ПРАВИТЕЛЬСТВО РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение высшего профессионального образования

Национальный исследовательский университет

«Высшая школа экономики»

Московский институт электроники и математики

Факультет Прикладной математики и кибернетики

Кафедра Кибернетики

Дипломная работа

по специальности 230401.65 «Прикладная математика»

Методы параллельного программирования нейронных сетей с использованием технологии CUDA

Студент группы М96 Климов П.А.

Руководитель доцент, к.т.н. Боголюбов Д.П.

Зав. кафедрой профессор, д.т.н. Афанасьев В.Н.

Москва 2014

Оглавление.

Введение.

Человечество с каждым днем генерирует все больше информации. Информационные потоки задействованы во всех областях нашей жизни и их роль неуклонно растет. Постоянное увеличение объемов информации, доступной для человека, приводит к проблеме ее восприятия мозгом. Сейчас необходимо уметь работать с информацией, правильно ее обрабатывать, анализировать, вычленять необходимые данные и в последующем знания из каждодневно обрушивающегося на мозг человека потока информации. Развитие ЭВМ позволяет работать с большими объемами информации быстрее, и, при учете всех необходимых факторов, - наиболее качественно. Поэтому с самого начала развития ЭВМ их использовали для помощи человеку в анализе информации и вычислениях.

В наше время наиболее актуальна тема принятия решений, для поддержки которых необходимо работать с большим количеством информации, уметь правильно классифицировать ее по необходимым критериям, для последующего анализа и вывода результата. В этом человеку помогают системы поддержки принятия решений, которые позволяют быстро и качественно, минимизируя риски, детально анализировать предметную область проблемы. Такие системы позволяют предоставлять оперативный, объективный и наиболее полный отчет об исследуемой области, предоставлять прогнозы, показатели и варианты наиболее подходящих под специфику предметной области проблемы в различных ее проявлениях. Для получения наиболее точных и подходящих для решения проблемы данных необходимо грамотно задать алгоритм отбора и классификации информации системе.

Но все же, во многих случаях невозможно найти точное решение проблемы или необходимые значения параметров системы, поэтому одним из главных направлений в системах поддержки принятия решений является прогнозирование, которое в последствии помогает человеку принять

подходящее под ситуацию оптимальное решение.

В современном мире многие задачи не поддаются простому алгоритмическому решению, а проведение точного расчёта может занять слишком много времени. В таких случаях может быть достаточно дать оценки возможным значениям, вывести тенденции изменений. Необходимо правильно определить ключевые показатели и дать им приблизительные оценки, которые будут основываться на анализе системы. Причем, необходимо учитывать состояние системы не только в настоящее время, но и проанализировать ее состояния в прошлом. Так как действующие системы находятся в постоянном развитии, существуют определённые тенденции их изменений. Цикличность событий и состояний системы зависит от череды предыдущих состояний и предпосылок к ним. Таким образом, при анализе исследуемой системы следует учитывать не только статистические данные ее показателей, но и временные интервалы и взаимосвязи событий — то есть собирать информацию о состоянии системы в различные периоды времени во временные ряды.

Существует довольно много систем поддержки в различным областях деятельности человека, которые могут учитывать статистические данные и способны прогнозировать изменения временных рядов. Но, чем более сложна анализируемая система и взаимосвязанность ее параметров, тем труднее предоставить наиболее значимую информацию для поддержки принятия решений. Так же существует проблема неопределенности данных.

Таким образом наиболее качественный результат будет достигнут при соотношении статистических данных о ключевых показателях в виде временных интервалов, что позволит наладить систему работы с неопределенными данными.

В качестве основы разработок современных систем поддержки принятия решений лежат нейронные сети, работа которых несколько подобна сложному процессу работы мозга человека. Способность обучения нейронных сетей позволяет им быстро адаптироваться к изменениям временных интервалов исследуемых систем. Нейронные сети используются в

различных областях и их использование позволит добиться конкурентного преимущества для разрабатываемой системы поддержки. Повышая точность и оперативность предоставления знаний из данных и прогнозов, повышается качество результатов системы. Таким образом нейронные сети востребованы в современном мире для решения проблем в разных областях деятельности человека.

С увеличением числа данных, однако, увеличивается время, необходимое на их обработку, тем более при применении нейронных сетей, которые будут рассматривать огромное число взаимосвязей ключевых показателей. Сокращение времени анализа данных и вычленения из них полезных знаний также одно из приоритетных направлений изучения алгоритмизации систем поддержки принятия решений при помощи нейронных сетей.

Искусственные нейронные сети по парадигме обучения разделены на два фундаментальных класса:

1. Сети, имеющие алгоритм обучения с учителем;
2. Сети, которые имеют алгоритм обучения без учителя.

Значительное ускорение можно получить используя для программирования нейронных сетей параллельные алгоритмы. Так, некоторые задачи можно решать намного быстрее при помощи параллельных алгоритмов, используя для обработки данных вычислительные мощности нескольких процессоров. Таким образом, решение проблемы увеличения временных затрат на обработку данных с помощью искусственных нейронных сетей будет реализовано в данной дипломной работе посредством использования параллельных алгоритмов. Доступным методом реализации параллельного алгоритма может служить технология CUDA, позволяющая реализовать на практике такой алгоритм, задействуя графический процессор в процессе обучения и работы системы. Технология CUDA является продуктом компании NVIDIA, которая поставляет на потребительский рынок графические процессоры разных ценовых категорий, что делает данную технологию пригодной для использования широкими массами людей.

Так в данной работе будут рассмотрены уровни параллелизации нейронных сетей и технология CUDA.

Таким образом, объект данной дипломной работы — нейронная сеть, а предмет — алгоритм написания нейронной сети.

Цель данной дипломной работы: анализ параллельных алгоритмов нейронных сетей для выявления наиболее эффективных способов реализации на практике наиболее быстрых и качественных приложений для обработки большого объёма информации. Это упростит процесс принятия решений для специалистов многих областей, позволит автоматизировать классификацию объектов, проводить постоянный мониторинг исследуемых объектов, прогнозировать фазы развития объекта.

Таким образом при написании данной дипломной работы были определены следующие задачи:

1. Выявить достоинства и недостатки нейронных сетей и особенности их программирования.
2. Рассмотреть примеры моделей нейронных сетей разных областей применения.
3. Выявить возможные способы распараллеливания нейронных сетей.
4. Изложить методы параллельного программирования с использованием технологии CUDA.
5. Показать на практике реализацию параллельных алгоритмов нейронных сетей.

В наше время нейронные сети активно исследуются и находят свое применение. Существует множество учебников по теории нейронных сетей, где подробно изложена математическая база данной модели. Но работ по параллелизации нейронных сетей всё ещё не достаточно много. С точки зрения программиста вопрос параллельного программирования освещён подробно на разных профильных интернет ресурсах, а также существует множество книг и справочников по технологиям, требующимся для

параллельного программирования. Например, технология CUDA подробно описана на официальном сайте компании NVIDIA.

Новизна данной дипломной работы состоит в подробном анализе реализаций параллельных алгоритмов нейронных сетей по средствам технологии CUDA, на примере двух различных моделей искусственных нейронных сетей.

Данная дипломная работа содержит введение, три главы, заключение и список литературы.

Во введении сделан краткий обзор проведённой работы, рассмотрены предмет и объект данной работы, поставлены цели и задачи. Так же приведена общая структура дипломной работы.

Первая глава диплома посвящена нейронным сетям. Рассмотрены сферы их применения, их плюсы и минусы. Так же подробно рассмотренных два примера нейронных сетей, а именно интервальные нейронные сети и самоорганизующиеся карты Кохонена.

Вторая глава содержит информацию о параллелизации нейронных сетей, рассмотрена параллелизация двух выбранных моделей нейронных сетей — модели интервальных нейронных сетей и модели самоорганизующихся карт Кохонена. Так же рассмотрены основные методы параллелизации и разобрана технология CUDA и ее вклад в параллельные методы.

Третья глава полностью посвящена практике применения технологии CUDA для написания параллельных алгоритмов нейронных сетей. Рассмотрена ее реализация на примерах моделей нейронных сетей и результаты их работы.

В заключении подведены итоги проделанной работы и даны выводы и рекомендации.

Глава 1. Искусственные нейронные сети.

1.1 Теория искусственных нейронных сетей.

Первое упоминание о нейронных сетях появилось в работе «Logical calculas of the ideas immanent in nervous activity» («Логическое исчисление идей, относящихся к нервной активности»), опубликованной МакКаллоком и Питсом в 1940-х годах. В ней были описаны модели искусственных нейронных сетей (далее ИНС), основывавшихся на структуре формального нейрона. Уже в 1965 году появилась первая завершенная модель искусственной нейронной сети - персептрон Розенблатта, которая основывалась на некоторых свойствах структуры работы головного мозга человека. После некоторого перерыва в совершенствовании нейронных сетей последовало их возрождение и последующее активное исследование их возможностей. Это стало возможным благодаря широкому распространению персональных компьютеров после 1985 года.

В наше время существуют и используются более 20 моделей нейронных сетей. Они нашли свое применение в различных сферах и задачах, в которых требуется обработка и анализ большого количества информации и незаменимы в решении трудноформализуемых или неформализуемых задач, где требуется не один алгоритм для решения проблемы, либо задач для алгоритмов с неявными параметрами или функциями в форме задаваемых входных сигналов. Дело в том, что для таких задач зачастую трудно подобрать описание, используя обычный набор математических методов. Необходимы новые методы, что в некотором роде послужило причиной развития модели искусственных нейронных сетей. С помощью ИНС можно описать задачи большой размерности, кластеризации, задачи по распознаванию тех или иных образов, поиска активных признаков и многие другие. Так, например, для сферы экономики используют подобные задачи для прогнозирования цен, анализа ситуации на рынке. В сфере

финансов ИНС используют для оценки рисков, определении курса валют. Таким образом, известно много примеров использования искусственных нейронных сетей в прогнозировании временных рядов.

Несмотря на высокую производительность ИНС, эффективность их распознавания образов все же далека от эффективности человеческого мозга, который способен распознавать интонации человека, реагировать на сигналы, знакомые с детства. Но отличительным преимуществом нейронных сетей перед мозгом человека является способность обрабатывать огромные массивы однотипных входных данных, что возможно благодаря структуре ИНС, которая состоит из взаимосвязанной системы обрабатывающих элементов. Обработка информации благодаря такой структуре нейронной сети реализуется путем синхронизации параллельного выполнения различных операций, состоящих из более простых операций, таких как сложение, умножение, а также нелинейное безынерционное преобразование. Также, нейронные сети отличаются отсутствием сложных и объемных иррациональных операций над операндами, которые соответствуют алгоритмам, создаваемых однопроцессорными машинами.

Таким образом структура нейронных сетей является идеально для расчёта их на графических процессорах, которые способны параллельно выполнять множество операций с одинарной точностью.

В работе МакКаллока и Питса искусственные нейронные сети базировались на представлении искусственного нейрона как простейшего логического устройства. В 1949 году Хебб[1] описал первое физиологическое правило обучения искусственного нейрона:

«Если аксон клетки А находится достаточно близко, чтобы возбуждать клетку В, и неоднократно или постоянно принимает участие в ее возбуждении, то наблюдается некоторый процесс роста или метаболических изменений в одной или обеих клетках, ведущий к увеличению

1 Канадский физиолог. Изучал роль нейронов в процессе обучения.

эффективности A, как одной из клеток, возбуждающих B».[2]

Опубликованная Френком Розенблаттом в 1958 году модель восприятия информации мозгом — Персептрон, а также представленная в 1960 году модель Видроу и Хоффа ADALINE являлись системами, которые включали в себя как вычислительные элементы, так и элементы их взаимосвязи, также встроенные схемы обучения. Развитие этих моделей внушало исследователям мысль о возможности выполнять через них вычисления практически любой сложности, но через несколько лет Минским и Пэйпертом была выявлена ограниченность возможностей персептрона в вычислениях. Так же они предполагали, что и возможности представления персептрона как многослойной сети тоже будут ограниченна подобным родом. Это положило начало ухода энтузиазма ученых в области исследований персептона и нейросетей.

Возобновление исследований все же состоялось. Так, исследованиями занимались Джеймс Андерсон, Тойво Кохонен, Стивен Гроссберг. Своеобразное возвращение нейронных сетей в поле зрения ученых произошло после представления модели (сети) Хопфилда и уже в 1986 году Хинтон и Руммельхарт представили алгоритм с обратным распространением ошибок. Группа занималась исследованием параллельных распределительных вычислений. Появление этого алгоритма повлекло за собой массовое исследование возможностей нейронных сетей.

Несмотря на все многообразие проявлений искусственных нейронных сетей, различные модели и концепции, всем моделям свойственны некоторые общие характеристики:

1. способность нейронных сетей к обучению, то есть наличие в них

2 Лахман Константин «Правило Хебба: «универсальный нейрофизиологический постулат» и великое заблуждение математиков» // http://habrahabr.ru/post/102305/

алгоритма, в котором описаны параметры нейросети, которые должны адаптироваться в процессе обучения.

2. обрабатывающие элементы (они отвечают за выбор нейронов, способных участвовать в вычислениях).

3. Все элементы нейросетей взаимосвязаны.

Если рассматривать нейронные сети структурно, то для каждой сети искусственный нейрон различен, однако и нейроны обладают общими принципами работы. Если рассматривать входные сигналы нейронных сетей, они могут быть получены из внешних источников, либо переданы от других нейронов системы. Нейрон обладает несколькими входными сигналами, их количество ограниченно. Собственно, сам нейрон, получая множество входных сигналов, которые измеряются обладают некоторыми весами, производя вычисления передает на выход один выходной сигнал. Такой сигнал в зависимости от модели работы искусственной нейронной сети может быть представлен в различных значениях, либо в действительных, либо бинарном варианте. Сигнал, выходящий из нейрона может быть далее использован следующим нейроном, в качестве входного сигнала, а может быть и выходным сигналом для всей нейросети.

