Если выполнялась аутентификация, то на этом всё и заканчивается. Для идентификации необходимо повторить этот процесс для всех отпечатков в базе данных.

АРМ оператора построено на базе персонального компьютера. Оператор должен следить за ведением журнала и решать вопросы доступа для посторонних лиц с помощью установленного на компьютер специализированного программного обеспечения. АРМ связано с модулями распознавания посредством проводной локальной сети. Также АРМ оператора выполняет ведение учёта всех входящих и выходящих людей, хранение данных о доступе в журнале доступа, хранение базы отпечатков, редактирование базы отпечатков и сведений о лицах имеющих допуск.

На АРМ оператора установлен модуль регистрации, который служит для снятия отпечатка пальца регистрирующегося лица, выделения ключевых точек и последующей передачи на нужный модуль контроля доступа, который представляет собой совокупность модуля распознавания и привода замка. Процесс регистрации отображается в специальной программе-диспетчере, установленной на АРМ оператора. Отпечатки нового сотрудника вносятся в общую базу отпечатков. Данное специализированное программное обеспечение позволяет контролировать процесс доступа сотрудников в охраняемые помещения, вести учет рабочего времени.

В процессе задания доступа можно указать доступные точки прохода для определенного круга лиц. В специальный раздел базы данных системы заносится информация о посетителях организации, включая задание ограничений доступа.

Функции АРМ оператора – это ведение учёта всех входящих и выходящих людей, хранение данных о доступе в журнале доступа, мониторинг работоспособности системы, хранение базы отпечатков, редактирование базы отпечатков и сведений о лицах имеющих допуск, осуществление взаимодействия между составными частями системы.

Результаты выполненного программного моделирования и макетирования отдельных компонентов системы подтверждают правильность предложенных системотехнических решений.

Таким образом, разрабатываемая система в случае ее реализации позволит автоматизировать контроль доступа в помещения, снизить затраты на обеспечение безопасности, а также упростить ведение учета рабочего времени в учреждениях и на предприятиях.

УДК 004.896

В.Ф. Гузик, Ю.В. Чернухин, М.Н. Десятерик

ВОЗМОЖНОСТИ ИСПОЛЬЗОВАНИЯ ГРАФИЧЕСКОГО АКСЕЛЕРАТОРА ДЛЯ РЕАЛИЗАЦИИ НЕЙРОСЕТЕВЫХ АЛГОРИТМОВ

В последнее время нейросетевой подход к решению различного рода прикладных задач получает все большую популярность. Однако алгоритмы классификации и обучения нейросетей требуют существенных вычислительных мощностей.

В настоящее время существует набор аппаратных решений, позволяющих увеличить скорость обучения нейросетей и выполнения процедуры классификации за счет распараллеливания нейросетевых алгоритмов на нескольких процессорах. Например, компания NVidia предлагает аппаратный комплекс Tesla C870 состоящий из 128 процессоров с общей производительностью 518 гигафлоп, Tesla D870 (256 процессоров, 1 терафлоп), Tesla S870 (512 процессоров, 2 терафлоп). Однако

стоимость этих устройств довольно высока и составляет 1500, 7500, и 12000 USD соответственно[1].

В то же время любой современный компьютер оборудован графическим ускорителем, являющимся многопроцессорной системой, и способной выполнять пользовательские программы (шейдеры) параллельно, что также может быть использовано при практической реализации нейросетевых алгоритмов.

Например, видеокарта ATI Radeon HD 2900 XT, построенная на базе графического чипа RV630, содержит 120 унифицированных графических процессоров, с общей производительностью 475 гигафлоп и стоимостью всего 400 USD. Для сравнения можно сказать, что четырёхпроцессорная система на базе двуядерных чипов Intel Itanium 2 (Montecito) обеспечивает производительность всего порядка 45 гигафлоп[2].

Отмеченные обстоятельства стимулируют исследование возможности применения современных графических акселераторов (Graphic Processor Unit, GPU) в нейросетевой технике.

