FAIR Sharing of Data in Autotuning Research (Vision Paper)

Jana Hozzová
Institute of Computer Science,
Masaryk University
Brno, Czech Republic
hozzova@mail.muni.cz

Jacob O. Tørring
Department of Computer Science,
Norwegian University of Science and Technology
Trondheim, Norway
jacob.torring@ntnu.no

Ben van Werkhoven
Leiden Institute of Advanced Computer Science, Leiden University
Leiden, The Netherlands
b.van.werkhoven@liacs.leidenuniv.nl

David Střelák
Institute of Computer Science,
Masaryk University
Brno, Czech Republic
National Biotechnology Center
Madrid, Spain
373911@mail.muni.cz

Richard Vuduc
Georgia Institute of Technology
Atlanta, USA
richie@cc.gatech.edu

ABSTRACT

Autotuning is an automated process that selects the best computer program implementation from a set of candidates to improve performance, such as execution time, when run under new circumstances, such as new hardware. The process of autotuning generates a large amount of performance data with multiple potential use cases, including reproducing results, comparing included methods, and understanding the impact of individual tuning parameters.

We propose the adoption of FAIR Principles, which stands for Findable, Accessible, Interoperable, and Reusable, to organize the guidelines for data sharing in autotuning research. The guidelines aim to lessen the burden of sharing data and provide a comprehensive checklist of recommendations for shared data. We illustrate three examples that could greatly benefit from shared autotuning data to advance the research without time- and resource-demanding data collection.

To facilitate data sharing, we have taken a community-driven approach to define a common format for the data using a JSON schema and provide scripts for their collection.

The proposed comprehensive guide for collecting and sharing performance data in autotuning research can promote further advances in the field and encourage research collaboration.

CCS CONCEPTS

• **General and reference** → *Measurement*; **Performance**; • **Software and its engineering** → *Collaboration in software development*.

KEYWORDS

autotuning; benchmarks; performance; measurements; open data; data sharing

ICPE '24 Companion, May 7–11, 2024, London, United Kingdom
© 2024 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-0445-1/24/05.
https://doi.org/10.1145/3629527.3651429

ACM Reference Format:
Jana Hozzová, Jacob O. Tørring, Ben van Werkhoven, David Střelák, and Richard Vuduc. 2024. FAIR Sharing of Data in Autotuning Research (Vision Paper). In *Companion of the 15th ACM/SPEC International Conference on Performance Engineering (ICPE '24 Companion), May 7–11, 2024, London, United Kingdom.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3629527.3651429

1 INTRODUCTION

Autotuning is an automated process, guided by experiments, that selects one from among a set of candidate computer program implementations to improve its execution time, energy, or another performance characteristic [2]. An *autotuner* refers to a system that implements this process and may be viewed as having two main parts: a *code generator* that can produce a space of possible implementations; and a *search mechanism* that explores this space, using models or automated benchmarking experiments, to find one that performs well. During this process, an autotuner may generate and collect a considerable amount of performance data.

These data have value for other researchers, but making them easily reusable is a complex task. As a focal point for thinking through data-sharing issues, we selected a research problem that arises in autotuning research: how to compare search methods fairly. For instance, there may be differences in how the samples were measured and the environment in which data were collected. In benchmarking search methods, it is essential to know if a data set covers the full configuration space or only some subset (as with a grid search, for instance). As another example, it might be useful to know whether or not so-called invalid configurations are included in the data set and how these can be recognized.

The field of autotuning research would benefit from a thorough discussion about how and what to collect and share so that other researchers might be able to find, access, integrate and reuse performance data. At the time of this writing, several autotuning systems are under active development, and data sharing is considered a prerequisite for better understanding which methods work best, under what scenarios, and why.

In this paper, we outline principles, guidelines, and mechanisms to reflect the questions about performance data sharing from the view of developers, practitioners, and users of autotuning systems.

It was initiated during a meeting of autotuning researchers in March 2022[1].

