УДК 004.383.8.032.26; 004.855.5

РЕАЛИЗАЦИЯ ОСЦИЛЛЯТОРНОЙ ХАОТИЧЕСКОЙ НЕЙРОННОЙ СЕТИ С ПРИМЕНЕНИЕМ ТЕХНОЛОГИИ NVIDIA CUDA ДЛЯ РЕШЕНИЯ ЗАДАЧ КЛАСТЕРИЗАЦИИ

Е. Н. Бендерская[a]***,*** *канд. техн. наук, доцент*
А. А. Толстов[a]***,*** *аспирант*
[a]*Санкт-Петербургский политехнический университет, Санкт-Петербург, РФ*

Постановка проблемы: *возрастающая актуальность решения задач в области искусственного интеллекта, а именно задач кластеризации, приводит к необходимости разработки производительной аппаратной реализации осцилляторных нейронных сетей, являющихся одним из перспективных методов в области кластерного анализа. Целью работы является упрощение процесса применения осцилляторных хаотических нейронных сетей за счет их аппаратной реализации, для чего требуется разработка набора подходов, алгоритмов и структурных решений.* ***Методы:*** *анализ особенностей осцилляторных нейронных сетей, определение основных этапов в решении задачи кластеризации на основе осцилляторных нейронных сетей, анализ альтернативных вариантов организации вычислительного процесса.* ***Результаты:*** *разработаны алгоритмы вычисления выходных значений нейронов с различными паттернами доступа к памяти в зависимости от размера сети и доступной мощности GPU. Анализ результатов тестирования показал, что методы X- и Y-потоков целесообразно использовать для сетей размером, не превышающим половину числа максимально возможных одновременно выполняющихся потоков для видеокарты, чтобы обеспечить выигрыш во времени выполнения вычислений. Предложены варианты распределения памяти для хранения матрицы синхронизаций с учетом размера сети на основе буферизации, а также варианты анализа результатов синхронизации между нейронами на основе неориентированных графов и системы непересекающихся множеств. На основе предложенных решений разработана эффективная реализация сети с использованием архитектуры CUDA, учитывающая особенности сети.* ***Практическая значимость:*** *результаты исследований и алгоритмические решения могут быть использованы при разработке аппаратных средств реализации осцилляторной хаотической нейронной сети. Они позволяют получить аппаратное решение, адекватное особенностям функционирования и применения хаотической нейронной сети в задачах кластеризации.*

Ключевые слова *— аппаратная реализация, графические процессоры, нейрокомпьютер, кластеризация, осцилляторная хаотическая нейронная сеть.*

Введение

Задача кластеризации является универсальной, так как используется при решении практических задач для многих предметных областей. Одним из перспективных средств ее решения, не использующих априорные знания о числе кластеров, является осцилляторная хаотическая нейронная сеть (ОХНС). Модель сети была разработана группой итальянских ученых под руководством Л. Ангелини [1] в 2000 г. Можно выделить следующие основные характеристики ОХНС:

1) сеть является однослойной, рекуррентной и полносвязной;

2) элементами сети являются нейроны с передаточной функцией «логистическое отображение»;

3) сеть обладает свойством неаттракторности — результат работы скрыт в динамике выходов нейронов;

4) для извлечения результата работы сети требуется анализ изменения выходов нейронов во времени: нейроны, демонстрирующие похожие колебания, относятся к одному кластеру.

Вследствие ресурсоемкости вычислений при решении задачи кластеризации как при использовании классических методов, так и при помощи ОХНС необходимо предложить эффективную аппаратную реализацию ОХНС. Однако здесь существует ряд трудностей.

1. Нейронная сеть имеет большую вычислительную сложность из-за значительного объема выполняемых операций. Вычисление выходов происходит в течение T итераций, после которых производится анализ значений выходов нейронов на всех T итерациях. Вследствие полносвязности сети алгоритмическая сложность вычислений пропорциональна числу нейронов в квадрате, что влечет за собой значительные временные затраты на моделирование работы сетей большого размера.

2. Для хранения значений выходов нейронов требуется большой объем памяти. При увеличении числа нейронов до десятков и сотен тысяч хранение даже матрицы весовых коэффициентов представляет сложность для современных ПК.

В статье рассматриваются основные принципы работы ОХНС, алгоритмы ее функционирования, проблемы, возникающие при организации вычислительного процесса для реализации ОХНС, и методы ускорения работы при аппаратной реализации с применением технологии CUDA.

Особенности технологии NVIDIA CUDA и постановка задачи

Технологии вычислений общего назначения на графических процессорах (GPGPU) широко распространены в наши дни, позволяя при помощи высокопроизводительных видеокарт эффективно решать поддающиеся распараллеливанию вычислительные задачи.