На изображении 1.1.1 представлена схема работы искусственного нейрона.

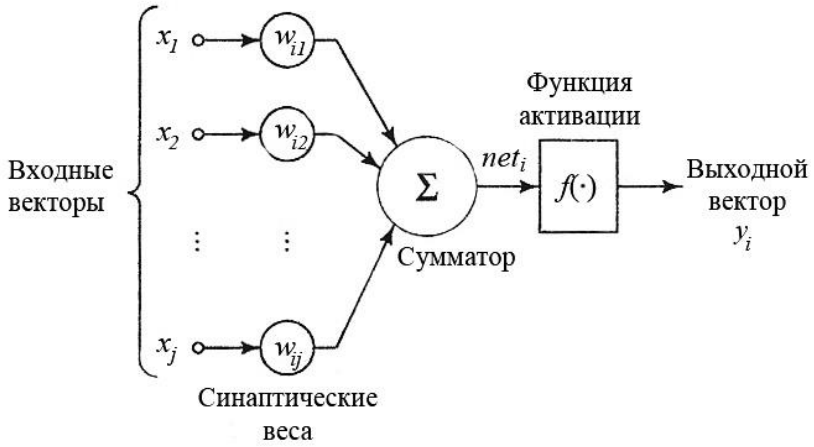


Рисунок 1.1.1 Схема искусственного нейрона.

Так, первоначально нейрон исчисляет входящий вектор по базисной функции:

net(w, x), где $w = \{w_1, w_2, \ldots w_d\}$ - соответствующие веса и $x=\{x_1, x_2, \ldots, x_d\}$.

Существует довольно много видов базисных функций, однако самыми распространенными являются следующие:

1. Линейная комбинация для ходов в нейрон — линейная базисная функция.

$$net_i = \sum_{j=1}^{d} x_j w_{ji}$$

2. Функция *net,* как значение является расстоянием до определенного шаблона, который предоставлен весами – это радиальная базисная функция.

$$net_i = \sqrt{\sum_{j=1}^{d} (x_j - w_{ji})^2}$$

Для выбора базисной функции нет формальных ограничений.

Для вывода результата преобразований данных нейроном, значение результата функции *net*(w,x) преобразуется путем применения активационной функции. Такие функции бывают:

1. Линейно возрастающие;

$$f(net) = \begin{cases} y_{min} \text{ если } net < y_{max} \\ net \text{ если } y_{min} \leq net \leq y_{max} \\ y_{max} \text{ иначе} \end{cases}$$

2. Пороговые функции (единого скачка);

$$f(net) = \begin{cases} 0 \text{ если } net < 0 \\ 1 \text{ иначе} \end{cases}$$

3. Сигмоидальные функции;

$$f(net) = \frac{1}{1 + e^{-net}}$$

4. Гауссова функция;

$$f(net) = ce^{\frac{-net}{\sigma^2}}$$

На схеме представлены все эти типы функций активации результирующего значения. На данной схеме функции положительны, однако они могут иметь и отрицательные значения.

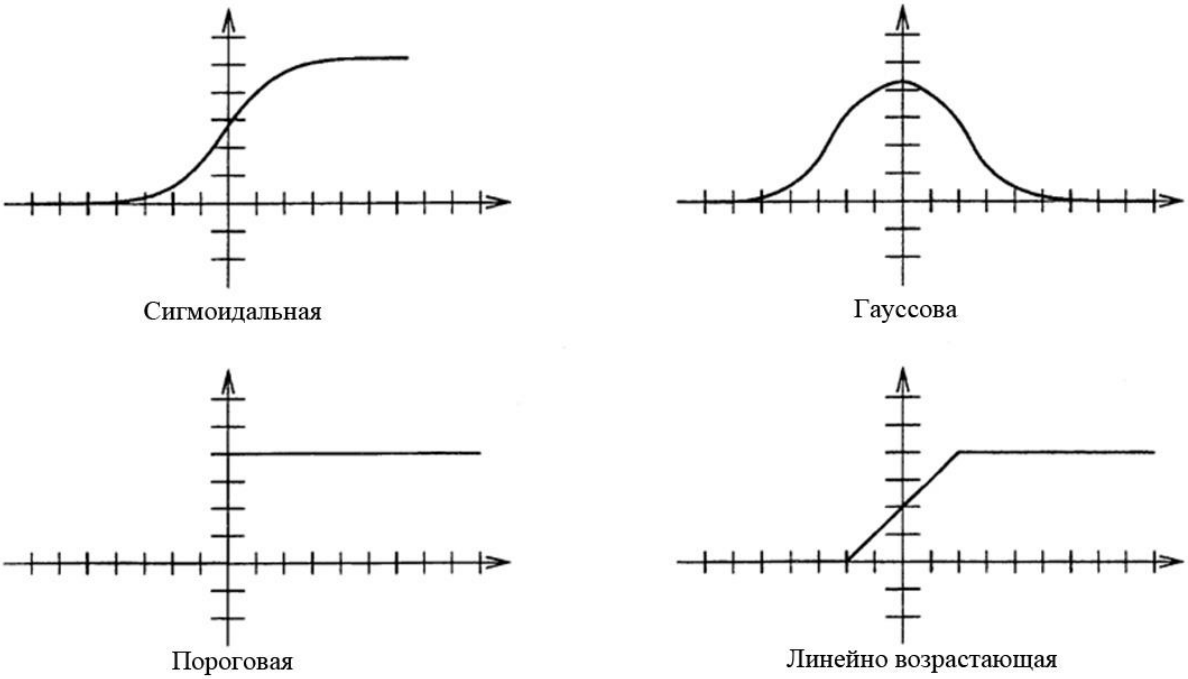


Рисунок 1.1.2. Примеры активационных функций.

Необходимо отметить, что не все модели искусственных нейронных сетей используют для преобразований активационные функции. Так, например, их не используют в случае самоорганизующихся карт. Однако, все же, и они имеют функцию активации, в форме линейной функции f(*net*)=*net.*

Способы взаимосвязи нейронов в искусственной нейронной сети также могут представлять собой различные варианты, в зависимости от выбранной модели. Необходимо учитывать, что нейросеть помимо входных и выходных слоев может иметь и всевозможные скрытые слои.

Одни из наиболее часто используемых типов взаимосвязей нейронов в ИНС:

1. Прямая взаимосвязь или прямое распространение.

Последовательная передача сигналов (выходных сигналов) с одного слоя на вход следующего слоя.

2. Обратная связь.

Выходные сигналы с одного слоя передаются назад, на вход предыдущего слоя.

3. Боковое соединение.

Взаимосвязь выходов слоя нейрона в пределах этого слоя.

Для использования искусственной нейронной сети в решении определенных задач, основными фазами на которые стоит обратить первоочередное внимание являются обучение и непосредственная работа с данными. Обучение как фаза состоит из настройки параметров сети (например, веса и порог активации в активационной функции), действуя при этом, основываясь на обучающем алгоритме.

Выделяют три основных класса обучающих алгоритмов:

1. обучение с учителем;
2. обучение без учителя;
3. обучающий алгоритм с фиксированными весами.

В первом случае, при обучении с учителем, в алгоритм входят данные входящего вектора, а также набор ожидаемого выходного вектора. Из так называемого тренировочного вектора в совокупности с соответствующим для него выходным вектором составляется тренирующая пара, где ожидаемый выходной вектор реализует функцию корректировки. Так, в число его функций входит коррекция весов, до состояния соответствия текущего выхода ожидаемому. Необходимо максимально приблизить их значения. Огромным плюсом нейронных сетей с учителем является их итеративность и способность выдавать результат максимально приемлемого уровня. Система не будет останавливать работу, пока выдаваемые сетью результаты не приведутся к приемлемому уровню расхождения. Одним из самых популярных алгоритмов обучения в искусственных нейронных сетях с

учителем является применение метода обратного распространения ошибки.

Во втором классе обучающих алгоритмов, который представляет собой искусственную нейросеть с алгоритмом обучения без учителя, в тренировочный набор входят только входные векторы. Алгоритм такого типа итеративен. При соответствии одинаковых входов одинаковым выходам нейронная сеть прекращает обучение, так как вся система принимает более организованный вид. Одним из примеров искусственных нейронных сетей, использующих такой обучающий алгоритм без учителя, являются самоорганизующиеся карты.

В третьем случае обучающих алгоритмов, когда используется алгоритм обучения с фиксированными весами, нейронные сети определяют веса, путем проведения некоторых предопределенных вычислений с помощью тренирующих векторов. В случае использования нейронных сетей Хопфилда (при помощи двунаправленной ассоциативной памяти) предопределенные обучающие алгоритмы действуют намного быстрее по сравнению с первыми двумя классами алгоритмов обучения.

Необходимо выделить, что по предоставляемым результатам обучающие алгоритмы без учителя являются более достоверными, так как предоставляют более отчетливые данные, по сравнению с нейронными сетями, использующими алгоритм обучения с учителем, который сравнивает текущий выход и желаемый результат.

После фазы обучения, искусственная нейронная сеть должна произвести расчет исходящих векторов, соответствующих каждому входящему вектору, что является фазой работы ИНС. На данной фазе работы, выведенные в результате обучения значения параметров остаются фиксированными, дальнейшее обучение не происходит во время рабочей фазы.

Обучение, которое представляет собой настройку параметров сети, может занимать довольно много времени. Для каждой задачи, поставленной

перед искусственной нейронной сетью, необходим подбор требуемых для их решения параметров. Выбор параметров сам по себе как процесс является очень времязатратным, так как значения параметров находятся путем перебора, методом проб и ошибок. Определение размера сети путем метода обучения также проводится несколько раз, пока не будет достигнут приемлемый результат. Каждый запуск нового сеанса обучения увеличивает общее время на настройку.

Для уменьшения затрачиваемого на обучение времени используют различные методы. Один из них — улучшение тренировочного алгоритма. Для улучшения алгоритма необходимо отталкиваться от стандартных моделей искусственных нейронных сетей, постепенно модернизируя механизм их работы. Улучшение стандартных моделей позволяет добиться большего процента точности и ускорить время обучения ИНС. Одним из самых эффективных способов уменьшения временных затрат является метод параллельной обработки данных.

1.2 Интервальные нейронные сети и их особенности.

Доказано, что нейронными сетями с любой непрерывной функцией активации одного переменного можно с наперед заданной точностью приблизить к любой непрерывной n мерной функции.[3] Нейронная сеть является высокоточной системой сложной многомерной нелинейной регрессии. К числу ее преимуществ относят:

1. Способность одновременного решения некоторого количества задач при обладании нейронной сетью таким же количеством выходов.

2. Способность обрабатывать зашумленные и неинформативные входные данные и возможность самостоятельно отфильтровывать неподходящие для решения определенной, поставленной перед нейронной сетью задачей с последующим сбросом их коэффициентов.

3. Способность при использовании заданных синаптических весов воссоздавать и проводить проверки возможных статистических моделей, и, при необходимости, улучшать их при помощи тренировки сети.

4. Способность использовать в качестве входных данных информацию разных типов и форматов. Во входном сигнале может быть одновременно непрерывная и дискретная, качественная и количественная информация. Для других методов реализация этой возможности является очень трудоёмкой задачей.

5. Способность некоторых алгоритмов решать обратные задачи, обучив нейронную сеть решению прямых задач. Чтобы реализовать это

3 В работах А.Н. Колмогоров (О представлении непрерывных функций нескольких переменных в виде суперпозиции непрерывных функций одного переменного) и В.И. Арнольд (О представлении функций нескольких переменных в виде суперпозиции функций меньшего числа переменных).

достаточно при обучении нейронной сети подавать выход нейронной сети, обученной решать прямую задачу, как входные параметры, а на выходе ожидать соответствующие входные параметры для прямой задачи.

6. Для работы с нейронной сетью не требуется столько квалификации работника, сколько необходимо для работы с массивными статистическими системами, реализующими подобный функционал.

Для обработки значений, представленных в виде интервалов следует использовать интервальные нейронные сети. Они необходимы, так как по исследованиям профессора Ishibuchi при использовании обычных нейронных сетей для прогнозирования возможны ошибки. Такие ошибки чаще всего возникают из-за превышений нижних границ интервалов по сравнению с верхними границами.

Интервальная нейронная сеть – это система взаимосвязанных и воздействующих друг на друга интервальных нейронов, у которых значения, поступающие на вход и получаемые на выходе, представлены в виде интервалов, то есть континуального множества значений, расположенных между двух границ интервала.

На рисунке 1.2.1 представлена интервальная нейронная сеть с одним скрытым слоем (где n – число входных нейронов, m – число скрытых нейронов, k – число выходных нейронов):

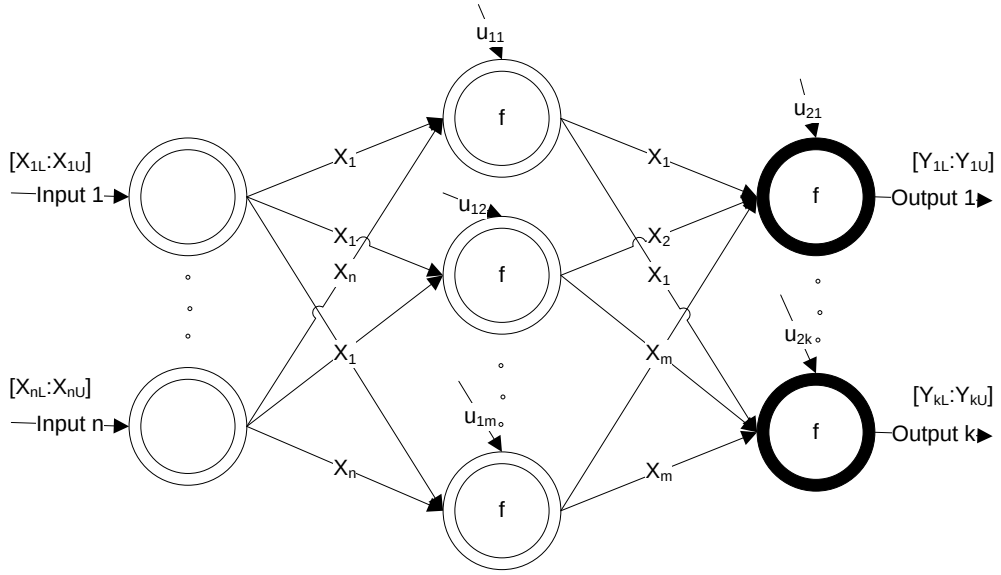


Рисунок 1.2.1. Интервальная нейронная сеть с одним скрытым слоем.

