«Санкт-Петербургский государственный электротехнический университет «ЛЭТИ» им. В.И.Ульянова (Ленина)»
(СПбГЭТУ «ЛЭТИ»)

Направление	01.03.02 – Прикладная математика и информатика
Профиль	Математическое и программное обеспечение вычислительных машин
Факультет	КТИ
Кафедра	МОЭВМ

К защите допустить

Зав. кафедрой Кринкин К.В.

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА БАКАЛАВРА

Тема: ОТЛАДЧИК ДЛЯ OPENCL

Студент		_____ *подпись*	Лосев М.Л.
Руководитель	доцент *(Уч. степень, уч. звание)*	_____ *подпись*	Беляева С.А.
Консультанты	ст. преподаватель *(Уч. степень, уч. звание)*	_____ *подпись*	Калишенко Е.Л.
	(Уч. степень, уч. звание)	_____ *подпись*	Лукин М.И.

Санкт-Петербург

2021

ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ

Утверждаю
Зав. кафедрой МОЭВМ
_____ Кринкин К.В.
«___»_____2021 г.

Студент Лосев М.Л. Группа 7383

Тема работы: Разработка отладчика для OpenCL

Место выполнения ВКР: СПбГЭТУ «ЛЭТИ», кафедра МО ЭВМ

Исходные данные (технические требования):
Разработка OpenCL-отладчика с функцией инспекции переменных

Содержание ВКР:
Обзор предметной области, формулировка требований к программе, описание решения, беспечение качества разработки, продукции, программного продукта

Перечень отчетных материалов: пояснительная записка, иллюстративный материал

Дополнительные разделы: Обеспечение качества разработки, продукции, программного продукта

Дата выдачи задания
«___»_____20__ г.

Дата представления ВКР к защите
«___»_____20__ г.

Студент _____ Лосев М.Л.

Руководитель доцент _____ Беляев С.А.
(Уч. степень, уч. звание)

Консультант _____ Лукин М.И.
(Уч. степень, уч. звание)

КАЛЕНДАРНЫЙ ПЛАН ВЫПОЛНЕНИЯ
ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЫ

Утверждаю
Зав. кафедрой МОЭВМ
_____ Кринкин К.В.
«___»_____20__ г.

Студент Лосев М.Л. Группа 7383

Тема работы: Разработка отладчика для OpenCL

№ п/п	Наименование работ	Срок выполнения
1	Обзор литературы по теме работы	11.02 – 05.03
2	Формулировка требований к программе	06.03 – 08.03
3	Разработка и отладка программы	09.03 – 15.05
5	Оформление пояснительной записки	16.05 – 24.05
6	Оформление иллюстративного материала	25.05 – 27.05

Студент _____ Лосев М.Л.

Руководитель _____ Беляев С.А.
(Уч. степень, уч. звание)

Консультант _____ Лукин М.Ю.
(Уч. степень, уч. звание)

РЕФЕРАТ

Пояснительная записка 45 стр., 5 рис., 3 табл., 9 ист., 10 прил.

OPENCL, ОТЛАДЧИК, Clang

Объектом исследования является алгоритм отладчика для модификации OpenCL-программы.

Цель работы - создание отладчика для OpenCL, работающего со всеми поддерживаемыми устройствами, который позволяет пользователю видеть на каждом шаге исполнения программы построчно состояние регистров выбранного рабочего элемента (ядра GPU) и локальной памяти (кэш GPU).

В настоящей работе были рассмотрены и сопоставлены существующие отладчики для OpenCL, существующие подходы к разработке отладчиков. Был разработан высокоуровневый кроссплатформенный устройствонезависимый отладчик для OpenCL-приложений, реализующий возможности инспекции состояния переменных программы в точке останова. Он обладает интерфейсом командной строки и предполагает возможность интеграции со средами разработки.

ABSTRACT

In this graduate work, the existing debuggers for OpenCL and the existing approaches to the development of debuggers were considered and compared. A high-level cross-platform device-independent debugger for OpenCL applications was developed, which implements the ability to inspect the state of program variables at a breakpoint. It has a command line interface and assumes the ability to integrate with development environments.

СОДЕРЖАНИЕ

Определения, обозначения и сокращения 7
Введение 8
1. Обзор предметной области 9
1.1. OpenCL 9
1.1.1. Модель платформы 9
1.1.2. Модель выполнения 10
1.1.3. Модель памяти 11
1.1.4. Язык программирования OpenCL 12
1.2. Отладчики 14
1.2.1. Классификация отладчиков 14
1.3. Обзор аналогов 15
1.2.1. Критерии сравнения 15
1.2.2. Существующие OpenCL-отладчики 15
1.2.3. Выводы по итогам сравнения 16
2. Формулировка требований к программе 17
2.1. Устройствонезависимость 17
2.2. Возможность интеграции 17
2.3. Интерфейс командной строки 17
2.4. Высокоуровневость 17
2.5. Платформонезависимость 18
3. Описание решения 19
3.1. Структура программной реализации 19
3.1.1. Структуры данных 19
3.1.2. Передача состояния устройства 23
3.1.3. Получение состояния устройства 28
3.2. Интерфейс пользователя 29
3.2.1. Формат входных данных приложения 30
3.2.2. Формат выходных данных приложения 30
4. Обеспечение качества разработки, продукции, программного продукта (Доп. раздел) 32
4.1. Юнит-тесты 32
4.1.1. Тестирование парсинга данных 32
4.1.2. Тестирование парсинга типов 40
4.2. Интеграционные тесты 44
Заключение 46
Список использованных источников 47
Приложение А. Код модуля Primitives 48
Приложение Б. Код модуля SourceProcessor 57
Приложение В. Код модуля OclSourceProcessor 59
Приложение Г. Код модуля LineInserter 60
Приложение Д. Код модуля KernelProcessor 61
Приложение Е. Код модуля OclDebugger 62

Приложение Ж. Код модуля filters **67**
Приложение З. Код модуля VariableTest **68**
Приложение И. Код модуля VarDeclarationTest **74**
Приложение К. Код модуля OclDebuggerTest **77**
Приложение Л. Код файла kernel.cl **80**

ОПРЕДЕЛЕНИЯ, ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ

В настоящей пояснительной записке применяют следующие термины с соответствующими определениями:

GPU – графический процессор (англ. Graphics Processing Unit)

SIMD – параллелизм на уровне данных (англ. Single Instruction Multiple Data)

CLI – интерфейс командной строки (англ. Command Line Interface)

LLVM – низкоуровневая виртуальная машина (англ. Low Level Virtual Machine)

AST – абстрактное синтаксическое дерево (англ. Abstract Syntax Sree)

DFS – поиск в глубину на графе (англ. Depth-First Search)

API – программный интерфейс приложения (англ. Application Program Interface)

TDD – разработка через тестирование (англ. Test-Driven Development)

JSON – объектная нотация ЖаваСкрипт (англ. JavaScript Object Notation)

ОС – операционная система (англ. Operating System)

ВВЕДЕНИЕ

Компания AMD прекратила поддержку инструмента разработки CodeXL для отладки и профилирования программ, использующих GPU, в 2019 году. Из-за этого отладка на новых устройствах, таких как Radeon RX 5700, невозможна. Поэтому нужны новые инструменты разработки под AMD-GPU.

Целью является создание отладчика для OpenCL, работающего со всеми поддерживаемыми устройствами, который позволяет пользователю видеть на каждом шаге исполнения программы построчно состояние регистров выбранного рабочего элемента (ядра GPU) и локальной памяти (кэш GPU).

Объектом исследования является алгоритм отладчика для модификации OpenCL-программы.

Предмет исследования – применение алгоритмов для получения состояния устройства (регистров и памяти) с целью его отображения.

Задачи:

1. Изучить подходы к разработке отладчиков.
2. Разработать алгоритм получения состояния устройства ("backend").
3. Реализовать вывод данных о состоянии устройства ("frontend").

Практическая значимость работы заключается в том, что результат работы сможет использоваться для повышения эффективности разработки OpenCL-приложений.

1. ОБЗОР ПРЕДМЕТНОЙ ОБЛАСТИ

1.1. OpenCL

OpenCL – это открытый стандарт параллельного программирования общего назначения на центральных, графических и иных процессорах [1]. Он включает в себя программный интерфейс приложений (API) для управления параллельными вычислениями на гетерогенных платформах и устройствах, а также одноименный язык программирования OpenCL [2].

1.1.1. Модель платформы

Схема модели платформы приведена на рисунке 1.1. Модель включает в себя одно или несколько устройств (Compute device), управляемых хостом (Host). Устройства состоят из одного или более вычислительных элементов (Compute units), которые делятся на один или несколько рабочих элементов (Processing element). Множественность рабочих элементов обеспечивает обработку данных по принципу SMID. Вычисления на устройстве выполняются рабочими элементами [3].

OpenCL-приложения выполняются на хосте в соответствии с моделью исполнения системы хоста (например, это могут быть исполняемые .exe-файлы для платформ семейства Windows). OpenCL-приложение отправляет команды на устройство, чтобы выполнить на нем параллельные вычисления.

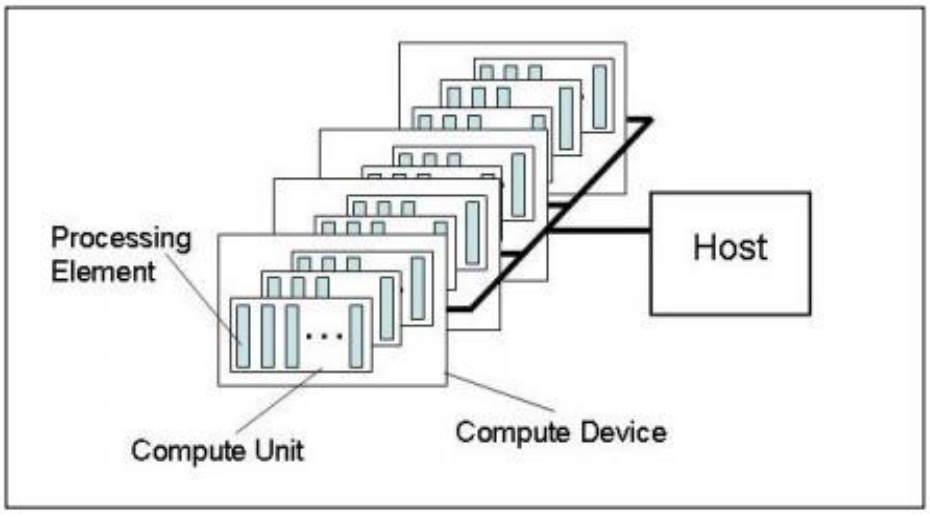

Рисунок 1.1 - Схема модели платформы

1.1.2. Модель выполнения

Выполнение OpenCL-программы происходит в двух частях: ядра (kernel), которое выполняется на одном или нескольких устройствах и хост-программа, которая выполняется на хосте. Хост-программа определяет контекст для ядер и управляет их выполнением.

Когда ядро отправляется хостом на выполнение устройством, определяется индексное пространство. Экземпляр ядра выполняется для каждой точки в этом индексном пространстве. Этот экземпляр ядра называется рабочим элементом (Work item) и определяется своей точкой в индексном пространстве, которая обеспечивает глобальный идентификатор рабочего элемента. Каждый рабочий элемент выполняет один и тот же код, но конкретный путь выполнения кода и обрабатываемые данные могут варьироваться в зависимости от рабочего элемента. Рабочие элементы организованы в рабочие группы (Work groups). Рабочие группы обеспечивают более крупное разложение индексного пространства. Рабочим группам присваивается уникальные идентификаторы с той же размерностью, что и индексное пространство, используемое для рабочих элементов. Заданиям присваивается уникальный локальный идентификатор в рабочей группе, так что отдельный рабочий элемент может быть однозначно идентифицирован по его глобальному идентификатору или комбинации его локального идентификатора и идентификатора рабочей группы. Рабочие элементы в данной рабочей группе выполняют одновременно элементы обработки одного вычислительного блока. Индексное пространство, поддерживаемое в OpenCL, называется NDRange. NDRange – это N-мерное индексное пространство, где N – натуральное число. NDRange определяется целочисленным массивом длины N, определяющим размер индексного пространства в каждом измерении, начиная с индекса смещения F (ноль по умолчанию). Глобальный идентификатор и локальный идентификатор каждого рабочего элемента представляют собой N-мерные кортежи. Глобальный идентификатор компоненты в некотором измерении - это значения в диапазоне от F до F плюс количество элементов в этом

измерении минус один. Рабочим группам присваиваются идентификаторы с использованием подхода, аналогичного тому, который используется для глобальных идентификаторов рабочих элементов. [1]

На рисунке 1.2. показана схема индексации элементов и групп.