Moreover, in collaboration with other researchers, we devised JSON schema to define a common format for sharing the data and a script to facilitate their collection. We suggest best practices for sharing autotuning data specifically while avoiding being overly prescriptive. We derive a comprehensive checklist of requirements for shared data using the following analysis: see Checklist for FAIR Sharing of Data in Autotuning Research or repository[2] for downloadable version. We consider these guidelines, the schema, the script and the checklist to be main contributions of this paper. Together, they provide a comprehensive view on sharing data in autotuning, offer a standard format of data and help with the collection.

The value of these data lies in multiple potential use cases. These can be categorized as:

Reproduction and Verification:

- Reproducing results of other researchers.
- Comparing search methods [17].
- Storing "wisdom" to cache best-results in specific tuning contexts [5].

Search Method Modifications and Code Generation:

- Using data to drive code generation, *a la* profile-guided optimizations in compilers.
- Using data to optimize the tuning budget [11].
- Constructing models to guide tuning, either by an expert or by a machine learning method [6].

Analysis and Insights:

- Running and developing statistical data analysis, machine learning, and visualization methods "tuned" to autotuning.
- Understanding the impact of individual tuning parameters and their interactions.
- Developing insights into the behavior of compilers and computer architectures.
- Studying the sensitivity of programs to their inputs and characteristics.

Curation and Benchmarking:

- Curating of entire tuning spaces used as "ground truth".
- Curating performance with respect to various metrics, including time, energy, storage, and accuracy.
- Making leaderboards or scoreboard reporting for specific classes of problems.

The rest of the paper is organized as follows. First, we propose data sharing guidelines in the context of autotuning (see Sharing autotuning data following FAIR principles). We adopted the doctrine of Findable, Accessible, Interoperable, and Reusable digital assets, also known as the FAIR Principles, [3] to organize these guidelines [16]. Second, we outline several use cases that show how shared performance data can stimulate research in autotuning (see Use Case Examples). Third, we summarize surveyed work on scientific data sharing and reproducibility in general, as well as performance data benchmarking, collection, and sharing in particular (see Related Work). Fourth, we describe the challenges of data sharing and lay out future work that would foster the discussion of these questions and the adoption of our approach (see Challenges and Future Work).

[1]Generic Autotuning Technology for GPU Applications at the Lorentz Center, March 7–13, 2022: https://www.lorentzcenter.nl/generic-autotuning-technology-for-gpu-applications.html

[2]https://github.com/odgaard/TuningSchema/tree/T4/checklist.md

[3]Developed, coincidentally, by others during a Lorentz Center workshop in 2014 https://www.lorentzcenter.nl/jointly-designing-a-data-fairport.html

2 SHARING AUTOTUNING DATA FOLLOWING FAIR PRINCIPLES

The FAIR Guiding Principles suggest that shared data will be most useful to the broader scientific community if it is findable, accessible, interoperable, and resuable [16]. We recommend how to apply these desiderata to autotuning datasets, focusing on what information should be included to enable the use cases mentioned in Introduction.

For concreteness, other researchers and we devised a JSON schema for sharing the metadata regarding the autotuning process and its results.[4] Moreover, we implemented a script that automatically collects recommended information about software and hardware involved in autotuning experiments.[5] Several autotuners under active development already export their outputs in accordance with JSON schema [12, 15]. Additionally, the BAT project [13, 14] supports exporting data from many Python-based tuners that do not natively support the format.

In this section, we define the terms we use, and then we list recommendations and guidelines by following FAIR principles.

2.1 Used nomenclature

In this proposal, we adopt the OpenML distinction between **raw datasets** and **run datasets**.[6] A raw dataset is an exhaustive search of a benchmark over the entirety of a searched configuration space. The dataset of an exhaustive search would contain the global optimum. In some cases, even a dataset of a partial exhaustive search of the configuration space might be useful to share.

A run dataset consists of data from runs of the search algorithm. Tuning configurations included in it represent the path of the search method, as it was going through the searched configuration space, looking for well-performing configuration. This dataset might not contain the global optimum.

