**УДК: 004.272.23**

*Д. П. Боголюбов, А.А. Чанкин, аспирант, К.В. Стемиковская, студент*

*Московский Государственный Институт Электроники и Математики*

**РЕАЛИЗАЦИЯ АЛГОРИТМА ОБУЧЕНИЯ САМООРГАНИЗУЮЩИХСЯ КАРТ КОХОНЕНА НА ГРАФИЧЕСКИХ ПРОЦЕССОРАХ**

**Аннотация**

*В настоящей статье предложен один из способов параллельной реализации самоорганизующихся карт Кохонена с помощью технологии CUDA. Описывается программная реализация и результаты её тестирования, показывающие рост производительности с увеличением размерности сети по сравнению с последовательной версией алгоритма.*

Ключевые слова: нейронные сети, параллельные вычисления, вычисления на графических процессорах, CUDA.

*Bogolyubov D.P., Chankin A.A., Stemikovskaya K.V.*

*Moscow State Institute of Electronics and Mathematics*

**IMPLEMENTATION OF A LEARNING ALGORITHM FOR KOHONEN’S SELF-ORGANIZING MAP ON GRAPHICAL PROCESSOR**

**Abstract**

*In this article we introduce a CUDA-based implementation of Kohonen self-organizing map. We describe software implementation and test results confirming performance growth with increasing size of neural network comparative to serial version of algorithm.*

Keywords: SOM, parallel computing, neural networks training algorithm, GPGPU, CUDA.

1. **Введение**

В настоящее время нейросетевые технологии являются популярным инструментом во многих областях научных исследований. С их помощью проводится анализ результатов экспериментов, так как нейросетевые методы позволяют эффективно справляться с обработкой больших массивов данных. Однако процесс обучения нейронной сети под определенную задачу зачастую превышает допустимые временные рамки эксперимента. Об одном из способов снижения временных затрат и пойдет речь в этой статье.

Рассматриваемая нами модель искусственных нейронных сетей носит название самоорганизующейся карты Кохонена (Kohonen self-organizing map, в дальнейшем SOM). Наиболее успешно с её помощью решаются задачи кластеризации массивов данных и построения контекстных карт признаков [1].

Принцип работы самоорганизующихся карт Кохонена частично воспроизводит один из способов обработки информации головным мозгом – отображение входов нервных сенсоров, таких как зрение или слух, на соответствующие области коры головного мозга. Модель Кохонена не концентрируется на нейробиологических особенностях и является более общей в том смысле, что осуществляет отображение многомерного входного пространства на двумерное (реже одномерное) выходное пространство, т.е. проводит сжатие данных.



*Рисунок 1. Модель работы самоорганизующихся карт.*

SOM принадлежат к классу нейронных сетей с алгоритмом обучением без учителя. В этом алгоритме не существует примеров, по которым проводится обучение, присутствует лишь некая мера качества, согласно которой настраиваются параметры сети.

Плюсом таких моделей является их близость к естественным методам обработки информации, поскольку трудно представить существование в головном мозге некоего механизма, сравнивающего получаемую информацию с существующими образцами.

Самоорганизующиеся карты реализуют правило конкурентного обучения – нейрон, наиболее «близкий» к входящему вектору, назначается победителем и переходит в активное состояние. Победитель определяет топологическую окрестность, в которой производится корректировка соответствующих возбуждаемых нейронов. Таким образом происходит упорядочивание нейронов в виде значимой системы координат [2].

Классический алгоритм обучения самоорганизующихся карт [1] выглядит следующим образом:

1. *Инициализация.* Вначале случайным образом инициализируются исходные значения синаптических весов где s – размерность входящего вектора.

2. *Подвыборка*. Происходит выбор вектора из входного пространства. Вектор представляет собой возбуждение, которое применяется к решетке нейронов.

3. *Поиск максимального подобия.* Производится поиск победившего нейрона с(*x*), используя критерий минимума Евклидова расстояния

4. *Коррекция.* Корректировка весов нейронов, входящих в окрестность нейрона-победителя, производится по следующей формуле (для *j*-го веса *i*-го нейрона на n-ной итерации)

где – коэффициент скорости обучения, зависящий от времени. Он принимает значения в области от 0 до 1. Значение α должно уменьшаться со временем, однако нет определенного закона, по которому должно происходить это уменьшение. В нашем случае коэффициент скорости обучения рассчитывался как:

где N – общее количество итераций алгоритма обучения.

