Crediting pull requests to open source research software as an academic contribution☆

Hartwig Anzt [a, b, *], Eileen Kuehn [a], Goran Flegar [c]

[a] Steinbuch Centre for Computing (SCC), Karlsruhe Institute of Technology, Germany
[b] Innovative Computing Lab (ICL), University of Tennessee, Knoxville, USA
[c] Departamento de Ingeniería y Ciencia de Computadores (ICC), Universidad Jaume I, Castellón, Spain

Keywords:
Scientific excellence paradigms
Conference contributions
Scientific reputation
Community software development

A B S T R A C T

Like any other scientific discipline, the High Performance Computing community suffers under the *publish or perish* paradigm. As a result, a significant portion of novel algorithm designs and hardware-optimized implementations never make it into production code but are instead abandoned once they served the purpose of yielding (another) publication. At the same time, community software packages driving scientific research lack the addition of new technology and hardware-specific implementations. This results in a very unsatisfying situation where researchers and software developers are working independently, and the traditional peer reviewing is reaching its capacity limits. A paradigm shift that accepts high-quality software pull requests to open source research software as conference contributions may create incentives to realize new and/or improved algorithms in community software ecosystems. In this paper, we propose to complement code reviews on pull requests to scientific open source software with scientific reviews, and allow the presentation and publication of high quality software contributions that present an academic improvement to the state-of-the-art at scientific conferences.

1. Motivation

Academic research in general – and the field developing algorithms for high performance computing in particular – suffers under the *publish or perish* paradigm. One consequence is the exponentially-increasing number of journal publications, workshop contributions, and conference proceedings [1]. Even though the development of High Performance Computing (HPC) algorithms is a relatively small research field, it is virtually impossible to keep track of the entire work contributed by peers.

The *publish or perish* paradigm primarily originates from the fact that academic reputation is still primarily based on scientific metrics such as the Hirsch-Index [2], the plain number of publications, or the more recently introduced altmetrics [3]. Acknowledging the merits of these metrics, the community benefits of classical publication formats are limited – particularly in comparison to other, more effective technology dissemination strategies like community code contributions. However, in particular against the background of the sluggish acceptance of *research software engineering* as an academic field, many researchers working on high performance algorithm development rely on scientific papers to ensure their own job security.

The papers presented at HPC conferences often present derivations of novel algorithms, the development of new implementations for large-scale parallelism or new hardware technology, or large-scale simulation runs. In many cases, the algorithms are realized in a prototypical implementation that fulfills the requirements for proposing and presenting the technology in a scientific paper or conference contribution. Yet, such implementations often fail to contribute to the community's software ecosystem: The publications typically lack the level of detail that is required to reproduce the technology, and, with prototype realizations often remaining private, the readers are unable to track the code.

☆ This work is an extension of the position paper "Are we Doing the Right Thing? – A Critical Analysis of the Academic HPC Community" presented in the context of the PDSEC workshop at IPDPS 2019. This work was supported by the "Impuls und Vernetzungsfond" of the Helmholtz Association under grant VH-NG-1241.

* Corresponding author at: Steinbuch Centre for Computing (SCC), Karlsruhe Institute of Technology, Germany.
E-mail addresses: hartwig.anzt@kit.edu (H. Anzt), eileen.kuehn@kit.edu (E. Kuehn), flegar@uji.es (G. Flegar).

In response to the situation, different publication formats now encourage (or even require) the release of source code and supporting data alongside the scientific content. These reproducibility efforts [4,5] providing reviewers access to the raw material aim at increasing the replicability, traceability, and general software quality. The side benefit is that the community can leverage the novel technology by accessing the sources and re-engineering the algorithms into already existing software libraries or simulation codes. These efforts have in common that the software developers initially have to submit original work in a classical paper format, and the reproducibility efforts in a second step check the validity and replicability of the software and results.

Rather disconnected from the publication-oriented high performance algorithm research, there exists a number of established open source software packages that are developed as a collaborative community effort [6] to provide domain scientists with the technology and the tools to realize scientific simulations and -analyses. These software packages typically feature a high standard in terms of software quality and software sustainability [7], and serve as the powertrain behind many of the recent research achievements. Well-known examples include the MFEM [8], deal.II [9], Trilinos [10], PETSc [11], hypre [12], and SuperLU [13] software packages.

At the same time, these community research software packages are dependent on high-quality contributions from software developers. And with scientists responding to the academic pressure on publishing scientific papers, the software packages are often lacking novel algorithm technology and hardware-specific optimizations. At best, software packages have access to prototype versions of novel algorithm technology, and are inclined to integrate those into the software stack to satisfy customer requests even though the code lacks the level of documentation, testability, and code readability that is preferred for software sustainability.

In a summarizing analysis of the field, the HPC community

- responds to academic pressure by publishing an increasing number of scientific papers (often containing novel algorithms and parallelization strategies);
- bears a significant amount of prototype implementations for novel algorithm technology, which remain in private possession of their authors;
- serves domain scientists by providing and contributing to open source software packages;
- falls short in releasing novel algorithm technologies as high-quality production-ready implementations featuring detailed documentation, tests, usage examples, and problem-specific efficiency analyses.

Obviously, it is not realistic to quickly change the academic system to endorse scientific software developers or base the promotion to tenure on software quality. However, we want to promote the importance of high-quality software contributions for accelerating computational science in HPC by proposing to base conference contributions no longer just on scientific papers in the traditional sense, but by introducing a new type of conference contribution based on pull requests of well-documented software contributions to established open source community software. To accept such a software contribution, it must meet certain minimal requirements. It must at least add new functionality or comprehensively improve existing functionality of the open source research software. Regarding the extent of a software contribution we follow the definition of semantic versioning [14] and define that such a software contribution supports at least a *minor* increment of the community software.