In a common scenario, a user could download a raw dataset and evaluate different search techniques using it as a surrogate for real hardware. We refer to this as a **simulated run**. Each run dataset is the full experimental result for how this search would have performed on the configurations contained in the raw dataset. A run dataset could also hold the results from a **hardware run** that is not simulated.

A raw dataset should include information that could be used to set up the same environment and run the benchmarks, thereby reproducing the exhaustive search's results. For a simulated run dataset, authors could provide a DOI to the raw dataset and thus

[4]https://github.com/odgaard/TuningSchema/tree/T4/metadata.py

[5]https://github.com/odgaard/TuningSchema/blob/T4

[6]https://openml.org

simplify the metadata that they need to provide. If the authors provide a hardware run dataset, there would be additional requirements to ensure that other researchers can reproduce the results.

We use the term **kernel** to refer to a unit of autotuned code. A kernel may be a CUDA kernel, the critical routine of an application, or an entire application if autotuned as a whole via, for instance, tuning compiler switches.

2.2 Findable

“Findability” refers to the ability to find and filter data of interest by other researchers. We strongly recommend that all datasets have DOIs to make citation possible. We mention other approaches to make datasets more available to other researchers in section Challenges and Future Work.

2.3 Accessible

“Accessibility” refers to the ability of other researchers to obtain a copy of the dataset. We recommend using repositories like Zenodo to host autotuning datasets. Zenodo data artifacts are citable via DOI and available for open or closed submission with versioning of datasets of 50 GB (or bigger if needed). We also recommend that authors provide contact details of the person responsible for collecting the data.

2.4 Interoperable

“Interoperability” means providing the information necessary to use the data in another autotuner or computing environment. Doing so implies a common data format or an otherwise completely and precisely described format and provides information related to invalid data points.

2.4.1 Standard Data Formats. It is a good practice to format data in standard and widely recognized formats for easy interpretation and integration. For example, JSON is recommended as it is universally recognized and easy to parse.

2.4.2 Performance Measurement Recommendations. For accurate performance measurements, it is necessary to report key details that align with the metrics for search method comparisons. This includes:

- Kernel experiment time, or the individual run time of kernel configurations.
- Time spent validating the output of the kernel configuration.
- Compilation time.
- Overhead details, such as the search method overhead (inclusive of model prediction time) and the autotuner's overhead.
- The timestamp of the measurement.

2.4.3 Additional Measurements and Metrics. It is beneficial to report supplementary measurements, such as power consumption, profiling counters, and clock frequencies, whenever possible. Derived metrics like compute performance in GFLOP/s or energy efficiency in GFLOPS/W can provide deeper insights. Importantly, if there is variability in some measurements due to factors like performance counter acquisition, it is crucial to describe the reasons and mark the data accordingly.

2.4.4 Detailed Information on Tuning Parameters. Tuning parameters should be described in detail. This encompasses the name, type (like int, float, string), and values or range (valid and used). Any conditions or restrictions on these parameters, especially complex ones, should be documented. It should be explicitly stated if there is a relationship between the input problem size and tuning parameters.

2.4.5 Handling of Invalid Data Points. Including invalid data points, especially in full configuration space explorations, provides a complete picture. Such data points should be marked based on their type, whether due to not meeting conditions, failed compilation, or computational issues during runtime.

2.4.6 Validation. Details about the validation of the results, including the benchmarked kernel configurations and their correctness criteria, must be provided.

2.4.7 Miscellaneous Recommendations. Minor details include the notation for decimal points (dot or comma). We also advise noting the timestamp of the experiment's start, as the GPU's runtime duration might influence the performance.

2.5 Reusable

The concept of "Reusability" in the context of dataset management extends beyond merely providing the dataset; it involves furnishing essential and supplementary information that aids in further research utilizing the dataset. This necessitates comprehensive metadata detailing both the data collection and processing methodologies.

