We thank the anonymous reviewer for taking the time to read this paper thoroughly and providing the authors with constructive and thoughtful feedback. Addressing this feedback has greatly improved the paper.

Addressing Reviewer #1's concerns:

Title: Best practice regarding the three P’s: profiling, portability and provenance when running HPC geoscientific applications
Author(s): Wendy Sharples et al.
This paper presents a utility for controlling the execution and initial evaluation of an application (the ParFlow model) running in a (primarily) HPC framework. There are two levels to the framework described: the "run control framework" (or RCF) which itself utilises a more generic JUBE benchmark framework as a workflow engine. Essentially these provide a method of systematically defining, running and analysing some benchmarks - the authors also suggest it would be suitable for use for production simulations as well.
The framework isolates multiple runs using parameters configured at set up time and keeps all the data produced along with a set of reports, allowing parameter sweeps with automated isolation of the various results (termed "provenance tracking" in the paper).
A case study is presented utilising the framework to examine weak scaling (increasing the domain size) for the ParFlow model.

This paper is a difficult read, because there are three different strands within it: motivation, tooling, and the results of using the tools. They are well isolated by the sections, but each is somewhat unsatisfying on its own, and the links between are not as strong as I would like to see in a GMD journal paper. As it stands I do not think it is fit for full publication in GMD, but I think it could be made so with some reworking to make the material more accessible and relevant to the GMD audience.

There is some good material in the motivation, but it falls uncomfortably between being either a complete description of the portability, performance and reproducibility issues associated with geo-scientific modelling or an introduction to those elements for which the tools discussed later are either well suited or applied. It would be stronger if it were the latter.

Our aim was to introduce those elements for which our RCF is a solution. We have therefore provided a better and more complete explanation of exactly what the RCF is, what its capabilities and functions are and how it is a best practice approach to aid portability, profiling and provenance tracking.

The discussion of the tooling itself is incomplete in important details, yet full of detail (like the XML files in the appendix) which cannot be easily consumed by the reader because of a lack of appropriate explanation. There is no discussion of why this tool is any different from any other tool (e.g. what are the strengths and weaknesses with respect to the two workflow tools introduced in section 1)?

This was an oversight on our part. We have now included a discussion of the relevant strengths and weaknesses of JUBE, ecFlow and cylc and have stated the reasons for choosing JUBE and that the RCF could be adapted for other workflow engines. We have also tried to better describe what the RCF is separate from the workflow engine used (JUBE) and how they are integrated with each other.

The results of the analysis of ParFlow are probably the strongest and most interesting parts of the paper, but because of the layout of the paper, introductory material such as the definitions of load balance, are mixed in with results and interpretation. I would rather this section had been organised to correspond to the (very useful) list of "health checks" which begins on the bottom of page 6. It might have been that the relevant definitions (equations 1 to 4) could have appeared in section 2, since that's where these issues are first introduced. The results and scientific consequences could then be clearly identified in 3.4.

Thanks for this great suggestion. We have moved these definitions up to section 2, and mapped each health check step back to the results presented and have provided a more in depth description of the profiling tools used and the function of the RCF in automating the health check workflow.

Addressing the specific comments:

1. I do not believe the title fairly reflects the material of the paper. The paper is not about best practice, it is primarily about one workflow/benchmarking application, although it does list elements of best practice and motivate some of the issues.

We wanted to present the RCF as a best practice approach, but we accept that the title does not totally reflect the material presented. We have updated the title to: A run control framework to streamline profiling, porting and tuning, simulation runs and provenance tracking of geoscientific applications

2. The paper begins with motivation with a selective list of references for how increased HPC might be used. The list is somewhat different from that usually presented which normally now includes increased use of data assimilation alongside increasing complexity, domain (spatial or temporal), resolution and ensemble size. It would be good to see this list inclusive of data assimilation and temporal extent and without quite so many references which don't add much value (there are so many it appears to be an attempt to be exhaustive, but it is clearly not exhaustive, better to have few or no exemplars than three or four each, because one is left wondering "why *these* ones"?).

This is a good point. We have updated the reference lists so that there are no more than 3 and e.g. is used more often to indicate the references are a subset of a broader scope of research (1st paragraph)

3. There is then some material on the upcoming difficulties with performance portability which adds to the motivation, but the implication is that these are issues which can be solved by optimisation. In particular the paragraph beginning on line4 of page 3 begins by recognising the massive investments required to get performance, then implies that this investment can leverage analysis of existing codes using benchmarking tools such as the RCF/JUBE one discussed here. I think this section would be stronger if there was a clean separation between the aspirations of parameter sweeping and performance analysis, which is primarily about optimisation, and that of massive structural reorganisation of code such as was involved in Leutwyler et al. This is not to denigrate the importance of the former, but just to realistically recognise the scope. As written, the paper overstates it.

Yes this is true. We meant to say that the RCF can streamline the investment as bottlenecks can be quickly identified because it has the relevant profiling tools integrated within it to automate the performance engineering approach. We have tried to clarify/add to this explanation in the paragraph mentioned in page 3 and we refer back to this automation in section 3.

4. In the context of scope it would also be useful to identify where the tool might have significant limitations, e.g, where it could interfere with other configuration and workflow managers (or be interfered with). This is not to suggest that the tool is not useful, or even powerful, for a particular class of problems; just that like all solutions, it almost certainly has limited applicability. It would be a service for potential users for

the authors to provide some clarity on any known scope issues.

Good suggestion. In the section on JUBE 2.1 paragraph 2 discussion of limitations have been added to and also what is common between JUBE, cylc and ecFlow is stated. In addition the advantages/disadvantages of each workflow engine is discussed along with an explanation of why we have chosen JUBE.

5. I think the paper confuses key issues around reproducibility. The implication of the discussions about reproducibility on page 3 and section 3.3 is that "if only the relevant parameters were documented, simulations would be reproducible". While this is undoubtedly *necessary*, it is far from *sufficient*, Baker et al. doi.org/10.5194/gmd-8-2829-2015 discuss the issue of ensuring that the science remains the same when hardware and software environments change. This paper would be stronger for identifying the distinction between these different issues of reproducibility and linking them to Irving's discussion and prior literature.

Our RCF actually goes much further than just documenting the relevant parameters. The whole simulation can be rerun as the code, the forcing data, the environment: source file with modules to be loaded, including a list of dependencies their versions, all extraneous scripts including job submission scripts and a complete model description are bundled together. If a user were to untar an output directory, they would be able to compile and rerun the experiment with JUBE using the RCF XML file in the directory, on the same machine and obtain the same results. We have improved the description on page 3 and section 3.3. Also we have mentioned that there are some things beyond the users control such as different hardware and compilers when porting models and ways to ensure the science remains the same.

6. This might address the issue that there is only a cursory discussion of the issues associated with porting ParView to JUQUEEN - indeed, one might have expected the use of this framework to help with that process. "It was found that the optimal flags which did not compromise accuracy" with accuracy determined "to six figures". This reviewer has no idea what they meant by "accuracy" in this context, and the cursory argument suggests that important issues around solution stability were not investigated (despite the motivation being reproducibility).

Yes the test was not very well explained. It was briefly explained in section 2.3. We have expanded this section and made reference to this section in section 3.2.

7. In the discussion of the tools itself, the overall workflow is well described (Figure 1 etc and the excellent list provided for the "health examination"), but the discussion of the tool provides names of files, and then exemplar files (in XML, in the appendices). It's not clear at all how and why this framework is better than a bunch of scripts with input files - it would be considerably stronger if there was some discussion of how the tool exploits the XML files (is there a semantic structure inherent in the files beyond that provided by the use of XML to control syntax)? It might be that this is what the description JUBE reference provides, but I was unable to access the description of JUBE hidden behind a paywall. Some kind of discussion about how the XML content links to JUBE actions would be helpful. In any case, I recommend removal of the XML files in the appendix, on their own they are inscrutable and provide little value. However, if they were provided in a repo with documentation as to function, they would provide useful complementary material.

Good suggestion. We have moved the description of the RCF from the appendix to section 2 to help provide a fuller understanding of exactly what it is. We have tried to emphasize the links between the XML content and JUBE actions in section 2.2 to explain how the RCF builds the overall configuration XML file to be run by JUBE. The xml files are provided in a tarred up directory submitted with this manuscript. There are readme files plus documentation in each script as to what it's functionality is. To that end we have removed the xml files from the appendix.

8. The bulk of the case study shows the ability of the tools to generate information to understand the performance of the ParView model on this platform, and introduces some of the plans to alleviate performance bottlenecks (such as Adaptive Mesh Refinement). However, I did not fully understand the argument as to why this is the obvious next step from the current arrangement where all cells know about all other cells (why)?. This piece of the argument was another place where I felt that there was blurring together in this paper around issues of performance portability (optimising for a target architecture, but not changing the science), versus algorithmic improvements in performance (which involve changing the science).

We agree that the integration with p4est was not very clear. This is a case of an algorithmic improvement which does not change the science. ParFlow coupled to p4est means that p4est is now the parallel mesh manager, currently focusing on uniform meshes. The approach was minimally invasive and preserves most of ParFlow's data structures, the configuration system and the setup and solver pipeline. We have now explained this in section 3.4 and also mentioned the state of the current mesh manager in section 3.3.

9. Somewhere in the paper there needs to be some comparison to prior art and other similar tools which may address some part of the scope of these tools. (The description of GMD Development and Technical papers states: "Development and technical papers usually include a significant amount of evaluation against standard benchmarks, observations, and/or other model output as appropriate.") While, the key word may be *usually*, in the context of *this* paper I think there should be a section similar to the "RelatedWork" section that appears in many computer science and software engineering papers covering relevant strengths and weaknesses.

There is related work in terms of scaling and profiling ParFlow, which has been reported in references Kollet et al. 2010 and Gasper et al. 2014 for example. There is a report on comparisons between workflow engines cylc and ecFlow in Manubens-Gil 2016 which is now mentioned in our manuscript. Our reasons for choosing JUBE over cylc and ecFlow are now outlined clearly in the manuscript, where the main reason for choosing JUBE is that it is the workflow engine most compatible for use on JSC machines. However the RCF itself is a new development to streamline scientific software profiling, porting and tuning, running production runs and to aid reproducibility. To date, there are no “standard” benchmarks as such.

10. Finally, I note that the code is apparently available on request, but I think it will not get significant uptake without being both open source and having an open development process. Neither are required for GMD publication, I only mention these factors to encourage the authors to build a community around what looks like a useful tool for a certain class of problems.

A tarball has been included with the manuscript which the users can use with JUBE and any version of ParFlow with the test case described in the paper straight away. This should be enough to get a new user familiarized with the RCF enough to add to it for their own purpose. The reason why the whole RCF code is not open source is because the real world models included are developed by different scientists, many of which have not yet been published.

We thank the anonymous reviewer for taking the time to read this paper thoroughly and providing the authors with constructive and thoughtful feedback. Addressing this feedback has greatly improved the paper.

Addressing reviewer 2's concerns:

The paper presents the some of the challenges of geoscientific modelling on HPC resources, with emphasis on profiling and processing workflows. In order to address provenance, portability and profiling best practices, a run control framework (RCF) based on JUBE is described. The demonstration of the RCF is conducted in a weak scaling experiment in ParFlow, an integrated watershed model. Presentation of the tests results constitutes a notable - and interesting - part of the paper.

The paper is relatively easy to read in sections, but difficult to read as a whole. It covers several domains of expertise such as HPC, geoscientific modelling, software engineering, run harness and profiling. More work is required to present a coherent and uni-formly detailed experiment in its full context. The profiling element is also substantially more detailed than the others to the point of overshadowing them. Furthermore, profiling results could be much better linked and/or mapped to the very informative “health check” section. As for layout of the “health check”, typical diagnosis tools appended for each item does not convey the specific (overlapping or not) role or features of each software very well.

The suggestion to map the results back to the health check is a good one. We have also moved definitions in section 3 up to section 2 to the health check, and mapped each health check step back to the results presented. We have provided a more in depth description of the profiling tools used and their overlap. In addition we have detailed the function of the RCF in automating the health check workflow to try and provide a coherent experiment with which to demonstrate the advantages of using the RCF.

The authors list in section 3.4 the outcomes of the profiling case study. This entices the reader to believe that profiling will also advance provenance and portability. The article presents only a few elements of future work to support the full scope of the study, at least as the title describes it. It is also difficult for a reader to consider the tests results - the most detailed element of the paper - as sufficient proof of application of best practices in profiling, portability and provenance.