Унифицированная архитектура вычислительного устройства (*Compute Unified Device Architecture* — CUDA) — технология GPGPU, предложенная для программирования на видеокартах, производимых компанией NVidia. Графические вычислительные устройства (GPU), произведенные NVidia, являются наиболее распространенными и широко применяющимися среди аналогичных устройств в сфере GPGPU.

Устройства с поддержкой CUDA обладают следующими особенностями.

1. Наличие большого числа одновременно выполняющихся потоков (threads). Такие потоки не являются производительными, но их число доходит до десятков тысяч. Следовательно, оптимальный режим использования ресурсов видеокарты достигается в случае одновременной работы максимального числа потоков.

2. Видеокарта разделена на мультипроцессоры. Потоки, запускаемые внутри одного мультипроцессора, используют принадлежащие ему ресурсы: L1-кэш, разделяемую (shared) память, регистры.

3. Потоки физически одновременно исполняются варпами (warp) по 32 потока на варп. Несколько варпов объединяются в блок (block), получая доступ к общей для блока разделяемой памяти, характеризующейся высокой скоростью работы в сравнении с глобальной памятью видеокарты.

Типичными задачами, эффективно решающимися с применением CUDA, являются всевозможные фильтры обработки изображений, задачи линейной алгебры, параллельные алгоритмы сортировки, алгоритмы быстрого преобразования Фурье и моделирования молекулярной динамики.

Ввиду вычислительных сложностей появляется естественное требование параллелизма при реализации ОХНС, как и для всех нейронных сетей. Структура ОХНС и алгоритмы анализа результатов являются адекватными для параллельного исполнения [2, 3], что при правильном подходе может обеспечить значительное ускорение вычислительного процесса.

Нейронные сети, близкие ОХНС по структуре, например сеть Хопфилда, также дружественны параллельному исполнению с применением CUDA [3, 4]. Причина этому — независимость вычисления выходов нейронов одного слоя сети. Несмотря на то, что большая часть проблем как программной, так и аппаратной реализации ОХНС схожа с теми же проблемами для других осцилляторных нейронных сетей, например, как для нейронной сети Хопфилда, тем не менее для ОХНС каждая из проблем имеет свои особенности ввиду отличительных черт и самой ОХНС [2]. Так, число итераций по вычислению выходов нейронов из-за хаотического аттрактора значительно больше, чем для осцилляторных нейронных сетей с аттракторами типа точка или замкнутый цикл, поэтому означенная во введении первая трудность аппаратной реализации имеет совсем другие временные характеристики — значительно большие. Специфика применения ОХНС для решения задач кластеризации также предъявляет большие требования и к объему памяти для хранения выходов, помимо матрицы весовых коэффициентов, вследствие необходимости последующей постобработки выходной динамики сети. Алгоритм реализации ОХНС также включает в себя и анализ выходов нейронов во времени, который является задачей параллельного исполнения, что будет показано далее.

В качестве решения части этих проблем предлагается использовать современные графические ускорители производства компании NVidia, поддерживающие технологию CUDA. Для того чтобы предложить эффективную реализацию ОХНС на выбранной аппаратной платформе, необходимо проанализировать различные варианты организации вычислительного процесса, учитывающие особенности алгоритмов функционирования ОХНС и различных стадий работы ОХНС, а также возможности, предоставляемые рассматриваемой аппаратной базой.

В данной работе будет использована ориентация на оптимизацию для архитектуры NVidia Fermi и выше [5], более старые версии не рассматриваются.

Обобщенный алгоритм реализации ОХНС

Алгоритм работы ОХНС состоит из нескольких стадий. В качестве входных данных выступает множество подлежащих кластеризации точек, в простейшем случае двумерных. Создается полносвязная однослойная нейронная сеть, состоящая из N нейронов, каждый из которых соответствует одной из точек исходного множества. Весовые коэффициенты сети определяются по формуле [1]

$$J_{ij} = \exp\left(-\left[r_i - r_j\right]^2 / 2a^2\right), \qquad (1)$$

где $[r_i - r_j]$ — евклидово расстояние между нейронами; a — масштабирующая константа. Уравнение

выходов сети во времени определяется соотношением

$$x_i(t+1)=\frac{1}{C_i}\sum_{j=1}^{N}J_{ij}f\left(x_j(t)\right), \quad (2)$$

где передаточная функция нейрона f и нормирующий коэффициент C_i определяются соотношениями

$$f(x)=1-2x^2; \quad (3)$$

$$C_i=\sum_{j=1}^{N}J_{ij}. \quad (4)$$

На основе топологической информации о входных данных, получаемой в результате выполнения триангуляции Делоне, формируются весовые коэффициенты обратных связей между нейронами по формуле (1). Происходит инициализация нейронов начальными значениями в виде случайных чисел в диапазоне от –1 до 1. Затем наступает фаза вычисления очередных значений нейронной сети на протяжении T итераций работы согласно формуле (2). Выходы каждой итерации должны быть обработаны и учтены алгоритмом поиска синхронизации между нейронами.

Общий алгоритм реализации ОХНС изображен на рис. 1.