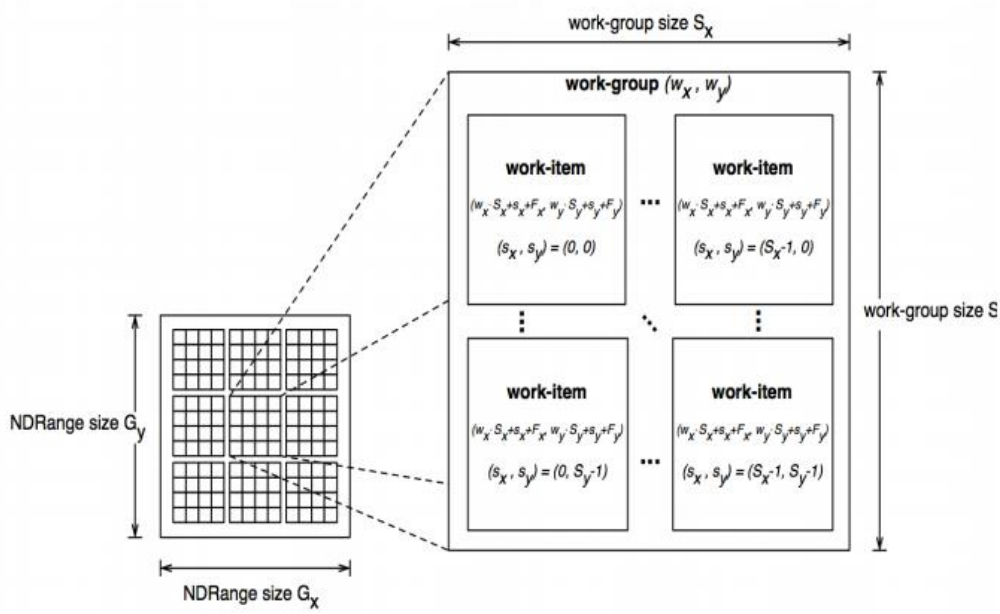

Рисунок 1.2 - Схема индексации элементов и групп

1.1.3. Модель памяти

Рабочие элементы, выполняющие ядро, имеют доступ к четырем отдельным областям памяти:

- Глобальная память. Эта область памяти разрешает доступ для чтения / записи всем рабочим элементам во всех рабочих группах. Рабочие элементы могут читать или записывать в любой элемент объекта памяти. Операции чтения и записи в глобальную память могут кэшироваться в зависимости от возможностей устройства.
- Постоянная память: область глобальной памяти, которая остается постоянной во время исполнения ядра. Хост выделяет и инициализирует объекты памяти, помещенные в постоянную память.

- Локальная память: область памяти, локальная для рабочей группы. Эта область памяти может быть использоваться для распределения переменных, которые являются общими для всех рабочих элементов в этой рабочей группе. Это может быть реализовано в виде выделенных областей памяти на устройстве OpenCL. В качестве альтернативы область локальной памяти может быть отображена на разделы глобальной памяти.
- Частная память: область памяти, принадлежащая рабочему элементу. Переменные, определенные в частной памяти рабочего элемента, не видны другому рабочему элементу.

На рисунке 1.3 приведена схема модели памяти.

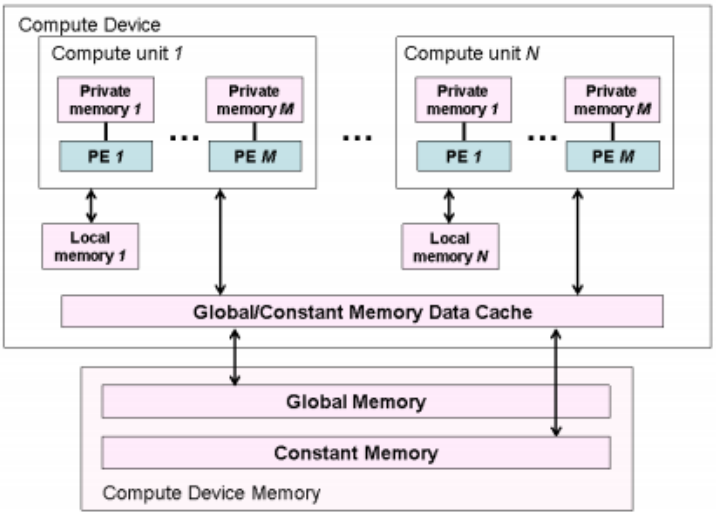

Рисунок 1.3 - схема модели памяти OpenCL

1.1.4. Язык программирования OpenCL

Язык программирования OpenCL C (также называемый OpenCL C) основан на спецификации языка C ISO / IEC 9899: 1999 (он же C99).

Язык поддерживает векторные типы данных, допускаются значения длины вектора: 2, 3, 4, 8 и 16. В таблице 1.1 приведены векторные типы данных.

Таблица 1.1 Векторные типы данных в OpenCL

длина / Базовый тип	2	3	4	8	16
char	char2	char3	char4	char8	char16
uchar	uchar2	uchar3	uchar4	uchar8	uchar16
short	short2	short3	short4	short8	short16
ushort	ushort2	ushort3	ushort4	ushort8	ushort16
int	int2	int3	int4	int8	int16
uint	uint2	uint3	uint4	uint8	uint16
long	long2	long3	long4	long8	long16
ulong	ulong2	ulong3	ulong4	ulong8	ulong16
float	float2	float3	float4	float8	float16
double	double2	double3	double4	double8	double16

Доступ к элементам вектора может осуществляться как к полям структуры через точку. Обращаться к элементам можно двумя способами: как sn для n-го элемента или x, y, z, w для первого, второго, третьего, четвертого элемента.

Так как переменные могут храниться в четырех различных адресных пространствах в соответствии с моделью памяти, перед именем переменной может идти модификатор адресного пространства: __private для приватной памяти, __local для локальной, __global для глобальной и __constant для постоянной. Если модификатор не указан явно, переменная будет размещена в приватной памяти. При объявлении массивов в локальной памяти запрещается использовать списки инициализации. Размерность массивов ограничена числом три. Массивы в глобальной памяти объявлять запрещается.

OpenCL поддерживает встроенные функции (builtins), которые, например, могут ускорять обращения к памяти. Здесь они рассмотрены не будут, так как не имеют отношения к теме.

1.2. Отладчики

Отладка – поиск и устранение ошибок в компьютерной программе. Отладка связана с определением текущих значений переменных и путей выполнения, по которым идет программа.

Отладчик – это программное средство, контролирующее отлаживаемое (целевое) приложение таким образом, что программист может проследить за его выполнением и в выбранной точке посмотреть состояние программы, чтобы проверить ее корректность [4].

1.2.1. Классификация отладчиков

Автономные отладчики – это приложения, предназначенные исключительно для отладки. Примером такого отладчика служит Borland Turbo Debugger.

Интегрированные отладчики включены в среду разработки с целью повышения продуктивности разработчика.

Низкоуровневые отладчики позволяют выполнять пошагово инструкции машинного кода целевого приложения. Поэтому, отладка с их помощью должна производиться отдельно на различных аппаратных средствах. Они не совершают декомпиляцию, поэтому проще в реализации [4].

Высокоуровневые (символьные) отладчики позволяют следить за пошаговым выполнением исходного кода (на языках программирования высокого уровня), абстрагируясь от инструкций процессора. Однако, иногда программисту требуется опуститься на низкий уровень, чтобы понять, как программа выполняется на аппаратном обеспечении. Поэтому символьные отладчики также должны предоставлять низкоуровневую информацию. Обычно это достигается путем получения состояния регистров и дампа памяти. [4]

1.3. Обзор аналогов

1.2.1. Критерии сравнения

Кроссплатформенность. Поскольку OpenCL-программы зачастую компилируются драйвером видеокарты непосредственно перед выполнением, машинный код зависит от драйвера и платформы. Это значит, что в большинстве случаев отладку необходимо выполнять под разными ОС. Будем считать отладчик платформонезависимым, если он представлен в виде работающих дистрибутивов для Windows и Linux.

Устройствонезависимость. Так как OpenCL-программы могут выполняться на большом количестве различных устройств и часто появляются новые устройства, важно, чтобы корректность работы отладчика не зависела от устройства. Работа отладчика не зависит от устройства, если он работает корректно со всеми устройствами, включая новые, и не требует для этого добавления их поддержки от разработчиков.

Возможность интеграции в среды разработки (IDE) требуется в связи с современной практикой разработки, подразумевающей использование IDE. Она повышает удобство и производительность программиста и расширяет круг возможных пользователей.

1.2.2. Существующие OpenCL-отладчики

CodeXL – это отладчик для устройств AMD. Позволяет профилировать программы [5]. Предоставляет информацию о количестве ипользуемых регистров, загруженности шин памяти и т.п. Не поддерживается с 2019 года. Не работает с новыми устройствами. Недоступен в виде дистрибутива. Не работает под линуксами.

Oclgrind – симулятор OpenCL-устройства (использует виртуальные устройства) [6]. При наличии профиля (набора характеристик) видеокарты позволяет отладку под нее. Позволяет автоматически находить ошибки обращений к памяти, гонки потоков.

OpenCL API Debugger – отладчик для CPU и GPU производства Intel. Входит в пакет Intel SDK For OpenCL [7]. Не работает с AMD-GPU.

NVIDIA Nsight – отладчик для GPU производства Nvidia. Не работает с AMD-GPU. Позволяет работать не только с OpenCL-программами, но и с программами на Cuda [8].

gDEBugger CL позволяет отлаживать OpenCL- и OpenGL-программы, профилировать, изменять ядра во время их выполнения, анализировать доступы к памяти [9].

В таблице 1.2 приведено сравнение существующих OpenCL-отладчиков по критериям, сформулированным выше.

Таблица 1.2 – Сравнение существующих отладчиков

Отладчик	CodeXL	Oclgrind	OpenCL API Debugger	NVIDIA Nsight	gDEBugger CL
Платформонезависимость	-	+	+	+	+
Устройствонезависимость	-	+	-	-	+
Интеграция с IDE	+	-	+	+	-

1.2.3. Выводы по итогам сравнения

Только Oclgrind и gDEBugger работают со всеми устройствами. Но Oclgrind использует виртуальные устройства, что не позволяет учитвать все особенности архитектуры. Кроме того, это делает профилирование невозможным. Ни один из устройствонезависимых отладчиков не имеет плагина для популярных сред разработки. Все отладчики, кроме CodeXL, работают как под ОС Windows, так и под Linux. Как инструмент разработки лучше всего подходит gDEBugger: он представляет все возможности для отладки и профилирования, работает со всеми устройствами и популярными ОС. Но он выполнен в виде отдельного приложения с gui, что делает невозможным создание адаптера для интегрирования его в IDE.

2. Формулировка требований к программе

2.1. Устройствонезависимость

Устройствонезависимость требуется, чтобы программа позволяла отлаживать OpenCL-ядра на любых устройствах (частности, интерес представляют новые устройства фирмы AMD). Кроме непосредственно возможности отладки, это делает ее удобнее: например, в системе с несколькими гетерогенными устройствами программист сможет использовать только один инструмент отладки.

2.2. Возможность интеграции

Возможность интеграции отладчика со средами разработки повышает удобство пользователя. Кроме того, для современных отладчиков это фактически стало стандартом.

2.3. Интерфейс командной строки

Интерфейс командной строки (Command line interface, CLI) упрощает интеграцию с другими программами (в том числе IDE), но не исключает возможности добавления в отладчик собственного графического интерфейса.

2.4. Высокоуровневость

OpenCL позволяет программисту абстрагироваться от сложных наборов инструкций конкретного устройства, которые, зачастую, в большой мере архитектурно-специфичны и насчитывают сотни инструкций. Поэтому, низкоуровневая отладочная информация не обязательна для разработки OpenCL-приложений. Низкоуровневый подход подразумевает работу с машинным кодом. OpenCL-программы могут запускаться на различных устройствах с разными архитектурами и наборами инструкций (например, 5 архитектур AMD GCN и соответствующих наборов инструкций), поэтому низкоуровневый подход исключает создание устройствонезависимого отладчика. Сим-

вольный подход позволяет достичь устройствонезависимости за счет работы с исходным кодом.

2.5. Платформонезависимость

OpenCL-приложения могут отлаживаться в системах под управлением различных операционных систем (например, семейств Linux и Windows). Поэтому, отладчик должен быть совместим с этими системами.

3. ОПИСАНИЕ РЕШЕНИЯ

Отладчик построен с использованием высокоуровневого подхода. Это значит, что он не работает с машинным кодом или кодом низкоуровневой виртуальной машины (LLVM), как это делают устройство-специфичные отладчики (например, CodeXL). Вместо модификации низкоуровневого кода целевого приложения (OpenCL-ядра), отладчик модифицирует исходный код, написанный на языке OpenCL.

Программа написана на языке программирования Python3, так как этот язык позволяет достичь высокой скорости и качества разработки.

3.1. Структура программной реализации

3.1.1. Структуры данных

Программная реализация структур данных, описанных в подразделе 3.1.1., представлена в приложении А.

3.1.1.1. ClTypes

Класс `ClTypes` хранит в статических полях информацию о системе типов OpenCL:

- названия скалярных типов данных;
- названия векторных типов данных;
- названия целочисленных типов данных;
- названия знаковых целочисленных типов данных;
- названия беззнаковых целочисленных типов данных;
- названия вещественных типов данных;
- синоним для типа указателя;
- названия скалярных типов, которым соответствуют векторные типы;
- возможные длины векторов (переменных векторного типа);
- словарь, определяющий парсер для каждого примитивного скалярного типа;
- словарь printf-флагов для примитивных скалярных типов;

- статический метод, реализующий генерацию printf-флагов для любых встроенных типов данных (включая векторные).

