МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РФ
федеральное государственное бюджетное образовательное учреждение высшего образования
КРАСНОЯРСКИЙ ГОСУДАРСТВЕННЫЙ ПЕДАГОГИЧЕСКИЙ УНИВЕРСИТЕТ им.В.П.АСТАФЬЕВА
(КГПУ им.В.П.Астафьева)

Институт/факультет Институт математики, физики и информатики
(полное наименование института/факультета/филиала)

Выпускающая кафедра Базовая кафедра информатики и информационных технологий в образовании
(полное наименование кафедры)

Иванов Антон Владимирович

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

Тема Терминальный сервер для поддержки обучения высокопроизводительным вычислениям с использованием графического ускорителя.

Направление подготовки 44.03.01 Педагогическое образование
(код и наименование направления)

Профиль Информатика
(наименование профиля для бакалавриата)

ДОПУСКАЮ К ЗАЩИТЕ
Заведующий кафедрой
д.п.н., профессор Пак Н.И.
(ученая степень, ученое звание, фамилия, инициалы)
11.06.2018
(дата, подпись)

Руководитель к.ф.-м.н., доцент Шикунов С.А.
(ученая степень, ученое звание, фамилия, инициалы)

Дата защиты 26.06.2018г.

Обучающийся Иванов А.В.
(фамилия, инициалы)
11.06.2018
(дата, подпись)

Оценка отлично
(прописью)

Красноярск 2018

Оглавление

Введение

С развитием графических процессоров появилась возможность сократить нагрузку на центральный процессор, выполняя вычисления с большими объемами данных на графическом ускорителе. Такое распределение в разы увеличило производительность вычислений. Появилось новое направление в программировании – параллельное программирование, позволяющие выполнять высокопроизводительные вычисления в том числе и на графических ускорителях. Параллельное программирование сегодня используется во всех областях деятельности человека. Умение программировать, используя графический ускоритель, дает учащимся возможность реализовать себя в более обширной области, увеличивает конкурентоспособность на рынке труда, являясь несомненным плюсом при принятии на работу.

Для успешного обучения параллельному программированию с использованием графического ускорителя необходим доступ к его ресурсам всем учащимся. На текущий момент аппаратные средства, используемые в высокопроизводительных вычислениях, одна из самых затратных статей бюджета любой организации. Чтобы сократить издержки на их приобретение, обслуживание, настройку необходима технология, обеспечивающая полный доступ к ресурсам сервера, в том числе графического ускорителя, посредством перенаправления их от сервера к компьютеру пользователя. Проблема заключается в том, что различные виды терминального доступа в той или иной мере ограничивают использование ресурсов аппаратных средств терминального сервера.

Виртуализация рабочих станций позволяет решить эти вопросы.

Гипервизоры позволяют быстро создать или изменить виртуальные машины, что позволяет использовать их для построения тестовых стендов, а также для обучения новым продуктам и технологиям, обеспечивает

разделение ресурсов сервера между виртуальными машинами, что повышает КПД использования ресурсов, а благодаря тому, что виртуальные машины состоят из стандартных виртуальных компонентов, проблемы совместимости с операционными системами и программным обеспечением отсутствуют.

Актуальность исследования обусловлено тем, что учебному учреждению в условиях ограниченного бюджета и отсутствия достаточного штата сотрудников для настройки и обслуживания компьютеров необходимо предоставить возможность учащимся полноценного обучения, в том числе и высокопроизводительным вычислениям. Использование терминального сервера для поддержки обучения позволяет сэкономить на покупке и обслуживании аппаратных и программных средств в долгосрочной перспективе.

Целью исследования является разработка рекомендаций для использования терминального сервера для поддержки обучения высокопроизводительным вычислениям.

Объектом исследования является техническое обеспечение учебного процесса в области информатики. Предмет исследования – терминальный сервер для выполнения лабораторных работ по высокопроизводительным вычислениям.

Для достижения поставленной цели были сформулированы следующие задачи:

1. Проанализировать требования к аппаратной части компьютера для выполнения лабораторных работ по высокопроизводительным вычислениям с использованием графического ускорителя.

2. Проанализировать возможность доступа к графическому ускорителю при различных видах виртуализации.

3. Определить наиболее эффективные способы программного доступа к графическому ускорителю из виртуальной среды.

4. Апробировать способы программного доступа к графическому ускорителю из виртуальной среды.

5. Разработать приёмы использования терминального сервера для выполнения лабораторных работ по высокопроизводительным вычислениям с использованием графического ускорителя.

6. Разработать рекомендации по созданию инфраструктуры для выполнения лабораторных работ по высокопроизводительным вычислениям с использованием графическому ускорителю.

1. Высокопроизводительные вычисления на графических ускорителях.

1.1. Аппаратные средства

Параллельное программирование возникло в 1962 г. с появлением каналов — независимых аппаратных контроллеров, за счет, которых центральный процессор мог выполнять новую прикладную программу одновременно с операциями ввода-вывода других (приостановленных) программ. Параллельное программирование (слово параллельное в данном случае означает "происходящее одновременно") первоначально было уделом разработчиков операционных систем. В конце 60-х годов были созданы многопроцессорные машины. В результате не только были поставлены новые задачи разработчикам операционных систем, но и появились новые возможности у прикладных программистов.

Сейчас стала заметной необходимость обработки данных с массовым параллелизмом, при котором для решения одной задачи используются десятки, сотни и даже тысячи процессоров. В широкое обращение вошли многопроцессорные рабочие станции. Параллельное аппаратное обеспечение используется больше, чем когда-либо, а параллельное программирование становится необходимым.

Многопроцессорные машины позволяют разным прикладным программам выполняться одновременно на разных процессорах. Они также ускоряют их выполнение, если приложение написано для многопроцессорной машины. Данная концепция архитектуры называется – множественный поток команд и данных (MIMD – Multiple Instruction Stream, Multiple Data Stream, рис.1). Каждое ядро работает отдельно от других, последовательно исполняя команды текущего процесса.

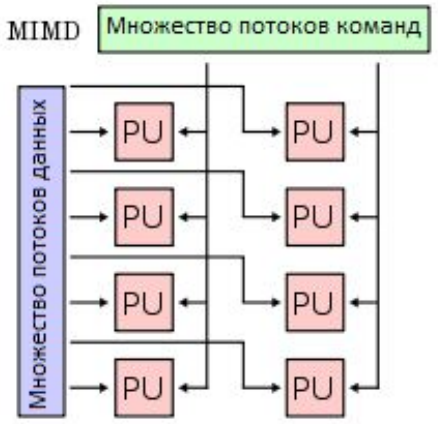


Рис 1. Схема работы многоядерной архитектуры MIMD.

За последние десятилетия во всех областях науки значительно выросла потребность в обработке огромного количества данных. Для этого необходимы мощные вычислительные системы. Развитие центральных процессоров практически остановилось (стало невозможно увеличивать тактовую частоту процессора). Графические ускорители, напротив, стали активно развиваться и изначально рассчитанные на обработку графических данных, стали использоваться для неграфических вычислений, что позволило достичь ускорения работы приложений в десятки, а то и в сотни раз.

Параллельная архитектура графических процессоров ориентирована на исполнение алгоритмов, в которых элементы больших входных массивов обрабатываются одинаковым образом независимо или почти независимо друг от друга, то есть использующих распараллеливание вычислений по данным. Множества элементов, подвергаемых однотипной независимой обработке,

называют потоками данных, так что графические процессоры осуществляют поточно-параллельную обработку данных.

Концепция программирования, заключающаяся в потоковой обработке данных, известна под аббревиатурой SIMD (Single Instruction — Multiple Data — одна инструкция для множества данных, рис. 2). Устройство, работающее по принципу SIMD, имеет множество вычислительных элементов, до нескольких тысяч, и каждый элемент устройства параллельно выполняет какую-либо инструкцию с потоком данных.

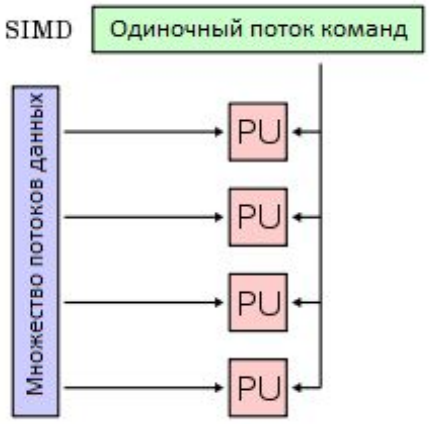


Рис 2. Схема работы многоядерной архитектуры SIMD.

По соотношению цены и производительности графические ускорители имеют большое преимущество перед другими вычислительными системами, в том числе и перед специализированными суперкомпьютерами.

Термин Graphics Processing Unit (GPU) был впервые использован корпорацией NVIDIA для обозначения того факта, что графический ускоритель, первоначально используемый только для ускорения трехмерной графики, стал мощным программируемым устройством – процессором, пригодным для решения значительно более широкого класса вычисленных задач (General Purpose computations on GPU, GPGPU).

Первые графические ускорители от NVIDIA с поддержкой CUDA появились в 2006 г. Серия, основанная на чипе G80. Самый мощный адаптер из серии имел 128 потоковых процессоров, производительностью около 600

Гфлопс. За неполных 20 лет компания усовершенствовала и увеличила производительность своих графических ускорителей почти в 1000 раз. На сегодняшний день самая мощная видеокарта – NVidia Titan V на чипе GV100 имеет 5120 скалярных процессоров, производительностью 110 Тфлопс.

В чем же основные отличия GPU и CPU? Во-первых, CPU имеет небольшое число ядер, работающих на высокой тактовой частоте независимо друг от друга. GPU же напротив работает на низкой тактовой частоте и имеет сотни и тысячи упрощенных вычислительных элементов.

Во-вторых, большую часть устройства CPU занято кэш-памятью, а вычислительными элементами лишь небольшая часть кристалла (Рис. 3).

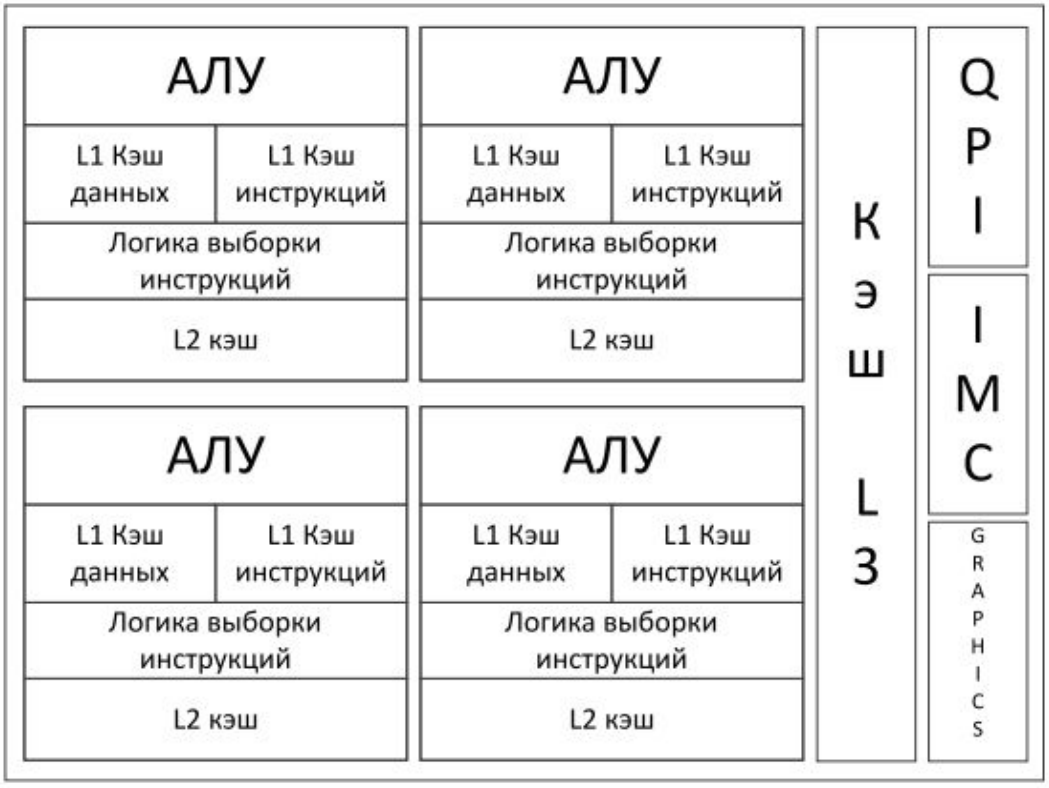


Рис 3. Схематическое устройство процессора Intel.

Кроме того, каждый отдельный вычислительный элемент является полноценным процессором. Он поддерживает все аппаратные прерывания, может взаимодействовать со всеми устройствами ввода/вывода, что, конечно, необходимо для его работы как центрального процессора, но мешает его работе в качестве векторного вычислительного модуля.

Более того, потоки, выполняемые на центральном процессоре, являются очень «тяжелыми», ими управляет операционная система, поэтому их не может быть много (максимальное количество не превышает несколькими тысячами).

Теперь сравним центральный процессор с графическим процессором (GPU). (Рис. 4) Значительная часть вычислительного кристалла занята элементарными вычислительными устройствами, кэш и устройства управления занимают очень мало пространства.

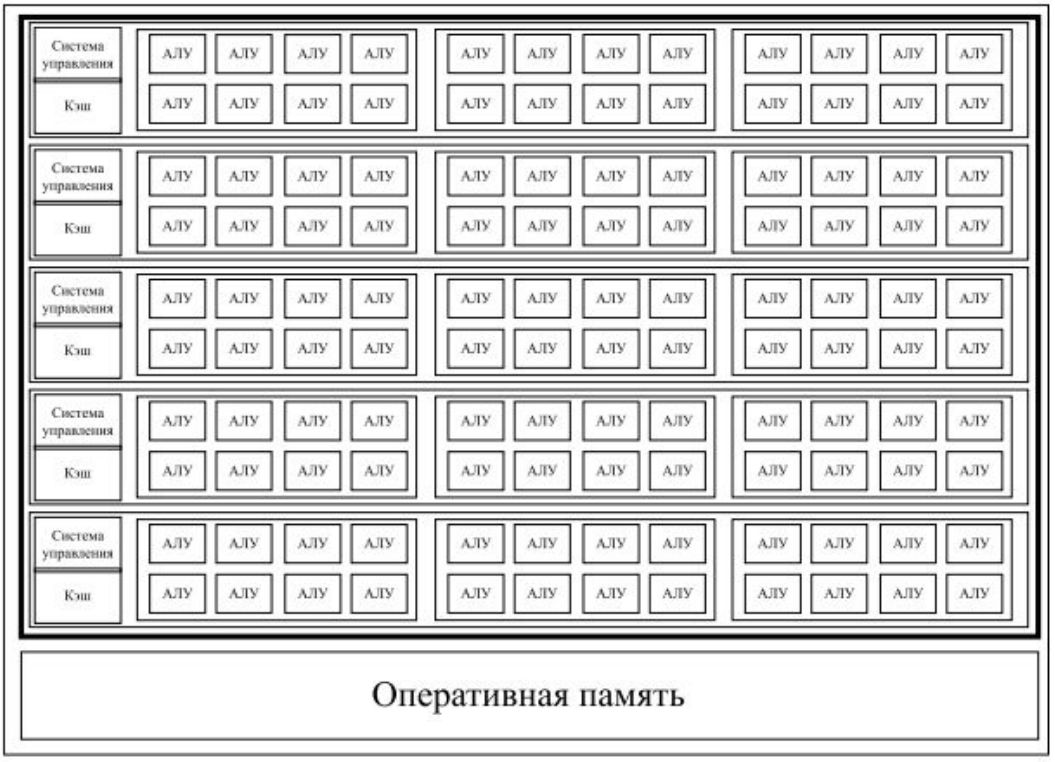


Рис 4. Схематическое устройство графического процессора.

