

Санкт-Петербургский государственный университет

Кафедра системного программирования

Васенина Анна Игоревна

Workflow Builder для библиотеки Brahma.FSharp

Курсовая работа

Научный руководитель:
магистр ИТ, ст. преп. Григорьев С.В.

Санкт-Петербург
2017

Оглавление

Введение	3
1. Обзор	5
1.1. Brahma.FSharp	5
1.2. Монады и монада reader	5
1.3. Computation expressions	5
2. Постановка задачи	7
3. Решение	8
Заключение	10
Список литературы	11

Введение

Сегодня вычисления на графическом процессоре (GPU) становятся всё более популярными, связано это с увеличением объема вычисляемых данных и необходимостью параллельного вычисления для эффективной их обработки. Одной из распространенных технологий для программирования на GPU является OpenCL ([3]), основным преимуществом которой является возможность работы на любых устройствах, поддерживающих данный стандарт .

Стандарт OpenCL реализован для множества языков программирования, таких как Python, Java. Реализацией стандарта OpenCL для языка F# стала библиотека Brahma.FSharp ([1]).

К сожалению, на данный момент в этой библиотеке существует ряд проблем. Одна из них - это неудобство явной передачи контекста как с точки зрения программиста при написании кода, так и с точки зрения реализации последовательных операций с одним элементом.

Сейчас работа с элементами в библиотеке Brahma осуществляется по схеме, изображенной на рис. 1

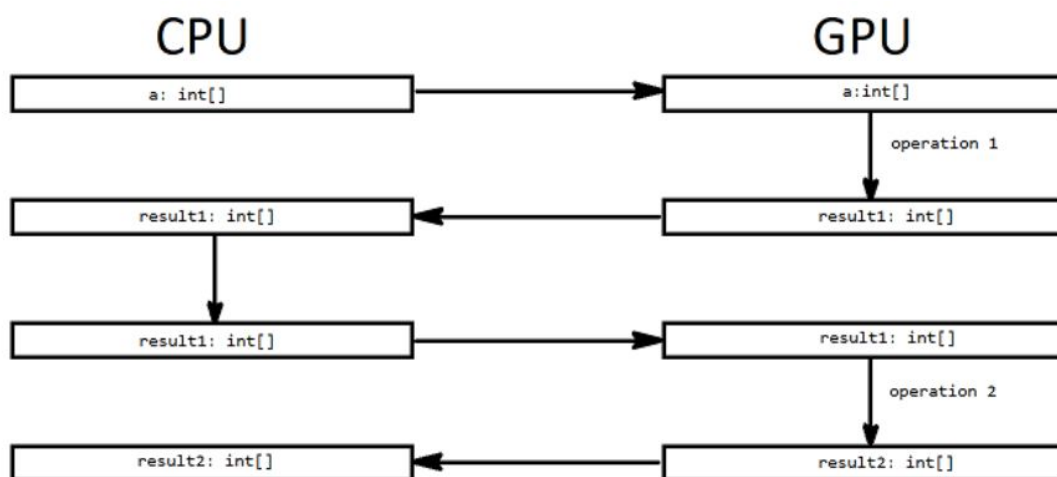


Рис. 1: Диаграмма модулей

Такая схема работы требует улучшения, потому что: Пользователю неудобно в явном виде пересылать контекст на каждом шаге Замедляется работа программы из-за пересылок Удобство написания кода в

библиотеке имеет не малое значение для программиста, использующего её. Поэтому необходимо давать как можно более удобные механизмы работы для того, чтобы пользователь был доволен библиотекой и предпочитал использовать именно её. Неявная передача контекста позволяет пользователю избежать возможных ошибок при манипуляциях с контекстом.

Возможность последовательно выполнять операции с одним массивом на GPU, не возвращая каждый раз массив не менее важна, так как обычно нам приходится иметь дело с небольшим количеством исходных массивов, над которыми нужно провести множество операций, а возвращая после каждой операции массив обратно в CPU, мы проигрываем не только в красоте и читаемости кода, но и в скорости выполнения операций.

