# Relaxed Differentiation and Differentiation of Relaxations: Implementation Considerations for the GPU

Neil Kichler*
RWTH Aachen University
Aachen, Germany
kichler@stce.rwth-aachen.de

Uwe Naumann
RWTH Aachen University
Aachen, Germany
naumann@stce.rwth-aachen.de

## Abstract

Derivative information is inherently local. Extending the information to a range of values to capture non-local behavior requires relaxations of Algorithmic Differentation (AD). Interval Arithmetic can be used to bound the derivatives in a given input range, giving what could be called *relaxed* AD. McCormick relaxations can further tighten those bounds by providing convex and concave under- and overapproximations, respectively. Use cases include significance analysis [11], pruning of neural networks [6], and various forms of optimization [3, 4, 10]. On the flip side, (sub)gradients of relaxations provide necessary linearizations of McCormick relaxations to be used in, e.g., deterministic global optimization. Motivated by larger-scale analysis and optimization, our aim is to make use of modern GPUs. Besides enabling massive parallelism, GPUs provide essential rounded intrinsic operations (e.g., `[f,d]add_r[u,n,z,d]`) without the additional cost occurred in most CPU architectures for repeatedly (re-)setting the global rounding mode. However, they also pose new challenges for an efficient implementation, including data-layout considerations, efficient seeding, shared memory use, etc.

We will consider Interval Arithmetic and McCormick relaxations in combination with Algorithmic Differentiation on GPUs, and report on recent developments. In particular, custom CUDA kernels are developed that, in combination with C++ operator-overloading, allow for arbitrary combinations of the above. In combination with CUDA graphs, the developed libraries can be used in the deterministic global optimizer MAiNGO. The corresponding open-source implementations CuInterval, CuMcCormick, and CuTangent will be shown, including examples in relevant contexts. We finally show initial comparisons to existing C++ based tools.

## CCS Concepts

• **Computing methodologies** → **Model verification and validation**; *Massively parallel and high-performance simulations*; *Uncertainty quantification*; • **Software and its engineering** → **Massively parallel systems**; • **Mathematics of computing** → **Mathematical software performance**; **Automatic differentiation**; **Interval arithmetic**; Nonconvex optimization.

## Keywords

Algorithmic Differentiation, McCormick Relaxations, GPGPU

**ACM Reference Format:**
Neil Kichler and Uwe Naumann. 2024. Relaxed Differentiation and Differentiation of Relaxations: Implementation Considerations for the GPU. In *Proceedings of Workshop on Differentiable Parallel Programming (DiffProg-PPoPP '25).* ACM, New York, NY, USA, 3 pages. https://doi.org/XXXXXXX.XXXXXXX


*DiffProg-PPoPP '25, March 01–02, 2025, Las Vegas, NV*

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM
https://doi.org/XXXXXXX.XXXXXXX

## 1 Introduction

Given a function $F : \mathbb{R}^n \to \mathbb{R}^m$ represented by a computer program, the aim in Algorithmic Differentiation (AD) is to reinterpret the execution in forward/tangent or reverse/adjoint mode to furthermore evaluate (higher-order) derivatives. A similar pattern emerges when considering the relaxed evaluation of the function $F$ in Interval Arithmetic. Now, the input is an interval $X = [a, b] = \{x \in \mathbb{R} \mid a \leq x \leq b\}$. By the fundamental theorem of Interval Arithmetic an interval function evaluation on $X$ will always bound all possible function evaluations on inputs $x \in X$, providing proper lower and upper bounds for the given interval input [9]. The newly developed C++/CUDA library CuInterval supports and allows overloading of all fundamental and set-based interval operations of the 2015 IEEE Standard for Interval Arithmetic [5].

In McCormick relaxations [7], the primitive functions require in addition a convex and concave relaxation (ideally its envelope) that with the McCormick composition rule establishes convex and concave relaxations for the entire function. Let $X \subseteq \mathbb{R}^n, Z \subseteq \mathbb{R}$ be nonempty convex sets. For a composite function $g = F \circ f$, where $f : X \to Z$ and $F : Z \to \mathbb{R}$, with known convex relaxations $f^{cv} : X \to \mathbb{R}$, $F^{cv} : Z \to \mathbb{R}$, and concave relaxations $f^{cc} : X \to \mathbb{R}$, $F^{cc} : Z \to \mathbb{R}$, the convex and concave relaxations of $g$ can be computed by

