Are we doing the right thing? — A Critical Analysis of the Academic HPC Community

Hartwig Anzt*†, Goran Flegar‡
*Steinbuch Centre for Computing, Karlsruhe Institute of Technology, Germany
†Innovative Computing Lab (ICL), University of Tennessee, Knoxville, USA
†Departamento de Ingeniería y Ciencia de Computadores (ICC), Universidad Jaume I, Castellón, Spain
hartwig.anzt@kit.edu, *flegar@uji.es*

***Abstract*—Like in any other research field, academically surviving in the High Performance Computing (HPC) community generally requires to publish papers, in the bast case many of them and in high-ranked journals or at top-tier conferences. As a result, the number of scientific papers published each year in this relatively small community easily outnumbers what a single researcher can read. At the same time, many of the proposed and analyzed strategies, algorithms, and hardware-optimized implementations never make it beyond the prototype stage, as they are abandoned once they served the single purpose of yielding (another) publication. In a time and field where high-quality manpower is a scarce resource, this is extremely inefficient. In this position paper we promote a radical paradigm shift towards accepting high-quality software patches to community software packages as legitimate conference contributions. In consequence, the reputation and appointability of researchers is no longer based on the classical scientific metrics, but on the quality and documentation of open source software contributions — effectively improving and accelerating the collaborative development of community software.**

***Index Terms*—Scientific Excellence Paradigms, Conference Contributions, Scientific Reputation, Community Software Development**

I. State-of-the-Art

Academic research in general – and the field developing algorithms for high performance computing in particular – experiences an increasing number of journal publications, workshop contributions, and conference proceedings [1]. Even though the development of HPC algorithms is a relatively small research field, it is virtually impossible to keep track of all work contributed by peers. This increasing number of scientific publications reflects the pressure imposed by appointability for tenure and the fact that the reputation is still primarily based on scientific metrics like the Hirsch-Index [2] and the plain number of publications. Acknowledging the merits of these traditional metrics, the community benefits of classical publication formats are limited — particularly in comparison to other, more effective technology dissemination strategies. Due to the focus on the traditional metrics and with regard to the sluggish acceptance of "scientific software engineering" as an academic field, many researchers working on high performance algorithm development rely on scientific papers for job security. The papers then 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 algorithm is realized in a prototype implementation that fulfills the requirements for proposing and presenting the technology in a scientific paper or conference contribution, but fails to contribute to the community's software ecosystem: the publications typically lack the level of detail that allows to reproduce the technology, and, with prototype realizations often remaining private, the readers are unable to track the code. In response to the situation, different publication formats now encourage (or even require) the release of source code and supporting data. These reproducibility efforts [3], [4] 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 in already existing software libraries or simulation codes. Unfortunately mostly disconnected from these efforts, there exists a number of established open source software packages that are developed as collaborative community effort [5] to provide domain scientists with the technology and the tools to realize scientific simulations. These software packages typically feature a high standard in terms of software quality and software sustainability [6], and serve as the powertrain behind many of the recent research achievements. At the same time, these community 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 the production-ready implementation of novel algorithms and hardware-specific efficient implementations. As a result, software packages are inclined to also accept contributions that lack the level of documentation and code readability that would be preferred for software sustainability.

In a summarizing field analysis, the community of high performance computing

- responds to academic pressure by publishing an increas-

This work was supported by the U.S. Department of Energy Office of Science, Office of Advanced Scientific Computing Research, Applied Mathematics program under Award Numbers DE-SC0016513, DE-SC-0016564, and DE-SC-0010042. H. Anzt was supported by the "Impuls und Vernetzungsfond" of the Helmholtz Association under grant VH-NG-1241. G. Flegar was supported by the CICYT projects TIN2014-53495-R and TIN2017-82972-R of the MINECO and FEDER.

ing number of scientific papers (often containing novel algorithms and parallelization strategies);
- bears a significant amount of prototype implementations for novel algorithm technology in private possession;
- serves domain scientists by providing open source software packages;
- falls short in releasing novel algorithm technologies as production-ready implementations featuring detailed documentation and problem-specific efficiency analysis.

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 encourage a journey in this direction by proposing to base conference contributions no longer only on scientific papers in the traditional sense, but also on the submission of well-documented software patches to established open source community software.

