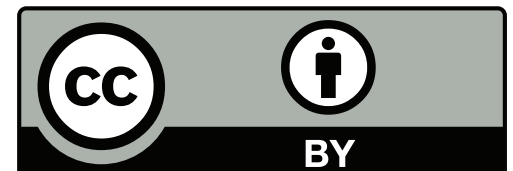


Open Science, Software Citation, and Reproducibility Best Practices

Kyle Niemeyer

ME 599: Software Development for Engineering Research
24 May 2019



Topics

- Software archival
- Software citation
 - Need for software citation
 - Principles of software citation
- Venues for sharing and publication
- Best practices for reproducibility

Sharing software (and data) openly has clear benefits to others *and* yourself

A paper that isn't accompanied by the software or data produced is just **advertising**.¹

People find reproducible results more trustworthy...

...and cite you more!²

Reduce duplicated effort *and* increase impact.

[1] Jon Claerbout & Martin Karrenbach (1992) "Electronic Documents Give Reproducible Research a New Meaning", <http://sepwww.stanford.edu/doku.php?id=sep:research:reproducible:seg92>

[2] Piwowar HA, Vision TJ. (2013) Data reuse and the open data citation advantage. *PeerJ* 1:e175 <https://doi.org/10.7717/peerj.175>

Open-Source Software

We've talked about:

- Version control
- Licensing
- Documentation
- Packaging/distribution

Great! All done?

Great! All done?



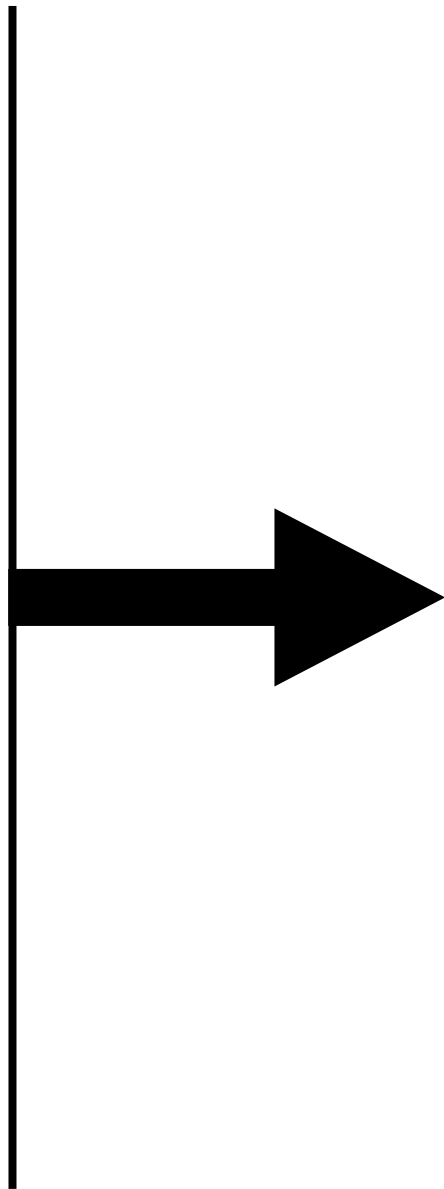
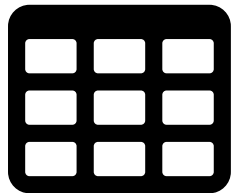
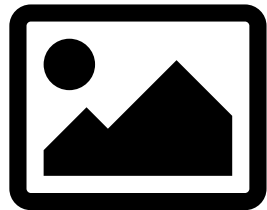
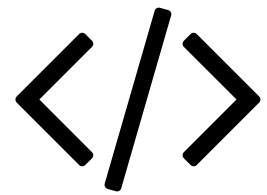
Great! All done?

For **research**, need one more step:

Archiving the software or data

Consider: what if you cite this,
and then someone modifies it?

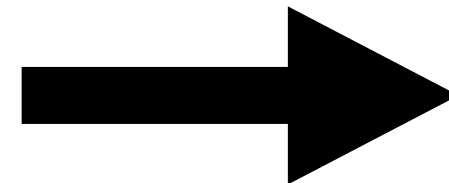
Archiving



zenodo



figshare



Live demo time

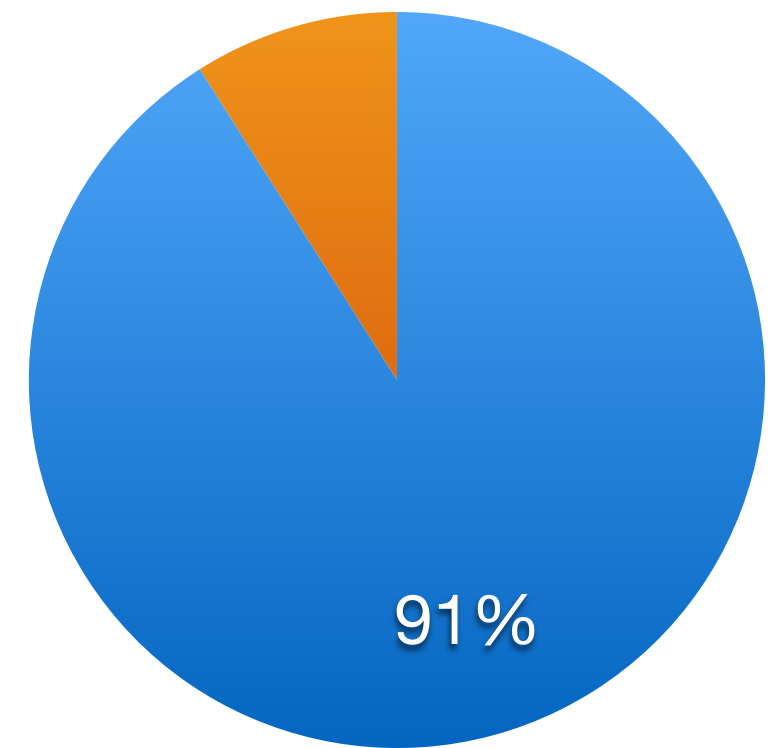


(show setting up GitHub -> Zenodo link)

Modern science and engineering research depends on software.