В рамках данной системы на каждый входной нейрон направляется интервальное значение, по которому впоследствии получают прогноз. В результате работы системы входные нейроны не изменяются, а остальные получают значения, рассчитываемые по формуле:

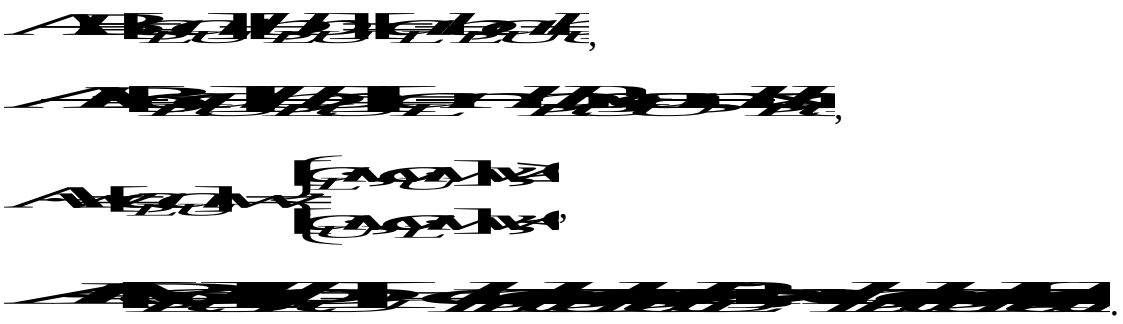
$$Y = f\left(\sum_{i=1}^{N} w_i X_i + u_i\right)$$

Здесь:

- Y – выход нейронной сети;
- w_i – весовой коэффициент i-ого входа;
- f – функция активации;
- N – число входов нейрона;
- u_i – весовой коэффициент смещения i-ого нейрона;
- X_i – интервальное значение поданное на i-ый вход;

Расчёт производится по правилам интервальной арифметики, что исключает возможность превышения нижней границы интервала верхней.

* представлена как любая операция.



При помощи интервальных значений, можно использовать в качестве входных параметров при прогнозировании следующие величины:

- Обычные интервальные величины (минимальная/максимальная температура за день, цена покупки/продажи);
- Значения измерительных приборов в интервале погрешности прибора;
- Обычные не интервальные величины, для которых нижняя и верхняя границы совпадают;
- Так же можно сократить количество входных значений, используя интервальные значения для указания интервала, в котором происходили изменения в течении часа или дня, вместо показаний за каждую минуту;

1.3. Самоорганизующиеся карты Кохонена.

Использование искусственных нейронных сетей часто связано с проведением научных исследований. Именно методы data-mining ИНС позволяют производить тщательную обработку полученных результатов, находить закономерности в больших объемах информации, распознавать изображения. Успешно справляется с такими задачами модель самоорганизующихся карт Кохонена, которая позволяет решить задачи выведения контекстных карт признаков по исследуемому объекту, а также проводить кластеризацию больших объемов данных.

Самоорганизующиеся карты Кохонена - это модель нейронной сети, предложенная ученым Т. Кохоненом, построенная на обучении без учителя и реализующая задачи кластеризации и визуализации данных.

Механизм работы самоорганизующихся карт схож с методом обработки окружающей среды мозгом человека. Принцип работы самоорганизующихся карт строится на модели отображения признаков. Так, схема работы напоминает работу мозга по обработке информации, полученной сенсорными датчиками тела человека, такими как слух, тактильная система, обоняние, зрение, вкус, сигналы которых распределяются по группам и отображаются на соответствующих полученной информации областях коры головного мозга человека.

Модель самоорганизующихся карт Кохонена не зацикливается на особенностях строения нервной системы человека, но использует подобную общую схему работы с информацией.

Таким образом, данная модель нейронной сети отображает многомерные входные пространства в виде двумерных выходных пространств. То есть происходит сжатие информации.

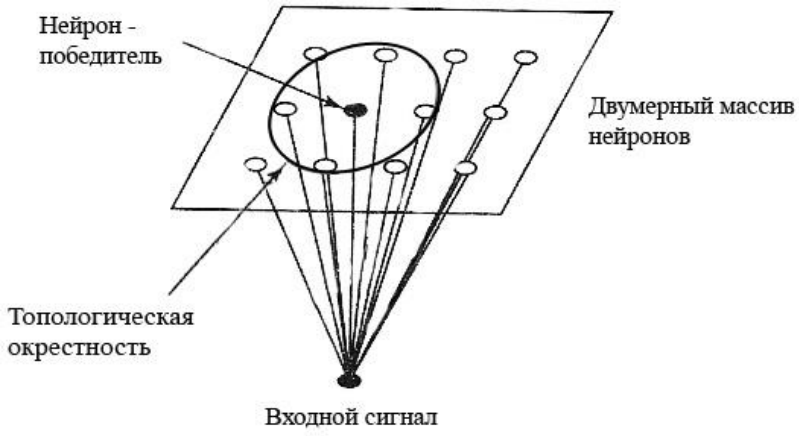


Рисунок 1.3.1. Модель работы самоорганизующихся карт.

Как было сказано выше, модель самоорганизующихся карт Кохонена относят к модели нейронных сетей с обучением без учителя, то есть в алгоритме нет конкретных схем обучения, только входные векторы обучения, по которым будет происходить настройка параметров нейронной сети.

Преимущества данной модели состоят как раз в подобии естественным схемам обработки информации мозгом человека в природе, по сравнению, например, с нейронными сетям, относящимися к классу обучения с учителем, где происходит сравнение информации с имеющимися в базе образцами. Такая схема работы мозга с информацией была бы крайне неудобной.

В обучении самоорганизующихся карт Кохонена используется правило конкурентного обучения. Так, нейрон, которые наиболее схож с входящим вектором становится главным. Именно такой нейрон переходит в активное состояние. Главный нейрон или победитель, в данной ситуации, определяет топологическую окрестность, где происходит корректировка необходимых возбуждаемых нейронов, то есть происходит упорядочивание нейронов в определенную систему координат.

Рассмотрим основной алгоритм системы обучения самоорганизующихся карт Кохонена.

Шаг 1: Процесс инициализации.

Первоначально исходные вектора синаптических весов

($w_i = \{w_{i1}, w_{i2}, \dots, w_{is}\}$, - s является размерностью входного вектора) заполняются случайными значениями.

Шаг 2: Процесс подвыборки.

Из входного пространства выбирают с определённой вероятностью вектор $x = \{x_1, x_2, \dots, x_s\}$. Этот вектор становится возбуждением, применяемым к решетке нейронов.

Шаг 3: Процесс поиска максимального подобия.

Происходит поиск главного из победивших нейронов (c(x)) по принципу критерия минимума Евклидова расстояния:

$$c(x) = \arg\min_j \|x - w_j\|$$

.

Шаг 4: Процесс коррекции.

Происходит корректирование весов нейронов, которые находятся в окрестности h(d,t) главное вектора (победителя). Для этого применяется следующая формула:

$$w_j(t+1) = w_j(t) + \alpha_i(t)h(d,t)(x_j - w_{ij}(t))$$

Формула применима для j-го веса *i*-го нейрона. В ней $\alpha(t)$ является коэффициентом скорости обучения, который зависим от времени обучения. Данный коэффициент принимает значения, входящие в область от 0 до 1.

Здесь α уменьшается в зависимости от уменьшения времени, но все же нет строгой зависимости, закона, по которому происходит такое уменьшение.

Для расчета коэффициента скорости используют следующую формулу:

$$\alpha_i = \alpha_0 exp^{-\frac{i}{t}}$$

Здесь i – это номер происходящей итерации, а *t* – общее количество

произведенных итерации в алгоритме обучения.

Для обозначения функции окрестности h(d,t) главное вектора используется гауссовская функция для нахождения окрестности:

$$h(d,t) = \begin{cases} 0, \text{при } d \geq \sigma(t) \\ e^{\left(-\frac{d}{2\sigma(t)}\right)}, \text{при } d < \sigma(t) \end{cases}$$

$$\sigma(t) = \sigma_0 * e^{\left(-\frac{t}{\mu}\right)}$$

здесь d – это расстояние на сетке координат нейронов, обозначенное как расстояние между текущим нейроном и нейроном-победителем, σ_0- это константа,

$$\mu = \frac{n}{\log_{10}(\sigma_0)}$$

– формула с n максимальным количеством производимых итераций.

Шаг 5: Процесс переходит снова к 2 шагу, происходит продолжение вычислений, пока не будет составлена окончательная карта признаков.

Такой процесс обучения нейронной сети также занимает довольно много времени, тем более при увеличении числа входных данных. Также в данном случае сам процесс настройки параметров является времязатратным, так как под каждую задачу подбирается несколько определенных параметров путем подбора. Итеративность процесса обучения также занимает много времени и в данной нейронной сети также целесообразно использовать параллельный алгоритм, который позволит понизить временные затраты и обрабатывать по нескольку элементов одновременно, ускоряя производимые вычисления.

В этой главе были рассмотрены нейронные сети обладающие огромным спектром областей применения, таких как классификация, кластеризация, категоризация образов, аппроксимация функций, прогнозирование,

оптимизация и управление. Был произведен подробный разбор двух примеров нейронных сетей, решающих разные задачи, а именно интервальные нейронные сети, хорошо справляющиеся с задачами прогнозирования и самоорганизующиеся карты Кохонена, реализующие механизм кластеризации. Обе рассмотренные модели обладают недостатком, связанным с долгим временем обучения и работы при больших входных данных. В то же время, они обладают хорошим потенциалом для проведения параллелизации их алгоритмов, что способно существенно сгладить недостаток, связанный с большой времязатратностью.

На основании этого было решено поставить следующие задачи:

1. Выявить возможные способы распараллеливания нейронных сетей.
2. Изложить методы параллельного программирования с использованием технологии CUDA.
3. Показать практическую реализацию параллельных алгоритмов нейронных сетей.

Глава 2. Методы и алгоритмы параллельного программирования.

2.1 Методы параллелизации искусственных нейронных сетей.

Параллельная обработка представляет собой объединение нескольких процессоров для решения задачи. Это позволяет ускорить время, которое затрачивается на вычисления, если сравнивать результаты использования параллельных процессов и результаты, предоставляемые одним процессором. Это позволяет решать объемные задачи и задачи фиксированной размерности намного быстрее. Если говорить о возможностях обработки информации человеческим мозгом, он так же использует метод параллельной обработки, так как содержит в себе огромное количество информации, обрабатывающих элементов и их взаимосвязи.

Модель искусственной нейронной сети содержит в себе несколько параллельных структур. Их используют для улучшения эффективности реализации на параллельных архитектурах. Существуют несколько типов уровней параллелизации в искусственных нейронных сетях.

Так Нордстремом были выделены основные уровни, где проводится параллелизации при фазе обучения:

1) уровень фазы обучения
2) уровень обучающей выборки
3) уровень слоя
4) уровень нейрона
5) уровень весов[4]

4 Боголюбов Д. П., Чанкин А. А., Стемиковская К. В. Реализация алгоритма обучения самоорганизующихся карт Кохонена на графических процессорах // Промышленные АСУ и контроллеры. 2012. № 10. С. 30-35

Все эти уровни параллелизации находятся в нейронной сети и могут использоваться одновременно. Для того, чтобы выбрать необходимый уровень необходимо отталкиваться от количества нейронов в сети, процессоров, особенностей компьютерной архитектуры ИНС и определенной задачи, поставленной перед ней.

При обучении искусственной нейронной сети происходит полное ее обучение по всей сети. Для того, чтобы определить необходимые параметры для решения определенной задачи, обучение состоит из множества сеансов экспериментального подбора. Например, подбираются параметры необходимого количества нейронов во всех слоях нейронной сети, скорости обработки данных (обучения).

Используя метод параллельной обработки данных при фазе обучения в искусственных нейронных сетях различные конфигурации сети могут исследоваться в одно и то же время. Так, на изображении представлен параллелизм тренировочной сессии, где различие между фазами обучения представлено в виде скорости обучения.

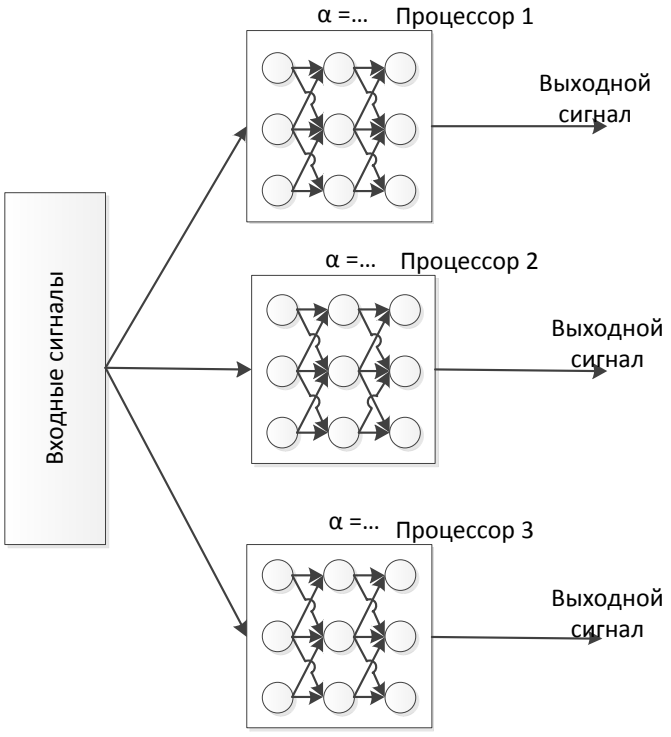


Рисунок 2.1.1. Схема параллелизации фазы обучения.