Парсинг скалярных величин реализован в библиотеке numpy.

3.1.1.2. Declaration

Класс `Declaration` – базовый класс объявления переменной и поля структуры. Имеет методы:

- `is_struct()` – возвращает `True` или `False` если объект является объявлением структуры или нет соответственно;
- `words_num()` – возвращает количество слов, разделенных пробелом, которые требуются для текстового представления значения объявленной переменной в формате, используемом отладчиком для получения значений переменных OpenCL-программы.

3.1.1.3. VarDeclaration

Класс `VarDeclaration` хранит информацию об объявлении переменной в следующих полях:

- `var_name` – имя переменной;
- `full_type` – полная информация о типе;
- `address_space` – модификатор адресного пространства;
- `is_struct` – булево значение, определяющее, имеет ли переменная составной тип (структура);
- `var_type` – название типа переменной;
- `pointer_rank` – порядок указателя (если переменная является указателем);
- `var_shape` – форма массива (если переменная является массивом).

Конструктор получает имя переменной и полный тип, по которому определяет значения полей.

3.1.1.4. FieldDeclaration

Класс `FieldDeclaration` хранит информацию об объявлении переменной в следующих полях:

- `var_name` – имя переменной;
- `full_type` – полная информация о типе;
- `is_struct` – булево значение, определяющее, имеет ли переменная составной тип (структура);
- `var_type` – название типа переменной;
- `pointer_rank` – порядок указателя (если переменная является указателем);
- `var_shape` – форма массива (если переменная является массивом).

Отличается от `VarDeclaration` тем, что не содержит информации об адресном пространстве, так как адресное пространство полей структуры совпадает с оным у самой структуры и не указывается явно.

3.1.1.5. StructDeclaration

Класс `StructDeclaration` хранит информацию об объявлении структуры:

- поле `fields` представляет из себя словарь и хранит объявления полей структуры;
- поле `name` содержит имя структуры.

Так же как `Declaration`, имеет метод `words_num()` – возвращает количество слов, разделенных пробелом, которые требуются для текстового

представления значения объявленной переменной в формате, используемом отладчиком для получения значений переменных OpenCL-программы.

3.1.1.6. Variable

Класс `Variable` хранит информацию об объявлении переменной в поле `decl` и ее значение в точке останова в поле value.

Конструктор получает объявление переменной и текстовое представление ее значения, полученное от целевого приложения, которому сопоставляется ее значение в виде, соответствующем типу. Для этого класс имеет следующие методы:

- `__parse_var` – возвращает значение переменной любого типа;
- `__parse_value` – возвращает значение одиночной переменной (не массива);
- `__parse_scalar_value` – возвращает значение переменной скалярного типа;
- `__parse_vector_value` – возвращает значение переменной векторного типа;
- `__parse_struct` – возвращает значение переменной пользовательского типа;
- `__parse_array` – возвращает значения элементов массива и указатель на его начало;
- `__parse_1d_array` – то же для одномерных массивов;
- `__parse_2d_array` – то же для двумерных массивов;
- `__parse_3d_array` – то же для трехмерных массивов;

Непосредственно в конструкторе вызывается только метод `__parse_var`, который, в свою очередь, вызывает остальные методы в зависимости от типа переменной. Иерархия вызовов методов представлена на рисунке 3.1

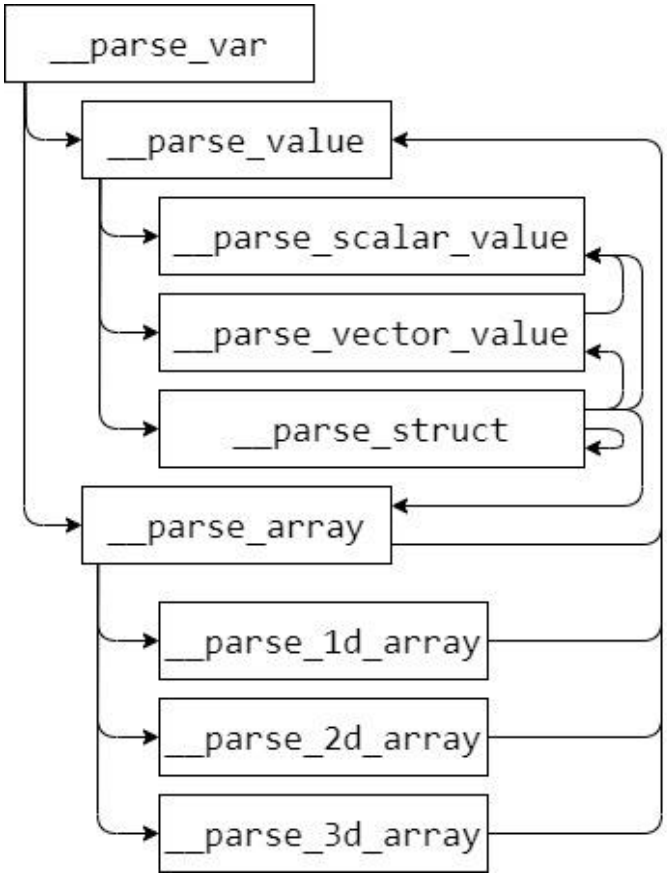

Рисунок 3.1 - Иерархия вызовов методов класса `Variable`

3.1.2. Передача состояния устройства

Для инспекции регистров и памяти OpenCL-устройства необходимо передать их состояния с устройства в систему, управляющую им. Сделать это можно тремя способами:

- Через пользовательские OpenCL-буферы – влечет возможность переполнения буферов, так как максимальный размер буфера, который может быть выделен в памяти устройства, определяется драйвером устройства;
- Через специальный printf-буфер (специальный буфер, определенный неявно, который служит для пользовательского отладочного вывода) – уменьшает пропускную способность канала между устройством и управляющей им системой из-за кодирования данных как текста;

- С использованием API драйвера устройства – исключает устройствонезависимость отладчика, так как драйвера специфичны к устройству (или его производителю).

Отладчик использует printf-буффер, так как им управляет драйвер устройства, что позволяет получить с устройства данные любого размера.

Для этого он модифицирует исходный код OpenCL-ядра таким образом, что ядро при выполнении выводит в него целевую информацию.

Чтобы модифицировать исходный код ядра, используется библиотека Clang (libclang). Она позволяет построить абстрактное синтаксическое дерево (AST) кода. AST содержит информацию о типах переменных программы и объявленных структурах. С помощью этой информации код меняется таким образом, что при выполнении выводит в printf-буффер состояния переменных и областей памяти.

3.1.2.1. Формат данных

При передаче данных от целевого приложения к отладчику, они представляются в текстовом виде.

Переменная любого типа (включая массивы, структуры, числа, векторы, указатели) представляется одной строкой, завершающейся символом перевода строки. Строка всегда начинается с имени переменной.

Для переменных скалярных типов непосредственно после имени переменной идет ее значение:

- для целочисленных типов – шестнадцатеричное представление. Например, переменная, объявленная как “int a = 16” представляется как “a 0x10”
- для вещественных типов – десятичное представление. Например, переменная, объявленная как “float b = 0.1000” представляется как “b 0.1”

Для переменных векторных типов после имени переменной идут значения элементов вектора, разделенные запятой. Правила представления элементов вектора те же, что для скалярных типов. Например, переменная, объ-

явленная как “int4 c = (int4)(1, 2, 3, 4)” представляется как “c 0x1,0x2,0x3,0x4”.

Для одномерных массивов после имени идет указатель на начало массива в шестнадцатеричной записи. В OpenCL указатели имеют длину 4 байта. После него через пробел идут значения элементов массива. Например, массив, объявленный как “char d[3] = {1, 2, 3}” может представляться как “d 0xffffffff 0x1 0x2 0x3”.

Двумерные массивы рассматриваются как массивы одномерных массивов, которые представляются по тем же правилам, что одиночные одномерные массивы. Правила представления трехмерных массивов аналогичны. . Например, массив, объявленный как “char e[3][1] = {1, 2, 3}” может представляться как “d 0x00000000 0x00000000 0x1 0x00000001 0x2 0x00000002 0x3”.

Массивов с размерностью большей, чем 3, в языке OpenCL нет.

Для структуры значение каждого поля представляется по правилам, указанным выше, последовательно через пробел. Если структура содержит другую структуру как поле, то правило представления структур принимает рекурсивный характер. Эта рекурсия не может быть бесконечной, так как число типов структур в программе конечно.

Например, если определено две структуры:

```
struct my_struct1 {
     int count = 1;
}
struct my_struct2 {
     my_struct1 f;
     int num = 4;
}
```

то переменная, объявленная как “`my_struct2 a;`” представляется как “a f count 1 num 4”.

3.1.2.2. Структура программной реализации

Построение AST кода, написанного на си-подобных языках, с помощью библиотеки Clang реализует класс `SourceProcessor.` Это абстрактный класс: он непосредственно реализует лишь методы построения AST и его текстового представления (которое требуется для отладки), оставляя реализацию методов изменения кода своим потомкам. Исходный код класса `SourceProcessor` представлен в приложении Б.

Построение AST OpenCL-кода имеет свою специфику. Класс `OclSourceProcessor` является наследником класса `SourceProcessor` и реализует его абстрактные методы `_prepare()` и `_defuse()`, которые подготавливают код к построению AST средствами Clang: определяют встроенные типы языка OpenCL. Это необходимо, потому что обертка Clang для языка Python не предоставляет полного доступа к функционалу библиотеки. Исходный код класса `OclSourceProcessor` представлен в приложении В.

Класс `LineInserter является наследником класса SourceProcessor` и реализует функционал добавления новых строк в исходный код. Он хранит номер строки, после которой вставляет новый код, в поле `_insert_line`, а код, подлежащий добавлению, в поле `_line_insertions` в виде массива строк. Метод `_apply_patches()` добавляет новые строки. Исходный код класса `LineInserter` представлен в приложении Г.

Класс `KernelProcessor` является наследником классов `OclSourceProcessor` и `LineInserter`. Иерархичная схема наследования этих классов представлена на рисунке 3.1. Такая схема называется ромбовидным наследованием. `KernelProcessor` реализует весь функционал, связанный с модификацией OpenCL-ядра. Сначала он обходит AST и сохраняет участки кода, содержащие точку останова (целевые), в иерархическом поряд-

ке в переменную `blocks`. Затем для каждого целевого участка кода он находит все переменные, объявленные в нем, и сохраняет данные об их объявлениях в список объектов типа VarDeclaration. Таким образом, формируется список всех переменных (включая массивы, структуры и указатели), которые существуют в программе в точке останова. После этого генерируется код, который будет вставлен в целевое OpenCL-ядро. Исходный код класса `LineInserter` представлен в приложении Д.

Так как OpenCL-программы не могут использовать сторонние библиотеки, они, зачастую, не подключают заголовочных файлов большого размера. Поэтому AST ядра имеет относительно маленькую высоту (по сравнению с деревьями C++- или C-кода). Из-за этого выбор алгоритма обхода абстрактного синтаксического дерева не очень значителен. Для обхода абстрактного синтаксического дерева используется DFS (поиск в глубину), так как он прост в реализации.

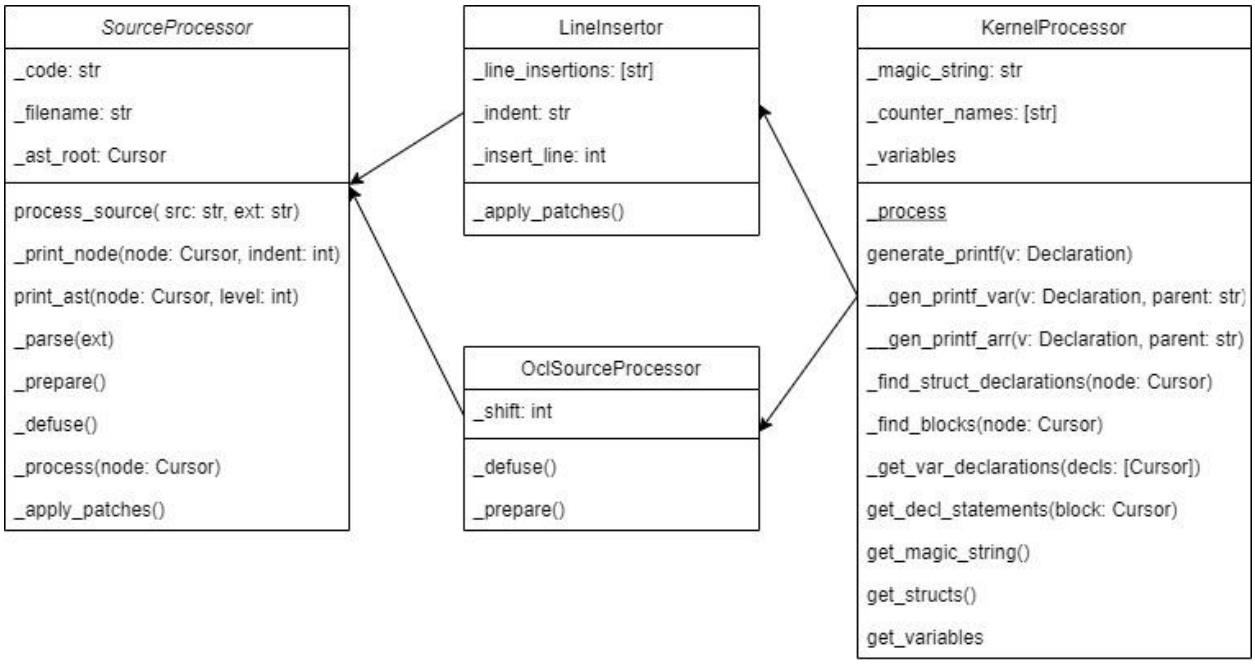

Рисунок 3.1 - Схема наследования классов SourceProcessor

3.1.3. Получение состояния устройства

OpenCL-приложения обычно компилируют ядро для целевого устройства средствами его драйвера из исходного кода. Код ядра может как читаться из отдельного файла во время выполнения OpenCL-приложения, так и храниться непосредственно в памяти приложения (иногда, в зашифрованном виде). Поэтому после модификации исходного кода OpenCL-ядра целевое приложение нужно заново скомпилировать и слинковать.

