характеристики вычислительной системы.

Таблица 1 – Технические характеристики вычислительной системы

Параметр	**Значение для одного узла**	**Значение для Вычислителя в целом**
Теоретическая пиковая производительность	1,447 Тфлопс	694,56 Тфлопс
Теоретическая пиковая производительность процессоров Intel Xeon X5680	0,371 Тфлопс	178,08 Тфлопс
Теоретическая пиковая производительность сопроцессоров Intel Xeon Phi	1,076 Тфлопс	206,592 Тфлопс
Количество сопроцессоров Intel Xeon Phi 7110X	1 (node[001-192])	192 (node[001-192])
Объем памяти сопроцессора Intel Xeon Phi	8 Гб	1 536 Гб
Тип используемых процессорных чипов	Intel Xeon X5680 6 ядер, тактовая частота ядра 3,33 ГГц, QPI 8,0 ГТ/с, размер кэша 12 МБ	
Характеристики коммуникационной и транспортной сети Вычислителя	Infiniband QDR 40 Гбит/с с полной бисекционной пропускной способностью.	
Сеть Gigabit Ethernet управления заданиями и мониторинга	Обеспечение доступа ко всем вычислительным узлам и управляющему серверу со скоростью 1 Гбит/с	
Количество процессорных чипов	2	960
Количество процессорных ядер	12	5760
Объем памяти	24 Гб (4 ГБ на ядро) на node[192-480], 48 Гб (8 ГБ на ядро) на node[001-192]	16128 ГБ
Количество узлов	–	480
Тип дисков	80 ГБ SATA SSD	
Количество дисков	1	480

В состав вычислительной системы входят Вычислитель, сервер управления Вычислителем, сервер доступа пользователей и система хранения данных. Для связи компонентов в единую систему и с сетевой инфраструктурой ЮУрГУ используется несколько сетей, каждая из которых служит для выполнения определенных функций.

Вычислитель представляет собой кластерную систему из множества узлов, объединенных между собой высокопроизводительной коммуникационной сетью, реализованной по технологии Infiniband QDR. Коммуникационная сеть поддерживает реализацию основных примитивов библиотеки

MPI и построена по топологии полной бисекционной пропускной способности.

Все узлы Вычислителя, сервер управления и сервер доступа пользователей, подключены к двум сетям Ethernet, одна из которых используется для управления и мониторинга вычислительных узлов, а вторая для управления заданиями.

Сопроцессором Intel Xeon Phi 7110X оснащены 192-а узла Вычислителя. Объем и тип оперативной памяти каждого сопроцессора 8 ГБ GDDR5. Данные сопроцессоры предназначены для ускорения выполнения команд x86 с векторными расширениями.

Выводы по первой главе

В результате проведенной работы была изучена реализация ресурса параллелизма численных алгоритмов на основе концепции Q-детерминанта, а также метод проектирования Q-эффективных программ.

Изучен алгоритм работы полносвязной нейронной сети. Для исследований был выбран метод стохастического градиентного спуска и метод обратного распространения ошибки.

Изучена архитектура суперкомпьютера «Торнадо ЮУрГУ», на котором будет производиться исследование.

2. РЕАЛИЗАЦИЯ

2.1. Построение Q-детерминанта для метода стохастического градиентного спуска

Определение алгоритмической проблемы

Построим Q-детерминант алгоритма стохастического градиентного спуска. Для этого определим алгоритмическую проблему.

Задача стохастического градиентного спуска – это оптимизация весов и смещений нейронной сети каждого слоя. Следовательно, на выходе этот алгоритм должен выдавать матрицу весов и вектор смещений (формула 14):

$$\vec{y}^{(l)} = \left(W^{(l)*}, B^{(l)*}\right), \tag{14}$$

где $W^{(l)*}$ – матрица новых весов слоя l, $B^{(l)*}$ – вектор новых смещений слоя l.

Исходя из формул (7) и (8), входными данными будут старая матрица весов $W^{(l)}$ и вектор смещений $B^{(l)}$ нейронной сети, а также градиенты весов и смещений слоя l (формула 15):

$$B = \left(W^{(l)}, B^{(l)}, \nabla_W C^{(l)}, \nabla_B C^{(l)}\right). \tag{15}$$

К параметрам размерности проблемы относится количество нейронов на слое l – r и на слое l минус 1 – h (формула 16):

$$N = (r, h). \tag{16}$$

Таким образом, алгоритмическая проблема имеет следующий вид (формула 17):

$$\left(W^{(l)*}, B^{(l)*}\right) = F\left((r, h), \left(W^{(l)}, B^{(l)}, \nabla_W C^{(l)}, \nabla_B C^{(l)}\right)\right). \tag{17}$$

Построение Q-детерминанта

Алгоритм стохастического градиентного спуска находит новый вес из матрицы $W^{(l)}$ по формуле (18):

$$w_{ij}^{(l)*} = w_{ij}^{(l)} - \varepsilon \cdot \nabla_w C_{ij}^{(l)}, \tag{18}$$

где $i \in H$ – количество нейронов на слое l минус 1, $j \in R$ – количество нейронов на слое l.

Для смещений алгоритм представлен аналогичным образом (формула 19):

$$b_i^{(l)*} = b_i^{(l)} - \varepsilon \cdot \nabla_b C_i^{(l)}, \quad (19)$$

где $i \in R$ – количество нейронов на слое l.

С учетом формул (12) и (13) Q-детерминанты для весов и смещений будут выглядеть следующим образом (формулы 20 и 21):

$$w_{ij}^{(l)*} = w_{ij}^{(l)} - \varepsilon \cdot \delta_i^{(l)} \cdot a_j^{(l-1)}, \quad (20)$$

$$b_i^{(l)*} = b_i^{(l)} - \varepsilon \cdot \delta_i^{(l)}. \quad (21)$$

Описание Q-эффективной реализации

Будем вычислять $\left\{\{w_{00}^*, \dots, w_{0j}^*\}, \dots, \{w_{i0}^*, \dots, w_{ij}^*\}\right\}$, где $i \in H, j \in R$ одновременно. Аналогичным образом поступим и с $\{b_0^*, \dots, b_i^*\}$, где $i \in R$. Данная реализация подходит для системы с общей памятью.

Таким образом, Q-детерминант метода стохастического градиентного спуска представлен в виде двух множеств безусловных Q-терм, для весов и смещений соответственно.

2.2. Построение Q-детерминанта для метода обратного распространения ошибки

Определение алгоритмической проблемы

Построим Q-детерминант алгоритма обратного распространения ошибки. Для этого определим алгоритмическую проблему.

Данный метод используется для вычисления градиентов меры влияния нейронов на величину ошибки слоя l. Следовательно, на выходе этот алгоритм должен выдавать вектор меры влияния нейронов на величину ошибки (формула 22):

$$\vec{y}^{(l)} = \left(\delta^{(l)}\right). \quad (22)$$

Так как для выходного слоя входные данные и формулы отличны, он не будет включен в Q-детерминант. Поэтому, исходя из формулы (11), входными данными будут вектор активационного потенциала $Z^{(l)}$ и вектор меры

влияния нейронов на величину ошибки $\delta^{(l+1)}$ и матрица весов $W^{(l)}$ (формула 23):

$$B = \left(Z^{(l)}, \delta^{(l+1)}, W^{(l)}\right). \quad (23)$$

К параметрам размерности проблемы относится количество нейронов на слое l – r и на слое l плюс 1 – k (формула 24):

$$N = (r, k). \quad (24)$$

Таким образом, алгоритмическая проблема имеет следующий вид (формула 25):

$$\left(\delta^{(l)}\right) = F\left((r, k), \left(Z^{(l)}, \delta^{(l+1)}, W^{(l)}\right)\right). \quad (25)$$

Построение Q-детерминанта

Алгоритм метода обратного распространения ошибки вычисляет меру влияния нейрона на величину ошибки следующим образом (формула 26):

$$\delta^{(l)}{}_i = \sigma'\left(z^{(l)}{}_i\right) \cdot \left(\delta^{(l+1)}{}_j \cdot w^{(l+1)}{}_{ji}\right), \quad (26)$$

где $i \in R$ – количество нейронов на слое l, $j \in K$ – количество нейронов на слое l плюс 1.

Описание Q-эффективной реализации

Будем вычислять $\left\{\delta^{(l)}{}_0, \dots, \delta^{(l)}{}_i\right\}$, где $i \in R$, одновременно. Данная реализация подходит для системы с общей памятью.

Таким образом, Q-детерминант метода обратного распространения ошибки представлен в виде множества безусловных Q-терм.

2.3. Описание Q-эффективной реализации для системы с распределенной памятью

Поскольку ранее описанные методы используют одни и те же данные, Q-эффективная реализация для систем с распределенной памятью будет предусматривать их совместное использование.

