Правительство Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего профессионального образования
«Санкт-Петербургский государственный университет»

Кафедра Системного Программирования

Анисимов Константин Александрович

Реализация динамического подключения физического дисплея к виртуальной машине на основе технологии виртуализации графики Intel

Дипломная работа

Допущена к защите.
Зав. кафедрой:
д. ф.-м. н., профессор Терехов А. Н.

Научный руководитель:
ст. преп. Баклановский М. В.

Рецензент:
ст. инженер-программист Котов Ю. А.

Санкт-Петербург
2015

SAINT-PETERSBURG STATE UNIVERSITY

Chair of Software Engeneering

Konstantin Anisimov

Implementation of dynamic physical display connection to virtual machine using Intel graphics virtualisation technology

Graduation Thesis

Admitted for defence.
Head of the chair:
Professor Andrey Terekhov

Scientific supervisor:
Sr. Lecturer Maxim Baklanovsky

Reviewer:
Sr. Software Engeneer Kotov Yurii

Saint-Petersburg
2015

Оглавление

Введение **5**

1. Обзор **7**
1.1. Виртуальный графический ускоритель 7
1.2. Эмуляция 9
1.3. Паравиртуализация 9
1.3.1. VMware SGVA II 10
1.3.2. VMGL 12
1.4. PCI pass-through 13
1.5. Аппаратно-програмная виртуализация GPU 14
1.5.1. Nvidia GRID 14
1.6. Mediated pass-through 15
1.6.1. Технология виртуализации графики Intel 16
1.7. Сравнение 16

2. Постановка Задачи **18**

3. Архитектура виртуализации графики Intel **19**
3.1. Архитектура Intel GPU 22
3.2. Виртуализация GPU 22
3.3. Механизм вывода изображения 22

4. Виртуализация механизма вывода изображения **25**
4.1. Схема виртуализации 26
4.1.1. MMIO 27
4.1.2. Interrupts 28
4.2. Динамическое переключение 28
4.3. Управление и настройка 28
4.4. Отладка и тестирование 29
4.5. Вывод 30

5. Тестовое внедрение **31**

Заключение **32**

Список литературы **33**

Введение

Одними из основных характеристик любой информационной системы являются её безопасность и надежность. Для многих компаний эти параметры являются важнейшими и учитываются в первую очередь. В IT-сфере под безопасностью (или информационной безопасностью) обычно понимают защищенность системы и ее окружения от целенаправленных или случайных действий, в результате которых может быть нанесен неприемлемый ущерб самой информационной системе или ее пользователям. Под надежностью обычно имеют в виду доступность и правильное функционирование системы, вне зависимости от различных внешних факторов влияющих на систему. Одним из способов повысить эти характеристики является виртуализация.

Виртуализация

Виртуализация - это процесс создания виртуальной(т.е не имеющей физической основы) версии какого-либо ресурса, для достижения изоляции ресурсов друг от друга. В основе виртуализации лежит отделение реализации от представления. Виртуальный ресурс обеспечивает большую гибкость при его использовании, так как он практически не ограничен аппаратной реализацией. То есть реализация этого ресурса может быть изменена незаметно для системы.

Таким образом одна и та же система может работать и в физическом, и в виртуальном окружении, и достигается значительная гибкость в управлении ресурсами. Так как виртуальные ресурсы отделены друг от друга, то и информационные системы, использующие разные виртуальные ресурсы, оказываются недоступны друг для друга. Таким образом увеличивается безопасность и надежность всей системы в целом, так как при ошибке, отказе или взломе из строя выходит только одна система.

В связи с описанными преимуществами многие компании, имеющие ИТ инфраструктуру, используют виртуализацию при её построе-

нии. Операционная система в месте с виртуальным окружением в котором она работает называется виртуальной машиной(ВМ).Программный продукт отвечающий за виртуализацию называется гипервизор или менеджер виртуальных машин.

Виртуальный рабочий стол

Частью этой инфраструктуры, которую можно подвергнуть виртуализации, является персональный компьютер. В большинстве случаев на персональном компьютере установлена и работает Операционная Система с графическим интерфейсом. В совокупности ОС, графический интерфейс и внешние устройства для управления ими обычно называют рабочим столом. И с точки зрения пользователя виртуализируемым ресурсом будет рабочий стол. Таким образом одной физической системой может пользоваться несколько человек одновременно. Но создание виртуальных рабочих столов имеет смысл не только в серверном варианте. Персональные компьютеры зачастую обладают избыточным количеством внешних интерфейсов и значительной производительностью, которая в основном используется не полностью. Можно огранизовать разделение различных внешних ресурсов между пользователями. Таким образом одна физическая система будет разделена на несколько самостоятельных частей, которые могут работать параллельно и независимо.

1. Обзор

Для создания полноценного виртуального рабочего стола необходимо виртуализировать следующие группы физических ресурсов.

- Базовые ресурсы без которых работа ОС, за редким исключением, невозможна в принципе. Например центральный процессор, оперативная память или ПЗУ.
- Устройства ответственные за обработку внешних событий, такие как мышь, клавиатура, сетевой адаптер
- Устройства обратной связи, которые позволяют пользователю видеть результат своих действий. Самым основным является дисплей.
- Ускорители. Благодаря аппаратной реализации некоторых вычислительно сложных задач, достигается увеличение производительности системы в целом.
- Другие необязательные ресурсы, реализующие очень специфичные функции.

В некотором виде первые 3 группы реализованы в любом гипервизоре. И их достаточно для простых сценариев использования. Но современные ПК позволяют решать широкий спектр задач. И для большинства пользователей приемлемая производительность при их решении предполагает использование ускорителей. Основным среди них является графический процессор.

1.1. Виртуальный графический ускоритель

Современный графический процессор реализует множество функций позволяющих ускорить обработку различных данных. Благодаря тому что эти функции реализованы аппаратно, то их выполнение на графическом процессоре оказывается в разы быстрее. В частности следующие функции присутствуют во многих из них:

- Ускорение 3D и 2D графики
- Ускорение параллельных вычислений
- Реализация различных алгоритмов кодирования видео

Поскольку использование этих функций сопряжено со значительными вычислениями, то виртуальный аналог графического процессора должен не только реализовывать необходимые функции, но и выполнять их с приемлемой производительностью. Помимо этого виртуальный ресурс в своей реализации использует физические ресурсы, доступность которых также является ограничением, которое должно учитываться. В частности подавляющем большинстве ПК физически присутствует только один графический процессор, что накладывает ограничение на выбор реализации виртуального адаптера.

Существует несколько способов создания виртуального графического ускорителя. Каждый из них обладает своими фундаментальными недостатками. Далее будут рассмотрены следующие технологии создания виртуального графического адаптера:

- Эмуляция
- Паравиртуализация
- PCI pass-through
- Nvidia GRID
- Mediated pass-through

Стандарт для создания аппаратно виртуализируемых PCI устройств - PCI SR-IOV[11] - здесь рассмотрен не будет, так как его применение в контексте видеокарт сопряжено со значительными трудностями, вследствие чего на данный момент не существует ни одного GPU поддерживающего данную технологию.

1.2. Эмуляция

Основным отличием эмуляции является то, что все операции, необходимые для полноценного использования графического процессора в виртуальной машине, выполняются без использования реального графического ускорителя - на центральном процессоре. Из-за этого графический процессор простаивает. А центральный процессор вынужден выполнять огромную работу, для которой он не приспособлен. По этому производительность виртуальной машины с эмулируемым графическим ускорителем значительно меньше чем у реальной машины.