После этого целевое приложение запускается отладчиком как дочерний процесс. Отладчик ждет, когда получит из потока вывода целевого приложения магическую строку – она сигнализирует о начале отладочного вывода. В качестве магической строки подбирается такая последовательность символов, которую целевое приложение никогда не выведет без модификации (то есть вероятность этого события должна быть пренебрежимо мала). Затем отладчик считывает данные и предоставляет пользователю.

Этот функционал реализует класс `OclDebugger`. Он имеет следующие методы:

- `__init__` - конструктор класса. Получает путь к файлу исходного кода ядра, путь к целевому приложению, команду для сборки целевого приложения.
- `_debug` – служебный метод. Модифицирует файл ядра с помощью объекта класса `KernelProcessor`, собирает целевое приложение, запускает его как свой подпроцесс, получает из его потоков вывода информацию о значениях переменных в точке останова. Запуск целевого приложения реализован асинхронно с помощью механизма async/await языка python. Это позволяет создать объект-генератор, выдающий строки из потоков вывода целевого приложения по мере их вывода. Генератор избавляет от необходимости получать весь вывод целевой программы целиком, разбивать его на строки и хранить в памяти отладчика. Это может снизить время простоя в ожидании данных от устройства, если инспектируется большое количество потоков.

- `_build_env` – служебный метод. Создает среду (словарь переменных окружения и их значений) для запуска целевого приложения.
- `_build` – реализует сборку целевого приложения с использованием механизма async/await языка python.
- `process_values` – создает объекты типа `Variable`, содержащие информацию о переменных ядра и их значениях в точке останова, полученную от модифицированного целевого приложения. Получает данные из генератора `value_generator`.
- `value_generator` – метод-генератор, выдающий последовательно значения переменных, полученных от модифицированного целевого приложения.
- `safe_debug` – получает как параметр точку останова, возвращает список объектов типа `Variable`, содержащих значения переменных в точке останова. По сути является оберткой над методом `_debug`. Перед вызовом последнего сохраняет копию файла OpenCL-ядра целевого приложения, а после отменяет изменения, сделанные в этом файле. Благодаря механизму try/except/finally языка Python, модификации ядра будут отменены в любом случае: даже если метод `_debug` спровоцировал ошибку. Таким образом обеспечивается безопасность отладчика для исходного кода целевой программы.

Исходный код класса `OclDebugger` представлен в приложении Д.

3.2. Интерфейс пользователя

Отладчик использует интерфейс командной строки (CLI-interface). Это дает возможность в будущем создать специальный адаптер (Adapter), через который отладчики взаимодействуют с IDE, что обеспечивает интеграцию в них.

3.2.1. Формат входных данных приложения

При запуске он получает аргументы из командной строки в следующем формате:

```
[--<аргумент> | -<сокращение> <значение аргумента>]
```

Все аргументы кроме build являются обязательными, так как задают необходимые для работы отладчика данные (входные).

В таблице 3.1 приведены пояснения назначения аргументов.

Таблица 3.1. Аргументы командной строки.

Аргумент	Сокращение	Пояснение
kernel	k	Путь к файлу целевого OpenCL-ядра
application	a	Путь к исполняемому файлу целевого приложения
build		Команда для сборки целевого приложения
breakpoint	b	Номер строки для точки останова
threads	t	Глобальные индексы целевых потоков

3.2.2. Формат выходных данных приложения

Приложение выводит JSON-представление данных об инспектируемых переменных и их значениях в точке останова.

Ниже приведены примеры выходных данных для массивов: они включают в себя указатель на начало массива. Для остальных типов значения записываются по правилам нотации JSON.

```
{"decl": {"var_type": "double", "var_shape": [2, 2], "is_array": true, "var_name": "a", "full_type": "__private double [2][2]", "address_space": "__private", "pointer_rank": 0}, "gid": 0, "value": [67108096, [[67108096, [0.1, 0.2]], [67108112, [0.1, 0.2]]]]}
```

```
{"decl": {"var_type": "double", "var_shape": [2], "is_array": true, "var_name": "c", "full_type": "__private
```

double [2]", "address_space": "__private", "pointer_rank": 0}, "gid": 0, "value": [67108128, [0.4, 0.5]]}

{"decl": {"var_type": "double", "var_shape": [2, 2], "is_array": true, "var_name": "a", "full_type": "__private double [2][2]", "address_space": "__private", "pointer_rank": 0}, "gid": 3, "value": [67108096, [[67108096, [0.1, 0.2]], [67108112, [0.1, 0.2]]]]}

{"decl": {"var_type": "double", "var_shape": [2], "is_array": true, "var_name": "c", "full_type": "__private double [2]", "address_space": "__private", "pointer_rank": 0}, "gid": 3, "value": [67108128, [0.4, 0.5]]}

4. ОБЕСПЕЧЕНИЕ КАЧЕСТВА РАЗРАБОТКИ, ПРОДУКЦИИ, ПРОГРАММНОГО ПРОДУКТА (ДОП. РАЗДЕЛ)

Сначала определяются четкие требования к формату входных и выходных данных элемента (алгоритма, функции, метода). После этого создается набор тестов, проверяющих соответствие результата работы тестируемого элемента или всей программы сформулированным требованиям. Такой подход к разработке называется TDD (test driven development). Он был применен при создании программы, описанной в этой работе.

4.1. Юнит-тесты

Качество разработки отладчика (программного продукта) достигается путем тестирования нового функционала по мере его реализации. Тестирование реализовано в виде юнит-тестов – тестов, проверяющих корректность работы единицы функционала программы. Такой подход называется модульным тестированием. Он упрощает отладку за счет модульной структуры: модули отлаживаются независимо друг от друга. Юнит-тесты были сделаны с использованием библиотеки unittest.

4.1.1. Тестирование парсинга данных

Для тестирования парсинга данных, получаемых отладчиком от целевого приложения, создан набор юнит-тестов (который называется тест-кейсом). Каждый тест проверяет корректность определенного сценария.

В соответствии с форматом данных, определенным ранее, значение переменной целочисленного типа передается в виде строки, содержащей название переменной и ее значение в шестнадцатиричном представлении, разделенные пробелом. Например, беззнаковая однобайтная переменная, объявленная, как “uchar c = 255” представляется как строка “c 0xff”, а переменная, объявленная как “char c = -1” представляется как строка “c 0xff”. То есть, знаковые и беззнаковые переменные могут представляться одинаково. По-

этому, для целочисленных типов данных тесты с отрицательными и положительными значениями проводятся отдельно.

Тесты парсинга данных реализованы как методы класса `VariableTest`. Исходный код класса представлен в приложении З.

Тест `test_char_positive` проверяет корректность парсинга значения переменной однобайтного целочисленного знакового типа: строка “a 12” записывает значение 0x12=18d переменной a.

- Входные данные: строка “a 12”
- Выходные данные: целое число 0x12

Тест `test_char_negative` проверяет корректность парсинга значения переменной однобайтного целочисленного знакового типа: строка “a ff” записывает значение 0xff=-1 переменной a, так как тип однобайтный, а значения представлены в регистрах (или памяти) в дополнительном коде.

- Входные данные: “a ff”
- Выходные данные: -1

Тест `test_char_prefix` проверяет корректность парсинга значения переменной однобайтного целочисленного знакового типа: строка “a 0x70” записывает значение 0x70=112 переменной a. Он показывает, что парсинг шестнадцатиричного числа может выполняться не только без префикса 0x, но и с ним.

- Входные данные: “a 0x70”
- Выходные данные: 0x70

Тест `test_char_too_much_digits` проверяет корректность парсинга значения переменной однобайтного целочисленного знакового типа в случае, если запись числа насчитывает слишком много цифр. В таком случае, знача-

щими считаются младшие цифры (две шестнадцатиричные цифры в случае однобайтного целочисленного типа).

- Входные данные: “a 0x1f100”
- Выходные данные: 0x00

Тест `test_uchar_positive` проверяет корректность парсинга значения переменной однобайтного целочисленного беззнакового типа: строка “a 12” записывает значение 0x12=18d переменной a.

- Входные данные: “a 0x12”
- Выходные данные: 0x12

Тест `test_uchar_negative` проверяет корректность парсинга значения переменной однобайтного целочисленного беззнакового типа: строка “a ff” записывает значение 0xff=255 переменной a.

- Входные данные: “a 0xff”
- Выходные данные: 255

Тест `test_uchar_prefix` проверяет корректность парсинга значения переменной однобайтного целочисленного безнакового типа: строка “a 0x70” записывает значение 0x70=112 переменной a. Он показывает, что парсинг шестнадцатиричного числа может выполняться не только без префикса 0x, но и с ним.

- Входные данные: “a 0x70”
- Выходные данные: 0x70

Тест `test_uchar_too_much_digits` проверяет корректность парсинга значения переменной однобайтного целочисленного знакового типа в случае, если запись числа насчитывает слишком много цифр. В таком случае,

значащими считаются младшие цифры (две шестнадцатиричные цифры в случае однобайтного целочисленного типа).

- Входные данные: “a 0x77700”
- Выходные данные: 0x00

Тест `test_int` проверяет корректность парсинга значения переменной четырехбайтного целочисленного знакового типа: строка “a 141” записывает значение 0x141 переменной a.

- Входные данные: “a 0x141”
- Выходные данные: 0x141

Тест `test_int_too_much_digits` проверяет корректность парсинга значения переменной однобайтного целочисленного знакового типа в случае, если запись числа насчитывает слишком много цифр. В таком случае, значащими считаются младшие цифры (восемь шестнадцатиричных цифр в случае четырехбайтного целочисленного типа).

- Входные данные: “a 0x777111111ff”
- Выходные данные: 0x111111ff

Тест `test_1d_arr` проверяет корректность парсинга значения одномерного массива целых четырехбайтных знаковых чисел длины 3. Такой массив представляется строкой, содержащей последовательно через пробел имя массива, указатель на его начало и три значения элементов массива. Например, строка “a 0xffffffff 1 2 3” записывает массив a = [1, 2, 3], начинающегося по смещению 0xffffffff – эти данные используются в тесте.

- Входные данные: “a 0xffffffff 1 2 3”
- Выходные данные: (0xffffffff, [1,2,3])

Тест `test_2d_arr` проверяет корректность парсинга значения двумерного массива (матрицы) целых четырехбайтных знаковых чисел формы (3,2). Такой массив представляется строкой, содержащей последовательно через пробел имя массива, указатель на его начало и три значения строк матрицы. Например, строка "`a 0x00000000 0x00000000 1 2 0x00000008 3 4 0x0000000C 5 6`" записывает массив a = [[1, 2], [3, 4], [5, 6]], начинающегося по смещению `0xffffffff` – эти данные используются в тесте.

- Входные данные: "a 0x00000000 0x00000000 1 2 0x00000008 3 4 0x0000000C 5 6"
- Выходные данные: (0x00000000, [(0x00000000, [1, 2]), (0x00000008, [3, 4]), (0x0000000C, [5, 6])])

Так как для чисел вещественного типа представление содержит знак, тесты для этих типов проводятся на положительных и отрицательных значениях.

Тест `test_float_positive` проверяет корректность парсинга положительного значения переменной четырехбайтного вещественного типа: строка " a 0.1" записывает значение 0.1 переменной a.

- Входные данные: "a 0.1"
- Выходные данные: 0.1

Тест `test_float_negative` проверяет корректность парсинга положительного значения переменной четырехбайтного вещественного типа: строка " a -0.133" записывает значение -0.13 переменной a.

- Входные данные: "a -0.133"
- Выходные данные: -0.133

Тест `test_double_positive` проверяет корректность парсинга отрицательного значения переменной четырехбайтного вещественного типа: строка “ a 0.1” записывает значение 0.1 переменной a.

- Входные данные: “a 0.1”
- Выходные данные: 0.1

Тест `test_double_negative` проверяет корректность парсинга отрицательного значения переменной четырехбайтного вещественного типа: строка “ a -0.133” записывает значение -0.13 переменной a.

- Входные данные: “a -0.133”
- Выходные данные: -0.133

Для тестов с векторными типами данных выбраны типы double2, double4, double8, double16.

Тест `test_double2` проверяет корректность парсинга положительного значения переменной четырехбайтного вещественного типа: строка “ a -0.133” записывает значение -0.13 переменной a.