2.5.1 General Dataset Information. Several key elements are integral to ensuring both clarity and traceability in dataset usage. These include providing links to associated research and data papers and offering a license recommendation, such as the Open Data Commons Open Database License (ODbL) 1.0 (available at [https://opendatacommons.org/licenses/odbl/1-0/]). Additionally, it is useful to clarify any data usage restrictions and establish clear citation guidelines for the dataset.

2.5.2 Computational Problem Description. Explaining the computational problem in lay terms would aid a broader understanding, particularly for those not specialized in the field. This should be complemented by an outline of common programming patterns into which the problem might fit and a clarification regarding whether the problem is predominantly memory-bound or compute-bound.

2.5.3 Kernel Implementation Details. Detailing the specifics of the kernel's programming aspects is important. This includes information on the source code's location and version, the programming language employed, details of the compiler and its options, the kernel's grid and thread size, and specifics about the kernel's arguments.

2.5.4 Tuning Parameter Insights. An in-depth exploration of the tuning parameters and their impacts is recommended. This should cover the effects of commonly used tuning parameters, details of the dataset's configuration space, and information on the dataset's run data, including the search methods and models utilized.

2.5.5 Input Data Details. Providing information on the dataset's inputs is suggested. This encompasses detailing the input's size and other relevant characteristics and whether this input data is included within the dataset.

2.5.6 Data Collection and Processing. A comprehensive dive into data management practices is necessary. This involves outlining the data acquisition methods, autotuner details, techniques used in data processing and filtering, visualization and other scripts, execution environment details, and the software and scripts employed to ensure reproducibility.

2.5.7 Hardware Specifications. Providing in-depth details about the hardware utilized in the dataset's creation and processing is significant to ensure reproducibility. This should include information on device details and model numbers, chipsets and memory specifics, methods for measuring power consumption, and details as recommended by the Supercomputing conference environment script[7].

2.5.8 Environment and Execution Details. Detailing the ecosystem surrounding the dataset makes the data actually reusable. This includes specifics on the software used, including operating systems and compilers, details of scripts and software for dataset acquisition, and information related to compilation and execution.

This comprehensive list of recommended shared information about data may be considered intimidating. Therefore, we encourage to use the script[8] we developed in collaboration with other researchers to automate the collection of bulk of these metadata. We provide an actual checklist in PDF and MD format for easier reference in the same repository and in the appendix of this paper.

3 USE CASE EXAMPLES

Shared performance data have many potential use cases; we listed several in section Introduction. In this section, we elaborate on three of them in greater detail. With these use cases, researchers could aim to devise more effective and faster search methods by better understanding current search methods or by deeper insight into tuning spaces.

One of the main issues related to the search in autotuning is that tuning spaces are hard to search. They are discrete, non-convex, and show little or no locality, i.e. similar configurations usually have vastly different performance. It means that many traditional approaches for search in optimization space do not perform well or they do not perform better than random search [3].

Reusing the performance data might be extremely difficult or even impossible in all use cases without all the necessary information. Our recommendations on sharing autotuning-related data have been chosen with these use cases (and more) in mind.

3.1 Comparing search methods

Comparing search methods and understanding which work best and under what scenarios facilitates the development and use of effective and fast search methods. However, fair comparison presents a difficult task in itself. The simplest way (although it also has its pitfalls) is to have all the search methods in one autotuner. Then, one could search for the best configuration of the same computational problem using the same input on the same hardware and see which search method finds the solution the fastest. To provide more robustness and fairness to comparison, one would repeat it many times to deal with the stochastic nature of the search, change out the input and hardware to deal with the sensitivity and possibly even change the parameters of the search method. Even if we ignore the implementation phase that might be needed, preparing and executing all of these experiments is heavily time- and resource-consuming.

Reusing performance data shared by other researchers would save a lot of time and resources, as someone else has already done the most consuming part. Clearly, a fair comparison of search methods implemented within different autotuners poses another set of challenges. It might even be impossible if the shared dataset does not include all the information needed, e.g. kernel code to ensure that we compare the same computational problem; details about performance measurements (whole experiment time vs. separately noting kernel time and compilation time and overhead time) to ensure we can account for the overhead of the search; or execution environment (hardware and software setup details) to ensure we compare experiments run on the same or comparable hardware.