Несмотря на постоянное повышение производительности современных центральных процессоров их производительности едва хватает на обработку простейших двух мерных интерфейсов. В обозримом будущем не будет достаточного увеличения производительности, которое позволит эмулировать операции обработки трёхмерный графики, со скоростью, достаточной для какого либо применения.

Тем не менее поскольку данный способ никак не использует графический процессор, только он может применяться на системах, в которых он отсутствует. Чего нельзя сказать о других технологиях виртуализации графики. Хотя в серверах это является существенным преимуществом, в условиях поставленной задачи это не так.

Таким образом единственным с чем успешно справляется полностью эмулированный графический ускоритель это двухмерная графика. Но для большинства современных ПК это только малая часть их возможностей. И для эмуляции полноценного рабочего места этого не достаточно.

1.3. Паравиртуализация

Паравиртуализация - способ виртуализации, когда драйвер в виртуальной машине знает, что он работает с виртуальным устройством, и благодаря этому, для реализации интерфейса этого устройства он может общаться напрямую с гипервизором, а не с виртуальным устрой-

ством, предоставляемым гипервизором. Таким образом достигается возможность использовать в ВМ универсальный драйвер для устройств одного типа, а последующим доступом к физическому устройству будет заниматься полноценный драйвер, запущенный в гипервизоре

Современные приложения, использующие графическое ускорение, не используют графический адаптер на прямую. В подавляющем большинстве случаев доступ к нему осуществляется через специальный стандартизированный интерфейс(API), предоставляемый ОС. Основными стандартами такого интерфейса являются OpenGL[10], Direct3D[8], Gallium3D[

Gallium3D является более низкоуровневым интерфейсом чем OpenGL и Direct3D. И также включает в себя реализацию функций не зависящих от аппаратной основы. Таким образом от драйвера не требуется полная реализация высокоуровнего графического интерфейса и достигается упрощение драйвера графического адаптера.

Таким образом при использовании паравиртуализации для создания виртуального 3d ускорителя нужно передать запросы к соответствующему интерфейсу в гипервизор. Переданные команды изменяются так, чтобы сделать вид, что они прошли не из виртуальной машины, а из обычного приложения. Затем используя оригинальный драйвер для физического GPU, эти команды исполняются на нем.

Есть различные способы, которыми драйвер графического адаптера может передать гипервизору команды графического интерфейса. Способ общения с гипервизором и API, чьи вызовы передаются в гипервизор, являются основным отличием в различных реализациях паравиртуализированного GPU.

1.3.1. VMware SGVA II

VMware в своей реализации виртуального GPU совместила два подхода - эмуляцию и паравиртуализацию. Передача вызовов графического интерфейса осуществляется через сэмулированное PCI устройство

- VMware SGVA II[4]. OpenGL и Direct3D вызовы транслируются во внутренние команды, которые затем записываются в очереди в памяти. Затем бекенд этого устройства транслирует эти комманды в вызовы Gallium3D интерфейса.

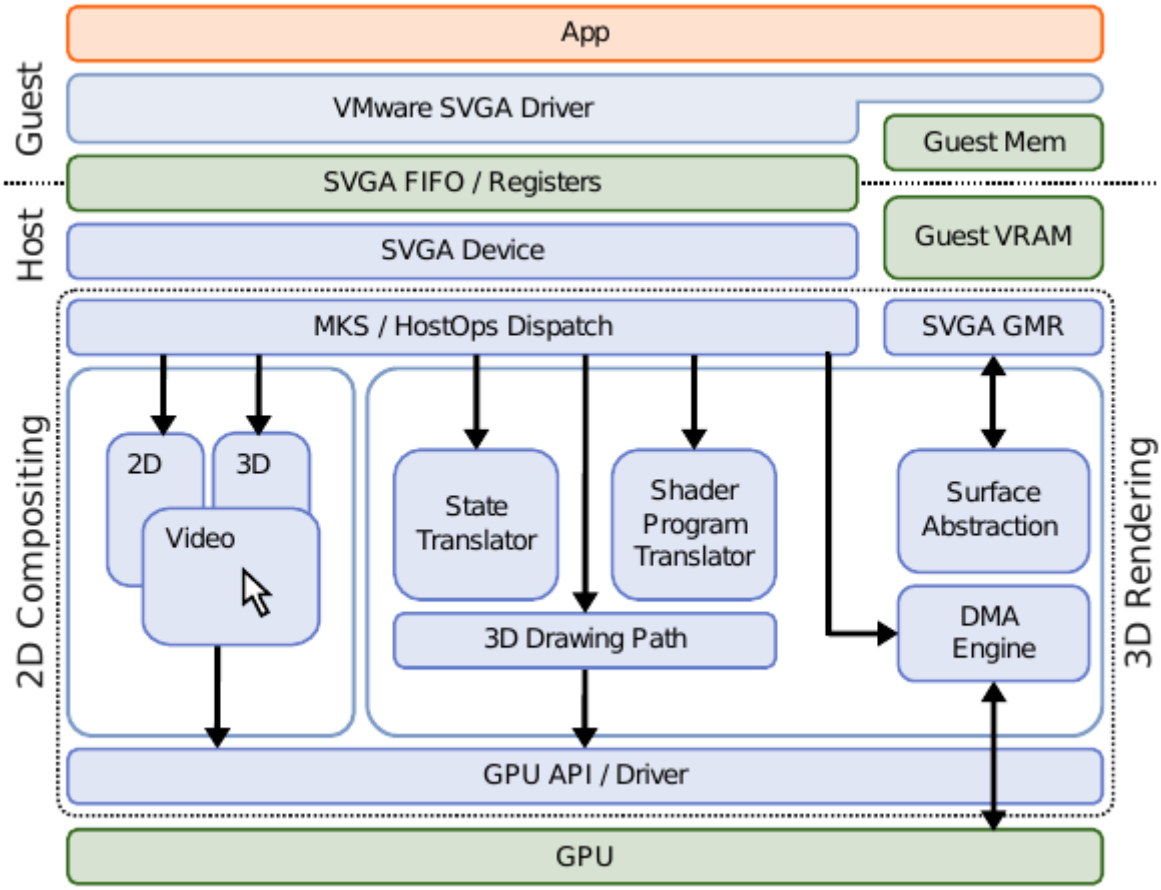


Рис. 1: Архитектура VMware SGVA II

Благодаря использованию PCI интерфейса, реализация виртуального GPU может использовать прямой доступ к памяти(Direct memory access, DMA) для доступа к графическим данным вируальной машины. Таким образом если реализация виртуального DMA в гипервизоре поддерживает доступ к реальной памяти виртуальной машины как к обычной памяти доступной гипервизору, то Gallium3D интерфейс может оперировать непосредственно с данными в виртуальной машине. Соответственно и физический GPU будет работать с данными в виртуальной машине. Таким образом если транслировать адреса виртуальной машины в адреса гипервизора, то отпадет необходимость ко-

пировать графическую память. Это позволяет добиться значительного ускорения по сравнению с другими способами передачи графических комманд в гипервизор. Тем не менее в болшинстве тестов производительность меньше физического GPU в среднем в 3 раза. И только в специальных тестах достигает 80%. Также данное решение применяется только в гипервизоре VMware - ESXi[14].