II. The Concept of Software Patches to Open Source Community Software

Community software packages are typically developed in the environment of a distributed versioning system like Git [7] or Mercurial [8]. 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 tools to track changes and orchestrate modifications introduced by distinct developers, therewith enabling the efficient development of software as a collaborative effort. In healthy software development, every new contribution to the *main branch* of a repository (which is the branch containing a stable version that has been tested to work correctly on all supported backends and in all supported environments) is submitted as a *software patch*. Technically, the concept of submitting a patch is, depending on the repository hosting platform, realized as a *pull request* on GitHub [9] or Bitbucket [10], or a *merge request* on GitLab [11]. Independent of the hosting site, the software patches compose of the new code contributions, a description of the features/bug fixes added, and are reviewed by other developers for correctness, consistency, and quality. If approved by the reviewers, the patch is merged into the main branch of the repository. The merge integrates the source code of the new feature. In addition to archiving the patch itself, an advanced repository hosting platform like GitHub or GitLab also archives its documentation and discussions that evolved during the review process. All secondary information can be retrieved at a later point to track changes and recall arguments 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 patch is accompanied with a detailed functionality description (e.g., in terms of Doxygen [12] comments), a usage example, and an efficiency analysis for relevant problem and hardware settings. This way, a patch not only extends the functionality of the software, but at the same time establishes a comprehensive documentation of the software and its features.

III. Software Patches as Conference Contribution

We propose to emphasize the significance of software patches by making them a contribution concept for conferences on high performance computing algorithms. The idea is that researchers submit a software patch that has been accepted as a pull request / merge request on a public software repository hosting site as a conference contribution. The program committee evaluates the patch in terms of software quality, feature significance, and sustainability against the question of whether this software contribution qualifies to be presented at the conference. Obviously, a patch submitted as conference contribution is required to come with a detailed algorithm description and feature specification, but also some functionality testing and efficiency analysis. In the end, the documentation of a software patch may not be too different from a scientific paper, however coming with significant benefits:

- Full reproducibility and traceability is ensured, as not only reviewers but the complete community can track the software patch;
- The versioning systems keeping track of the authors of each line helps to identify the main contributors of a software contribution, but also to link to the right person in case of technical questions;
- Novel algorithms and hardware optimized implementations are integrated into open source software already at the point of publishing the new technology (or shortly after);
- The whole community can contribute to the development of a novel algorithm by commenting on software contributions – without the individuals losing the recognition for the ideas as the comments are publicly available and tracked by the collaboration platform;
- Designing software patches as conference contributions naturally implies an extremely high level of code documentation, and efficiently enables users to evaluate (based on the patch and the included efficiency analysis) the appropriateness of a software feature for a specific problem;
- Presenting patches 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 acceptance rate far higher, as each submission will most likely pass at least some pre-review process by library developers, and the authors of the papers will be forced to produce a higher quality contribution.

Since the patch already passed a review process as part of its acceptance to the community software project, the program committee does not need to focus on verifying every detail of the implementation, but rather on general aspects such as the novelty of the work and the clarity and completeness of the user-facing documentation. Combined with the expected lower submission rate, we do not expect that the total effort for reviewing the contributions for a conference based on

Fig. 1. Screenshot of the patch description on the collaboration platform.

software patch contributions exceeds the reviewing effort for a conference with traditional total manpower needed to complete the reviews for a conference with traditional contributions.

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 patches as scientific publications. Technically, these publications may differ from "traditional" papers by featuring a shorter general introduction, as it is not necessary to motivate the importance of scientific high performance computing. On the other hand, we expect the technical aspects to be discussed more elaborately, as, beside 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 would reflect 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 does not harm the readability, but instead makes the publication more attractive to researchers active in the fields of algorithm engineering and scientific

computing.

IV. Example of a Well-Designed Software Patch

We use an example to illustrate how to design a software patch that qualifies as a conference contribution. Instead of creating an artificial patch, we recall an already existing pull request to the Ginkgo[1] Open Source library publicly hosted on the GitHub [9] repository hosting site. We emphasize that we do not select the pull request #159[2] because of its technical 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 patch starts off with a description of the new capability, and illustrates the strategy used to realize the feature, see screenshot of the collaboration platform shown in Figure 1.

The repository hosting platform (in this case GitHub [9]) makes it easy to identify the name of the patch (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), and also allows to add labels that are somewhat 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. Finally, all individuals that participated in the related discussions are listed. The description of the new functionality is straight forward, refers to previous patches, and uses a figure to sketch out the strategies. We note that the header of the patch also provides information that the patch was successfully merged into the *develop* branch of the project on November 26th, 2018.