2009 survey: 91 % of scientists consider software “important” or “very important” to research¹.

[1] Hannay et al. (2009) How do scientists develop and use scientific software? SECSE'09. <http://dx.doi.org/10.1109/SECSE.2009.5069155>



But, 40–70% of software used is not cited^{2,3}.

[2] Pan et al. (2015) Assessing the impact of software on science: A bootstrapped learning of software entities in full-text papers. *J Informetrics* 9:860–71. <https://doi.org/10.1016/j.joi.2015.07.012>

[3] Howison et al. (2016) Software in the scientific literature: Problems with seeing, finding, and using software mentioned in the biology literature. *J Assoc Inf Sci Technol* 2016;67:2137–55. <https://doi.org/10.1002/asi.23538>

Citing software & data is important.

Our research results depend on software and data—**different versions of software and data changes our answers.**

Without proper citations,
your work is not **reproducible**.

Also, academia relies on citations for **credit**.
(for better or worse)

Software citation principles



Software citation principles

Arfon M. Smith^{1,*}, Daniel S. Katz^{2,*}, Kyle E. Niemeyer^{3,*} and
FORCE11 Software Citation Working Group

¹ GitHub, Inc., San Francisco, California, United States

² National Center for Supercomputing Applications & Electrical and Computer Engineering
Department & School of Information Sciences, University of Illinois at Urbana-Champaign,
Urbana, Illinois, United States

³ School of Mechanical, Industrial, and Manufacturing Engineering, Oregon State University,
Corvallis, Oregon, United States

* These authors contributed equally to this work.

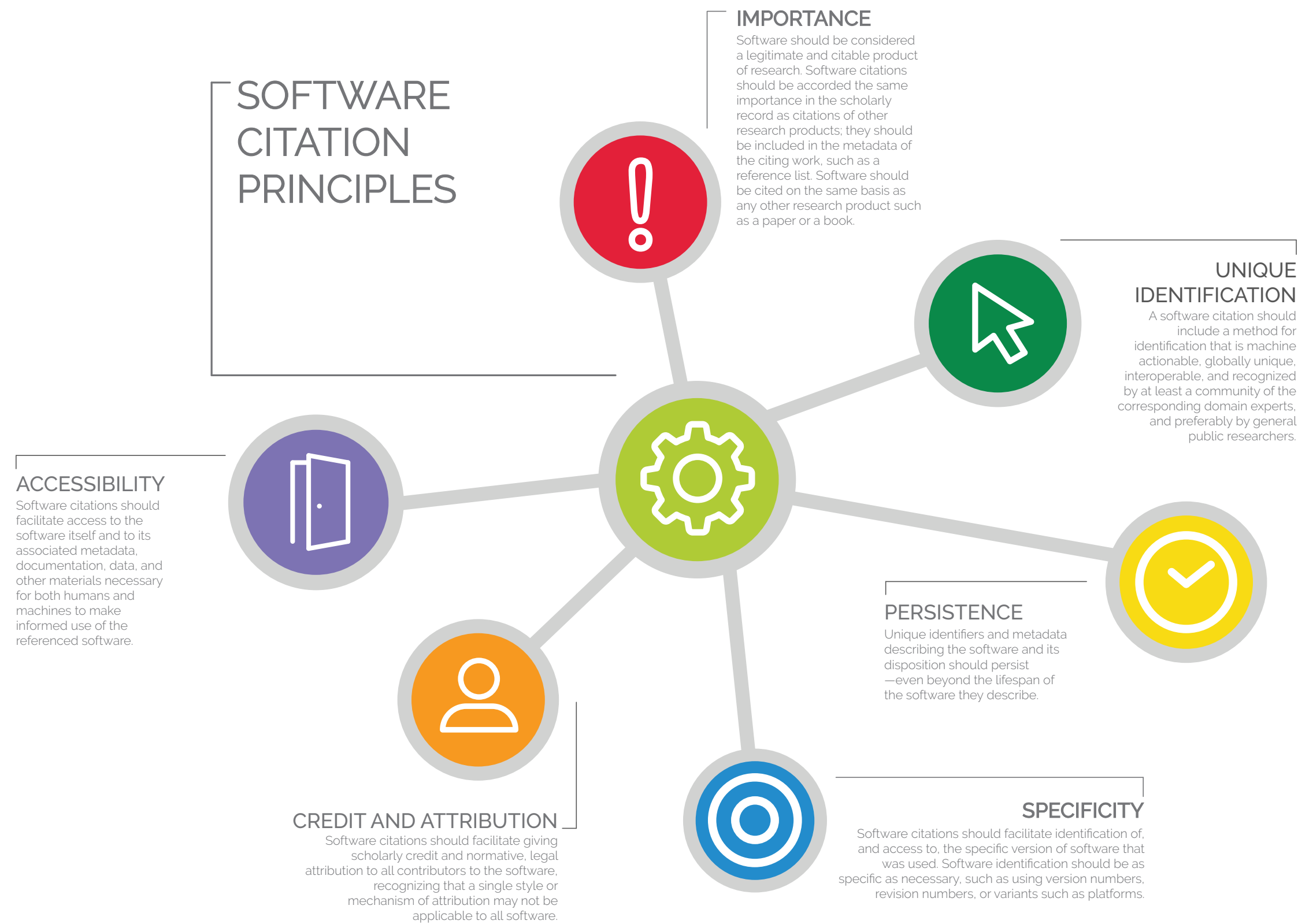
ABSTRACT

Software is a critical part of modern research and yet there is little support across the scholarly ecosystem for its acknowledgement and citation. Inspired by the activities of the FORCE11 working group focused on data citation, this document summarizes the recommendations of the FORCE11 Software Citation Working Group and its activities between June 2015 and April 2016. Based on a review of existing community practices, the goal of the working group was to produce a consolidated set of citation principles that may encourage broad adoption of a consistent policy for software citation across disciplines and venues. Our work is presented here as a set of software citation principles, a discussion of the motivations for developing the principles, reviews of existing community practice, and a discussion of the requirements these principles would place upon different stakeholders. Working examples and possible technical solutions for how these principles can be implemented will be discussed in a separate paper.