1.3.2. VMGL

VMGL[13] обеспечивает виртуализацию на уровне OpenGL API в виртуальной машине. Для передачи вызовов OpenGL используется сетевое соединение. Так как OpenGL и сетевой стек поддерживаются подавляющим большинством ОС и гипервизоров, то реализация VMGL является универсальной и практически не зависит от гипервизора. Также использование VMGL library и VMGL Extention не предполагает доступ к ядру ОС, то есть гостевая часть VMGL может работать в пространстве пользователя. Таким образом обеспечивается значительная гибкость при использовании данной технологии.

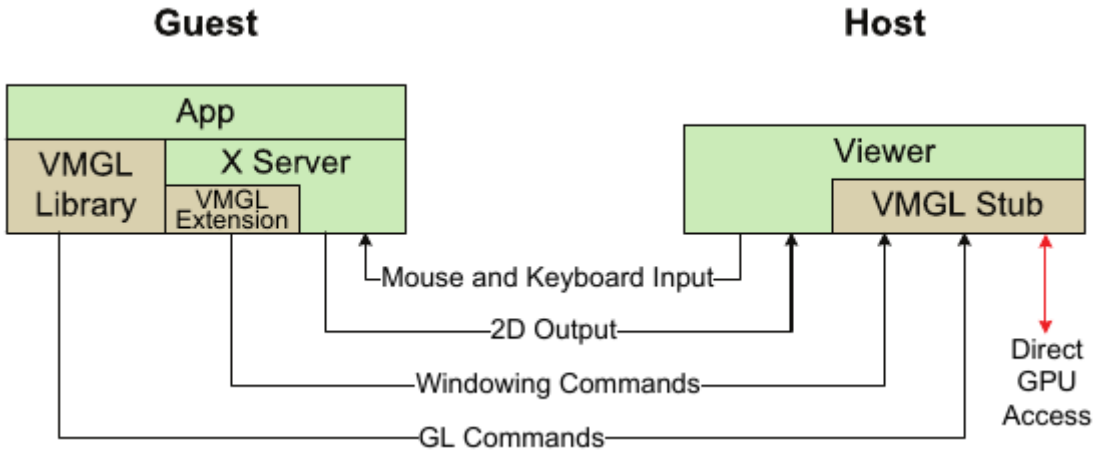


Рис. 2: Архитектура VMGL

Хотя VMGL для передачи команд и использует более производительный протокол Chromium[3], но объем передаваемых данных все равно значителен. Для графически требовательных приложений объем передаваемых данных составляет более 2Гбит/с. В итоге основным

фактором производительности VMGL является качество сетевого соединения. Таким образом для достаточной производительности сети между виртуальной машиной и гипервизором необходимо использовать паравиртуализированный сетевой адаптер. А это уже накладывает ограничение на гипервизор и гостевую ОС. Также помимо непосредственно графической нагрузки, на передачу информации тратится часть возможностей ЦП и виртуальной сети, что также является значительным минусом.

1.4. PCI pass-through

PCI pass-through - технология, которая обеспечивает доступ виртуальной машины непосредственно к физическому устройству, в данном случае к графическому адаптеру. Доступ к нему может осуществляться через PCI config space и через DMA(MMIO). Обычно PCI устройства могут иметь доступ к любой части физической памяти, и это может привести к различным проблемам. Во-первых гостевая ОС, контролируемая злоумышленником, может получить доступ к памяти другой ВМ. Во-вторых гостевая реальная память не соответствует физической, при этом графический адаптер, как и его драйвер в ВМ, об этом не знают. Вторую проблему можно решить используя незначительную паравиртуализацию в драйвере таким образом, чтобы он транслировал адреса из памяти виртуальной машины в физический адрес перед тем как отправить их на графический процессор. Но это не решает проблему полностью, так как большая часть драйверов современных GPU не поставляется с открытым исходным кодом.

Тем не менее существует аппаратно-програмное решение этих проблем. Это так называемый IOMMU(I/O Memory management unit), или Intel VT-d(Virtualization for Directed I/O)[2]. IOMMU позволяет гипервизору дать устройству доступ к той же самой части памяти, что и виртуальной машине. Таким образом и устройство и драйвер в виртуальной машине видят одну и туже память. Это требует программной поддержки со стороны BIOS(UEFI) для управления таблицами транс-

ляции адресов и прерываний. И аппаратной поддержки со стороны процессора и менеджера памяти для использования этих таблиц. Хотя сама технология была представлена компанией Intel достаточно давно, большая часть систем предназначенных для создания рабочих мест так и не оснащена полноценной реализацией IOMMU.

Также для поддержки динамического переключения GPU между виртуальными машинами, требуется аппаратная поддержка опциональной функциональности PCI интерфейса - Function Level Reset(FLR). Она позволяет сбросить состояние PCI устройства в изначальное состояние без его фактического перезапуска. Иначе для запуска новой ВМ потребуется перезагрузить всю систему. На данный момент в основном профессиональные видеокарты поддерживают эту возможность. Хотя Intel в последнем поколении встроенных GPU также реализовала эту возможность.

В итоге хотя технология и позволяет дать ВМ максимально возможную производительность и обеспечить максимальную изоляцию виртуальной машины он остальной инфраструктуры, но она не позволяет использовать один адаптер в разных виртуальных машинах одновременно.Таким образом для получения ощутимого преимущества от этой технологии необходимо наличие хотя бы двух физических GPU. Это, несмотря на значительные преимущества, не позволит использовать этот способ для решения поставленной задачи.

1.5. Аппаратно-програмная виртуализация GPU

1.5.1. Nvidia GRID

Nvidia GRID обеспечивает аппаратную реализацию виртуальных GPU[9].

Гипервизор содержит менеджер виртуальных GPU, который разделяет количественные ресурсы между виртуальными машинами. Но при этом распределение производительности реализовано на аппаратном уровне. Таким образом каждая ВМ обеспечивается необходимым

