Automatic Differentiation of Parallelised Convolutional Neural Networks - Lessons from Adjoint PDE Solvers

Jan Hückelheim
Department of Earth Science and Engineering
Imperial College London
London, UK
j.hueckelheim@imperial.ac.uk

Paul Hovland
Mathematics and Computer Science
Argonne National Laboratory
Lemont, IL
hovland@mcs.anl.gov

Abstract

Convolutional Neural Networks often consist of several layers whose data access patterns closely resemble that of structured-mesh solvers for partial differential equation (PDE) solvers. Past work by us and others showed that the reverse-mode automatic differentiation of such solvers is challenging if the forward model uses shared-memory parallelisation. This is because communication between threads happens implicitly through shared memory locations, and is not always detectable at compile-time. This work presents an overview of past results on the relationship between data access in the forward and reverse computations, and methods to transform shared-memory-parallel forward models into equally scalable reverse models, with a case study on multi-core CPUs and Intel XeonPhi processors.

1 Introduction

Automatic Differentiation (AD) in reverse-mode has been successfully used to compute adjoints in applications as diverse as engineering, trading, weather forecasting, or climate science. These applications have in common a need for computational efficiency and scalability on parallel computing systems. With the stall in achievable processor clock speeds in the last decade, many-core shared-memory parallelism and vectorisation have become increasingly important.

Shared-memory parallelism is posing unique challenges for AD tools. Unlike parallelism using message passing (MPI), which has a long history of support in the AD community [4] and has good AD tool support, shared-memory parallelism (e.g. using OpenMP) is less well supported. One reason is that data dependencies between threads are not explicit in the source code of a program. For example, programs may contain parallel loops where each iteration reads from, and writes to, a shared array at some computed index. Depending on whether these indices overlap between iterations, this loop can be either trivial or impossible to parallelise, or may require a certain partial ordering of iterations. Furthermore, a parallel forward model does not imply that the reverse model can be run in parallel, and an AD tool must therefore fully understand the data dependencies.

In this work, we demonstrate the close relationship between convolution layers found in neural networks, and stencil computations in structured-mesh partial differential equation solvers in Section 2. We then illustrate in Section 3 how back-propagation or reverse-differentiation change the data flow, and the effect of this on parallelisation. Section 4 summarises results on the reverse-differentiation of shared-memory parallel code, and presents a small case study, before we conclude with a discussion in Section 5.

31st Conference on Neural Information Processing Systems (NIPS 2017), Long Beach, CA, USA.

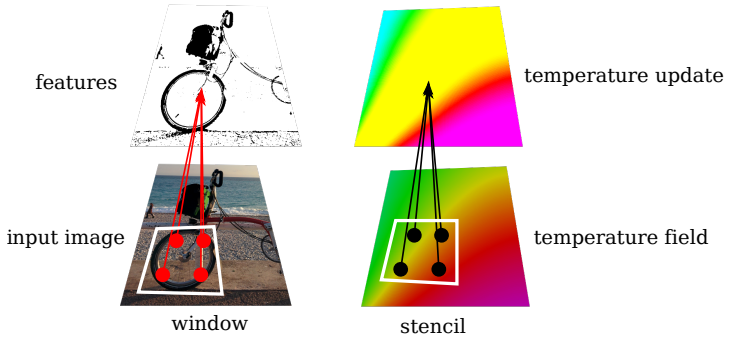

Figure 1: Analogy of PDE stencil updates and convolutional layers.

2 Convolution vs. PDE solving

Partial Differential Equations (PDE) are a model for many problems in science and industry. A simple example is the heat equation, which describes for a given point in space and time the relationship between spatial temperature variations, and the speed at which heat flows to change the temperature over time. A PDE solver applied to this problem is a forward model of heat conduction that can be used e.g. to determine the performance of a processor heat sink. A reverse model of this can be used e.g. to automatically find optimal heat sink shapes [2].

The forward model can be implemented by discretising the domain, i.e. by choosing a resolution in space and time and a representation of the state at each of these discrete points. Following this, a standard technique is to advance the solution in time by computing the new value at some spatial location as a function (often a weighted sum) of the old values within some area surrounding that location. The structure of this computation closely resembles that of a layer in a Convolutional Neural Network (CNN) with unit stride. This similarity is illustrated in Figure 1.