Smith AM, Katz DS, Niemeyer KE, FORCE11 Software Citation Working
Group. (2016) Software citation principles. PeerJ Computer Science
2:e86 <https://doi.org/10.7717/peerj-cs.86>



Software is a critical part of modern research...



... yet there is little support for its acknowledgement and citation

Principles

1. **Importance:** software as important as other research products
2. **Credit & attribution:** citations should facilitate scholarly credit and attribution to all contributors
3. **Unique identification:** citation should include machine actionable, globally unique, interoperable, and recognized identification method
4. **Persistence:** Unique identifiers and metadata should persist
5. **Accessibility:** Citations should facilitate access to software and associated metadata
6. **Specificity:** Citations should facilitate identification of, and access to, specific version of software used

How to cite?

Name/description

Author(s)/developer(s)

DOI or other unique/persistent identifier

Location (e.g., GitHub repo)

How to cite?

Name/description

Author(s)/developer(s)

Version number/commit hash

Location (e.g., GitHub repo)

(If there's a paper describing it, cite that *too*)

Smith AM, Katz DS, Niemeyer KE, FORCE11
Software Citation Working Group. (2016) Software
citation principles. *PeerJ Computer Science* 2:e86
<https://doi.org/10.7717/peerj-cs.86>

Starr J, et al. (2015) Achieving human and
machine accessibility of cited data in scholarly
publications. *PeerJ Computer Science* 1:e1
<https://doi.org/10.7717/peerj-cs.1>

Where to cite?

In the text with the references/bibliography.

The next three sections explain the implementation of each primary module. PyTeCK also includes the module `detect_peaks`, based on the work of Duarte [Duarte2015], for detecting peaks in targeted quantities (e.g., pressure, temperature) to determine the ignition delay time. Supporting modules in PyTeCK include `exceptions` for raising exceptions while reading YAML files, `utils` that initializes a single Pint-based unit registry [Grecco2016], and `validation` that provides quantity validation functions.

PyTeCK relies on well-established scientific Python software tools. These include NumPy [vanderWalt2011] for large array manipulation, SciPy [Jones2001] for interpolation, Pint [Grecco2016] for interpreting and converting between units, PyTables [Alted2002] for HDF5 file manipulation, Cantera [Goodwin2016] for chemical kinetics, and `pytest` [Krekel2016] for unit testing. Travis-CI [Travis2016] also provides continuous integration testing.

The Python [59] package `pyJac` implements the methodology for producing analytical Jacobian matrices described in the previous sections, which we released openly online [60] under the MIT license. `pyJac` requires the Python module NumPy [61]. The modules used to test the correctness and performance of `pyJac` are included in this release, and additionally require Cython [62], Cantera [54], PyYAML [63], and Adept [64]; however, these are not required for Jacobian/rate subroutine generation. In addition, interpreting Cantera-format models [54] requires installing the Cantera module for any purpose, while `pyJac` includes native support for interpreting Chemkin-format models [53].

KE Niemeyer, “PyTeCK: a Python-based automatic testing package for chemical kinetic models”.

Proceedings of SciPy 2016.

http://conference.scipy.org/proceedings/scipy2016/kyle_niemeyer.html

KE Niemeyer, NJ Curtis, & CJ Sung. “pyJac: analytical Jacobian generator for chemical kinetics” (2017)

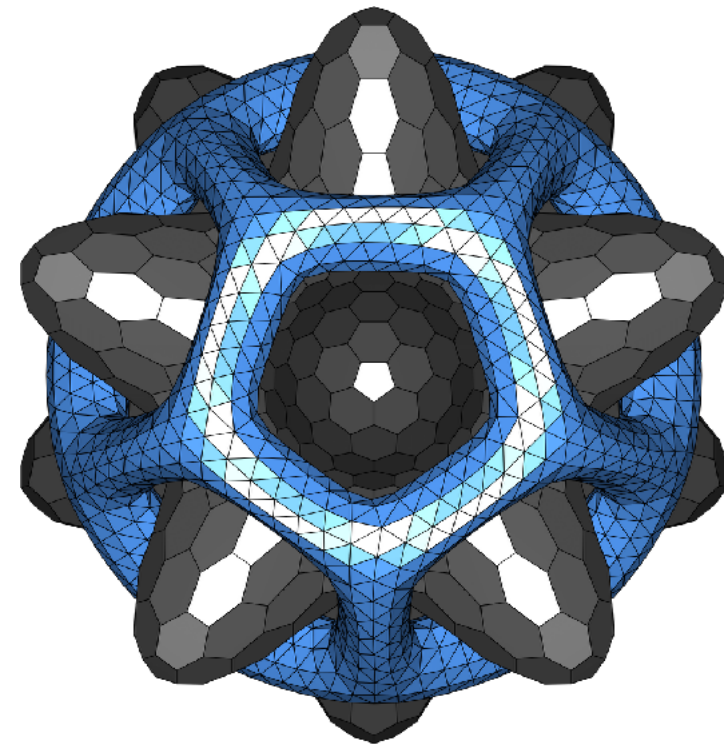
Computer Physics Communications, 215:188–203.