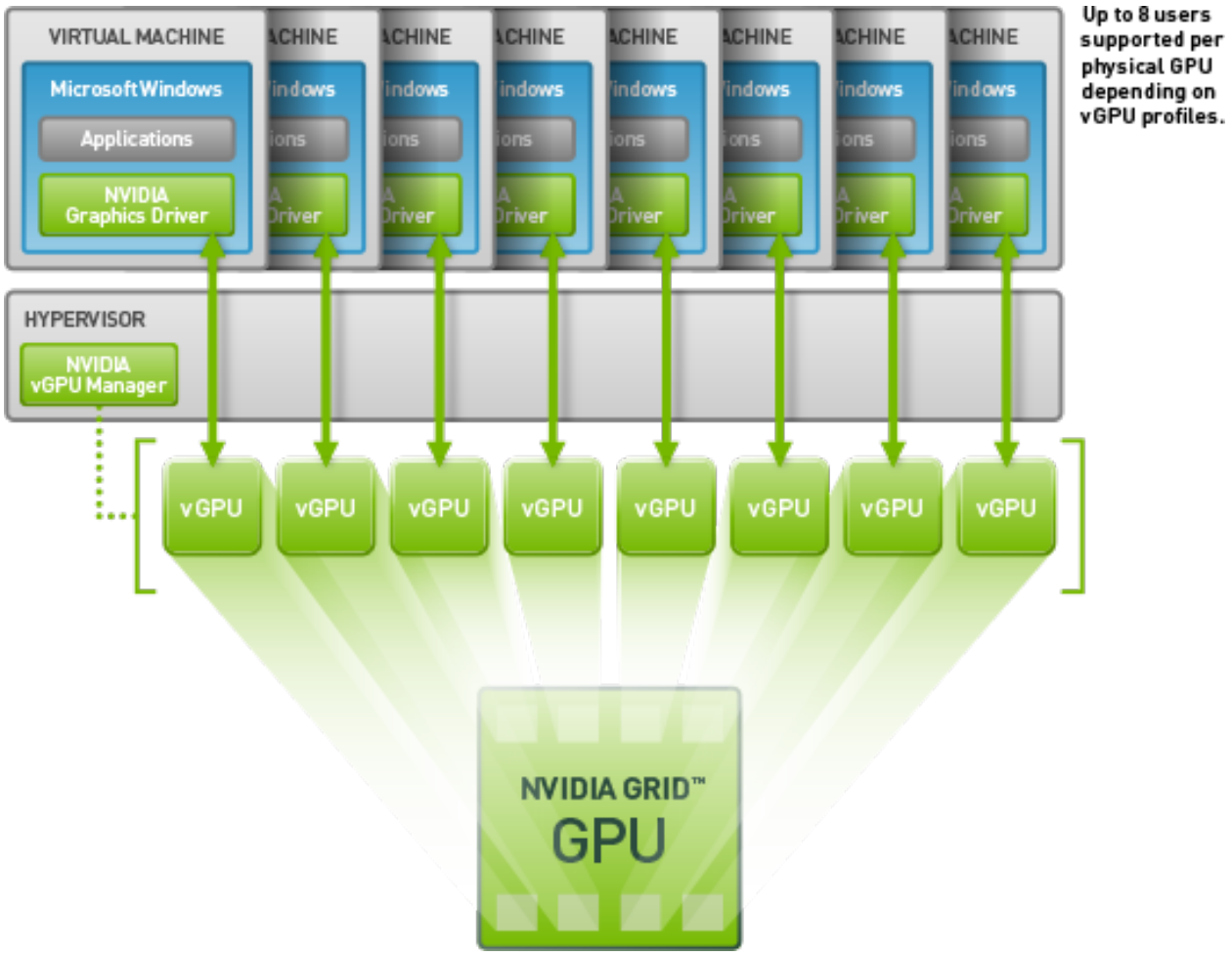


Рис. 3: Архитектура Nvidia GRID

количеством графической памяти и производительностью, точно также как и на ПК. Nvidia GRID поддерживает до 8 пользователей на физический GPU. Данная технология предназначена для серверного применения. По-этому в данном решении отсутствуют физические видеовыходы, а вывод изображения из виртуального GPU реализуется средствами гипервизора.

1.6. Mediated pass-through

Mediated pass-through расширяет подход PCI pass-through используя программный подход. Каждая виртуальная машина получает доступ к части ресурсов практически без участия гипервизора, а привелегированные операции, меняющие глобальное состояние устройства перехватываются и обрабатываются в программном слоею. Таким об-

разом сохраняется производительность PCI pass-through, и избегаются сложности аппаратной реализации устройства, присутствующие в стандарте PCI SR-IOV. Тем не менее для качественной реализации данный подход требует глубокого знания принципов работы устройства. На текущий момент единственной публично доступной полноценной реализацией этой технологии является технология виртуализация графики Intel.

1.6.1. Технология виртуализации графики Intel

Intel GVT-g[12] - применение технологии mediated pass-through к графическим процессорам Intel. В виртуальной машине используется оригинальный графический драйвер. Это позволяет упростить разработку, добиться высокой производительности и уменьшить накладные расходы на виртуализацию. Intel GVT-g требует наличия графического адаптера Intel как минимум четвертого и выше поколений. Это все GPU от Intel выпущенные за последние 2 года.

1.7. Сравнение

	Emulation	Paravirt	Nvidia Grid	PCI path-through	Intel GVT
ВМ одновременно	Ограничено произв.	Ограничено произв.	До 8	1	До 4
Производительность в % от физической	-	20%-50%	Близко к физической	95%	80%-90%
Необходимость аппаратной поддержки	-	-	+	+	-
Вывод ВМ на разные дисплеи	+/-	+/-	-	-	-
Использование оригинального драйвера	-	-	+/-	+	+

Рис. 4: Результат сравнения

По результатам сравнения было выяснено, что любая из этих технологий, кроме эмуляции, обеспечивает ускорение различных графических задач в виртуальной машине. Тем не менее не смотря на то, что производительности виртуального графического адаптера хватает для обеспечения плавной работы, возникают проблемы при выводе изображения. В большинстве случаев при отрисовке экрана виртуальной машины меняется только его небольшая часть, и нужно на вывод передать только это изменение. Но в случае запуска в виртуальной машине приложений, которые значительно меняют содержимое экрана от кадра к кадру, требуется передавать значительный объем данных. В случае удаленного доступа единственное, что можно сделать - это сжимать изображение. Тем не менее некоторая нагрузка на центральный и графический процессор сохраняется.

Из всех рассмотренных технологий только mediated pass-through и PCI pass-through могут выводить изображение напрямую на экран, минуя хостовую ОС. Но при наличии одного графического процессора применить технологию PCI pass-through не представляется возможным.

2. Постановка Задачи

В результате сравнения разных технологий создания виртуального графического процессора для использования в поставленных целях была выбрана технология виртуализации графики Intel. Но она не позволяла выводить разные ВМ на разные дисплеи одновременно. Таким образом дальнейшей задачей стало:

- Изучение технологии виртуализации графики Intel
- Обеспечение вывода разных ВМ на разные физические дисплеи и поддержка изменения настроек из ВМ.
- Реализация динамического переключения физического дисплея между виртуальными машинами
- Тестовое внедрение в уже существующую инфраструктуру.