При стадии подбора параметров необходимо учитывать, что результаты, полученные при обучении зависят от изначальных значений весов. Так, при применении алгоритма обратного распространения ошибки с градиентной поисковой методикой результаты обучения очень чувствительны к первичным значениям весов. Аналогичная ситуация присутствует и в сетях Хопфилда при решении задач оптимизации.

При использовании же метода параллельной обработки данных при фазе обучения можно производить исследование сети при разных первичных установках. Так же при использовании параллелизации обучения среди плюсов выделяется линейный прирост быстродействия фазы. Это возможно благодаря отсутствию необходимости в взаимодействии между процессорами.

При использовании параллелизации на уровне обучения выборки используют обучение одновременно на нескольких разных обучающих выборках. Это важно, так как часто требуется довольно большое количество обучающих векторов в искусственной нейронной сети для решения определенной задачи объемного размера.

При системе, использующей один поток векторов, они будут направлены в сеть последовательно, поочередно, а при параллельной системе — обучающие вектора разделяются по процессорам, где каждый из процессоров обладает копией всех данных искусственной нейронной сети. То есть каждый процессор обучается на нескольких выборках. На схеме представлена данная схема работы:

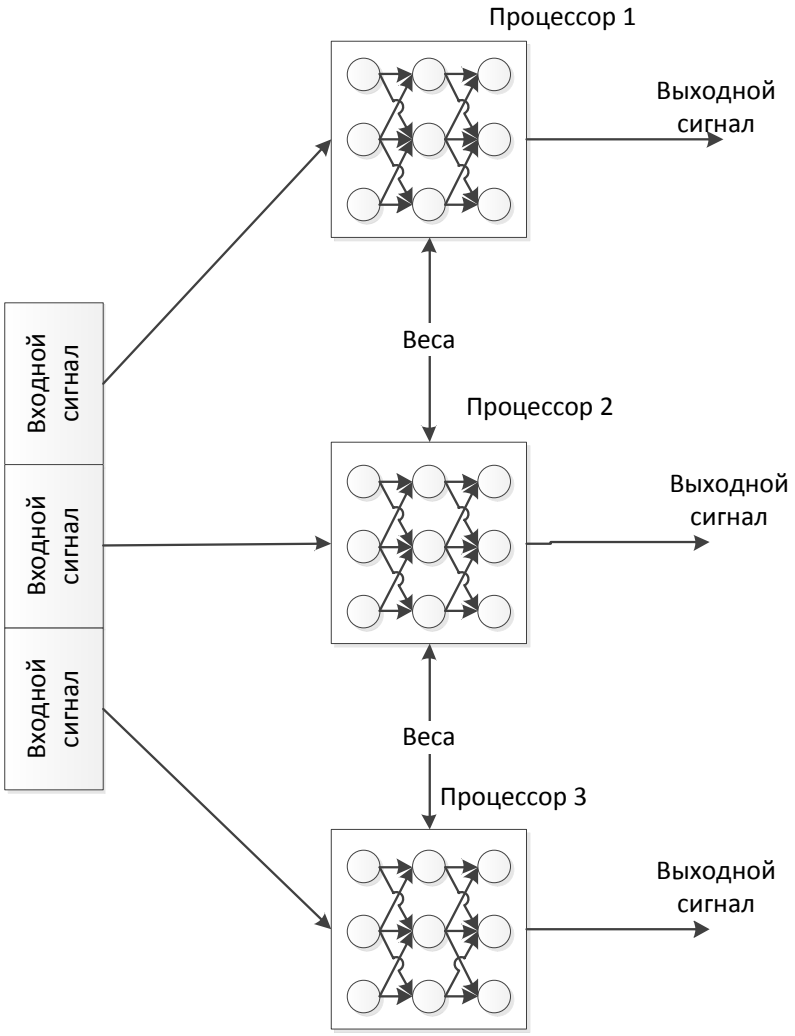


Рисунок 2.1.2. Схема параллелизации обучающей выборки.

При параллелизации уровней слоев, используемых в моделях неокогнитрона и в моделях с обратным распространением ошибки векторы, находящиеся на стадии обучения идут друг за другом по сети и находятся в сети в одно время. В то же время каналы некоторых сетей нейронной сети могут идти по обратному направлению. В модели с обратным распространением ошибки, они могут переходить и на обратные слои назад. Это позволяет передать операцию другому процессору. Такой же принцип может быть и в моделях искусственных нейроных сетей, которые не имеют слоев, но вектора так же идут через процессоры по принципу конвейера. Самые точные вектора, прошедшие через все процессоры, берутся за эталон.

После их повторного перехода по процессорам сети, процессоры обновляют веса в соответствии с этими эталонами.

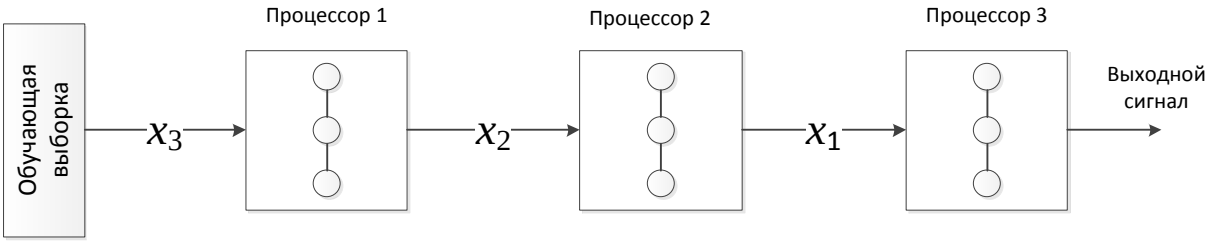


Рисунок 2.1.3. Схема параллелизации на уровне слоя.

При использовании метода параллелизации на уровне нейронов, нейрон по функциям аналогичен обрабатывающему процессору, так как является также обрабатывающим элементом. Параллельная обработка как процесс на данном уровне разделяет нейроны в рамках одного слоя по процессорам, а также при параллельных вычислениях. В рамках данной системы обработки нейрон или некоторое их количество соотносится с каждым процессором.

Такая модель параллелизации есть в каждой из моделей искусственных нейронных сетей и является одним из популярных методов. Схема работы данного метода:

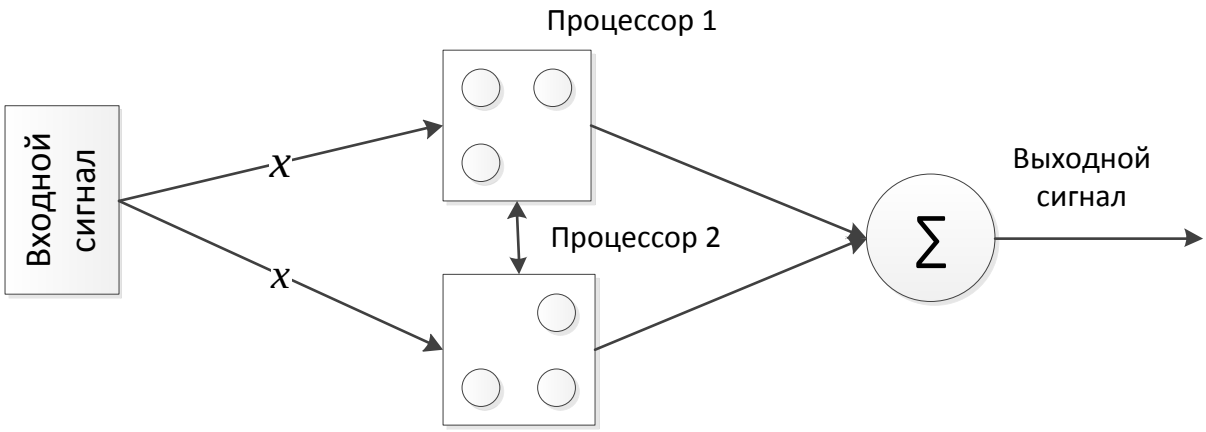


Рисунок 2.1.4. Схема параллелизации на уровне нейрона.

Уровень весов при методе параллелизации является мелкомодульным. Чаще всего данный метод используют в аппаратных реализациях. При данном методе вычисления в рамках одного нейрона разделены по нескольким процессорам.

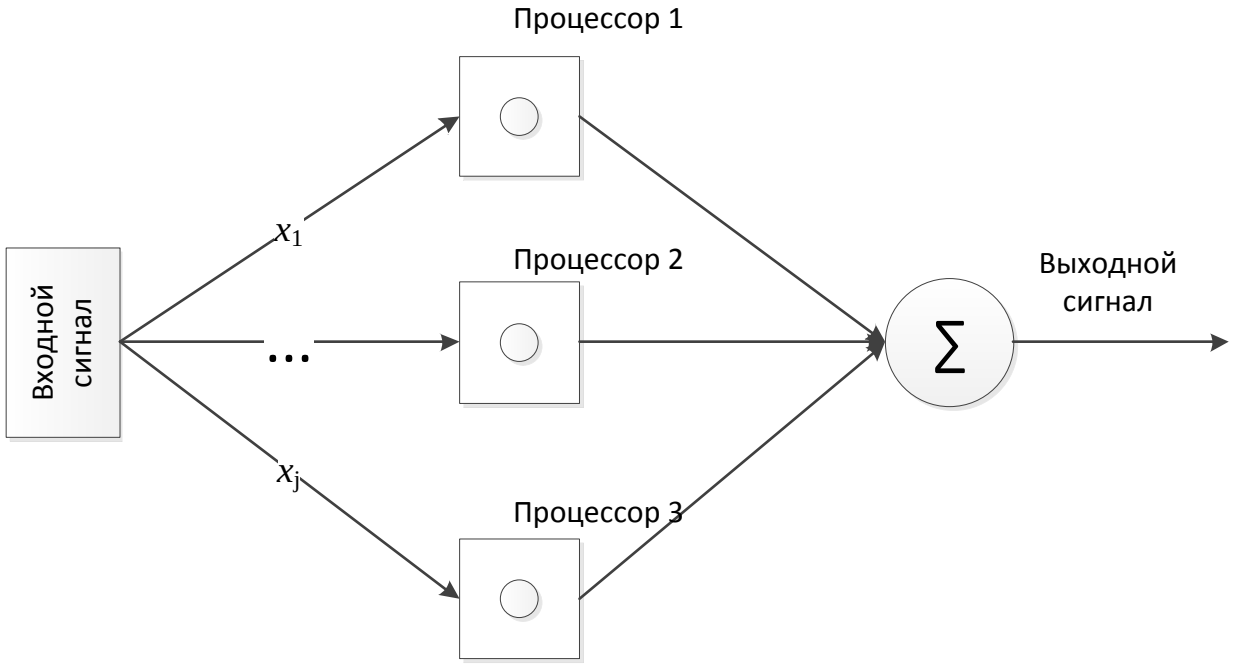


Рисунок 2.1.5. Схема параллелизации на уровне весов.

Для определения степени успешности применения вышеуказанных методов параллелизации необходимо выработать систему релевантных показателей. Для оценки успешности параллельного алгоритма используют такие релевантные показатели производительности как эффективность, увеличение скорости (прирост), масштабируемость системы. При этом сравнивается эффективность при использовании метода параллельной обработки и при обычной последовательной работе. При сравнении должен быть найден наиболее быстрый алгоритм для решения проблемы при использовании последовательной обработки, в таком случае будут видны преимущества использования параллельного метода. Также необходимо

иметь ввиду, что может быть несколько параллельных алгоритмов, однако не все наиболее быстрые алгоритмы могут быть подходящими для процесса параллелизации.

Для оценки скорости работы алгоритма сравним время самого быстрого из алгоритмов решения определенной проблемы при последовательной системе обработки данных и время, затраченное параллельным алгоритмом на определенном числе процессоров для решения той же проблемы. Если скорость выполнения параллельного алгоритма увеличилась во столько же раз, сколько процессоров было задействовано при расчёте, то мы имеем дело с линейным ростом. Если скорость выполнения увеличилась сильнее, то алгоритм суперлинейный, если скорость возросла меньше, чем при линейном росте, то алгоритм сублинейный. Так же нужно учитывать, что здесь при использовании параллельного алгоритма обрабатывающий элемент равнозначен последовательному алгоритму.

Рассмотрим эффективность процесса параллелизации. Эффективностью в данном случае является степень задействованности процессоров, которые используются при параллельном алгоритме обработки. При идеальном случае с линейным увеличением быстроты работы эффективность будет равнозначна 1. Но на деле в большинстве случаев при реализации параллельного алгоритма обработки бывает сублинейный прирост быстроты. Результирующая эффективность при этом находится на отметке от 0 до 1.

Показатель масштабируемости системы показывает способность системы воздействовать на ускорение обработки пропорционально количеству процессоров. Чем больше процессоров используется, тем более падает эффективность при фиксированном размере задачи. При поддержке эффективности системы путем увеличения размерности задачи, система параллелизации будет масштабируемой.

При увеличении числа процессоров для заданной задачи появляется

проблема сбалансированности нагрузки. При постоянном размере стоящей перед системой задачи нагрузка на процессоры уменьшается. Чтобы не было проблемы недостачи работы процессоров, используют масштабируемость задачи.

При увеличении числа процессоров эффективность падает. Так Амдал установил верхний порог прироста производительности, которого возможно достигнуть при использовании метода параллелизации. По его теории, на практике вычисления могут выполняться последовательно, не параллельно и, собственно, параллельно. Так как количество процессоров увеличивается, время работы, затраченное на параллельные вычисления уменьшается. При этом последовательная часть работы над вычислениями остается постоянной.