Реализацию для системы с распределенной памятью опишем с использованием принципа «master-slave». Узел «master» обозначим буквой M, а узел «slave» – буквой S.

Общими данными для методов СГС и обратного распространения ошибки являются матрица весов W, вектор выходных сигналов нейронов A и вектор активационных потенциалов Z. Из формулы (3) становится ясно, что вектор A может быть легко получен из вектора Z. Поэтому общими данными остаются только матрица весов W и вектор активационных потенциалов Z.

В целях оптимизации распределения данных между узлами S будут разделены так, что матрица W будет разделена на строки, и каждому узлу S будет передано некоторое количество строк $\{\{w_0, \dots, w_r\}_0, \dots, \{w_0, \dots, w_r\}_{i+t}\}$, t – количество строк, которые будут переданы каждому узлу. Вектор Z будет разделен так, что каждому узлу достанется некоторое количество элементов $\{z_i, \dots, z_{i+t}\}$. Количество строк $\{w_0, \dots, w_j\}$ и элементов z_i будет передано в одинаковом количестве.

Теперь перейдем к частным данным, которые нужны каждому методу отдельно. Метод СГС также использует вектор смещений B, размерность которого такая же, как и у вектора Z. Поэтому вектор B будет распределен между узлами S перед началом вычислений аналогично вектору Z. Вектор меры влияния нейронов на величину ошибки $\{\delta_0, \dots, \delta_j\}$ необходимо передать всем узлам S целиком, так как этого требуют вычисления. Передача всех общих и частных данных для метода СГС будет осуществлять либо перед началом обучения нейронной сети, либо перед началом работы метода.

Для метода обратного распространения ошибки требуется вектор меры влияния нейронов на величину ошибки $\{\delta_0, \dots, \delta_j\}$, полученный из предыдущей итерации этого же метода, причем в полном объеме. Распределим вычисление этого вектора так, чтобы каждый узел нашел некоторое количество элементов этого вектора, а в конце каждой итерации все элементы будут собраны узлом M в один вектор $\{\delta_0, \dots, \delta_j\}$ и распределены снова по всем узлам S. Так как к моменту начала метода СГС все векторы $\{\delta_0, \dots, \delta_j\}$

будут вычислены и пересланы всем узлам, дополнительно пересылать их не придется.

Для работы методов СГС и обратного распространения ошибки не требуется собирать общие данные и заново распределять их. Однако это потребуется в методе прямого распространения. Поэтому при работе этого метода матрица весов W будет переслана в полном объеме.

2.4. Архитектура нейронной сети

Для обучения нейронной сети был выбран набор данных рукописных цифр MNIST [16], состоящий из 60000 образцов для обучающей выборки и 10000 образцов для тестовой выборки. Для каждого образца существует метка от 0 до 9.

Так как основной задачей данной работы является применение метода проектирования Q-эффективных программ к методу стохастического градиентного спуска для обучения нейронных сетей, основным критерием оценки будет время выполнения методов, для которых описана Q-эффективная реализация. Поэтому из набора данных будет взято только 6000 образцов.

Загрузка и структуризация данных осуществляется с помощью кода, указанного в листинге 1.

Листинг 1 – Загрузка обучающего набора

```
struct Data
{
  int y;
  vector<double> x;
};

vector <Data> load_data(string path, int count_x)
{
  int examples;

  ifstream file;
  file.open(path);
  file >> examples;
  examples = (int)(examples * 0.1);
  vector <Data> data(examples);
  for (int i = 0; i < examples; ++i){
    data[i].x.resize(count_x);
  }
  for (int i = 0; i < examples; ++i){
    file >> data[i].y;
    for (int j = 0; j < count_x; ++j){
```

```
            file >> data[i].x[j];
        }
    }
    return data;
}
```

Для более удобного кода процесса обучения все образцы будут представлена в виде набора пользовательского типа данных struct [17].

Архитектура нейронной сети будет состоять из 6 слоев:

1) входной слой имеет 784 нейрона (в соответствие с размером изображения);

2) четыре скрытых слоя по 80, 60, 40, 20 нейронов соответственно;

3) выходной слой на 10 нейронов (в соответствие с количеством распознаваемых классов).

Графическое представление архитектуры показано на рисунке на рисунке 2.

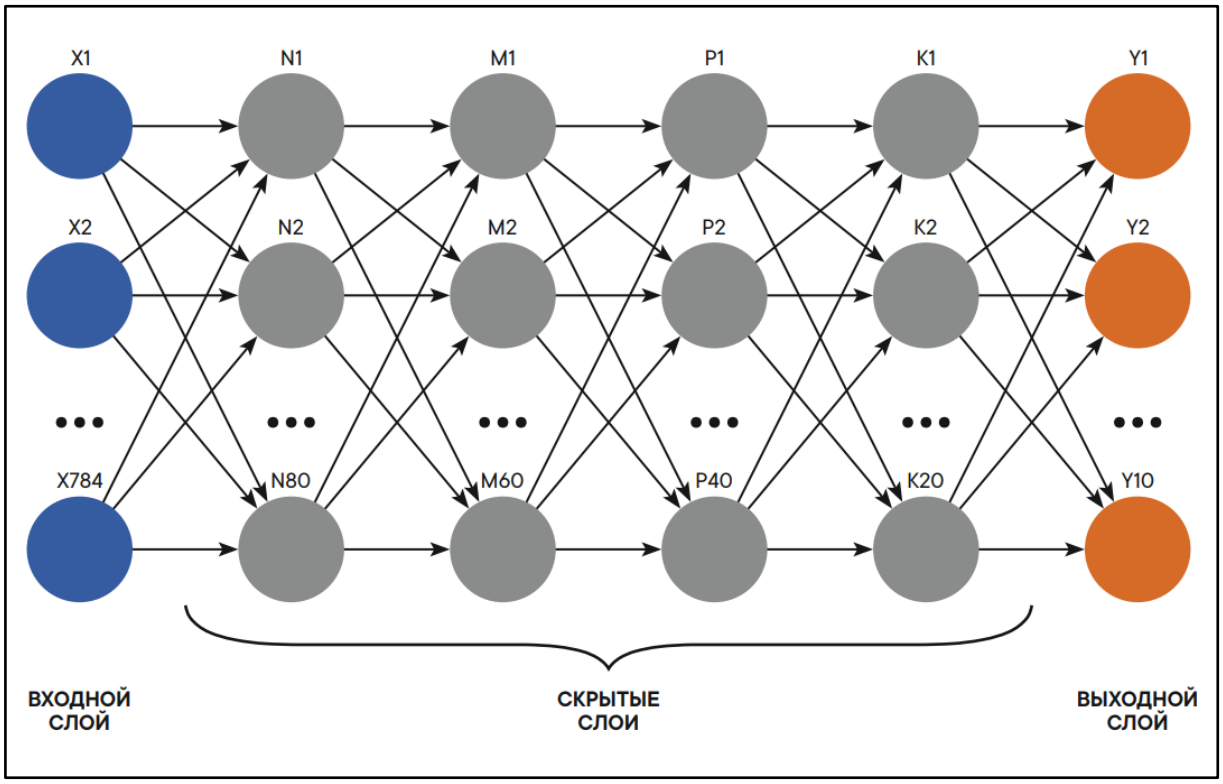

Рисунок 2 – Архитектура нейронной сети

Общее количество параметров такой сети составляет более 70 тысяч.

При инициализации нейронной сети между нейронами разных слоев создаются связи. Также у каждого нейрона есть смещения. Значения связей

и смещений определяются случайно. Это может привести как к ускорению обучения, так и к его замедлению. В случае, если начальные веса и смещения окажутся слишком большими, процесс обучения может прекратиться, так как метод СГС является итерационным и может не сойтись.

Поэтому выбор начальных весов и смещений следует определять рационально. В работе Glorot and Bengio [18] предлагается использовать метод нормированной инициализации, программная реализация которого показана в листинге 2.

Листинг 2 – Функция, реализующая метод нормированной инициализации

```
double create_number(int n, int m){

  double low = -sqrt(6) / sqrt(n + m);
  double high = sqrt(6) / sqrt(n + m);
  double k = low + (double)(rand()) / RAND_MAX * (high - low);

  return k;
}
```

2.5. Q-эффективная программа для систем с общей памятью

Все Q-эффективные программы будут написаны на языке C++ [11].

Исследование ресурса параллелизма метода стохастического градиентного спуска и метода обратного распространения ошибки будет проводиться для систем с общем памятью с помощью технологии OpenMP [12,13], а для систем с распределенной памятью с помощью технологий OpenMP и MPI [14,15]. При этом Q-эффективные программы будут работать в составе нейронной сети, архитектура которой была предоставлена ранее.

После завершения предсказания нейронной сети с помощью метода обратного распространения ошибки вычисляется мера влияния нейронов на величину ошибки. Код, реализующий Q-эффективную реализацию метода обратного распространения ошибки показан на листинге 3.