Типовая архитектура современных GPU

Современный графический акселератор, поддерживающий версию шейдеров 3.0, позволяет выполнять шейдеры двух типов:

- вершинные шейдеры;
- пиксельные шейдеры;

Вершинный шейдер – это программа, предназначенная для обработки одной вершины (вертекса). При формировании изображения каждый вертекс проходит заданную пользователем обработку, в процессе которой могут выполняться преобразование координат (применение мировой, видовой и проекционной матриц) вершины, расчет освещения, генерация текстурных координат и т.п. (рис. 1).

Рис. 1. Выполнение вершинного шейдера

После выполнения вершинного шейдера выполняется этап растеризации, в процессе которого, треугольники делятся на фрагменты (пиксели), для которых интерполируются текстурные координаты и цвет. В результате растеризации, полученный фрагмент помещается в буфер кадра, который является целевым объектом рендеринга (render target). На этапе растеризации выполняется пиксельный шейдер (рис. 2).

Пиксельный шейдер – это программа, предназначенная для расчета цвета пиксела (задаваемого в координатах целевого объекта рендеринга) в процессе растеризации изображения.

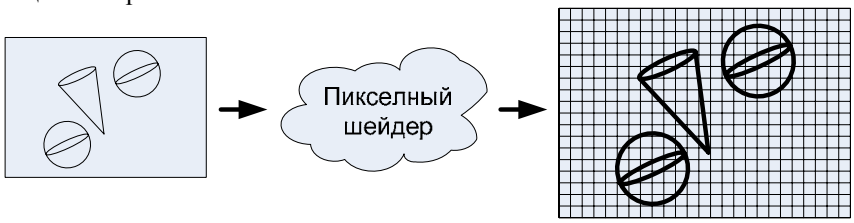

Рис. 2. Пикселный шейдер

Код программы (шейдера) выполняется графическим акселератором параллельно (как вершинного, так и пикселного), степень параллелизма зависит от аппаратной архитектуры графического акселератора.

Для достижения сложных графических эффектов часто используют несколько шейдеров, выполняемых последовательно, каждый из которых реализует требуемый визуальный эффект, например, отражение объектов, наложение бликов, эффектов тумана, дождя и т.п. (рис. 3,а). При этом прослеживается аналогия по отношению к расчету состояния многослойных нейросетей, когда для расчета состояния нейросети также последовательно выполняется расчет состояний каждого слоя нейросети (рис. 3,б).

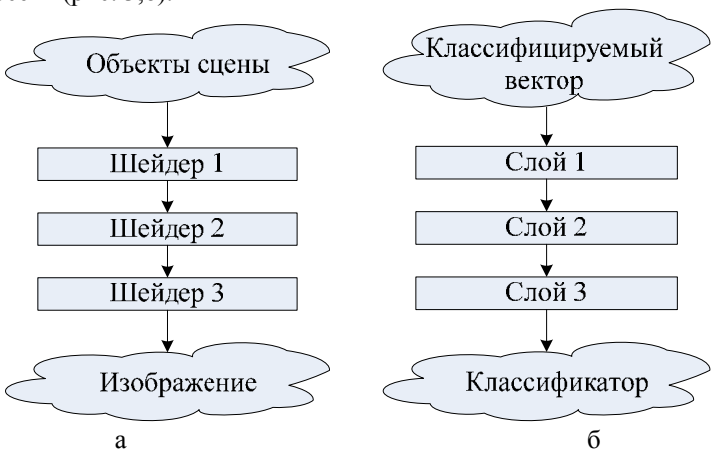

Рис. 3. Последовательное применение шейдеров (а) и расчет состояния нейросети (б)

Таким образом, становится возможным применение современных способов формирования графических изображений на основе GPU для расчета состояния многослойных нейросетей и реализации нейросетевых алгоритмов с применением шейдеров. При таком подходе можно рассчитывать состояние одного слоя нейросети за один кадр генерации изображения, при этом, каждый нейрон слоя нейросети будет моделироваться пикселом формируемого изображения, а пиксельный шейдер будет реализовывать активационную функцию нейрона. Вершинный шейдер при том никакой смысловой нагрузки не несет (так как не требуется выполнять какую-либо предварительную модификацию классифицируемого вектора), но должен быть формально реализован.