3. Архитектура виртуализации графики Intel

Как было сказано ранее Intel GVT-g реализует архитектуру mediated pass-through

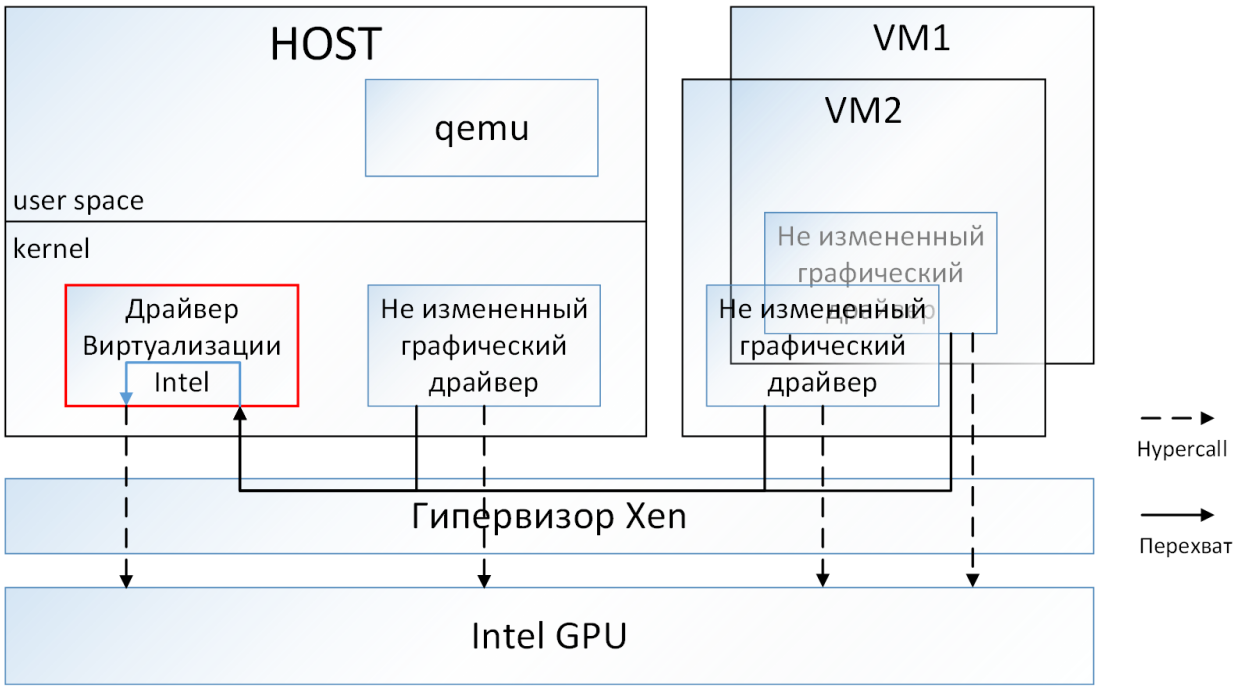


Рис. 5: Архитектура Mediated pass-through

Каждой виртуальной машине доступна часть критичных к производительности ресурсов без участия гипервизора. Привилегированные операции, которые значительно меняют состояние устройства, перехватываются гипервизором и направляются на обработку медиатором для эмуляции. Медиатор создает образ виртуального графического процессора для каждой виртуальной машины и эмулирует привилегированные команды в виртуальном образе. Медиатор периодически переключает контекст работы физического графического процессора между виртуальными. Intel GVT-g это реализация медиатора в виде драйвера хостовой ОС. При этом с точки зрения медиатора хостовая ОС является такой же виртуальной машиной как и все остальные. Это позволяет сделать архитектуру медиатора более однородной и независимой.

Следующие ресурсы, для которых важна производительность, частично выделяются каждой виртуальной машине:

- Часть глобальной графической памяти
- Виртуальная графическая память процессов, своя для каждой ВМ
- Выделенный буфер команд для каждой ВМ

Это минимизирует вмешательство гипервизора в виртуальную машину в критической части рендеринга изображения. Виртуальные машины могут подавать команды на исполнение параллельно и независимо, хотя физический GPU будет выполнять их по очереди.

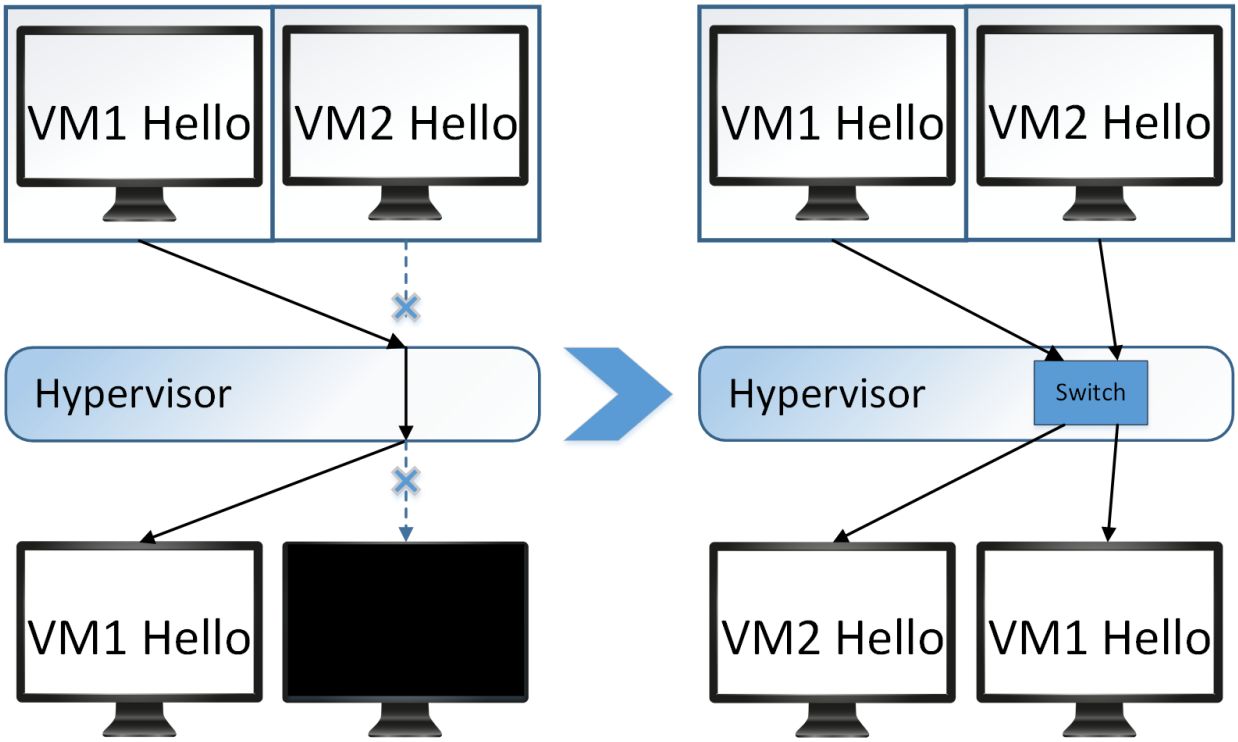
Другие операции перехватываются гипервизором и эмулируются в медиаторе, среди них:

- Регистры отображенные в память
- Конфигурационные регистры PCI
- Таблицы трансляции графической памяти
- Отправка графических команд из буфера на исполнение

Медиатор планирует использование графического процесора между виртуальными машинами и переключает контекст физического GPU между ними. Для виртуализациии VGA режима используется эмулятор QEMU[1].

Для непосредственного изучения возможностей данной технологии использовались следующие версии ее реализации Q4.2014 и Q1.2015 для гипервизора Xen[15]. В качестве хостовой ОС использовалась Ubuntu 14.04 на основе ядра Linux. В процессе использования были проведены тесты на производительность этого решения. Сравнивались результаты полученные в виртуальном окружении и в реальном. В итоге были получены следующие результаты.

- В случае одной виртуальной машины производительность составляла от 85% до 90% от реальной, в зависимости от задачи. При этом нагрузка на ЦП незначительно отличалась от фоновой.
- В случае запуска нагрузки в нескольких виртуальных машинах одновременно суммарная производительность этих виртуальных машин составляла 80-85%. При этом нагрузка на ЦП была 4-5%.



(а) (б)

Рис. 6: Постановка задачи

Тем не менее при попытке применить эту технологию для решения поставленной цели было выяснено, что текущая реализация не поддерживает параллельный вывод изображения из разных виртуальных машин на разные физические дисплеи. То есть если физически подключено два дисплея, и запущено две виртуальные машины, к каждой из которых было подключено по одному дисплею, то изображение будет либо на одном физическом дисплее, либо на другом, но не на обоих сразу(Рис. 6а). Также любые изменения настроек в драйвере в виртуальной машине, не давали эффекта на итоговом изображении.