В качестве функции окрестности нами использовалась гауссова функция окрестности

где *–* расстояние на координатной сетке между текущим нейроном i и нейроном-победителем,

 *σ0* - начальное значение эффективной ширины

Значение временной константы определяется как  *.*

5. *Продолжение.* Происходит возврат к шагу 2, вычисления продолжаются до тех пор, пока в карте признаков не перестанут происходить заметные изменения.

Однако, несмотря на кажущуюся простоту алгоритма, обучение нейронной сети может оказаться достаточно времязатратным. Для эффективного обучения под определенную задачу требуется произвести подбор таких, параметров, как размерность входящих векторов, количество нейронов, а также коэффициент скорости обучения. Даже после того, как размер сети определен, обучаемая сеть может не давать приемлемого результата, и нужно будет проводить новое обучение. Каждый из таких сеансов может занять большое количество времени, как вычислительного, так и реального.

Одним из методов повышения производительности является использование параллельной обработки, при которой несколько обрабатывающих элементов одновременно участвуют в производимых вычислениях. В приложении к нейронным сетям это может использоваться, в частности, для ускорения фазы обучения.

1. **Возможные способы параллелизации.**

Согласно Нордстрёму [3] существует 5 методов параллелизации фазы обучения ИНС. В равной степени это относится и к самоорганизующимся картам, однако эффективность применения того или иного метода для SOM отличается от других моделей ИНС. Ниже мы вкратце расскажем об этих принципах параллелизации.

1. Параллелизация фазы обучения

При решении задачи с помощью нейронной сети обычно проводится множество экспериментов с целью установления необходимых параметров, таких как параметр скорости обучения и количество нейронов в различных слоях. С использованием параллелизации фазы обучения несколько различных конфигураций нейронной сети могут быть исследованы одновременно. Единственное различие между двумя фазами обучения будет состоять в единственном параметре, например, скорости обучения.

При использовании параллелизации фазы обучения линейный прирост быстродействия легко достижим, поскольку исчезает необходимость межпроцессорного взаимодействия.

1. Параллелизация обучающей выборки

Часто размер обучающей выборки, необходимой для обучения сети применительно к задаче может быть достаточно большим. В однопоточной системе обучающие вектора будут направлены в сеть по одному. В параллельной системе эти обучающие вектора можно разделить между процессорами. Тогда каждому процессору необходима полная копия нейронной сети. Таким образом, несколько копий сети могут обучаться одновременно на разных обучающих выборках.

1. Параллелизация на уровне слоя.

В таких моделях ИНС, как сеть с обратным распространением ошибки и неокогнитрон, обучающие вектора проходят через сеть по принципу конвейера. Таким образом, несколько обучающих векторов могут находиться в сети одновременно.

В отличие от вышеперечисленных, самоорганизующиеся карты не являются многослойной моделью, кроме случая, когда распространение обучающих векторов представлено в виде слоя. Поскольку входной слой не производит никаких вычислений, а только распределяет входные вектора на нейроны основного слоя, то при параллелизации на уровне слоя в модели SOM нельзя достигнуть никакого прироста производительности.

1. Параллелизация на уровне нейрона

Параллелизация на уровне нейрона является наиболее очевидной из присутствующих в модели нейронной сети, поскольку нейрон как обрабатывающий элемент нейронной сети аналогичен отдельному процессору. Параллелизм на уровне нейрона состоит в разделении нейронов (внутри слоя, если используется модель с несколькими слоями) среди процессоров, и последующем параллельном вычислении. Каждому процессору выделяются один или более нейронов сети. Параллелизация на уровне нейрона присутствует во всех моделях нейронных сетей, и это наиболее популярный метод в большинстве реализаций, вне зависимости от используемой модели нейронной сети.

1. Параллелизация на уровне весов.

Вычисления в пределах нейрона также могут быть разделены между несколькими процессорами. Это является мелкомодульным параллелизмом, в первую очередь он представлен в аппаратных реализациях.

1. **Технологии реализации параллельных алгоритмов ИНС.**

Почти все вышеописанные методы параллелизации успешно используются для ускорения работы алгоритмов нейронных сетей. Для реализаций используются как постоянно наращивающие мощность высокопроизводительные суперкомпьютеры, так и нейрокомпьютеры, построенные на нефоннеймановской архитектуре.

Суперкомпьютеры являются наиболее распространенным решением для реализаций различных моделей нейронных сетей. С различной эффективностью для высокопроизводительных вычислений используются массивно-параллельные машины, вычислительные кластеры с производительностью в несколько тысяч Gflops [4].

Реализации на нейрочипах являются куда менее распространенным решением, в первую очередь за счет своей высокой стоимости. В настоящее время на рынке отсутствуют нейрочипы массового производства, большинство их разработок производится под заказ, для выполнения специфических задач, что делает их малодоступными для рядового пользователя [5].