Реализация алгоритма классификации на GPU

Для того чтобы заставить графический акселератор выполнить рендеринг кадра, необходимо наполнить список объектов рендеринга графическими примитивами – для этого достаточно формально сформировать сцену, состоящую из двух треугольников покрывающих 2D-пространство с координатами (0,0), (1,1) путем вызова функции DirectX DrawPrimitiveUP.

При программировании шейдеров для графической подсистемы DirectX используется язык программирования High Level Shading Language (HLSL).

Всю реализацию нейросетевого алгоритма классификации предлагается реализовать на базе пикселного шейдера. При этом достаточно создать вершинный шейдер, не выполняющий никаких преобразований над вертексами:

```
struct VS_OUTPUT
{
        float4 Position : POSITION;
        float3 TextCoord : TEXCOORD0;
};
VS_OUTPUT ShaderFunc(float4 Pos : POSITION)
{
        VS_OUTPUT Out = (VS_OUTPUT)0;
        Pos.w = 1.0f;
        Out.Position = Pos;
        Out.TextCoord = Pos;
        return Out;
}
```

После применения вершинного шейдера, графический акселератор начинает выполнять код пикселного шейдера для каждого пиксела целевого объекта рендеринга. При этом код шейдера, выполняемый графическим акселератором для каждого пиксела целевого объекта рендеринга, реализует заданную активационную функцию нейрона. Данный факт накладывает следующее ограничение на организацию архитектуры нейросети – активационная функция каждого нейрона слоя нейросети должна быть одинаковой в пределах нейросетевого слоя. Практически, существует возможность реализации уникальной активационной функции для каждого нейрона в слое, путем задания дополнительной управляющей текстуры, значение элементов которой (текселов) будет трактоваться шейдером как тип активационной функции каждого нейрона в слое (пороговая, логистическая, и т.п.). Однако такой подход не рекомендуется к применению, так как он требует организации ветвлений в коде шейдера (логических условий и переходов), что негативно сказывается на производительности шейдера. Более того, типовые схемы построения многослойных нейросетей подразумевают наличие нейронов с одинаковой активационной функцией в пределах одного слоя нейросети[3].

Для реализации алгоритмов классификации на основе классического многослойного перцептрона необходимо обеспечить представление следующих структур данных:

- набор входных значений каждого слоя нейронов $X_{ik} = \{x_{11},...,x_{M1},...,x_{1N},...,x_{MN}\}$, где M – число входов нейрона в слое, N – число нейронов в слое;
- набор весовых коэффициентов для каждого нейрона в слое $\mathrm{W}_{ik} = \{w_{11},...,w_{M1},...,w_{1N},...,w_{MN}\}$, где M – число входов нейрона в слое, N – число нейронов в слое;
- набор значений выходов нейронов $\mathrm{Y}_k = \{y_1,...,y_N\}$, где N – число нейронов в слое.

При этом, значения выходов Y нейронов k-го слоя будут являться входами X для нейронов k+1 слоя.

Команды пикселного шейдера не могут обращаться к данным, расположенным непосредственно в оперативной памяти компьютера – в качестве исходных данных пикселный шейдер может обращаться только к текстурам, значения элементов которых (тексели) трактуются GPU по разному. При применении типа данных D3DFMT_R32F можно заставить GPU трактовать элементы текстуры в виде вещественных чисел одинарной точности. Тогда можно представить внутреннее

состояние нейросети с помощью вышеперечисленных структур данных в виде 2D-текстур (пример для двухслойной нейросети приведен на рис. 4).