Время, затраченное на реализацию последовательных вычислений не масштабируется при увеличении числа процессоров и скоростной линейный прирост достигается только при отсутствии последовательной части при решении определенной задачи. Но так как большинство задач все же содержат в себе последовательные вычисления, максимальный прирост скорости ограничивается.

2.2 Алгоритм параллельного программирования интервальной нейронной сети и самоорганизующихся карт Кохонена.

Интервальные нейронные сети начинают работать заметно дольше при увеличении числа входных и выходных нейронов и при увеличении обучающей выборки. Для параллелизации интервальной нейронной сети эффективно использовать параллелизацию стадии обучения. При обучении интервальной нейронной сети методом обратного распространения ошибки невозможно создать тривиального алгоритма распараллеливания циклов обучения, внутри каждого цикла обучения можно эффективно применять параллелизацию. Так в классическом алгоритме поочередно рассчитывается погрешность прогнозирования для каждого входного вектора и на основании этих погрешностей пересчитываются соответствующие весовые коэффициенты. Но прогнозирование значений для каждого входного вектора можно делать не зависимо друг от друга на разный процессорах, что даст значительный прирос в скорости выполнения программы. Расчёт погрешностей и перерасчёт весовых коэффициентов тоже можно сделать параллельно.

Таким образом, благодаря распараллеливанию стадии обучения, возможно, при пропорциональном увеличении числа процессоров, увеличивать обучающую выборку и число входных и выходных нейронов, без увеличения временных затрат.

Одной из основных причин необходимости параллельной обработки в модели самоорганизующихся карт Кохонена является потребность в увеличении скорости обработки данных. Сама модель самоорганизующихся карт в принципе является удобным методом обработки большого количества информации, благодаря своему механизму работы, что позволяет сокращать время, которое человек может затратить на обработку своими силами. Использование же метода параллелизации в обучении и работе самой нейронной сети позволит существенно сократить время на обработку

информации даже при увеличении количества входных данных.

В данной главе были рассмотрены возможные схемы параллелизации для применения в искусственных нейронных сетях на различных уровнях. Самоорганизующиеся карты Кохонена являются моделью искусственных нейронных сетей класса обучения без учителя, для них также могут быть использованы рассмотренные уровни параллелизации.

При использовании параллелизации на уровне всей системы обучения нейронной сети Кохонена создаются копии всей сети и передаются на разные процессоры. При этом в них идет обучение на разных векторах и обмен результатами. В итоге собираются и обмениваются результаты с каждого процессора. Для того, чтобы при использовании данного метода параллелизации достигать эффективности стремящейся к 1, необходимо равномерно распределять между обрабатывающими элементами вычислительную нагрузку. В принципе, сама модель изначально осуществляет равномерное распределение нагрузки, путем передачи равного количества нейронов в разные обрабатывающие процессоры. При этом предполагается, что элементы, представленные для обработки идентичные.

Все же необходимо учитывать, что при корректировке коэффициентов могут быть небольшие различия по времени, затрачиваемом на требуемые вычисления, например, из-за того, что размер окрестности может быть даже больше самой сети. Также, следует учитывать, что так как обучающие вектора передаются на разные участки для обработки, они могут в итоге представляться в фазу корректировки с различным количеством нейронов.

Если рассматривать параллелизацию на уровне слоя, то надо иметь ввиду, что самоорганизующиеся карты Кохонена не обладают слоями. За исключением случаев, при которых обучающие вектора представляются в виде слоя. Следовательно, данный метод параллелизации не принесет увеличения скорости, тем более, что входной слой нейронной сети Кохонена не занимается вычислениями.

Метод параллелизации на уровне нейрона один из самых удачных

методов для реализации в самоорганизующихся картах Кохонена. Это связано с тем, что нейрон является обрабатывающим элементом, и применение параллелизации на уровне нейрона является естественным методом, напоминающим природный механизм работы мозга. Так, метод параллелизации нейрона часто используют в аппаратной параллелизации, например, на системах транспьютеров или на MIMD системе.

Отдельным сложным вопросом в алгоритме для самоорганизующихся карт является измерение его производительности. Обычно в ИНС она зависит различных параметров сети, например, от общего количества нейронов или от размера присущего входному вектору. В модели самоорганизующихся карт же трудно сравнивать производительность различных обрабатывающих модулей, настолько они различны.

Практически все параметры сети влияют на производительность.

Если брать окрестность, то она влияет на производительность по двум параметрам. К примеру, ее размер определяет количество нейронов для корректировки всех обучающих векторов. А также есть зависимость производительности от размера окрестности при завершающем этапе работы.

Также для обозначения количества изменений значений весов вводят CUPS, который показывает данное число изменений, произошедших за секунду. В самоорганизующихся картах CUPS зависит и изменяется от размера окрестности, но выходные коэффициенты нейронов вычисляются независимо от ее размера. Если окрестность большая, то больше времени идет на корректирование весовых значений. Также бывают случаи, при которых результаты выделяют как произведение произошедших итераций обучения за определенное время и весов. Такое возможно даже при не скорректированных весах.

Также, на производительность параллельного алгоритма влияют размерность вектора обучения, общее количество нейронов сети. При алгоритме параллелизации на уровне нейрона зависимость идет в основном от размера обучающего вектора. Это происходит в ситуациях распределения

всех векторов между процессорами еще до начала непосредственного процесса обучения искусственной нейронной сети. При увеличении числа нейронов в нейронной сети и увеличении обучающего вектора - время, необходимое на процесс коммуникации внутри системы уменьшается, при этом основное время затрачивается на процесс вычислений. От количества нейронов в сети и размера обучающего вектора также зависят количество вычислений, которые нужны при фазе вычислений, и при фазе корректирования.

На производительность также влияет насколько часто корректируются веса. Если их требуется довольно много, и они накапливаются, то число итераций между ними влияет на производительность. Так, к примеру, если рассматривать метод с обратным распространением ошибки, то число итераций в обучении растет при увеличении временных промежутков между процессами весового корректирования.

Для самоорганизующихся карт Кохонена значимым может быть использование online или offline алгоритмов. Так, при втором случае, в варианте с редким корректированием весов само обучение в принципе носит не очень успешный результат. То есть также необходимо определить необходимое число корректировок, для успешного проведения обучения.

Рассмотрим модель параллелизации самоорганизующихся карт Кохонена, где параллельный алгоритм основывается на процессе обучения. В данном алгоритме применена параллелизация на уровне нейрона и на уровне весов. Это позволяет добиться максимально эффективного результата.

Под структуру самоорганизующихся карт Кохонена хорошо подходит мелкомодульная параллелизация. При таком методе происходит разделение довольно простых операций между отдельными элементами, которые занимаются обработкой.

В рассматриваемом методе параллелизации к начальному алгоритму обучения самоорганизующихся карт внесли некоторые изменения. Будут указаны эти изменения применительно к рассматриваемому ранее

пошаговому методу работы самоорганизующихся карт Кохонена:

1 шаг: Процесс инициализации

Данный этап происходит без изменений классического алгоритма обучения карт Кохонена, так как происходит в самом начале обучения однократно и не несет в себе особенной временной нагрузки. На данном этапе применение метода параллелизации не совсем уместно.

2 шаг: Процесс определения подвыборки

В классическом варианте нейронной сети Кохонена обучающие вектора проходят в сети поочередно. В данном случае имеет место реализовать одновременную обработку сразу нескольких различных входящих векторов. Это происходит в ситуациях, при которых окрестность очень маленькая, но вектора находятся на большом расстоянии друг от друга. Для реализации подобного параллельного алгоритма нужно предварительно производить обработку всех входных данных.

3 шаг: Процесс поиска максимального подобия

Для увеличения скорости путем применения параллельного алгоритма процесс поиска разделяют на две части. С одной стороны, необходимо определить расстояние от нейронов до входных векторов. Здесь используется распараллеливание на уровне отдельных весов. Вычисления происходят в этом случае на всех потоках. По модели карт Кохонена такое расстояние между элементами рассчитывается как Евклидово.

С другой стороны, необходимо найти самый подходящий элемент. Например, поиск минимального элемента происходит с введением параллелизации с помощью редукции объема расстояний между нейроном и входным вектором с предыдущего подэтапа. Здесь редукция будет реализоваться так: в случае редукции массива, происходит попарное сравнение элементов. Происходит замена i-ым элементом. Также процедуру повторяют для подмассивов. Необходимо продолжать процесс до тех пор, пока подмассив не станет единичной длины.

4 шаг: Процесс корректирования

Для более эффективного проведения алгоритма обучения данный процесс также необходимо разделить. Сначала необходимо произвести расчет между нейронами, главным образом между всеми нейронами и главным нейроном-победителем. Это необходимо для определения расстояния окрестности. Необходимо выяснить, попадают ли туда нейроны. Это расстояние также находится путем применения формулы Евклидова расстояния. Однако есть различия в подсчете при латеральном расстоянии, где тоже расстояние рассчитывают в выходном пространстве.

Вторая фаза процесса корректировки включает в себя корректирование весов нейронов, которые попадают в область окрестности главного нейрона-победителя. Здесь процесс параллелизации корректировки весов таких нейронов как раз происходит при помощи параллелизации на уровне отдельных нейронов. Для начала с помощью формулы expression необходимо определить относится ли рассматриваемый нейрон к окрестности победившего нейрона, а только затем произвести корректирование весов по формуле expression. Для каждой итерации вычисляется свой коэффициент обучения исходя из общего числа произведённых итераций.

5 шаг: перехода ко второму шагу, продолжение процесса обучения нейронной сети. В сеть передается следующий входной вектор из общей совокупности обучающих элементов. Процесс обучения продолжает свою работу до тех пор, пока все входные вектора не пройдут через данную схему.

На схеме представлен рассматриваемый параллельный алгоритм самоорганизующихся карт Кохонена. Для описания представленного алгоритма необходимо рассмотреть такие понятия как барьер и поток. Так, поток представляет собой наименьший элемент обработки данных, а барьер представляет собой механизм взаимосвязи, синхронизации потоков при реализации параллельного алгоритма.

После процесса инициализации весов нейронов случайными значениями в сеть начинают поступать первые входные векторы из общей

базы векторов. Для каждого из потоков с помощью проведения вычислений находится расстояние между компонентами векторов весов нейронов и входящих векторов. Полученный результат сохраняется в общем массиве. С помощью барьерной синхронизации происходит координирование потоков обработки данных между собой. Данный процесс происходит на этом и последующем этапе обучения самоорганизующихся карт Кохонена. Такое координирование прекращается только после того как все потоки дойдут до данного барьера.

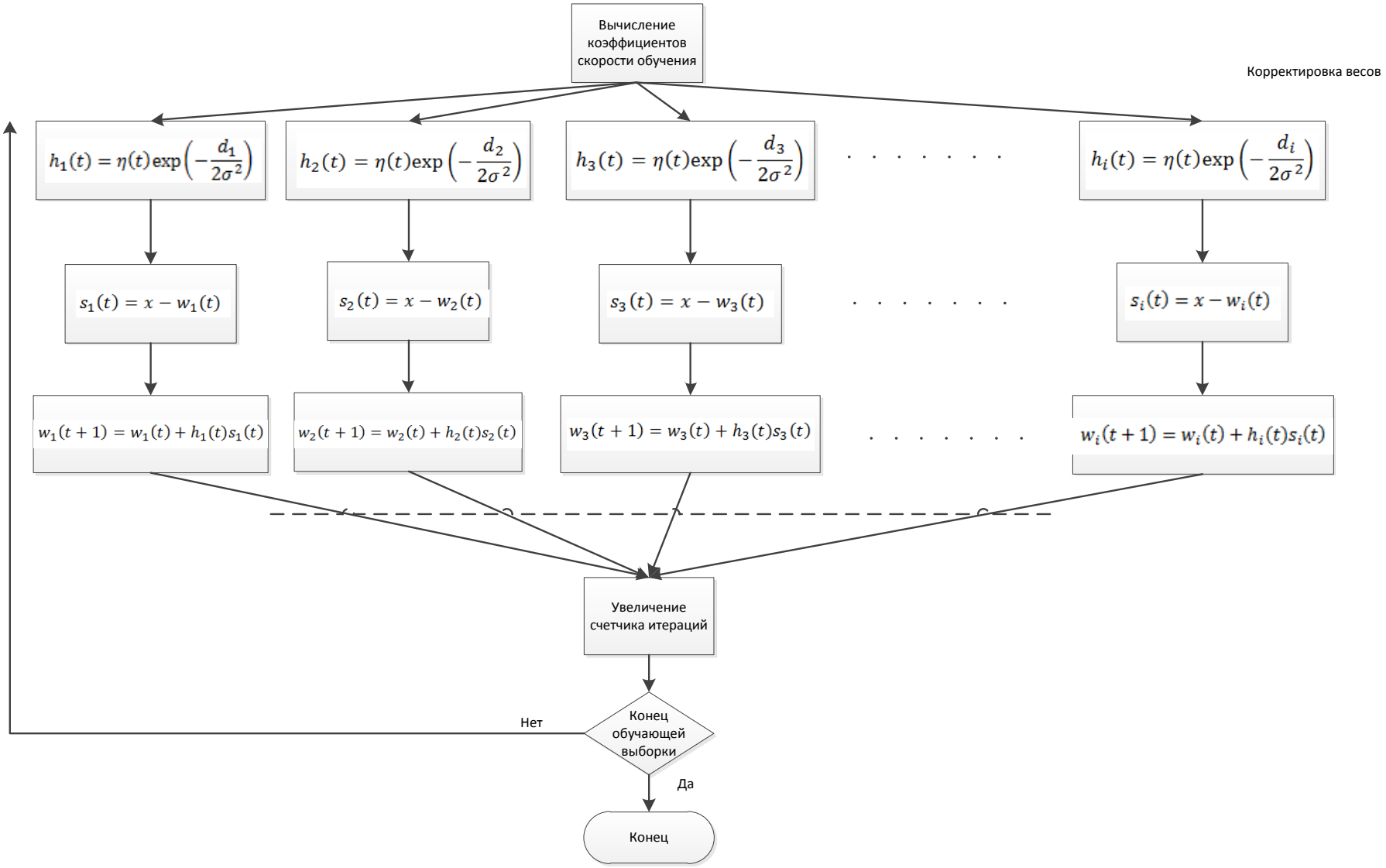


Рисунок 2.2.1. Схема рассматриваемого параллельного алгоритма обучения самоорганизующихся карт Кохонена.