Представление данных ОХНС:

1. Матрица весовых коэффициентов размера $N\times N$, состоящая из вещественных чисел в диапазоне от 0 до 1. Заполняется единожды и используется на протяжении вычисления выходов нейронной сети.

2. Вектор размера N, хранящий значения выходов нейронов предыдущей итерации — вещественные числа в диапазоне [–1 1].

3. Вектор размера N, в который будут записаны текущие значения выходов нейронов — вещественные числа в диапазоне [–1 1]. После очередной итерации содержимое текущего вектора переходит в вектор из п. 2, а текущий вектор вычисляется заново.

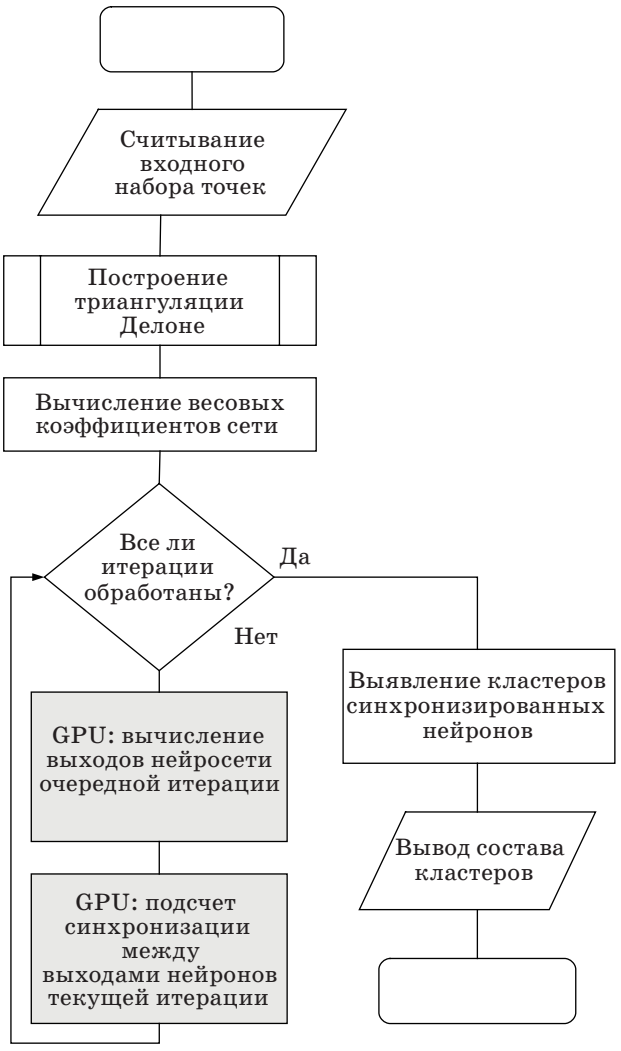
4. Матрица синхронизации размера $N\times N$, хранящая неотрицательные целочисленные значения. Значение в строке i и столбце j обозначает количество итераций, в которых значения нейронов i и j были синхронизированы согласно используемому типу синхронизации. Заполняется в течение работы алгоритма.

Занесение входных данных в разработанный программный продукт происходит путем чтения файла в формате, совместимом с файлами базы FCPS [6].

Следующим этапом является построение триангуляции Делоне входного набора для получения масштабирующей константы [7, 8]. Триангуляция может быть построена при помощи алгоритма Форчуна, реализованного в рамках исследовательской задачи, либо алгоритма quickhull, поставляемого в составе библиотеки qhull. Оба эти алгоритма имеют вычислительную сложность O(Nlog$N$), однако алгоритм quickhull является более универсальным, так как позволяет проводить вычисления для многомерных данных, тогда как алгоритм Форчуна ограничен двумерными данными. Оба алгоритма являются последовательными и выполняются на CPU.

После получения триангуляции происходит процесс вычисления масштабирующей константы — для каждого нейрона рассматриваются его соседи по триангуляции и усредняется расстояние до них. Среднее среди расстояний всех нейронов будет являться масштабирующей константой.

Затем по формуле (1) устанавливаются весовые коэффициенты $\mathbf{J}_{ij}$ с учетом расстояния между нейронами i, j и масштабирующей константой.



■ ***Рис. 1.*** Обобщенная схема алгоритма работы ОХНС

Для хранения весовых коэффициентов достаточно 32-битного вещественного числа с плавающей точкой (одинарная точность). Матрица весовых коэффициентов будет использоваться при вычислениях на GPU, поэтому ее необходимо скопировать в память GPU. Заметим, что количество весовых коэффициентов равняется N^2. В целях оптимизации объема используемой памяти хранить коэффициенты не обязательно — их можно пересчитывать в процессе вычисления. Смысл такой оптимизации может возникнуть в случае работы с большими нейронными сетями (от 20 000 нейронов), когда матрица занимает гигабайты данных (отметим, что объем памяти современных графических ускорителей ограничен 3–12 ГБ).