Процессоры на графических ускорителях куда проще, чем на центральных процессорах, они могут исполнять только математические операции и неспособны к самостоятельной работе (являются сопроцессорами к центральному процессору). На центральном процессоре (хост, host) выполняются исключительно последовательные части алгоритма программы: подготовка и копирование данных на графический ускоритель, задание инструкций для ядра (kernel – параллельная часть алгоритма) и их запуск. Взаимодействие с центральным процессором осуществляется через мост,

который может быть как интегрированным в CPU (Intel Core i7), так и вынесенным в чипсет (Intel Core 2 Duo). Параллельные части алгоритма оформляются в потоки данных, которые выполняются на большом количестве ядер на графическом ускорителе.

Процессор графического ускорителя (GPU, устройство, device) состоит из планировщика блоков потоков, мультипроцессоров (SM, Streaming Multiprocessor) и кэша L2. Мультипроцессоры состоят из потоковых процессоров (SP, Streaming Processor), объединенных в независимые блоки, планировщика потоков, разделяемой памяти, банка регистров, а также текстурного, константного и L1 кэшей. Кэш L2 автоматически кэширует данные при обращении к глобальной памяти, чем ускоряет как последующий повторный доступ, так и доступ к соседним данным. Кэш L2 появился в архитектуре CUDA, начиная с версии 2.x (Fermi). В зависимости от модели графического ускорителя может меняться количество мультипроцессоров и их составляющих, так же может меняться тип и количество микросхем памяти DRAM и, соответственно, общий объем памяти всего устройства. Микросхемы памяти имеют кэш второго уровня (L2) и конвейер растровых операций (ROP — Raster Operations Pipeline). Они объединены между собой коммуникационной сетью, в которую также подключены все мультипроцессоры. Меняя количество мультипроцессоров, которое определяется как принадлежностью к тому или иному семейству процессоров (GP108 – 3, GP 107 – 5-6, GP106 – 9-10, GP104 – 15-20, GP102 – 28 до 30, GV100 – 80), так и конкретной моделью (семейство GP102: GTX 1080Ti, Titan X – 28, TITAN Xp – 30), производитель пропорционально меняет потребляемую мощность и производительность графического ускорителя, и его цену.

Планировщик блоков потоков следит за загруженностью мультипроцессоров, где исполняются блоки потоков, и, по завершении

работы одних блоков, отправляет еще неотработанные блоки на освободившиеся мультипроцессоры. (Рис. 5)

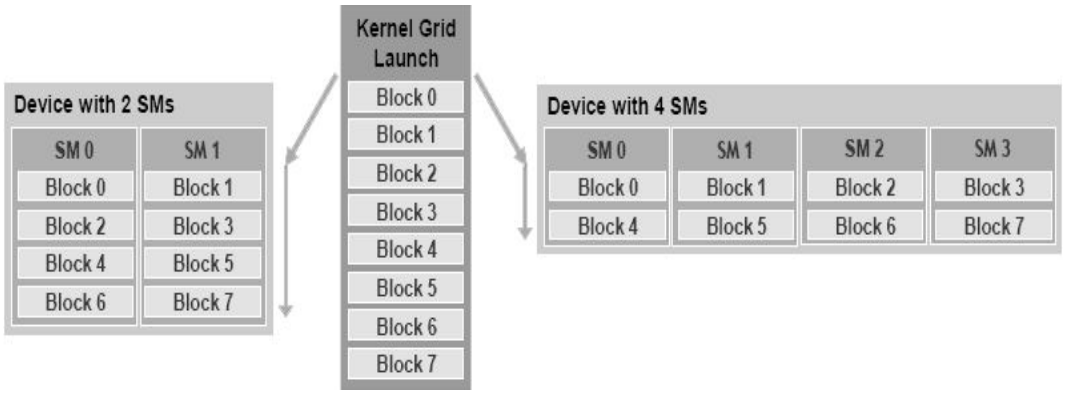


Рис. 5. Схема работы планировщика блоков.

На одном графическом ускорителе могут исполняться несколько блоков потоков. Минимальной единицей исполнения на мультипроцессоре является варп (warp) – группа до 32 потоков одного блока, выполняется физически одновременно. Поток (тред, thread) – минимальный элемент выполнения программы, обладает уникальный идентификатором внутри блока. Блок (block) – совокупность потоков, которое выполняется целиком на одном мультипроцессоре, обладает уникальный идентификатором внутри грида. Грид (grid) – объединение блоков, которые выполняются на одном устройстве. (Рис. 6)

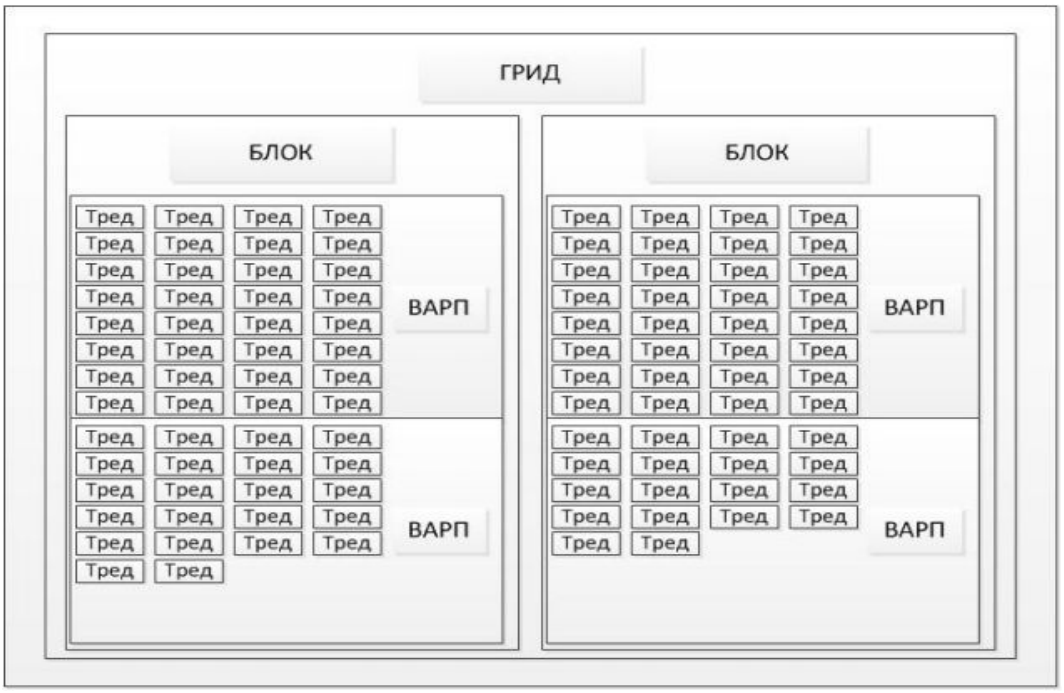


Рис. 6. Схема организации параллельных процессов.

Следует отметить, что взаимодействие потоков из разных блоков во время работы ядра затруднено: отсутствуют явные инструкции синхронизации, взаимодействие возможно через глобальную память и использованием атомарных функций (другим вариантом является разбиение ядра на несколько ядер без внутреннего взаимодействия между потоками разных блоков).

Еще одним из серьезных отличий между CPU и GPU являются организация памяти и работа с ней. Обычно большую часть CPU занимают кэши различных уровней. А основная часть GPU отведена на вычисления. Как следствие, в отличие от CPU, где есть всего один тип памяти с несколькими уровнями кеширования в самом CPU, GPU обладает более сложной, но в то же время гораздо более документированной структурой памяти.

Чисто физически память GPU можно разделить на DRAM и на память, размещенную непосредственно на GPU (точнее, в потоковых мультипроцессорах). Однако, классификация памяти в CUDA не ограничивается только ее физическим расположением.

В таблице 1 указаны сравнительные характеристики разных типов памяти, доступных на графическом ускорителе от NVIDIA.

Таблица 1. Типы памяти в CUDA.

Тип памяти	Доступ	Уровень выделения	Скорость работы
Регистры (register)	R/W	на поток	Высокая (on-chip)
Локальная (local)	R/W	на поток	Низкая (DRAM)
Разделяемая (shared)	R/W	на грид	Высокая (on-chip)
Глобальная (global)	R/W	на грид + хост	Низкая (DRAM)
Константная (constant)	R/O	на грид + хост	Высокая (L1 cache)
Текстурная (texture)	R/O	на грид + хост	Высокая (L1 cache)

Регистры – применяют для хранения локальных переменных. Физически находятся на кристалле графического процессора; скорость доступа самая быстрая. Выдаются отдельно для каждого потока.

Локальная память – это область в глобальной памяти, выделенная компилятором для хранения локальных значений потоков. Она применяют для хранения локальных данных потоков при отсутствии свободных регистров или при объявления локальных массивов внутри ядра без ключевого слова. Скорость доступа низкая, так как находится в микросхемах DRAM, находящихся вне кристалла. Выдается отдельно для каждого потока.

Разделяемая – применяют для хранения массивов данных, используемых совместно всеми потоками в блоке. Находится на кристалле GPU; имеет чуть меньшую скорость доступа, чем регистры (около 10 тактов). Выдается на блок.

Глобальная – основная память графического ускорителя. Применяют для хранения больших массивов данных. Находится в микросхемах DRAM и имеет медленную скорость доступа (около 80 тактов). Выдается целиком на грид.

Константная – память, находится в микросхемах DRAM, снабжена специальным константным кэшем. Применяют для передачи в ядро аргументов, превышающих допустимые размеры для параметров ядра (для чипа Fermi – 256 байт). Выдается целиком на грид.

Текстурная – память, находится в микросхемах DRAM, кэшируется. Применяют для хранения больших массивов данных. Выдается целиком на грид.

Все типы памяти, кроме регистровой и локальной, могут быть использованы в программе как эффективно, так и неэффективно. Для эффективного обращения к памяти необходимо четко понимать особенности и области применения разных её типов.

Вычисления с использованием графических адаптеров показывают максимальную эффективность в задачах, не требующих интенсивного обращения к памяти. Чуть хуже решаются задачи, для которых памяти требуется много, но при этом есть возможность использовать разделяемую память. Меньшее ускорение будет получена, когда требуется доступ по случайному адресу, но есть переиспользование данных (используется текстурная память). Совсем плохо будут решаться те задачи, для которых не выполняется ни одно из этих требований. Кроме того, если задача требует большого количества памяти, то, скорее всего, на данном этапе развития технологии CUDA её вообще не целесообразно решать при помощи GPU.

1.2. Программные средства.

На данный момент существует достаточно много технологий использования графического ускорителя для высокопроизводительных вычислений. Самыми известными являются CUDA от NVIDIA, OpenCL от Khronos Group, DirectCompute от Microsoft. Главным недостатком CUDA является работоспособность только на графических картах от NVIDIA.

Однако, у CUDA есть альтернатива в лице OpenCL, который изначально задумывался как кроссаппаратный и кроссплатформенный. Он позволяет создавать программы как для адаптеров NVIDIA, так и для устройств от AMD. DirectCompute также работает на видеокартах разных производителей, но только на операционных системах Windows.

OpenCL (Open Computing Language, открытый язык вычислений) — это открытый стандарт для параллельного программирования, предлагающий эффективный и переносимый способ использования возможностей разнородных вычислительных многоядерных платформ (CPU, GPU и др.). Он включает в себя программный интерфейс (API) для координирования параллельных вычислений в среде разнородных процессоров и кроссплатформенный язык, используемый в определённом вычислительном окружении.

На конкретной системной платформе стандарт реализуется в виде исполняемых модулей (библиотек), достаточных для запуска пользовательских OpenCL-приложений. Разработка подобных приложений предполагает наличие также C/C++-компилятора системной платформы и набора необходимых заголовочных файлов.

OpenCL — это стандарт, предложенный Apple в 2008 году. Он позволяет создавать платформенно-независимые параллельные программы. OpenCL рассчитан на мультипроцессоры, GPU и всевозможные ускорители.

Идея Apple заключалась в том, чтобы разогнать свой графический интерфейс с помощью простаивающих специализированных мощностей во время неигровой деятельности. Позже к Apple решила примкнуть фирма AMD, теперь OpenCL официально поддерживают фактически все основные производители вычислительной техники, включая IBM, Intel, AMD и NVIDIA. Разработкой OpenCL занимается некоммерческая независимая организация Khronos Group (http://www.khronos.org/).

OpenCL позволяет работать как с параллелизмом по данным, когда одна и та же операция одновременно применяется к разным комплектам данных, так и с параллелизмом на уровне задач, когда независимые потоки команд работают одновременно и выполняют разные функции.

В мире современных вычислительных систем есть весьма разнообразный и неоднородный набор вычислителей. Во-первых, есть мощные центральные процессоры, также легко доступны графические процессоры, которые можно использовать для счета. Есть и другие разнообразные специализированные вычислительные ускорители.

Даже в совершенно обычном компьютере уже довольно давно сожительствуют несколько разнотипных вычислителей — как минимум один процессор и графическая карта. А если посмотреть на самые высокопроизводительные кластеры, то размещение в их узлах нескольких многоядерных центральных процессоров и нескольких GPU стало теперь уже практически стандартным. Но на этом эволюция не закончилась.

Цель создания OpenCL — дать единую технологию построения программ на разнородных вычислителях. Написав только одну программу, можно запустить ее фактически на всех параллельных устройствах. А если сейчас этого сделать нельзя — то это ненадолго. Более того, OpenCL, в теории, позволит запустить программу, написанную сегодня, на вычислителе, который создадут в будущем, даже без необходимости перекомпилировать ее заново. Следует, однако, осознавать, что хоть правильно отлаженная программа под соответствующую архитектуру не уступает по производительности той же CUDA, но при переносе на принципиально другой вычислитель с другой производительностью безусловно могут возникнуть серьезные проблемы. необходимо отметить, что использование

Необходимо отметить, что OpenCL не исключает возможность применения других технологий построения параллельных программ в одной

программе. Например, на кластерах с многопроцессорными узлами или узлами с GPU целесообразно строить гибридные программы на базе OpenCL и MPI. OpenCL в них будет отвечать за организацию параллельного счета внутри одного узла, а MPI — за параллельное исполнение на уровне кластера, между несколькими узлами.

DirectCompute — интерфейс программирования приложений (API), который входит в состав DirectX — набора API от Microsoft, который предназначен для работы на IBM PC-совместимых компьютерах под управлением операционных систем семейства Microsoft Windows. DirectCompute предназначен для выполнения вычислений общего назначения на графических процессорах, являясь реализацией концепции GPGPU.

DirectCompute, впервые появившись в составе DirectX 11, стал одним из важнейших его нововведений, первой технологией в составе DirectX, предоставившей доступ к вычислениям общего назначения на графических процессорах.

Особенный интерес DirectCompute вызывает при просчёте алгоритмов физических движков. С помощью DirectCompute можно обрабатывать механику твёрдых тел, физику тканей и гидрогазодинамику. Компания AMD активно работает с DirectCompute в рамках проекта Open Physics Initiative.

DirectCompute поддерживается всеми основными компаниями на рынке производства графических процессоров: AMD и NVIDIA. На графических процессорах производства AMD технология AMD FireStream работает «поверх» DirectCompute. AMD в сотрудничестве с Pixelux Entertainment в рамках проекта Open Physics Initiative работает над переносом физических вычислений на GPU. Кроме этого, в сотрудничестве с компанией CyberLink AMD работает над «переносом» на DirectCompute алгоритмов кодирования и декодирования видеоданных, редактирования видео, распознавания лиц.

На графических процессорах производства NVIDIA DirectCompute работает «поверх» CUDA.

До появления CUDA единственный способ использования видеочипов для решения неграфических задач, то есть для задач общего назначения GPGPU, было использование графических библиотек с кодированием и декодированием данных в графические объекты. Естественно, такой способ использования был неудобным, что и побудило на создание платформы для обычных вычислений.

С появлением программно-аппаратной архитектуры CUDA графических ускорителей компании NVIDIA ситуация кардинально изменилась. Программы стали составляться не на специальном языке описания шейдеров, а на знакомом C/C++, что позволило программистам использовать уже имеющиеся навыки для параллельных вычислений.