<https://doi.org/10.1016/j.cpc.2017.02.004>

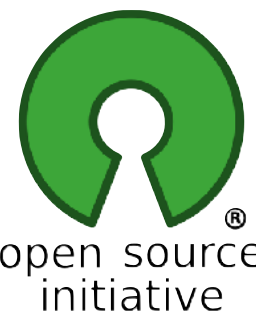
JOSS: Journal of Open Source Software

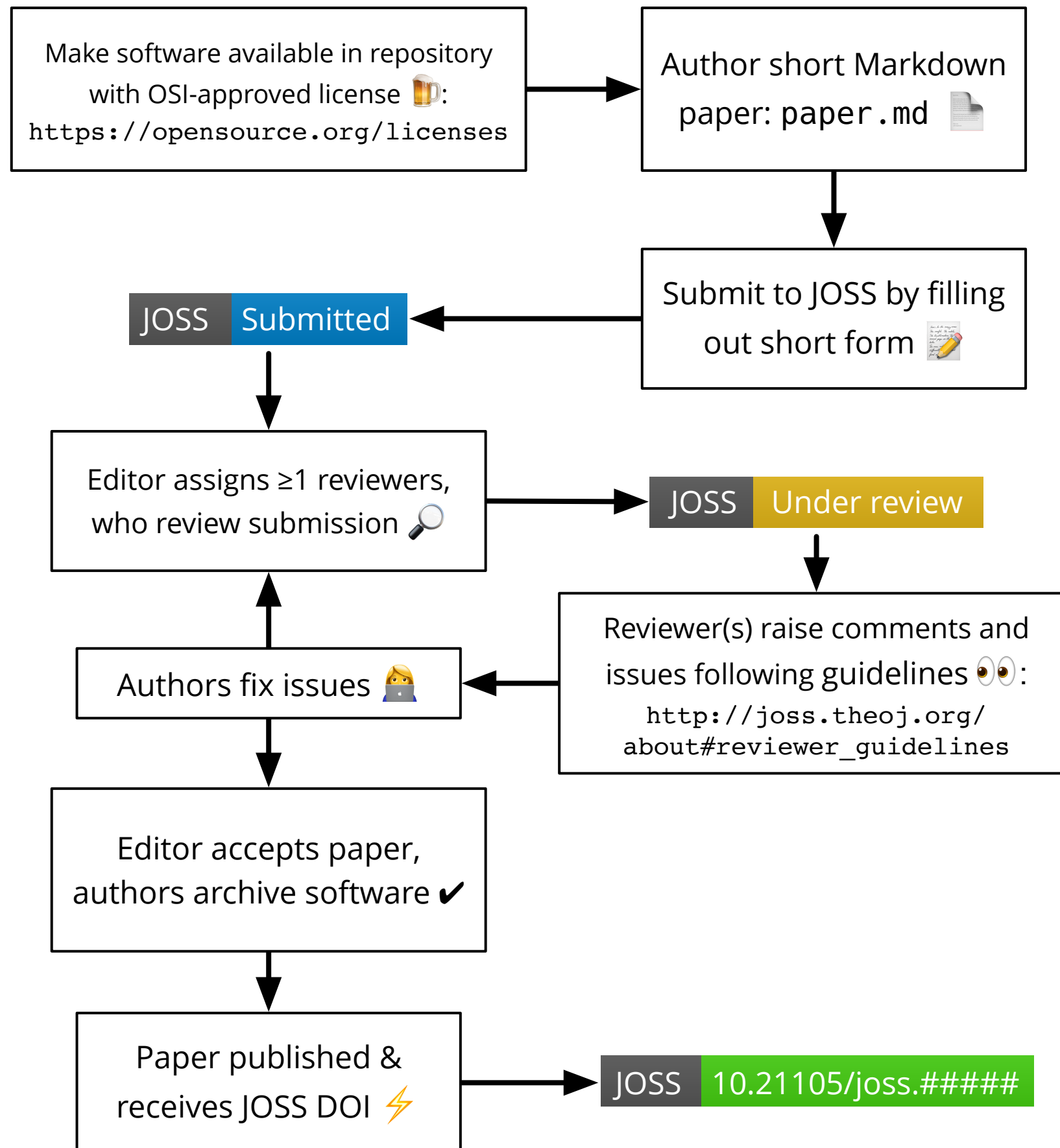
- <http://joss.theoj.org/>

- “Developer-friendly journal for research software packages”
- Affiliate of Open Source Initiative
- Open access, no fees



“If you've already licensed your code and have good documentation then we expect that it should take less than an hour to prepare and submit your paper to JOSS.”





The Journal of Open Source Software

SubmitPapersAboutArfon Smith · Sign out

Submit software for review

Before you submit

Please make sure you've read the [submission instructions](#) before submitting. In particular please make sure there is a `paper.md` present in your repository that is structured [like this](#). We promise this will make things go much more quickly during the review process 🚀

Title

Repository address

Software version

Suggested editor. [View editors here »](#)

Suggested editor


Description

Please give short (1-2 line) description of your software.

☐ I certify that I am submitting software for which I am a primary author

☐ I confirm that I read and will adhere to the JOSS [code of conduct](#)

Submit paper

The Journal of Open Source Software is
an affiliate of the [Open Source Initiative](#).

© The Journal of Open Source Software

JOSS paper submission

Issues · openjournals/joss-reviews

GitHub, Inc. [US] https://github.com/openjournals/joss-reviews/issues

This repository Search Pull requests Issues Gist

openjournals / joss-reviews Watch 31 Star 53 Fork 0

Code Issues 44 Pull requests 0 Projects 0 Pulse Graphs Settings

Filters is:issue is:open Labels Milestones New issue

44 Open 176 Closed Author Labels Projects Milestones Assignee Sort

- [REVIEW]: Brightway: An open source framework for life cycle assessment review #236 opened a day ago by whedon 0 of 17
- [REVIEW]: Text detection in screen images with a Convolutional Neural Network review #235 opened 4 days ago by whedon 0 of 17
- [REVIEW]: blogo/ncbi: interfaces to NCBI services for the Go language review #234 opened 6 days ago by whedon 0 of 17
- [REVIEW]: nails: Network Analysis Interface for Literature Studies review #233 opened 7 days ago by whedon 14 of 17
- [REVIEW]: libqcpp: A C++14 sequence quality control library review #232 opened 7 days ago by whedon 0 of 17
- [REVIEW]: flusight review #231 opened 8 days ago by whedon 0 of 17
- [PRE REVIEW]: qGaussian: An R package that deal with the Tsallis statistics pre-review #229 opened 9 days ago by whedon
- [REVIEW]: cbcbeat: an adjoint-enabled framework for computational cardiac electrophysiology review #224 opened 10 days ago by whedon 0 of 17