In this paper, we outline a workflow that combines the code review of pull requests to scientific open source software with a scientific review of the academic contribution to allow for the presentation and publication of these contributions at scientific conferences. For that purpose, we initially review in Section 2 existing efforts, either aiming at the publication of open source code like the Journal on Open Source Software (JOSS), or pushing for the publication of raw data and used code in reproducibility initiatives. In Section 3 we characterize the field of computer-based science from a “helicopter position” and identify the need for a new peer-review workflow that scales with the number of scientific papers and software contributions. The idea of crediting pull requests to open source research software as an academic contribution is outlined in Section 4, a realistic workflow for this approach is detailed in Section 5. In Section 6 we provide an example for a possible design of a pull request to be considered as a conference contribution. Like virtually all peer reviewing concepts, also the idea presented in this paper has its limitations, as we elaborate in Section 7 before summarizing the current status and positioning it in the wider field in Section 8.

2. Existing efforts

There already exist several strong efforts to improve the academic peer reviewing system for scientific software. Among the most well-known examples are the Replicated Computational Results (RCR [15]) initiative of the ACM Transactions on Mathematical Software (ACM TOMS [16]), and the Journal of Open Source Software (JOSS [17]). The two have orthogonal intentions, putting their main focus either on the scientific or software contribution.

ACM TOMS is a journal in the traditional sense. The RCR initiative aims at enhancing the traditional peer review by a reviewing of computational results. Therewith, the purpose of RCR activities is to provide independent confirmation that results contained in a manuscript are correct and reproducible. Successful completion of the RCR process gives the manuscript a Replicated Computational Results designation, which is noted on the first page of the published article [15].

While this strategy certainly enhances the quality of the contributions and ensures full reproducibility of the results, the authors are still required to submit a scientific article in the traditional form. This is not equivalent to submitting software as the main contribution itself, as we propose. Instead, authors need to put comprehensive efforts into the scientific elaboration as well as the experimental setup and reproducibility that might lead to low sustainability of the software contribution itself. Furthermore, ACM TOMS is a traditional journal publishing original work and not connected to any conference where the authors can present their work. Therewith, the RCR initiative is somewhat orthogonal to the process we suggest, but remains as a pioneer aiming at reproducibility and crediting software development as one of the lighthouse examples in the academic community.

The Journal of Open Source Software takes a radically different approach. As the name suggests, JOSS publishes articles about research software. This includes software that: (1) solves complex modeling problems in a scientific context (physics, mathematics, biology, medicine, social science, neuroscience, engineering); (2) supports the functioning of research instruments or the execution of research experiments; (3) extracts knowledge from large data sets; and (4) offers

a mathematical library, or similar [17]. A submission should consist of software that is open source as per the OSI definition and have an obvious research application. In contrast to the ACM TOMS RCR process, the submission should not focus on new research results accomplished with the software, but on the research software itself [17]. Upon acceptance into JOSS, a Crossref DOI is minted and the paper gets listed on the JOSS website.

A submission to JOSS consists of a markdown file (eventually accompanied with bibtex files and figures) whose metadata is submitted in a feedback form to the JOSS paper repository. During the peer review, the scientific quality, relevance, correctness, and sustainability of the software is checked. The review is open and public and happens within an issue in the JOSS repository, detached from the software contribution and release itself. This implies scientists working with a certain software package will likely miss the JOSS review discussion of this software package. This, in particular, is true for application scientists not directly working on the software but using it as a component in a larger application.

Furthermore, reviewing an entire, complex software package is a significant effort, and can never cover all details of a specific (e.g. new) algorithm. At the same time, JOSS is the first platform that allows for submitting complete software packages, therewith giving software development and research software engineers visibility and credit they would not receive otherwise. Furthermore, the reviewers conducting the software review also receive credit by being visible to the scientific community. The exponentially-increasing number of submissions to JOSS proves the success of this concept.

While we acknowledge that these efforts are extremely effective in pursuing the same goal, our approach complements them by targeting the problem from a different angle. In particular, we see further potential for minimizing the efforts of researchers, reviewers, and the open source community. In the case of JOSS, the reviewers are confronted with a massive code review effort, which is not only defined by the code size of software packages, but also by a possibly very extensive scientific functional range, which may require expertise in more than one field. In the case of the ACM TOMS RCR, the review effort of the software artifacts remains manageable, but focuses on the reproducibility of the results and not on the reusability and compatibility of the software itself. As the integration into existing open source community projects or software compatibility proofs are not required, the novel algorithm features can still remain in prototype code outside the community software stack.

3. The collaborative development effort of open source community software

Community software packages are typically developed in the environment of a distributed versioning system such as Git [18] or Mercurial [19]. These versioning systems are not only able to take snapshots of source code that can be revisited or retrieved at a later point, but also provide the means to track changes and orchestrate modifications introduced by several developers, therewith enabling the efficient development of software in a collaborative effort. The underlying concept to efficiently put collaborative efforts in practice is the concept of branches. Branches can encapsulate an independent thread of development as a well-defined sequence of changes, and can be based on or incorporate the state of other branches. During development, branches can even be synchronized with each other to share common changes.

In many Open Source projects, the *main branch* of a repository contains the stable version of the software that has been tested to work correctly on all supported backends and in all supported environments. The main branch is commonly protected from new software contributions and patches. Instead, changes and additions to the existing software are developed in separate branches until they are considered (1) functionally complete and self-contained, (2) sufficiently documented, and (3) verified regarding their expected functioning as well as supported backends and environments. These branches can then be submitted as a *software contribution* to the main branch. Technically, the concept of submitting a software contribution is, depending on the repository hosting platform, realized as a *pull request* on GitHub [20] or Bitbucket [21], or as a *merge request* on GitLab [22].

3.1. Pull requests to submit software contributions

Independent of the repository hosting platform, the software contributions are composed of new code contributions, including tests and code documentation, as well as a general commentary on motivation and description of the contribution with the main purpose to enable target-oriented reviews. The code and its auxiliary information are reviewed by other developers for correctness, consistency, and quality. If approved by the reviewers, the contribution is merged into the main branch of the repository. The merge integrates the source code of the new feature.

