Under consideration for publication in J. Functional Programming

Arrows for Parallel Computations

MARTIN BRAUN
University Bayreuth, 95440 Bayreuth, Germany
[illegible]NDER
Glasgow University, Glasg[illegible]12 8QQ, Scotland
[illegible]d OLEG LO[illegible]EV
University [illegible]euth, 9544[illegible]euth, Germany

Abstract

Arrows form**OL: are?** a general interface for computation and pose therefore as an alternative to monads for API design. We express parallelism using this concept. This is a new way to represent parallel computation. We define an Arrows-based interface for parallelism and implement it using multiple parallel Haskells. **OL: Benefits:** In this manner we are able to bridge across various parallel Haskells with a common interface.

This new way of writing parallel program[illegible] benefit of being portable across flavours of parallel Haskells.**OL: Wdh?** Each parallel comput[illegible] an arrow, they can be composed and transformed as such. We thus introduce some syntactic sugar to provide parallelism-aware arrow combinators.

We also define several parallel skeletons with our framework. Benchmarks shows that our framework does not induce too much overhead performance-wise.

Contents

Contents

1 Introduction

OL: todo, reuse 5.5, "Impact" at the end and more

blablabla arrows, parallel, haskell.

Contribution HIT HERE REALLY STRONG

Structure The remaining text is structures as follows. Section **??** briefly introduces known parallel Haskell flavours and gives an overview of Arrows to the reader (Sec. **??**). Section **??** discusses related work. Section **??** defines Parallel Arrows and presents a basic interface. Section **??** defines Futures for Parallel Arrows, this concept enables better communication. Section **??** presents some basic algorithmic skeletons (parallel `map` with and without load balancing, `map-reduce`) in our newly defined dialect. More advanced ones are showcased in Section **??** (`pipe`, `ring`, `torus`). Section **??** shows the benchmark results. Section **??** discusses future work and concludes.

2 Background

2.1 Shor[illegible]on to parallel Haskells

There are already several w[illegible]s to write pa[illegible]rograms in Haskell. As we will base our parallel arrows on existing [illegible]llel Haskells[illegible]ill now give a short introduction to the ones we use as backends in [illegible] paper.

In its purest form, parallel computation ([illegible]ctions) can be looked at as the execution of some functions `a -> b` in parallel, as a[illegible]igure **??** symbolically shows:

$$parEvalN :: [a \to b] \to [a] \to [b]$$

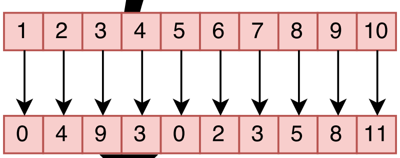

Figure 1: Schematic illustration of `parEvalN`.

Before we go into detail on how we can use this idea of parallelism for parallel Arrows, as a short introduction to parallelism in Haskell we will now implement `parEvalN` with several different parallel Haskells.

2.1.1 Multicore Haskell

Multicore Haskell (**??**) is way to do parallel processing found in standard GHC.[1] It ships with parallel evaluation strategies (**??**) for several types which can be applied with `using :: a -> Strategy a -> a`. For `parEvalN` this means that we can just apply the list of functions `[a -> b]` to the list of inputs `[a]` by zipping them with the application operator `$`. We then evaluate this lazy list `[b]` according to a `Strategy [b]` with the `using :: a -> Strategy a -> a` operator. We construct this strategy with `parList :: Strategy a -> Strategy [a]` and `rdeepseq :: NFData a => Strategy a` where the latter is a strategy which evalutes to normal form. To ensure that programs that use `parEvalN` have the correct evaluation order, we annotate the computation with `pseq :: a -> b -> b` which forces the compiler to not reorder multiple `parEvalN` computations. This is particularly necessary in circular communication topologies like in the `torus` or `ring` skeleton that we will see in chapter **??** which resulted in deadlock scenarios when executed without `pseq` during testing for this paper.

[1] Multicore Haskell on Hackage is available under `https://hackage.haskell.org/package/parallel-3.2.1.0`, compiler support is integrated in the stock GHC.