Особенность структуры ОХНС такова, что вся информация о входных данных выражается в числе нейронов и весовых коэффициентах между ними, т. е. на вход нейронов не подается какой-то особый сигнал. Однако входы нейронов должны быть инициализированы случайными значениями в диапазоне от –1 до 1. Генерацию таких значений ввиду их небольшого объема и простоты операции удобно провести традиционными средствами на CPU, а массив полученных значений скопировать в память GPU.

Итак, можно выделить две основные составляющие вычислительного процесса для ОХНС — вычисление выходных значений нейронов и определение синхронных нейронов и затем на их основе определение сформировавшихся в процессе самоорганизации кластеров. Рассмотрим варианты реализации каждой из составляющих.

Алгоритмы вычисления значений на выходах нейронов

Реализованная для исполнения на GPU процедура подсчета выходов вызывается на каждой итерации, принимая на вход значения выходов предыдущей итерации.

Операция вычисления значений нейронов есть результат умножения вектора предыдущих значений на матрицу весовых коэффициентов и умножение полученного значения на коэффициент нормирования [см. формулу (2)]. В случае ОХНС матрица весовых коэффициентов является квадратной и симметричной. Данные свойства можно использовать для повышения эффективности работы алгоритма.

Ключевыми компонентами реализации такой процедуры являются эффективное обращение к памяти, в которой хранятся весовые коэффициенты; максимально возможное повторное использование памяти, хранимой вектором предыдущих значений, и применение механизма shared-памяти для разделения работы внутри блоков [9].

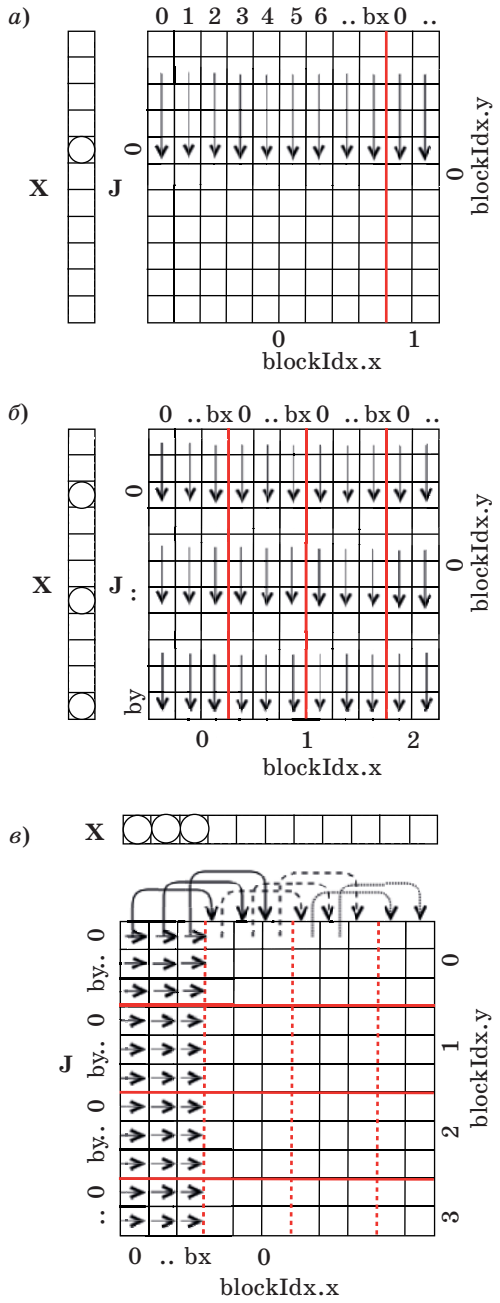
В рамках задачи вычисления значений нейронов было разработано несколько решений, рассмотрим наиболее эффективные.

1. Поток на нейрон (thread-per-neuron). Использование буфера из shared-памяти для вектора. Один поток отвечает за выход одного нейрона и двигается по столбцу матрицы вниз в течение своей работы. Потоки группируются для считывания значений матрицы и читают один и тот же элемент вектора. Недостаток подхода: неэффективно используется ресурс GPU на производительных видеокартах и в случае нейронных сетей малого размера. Плюс подхода: отсутствие лишних операций, низкий overhead, высокая производительность в случае соответствия размера матрицы объему вычислительных мощностей.

2. Несколько Y-потоков на нейрон с максимальной утилизацией GPU и атомарными операциями по shared-памяти. Решает проблему «простаивания» GPU. Являет собой аналог метода thread-per-neuron, но отличается тем, что за один выход нейрона отвечают несколько потоков из разных варпов, двигающихся с определенной дистанцией по столбцу матрицы вниз. После завершения работы каждого из потоков они суммируют свои значения при помощи атомарного сложения на разделяемой памяти, а затем главный поток записывает значение в глобальную память.