JOSS paper reviews

[REVIEW]: hdbscan: A high performance implementation of HDBSCAN* clustering. #205

Closed whedon opened this issue on 13 Mar · 17 comments

whedon commented on 13 Mar • edited

Submitting author: @lmcinnes (Leland McInnes)
Repository: <https://github.com/scikit-learn-contrib/hdbscan>
Version: v0.8.8
Editor: @danielskatz
Reviewer: @zhaozhang
Archive: [10.5281/zenodo.401403](https://zenodo.org/record/401403)

Status

JOSS 10.21105/joss.00205

Status badge code:

```
HTML: <a href="http://joss.theoj.org/papers/b5c5dd4b7491890b711c06225dcc9649">
Markdown: [[status]](http://joss.theoj.org/papers/b5c5dd4b7491890b711c06225dcc9649/status.s
```

Reviewers and authors:

Assignees: No one—assign yourself

Labels: **accepted**, review

Projects: None yet

Milestone: No milestone

Notifications: **Unsubscribe**

You're receiving notifications because you modified the

JOSS paper review

Journal of Open Research Software (JORS)

- Open access, peer-reviewed journal for “software metapapers” describing research software with high reuse potential
- Also publishes papers on aspects of creating, maintaining, and evaluating open-source software
- Fee: £100 (waivable)



<http://openresearchsoftware.metajnl.com/>



Let's talk papers.

Traditional journal publication

- Most traditional journals are only available via subscription, or per-article basis (\$10–100 each).
- Universities pay beaucoup \$\$\$ (on the order of millions) for these subscriptions, but some universities are starting to refuse.
- Many people can't access these articles: other universities, industry, researchers around the world, the public.
- Many funders are beginning to require public access to articles they support.
- (Also, academic publishers make boatloads, with their content given to them for free. What gives??)

<https://www.theguardian.com/science/2017/jun/27/profitable-business-scientific-publishing-bad-for-science>

<https://www.timeshighereducation.com/news/elseviers-profits-swell-more-ps900-million>

Types of open access



Green OA
self archiving



Gold OA
open access journal

Gold OA

Either fully open journal (good)
or hybrid (bad).

Typically (though not always) both
require **article processing charge**.

OSU pays for some, like *PeerJ/PeerJ CS*.

Green OA

Meaning: publish in traditional (closed) venue, but also make available openly.

Where? eprint/preprint archives

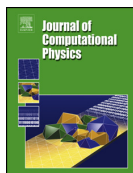




Contents lists available at ScienceDirect

Journal of Computational Physics

www.elsevier.com/locate/jcp



Accelerating moderately stiff chemical kinetics in reactive-flow simulations using GPUs

Kyle E. Niemeyer^{a,b,*}, Chih-Jen Sung^b^a Department of Mechanical and Aerospace Engineering, Case Western Reserve University, Cleveland, OH 44106, USA^b Department of Mechanical Engineering, University of Connecticut, Storrs, CT 06269, USA

ARTICLE INFO

Article history:

Received 11 May 2013

Received in revised form 22 August 2013

Accepted 13 September 2013

Available online 25 September 2013

Keywords:

Reactive-flow modeling

GPU

Chemical kinetics

Stiff chemistry

CUDA

ABSTRACT

The chemical kinetics ODEs arising from operator-split reactive-flow simulations were solved on GPUs using explicit integration algorithms. Nonstiff chemical kinetics of a hydrogen oxidation mechanism (9 species and 38 irreversible reactions) were computed using the explicit fifth-order Runge–Kutta–Cash–Karp method, and the GPU-accelerated version performed faster than single- and six-core CPU versions by factors of 126 and 25, respectively, for 524,288 ODEs. Moderately stiff kinetics, represented with mechanisms for hydrogen/carbon-monoxide (13 species and 54 irreversible reactions) and methane (53 species and 634 irreversible reactions) oxidation, were computed using the stabilized explicit second-order Runge–Kutta–Chebyshev (RKC) algorithm. The GPU-based RKC implementation demonstrated an increase in performance of nearly 59 and 10 times, for problem sizes consisting of 262,144 ODEs and larger, than the single- and six-core CPU-based RKC algorithms using the hydrogen/carbon-monoxide mechanism. With the methane mechanism, RKC-GPU performed more than 65 and 11 times faster, for problem sizes consisting of 131,072 ODEs and larger, than the single- and six-core RKC-CPU versions, and up to 57 times faster than the six-core CPU-based implicit VODE algorithm on 65,536 ODEs. In the presence of more severe stiffness, such as ethylene oxidation (111 species and 1566 irreversible reactions), RKC-GPU performed more than 17 times faster than RKC-CPU on six cores for 32,768 ODEs and larger, and at best 4.5 times faster than VODE on six CPU cores for 65,536 ODEs. With a larger time step size, RKC-GPU performed at best 2.5 times slower than six-core VODE for 8192 ODEs and larger. Therefore, the need for developing new strategies for integrating stiff chemistry on GPUs was discussed.

© 2013 Elsevier Inc. All rights reserved.

1. Introduction

The heavy computational demands of high-fidelity computational fluid dynamics (CFD) simulations, caused by fine grid resolutions and time step sizes in addition to complex physical models, are the primary bottleneck preventing most industrial and academic researchers from performing and using such studies. Reactive-flow simulations considering detailed chemistry in particular pose prohibitive computational demands due to (1) chemical stiffness, caused by rapidly depleting species and/or fast reversible reactions, and (2) the large and ever-increasing size of detailed reaction mechanisms. While reaction mechanisms for fuels relevant to hypersonic engines, such as hydrogen or ethylene, may contain 10–70 species [1,2], a recent surrogate mechanism for gasoline consists of about 1550 species and 6000 reactions [3]; a surrogate mechanism for biodiesel contains almost 3300 species and over 10,000 reactions [4]. Strategies for incorporating such large, realistic

* Corresponding author. Current address: School of Mechanical, Industrial, and Manufacturing Engineering, Oregon State University.
E-mail addresses: Kyle.Niemeyer@oregonstate.edu (K.E. Niemeyer), cjsung@engr.uconn.edu (C.-J. Sung).