Листинг 3 – Q-эффективная реализация метода обратного распространения ошибки для системы с общей памятью

```
for (int i = 0; i < size_network[L - 1]; i++){
  if (i != train_data[v].y){
    delta[L - 1][i] = neurons_value[L - 1][i] * derivative_sigmoid(activa-
tions[L - 1][i]);
  }
```

```
  else{
    delta[L - 1][i] = (neurons_value[L - 1][i] - train_data[v].y) * deriva-
tive_sigmoid(activations[L - 1][i]);
    }
}

for (int i = L - 2; i > 0; i--){
#pragma omp parallel num_threads(size) {
#pragma omp for schedule(static, 1)
    for (int j = 0; j < size_network[i]; j++){
      double sum = 0;
      for (int k = 0; k < size_network[i + 1]; k++){
        sum += delta[i + 1][k] * weights[i + 1][k][j];
      }
      delta[i][j] = derivative_sigmoid(activations[i][j]) * sum;
    }
  }
}
```

Данный алгоритм представлен для одного набора входных данных. Но обучение производится только после того, как пройдет целый мини-пакет набора данных.

После того, как нейронная сеть обработает мини-пакет набора данных, применяется метод СГС для вычисления градиентов и обновления весов и смещений на их основе. Q-эффективная реализация этого метода для систем с общей памятью представлена в листинге 4.

Листинг 4 – Q-эффективная реализация метода СГС для системы с общей памятью

```
double learning_rate = 0.35 * exp(-p / epoch);
for (int i = 1; i < L; i++){
#pragma omp parallel num_threads(size) {
#pragma omp for schedule(static, 1)
    for (int j = 0; j < size_network[i]; j++){
      biases[i][j] = biases[i][j] - learning_rate * delta[i][j];
      for (int k = 0; k < size_network[i - 1]; k++){
        weights[i][j][k] = weights[i][j][k] - learning_rate * delta[i][j] *
neurons_value[i - 1][k];
      }
    }
  }
}
```

По завершению обновления весов и смещений начинает обучение уже со следующим мини-пакетом. Один мини-пакет содержит 100 образцов для обучения.

Весь ранее описанный процесс может повторяться несколько раз для достижения нужной точности. Каждая из таких итераций называется эпохой. В данной работе для обучения будет выставлено 5 эпох.

2.6. Q-эффективная программа для систем с распределенной памятью

Для загрузки обучающих наборов будет использоваться тот же, код, что и для системы с общей памяти.

Как было ранее сказано в описании Q-эффективной реализации для системы с распределенной памятью, общие данные будут распределены между узлами, так, чтобы каждому узлу досталась некоторая часть данных, с которыми он будет работать.

Перед началом обучения на каждом узле создается свое хранилище для общих и частных данных.

В методе обратного распространения ошибки для последнего слоя алгоритм действий отличается, поэтому последний слой узел M обрабатывает самостоятельно. Для остальных слоем узел M использует все доступные узлы S (листинг 5).

Листинг 5 – Q-эффективная реализация метода обратного распространения ошибки для системы с распределенной памятью

```
if (rank_process == 0)
{
  for (int i = 0; i < network_process[L - 1]; i++)
  {
    if (i != train_data[v].y)
    {
      delta[L - 1][i] = sigmoid(activations[L - 1][i]) * derivative_sig-
moid(activations[L - 1][i]);
    }
    else{
      delta[L - 1][i] = (sigmoid(activations[L - 1][i]) - train_data[v].y)
* derivative_sigmoid(activations[L - 1][i]);
    }
  }
}
MPI_Bcast(delta[L - 1], network_process[L - 1], MPI_DOUBLE, 0,
MPI_COMM_WORLD);
for (int k = L - 2; k > 0; k--)
{
  double* new_delta = new double[network_process[k] / size_process];
#pragma omp parallel num_threads(size_threads)
  {
#pragma omp for schedule(static, 1)
```

```
    for (int i = 0; i < network_process[k] / size_process; i++)
    {
      double sum = 0;
      for (int j = 0; j < network_process[k + 1]; j++)
      {
        sum += process_weights[k + 1][i * network_process[k + 1] + j] *
delta[k + 1][j];
      }
      new_delta[i] = derivative_sigmoid(activations[k][i + rank_process *
network_process[k] / size_process]) * sum;
    }
  }
  MPI_Allgather(new_delta, network_process[k] / size_process, MPI_DOUBLE,
delta[k], network_process[k] / size_process, MPI_DOUBLE, MPI_COMM_WORLD);
  delete[] new_delta;
}
```

По окончании итерации вычисленные меры влияния нейронов на величину ошибки распределяются аналогично активационным потенциалам в методе прямого распространения.

Все описанные процессы производятся для всего мини-пакета.

В методе СГС отсутствуют пересылки данных, а вычисления новых весов и смещений распределены между узлами так, чтобы каждый узел вычислял только свою часть согласно описанию Q-эффективной реализации (листинг 6).

Листинг 6 – Q-эффективная реализация СГС для системы с распределенной памятью

```
for (int k = 1; k < L; k++)
{
  double learning_rate = 0.35;
#pragma omp parallel num_threads(size_threads)
  {
#pragma omp for schedule(static, 1)
    for (int i = 0; i < network_process[k - 1] / size_process; i++)
    {
      for (int j = 0; j < network_process[k]; j++)
      {
        process_weights[k][i * network_process[k] + j] = pro-
cess_weights[k][i * network_process[k] + j] - learning_rate * deriva-
tive_sigmoid(activations[k][i + rank_process * network_process[k - 1] /
size_process]) * delta[k][j];
      }
    }
#pragma omp for schedule(static, 2)
    for (int j = 0; j < network_process[k] / size_process; j++)
    {
      process_biases[k][j] = process_biases[k][j] - learning_rate *
delta[k][j + rank_process * network_process[k] / size_process];
    }
  }
}
```

Выводы по второй главе

В результате проведенной работы были построен Q-детерминант для метода стохастического градиентного спуска, который состоит из двух множеств безусловных Q-терм. Q-детерминант метода обратного распространения ошибки состоит из одного множества безусловных Q-терм.

Из описания Q-эффективной реализации видно, что оба метода имеют общие данные, которые можно разделить между узлами и уменьшить количество пересылок данных.

Для проверки работы Q-эффективных программ была разработана архитектура полносвязной нейронной сети с общим количеством параметров более 70 тысяч.

Q-эффективная реализация для систем с общей памятью показывает, что с помощью технологии OpenMP можно легко разделить итерации рассматриваемых методом между ядрами одного узла.

Q-эффективная реализация для систем с распределенной памятью показывает, что вычисления в методе СГС можно легко разделить между узлами, не прибегая к пересылке данных внутри итераций, однако в методе обратного распространения ошибки избежать пересылок данных не получится из-за особенностей этого метода.

3. ТЕСТИРОВАНИЕ Q-ЭФФЕКТИВНЫХ ПРОГРАММ

Прежде, чем приступать к вычислительным экспериментам, необходимо протестировать Q-эффективные программы, а также последовательную программу. Тестирование будет проводиться на суперкомпьютере «Торнадо ЮУрГУ».

Выполнение программы на суперкомпьютере «Торнадо ЮУрГУ» начинается с постановки задачи в очередь. Запуск задач может осуществляться как в интерактивном режиме, так и в пакетном. Для пакетного запуска в задаче необходимо указать несколько параметров: количество используемых узлов, количество процессов, запускаемых на одном узле и тип задачи. На листинге 7 показан пример скрипта, написанного на языке bash [19].

Листинг 7 – Код скрипта для запуска задач

```
mpicc -openmp network_3.cpp -o network_3.x
sbatch -n 12 -p quick --ntasks-per-node=1 mpirun ./network_3.x
```

При тестировании разработанные программы ставились в очередь задач в интерактивном режиме, а результаты тестирования можно увидеть на рисунках 3–5.