CUDA строится на концепции, что GPU (называемый «устройством», device) выступает в роли массивно-параллельного сопроцессора к CPU (называемый «хост», host). Приложение на CUDA использует как CPU, так и GPU. Последовательный, то есть непараллельный код выполняется на CPU, а для массивно-параллельных вычислений соответствующий код выполняется на GPU как набор одновременно выполняющихся нитей (потоков, threads).

Таким образом, GPU рассматривается как специализированное вычислительное устройство, которое:

- является сопроцессором к CPU;
- обладает собственной памятью;
- обладает возможностью параллельного выполнения огромного количества отдельных нитей.

При этом очень важно понимать, что между нитями на CPU и нитями на GPU есть принципиальные различия:

- нити на GPU обладают крайне небольшой стоимостью создания, управления и уничтожения (контекст нити минимален, все регистры распределены заранее);
- для эффективной загрузки GPU необходимо использовать много тысяч отдельных нитей, в то время как для CPU обычно достаточно 10 – 20 нитей.

Основные характеристики CUDA:

- общее программно-аппаратное решение для параллельных вычислений на видеочипах NVIDIA;
- является расширением языков C и C++, что позволяет не учить новый язык, а выучить только сами нововведения в язык;
- присутствуют библиотеки численного анализа и линейной алгебры;
- оптимизирован обмен между CPU и GPU;
- кроссплатформенность – поддержка известных операционных систем: Windows, Linux, Mac OS X;
- возможность разработки на нижнем уровне.

CUDA является полностью бесплатной, содержит средства разработки, документацию и примеры, можно скачать в открытом доступе с сайта https://developer.nvidia.com.

Следует различать версии программных пакетов CUDA Toolkit (набор инструментов для разработчиков) и версии CUDA, которую поддерживает графический ускоритель. В документации Nvidia C Programming Guide версия, которую поддерживает графический ускоритель, называется Compute capability. Compute capability введена для определения семейства графического ускорителя с идентичной архитектурой, состоит из двух чисел – старшего и младшего, например, 7.1. Графические ускорители с одинаковым старшим числом имеют одну архитектуру ядра. Младшее число

указывает на улучшения в архитектуре ядра. Какой версии соответствует устройство, можно узнать на сайте NVIDIA https://developer.nvidia.com/cuda-gpus.

Последняя версия на сегодня ComputeCapability 7.1. Поддержка первых графических ускорителей с ComputeCapability 1.0 – 2.1 прекращена.

Таблица 2. Версии ComputeCapability в зависимости от архитектуры и чипа GPU.

Версия Compute Capability	Архитектура	GPU
1.0	Tesla	G80
1.1		G92, G94, G96, G98, G84, G86
1.2		GT218, GT216, GT215
1.3		GT200, GT200b
2.0	Fermi	GF100, GF110
2.1		GF104, GF106 GF108, GF114, GF116, GF117, GF119
3.0	Kepler	GK104, GK106, GK107
3.5		GK110, GK208
3.7		GK210
5.0	Maxwell	GM107, GM108
5.2		GM200, GM204, GM206
6.0	Pascal	GP100
6.1		GP102, GP104, GP106, GP107, GP108
7.0	Volta	GV100

По мере развития графических адаптеров, меняются и возможности CUDA (Таб. 3):

Таблица 3. Поддержка функций CUDA в разных версиях Compute capability.

Функции CUDA	Версия Compute capability

	3.0	3.2	3.5, 3.7, 5.0, 5.2	5.3	6.x	7.x
Shuffle	Да					
64-битовый сдвиг, циклический сдвиг	Нет	Да				
Динамический параллелизм	Нет		Да			
Операции над числами половинной точности с плавающей запятой	Нет			Да		
Атомарное сложение с числами двойной точности с плавающей запятой в глобальной и общей памяти	Нет				Да	
Ядра Tensor	Нет					Да

Рассмотрим подробнее данные функции:

Функции shuffle.

Начиная с архитектуры Kepler (600 и 700 серии GeForce) поддерживается новая команда – shuffle. Данная команда позволяет быстро передавать данные между потоками одного варпа. Ранее для этой операции использовалась разделяемая память – в одном потоке было необходимо записать в память данные, а в другом – считать. Теперь это происходит в один шаг – просто считываются данные соответствующего регистра другого потока варпа. Данная операция быстра и позволяет выполнять как передачу значений от одного потока другому, так и передачу значения от одного потока всем остальным.

Динамический параллелизм.

С версии CUDA 3.5 (архитектура Kepler) реализован динамический параллелизм, позволяющий запускать ядра с самого графического ускорителя без вмешательства CPU и синхронизироваться по результату. Вызовы ядер могут быть вложенными, что открывает большие возможности эффективной

реализации задач с нерегулярным и динамическим параллелизмом, а также по переводу части кода, управляющего запуском ядер, на графический ускоритель. Другими словами, потоки GPU могут динамически рождать новые потоки, позволяя GPU адаптироваться к новым данным. Сводя к минимуму пересылку данных в CPU и обратно, динамический параллелизм значительно упрощает параллельное программирование. Это также позволяет применять GPU-ускорение к более широкому спектру распространенных алгоритмов. Впрочем, от программистов тоже требуются усилия, поскольку им следует учитывать неравномерности обработки GPU и запросы из памяти. Если созданные потоки превысят возможности доступной памяти GPU, то будет проводиться обращение через шину памяти PCI Express Interface, что может вновь замедлить весь процесс.

Операции над числами половинной точности.

Число половинной точности — компьютерный формат представления чисел, занимающий в памяти половину машинного слова. В силу невысокой точности этот формат представления чисел с плавающей запятой обычно используется в видеокартах, где небольшой размер и высокая скорость работы важнее точности вычислений.

Благодаря новой архитектуре NVIDIA Pascal графический ускоритель обеспечивает максимально высокую производительность для решения задач в области высокопроизводительных вычислений (HPC, High-performance computing) и сверхмасштабирования. Производительность более 21 Терафлопс для вычислений половинной точности открывает новые возможности для приложений глубокого обучения.

Лучшая атомарность.

Операции атомарности памяти важны в параллельном программировании, позволяя параллельным потокам правильно выполнять операции чтения-изменения-записи в совместно используемых структурах.

Kepler был снабжён операциями атомичности памяти в том же виде, что и Fermi. Обе архитектуры реализовывали атомичность совместно используемой памяти применяя шаблон блокировка/изменение/снятие блокировки, который может быть затратным в случае при высокой конкуренции на обновления в определённых местоположениях в совместно используемой памяти.

Maxwell улучшил операции атомарности реализовав встроенную аппаратную поддержку для операций атомарности совместно используемой памяти для 32-битных целых и встроенную совместно используемую память 32-бит и 64-бит CAS (compare-and-swap), которая может быть использована для реализации других функций атомарности с уменьшенными накладными расходами (по сравнению с методами Fermi и Kepler, которые были реализованы программно).

GP100 строится поверх Maxwell, но также улучшает операции атомарности с применением новых свойств унифицированной памяти. Дополнительные операции атомарности в глобальной памяти были расширены и включили данные FP64.

Ядра Tensor.

Одна из наиболее важных инноваций архитектуры Volta, вернее, GPU GV100, заключается в ядрах Tensor. В случае Tesla V100 доступны 640 таких ядер.

Для процесса тренировки сетей глубокого обучения наиболее важны операции матричного умножения, именно на них ориентированы ядра Tensor в GPU GV100.

С ядрами Tensor NVIDIA смогла увеличить производительность вычислений FP32, важных для глубокого обучения, до 120 TFLOPS. То есть мы получаем прирост вычислительной производительности в 12 раз по

сравнению с GPU GP100. NVIDIA также говорит о шестикратном увеличении производительности обработки запросов.

Официально для разработки приложений с использованием CUDA поддерживаются видеокарты серий GeForce на рабочих станциях, Quadro и Tesla на высокопроизводительных машинах или серверах и другие узкоспециализованные серии. Компания обещает, что все графические ускорители их производства независимо от серии будут иметь сходную архитектуру, которая полностью поддерживает программную часть CUDA технологии.

1.3. Использование IDE и CUDA

За счет того, что программы в CUDA пишутся фактически на обычном языке C/C++, в который добавлено небольшое число новых конструкций (спецификаторы типа, встроенные переменные и типы, директива запуска ядра), написание программ с использованием технологии CUDA оказывается заметно проще, чем при использовании традиционного GPGPU (то есть использующего графические API для доступа к GPU). Кроме того, в распоряжении программиста оказывается гораздо больше контроля и возможностей по работе с GPU.

Для реализации программ под графические ускорители компания NVIDIA разработала свои расширения для языка C, компилятор nvcc для сборки таких программ, ввела в обращение новое расширение *.cu для приложений, которые содержат CUDA вызовы. К расширениям языка C относятся:

- спецификаторы для функций и переменных;
- новые встроенные типы данных;
- встроенные переменные (внутри ядра);
- директива для запуска ядра из C кода.

Для программирования на CUDA нет жестких ограничений на используемую операционную систему, можно выбрать из большого списка – Windows: 7, 8.1, 10, Server 2012 R2, 2016, Linux: RHEL 6.x,7.x, CentOS 6.x,7.x, Fedora 27, OpenSUSE Leap 42.3, SLES (SUSE Linux Enterprise Server) 12 SP3, Ubuntu 16, 17, MacOS X 10.13.

Среда разработки CUDA (CUDA Toolkit) включает:

- компилятор nvcc;
- библиотеки;
- профилировщик;
- отладчик gdb для GPU;
- CUDA runtime драйвер в комплекте стандартных драйверов NVIDIA;
- руководство по программированию;
- CUDA Developer SDK (исходный код, утилиты и документация).

  Для установки CUDA на компьютер необходимо загрузить дистрибутив по адресу http://www.nvidia.com/object/cuda_get.html и установить на свой компьютер следующие файлы: CUDA driver, CUDA Toolkit и CUDA SDK.

Версия CUDA для платформы Linux также содержит CUDA-отладчик (cuda-gdb), устанавливаемый как отдельная компонента.

После установки всех этих компонент можно будет посмотреть готовые примеры, изучить прилагаемую документацию и начать писать свои программы с использованием CUDA.

Обратите внимание, что для компиляции программ на CUDA также потребуется установленный компилятор с C/C++. В качестве такого компилятора может выступать компилятор, входящий в состав Microsoft Visual Studio, а также компиляторы mingw и cygwin.

Это связано с тем, что используемый для компиляции программ на CUDA компилятор nvcc использует внешний компилятор для компиляции частей кода, выполняемых на CPU.

При программировании графического ускорителя необходимо быть в курсе диктуемой архитектурой иерархии исполняемых потоков и памяти. Это ключ к ускорению параллельных вычислений. К сожалению, в этом месте очень легко ошибиться и убить весь эффект большого числа процессоров неэффективным копированием данных. При запуске программы на графическом ускорителе порождается множество потоков (thread). Все потоки поделены на группы одного размера – блоки потоков (block). У каждого потока и блока потоков имеются свои уникальные идентификаторы, называемые координатами. Тем самым, для разных потоков аргументы исполняемых инструкций и их последовательность могут различаться, поскольку могут зависеть от координат потока и блока потоков. Множества координат потоков и блоков потоков образуют одно-, двух- или трехмерные массивы-сетки(grid). Размеры блока потоков и массива блоков потоков задаются при запуске ядра. Во время исполнения ядра потоки одного блока могут синхронизироваться между собой посредством барьеров, а потоки разных блоков исполняются независимо. Кроме возможности барьерной синхронизации, потоки одного блока могут взаимодействовать посредством разделяемой памяти. Потоки разных блоков могут взаимодействовать лишь через глобальную память, аналог оперативной памяти в компьютере. Кроме разделяемой и глобальной, есть константная и текстурная памяти, которые доступны из потоков только на чтение. На аппаратном уровне глобальная, константная, текстурная и разделяемая памяти оптимизированы под различные варианты использования.

Для эффективного программирования на CUDA необходимо учесть следующие факторы, существенно влияющие на производительность.

Количество вычислительных потоков.

От размера сетки блоков потоков и самого блока потоков зависит степень загруженности планировщиков графического ускорителя. Чем больше варпов планирует планировщик, тем больше у него возможностей скрыть задержки, связанные с обращением в глобальную память. Кроме того, достаточно большие размеры сетки и блока обеспечат эффективное масштабирование на новых графических ускорителях без переписывания и перекомпилирования программы. Обычно рекомендуется порождать порядка ста потоков за один запуск ядра.

Равномерность загрузки вычислительных потоков

Неравномерность загрузки – очень частое явление, например, вычисления значений во внутренних и граничных узлах сетки явной разностной схемы обычно различаются. Однако неравномерность загрузки потоков может привести к существенной деградации производительности. Во-первых, это связано с тем, что блок потоков завершается и освобождает мультипроцессор, когда все его потоки завершили исполнение. Во-вторых, запуск ядра завершается, когда все блоки потоков завершили исполнение.

Преобладание вычислений по отношению к загрузкам данных.

Для повышения производительности необходимо максимизировать количество производимых вычислений на единицу загружаемых данных из глобальной памяти. Минимизировать количество доступов в память помогает использование разделяемой памяти и кэшей, за счет переиспользования одним и тем же или другими потоками близко расположенных данных. Локальность загружаемых данных.

Этот фактор напрямую следует из предыдущего. Увеличение локальности загружаемых данных на уровне варпа позволяет уменьшить количество и объем транзакций с памятью и повысить эффективность работы кэшей. Увеличение локальности загружаемых данных на уровне блока потоков

позволяет увеличить эффективность использования разделяемой памяти и работы кэшей.

Деление варпов на условных переходах.

Когда разные потоки одного варпа разбиваются по разным ветвям условного перехода, время исполнения условного перехода складывается из времен исполнения его ветвей. Т.е. частое деление варпов по ветвям приводит к деградации производительности; ее степень зависит от числа делений варпов и от размера ветвей перехода.

1.4. Примерные варианты организации проведения лабораторных работ.

Целью лабораторных работ по параллельному программированию с использование графического ускорителя является практическое освоение учащимися содержания и методологии предмета с использованием компьютера. В ходе работы учащиеся должны освоить написание кода программы, компиляцию, запуск и отладку.

Для организации проведения лабораторных работ по программированию необходимы компьютеры с установленным на них всем необходимым программным обеспечением: операционной системой, средой разработки.

Для Linux-подобных операционных систем, использующих графический интерфейс, и Mac OS NVIDIA предоставляет среду разработки Nsight Eclipse Edition (рис. 6), в которой и проходит разработка программы, компиляция и отладка.

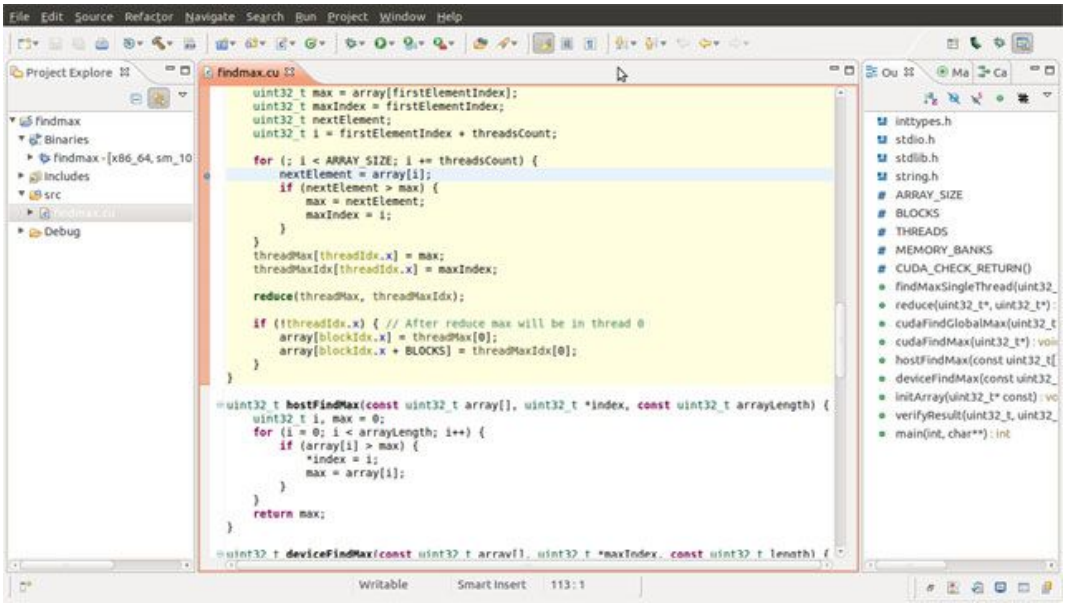


Рисунок 6. Интерфейс Nsight Eclipse Edition на Linux и MacOS

В операционных системах семейства Windows разработана надстройка для Visiual Studio – NVIDIA Nsight Visual Studio Edition (рис. 7), где так же наглядно и доступно происходит процесс обучения программированию.