Однако и многопроцессорные системы также являются решением, зачастую подходящим только для крупных научных центров. Кроме того, за счет узкой специализации большинства решений сильно затрудняется возможность их масштабирования и использования для задач более общего типа.

Однако, существует технология, позволяющая осуществлять высокопроизводительные вычисления с помощью средств, находящихся практически в каждом современном ПК. Речь идет о технологии GPGPU, в перевод c англ. General-purpose graphics processing units - графические процессоры общего назначения. Первоначально графические процессоры использовались исключительно для обработки компьютерной графики, и для того, чтобы осуществить сторонние вычисления, были необходимы специфические знания в области обработки трехмерной графики.

Технология CUDA является программно-аппаратной архитектурой, позволяющей программисту писать полнофункциональные приложения с использованием дополнительной библиотеки CUDA Cи.

К настоящему времени в мире продано более 128 миллионов графических процессоров, поддерживающих технологию CUDA. NVIDIA предлагает широкий выбор устройств, от мощных вычислительных систем Tesla, стоимостью в несколько тысяч долларов и сконструированных специально для высокопроизводительных вычислений, до дискретных видеокарт ION, использующихся в портативных компьютерах, на которых также возможно достичь небольшого прироста скорости по сравнению с вычислениями на CPU [6]. Для наших целей была выбрана оптимальная по соотношению цены и производительности видеокарта GeForce GTX 680 с объемом памяти 2048 MB GDDR5 [7].

1. **Основные принципы технологии CUDA**

Архитектура CUDA спроектирована в виде масштабируемого массива потоковых мультипроцессоров (Streaming Multiprocessors). Используемая нами видеокарта GTX680 содержит 1526 CUDA ядер, объединенных в 8 мультипроцессоров. Основная идея архитектуры CUDA состоит в том, что GPU (обозначаемое как device) выступает в роли массивно-параллельного сопроцессора к CPU (используется термин host). При этом последовательный код выполняется на CPU, а для параллельных вычислений соответствующий код выполняется на GPU N раз на N параллельных нитях. Функции, выполняемые параллельно на всех создаваемых нитях, носят название ядер (kernel), и представляют собой расширенные функции языка Си.

В основе идеи библиотеки расширений языка Си для CUDA лежат три абстрактных понятия – иерархия групп нитей, разделение памяти и барьерная синхронизация. API интерфейс CUDA скрывает низкоуровневые подробности создания нитей на устройстве, однако распределение вычислительных функций между нитями, а также работа с многоуровневой памятью остаются задачей разработчика [8].

 CPU взаимодействует с GPU для инициализации и прекращения работы ядер. В нашем случае GPU и CPU работают в блокирующем режиме. В этом режиме CPU запускает одно ядро и ждет до полного окончания его выполнения.

Все запущенные на выполнение нити организуются в следующую структуру:



*Рисунок 2. Организация иерархии нитей.*

Верхний уровень организации соответствует всем нитям, выполняющим данное ядро. Для его обозначения используется термин «сетка» - grid. Сетка представляет собой одномерный или двумерный массив блоков. Каждый блок – это одномерный, двумерный или трехмерный массив нитей, в блоке их может содержаться до 512. Все блоки, образующие сетку, имеют одинаковую размерность (для используемой нами видеокарты GTX680 максимальная поддерживаемая размерность составляет 1024 x 1024 x 64). Каждый блок в сетке имеет свой адрес, индекс блока в сети, также как и нить имеет свой индекс внутри блока.

Когда мультипроцессор получает один или несколько блоков нитей для выполнения (в GTX680 на каждом мультипроцессоре может выполняться одновременно до 2048 нитей), он разделяет их на группы по 32 нити - warp’ы. При этом только нити в пределах одного warp’а выполняются физически одновременно. Нити из разных warp’ов могут находиться на разных стадиях выполнения программы, поэтому вызываемые нити не должны зависеть от очередности работы друг друга. Иначе существует опасность возникновения так называемого состояния гонки - когда результат работы программы зависит от того, какая нить первой получит доступ к ячейке данных.

Блоки также могут выполняться в произвольном порядке, одновременно или последовательно, в зависимости от ресурсов системы, что позволяет обеспечивать высокую масштабируемость приложения. Это становится возможным за счет ограничений в коммуникации между нитями, что составляет второй принцип архитектуры CUDA - разделениe памяти по уровням доступа.