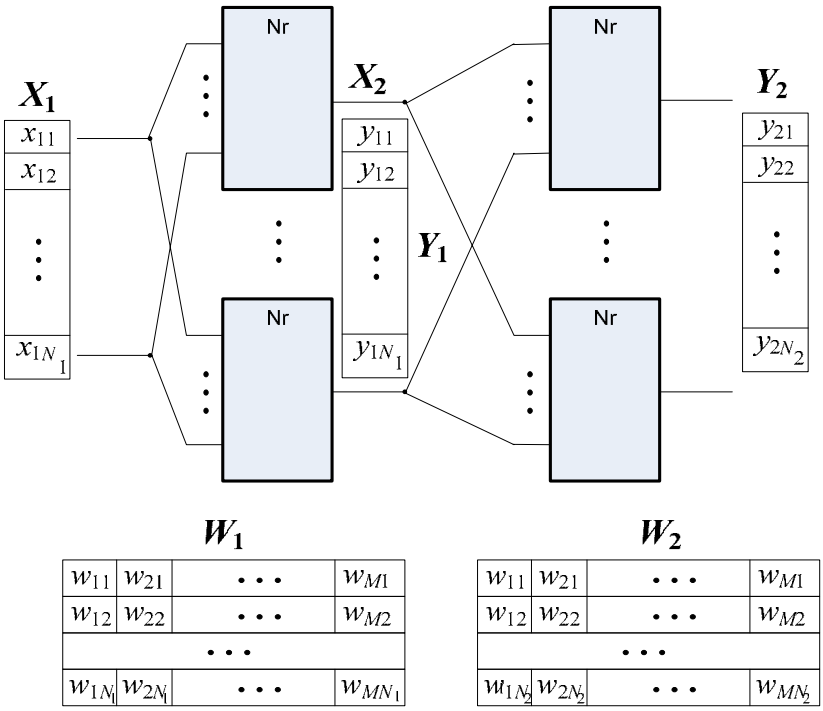

Рис. 4. Представление состояния нейросети в виде текстур

В процессе выполнения рендеринга удобно трактовать каждый пиксел целевого объекта рендеринга как один нейрон слоя, при этом за один кадр можно рассчитать значение выходов всех нейронов в пределах одного слоя нейросети. Для расчета многослойных нейросетей достаточно выполнить рендеринг L кадров, где L – число слоев нейросети.

Тогда для пикселного шейдера исходными данными будут являться:

- текстура X входных значений нейронов;
- текстура W весовых коэффициентов входов нейронов.

Целевым объектом рендеринга будет являться текстура Y, заполняемая рассчитанными значениями выходов нейронов.

Таким образом, графический акселератор выполняет код шейдера для каждого пиксела целевого объекта рендеринга, который в данном случае вычисляет значение выхода нейрона Y на основе заданной активационной функции с учетом входных значение нейрона X и весовых коэффициентов W, значения которых передаются в шейдер в виде текстур. Результат выполнения рендеринга представляется также в виде текстуры Y (рис. 5).