Recently, Willemsen et al. [17] published a method for comparing search methods in different autotuners fairly. Our list includes all the data needed to use their comparison metric.

3.2 Creating models to guide tuning

One of the ways to search faster is to give it domain- and problem-specific information, for example, a performance model created either by an expert or a machine learning technique. If trained properly, it can assess the shape of the tuning space of the given computational problem and guess what configuration would be the best next step.

In order to create and train the performance model, one needs performance data above the usual time or energy spent in the kernel. Performance profiling counters, also called hardware counters, provide more reusable information. However, their collection is heavily time- and resource-consuming; kernel run with profiling turned on runs much slower than usual. Thus, reusing the data collected by others would save many resources. Apart from profiling counters, detailed information about the tuning space and computing environment is required.

Data from previous runs on other hardware or input have been used to guide the tuning in [6]. Filipovič et al. gathered all profiling counters themselves and explicitly noted how demanding it was. They shared the data in [9]. Machine learning method based on statistical analysis guides the profiling also in [10].

3.3 Analyzing data to gain insight

Gaining deeper insight into tuning spaces would help researchers make more informed decisions while developing search methods or creating performance models to guide the tuning. Via statistical analysis, they can inspect the interactions between tuning parameters, look at input sensitivity and learn the intricate details

[7]https://submissions.supercomputing.org/?page=SampleForm&id=PaperArtifactDescriptionArticleEvaluationADAEAppendix&site=sc21
[8]https://github.com/odgaard/TuningSchema/blob/T4

about hardware and programming models beyond often poorly documented behaviour.

Running all the experiments "just" for statistical analysis seems wasteful and worthless. By using shared data, one can build upon the work of others and move the research further.

4 RELATED WORK

Several related initiatives have been created for or could be reasonably used towards sharing data in autotuning research. For instance, there are many machine learning and data science competition platforms, such as Kaggle[9] or OpenML[10], that allow researchers to publish or share datasets. While the data sharing principles we have proposed could be implemented using any of these platforms, our guidelines for autotuning go beyond the typical use case of these platforms by providing specific recommendations with regard to what metadata to collect and how datasets can be made FAIR.

Several computer science conferences and journals are currently trying to improve the reproducibility and replicability of results by implementing artifact description and evaluation initiatives. We have drawn inspiration from the questionnaire used for artifact evaluations in SuperComputing[11] and specialized them to cover information required to enable FAIR sharing of autotuning performance measurement data.

In terms of autotuning research specifically, a set of initiatives by Grigori Fursin under the names of cTuning[12], Collective Mind [8], Collective Knowledge (CK) [7], and work of Cho et al. [4] are probably the most closely related works. The cTuning initiative started with a similar goal: to enable or bootstrap sharing of performance data within the autotuning research community. Collective Mind aimed to crowd-source such autotuning performance data for many different computing hardware systems. It seems these systems have inspired Collective Knowledge, a system for automating research actions similar to CodaLab.[13] It appears that cTuning can only be used through the Collective Knowledge workflow. It is unclear how, where, and in what form the data or metadata is stored. The main contribution of the initiative appears to be in automating the experimentation workflow through the Collective Knowledge system.

Finally, Cho et al. have proposed a JSON database to leverage historical data for Bayesian optimization and provide CK for metadata collection [4].[14]. They also propose a website where users can upload their results.[15] However, this repository is specific to their autotuner GPTune (which is limited to integer, real, and categorical types of parameters and the Bayesian optimization process), whereas what we are proposing tries to be more general and inclusive of other autotuning frameworks. The developers of GPTune also collect additional metadata, such as software version, machine information and evaluation data. These are publicly available, with over 14 thousand evaluations spanning 23 problems. Capitalizing on these data-sharing opportunities is a central goal of our more general proposal.