Для решения поставленной задачи необходимо было реализовать поддержку вывода различных виртуальных экранов на различные физические дисплеи(Рис 6б).

Для дальнейшего объяснения потребуется знать устройство GPU. Его описание можно найти в Intel Programmers Reference Manual 2013[6]. Большая часть последующей информации была взята их этой документации.

3.1. Архитектура Intel GPU

В грубом приближении любой GPU состоит из 3 основных частей:

- Механизм рендеринга изображения
- Механизм вывода изображения
- Графическая память

Их взаимоотношение можно видеть на рис. 7. XenGT в исходном варианте реализует виртуализацию доступа к механизму рендеринга изображения и к графической памяти.

3.2. Виртуализация GPU

Для виртуализации памяти медиатор перехватывает изменение из виртуальных машин таблиц трансляции графической памяти таким образом, что каждой машине выделяется часть этой памяти.

Для виртуализации доступа к механизму рендеринга изображения используется встроенный механизм переключения контекста графического процессора, который применяется в графическом драйвере для организации многозадачности при работе с графическим процессором.

3.3. Механизм вывода изображения

При выводе изображения в физический видео-порт механизм вывода изображения, реализованный в GPU Intel, производит значительное

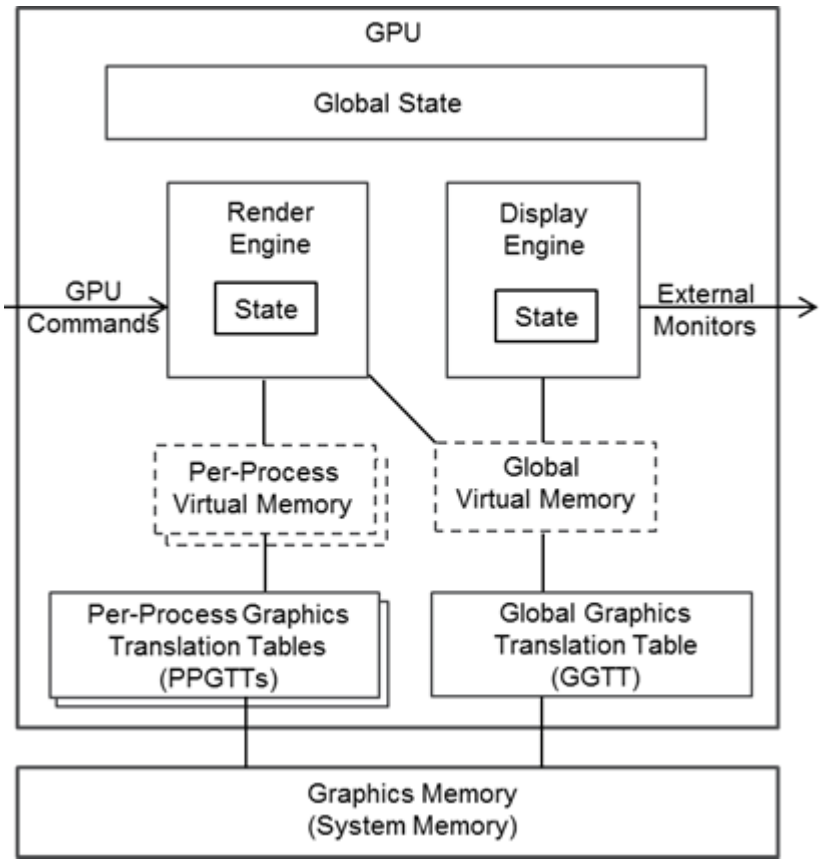


Рис. 7: Архитектура Intel GPU

количество операций. Различные компоненты, реализующие эти операции, и их связь можно видеть на схеме. Прежде чем изображение будет передано на физический дисплей механизм вывода изображения производит следующие операции.

- Преобразование адреса. В памяти виртуальной машины изображение, подготовленное для вывода, хранится в измененном виде. Оно разбито на блоки высотой в 8 строк и размером в 4Кб - страницу памяти. Также применяется подмена адреса на уровне блоков в 64 байта. Это обеспечивает возможность использовать разные каналы памяти более равномерно, что увеличивает скорость доступа(Pipe).

- Компоновка итогового изображения из 3-х частей - основная поверхность(Pipe), итоговый спрайт, курсор.

- Постобработка. Состоит из преобразования цветового простран-

ства и гамма коррекции(Pipe).

- Изменение масштаба(Panel fitter).
- Конвертация в соответствующий протокол HDMI/DisplayPort(Transcoder
- Непосредственно передача по физическому каналу(Port).

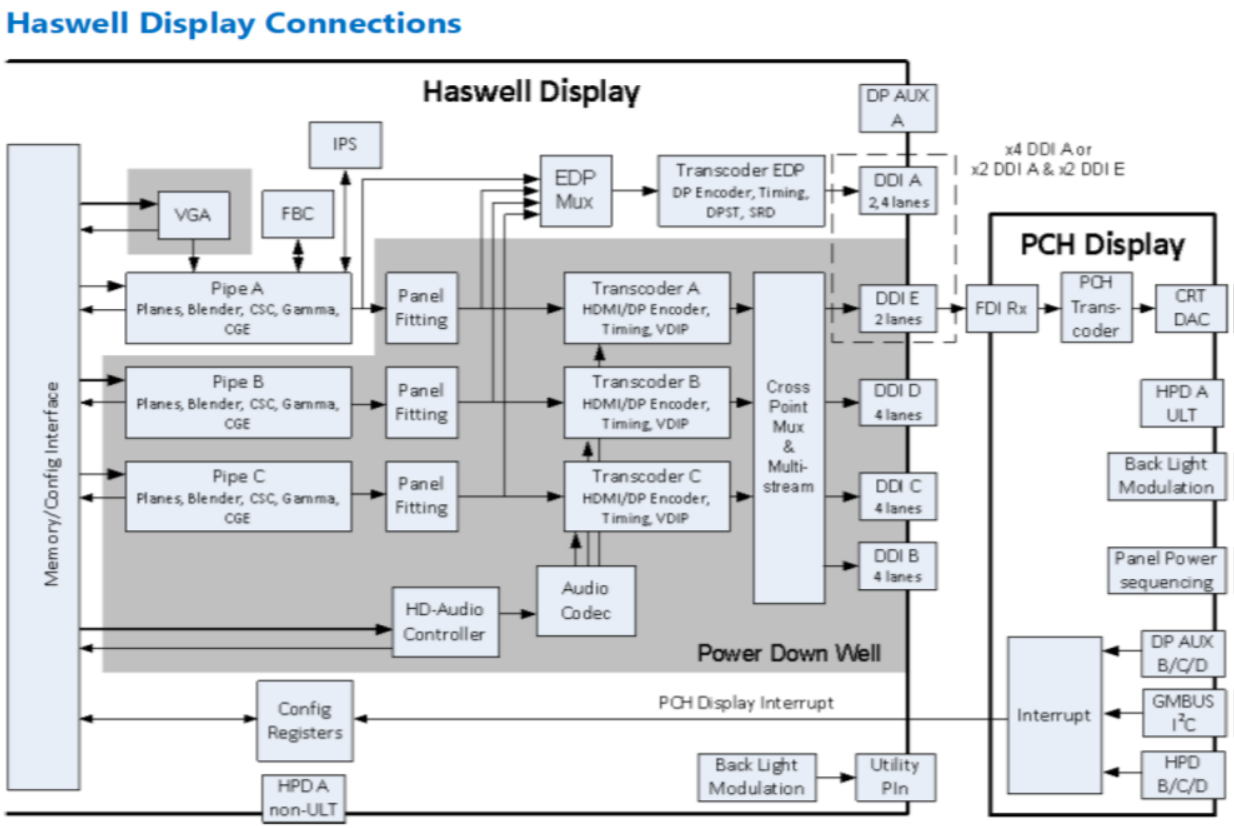


Рис. 8: Архитектура механизма вывода изображения

Помимо этого параллельно и независимо происходит синхронизация драйвера и этих компонентов.