The reverse model can be implemented in CNNs or PDE solvers by using automatic differentiation. CNN models are trained (i.e. there weights are adjusted) using back-propagation. Although most applications of PDE solvers use reverse-differentiation to determine the sensitivity to program inputs (e.g. the shape of the heat sink), there are cases where reverse-differentiation is used to determine the finite difference weights and in a sense "train" the model, e.g. [9].

3 Data Flow Reversal: Pulling vs. Pushing

Back-propagation or reverse-differentiation leads to *data flow reversal* [7]. This concept is known in the CNN literature, for example as *pulling* and *pushing* [8]. A forward model that is implemented by pulling all the required data from surrounding indices to compute one new value naturally leads to a reverse model that pushes data to all indices that influenced the forward value. This has consequences for the parallelisation. As an example, consider the equation

$$y_1 = w_1 \cdot x_1 + w_2 \cdot x_2 \tag{1}$$
$$y_2 = w_3 \cdot x_1 + w_4 \cdot x_2, \tag{2}$$

which could be implemented in a program such that (1) and (2) are computed concurrently in a parallel loop. Each iteration would compute the final result of its own index in y, and would *pull* the required data from various locations in x.

Applying AD to such a forward loop would result in a reverse loop where the first iteration computes

$$\bar{x}_1 += w_1 \cdot \bar{y}_1$$
$$\bar{x}_2 += w_2 \cdot \bar{y}_1,$$

and the second iteration computes

$$\bar{x}_1 += w_1 \cdot \bar{y}_2$$
$$\bar{x}_2 += w_2 \cdot \bar{y}_2.$$

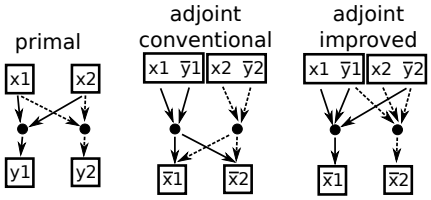

Figure 2: Data dependencies in a stencil computation forward model (left), conventional reverse model derived by AD (middle), and ideal reverse stencil model (right). The solid lines and dotted lines show the application of two stencils / windows, and the aim is to perform both in parallel. Variables with a bar on top represent derivative variables in the reverse model.

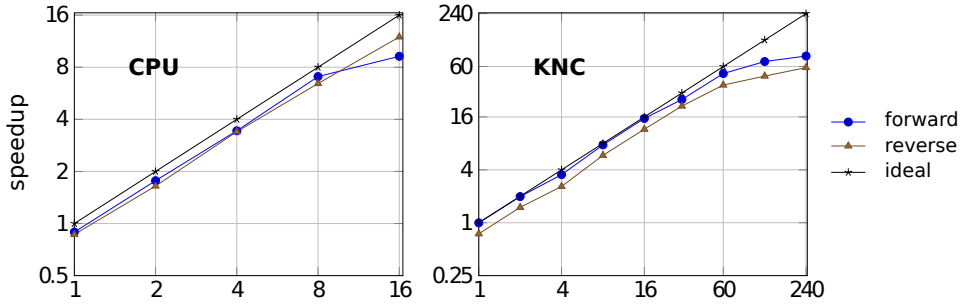

Figure 3: Scalability of an OpenMP-parallel forward stencil, and an equally parallelised reverse stencil created with transposed forward-mode automatic differentiation.

The reverse loop is thus not trivially parallel, since both iterations write to both indices in $\bar{x}$. However, it is possible to re-organise this code by placing all statements that affect a given index in the output vector into one iteration, resulting in a parallel reverse computation. In this re-organised loop, the first iteration computes

$$\bar{x}_1 += w_1 \cdot \bar{y}_1 + w_1 \cdot \bar{y}_2,$$

while the second iteration can simultaneously update

$$\bar{x}_2 += w_2 \cdot \bar{y}_1 + w_2 \cdot \bar{y}_2.$$

This transformation is the idea behind *transposed forward-mode automatic differentiation*, see Figure 2 for an illustration. However, not every code can be reorganised in this fashion, and only a subset of those can easily be handled by an AD tool.

4 Shared-memory parallelisation and AD

The reorganised, parallel reverse model in Section 3 is an example for a stencil computation that can be reorganised such that the reverse model is also a stencil computation, as formalised in [5]. In other words, a forward model that is implemented by pulling data can yield a reverse model that also pulls data, and this can in some cases be automated in an AD tool. The prerequisite for this is that the forward stencil is *reflexive*, that is, if some index i in the output is influenced by another index j in the input, then the index j in the output must also be influenced by index i in the input.

