F# and GPGPU

Kirill Smirenko
St. Petersburg State University
7/9 Universitetskaya nab.
St. Petersburg, 199034,
Russia
k.smirenko@gmail.com

Polina Lunina
St. Petersburg State University
7/9 Universitetskaya nab.
St. Petersburg, 199034,
Russia
lunina_polina@mail.ru

Julia Susanina
St. Petersburg State University
7/9 Universitetskaya nab.
St. Petersburg, 199034,
Russia
jsusanina@gmail.com

Semyon Grigorev
St. Petersburg State University
7/9 Universitetskaya nab.
St. Petersburg, 199034,
Russia
semen.grigorev@jetbrains.com

ABSTRACT

Aaabstraaact!!!! abstract

CCS Concepts

•**Software and its engineering** → **Automated static analysis; Software maintenance tools;** •**Theory of computation** → *Program analysis; Parsing;*

Keywords

GPGPU, OpenCL, F#, metaprogramming, !!!!!

1. INTRODUCTION

GPGPU is popular technique for....

Tools and languages are low level. It is ood for high performance, but bad for developers.

OpenCL, CUDA etc.

ACM ISBN 978-1-4503-2138-9.
DOI: 10.1145/1235

Complex problems, geterogenious platforms: multicore, multi GPGPU etc. Special tools, libs required for development simlification. High level languages and platforms are used for application development.

F# primitives are helpful for metaprogramming and parallel/asyncronious programming.

General reqirenments: highlevel languge, existing code/dlls/other stuff reusing

Existing solutions, auch as Alea.GPU, FCSL, are not good enough. Why? Many different attempts for high level platform, such as JVM

Brahma.FSharp — the best platform for GPGPU programming!!!! Quotations to OpenCL translator with many cool features.

2. F# PROGRAMMING LANGUAGE

In this section F# [**?**] — is a functional-first multiparadigmal programming language for .NET platform.

Main important features described.

2.1 Code quotation

Cool feature for metaprogramming.

```
let quoted =
    <@
        fun x y -> x + y
    @>
```

Listing 1: Example of F# code quotation

2.2 Type providers

Yet another feature fore metaprogramming.

2.3 Async MBP etc

F# immutable by default — useful for parallel programming.

F# provides huge amount of parallel and asyncronious programming primitives.

Message-passing model.

3. RELATED WORK

Existing solution for GPGPU programming...

3.1 FSCL

Status [**?**]

3.2 Alea CUDA

CUDA only [**?**]

3.3 Managed Cuda etc

Hm...

4. BRAHMA.FSHARP

In this section we present our platform for GPGPU programming in F#.

Research project: SPbU and JetBrains Research.

Blah-Blah-Blah!!!!

4.1 Architecture

Based on F# code quotation to OpenCL translator.

Driver is OpenCL.NET [1]

Picture.

Workflow:

Details of some blocks are described below.

4.2 Translator

Classical techniques: var scopes,

Structs, tuples, etc

4.3 OpenCL specific operations

Atomic functions.

Memory transferring.

4.4 OpenCL type provider

5. EVALUATION

Implemented. Sources available here: https://github.com/YaccConstructor/Brahma.FSharp Binary package available here: https://www.nuget.org/packages/Brahma.FSharp/ Examples: https://github.com/YaccConstructor/Brahma.FSharp.Examples

Matrix multiplication

Substring matching

Substring matching with agents [**?**] [**?**] [**?**]

Results of performance test of GPGPU calculation using Brahma.FSharp and MailboxProcessor composition are presented. Problem to solve is substring matching for data carving. Rabin-Karp algorithm was implemented using Brahma.FSharp for substring matching. F# MailboxProcessor used for composing of data reading, data processing on GPGPU, and data processing on CPU. Library for fast and flexible configuration of MailboxProcessors was created. Set of templates for search was fixed. Tests were performed for HDD and SSD storages. Low level sequential reading was implemented. First 16.5 Mb was processed.

- OS: Microsoft Windows 8.1 Pro
- System Type: x64-based PC
- Processor: Intel(R) Core(TM) i7-4790 CPU @ 3.60GHz, 3601 Mhz, 4 Core(s), 8 Logical Processor(s)
- RAM: 16.0 GB
- HDD for test:
  - Model: ST3250410AS
  - Size: 232.88 GB
  - 7200 rpm
- SSD for test
  - Model: INTEL SSDSC2BW240A4
  - Size: 223.57 GB
  - Max read speed: 540 Mb/sec
- GPGPU:
  - NVIDIA GeForce GTX 560 Ti
  - CUDA Cores: 384
  - Core clock: 822 MHz
  - Shader clock: 1645 MHz
  - Memory data rate: 4008 MHz
  - Memory interface: 256-bit
  - Memory bandwidth: 128.26 GB/s
  - Total available graphics memory: 4095 MB
  - Dedicated video memory: 2048 MB GDDR5
  - Shared system memory: 2047 MB

Tables below present results of tests. “buffers for data” — a number of arrays to fill by disc reader for each MailboxProcessor which communicate with GPGPU. “threads” — a number of MailboxProcessors which communicate with GPGPU. In current configuration we have only one GPGU, so all MailboxProcessors use it. For multi-GPGPU systems we can configure k MailboxProcessors for each GPGPU.

In each cell — total time and GPGPU loading graph.

Conclusion: Data reading bufferization can sufficiently increase performance. Especially for HDD, where speed of reading is low. For SSD processing with multi-GPGPU systems may be useful. Data reading is not so critical as for HDD and more than one GPGPU can be fully loaded by using flexible MailboxProcessors configuration. Configuration with two MailboxProcessors and two buffers for each of them can fully load one GPGPU.

6. CONCLUSION AND FUTURE WORK

Platform presented.

Education. Metaprogramming, translators development, GPGPU programming, etc.

Graph parsing.

Geterogenious porgramming generalization. Hopac is better then MBP [2].

Research: Automatic memory management.

Data to code translation (automata can be translated into code instead of data structures in memory)

Other technical improvements: IDE support, type provider improvements, new OpenCL standard support, runtime extension, etc.

[1] OpenCL.NET — low level .NET bindings for OpenCL. Project site [visited: 20.06.2017]: https://openclnet.codeplex.com/.

[2] https://vasily-kirichenko.github.io/fsharpblog/actors

Table 1: WEWEW

1	2	3
4	5	6
7	8	9

Acknowledgments

We are grateful to the !!!! and !!! for their careful reading, pointing out some mistakes, and invaluable suggestions. This work is supported by grant from JetBrains Research, and by grant UMNIK!!!!.

7. REFERENCES

[1] L. Bläser, D. Egloff, O. Knobel, P. Kramer, X. Zhang, and D. Fabian. Alea reactive dataflow: Gpu parallelization made simple. In *Proceedings of the companion publication of the 2014 ACM SIGPLAN conference on Systems, Programming, and Applications: Software for Humanity*, 2014.

[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] P. Kramer, D. Egloff, and L. Blaser. The alea reactive dataflow system for gpu parallelization. In *Proc. of the HLGPU 2016 Workshop, HiPEAC*, 2016.

[4] D. Mikhaylov. *Transparent usage of massively parallel architectures for local optimizations of .NET applications*. PhD thesis, Graduation Thesis.\Dmitry Mikhaylov.-St. Petersburg State University, 2013.-47 p, 2013.

[5] QuanAlea Inc. Alea gpu. https://www.quantalea.net/. Accessed: 2017-06-20.

[6] D. Syme, A. Granicz, and A. Cisternino. *Expert F# 3.0*. Springer, 2012.