Proceeding Series of the Brazilian Society of Applied and Computational Mathematics, Vol. 5, N. 1, 2017.

Trabalho apresentado no CNMAC, Gramado - RS, 2016.

Proceeding Series of the Brazilian Society of Computational and Applied Mathematics

Transactional Boosting in a Purely Functional Language

Jonathas A. O. Conceição, André R. Du Bois and Renata Reiser[1]
Universidade Federal de Pelotas, UFPel, Pelotas, RS

1 Introduction

Functional programming is a different paradigm than imperative or object oriented programming. A programmer writes programs by defining functions and their composition. Functions are first class values, so functions can receive functions as arguments and return functions as results. The programs is basically an expression that must be evaluated by the language runtime to find the result of a computation. Haskell is a pure functional, statically typed and lazy programming language based on lambda calculus.

A great problem of parallel programming is the increasing complexity of programs. This complexity comes mainly from the use of Locks which are necessary tools, although hard to use. Software Transactional Memory(STM) is an abstraction of parallel programming that aims at simplifying the coding of programs. Therefore the concurrency control is handled by the virtual machine, this way the coding is easier and problems such as DeadLocks are completely avoided.

A library for STM was implemented to the Glasgow Haskell Compiler(GHC) in 2006 [4], since then research in transactional memories has been increased as well expanded to other languages [3] [2]. Among these research there's transactional boosting, that has shown great results when compared to ordinary functions of the STM Haskell libraries [1]. Our job here is to implement a specific transactional boosting function for STM on GHC.

2 GHC and Transactional Boosting

The GHC is an open source compiler and interpreter for the functional language Haskell. It runs in many environments such as Windows, Mac, Linux, most of the Unix based ones and several different processor architectures. The GHC follows some compiling steps, a Front End from the Haskell code to a Core code where Lexer, Parser, Renamer, Typecheck and Desugar are handled. A Middle from Core to Core where a optimization is made. And then a Backend, from Core to Assembly. When a code compiled by GHC is executed, it runs along with the RunTime System(RTS). The RTS is a system that gives support for multiple low-level aspects for the running program, such as garbage collection, transaction support, exceptions, scheduling, concurrency control, among other things.

[1]{jadoliveira, dubois, reiser}@inf.ufpel.edu.br

Transactional Memory works by recording data alterations made to the memory and then comparing to registered data to detect conflicts, if no conflict is found then a Commit is made, making the data of this memory segment public. If any conflict is found then an Abort is made to undo any change to the memory. A conflict occurs when two or more read/write actions are done to the same memory address. However, the methods used to find conflicts may, in some cases, find some false conflicts, which will result in performance loss.

To avoid these false conflicts, techniques such as Transactional Boosting can be applied to transform highly concurrent linearizable objects into highly concurrent transactional objects [1]. In such cases the object is treated as a black box and all conflicts are handled as soon as they are found.

This paper proposes the implementation of a polymorphic mechanism for Transactional Boosting in the Haskell language. The function to be boosted using this new primitive, only needs to have an inverse. The boost primitive creates a new version of the function warping it in an STM action providing ways for it to be called inside a transaction as well to be aborted if needed.

3 Current State and Future Steps

Currently we have an understanding of how the RunTime System of STM Haskell, which is implemented in C, works. In the Front End we know how the transactional memory functions work, on the Backend we understand how functions in Haskell make the call to the Runtime System. To continue the project now we have to add our own modifications to the RTS for transactional boosting so the primitive will be able to be called them in the purely functional world of Haskell.

References

[1] André Rauber Du Bois, Maurício Lima Pilla, and Rodrigo Duarte. Transactional boosting for Haskell. In FernandoMagno Quintão Pereira, editor, *Programming Languages*, volume 8771 of *Lecture Notes in Computer Science*, pages 145 to 159. Springer International Publishing(2014).

[2] Rafael Bandeira, André R. Du Bois, Maurício Pilla, Juliana Vizzotto, and Marcelo Machado. Composable memory transactions for java using a monadic intermediate language. In Alberto Pardo and S. Doaitse Swierstra, editors, *Programming Languages*, volume 9325 of *Lecture Notes in Computer Science*, pages 128 to 142. Springer International Publishing(2015).

[3] Sandro Rigo, Paulo Centoducatte, and Alexandro Baldassin. Memórias transacionais uma nova alternativa para programação concorrente. *Laboratório de Sistemas de Computação, Instituto de Computação, Unicamp*(2007).

[4] Tim Harris, Simon Marlow, Simon Peyton Jones, and Maurice Herlihy. Composable memory transactions. Commun. ACM, 51(8):91 to 100 (August, 2008).