Accelerating moderately stiff chemical kinetics in reactive-flow simulations using GPUs

Kyle E. Niemeyer^{a,b,1,*}, Chih-Jen Sung^b^a Department of Mechanical and Aerospace Engineering
Case Western Reserve University, Cleveland, OH 44106, USA^b Department of Mechanical Engineering
University of Connecticut, Storrs, CT 06269, USA

Abstract

The chemical kinetics ODEs arising from operator-split reactive-flow simulations were solved on GPUs using explicit integration algorithms. Nonstiff chemical kinetics of a hydrogen oxidation mechanism (9 species and 38 irreversible reactions) were computed using the explicit fifth-order Runge–Kutta–Cash–Karp method, and the GPU-accelerated version performed faster than single- and six-core CPU versions by factors of 126 and 25, respectively, for 524,288 ODEs. Moderately stiff kinetics, represented with mechanisms for hydrogen/carbon-monoxide (13 species and 54 irreversible reactions) and methane (53 species and 634 irreversible reactions) oxidation, were computed using the stabilized explicit second-order Runge–Kutta–Chebyshev (RKC) algorithm. The GPU-based RKC implementation demonstrated an increase in performance of nearly 59 and 10 times, for problem sizes consisting of 262,144 ODEs and larger, than the single- and six-core CPU-based RKC algorithms using the hydrogen/carbon-monoxide mechanism. With the methane mechanism, RKC-GPU performed more than 65 and 11 times faster, for problem sizes consisting of 131,072 ODEs and larger, than the single- and six-core RKC-CPU versions, and up to 57 times faster than the six-core CPU-based implicit VODE algorithm on 65,536 ODEs. In the presence of more severe stiffness, such as ethylene oxidation (111 species and 1566 irreversible reactions), RKC-GPU performed more than 17 times faster than RKC-CPU on six cores for 32,768 ODEs and larger, and at best 4.5 times faster than VODE on six CPU cores for 65,536 ODEs. With a larger time step size, RKC-GPU performed at best 2.5 times slower than six-core VODE for 8192 ODEs and larger. Therefore, the need for developing new strategies for integrating stiff chemistry on GPUs was discussed.

Keywords: Reactive-flow modeling, GPU, Chemical kinetics, Stiff chemistry, CUDA

1. Introduction

The heavy computational demands of high-fidelity computational fluid dynamics (CFD) simulations, caused by fine grid resolutions and time step sizes in addition to complex physical

*Corresponding author

E-mail addresses: Kyle.Niemeyer@oregonstate.edu (Kyle E. Niemeyer), cjsung@engr.uconn.edu (Chih-Jen Sung)

¹Present address: School of Mechanical, Industrial and Manufacturing Engineering, Oregon State University, Corvallis, OR 97331