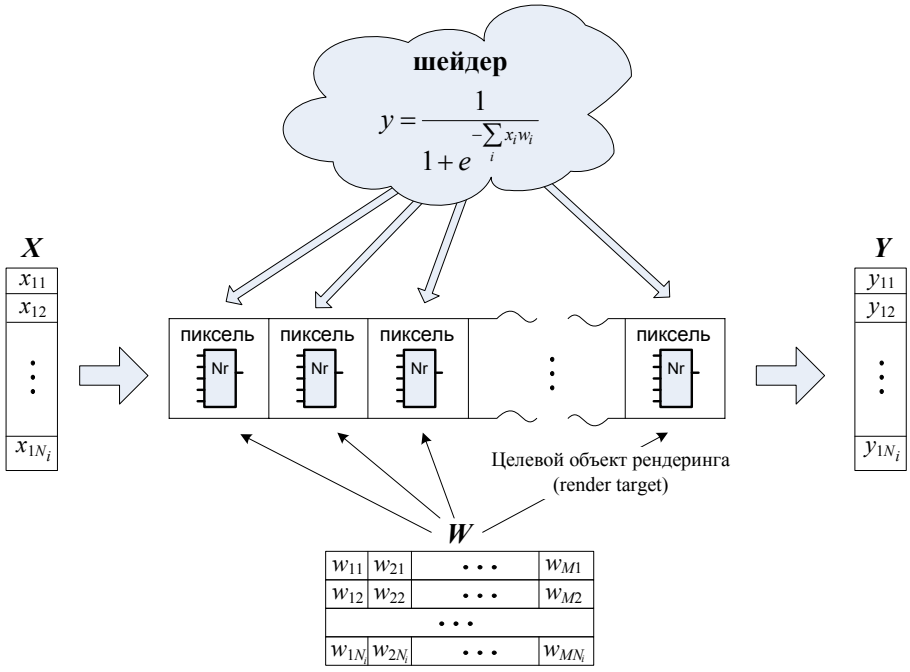

Рис. 5. Расчет состояния нейронов слоя нейросети

Например, для слоя нейросети, каждый нейрон которой имеет по два входа и реализует логистическую активационную функцию, получим следующий текст пикселного шейдера:

```
float4 Steps : register(c0);
float4 Offsets : register(c1);
sampler2D WTexture : register(s1);
sampler2D XTexture : register(s3);
struct PS_INPUT
{
   float4 Position : POSITION;
   float3 TextCoord : TEXCOORD0;
   float2 vPos : VPOS;
};
float4 PixelSh(PS_INPUT Vertex):COLOR0
{
   const float MaxNet = 45;
   int i = 0;
   float2 WPos = {Offsets.x, Vertex.vPos.x*Steps.z + Offsets.z};
   float2 XPos = {Offsets.x, Vertex.vPos.y*Steps.y + Offsets.y};
   float sum = 0;
   sum = sum + tex2D(XTexture, XPos)*tex2D(WTexture, WPos);
   XPos.x = XPos.x + Steps.x;
   WPos.x = WPos.x + Steps.x;
   sum = sum + tex2D(XTexture, XPos)*tex2D(WTexture, WPos);
   XPos.x = XPos.x + Steps.x;
   WPos.x = WPos.x + Steps.x;
   sum = clamp(sum, -MaxNet, MaxNet);
```

```
    sum = 1 + exp(-sum);
    sum = sum*sum;
    sum = rsqrt(sum);
    return float4(sum,0,0,0);
}
```

Следует обратить внимание на то, что текст пикселного шейдера может генерироваться программой динамически и компилироваться с помощью функции D3DXCompileShader, что позволяет создавать пикселные шейдеры “на лету” для практически любой конфигурации нейросети. Одним из важных ограничений, накладываемых на пикселные шейдеры является то, что большинство современных видеокарт позволяют работать с текстурами размером не более 4096 пикселов по каждому размерению (по ширине или высоте). Исходя из этого ограничения следует, что предложенная схема представления внутреннего состояния нейросети (рис. 4) позволяет выполнять процедуру классификации для нейросетей, где число нейронов в каждом слое не превышает 4096. Данное ограничение может быть снято за счет усложнения представления внутреннего состояния нейросети и пикселного шейдера.

Сравнительный анализ быстродействия нейросетевого алгоритма классификации на базе CPU и GPU проводился для однослойной нейросети с размерностью классифицируемого вектора в 1000 элементов одинарной точности и варьируемым числом нейронов в слое. Проведенные эксперименты показывают, что графический акселератор (GPU) не всегда обладает превосходством над CPU по быстродействию (рис. 6). Для использованной аппаратной конфигурации баланс производительности наступает для порядка 45 000 связей (45 нейронов × 1000 входов) в слое.

