Hardware aware code generation for Rust

Klas Segeljakt <klasseg@kth.se>

Contents

List of Tables

List of Figures

1 Abstract

2 Introduction

- ***General introduction to the area.***

2.1 Problem

- ***Problem definition.***
- ***Problem statement.***
- ***References.***

Current distributed systems are not able to support continuous deep analytics (CDA) at a large scale (**???**). In addition, distributed systems are becoming more heterogenous (Chafi *et al.*, 2010). This requires developers of CDA to have expertise with multiple APIs and programming models which interface with the drivers, e.g., CUDA, OpenCL, OpenMP, MPI. Developers must also stay up to date with the regularly-Due to this complexity, many general-purpose distributed systems, e.g., Spark and Flink, are written in high level languages such as Java and Scala. This in turn sacrifices performance for clarity, as running applications on the JVM incurs overhead. Evaluation by (Essertel *et al.*, 2017) has shown that a 128 node native Spark cluster running PageRank can be outperformed by a single laptop.

A possible approach to mitigate the performance loss is to use a Domain Specific Language (DSL) (Brown *et al.*, 2011). DSLs are minimalistic languages, tailor-suited to a certain domain. They bring domain-specific optimizations which general-purpose languages such as Java are unable to provide. DSLs can optimize further by generating code in a low level language, and compiling it to binary code which naturally runs faster than byte code. C and C++ are commonly used as the target low-level language. We regard Rust as a candidate as it provides safe and efficient memory management through ownership.

For a given query, the DSL must both optimize each stage of the query and the query plan as a whole. Another issue is User Defined Functions (UDFs). UDFs are essentially black boxes whose functionality might not be known at compile time. The task of optimizing these is a monumental challenge, and will be left out for future work. Another topic of interest is whether code also could be generated for the network layer. Most modern day switches are programmable, and could allow for further optimizations (**???**).

Previous work by (**???**) has shown that Spark's performance can be improved to be much faster through the use of DSLs. There is no equivalent solution yet for Flink, and this thesis aims to address this. The problem can be summarized as the following problem statement: How can Apache Flink's performance be improved through a Rust DSL?

2.2 Purpose

- ***Motivation behind writing this report.***

The purpose of this thesis is to provide a useful resource for developers of distributed systems. It is specifically aimed at those who seek to boost the performance of distributed systems which are built on top of the JVM.

2.3 Goal/Contributions

- ***Concrete deliverables.***

The goal of this thesis is to explore the area of DSLs with respect to Rust and Flink. The following deliverables are expected:

- A Rust DSL written in Scala.
- An evaluation of Apache Flink's performance before and after integrating the Rust DSL.

2.4 Benefits, Ethics and Sustainability

- ***References.***

Flink is being used by large companies such as Alibaba, Ericsson, Huawei, King, LINE, Netflix, Uber, and Zalando. As performance is often a crucial metric, these companies may see benefits in their business from the outcome of this thesis. As an example, Alibaba uses Flink for optimizing search rankings in realtime. By improving Flink's performance, it may allow for more complex and data intensive search rank optimizations. This will in consequence be beneficial for the customer whom will have an easier time finding their product.

Low level code is able to utilize hardware more efficiently than high-level code. Thus, better performance in this case also implicates less waste of resources. As a result the power usage goes down, which is healthy and sustainable for the environment.

In terms of ethics, it is crucial that the code generator does not generate buggy code which could compromise security. Keeping the code generation logic decoupled from the client code is also important. Without this consideration, an attacker would be able to generate malicious code to stage an attack with ease.

2.5 Related Work

- ***What are the existing solutions?***

Our approach to optimizing Flink draws inspiration from Flare which is a back-end to Spark (Essertel *et al.*, 2017). Flare bypasses Spark's inefficient abstraction layers by 1. Compiling queries to native code, 2. Replacing parts of the Spark runtime, and by 3. Extending the scope of optimizations and code generation to UDFs. Flare is built on top of Delite, a compiler framework for high performance DSLs, and LMS, a generative programming technique. When applying Flare, Spark's query performance improves significantly and becomes equivalent to HyPer, which is one of the fastest SQL engines.

Weld is a framework

2.6 Delimitations

- ***What was intentionally left out?***
- ***References.***

Only Rust will be used as the target language for code generation. It would be interesting to compare its performance to C and C++, but this is out of scope for this thesis. Another delimitation is regarding the problem of optimizing UDFs. Existing solutions to this problem will be described in the background. The task of inventing a new solution, specialized for Flink, will be left for future studies.

3 Background

Apache Flink is a streaming engine

3.1 Domain Specific Languages (DSL)

3.2 Code generation

- Generate binary directly, or generate Rust and compile?

3.2.1 Lightweight Modular Staging (LMS)

3.2.2 Delite

3.3 Apache Flink

3.4 Bottlenecks

3.5 Hardware acceleration

4 Design

5 Implementation

6 Evaluation

7 Discussion

8 Conclusions

```c
int main(int argc, char **argv) {
  return 0;
}
```

Listing 1: Hello

8.0.1 H3 - b

Blah blah blah...

8.0.2 H3 - c

Blah blah blah...

Mahgoub *et al.* (2017)

Brown, K. J., Sujeeth, A. K., Lee, H. J., Rompf, T., Chafi, H., Odersky, M. and Olukotun, K. (2011) ‘A heterogeneous parallel framework for domain-specific languages’, in *Parallel architectures and compilation techniques (pact), 2011 international conference on*. IEEE, pp. 89–100.

Chafi, H., DeVito, Z., Moors, A., Rompf, T., Sujeeth, A. K., Hanrahan, P., Odersky, M. and Olukotun, K. (2010) ‘Language virtualization for heterogeneous parallel computing’, in *ACM sigplan notices*. ACM (10), pp. 835–847.

Essertel, G. M., Tahboub, R. Y., Decker, J. M., Brown, K. J., Olukotun, K. and Rompf, T. (2017) ‘Flare: Native compilation for heterogeneous workloads in apache spark’, *arXiv preprint arXiv:1703.08219*.

Mahgoub, A., Ganesh, S., Meyer, F., Grama, A. and Chaterji, S. (2017) ‘Suitability of nosql systems—cassandra and scylladb—for iot workloads’, in *Communication systems and networks (comsnets), 2017 9th international conference on*. IEEE, pp. 476–479.