5 CHALLENGES AND FUTURE WORK

Many researchers may view data sharing as a hardly feasible task - it is undoubtedly a challenge. We have addressed a few perceived barriers in this paper; several remain for future work.

The main challenges of data collection include that researchers do not know *what* to collect and *how* to collect data without too much effort. Data sharing brings additional worries: *where* to share and *what* to share so that the data can be reused. Moreover, data collection and sharing are usually not at the center of the researcher’s attention; they are just a byproduct of the ongoing research. Therefore, anything that makes data collection and sharing quicker and easier can sway the decision to make public or to keep private.

In our guidelines, we address the question about *what* data should be collected, both in terms of actual, primary data and additional descriptions of the environment and processes. All these suggestions have been made with a focus on the future reusability of data. Having a checklist gives researchers a solid starting point, ensuring they do not omit anything critically important.

The question of *how* to collect data can be divided into three parts: primary data (measurements and tuning space details), environment data (hardware information, code location) and process data (details about the computational problem, search method and data processing). Collecting the measurements and details about tuning space in our proposed JSON format is automatic in some autotuning frameworks. So far, KTT [12] and KernelTuner [15] export data in this format, and BAT project [13, 14] can export data from several Python-based tuners, most notably OpenTuner [1]. For the environment data, we offer the script that automatically collects most of them. The last part is the most time-consuming; details about methods and processes need to be written down. However, this effort pays off later, as these descriptions most probably should also appear in the research article; they are not relevant only to data sharing.

We address the question of *where* to share the data by suggesting both the repository and licence. To make autotuning data more readily findable, we foresee a website that could serve as a primary entry point. For those creating data, it should provide an easy way to submit links to their datasets and facilitate data sharing. For those seeking data, it should offer basic filtering on the metadata properties of the datasets to help look up by criteria of specific interest in autotuning, such as what hardware or performance metrics were used. For example, EOSC initiative[16] aims to provide a venue for such websites.

We realize that even with the guidelines, a checklist and scripts at our disposal, the feat of data sharing might seem unreachable. A sample dataset and a sample description of data that would abide by our guidelines ease the hurdle and make it conceivable to undertake. We have created a simple example of shared data https://zenodo.org /records/7212426. It contains only automatically generated data, no detailed descriptions of all processes. Nevertheless, it can illustrate how the shared data generated by our scripts and schemas look like.

[9]https://www.kaggle.com/
[10]https://www.openml.org/
[11]https://submissions.supercomputing.org/?page=SampleForm&id=PaperArtifactDescriptionArticleEvaluationADAEAppendix&site=sc21
[12]https://ctuning.org/
[13]https://worksheets.codalab.org/
[14]https://github.com/gptune/CK-GPTune
[15]https://gptune.lbl.gov
[16]https://eosc-portal.eu/

6 CONCLUSION

We have formulated several recommendations and guidelines to enable sharing FAIR data with the autotuning research community based on twelve foreseen use cases. Based on the FAIR principles (Findable, Accessible, Interoperable, and Reusable), we recommend specific actions the autotuning community can take, including what metadata to collect to sustain the use cases and ultimately make the autotuning data FAIR. We hope that the community's refinement and ultimate adoption of this proposal will help address data sharing issues in autotuning. Indeed, we strongly believe such data have high intrinsic value, not just within the autotuning community but also for improving our overall understanding of computer system performance.

ACKNOWLEDGMENTS

The CORTEX project has received funding from the Dutch Research Council (NWO) in the framework of the NWA-ORC Call (file number NWA.1160.18.316). The *Generic Autotuning Technology for GPU Applications* workshop has received funding from the Lorentz Center, Netherlands eScience Center, and the CORTEX consortium.

This work was supported by the Ministry of Education, Youth and Sports of the Czech Republic through the e-INFRA CZ LM2018140.

This research work was also funded by the European Commision – NextGenerationEU(Regulation 2020/2094), through CSIC´s Global Health Platform (PTI Salud Global).