Preprint submitted to Journal of Computational Physics

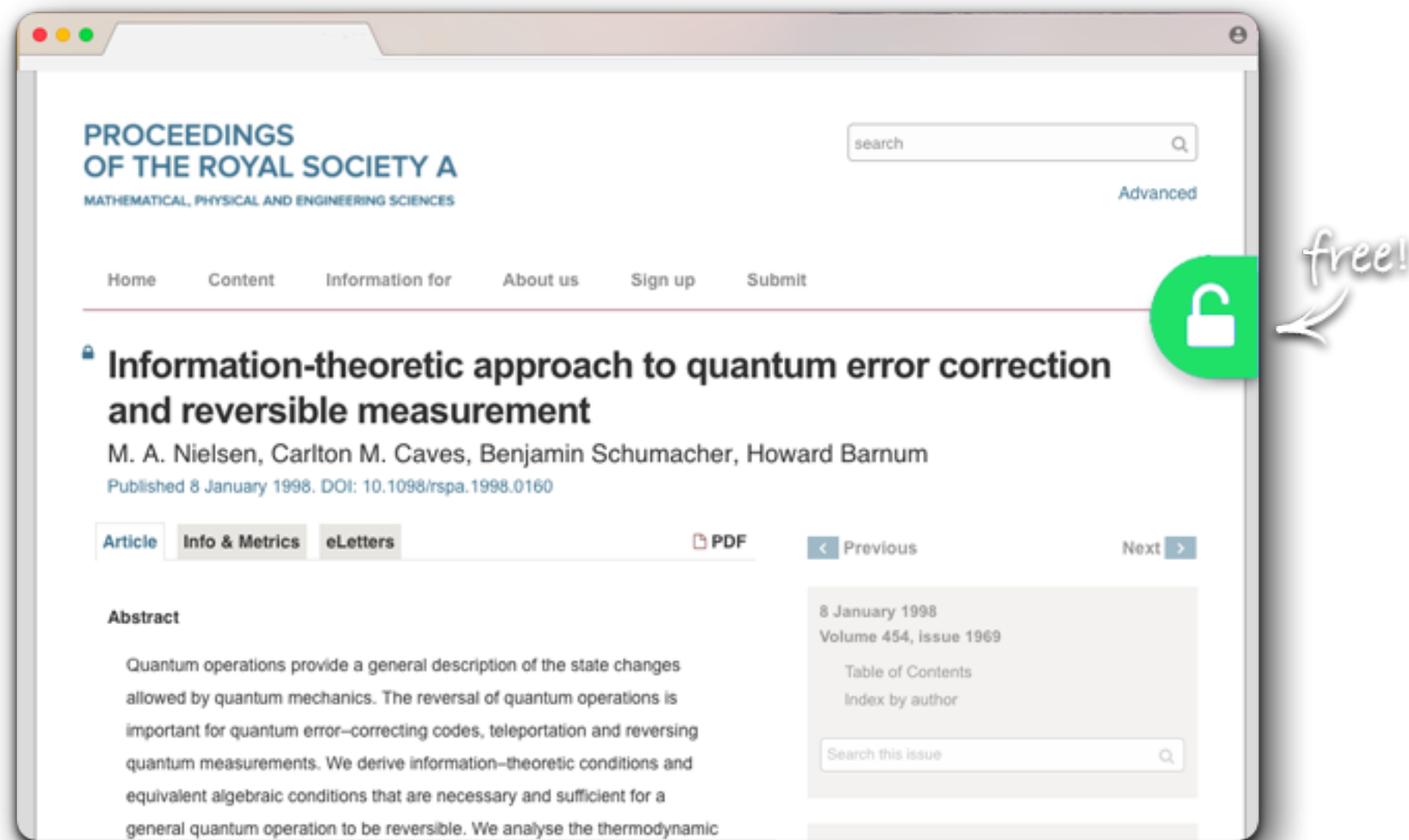
November 5, 2013

arXiv:1309.2710v2 [physics.comp-ph] 4 Nov 2013

Journal version

Author-accepted
version

Unpaywall browser extension



Automagically finds open-access version of paywalled journal articles.

What about data?

You should also share data

- If your study generates (or relies on) data, that should also be archived and cited in your papers.
- We have great resources now to easily—and freely—upload huge datasets.
- The goal should be for someone else to be able to download your data and software and reproduce your paper.

zenodo



“Repro-Pack”

Prof. Lorena Barba describes “reproducibility packages” associated with papers, with figures shared under CC-BY license:

“For every figure that presents some result, we bundle the files needed to reproduce it — input or configuration files used to run the simulation(s) behind the result; code to process raw data into derived data; and scripts to create output graphs — and deposit them together with the figure into an open-data repository, such as [Figshare](#). Figshare assigns the bundle a DOI, which we then include in the figure caption so readers can easily find the data and re-create the result. Our lab uses these packages as test beds for our in-house software, to verify that the results haven’t been compromised by software modifications. And because we maintain a public history of all changes, we achieve what one of my students calls ‘unimpeachable provenance’.”

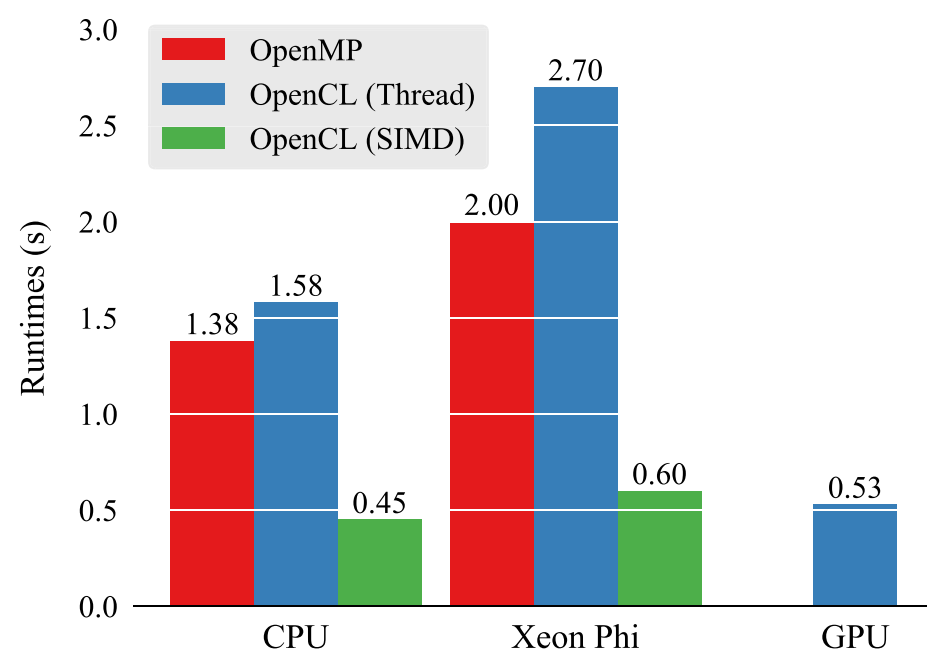
My practice

- Produce a single “repro-pack” for an entire paper, which contains:
 - Python plotting scripts and associated results data
 - Figures (PDFs for plots, always)
 - Any other relevant data: input files, configuration files, etc.
- Upload to Figshare/Zenodo under CC-BY license
- Cite using the resulting DOI in the associated paper(s)

Benefits

- Improving reproducibility and impact of your work
- Reviewers will love you with this one great trick!
- It also lets you reuse your figures without violating the journal copyright. (Yes, when published, the journal owns the paper and everything in it that isn't licensed from somewhere else.)

