Report from Dagstuhl Seminar 22101

Tensor Computations: Applications and Optimization

Paolo Bientinesi[*1], **David Ham**[*2], **Furong Huang**[*3], **Paul H. J. Kelly**[*4], **P. (Saday) Sadayappan**[*5], **and Edward Stow**[†6]

1 **University of Umeå, SE.** pauldj@cs.umu.se
2 **Imperial College London, GB.** david.ham@imperial.ac.uk
3 **University of Maryland – College Park, US.** furongh@cs.umd.edu
4 **Imperial College London, GB.** p.kelly@imperial.ac.uk
5 **University of Utah – Salt Lake City, US.** psaday@gmail.com
6 **Imperial College London, GB.** edward.stow16@imperial.ac.uk

Abstract

This report documents the program and the outcomes of Dagstuhl Seminar 22101 "Tensor Computations: Applications and Optimization". Tensors are higher dimensional analogs of matrices, and represent a key data abstraction for many applications in computational science and data science. Widely used shared infrastructure exists for linear algebra, while, in contrast, for tensor computations, there is no consensus on standard building blocks. This Dagstuhl Seminar aimed to bring together users, and performance optimization specialists, to build such foundations.

We present the abstracts of the 5 tutorials and 14 talks given. The working groups and their outcomes so far are then presented.

Seminar 06.–11. March, 2022 – http://www.dagstuhl.de/22101
2012 ACM Subject Classification General and reference → General literature; Computing methodologies → Linear algebra algorithms; Applied computing → Physical sciences and engineering; Mathematics of computing → Mathematical software performance; Mathematics of computing → Computations on matrices
Keywords and phrases Tensor, Optimisation, Linear Algebra, Compilers, Benchmark, Domain Specific Language
Digital Object Identifier 10.4230/DagRep.12.3.1

1 Executive Summary

Paolo Bientinesi (University of Umeå, SE, pauldj@cs.umu.se)
David Ham (Imperial College London, GB, david.ham@imperial.ac.uk)
Furong Huang (University of Maryland, College Park, US, furongh@umd.edu)
Paul H. J. Kelly (Imperial College London, GB, p.kelly@imperial.ac.uk)
P. Sadayappan (University of Utah, Salt Lake City, US, saday@cs.utah.edu)

Linear relationships between quantities are one of the most fundamental and pervasive phenomena in mathematics, science and computing. While matrices encode linear relationships between exactly two quantities, tensors are an abstraction representing linear relationships between multiple variables. Tensor computations therefore provide an abstract language for computations that span an enormous range of application domains, including machine learning, quantum information systems, simulations based on solving partial differential equations,

[*] Editor / Organizer
[†] Editorial Assistant / Collector

computational chemistry and beyond. The tensor abstraction enriches our understanding of the structure of computations, and exposes common challenges and solutions that cut across different research communities.

While the mathematics of tensors is well-developed and extensively applied across all of these applications and beyond, there is far less commonality in the software abstractions and tools deployed to execute tensor computations. This is in stark contrast to matrix computations, where common abstractions and stable interfaces have led to widely used tools that bring high performance to across diverse application domains.

This Seminar explored this challenge, and made significant progress towards establishing foundations for common implementations – embodying the substantial body of knowledge on high-performance tensor computation strategies in common software libraries and domain-specific program generation tools.

The Seminar began with five tutorial lectures, offered by the organisers in partnership with selected leading figures in some of the relevant communities. We began by mapping some of the diverse terminology. We then provided tutorials exposing the quantitative and qualitative diversity in how different communities use tensor computations – aiming to build a common understanding of key concepts, notations, and building blocks. We focused on the following application areas:

- Quantum physics and chemistry
- Mesh-based discretisations for solution of partial differential equations
- Machine learning.

The final tutorial reviewed the challenge of establishing unifying software tools, highlighting the enormous body of work that has been done within application areas.

The second phase of the Seminar consisted of more detailed presentations from the participants. These included motivating applications, but focusing on the fundamental computational workloads, methods, and performance challenges. Building on this, we also had contributions focused on implementation – low-level performance considerations, algorithmic proposals, compiler algorithms and compiler infrastructure.