4. Виртуализация механизма вывода изображения

Для реализации виртуального механизма вывода изображения можно рассмотреть два подхода.

1. Полную виртуализацию
2. Проброс части механизма, необходимой для работы одного конкретного видео-выхода, в виртуальную машину.

В первом случае можно получить значительно более гибкое решение за счет того, что можно будет средствами хостовой ОС полностью контролировать как выводить изображение. Таким образом, если можно будет вывести экран виртуальной машины, как обычное изображение, то его можно будет выводить как обычное приложение.

Но при попытке реализации данного подхода были выявлены следующие проблемы

- Изображение хранится в той части памяти, к которой имеет доступ физическое устройство - графический адаптер. По этому Xen не позволяет иметь доступ к одной и той же странице памяти из разных ВМ
- Не смотря на то, что Xen позволяет хостовой ОС читать любую физическую память, ядро Linux не позволяет добавить эти страницы в виртуальную память. По-этому приложению, работающему в пространстве пользователя невозможно дать доступ к этой памяти.
- Даже если бы была возможность дать приложению непосредственный доступ к изображению экрана виртуальной машины, это изображение необходимо обработать в связи с особенностями его хранения. Это потребовало бы некоторого времени.

- Изменение кадрового буфера может происходить до 60 раз в секунду. И каждый раз нужно переконфигурировывать доступ к нему, что так-же требует некоторых ресурсов.

По этим причинам у данного подхода нет никаких преимуществ по сравнению с любой технологией удаленного доступа.

В итоге был реализован второй вариант. Он позволяет достичь следующих преимуществ.

- Нет дополнительной нагрузки на сеть и ЦП
- Отсутствуют дополнительные задержки при выводе изображения

4.1. Схема виртуализации

В некотором виде данный подход уже был реализован в существующей версии драйвера виртуализации. Но как уже было сказано ранее, он не позволял выводить изображение из разных виртуальных машин на разные физические дисплеи. Это происходило потому, что исходная реализация виртуализации механизма вывода изображения в один момент времени может работать только с одной виртуальной машиной.

Но если посмотреть на компоненты, используемые для работы одного дисплея, можно заметить, что для каждого дисплея используется по одному Pipe, Transcoder и Port. При этом Port настраивается один раз и практически не меняется в процессе работы. Таким образом если выделить каждому виртуальному дисплею по Pipe и Transcoder, то есть дать драйверу в виртуальной машине доступ к изменению конфигурации соответствующих компонентов, то виртуальные машины смогут выводить изображение независимо, каждая на свой дисплей. Например, можно дать одной ВМ доступ к Pipe A и Transcoder A, а другой к Pipe C и Transcoder C(рис. 9). И затем направить выход из соответствующего Transcoder в нужный нам Port.

Тем не менее возникает следующая проблема. Драйвер виртуальной

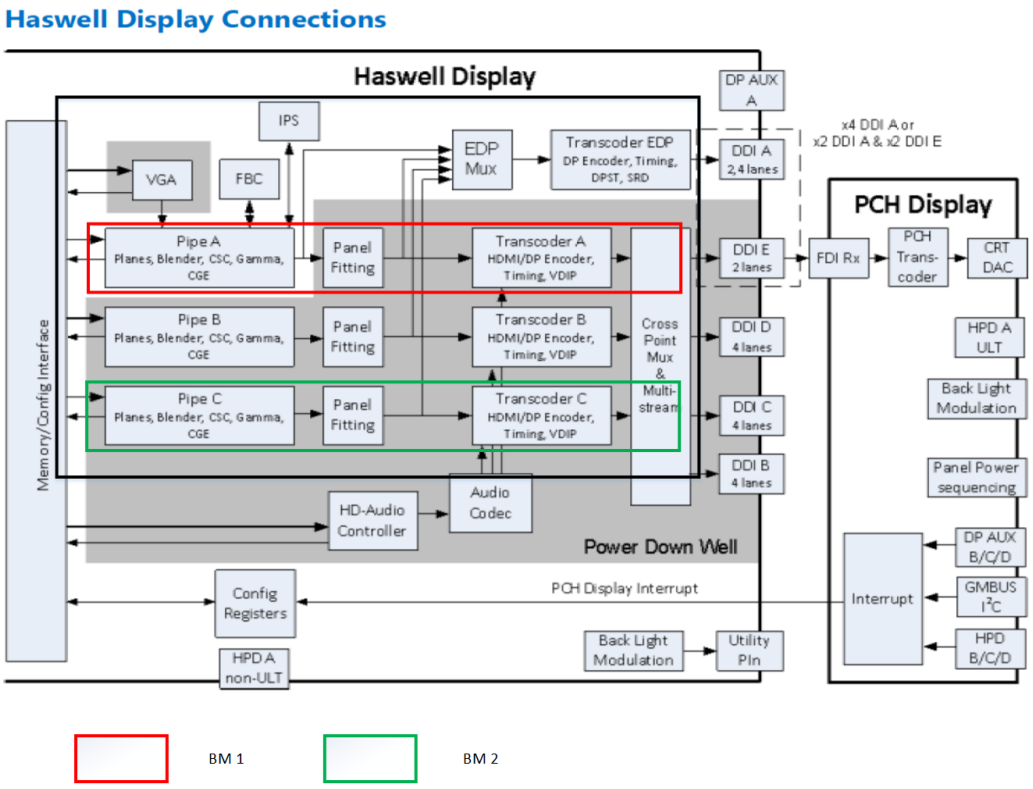


Рис. 9: Архитектура display engine

машины стремится использовать сначала Pipe A, по-этому требуется не только разделить доступ, но и перенаправить его. То есть работа драйвера первой ВМ с виртуальным Pipe A должна отражаться на физическом Pipe A. А работа второй ВМ со своим виртуальным Pipe A должна отражаться на другом Pipe, например, Pipe B. Тоже самое для Transcoder. При этом нужно правильно перевести виртуальные соединения различных компонентов в физические.

4.1.1. MMIO

Конфигурация этих компонентов осуществляется через отображенные в память регистры В процессе работы было рассмотрено около 30 различных конфигурационных регистров. Хотя виртуализация некоторых из них уже была сделана, но она не позволяла перенаправлять доступ к регистрам соответствующим образом. Также для некоторых регистров виртуализация отсутствовала полностью, по этому она была также реализована, что позволило управлять многими характеристи-

ками изображения из виртуальной машины.