In addition to archiving the contribution itself, an advanced repository hosting platform such as GitHub or GitLab also archives the description and discussions that evolved as part of the pull request during the review process. All secondary information can be retrieved at a later point to track changes and recall argumentation or design choices. Archiving all secondary information also ensures contributors receive recognition for adding features, and allows to track who proposed changes or participated in discussions.

In the best case, a software contribution is accompanied with a detailed source code documentation (e.g. in terms of Doxygen [23] comments), unit tests, a realistic usage example, and an efficiency analysis for relevant problem and hardware settings. This way, a contribution not only extends the functionality of the software, but at the same time establishes a comprehensive documentation of the software and its features.

3.2. The wide spectrum of pull request reviews

Each pull request is immediately visible to the maintainers of the community software package, and automatic notification of maintainers is common. Yet, not only maintainers and code owners can comment and send feedback about the meaningfulness, completeness, validity, and quality of the code contributions and the pull request itself. Instead, the pull request is open for discussion for the whole community and, therefore, potentially can generate a reputation for all contributors including authors of the software contribution, official reviewers as well as commenting peers.

The workflow in established community research software packages usually foresees an extensive support of human review by automatic reviews. Automatic reviews can cover the general and idiomatic characteristics of the source code, e.g. with services, or bots checking common formatting style guides, linting rules, maintainability or security

Table 1
Criteria for software pull requests to be treated as conference publications and their relevance for the review process. This review process consists of a pull request review by the community of the software package, a scientific review by the program committee of a given conference, as well as some automated technical checks in the source code repository ensuring the code quality standards are adhered. The relevance of the given criteria ranges from low (+) to high (+++). Criteria that do not apply are marked by . For example, checking the quality of a user documentation is virtually impossible to automate, and thus marked by . The column *scope* shows where the information is archived and accessible to the user community. The list is based on selected criteria for software in large community software packages and extended by criteria relevant to the scientific community. The list of criteria is not exhaustive, and relevant criteria should be selected or supplemented according to the specific use case.

Criterion	Scope	Relevance for review		
		Community	Science	Auto
Detailed description				
Motivation, Summary	Pull Request	+++	+++	−
Scope, Use Cases	Pull Request	+	+++	−
Related Work	Pull Request	+	++	−
Originality	Pull Request	++	+++	−
Documentation				
User Documentation	Branch	++	+	−
Developer Documentation	Branch	++	−	−
Performance Analysis	Branch	++	++	−
Example Usage, Tutorial	Branch	++	++	−
Tests				
Tests and Coverage	Branch	+++	-	+++
Quality of Tests	Branch	++	−	−
Compatibility Tests	Repository	++	−	+++
Authors, Co-Authors, Contributors	Branch, Pull Request	++	+++	−
Deployment	Repository	++	++	+++
Integration with Community Package	Branch	+++	++	−

issues. This automation allows to enforce a general consensus on quality across an entire community. Furthermore, more complex processes such as continuous integration services can be triggered that check the functioning of the software contribution for various backends and supported environments, ensure integration with associated software packages, perform benchmarks and unit tests, or pre-built documentation and perform a deployment in a testing environment to support human reviewers. Such customized verification allows to guarantee specific quality requirements of individual projects.

Automatic reviews serve to ensure the well-defined desired characteristics of the software. Together with the expert judgement of a human reviewer, they form the software contribution review to ensure technical viability and quality.

4. Software pull requests as a conference contribution

We propose to emphasize the significance of software contributions by making them a contribution concept for conferences on HPC algorithms. Obviously, a software contribution submitted as a conference contribution is required to satisfy not just technical but also scientific requirements, such as a detailed algorithm description and feature specification, but also functionality testing and efficiency analysis. The idea is that researchers directly submit a software pull request of a legitimate software contribution – which has successfully passed automatic reviews and code reviews by code owners, maintainers, and the associated community of an established open source research software package – as a conference contribution. The information about the submission of the pull request as a conference contribution will automatically inform the editor of the designated conference and allows them to assign a number of anonymous reviewers of the conference's program committee.

Since the contribution already passed a review process as part of its acceptance to the community software project, the reviewers of the program committee do not need to focus on verifying every detail of the implementation, but rather on general aspects such as the novelty of the work, the clarity, scientific correctness, and completeness of description and the user-facing documentation. Thus, the reviewers evaluate the contribution in terms of scientific value, feature significance, and whether this software contribution qualifies to be presented at the conference. In the end, the documentation of a software contribution in a pull request may not be too different from a scientific paper, however coming with significant benefits:

- Full reproducibility and traceability for technical as well as scientific value is ensured, as not only reviewers but the entire community can track the software contribution;
- The versioning systems keeping track of the authors of each line helping to identify the main contributors of a software contribution, but also to link to the right person in case of questions;
- Novel algorithms and hardware optimized implementations are ready for integration into open source software repositories already at the point of submitting the new contributions;
- The whole community can contribute to the development of a novel algorithm by commenting or even extending on software contributions – without the individuals losing the recognition for their ideas as the interactions are publicly available and tracked by the collaboration platform;
- Designing software pull requests as a conference contributions naturally implies a high quality of code documentation, and efficiently enables users to evaluate (based on the contribution and the