3. Несколько X-потоков на нейрон внутри блока. Отличие от метода с Y-потоками заключается в том, что один нейрон просчитывает потоки из одной строки, идущие последовательно (такие группы потоков могут быть варпами либо полуварпами). В одном блоке располагается несколько групп таких потоков. Дополнительно используется буфер из shared-памяти для чтения данных из вектора значений предыдущей итерации. Атомарные сложения в данном методе не выполняются из-за высоких накладных расходов, обусловленных тем, что внутри варпа их проведение неэффективно из-за конфликтов банков shared-памяти. Вместо этого применяется суммирование значений в цикле за логарифмическое число операций [5].

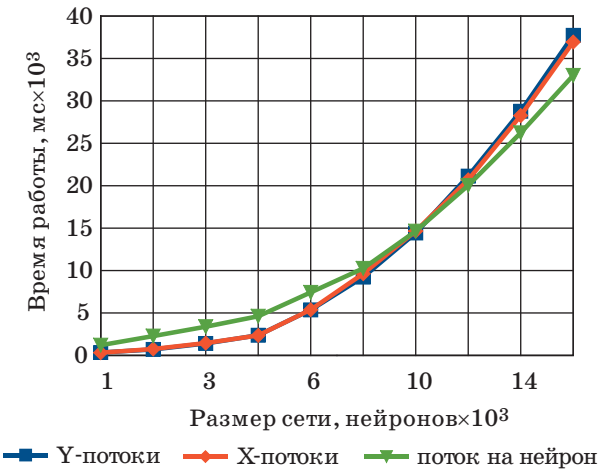
Графические схемы алгоритмов, демонстрирующие паттерн доступа к памяти, представлены на рис. 2, *а–в*, где потоки исполнения (threads) показаны стрелками. Стрелки указывают на элемент памяти матрицы **J**, считываемый в текущий момент времени, и показывают направление перехода к следующему элементу. Кружки в ячейках вектора **X** показывают, что значения соответствующих ячеек считываются в текущий момент. Линии, выделенные полужирным, разграничивают зоны работы блоков, полужирным пунктирным отделяют текущий регион обработки блоком. Пунктирные стрелки (см. рис. 2, *в*) подсказывают переход потоков к следующим ячейкам в матрице.



■ ***Рис. 2.*** Схемы паттернов доступа к памяти для алгоритмов вычисления значений нейронов: *а* — поток на нейрон; *б* — Y-потоки; *в* — X-потоки

С целью сравнить методы было проведено тестирование для сетей, соответствующих случайно сгенерированным входным множествам. Координаты x и y точек множества лежат в диапазоне [–100..100]. В силу того, что конкретные значения в исходных данных для каждой отдельной задачи не влияют на количество выполняемых операций, необходимости накапливать статистическую информацию по серии запусков одной и той же задачи нет. Даже задание начальных условий случайным образом не влияет на конечный результат работы сети в силу образования уникальных динамик, соответствующих каждому из кластеров [7]. Результаты тестирования приведены в табл. 1.

Проведенное тестирование методов позволяет сделать вывод о применимости каждого из методов в зависимости от размера нейронных сетей и доступной мощности GPU. Поскольку имеется два параметра — число одновременно запускаемых потоков на видеокарте и размер нейросети, то предпочтительно использовать метод X- или Y-потоков для сетей размером, не превышающим половину числа максимально возможных одновременно выполняющихся потоков для видеокарты. Как можно оценить по данным на рис. 3,



■ ***Рис. 3.*** Зависимость скорости вычисления значений нейронов от выбранного метода

■ ***Таблица 1.*** Время работы алгоритма вычисления выходов нейронов в зависимости от метода и размера сети на GPU NVidia GeForce GTX580

Метод	Время работы 5000 итераций для сети из указанного числа нейронов, мс						
	1000	2000	3000	4000	8000	12 000	16 000
Y-потоки	342	700	1408	2372	9229	21 127	37 722
X-потоки	322	755	1454	2354	9631	20 692	36 937
Поток на нейрон	1201	2273	3381	4611	10 254	20 036	33 003

методы X- и Y-потоков являются адекватно применимыми и для больших сетей, проигрывая не более 10 % методу «поток на нейрон». Тенденция характерна также и для других GPU — лишь положение точки пересечения кривых скорости работы будет зависеть от вычислительной мощности: чем больше потоков можно запустить одновременно, тем при большем размере сети будет наблюдаться паритет между алгоритмами.