В результате, полученный массив расстояний обрабатывается с помощью параллельного алгоритма по методу редукции, что представляет собой следующий этап работы. Так полученный массив, содержащий в себе расстояния между нейронами разделяется на пару частей, а каждому потоку соотносится элемент из каждого из подмассивов. Происходит процесс сравнения, в результате которого самые маленькие элементы относятся в новый массив. Процесс такого отбора элементов приводит к тому, что в результате остается один главный элемент — победитель.

На следующем этапе корректирования весов происходит параллелизация на уровне нейронов. Здесь использование данного метода параллелизации позволяет обойти проблему создания и загруженности потоков, которые потом не участвуют в корректировке. Для каждого потока соотносится по одному нейрону для обработки данных. Происходит расчет латерального расстояния, находящегося между главным победившим нейроном и обрабатываемым нейроном. Так определяется принадлежность обрабатываемого нейрона к окрестности победившего элемента. Если нейрон относится к окрестности победителя, то на потоке начинается последовательное корректировение весовых коэффициентов нейрона.

2.3 Методы параллельного программирования на CUDA.

Современные графические процессоры прошли длинный путь от машин для обработки и рендеринга графики до современных массово-параллельных процессоров, способных выполнять полный спектр вычислительных задач. Самые передовые игровые графические процессоры уже достигли производительность в 8.988 TFLOPS[5].

С появлением первых полноценных графических процессоров от компании NVIDIA они привлекли внимание не только разработчиков компьютерных игр, но и учёных, которые обратили внимание насколько эффективны GPU в работе над вычислениями с плавающей точкой. Благодаря учёным из Стэндфордского университета в 2003 году появилось расширение языка Си, которое позволило работать с параллельными структурами данных на графическом процессоре. Компания NVIDIA не осталась в стороне, и уже в 2006 году была анонсированная разработка программно-аппаратной технологии CUDA, которая позволила писать программы для графических процессоров на высокоуровневом языке программирования и задействовать GPU как процессор общего назначения.

В основе архитектуры CUDA лежит масштабируемый массив потоковых мультипроцессоров (Streaming Multiprocessors, дальше SM). Такой мультипроцессор способен обрабатывать параллельно сотни нитей. Для управления работой такого массива потоков была разработана уникальная архитектура – Single-Instruction, Multiple-Thread.

SIMT – это подход к параллельным вычислениям при котором несколько потоков выполняют одни и те же операции на разных данных.

Технология CUDA строится на следующем принципе работы: GPU или по-другому device — устройство является массивно-параллельным

[5] Nvidia GeForce GTX Titan Z // http://versus.com/ru/nvidia-geforce-gtx-titan-z-vs-nvidia-quadro-k600

сопроцессором для CPU или host, где выполняется последовательный код. Параллельный код, применяемый для параллельных вычислений производится на GPU несколько раз на нескольких параллельных нитях, а функции, которые работают параллельно на всех нитях графического процессора являются ядрами kernel. Такие функции являются расширенными функциями языка C.

По принципу технологии CUDA CPU и GPU взаимодействуют для обеспечения работы ядра, что включает в себя также создание и прекращение рабочего цикла ядра.

Существует три понятия, которые лежат в основе идеи расширения языка C для его применении, воплощенном в технологии CUDA:

- Разделение памяти,
- Барьерная синхронизация,
- Иерархия групп нитей.

В соответствии с данными понятиями выстраивается следующая структура работы нитей:

Верхний уровень grid или сетка — это массив блоков blocks, который может быть, как одномерным, так и двумерным. Среди отличительных особенностей сетки является ее непредсказуемость очередности запуска блоков и отсутствие общей памяти между блоками. Сетка соответствует всем нитям, которые выполняют данные функции (ядро). Каждый блок на данном верхнем уровне представляет собой массив нитей, который также может быть одномерным, двумерным или трехмерным. Нитей в составе блока может быть до 512. Блоки в сетке являются соразмерными и имеют свои адреса, индексы. Нить внутри блоков также имеет свой индекс.

Процессы, объединенные в блоки, имеют внутри общий объем памяти shared memory и синхронное исполнение.

Таким образом, нити, выполняющие какую-нибудь определенную задачу организовываются в структуру следующим образом:

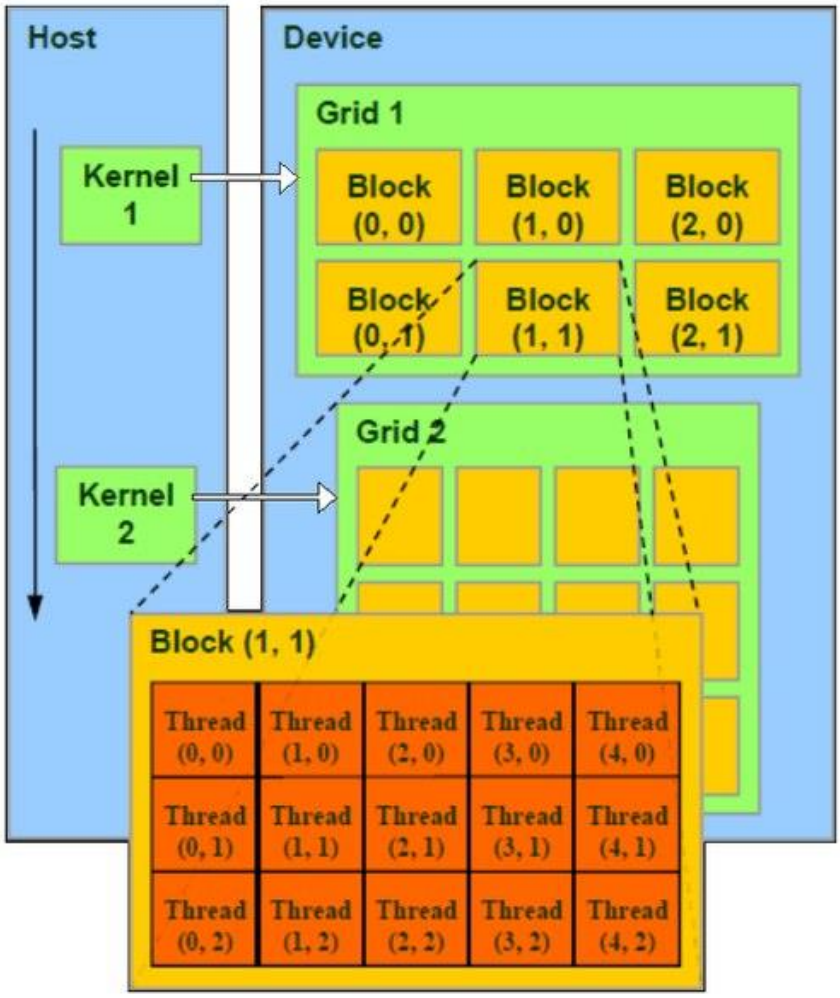


Рисунок 2.3.1. Схема организации иерархии нитей.[6]

При получении мультипроцессором одного или нескольких блоков нитей для выполнения функции, эти блоки разделяются на группы - warp, которые состоят из 32 нитей. Каждый warp получает данные о своей очереди выполнения через warp scheduler. Разделение на warp происходит изначально от схемы работы компьютерной графикой, где при ее обработке использовалось считывание больших пакетов, данных и распределение его между нитями. При считывании непоследовательных данных, не

6 Сайт компании NVIDIA http://www.nvidia.ru/

объединенных в общий массив созданные нити практически не используются и процесс не организован.

Поэтому следует тщательно отслеживать распределение данных в памяти, оптимизировать его. Необходимо учитывать и то, что нити разных из warp могут быть на разных уровнях выполнения программы, а нити из одного warp выполняются все в одно время.

Для выполнения программы каждому блоку данных соответствует один мультипроцессор, где блок функционирует отдельно от остальных блоков. Нити в рамках блока могут использоваться на одном мультипроцессоре одновременно, также возможно использование на одном мультипроцессоре нескольких отдельных блоков. Такая обработка данных на мультипроцессоре является последовательной, блоки на нем обрабатываются поочередно.

В зависимости от используемой системы и ее ресурсов выполнение работы блоков может происходить по-разному, например, в произвольном порядке, а также может выполняться последовательно или одновременно.

Масштабируемость системы происходит благодаря коммуникативным ограничениям между нитями, которые построены путем организации различных видов памяти и соответствующими им уровнями доступа.

У каждой нити есть свой личный объем памяти, так называемая локальная память local memory. Сообщение между нитями в рамках одного блока происходит благодаря разделяемой памяти shared memory, где сообщение происходит с довольно низкой задержкой.

Global memory — глобальная общая память, является объемным участком памяти и передает информацию с высокой задержкой. Тем не менее глобальная память доступна из CPU и является по сути единственными каналом сообщения между GPU и CPU.

На схеме представлена организация памяти в CUDA.

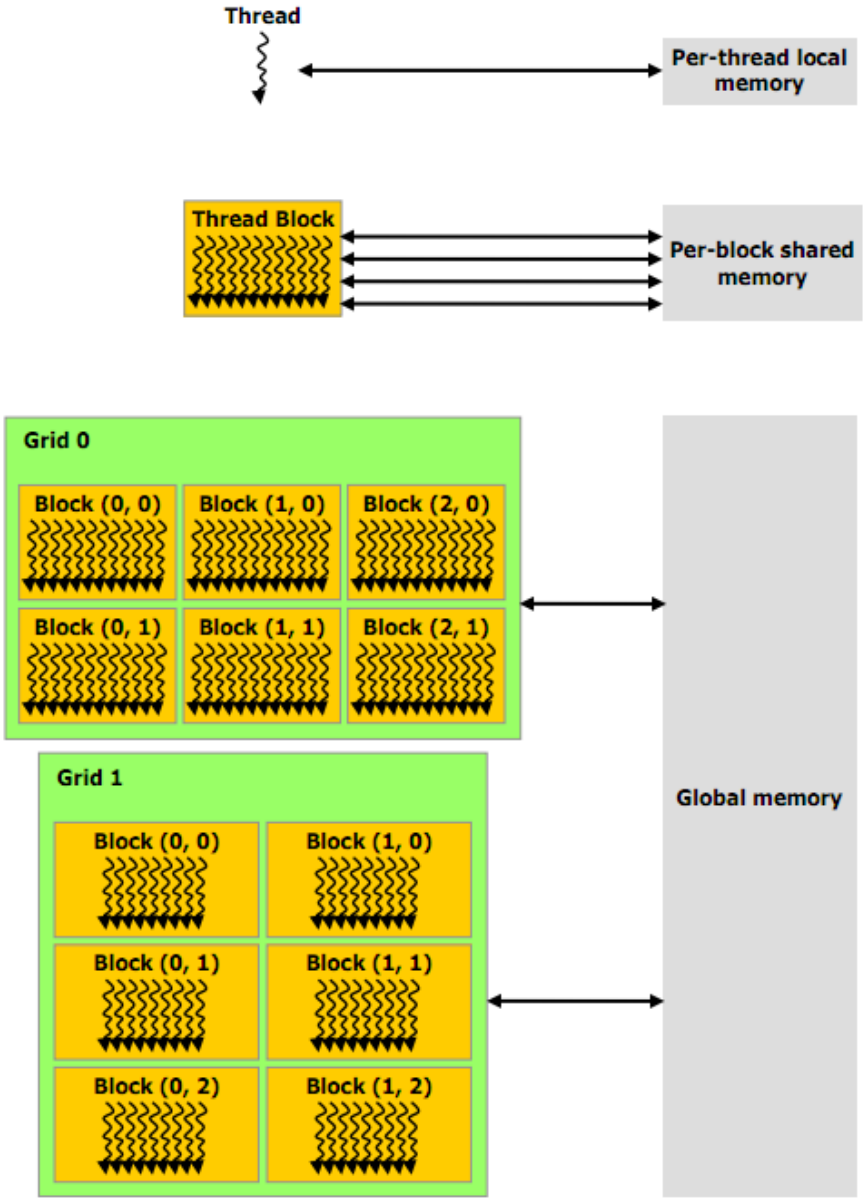


Рисунок 2.3.2. Организация уровней памяти.

Существует также два дополнительных вида памяти — текстурная память и константная память. Они доступны для всех нитей программы, но открыты лишь для чтения. Также они являются оптимизированными для некоторых отдельных форматов памяти, обладающих своей спецификой.

Взаимодействие нитей между собой происходит благодаря встроенной функции _syncthreads(). Также с помощью нее происходит барьерная синхронизация, характерная для CUDA. С помощью этой функции

обозначают точки синхронизации внутри ядер. Также барьерная синхронизация запрещает нитям выполнять команды, пока все нити не будут выполнять эту функцию.

В данной главе были рассмотрены методы параллелизации нейронных сетей в общем случае и на двух конкретных примерах. Так же была рассмотрена технология CUDA, которая является доступным и очень удобным инструментом для создания параллельный алгоритмов для графических процессоров. Рассмотренные примеры показали, что параллелизация нейронных сетей действительно способна ускорить существующие алгоритмы.

Глава 3. Реализация параллельного алгоритма на CUDA.

3.1 Реализация параллельного алгоритма интервальной нейронной сети и самоорганизующихся карт Кохонена.

Для начала будет рассмотрена реализация системы поддержки принятия решений на основе интервальной нейронной сети с обратным распространением ошибки и генетическим алгоритмом. Особое внимание будет уделено именно реализации параллельного алгоритма и использованию для этого технологии CUDA.