- Входные данные: “ a -0.133,0.1”
- Выходные данные: [0.133, 0.1]

Тест `test_double4` проверяет корректность парсинга положительного значения переменной четырехбайтного вещественного типа: строка “ a -0.133” записывает значение -0.13 переменной a.

- Входные данные: “ a -0.133,0.1,0.2,0.3”
- Выходные данные: [0.133, 0.1, 0.2, 0.3]

Тест `test_double8` проверяет корректность парсинга положительного значения переменной четырехбайтного вещественного типа: строка “ a -0.133” записывает значение -0.13 переменной a.

- Входные данные: “ a -0.133,0.1,0.2,0.3,-0.133,0.1,0.2,0.7”
- Выходные данные: [-0.133, 0.1, 0.2, 0.3, -0.133, 0.1, 0.2, 0.7]

Тест `test_double16` проверяет корректность парсинга положительного значения переменной четырехбайтного вещественного типа: строка “ a -0.133” записывает значение -0.13 переменной a.

- Входные данные: “a -0.133,0.1,0.2,0.3,-0.133,0.1,0.2,0.7,-0.133,0.1,0.2,0.3,-0.133,0.1,0.2,0.7”
- Выходные данные: [-0.133, 0.1, 0.2, 0.3, -0.133, 0.1, 0.2, 0.7, -0.133, 0.1, 0.2, 0.3, -0.133, 0.1, 0.2, 0.7]

Для тестов, работающих со структурами, было определено пять структур:

```
struct my_struct_1
{
    int count;
    double2 v;
};

struct my_struct_2
{
    int count;
    my_struct_1 v;
};

struct my_struct_3
{
    int count;
    int a[3];
```

```
};
struct my_struct_4
{
    int count;
    int a[3][2];
};
struct my_struct_5
{
    int count;
    int a[3][2][1];
};
```

Это делается в методе `setUp()` класса, реализующего тест-кейс.

Тест `test_simple_struct1` проверяет корректность парсинга значения переменной составного типа (структуры) с именем s, содержащей только поля встроенных типов:

- Входные данные: “s count 0x1488 v 0.1,0.2”
- Выходные данные: {'count': 0x1488, 'v': [0.1, 0.2]}

Тест `test_simple_struct2` проверяет корректность парсинга значения переменной составного типа (структуры) с именем s, содержащей только поля встроенных типов:

- Входные данные: “s count 0x145 v 0.1,0.3”
- Выходные данные: {'count': 0x145, 'v': [0.1, 0.3]}

Тест `test_struct_with_struct_as_a_field` проверяет корректность парсинга значения переменной составного типа (структуры) с именем s, содержащей другую структуру как поле:

- Входные данные: “s count 0x145 v count 0x231 v 0.1,0.3”
- Выходные данные: {'count': 0x145, 'v': {'count': 0x231, 'v': [0.1, 0.3]}}

Тест `test_array1d_struct` проверяет корректность парсинга значения переменной составного типа (структуры) с именем s, содержащей одномерный массив как поле:

- Входные данные: “ `s count 0x1488 a 0x00000000 0x71 0x198 0x43`”
- Выходные данные: `{'count': 0x1488, 'a': (0, [0x71, 0x198, 0x43])}`

Тест `test_array2d_struct` проверяет корректность парсинга значения переменной составного типа (структуры) с именем s, содержащей двумерный массив как поле:

- Входные данные: “ `s count 0x1488 a 0x00000000 0x00000000 0x00000000 1 0x00000004 2 0x00000008 0x00000008 3 0x0000000C 4 0x00000010 0x00000010 5 0x00000014 6`”
- Выходные данные: {'count': 0x1488, 'a': (0x00000000, [(0x00000000, [(0x00000000, [1]), (0x00000004, [2])]), (0x00000008, [(0x00000008, [3]), (0x0000000C, [4])]), (0x00000010, [(0x00000010, [5]), (0x00000014, [6])])])})

4.1.2. Тестирование парсинга типов

Библиотека Clang представляет тип переменной как строку. Эта строка может иметь сложную структуру: она содержит информацию о типе, размерах (для массивов), является ли переменная указателем, а в OpenCL еще и модификатор адресного пространства, указывающий физическую область памяти переменной (регистры, кэш, оперативная память). Поэтому требуется

получать эту информацию из строки, характеризующей тип переменной, которую предоставляет Clang в одном из полей узла AST, описывающего объявление переменной. Этот функционал реализован в классах `VarDeclaration` и `FieldDeclaration`.

Тесты для парсинга типов, реализованного в `VarDeclaration`, представлены классом `VariableDeclarationTest.` Исходный код класса представлен в приложении И.

Тест `test_private_int1` проверяет корректность распознавания целочисленной четырехбайтной переменной, содержащейся в приватной памяти рабочего элемента.

- Входные данные: “__private int”
- Выходные данные: {var_type: 'int', address_space: '__private', is_array: False, var_shape: None, pointer_rank: 0, words_num(): 2, is_struct(): False}

Тест `test_private_int2` проверяет корректность распознавания целочисленной четырехбайтной переменной, содержащейся в приватной памяти рабочего элемента. Отличается от предыдущего порядком слов: в OpenCL оба варианта допустимы.

- Входные данные: “__private int”
- Выходные данные: {var_type: 'int', address_space: '__private', is_array: False, var_shape: None, pointer_rank: 0, words_num(): 2, is_struct(): False}

Тест `test_private_1d_array` проверяет корректность распознавания массива целочисленных четырехбайтных величин, содержащегося в приватной памяти рабочего элемента.

- Входные данные: “__private int [41]”
- Выходные данные: {var_type: 'int', address_space: '__private', is_array: True, var_shape: [41], pointer_rank: 0, words_num(): 43, is_struct(): False}

Тест `test_private_2d_array` проверяет корректность распознавания двумерного массива целочисленных четырехбайтных величин, содержащегося в приватной памяти рабочего элемента.

- Входные данные: “__private int [41][5]”
- Выходные данные: {var_type: 'int', address_space: '__private', is_array: True, var_shape: [41, 5], pointer_rank: 0, words_num(): 248, is_struct(): False}

Тест `test_private_3d_array` проверяет корректность распознавания трехмерного массива целочисленных четырехбайтных величин, содержащегося в приватной памяти рабочего элемента.

- Входные данные: “__private int [2][3][1]”
- Выходные данные: {var_type: 'int', address_space: '__private', is_array: True, var_shape: [2, 3, 1], pointer_rank: 0, words_num(): 16, is_struct(): False}

Тест `test_local_1d_array` проверяет корректность распознавания массива целочисленных четырехбайтных величин, содержащегося в локальной памяти рабочего элемента.

- Входные данные: “__local int [41]”
- Выходные данные: {var_type: 'int', address_space: '__ local, is_array: True, var_shape: [41], pointer_rank: 0, words_num(): 43, is_struct(): False}

Тест `test_local_2d_array` проверяет корректность распознавания двумерного массива целочисленных четырехбайтных величин, содержащегося в локальной памяти рабочего элемента.

- Входные данные: “__ local int [41][5]”

- Выходные данные: {var_type: 'int', address_space: '__ local, is_array: True, var_shape: [41, 5], pointer_rank: 0, words_num(): 248, is_struct(): False}

Тест `test_local_3d_array` проверяет корректность распознавания трехмерного массива целочисленных четырехбайтных величин, содержащегося в локальной памяти рабочего элемента.

- Входные данные: “__ local int [2][3][1]”
- Выходные данные: {var_type: 'int', address_space: '__ local, is_array: True, var_shape: [2, 3, 1], pointer_rank: 0, words_num(): 16, is_struct(): False}

Тест `test_struct` проверяет корректность распознавания переменной типа my_struct_1, который является структурой, содержащегося в локальной памяти рабочего элемента.

- Входные данные: `'__private struct my_struct_1`”
- Выходные данные: {var_type: 'my_struct_1', address_space: '__private, is_array: False, var_shape: None, pointer_rank: 0, words_num(): 5, is_struct(): True}

  Для этого теста специально была определена структура:

```
struct my_struct_1
{
    int count;
    double2 v;
};
```

Тест `test_undefined_struct` проверяет корректность распознавания переменной типа my_struct1, который не был определен. Это должно повлечь исключение, что и проверяется тестом.

- Входные данные: `'__private struct my_struct1`”

4.2. Интеграционные тесты

Интеграционный тест проверяет корректность работы класса `OclDebugger`, который реализует весь функционал отладчика. Тест представлен в классе `OclDebuggerTest`. Исходный код класса представлен в приложении К Он получает состояния переменных специально написанного OpenCL-ядра, код которого представлен в приложении Л. Для однозначности переменные проинициализированы явным образом при объявлении. В коде объявлено 15 переменных различных типов (включая скалярные, векторные, массивы, структуры):

1. Переменная объявлена как “`char c = 1;`”. Ее ожидаемое значение – 1.
2. Переменная объявлена как “`uchar uc = 2;`”. Ее ожидаемое значение – 2.
3. Переменная объявлена как “short `s = 3;`”. Ее ожидаемое значение – 3.
4. Переменная объявлена как “ushort `us = 4;`”. Ее ожидаемое значение – 4.
5. Переменная объявлена как “int `i = 5;`”. Ее ожидаемое значение – 5.
6. Переменная объявлена как “uint `ui = 6;`”. Ее ожидаемое значение – 6.
7. Переменная объявлена как “long `l = 7;`”. Ее ожидаемое значение – 7.
8. Переменная объявлена как “ulong `ul = 8;`”. Ее ожидаемое значение – 8.
9. Переменная объявлена как “float `f = 14.41;`”. Ее ожидаемое значение – 14.31.
10. Переменная объявлена как “double `d = -147.1;`”. Ее ожидаемое значение – `-147.1`.

11. Переменная объявлена как “`arr_1d[2] = {14, 17}`”. Ее ожидаемое значение – (<любое число>, [14, 17]).

12. Переменная объявлена как “`arr_2d[2][2] = {{14, 17}, {15, 31}}`”. Ее ожидаемое значение – (<любое число>, [(<любое число> [14, 17]), (<любое число>, [15, 31])]).

13. Переменная объявлена как “`arr_3d[2][2][2] = {{{14, 17}, {15, 31}}, {{1, 2}, {3, 4}}}`”. Ее ожидаемое значение – (<любое число>, [(<любое число>, [(<любое число> [14, 17]), (<любое число>, [15, 31])]), (<любое число>, [(<любое число> [1, 2]), (<любое число>, [3, 4])])]).

14. Переменная объявлена как “`struct my_struct s1 = {{{14, 17}, {15, 31}}, (double2)(0.1, 0.2)};`”. Ее ожидаемое значение – {count: `{{14, 17}, {15, 31}}, w1: [0.1, 0.2]`}.

15. Переменная объявлена как “ `struct my_struct2 s2 = {{{{14, 17}, {15, 31}}, (double2)(0.1, 0.2)}, 0.12}`”. Ее ожидаемое значение – {a: {count: `{{14, 17}, {15, 31}}, w1: [0.1, 0.2]`}, w2: 0.12}.

Для массивов не проверяется значение указателя на начало массива, так как оно может быть различным в зависимости от устройства и его состояния.

ЗАКЛЮЧЕНИЕ

В работе были рассмотрены существующие отладчики для OpenCL. Среди них есть лишь один отладчик, который работает со всеми устройствами, поддерживающими технологию OpenCL. Он обладает собственным графическим интерфейсом пользователя, что делает невозможным создание адаптера для интеграции отладчика в IDE.

Были рассмотрены подходы к созданию отладчиков. Только символьный подход обеспечивает устройствонезависимость.

В рамках этой работы был разработан отладчик для OpenCL, позволяющий пользователю инспектировать значения переменных OpenCL-программы в точке останова. Разработанная программа может запускаться как под управлением ОС семейства Windows, так и семейства Linux. Она обладает CLI-интерфейсом, что позволяет в будущем интегрировать отладчик в IDE посредством создания адаптера – механизма конвертации данных в формат IDE и передачи их через API IDE. Созданный отладчик достигает устройствонезависимости (то есть он может работать с любыми OpenCL-устройствами) за счет реализации высокоуровневого подхода.

Разработка осуществлялась по принципу TDE, что позволило достичь корректности работы всех модулей. Для этого были написаны юнит-тесты.

Разработанный отладчик был протестирован на специально написанной OpenCL-программе, что показало корректность его работы (инспекции состояния пременных).

