Master's Thesis Proposal

Simon van Hus
6147879
s.vanhus@students.uu.nl

December 4, 2022

1 Introduction

The aim of this research project is to built an implementation of reverse-mode automatic differentiation that – through program transformation – preserves parallelism structures present in the source program in the differentiated program. The goal is to implement this using a dual-numbers approach, and on top of Accelerate, a parallelized array language extension for Haskell [4].

Automatic differentiation (AD) is an approach for computers to differentiate programs as if they were mathematical functions. Differentiation is useful for application all throughout science, and being able to do it on an efficient basis therefor becomes very important. Current approaches can provide program transformations to create differentiated programs in polynomial time compared to the size of the input program. However, in these approaches using program transformation, complex structures in the program – like parallelism – are generally discarded. This is a shame, because many parallelized operations have parallelizable equivalents in the differentiated form. So discarding these structures means that the differentiated program can be much slower than the primal program, while it really doesn't have to be.

In the rest of this proposal, I'll go over the literature about this topic (in Section 2), the research questions and goals (in Section 3), and finally the planning going forward (in Section 4).

2 Literature

2.1 Forward and Reverse Mode

Automatic differentiation differentiates a program as a function $f : \mathbb{R}^n \to \mathbb{R}^m/$. In general, there's two major techniques for doing so [3, 2]. We have forward

mode, which differentiates the program recursively using the chain rule. This means generating the derivatives in the same order of operations as the primal function, which is both intuitive and easy to implement.

Reverse mode, like the name suggests evaluates the program in reverse of the execution order. Rather than fixing the independent variables in the function, like in forward mode, reverse mode fixes the dependent variables and computes the so-called "adjoint" tangents which are achieved by performing a reverse pass over the computation tree.

The most important distinction between these two modes is what part of the Jacobian matrix they produce. As forward mode evaluates all dependent variables for a single dependent variable, effectively calculating a column in the Jacobian. This then also means that reverse mode – evaluating all dependent variables based on a single fixed independent variable – calculates a row of the Jacobian matrix of the function. This is important for efficient computation. Both forward-mode and reverse-mode can be calculated in polynomial time based on the size of the input program, however the dimensions of the Jacobian (or the number of independent and dependent variables) dictates which is more efficient. Namely, the derivative of a program or function $f : \mathbb{R}^n \to \mathbb{R}^m$ can be calculated more efficiently using forward mode AD when the function has (a lot) more outputs than it has inputs – so $m \gg n$, as only n sweeps will be necessary to calculate the full Jacobian. Conversely, reverse mode is more efficient when the number of inputs far outweigh the number of outputs, so $m \ll n$, requiring only m sweeps.

Reverse mode AD is especially current due to many applications in Artificial Intelligence using backpropagation, a special case of reverse mode AD [1, 5]. This is appropriate because many of these systems have many more inputs than outputs.

2.2 Dual-Numbers

2.3 Parallelism

3 Research Questions and Goals

4 Planning

References

[1] Baydin, A. G., Pearlmutter, B. A., Radul, A. A., and Siskind, J. M. Automatic differentiation in machine learning: a survey. *Journal of Machine Learning Research 18*, 153 (2018), 1–43.

[2] Elliott, C. The simple essence of automatic differentiation. *Proc. ACM Program. Lang. 2*, ICFP (jul 2018).

[3] Margossian, C. C. A review of automatic differentiation and its efficient implementation. *Wiley interdisciplinary reviews: data mining and knowledge discovery 9*, 4 (2019), e1305.

[4] Marlow, S., et al. Haskell 2010 language report. *https://www.haskell.org/definition/haskell2010.pdf* (2010).

[5] Wang, F., Zheng, D., Decker, J., Wu, X., Essertel, G. M., and Rompf, T. Demystifying differentiable programming: Shift/reset the penultimate backpropagator. *Proceedings of the ACM on Programming Languages 3*, ICFP (2019), 1–31.