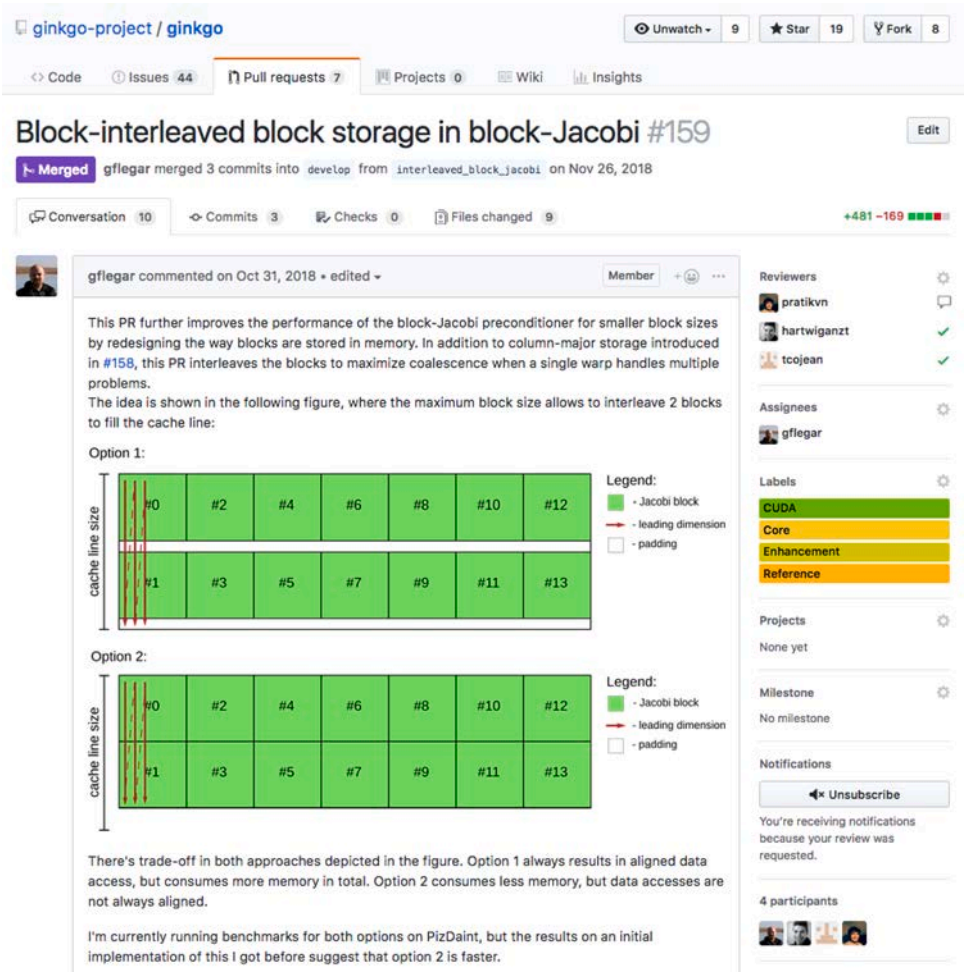

Fig. 1. Contribution description on the collaboration platform.

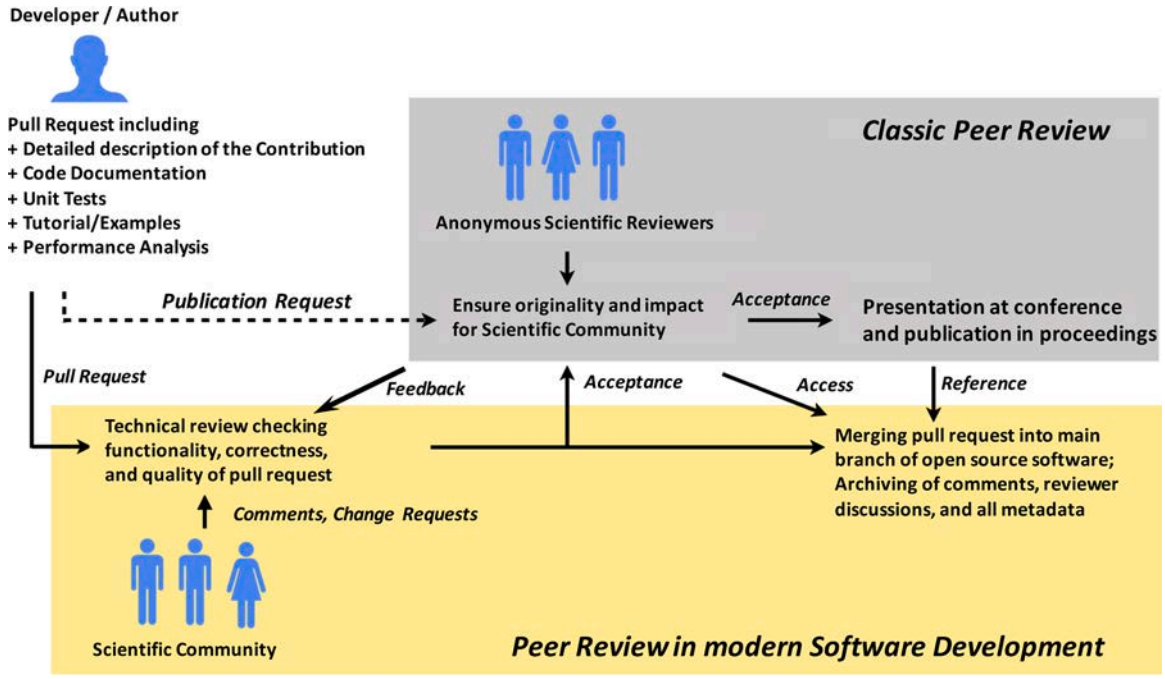

Fig. 2. Peer review workflow accepting pull requests as a conference contributions.

included efficiency analysis) the appropriateness of a software feature for a specific problem;

- Presenting contributions at a conference not only makes the whole community aware of a new feature, but domain scientists can directly approach the developers, establish contact, and discuss technical aspects;
- The submission rate will be far lower, and the acceptance rate far higher, as each submission must pass at least some pre-review by community software developers and the authors of the papers will be forced to produce a higher quality contribution.

Combined with the expected lower submission rate, we do not expect that the total reviewing effort for the reviewers of the program committee exceeds the reviewing effort for a conference with traditional contributions. We even expect a positive impact on reviewing effort for technical reviews as reviewers of the community software project have additional information facing the conference reviewers but supporting the overall logic and motivation of the software contribution.

Even though the benefits for the community are obvious, a contribution of this type alone may be unable to provide the same academic reward a scientific paper comes with. Hence, in order to boost the appeal and benefits for the contributing researcher(s), we propose to complement the concept with post-conference proceedings that accept software contributions as scientific publications.