1. Обзор

Для решения данных проблем было решено создать workflow builder. Средствами создания его выступили монада reader и computation expressions.

1.1. Brahma.FSharp

Brahma.Fsharp нацелена на трансляцию кода на FSharp в OpenCL с минимизацией различных пользовательских типов и wrappers. Данная библиотека предоставляет пользователю следующие возможности:

- использование OpenCL для работы на GPU с любыми устройствами, которые поддерживают OpenCL, например с устройствами AMD и Nvidia;
- поддерживает кортежи и структуры;
- использование строго типизированных ядер из OpenCL.

1.2. Монады и монада reader

([4]) Монада это конструкция функционального программирования, которая позволяет применить функцию, возвращающую упакованное значение к упакованному значению. Монада reader (см. рис.2) (1.2) позволяет нам неявно передать какие-то настройки в функцию неявно, скрывая сам процесс передачи этих настроек за кулисами. Она организует упаковку функции в контекст и его распаковку для применения функции. Монаду характеризуют функции bind (\gg) для соединения содержательной части с контекстом и run для их отделения.

1.3. Computation expressions

([2]) По сути своей computation expressions представляют некую именованную среду, ограниченную фигурными скобками (например,

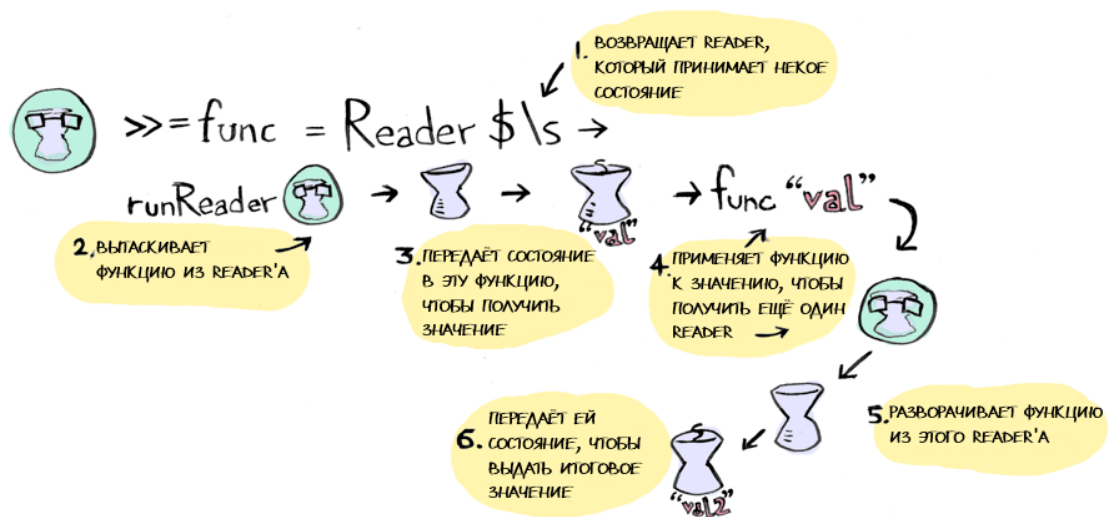


Рис. 2: Рис. 2 Схематичная работа монады Reader Источник: <https://habrahabr.ru/post/184722/>

гри...), для которой мы переопределяем ключевое слово `let`! необходимым нам образом, а так же определяем другие ключевые слова, такие как `return`. Каждый `computation expression` работает с некоторым типом-оберткой, именно его мы и используем для упаковки контекста и содержательной составляющей данных.

2. Постановка задачи

При написании данной работы преследовалось две основные цели:

- повысить удобство написания кода для пользователя путем неявной передачи в функции заданный в начале работы контекст;
- возможность выполнять последовательно операции с массивами на GPU, не возвращая массив каждый раз после исполнения функции.

Для их выполнения были поставлены следующие задачи:

- Реализовать модуль `gpu...` и конструктор таких модулей для разных контекстов. Внутри которых происходит работа с одним контекстом, передающимся выполняемым функциям неявно.
- Протестировать данный модуль на нескольких примерах с использованием функций массивов из модуля `ArrayGPU`, снабдить их комментариями.

