Национальный исследовательский университет ИТМО

Факультет программной инженерии и компьютерной техники

Лабораторная работа №1

по дисципение Проектирование вычислительных систем

Студент: Саржевский Иван

Группа: P3402

г. Санкт-Петербург

2020 г.

Проект

Radeon Asm Debugger Extension for Visual Studio

https://github.com/vsrad/radeon-asm-tools

Краткое описание проекта в свободной форме

Хотя высокоуровневые GPGPU языки достаточны для большинства задач, некоторые возможности железа можно раскрыть лишь на уровне инструкций. Программирование на ассемблере осложняется отсутствием удобных средств разработки.

Настоящий дебаг шейдерного кода довольно сложен, как минимум, он требует поддержки со стороны железа. Зачастую такой роскоши у разработчика нет. Поэтому можно воспользоваться псевдо дебагером.

Почему псевдо? Для каждого брейкпойнта в реальности запускается шейдер каждый раз заново.

Таким образом, мы можем вставлять на место брейкпоинта кусок кода, который сдампит необходимые значения, а затем остановит выполнение шейдера и можно обойтись даже без поддержки дебага от производителей железа.

Чтобы автоматизировать работу дебаг вставки и визуализировать информацию в удобном формате были разработаны плагины для VS.

Плагин предполагает наличие ремоут машины т.к. зачастую случается так, что при некорректном поведении шейдера (для диагностики которого и нужна отладка) машина, которая выполняет шейдер, утрачивает работоспособность, необходимо заново запускать машину, терять время и несохраненные изменения. Также один и тот же шейдер необходимо запускать на разных машинах с различной конфигурацией железа. Поэтому была выбрана концепция host-remote, при которой на стороне хоста находится код шейдера и список профилей (об этом дальше). Но при этом локальный хост также может выступать как ремоут машина.

Жизненный цикл

1. Планирование

   Роль в проекте: Оценка требований, анализ существующих подобных систем, оценка трудоемкости

   Методы и средства: Ознакомление с кодом других проектов на `github`, комуникация с заказчиком, изучение существующей системы отладки

2. Проектирование системы

   Роль в проекте: Определение архитектурных проблем и принятие решений по ним, построение архитектуры всех компонентов системы и архитектуры системы в целом

   Методы и средства: Создание entity-диаграммы, UML-диаграммы, прецедентов

3. Разработка мокапов интерфейсов

   Роль в проекте: Разработка мокапов, согласно которым будут верстаться пользовательские интерфейсы согласно требованиям системы и из соображений эргономики

   Методы и средства: Инструменты прототипирования

4. Реализация

   (a) Реализация сервера

   (b) Реализация расширения для VS

   Роль в проекте: Каждая часть системы должна быть реализована и оттестирована

   Методы и средства: Написание кода, Unit-тесты

5. Релиз

   Роль в проекте: Должна быть построена релизная сборка, которая должна быть протестирована на всех целевых платформах, написан change log, все это должно быть оформлено в виде github-релиза с версией отвечающей дате релиза

   Методы и средства: Ручное тестирование, github

6. Анализ текущего релиза

   Роль в проекте: Получение новых feature-реквестов и баг репортов

   Методы и средства: github issues

7. Планирование следующего релиза

   Роль в проекте: Определение списка новых опций и багов, которые необходимо исправить для того, чтобы произвести новый релиз

   Методы и средства: github issues, системы планирования задач

   После этого возвращаемся на пункт 4, реализация новых опций с последующим релизом

Архитектурные проблемы

1. Выбор протокола передачи данных между хостом и сервером.

   Так как клиент и сервер находятся физически на разных машинах, необходимо определить протокол передачи сообщений между ними. В качестве протокола был выбран `TCP`, так как он позволяет передавать сообщения по сети и имеет подтверждение факта приема/передачи пакетов, что обеспечивает надежность.

2. Тип протокола взаимодействия расширения и сервера: `Stateful/Stateless`.

   Изначально было принято решение четко определить последовательность сообщений, которыми хост и сервер обмениваются в процессе отладки, однозначно определив сессию и, соответственно, сервер был спроектирован как `stateful` компонент. Однако в процессе поддержки появлялось все больше различных сценариев взаимодействия пользователя с системой, что требовало добавления все новых состояний и типов пользовательских сессий, что значительно затрудняло разработку и увеличивало количество возможных багов. Это подтолкнуло нас к редизайну сервера, теперь это `stateless` компонент, который обслуживает несколько базовых команд.

3. Пользовательский интерфейс конфигурации шагов отладки.

   Одним из требований системы было наличие т. н. "Профилей" отладки. Профиль включал в себя окружение и конфигурацию для основных шагов - дебага, запуска препроцессора, профайлера и дизассемблера. Благодаря наличию профилей можно быстро переключаться между целевыми платформами для отладки. Однако в процессе использования данный подход зарекомендовал себя как недостаточно гибкий - для более сложных сценариев требовалось реализовывать логику в вызываемом скрипте (так как каждый шаг вызывал ровно один скрипт), что приводило к неудобству в работе со стороны пользователя. Было принято решение внести изменение в дизайн профилей - отныне каждое детерменированное действие может строиться из набора простых действий в произвольном порядке. Помимо этого, пользователь теперь может определять кастомные действия и вызывать их в любой момент. Профиль теперь определяет совокупность всех определенных действий. Данное архитектурное решение позволило добиться необходимой гибкости работы пользователя при относительной простоте настройке расширения.

4. Выбор оптимального интерфейса представления данных с учетом специфики системы.

   Так как отладчик предназначен, в первую очередь, для отладки шейдеров на GPU, привычные интерфейсы для отображения значений переменных использовать невозможно. Дело в том, что из-за высокой степени параллелизма у каждого потока будет свое значение заданной переменной. Для решения этой проблемы было решено разработать собственный интерфейс отображения переменных, который представляет собой таблицу, строки которой являются переменной, а каждый столбец представляет один лейн.

5. Поддержка конфигураций нескольких проектов в рамках одного решения.

   VS поддерживает одновременную работу с несколькими проектами в рамках одного решения. Необходимо было сделать выбор каким образом это обрабатывать. Проще было бы иметь один сет профилей для всего решения, однако для большей гибкости настройки было принято решение сохранить сет профилей за каждым проектом. Активный сет профилей при этом определяется текущим `startup project` в решении.

Вывод

В ходе выполнения ланной лабораторной работы были приобретены навыки анализа архитектурных проблем, а также краткого описания системы и её жизненного цикла.