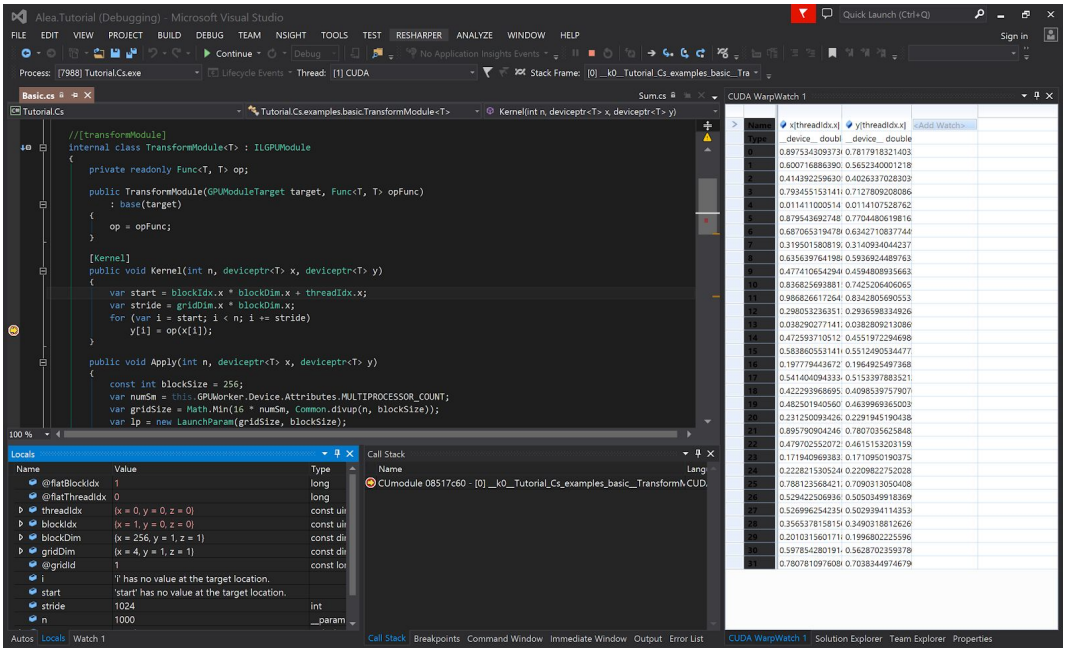


Рисунок 7. Интерфейс Visual Studio на Windows

NVIDIA Nsight предоставляет широкие возможности для обучения программированию:

- простое и быстрое начало работы, благодаря новым шаблонам и интеграции образцов CUDA SDK;

- простая навигация, выделение кода CUDA, автоматические подсказки и подстановки;
- одновременная отладка кода для GPU и CPU
- объединенный мониторинг CPU и GPU во время работы приложений позволяет определить узкие места производительности.

Таким образом, если у преподавателя имеется в распоряжении оборудованный класс для проведения лабораторных работ по параллельному программированию с использование графического ускорителя, то никаких технических проблем и сложностей в процессе обучения и проведения работы не предвидится. Преподаватель выбирает удобную для обучения операционную систему и среду разработки, а каждый учащийся работает независимо от других в удобном интерфейсе среды разработки, вникает в принципы параллельного программирования.

В случае, если на рабочих станциях нет графического ускорителя, то он может быть установлен на сервере, к которому организован доступ посредством текстового терминала, написание кода заключается в редактировании файла в обычном текстовом редакторе, запущенном в SSH сеансе, без каких-либо подсказок, подстановок. После написания кода его сборку на сервере осуществляет компилятор nvcc, поставляемый в комплекте CUDA Toolkit.

Ни о каком удобстве и наглядности в таком случае говорить не приходится.

Для написания кода можно воспользоваться предоставляемой средой программирования, установленной на рабочей станции, но компиляция и отладка все равно возможна только на сервере. Много времени затрачивается на отладку – сложная процедура в условиях командной строки, необходимо знание ключей и тонкостей работы компилятора и отладчика. К тому же

запуск нескольких программ на одном устройстве может вызвать ошибки, или имеет ограничение по количеству одновременно запущенных задач, что так же осложняет процесс отладки и обучения в целом.

2. Организация терминального сервера для проведения лабораторных работ.

2.1. Требования к аппаратной части компьютера для проведения лабораторных работ по параллельному программированию с использованием графического ускорителя.

Исходя из требований для работы современного программного обеспечения и актуальности аппаратного обеспечения, минимальной конфигурацией рабочей станции для проведения лабораторных работ по программированию с использованием графического ускорителя рекомендуется:

- процессор Intel Celeron G3930, частота 3000MHz (со встроенными графическим ускорителем);
- совместимая материнская плата;
- 4Gb оперативной памяти;
- 500Gb дискового пространства;
- видеокарта NVIDIA GT 1030;
- блок питания от 400W;
- другие периферийные устройства (монитор, клавиатура, мышь и т.д).

Конфигураций в зависимости от производительности оборудования, производителей и цен очень много, но в рамках изучения программирования с использованием графического ускорителя выбор существенно не повлияет на качество обучения.

В настройка BIOS обязательно включить поддержку Intel VT и включить использование встроенного в процессор графического ускорителя по умолчанию.

Также необходимо наличие на рабочей станции операционной системы из рекомендованных: Windows, Linux с графическим интерфейсом или Mac OS. Установленные соответствующие для нее драйвера оборудования, особенно для видеокарты.

Для Windows необходимо установить Visual Studio 2012 версии или новее. И, наконец, NVIDIA CUDA Toolkit.

Для Linux достаточно установить только NVIDIA CUDA Toolkit.

Для Mac OS, кроме NVIDIA CUDA Toolkit, понадобится компилятор Clang, устанавливаемый посредством среды разработки приложений Xcode.

Дополнительным положительным фактором является организация локальной сети, включающую в себя компьютеры для проведения лабораторных работ, с выходом в интернет для оперативного получения справочного материала.

2.2. Существующие технологии организации терминального доступа.

Многотерминальные системы – сетевые технологии, благодаря которым большое количество пользователей могут изолированно работать на одной машине, используя несколько компьютеров, включенных в одну сеть так, как если бы они работали непосредственно на своем компьютере. Для каждого пользователя, сразу после авторизации, на сервере создаётся уникальный сеанс, в котором он и продолжает работать.

Используется принцип "клиент-сервер", где в качестве сервера терминалов выступает техника большой вычислительной мощи, а клиентом может быть компьютер с минимальной аппаратной частью, в зависимости от предназначения системы.

Этот подход больше не привязывает пользователя к своему физическому рабочему месту в офисе, что позволяет пользователю работать с его привычными приложениями и данными с любого устройства и из любого места. Устройство может быть, к примеру, – планшет, телефон, тонкий клиент и т д.

Клиенты в терминальной системе бывают "тонкие" и "толстые". Для работы тонкого клиента необходим терминальный сервер, с него происходит загрузка, и на нем же производятся все вычисления. Толстый клиент, напротив, производит обработку информации независимо от сервера, используя его в основном для хранения данных.

Виртуализация – многозначное понятие – создание аппаратных и/или программных, а также иных систем, процессов, отличающихся от реальных. Так, на базе совокупности множества реальных процессоров с помощью программно-аппаратных средств обеспечивают возможность обращаться к ним из программы как к одному виртуальному процессору с большой производительностью, что упрощает программирование.

Основная идея виртуализации — использование одной физической машины для выполнения нескольких виртуальных. Это немного напоминает разбиение одного физического жесткого диска на несколько логических. Виртуализация использует физические устройства машины и предоставляет вместо них виртуальные средства. Например, жесткий диск виртуальной машины — это просто файл в файловой системе физической машины. Кроме того, в виртуальной машине есть сетевая карта, видеокарта, порты для подключения периферийных устройств, процессор и память.

Несмотря на то, что виртуализация в совместимых с процессором x86 информационных средах получила бурное развитие только в текущем десятилетии, сама эта технология появилась более 40 лет назад.

Виртуализация включает в себя различные методики для обеспечения работы на одной физической хост-системе нескольких безопасных и изолированных разделов (совместно и одновременно использующих ресурсы). Эти подходы отличаются по плотности разделов (количеству одновременных разделов), масштабируемости, производительности и разнообразию поддерживаемых операционных систем.

При реализации гипервизора – интерфейса между виртуальными машинами и виртуализированными системными ресурсами используются три возможных метода:

- полная виртуализация;
- паравиртуализация;
- виртуализация приложений;
- виртуализация представлений;
- виртуализация на уровне ядра операционной системы;
- собственная виртуализация.

В настоящее время наиболее широко признанной является парадигма полной виртуализации. В этой модели операционная система не знает, что она функционирует на виртуализированной платформе, а между виртуальными машинами (гостями) и аппаратным обеспечением инсталлируется гипервизор (hypervisor), известный также как монитор виртуальной машины (Virtual Machine Monitor, VMM).

Гипервизор отвечает за создание и изоляцию виртуальной машины и сохранение ее состояния, а также за организацию доступа к системным ресурсам. Схема гипервизора зависит от архитектуры конкретного процессора; несмотря на то, что гипервизор позволяет работать внутри виртуальной машины немодифицированным операционным системам, вы обычно ограничены теми операционными системами, которые могут работать на данном физическом системном процессоре.

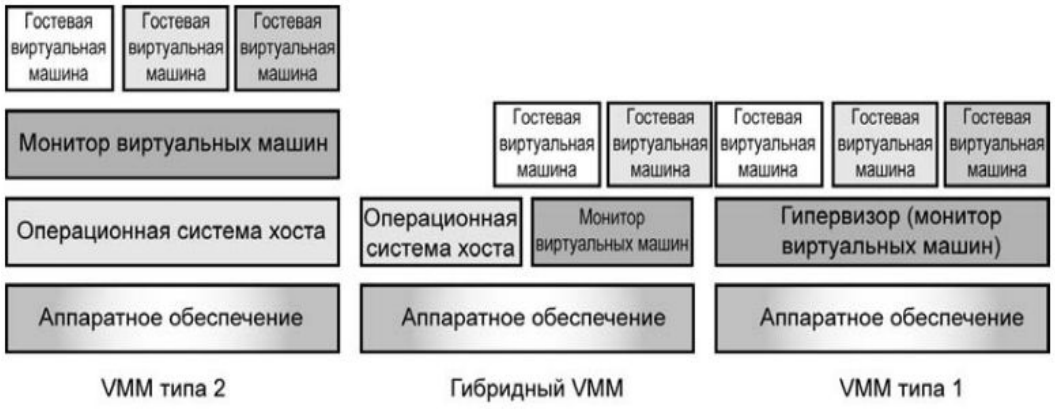


Рис. 7. Типы гипервизора виртуальных машин

На рис. 7 показаны три разные реализации гипервизора: тип 2, гибридная модель и тип 1. гипервизор типа 2 работает над операционной системой хоста (как Java VM). В гибридной модели гипервизор работает как равноправный с операционной системой модуль. Так реализован сервер Virtual Server 2005 R2. В отличие от него гипервизор типа 1 (гипервизор) работает непосредственно на аппаратном обеспечении (ниже разделов виртуальных машин).

В смысле производительности гипервизор (VMM типа 1) обычно способен дать более высокий уровень эффективности и, следовательно, большую плотность виртуальных машин. Другие типы гипервизора при доступе к ресурсам зависят от операционной системы хоста, что приводит к более дорогим переключениям контекста и более значительным потерям производительности.

Такие гипервизоры называются автономными, потому что они управляют аппаратным обеспечением. Гипервизор обеспечивает уровень эмуляции для всех аппаратных устройств узла. Гостевая операционная система не модифицируется. Гости выполняют прямые запросы к виртуализированному аппаратному обеспечению, и любые привилегированные инструкции, которые гостевые ядра пытаются выполнить, обрабатываются гипервизором.

Полная виртуализация – наиболее безопасный вид виртуализации, потому что гостевые операционные системы изолированы от базового аппаратного

обеспечения. Кроме того, не требуется вносить никаких изменений в ядра операционных систем, и гости могут выбирать разные базовые архитектуры. Если имеется программное обеспечение для виртуализации, то гость может функционировать на любой архитектуре процессоров.

Такая технология применяется, в частности, в VMware Workstation, VMware Server, Parallels Desktop, Parallels Server, MS Virtual PC, MS Virtual Server. К достоинствам данного подхода можно отнести относительную простоту реализации, универсальность и надежность решения, так как все функции управления берет на себя операционная система хоста. Она получает ресурсы таким же образом, как и гостевые системы. В таком типе виртуализации производительность при работе с виртуальными машинами приближается к 100 %, т.е. к тому уровню, который достигался, если бы операционная система на виртуальной машине выполнялась на физическом оборудовании.

Паравиртуализация – это технология, которая позволяет нескольким операционным системам согласованно функционировать на одной машине. Однако ядра каждой операционной системы должны быть модифицированы, чтобы поддерживать “гипервызовы”, т.е. перевод определенных инструкций, предназначенных для центрального процессора. Модификация ядра гостевой ОС выполняется таким образом, что в нее включается новый набор API, через который она может напрямую работать с аппаратурой, не конфликтуя с другими виртуальными машинами. При этом нет необходимости задействовать полноценную операционную систему в качестве хостовой, функции которой в данном случае исполняет гипервизор. Приложения, действующие в адресном пространстве пользователя, не требуют модификации и выполняются на виртуальных машинах естественным образом. Гипервизор в паравиртуализации используется точно так же, как и в полной виртуализации.

Уровень трансляции в паравиртуализированной системе связан с меньшими накладными расходами ресурсов, поэтому паравиртуализация ведет к номинальному повышению производительности. Однако необходимость модифицировать гостевые операционные системы резко снижает эффект и является основной причиной того, почему технология паравиртуализации слабо поддерживается за пределами сообществ с открытым исходным кодом.

Среда паравиртуализации похожа на полностью виртуализированную систему, в которой гостевые операционные системы взаимодействуют с гипервизором посредством определенного интерфейса, но в данной системе первый гость является привилегированным. Именно этот вариант является сегодня наиболее актуальным направлением развития серверных технологий виртуализации и применяется в VMware ESX Server, Xen (и решениях других поставщиков на базе этой технологии), Microsoft Hyper-V. Достоинство данной технологии — отсутствие потребности в хостовой ОС. Однако паравиртуализация требует изменения кода ядра, а подобное изменение возможно лишь в случае, если гостевые операционные системы имеют открытые исходные коды, которые можно модифицировать согласно лицензии.

Виртуализация приложений – это технология, которая направлена на разделение и изоляцию приложений на стороне клиента (работающих под местной операционной системой). Приложения изолируются в виртуальной среде, находящейся между операционной системой и стеком приложений. Виртуальная среда загружается до приложения, изолирует его от других приложений и операционной системы, а также предотвращает модификацию приложением локальных ресурсов (таких, как файлы и настройки реестра). Приложения могут читать информацию из локального системного реестра и файлов, но пригодные для записи версии этих ресурсов поддерживаются

внутри виртуальной среды. Фактически приложение не обязательно даже инсталлировать на настольный компьютер; код можно получать динамически (по мере необходимости) и кэшировать в виртуальной среде.