Правила определения синхронизации между нейронами и организация вычислительного процесса для анализа результатов

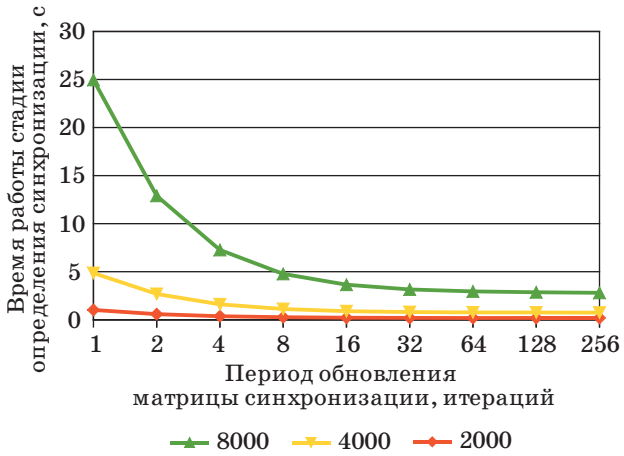
Когда вычисление новых значений нейронов завершено, проводится промежуточный этап анализа — учитывается число совпадений значений нейронов согласно фрагментарной синхронизации с определенным значением невязки ε либо выполняется фазовая синхронизация, определяющая, увеличилось ли значение нейрона на текущей итерации по сравнению с предыдущей. Результаты промежуточного анализа аккумулируются в целочисленной матрице синхронизаций размером $N \times N$. Фаза анализа выполняется за квадратичное время.

Решение выполнять фазу анализа после каждой итерации выглядит логично с точки зрения минимизации затрат памяти. Но в данном случае имеется трудность для реализаций, чувствительных к записи в память. Потенциально на каждом этапе может изменяться порядка N^2 ячеек глобальной памяти CUDA.

В качестве оптимизирующего решения обновление матрицы синхронизаций следует производить один раз за несколько итераций, снижая частоту записи в глобальную память (число операций сравнения остается прежним). Следовательно, необходимо хранить дополнительный буфер из предыдущих значений нейронов; размер такого буфера кратен числу итераций, по которым выполняется одновременный анализ синхронизации.

Для определения размера буфера, близкого к оптимальному, были протестированы нейронные сети размером 2000, 4000 и 8000 нейронов, сгенерированные на основе случайных исходных множеств. Для каждой нейронной сети выполнялись 1000 итераций ее полной обработки и измерялось суммарное время работы процедуры, ответственной за подсчет синхронизаций. Для теста использовалось GPU NVidia CUDA GTX580.

Из графиков, представленных на рис. 4, становится очевидной зависимость времени работы от периода между записями. С увеличением периода сначала происходит резкий рост скорости, но потом достигается насыщение, связанное с установлением баланса времени, затраченного



■ ***Рис. 4.*** Зависимость времени работы стадии определения синхронизации от периода обновления матрицы синхронизаций

на операции сравнения и операции записи в глобальную память.

Основываясь на данных экспериментов, можно воспользоваться константой 64 в качестве стандартного параметра для размера буфера выходов ОХНС. С ростом числа нейронов в сети характер зависимости остается практически неизменным.

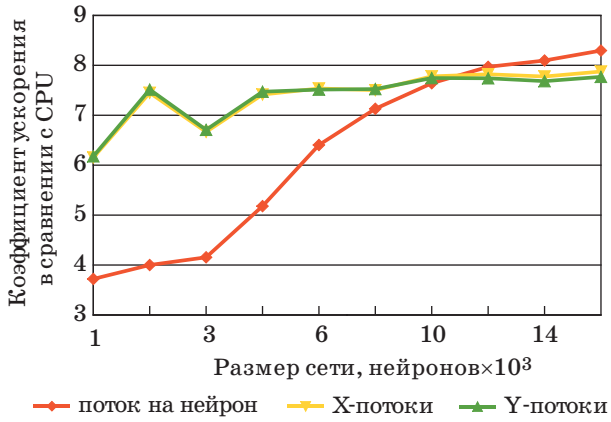
После выполнения всех T итераций аккумулированные значения синхронизации копируются в оперативную память, и за этим копированием следует последняя фаза вычислений — в зависимости от заданного порогового коэффициента два нейрона либо могут быть признаны синхронизированными, либо нет. Такая информация вычисляется для каждой пары нейронов, и на ее основе строится неориентированный граф, вершинами которого являются нейроны. Ребро единичного веса соединяет два нейрона, если они признаны синхронизированными. Компоненты связности полученного графа есть кластеры, на которые разбивается входное множество. Получение компонент связности при обходе графа в ширину либо в глубину может иметь вычислительную сложность $O(N + M)$, где M — число пар синхронизированных нейронов. Второй вариант получения компонент связности — использование структуры данных «система непересекающихся множеств»; в этом случае сложность алгоритма составит $O(N^2 + N\log N)$. Результат данной стадии есть результат работы нейронной сети.