Technically, these publications may differ from *traditional* papers by featuring a shorter general introduction, as it is not necessary to fundamentally motivate the importance of scientific high performance computing. On the other hand, we expect the technical aspects to be discussed more elaborately, as, besides the algorithm presentation, the feature description also has to include the user documentation, usage examples or tutorials, and a scalability or efficiency analysis. In fact, the technical content of the publication should comprise all information necessary to understand the functionality, its application field, and usage. We also expect that the acknowledgment list reflects the community- and reviewer comments, as well as the hardware facilities accessed to ensure cross-platform portability of the contribution. We think that this adapted design of a conference proceeding contribution does not harm the readability, but instead makes the publication more attractive to researchers active in the fields of algorithm engineering and scientific computing.

5. Implementing a workflow for accepting pull requests as a conference publication

In Fig. 2 we outline the peer review workflow we envision for accepting pull requests to community software as a conference publication. To make the submission of a pull request as a conference contribution as convenient as possible, we propose to facilitate software contribution templates. These templates provide the creator of a software contribution with the information on what needs to be provided for a successful submission, and guidelines on how to best do so. Table 1 summarizes relevant criteria for the community along with scientific and technical requirements. For example, the description of the pull request is of major importance for the scientific review but also helps to guide the community review. However, the main publication asset remains the code contribution and, therefore, the author also needs to include the functionality and contribution information in preserved locations such as documentation and tests. The template tries to also reflect a weighting of the different aspects. Conferences supporting the proposed contribution type for software contributions can define their own set of minimum requirements by providing a customized software contribution template. Technically speaking, software contribution templates can be realized by *pull request templates* in case of GitHub or by *merge request description templates* in the case of GitLab.

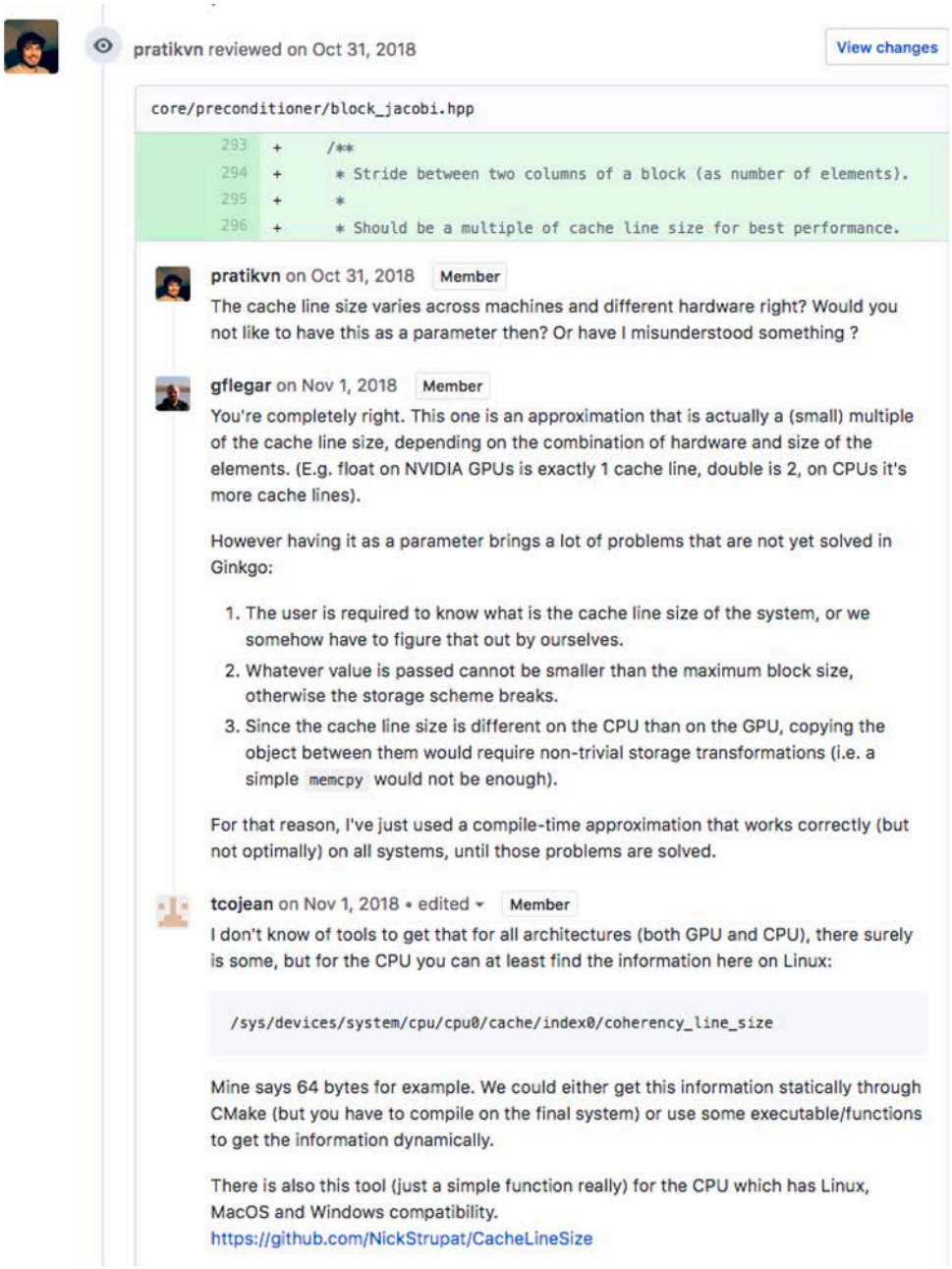

Fig. 3. Technical discussions of the implementation make up the software contribution.