Каждая нить имеет собственный, доступный только ей небольшой участок памяти, называемый локальной памятью (local memory). Нити внутри одного блока могут сообщаться друг с другом с помощью разделяемой памяти (shared memory) с низкой задержкой. Более объемная общая память (global memory) имеет высокую задержку и является доступной из CPU, она служит единственным каналом сообщения между CPU и GPU.

Кроме того, существует ещё два дополнительных вида памяти, доступных для всех нитей только на чтение. Это константная и текстурная память, оптимизированные для некоторых специфических форматов данных.

Механизм взаимодействия нитей друг с другом, барьерная синхронизация, в CUDA реализован при помощи встроенной функции \_syncthreads() [9]. Ей обозначаются точки синхронизации в ядре. \_syncthreads() запрещает нитям выполнение дальнейших команд до тех пор, пока все нити ядра не вызовут эту функцию.

1. **Описание реализации параллельного алгоритма.**

В разработанном нами алгоритме обучения карты Кохонена используется одновременно два принципа параллелизации: как на уровне нейрона, так и мелкомодульная параллелизация на уровне весов. В классическом алгоритме обучения самоорганизующихся карт мы выделяем несколько потенциально параллелизуемых подэтапов. Они характеризуются большим количеством независящих друг от друга простых вычислений. Для нашей реализации было выделено три подобных подэтапа:

1. Расчет расстояния между входящим вектором и нейронами сети;
2. Расчет расстояния на сетке между нейроном-победителем и остальными нейронами сети, для определения области корректировки;
3. Корректировка отдельных весов нейронов, входящих в окрестность.

На блок-схеме программы на рисунке 3 можно увидеть отображение выделенных подэтапов на архитектуру программы как принцип организации ядер CUDA.

 

*Рис.3. Блок-схема программы.*

Безусловно, можно выделить ещё несколько подэтапов с подобными характеристиками. Однако их параллелизация не дает ощутимого прироста скорости. Примером может служить инициализация векторов синаптических весов нейронов, которая осуществляется один раз за все время работы сети. Её параллельная реализация дает пренебрежимо малый прирост быстродействия.

 Работа ядер *fv\_distance\_kernel*, *xy\_distance\_kernel* и *nodes\_update\_kernel* осуществляется по единому принципу. Каждой нити выделяется либо один нейрон, либо, в случае ядра *fv\_distance\_kernel*, отдельный вес нейрона. Затем по заданной формуле на каждой нити вычисляется результат для текущего элемента и полученное значение записывается в соответствующий массив данных для последующей обработки.

 Этап корректировки является потенциально параллелизуемым на уровне весов, поскольку для каждого веса корректировка рассчитывается независимо. Теоретически это должно ускорить работу алгоритма. Однако в нашем случае каждой нити выделяется не вес, а весь нейрон целиком, с последовательной коррекцией весов каждого нейрона на одной нити. Это сделано во избежание дивергенции потока управления - простоя большого количества нитей, созданных перед условным выбором [10].

Четвертое ядро *bmu\_search\_kernel* отличается от других ядер тем, что оно представляет собой не параллелизацию подэтапа алгоритма, а отдельно разработанную функцию.

Редукция массива реализуется следующим образом – для массива , проводится попарное сравнение элементов , для . Если , *i*-ый заменяется элементом . Далее процедура повторяется для подмассива . В случае, если *n* нечетно, элемент добавляется в . Этот процесс повторяется до тех пор, пока подмассив не уменьшится до единичной длины. Единственный элемент результирующего подмассива и является нейроном-победителем, его координаты передаются в ядро xy\_distance\_kernel для поиска нейронов, входящих в окрестность победителя.

1. **Тестовые результаты.**

Программная CUDA-реализация нашего алгоритма была запущена на нескольких тестовых примерах. Чтобы оценить прирост производительности, на аналогичных тестах также запускалась последовательная версия алгоритма обучения, написанная на языке Си.

В экспериментах использовался ПК со следующими техническими характеристиками: процессор Intel Core i7 920 (4 ядра с тактовой частотой 2.67GHz), на операционной системе Windows 7 64-bit. В качестве графического процессора использовалась вышеописанная видеокарта NVIDIA GeForce GTX 680. Компиляция параллельной и последовательной версий осуществлялась в Microsoft Visual Studio 2010, для параллельной версии использовалась надстройка NVIDIA Parallel Nsight v.2.1 и программный пакет NVIDIA CUDA Toolkit v4.2.

Тесты проводились на картах различной размерности: 25х25, 50х50 и 100х100 нейронов. Массив обучающих векторов для каждой карты состоял из 60000 векторов размерности 100. Обучающая выборка генерировалась случайным образом по нормальному распределению со значениями в интервале от нуля до единицы не включительно. Для каждой конфигурации сети производилось 50 последовательных запусков CPU и CUDA алгоритмов.