Yes we can see how this might cause some confusion. We have deleted Figure 9 and also the text to surrounding the NetCDF reader/writer to limit confusion for the reader. We have also clarified that the RCF is a best practice approach to profiling, portability and provenance by altering the title and also providing a better explanation of what the RCF is and how it is integrated with the workflow engine, JUBE. We have updated the title to: A run control framework to streamline profiling, porting and tuning, simulation runs and provenance tracking of geoscientific applications

The test case experimental design for ParFlow reads like a completely disjoint section to the rest of the paper. It is only very lightly linked with previous or subsequent sections, for instance on the motivations behind that particular test case and how it leverages the RCF, the workflows or the HPC resources. It is unclear looking at the profiling results what are the implications for the test case or to ParFlow itself. This section offers a great potential to present ParFlow software and models (graphically), to show alternative test configurations or to contrast with real-world scenarios.

Thanks for these suggestions. We have now included the motivations for using this particular test case in order to illuminate bottlenecks using the health check procedure and the reasons why a “real world” test case might obscure certain results such as load balance. We have also shifted the test case model description to section 3 for better continuity.

The paper mentions two other workflow engines (p.3 33) before rapidly shifting to JUBE (p.4 25). What are the advantages, limitations, similarities of JUBE compared to these other solutions? As is, considering that the results analysis of ParFlow provided by the RCF is a large (and interesting) part of the paper, a reader might be tempted to think that JUBE was selected first and pitted against other solutions a posteriori. It might be sufficient to cite major findings for the associated publication.

This was an oversight. We have now included a discussion of the relevant strengths and weaknesses of JUBE, ecFlow and cylc and have stated the reasons for choosing JUBE but have also made it clear that the RCF could be adapted for other workflow engines and they share a lot of the same functionality.

Address specific comments

Specific Comments
1. Some sentences are too long (p.1 8-10, p.3 8-10, p.17 6-10). The messages conveyed by these long sentences is important for the coherence of the whole.

Agreed- we have fixed these sentences.

2. Figure 9 could be changed to a textual list without loss of content. Is there an overlap or interrelation between those developments? Is there development to do outside the scope of ParFlow, for instance in the run harness or workflows? What does “towards exascale” refers to exactly (cite or describe)? At what scale is the system operating at right now? What are the hurdles from terascale/petascale onto exascale that the presented work will limit or remove?

We have followed your suggestion and removed Figure 9. From the profling results it can be seen that due to memory required being greater than memory available, at around 64000 cores, ParFlow as it stands is not approaching exascale. *Exascale* computing refers to computing systems capable of at least one exaFLOPS, or a billion billion calculations per second. ParFlow coupled with p4est as the parallel mesh handler, allows ParFlow to scale to the whole Juqueen machine (petascale). We have made this clearer in section 3 and introduced the exascale concept in the same section.

3. A very large part of the relevant information related to Figure 2 is in its legend. The reader might not care very much about the screen layout of the output. The reader might be interested in numerical values for each result on some occasions, but most values aren’t described or introduced previously in the article. Most of all, the reader will most probably be interested in the metrics themselves and how they relate to profiling - or to some extend to portability or provenance, if applicable.

This is a good idea. The reader would benefit from knowing what each metric is used for. We have tried to explain what each metric means and why it might be useful in table format in appendix A2 and thus reduced the length of the figure caption.

4. Section 3.1 is very short compared to others at this level, which diminishes its impact. It does not help with readability and flow. The section would benefit from an extension of concepts or reinforcement of links with other sections. It may also be merged elsewhere.

We have altered section 2 to include a more thorough description of the accuracy test we used to test compilation flags. We have expanded section 3.1 to include a reference to this test as well as a discussion of the relevant compilation flags used for the specific IBM XL and why we focused on the two specific flags as opposed to others available.

5. Some of the claims in the article are very lightly substantiated, insufficiently nuanced or lacking

details. For instance (p.1 16 and p.19), the author claims that RCF is less time consuming and more robust, but less/more than what? Than without use of an RCF? The article concludes by claiming a more efficient use of HPC resources that
was not clearly demonstrated; it reads like this is only implied because best practices
were followed. If that is the case, it is suggested to better define and highlight these
best practices.

Yes less time consuming than running by hand or developing a series of specialized scripts which only work for one model without integrated profiling tools. We have tried to make this clearer in all sections of the manuscript where this is mentioned and have updated the conclusions accordingly.

6. The paper also mention in a few places costs notions (“invested effort”, “paid off”,
“cost in resources”) without providing any data or basic financial analysis. What was
the approximate amount invested by articles cited? How much money or energy was
saved? This comment is provided without diminishing the (substantial) presented work
or assuming that this information is essential.

We are assuming you mean the references mentioned in page 3, 2nd paragraph and have updated this to include an example (Leutwyler et al. 2016) of speedup values (~3.6) and also the invested effort required (a team of developers)

7. The paper briefly mentions hybrid and heterogeneous architectures, but do not
mention cloud computing. While a very different architecture than HPC, a reader might
interested to see if any of the work presented can be applied in cloud computing environments (worfklows? packaged code? profiling? tools? models?). Commercial
and scientific offers in HPC-as-a-service might prove an interesting option for the RCF.
Absence of cloud computing discussion is not seen here as a limitation of the article,
only a potential topic of interest.

Thanks for pointing this out. Yes the RCF could be adapted for cloud computing or a web based interface. However this would be a substantial effort for a technology that is as yet severely limited by bandwidth such that it performs substantially worse than just using one HPC system. So we think this is out of scope for what is currently available to HPC users.

8. There are also almost no mention of standards, except a brief sentence on (p.10
3-5). A reader might expect that such large scale systems with claims on portability
and provenance do indeed follow standards instead of reinventing the wheel.

A brief mention of these standards without explanation was because the authors assumed that the readers would be familiar what standards we follow (CF and CMOR Metadata Conventions) in the section on provenance tracking. We've now expanded this section to include a discussion to describe the CF conventions and their purpose. We have also now added to this section a discussion on how Irving 2016's minimum standard points 1-4 are covered by the RCF.

9. The code profiling section (p.6 6) makes it difficult to the reader to separate author
contribution - by means of the RCF - to outputs from ScoreP and Scalasa. There
is a long list of what software can “examine” those outputs. Why are those outputs
compatible with all these software? Is it a standardized file format or structured data?

This is true. We have tried to make this clearer by explaining what ScoreP, Scalasca and Cube are in the text (rather than in the appendix) and to clarify what RCF does (collect and collate the performance metrics by parsing the various reports generated by the tools). There is a standard format for ScoreP and Scalasca output which Cube can parse and read.

10. Alinea Performance Reports and Intel Vectorization Advisor are present in each
item of the health examination, but both software weren’t used. Still, they are recommended by the authors. Is the toolset of the experiment sufficient? What is the

additional insight offered by Alinea and Intel's products that the other tools can't?

The toolset used was determined by what was available on JUQUEEN. We've now explained this in section 2. There is a lot of overlap with different tools and we have now explained this in the same section. As to what tool one uses it depends on 1. what is available on a given machine and 2. personal preference. For example, Alinea products offer a nice way of presenting coarse-grained metrics for a novice user but do not offer more features than many other tools. And Intel vector advisor additionally can provide some more guidance than other software when it comes to identifying potential loops to be vectorized but this is not part of the initial health check guidelines and so these results are not mentioned.

11. Interoperability is largely undefined throughout the text. There is only a single mention of interoperability (p.10 6) for "extra" features. Interoperability between what and what exactly? Was interoperability a critieria either in the conception of the test, the run harness or the workflow?

What the authors meant by interoperability and did not explain very well, is that CMORized NetCDF files can be used on different architecture (big and little endian) and for different software (for use in various terrestrial systems software and visualization software). In the section we've added about CF conventions we have also mentioned how this facilitates interoperability and what we mean by interoperability in this instance.

12. The subsequent paragraph - a very long sentence - mention download and rerun. Results on reruns would be welcomed if possible, either on JUQUEEN or better, on other infrastructures.

Yes, the profiling results obtained are the result of running the benchmark described 10 times to get an average and make sure the benchmark is running as it should- i.e. there should not be a huge variance between results. And we have now added this explanation to section 3. The authors think that showing how this benchmark performs on other architecture is really beyond the scope of the paper as we are primarily discussing the RCF and how it aids portability, profiling and provenance tracking. Of course discussion of the portability aspect should include what other machines the RCF is running on (good suggestion) so we have mentioned that this RCF is also being run on Julia (a prototype KNL cluster at JSC), Jureca (a general purpose cluster at JSC) and Juropa3 (a prototype testbed system at JSC) in the description of the RCF.

13. The paper states that platform.xml can be easily extended or altered to include new systems. It is always easy to modify XML files, but not trivial to know what constitutes a valid modification and to successfully deploy it on other systems. Is there any tools to help a user, a developer, an administrator? Most discussion on portability revolves around XML files, compiler/linker flags and use of Python language. The paper concludes that the RCF using a workflow engine leads to code that can be ported easily. These conditions are important, but insufficient. A more thorough description of "environment preparation setup" might help a reader to better assess how close this particular run harness is compared to his own environment(s).The article would benefit from a better definition of portability. Ported from where to where? Any example of a second HPC infrastructure in your network? Precise what future work will advance portability. Other topics that could help a reader - this reviewer in particular - to assess portability could include virtual environments, software containers, software repositories and continuous integration frameworks.

This is a valid point. The RCF facilitates the environment set up through the use of loading modules, most if not all HPC systems contain this feature, thus this is one of the reasons why the RCF is portable- we have now mentioned this in section 2. Within the platform XML file, there are structs defined with standard parameters for run time arguments and compilation flags which can then be used for any HPC system with particular versions of compilers and profiling tools. We have added this to the description in section 2.3 (code portability).

14. Table 2 presents efficiencies measured during the weak scaling experiment. The authors states that more in-depth analysis is needed, but no strategy, best practices or future work is offered to the reader. Is this analysis to be conducted by a specialist, is it tool-assisted, what is the state of the art?

Good point. More in depth profiling would be much more effective together with performance analysis engineers. We have added this explanation and have put in an example of future work that we are currently conducting with the aid of performance analysis specialists: vectorization of individual loops.

15. There is an assumption that the directory structure (Figure B1) “allows for run time provenance tracking” and “such that the model can be rerun without using any other external tools”. The directory structure presented is most probably correct as a part of the RCF implementation. Still, it is unclear this is sufficient to insure provenance tracking or rerun. Is a tool set available to explore these directories and/or rerun models? Is it indexed in some form? Is there some other semantic information available that can be used?

What is meant is that whole simulation can be rerun as the code, the forcing data, the environment: modules loaded, including a list of dependencies their versions, all extraneous scripts including job submission scripts and a complete model description are bundled together in one output directory. If a user were to untar an output directory, they would be able to compile and rerun the experiment using the XML file contained in the directory, with JUBE, on the same machine and obtain the same results. We have tried to improve the description on page 3 and in section 3. Also we have mentioned that there are additional features beyond the users control such as different hardware when porting models and ways to ensure the scientific results remain the same.

List of changes:

1. Title and running title

2. Abstract shortened and long sentences shortened.

3. Introduction: took out some unnecessary references. Page 3, lines 7-9, a specific example of investment has been given. Page 3, line 11: Extra sentence added to tie RCF into automated performance engineering approach.

5. Section 2, Introduction: reference to the appendix taken out and last two sentences changed to better reflect what is presented in this section. Section has been changed in structure and content in order to follow the suggestions and feedback from the reviewers. In addition subsection titles in Section 2 have been changed. Section 2.1: Discussion of other workflow engines added along with discussion of advantages and disadvantages and commonalities between them as well as reasons given as to why the authors chose JUBE. Section 2.2: Added description of RCF itself. Sections 2.1.2 and 2.1.3 taken out as stand alone sections. Section 2.3: Code portability expanded to include a better description of the result comparison test. Section 2.4 added an extra health check step to match the study presented and added in relevant calculations. Also added a better explanation of tools and overlap and what is available on JUQUEEN. Section 2.5 has been expanded to contain more in depth explanations and justifications.