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Aaftab Munshi. The OpenCL Specification – 2012 – c.22-29, 195-206 URL: https://developer.amd.com/wordpress/media/2012/10/opencl-1.2.pdf
2. Aaftab Munshi, Benedict R. Gaster,Timothy G. Mattson, James Fung, Dan Ginsburg. OpenCL Programming Guide: Addison-Wesley – 2011 – c. 30-33 – URL: http://asu-cs.donntu.org/sites/default/files/images/doc/opencl.programming.guide.pdf
3. Raymond Tay. OpenCL Parallel Programming Development Cookbook: Packt Publishing – 2013 – c.10
4. Jonathan B. Rosenberg. How Debuggers Work: Algorithms, Data Structures, and Architecture: Wiley Computer Publishing – 1996 – c.1-19.
5. GpuOpen [Электронный ресурс] – URL: https://web.archive.org/web/20180627034628/https://gpuopen.com/codexl-2-0-is-here-and-open-source/
6. Oclgrind readme GpuOpen [Электронный ресурс] – URL: https://github.com/jrprice/Oclgrind

   Отладчики:
7. Intel Software [Электронный ресурс] – URL: https://software.intel.com/content/www/us/en/develop/articles/api-debugger-tutorial.html
8. Developer Nvidia [Электронный ресурс] – URL: https://developer.nvidia.com/nsight-visual-studio-edition
9. gDEBugger Tutorial [Электронный ресурс] – URL: http://www.gremedy.com/tutorial/

ПРИЛОЖЕНИЕ А

КОД МОДУЛЯ PRIMITIVES

```
import abc
import json
import re
from abc import ABC
from typing import List, Tuple

import numpy as np
from clang.cindex import Cursor, CursorKind

from filters import filter_node_list_by_node_kind


class ClTypes:
    # TODO: add half types
    pointer_type = 'uint'
    signed_integer_types = ['char', 'short', 'int', 'long']
    unsigned_integer_types = ['uchar', 'ushort', 'uint', 'ulong']
    integer_types = signed_integer_types + unsigned_integer_types
    float_types = ['float', 'double']
    vector_len = [2, 4, 8, 16]
    vector_base = float_types + integer_types
    vector_types = [f'{t}{n}' for t in vector_base for n in [2, 4, 8,
16]]
    scalar_types = ['char', 'uchar', 'short', 'ushort', 'int', 'uint',
'long', 'ulong', 'float', 'double']

    struct_declarations: [] = []

    parser = {
        'char': np.int8,
        'uchar': np.uint8,
        'short': np.int16,
        'ushort': np.uint16,
        'int': np.int32,
        'uint': np.uint32,
        'long': np.int64,
        'ulong': np.uint64,
```

```
        'float': np.float32,
        'double': np.float64
    }
    flags = {
        'char': 'x',
        'uchar': 'x',
        'short': 'hx',
        'ushort': 'hx',
        'int': 'x',
        'uint': 'x',
        'long': 'lx',
        'ulong': 'lx',
        'float': 'f',
        'double': 'lf'
    }

    @staticmethod
    def get_printf_flag(var_type: str):
        if var_type in ClTypes.scalar_types:
            return f'%{ClTypes.flags[var_type]}'
        if var_type in ClTypes.vector_types:
            n = re.search('[0-9]+', var_type).group(0)
            base = re.search('[a-z]+', var_type).group(0)
            return f'%v{n}{ClTypes.flags[base]}'
        assert False  # Fail if it's not a primitive type

    @staticmethod
    def get_struct_decl(struct_name: str):
        struct_names = [s.name for s in ClTypes.struct_declarations]
        if struct_name not in struct_names:
            raise Exception("Undefined struct name")
        struct = [s for s in ClTypes.struct_declarations if s.name ==
struct_name]
        assert len(struct) == 1
        struct = struct[0]
        return struct

    @staticmethod
    def get_struct_types():
        return [e.name for e in ClTypes.struct_declarations]
```

```
class Declaration(object):
    __metaclass__ = abc.ABCMeta

    def __init__(self):
        self.var_type = None
        self.var_shape = None
        self.is_array = None

    def is_struct(self):
        return self.var_type not in ClTypes.scalar_types and \
               self.var_type not in ClTypes.vector_types

    def words_num(self):
        val_words = 0
        if self.is_array:
            t = 1
            shape_rev = reversed(self.var_shape)
            for d in shape_rev:
                t *= d
                t += 1
            val_words = t
        elif self.is_struct():
            struct_decl = ClTypes.get_struct_decl(self.var_type)
            val_words = struct_decl.words_num()
        else:
            val_words = 1

        return 1 + val_words  # 1 for variable name signing

    def __str__(self):
        return json.dumps(self, default=lambda o: o.__dict__)

    def __repr__(self):
        return str(self)


class VarDeclaration(Declaration, ABC):
    _address_space_modifiers: [str] = ['__private', '__local',
'__global']

    def __init__(self, var_name: str, full_type: str):
        super().__init__()
```

```
        self.var_name = var_name
        # Make ASM come first
        words = full_type.split(' ')
        assert words is not None
        assert len(words) > 1
        if words[0] not in self._address_space_modifiers:
            words[0], words[1] = words[1], words[0]
        assert words[0] in self._address_space_modifiers
        self.full_type = ' '.join(words)
        self.address_space = words[0]
        is_struct = words[1] == 'struct'
        if is_struct:
            self.var_type = words[2]
        else:
            self.var_type = words[1]

        # Check if it's an array
        if len(words) > 2 + int(is_struct):
            match = re.findall('\[[0-9]+\]', self.full_type)
            self.is_array = bool(len(match))
            self.var_shape = [int(re.search('[0-9]+', m).group(0)) for
m in match]
        else:
            self.is_array = False

        # Check if it's a pointer
        match = re.findall('\*', self.full_type)
        self.pointer_rank = len(match)


class FieldDeclaration(Declaration, ABC):
    def __init__(self,  var_name: str, full_type: str):
        super().__init__()

        self.var_name = var_name

        # Make ASM come first
        words = full_type.split(' ')
        assert words is not None
        assert len(words) > 0
        self.full_type = ' '.join(words)
```

```
        is_struct = words[0] == 'struct'
        if is_struct:
            self.var_type = words[1]
        else:
            self.var_type = words[0]

        # Check if it's an array
        if len(words) > 1 + int(is_struct):
            match = re.findall('\[[0-9]+\]', self.full_type)
            self.is_array = bool(len(match))
            self.var_shape = [int(re.search('[0-9]+', m).group(0)) for
m in match]
        else:
            self.is_array = False

        # Check if it's a pointer
        match = re.findall('\*', self.full_type)
        self.pointer_rank = len(match)


class Variable(object):
    def __init__(self, decl: VarDeclaration, value: str, gid: int =
None):
        self.decl = decl
        self.gid = gid

        values = value.split(' ')
        assert values[0] == decl.var_name
        value = ' '.join(values[1:])
        self.value = self.__parse_var(value, decl)

    @staticmethod
    def __parse_var(value, decl):
        if decl.is_array:
            return Variable.__parse_array(value.split(' '),
decl.var_shape, decl.var_type)
        else:
            return Variable.__parse_value(value, decl.var_type)

    @staticmethod
    def __parse_scalar_value(value, var_type):
        if var_type not in ClTypes.float_types:
```

```
            value = int(value, base=16)
            return int(ClTypes.parser[var_type](value))
        return float(ClTypes.parser[var_type](value))

    @staticmethod
    def __parse_vector_value(value, var_type):
        elements = value.split(',')
        assert len(elements) in [2, 4, 8, 16]
        base = re.search('[a-z]+', var_type).group(0)
        return [Variable.__parse_scalar_value(value=e, var_type=base)
for e in elements]

    @staticmethod
    def __parse_value(value, var_type):
        if var_type in ClTypes.scalar_types:
            return Variable.__parse_scalar_value(value=value,
var_type=var_type)
        elif var_type in ClTypes.vector_types:
            return Variable.__parse_vector_value(value=value,
var_type=var_type)
        else:
            return Variable.__parse_struct(value=value,
var_type=var_type)

    @staticmethod
    def __parse_struct(value, var_type):
        struct_type = var_type
        struct_names = [s.name for s in ClTypes.struct_declarations]
        if struct_type not in struct_names:
            raise Exception("Undefined struct name")
        struct = [s for s in ClTypes.struct_declarations if s.name ==
struct_type]
        assert len(struct) == 1
        struct = struct[0]

        retval = {}
        elements = value.split(' ')
        i = 0
        for f in struct.fields.keys():
            field_decl = struct.fields[f]
            field_type = field_decl.var_type
            assert elements[i] == field_decl.var_name
```

```
            i += 1
            if field_decl.is_array:
                values = elements[i:i+field_decl.words_num() - 1]
                i += field_decl.words_num() - 1
                retval[f] = Variable.__parse_array(values,
field_decl.var_shape, field_decl.var_type)
            else:
                if field_type in ClTypes.scalar_types:
                    retval[f] = Varia-
ble.__parse_scalar_value(elements[i], field_type)
                    i += 1
                elif field_type in ClTypes.vector_types:
                    retval[f] = Varia-
ble.__parse_vector_value(elements[i], field_type)
                    i += 1
                elif field_type in ClTypes.get_struct_types():
                    l = field_decl.words_num()
                    retval[f] = Variable.__parse_struct('
'.join(elements[i: i + l - 1]), field_type)
                    i += l - 1

        return retval

    @staticmethod
    def __parse_array(arr: [str], var_shape: [int], var_type: str):
        n_dims = len(var_shape)
        if n_dims == 1:
            return Variable.__parse_1d_array(arr, var_shape, var_type)
        elif n_dims == 2:
            return Variable.__parse_2d_array(arr, var_shape, var_type)
        elif n_dims == 3:
            return Variable.__parse_3d_array(arr, var_shape, var_type)

    @staticmethod
    def __parse_1d_array(arr: [str], var_shape: [int], var_type: str)
-> Tuple:
        assert len(arr) == 1 + var_shape[0]
        return (Variable.__parse_value(arr[0], ClTypes.pointer_type),
                [Variable.__parse_value(v, var_type) for v in
arr[1:]])

    @staticmethod
```

```
    def __parse_2d_array(arr: [str], var_shape: [int], var_type: str)
-> Tuple:
        assert len(arr) == 1 + var_shape[0] * (1 + var_shape[1])
        values = []
        for i in range(var_shape[0]):
            array_1d = arr[1 + (i + 0) * (1 + var_shape[1]): 1 + (i +
1) * (1 + var_shape[1])]
            values.append(Variable.__parse_1d_array(array_1d,
(var_shape[1],), var_type))
        return Variable.__parse_value(arr[0], ClTypes.pointer_type),
values

    @staticmethod
    def __parse_3d_array(arr: [str], var_shape: [int], var_type: str)
-> Tuple:
        assert len(arr) == 1 + var_shape[0] * (1 + var_shape[1] * (1 +
var_shape[2]))
        retval = []
        for i in range(var_shape[0]):
            array_2d = arr[1 + (i + 0) * (1 + var_shape[1] * (1 +
var_shape[2])):
                           1 + (i + 1) * (1 + var_shape[1] * (1 +
var_shape[2]))]
            retval.append(Variable.__parse_2d_array(array_2d,
(var_shape[1], var_shape[2]), var_type))
        return Variable.__parse_value(arr[0], ClTypes.pointer_type),
retval

    def __str__(self):
        return json.dumps(self, default=lambda o: o.__dict__)

    def __repr__(self):
        return str(self)


class StructDeclaration(object):
    def __init__(self, node: Cursor):
        self.fields = {}
        self.name = None
        if node is None:
            return
        assert node.kind == CursorKind.STRUCT_DECL
```

```
        self.name = node.spelling
        fields = filter_node_list_by_node_kind(node.get_children(),
[CursorKind.FIELD_DECL])
        for f in fields:
            self.fields[f.spelling] = FieldDeclaration(f.spelling,
f.type.spelling)

    def words_num(self):
        retval = 0
        for f in self.fields.keys():
            field_decl = self.fields[f]
            retval += field_decl.words_num()
        return retval

    def __str__(self):
        return json.dumps(self, default=lambda o: o.__dict__)

    def __repr__(self):
        return str(self)
```

ПРИЛОЖЕНИЕ Б

КОД МОДУЛЯ SOURCEPROCESSOR

```
from clang.cindex import Cursor, Index


class SourceProcessor:
    def __init__(self):
        self._edit = []
        self._code = ""
        self._filename = None
        self._ast_root = None

    def process_source(self, src: str, ext='cl') -> str:
        self._code = src
        self._prepare()

        self._parse(ext)
        root = self._ast_root
        self._process(root)
        self._apply_patches()

        self._defuse()
        return self._code

    @staticmethod
    def _print_node(node: Cursor, indent: int):
        print('%s %-35s %-20s %-10s [%-6s:%s - %-6s:%s] %s %s ' % (' '
* indent,

node.kind, node.spelling, node.type.spelling,

node.extent.start.line, node.extent.start.column,

node.extent.end.line, node.extent.end.column,

node.location.file, node.mangled_name))

    def print_ast(self, node: Cursor, level: int = 0):
        self._print_node(node, level)
```

```
        for child in node.get_children():  # fil-
ter_node_list_by_file(node.get_children(), self._filename):
            self.print_ast(child, level + 1)

    def _parse(self, ext='cl'):
        index = Index.create()
        translation_unit = in-
dex.parse(unsaved_files=[(f'__file__.{ext}', self._code)],
                                        path=f'__file__.{ext}',
                                        args=['-cc1'])
        self._ast_root = translation_unit.cursor

    def _prepare(self):
        pass

    def _defuse(self):
        pass

    def _process(self, node: Cursor):
        pass

    def _apply_patches(self):
        pass
```