Виртуализация на уровне приложения дает несколько преимуществ. Главные из них:

- повышенная стабильность локального настольного компьютера;
- простое удаление приложений без таких изменений локальной среды, которые могли бы негативно сказаться на других приложениях;
- бесконфликтное одновременное выполнение нескольких экземпляров одного и того же приложения.

Виртуализация представлений – это тип виртуализации, при котором приложение выполняется на удалённом сервере, а пользователю на экран выводятся результаты его выполнения. Классический терминальный доступ, предоставляющий серверную операционную систему, для нескольких пользователей в конкурентном режиме. Пользователь подключается к удаленному серверу, и работает с собственным пользовательским сеансов, который изолирован от сеансов других пользователей. В то же время общие ресурсы сервера остаются не абстрагированными. Благодаря этому данные могут быть централизованы и надежно сохранены на центральном сервере, а не на нескольких настольных компьютерах. За счет этого повышается безопасность, поскольку информация не распределяется между различными системами. Кроме того, можно значительно сократить затраты на управление приложениями. Вместо обновления каждого приложения на каждом отдельно взятом компьютере необходимо изменить лишь одну общую копию на сервере.

Виртуализация представлений также позволяет использовать более простые образы операционных систем и специализированные настольные

компьютеры, называемые «тонкими» клиентами, что способствует снижению расходов на управление. Проблема несовместимости приложений и установленной на компьютере операционной системы решена за счет виртуализации настольных систем, это позволяет просто запустить приложение на центральном сервере, а затем сделать его доступным для клиентских компьютеров с любой операционной системой. Виртуализация позволяет повысить производительность. Одним из способов повышения производительности является запуск всего приложения (и клиентской, и серверной части) на компьютере с высокоскоростным подключением к данным, а затем с помощью виртуализации представлений это приложение можно сделать доступным для пользователей.

Недостатков у таких систем два: во-первых – необходимость покупки более мощных серверов (хотя это может быть дешевле, чем множество клиентских рабочих станций с оборудованием, достаточным для запуска приложений локально), во-вторых – появление единой точки отказа в виде терминального сервера. Эта проблема решается за счет использования кластеров, или ферм серверов, но это приводит к еще большему удорожанию системы.

Виртуализация на уровне ядра операционной системы подразумевает использование одного ядра хостовой операционной системы для создания независимых параллельно работающих операционных сред. Для гостевого приложения создается только собственное сетевое и аппаратное окружение. Такой вариант используется в Virtuozzo (для Linux и Windows), OpenVZ (бесплатный вариант Virtuozzo), IBM AIX, HP-UX, Free BSD, Solaris Containers. Достоинства — высокая эффективность использования аппаратных ресурсов, низкие накладные технические расходы, хорошая управляемость, минимизация расходов на приобретение лицензий. Недостатки — реализация только однородных вычислительных сред.

Наличие всего одной операционной системы позволяет устанавливать обновления одновременно на все виртуальные среды, размещенные на одном сервере. Однако на каждый сервер устанавливается только одна ОС. Таким образом, если, например, клиентам требуется предоставить и Windows, и Linux, то понадобится как минимум 2 сервера.

Собственная виртуализация зависит от архитектуры виртуализируемого процессора (например, такой, как в сериях процессоров AMD-V и Intel VT). Эти процессоры имеют в своей аппаратной части новые режимы выполнения, инструкции и структуры данных, которые предназначены для уменьшения сложности VMM.

При собственной виртуализации монитору VMM больше не требуется поддерживать в программном обеспечении характеристики ресурсов виртуальной машины и ее состояние. Точно так же, как и в случае полной виртуализации, внутри виртуальных машин операционные системы могут выполняться без их модификации. Hyper-V использует этот метод для работы устаревших операционных систем.

Такой тип реализации имеет много потенциальных преимуществ — от упрощения архитектуры VMM и до существенного повышения производительности (в результате снижения накладных расходов программного обеспечения). При помощи снижения накладных расходов на виртуализацию можно достичь более высокой плотности разделов в системе.

Виртуальная машина – программная и/или аппаратная система, эмулирующая аппаратное обеспечение или компьютер в целом. Эмулирующая система (платформа) называется главной или host (хост, хозяин), эмулируемая – target (целевая или гостевая) платформа. Также виртуальная машина может создавать среды, изолирующие друг от друга программы и даже операционные системы. Виртуальный жесткий диск - это файл в файловой системе жесткого диска базового компьютера, который для

гостевой ОС определяется как реальный физический диск. Виртуальная машина исполняет машинно-независимый код или код реального процессора, также может эмулировать как работу отдельных аппаратных частей компьютера, так и целого компьютера (включая BIOS, оперативную память, жёсткий диск и другие периферийные устройства). В последнем случае на виртуальную машину, как и на реальный компьютер, можно устанавливать операционные системы. На одном компьютере может функционировать несколько виртуальных машин.

На физическом компьютере после нажатия кнопки включения питания или сброса центральный процессор находит команду по адресу (номер ячейки памяти), находящемуся в специальном регистре – счетчике команд. Происходит исполнение команд и изменение рабочего пространства – содержимого оперативной, внешней памяти и регистров, являющихся разновидностью памяти. Далее содержимое счетчика команд наращивается в зависимости от длины команды с операндами на определенное число так, чтобы процессор считал следующую команду. Условные и безусловные переходы, циклы, вызовы подпрограмм происходят путем перезаписи содержимого счетчика команд. В виртуальной машине те же команды и в том же порядке считывает не процессор, а программа виртуальной машины, и только затем для выполнения отправляет их на реальный процессор, далее результаты записываются, но не в реальные регистры, а в область оперативной памяти, выделенной для виртуальной машины. При необходимости записи/считывания информации в реальные регистры (например, при переброске портов или устройств (переключение на виртуальную машину)) это делается только после тщательного контроля безопасности этих операций. Таким образом, программы в виртуальной машине вообще не имеют прямого доступа к ресурсам реальной машины (только через посредника в виде программы, реализующей виртуальную

машину). Но и хост – не имеет доступа к области памяти, выделенной для виртуальной машины. Хост только видит, что память выделена для чего-то, но доступ к этой памяти закрыт. В результате в реальном компьютере могут одновременно и совершенно независимо функционировать несколько совершенно разных операционных систем на базе одной или разных программ виртуальных машин со своими приложениями, в том числе хост. Виртуальная машина является платформенно-независимой, т.е. должна работать на любом компьютере, в том числе с разным аппаратным обеспечением (системная плата, процессор и т.д.).

Технология виртуализации является той основой, которая может помочь предприятию ускорить достижение следующих целей:

- снижение капитальных и эксплуатационных затрат на информационные технологии;
- реализация упрощенной, динамичной инфраструктуры;
- повышение готовности вычислительных ресурсов;
- уменьшение времени, необходимого для подготовки или предоставления новых услуг;
- уменьшение сложности управления;
- использование клиентской операционной системы в терминальном режиме;
- наличие административных прав в терминальной среде для пользователя;
- возможность сохранения состояния рабочей среды и последующего возобновления работы;
- повышенную изоляцию пользовательских рабочих сред друг от друга, решающую вопросы нестабильности используемых приложений или возможные пики нагрузки на аппаратные ресурсы сервера, влияющие на других пользователей;

- перенос рабочей среды на локальный компьютер путем копирования виртуального жесткого диска.

Для подключения к гипервизору применяются самые разнообразные устройства: компьютеры, планшеты, смартфоны и т.д., самым экономически выигрышным вариантом на сегодняшний день является использование тонких клиентов вместо традиционных рабочих станций. Тонкие клиенты поддерживают практически все известные на рынке инфраструктуры виртуализации десктопов (Microsoft, VMware, Citrix), а также возможность централизованного управления с помощью соответствующего программного обеспечения. Под поддержкой подразумевается наличие соответствующего протокола на стороне тонкого клиента.

Технология виртуализации является дальнейшим развитием терминальных решений. В отличие от классического подхода, при использовании виртуализации ресурсы сервера разделяются между пользователями не за счет создания сессий в среде одной операционной системы, а с помощью запуска виртуальных машин, интерфейс которых передается пользователям системы на рабочие станции и тонкие клиенты.

2.3. Терминальный доступ с возможностью программирования с использованием графического ускорителя

На сегодняшний день первенство в области виртуализации принадлежит компаниям Microsoft, VMware, Citrix, существуют так же менее популярные open source проекты, наиболее известный из них KVM/QEMU. Рассмотрим в целом технологии виртуализации графического ускорителя.

В гипервизоре Hyper-V от Microsoft, используются технологии DDA (Discrete Device Assignment, выделение дискретного устройства) и RemoteFX vGPU, которые позволяют использовать ресурсы видеокарты в виртуальной машине.

Технология DDA является решением аппаратного проброса, которое предоставляет наилучшую производительность, предоставляя конкретной виртуальной машины полный доступ к определённому GPU с применением естественного драйвера (драйвера самого производителя оборудования). Пользователь виртуальной машины получает доступ ко всем возможностям своего устройства, а также к самому драйверу данного устройства. Это означает, что все свойства и возможности работы данного устройства в некоторой виртуальной машине зеркально отражают работу того же самого устройства в голом железе. Это аппаратное перенаправление, поэтому работает только в режиме: один графический ускоритель – одна виртуальная машина, к тому же, чтобы организовать такой доступ нужна видеокарта из семейств Quadro или Tesla. GeForce в таком режиме работать не будет из-за ограничений, введенных NVIDIA.

Другой режим виртуализации графического ускорителя – RemoteFX. Технология RemoteFX появилась в Windows Server 2008 R2 и была улучшена в Windows Server 2012. Она значительно повышает графическую производительность для сеансов удаленных рабочих столов. Обычно при отображении графики в сеансе удаленного рабочего стола (воспроизведение полноценного видеофильма или вывод приложений со значительным графическим компонентом) центральный процессор компьютера должен поддерживать не только работу в сеансе удаленного рабочего стола, но и отображение графики. Поэтому приложения со значительным графическим компонентом существенно понижают производительность серверов удаленных рабочих столов, вплоть до того, что пользователи отказываются выполнять приложения в системах удаленных рабочих столов.

Однако технология RemoteFX переносит обработку видеоданных для их отображения с центрального процессора (CPU) удаленного рабочего стола на графический процессор (GPU) на отдельном видеоадаптере сервера RDS.

Перенос графических процессов на GPU позволяет организациям предоставлять графическую поддержку для производственных приложений, что очень важно в наши дни для работы с насыщенным контентом вроде потокового видео, графических данных и т.п. С выходом нового программного клиента RDP версии 8, в который вошел RemoteFX второго поколения, фрейм-буфер анализируется и для различных участков экрана (графика, статичные изображения, анимация и видео) выбираются разные кодеки, участки кодируются и по отдельности отдаются на «тонкий клиент». Для изображений применяется прогрессивный рендеринг, то есть изображение в низком разрешении клиент увидит мгновенно, а детали догрузятся так быстро, как это позволит пропускная способность канала. RemoteFX научился подстраиваться под особенности канала связи, используя адаптивные кодеки. Он позволяет разделить ресурсы графического ускорителя на несколько виртуальных машин, но не дает доступа к возможностям CUDA графического ускорителя, программы при таком режиме не будут компилироваться и выполняться.

Гипервизор от Citrix XenServer использует 3 типа виртуализации графического ускорителя:

vSGA (Virtual Shared Graphics Acceleration) – это самый простой и эффективный способ использовать аппаратное ускорение для 3D-графики в виртуальных машинах. Он ограничивает системного администратора только объемом видеопамяти, которой, однако, у адаптера тоже не бесконечно много. Этот режим позволяет использовать один графический ускоритель для нескольких виртуальных машин, но у него крайне низкая производительность и нет поддержки CUDA технологии.

vDGA (Virtual Dedicated Graphics Acceleration) – технология графического ускорения vDGA обеспечивает высочайшую производительность за счет поддержки встроенных драйверов NVIDIA, что необходимо пользователям,

работающим на выделенных рабочих станциях. Однако данная технология не обеспечивает экономически эффективной масштабируемости. При использовании технологии vDGA к одному графическому процессору NVIDIA может подключиться лишь один пользователь. Такой подход целесообразно применять в тех случаях, когда производительность важнее экономических преимуществ совместного использования графического процессора. В отличие от предыдущих двух режимов используется драйвер от производителя ускорителя (на сегодня только карты nVidia). Поддерживаются DirectX, OpenGL, CUDA.

В данном режиме графическое ядро целиком выделяется в виртуальную машину и не разделяется между другими виртуальными машинами. Их максимальное количество в режиме vDGA определяется количеством графических ядер в установленных ускорителях.

NVIDIA GRID vGPU (Virtual Graphics Processing Unit) – проброс части ресурсов GPU в виртуальную машину с использованием специального драйвера NVIDIA. Это самый интересный и долгожданный вариант реализации, так как vGPU обеспечивает высокую плотность виртуальных машин с использованием аппаратного 3D-ускорения и поддержкой CUDA.

NVIDIA GRID vGPU — это технология ускорения графики, которая обеспечивает разделение ресурсов одного графического процессора между несколькими виртуальными компьютерами. Установка карт NVIDIA GRID на в среде виртуализированных компьютеров обеспечивает высочайшую производительность графических возможностей приложений по сравнению с возможностями сред без аппаратного ускорения.

Эта технология идеально подходит для сценариев использования, связанных с большим объемом графических данных, например, в работе дизайнеров, архитекторов, инженеров, в области высшего образования, в нефтегазовой промышленности, в здравоохранении, а также в работе

привилегированных пользователей, которым требуется доступ к интерфейсам двухмерных и трехмерных графических данных. Благодаря технологии GRID vGPU графические команды каждой виртуальной машины передаются непосредственно на графический ЦП без трансляции гипервизором. Графический ЦП использует временные интервалы для обеспечения производительности при совместном использовании виртуализированной графики.

Технология заключается в том, что встроенный в гипервизор продукт NVIDIA Virtualization Software осуществляет управление виртуальными графическими ускорителями, которые прикрепляются к виртуальным машинам.

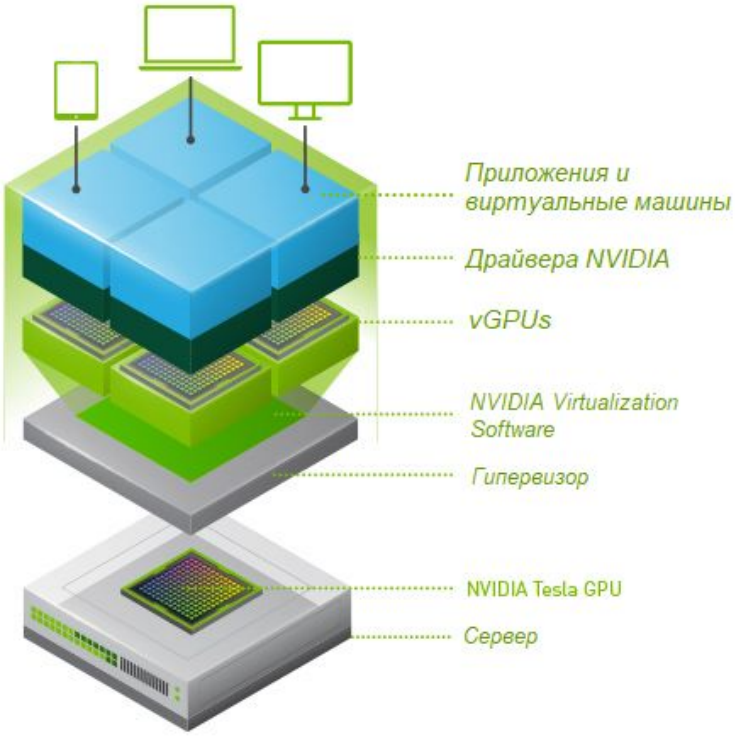


Рис. 8. Структура NVIDIA GRID

В операционной системе дополнительно устанавливается GRID Software Driver. Для управления виртуальными графическими ускорителями выбирается профиль – предустановленные настройки в зависимости от модели физического графического ускорителя.

В таблице 3 представлены возможные варианты профилей для количества пользователей от 16 на 1 графический ускоритель.