REFERENCES

[1] Jason Ansel, Shoaib Kamil, Kalyan Veeramachaneni, Jonathan Ragan-Kelley, Jeffrey Bosboom, Una-May O'Reilly, and Saman Amarasinghe. 2014. Opentuner: an extensible framework for program autotuning. In *International Conference on Parallel Architectures and Compilation Techniques (PACT)*. Edmonton, Canada, (Aug. 2014). http://groups.csail.mit.edu/commit/papers/2014/ansel-pact14-opentuner.pdf.

[2] Prasanna Balaprakash, Jack Dongarra, Todd Gamblin, Mary Hall, Jeffrey K Hollingsworth, Boyana Norris, and Richard Vuduc. 2018. Autotuning in high-performance computing applications. *Proceedings of the IEEE*, 106, 11, 2068–2083.

[3] Prasanna Balaprakash, Stefan M. Wild, and Paul D. Hovland. 2011. Can search algorithms save large-scale automatic performance tuning? *Procedia Computer Science*. Proceedings of the International Conference on Computational Science, ICCS 2011 4, (Jan. 1, 2011), 2136–2145. DOI: 10.1016/j.procs.2011.04.234.

[4] Younghyun Cho, James W Demmel, Xiaoye S Li, Yang Liu, and Hengrui Luo. 2021. Enhancing autotuning capability with a history database. In *2021 IEEE 14th International Symposium on Embedded Multicore/Many-core Systems-on-Chip (MCSoC)*. IEEE, 249–257.

[5] FFTW.org. 2023. Words of wisdom - saving plans. Web page. (2023). Retrieved Feb. 8, 2023 from https://www.fftw.org/fftw3_doc/Words-of-Wisdom_002dSaving-Plans.html#Words-of-Wisdom_002dSaving-Plans.

[6] Jiří Filipovič, Jana Hozzová, Amin Nezarat, Jaroslav Ol'ha, and Filip Petrovič. 2022. Using hardware performance counters to speed up autotuning convergence on GPUs. *Journal of Parallel and Distributed Computing*, 160, (Feb. 1, 2022), 16–35. DOI: 10.1016/j.jpdc.2021.10.003.

[7] Grigori Fursin. 2021. Collective knowledge: organizing research projects as a database of reusable components and portable workflows with common interfaces. *Philosophical Transactions of the Royal Society A*, 379, 2197, 20200211.

[8] Grigori Fursin. 2013. Tutorial at hpsc 2013 at ntu, taiwan: collective mind: novel methodology, framework and repository to crowd-source auto-tuning. In *HPSC-Conference on Advanced Topics and Auto Tuning in High Performance and Scientific Computing-2013*.

[9] Jana Hozzová, Jiří Filipovič, Amin Nezarat, Jaroslav Ol'ha, and Filip Petrovič. 2021. Searching CUDA code autotuning spaces with hardware performance counters: data from benchmarks running on various GPU architectures. *Data in Brief*, 39, (Dec. 1, 2021), 107631. DOI: 10.1016/j.dib.2021.107631.

[10] Edward Hutter and Edgar Solomonik. 2021. Accelerating Distributed-Memory Autotuning via Statistical Analysis of Execution Paths. In *2021 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. 2021 IEEE International Parallel and Distributed Processing Symposium (IPDPS). (May 2021), 46–57. DOI: 10.1109/IPDPS49936.2021.00014.

[11] Jaroslav Oľha, Jana Hozzová, Matej Antol, and Jiří Filipovič. 2024. Estimating resource budgets to ensure autotuning efficiency. *Journal of Parallel and Distributed Computing*. Submitted. http://dx.doi.org/10.2139/ssrn.4661862.

[12] Filip Petrovič, David Střelák, Jana Hozzová, Jaroslav Ol'ha, Richard Trembecký, Siegfried Benkner, and Jiří Filipovič. 2020. A benchmark set of highly-efficient cuda and opencl kernels and its dynamic autotuning with kernel tuning toolkit. *Future Generation Computer Systems*, 108, 161–177.