In the third phase of the Seminar, we separated into three teams. One explored benchmarking and datasets, another made substantial progress on proof-of-concept implementation work to connecting the high-level Tensorly library for tensor decompositions in machine learning to a lower-level tensor-vector products – achieving considerable performance advantage. Finally there was also a major and continuing effort to define a common domain-specific language and compiler representation for tensor contractions that supports both high-level optimisations and the use of high-performance low-level libraries.

This 2021 seminar built on progress made at an earlier seminar with the same title, in March 2020 – which was very heavily impacted by the coronavirus pandemic. This seminar was also affected, to a lesser extent – with a reduced number of on-site participants, partly compensated by very useful engagement with researchers joining online, albeit from distant timezones.

This seminar benefited from broader engagement with application domains – partly as a result of the work that was done on tutorials – which we hope to publish in due course. It also benefited from deeper engagement with developers of high-performance building blocks. Finally, we initiated a new and continuing effort to define a common language and a common intermediate language for code generation tools.

2 Table of Contents

3 Overview of Tutorials

3.1 What is a tensor? What might a tensor abstraction look like?

David Ham (Imperial College London, GB, david.ham@imperial.ac.uk)

The 2020 Dagstuhl Seminar[1] spent a long time developing a common understanding of what tensors are. It also resulted in some ideas about multiple levels of abstraction for tensor computations. This presentation introduces these ideas and considers how different communities' expectations about tensors map onto them.

References

1 Paolo Bientinesi, David Ham, Furong Huang, Paul H. J. Kelly, Christian Lengauer, and Saday Sadayappan. Tensor computations: Applications and optimization (dagstuhl seminar 20111), 2020.

3.2 Computing with tensors in mesh-based PDE discretisations

Lawrence Mitchell (NVIDIA Corporation, Santa Clara, US, lmitchell@nvidia.com)

At the core of residual assembly in finite element computations is a combination of tensor contractions and 'pointwise' non-linear operations. For efficient implementation, one often wishes to exploit structure in the tensors. I showed some structure that we use in the TSFC compiler [1, 2] to do this, and discussed some places where we might imagine using more generic technology.

References

1 Miklós Homolya, Robert C. Kirby, and David A. Ham. Exposing and exploiting structure: optimal code generation for high-order finite element methods, 2017.
2 Miklós Homolya, Lawrence Mitchell, Fabio Luporini, and David A. Ham. TSFC: a structure-preserving form compiler. *SIAM Journal on Scientific Computing*, 40(3):C401–C428, 2018.

3.3 Tensors in Machine Learning

Jeremy Cohen (CREATIS, CNRS, Villeurbanne, FR, jeremy.cohen@cnrs.fr)
Furong Huang (University of Maryland, College Park, US, furongh@umd.edu)

Machine Learning is relying more and more on tensor computations. Tensors may represent extremely diverse data stemming from fluorescence spectroscopy, remote sensing, music information retrieval, image and video processing, but also parameters in high order statistics and deep learning. In this tutorial, after introducing several widely used tensor notations and low-rank approximation models, we detail how tensors and tensor models are used in several of these applications. In particular, we review the use of tensors in recommendation systems, blind source separation, dictionary learning, compression of neural networks (notably transformers) and classification. We finish this tutorial by pointing out a few important required tensor computations building blocks in the machine learning community.

3.4 Software for tensor computations: What is happening?

Paolo Bientinesi (University of Umeå, SE, pauldj@cs.umu.se)

Main reference Christos Psarras, Lars Karlsson, Paolo Bientinesi: "The landscape of software for tensor computations", CoRR, Vol. abs/2103.13756, 2021.
URL https://arxiv.org/abs/2103.13756

In stark contrast to the world of matrix computations, that of tensor computations still lacks a well defined set of building blocks. From an ongoing survey of the existing software for tensor computations, it emerged that many similar libraries are developed independently in different application domains. This inevitably leads to redundant effort and sub-optimal results (in terms of efficiency). In this talk we present and discuss possible building blocks to support high-performance tensor operations across different application domains.