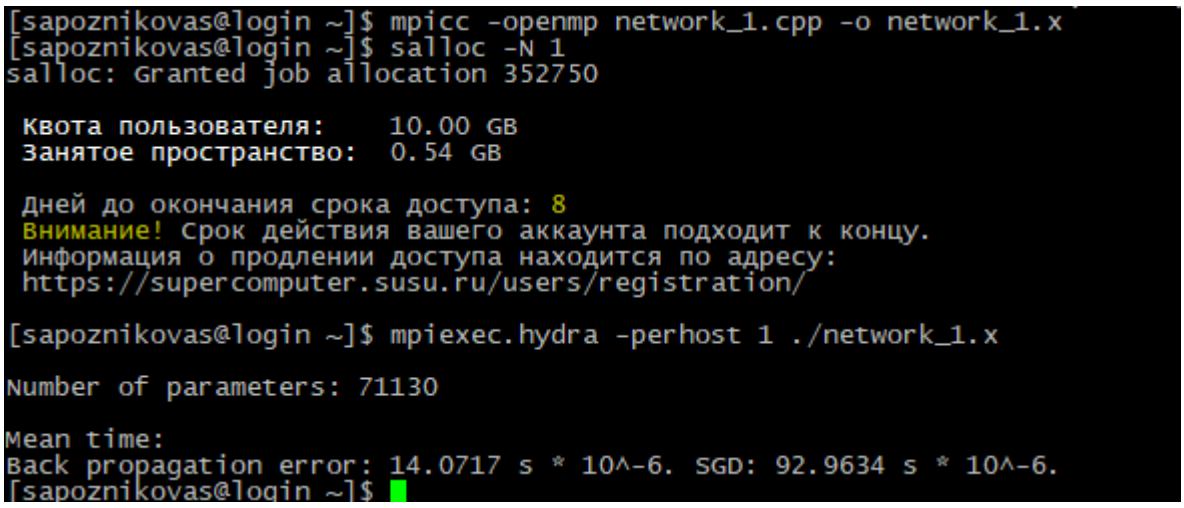

Рисунок 3 – Пример запуска последовательной программы

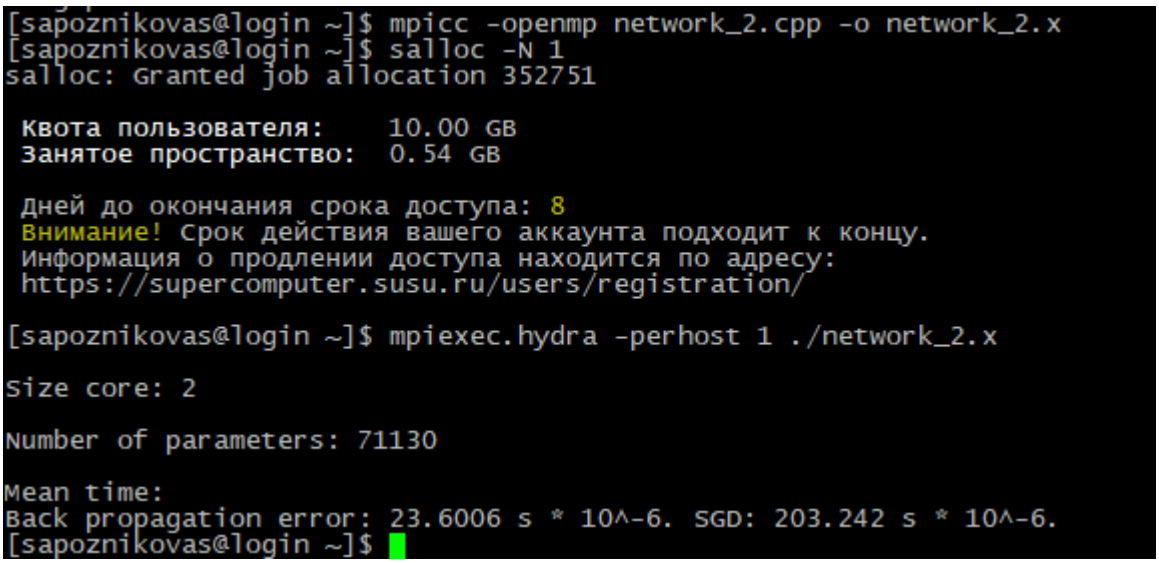

Рисунок 4 – Пример запуска Q-эффективной программы для общей памяти

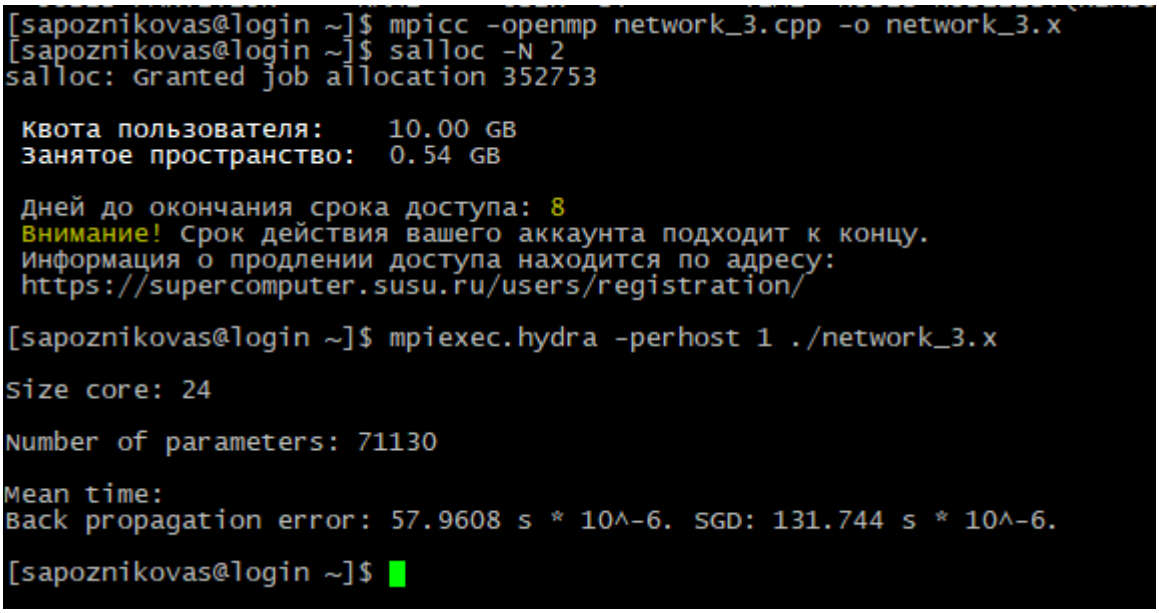

Рисунок 5 – Пример запуска Q-эффективной программы для распределенной памяти

Выводы по третьей главе

В результате проведенной работы был определен порядок запуска задач на суперкомпьютере «Торнадо ЮУрГУ».

Были протестированы последовательная и Q-эффективные программы.

4. ЭКСПЕРИМЕНТАЛЬНОЕ ИССЛЕДОВАНИЕ Q-ЭФФЕКТИВНЫХ ПРОГРАММ

При экспериментальном исследовании рассматривались нейронные сети с разным общим количеством параметром, но с одинаковым количеством слоев.

Все исследования проводились с использованием ресурсов суперкомпьютера «Торнадо ЮУрГУ». Результаты экспериментов последовательной программы занесены в таблицу 2.

Таблица 2 – Результаты экспериментов последовательной программы

Общее количество параметров	**Метод обратного распространения ошибки, мкс**	**Метод стохастического градиентного спуска, мкс**
71130	14,13	92,79
158250	43,02	196,22
261370	89,84	312,78
380490	158,70	454,74
515610	238,95	615,23
666730	365,33	795,27
833850	506,62	993,36
1016970	780,24	1209,60
1216090	1018,48	1452,40
1431210	1259,55	1743,13
1662330	1572,05	2074,02

Эксперименты с Q-эффективной программой для систем с общей памятью будут проводиться с количеством ядер от 2 до максимального количества физических ядер на одном узле, то есть до 12, а также с изменением количества параметров нейронной сети, как в случае с последовательной программы. Результаты вычислительных экспериментов представлены в таблицах 1 и 2 приложения.

Эксперименты с Q-эффективной программой для систем с распределенной памятью будут проводиться с количеством узлов от 2 до 12 и изменении количества параметров нейронной сети. При это на каждом узле будут использоваться все физические ядра. Результаты вычислительных экспериментов представлены в таблицах 3 и 4 приложения.

Для оценки Q-эффективных программ воспользуемся следующими метриками: ускорение и эффективность.

4.1. Расчет ускорения

Под ускорением понимается безразмерная величина, равная отношению времени выполнения последовательной программы $T_{\text{п}}$ к времени выполнения параллельной программы T_p с использованием некоторого количества ядер p.

