Brahma.FSharp: Power of Functional Programming to Create Portable GPGPU-enabled .NET Applications

Nikolai Ponomarev[1] [0009−0000−2382−5687], Vladimir Kutuev[1] [0000−0001−7749−4940], and Semyon Grigorev[1] [0000−0002−7966−0698]

[1] Saint Petersburg State University, 7-9 Universitetskaya Embankment, St. Petersburg, Russia

[2] n.ponomarev@spbu.ru, v.kutuev@spbu.ru, s.v.grigoriev@mail.spbu.ru

Abstract. Widening of GPGPU applicability increases the interest to high-level languages for GPGPU programming development. One of the challenges is to create a portable solution which provides static checks and being integrated with application platforms. We propose a tool — Brahma.FSharp — that allows one to utilize OpenCL-compatible devices in .NET applications, and to develop homogeneous code using familiar .NET tools. Brahma.FSharp provides ability to create GPU kernels using F# programming language that is functional-first statically typed .NET language. Compile-time metaprogramming techniques, provided by F#, allows one to develop generic type-safe kernels. We show portability of the proposed solution by running several algorithms developed with it across different platforms and devices.

Keywords: Functional programming · GPGPU · FSharp · .NET · OpenCL

1 Introduction

Last decades utilization of GPGPUs not only in scientific or dedicated applications, but also in regular business applications becomes more popular. In such cases not peak performance, but transparent offloading of computations to accelerator has come into focus. As a result, respective tools for integration of GPGPUs into such platforms as JVM [7,12,6] or .NET [2,4] are developed. Note that in real-world application the problem no only to offload some computations on GPGPU, but to orchestrate heterogenous asynchronous application that involves computations on possible several GPGPUs.

At the same time, utilization of existing functional languages and creation new ones for GPGPU programming, looks promising due to them safety, flexibility, ability to use advanced optimization techniques and to create high-level abstractions. That lead to such projects as Futhark [3], Lift [8], AnyDSL [5], Accelerate [1].

Nowadays there are very few combination of mature business application development platform and functional programming language. One of them is a

.NET platform and F# programming language. There are several tools, such as Alea.GPU [4], FCSL [2], ILGPU [3], that allows one to integrate GPGPUs into .NET application without using low-lwvwl mechanisms like string-level kernels creation. While FSCL and Alea.GPU use F# to create kernels, ILGPU works on IL level that limits ability to use high-level features and nontrivial optimizations.

In this work we propose a **Brahma.FSharp**[4] — the tool for portable GPGPU-enabled .NET applications development that provides transparent and safe integration with accelerators — and demonstrate it's portability across variety of platforms and devices.

2 Brahma.FSharp

Brahma.FSharp is a tool that allows one to utilize GPGPUs in .NET applications and write kernels and all supplementary code in pure F# [10] that is a functional-first multiparadigmal programming language for .NET platform. This language combines functional programming, including first-class functions, generics, static strong typing with automatic type inference, with transparent integration with the a platform for business applications development with mature infrastructure. At the same time, F# provides an ability to write imperative code that is native for kernel programming.

Core of the tool is a translator of F# subset to OpenCL C that is based on *code quotations* [9] that allows one to get access to annotated tree of the F# code and transform it during program execution. This tree can be transformed using regular F# functions: for example, it can be translated to other language, that is we do to generate OpenCL C code for kernels. Other words, code quotations is a running time metaprogramming feature that allows us to create running-time configurable kernels. For example, it is possible, in opposite to compile time metaprogramming, configure work group size dependent parts of kernel (e.g. local buffer size) without recompilation of whole program (look at line 9 of listing 1). The main feature is that all is strongly and statically typed: no unsafe code that uses strings, pointers, objects, etc. At the user side, compiled quotation (compiled kernel) has the signature that requires parameters of types that are in agreement with initial quotation.

An example of quoted code (actually, part of the generalized mXm kernel) is presented in listing 1 (lines 6–12). This code also demonstrates typed composition of quotations: operations `opAdd` and `opMult`, and identity element `zero`, have agreed types and can be specified outside the kernel in run time. So, we can write highly configurable kernels generator and instantiate specific kernels later, as shown in lines 15–16.

The translator supports not only imperative subset of F# and primitive types, but also F#-specific features like structs, tuples, discriminated unions, pattern matching, nested bindings. Also it supports OpenCL specific features

[3] ILGPU project web page: `https://ilgpu.net/`

[4] Sources of Brahma.FSharp: `https://github.com/YaccConstructor/Brahma.FSharp`.

like atomic functions, barriers, local and thread-local allocation arrays allocation. For data transferring and manipulation Brahma.FSharp provides specified memory primitives (`ClArray<'t>` and `ClCell<'t>`) that is F#-array-friendly wrappers around `ClBuffer`.

Brahma.FSharp provides typical workflow to run kernels and implements respective typed wrappers for it[5]. It is worth noting that F# is friendly to asynchronous programming and provides huge amount of parallel and asyncronious programming primitives [11]. Utilization of *mailbox processor*, that is F#-native massage passing primitive, to wrap command queue allows us to make communication with GPGPU friendly for asynchronous programming in F#.

3 Evaluation

In this section we provide experiments[6] with Brahma.FSharp platform which are aimed to demonstrate its main features.

We evaluated Brahma.FSharp in two cases listed below and described in respective sections.

1. The first one is an image convolution. to demonstrate async
2. Second one is a matrix multiplication. to demonstrate generics, local and private memory support. To evaluate on different devices.

On several platforms.

- Intel
- NVIDIA
- ImTech, PowerVR, RISC-V
- Qualcomm, Mali, ARM

3.1 Image Convolution

Reading and writing. F# MailboxProcessor used for composing of data reading, data processing on GPGPU, and data processing on CPU.