4 Overview of Talks

4.1 Infrastructure for Tensor Compilers: Lessons and Ongoing Developments from the MLIR Project

Albert Cohen (Google – Paris, FR, albertcohen@google.com)

Peeking into MLIR for code generation and domain-specific compilation, with a focus on tensor algebra. Covering both graph-level and loop/vector-level optimization.

4.2 Tensorly toolbox, and the tensoptly project

Jeremy Cohen (CREATIS, CNRS, Villeurbanne, FR, jeremy.cohen@cnrs.fr)

The Tensorly toolbox is a high level library that provides tensor manipulations and decompositions in python[1]. After presenting the high level API and some recent additions from the Tensoptly project, we explore how bottleneck operations such as MTTKRP are implemented.

References

1 Jean Kossaifi, Yannis Panagakis, Anima Anandkumar, and Maja Pantic. Tensorly: Tensor learning in python. *Journal of Machine Learning Research (JMLR)*, 20(26), 2019.

4.3 Overview of the Boost.uBlas Tensor Extension

Cem Bassoy (Fraunhofer IOSB, DE, cem.bassoy@iosb.fraunhofer.de)

Boost.uBLAS has been recently extended with dense tensor types and basic tensor operations. In this talk interfaces and implementations of the extension are presented. Runtime results of tensor-time-vector and tensor-times-matrix operations are presented and discussed as well.

4.4 Sequences of tensor contractions: A design space exploration

Carsten Uphoff (Intel Deutschland GmbH, Feldkirchen, DE, carsten.uphoff@intel.com)

Kernels in finite element methods can be abstracted as sequences of tensor contractions. The number of implementation variants typically grows exponentially with the number of tensors. Sources of exponential growth are in particular the order of operations and the data structures of intermediate tensors. I discuss the design space of sequences of tensor contractions and algorithms to automatically select a fast implementation variant.

4.5 Hashing for the zeros/nonzeros of a sparse tensor

Bora Ucar (CNRS and ENS Lyon, FR, bora.ucar@ens-lyon.fr)

Main reference Jules Bertrand, Fanny Dufossé, Somesh Singh, Bora Uçar: "Algorithms and data structures for hyperedge queries", p. 28, 2022.
URL https://hal.inria.fr/hal-03127673

We consider the problem of querying the existence of hyperedges in hypergraphs. More formally, we are given a hypergraph, and we need to answer queries of the form "does the following set of vertices form a hyperedge in the given hypergraph?". Our aim is to set up data structures based on hashing to answer these queries as fast as possible. We discuss an adaptation of a well-known perfect hashing approach for the problem at hand. This is joint work with Jules Bertrand, Fanny Dufossé, and Somesh Singh and available as a technical report. There is also an efficient shared-memory parallel implementation [1].

References

1 Somesh Singh and Bora Uçar. An Efficient Parallel Implementation of a Perfect Hashing Method for Hypergraphs. In *GrAPL 2022 – Workshop on Graphs, Architectures, Programming, and Learning*, pages 1–10, Lyon, France, May 2022. IEEE. to appear.

4.6 Portable and efficient array redistribution

Norman A. Rink (DeepMind, London, GB, nrink@deepmind.com)

Main reference Norman A. Rink, Adam Paszke, Dimitrios Vytiniotis, Georg Stefan Schmid: "Memory-efficient array redistribution through portable collective communication", CoRR, Vol. abs/2112.01075, 2021.
URL https://arxiv.org/abs/2112.01075

Computing on partitioned arrays in a distributed fashion has become commonplace, especially in the context of large-scale machine learning, where the size of large models necessitates working on partitioned arrays in order to fit into device memory. Computing on partitioned arrays typically requires communication in the form of redistributing chunks of arrays, which can easily become a performance bottleneck. I present a type-directed approach that synthesizes efficient communication sequences for array redistribution.