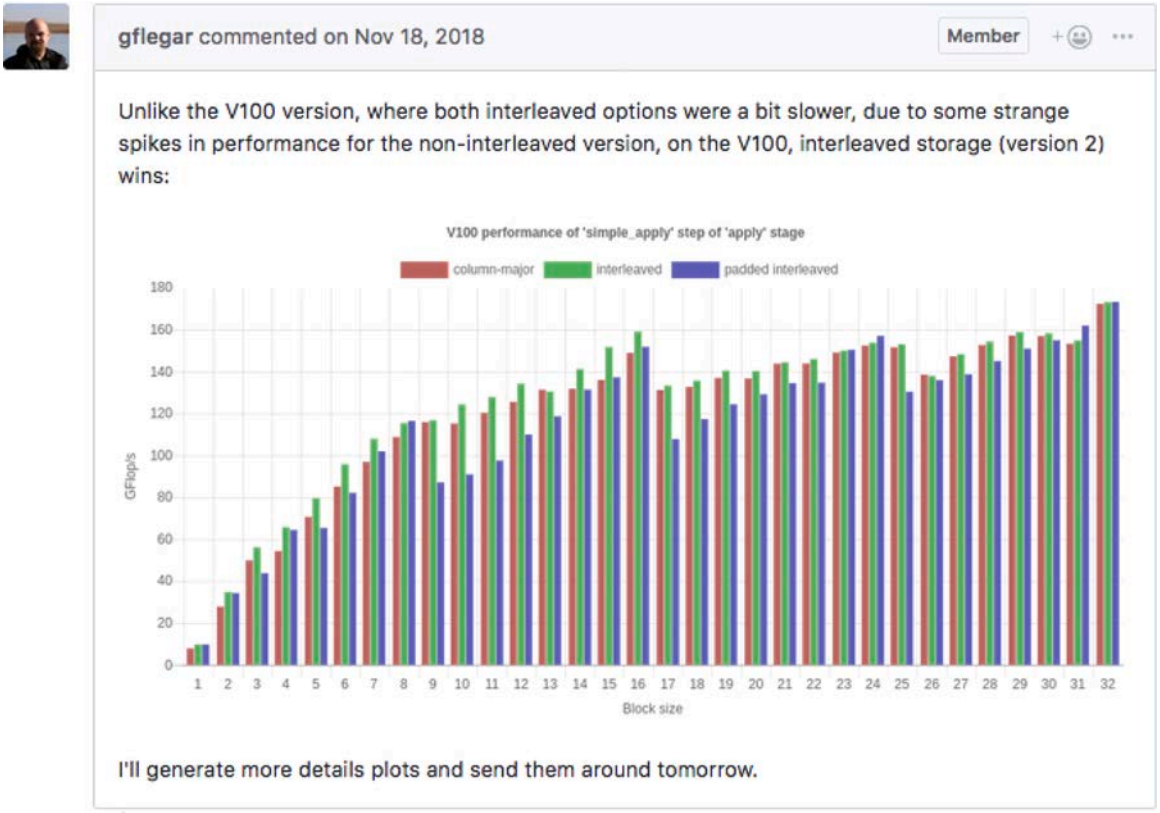

Fig. 4. Performance aspects provided alongside the software contribution on the collaboration platform.

The pull request template allows the creator to indicate that he wants the pull request to an open source community software to be considered as a submission to a certain conference. While the pull request template can automatically extract information such as author, target software package, license model, and programming language, the creator should additionally specify keywords indicating research area, algorithm functionality, and scope of the contribution. All additional information and performance analysis are then part of the detailed functionality description of the pull request.

In addition to the standardized metadata collection, the pull request template improves the overall automation and orchestration of the review process via preset assignment of labels and actions. For example, this allows to automatically inform the chair of a conference once the technical code review of the software contribution is completed. At this point, the automated workflow requests a scientific review, and the review template allows to track the progress of the review. Intertwining technical and scientific review, it is possible for the scientific reviewers to request changes that are automatically passed on to the authors and technical reviewers.

While a classical software pull request would be merged into the master branch of the software package once the technical review is completed, delaying the merge until also the scientific review has completed and automatically forwarding reviewer comments and change requests to the authors and technical reviewers improves the contribution before disseminating the new functionality to the community. Only after the scientific review has passed and also the technical review team addresses all suggested changes and additions, the software pull request is merged and closed. This strategy is likely to reduce the number of patches necessary to improve new features released in community software packages. Once the scientific reviewers and the technical reviewers are satisfied, the pull request is merged, and the template automatically initiates the conference invitation as well as the publication in the conference proceedings.

If the scientific reviewers do not recommend the acceptance of the pull request as a conference contribution, the pull request template falls back to the standard software development workflow that allows to merge to the main branch of the community software once the technical reviewing team accepted the contribution. In that sense, the scientific reviewers do not have the power to intervene the acceptance of the pull request, but the decision of the acceptance of the pull request is left to the code reviewers. We are sure that this concept allows to propagate scientific input and ideas from the reviewers to the developers to improve their algorithms and methods.

6. Example of a well-designed software contribution

We use an example to illustrate how to design a software pull request that qualifies as a conference contribution. Instead of discussing an artificial contribution, we recall an already existing pull request to the Ginkgo[1] Open Source library publicly hosted on the GitHub repository hosting site. We emphasize that we do not select the pull request #159[2] because of its technical and scientific content (qualifying as a conference contribution), but instead because of its compactness (qualifying as a short example) and its completeness in terms of documentation and efficiency assessment.

The repository hosting platform makes it straightforward to identify relevant parts of the pull request, see screenshot of the pull request on the collaboration platform shown in Fig. 1. The name of the contribution

[1] https://ginkgo-project.github.io/.
[2] https://github.com/ginkgo-project/ginkgo/pull/159.

(Block-interleaved block storage in block-Jacobi #159), the contributor (on the left: gflegar), the reviewers that approved the pull request (on the right: hartwiganzt and tcojean) are clearly visible. Furthermore, the platform allows us to add labels that are similar to keywords in a classical scientific application (here: cuda, core, enhancement, reference). Not employed in this examples is the possibility to link to a project and a milestone, which would allow for further context of the contribution. Finally, all individuals that participated in the related discussions are listed.

The pull request starts off with a description of the new capability, and illustrates the strategy used to realize the feature. The description of the new functionality is straight forward, refers to previous contributions, and uses a figure to sketch out the strategies. We note that the header of the contribution also provides information on the current status of the pull request. In this case it shows, that the contribution was successfully merged into the *develop* branch of the project on November 26th, 2018.