Результаты тестирования эффективности реализации ОХНС на CUDA

Для тестирования эффективности был использован тестовый стенд с характеристиками: процессор Intel Core i7 920 @2.67 ГГц, оперативная

■ ***Таблица 2.*** Время работы высокопроизводительных алгоритмов в зависимости от размера ОХНС

Алгоритм	Время работы в зависимости от числа нейронов, с					
	1000	2000	4000	10 000	14 000	16 000
CPU	1,6	2,6	7,8	47,6	92,7	121,2
Поток на нейрон	0,43	0,65	1,51	6,23	11,45	14,61
X-потоки	0,26	0,35	1,05	6,12	11,92	15,39
Y-потоки	0,26	0,35	1,04	6,15	12,07	15,61



■ ***Рис. 5.*** Ускорение реализации ОХНС на CUDA по сравнению с CPU (8 потоков исполнения)

память объемом 12 ГБ, видеокарта NVidia GeForce GTX 580 с объемом памяти 3 ГБ, ОС Windows 7 Professional. Проводилось сравнение трех вышеописанных алгоритмов вычислений, реализованных на CUDA, с высокопроизводительной реализацией для центрального процессора, использующей многопоточность (процессор поддерживает 8 потоков исполнения) и автоматическую векторизацию. Используемые компиляторы: CUDA Toolkit 5.0, Microsoft Visual C++ 2008 для CUDA-реализации и Intel Composer XE 2013 для CPU. Тестирование проводилось для нейросетей размером от 1000 до 16 000 нейронов, сгенерированных на основе случайных данных.

Результаты тестирования демонстрируют эффективность применения CUDA для работы ОХНС на сетях от 1000 элементов. Сравнение показывает преимущество реализованной на CUDA модели в 6–8 раз. Сравнительное время работы алгоритмов для подмножества размеров сетей, отображенных на рис. 5, представлено в табл. 2.

Для теста использовались GPU NVidia CUDA GTX580 и Intel Core i7 920. Замеры проводились на 1000 итерациях работы.

Заключение

Реализация ОХНС на CUDA, выполненная в рамках исследования аппаратной поддержки осцилляторных хаотических сетей, подтверждает эффективность использования GPU для ускорения вычислений и позволяет более быстро обрабатывать ОХНС по сравнению с традиционным способом запуска на CPU. Эффективность предложенного способа организации вычислительного процесса подтверждена экспериментальным сравнением времени работы сети на CUDA с эффективной CPU-реализацией. Путем модификации алгоритмов вычисления нейросети удалось получить как выигрыш во времени обработки сети, так и экономию объемов занимаемой памяти, предложив эффективную реализацию с использованием технологии NVidia CUDA. Были достигнуты показатели ускорения обработки в 68 раз для устройств одного поколения по сравнению с оптимизированной реализацией для центрального процессора. Исходный код выполненной разработки находится в свободном доступе в сети Интернет по адресу http://github.com/alex-tolstov/OCNN/.

Дальнейшее совершенствование алгоритмов ОХНС может быть выполнено из соображений качественного уменьшения времени работы за счет включения в весовые коэффициенты сети новой метрики, извлекаемой из входных данных, что должно позволить получить более простые типы синхронизации нейронов, для идентификации которых требовалось бы меньше итераций. Другое направление деятельности — аппаратная реализация хаотических структур в реакционно-диффузионных средах.

Литература

1. **Angelini L.** et al. Clustering Data by Inhomogeneous Chaotic Map Lattices // Physical Review Letters. 2000. N 85. P. 78–102. doi:1103/PhysRevLett.85.554

2. **Бендерская Е. Н., Толстов А. А.** Тенденции развития средств аппаратной поддержки нейровычислений // Научно-технические ведомости СПбГПУ. Информатика. Телекоммуникации. Управление. 2013. № 3(174). С. 9–18.

3. **Бендерская Е. Н.** Проблемы и перспективы параллельных вычислений на базе нелинейных динамических элементов // Суперкомпьютеры. 2013. № 1(13). С. 40–43.
4. **Liang L.** Parallel Implementation of Hopfield Neural Networks on GPU. Oct. 2011. http://dumas.ccsd.cnrs.fr/dumas-00636458 (дата обращения: 20.01.2014).
5. **NVidia** C Best Practices Guide. Design Guide. DG-05603-001_v5.5. July 2013. http://docs.nvidia.com/cuda/pdf/CUDA_C_Best_Practices_Guide.pdf (дата обращения: 05.04.2014).
6. **Ultsch A.** Clustering with SOM: U*C // Proc. Workshop on Self-Organizing Maps, Paris, France, 2005. P. 75–82.
7. **Бендерская Е. Н., Жукова С. В.** Осцилляторные нейронные сети с хаотической динамикой в задачах кластерного анализа // Нейрокомпьютеры: разработка, применение. 2011. № 7. С. 74–86.
Benderskaya E. N., Zhukova S. V. Nonlinear Approaches to Automatic Elicitation of Distributed Oscillatory Clusters in Adaptive Self-Organized System // Distributed Computing and Artificial Intelligence (DCAI — 2012): 9th Intern. Conf., Salamanca (Spain), Mar. 28–30, 2012, Advances in Intelligent and Soft Computing (AISC). Springer-Verlag Berlin Heidelberg, 2012. Vol. 151. P. 733–741. doi:10.1007/978-3-642-28765-7_88
8. **Sørensen H. H. B.** High-Performance Matrix-Vector Multiplication on the GPU // Proc. of Euro-Par 2011 Workshops. Part I: Lecture Notes in Computer Science (LNCS). Springer-Verlag Berlin Heidelberg, 2012. Vol. 7155. P. 377–386. doi:10.1007/978-3-642-29737-3_42