4.7 How will End-of-Moore impact high-performance tensor-centric applications?

P. Sadayappan (University of Utah, Salt Lake City, US, saday@cs.utah.edu)

The high-performance computing world has seen two "sea changes" so far. The first was the "attack of the killer micros" – change from vector supercomputers to highly parallel clusters of commodity processors. The next disruptive change was the emergence of GPUs with roughly an order-of-magnitude performance edge over CPUs for tensor computations. While tensor-centric applications in ML have largely adapted to this change, other domains are yet to benefit from the power of GPUs. It appears that the next disruptive change is looming with the end of Moore's Law scaling of VLSI. The latest breed of accelerators for ML all appear to be fully distributed-memory systems with thousands of processors on a chip without any shared-memory. The challenges in developing efficient tensor applications for these systems is even more daunting than GPUs. Compilers will need to play a significant role moving forward.

4.8 Successes and Challenges for continuous matrix product states

Jutho Haegerman (University of Ghent, BE, Jutho.Haegeman@UGent.be)

Main reference Benoît Tuybens, Jacopo De Nardis, Jutho Haegeman, Frank Verstraete: "Variational Optimization of Continuous Matrix Product States", Phys. Rev. Lett., Vol. 128, p. 020501, American Physical Society, 2022.
URL https://doi.org/10.1103/PhysRevLett.128.020501

A particular limit of the tensor train / matrix product state construction gives rise to a class of quantum states known as continuous matrix product states. The energy minimisation or dynamical evolution thereof gives rise to coupled set of non-linear matrix-valued partial differential equations. I discuss our successes in dealing with them so far, and the remaining challenges that we face for further applications.

4.9 Simulation of the Sycamore quantum circuits with tensor networks

Pan Zhang (Chinese Academy of Science, Beijing, CN, panzhang@itp.ac.cn)

The sampling problem of the Sycamore circuits has been used by Google to demonstrate the quantum advantage. I will introduce tensor network methods based on the sparse-state representation to generate one million uncorrelated samples from the final state of the Sycamore circuits with fidelity greater than Google's experiments.

4.10 Implicit Regularization in Deep Learning: Lessons Learned from Tensor Factorizations

Nadav Cohen (Tel Aviv University, IL, cohennadav@cs.tau.ac.il)

Main reference Noam Razin, Asaf Maman, Nadav Cohen: "Implicit Regularization in Tensor Factorization", in Proc. of the 38th International Conference on Machine Learning, Proceedings of Machine Learning Research, Vol. 139, pp. 8913–8924, PMLR, 2021.
URL https://proceedings.mlr.press/v139/razin21a.html

The mysterious ability of deep neural networks to generalize is believed to stem from an implicit regularization, a tendency of gradient-based optimization to fit training data with predictors of low "complexity" [1, 2]. A major challenge in formalizing this intuition is that we lack measures of complexity that are both quantitative and capture the essence of data which admits generalization (images, audio, text, etc.). With an eye towards this challenge, I will present recent analyses of implicit regularization in tensor factorizations, equivalent to certain non-linear neural networks. Through dynamical characterizations, I will establish implicit regularization towards low rank, different from any type of norm minimization, in contrast to prior beliefs. Then, motivated by tensor rank capturing implicit regularization of non-linear neural networks, I will suggest it as a measure of complexity, and show that it stays extremely low when fitting standard datasets. This gives rise to the possibility of tensor rank explaining both implicit regularization of neural networks, and the properties of real-world data translating it to generalization.

References

1 Sanjeev Arora, Nadav Cohen, Wei Hu, and Yuping Luo. Implicit regularization in deep matrix factorization. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

2 Noam Razin and Nadav Cohen. Implicit regularization in deep learning may not be explainable by norms. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, NIPS'20, Red Hook, NY, USA, 2020. Curran Associates Inc.

4.11 The Tensor Brain: Semantic Decoding for Perception and Memory

Volker Tresp (Ludwig-Maximilians-Universität München & Siemens, München, DE, volker.tresp@siemens.com)

I start with an introduction to PyKEEN [1], the main talk is on the tensor brain. It is a unified computational theory of an agent's perception and memory. In our model [2], perception, episodic and semantic memory are refined by different functional and operational modes of the oscillating interaction between index layer and a representation layer (global workspace) in a bi-layer tensor network (BTN).