Applying this method to a stencil stencil solver running on two multi-core CPUs and a Intel XeonPhi (KNC) resulted in a scalability that matched that of the forward model. This is shown in Figure 3. The rest of this section gives an overview of other past results regarding the application of AD to shared-memory parallel codes using OpenMP.

Exclusive Read Property. A comprehensive study on the automatic differentiation of pragma-based parallelisation [1] shows that a parallel forward model only leads to a trivially parallel reverse model if the forward model satisfies the *exclusive read property*. In essence, this means that the set of memory locations that the threads read from do not overlap between synchronisation barriers. For instance, a non-overlapping pooling layer or a convolutional

layer with a stride greater than the window size satisfy this property, and naturally lead to a reverse model that can be executed in parallel.

Symmetric Updates. Some PDE solvers implement neither pulling nor pushing. For example, a convolution with window size 3 could be implemented as a loop where each iteration reads from and writes to the same two indices in the input and output arrays. This leads to forward and reverse models that can be parallelised using the same strategy [6].

Critical sections: The reverse model could be parallelised by an AD tool by placing all updates to shared variables into a critical section, or by using atomic updates. This conservative step guarantees a correct result, but only leads to reasonable performance if the computation is expensive and there are few updates to shared variables.

Reduction pragmas: The reverse model could also be parallelised using an OpenMP reduction pragma for shared variables [3]. However, this is often implemented in OpenMP runtimes by creating a private copy on each thread, thus multiplying the memory footprint with the number of threads.

5 Discussion

Parallelism can be expressed in many languages, such as OpenMP, CUDA, OpenCL, or vector intrinsics. This is a challenge for AD tools, especially since the data flow of reverse and forward models can be radically different. Languages and hardware are evolving fast, which is not only a challenge, but in the case of e.g. collision-avoiding vector instructions in AVX-512 also an opportunity to generate more efficient parallel reverse models. This makes it important for AD tool developers to focus their efforts on the things that are most useful to potential users.

It is often beneficial to exploit the structure of the problem instead of brute-force applying AD tools. This leads to the question how to exploit high-level knowledge of the problem structure, while still using AD. This requires identification and formalisation of program properties that are helpful to AD tools. A close collaboration between users and developers of AD tools is essential for this.

References

[1] M. Förster, *Algorithmic differentiation of pragma-defined parallel regions: Differentiating computer programs containing OpenMP*, Ph.D. diss., RWTH Aachen, 2014.

[2] A. Gersborg-Hansen, M.P. Bendsøe, and O. Sigmund, *Topology optimization of heat conduction problems using the finite volume method*, Structural and Multidisciplinary Optimization 31 (2006), pp. 251–259, Available at `https://doi.org/10.1007/s00158-005-0584-3`.

[3] R. Giering, T. Kaminski, and T. Slawig, *Generating efficient derivative code with taf: adjoint and tangent linear euler flow around an airfoil*, Future generation computer systems 21 (2005), pp. 1345–1355.

[4] P.D. Hovland, *Automatic differentiation of parallel programs*, Ph.D. diss., University of Illinois at Urbana-Champaign, 1997.

[5] J. Hückelheim, P. Hovland, M. Strout, and J.D. Müller, *Parallelisable adjoint stencil computations using transposed forward-mode algorithmic differentiation*, 2017. in review.

[6] J. Hückelheim, P.D. Hovland, M.M. Strout, and J.D. Müller, *Reverse-mode algorithmic differentiation of an OpenMP-parallel compressible flow solver*, International Journal for High Performance Computing Applications (2017).

[7] U. Naumann, *The art of differentiating computer programs: an introduction to algorithmic differentiation*, Vol. 24, SIAM, 2012.

[8] P.Y. Simard, D. Steinkraus, J.C. Platt, *et al.*, *Best practices for convolutional neural networks applied to visual document analysis.*, in .

[9] G. Yao, D. Wu, and H.A. Debens, *Adaptive finite difference for seismic wavefield modelling in acoustic media* 6 (2016), pp. 30302 EP –, Available at `http://dx.doi.org/10.1038/srep30302`.