6. Section 3, Introduction: first sentence and last few sentences changed to better reflect the updated content and structural changes following the suggestions and feedback from the reviewers. Section 3.1: Case study explanation moved to here and justifications as to why we would use this particular study are added. Section 3.2: Better explanation of the different compilation flags and justification as to why we chose to tune particular compliation flags has been added to this section along with a clear reference back to the accuracy test described in Section 2. Section 3.3: Mapping between the health check described in Section 2.4 and profiling steps taken has been added, definitions have been moved to Section 2.4 and structural changes have been made to incorporate each health check step separately. Section 3.5: All irrelevant information has been taken out and redundant figures plus additional explanations have been added, following feedback from reviewers.

7. Conclusion: 2nd and 3rd paragraphs have been updated to better justify the RCF as a best practice approach.

8. Appendix B: Appendix B contains the description of the performance metrics in the automated profiling workflow described in Figure 3.

Please note that the marked up manuscript contains highlighted sections which indicate changes from the original manuscript, pertaining to all the changes mentioned in this document. Thank you for your consideration.

A run control framework to streamline profiling, porting and tuning, simulation runs and provenance tracking of geoscientific applications

Wendy Sharples[1,2,3], Ilya Zhukov[1], Markus Geimer[1], Klaus Goergen[2,4], Sebastian Luehrs[1], Thomas Breuer[1], Bibi Naz[2,4], Ketan Kulkarni[1,4], Slavko Brdar[1,4], and Stefan Kollet[2,4]

[1]Jülich Supercomputing Centre, Forschungszentrum Jülich, Jülich, Germany
[2]Institute of Bio- and Geosciences, Agrosphere (IBG-3), Forschungszentrum Jülich, Jülich, Germany
[3]Meteorological Institute, University of Bonn, Bonn, Germany
[4]Centre for High-Performance Scientific Computing in Terrestrial Systems, Geoverbund ABC/J, Jülich, Germany

Correspondence to: Wendy Sharples (w.sharples@fz-juelich.de)

Abstract. Geoscientific modeling is constantly evolving, with next generation geoscientific models and applications placing large demands on high performance computing (HPC) resources. These demands are being met by new developments in HPC architectures, software libraries, and infrastructures. New HPC developments require new programming paradigms leading to substantial investment in model porting, tuning, and refactoring of complicated legacy code in order to use these resources effectively. In addition to the challenge of new massively parallel HPC systems, reproducibility of simulation and analysis results is of great concern. This is due to the fact that next generation geoscientific models are based on complex model implementations and profiling, modeling, and data processing workflows. Thus, in order to reduce both the duration and the cost of code migration, aid in the development of new models or model components, while ensuring reproducibility and sustainability over the complete data life cycle, an automated approach to profiling, porting, and provenance tracking is necessary. We propose a run control framework (RCF) integrated with a workflow engine as a best practice approach to automate profiling, porting, provenance tracking and simulation runs. Our RCF encompasses all stages of the modeling chain: 1. preprocess input, 2. compilation of code (including code instrumentation with performance analysis tools), 3. simulation run, 4. postprocess and analysis, to address these issues. Within this RCF, the workflow engine is used to create and manage benchmark or simulation parameter combinations and performs the documentation and data organization for reproducibility. We show that in using our run control framework, testing, benchmarking, profiling, and running models is less time consuming and more robust than running geoscientific applications in an ad hoc fashion, resulting in more efficient use of HPC resources, more strategic code development, and enhanced data integrity and reproducibility.

1 Introduction

Geoscientific modeling is constantly evolving, leading to higher demands on HPC resources. We distinguish four main developments which increase HPC demands. (i) Higher spatial resolution, where the added value inherent to simulations at high spatial resolutions has been shown, for example, in many studies of regional convection permitting climate simulations (e.g., Prein et al., 2015; Eyring et al., 2015; Heinzeller et al., 2016) and continental hyper-resolution hydrological modeling approaches (e.g., Kollet and Maxwell, 2008; Maxwell et al., 2015; Keune et al.); (ii) increased model domain size, where models are now being run at larger scales, for example, global convection permitting models (Schwitalla et al., 2016), high resolution continental RCMs (Leutwyler et al., 2016), and global hydrology and land surface models which are needed for water resources modeling (Bierkens et al., 2015); (iii) increased model complexity in which the desire to explore the feedbacks between the surface, subsurface, oceans, and atmosphere have led to fully coupled multi-physics global or regional earth system models (ESM) (e.g., Shrestha et al., 2014; Ruti et al., 2016) posing load balancing issues (Gasper et al., 2014), and (iv) increasing number of ensemble members in modeling and data assimilation experiments (e.g., Han et al., 2016; Kurtz et al., 2016). These developments, combined with long climate scenario simulation time spans pose specific challenges in terms of computational resources, data volume, data velocity, data handling, and analysis.

The evolution of geoscientific models is closely linked to and enabled by technical advancements in supercomputing, for example, advances in computer chip development, low cost growth in parallelism, low latency high bandwidth interconnects, or parallel file systems (e.g., Smari et al., 2016; Navarra et al., 2010). Current HPC systems are massively parallel distributed memory supercomputers. Shared memory compute nodes with two or more multi-core CPUs are linked with low latency interconnects (Beanato et al., 2013). Hybrid parallelization, combining multithreading (e.g., via OpenMP or POSIX threads) on a shared memory node and inter-node distributed memory parallelism (via Message Passing Interface, MPI) reduces communication overhead, improves partitioning, load balancing, and communication overlapping as well as memory footprint (Jin et al., 2011). The inter-process communication can be further optimized by explicitly mapping threads onto specific nodes and CPU cores, taking into account the interconnect topology and the model's internal communication pattern (Balaji et al., 2011).

HPC hardware, software, and tools are developing at a rapid pace. For example, heterogeneous HPC architectures that combine multi-core CPUs with accelerators on the same compute node (Brodtkorb et al., 2010) are considered a suitable architecture for future exascale systems, that is, supercomputers with more than 10^{18} floating-point operations per second, because of their energy efficiency (i.e., Flops/Watt), low latency data management, and peak performance per accelerator of currently more than 1 TFLOP/s (Davis et al., 2012). Accelerator design is currently either based on graphic processing units (GPUs) or on Many Integrated Core chips (MICs). These coprocessors have tens of cores and can host hundreds of threads per chip, include their own memory with very high bandwidth (Liu et al., 2012), albeit using different memory architectures (e.g., cache coherence) or parallel programming models (e.g., CUDA, OpenCL, OpenACC). While these HPC developments are instrumental towards next-generation exascale HPC systems, during the next decade (Attig et al., 2011; Keyes, 2011; Davis et al., 2012), MPI-parallel simulation codes on multi-core shared or distributed memory architectures need a substantial amount of porting, profiling, tuning, and refactoring (Hwu, 2014) to efficiently use such kind of hardware, in particular because a very high level of vectorization is needed to take advantage of the ever increasing SIMD units. Moreover, offloading compute intensive code sections to accelerators can also become a performance bottleneck due to excessive data transfers between host and accelerator. Thus, special care needs to be taken with respect to data layout, placement, and reuse. Concurrent to improvements in the simulation codes themselves and development of established parallel solver libraries such as SUNDIALS and PETSc has been underway to enable the use of heterogeneous architectures (e.g., Minden et al., 2013; Breß and Saake, 2013; Kraus et al., 2013). In addition, compilers are constantly improved to provide some level of vectorization when using appropriate optimization flags (Petersen et al., 2013), though additional effort may be required to structure the source code such that compilers are enabled to recognize vectorization opportunities.

In the geosciences, software applications are often complicated by the new developments outlined above. In many cases, complicated legacy codes need substantial investments. Investments in model porting, tuning, and refactoring are necessary in order to efficiently use these upcoming and already existing HPC architectures, software libraries, and infrastructures and achieve a high level of performance. Invested effort has already paid off for many codes (Meadows, 2012; Hammond et al., 2014; Leutwyler et al., 2016; Heinzeller et al., 2016), in the form of significantly reduced run times, however, at a significant cost in resources. For example in Leutwyler et al. (2016), porting COSMO to GPUs brought reduced simulation times (speedup on the order of 3.6) but needed a team of developers to bring this about (see article acknowledgements). In order to reduce both the duration and the cost of code migration, and also aid in the development of new models or model components, a systematic, rigorous approach is needed to fully analyze and understand the runtime behavior and I/O characteristics in detail, and identify performance bottlenecks. In this context, the use of performance analysis tools is crucial. A run control framework with integrated performance analysis tools can automate a performance engineering approach as well as gather information from the resulting output and analyse that output. However, depending on the current focus of the analysis, different tools and techniques may have to be used—sometimes even in combination. For example, while various tools provide generic information about the runtime behavior of an application, specialized tools exists that focus on a particular aspect such as vectorization, threading, communication and synchronization, or I/O. Likewise, while profiling—i.e., the process-local generation of aggregated performance metrics during the execution—can provide a summarized overview of the performance for the entire application run, it is not able to capture the dynamic runtime behavior. Thus, it can be complemented by using event tracing, which collects performance-related events in chronological order and therefore allows to reconstruct the dynamic application behavior in detail. However, care has to be taken when using event tracing, as it is more expensive than profiling as the amount of data in the trace increases with the runtime of the application (e.g., Geimer et al., 2010; Carns et al., 2011). Thus, it is usually only applied to selected parts of the execution, for example, a few time steps or iterations of a solver, that have been identified using more lightweight techniques such as profiling.

In addition to making efficient use of massively parallel HPC systems, reproducibility of simulation results, based on complex model implementations, profiling, modeling, and data processing workflows, must be a fundamental principle in computational research (Hutton et al., 2016). Recently, Stodden et al. (2016) presented "Reproducibility Enhancement Principles" (REP) to help ensure that the computational steps in data processing and generation are similarly important as access to the data themselves. Hence sharing not only data but also details of software, workflows, and the computational environment via open repositories is likewise important. Similarly, Hutton et al. (2016) recommend for computational hydrology that workflows, which combine data and reusable code, are needed in order to ensure provenance of scientific results. Given that in the weather and climate sciences data and primary code availability is often ensured, ancillary code availability is addressed in Irving (2016) as one of the root causes for irreproducibility. With this in mind, we consider the aspect of documenting the porting and performance optimization steps as well as provenance tracking during production simulations as highly relevant to ensure reproducibility. Workflow engines such as ecFlow (Bahra, 2011) or cylc (Oliver et al., 2017) can connect all relevant steps of a modeling chain, submit jobs with dependencies, and help with necessary parameter sweeps for application software porting and tuning alike. At the same time they allow for extensive, systematic logging of the processing steps themselves as well as the log outputs from the individual applications.

In this article, we present a run control framework (RCF) as a best practice approach to porting, profiling, and documenting legacy code using the script-based benchmarking framework JUBE (Lührs et al., 2016) as a workflow engine. We developed profiling, run control, and testing frameworks which are dynamically built with user input into interdependent tasks and these tasks are run using JUBE. While the use case for this portable run control, profiling, and testing workflow engine system discussed in this paper is the software application ParFlow, an integrated parallel watershed model, the RCF is generic and can be applied to any other simulation software. In the remainder of this paper we outline this approach and highlight the subsequent developments scheduled for implementation born out of the extensive profiling of ParFlow. Additionally, we highlight other uses for employing a workflow engine which have enabled us to streamline the run control process for model runs.

2 RCF approach to profiling, portability, and provenance tracking

In this section, the run control framework which could be described as a run harness, along with JUBE is introduced, where a harness in this case is used to describe the framework of scripts and other supporting tools that are required to execute a workflow. The RCF is presented as a means to facilitate portability, profiling, and provenance tracking. This is followed by the description of the standard profiling toolset which is currently built into our RCF to aid in the ParFlow hydrological model development and for production simulation run control as well as the hardware characteristics and the profiling tools available on the supercomputer used in this study. The RCF for the case study in this paper is given in the supplementary material as a tarball file.

2.1 JUBE as a workflow engine

Benchmarking scientific code can assess impacts of changes of the underlying HPC software stack (e.g., compiler or library upgrades) and hardware (e.g., interconnect upgrades), aid in testing as part of software engineering and code refactoring, and in finding optimum numerical model configurations. Benchmarking a numerical model system usually involves several runs with different configurations (compiler, domain, physics parameterizations, solver settings, load balancing), including compilation, instrumentation (i.e., the injection of special monitoring "hooks" into the program to enable profiling and/or event tracing), various simulations, profiling, result verification, and analysis. However, with increasing model and HPC environment complexity, the parameter space for benchmarking can be large. To avoid errors in managing benchmark parameter combinations, to reduce the overall temporal effort, and to ensure reproducibility and comparability, benchmarking must be automated. This task can be accomplished using a workflow engine, which is an application for workflow automation, like the JUBE benchmarking environment (Lührs et al., 2016).