References

1 Mehdi Ali, Max Berrendorf, Charles Tapley Hoyt, Laurent Vermue, Sahand Sharifzadeh, Volker Tresp, and Jens Lehmann. Pykeen 1.0: a python library for training and evaluating knowledge graph embeddings. *Journal of Machine Learning Research*, 22(82):1–6, 2021.

2 Volker Tresp, Sahand Sharifzadeh, and Dario Konopatzki. A model for perception and memory. In *Conference on Cognitive Computational Neuroscience*, 2019.

4.12 Matricized Tensor Times Khatri-Rao Product (MTTKRP)

Christos Psarras (RWTH Aachen, DE, psarras@aices.rwth-aachen.de)

Given the apparent interest at the start of the week on the MTTKRP operation, especially among the machine learning audiences, I decided to present the challengers and my insights on providing a generic black-box implementation of MTTKRP for 3D dense tensors and beyond. I presented a graph which listed the different ways of computing MTTKRP for 3D tensors, without explicitly transposing the underlying tensor, and emphasized on their (often significant) differences in performance. My findings pointed to the difficulty of creating a mechanism that can accurately predict which method for computing MTTKRP is most efficient depending on the target platform (CPU or GPU), the sizes of the dimensions of the underlying tensor, and the number of components (columns) of the factor matrices.

4.13 Domain-Extensible Compilers and Controllable Automation of Optimizations

Thomas Koehler (University of Glasgow, GB, t.koehler.1@research.gla.ac.uk)

I presented my work on a rewrite-based domain-extensible compiler with controllable automation of optimisations [1, 2, 3]. To encourage discussion, I presented two opinions on how future tensor compilers should be designed. "Will you agree, or have a different opinion?"

References

1 Bastian Hagedorn, Johannes Lenfers, Thomas Koehler, Xueying Qin, Sergei Gorlatch, and Michel Steuwer. Achieving high-performance the functional way: A functional pearl on expressing high-performance optimizations as rewrite strategies. *Proc. ACM Program. Lang.*, 4(ICFP), aug 2020.

2 Thomas Koehler and Michel Steuwer. Towards a domain-extensible compiler: Optimizing an image processing pipeline on mobile cpus. In *2021 IEEE/ACM International Symposium on Code Generation and Optimization (CGO)*, pages 27–38, 2021.

3 Thomas Koehler, Phil Trinder, and Michel Steuwer. Sketch-guided equality saturation: Scaling equality saturation to complex optimizations in languages with bindings, 2021.

5 Working Groups

5.1 TTV in Tensorly

Jeremy Cohen (CREATIS, CNRS, Villeurbanne, FR, jeremy.cohen@cnrs.fr)
Cem Bassoy (Fraunhofer IOSB, Ettlingen, DE)
Lawrence Mitchell (NVIDIA Corporation, Santa Clara, US, lmitchell@nvidia.com)

Tensorly is a python library that provides high-level API for tensor decompositions [3]. It contains highly-efficient implementations of state-of-the-art algorithms for tensor decompositions. One key operation for the tensor decomposition is the so-called TTV (tensor-times-vector) operation. Tensorly's TTV implementation unfolds tensors in order to use efficiently implemented matrix-vector operations with unfolded tensors. The unfolding operation however consumes more memory than a naive implementation and introduces additional runtime overhead compared to high-performance tensor multiplication algorithms.

The goal of this breakout group was to investigate the usability of an high-performance open-source C++ implementation of TTV [1] and to estimate potential performance gains for Tensorly. One major benefit of Cem's C++ library is it's ability to provide fast tensor-vector multiplication for tensors that are stored according to last-order storage format which is (if not otherwise specified) the common format used in NumPy. We were able to integrate Cem's TTV library into Tensorly without modifying the original C++ code. We observed considerable speedups for certain tensor shapes. With promising results, we intend to continue our collaboration and to continuously report our latest findings in [2].