4.1.2. Interrupts

Для реализации синхронизации с дисплеем используется вертикальная синхронизация. Графический процессор для синхронизации использует прерывания. Для реализации виртуальных прерываний требовалось те прерывания, которые идут от физических дисплеев, перенаправить в виртуальные машины, при этом сохранить существующую генерацию виртуальных прерываний.

4.2. Динамическое переключение

Для реализации динамического переключения активной ВМ на конкретном физическом дисплее было реализовано изменение всех необходимых регистров на уровне отдельных компонентов в соответствии с документированными процессами их настройки.

4.3. Управление и настройка

Для управления, наблюдения и настройки XenGT использует интерфейс ядра Linux - Sysfs.

Sysfs - это виртуальная файловая система, созданная для предоставления информации о различных устройствах доступных системе и для изменения их конфигурации.

Оригинальный Sysfs интерфейс XenGT поддерживал управление портами только для создания виртуального порта. В процессе работы для полного управления и динамического переключения дисплеев между виртуальными машинами он был расширен. Были добавлены следующие возможности:

- Вывод информации о том, какая виртуальная машина является активной на каждом из физических портов.
- Вывод и изменение активного виртуального дисплея на каждом из физических портов.

- Изменение активного виртуального дисплея на следующий в очереди. Это позволяет использовать сочетание клавиш для удобного переключения между виртуальными машинами.

Таким образом можно независимо управлять выводом каждого виртуального дисплея на любой физический.

4.4. Отладка и тестирование

Отладка осложняется тем, что в данном случае практически невозможно использовать пошаговую отладку и точки останова. Это связано с тем, что графический драйвер виртуальной машины работает параллельно с драйвером виртуализации, при этом он ожидает реакцию на взаимодействие с графическим адаптером в течение разумного времени, которой не произойдет, в случае использования отладчика.

По этому для отладки используется лог ядра и интерфейс Debugfs.

Для тестирования требовалось создать множество конфигураций с различными виртуальными машинами использующими различные физические дисплеи. Разработка и тестирование производились на следующих тестовых стендах: Ноутбук Lenovo Thinkpad T440s в следующей конфигурации:

- Intel Core i5-4300U
- Intel HD graphics 4400
- Видео-выходы: Internal DisplayPort, 2 external DisplayPort
- Ubuntu 14.04 с гипервизором Xen

Настольный ПК:

- Материнская плата Asus Z97-Z
- Intel Core i5-4590
- Intel HD graphics 4600

- Видео-выходы: VGA, DisplayPort, HDMI, DVI-D
- Ubuntu 14.04 с гипервизором Xen

В качестве виртуальных машин использовались Windows 7 Professional, Windows 8.1 Professional, Ubuntu 14.04 Desktop.

4.5. Вывод

Таким образом в результате получена система, которая позволяет на одном ПК работать с двумя ОС одновременно таким образом, что со стороны пользователя эти системы выглядят также, как и системы запущенные на различных физических машинах. Это происходит благодаря тому, что графический процессор работает с информацией в виртуальной машине напрямую, практически без вмешательства. При этом при нагрузке производительность может как делиться между машинами, так и выделяться конкретной ВМ полностью.

5. Тестовое внедрение

Для тестирования и изучения дальнейших перспектив разработки было проведено внедрение в существующую инфраструктуру. Для этого потребовалось сделать следующие вещи.

- Миграция операционной системы в виртуальное окружение
- Интеграция с системой шифрования McAfee Endpoint Encryption[7]
- Настройка сетевой инфраструктуры.
- Сохранение OEM активации в виртуальном окружении.

Миграция ОС и интеграция шифрования, благодаря использованию не измененных драйверов операционной системы, прошла совершенно незаметно для операционной системы.

Для интеграции в сетевую инфраструктуру потребовалось настроить трансляцию сетевых адресов(NAT) и сервер протокола DHCP. Конфигурация NAT осуществлялась с помощью iptables. Из внешней сети система должна выглядеть не измененной. Для этого при открытии соединения из вне оно для большинства портов автоматически перенаправлялось в виртуальную машину.

OEM активация продуктов в ОС использует стандарт ACPI[16]. Поэтому для сохранения OEM активации в виртуальном окружении нужно дать доступ виртуальной машине к ACPI таблице, хранящей соответствующую информацию - SLAT таблице. Для этого был незначительно изменен виртуальный BIOS. Таким образом виртуальный BIOS добавляет эту таблицу в виртуальный ACPI интерфейс и ВМ получает доступ к SLAT таблице.

Поскольку на данный момент в гипервизоре Xen динамическое подключение USB устройств к виртуальной машине находится в процессе разработки, отсутствие этой функциональности приводит к некоторым неудобствам при использовании системы.

Заключение

В рамках данной работы были достигнуты следующие результаты:

- Проведено исследование и сравнение существующих технологий создания виртуального графического адаптера. Выбрана наиболее подходящая для решения поставленной задачи.
- Добавлена необходимая дополнительная функциональность.
- Проведено тестирование новой функциональности в различных физических и виртуальных окружениях.
- Проведено тестовое внедрение в существующую ИТ инфраструктуру

Список литературы

[1] Bellard F. QEMU, a Fast and Portable Dynamic Translator. — Usenix Technical Conference, pages 47–60, Anaheim, CA, 2005.

[2] Burger T. Intel Virtualization Technology for Directed I/O (VT-d): Enhancing Intel platforms for efficient virtualization of I/O devices. — 2012. — URL: https://software.intel.com/en-us/articles/intel-virtualization-technology-for-directed-io-vt-d-enhancing-ir

[3] Chromium - Interactive cluster rendering system. — URL: http://chromium.sourceforge.net/.

[4] Dowty Micah, Sugerman Jeremy, VMware Inc. GPU Virtualization on VMware's Hosted I/O Architecture. — URL: http://www.vmware.com/appliances/.

[5] Gallium3D Technical Overview. — URL: http://www.freedesktop.org/wiki/Software/gallium/.

[6] Intel Graphics for Linux - Driver Documentation - PRMs. — URL: https://01.org/linuxgraphics/documentation/driver-documentation-prms.

[7] McAfee. Endpoint Security Protection. — URL: http://www.mcafee.com/us/products/endpoint-protection/index.aspx.

[8] Microsoft. DirectX. — URL: http://www.microsoft.com/windows/directx/default.mspx.

[9] NVidia GRID. Network-delivered GPU acceleration for games, virtual desktops, and cloud applications. — URL: http://www.nvidia.com/object/nvidia-grid.html.

[10] OpenGL – The Industry Standard for High Performance Graphics. — URL: http://www.opengl.org.

[11] Single Root I/O Virtualization and Sharing Specification, PCI Special Interest Group. — URL: http://www.pcisig.com/specifications/iov.

[12] Tian Kevin. Intel Graphics Virtualization Technology. — URL: https://01.org/igvt-g.

[13] VMM-Independent Graphics Acceleration / H. Andres Lagar-Cavilla, Niraj Tolia, M. Satyanarayanan, Eyal de Lara. — 2015.

[14] VMware. VMware ESXi. — URL: http://www.vmware.com/resources/techresources/1009.

[15] The Xen Project, the powerful opensource industry standart for virtualization. — URL: http://www.xenproject.org/.

[16] forum UEFI. Advanced Configuration Power Interface. — URL: http://www.uefi.org/acpi/specs.