Graphics, tables.

[4]

Gray scale.

Multiple GPU-s.

3.2 Matrix Multiplication

Classical task for GPGPU.

Several optimizations.

Generic kernels parametrized by types and operations.

[5] Configure of path to `libopencl` making the solution portable.

[6] Related sources: `!!!`

Code examples.

Sequence of optimizations inspired by !!![7]. Not all, but memory Square matrix.

Flexibility. Kernels are parametrized by operations and id. Unsafe Min-plus using max value as ID. Matrix of 'e

More accurate min-plus using options.

```
let mXmKernel
   (opAdd: Quotations.Expr<'a -> 'b -> 'a>)
   (opMult: Quotations.Expr<'e -> 'f -> 'b>)
   (zero: Quotations.Expr<'a>) ... (* other parameters *)  =
      ... // Supplementary code
      let kernel = <@ fun 2dRange m1 m2 res ->  // Quoted code
        ...
        let acc = %zero // Embedded identity value
        let lBuf = localArray lws // captured from context
        ...
        acc <- (%opAdd) acc ((%opMult) x y) // Embedded operations
        ... @>
      ... // Supplementary code

let intArithmeticKernel = mXmKernel <@ (+) @> <@ ( * ) @> <@ 0 @>
let intMinPlusKernel =
    mXmKernel <@ (min) @> <@ (+) @> <@ Int.MaxValue @>
```

Listing 1: An example of masking operation definition

Graphics, tables.

4 Conclusion and Future Work

Brahma.FSharp — a tool to create cross-platform GPGPU-enabled .NET application presented. We demonstrated portability of application by evaluating them on a set of platforms including RISC-V with PowerVR GPPGU and ARM with Mali GPGPU.

While work still in progress, Brahma.FSharp allows one to create linear algebra related kernels performant enough to be integrated in libraries like Math.Net Numerics to offload generic linear algebra transparently to GPGPU. Such an integration is planned to the nearest future.

Also, within the translator improvements, it is necessary to improve performance of data transferring between managed and native memory for complex types such as discriminated unions.

[7] !!!

While agent-based approach for communications is a native for both OpenCL and F#, mailbox processor may not be the best choice for it especially in cases with high-frequent CPU-GPU communications. It may be better to use more performant libraries like Hopac [8] or even provide light-weight wrapper for direct access to command queue for latency-critical code.

One of nontrivial problem for the future research is an automatic memory management. For now, GPGPU-related memory should be cleaned manually, but .NET has automatic garbage collector. How can we offload buffers management on it with ability ti switch to manual control if required.

References

1. M. M. Chakravarty, G. Keller, S. Lee, T. L. McDonell, and V. Grover. Accelerating haskell array codes with multicore gpus. In *Proceedings of the Sixth Workshop on Declarative Aspects of Multicore Programming*, DAMP '11, page 3–14, New York, NY, USA, 2011. Association for Computing Machinery.
2. G. Coco. *Homogeneous programming, scheduling and execution on heterogeneous platforms*. PhD thesis, Ph. D. Thesis.\Gabriel Coco.-University of Pisa, 2014.-254 p, 2014.
3. T. Henriksen, N. G. W. Serup, M. Elsman, F. Henglein, and C. E. Oancea. Futhark: purely functional gpu-programming with nested parallelism and in-place array updates. *SIGPLAN Not.*, 52(6):556–571, June 2017.
4. P. Kramer, D. Egloff, and L. Blaser. The alea reactive dataflow system for gpu parallelization. In *Proc. of the HLGPU 2016 Workshop, HiPEAC*, 2016.
5. R. Leißa, K. Boesche, S. Hack, A. Pérard-Gayot, R. Membarth, P. Slusallek, A. Müller, and B. Schmidt. Anydsl: a partial evaluation framework for programming high-performance libraries. *Proc. ACM Program. Lang.*, 2(OOPSLA), Oct. 2018.
6. N. Nystrom, D. White, and K. Das. Firepile: run-time compilation for gpus in scala. In *ACM SIGPLAN Notices*, volume 47, pages 107–116. ACM, 2011.
7. P. C. Pratt-Szeliga, J. W. Fawcett, and R. D. Welch. Rootbeer: Seamlessly using gpus from java. In *High Performance Computing and Communication & 2012 IEEE 9th International Conference on Embedded Software and Systems (HPCC-ICESS), 2012 IEEE 14th International Conference on*, pages 375–380. IEEE, 2012.
8. M. Steuwer, T. Remmelg, and C. Dubach. Lift: a functional data-parallel ir for high-performance gpu code generation. In *Proceedings of the 2017 International Symposium on Code Generation and Optimization*, CGO '17, page 74–85. IEEE Press, 2017.
9. D. Syme. Leveraging. net meta-programming components from f#: integrated queries and interoperable heterogeneous execution. In *Proceedings of the 2006 workshop on ML*, pages 43–54. ACM, 2006.
10. D. Syme, A. Granicz, and A. Cisternino. *Expert F# 3.0.* Springer, 2012.
11. D. Syme, T. Petricek, and D. Lomov. The f# asynchronous programming model. In *International Symposium on Practical Aspects of Declarative Languages*, pages 175–189. Springer, 2011.

[8] Hopac and mailbox processor performance comparison: `https://vasily-kirichenko.github.io/fsharpblog/actors`

12. Y. Yan, M. Grossman, and V. Sarkar. Jcuda: A programmer-friendly interface for accelerating java programs with cuda. In *European Conference on Parallel Processing*, pages 887–899. Springer, 2009.