ПРИЛОЖЕНИЕ В

КОД МОДУЛЯ OCLSOURSEPROCESSOR

```
from SourceProcessor import SourceProcessor
from primitives import ClTypes


class OclSourceProcessor(SourceProcessor):
    _shift: int = 0
    _break_line: int = 0

    def __init__(self):
        SourceProcessor.__init__(self)

    # For some reason clang doesn't parse u* and vector types so it
needs a little help
    def _prepare(self):
        line_insertions = []
        # a) u* types
        line_insertions.extend([f'typedef unsigned {t} u{t};' for t in
ClTypes.signed_integer_types])
        # b) vector types.
        for t in ClTypes.vector_base:
            for n in ClTypes.vector_len:
                line_insertions.append(f'typedef {t} {t}{n}
__attribute__((ext_vector_type({n})));')
        # 1. Insert
        lines = self._code.split('\n')
        lines = line_insertions + lines[:]
        self._code = '\n'.join(lines)
        # 2. Shift
        self._shift = len(line_insertions)
        self._break_line += self._shift

    def _defuse(self):
        # Shift back
        lines = self._code.split('\n')
        self._code = '\n'.join(lines[self._shift:])
```

ПРИЛОЖЕНИЕ Г

КОД МОДУЛЯ LINEINSERTER

```
from SourceProcessor import SourceProcessor


class LineInserter(SourceProcessor):
    _line_insertions: [str] = []
    _indent: str = ''
    _insert_line: int = 0

    def __init__(self):
        SourceProcessor.__init__(self)

    def _apply_patches(self):
        line_insertions = []
        for line in self._line_insertions:
            line_insertions.extend([self._indent + e for e in
line.split('\n')])
        lines = self._code.split('\n')
        lines = lines[:self._insert_line] + line_insertions +
lines[self._insert_line:]
```

ПРИЛОЖЕНИЕ Д

КОД МОДУЛЯ KERNELMODIFIER

```
from   self._code = '\n'.join(lines[:])
```

ПРИЛОЖЕНИЕ Е

КОД МОДУЛЯ OCLDEBUGGER

```
import argparse
import asyncio
import os
import sys
from logging import fatal, warning
from shutil import copyfile  # TODO: find out whether it works with
Windows

from typing import List

from KernelProcessor import KernelProcessor
from primitives import VarDeclaration, Variable, ClTypes


class OclDebugger(object):
    _break_line: int = None
    _threads: [int] = None

    def __init__(self, kernel_file: str, binary: str, build_cmd: str):
        self._kernel_file = kernel_file
        self._binary = os.path.basename(binary)
        self._binary_dir = os.path.dirname(binary)
        # TODO: find out whether it works with Windows
        self._cmd = f'cd {self._binary_dir} && ./{self._binary}'
        self._build_cmd = build_cmd

    def safe_debug(self, break_line: int, threads: [int]):
        self._break_line = break_line
        self._threads = threads

        copyfile(self._kernel_file, 'kernel_backup')
        try:
            variables = self._debug()
        finally:
            copyfile('kernel_backup', self._kernel_file)
        return variables
```

```
    def _debug(self):
        kernel_processor = KernelProcessor(self._break_line,
self._threads)
        with open(self._kernel_file, 'r') as source_kernel_file:
            kernel = ker-
nel_processor.process_source(str(source_kernel_file.read()), 'cl')

        ClTypes.struct_declarations = kernel_processor.get_structs()

        with open(self._kernel_file, 'w') as kernel_file:
            kernel_file.write(kernel)
            kernel_file.close()

        with open('kernel.cl', 'w') as kernel_file:
            kernel_file.write(kernel)
            kernel_file.close()

        loop = asyncio.get_event_loop()
        try:
            loop.run_until_complete(self._build())
        finally:
            # loop.run_until_complete(loop.shutdown_asyncgens())
            pass

        try:
            variables =
loop.run_until_complete(self.process_values(kernel_processor.get_varia
bles(),

self.value_generator()))
            return variables
        finally:
            # see: https://docs.python.org/3/library/asyncio-
eventloop.html#asyncio.loop.shutdown_asyncgens
            loop.run_until_complete(loop.shutdown_asyncgens())
            loop.close()
            pass

    def _build_env(self) -> dict:
        env = os.environ.copy()

        dirs = env['PATH'].split(':')
```

```
        dirs[0] = self._binary_dir
        path = ':'.join(dirs)

        env['PATH'] = path
        env['PWD'] = self._binary_dir
        return env

    async def process_values(self, info: List[VarDeclaration], val-
ues):
        while True:
            try:
                if await values.__anext__() == KernelProces-
sor.get_magic_string():
                    break
            except StopAsyncIteration as e:
                fatal('No debugging data received')
                exit(-1)
        variables = []
        for t in self._threads:
            for i in info:
                v = await values.__anext__()
                variables.append(Variable(i, v, t))
        return variables

    async def _build(self):
        if self._build_cmd is not None:
            env = self._build_env()
            create = asyncio.create_subprocess_exec(self._build_cmd,
env=env,

stdin=asyncio.subprocess.PIPE,

stdout=asyncio.subprocess.PIPE,

stderr=asyncio.subprocess.STDOUT)
            try:
                proc = await create
            except Exception as e:
                fatal(f'Could not execute {self._binary}')

    async def value_generator(self):
        env = self._build_env()
```

```
        create = asyncio.create_subprocess_shell(self._cmd, env=env,

stdin=asyncio.subprocess.PIPE,

stdout=asyncio.subprocess.PIPE,

stderr=asyncio.subprocess.STDOUT)
        try:
            proc = await create
        except FileNotFoundError as e:
            warning(f'Could not find {self._binary};')
            exit(-1)
        except Exception as e:
            fatal(f'Could not execute {self._binary}')

        while True:
            line = await proc.stdout.readline()
            if not line:
                break
            line = line.decode('ascii').rstrip()
            yield line


def main():
    parser = argparse.ArgumentParser(prog='OclDebugger.py')
    parser.add_argument('-k', '--kernel', help='<Required> Kernel file
location', required=True)
    parser.add_argument('-a', '--application', help='<Required> Target
application location', required=True)
    parser.add_argument('--build', help='<Optional> build command if
needed', type=str, required=False)
    parser.add_argument('-b', '--breakpoint', help='<Required> Break-
point', type=int, required=True)
    parser.add_argument('-t', '--threads', nargs='+', help='<Required>
Target threads (global ids)', type=int, required=True)
    args = parser.parse_args()

    debugger = OclDebugger(
        kernel_file=args.kernel,
        binary=args.application,
        build_cmd=args.build
    )
```

```
    variables = debugger.safe_debug(break_line=args.breakpoint,
threads=args.threads)
    for v in variables:
        print(v)


if __name__ == '__main__':
    exit(main())
```

ПРИЛОЖЕНИЕ Ж

КОД МОДУЛЯ FILTERS

```
import typing
import clang


def filter_node_list_by_file(nodes: typ-
ing.Iterable[clang.cindex.Cursor], file_name: str) \
        -> typing.Iterable[clang.cindex.Cursor]:
    result = [n for n in nodes if n.location.file and
n.location.file.name == file_name]
    return result


def filter_node_list_by_node_kind(nodes: typ-
ing.Iterable[clang.cindex.Cursor], kinds: list) \
        -> typing.Iterable[clang.cindex.Cursor]:
    result = [n for n in nodes if n.kind in kinds]
    return result


def filter_node_list_by_start_line(nodes: typ-
ing.Iterable[clang.cindex.Cursor], by_line: int) \
        -> typing.Iterable[clang.cindex.Cursor]:
    result = [n for n in nodes if n.extent.start.line <= by_line]
    return result
```

ПРИЛОЖЕНИЕ 3

КОД МОДУЛЯ VARIABLETEST

```
import unittest

from primitives import Variable, VarDeclaration, StructDeclaration,
FieldDeclaration, ClTypes


class VariableTest(unittest.TestCase):
    def setUp(self) -> None:
        s1, s2 = StructDeclaration(None), StructDeclaration(None)
        arr1dstruct, arr2dstruct, arr3dstruct = StructDeclara-
tion(None), StructDeclaration(None), StructDeclaration(
            None)

        s1.name = 'my_struct_1'
        s1.fields = {'count': FieldDeclaration('count', 'int'),
                     'v': FieldDeclaration('v', 'double2')}

        s2.name = 'my_struct_2'
        s2.fields = {'count': FieldDeclaration('count', 'int'),
                     'v': FieldDeclaration('v', 'my_struct_1')}

        arr1dstruct.name = 'my_struct_3'
        arr1dstruct.fields = {'count': FieldDeclaration('count',
'int'),
                              'a': FieldDeclaration('a', 'int [3]')}

        arr2dstruct.name = 'my_struct_4'
        arr2dstruct.fields = {'count': FieldDeclaration('count',
'int'),
                              'a': FieldDeclaration('a', 'int [3]
[2]')}

        arr3dstruct.name = 'my_struct_5'
        arr3dstruct.fields = {'count': FieldDeclaration('count',
'int'),
                              'a': FieldDeclaration('a', 'int [3] [2]
[1]')}
```

```
        ClTypes.struct_declarations = [s1, s2, arr1dstruct,
arr2dstruct, arr3dstruct]

    def tearDown(self) -> None:
        pass

    def test_char_positive(self):
        var = Variable(decl=VarDeclaration(var_name='a',
full_type='__private char'),
                       value='a 12')
        self.assertEqual(var.value, 0x12)

    def test_char_negative(self):
        var = Variable(decl=VarDeclaration(var_name='a',
full_type='__private char'),
                       value='a ff')
        self.assertEqual(var.value, -1)

    def test_char_prefix(self):
        var = Variable(decl=VarDeclaration(var_name='a',
full_type='__private char'),
                       value='a 0x70')
        self.assertEqual(var.value, 0x70)

    def test_char_too_much_digits(self):
        var = Variable(decl=VarDeclaration(var_name='a',
full_type='__private char'),
                       value='a 0x1f100')
        self.assertEqual(var.value, 0x00)

    def test_uchar_positive(self):
        var = Variable(decl=VarDeclaration(var_name='a',
full_type='__private uchar'),
                       value='a 12')
        self.assertEqual(var.value, 0x12)

    def test_uchar_negative(self):
        var = Variable(decl=VarDeclaration(var_name='a',
full_type='__private uchar'),
                       value='a ff')
        self.assertEqual(var.value, 255)
```

```
    def test_uchar_prefix(self):
        var = Variable(decl=VarDeclaration(var_name='a',
full_type='__private uchar'),
                       value='a 0x70')
        self.assertEqual(var.value, 0x70)

    def test_uchar_too_much_digits(self):
        var = Variable(decl=VarDeclaration(var_name='a',
full_type='__private uchar'),
                       value='a 0x1f100')
        self.assertEqual(var.value, 0x00)

    def test_int(self):
        var = Variable(decl=VarDeclaration(var_name='a',
full_type='__private int'),
                       value='a 0x141')
        self.assertEqual(var.value, 0x141)

    def test_int_too_much_digits(self):
        var = Variable(decl=VarDeclaration(var_name='a',
full_type='__private int'),
                       value='a 0x777777111111ff')
        self.assertEqual(var.value, 0x111111ff)

    def test_1d_arr(self):
        var = Variable(decl=VarDeclaration(var_name='a',
full_type='__private int [3]'),
                       value='a 0xffffffff 1 2 3')
        self.assertEqual(var.value,
                         (0xffffffff, [1, 2, 3])
                         )

    def test_float_positive(self):
        var = Variable(decl=VarDeclaration(var_name='a',
full_type='__private float'),
                       value='a 0.1')
        self.assertAlmostEqual(var.value, 0.1)

    def test_float_negative(self):
        var = Variable(decl=VarDeclaration(var_name='a',
full_type='__private float'),
```

```
                        value='a -0.133')
        self.assertAlmostEqual(var.value, -0.133)

    def test_double_positive(self):
        var = Variable(decl=VarDeclaration(var_name='a',
full_type='__private double'),
                        value='a 0.1')
        self.assertAlmostEqual(var.value, 0.1)

    def test_double_negative(self):
        var = Variable(decl=VarDeclaration(var_name='a',
full_type='__private double'),
                        value='a -0.133')
        self.assertAlmostEqual(var.value, -0.133)

    def test_double2(self):
        var = Variable(decl=VarDeclaration(var_name='a',
full_type='__private double2'),
                        value='a -0.133,0.1')
        expected = [-0.133, 0.1]
        self.assertEqual(len(var.value), len(expected))
        for a, e in zip(var.value, expected):
            self.assertAlmostEqual(a, e)

    def test_double4(self):
        var = Variable(decl=VarDeclaration(var_name='a',
full_type='__private double4'),
                        value='a -0.133,0.1,0.2,0.3')
        expected = [-0.133, 0.1, 0.2, 0.3]
        self.assertEqual(len(var.value), len(expected))
        for a, e in zip(var.value, expected):
            self.assertAlmostEqual(a, e)

    def test_double8(self):
        var = Variable(decl=VarDeclaration(var_name='a',
full_type='__private double8'),
                        value='a -0.133,0.1,0.2,0.3,-
0.133,0.1,0.2,0.7')
        expected = [-0.133, 0.1, 0.2, 0.3, -0.133, 0.1, 0.2, 0.7]
        self.assertEqual(len(var.value), len(expected))
        for a, e in zip(var.value, expected):
            self.assertAlmostEqual(a, e)
```

```
    def test_double16(self):
        var = Variable(decl=VarDeclaration(var_name='a',
full_type='__private double16'),
                       value='a -0.133,0.1,0.2,0.3,-
0.133,0.1,0.2,0.7,-0.133,0.1,0.2,0.3,-0.133,0.1,0.2,0.7')
        expected = [-0.133, 0.1, 0.2, 0.3, -0.133, 0.1, 0.2, 0.7, -
0.133, 0.1, 0.2, 0.3, -0.133, 0.1, 0.2, 0.7]
        self.assertEqual(len(var.value), len(expected))
        for a, e in zip(var.value, expected):
            self.assertAlmostEqual(a, e)

    def test_2d_arr(self):
        var = Variable(decl=VarDeclaration(var_name='a',
full_type='__private int [3] [2]'),
                       value='a 0x00000000 0x00000000 1 2 0x00000008 3
4 0x0000000C 5 6')
        self.assertEqual(var.value, (0x00000000, [(0x00000000, [1,
2]), (0x00000008, [3, 4]), (0x0000000C, [5, 6])]))

    def test_3d_arr(self):
        var = Variable(decl=VarDeclaration(var_name='a',
full_type='__private int [3] [2] [1]'),
                       value='a 0x00000000 0x00000000 0x00000000 1
0x00000004 2 0x00000008 0x00000008 3 0x0000000C 4 0x00000010
0x00000010 5 0x00000014 6')
        self.assertEqual(var.value, (0x00000000, [(0x00000000,
[(0x00000000, [1]), (0x00000004, [2])]),
                                                  (0x00000008,
[(0x00000008, [3]), (0x0000000C, [4])]),
                                                  (0x00000010,
[(0x00000010, [5]), (0x00000014, [6])])]))

    def test_simple_struct1(self):
        var = Variable(decl=VarDeclaration(var_name='s',
full_type='__private my_struct_1'),
                       value='s count 0x1488 v 0.1,0.2')
        self.assertEqual(var.value, {'count': 0x1488, 'v': [0.1,
0.2]})

    def test_simple_struct2(self):
```

```
        var = Variable(decl=VarDeclaration(var_name='s',
full_type='__private my_struct_1'),
                       value='s count 0x145 v 0.1,0.3')
        self.assertEqual(var.value, {'count': 0x145, 'v': [0.1, 0.3]})

    def test_struct_with_struct_as_a_field(self):
        var = Variable(decl=VarDeclaration(var_name='s',
full_type='__private my_struct_2'),
                       value='s count 0x145 v count 0x231 v 0.1,0.3')
        self.assertEqual(var.value, {'count': 0x145, 'v': {'count':
0x231, 'v': [0.1, 0.3]}})

    def test_array1d_struct(self):
        var = Variable(decl=VarDeclaration(var_name='s',
full_type='__private my_struct_3'),
                       value='s count 0x1488 a 0x00000000 0x71 0x198
0x43')
        self.assertEqual(var.value, {'count': 0x1488, 'a': (0, [0x71,
0x198, 0x43])})

    def test_array2d_struct(self):
        var = Variable(decl=VarDeclaration(var_name='s',
full_type='__private my_struct_5'),
                       value='s count 0x1488 a 0x00000000 0x00000000
0x00000000 1 0x00000004 2 0x00000008 0x00000008 3 0x0000000C 4
0x00000010 0x00000010 5 0x00000014 6')
        self.assertEqual(var.value,
                         {'count': 0x1488, 'a': (0x00000000,
[(0x00000000, [(0x00000000, [1]), (0x00000004, [2])]),

(0x00000008, [(0x00000008, [3]), (0x0000000C, [4])]),

(0x00000010, [(0x00000010, [5]), (0x00000014, [6])])])})


if __name__ == "__main__":
    unittest.main()
```