$$g^{cv}(\boldsymbol{x}) = F^{cv}(\mathtt{mid}(f^{cv}(\boldsymbol{x}), f^{cc}(\boldsymbol{x}), z_{\min})), \tag{1}$$

and

$$g^{cc}(\boldsymbol{x}) = F^{cc}(\mathtt{mid}(f^{c}(\boldsymbol{x}), f^{cc}(\boldsymbol{x}), z_{\max})), \tag{2}$$

respectively. Here, the `mid` function returns the middle value of the three scalar arguments and

$$z_{\min} = \arg\min_{z \in Z} F^{cv}(z),$$

$$z_{\max} = \arg\min_{z \in Z} F^{cc}(z).$$

Note that the McCormick composition therefore might result in nonsmooth functions such that in general subgradients are required. This is taken care of in AD by extending derivatives of the `min`, `max`, and `mid` functions in the desired way as in [2, 8]. Most commonly used primitive functions are implemented in the newly developed C++/CUDA library CuMcCormick.

However, efficient combinations of the above with AD are often not considered, especially once the target architecture is a highly-parallel machine like a GPU. The efficient implementation of vector-mode AD in conjunction with relaxations is of particular interest. Motivating examples are plentiful and will guide implementation efforts.

In this work, we consider combinations of the above with a focus on the combined execution on GPUs. In particular, combinations of AD and McCormick relaxations as well as Interval Arithmetic will be developed in C++ via operator overloading and are used as a testbed for performance evaluations. Furthermore, in dynamic environments where the computational graph is built up at runtime, the use of CUDA graphs is employed and analyzed.

## 2 Preliminary Results

Initial results based on Tangents-of-McCormick are provided in the below figures. Figure 1 shows the execution of the arbitrarily chosen function in Equation 3 being written in a single kernel.

$$F(\boldsymbol{x}) = -1 + \sum_{i=0}^{n} \cos(x_i - 1)^2 + (x_i - 1)^3 + (x_i - 1)^4 \qquad (3)$$

As in most other domains, the GPU requires a large enough workload to overcome the initial overhead of CUDA context initialization, memory transfers, and slower single-core performance. The benchmarks were performed on a single node of the CLAIX-2023-ML cluster [1] using 1 upto 96 cores of the two Intel Xeon 8468 Sapphire (2.1 GHz, 48 cores each) CPUs and one H100 for the GPU scenario.

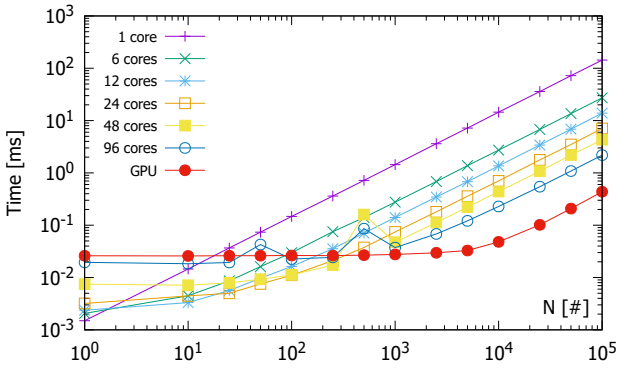


**Figure 1: Tangent of McCormick evaluation of function with 1 variable, on 1 node @CLAIX-2023-ML [1]. CPU versions make use of OpenMP and MC++ while the GPU version uses a single CUDA kernel.**

In Figure 2, the Tangents-of-McCormick computation of the same function is performed on a computational graph, using the MC++ graph and evaluation on the CPU and comparing it against the new CUDA graph implementation using CuMcCormick and CuTangent. While the MC++ graph evaluation might not be the most efficient possible CPU implementation, Figure 2 undoubtedly shows a clear opportunity for large-scale evaluations of relaxations