*Рисунок 5. Результаты работы алгоритмов на различных конфигурациях сети.*

Средний прирост скорости обучения относительно последовательной версии алгоритма для карты размера 25х25 нейронов составил 12,39 раз, для карты размера 50х50 нейронов 24,27 раз, а для карты размера 100х100 нейронов время работы параллельного алгоритма превышает время работы последовательного уже в 55,78 раз.

Такой рост эффективности параллельного алгоритма на картах большой размерности связан с тем, что с увеличением количества нейронов его последовательная часть, включающая в себя инициализацию весов и расчет параметров сети на каждой итерации, остается практически неизменной. А для версии алгоритма, выполняемой на CPU, количество осуществляемых последовательных вычислений растет пропорционально размерности карты. Как следствие, при фиксированной выборке и размерности входящих векторов время выполнения последовательного алгоритма напрямую зависит от количества нейронов. С ростом нейронов в 4 раза, приблизительно в 4 раза увеличивается и время выполнения последовательного алгоритма. Для карты размером 500х500 нейронов расчетное время одного запуска будет занимать приблизительно 28 часов, что с учетом необходимости проводить подбор параметров сети, делает проведение экспериментов с данными такого масштаба практически невозможным.

1. **Заключение.**

В статье нами была описана эффективная и доступная методика ускорения большого объема вычислений, который неизбежно возникает в процессе решения реальных задач с применением самоорганизующихся карт Кохонена.

Технология CUDA применима не только для ускорения работы SOM, но и для других моделей ИНС, в которых имеются потенциально параллелизуемые структуры [11].

Кроме того, поскольку основной код алгоритма написан на языке C с добавлением некоторых функций из библиотеки CUDA, становится возможной его адаптация для работы с недавно анонсированной компанией Microsoft технологией параллельного программирования C++ AMP [12], что даёт новые возможности для дальнейших исследований в этом направлении.

Литература.

1. **Хайкин С. Нейронные сети: полный курс, 2-е издание. : Пер. с англ. – М.: Издательский дом «Вильямс», 2008**
2. **Kohonen T. “The self-organizing map”, Proceedings of the Institute of Electrical and Electronics, 1990, vol. 78, p. 1464 – 1480**
3. Nordstrom T. Designing parallel computers for self-organizing maps. Forth Swedish Workshop on Computer System Architecture, Linkoping, 1992. [Труды конференции]
4. D. Valencia, A. Plaza, P. Martinez and J. Plaza. Parallel Processing of High-Dimensional Remote Sensing Images Using Cluster Computer Architectures // International Journal of Computers and their Applications, USA, NC, Cary, March 2007, vol. 14, no. 1, pp. 23-34
5. **Калитин Д. Использование технологии CUDA фирмы NVIDIA для САПР нейронных сетей // Электронное научное издание «Устойчивое инновационное развитие: проектирование и управление», том 4, 1999. URL:** [www.rypravlenie.ru](http://www.rypravlenie.ru) **(Дата обращения 21.05.2012)**
6. **Официальный сайт компании NVIDIA. Раздел** Продукты с поддержкой NVIDIA CUDA**//** <http://www.nvidia.ru/object/cuda_gpus_ru.html> **(Дата обращения 21.05.2012)**
7. **Официальный сайт компании NVIDIA. Технические характеристики графической карты NVIDIA GeForce GTX 680 //** <http://www.nvidia.ru/object/geforce-gtx-680-ru.html#pdpContent=2>
8. Руководство по программированию на CUDA C вер. 4.1 // CUDA C Programming Guide v.4.1. // <http://developer.download.nvidia.com/compute/DevZone/docs/html/C/doc/CUDA_C_Programming_Guide.pdf>
9. **CUDA API Reference Manual v.4.2 //** <http://developer.download.nvidia.com/compute/DevZone/docs/html/C/doc/CUDA_Toolkit_Reference_Manual.pdf>
10. Сандерс Дж., Кэндрот Э. Технология CUDA в примерах: введение в программирование графических процессоров: Пер. с англ. – М.: ДМК Пресс, 2011.
11. Чанкин А.А. Реализация алгоритма прогнозирования временных рядов при помощи интервальной нейронной сети. Сб. трудов XIV Всероссийской научно- технической конференции «Новые информационные технологии». 2011 – М.: МГУПИ. 2011, сс.64-73.
12. Анонс о выходе С++ AMP в официальном блоге компании Microsoft // http://blogs.msdn.com/b/vcblog/archive/2011/06/15/introducing-amp.aspx