UDC 004.383.8.032.26; 004.855.5

Hardware Implementation of a Chaotic Oscillatory Neural Network by NVidia CUDA Technology

Benderskaya E. N.[a], PhD, Associate Professor, helen.bend@gmail.com
Tolstov A. A.[a], Post-Graduate Student, gm.alext@gmail.com
[a] Saint-Petersburg State Polytechnical University, 21, Politechnicheskaya St., 194021, Saint-Petersburg, Russian Federation

Purpose: The paper deals with the problem of hardware implementation of chaotic oscillatory neural networks used for clusterization tasks. The goal of this paper is to simplify the process of using oscillatory chaotic neural networks in practical applications through a number of approaches, algorithms and structural solutions for their hardware implementation. **Methods:** Analyzing features of oscillatory neural networks, defining the basic stages in the clusterization task on the basis of oscillatory neural networks, analyzing alternative variants of the computing process organization. **Results:** Algorithmic solutions for hardware implementation of chaotic oscillatory neural networks were developed and discussed. Algorithms for finding the output values of neurons with different patterns of memory access and with different network size were developed. The results of the tests showed that the methods of X-flow and Y-flow streams should be used in networks of a size not exceeding half the maximum possible number of concurrent videocard streams, as this approach provides the shortest computation time. For the storage of the synchronization matrix, some variants of memory allocation were proposed, taking into account the network size on basis of buffering. Several ways were discussed to analize the results of inter-neuron synchronization results, on basis of undirected graphs and systems of disjoint sets. An effective network implementation was developed, based on CUDA, taking into account the features of the network. **Practical relevance:** The research results and algorithmic solutions can be used in developing oscillatory chaotic neural network hardware, adequate for the features of chaotic neural network functioning, usage and application for problems of clustering.

Keywords — Hardware Implementation, Graphics Processors, Neurocomputer, Clustering, Oscillatory Chaotic Neural Network.

References

1. Angelini L., Carlo F., Marangi C., Pellicoro M., Nardullia M., Stramaglia S. Clustering Data by Inhomogeneous Chaotic Map Lattices. *Physical Review Letters*, 2000, no. 85, pp. 78–102. doi:1103/PhysRevLett.85.554
2. Benderskaya E. N., Tolstov A. A. Trends of Hardware Implementation of Neural Networks. *Nauchno-tekhnicheskie vedomosti SPbGPU. Informatika. Telekommunikatsii. Upravlenie*, 2013, no. 3(174), pp. 9–18 (In Russian).
3. Benderskaya E. N. Perspective and Problems of Parallel Computing Based on Non-Linear Dynamic Elements. *Superkomp'iutery*, 2013, no. 1(13), pp. 40–43 (In Russian).
4. Liang L. *Parallel Implementation of Hopfield Neural Networks on GPU*. October 2011. Available at: http://dumas.ccsd.cnrs.fr/dumas-00636458 (accessed 20 January 2014).
5. *NVidia C Best Practices Guide. Design Guide. DG-05603-001_v5.5*. July 2013. Available at: http://docs.nvidia.com/cuda/pdf/CUDA_C_Best_Practices_Guide.pdf (accessed 05 April 2014).
6. Ultsch A. Clustering with SOM: U*C. *Proc. Workshop on Self-Organizing Maps*. Paris, France, 2005, pp. 75–82.
7. Benderskaya E. N., Zhukova S. V. Oscillatory Neural Networks with Chaotic Dynamics for Cluster Analysis Problems. *Nejrokomp'jutery: razrabotka, primenenie*, 2011, no. 7, pp. 74–86 (In Russian).
8. Benderskaya E. N., Zhukova S. V. Nonlinear Approaches to Automatic Elicitation of Distributed Oscillatory Clusters in Adaptive Self-Organized System. *Distributed Computing and Artificial Intelligence (DCAI — 2012). 9th Intern. Conf., Salamanca (Spain), 2012, Advances in Intelligent and Soft Computing (AISC)*. Springer-Verlag Berlin Heidelberg, 2012, vol. 151, pp. 733–741. doi:10.1007/978-3-642-28765-7_88
9. Sørensen H. H. B. High-Performance Matrix-Vector Multiplication on the GPU. *Proc. of Euro-Par 2011 Workshops. Part I: Lecture Notes in Computer Science (LNCS)*. Springer-Verlag Berlin Heidelberg, 2012, vol. 7155, pp. 377–386. doi:10.1007/978-3-642-29737-3_42