Ускорение вычисляется по формуле (27):

$$S_p = \frac{T_{\text{п}}}{T_p}. \tag{27}$$

На рисунках 6 и 7 показаны графики зависимости ускорения Q-эффективной программы для метода обратного распространения ошибки от количества используемых ядер и размера архитектуры нейронной сети для систем с общей и распределенной памятью.·

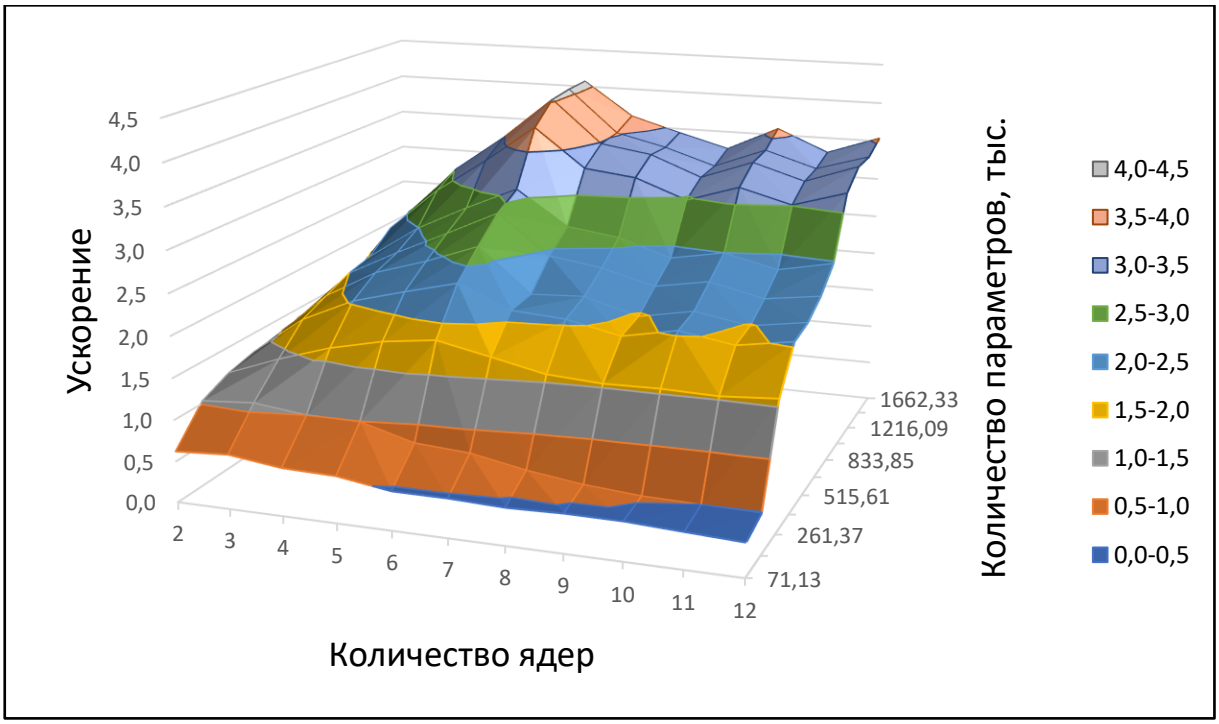

Рисунок 6 – Ускорение Q-эффективной программы для метода обратного распространения ошибки для системы с общей памятью

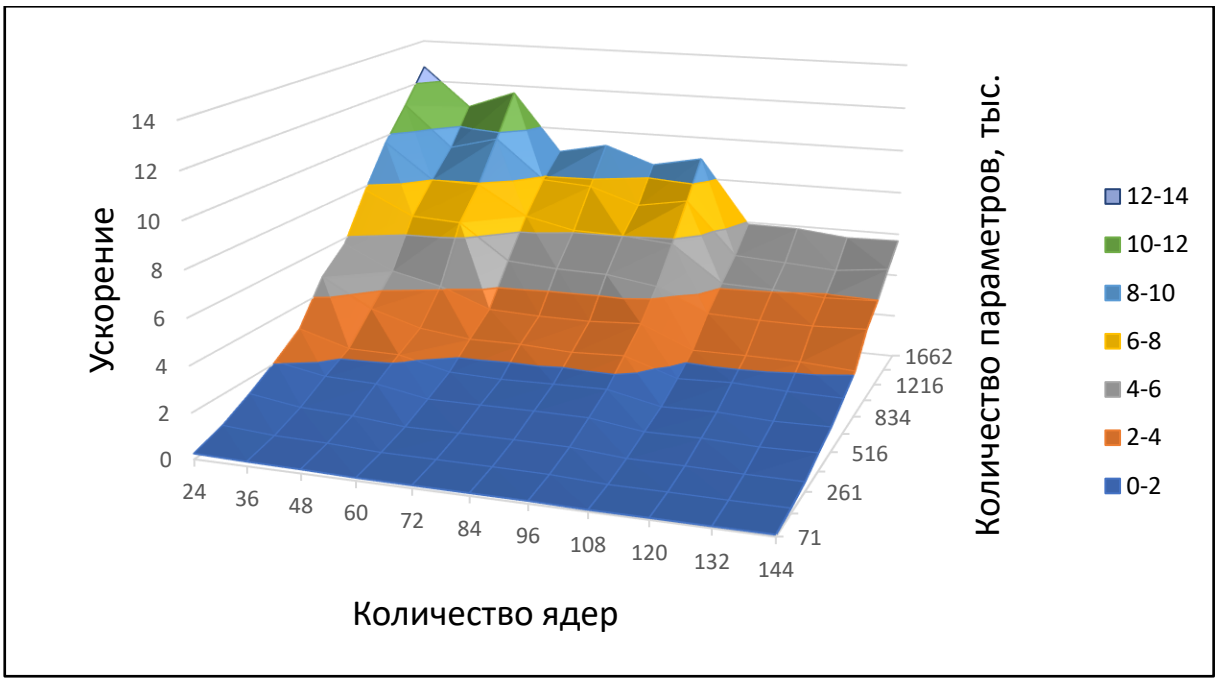

Рисунок 7 – Ускорение Q-эффективной программы для метода обратного распространения ошибки для системы с распределенной памятью

Для Q-эффективной программы для метода СГС аналогичные графики показаны на рисунках 8 и 9 соответственно.

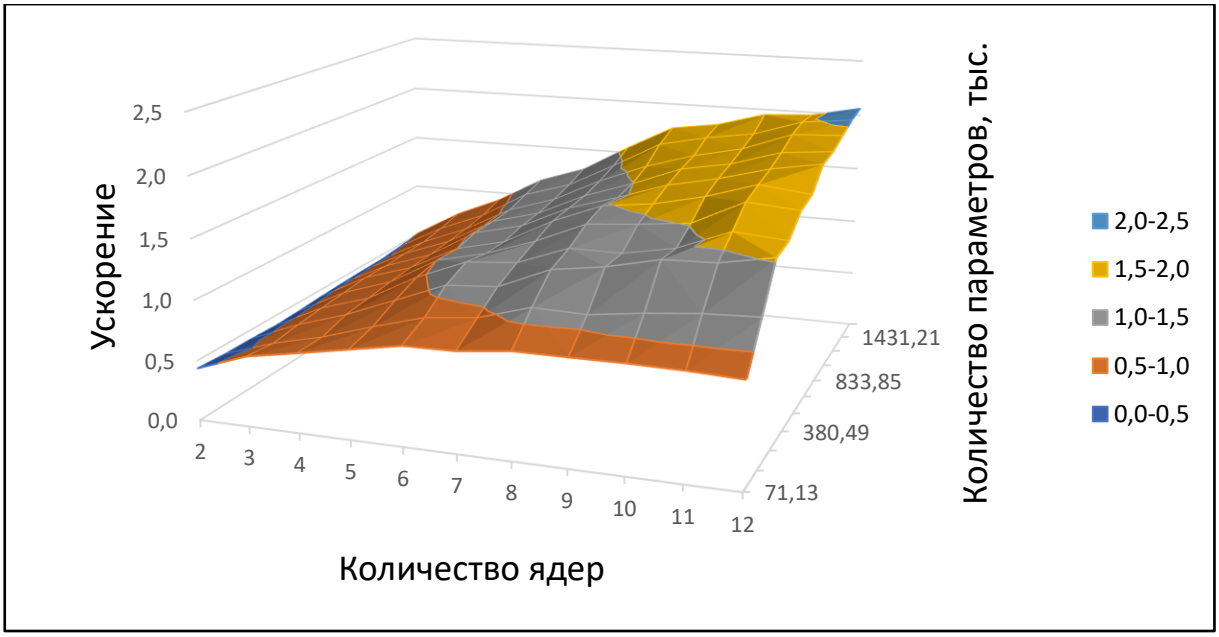

Рисунок 8 – Ускорение Q-эффективной программы для метода СГС для системы с общей памятью