Изначально идёт последовательный код, выполняемый на CPU. Здесь интересна только эта часть:

```
cudaError_t cudaStatus = cudaSetDevice(0);
if (cudaStatus != cudaSuccess)
{
  free(INPUT_NEURONS);
  return 5;
}
```

Данный код проверяет наличие на компьютере пользователя CUDA совместимой графической карты.

После подготовки всех входных данных для работы с ними последовательной частью программы, программа начинает подготовку графической карты для обработки этих данных:

```
for(int z=0;z<net_num;z++)
{
    cudaStreamCreate(&stream1[z]);
    cudaStreamCreate(&stream2[z]);
    cudaMalloc((void**)&gpuWih[z], (INPUT_NEURONS[z]+1)*HIDDEN_NEURONS[z]*sizeof(float));
    cudaMalloc((void**)&gpuWho[z], (HIDDEN_NEURONS[z]+1)*OUTPUT_NEURONS[z]*sizeof(float));

    cudaMalloc((void**)&gpuDWih[z], (INPUT_NEURONS[z]+1)*HIDDEN_NEURONS[z]*sizeof(float));
    fillByNull<<<INPUT_NEURONS[z]+1,HIDDEN_NEURONS[z],0,stream1[z]>>>(gpuDWih[z]);
    cudaMalloc((void**)&gpuDWho[z], (HIDDEN_NEURONS[z]+1)*OUTPUT_NEURONS[z]*sizeof(float));
    fillByNull<<<HIDDEN_NEURONS[z]+1,OUTPUT_NEURONS[z],0,stream2[z]>>>(gpuDWho[z]);

    cudaMalloc((void**)&gpuInputs[z], (num_inp[z]*(inp_win_size[z]+k[z]-1))*2*sizeof(float));
    cudaMalloc((void**)&gpuActual[z], OUTPUT_NEURONS[z]*2*sizeof(float));
    cudaMalloc((void**)&gpuTarget[z], (OUTPUT_NEURONS[z]+k[z]-1)*2*sizeof(float));
```

```
    cudaMemcpy(gpuWih[z], wih[z], (INPUT_NEURONS[z]+1)*HIDDEN_NEURONS[z]*sizeof(float),
    cudaMemcpyHostToDevice);
    cudaMemcpy(gpuWho[z], who[z], (HIDDEN_NEURONS[z]+1)*OUTPUT_NEURONS[z]*sizeof(float),
    cudaMemcpyHostToDevice);
    threads[z]=dim3(INPUT_NEURONS[z],HIDDEN_NEURONS[z]);
    max_NEURONS = INPUT_NEURONS[z];
    if (INPUT_NEURONS[z]<OUTPUT_NEURONS[z])
      max_NEURONS = OUTPUT_NEURONS[z];
    sharedSize[z]=(INPUT_NEURONS[z]+HIDDEN_NEURONS[z]+OUTPUT_NEURONS[z]*2+(INPUT_NEURONS[z]+
    1)*HIDDEN_NEURONS[z]+(HIDDEN_NEURONS[z]+1)*OUTPUT_NEURONS[z]+HIDDEN_NEURONS[z]+
    OUTPUT_NEURONS[z]+max_NEURONS*HIDDEN_NEURONS[z])*2*sizeof(float);
    sample[z]=0;

    cudaStreamCreate(&stream1[z]);
    cudaStreamCreate(&stream2[z]);
}
```

В этой части программы при помощи функции cudaMalloc выделяется память в необходимом объёме на графической карте для переменных. При этом те переменные, которые необходимо предопределить нулями заполняются при помощи функции fillByNull. Ниже представлен её код:

```
__global__ void fillByNull(float*c)
{
    c[blockDim.x*blockIdx.x+threadIdx.x]=0;
}
```

Данная функция вызывается с центрального процессора и выполняется параллельно на графической карте. Вместе с вызовом функции для параллельной обработки запускается ядро с нужным числом блоков и нужным числом нитей в созданном заранее потоке обработки. Все необходимые для этого параметры передаются в угловых скобках после названия функции и перед параметрами, передаваемыми ей.

Позже, в этом же коде, данные из переменный с CPU копируются в соответствующие переменные на GPU при помощи функции cudaMemcpy с параметром cudaMemcpyHostToDevice.

Далее в программе в бесконечном цикле идет следующая

последовательность действий:

1. последовательный код, который передаёт на графический процессор входной нейрон и значение, которое он должен выдать на выходе нейронной сети. Здесь интересны эти команды:

```
cudaMemcpyAsync(gpuInputs[z], inputs[z], (num_inp[z]*(inp_win_size[z]+m[z]))*2*
sizeof(float),cudaMemcpyHostToDevice, stream1[z]);

cudaMemcpyAsync(gpuTarget[z], target[z], (OUTPUT_NEURONS[z]+m[z])*2*sizeof(float),
cudaMemcpyHostToDevice, stream2[z]);
```

Команда cudaMemcpyAsync с параметром cudaMemcpyHostToDevice так же, как cudaMemcpy передаёт данные с центрального процессора на графическую карту с тем отличием, что в данном случае программа не ждёт завершения копирования данных. За счёт этого происходит уменьшение времени работы программы.

2. Далее, для всех обучаемых нейронных сетей выполняется следующий код:

```
for(int z=0;z<net_num;z++)
{
    cudaStreamSynchronize(stream1[z]);
    cudaStreamSynchronize(stream2[z]);

    backPropagate<<<1,threads[z],sharedSize[z],stream2[z]>>>(gpuWih[z], gpuDWih[z],
    gpuWho[z], gpuDWho[z], &gpuInputs[z][(sample[z]%k[z])*num_inp[z]*2], gpuActual[z], &
    gpuTarget[z][(sample[z]%k[z])*2], INPUT_NEURONS[z], HIDDEN_NEURONS[z], OUTPUT_NEURONS
    [z]);
    sample[z]++;
}
```

Здесь команда cudaStreamSynchronize заставляет программу дождаться выполнения всех процессов, запущенных в указанном потоке. Если не указать эту команду, то могут возникнуть проблемы с тем, что программа будет обрабатывать не те данные, которые необходимо.

После синхронизации запускается функция backPropagete в одном блоке, с необходимым количеством нитей. Код данной функции почти полностью последователен и только в местах, где необходимо дождаться выполнения всех нитей в данном блоке, в коде, встречается

команда __syncthreads().

Выполнение цикла прекращается после того как число итераций превысит установленное ограничение.

После обучения сети при помощи метода обратного распространения ошибки идет тестирование сети, при котором, после подготовки всех необходимых входных данных, запускается функция прогнозирования:

```
feedForward<<<1,threads[z],sharedSize[z],stream1[z]>>>(gpuWih[z], gpuWho[z],
 gpuInputs[z], gpuActual[z], INPUT_NEURONS[z], HIDDEN_NEURONS[z],
OUTPUT_NEURONS[z], stream1[z]);

cudaStreamSynchronize(stream1[z]);

cudaMemcpy(actual[z], gpuActual[z], OUTPUT_NEURONS[z]*2*sizeof(float),
cudaMemcpyDeviceToHost);
```

Функция прогнозирования запускается так же, как и функция обучения в одном блоке на множестве нитей. Каждая нить делает прогноз для своего входного вектора. Таким образом, одновременно может выполняться до тысячи прогнозов вместо одного при последовательном подходе.

Затем программа дожидается завершения прогнозирования для всех входных данный и, при помощи команды cudaMemcpy с параметром cudaMemcpyDeviceToHost, копирует спрогнозированные данные с графической карты в оперативную память.

На последнем шаге спрогнозированные интервалы сравнивают с целевыми и рассчитывается погрешность расчёта. Полученные ошибки выводятся как результат работы сети. Все полученные весовые коэффициенты нейронной сети передаются на CPU.

Перед окончание работы происходит освобождение всей выделенной памяти при помощи операций: cudaFree(<имя переменной>) для переменной на графическом процессоре и free(<имя переменной>) для переменной на центральном процессоре; уничтожение созданных потоков при помощи операции cudaStreamDestroy(<имя потока>).

В этой реализации просматривается множество случаев

параллелизации алгоритма в тех местах, где при последовательном алгоритме находился бы цикл. В результате незначительных изменений последовательного кода произошёл заметный рост быстродействия системы.

Дальше рассмотрена реализация параллельного алгоритма самоорганизующихся карт Кохонена.

Как всегда, алгоритм начинается с последовательного кода, который подготавливает данную систему. После подготовки идет вызов функции:

```
cudaStatus = CudaKsomRun(&map, test_array,&num_test);
if (cudaStatus != cudaSuccess) {
    fprintf(stderr, "CudaKsomRun failed!");
    return 1;
}
```

Так, в программе вызвана функция CudaKsomRun, которая запускает обучение заданной карты заданным тестовым массивом, сгенерированным в последовательном коде. В случае ошибки запуска об этом совершается запись в файл.

После того как программа приступает к выполнению функции CudaKsomRun, первым делом происходит инициализация всех переменных, которые понадобятся программе при работе на графической карте. Следующая команда выбирает устройство для работы с технологией CUDA.

```
cudaStatus = cudaSetDevice(0);
if (cudaStatus != cudaSuccess) {
    fprintf(stderr, "cudaSetDevice failed!  Do you have a CUDA-capable GPU installed?");
    goto Error;
}
```

Если необходимое устройство не найдено об этом создаётся запись в log файле. Так же здесь и в дальнейших участках текста указатель goto Error; направляет программу к моменту освобождения всей выделенной памяти и сбросу выбранных устройств в самом конце функции CudaKcomRun.

```
cudaStatus = cudaMalloc((void**)&dev_nodes, map->num_nodes * map->options.fv_size *
sizeof(float));
if (cudaStatus != cudaSuccess) {
    fprintf(stderr, "cudaMalloc failed!");
    goto Error;
}
```

Команда cudaMalloc из фрагмента сверху резервирует место на графической карте для переменной dev_nodes необходимого размера. Таким же образом происходит резервирование места и для всех остальных переменных, необходимых при вычислениях на графическом процессоре. Как и в предыдущих случаях при ошибке выделения памяти сообщение об этом записывается в файл.

```
cudaStatus = cudaMemcpy(dev_nodes, n_nodes, map->num_nodes * map->options.fv_size *
sizeof(float), cudaMemcpyHostToDevice);
if (cudaStatus != cudaSuccess) {
    fprintf(stderr, "cudaMemcpy failed!");
    goto Error;
}
```

Все входные данные при помощи команды cudaMemcpy с параметром cudaMemcpyHostToDevice копируются из оперативной памяти в память графического процессора. В случае неудачи, сохраняется сообщение об ошибке работы с памятью.

После подготовительной работы вызывается следующая функция:

```
ksom_fv_distance<<<num_nodes_c, map->options.fv_size>>>(dev_nodes, dev_fv,
dev_fv_dist,dev_fv_size);
```

Данная функция выполняется параллельно для каждого входного вектора и вычисляет расстояние от входного вектора до каждого из узлов.

```
cudaStatus = cudaDeviceSynchronize();
if (cudaStatus != cudaSuccess) {
    fprintf(stderr, "cudaDeviceSynchronize returned error code %d after
    launching ksom_fv_distance!\n", cudaStatus);
    goto Error;
}
```

Команда cudaDeviceSynchronize дожидается завершения всех параллельных процессов, после чего продолжается выполнение программы.

После некоторой подготовки переменных происходит параллельный вызов функции:

```
ksom_bmu_search<<<grid_size, block_size>>>(dev_fv_dist, dev_bmu,
dev_half_size);
```

Параллельная функция ksom_bmu_search сравнивает два поданных на вход расстояния и выбирает из них меньшее. После программа синхронизирует потоки, дожидаясь завершения всех работающих нитей.

Далее программа, последовательно запустив две параллельные функции ksom_xy_distance и ksom_nodes_update, предварительно подождав синхронизации потоков, первой функцией высчитала необходимое смещение к нужному узлу для каждого из входных нейронов, а второй обновила узлы.

После выполнения этих функций происходит обновление данных карты Кохонена.

```
cudaStatus = cudaMemcpy(n_nodes, dev_nodes, num_nodes_c * map->options.fv_size *
sizeof(float), cudaMemcpyDeviceToHost);
if (cudaStatus != cudaSuccess) {
    fprintf(stderr, "cudaMemcpy failed!");
    goto Error;
}
```

Перед завершением заботы функция cudaMemcpy с параметром cudaMemcpyDeviceToHost копирует данные об узлах обученной только что программой карты Кохонена с графической карты в оперативную память. После этого функция освобождает всю выделенную на графической карте память и завершает свою работу, возвращая управление основной функции, которая, выводя на экран данные о времени работы, тоже завершает свою работу.

После проведенного анализа видно, что почти всё время алгоритм карты Кохонена работал в параллельном режиме, что заметно уменьшает временя работы программы.

Таким образом, подробно разобрав параллельную часть алгоритма работы двух нейронных сетей, перейдём к рассмотрению компьютерного эксперимента и проанализируем ускорение работы предложенных алгоритмов.

3.2 Компьютерный эксперимент.

Используемые в данной работе реализации параллельных алгоритмов нейронных сетей являются готовыми программами и правильность их работы была проверена за рамками данной дипломной работы. Здесь проведены тесты показывающие эффективности параллельных алгоритмов по сравнению с последовательными.

Для проверки эффективности работы данного алгоритма интервальной нейронной сети были произведены тесты времени работы последовательной, работающей только на CPU и параллельной, использующей при работе GPU, реализации разработанной оболочки. Для этого запускали одновременное обучение одной, двух и трех сетей, содержащих по 27 нейронов, состоящее из 10 итераций по 1000 циклов обучения. Результаты этого теста представлены на графике.

Рисунок 3.2.1. Сравнительные характеристики скорости последовательной и параллельной реализации.

Из графика видно, что выигрыш от параллельного обучения нескольких сетей почти линейно увеличивается при увеличении числа обучаемых сетей.

Для проверки эффективности параллельного алгоритма самоорганизующейся карты Кохонена сравним время выполнения последовательного и параллельного алгоритма.