What now follows is a community discussion on the functionality, its algorithmic realization, the software quality, and implementation aspects. For this patch, there was no discussion about the functionality itself or its properties. For #159, only implementation aspects were discussed, with an example reported in Figure 2.

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 code lines makes it easy for the reader to link these aspects to the implementation.

Finally, the patch was accompanied with some efficiency /performance assessment we list in Figure 3. 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 patch 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 patch. For a post-conference proceeding, 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 simply refer to the patch archived in the collaboration platform.

The graphical comparison of the original code and the code enhancement of the software patch make it easy to evaluate the algorithm realization. For brevity, we do not list the complete

[1]https://ginkgo-project.github.io/

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

Fig. 2. Screenshot of technical discussions of the patch implementation.

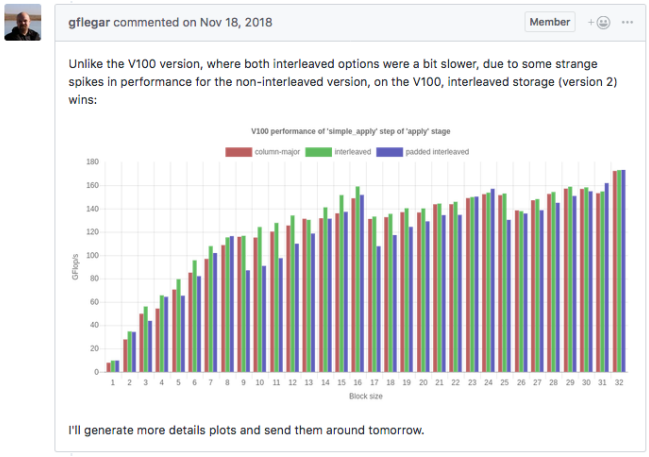

Fig. 3. Screenshot of performance aspects the patch is accompanied with on the collaboration platform.

code of the patch (interested readers may find that under #159[3]), but instead show an example of code created by the patch and code that is modified by the patch. Figure 4 in the Appendix reports new code contributed by the patch that is enhanced with Doxygen [12] documentation and comments indicating future steps. The file shown in Figure 5 is heavily modified by the patch. The collaboration platform employs colors to visualize Git's "diff" command [7], which makes it easy to track the changes introduced.

V. 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 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 patch would be an appropriate contribution, there could be special circumstances which do not allow for the publication of a patch. 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].

VI. Summary

In this position paper, we propose to base conference contributions on software patches to open source community software. The program committee evaluates the patch in terms of software quality, feature significance, and sustainability against the question of whether this patch qualifies to be presented at the conference. Publishing the algorithm description, the technical analysis, and the performance assessment of the software patch as post-conference proceeding aims at providing the authors with the same academic rewards like publishing the new technology as a traditional journal paper. At the same time, the community benefits with outstanding traceability, fast propagation of new technology via community software, and excellent documentation of source code. In a larger picture, accepting software patches as conference contribution is another step in the direction of entrenching scientific software development as an academic field, and moving the academic evaluation system from traditional metrics (like the Hirsch-Index) towards community-advancing software contributions.

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

[4]Since both 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.

Acknowledgments

The authors want 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.

References

[1] *UNESCO science report: towards 2030*, ser. UNESCO Reference Works Series. UNESCO, 2015. [Online]. Available: https://books.google.de/books?id=SDHwCgAAQBAJ

[2] J. E. Hirsch, "An index to quantify an individual's scientific research output," *Proceedings of the National Academy of Sciences of the United States of America*, vol. 102, no. 46, pp. 16 569–16 572, 2005. [Online]. Available: http://www.pnas.org/content/102/46/16569.abstract

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

[4] S. Conference, "SC Reproducibility Initiative," *https://sc18.supercomputing.org/submit/sc-reproducibility-initiative/*, 2018.

[5] "xSDK: Extreme-scale Scientific Software Development Kit https://xsdk.info/," accessed in August 2018.

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

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

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

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

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

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

[12] D. van Heesch, "Doxygen: Source code documentation generator tool," 2008. [Online]. Available: http://www.stack.nl/~dimitri/doxygen/

APPENDIX

Fig. 4. The patch adds a significant amount of new code to an existing file.

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.