Таблица 3. Профили NVIDIA GRID

Физических GPU	Виртуальный GPU	Память, Мбайт	Максимум vGPU на GPU	Максимум vGPU на устройство
Tesla M60				
2	M60-1Q	1024	8	16
2	M60-0Q	512	16	32
2	M60-1B	1024	8	16
2	M60-0B	512	16	32
2	M60-1A	1024	8	16
Tesla M10				
4	M10-2Q	2048	4	16
4	M10-1Q	1024	8	32
4	M10-0Q	512	16	64
4	M10-1B	1024	8	32
4	M10-0B	512	16	64
4	M10-2A	2048	4	16
4	M10-1A	1024	8	32
Tesla M6				
1	M6-0Q	512	16	16
1	M6-0B	512	16	16

Полный список профилей доступен по адресу http://images.nvidia.com/content/grid/pdf/GRID-vGPU-User-Guide.pdf

Благодаря сертификации ведущими разработчиками программного обеспечения, технология NVIDIA GRID vGPU позволяет организовывать

доступ к GPU нескольким виртуальным машинам и запускать полноценную сборку драйверов NVIDIA, что означает 100% совместимость приложений.

В высших учебных заведениях технология NVIDIA GRID vGPU позволяет создавать масштабируемую лабораторную среду.

Таким образом, исходя из вышеперечисленных технологий организации терминального доступа можно сделать выбор в сторону виртуализации серверов с поддержкой виртуализации vGPU, как единственный подходящий способ для организации лабораторных работ по параллельному программированию с использованием графического ускорителя, только такой вариант может предоставить полный доступ к аппаратным средствам сервера, а именно, к графическому ускорителю.

Рассмотрим рекомендованные аппаратные требования для гипервизоров XenServer и VM Ware ESXi.

Для работы гипервизора XenServer разработчик рекомендует использовать:

- один и более 2 GHz процессор, с поддержкой технологий Intel VT или AMD-V. Материнская плата должна иметь поддержку этих технологий, которая включается в BIOS;
- 4 или более гигабайта оперативной памяти;
- наличие 60 гигабайт дискового пространства;
- наличие одного или более сетевого адаптера 1000 М/бит или быстрее.

Для VM Ware ESXi:

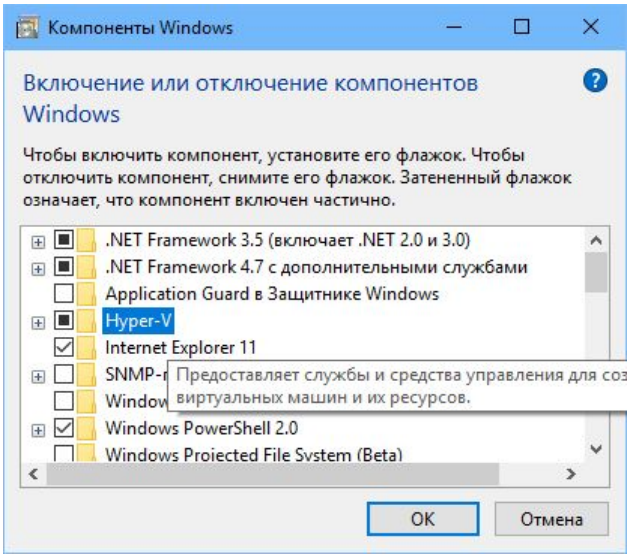
- один 64-bit двухъядерный процессор, с тактовой частотой 2.0 GHz или выше, с поддержкой технологий Intel VT или AMD-V. Материнская плата должна иметь поддержку этих технологий, которая включается в BIOS. Itanium (IA64) процессор не поддерживается.

- 4 и более гигабайта оперативной памяти;
- 4 гигабайта дискового пространства, если база, хранящая данные сервера, будет находиться на этом же сервер, то больше;
- наличие одного или более сетевого адаптера 1000 М/бит или быстрее.

Для использования NVIDIA GRID необходимы графические ускорители, поддерживающие данную технологию: видеокарты Tesla M6, M10, M60.

Для апробации технологий и написания рекомендации по созданию инфраструктуры для выполнения лабораторных работ в качестве тестового стенда была выбрана рабочая станция с процессором i7-4790k, 16GB RAM, видеокарта GeForce GTX 770.

Гипервизор от Microsoft является встроенным продуктом для операционной системы Windows 10, для его установки достаточно поставить одну галочку:



После перезагрузки получаем готовый к использованию гипервизор. Но, как и ожидалось, в режиме DDA видеокарта GeForce не заработала. Оборудование определилось, драйвера установились, но работа графического ускорителя стопорилась с ошибкой – ограничение NVIDIA в действии.

Второй попыткой стал гипервизор от Xen, но уже на этапе установки он не обнаружил сетевую карту (необходимое устройство для работы гипервизора), при дальнейшей установке с установленной дополнительной сетевой картой, проблем не возникло, но в процессе создания и настройки перенаправления графического ускорителя возникли проблемы, гипервизор не «отдавал» видеокарту. Возможно, связано с тем же ограничением, что и у Hyper-V.

При установке ESXi проблем не возникло, к тому же был позитивной опыт работы с продуктами данного разработчика. Подтвердилась статистика использования гипервизоров.

В дальнейшем ESXi был настроен, создана виртуальная машина, но vDGA и vGPU так же не доступны для GeForce семейства графического ускорителя.

С таблицы совместимости графических ускорителей в разных режимах можно ознакомиться в базе знаний VMware https://www.vmware.com/resources/compatibility/search.php.

По совместимости vDGA и vGPU: https://www.vmware.com/resources/compatibility/pdf/vi_vdga_guide.pdf https://www.vmware.com/resources/compatibility/pdf/vi_sptg_guide.pdf

2.4. Приемы использования терминального сервера для выполнения лабораторных работ по параллельному программированию с использованием графического ускорителя.

Организация и проведение лабораторных работ по параллельному программированию используя терминальный сервер виртуализации рабочих станций, ничем не отличается от обычного проведения лабораторной работы. Благодаря виртуализации, учащиеся получают идентичный набор средств для выполнения заданий, как если бы они работали за обычной физической машиной, оборудованной согласно требованиям. Сервер виртуализации

организовывает работу таким образом, что независимо от физической машины, каждый учащийся подключается к «своей» виртуальной машине и работает со своим рабочим пространством, не ограничивая в ресурсах других.

Еще одним существенным плюсом является вариативность проведения лабораторных работ. Предположим, необходимо провести лабораторную работу по программированию на операционной системе Windows, но хотелось бы показать, как те же операции выполняются в Linux, сравнить производительность программ в разных средах выполнения. В условиях работы с физическими рабочими станциями это либо занимает очень много времени на подготовку: на каждый компьютер поставить 2 операционные системы, на каждой свой набор программного обеспечения, каждую настроить для работы, либо страдает наглядность – разбивать класс на две части, одна с Window, другая с Linux.

При схеме работы на гипервизоре с виртуальными машинами, такая лабораторная работа организовывается путем создания «золотого образа» – это виртуальная машина, с которой происходит загрузка других машин, т.е. загружаемая с «золотого образа» машина имеет ту же операционную систему и набор программного обеспечения, что и «золотой образ», но другие данные (рабочий стол пользователя, документы, запускаемые программы) она имеет свои. Создав всего два «золотого образа», один с Windows, другой с Linux, мы получим работающий класс на двух разных операционных системах. Причем открыть виртуальные машины можно как по очереди, так одновременно, если позволят характеристики сервера, что удобно и наглядно при сравнении производительности программ в разных средах.

Благодаря быстрому редактированию аппаратных средств виртуальной машины, можно «на лету» изменять количество процессоров, размер оперативной памяти, видеопамяти, что так же дает возможность

анализировать зависимость производительности программы от оборудования.

Использование виртуализации делает возможным проведение дистанционных лабораторных работ. При наличии доступа к серверу-гипервизору через интернет, появляется возможность организовать работу учащихся с их виртуальными машинами из любого места и практически с любого устройства.

2.5. Рекомендации по созданию инфраструктуры для выполнения лабораторных работ по параллельному программированию с использованием графического ускорителя.

Основой инфраструктуры для проведения лабораторных работ является сервер, на котором установлен гипервизор. В качестве рекомендаций по техническому оборудованию из расчета на 16 виртуальных машин, для сервера я рекомендую использовать процессор Intel семейства Xeon от восьми ядер, совместимую материнскую плату поддерживаемую технологию Intel VT, от 16 GB оперативной памяти, 1 TB дискового пространства и графический ускоритель Tesla M10. Также обязательным условием является организация локальной сети и включении сервера в нее.

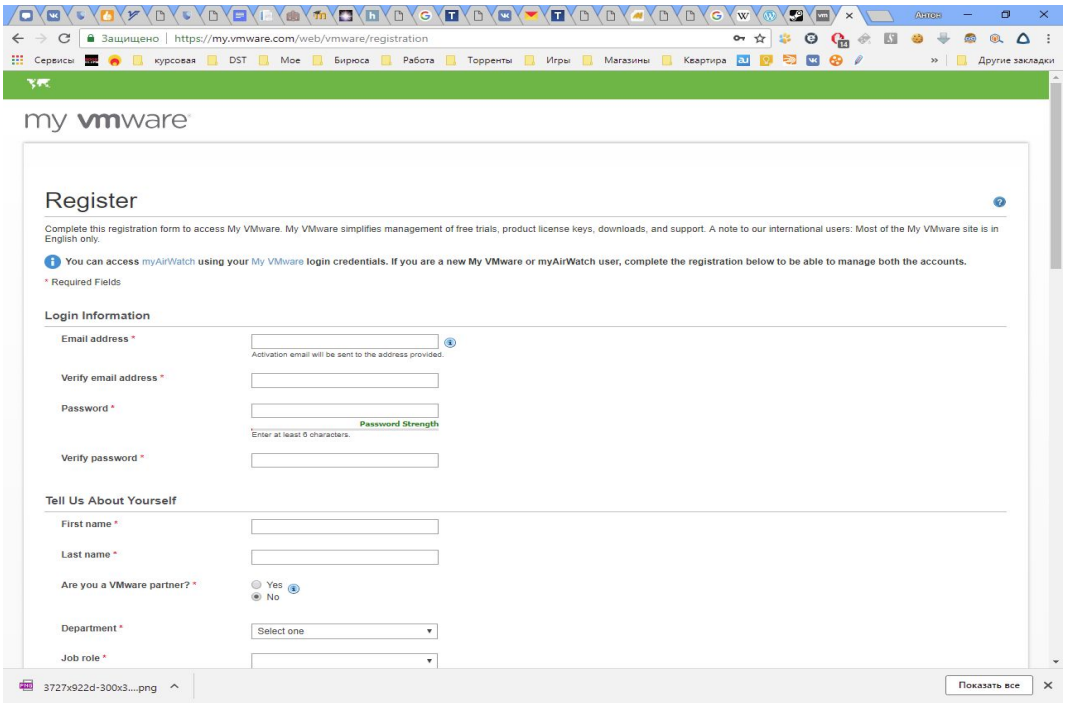
Для клиентских машин подойдет минимальная аппаратная конфигурация, соответствующая требованиям операционной системы, которая будет установлена на физическую машину. Например, требования для Ubuntu:

- двухъядерный процессор с частотой 1,6GHz и мощнее;
- 2GB RAM;
- 5GB Жесткий диск;
- любая видеокарта, для которой есть драйвера
- блок питания от 300W

- другие периферийные устройства (монитор, клавиатура, мышь и т.д.)

Основой инфраструктуры является гипервизор VMware vSphere Hypervisor (ESXi).

Для того, чтобы получить дистрибутив необходимо зарегистрироваться на сайте https://my.vmware.com/web/vmware/registration:



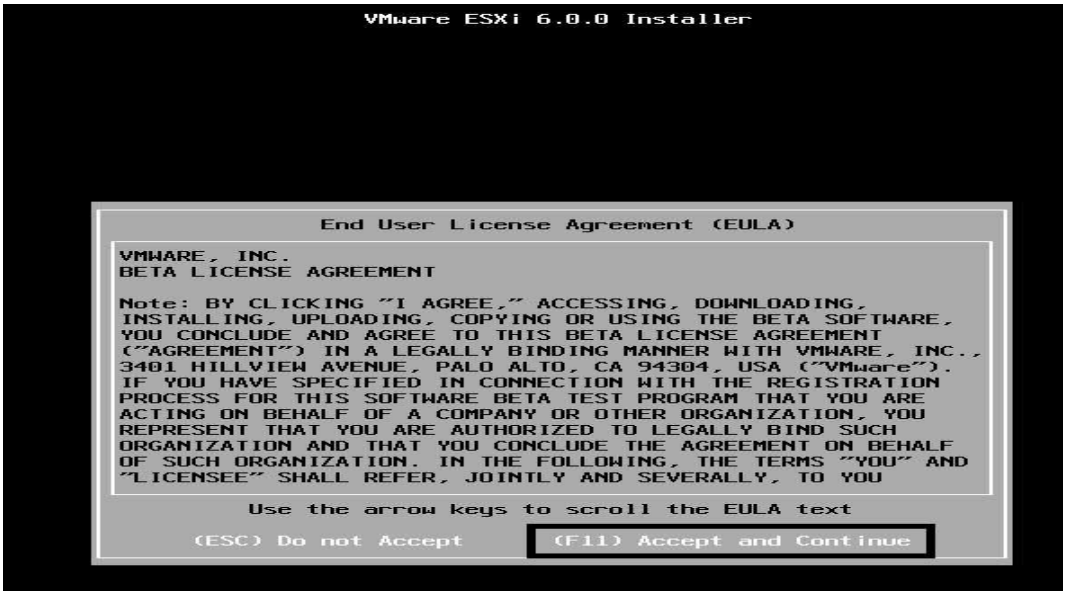
после это станет доступен раздел с загрузками, и на почту будет выслан ключ для регистрации лицензии для бесплатного использования гипервизора.

Перед установкой необходимо записать на диск скачанный дистрибутив. В BIOS включить поддержку технологии Intel VT и загрузку с диска.