JUBE is a script-based framework designed to efficiently and systematically define, setup, run, and analyze benchmarks and production simulations. The current JUBE v2.1.4 is a Python-based implementation released under GNU GPLv3 actively developed at the Jülich Supercomputing Centre (JSC, https://www.fz-juelich.de/ias/jsc/EN). JUBE allows to easily define benchmark sets via an XML configuration file, in which the workflow and parameter sweeps are specified. When run, JUBE executes the user-defined steps. JUBE controls the automatic execution of the designed workflow and takes care of the underlying file structure to allow an individual execution per run. Automatic bookkeeping separates the different runs and parameter combinations and allows reproducible executions. To generate an overview of the overall workflow execution, the user can configure JUBE how to analyze the different output files to extract information such as the overall runtime or other application-specific data. This allows the system to create a combined overview of the underlying parameterization and the application outputs. The features above combined with the RCF described means that ParFlow users can very quickly get their complicated model workflows up and running without resorting to developing their own specialized bash or Python scripts to run simulations, which are usually bereft of the features contained in our RCF. JUBE, ecFlow, and cylc are all written in Python, all have tasks/steps which can be triggered based on dependencies (e.g., a run task would only run on successful compilation and have variable inheritance where general variable definitions can be overwritten with specific parameters at run time), and both JUBE and ecFlow can read in XML scripts as input, while cylc has its own custom scripting (Bahra, 2011; Oliver et al., 2017). However, cylc and ecFlow can continuously run on the login nodes, even after the job/s are submitted, so that they can interact with the job scheduler, if a user should require it (Bahra, 2011; Oliver et al., 2017). This is hugely beneficial for geoscientific models which run in the order of years and thus require a chain job submission style of running. Other advantages of cylc and ecFlow over JUBE include a GUI, which can be very powerful for non-developers. Unfortunately, a process running on the login nodes on JSC machines has an enforced time limitation of 2 hours. The writers of JUBE are currently developing a version to be released early next year, which can interact with the job scheduler, and can circumvent the two hour time limitation. As we are running models predominantly on JSC machines, this is the main reason we chose JUBE over cylc or ecFlow. In addition, we choose JUBE because we have direct access to the developers and can influence the development of added functionality such as builtin Python scripting for variable declaration, parameter space creation, environment handling, loading files, and substitution. In the case that we could run either cylc or ecFlow continuously on JSC machines, it would be hard to pick one over the other. Both seem to have the same features and functionality, although cylc has been reported to be more complicated when it comes to building workflows (Manubens-Gil et al., 2016).

2.2 RCF Description

Whatever workflow engine one decides to use, someone still needs to integrate the workflows or tasks themselves—JUBE, ecFlow, and cylc are merely tools for task submission. Leveraging the generic JUBE framework, we developed a run control framework, suitable for a typical geoscience model, from a series of XML files integrated with Python scripts to be executed with JUBE (see Figure 1). These jobs are usually run with the following modeling chain: 1. preprocess input, 2. compilation of code (includes code instrumentation with profiling tools) 3. simulation run 4. postprocess and analysis. This modeling chain can be thought of as interdependent tasks set up by the RCF which are then submitted as steps by JUBE. The current run control framework is under version control and can be cloned from GitLab (Hethey, 2013).

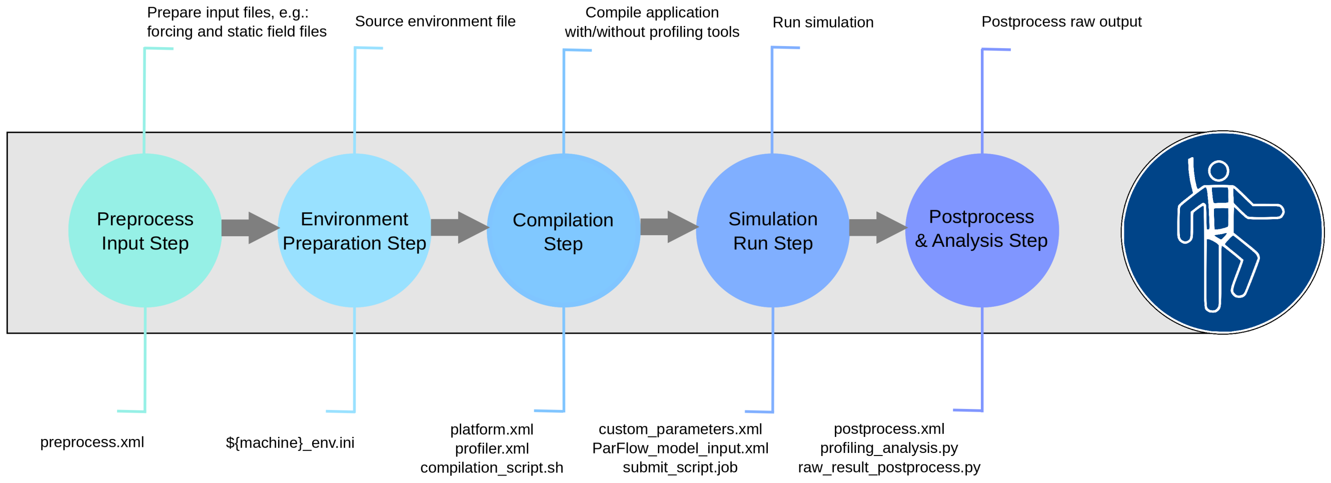


Figure 1. Schematic overview of the modeling chain as supported by our JUBE-based run harness. Each step is annotated with a brief description (top) as well as the respective RCF infrastructure (XML files and scripts, bottom).

The directory structure for the run harness used in the case study in this paper is shown in Figure 2. A top-level Python script, `jubeRun.py`, combines the custom job specifications (`custom/weakScalingSinusoidal_Job_Juqueen.xml`) with the run control benchmark XML script (`driver/ParFlowRC_Benchmark.xml`) into one XML configuration file `execute.xml` which can be parsed through JUBE, and then calls the JUBE run command with the newly created file as an argument. Machine-specific profiling and job submission parameter sets are imported from XML structs given in the scripts `templates/platform.xml` and `templates/profiler.xml`, respectively, and the ParFlow model input parameter sets are imported from the structs given in `templates/ParFlow_model_input.xml`, based on the options specified in the custom job. All environment and submission scripts are stored in directories `${machine}_files`, all the profiling specific wrappers and filter files are stored in the directory `profiler_data` and all ParFlow model input is stored in the directory `model` (see Figure 2).

The driver script contains the steps to run the modeling chain, where the steps themselves can be dependent on the successful completion of the previous step/s. Compilation parameters are set based on which HPC platform or machine the benchmark suite is run on and the profiling tool/s chosen. The user can specify the machine, the profiling tool/s, the ParFlow model, the domain size, the scaling parameters, and overwrite the default compilation and job submission parameters via a custom job XML script. The user can also describe an analysis step for the postprocessing of output.

The template scripts are used in our run harness to capture default settings for HPC platform and model related parameters. Specific default settings can be overwritten when specified in the custom job XML script. In the case study, the template scripts used contain HPC platform-related default settings for profiling tools (`profiler.xml`), compilers (`platform.xml`), and

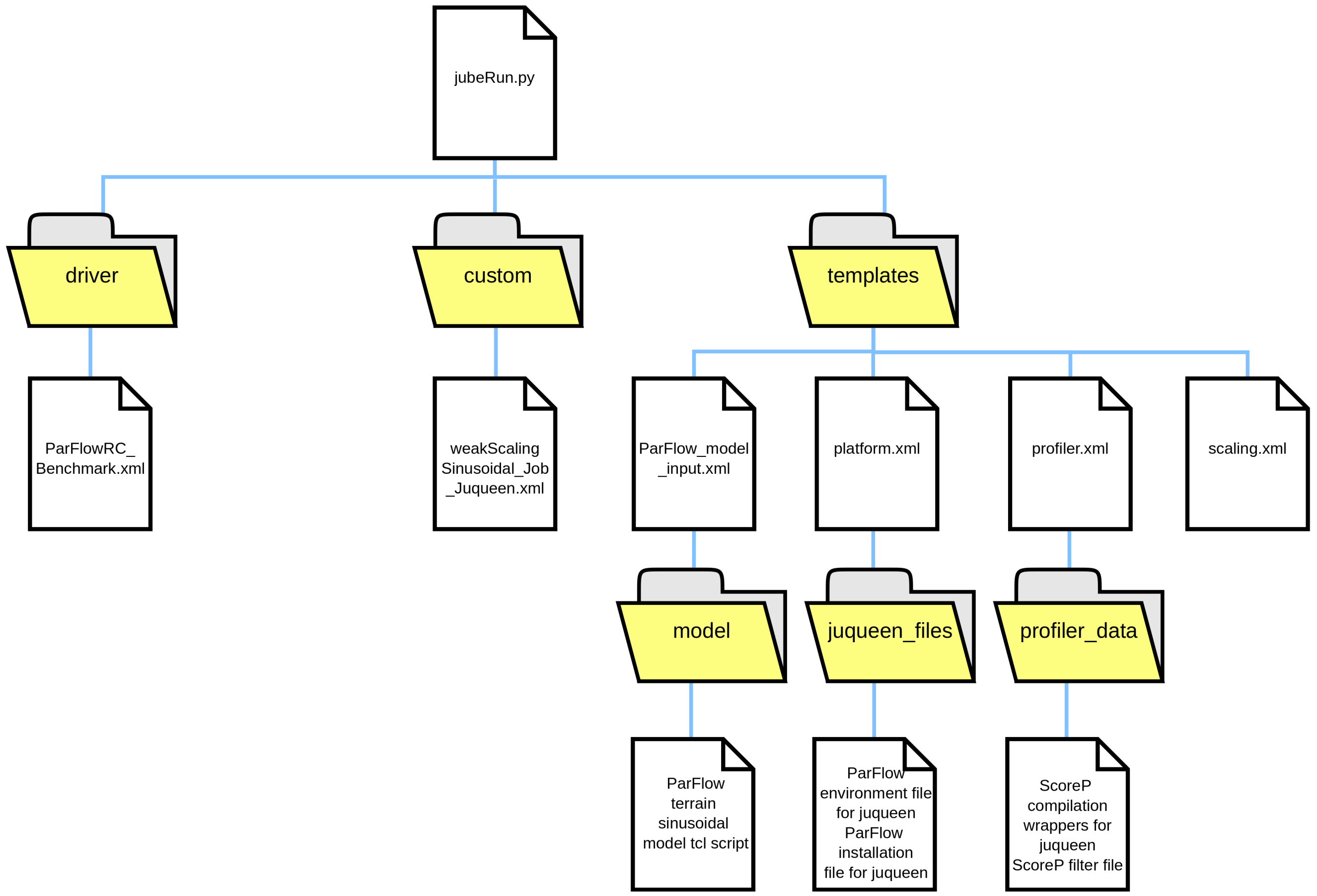


Figure 2. Directory structure of the run control framework used in the case study (Section 3).

job submission (`juqueen_files`) (see Figure 2). In addition there are templates for each type of ParFlow model, where the idealized overland flow model is defined (see Section 3.1), and the type of scaling typically used for a benchmark suite, i.e., weak scaling and strong scaling. The template ParFlow model XML script sets the default settings for a series of ParFlow models set up over different domains. The template scaling XML script sets the default domain settings for a given ParFlow model. The scaling parameters can be set such that either one subdomain is spawned per thread (weak scaling) or such that the domain size does not change (strong scaling). The custom XML script sets the HPC platform used, the profiling tool, the parflow model type, the domain extent and type of scaling and the postprocessing analysis step. The custom job script can also overwrite default settings such as compiler settings and job submission settings.

Our RCF creates benchmark or production model simulation suites, which can run on multiple computer systems and whose results can be postprocessed and analysed, via the execution of the driver file using JUBE, where the driver file ingests custom and template input. All parameters which are comma separated are parsed by JUBE as a parameter sweep, so that each comma-separated variable is iterated over to become a separate run. At the time of writing, our RCF is running on several different machines, namely JUQUEEN (a highly scalable cluster at JSC), JULIA (a prototype KNL cluster at JSC), JURECA (a general purpose cluster at JSC), and JUROPA3 (a prototype testbed system at JSC). All files mentioned in Figure 2 are available in the tarball supplied as supplementary material for this paper.