[13] Ingunn Sund, Knut A. Kirkhorn, Jacob O. Tørring, and Anne C. Elster. 2021. BAT: a benchmark suite for AutoTuners. *Norsk IKT-konferanse for forskning og utdanning*, 1, (Nov. 14, 2021), 44–57. Number: 1. Retrieved Dec. 10, 2021 from https://ojs.bibsys.no/index.php/NIK/article/view/915.

[14] Jacob O. Tørring, Ben van Werkhoven, Filip Petrovič, Floris-Jan Willemsen, Jiří Filipovič, and Anne C. Elster. 2024. Towards a Benchmarking Suite for Kernel Tuners. *2023 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. Submitted.

[15] Ben van Werkhoven. 2019. Kernel tuner: a search-optimizing gpu code autotuner. *Future Generation Computer Systems*, 90, 347–358. DOI: https://doi.org/10.1016/j.future.2018.08.004.

[16] Mark D. Wilkinson et al. 2016. The FAIR Guiding Principles for scientific data management and stewardship. *Scientific Data*, 3, 160018, (Mar. 2016). DOI: 10.1038/sdata.2016.18.

[17] Floris-Jan Willemsen, Richard Schoonhoven, Jiří Filipovič, Jacob O. Tørring, Rob van Nieuwpoort, and Ben van Werkhoven. 2024. A methodology for comparing optimization algorithms for auto-tuning. *Future Generation Computer Systems*. Submitted.

A CHECKLIST FOR FAIR SHARING OF DATA IN AUTOTUNING RESEARCH

This appendix contains a checklist of recommended information to share. Information that gets automatically collected by using our scripts or those that are present in our proposed JSON schemas available in repository https://github.com/odgaard/TuningSchema/blob/T4/ is marked by ⊡.

- General information
  - ☐ name of the dataset for easier future reference
  - ☐ DOI and link to repository
  - ☐ contact information to authors
  - ☐ how to cite
  - ☐ licence and usage restrictions
  - ☐ link to related papers
- Measurements
  - ⊡ kernel experiment time
  - ⊡ validation time
  - ⊡ compilation time
  - ⊡ overhead details (search method overhead, autotuner overhead, model overhead)
  - ⊡ timestamp
  - ⊡ if possible, additional measurements, such as power consumption, profiling counters or clock frequencies
- Tuning space
  - ⊡ names and types of tuning parameters
  - ⊡ values or ranges of tuning parameters
  - ⊡ conditions of tuning parameters
  - ☐ details about how different types of invalid data points are handled
  - ⊡ details about how the results are validated
  - ☐ description of the effects of tuning parameters
  - ☐ details about search space, i.e. raw dataset or run dataset
- Computational problem and its implementation
  - ☐ explanation for non-experts
  - ☐ common programming patterns it fits into
  - ☐ memory- or compute-bound
  - ☐ source code location and version
  - ⊡ programming language used
  - ⊡ grid and thread size
  - ⊡ kernel argument details
- Search method and models
  - ⊡ hyperparameters of the search method
  - ⊡ budget
  - ⊡ performance metric and optimization objective function
  - ☐ details about how models were created and trained
- Environment and execution
  - Input data
    - ☐ size and other relevant characteristics
    - ☐ whether it is included within the dataset
  - Hardware
    - ⊡ details about the device and the model
    - ⊡ chipsets and memory specifics
    - ☐ details about how power consumption is measured
    - ⊡ details provided by the recommended Supercomputing conference environment script
  - Software
    - ⊡ software specifics, OS and compilers
    - ⊡ details about compilation
    - ⊡ details about execution environment
  - Data processing
    - ☐ details about how data were acquired
    - ☐ details about how the autotuner was set and executed
    - ☐ details about data processing and filtering
    - ☐ if relevant, details about analysis and visualization
    - ☐ software and scripts used for dataset acquisition, processing, analysis and visualization