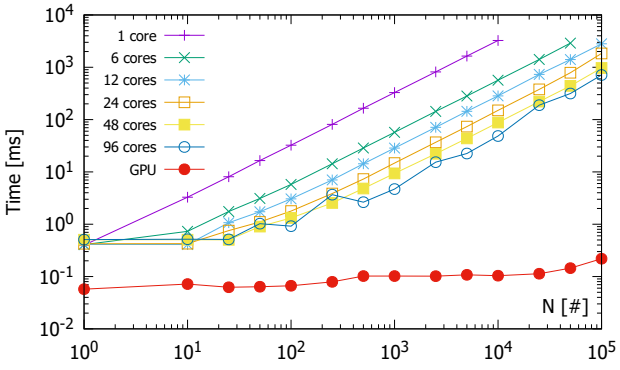


**Figure 2: McCormick evaluation with CUDA graphs of function with 1 variable, on 1 node @CLAIX-2023-ML [1]. CPU versions make use of OpenMP and MC++ graphs, while the GPU version uses a CUDA graph.**

on the GPU with potentially orders of magnitude improved performance. The graph evaluation includes the initial creation time of the computational graph. Larger graph overheads are amortized by repeated usage - as in deterministic global optimization - but also point to the need for lower-overhead alternatives in latency sensitive scenarios.

## 3 Outlook

Ultimately, next-generation scientific software frameworks should be open to extension and allow reinterpretation of existing code in new ways unknown to the original authors. This work highlights some of the recent developments and challenges one encounters when trying to combine the flexibility of operator-overloading and efficiency of CUDA graphs. It should also hopefully give insights into possible integration of the mentioned relaxations in AD-aware JIT-compilers and other compiler infrastructure.

## References

[1] RWTH Aachen. 2023. *CLAIX 2023 hardware overview.* https://help.itc.rwth-aachen.de/service/rhr4fjjutttf/article/fbd107191cf14c4b8307f44f545cf68a/

[2] Markus Beckers, Viktor Mosenkis, and Uwe Naumann. 2012. Adjoint Mode Computation of Subgradients for McCormick Relaxations. In *Recent Advances in Algorithmic Differentiation*, Shaun Forth, Paul Hovland, Eric Phipps, Jean Utke, and Andrea Walther (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 103–113. https://doi.org/10.1007/978-3-642-30023-3_10

[3] Dominik Bongartz, Jaromił Najman, and Alexander Mitsos. 2020. Deterministic global optimization of steam cycles using the IAPWS-IF97 model. *Optimization & Engineering* 21 (2020), 1095–1131. https://doi.org/10.1007/s11081-020-09502-1

[4] Jens Deussen and Uwe Naumann. 2023. Subdomain separability in global optimization. *Journal of Global Optimization* 86, 3 (2023), 573–588. https://doi.org/10.1007/s10898-022-01265-6

[5] IEEE. 2015. IEEE Standard for Interval Arithmetic. *IEEE Std 1788-2015* (2015), 1–97. https://doi.org/10.1109/IEEESTD.2015.7140721

[6] Neil Kichler, Sher Afghan, and Uwe Naumann. 2024. Towards Sobolev Pruning. In *Proceedings of the Platform for Advanced Scientific Computing Conference* (Zurich, Switzerland) *(PASC '24)*. Association for Computing Machinery, New York, NY, USA, Article 1, 11 pages. https://doi.org/10.1145/3659914.3659915

[7] Garth P McCormick. 1976. Computability of global solutions to factorable non-convex programs: Part I—Convex underestimating problems. *Mathematical programming* 10, 1 (1976), 147–175. https://doi.org/10.1007/BF01580665

[8] Alexander Mitsos, Benoit Chachuat, and Paul I Barton. 2009. McCormick-based relaxations of algorithms. *SIAM Journal on Optimization* 20, 2 (2009), 573–601. https://doi.org/10.1137/080717341

[9] Ramon E. Moore. 1979. *Methods and Applications of Interval Analysis*. Society for Industrial and Applied Mathematics (SIAM). https://doi.org/10.1137/1.9781611970906

[10] Hermann Schichl and Arnold Neumaier. 2005. Interval analysis on directed acyclic graphs for global optimization. *Journal of Global Optimization* 33 (2005), 541–562. https://doi.org/10.1007/s10898-005-0937-x

[11] Vassilis Vassiliadis, Jan Riehme, Jens Deussen, Konstantinos Parasyris, Christos D. Antonopoulos, Nikolaos Bellas, Spyros Lalis, and Uwe Naumann. 2016. Towards automatic significance analysis for approximate computing. In *Proceedings of the 2016 International Symposium on Code Generation and Optimization* (Barcelona, Spain) *(CGO '16)*. Association for Computing Machinery, New York, NY, USA, 182–193. https://doi.org/10.1145/2854038.2854058