2.3 RCF: Aiding code portability

Within the RCF, we have separated all the information pertinent to the existing compilers, required environment modules, and workload manager job submission specifics for a given system into a single XML file (see Figure 1, `platform.xml`). This `platform.xml` file can easily be extended to include any new system. When compilers, environment modules, and workload managers are updated or new features or functionality are added, the `platform.xml` can be easily altered to include these updates. For example, as new C and Fortran compiler versions are released with improved code generation and potentially new optimization flags, it is useful to reassess which compilation flags give the best run time without compromising the accuracy of the result. In the case of our RCF, this is as simple as altering the compiler flags parameter in `platform.xml` with comma-separated values for each different compiler flag, which produces a benchmark suite. In order to ensure accuracy is not compromised we built in a result comparison test, where the user can compare the result with previously generated output. However, some thought needs to be taken in setting up the test as ultimately it is up to the user to decide which models and previously generated output is accurate enough to be the gold standard to test against. If the user is uncomfortable in using their own models for this test, most geoscience applications (ParFlow included) have a plethora of tests with previously generated output to use as a gold standard to test against.

2.4 RCF: Facilitating code profiling

In order to analyze ParFlow's runtime behavior, determine optimal runtime settings, as well as identify performance bottlenecks during model development, we use several complementary performance analysis tools. Setup, compilation wrappers, and analysis profiling steps were built into our RCF (Figure 1: `profiler.xml`, Appendix A) with support for the following tools: Score-P v3.1 (Knüpfer et al., 2012) and Scalasca v2.3.1 (Geimer et al., 2010; Zhukov et al., 2015), where results collected with Score-P and Scalasca can be examined using the interactive analysis report explorer Cube v4.3.5 (Saviankou et al., 2015), Allinea Performance Reports v7.0.4 (January et al., 2015), Extrae v3.4.3 (Alonso et al., 2012), Paraver v4.6.3 (Labarta et al., 2006), Intel Advisor 2015 (Rane et al., 2015), and Darshan v3.0.0 (Carns et al., 2011) (see Table A1 in Appendix A for a more detailed description of each performance analysis tool listed above). The modeling chain for the profiling workflow is as follows:

1. Prepare the input data;

2. Load environment modules and set up performance analysis tool specific parameters;

3. Compile or link ParFlow using scripts and wrappers, depending on what is required by the profiling tool; e.g., Score-P requires compilation and linking using wrappers, whereas Darshan requires only linking after compilation;

4. ParFlow execution with the necessary tool flags (e.g., Scalasca has various runtime measurement, collection, and analysis flags which can be turned on or off);

5. Parse and analyze the results interactively (e.g., using interactive visual explorers like Paraver, Cube, etc.) or generate a textual performance metric report via a post processing step.

Note that code instrumentation with performance analysis tools—that is, the insertion of tool-specific measurement calls into the application code which are executed at relevant points (events) during runtime—can introduce significant overhead, which can be assessed by comparing to an uninstrumented reference run. If the runtime of the instrumented version of the code under inspection is much longer than the reference run (more than 10-15%), it is recommended to reduce instrumentation overhead as the measurement may no longer reflect the actual runtime behavior of the application. Typical measures to reduce the runtime overhead include turning off automatic compiler instrumentation, filtering out short but frequently called functions, and applying manual instrumentation using specific APIs provided by the tools.

A typical workflow when performing an initial "health examination" on a scientific code can be described as follows:

1. “Which function(s) or code region(s) in my program consume(s) the most wallclock time?” This question can usually be answered using a flat profile, which breaks down the application code into separate functions or manually annotated source code regions (e.g., initialization vs. solver phase) and aggregates the wallclock time spent in each function/region. This ascertains the area(s) of interest in order to streamline performance analysis efforts.

   Typical diagnosis tools: Allinea Performance Reports, Score-P + Cube, Extrae + Paraver

2. “Does my application scale as expected?” Typically all scientific applications aim to perform well at scale. To address this question, profiles need to be collected with varying numbers of processes and the scalability of functions/code regions within the areas of interest can be examined.

   There are two types of scaling: strong and weak scaling. In case of strong scaling, the overall problem size (workload) stays fixed but the number of processes increases. Here, the runtime is expected to decrease with increasing number of processes. By contrast, in case of weak scaling, the workload assigned to each process remains constant with an increase of processors and, thus, the runtime is ideally expected to be constant as well.

   The strong scaling efficiency, E_{ss} is computed as a relation of speedup to the number of processes. Speedup is computed as a relation of the amount of time to complete a work unit with one process to the amount of time to complete N of the same work units with N processes:

   $$E_{ss} = \frac{S}{N} = \frac{T_1}{NT_N} \tag{1}$$

   The weak scaling efficiency, E_{ws} is computed as a relation of the amount of time to complete a work unit with one process to the amount of time to complete N of the same work units with N processes:

   $$E_{ws} = \frac{T_1}{T_N} \tag{2}$$

E_{ss} and E_{ws} are very common metrics in HPC to quantify and qualify the scalability of the application. These metrics indicate how efficient an application is when using increasing numbers of processes.

Typical diagnosis tools: Allinea Performance Reports, Score-P + Cube, Extrae + Paraver

3. "Does my program suffer from load imbalance?" If this is the case, some processes will be performing significantly more or less work than the others. Load balance is an indication of how well the load is distributed across processors. If a code is not well balanced, HPC resources will be used inefficiently as imbalances usually materialize as wait states in communication/synchronization operations between processes/threads. Thus, this may be an area to concentrate code refactoring efforts.

   To characterize load imbalance Rosas et al. (2014) invented the load balance efficiency metric, E_{lb}, which is defined as a relation between average computation, $\overline{T}$, and maximal computation time, T_{max}:

   $$E_{lb} = \frac{\overline{T}}{T_{max}} \tag{3}$$

   Note that load imbalance can either be static or dynamic. While the former can usually be easily identified in profiles, pinpointing the latter may require more heavyweight measurement and analysis techniques such as event tracing, as imbalances may cancel out each other in aggregated profile data.

   Typical diagnosis tools: Score-P + Cube, Score-P + Scalasca + Cube, Extrae + Paraver

4. "Is there a disproportionate time spent in communication or synchronization?" Communication and synchronization overheads can be caused by network latencies (e.g., due to an inefficient process placement onto the compute resources), or wait states and other inefficiency patterns (e.g., caused by load or communication imbalances). If these overheads increase significantly with an increase in resources, this can be a further barrier to scalability.

   To quantify and qualify disproportionate time spent in communication or synchronization, Rosas et al. (2014) developed the communication efficiency metric, E_{com}, which is defined as a relation between T_{max} and total execution time, T_{tot}:

   $$E_{com} = \frac{T_{max}}{T_{tot}} \tag{4}$$

   Typical diagnosis tools: Allinea Performance Reports, Score-P + Scalasca + Cube, Extrae + Paraver

5. "Is my parallelization strategy efficient?" To answer this question, Rosas et al. (2014) developed an auxiliary efficiency metric, parallel efficiency, which quantifies and qualifies the parallelization strategy as a whole.

   Parallel efficiency is computed as the product of the previously defined metrics load balance efficiency (Step #3) and communication efficiency (Step #4):

   $$E_{par} = E_{lb}E_{com} = \frac{\overline{T}}{T_{tot}} \tag{5}$$

   Where a minor value reduction of any component will result in a significant reduction of parallel efficiency.

Efficiency values range from zero to one, where a value of one is the most efficient. Using Cube's derived metric feature (Zhukov et al., 2015), we can derive these efficiency metrics from the Score-P profile data automatically.

6. "Is my application limited by resource bounds?" There are several bounds one can reach, such as

   (a) CPU bound, i.e., the rate at which processes operate is limited by the speed of the CPU. For example, a tight loop that can be vectorized and operates only on a few values held in CPU registers is likely to be CPU bound.

   (b) Cache bound, i.e., the simulation is limited by the amount and the speed of the cache available. For example, a kernel operating on more data than can be held in registers but which fits into cache is likely to be cache bound.

   (c) Memory bound, i.e., the simulation is limited by the amount of memory available and/or the memory access bandwidth. For example, a kernel operating on more data than fits into cache is likely to be memory bound.

   (d) I/O bound, i.e., the simulation is limited by the speed of the I/O subsystem. For example, counting the number of lines in a file is likely to be I/O bound.

   Typical diagnosis tools: Score-P + PAPI + Cube, Extrae + Paraver, Intel Advisor, Darshan

7. There are additional questions one can add to the survey, for example: "How many pipeline stalls, cache misses, and mis-predicted branches are occurring?", "How can we assess serial performance", etc.

   To describe serial performance, for example, we use the instructions per cycle metric (IPC), i.e., the ratio of total instructions executed and the total number of CPU cycles. Potential reasons for low IPC values are pipeline stalls, cache misses, and mis-predicted branches (John and Rubio, 2008). Therefore, additional measurements with hardware counters should be made to determine the number of cache misses in L1, stalls, and mis-predicted branches when a low IPC value is computed.

   Typical diagnosis tools: Score-P + PAPI + Cube, Extrae + Paraver, Intel Advisor, Darshan, etc.

As can be seen, different tools can be used to answer the various questions mentioned above. However, they usually employ different measurement and analysis techniques which may prohibit the use of a particular tool under certain circumstances. For example, the combination Extrae + Paraver uses event tracing in conjunction with a manual/visual analysis, which is only applicable to selected parts of the execution (i.e., the current region of interest). In contrast, Score-P profiling + Cube uses a more lightweight measurement technique that can handle arbitrarily long executions, but requires a second measurement if there is a need for a focused in-depth analysis based on event tracing. Moreover, the level of detail provided by the various tools usually differs. For example, Allinea Performance Reports can provide a very coarse-grained, initial performance overview. Such an overview can be sufficient to already rule out certain classes of performance issues, but does not provide enough detail to track down the root causes of the issues being identified. Thus, it can only give an indication on which more in-depth analysis to carry out in the next step/s. Also, not all performance analysis tools are available on every platform. For example, Allinea Performance Reports and Intel Advisor are only available on x86 architectures, but not for the IBM Blue Gene/Q platform used in this case study. And finally, if two tools provide comparable functionality, users are inclined to use the tool/s they feel most comfortable with. Thus, our RCF implements support for all of the diagnosis tools mentioned in the section above (see Appendix A for more details), covering many different use cases and preferences.

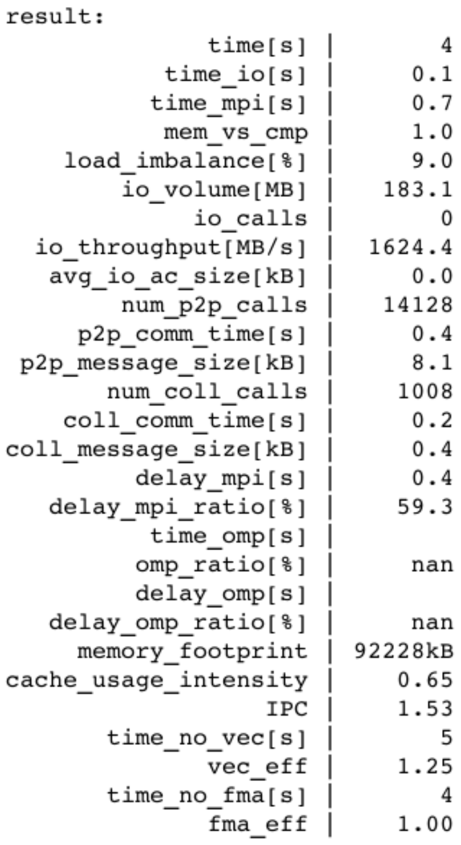
In this case study, we followed the health check using the diagnosis tools (co-)developed by JSC and available on the JUQUEEN platform, namely 1. Score-P profile measurements, including hardware performance counters collected via PAPI (Moore et al.), 2. Score-P trace measurements followed by a subsequent automatic Scalasca trace analysis, 3. Manual analysis of measurements from the abovementioned steps with an interactive visual browser, i.e., Cube. Where Score-P is a community-maintained scalable instrumentation and performance measurement infrastructure for parallel codes that can collect both profiles and event traces, Scalasca Trace Tools are a collection of scalable trace-based tools for in-depth analyses of concurrent behavior, and Cube is an interactive analysis report explorer for Score-P profiles and Scalasca trace analyis reports. Additionally we used Darshan, a tool to capture and characterize the I/O behavior of an application, for I/O profiling. Both Score-P and Scalasca output their results in CUBE4 format, which can be processed by the Cube GUI and command-line tools. The latter are used by the RCF to process the result files and to collate specific information from each run in tabular format.