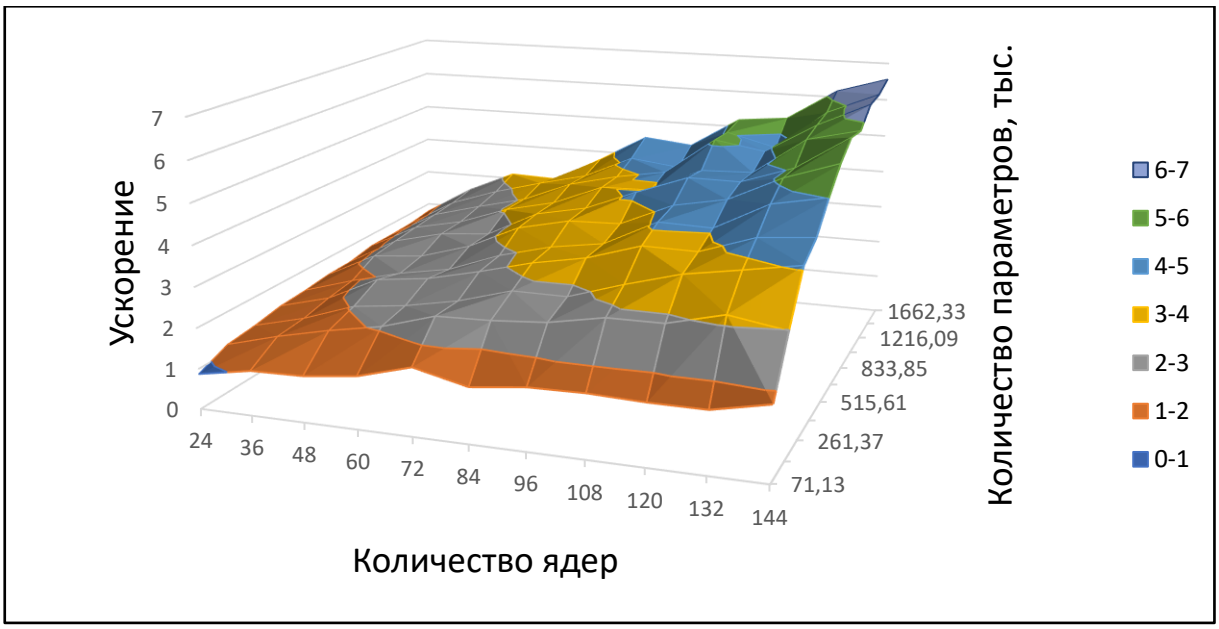

Рисунок 9 – Ускорение Q-эффективной программы для метода СГС для системы с распределенной памятью

Как можно заметить, ускорение Q-эффективной программы для метода обратного распространения ошибки не зависимо от того имеет система общую или распределенную память, со временем выходит на плато, в то время как для Q-эффективной программы для метода СГС совсем иная ситуация: ускорение продолжает расти.

4.2. Расчет эффективности

Под эффективностью понимается безразмерная величина, равная отношению ускорения к количеству используемых ядер p.

Эффективность вычисляется по формуле (28):

$$E_p = \frac{S_p}{p}. \tag{28}$$

На рисунках 10 и 11 показана зависимость эффективности Q-эффективной программы для метода обратного распространения ошибки от количества ядер и размера архитектуры нейронной сети для систем с общей и распределенной памятью.

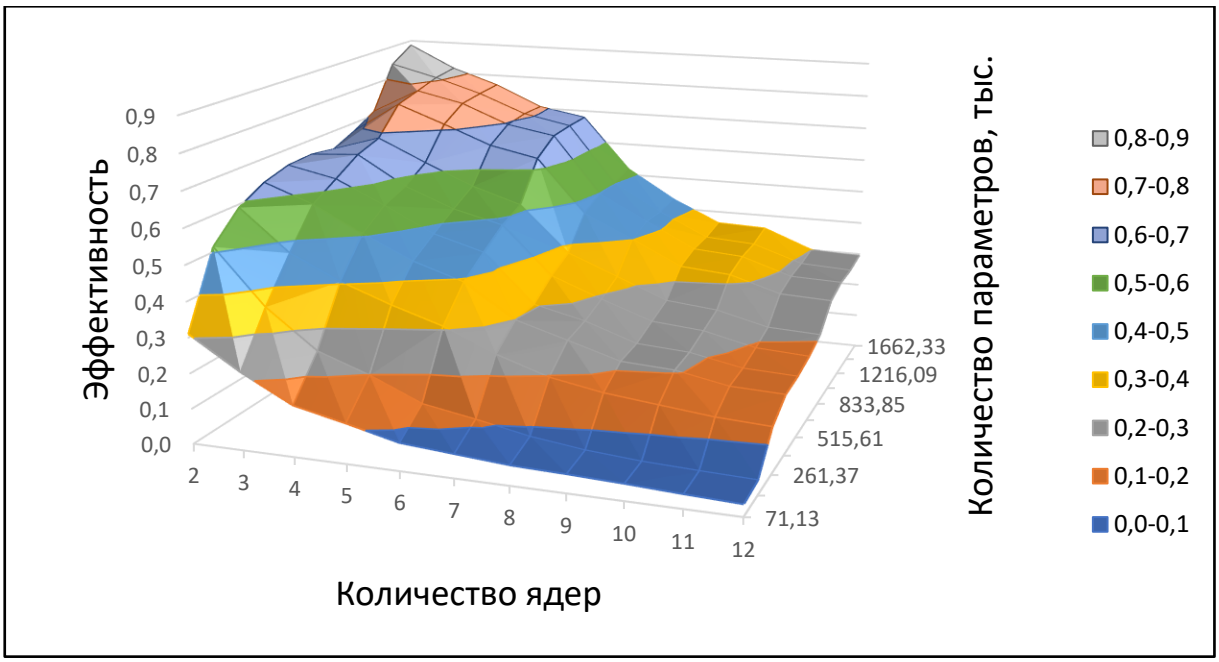

Рисунок 10 – Эффективность Q-эффективной программы для метода обратного распространения ошибки для системы с общей памятью

Из рисунка 10 становится ясно, что даже выбранного максимального количества параметров нейронной сети слишком мало для Q-эффективной реализации метода обратного распространения ошибки.

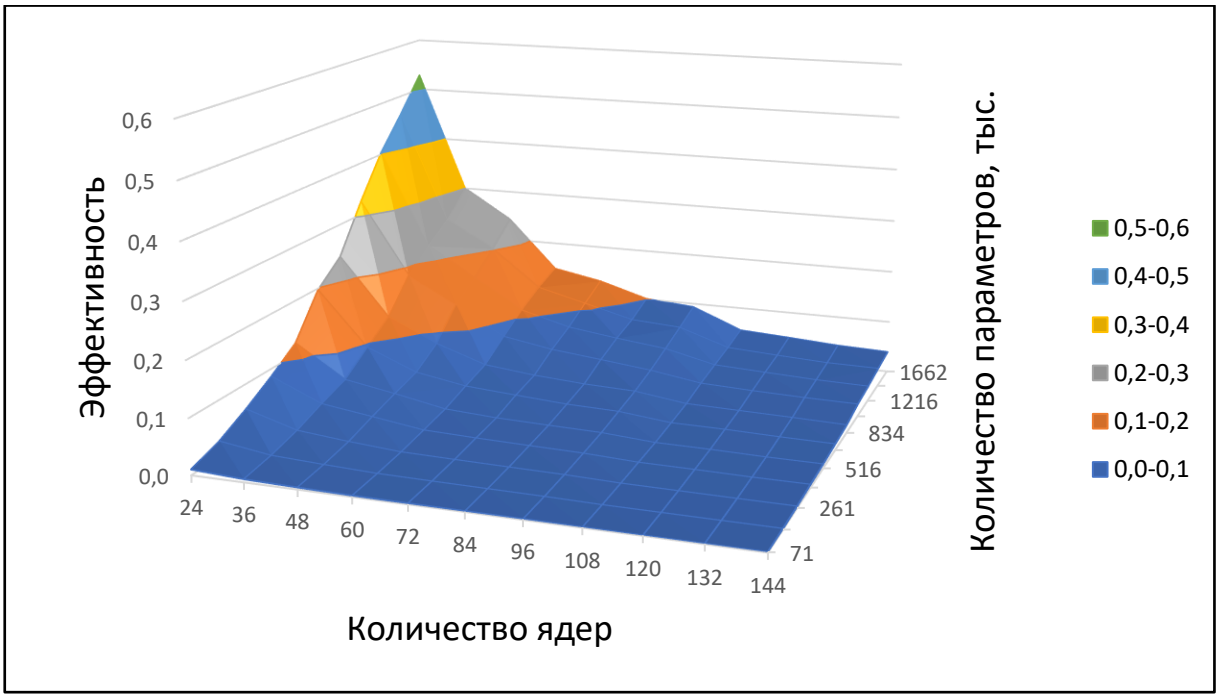

Рисунок 11 – Эффективность Q-эффективной программы для метода обратного распространения ошибки для системы с распределенной памятью

Для Q-эффективной программы для метода СГС аналогичные графики показаны на рисунках 12 и 13 соответственно.