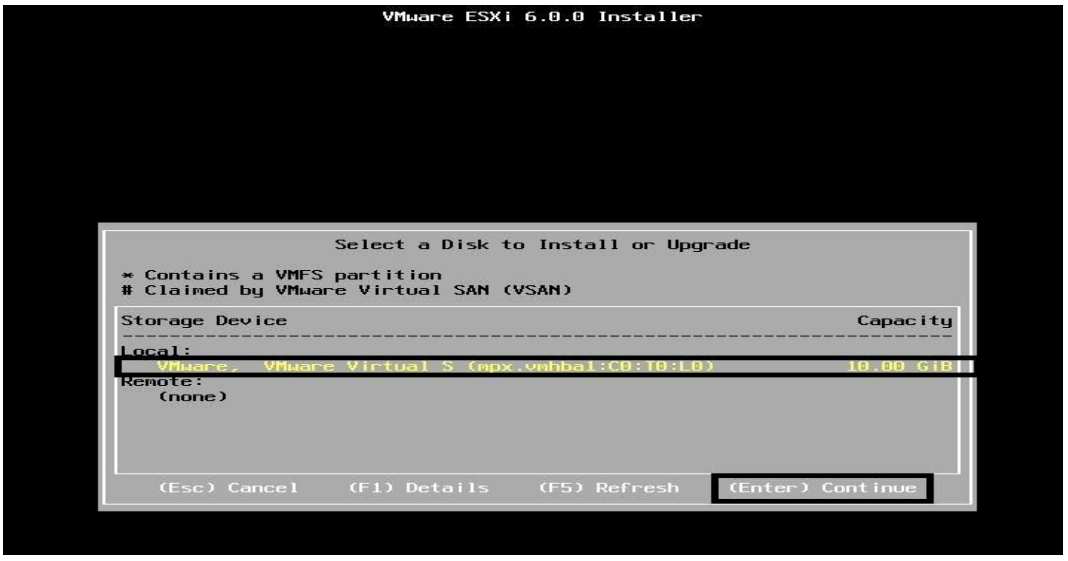
При загрузке с диска запускаем программу установки ESXi:

после загрузки самой программы установки, инициализации оборудования и загрузки модулей программы продолжаем установку гипервизора:

принимаем лицензионное соглашение:



выбираем место установки:



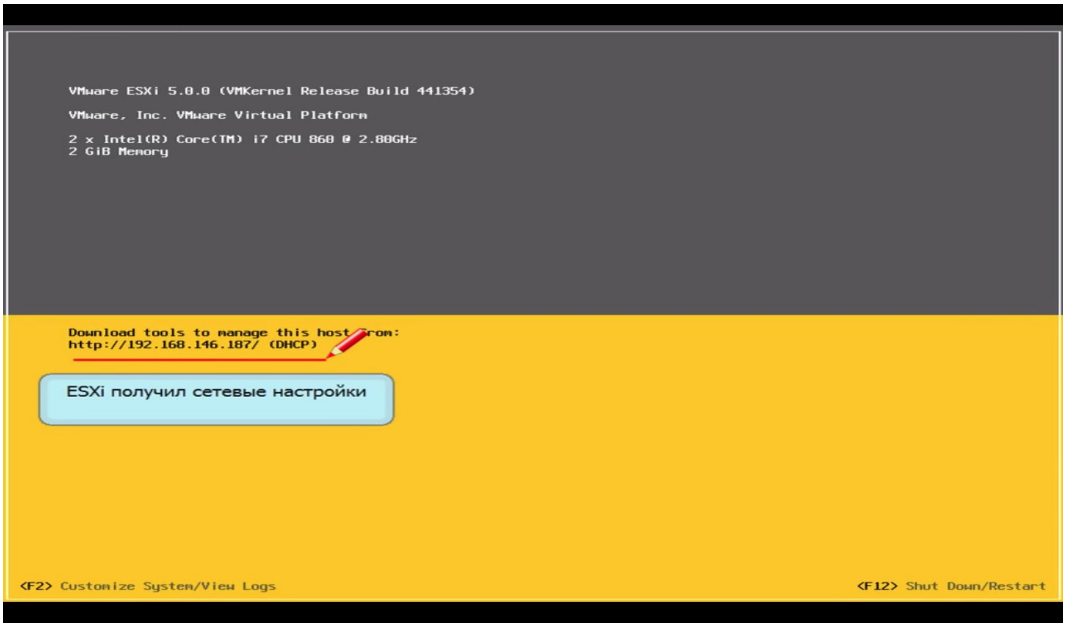
раскладку клавиатуры:

Далее нас просят ввести пароль для входа под администраторским пользователем root в последней версии необходимо соблюдать требования к сложности пароля – от 7 символов, заглавная, строчная буквы, цифры и спецсимволы:

и подтвердить установку, учитывая, что данные диска будут потеряны:

После установки гипервизора, необходимо извлечь установочный диск и перезагрузить сервер:

На этом можно считать установку завершенной. После загрузки система отобразит на экране сетевой адрес, по которому нужно пройти для работы с гипервизором.



Для подключения к серверу необходимо с любой машины в той же сети в адресной строке веб-браузера ввести указанный адрес.

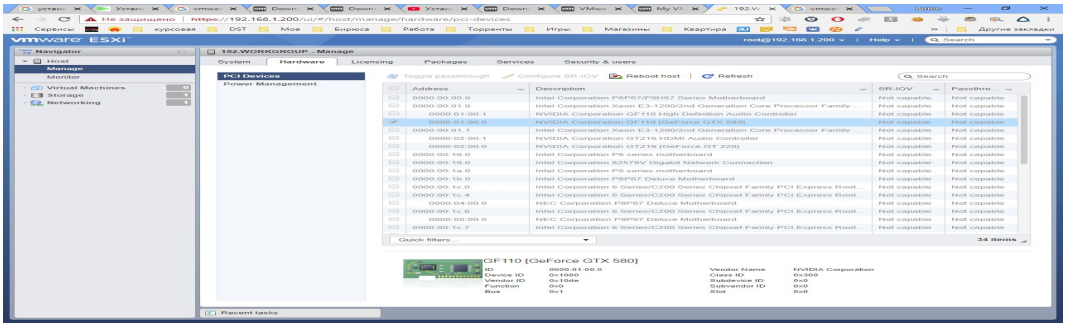
После подключения к хосту ESXi получаем следующие основные возможности:

- создать виртуальную машину;
- подключение к системе хранения данных.

Далее нужно установить компонент NVIDIA vGPU Manager, который создает связь между виртуальной GPU видеокарты и виртуальной машиной. Устанавливается он как обычный VIB-пакет:

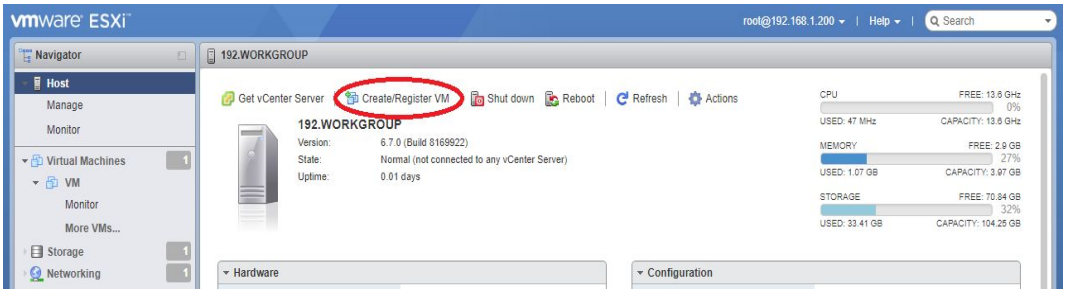
```
user@ubuntu-11-10:~$ esxcli -s 10.1.102.154 -u root software vib install -v http://IP/vmware-esx-provider-arcconf.vib
Enter password:
Installation Result
   Message: Operation finished successfully.
   Reboot Required: false
   VIBs Installed: Adaptec_bootbank_arcconf_1.00-1
   VIBs Removed:
   VIBs Skipped:
```

Перед созданием виртуальной машины нужно разрешить доступ к видеокарте. Это делается в пункте Manage, вкладка Hardware. Необходимую видеокарту отмечаем галочкой и перезагружаем гипервизор.

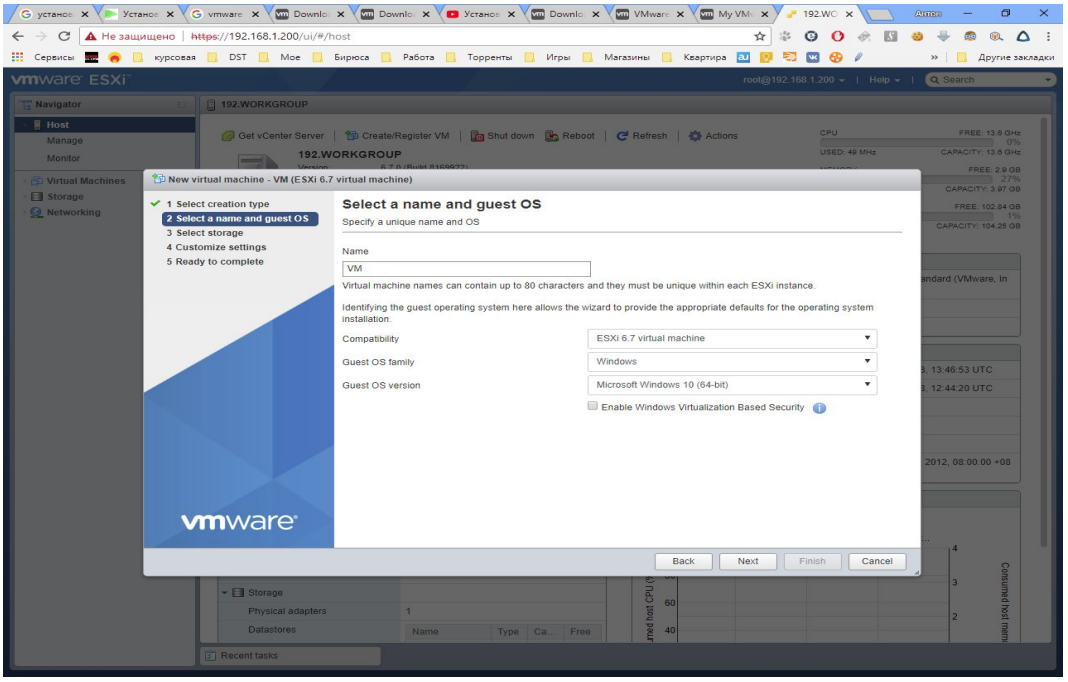


Еще раз напомню, что необходимо наличие двух графических адаптеров, один любой для работы самого гипервизора и один или более для работы с CUDA.

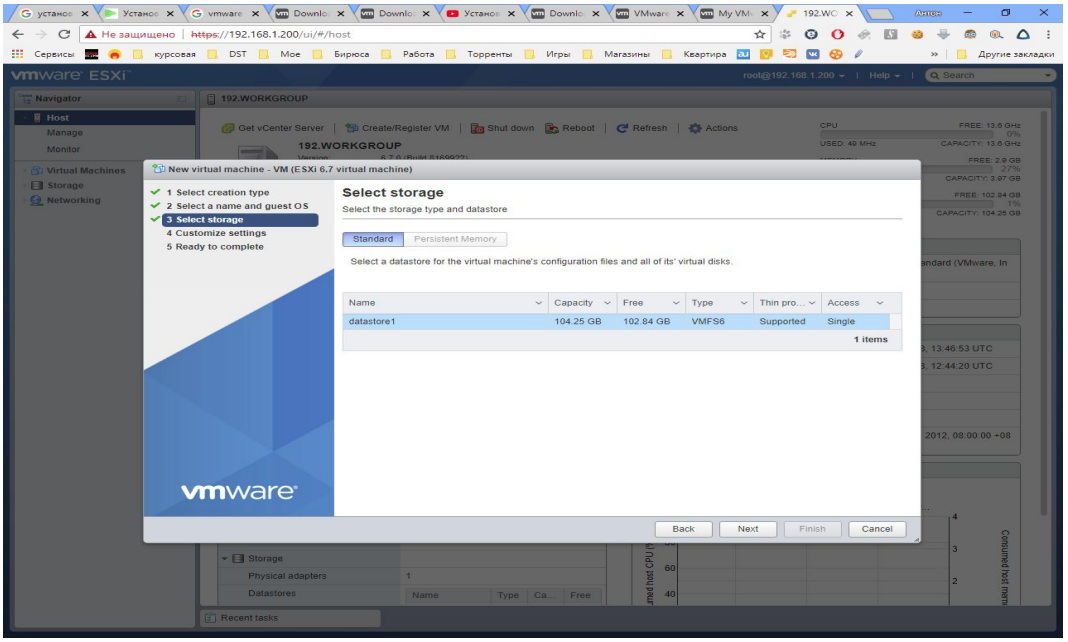
После перезагрузки гипервизора создаем виртуальную машину: выбираем Host и открываем форму создания виртуальной машины.



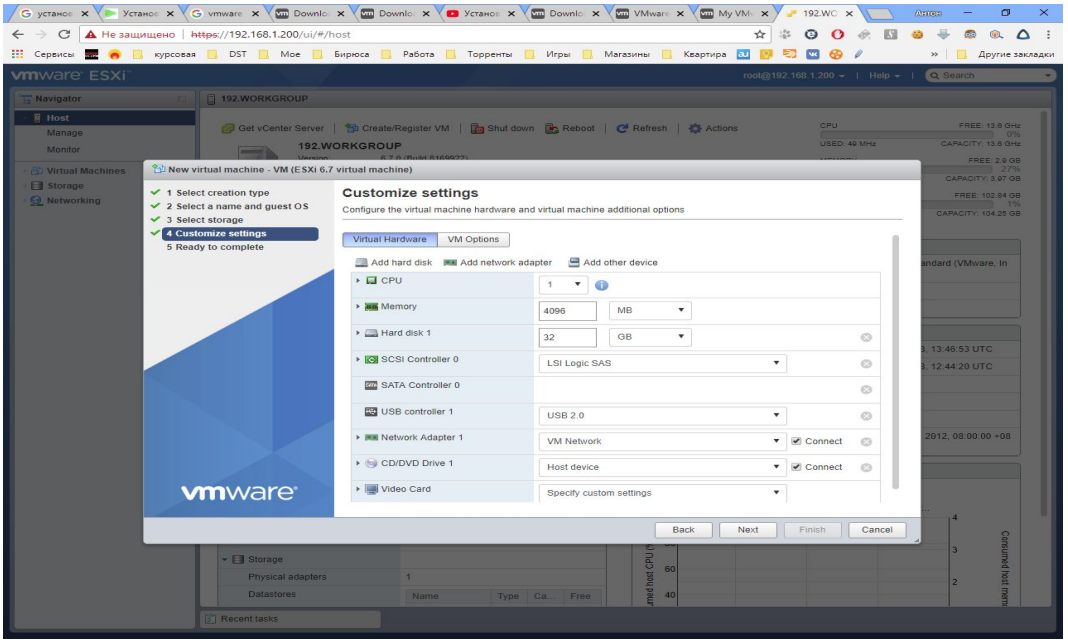
Следуя подсказкам мастера создания новой виртуальной машины, вводим имя, совместимость с другими версиями гипервизора ESXi, семейство и версию операционной системы:



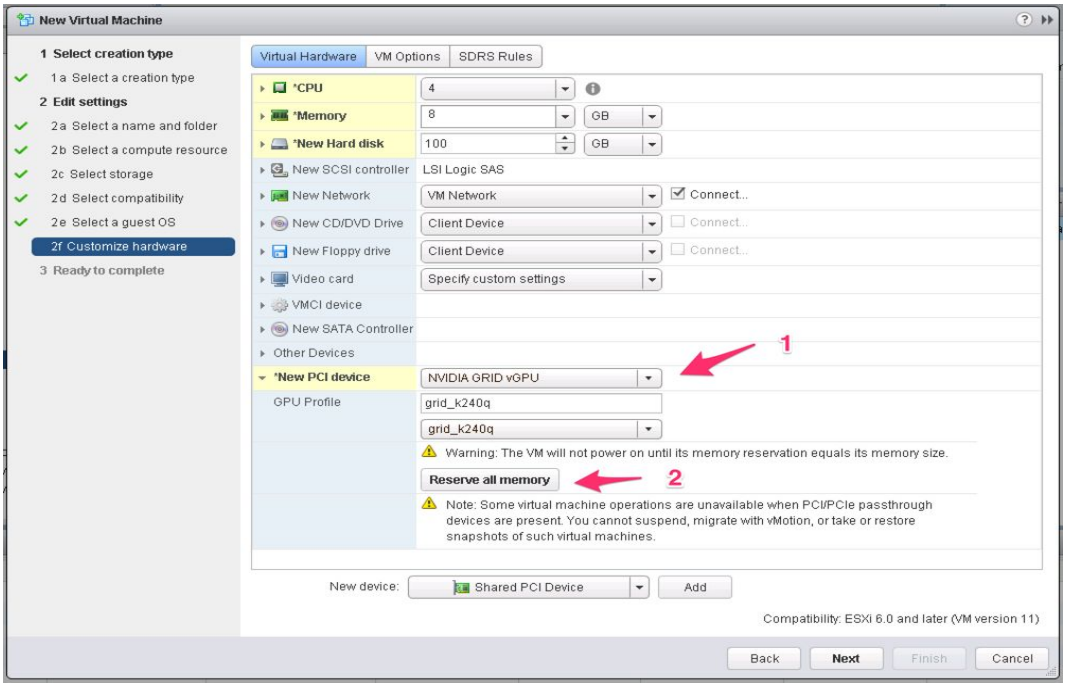
Выбираем место для хранения виртуальной машины:



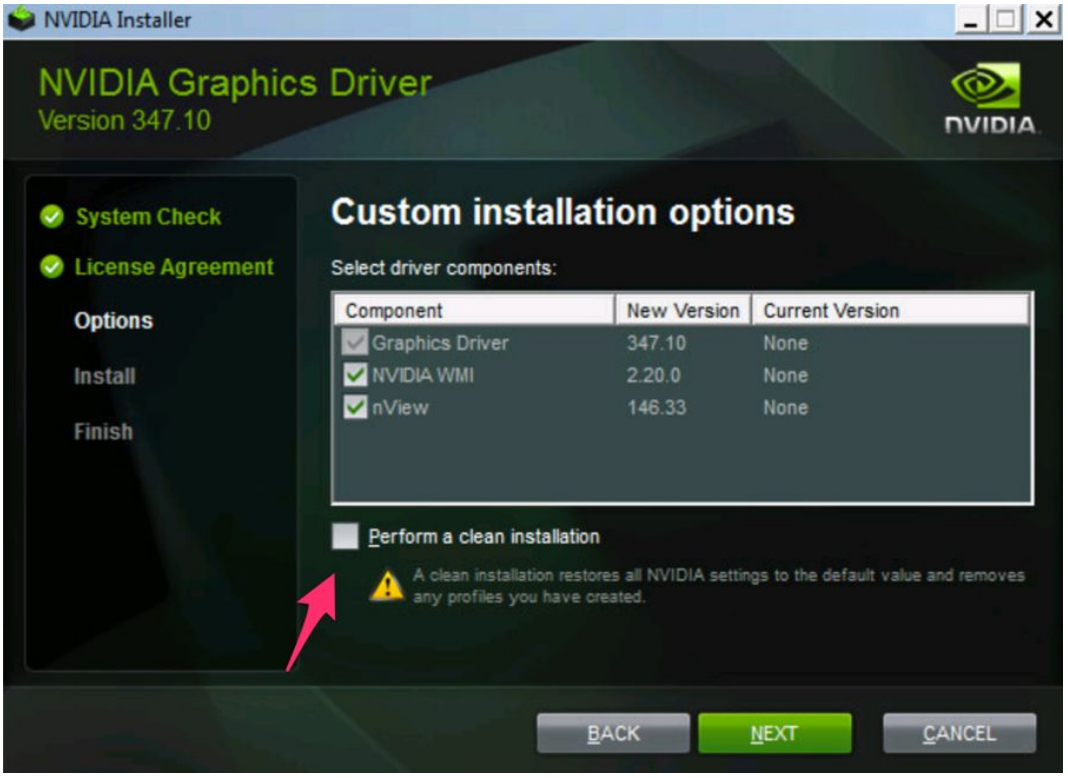
Затем задаем окончательные параметры виртуальной машины: количество CPU, оперативная память, дисковое пространство и другие настройки. Рекомендованные параметры для Windows 10: 2 CPU, 2GB Memory, 20 GB Hard disk.



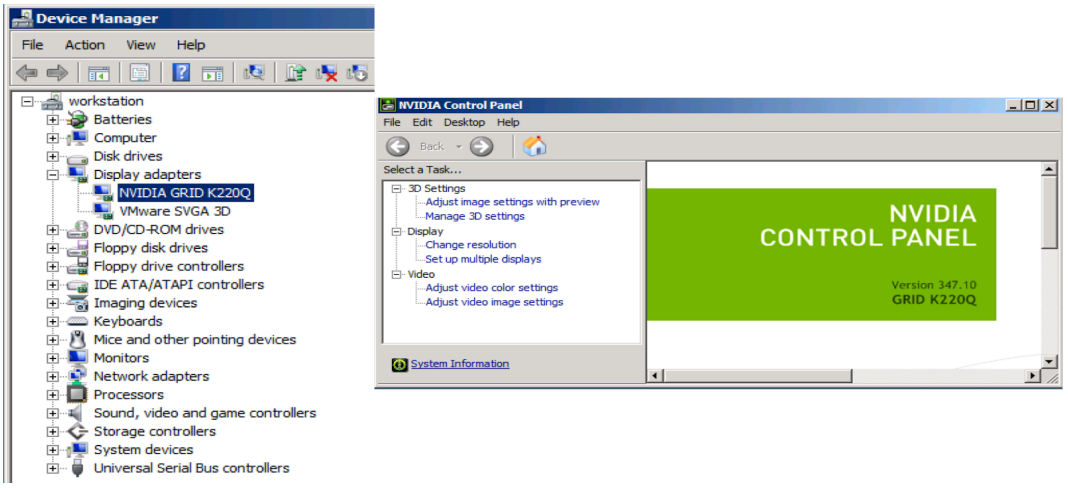
На этом же этапе можно выбрать подключение графического ускорителя. Нажать кнопку Add other device и выбрать пункт PCI device и выбрать ту видеокарту, которую собираемся использовать и профиль GPU (1), и резервируем память (2), либо сделать это после создания виртуальной машины через ее редактирование:



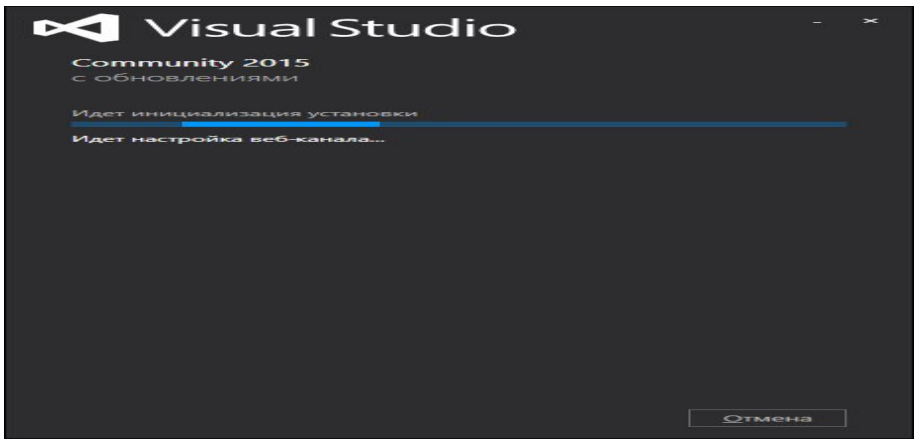
На этом создание виртуальной машины завершается. Остается установить на виртуальную машину операционную систему (пример, Windows 7 или новее), драйвера на виртуальный ускоритель NVIDIA GRID driver:



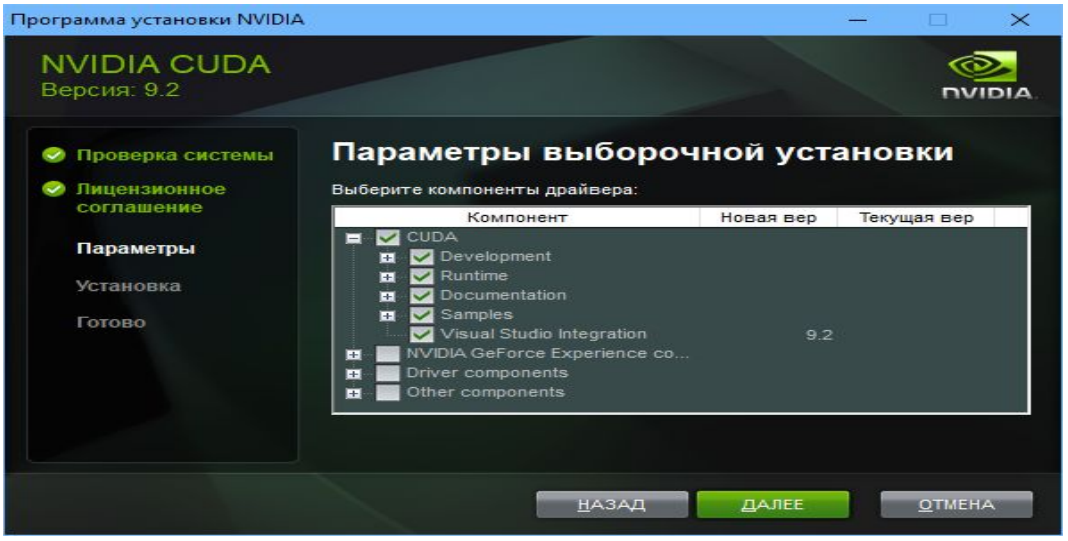
В диспетчере устройств виртуальной машины отобразится доступный для использования графический ускоритель:



Дополнительно нужно установить Visio Studio:



и CUDA Toolkit:



Теперь на основе этой виртуальной машины можно создать так называемый «золотой образ», с которого другие виртуальные машины будут загружаться, все аппаратные параметры и программные у них будут идентичны, а данные, которые появятся в процессе работы индивидуальны.

Это заметно упрощает и ускоряет процесс внедрения новой инфраструктуры. Не нужно настраивать каждую машину в отдельности, как это происходит при физической реализации.

Заключение.

В настоящее время мы наблюдаем за интересными процессами в области параллельных (высокопроизводительных) вычислений. Графические укорители прочно заняли эту нишу и пока никому не уступают, а только продолжают наращивать производительность. Технология CUDA развивается лавинообразно, от года в год набирая мощности и скорости вычислений, позволяя самым различным областям деятельности человека сократить время на обработку данных. Дизайн, производство, маркетинг, архитектура, медицина, космонавтика, область искусственного интеллекта и много-много других областей, где используются параллельные вычисления, и куда требуются квалифицированные программисты.

В тоже время гонка за уменьшением затрат на информационную структуру предприятий и учреждений заставляет разработчиков технологий виртуализации предлагать новые варианты, учитывая растущие потребности пользователей. Но даже сейчас уже имеющиеся и с успехом используемые технологии позволяют обеспечить необходимые аппаратные и программные ресурсы для выполнения поставленных задач при соразмерных затратах на внедрении, и экономии на обслуживании, содержании и модернизации IT-структуры организаций.

На сегодняшний день в большинстве учебных заведениях классы не укомплектованы для проведения лабораторных работ по программированию с использованием графического ускорителя. Сказывается ограничение бюджета, отсутствие знаний возможных альтернатив в виде применения сервера виртуализации. Что ограничивает учащихся в получении полных качественных знаний по параллельному программированию.

Учитывая эти обстоятельства было решено разработать общие рекомендации для организации инфраструктуры для выполнения

лабораторных работ по программированию с использованием графического ускорителя.

В результате реализации цели, были выполнены следующие задачи:

- изучена литература и информационные ресурсы с целью выявления и понимания сущности и особенностей параллельного программирования в целом, и с использованием графического ускорителя в частности, а также виртуализации – как разновидности терминального доступа;
- проанализированы требования к аппаратной части компьютера для выполнения лабораторных работ по высокопроизводительным вычислениям с использованием графического ускорителя,
- проанализирована возможность доступа к графическому ускорителю при различных видах виртуализации
- апробированы способы виртуализации и определен наиболее эффективный способ программного доступа к графическому ускорителю из виртуальной среды.

Результатом выполнения целей стала рекомендация по созданию инфраструктуры для выполнения лабораторных работ по программированию с использованием графического ускорителя.

Реализация виртуализации в учебном классе создает благоприятную среду для проведения лабораторных работ. Виртуализация рабочих станций предоставляет вариативность, безопасность, наглядность процессов и результатов проведения лабораторных работ. Работа в виртуальных машинах ничем не отличается от физических, и их можно использовать для исследования новых операционных систем, апробации нового программного обеспечения, проведения тестов производительности приложений на различных архитектурах и конфигурациях машин.

Список использованных источников.

1. CUDA Toolkit документация [Электронный ресурс] URL: https://docs.nvidia.com/cuda/ (дата обращения 5.06.2018)

2. NVIDIA Developer. Портал для разработчиков. [Электронный ресурс] URL: https://developer.nvidia.com/ (дата обращения 7.06.2018)

3. NVIDIA VIRTUAL GPU RESOURCES [Электронный ресурс] URL: http://www.nvidia.com/object/grid-enterprise-resources.html (дата обращения 7.06.2018)

4. База знаний Citrix. [Электронный ресурс] URL: https://support.citrix.com/en/products/xenserver (дата обращения 7.06.2018)

5. База знаний VMware. [Электронный ресурс] URL: https://www.vmware.com/support/vsphere.html (дата обращения 7.06.2018)

6. Библиотека технической информации для Windows Server. Виртуализация. [Электронный ресурс] URL: https://docs.microsoft.com/ru-ru/windows-server/virtualization/virtualization (дата обращения 7.06.2018)

7. Богачёв К.Ю. Основы параллельного программирования. М.: Бином. Лаборатория знаний, 2003

8. Боресков А.В., Харламов А. А. Параллельные вычисления на GPU. Архитектура и программная модель CUDA: Учебное пособие. М.: Издательство Московского университета, 2012

9. Васильковский В.А., Котов А.Г., Марчук А.Г., Миренков Н.Н. Элементы параллельного программирования; Под ред. В.Е. Котова, М.:Радио и связь, 1983

10. Воеводин В.В., Воеводин Вл.В. Параллельные вычисления. СПб.: БХВ-Петербург, 2002

11. Гультяев А.К. Виртуальные машины: несколько компьютеров в одном (+CD). СПб.: Питер, 2006

12. Диттнер Р. И др. Виртуализация и Microsoft Virtual Server 2005. Пер. с англ., М.: ООО «Бином-Пресс», 2008

13. Качко Е.Г. Параллельное программирование: Учебное пособие. Харьков: Форт, 2011

14. Краузе Дж. Windows Server 2016 Книга рецептов. [Электронный ресурс]
URL: http://onreader.mdl.ru/WindowsServer2016Cookbook/content/index.html (дата обращения 7.06.2018)

15. Кулагин К., Балдин Е. CUDA: Пустим. Часть 2. Статья. Linux Format, 2013. – Май. – С. 88-91.

16. Кулагин К., Балдин Е. CUDA: Ускоряем. Часть 1. Статья. Linux Format, 2013. – Апрель. – С. 86-89.

17. Кэндрот Э., Сандерс Д. Технология CUDA в примерах. Введение в программирование графических процессоров.

18. Ларсон Р., Карбон Ж. Платформа виртуализации Hyper-V. Ресурсы Windows Server 2008 : Пер. с англ. М.: Издательство «Русская редакция»; СПб.: «БХВ-Петербург», 2010

19. Лупин С.А., Посыпкин М.А. Технологии параллельного программирования. М.: ИД «ФОРУМ»: ИНФРА-М, 2011

20. Михеев М.О. Администрирование VMware vSphere. М.: ДКМ Пресс, 2010

21. Моримото Р., Ноэл М., Ярдени Г, и др. Microsoft Windows Server 2012. Полное руководство. : Пер. с англ., М.: ООО «И.Д. Вильямс», 2013

22. Немет Э.,Снайдер Г., Хейн Т., Уэйли Б. Unix и Linux: руководство системного администратора, 4-е изд. : Пер. с англ., М.: ООО «И.Д. Вильямс», 2012

23. НОУ «ИНТУИТ». Обзор архитектуры современных многоядерных процессоров. Лекция 1. [Электронный ресурс] URL:

https://www.intuit.ru/studies/courses/10611/1095/info (дата обращения 5.06.2018)

24. Ньюмен С. Создание микросервисов. СПб.: Питер, 2016

25. Остапкевич М., Балдин Е. OpenCL: Стандарт. Статья. Linux Format, 2013. – Июнь. – С. 86-89.

26. Развертывание RemoteFX на Windows Server 2012 [Электронный ресурс] URL: https://habr.com/post/155129/ (дата обращения 7.06.2018)

27. Сандерс Дж., Кэндрот Э. Технология CUDA в примерах: введение в программирование графических процессоров. : Пер. с англ., М.: ДМК Пресс, 2011

28. Турулин И.И. Виртуализация: Учебное пособие. Таганрог: ТТИ ЮФУ (ТРТИ, ТРТУ), 2012

29. Эндрюс Г.Р. Основы многопоточного, параллельного и распределенного программирования. : Пер. с англ., М.: ООО «И.Д. Вильямс», 2003

30. Яковлев В.В. Технологии виртуализации и консолидации информационных ресурсов: Учебное пособие. М.: ФГБОУ «Учебно-методический центр по образованию на железнодорожном транспорте», 2015