In order to track the health of ParFlow with each new release, we developed an automated performance metric extraction workflow and integrated this into our RCF to obtain key performance indicators such as MPI wait time, memory footprint, cache intensity, etc. in order to quickly assess whether new developments or additions to the code improve or degrade the overall performance. An example of such a workflow output is given in Figure 3. Metrics shown in Figure 3 not only describe the application in general, but also assess potential bottlenecks, i.e., I/O, communication, node and core performance, memory usage (see Appendix B for more details).

2.5 RCF: Provenance tracking

JUBE has many provenance tracking features and tools. JUBE automatically stores the benchmark suite data for each workflow execution, which can be parsed by JUBE's analysis tools. Workflow metadata is automatically parsed by JUBE and then compiled into a report detailing the run and which settings were used for each suite. Subsequent analysis procedures can be predefined, added, or altered by the user after the experiment to automate data processing. These features and tools are designed to facilitate documentation and archiving. Additionally, JUBE's workflow execution directory structure allows for run time provenance tracking. JUBE's workflow management system automatically creates a suite of the parameter sets and steps for each workflow. JUBE then creates a unique execution unit or work package for a specific step and parameter combination. Each workflow execution has its own directory named by a unique numeric identifier which gets incremented for subsequent runs. Inside this directory, JUBE handles the workflow execution's metadata and creates a directory for each separate work package. This avoids interference between different work package runs and creates a reproducible structure. For dependent work packages, symbolic links are created to the parent work package, for user access.

We added extra provenance tracking features to ParFlow simulation runs such as configuration management (e.g. logging of time spent in important routines, performance tracking etc), and postprocessing of output to a standardized format enriched with metadata. The post processing of output entails converting unannotated ParFlow binary file model simulation output to a more portable NetCDF output containing standardized meta-data enrichment using CMOR and CF standards and incorpora-



```
result:
                  time[s] |       4
               time_io[s] |     0.1
              time_mpi[s] |     0.7
               mem_vs_cmp |     1.0
        load_imbalance[%] |     9.0
            io_volume[MB] |   183.1
                 io_calls |       0
      io_throughput[MB/s] |  1624.4
        avg_io_ac_size[kB] |     0.0
             num_p2p_calls |   14128
          p2p_comm_time[s] |     0.4
     p2p_message_size[kB] |     8.1
            num_coll_calls |    1008
         coll_comm_time[s] |     0.2
    coll_message_size[kB] |     0.4
              delay_mpi[s] |     0.4
       delay_mpi_ratio[%] |    59.3
              time_omp[s] |
             omp_ratio[%] |     nan
              delay_omp[s] |
       delay_omp_ratio[%] |     nan
          memory_footprint | 92228kB
     cache_usage_intensity |    0.65
                       IPC |    1.53
            time_no_vec[s] |       5
                   vec_eff |    1.25
            time_no_fma[s] |       4
                   fma_eff |    1.00
```

Figure 3. Example output from a performance metric extraction workflow. See Appendix B for a complete explanation of these performance metrics and why they might be useful.

tion of all ParFlow model settings (Eaton et al., 2003)). The CF conventions for climate and forecast metadata are designed to promote the processing and sharing of files created in NetCDF format. The conventions define metadata that provide a definitive description of what the data in each variable represents, and of the spatial and temporal properties of the data. Use of the convention ensures that users of data from different sources can properly compare quantities, along with facilitating interoperability and portability. Interoperability in this case means that CMORized NetCDF files can be used on different architectures (big/little endian) and for different software (for use in various terrestrial systems software and visualization software). The data conversion postprocessing step was developed in accordance with state-of-the-art data lifecycle management and to maintain interoperability (Stodden et al., 2016).

In addition, the postprocess and analysis step we developed contains an archive process at the end of the modeling chain, which documents and collates the environment variables, model input, model simulation scripts, model submission scripts, log files, postprocessed output, and application code in such a fashion that the archived output can be downloaded and rerun following the instructions in the simulation documentation, without need for any additional input, a practice recommended by Hutton et al. (2016). That is, if a user were to untar an output directory they would be able to compile and rerun the simulation, using the XML configuration file with JUBE, on the same machine, without having to obtain information elsewhere. The information contained in the tarred directory would also almost cover points 1-4 from Irving (2016)—an option for the user to reproduce published figures from the postprocessed NetCDF output files produced is not yet built-in. However ensuring the science remains the same on different HPC architectures needs to be considered when porting models. The developers of the software need to employ strategies such as continuous integration on multiple machines to ensure consistency of science across architectures and compilers. Our RCF could also be adapted to facilitate these strategies.

3 Case study: RCF profiling workflow

In this section, we present the experimental design of the test case, to be used in the profiling study, steps we took in porting the test case, and results of the profiling study, which demonstrate the usefulness of our RCF. For this study we use the highly scalable IBM Blue Gene/Q system JUQUEEN. JUQUEEN features a total of 458 752 cores from 1 024 PowerPC A2 16-core, four-way simultaneous multithreading CPUs in each of the 28 racks and a total of 448 TB main memory with a Linpack performance of 5.0 Petaflops. ParFlow is running under Linux microkernels on compute nodes using IBM XL compilers and a proprietary MPI library and a GPFS filesystem. All profiling results shown in this paper are the result of running the benchmark described in Section 3.1 10 times to get an average and make sure the benchmark is running as it should, taking notice of the variance between results. See the tarball supplied as supplementary material for this manuscript for the complete RCF used for this case study.

3.1 Case Study: Experimental design

In order to demonstrate the applicability of the RCF, a weak scaling demonstration study with an idealized overland flow test case was set up for ParFlow (Maxwell et al., 2015). ParFlow (v3.2, https://github.com/parflow) is a massively parallel, physics-based integrated watershed model, which simulates fully coupled, dynamic 2D/3D hydrological, groundwater and land-surface processes suitable for large scale problems. ParFlow is used extensively in research on the water cycle in idealized and real data setups as part of process studies, forecasts, data assimilation experiments, hind-casts as well as regional climate change studies from the plot-scale to the continent, ranging from days to years. Saturated and variably saturated subsurface flow in heterogeneous porous media are simulated in three spatial dimensions using a Newton-Krylov nonlinear solver (Ashby and Falgout, 1996; Jones and Woodward, 2001; Maxwell, 2013) and multigrid preconditioners, where the three-dimensional Richards equation is discretized based on cell-centered finite differences. ParFlow also features coupled surface-subsurface flow which allows for hillslope runoff and channel routing (Kollet and Maxwell, 2006). Because it is fully coupled to the Common Land Model (CLM), a land surface model, ParFlow can incorporate exchange processes at the land surface including the effects of vegetation (Maxwell and Miller, 2005; Kollet and Maxwell, 2008). Other features include a parallel data assimilation scheme using the Parallel Data Assimilation Framework (PDAF) from Nerger and Hiller (2013), with an ensemble Kalman filter, allowing observations to be ingested into the model to improve forecasts (Kurtz et al., 2016). An octree space partitioning algorithm is used to depict complex structures in three-dimensional space, such as topography, different hydrologic facies, and watershed boundaries. ParFlow parallel I/O is via task-local and shared files in a binary format for each time step. ParFlow is also part of fully coupled model systems such as the Terrestrial Systems Modeling Platform (TerrSysMP) (Shrestha et al., 2014) or PF.WRF (Maxwell et al., 2011), which can reproduce the water cycle from deep aquifers into the atmosphere.

A three-dimensional sinusoidal topography as shown in Figure 4 was used as the computational domain with a lateral spatial discretization of $\Delta x = \Delta y = 1$ m and a vertical grid spacing of $\Delta z = 0.5$ m; the grid size, n, was set to nx = ny = 50 and nz = 40 resulting in 100 000 unknowns per CPU core, with one MPI task per core. In order to simulate surface runoff from the high to the low topographic regions with subsequent water pooling and infiltration, a constant precipitation flux of 10 mm/hour was applied. This results in realistic non-linear physical processes and thus compute times. The water table was implemented as a constant head boundary condition at the bottom of the domain with an unsaturated zone above, 10m below the land surface. The heterogeneous subsurface was simulated as a spatially uncorrelated, log-transformed Gaussian random field of the saturated hydraulic conductivity with a variance ranging over one order of magnitude. The soil porosity and permeability were set to 0.25 m/day. This idealized setup was used for the profiling case study as opposed to a real world set up due to the symmetry inherent in the setup. In contrast, a real world experiment has asymmetry in both the meteorological forcing and also the model topography which naturally lead to load imbalances. These asymmetries could therefore obscure whether there are actually load imbalances due to poor software design.

The weak scaling experiment is defined as how the solution time varies with the number of processors for a fixed problem size of 100 000 degrees of freedom per processor. The horizontal (nx,ny) grid size is increased but the number of cells in the vertical direction, nz, remain constant. All model configurations were run for 10 hours with a time step size of $\Delta t = 0.5$ hr.

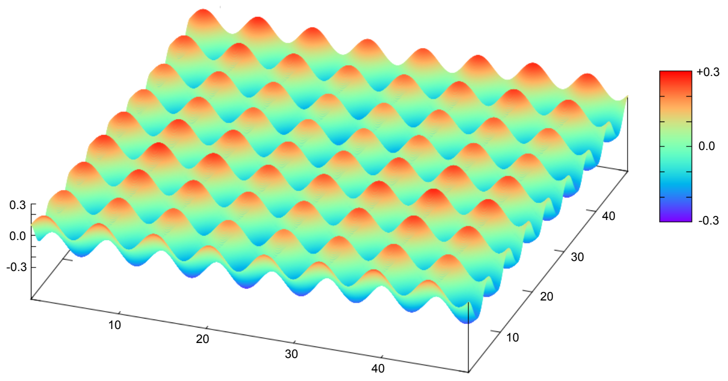


Figure 4. Model setup, showing cross-sectional domain and sinusoidal topography variation from the top of the model (z=20) for each processor.

3.2 Porting ParFlow to JUQUEEN

When porting ParFlow onto JUQUEEN, we used the IBM XL C compiler available on the platform (v12.1), which provides several compiler options that can help control the optimization and performance of C programs. We focused on two aspects of optimization, namely, loop optimization (`-qhot`) and general optimization levels: `-O1` to `-O3`, where these optimizations range from local basic block to whole-program analysis. The higher the optimization level, the more sophisticated optimization techniques are applied. For example using optimization level `-O1` performs only quick local optimizations such as constant folding and elimination of local common subexpressions, whereas optimization level `-O3` performs rewrites of floating point expressions, aggressive code motion, scheduling on computations, and loop optimizations and additionally the compiler replaces any calls in the source code to standard math library functions with calls to the equivalent MASS library functions. The focus on these two aspects was a result of following the user guidelines set out by IBM in using the XL C compiler IBM (2012).

We set up the accuracy test described in Section 2.3 with the gold standard output to test against being results previously generated with the model described in Section 3.1. The gold standard output was verified via inbuilt water balance and energy balance tests using ParFlow pftools (a package of utilities and a TCL library that is used to setup and postprocess Parflow binary files). Accuracy was determined to be met when the output generated does not vary with the gold standard to six significant figures.

We found the optimal commonly used compilation flag which did not compromise accuracy with the IBM XL C compiler for ParFlow to be `-O3 -qhot -qarch=qp -qtune=qp`, where the architecture and tuning flag was set to be `qp`, which indicates the specific architecture of JUQUEEN (see Table 1). Using the `-O3` compilation flag resulted in a speedup of close to factor of 2 when running with 16 MPI tasks on 16 CPU cores (one compute node on Blue Gene/Q, no multithreading). The timing results were compiled using JUBE's result parser functionality, which was run as a post processing step. This is in agreement with the results in running the fully coupled TerrSysMP model system on JUQUEEN in Gasper et al. (2014).

IBM XL COMPILER FLAGS	TIME [S]
-O1 -qhot -qarch=qp -qtune=qp	203
-O2 -qhot -qarch=qp -qtune=qp	203
-O3 -qhot -qarch=qp -qtune=qp	110

Table 1. Time taken to run the ParFlow test case on JUQUEEN (IBM Blue Gene/Q) with 16 MPI processes using three different commonly used compiler flag optimizations.