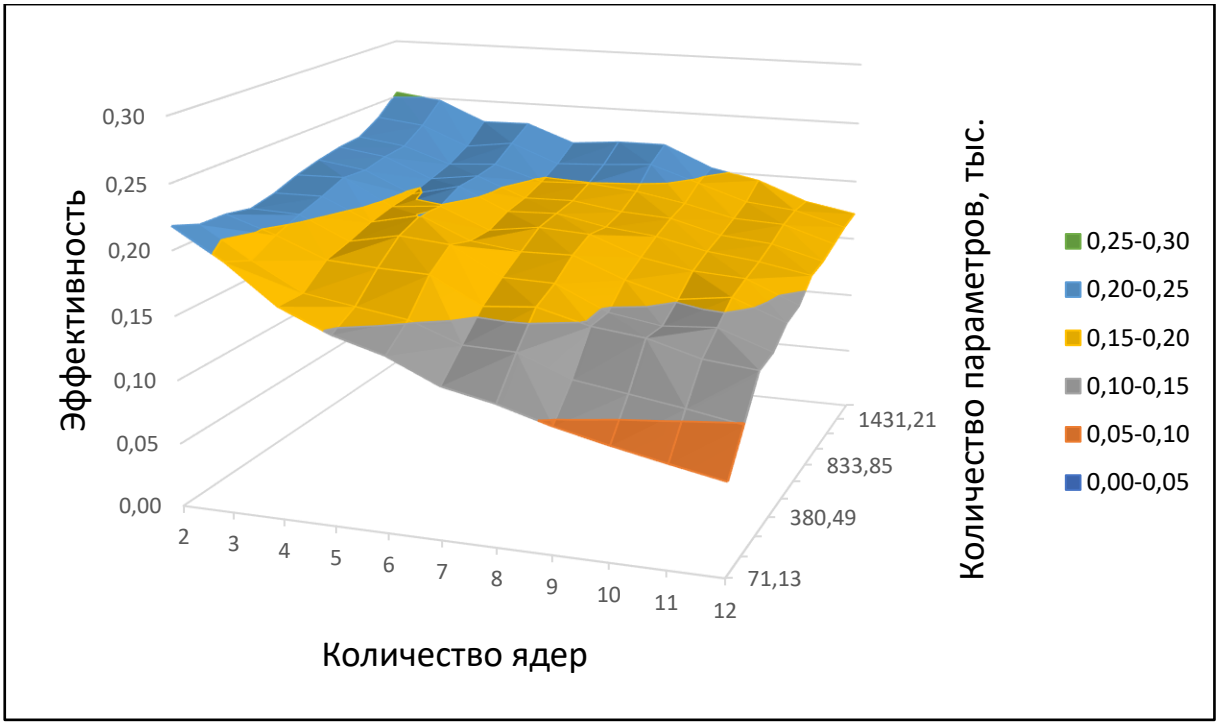

Рисунок 12 – Эффективность Q-эффективной программы для метода СГС для системы с общей памятью

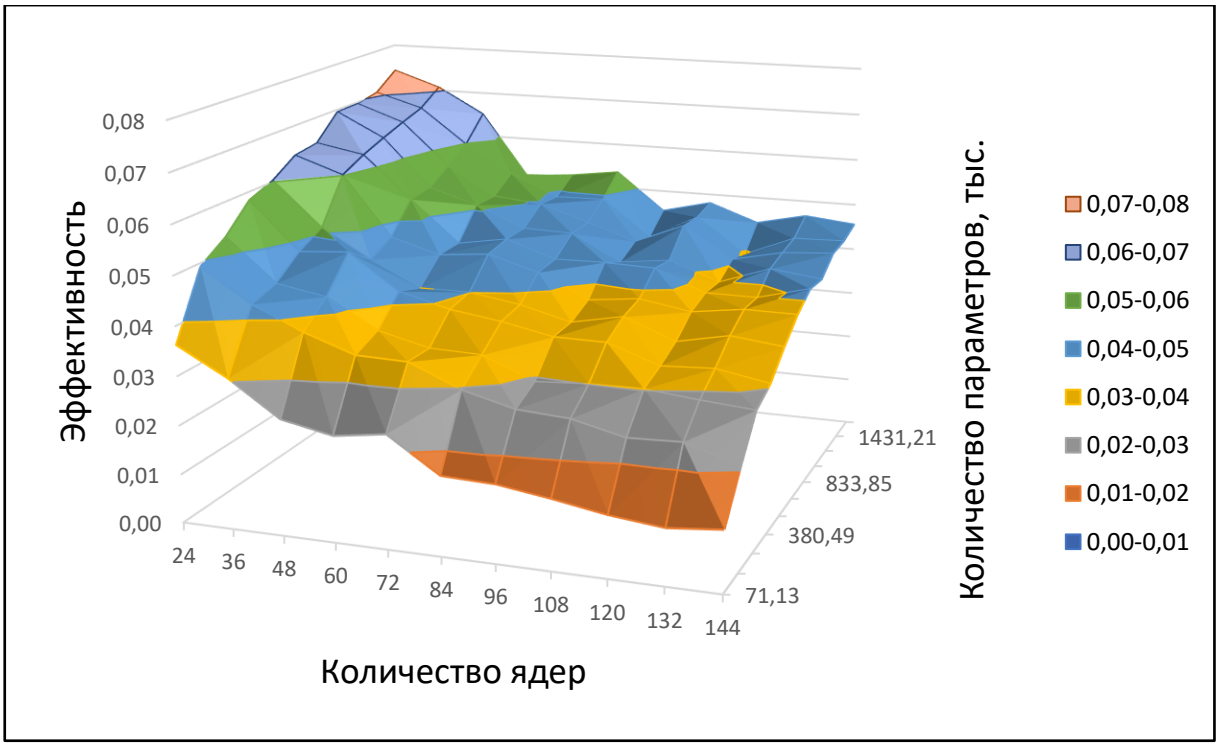

Рисунок 13 – Эффективность Q-эффективной программы для метода СГС для системы с распределенной памятью

Выводы по четвертой главе

В результате проведенной работы было проведено экспериментальное исследование Q-эффективной программы для метода стохастического градиентного спуска и Q-эффективной программы для метода обратного распространения ошибки для систем с общей и распределенной памятью.

Для всех проведенных экспериментов было зафиксировано время выполнения программы, по которому были рассчитаны ускорение и эффективность.

По графикам ускорения метода обратного распространения ошибки (рисунки 6–7) можно увидеть, что этот метод полностью исчерпывает свой ресурс параллелизма при условиях экспериментов. Особенно заметно падение ускорения на системе с распределенной памятью. Это связано с наличием пересылки данных, происходящей перед во время каждой итерации этого метода.

В то же время на рисунках 8–9 видно, что ускорение метода стохастического градиентного спуска продолжает расти на системах с общей и распределенной памятью. Это связано с тем, что в Q-эффективной реализации этого метода отсутствует какая-либо пересылка данных, за счет чего даже на при максимальном количестве параметров нейронной сети виден прирост ускорения.

ЗАКЛЮЧЕНИЕ

В рамках данной работы был применен метод проектирования Q-эффективных программ к методу стохастического градиентного спуска. При этом были решены следующие задачи.

1. Проведен анализ предметной области.
2. Был применен метод проектирования Q-эффективных программ к методу стохастического градиентного спуска и методу обратного распространения ошибки.
3. Проведено тестирование разработанных Q-эффективных программ.
4. Проведено экспериментальное исследование разработанных Q-эффективных программ.

Исходный код Q-эффективных программ доступен по URL-адресу: https://github.com/Snezinka/Parallel-neural-network.

ЛИТЕРАТУРА

1. Алеева В.Н., Зотова П.С., Склезнев Д.С. Расширение возможностей исследования ресурса параллелизма численных алгоритмов с помощью программной Q-системы // Вестник ЮУрГУ. Серия: Вычислительная математика и информатика. 2021. – С. 66–81.

2. Алеева В.Н., Шатов М.Б. Применение концепции Q-детерминанта для эффективной реализации численных алгоритмов на примере метода сопряженных градиентов для решения систем линейных уравнений. // Вестник ЮУрГУ. Серия: Вычислительная математика и информатика. 2021. – С. 56–71.

3. Алеева В.Н. Архитектура вычислительных систем: курс лекций / В.Н. Алеева. – Челябинск: Издательский центр ЮУрГУ, 2023. – 150 с.

4. Aleeva V.N. Improving Parallel Computing Efficiency. // Proceedings – 2020 Global Smart Industry Conference, 2020. – 7 с.

5. Aleeva V.N., Aleev R.Zh. Investigation and Implementation of Parallelism Resources of Numerical Algorithms. // ACM Transactions on Parallel Computing, 2023. – С. 1–64.

6. Каширина И.Л., Демченко М.В. Исследование и сравнительный анализ методов оптимизации, используемых при обучении нейронных сетей. // Вестник ВГУ. Серия: Системный анализ и информационные технологии. 2018 – С. 123-132.

7. Гудфеллоу Я., Бенджио И., Курвилль А. Глубокое обучение / пер. с анг. А. А. Слинкина // М.: ДМК Пресс, 2018. – 652 с.