What now follows is what we define as the first part of the review: a community discussion on the functionality, its algorithmic realization, the software quality, and implementation aspects. For this contribution, there was no discussion about the functionality itself or its properties. For #159, only implementation aspects were discussed, with an example reported in Fig. 3.

The collaboration platform tracks the complete discussion along with the participants and timestamps of the contributions. The possibility to add code fragments or tie comments to lines of code makes it easy for the reader to link these aspects to the implementation.

Finally, the contribution was accompanied with some efficiency/performance assessment we list in Fig. 4. This also enables readers and reviewers that do not have access to the target hardware to follow the argumentation or assess the quality of the contribution.

When submitting the software contribution as a conference contribution, the program committee can (but is not required to) access and dissect the code on the collaboration platform hosting the repository. This enables to review the contribution and to assess the quality of the software contribution. For a post-conference proceedings, one option is to append the most relevant code segments in the appendix – which is what we do in this example. An alternative is to complement the proceeding with a digital artifact, or directly refer to the contribution archived in the collaboration platform.

The graphical comparison of the original code and the code enhancement of the software contribution makes it easy to evaluate the algorithmic realization. For brevity, we do not list the complete code of the contribution (interested readers may find that under #159[3]), but instead show an example of code created by the contribution and code that is modified by the contribution. Fig. 6 in the Appendix reports new code contributed by the contribution that is enhanced with Doxygen [23] documentation and comments indicating future steps. The file shown in Fig. 5 is heavily modified by the contribution. The collaboration platform employs colors to visualize Git's `diff` command, which makes it easy to track the changes introduced.

7. Scope and limitations

We recognize that the contribution format proposed in this work is not suitable for all types of conference contributions. One example would be a purely theoretical exposition of a new algorithm or method that does not yet have a high performance implementation, and whose practical implementation or performance is not part of the contributions. Another example are papers that do not aim at contributing an algorithm or software component, with this paper being such an example. Thus, we do not propose to completely abandon the traditional concept of scientific papers, but to allow for a wider variety of contribution formats that are in line with the contribution type. Ultimately, it remains the program committee's responsibility to judge whether the format of a specific contribution is adequate for its type.

Even in cases where a software contribution would be an appropriate contribution, there could be special circumstances which do not allow for the publication of a software contribution. Such examples include cases where the implementation itself is classified or proprietary due to a third-party contract. One possible approach would be to allow the contribution in a classical paper format, augmented with a statement from the third party that verifies that the software in question is indeed protected by the contract, and that the claims made about it in the contribution are valid.[4]

8. Summary

Like in any other academic field, scientists in High Performance Computing suffer under the *publish or perish* paradigm. As a result, novel algorithm designs and high performance kernel implementations often reside as prototype implementation and are never adopted as production-ready community code. To counteract this inefficiency, we propose to establish a new form of conference contribution that is based on software pull requests to open source community software.

The idea is to complement a technical review as it is established in modern open source software development to ensure technical quality such as maintainability, testability and sustainability, with a scientific review assessing originality and impact. An accepted conference contribution then consists of the algorithm description, the technical analysis, and the performance assessment of the software contribution. Publishing the contribution as post-conference proceeding aims at providing the authors with the same academic rewards as publishing the new technology as a traditional journal paper. At the same time, the community benefits with outstanding traceability, fast propagation of new technology into the community software, and excellent documentation of source code.

In a larger picture, accepting software pull requests as conference contribution increases the sustainability of open source community software, ensures public research money for software development, advances the scientific community as a whole, and enhances the strengths of open source software against commercial software packages. Furthermore, it is another step in the direction of entrenching scientific software development as an academic field, and moving the academic evaluation system from traditional metrics (such as the Hirsch-Index) towards community-advancing software contributions.

Conflict of interest

The authors declare no conflict of interest.

Declaration of Competing Interest

The authors report no declarations of interest.

Acknowledgments

The authors want to express their appreciation for comments of the anonymous reviewers of the PDSEC'19 workshop. Acknowledging that this paper is provocative and we sure failed to consider all aspects of this controversial topic, we are highly thankful for the valuable feedback and input. We also thank Fabian Brunk for comments and discussion on an earlier version of the paper.

[3] https://github.com/ginkgo-project/ginkgo/pull/159.

[4] Since the authors of this work are only supported through public funding, this approach is only a suggestion. The actual solution should be discussed with scientists that have third party arrangements.

Appendix A

106 cuda/preconditioner/block_jacobi_kernels.cu — Copy path | View file

```
@@ -48,16 +48,28 @@ SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 namespace gko {
 namespace kernels {
 namespace cuda {
+
+
+/**
+ * A compile-time list of block sizes for which dedicated generate and apply
+ * kernels should be compiled.
+ */
+using compiled_kernels = syn::compile_int_list<1, 13, 16, 32>;
+
+
 namespace kernel {
 namespace {

+
 template <int max_block_size, int subwarp_size, int warps_per_block,
           typename ValueType, typename IndexType>
 __global__ void __launch_bounds__(warps_per_block *cuda_config::warp_size)
     generate(size_type num_rows, const IndexType *__restrict__ row_ptrs,
              const IndexType *__restrict__ col_idxs,
              const ValueType *__restrict__ values,
-             ValueType *__restrict__ block_data, size_type stride,
+             ValueType *__restrict__ block_data,
+             preconditioner::block_interleaved_storage_scheme<IndexType>
+                 storage_scheme,
              const IndexType *__restrict__ block_ptrs, size_type num_blocks)
 {
     const auto block_id =
@@ -79,15 +91,18 @@ __global__ void __launch_bounds__(warps_per_block *cuda_config::warp_size)
                                     trans_perm);
         copy_matrix<max_block_size, and_transpose>(
             subwarp, block_size, row, 1, perm, trans_perm,
-            block_data + (block_ptrs[block_id] * stride), stride);
+            block_data + storage_scheme.get_global_block_offset(block_id),
+            storage_scheme.get_stride());
     }
 }


 template <int max_block_size, int subwarp_size, int warps_per_block,
           typename ValueType, typename IndexType>
 __global__ void __launch_bounds__(warps_per_block *cuda_config::warp_size)
-    apply(const ValueType *__restrict__ blocks, int32 stride,
+    apply(const ValueType *__restrict__ blocks,
+          preconditioner::block_interleaved_storage_scheme<IndexType>
+              storage_scheme,
           const IndexType *__restrict__ block_ptrs, size_type num_blocks,
           const ValueType *__restrict__ b, int32 b_stride,
           ValueType *__restrict__ x, int32 x_stride)
```

Fig. 5. The patch applies significant changes to already existing code.