ПРИЛОЖЕНИЕ И

КОД МОДУЛЯ DECLARATIONTEST

```
 import unittest

from primitives import VarDeclaration, StructDeclaration, FieldDecla-
ration, ClTypes


class VariableDeclarationTest(unittest.TestCase):
    def setUp(self) -> None:
        s1, s2 = StructDeclaration(None), StructDeclaration(None)

        s1.name = 'my_struct_1'
        s1.fields = {'count': FieldDeclaration('count', 'int'),
                     'v': FieldDeclaration('v', 'double2')}

        s2.name = 'my_struct_2'
        s2.fields = {'count': FieldDeclaration('count', 'int'),
                     'v': FieldDeclaration('v', 'my_struct_1')}

        ClTypes.struct_declarations = [s1, s2]

    def test_private_int1(self):
        v = VarDeclaration(var_name='a', full_type='__private int')
        self.assertEqual(v.var_type, 'int')
        self.assertEqual(v.address_space, '__private')
        self.assertEqual(v.is_array, False)
        self.assertEqual(v.var_shape, None)
        self.assertEqual(v.pointer_rank, 0)
        self.assertEqual(v.words_num(), 2)
        self.assertEqual(v.is_struct(), False)

    def test_private_int2(self):
        v = VarDeclaration(var_name='a', full_type='int __private')
        self.assertEqual(v.var_type, 'int')
        self.assertEqual(v.address_space, '__private')
        self.assertEqual(v.is_array, False)
        self.assertEqual(v.var_shape, None)
        self.assertEqual(v.pointer_rank, 0)
        self.assertEqual(v.words_num(), 2)
        self.assertEqual(v.is_struct(), False)
```

```
    def test_private_1d_array(self):
        v = VarDeclaration(var_name='a', full_type='__private int
[41]')
        self.assertEqual(v.var_type, 'int')
        self.assertEqual(v.address_space, '__private')
        self.assertEqual(v.is_array, True)
        self.assertEqual(v.var_shape, [41])
        self.assertEqual(v.pointer_rank, 0)
        self.assertEqual(v.words_num(), 43)
        self.assertEqual(v.is_struct(), False)

    def test_private_2d_array(self):
        v = VarDeclaration(var_name='a', full_type='__private int [41]
[5]')
        self.assertEqual(v.var_type, 'int')
        self.assertEqual(v.address_space, '__private')
        self.assertEqual(v.is_array, True)
        self.assertEqual(v.var_shape, [41, 5])
        self.assertEqual(v.pointer_rank, 0)
        self.assertEqual(v.words_num(), 248)
        self.assertEqual(v.is_struct(), False)

    def test_private_3d_array(self):
        v = VarDeclaration(var_name='a', full_type='__private int [2]
[3] [1]')
        self.assertEqual(v.var_type, 'int')
        self.assertEqual(v.address_space, '__private')
        self.assertEqual(v.is_array, True)
        self.assertEqual(v.var_shape, [2, 3, 1])
        self.assertEqual(v.pointer_rank, 0)
        self.assertEqual(v.words_num(), 16)
        self.assertEqual(v.is_struct(), False)

    def test_local_1d_array(self):
        v = VarDeclaration(var_name='a', full_type='__local int [41]')
        self.assertEqual(v.var_type, 'int')
        self.assertEqual(v.address_space, '__local')
        self.assertEqual(v.is_array, True)
        self.assertEqual(v.var_shape, [41])
        self.assertEqual(v.pointer_rank, 0)
        self.assertEqual(v.words_num(), 43)
```

```
        self.assertEqual(v.is_struct(), False)

    def test_local_2d_array(self):
        v = VarDeclaration(var_name='a', full_type='__local int [41]
[5]')
        self.assertEqual(v.var_type, 'int')
        self.assertEqual(v.address_space, '__local')
        self.assertEqual(v.is_array, True)
        self.assertEqual(v.var_shape, [41, 5])
        self.assertEqual(v.pointer_rank, 0)
        self.assertEqual(v.words_num(), 248)
        self.assertEqual(v.is_struct(), False)

    def test_local_3d_array(self):
        v = VarDeclaration(var_name='a', full_type='__local int [2]
[3] [1]')
        self.assertEqual(v.var_type, 'int')
        self.assertEqual(v.address_space, '__local')
        self.assertEqual(v.is_array, True)
        self.assertEqual(v.var_shape, [2, 3, 1])
        self.assertEqual(v.pointer_rank, 0)
        self.assertEqual(v.words_num(), 16)
        self.assertEqual(v.is_struct(), False)

    def test_struct(self):
        v = VarDeclaration(var_name='a', full_type='__private struct
my_struct_1')
        self.assertEqual(v.var_type, 'my_struct_1')
        self.assertEqual(v.address_space, '__private')
        self.assertEqual(v.is_array, False)
        self.assertEqual(v.var_shape, None)
        self.assertEqual(v.pointer_rank, 0)
        self.assertEqual(v.words_num(), 5)
        self.assertEqual(v.is_struct(), True)

    def test_undefined_struct(self):
        self.assertRaises(Exception, VarDeclaration(var_name='a',
full_type='__private struct my_struct1'))


if __name__ == "__main__":
    unittest.main()
```

ПРИЛОЖЕНИЕ К

КОД МОДУЛЯ OCLDEBUGGERTEST

```
import unittest
from shutil import copyfile

from OclDebugger import OclDebugger


class OclDebuggerTest(unittest.TestCase):
    def setUp(self) -> None:
        pass

    def test_it_all(self):
        debugger = OclDebugger(
            kernel_file='res/ocl_app/kernel.cl',
            binary='res/ocl_app/app',
            build_cmd=None  # This application doesnt need rebuilding
        )

        variables = debugger.safe_debug(break_line=35, threads=[0])
        c = variables[0]
        self.assertEqual(c.decl.var_name, 'c')
        self.assertEqual(c.value, 1)

        uc = variables[1]
        self.assertEqual(uc.decl.var_name, 'uc')
        self.assertEqual(uc.value, 2)

        s = variables[2]
        self.assertEqual(s.decl.var_name, 's')
        self.assertEqual(s.value, 3)

        us = variables[3]
        self.assertEqual(us.decl.var_name, 'us')
        self.assertEqual(us.value, 4)

        i = variables[4]
        self.assertEqual(i.decl.var_name, 'i')
        self.assertEqual(i.value, 5)
```

```
ui = variables[5]
self.assertEqual(ui.decl.var_name, 'ui')
self.assertEqual(ui.value, 6)

l = variables[6]
self.assertEqual(l.decl.var_name, 'l')
self.assertEqual(l.value, 7)


ul = variables[7]
self.assertEqual(ul.decl.var_name, 'ul')
self.assertEqual(ul.value, 8)


f = variables[8]
self.assertEqual(f.decl.var_name, 'f')
self.assertAlmostEqual(f.value, 14.31, delta=0.00001)

d = variables[9]
self.assertEqual(d.decl.var_name, 'd')
self.assertAlmostEqual(d.value, -147.1, delta=0.00001)

arr_1d = variables[10]
self.assertEqual(arr_1d.decl.var_name, 'arr_1d')
self.assertEqual(arr_1d.value[1], [14, 17])

arr_2d = variables[11]
self.assertEqual(arr_2d.decl.var_name, 'arr_2d')
self.assertEqual(arr_2d.value[1][0][1], [14, 17])
self.assertEqual(arr_2d.value[1][1][1], [15, 31])

arr_3d = variables[12]
self.assertEqual(arr_3d.decl.var_name, 'arr_3d')
self.assertEqual(arr_3d.value[1][0][1][0][1], [14, 17])
self.assertEqual(arr_3d.value[1][0][1][1][1], [15, 31])
self.assertEqual(arr_3d.value[1][1][1][0][1], [1, 2])
self.assertEqual(arr_3d.value[1][1][1][1][1], [3, 4])

s1 = variables[13]
self.assertEqual(s1.decl.var_name, 's1')
arr_2d = s1.value['count']
```

```
        self.assertEqual(arr_2d[1][0][1], [14, 17])
        self.assertEqual(arr_2d[1][1][1], [15, 31])
        self.assertEqual(s1.value['w1'], [0.1, 0.2])

        s2 = variables[14]
        self.assertEqual(s2.decl.var_name, 's2')
        s1 = s2.value['a']
        arr_2d = s1['count']
        self.assertEqual(arr_2d[1][0][1], [14, 17])
        self.assertEqual(arr_2d[1][1][1], [15, 31])
        self.assertEqual(s1['w1'], [0.1, 0.2])
        self.assertEqual(s2.value['w2'], 0.12)


if __name__ == "__main__":
    unittest.main()
```

ПРИЛОЖЕНИЕ Л

КОД ФАЙЛА KERNEL.CL

```
struct my_struct {
      uint count[2][2];
      double2 w1;
};

struct my_struct2 {
      struct my_struct a;
      double w2;
};

__kernel void test(__global int* message, __global char* debugging-
Buffer)
{
    char c = 1;
    uchar uc = 2;
    short s = 3;
    ushort us = 4;
    int i = 5;
    uint ui = 6;
    long l = 7;
    ulong ul = 8;
    float f = 14.31;
    double d = -147.1;

     int arr_1d[2] = {14, 17};
    int arr_2d[2][2] = {{14, 17}, {15, 31}};
    int arr_3d[2][2][2] = {{{14, 17}, {15, 31}}, {{1, 2}, {3, 4}}};

    struct my_struct s1 = {{{14, 17}, {15, 31}}, (double2)(0.1, 0.2)};
    struct my_struct2 s2 = {{{{14, 17}, {15, 31}}, (double2)(0.1,
0.2)}, 0.12};




}
```