3.3 Profiling results and analysis

The following section describes the results from the demonstration scaling study, following the code performance "health check" protocol given in Section 2.4.

Analysis of time spent in ParFlow functions: Health check step #1

As a first step, a breakdown of the time spent in each annotated region of ParFlow was obtained via internal timings in ParFlow and a Score-P profile measurement, visualized as a bar chart in Figure 5. From the breakdown it is clear that the core component of ParFlow is the computation of the solution to a system of nonlinear equations, reflected in Figure 5, where most of the wallclock time is spent in the blue regions which make up the time spent getting a solution via a nonlinear solve step. A large part of the nonlinear solver's workflow can be summarized in two steps, which are as follows: the initialization of the problem for the specific input and the actual solver loop. The last two steps reside in the `KINSol` routine which is a component of the SUNDIALS solver library (Hindmarsh et al., 2005). Therefore, the nonlinear solve routine and its components are the focus of interest for reducing ParFlow's runtime. The nonlinear solve loop performs the computational process of computing an approximate linear solution (`KINSpgmrSolve`) where the intermediate solution is updated every iteration until the desired convergence or tolerance is reached. Those two aforementioned steps within the nonlinear solve loop are manually annotated in the source code and we will focus on these for our results. For simplicity, we have shortened these two steps to `setup_solver` and `solver_loop` respectively and will refer to them by this nomenclature henceforth.

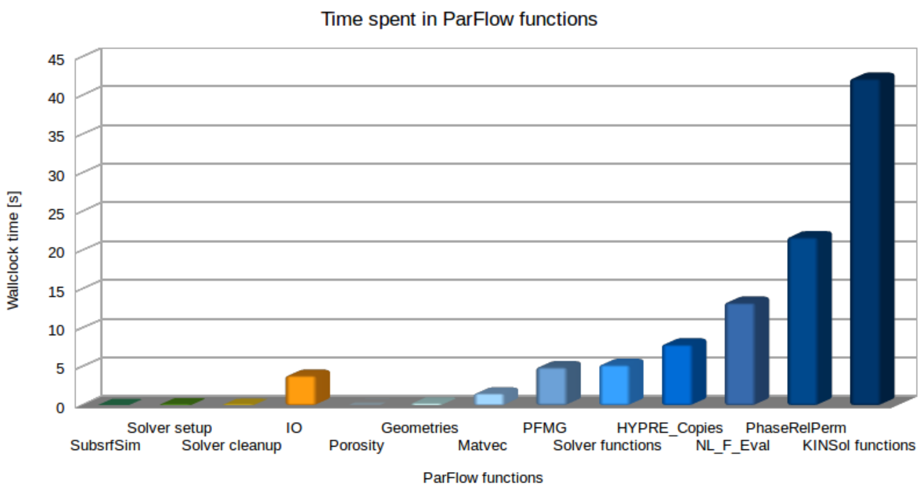


Figure 5. Time spent in ParFlow functions or routines, where the functions/routines can be divided into four categories, set up, clean up, I/O, and solve. The functions in the category "set up" are depicted in green: SubsrfSim—setting up the domain, Solver setup—initializing the solver. The functions in the category "clean up" are depicted in yellow: Solver cleanup—finalizing the solver. The functions in the category "I/O" are depicted in orange: PFB I/O—ParFlow binary I/O. The functions in the category "solve" are depicted in blue: Porosity—calculation of the porosity matrix, Geometries—calculation of the simulation domain, MatVec—matrix and vector operations, PFMG—Geometric Multigrid Preconditioner from HYPRE, Solver functions—miscellaneous functions, HYPRE_Copies—copying data within HYPRE, NL_F_Eval—setting up the physics and field variables for the next iteration, PhaseRelPerm—setting up the permeability matrix, KINSol functions—nonlinear solver functions from SUNDIALS.

Scalability: Health check step #2

Our scalability analysis of ParFlow is again based on the Score-P profile measurement. Figure 6 shows a plot of the execution time versus the number of MPI processes when running the weak scaling experiment as outlined in Section 3.1, broken down into the two regions of interest: `setup_solver` and `solver_loop`. The behavior of both regions show an increase of execution time with an increase in the number of processes, though the `setup_solver` region shows better performance in comparison to the `solver_loop`. However, the strong scaling efficiency profile is comparable to similar codes (Mills et al., 2007).

Upon examination of the results shown in Table 2, at 32 768 MPI processes, the weak scaling efficiency E_{ws} (see Equation 2), drops to approximately 21%. To try to acertain the routines which hinder scalability, further inspection of the breakdown between computation and communication (Figure 7) shows that total communication is considerably increasing at scale. This indicates that communication could be a scalability breaker and should be investigated further (see health check step #4).

# MPI processes	Weak Scaling Efficiency	Load Balance Efficiency	Communication Efficiency	Parallel Efficiency [%]
1 024	1.0	0.96	0.97	0.93
2 048	0.84	0.97	0.97	0.94
4 096	0.84	0.97	0.82	0.80
8 192	0.55	0.98	0.75	0.73
16 384	0.50	0.98	0.69	0.67
32 768	0.21	0.98	0.64	0.63

Table 2. Weak scaling efficiency, load balance efficiency, communication efficiency, and parallel efficiency, running the weak scaling experiment up to 32 768 MPI processes.

Load balance: Health check step #3

Table 2 shows that the load balance efficiency at the different scales is between 0.96 and 0.98. This means that the workload is equally distributed among the processes for our idealized test case, that is, that there is no inherent load imbalance issue in the algorithm.

Communication: Health check step #4

Values of communication efficiency shown in Table 2 reduce from 0.97 (1 024 processes) to 0.64 (32 768 processes). This means that time spent in communication is growing at scale. Therefore, it is worthwhile taking a closer look at what is happening (Figures 7 and 8).

Time spent in communication grows with increasing number of processes. For example, the communication time constitutes 37% of the total time when running the test case with 32 768 MPI processes. The main contributors to communication time within the regions of interest are `MPI_Allreduce` in `setup_solver` and `MPI_Waitall` in `solver_loop`. However,

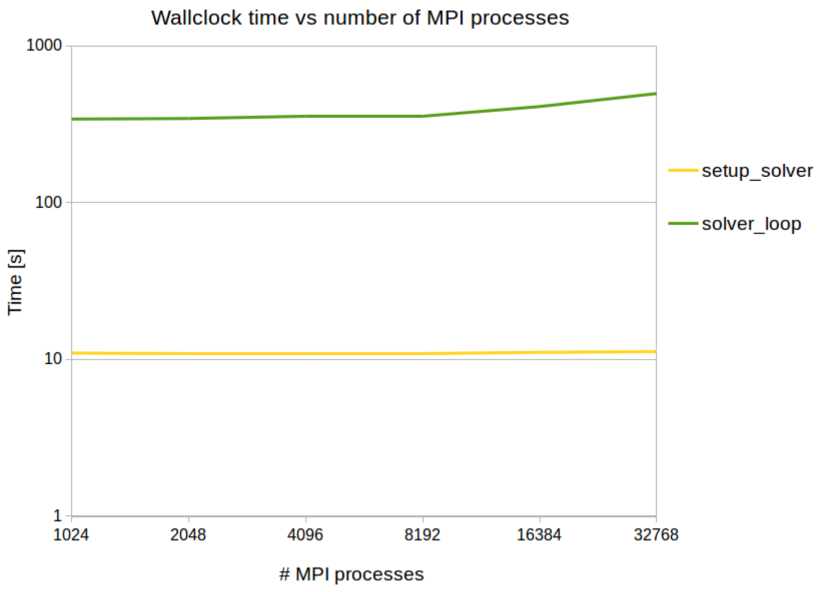


Figure 6. Execution time versus the number of MPI processes for the regions of interest, running the weak scaling experiment up to 32 768 MPI processes.

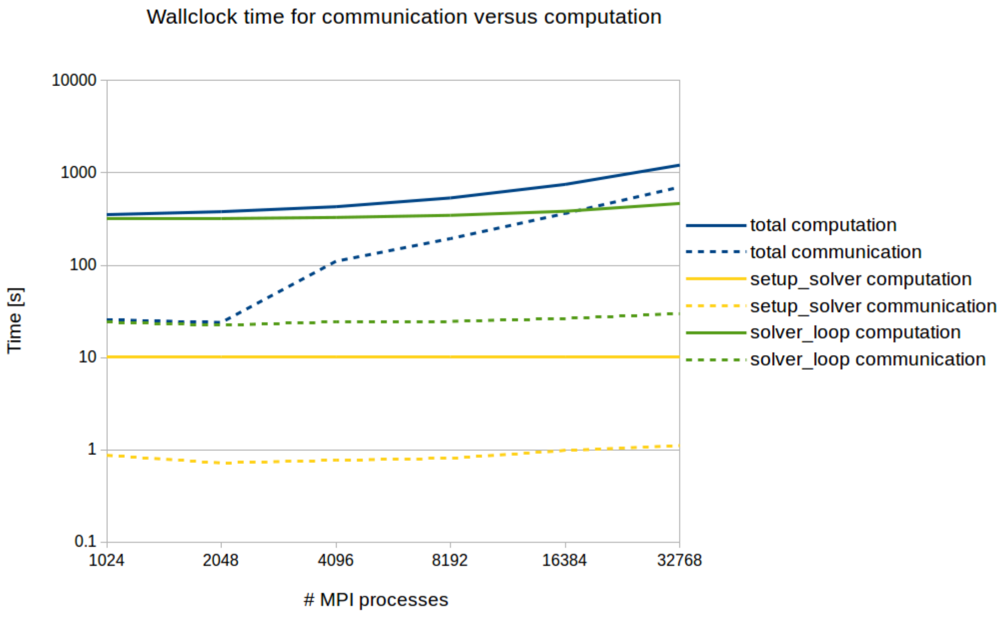


Figure 7. Wallclock time for communication versus computation for the regions of interest for the weak scaling experiment using the test case described in Section 3.1.

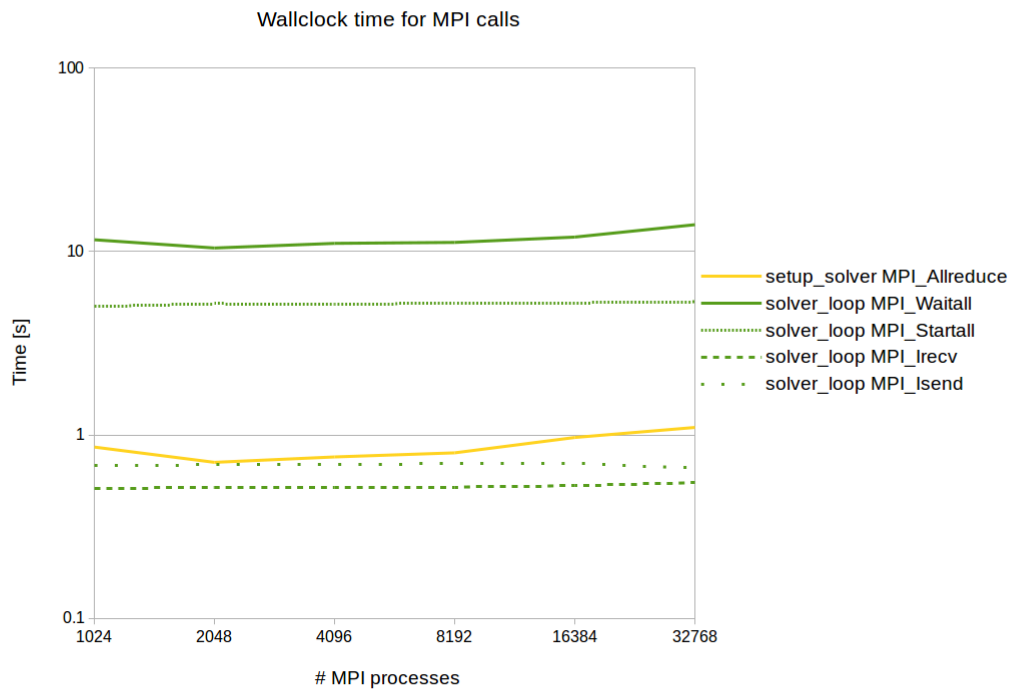


Figure 8. Wallclock time for MPI calls for the weak scaling experiment using the test case described in Section 3.1.

the main communication problem is outside of `setup_solver` and `solver_loop` e.g., in the initialization phase or the preconditioner as `setup_solver` and `solver_loop` communication time do not contribute much. The slight increase in communication time in those two routines could be attributed to `MPI_Allreduce` in `setup_solver` and `MPI_Waitall` in `solver_loop` by further breaking down the communication routines (see Figure 8). A trace analysis using Scalasca identified a costly wait-state pattern constituting 23% of total time in the initialization phase occurring in `MPI_Allreduce` in the preconditioner of the HYPRE v2.10.1 library. This is an example where more in-depth analysis is needed after the initial health examination to clarify which part of the code must be improved.