References

1 Cem Bassoy. Design of a high-performance tensor-vector multiplication with blas. In João M. F. Rodrigues, Pedro J. S. Cardoso, Jânio Monteiro, Roberto Lam, Valeria V. Krzhizhanovskaya, Michael H. Lees, Jack J. Dongarra, and Peter M.A. Sloot, editors, *Computational Science – ICCS 2019*, pages 32–45, Cham, 2019. Springer International Publishing.

2 `https://github.com/cohenjer/tensorly/tree/tensordot_and_ttv/ttv_and_tensordot`.

3 Jean Kossaifi, Yannis Panagakis, Anima Anandkumar, and Maja Pantic. Tensorly: Tensor learning in python. *Journal of Machine Learning Research (JMLR)*, 20(26), 2019.

5.2 A unified domain specific language for tensor contractions: Dagstuhl Tensor Language

Albert Cohen (Google – Paris, FR, albertcohen@google.com)
Teodoro F. Collin (MIT CSAIL, US, teoc@mit.edu)
David Ham (Imperial College London, GB, david.ham@imperial.ac.uk)
Paul H. J. Kelly (Imperial College London, GB, p.kelly@imperial.ac.uk)
Thomas Koehler (University of Glasgow, GB, t.koehler.1@research.gla.ac.uk)
Norman A. Rink (DeepMind, London, GB, nrink@deepmind.com)
Edward Stow (Imperial College London, GB, edward.stow16@imperial.ac.uk)
Carsten Uphoff (Intel Deutschland GmbH, Feldkirchen, DE, carsten.uphoff@intel.com)
Sophia Vorderwuelbecke (Imperial College London, GB, s.vorderwuelbecke18@imperial.ac.uk)
Connor Ward (Imperial College London, GB, c.ward20@imperial.ac.uk)

Dagstuhl Tensor Language (DTL) is the continuation of work done in the previous Dagstuhl Seminar, the aim being to bring together a common language at a highly abstract level that sufficiently describes the mathematics of tensor computations. The language is designed to be minimal and general in that it makes no attempt to specialise towards any hardware or algorithmic feature beyond tensor contractions; but with a view that is must become extensible and through directives or annotations a policy for decision making and optimisation for specific use cases could be produced.

5.2.1 Design

Our primary goal in the design of DTL was to build a minimal and unambiguous tensor language that targeted a minimal set of optimizations that could be done with just formulas for the sizes of the tensors. Strength reduction was the primary motivator. We were able to eventually sketch out a design by using an insight from several other languages for tensor computations: the need for an explicit unbind operator (an operator that builds a tensor by iterating over a given set of indexes) to avoid ambiguities in index notation. Several other projects, targeted at different problems (e.g Chiw's EIN for Diderot, GEM from Firedrake, Dex for ML, ATL for optimization problems) have postulated and to various extents used such an operation. We have incorporated it as a primary and fundamental feature for our minimal set of operations.

5.2.2 Open Problems

There are disagreements on the precise way we would like others to use this language. For some, we would hope this to basically be an IR and to be used in other people's stack after a point of lowering. For others, we want this to be a base language that is easily extended to other languages (i.e to be functorial in the sense of standard ML modules). In either case, we did not figure out how this language could easily modified or extended by users towards other means. In particular, although the base language works, we left several design issues unsolved that could be rephrased as areas where we need to enable extensibility via some means:

- Meta data associated to tensors and how it could be use by general strength reduction algorithms or passed down a compiler stack or used to call special math operations

- Special constant tensors with special algebraic relations (e.g. $\epsilon_{ijk\ldots}$ and δ_{ij}) or even languages for constant tensors built in some other expression language
- Arithmetic operations on indices, affine, modular, or maybe non-linear
- Special operations on tensor indices to transform them so that operations such as Ax recreate convolution. For example, circular indexing and the topelitz transform employed in ML to do convolution with GEMM
- Non-linear mathematical operations on tensors as opposed to scalars (e.g inverse of a matrix)
  - Even if there should be linear operations on tensors
- Support for non-rectangular iteration spaces.
- Support for non-affine or even sparse iteration spaces.
- General support for iteration spaces
- Use of index expressions to make constant tensors (the algebraic use of indexes, allowing `i` to be promoted to a tensor $[0, 1, 2, 3, ..., n]$)
- Support for algebraic sparsity via conditionals in constant tensors and/or via more complex indexing operations
- Support for sequence spaces or semi-rings or other basic algebraic operations