Fig. 6. The patch adds a significant amount of new code to an existing file.

References

[1] UNESCO Science Report: Towards 2030, UNESCO Reference Works Series, UNESCO, 2015. https://books.google.de/books?id SDHwCgAAQBAJ.

[2] J.E. Hirsch, An index to quantify an individual's scientific research output, Proc. Natl. Acad. Sci. USA 102 (46) (2005) 16569–16572, https://doi.org/10.1073/pnas.0507655102. , http://www.pnas.org/content/102/46/16569.Abstract, http://www.pnas.org/content/102/46/16569.full.pdf+html.

[3] J. Priem, D. Taraborelli, P. Groth, C. Neylon, Altmetrics: A Manifesto, altmetrics.org, 2011.

[4] Michael Allen Heroux, The TOMS Initiative and Policies for Replicated Computational Results (RCR), 2017. https://toms.acm.org/replicated-computational-results.cfm.

[5] Supercomputing Conference Reproducibility Initiative, 2018. https://sc18.supercomputing.org/submit/sc-reproducibility-initiative/.

[6] xSDK: Extreme-scale Scientific Software Development Kit https://xsdk.info/ (accessed in August 2018).

[7] Better Scientific Software (BSSw) https://bssw.io/ (accessed in August 2018).

[8] MFEM: Modular finite element methods library, mfem.org. https://doi.org/10.11578/dc.20171025.1248.

[9] W. Bangerth, R. Hartmann, G. Kanschat, deal.II – a general purpose object oriented finite element library, ACM Trans. Math. Softw. 33 (4) (2007), 24/1-24/27.

[10] T. Trilinos Project Team, The Trilinos Project Website.

[11] S. Balay, S. Abhyankar, M.F. Adams, J. Brown, P. Brune, K. Buschelman, L. Dalcin, A. Dener, V. Eijkhout, W.D. Gropp, D. Karpeyev, D. Kaushik, M.G. Knepley, D. A. May, L.C. McInnes, R.T. Mills, T. Munson, K. Rupp, P. Sanan, B.F. Smith, S. Zampini, H. Zhang, H. Zhang, PETSc Web Page, 2019. https://www.mcs.anl.gov/petsc.

[12] R.D. Falgout, U.M. Yang, hypre: a library of high performance preconditioners, in: P.M.A. Sloot, A.G. Hoekstra, C.J.K. Tan, J.J. Dongarra (Eds.), Computational Science – ICCS 2002, Springer Berlin Heidelberg, Berlin, Heidelberg, 2002, pp. 632–641.

[13] X.S. Li, An overview of superlu: algorithms, implementation, and user interface, ACM Trans. Math. Softw. 31 (3) (2005) 302–325, https://doi.org/10.1145/1089014.1089017.

[14] T. Preston-Werner, Semantic Versioning 2.0, 0, línea], 2013. Available from: http://semver.org.

[15] M.A. Heroux, Editorial: Acm toms replicated computational results initiative, ACM Trans. Math. Softw. 41 (3) (2015 Jun), https://doi.org/10.1145/2743015.

[16] ACM, Transactions on Mathematical Software (ACM TOMS), vol. 33(4, 2007, p. 24/1. https://dl.acm.org/journal/toms.

[17] Journal of open source software, https://joss.theoj.org/.

[18] Git https://git-scm.com/.

[19] O'Sullivan, Bryan, Mercurial: The Definitive Guide, O'Reilly Media, Inc., 2009.

[20] GitHub https://github.com/.

[21] Bitbucket https://bitbucket.org/.

[22] GitLab https://gitlab.com/.

[23] D. van Heesch, Doxygen: Source Code Documentation Generator Tool, 2008. http://www.stack.nl/dimitri/doxygen/.

Hartwig Anzt is a Helmholtz-Young-Investigator Group leader at the Steinbuch Centre for Computing at the Karlsruhe Institute of Technology. He obtained his PhD in Mathematics at the Karlsruhe Institute of Technology, and afterwards joined Jack Dongarra's Innovative Computing Lab at the University of Tennessee in 2013. Since 2015 he also holds a Senior Research Scientist position at the University of Tennessee. Hartwig Anzt has a strong background in numerical mathematics, specializes in iterative methods and preconditioning techniques for the next generation hardware architectures. Hartwig Anzt has a long track record of high-quality software development. He is author of the MAGMA-sparse open source software package managing lead and developer of the Ginkgo numerical linear algebra library.

Eileen Kuehn received her PhD in computer science in 2017. She currently works at the Karlsruhe Institute of Technology in the domain of quantum computing. Her career includes work as research associate and project coordinator in several EU projects. For many years already, she is working in close collaboration with diverse domains including High Performance Computing, High Energy Physics, Climatology, or Museology. Her research activities focus on scalable data analytics for highly parallel, distributed systems and sustainable software.

Goran Flegar received his PhD from the University of Jaume I with focus on High Performance Computing. His research interests include sparse linear algebra, accelerator computing and software design. He also holds a bachelor's degree in mathematics and a master's degree in computer science and mathematics from the University of Zagreb. He is one of the founders of the Ginkgo software package, a modern C++ library primarily focused on the iterative solution of sparse linear systems via preconditioned Krylov subspace methods on high performance GPU and multicore architectures.