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 [3], CUDA [6] 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 [9, 11, 16, 7, 2]

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

2. F# PROGRAMMING LANGUAGE

In this section F# [14] — is a functional-first multiparadigmal programming language for .NET platform.

Main important features described.

2.1 Code quotation

Cool feature for metaprogramming [12].

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

Listing 1: Example of F# code quotation

2.2 Type providers

Yet another feature fore metaprogramming [13, 8].

2.3 Async MBP etc

F# immutable by default — useful for parallel programming.

F# provides huge amount of parallel and asyncronious programming primitives [15].

Message-passing model.

3. RELATED WORK

Existing solution for GPGPU programming...

3.1 FSCL

Status [1]

3.2 Alea CUDA

CUDA only [10]

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.

GPGPU driver is OpenCL.NET [1]

Picture.

Workflow:

Detailed description of some blocks are provided below.

4.2 Translator

Subset of F# to openCL. Classical techniques: var scopes,

Strongly typed: exactly same signature, but logic is changed.

Structs, tuples, etc

4.3 OpenCL specific operations

Atomic functions. For kernal code only.

Memory transferring.

4.4 OpenCL type provider

It is necessary to provide mechanism for existing kernels using. OpenCL kernels destribution — source code

Create strongly typed functions from existing OpenCL kernels code.

5. EVALUATION

In this section we provide results of some experiments with Brahma.FSharp platform which are aimed to demonstarte its main features.

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

5.1 Matrix multiplication

Classical task for GPGPU.

Naive, optimized in F#, optimized via type provider.

Code examples.

And with type providers too

5.2 Substring matching

Data recovery.

CPU vs GPGPU.

Algorithm is not important for real data. Data transferring is bottleneck.

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

5.3 Substring matching with agents

Agents forever [5] [4]

Geterogenious, multi-GPGPU platforms

Substring matchng from previous section.

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.

Table 1: WEWEW

1	2	3
4	5	6
7	8	9

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.

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 from Russian Foundation for Assistance to Small Innovative Enterprises (UMNIK program, №162GU1/2013 and №6919GU2/2015).

7. REFERENCES

[1] G. Coco. *Homogeneous programming, scheduling and execution on heterogeneous platforms*. PhD thesis, Ph. D. Thesis.\Gabriel Coco.-University of Pisa, 2014.-254 p, 2014.

[2] E. Holk, M. Pathirage, A. Chauhan, A. Lumsdaine, and N. D. Matsakis. Gpu programming in rust: implementing high-level abstractions in a systems-level language. In *Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW), 2013 IEEE 27th International*, pages 315–324. IEEE, 2013.

[3] Khronos Group. The open standard for parallel programming of heterogeneous systems. opencl 2.2. https://www.khronos.org/opencl/. Accessed: 2017-06-20.

[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] 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.

[6] Nvidia Inc. Cuda c programming guide. version 8.0. http://docs.nvidia.com/cuda/cuda-c-programming-guide. Accessed: 2017-06-20.

[7] 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.

[8] T. Petricek, G. Guerra, and D. Syme. F# data: Making structured data first class citizens. *Submitted to ICFP*, 2015, 2015.

[9] 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.

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

[11] J. Svensson, M. Sheeran, and K. Claessen. Obsidian: A domain specific embedded language for parallel programming of graphics processors. In *Symposium on Implementation and Application of Functional Languages*, pages 156–173. Springer, 2008.

[12] 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.

[13] D. Syme, K. Battocchi, K. Takeda, D. Malayeri, J. Fisher, J. Hu, T. Liu, B. McNamara, D. Quirk, M. Taveggia, et al. Strongly-typed language support for internet-scale information sources. *Technical Report MSR-TR-2012–101, Microsoft Research*, 2012.

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

[15] 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.

[16] 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.

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