Final Project Proposal: A framework for writing parallel and (doubtfully) distributed graph algorithms

Jared Roesch and Tristan Konolige

Introduction

Both being Haskell programmers has given us an appreciation for the balance between abstraction and high performance. Often times these two goals are at odds, for example the goal of MPI for example is to not give up performance for the sake of abstraction. We are greedy and we want both. One approach is to build a language that only allows one to express operations that are inherently performant, parallel, distributed ect. This is the appraoch taken by MapReduce who only provides the primitives of map which emits a (K, V) pair and then allows one to reduce all values that map to a specific key. This programming style has allowed multiple different efforts to build higher level programming abstractions but they are restricted the core language, and all operations are then gaurented to be distributed.

The current state of the art in large scale parallel computing is depressing as it is designed around the abstraction capibilities of C which are known to be poor. We really wanted abstraction and expressivity without giving up performance and so we decided on Haskell as it is the perfect solution for expressing these goals. Haskell has been made performant in vareity of situations from string operations (recent paper from intel), SDN controllers, ect.

One area of active research is the automatic parallelization of a language like Haskell which has the great properties of laziness and purity which enables reasoning not easily done about other languages. Haskell is filled with great approaches to automatically parallel code, from accelerate a DSL for GPU programming, repa for parallel arrays, data parallel haskell for SIMD style programming, the par monad for speculative paralleism, lightweight system threads for low cpu high througput concurrency/parallelism, and bound OS threads. As well Haskell provides a mature FFI for interfacing with C allowing one to write critical components as needed as well as provide pointers and other low level machinery.

We want to extend and explore the space of graph algorithms similar to Combinatorial BLAS but exploit some of the type level reasoning and fusion properties Haskell provides. We see our basis as Repa a library for parallel arrays. Repa provides a vareity of tools for dealing with Arrays with varyin representations and allow for GHC to optimize them heavily. This provides us with a battle-tested framework for dealing with parallelism and arrays, but there are still limitations. The biggest of which is that Repa provides no support for nested parallelism and no way to distribute across multiple nodes. We propose a novel extension on top of repa jokingly named doubtful distributed arrays which will provide primitives for dealing with distributed arrays. We initially visualize our distributed arrays as a type of repa repreprsentation which has special properties. We assume all machines to be running identical code similar to the approach taken by OpenMPI and other frameworks. We them implement some combinators over these distributed arrays that allow for one to run node local parallel code on the slice of the array avalible to the machine. Repa provides functionality for “manifesting” arrays which will result in the entire distributed array being pulled into memory at the node.