3. Решение

Для решения поставленной задачи был создан специальный тип `ReaderM <'d,'out>` являющийся оберткой для нашего `computation expression`. Также необходимо было научить работать данный `computation` с этим типом. Для этого были применены методы монады `reader`: `bind` для переопределения ключевого слова `let!` и `constant` для того, чтобы обернуть значение при вызове функций `return` и `yield`. `Reader.run` явно вызывается только после получения результата `computation`'а для того, чтобы отделить нужные данные от контекста.

Благодаря функции `bind` мы больше не должны манипулировать контекстом: внутри модуля `gpu` получаем следующую ротацию типов.

<pre>let comp1 = gpu { let a = [5; 7; 8; 22; 16] let! c = <u>ArrayGPU.Reverse</u> a return c }</pre>	<pre>a: int [] <u>ArrayGPU.Reverse</u>: fun 'a[] -> context -> 'a[] c: int [] comp1: ReaderM<context, 'a[]></pre>
--	---

Как можно видеть, модуль `gpu` решает вопрос неявной передачи контекста для удобства пользователя.

Контекст, который передается функциям, монада `reader` берет первый подходящий, который она встретит в модуле. Однако, при этом все равно необходимо передавать контекст в билдер для того, чтобы не пришлось вручную переводить массив на CPU при выполнении функции `return`, а также очищать буфера и сбрасывать очередь команд после завершения работы - команда `return` сделает это за пользователя.

Однако, нам необходимо иметь также возможность вернуться из композиции `computation`'ов, не проводя манипуляций с контекстом. Для этого мы используем метод `Yield`, который так же, как и `return` использует функцию `constant` для того, чтобы обернуть результат типом `ReaderM <'d,'out>`, но не очищает буфера и не сбрасывает очередь ко-

манд. Благодаря этому мы получаем возможность построения таких конструкций:

```
let outerComputation inArr =  
    gpu  
    {  
        let! e = ArrayGPU.Reverse inArr  
        let! f = ArrayGPU.Map <@ fun a -> a + 1 @> e  
        yield f  
    }  
let computation 1 = gpu  
{  
    let a = [5; 7; 8; 22; 16]  
    let! c = ArrayGPU.Reverse a  
    let! d = outerComputation a  
    let! g = ArrayGPU.Map2 <@ fun a b -> a + b @> c d  
    return g  
}
```

Для того, чтобы использовать метод `yield` необходимо определить метод `Zero`, который говорит, какое значение присвоить всему выражению, если `computation` пытается вернуть тип `unit`. В данном решении мы определили, что `computation` должен вернуть обернутый `None`.

Заключение

В ходе работы получены следующие результаты:

- реализована возможность выполнения последовательных операций на GPU;
- внутри модуля `gru..` производится работа с заранее определенным контекстом вида `(ComputeProvider * CommandQueue * int * int)`. Работать с функциями получающими такой контекст последним из аргументов можно без указания контекста.

В качестве дальнейшего развития можно рассмотреть следующее направление: в некоторых случаях возникает необходимость работать в разных контекстах с одними данными. На данный момент работа с разными контекстами не поддерживается, но в рамках `workflow builder`'а это можно реализовать при использовании композиции нескольких `computation expressions` (например, `gru1..` и `gru2..`), каждый из которых работает со своим контекстом.

Список литературы

- [1] Проект Brahma.FSharp [Электронный ресурс]. — URL: <https://github.com/YaccConstructor/Brahma.FSharp> (online; accessed: 17.10.2017).
- [2] Серия статей 'Computation Expressions' [Электронный ресурс]. — URL: <https://fsharpforfunandprofit.com/series/computation-expressions.html> (online; accessed: 17.10.2017).
- [3] Спецификация OpenCL 1.2 [Электронный ресурс]. — URL: <https://www.khronos.org/registry/OpenCL/specs/opencvl-1.2.pdf> (online; accessed: 17.10.2017).
- [4] Статья 'Тройка полезных монад' [Электронный ресурс]. — URL: <https://habrahabr.ru/post/184722/> (online; accessed: 17.10.2017).