How to cite/mention

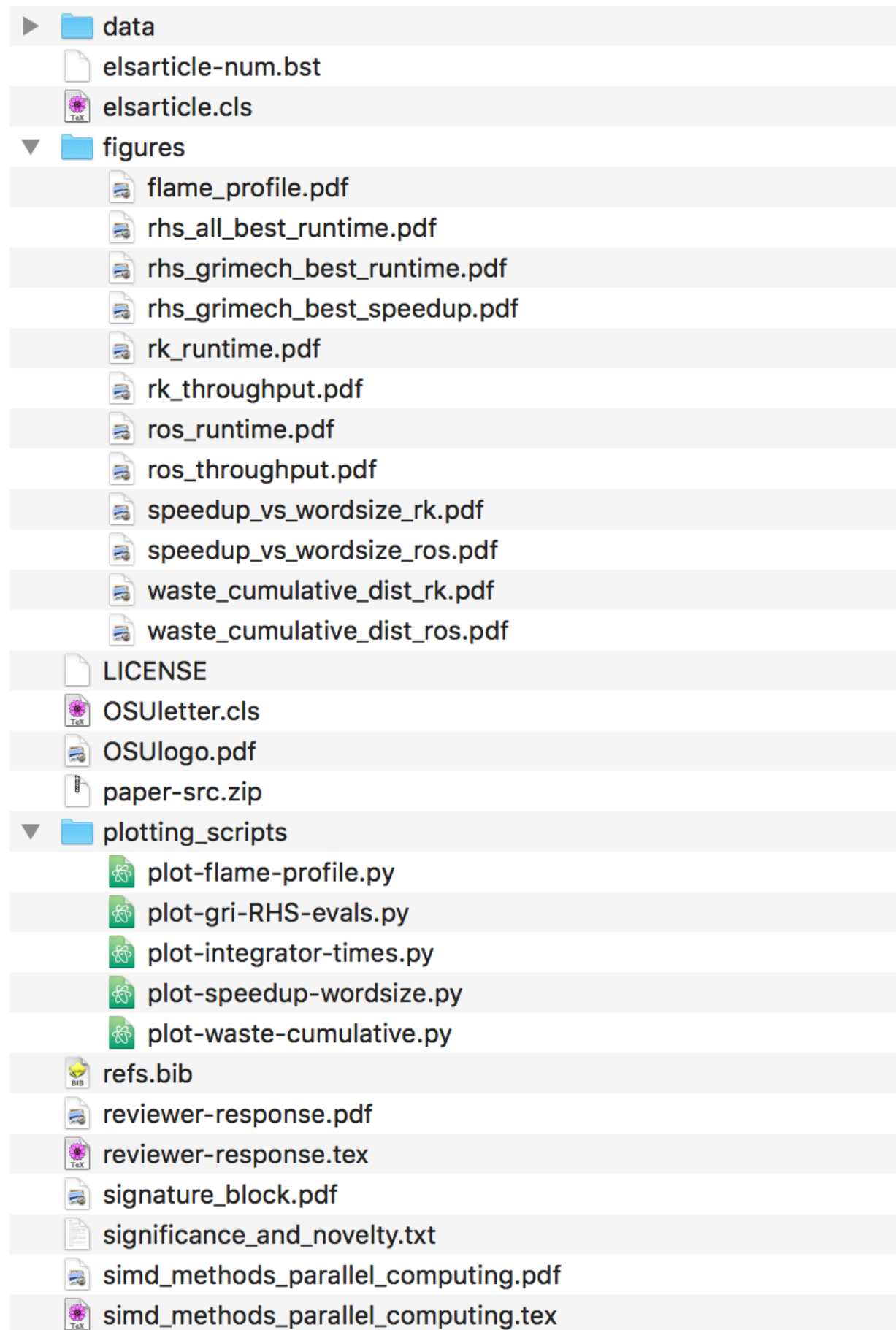


Appendix A. Availability of material

The integrators used to perform this study are available openly via the `accelerInt` software package [50]. The most recent version of `accelerInt` can be found at its GitHub repository: <https://github.com/SLACKHA/accelerInt>. All figures, and the data and plotting scripts necessary to reproduce them, are available openly under the CC-BY license [41].

Fig. 1. Wall-clock runtimes (seconds) for one million RHS evaluations of the GRI Mech 3.0 model on the host CPU, MIC, and Kepler GPU using the OpenCL SIMD (CL-SIMD) and thread-parallel (CL-Thread) implementations. OpenMP uses 16 threads with compiler auto-vectorization. Data, plotting scripts, and figure file are available [41].

- [41] C.P. Stone, A.T. Alferman, K.E. Niemeyer, Data, plotting scripts, and figures for “Accelerating finite-rate chemical kinetics with coprocessors: comparing vectorization methods on GPUs, MICs, and CPUs”, Figshare, 2017. <http://dx.doi.org/10.6084/m9.figshare.5353183>.



```

from __future__ import print_function, division
import sys
import os.path
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.backends.backend_pdf import PdfPages
from matplotlib.ticker import ScalarFormatter
from cycler import cycler

import brewer2mpl

home_dir = os.path.join(sys.path[0], '../')
home_dir = os.path.realpath(home_dir)

figures_dir = os.path.join(home_dir, 'figures')
data_dir = os.path.join(home_dir, 'data')

font = {'weight' : 'normal',
        'size' : 16,
        'family' : 'serif',
        'serif' : ['Times New Roman']}
plt.rc('font', **font)
axis_font = {'fontname':'Times New Roman', 'size':'16'}
leg_font = {'family':'serif', 'size':'13'}

...

# get colors that show qualitative differences
colors = brewer2mpl.get_map('Set1', 'qualitative', 3).mpl_colors

fig, ax = plt.subplots()

# Change the default colors
ax.set_prop_cycle(cycler('color', colors) + cycler('lw', [2, 2, 2]))

...

# save to file
pp = PdfPages(os.path.join(figures_dir, 'flame_profile.pdf'))
pp.savefig()
pp.close()

```

Ideal computational workflow

1. If software project, create repo on GitHub
2. Write software (easy!), then test & validate
3. Do your research, generating data
4. Write scripts to create plots out of data
5. Write paper about data and plots
6. Archive version of software used in paper, and cite
7. Archive data generated by software, and cite
8. Archive figures and scripts, and cite
9. Upload paper to appropriate eprint server
10. Submit to journal!

Example: pyJac

Creates C and CUDA analytical Jacobian matrices for chemical kinetics ODE systems

- Source code at github.com/SLACKHA/pyJac has README with basic usage
- Full documentation website with API docs, installation guide, and examples: slackha.github.io/pyJac/
- Functional and performance testing suites built-in
- Software paper published with full theory details (doi.org/10.1016/j.cpc.2017.02.004)
- full source of paper also available via [niemeyer-research-group.github.io/pyJac-paper/](https://github.com/niemeyer-research-group/pyJac-paper/)
- Data, figures, and figure scripts from paper available openly via Figshare: <https://doi.org/10.6084/m9.figshare.4578010>