We could probably generate a longer list, but the critical point is that the basic objects of the language (tensors, constants, operations, indexes, iteration spaces?) could be plausible extended in a variety of potentially overlapping ways (e.g. two of these extensions enable convolution support) or at least users might want to pass information about this stuff through a compilation pipeline that uses DTL (e.g. tagging where a tensor or operation comes from). We did not resolve these issues, but we recognize them and hope to solve them.

5.2.3 Future Work

One potential avenue we want to investigate is building this system within MLIR and seeing to what extent MLIR helps us. But even there, we must return to a central set of optimizations that we want to share, and we must also do the algorithmic work of how those optimizations could be extended if we allow the language to be extended along some of these axises. This was one of the core arguments against enabling too much extensionality as it makes the core rationale less reasonable.

5.2.4 Ongoing Work

Since the seminar, work has continued in rounding out a python-embedded implementation of the language with a project architecture that focuses on keeping the language minimal while allowing different back-ends to produce implementations or eagerly compute results. We have produced a work-in-progress reference implementation that uses python with Numpy for un-optimised but semantically correct outputs. In parallel, we have begun mapping the python-embedded implementation into an xDSL dialect with the intention to make use of existing work to enable lower levels of optimisations through xDSL's own linear algebra dialects and also further mapping into the MLIR compiler infrastructure.

Participants

- Cem Bassoy
Fraunhofer IOSB – Ettlingen, DE
- Paolo Bientinesi
University of Umeå, SE
- Simon Bonér
University of Umeå, SE
- Albert Choen
Google – Paris, FR
- Jeremy Cohen
CNRS – IRISA – Rennes, FR
- Teodoro F. Collin
MIT – Cambridge, US
- Jutho Haegerman
Ghent University, BE
- David Ham
Imperial College London, GB
- Paul H. J. Kelly
Imperial College London, GB
- Thomas Koehler
University of Glasgow, GB
- Lawrance Mitchell
NVIDIA – Santa Clara, US
- Christos Psarras
RWTH Aachen, DE
- Norman A. Rink
Google DeepMind – London, GB
- P. Sadayappan
University of Utah, US
- Paul Springer
Nvidia Corp. – Santa Clara, US
- Edward Stow
Imperial College London, GB
- Volker Tresp
LMU München & Siemens, Müchen, DE
- Bora Ucar
CNRS and ENS Lyon, FR
- Carsten Uphoff
Intel Deutschland GmbH – Feldkirchen, DE
- Edward Valeev
Virginia Tech – Blacksburg, US
- Sophia Vorderwuelbecke
Imperial College London, GB
- Connor Ward
Imperial College London, GB

Remote Participants

- Charisee Chiw
Google – San Francisco, US
- Nadav Cohen
Tel Aviv University, IL
- Edoardo Di Napoli
Jülich Supercomputing Centre, DE
- Rong Ge
Duke University – Durham, US
- Johnnie Gray
Caltech – Pasadena, US
- Vinod Grover
NVIDIA – Redmond, US
- Furong Huang
University of Maryland – College Park, US
- Katharina Kormann
Uppsala University, SE
- Jean Kossaifi
NVIDIA – Redmond, US
- Jiajia Li
Pacific Northwest National Lab. – Richland, US
- Devin Matthews
SMU – Dallas, US
- Luke Panayi
Imperial College London, GB
- Vivek Srikumar
University of Utah – Salt Lake City, US
- Edwin Miles Stoudenmire
Flatiron Institute – New York, US
- Richard M. Veras
University of Oklahoma – Norman, US
- Qi (Rose) Yu
University of California – San Diego, US
- Pan Zhang
Chinese Academy of Science – Beijing, CN