Для этого проведём серию экспериментов для сетей с размерностями 25x25, 50x50 и 100x100. Каждый раз для обучения будем использовать один и тот же массив из 60000 векторов размерности 100 сгенерированных случайным образом по нормальному распределению из интервала (0,1).

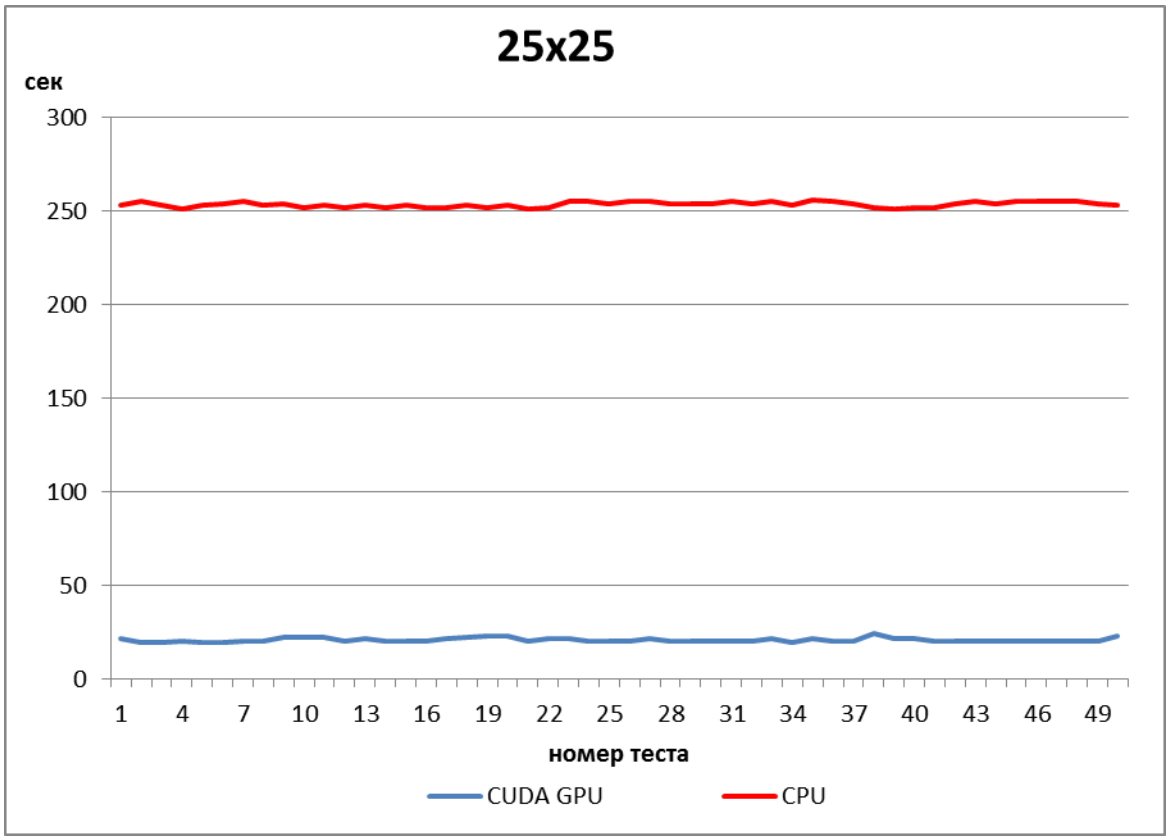


Рисунок 3.2.2. Иллюстрация серии экспериментов на карте размеров 25x25.

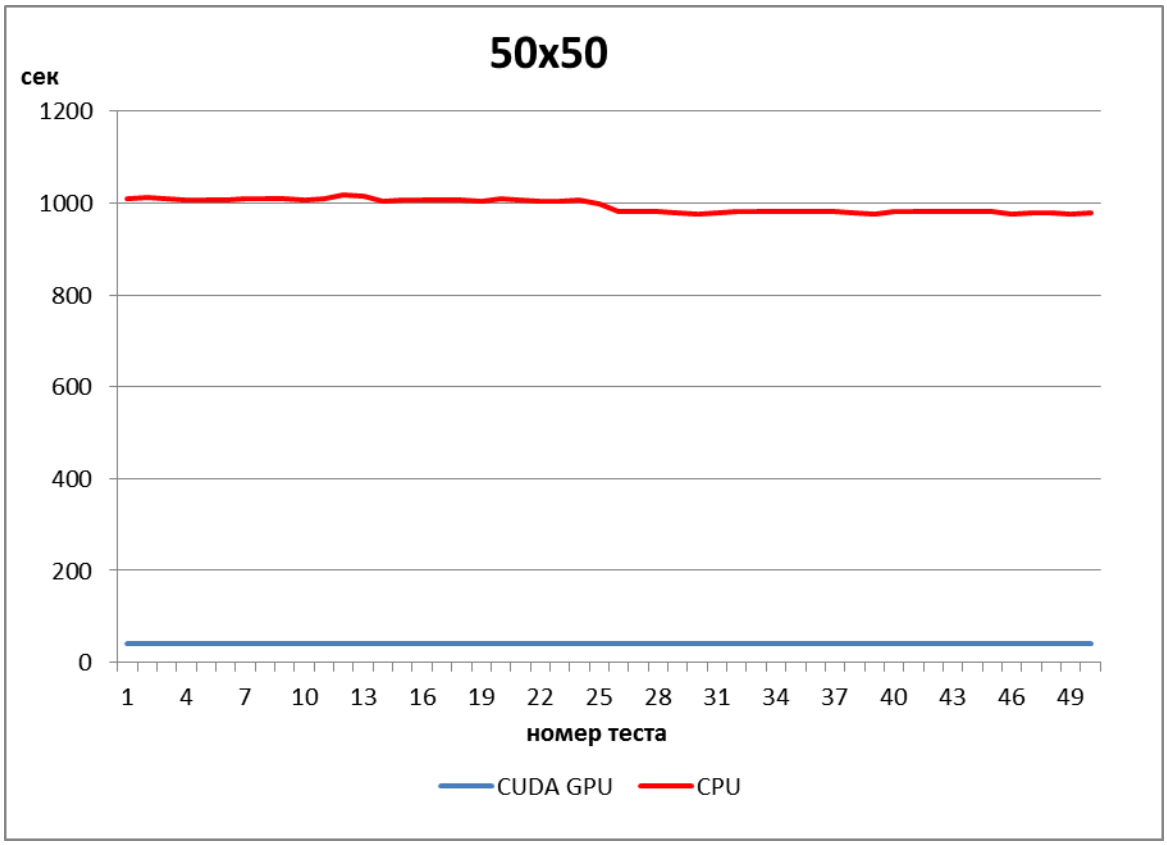


Рисунок 3.2.3. Иллюстрация серии экспериментов на карте размеров 50x50.

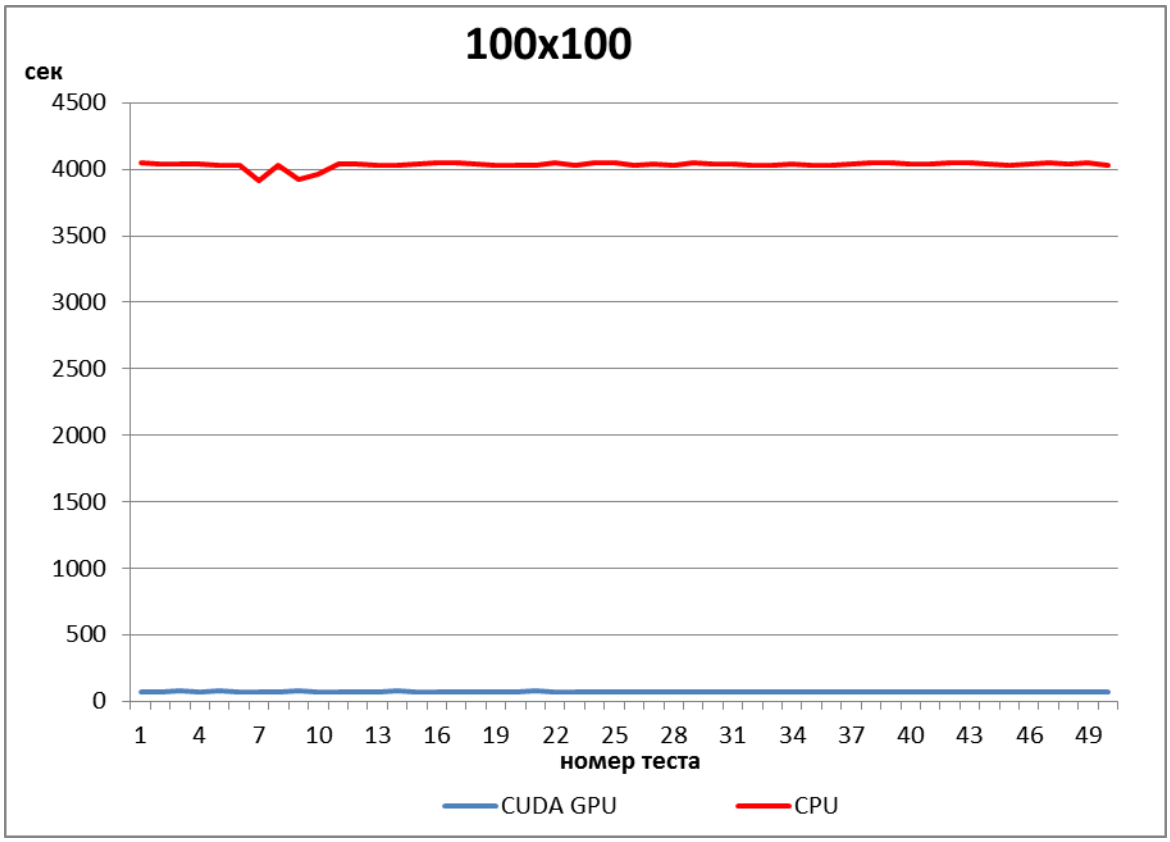


Рисунок 3.2.4. Иллюстрация серии экспериментов на карте размеров 100x100.

После приведённых тестов можно смело утверждать, что использование параллельных алгоритмов даёт существенный прирост в

скорости работы алгоритма самоорганизующихся карт Кохонена. При этом чем больше размерность карты, тем существеннее выигрыш при использовании параллельного подхода. Это происходит потому, что время выполнения последовательной части кода, которая содержится в параллельной реализации, при увеличении размерности карты практически не увеличивается, тогда как время выполнения полностью последовательной реализации растёт пропорционально размерности карты.

Заключение.

В ходе выполнения дипломной работы были реализованы все поставленные задачи. Так же была достигнута цель работы, а именно были проанализированы параллельные алгоритмы нейронных сетей для выявления наиболее эффективных способов реализации на практике наиболее быстрых и качественных приложений для обработки большого объёма информации.

В данной дипломной работе были рассмотрены нейронные сети и два частных примера нейронных сетей, интервальные нейронные сети и самоорганизующиеся карты Кохонена. После подробного рассмотрения примеров был выявлен существенный недостаток нейронных сетей, а именно, при большом объёме входных данных время их работы становится не приемлемым для использования их для решения практических задач.

Было решено, что данный недостаток возможно исправить, используя параллельные алгоритмы программирования при реализации нейронных сетей на практике. Был произведён общий обзор методов параллелизации нейронных сетей, что подтвердило предположение, что использование параллелизации нейронных сетей способно ускорить их работу. Были рассмотрены два примера параллелизации алгоритмов нейронных сетей.

Была рассмотрена технология CUDA, позволяющая писать программы, поддерживающие возможность распараллеливания вычислений на графическом процессоре. После чего был произведён подробный разбор параллельных алгоритмов интервальной нейронной сети и самоорганизующихся карт Кохонена реализованных с использованием технологии CUDA. Стоит отметить лёгкость написания программы для графического процессора при помощи технологии CUDA. Программный код совершенно не значительно отличатся от последовательного аналога этому коду. При этом, согласно проведенному в данной работе исследованию производительности распараллеленного кода, программы, написанные для графического процессора, работают значительно быстрее своих

последовательных аналогов.

Резюмируя проделанную работу заметим, что нейронные сети при использовании параллельных алгоритмов являются очень мощным инструментом, который способен помочь специалистам из разных областей, а технология CUDA является крайне удобной технологией для реализации этих алгоритмов. На рынке существует богатый выбор графических процессоров, поддерживающих технологию CUDA, что позволяет использовать в своей работе нейронные сети, реализованные для таких графических процессоров, большему числу людей.

Список литературы.

1. Kohonen T. “The self-organizing map”, Proceedings of the Institute of Electrical and Electronics, 1990, vol. 78, p. 1464 – 1480
2. Nordstrom T. Designing parallel computers for self-organizing maps. Forth Swedish Workshop on Computer System Architecture, Linkoping, 1992. [Труды конференции]
3. Боголюбов Д. П., Чанкин А. А., Стемиковская К. В. Реализация алгоритма обучения самоорганизующихся карт Кохонена на графических процессорах // Промышленные АСУ и контроллеры. 2012. № 10. С. 30-35
4. В.В. Круглов, М.И. Дли, Р.Ю. Голунов "Нечеткая логика и искусственные нейронные сети": учебное пособие для студентов вузов, обучающихся по специальности "Прикладная информатика". - М.: Физматлит, 2001
5. Лахман Константин «Правило Хебба: «универсальный нейрофизиологический постулат» и великое заблуждение математиков» // http://habrahabr.ru/post/102305/
6. Сандерс Дж., Кэндрот Э. Технология CUDA в примерах: введение в программирование графических процессоров: Пер. с англ. – М.: ДМК Пресс, 2011.
7. Старченко А. В. и др. «Практикум по методам параллельных вычислений» Учебник/Под ред. А.В. Старченко. - М.: Издательство Московского университета, 2010
8. Хайкин С. Нейронные сети: полный курс, 2-е издание. : Пер. с англ. – М.: Издательский дом «Вильямс», 2008