



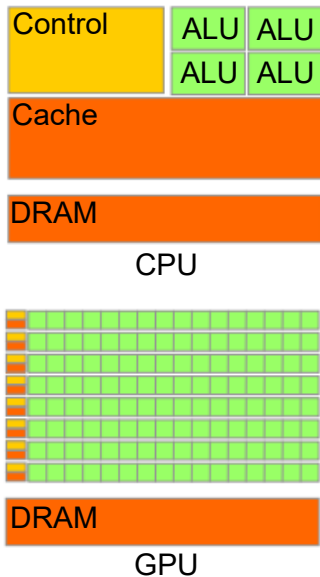
F# OpenCL C Type Provider

Kirill Smirenko, **Semyon Grigorev**

JetBrains Research, Programming Languages and Tools Lab
Saint Petersburg University

September 27, 2018

GPGPU



General purpose computations on graphical processor units

- (Almost) SIMD architecture
- Huge amount of “simple” ALUs on single chip
- Initially for computer graphic/games etc
- Good choice for big data processing

General purpose applications of GPGPU

- Initially for scientific computations
 - ▶ Physics
 - ▶ Math
 - ▶ Chemistry
- But more and more for applications
 - ▶ Finance/Banking
 - ▶ Data Analytics and Data Science (Hadoop, Spark ...)
 - ▶ Security analytics (log processing)
 - ▶ Some “scientific computations” today are daily-used applications (bioinformatics, chemistry , ...)

High level languages and GPGPU

Low-level platforms and languages
for GPGPU programming

- NVIDIA CUDA: Cuda C,
Cuda Fortran
- **OpenCL: OpenCL C**

High-level platform and languages
for applications

- C++
- Python, Haskell, OCaml, ...
- JVM: Java, Scala, ...
- .NET: C#, F#, ...

High level languages and GPGPU

Low-level platforms and languages
for GPGPU programming

- NVIDIA CUDA: Cuda C,
Cuda Fortran
- **OpenCL: OpenCL C**

High-level platform and languages
for applications

- C++
- Python, Haskell, OCaml, ...
- JVM: Java, Scala, ...
- .NET: C#, F#, ...

Interaction is a problem!

Existing solutions and its problems

- Generative approach (haskell, Alea, etc): good but what about code reusing?
- Simple drivers (textual, for example) — flexible but not safe.

- F# quotations to OpenCL C translator
- Runtime
 - ▶ Command queue
 - ▶ Context management
 - ▶ Memory management
 - ▶ F# aliases for OpenCL-specific functions

F# type providers

- Compile-time metaprogramming technique for compile-time types creation
 - ▶ Type provider is a “function which constructs type”
- Design-time features in IDE
 - ▶ Completion
 - ▶ Type information
- Used for type-safe integration of external data with “fixed schema”
 - ▶ Type providers for XML, JSON, INI, etc
 - ▶ R, SQL, ets type providers

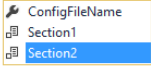
Example of INI type provider

```
[Section1]
intSetting = 2
stringSetting = stringValue
[Section2]
floatSetting = 1.23
boolSetting = true
anotherBoolSetting = False
emptySetting =
stringWithSemiColonValue = DataSource=foo@bar;UserName=blah
```

```
open FSharp.Configuration
```

```
type Config = IniFile<"Config.ini">
```

```
Config.
```



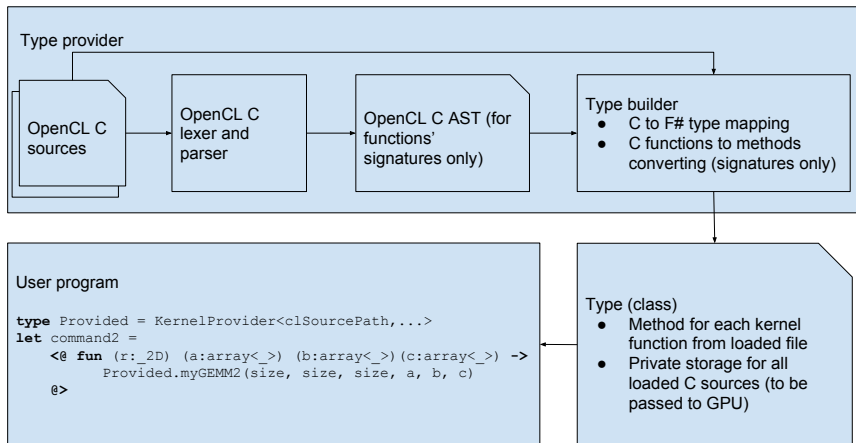
ConfigFileName

Section1

Section2

```
type Section2 =
    static member anotherBoolSetting : bool
    static member boolSetting : bool
    static member emptySetting : string
    static member floatSetting : float
    static member stringWithSemiColonValue : string
```

OpenCL C type provider



Really, typical type provider

Limitations

- Only (small) subset of OpenCL C
 - ▶ h files is not supported
 - ▶ preprocessor is not supported
 - ▶ only small subset of syntax is supported
- Very simple C to F# type mapping
- ...

Examples

Screens

Future work

- Improve OpenCL C support
 - ▶ Lexer and parser
 - ▶ Translator
 - ▶ Types mapping
 - ▶ Headers files processing
 - ▶ ...
- Unify kernels on client side
 - ▶ Currently native Brahma.FSharp's kernel and kernel loaded by type provider are different types
- Improve mechanism of kernels composition

Summary

- F# OpenCL C type provider
 - ▶ Type-safe integration of existing OpenCL C code in F# applications
 - ▶ Prototype with limitations
- Source code on GitHub:
<https://github.com/YaccConstructor/Brahma.FSharp>
- Package on NuGet:
<https://www.nuget.org/packages/Brahma.FSharp/>

Contact Information

- Semyon Grigorev: s.v.grigoriev@spbu.ru
- Kirill Smirenko: k.smirenko@gmail.com
- Brahma.FSharp:
<https://github.com/YaccConstructor/Brahma.FSharp>

Thanks!