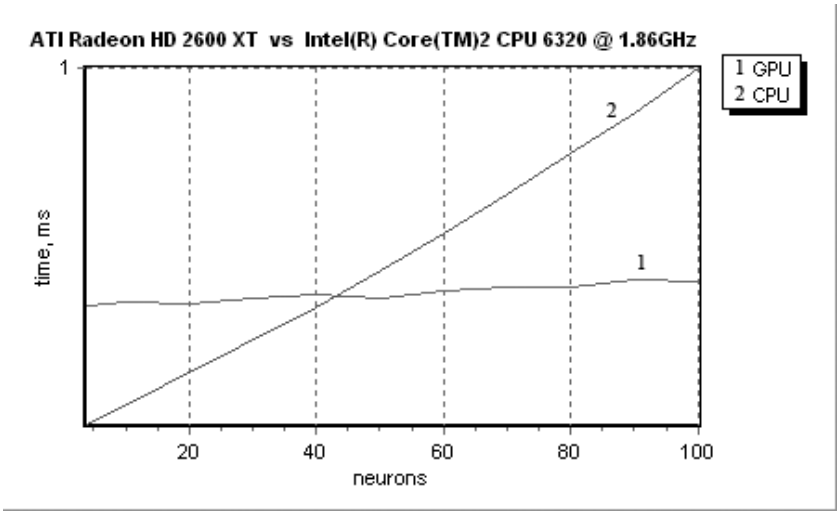

Рис. 6. Быстродействие CPU и GPU для нейросетей небольшой размерности

Для нейросетей большой размерности, порядка 10^6 связей в слое (1000 нейронов × 1000 входов), выигрыш производительности достигает 800% (рис. 7).

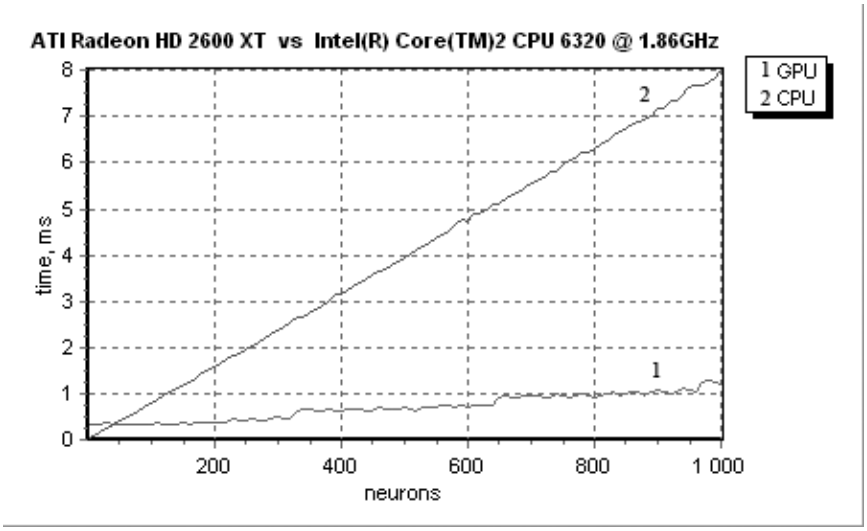

Рис. 7. Быстродействие CPU и GPU для нейросетей большой размерности

Современное развитие архитектуры графических акселераторов позволяет использовать имеющиеся вычислительные мощности для реализации нейросетевых алгоритмов и, в частности, алгоритма классификации.

Применение подобного подхода на практике позволяет в ряде случаев получить существенное увеличение быстродействия работы нейросетевых алгоритмов.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. NVidia. “Высокопроизводительные вычислительные решения”. http://www.nvidia.ru/page/tesla_computing_solutions.html.
2. *Вайцман И.* “Обзор видеоадаптера HIS HD 2900XT”. http://www.radeon.ru/articles/r6xx/his2900xt/1/.
3. *Уоссермен Ф.* Нейрокомпьютерная техника: Теория и практика. – М.: Мир, 1992. – 240 с.

УДК 681.3.78

А.А. Зори, В.П. Тарасюк

ЭЛЕМЕНТЫ КОМПЬЮТЕРИЗАЦИИ ПОДГОТОВКИ БАКАЛАВРОВ ПО НАПРАВЛЕНИЮ «ЭЛЕКТРОННЫЕ УСТРОЙСТВА И СИСТЕМЫ» СОГЛАСНО ТРЕБОВАНИЯМ БОЛОНСКОЙ КОНВЕНЦИИ

Проблема повышения качества подготовки специалистов в высших учебных заведениях обуславливает необходимость постоянного совершенствования организации учебного процесса. Это касается подготовки бакалавров по направлению «Электронные устройства и системы», поскольку развитие информационных технологий в электронике значительно опережает темпы модификации учебного процесса.

Решение проблемы можно разбить на несколько этапов: усовершенствование учебного плана, обучение, закрепление знаний, подготовка вопросов и тестов для промежуточной аттестации и бакалаврских экзаменов. Результат зависит, в значительной мере, от того, насколько продуманными и логически последовательными будут перечисленные этапы. Важно, как будут они расположены оптималь-