8. Николенко С., Кадурин А., Архангельская Е. Глубокое обучение. – СПб.: Питер, 2018. – 480 с.

9. Nielsen M.A. Neural Networks and deep Learning [Электронный ресурс] URL: http://neuralnetworksanddeeplearning.com/chap2.html (дата обращения: 13.05.2024 г.).

10. Пользовательская документация суперкомпьютера «Торнадо ЮУрГУ». [Электронный ресурс] URL: http://supercomputer.susu.ru/users/support/ (дата обращения: 09.04.2024 г.).

11. Страуструп Бьярне Язык программирования C++. Краткий курс, 2-ое изд.: Пер. с англ. – СПб.: ООО «Диалектика», 2019. – 320 с.

12. Антонов А.С. Параллельное программирование с использованием технологии OpenMP: Учебное пособие. – М.: Изд-во МГУ, 2009. – 77 с.

13. Официальный сайт проекта OpenMP. [Электронный ресурс] URL: http://www.openmp.org (дата обращения: 01.05.2024 г.).

14. Антонов А.С. Параллельное программирование с использованием технологии MPI: Учебное пособие. – М.: Изд-во МГУ, 2004. – 71 с.

15. Официальный сайт проекта MPI. [Электронный ресурс] URL: https://www.open-mpi.org (дата обращения: 05.05.2024 г.).

16. База данных MNIST образцов рукописного написания цифр [Электронный ресурс] URL. http://yann.lecun.com/exdb/mnist/ (дата обращения: 09.05.2024 г.).

17. Техническая документация пользовательскому типу данных struct языка C++ [Электронный ресурс] URL: https://cplusplus.com/doc/tutorial/structures/ (дата обращения: 13.05.2024 г.).

18. Glorot, X. and Bengio, Y. Understanding the difficulty of training deep feedforward neural networks. // University of Montreal, 2010. – C. 249–256.

19. Техническая документация к языку Bash. [Электронный ресурс] URL: http://www.gnu.org/software/bash/manual/ (дата обращения: 13.05.2024 г.).

20. Презентации лекций по курсу «Глубокие нейронные сети» [Электронный ресурс] URL: https://sok.susu.ru/courses/MachineLearnig/lectures/ (дата обращения: 15.05.2024 г.).

ПРИЛОЖЕНИЕ. Результаты экспериментов

Результаты экспериментов Q-эффективной реализации метода обратного распространения ошибки и метода стохастического градиентного спуска представлены в таблицах 1 и 2 соответственно.

Таблица 1 – Время выполнения Q-эффективной реализации метода обратного распространения ошибки для системы с общей памятью, мкс

Общее количество параметров	**Количество ядер**										
	2	3	4	5	6	7	8	9	10	11	12
71130	22,81	21,42	24,42	25,06	30,24	30,21	31,74	30,43	30,59	32,85	35,35
158250	41,85	39,98	42,95	43,20	52,66	55,52	65,96	76,12	83,58	86,15	88,11
261370	76,04	63,24	55,32	50,79	48,54	52,31	56,88	58,37	57,92	58,52	56,88
380490	125,67	91,35	78,21	72,06	67,12	75,70	77,27	82,88	77,61	81,90	77,61
515610	183,14	128,01	107,58	98,70	89,89	112,41	110,68	120,97	113,79	120,16	114,25
666730	278,41	183,01	157,28	139,37	124,69	149,25	155,43	167,49	159,10	170,26	163,37
833850	393,29	245,42	203,30	180,32	163,54	189,65	196,12	213,06	202,79	216,18	204,56
1016970	578,47	338,95	273,09	236,99	206,99	237,41	242,94	262,56	245,16	259,37	243,38
1216090	715,03	431,80	337,42	292,25	252,92	292,25	300,80	324,99	302,00	320,70	298,82
1431210	747,52	520,80	412,90	356,73	310,18	358,40	374,71	399,02	369,38	392,20	369,37
1662330	888,18	642,41	511,40	446,04	386,08	432,75	456,37	480,72	439,08	475,61	444,56

Таблица 2 – Время выполнения Q-эффективной реализации метода стохастического градиентного спуска для системы с общей памятью, мкс

Общее количество параметров	**Количество ядер**										
	2	3	4	5	6	7	8	9	10	11	12
71130	212,49	158,89	140,76	125,86	113,78	112,90	106,57	106,44	106,08	106,68	108,82
158250	466,21	351,77	298,25	248,40	215,40	205,17	181,17	183,09	174,00	167,65	165,00
261370	746,36	531,14	433,87	352,35	296,12	287,76	246,52	234,93	218,79	216,60	205,19
380490	1107,64	744,25	613,08	470,39	399,62	387,88	341,54	342,16	322,63	287,47	293,77
515610	1468,52	977,43	801,54	613,64	559,39	493,65	430,26	419,84	407,21	357,37	364,74
666730	1838,24	1220,67	1015,19	797,84	724,94	602,95	534,25	516,80	486,26	449,06	461,86
833850	2245,39	1537,58	1254,38	957,59	873,58	743,07	678,35	617,90	585,40	548,26	526,55
1016970	2690,72	1839,41	1490,61	1161,96	1047,87	894,70	808,63	728,45	696,24	647,08	634,40
1216090	3207,34	2190,91	1747,49	1380,06	1221,82	1039,26	927,14	856,08	811,84	746,06	738,56
1431210	3678,97	2501,79	2016,60	1574,31	1409,85	1196,80	1032,46	979,57	945,03	879,50	859,99
1662330	4094,80	2787,30	2244,07	1789,95	1592,48	1343,39	1174,26	1134,68	1064,00	1051,38	1006,19

Результаты экспериментов Q-эффективной реализации метода обратного распространения ошибки и метода стохастического градиентного спуска представлены в таблицах 3 и 4 соответственно.

Таблица 3 – Время выполнения Q-эффективной реализации метода обратного распространения ошибки для системы с распределенной памятью, мкс

Общее количество параметров	**Количество ядер**										
	24	36	48	60	72	84	96	108	120	132	144
71130	57,63	72,39	82,28	114,17	117,69	124,30	124,80	190,43	181,80	193,15	181,15
158250	64,30	85,60	93,49	120,66	122,14	123,43	130,07	193,81	189,74	203,30	200,08
261370	68,06	91,95	102,52	133,45	131,02	131,51	135,59	188,82	195,18	212,83	215,66
380490	76,89	96,66	101,43	138,73	137,95	137,00	139,33	194,69	191,97	213,93	200,97
515610	82,22	107,62	109,30	145,50	152,20	147,80	145,11	208,98	201,82	225,60	217,32
666730	77,22	106,23	132,75	147,63	147,23	154,81	148,86	218,99	208,56	224,93	230,26
833850	88,59	109,39	123,16	156,53	161,20	161,80	155,18	223,56	222,98	226,33	244,00
1016970	99,94	115,96	118,16	157,21	163,04	164,88	177,95	234,43	241,09	242,90	236,50
1216090	105,21	127,20	128,28	154,63	167,50	169,21	175,05	245,92	250,66	261,36	255,95
1431210	113,24	137,48	129,32	159,83	162,99	180,70	171,14	256,70	264,39	273,70	258,91
1662330	123,87	145,39	135,07	177,85	168,99	184,94	175,53	266,45	267,66	279,61	277,28

Таблица 4 – Время выполнения Q-эффективной реализации метода стохастического градиентного спуска для системы с распределенной памятью, мкс

Общее количество параметров	**Количество ядер**										
	24	36	48	60	72	84	96	108	120	132	144
71130	106,65	84,16	81,92	71,89	55,91	68,59	61,36	60,35	62,15	62,09	51,99
158250	165,86	127,93	107,43	94,88	79,67	78,45	75,50	68,82	68,53	61,29	65,74
261370	246,35	196,35	153,47	142,60	127,79	110,47	110,71	96,26	86,29	81,27	76,64
380490	324,97	261,21	192,92	180,91	161,30	143,77	133,33	118,80	119,40	106,61	103,73
515610	425,12	334,85	284,06	256,54	213,22	190,68	179,38	150,38	151,74	129,44	122,72
666730	520,48	367,11	348,33	297,21	242,14	222,23	219,69	189,23	178,32	155,90	143,19
833850	642,42	441,29	386,90	370,46	296,97	275,20	239,78	214,68	215,03	195,74	167,18
1016970	725,21	522,07	466,69	396,28	361,05	311,44	309,14	266,35	263,71	218,09	206,34
1216090	868,99	610,82	532,78	477,19	405,62	372,79	352,15	284,21	297,21	255,49	230,19
1431210	1029,33	716,72	606,23	540,16	492,11	420,15	437,04	343,78	368,11	301,06	274,16
1662330	1165,70	814,55	663,77	674,09	551,88	460,50	475,13	399,68	390,08	336,71	316,11