Assessing the parallelization strategy : Health check step #5

To assess the parallelization strategy of ParFlow as a whole it is necessary to perform steps #3 and #4. Now we are ready to compute the parallel efficiency which is shown in Table 2. We can see, values reduce from 0.93 (1 024 processes) to 0.63 (32 768 processes) where the loss in communication efficiency is the main cause of the reduction in parallel efficiency.

Resource bounds: Health check step #6

Using the idealized weak scaling test case described in Section 3.1, Score-P was used to track memory usage of the test case on JUQUEEN. Due to the idealized behavior (symmetry) of the test case, all MPI ranks needed roughly the same amount of memory. For example, at 1 024 processes, each rank needed roughly 95 MB and at 32 678 processes roughly 325 MB. Memory usage per MPI rank increases with scale as the mesh manager in ParFlow is implemented in such a way that the entire grid information is redundantly stored on each MPI rank. This becomes a scalability breaker for ParFlow as we can see from Figure 9; there is a point at which the memory required will eclipse the memory available (at around 64 000 cores for this test case) and this is due to storage of the grid information for the mesh manager. In ParFlow, the most memory consuming routines are: `GetGridNeighbors`, `PFMGInitInstanceXtra`, `KinSolPC`, and `AllocateVectorData`.

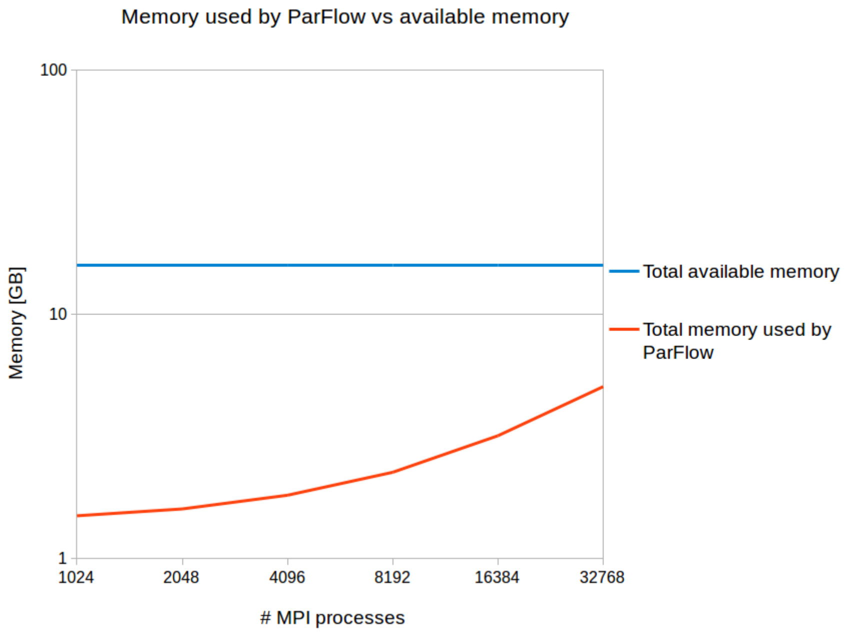


Figure 9. Memory usage of the weak scaling experiment described in Section 3.1 versus the total amount of memory available.

Serial performance: Health check step #7

A Score-P profile measurement with hardware performance counters was used to inspect serial performance (IPC). The serial performance for the test case (Section 3.1) with 1 024 MPI processes shows lower than ideal values of IPC. For example, `solver_loop` has an IPC value of 0.31 out of 2 (the theoretical limit on the Blue Gene/Q platform).

Therefore, additional measurements with hardware counters were collected which show that a significant number of cache misses in L1, stalls, and mis-predicted branches occur in the following routines: `RichardsJacobianEval`, `PhaseRelPerm`, `Saturation`, and `NlFunctionEval`. Since JUQUEEN is based on an in-order instruction execution model, meaning instructions are fetched, executed, and committed in compiler-generated order, in case of an instruction stall, all ensuing instructions will stall as well. Branching on JUQUEEN is therefore very expensive and can cause pipeline stalls. Thus, the aforementioned routines may account for the low IPC values.

3.4 Reproducibility

All simulation runs in the scaling study are separated into different subdirectories for each simulation run. Each subdirectory includes the environment description, the XML scripts used by JUBE, the compilation scripts, the job submission scripts, the job logs, the model scripts, the postprocessing analysis, and a description of the version of ParFlow used along with the ParFlow binary itself. Each directory is self-contained such that the model can be rerun on JUQUEEN without using any other external tools or files. After the simulation is run and the postprocessing step has been executed, the directory is automatically archived for long term storage.

3.5 Outcomes of profiling case study and future developments using RCF

The detailed profiling work illuminated the main bottlenecks to scalability. So that ParFlow can scale to upcoming exascale machines that are capable of at least one Exaflop, or even to petascale (e.g., the full JUQUEEN system), memory use, time spent in communication, and time spent in acquiring the solution for each time step need to be addressed.

To reduce the memory usage and to reduce the time spent in communication, an Adaptive Mesh Refinement (AMR) library, p4est, is currently being implemented into ParFlow to function as the parallel mesh manager. The approach was minimally invasive and preserves most of ParFlow's data structures, the configuration system, and the setup and solver pipeline. The current mesh manager is a barrier to scalability as it requires that all cells store information about every other cell. This is reduced to neighboring cells under p4est, which results in a decrease in memory use (storage reduction) and a decrease in time spent in communication (communication reduced to neighboring cells only), allowing ParFlow to scale over all 458,752 cores on JUQUEEN (Burstedde et al., 2017). Using p4est as the parallel mesh manager has the additional potential benefit of integrating the adaptive mesh refinement functionality into ParFlow in order to address inactive regions (due to heterogeneous forcing, permeability, etc.) causing load imbalances in the real world models.

To further improve simulation run times using ParFlow, the RCF is being used to benchmark different accelerator-enabled numerical libraries, for a simplified version of ParFlow, across different HPC architectures. To reduce time spent in preprocessing model input and postprocessing model output, a NetCDF reader and writer is under development, with testing of this new feature integrated into the RCF. There is still room for improvement with regards to serial performance. However to tackle this problem effectively, more in-depth profiling is needed with the aid of performance analysis engineers. For example, we are currently working in conjunction with performance analysis specialists to identify and refactor individual loops in specific functions for vectorization in order to speedup serial performance. Naturally, we will use our RCF to then validate the effectiveness of these new developments and tuning efforts.

4 Conclusions

Adapting to new developments in HPC architectures, software libraries, and infrastructures, while ensuring reproducibility of simulation and analysis results has become challenging in the field of geoscience. Next generation massively parallel HPC systems require new coding paradigms, and next generation geoscientific models are based on complex model implementations, and profiling, modeling and data processing workflows. Thus there is a strong need for a streamlined approach to model simulation runs, including profiling, porting, and provenance tracking.

In this article, we presented our run control framework as a best practice approach to porting, profiling and provenance tracking. Implementing an RCF using a workflow engine for the complete modeling chain consisting of preprocessing, simulation run, and postprocessing leads to code that can be ported easily and tuned to any platform and combination of compilers, where dependencies are available in module format. Each simulation is self-contained and automatically documented, accounting for provenance tracking, which leads to better supplementary code sharing and ultimately reproducibility. The relevant profiling toolset can be applied on any platform where the toolset is available, leading to identification of bottlenecks, code tuning, refactoring, and ultimately more efficient use of HPC resources. For example, the detailed profiling study of ParFlow led to the identification of bottlenecks and scalability breakers.

The proposed approach helps the novice user as well as the developer and can be embedded into regression testing and a continuous integration approach. Using our RCF, testing, benchmarking, profiling, and running models is less time consuming and more robust than running models in an ad hoc fashion, resulting in more efficient use of HPC resources, more strategic code development, and enhanced data integrity and reproducibility.

Code and data availability. The run control framework, data and version of ParFlow used in this paper are available for download via https://gitlab.maisondelasimulation.fr/EOCOE/Parflow upon request for access. A tarball containing the RCF, data and version of ParFlow relevant to this study has been included as supplementary information.

Appendix A: Profiling tools implemented into the RCF

Table A1 describes the profiling tools which are currently supported by our RCF. New profiling tools can easily be added into the framework by adding to the `profiler.xml` file (see Figure 2).

Appendix B: Description of performance metrics gathered from the automated performance metric workflow

The list of performance metrics gathered in the automated performance metrics workflow (see Figure 2), with an explanation of why these particular metrics are useful is given in the tables below. Table B1 contains general performance metrics and Tables B2 - B5 contain performance metrics pertaining to specific areas such as I/O, communication, memory use, node-level performance, and core-level (serial) performance.

Performance Analysis Tool	Description
Score-P	Score-P is a community-maintained scalable instrumentation and performance measurement infrastructure for parallel codes. It can collect both profiles and event traces. https://www.score-p.org
Scalasca	The Scalasca Trace Tools are a collection of scalable trace-based tools for in-depth analyses of concurrent behavior, in particular regarding wait states in communication and synchronization operations as well as their root causes. Supports Score-P traces since v2.0. https://www.scalasca.org
Cube	Cube is an interactive analysis report explorer for Score-P profiles and Scalasca trace analyis reports. http://www.scalasca.org/software/cube-4.x/
Allinea Performance Report	Allinea Performance Report is a performance tool which provides a high-level overview of the runtime using a single-page report. https://www.allinea.com/products/allinea-performance-reports
Extrae	Extrae is a measurement system which is able to collect traces for use with Paraver. https://tools.bsc.es/extrae
Paraver	Paraver is a flexible and configurable performance analysis tool based on traces collected by the Extrae measurement system. It supports time-line views as well as histogram/statistics views on the trace data. https://tools.bsc.es/paraver
Intel® Advisor	Advisor is a tool to analyze node-level performance issues, in particular regarding code vectorization and threading. https://software.intel.com/en-us/intel-advisor-xe
Darshan	Darshan is tool to capture and characterize the I/O behavior of an application. http://www.mcs.anl.gov/project/darshan-hpc-io-characterization-tool
PAPI	PAPI is a library providing a consistent interface for accessing hardware performance counters of CPUs and other components. While it can be called from application code directly, PAPI is more often used through other performance measurement systems such as Extrae and Score-P. http://icl.utk.edu/papi

Table A1. Description of of the performance analysis tools currently supported by our RCF.

General Performance Metric	Description
time[s]	Total application wall time to use as a reference.
time_io[s]	Average time spent in input/output operations for each rank. When this value is large compared to the overall run time, the application spends a significant amount of time in input/output operations. Further measurements can be taken to illuminate problem areas (see I/O section of Table B2).
time_mpi[s]	Average time spent in MPI for each rank. When this value is large compared to the overall run time, the application spends a significant amount of time in MPI operations. Further measurements can be taken to illuminate problem areas (see MPI section of Table B3).
mem_vs_comp	Memory bound versus compute bound. Memory bound means that the application would be faster if the memory bandwidth was larger and compute bound means that the application would be faster if the CPU was faster. (close to 1.0 means the application is strongly compute bound, close to 2.0 means the application is strongly memory bound).
load_imbalance	Ratio of the load imbalance overhead towards the critical path duration. This ratio signifies the potential for speedup if the load imbalance was non-existent. For example, if a 20% load imbalance is measured, fixing this load imbalance would improve the run time of the code by 20%.

Table B1. Description of general performance analysis metrics currently supported by our RCF.

I/O Performance Metric	Description
io_volume[MB]	Total amount of data in I/O. Can indicate whether I/O is going to be a bottleneck for the application.
io_calls[nb]	Total number of I/O calls. Can indicate whether the I/O subroutines are inefficient.
io_throughput[MB/s]	Speed of I/O. Can indicate whether the HPC architecture is suitable for the application.
avg_io_ac_size[kB]	Average amount of data per I/O call. Can indicate whether performance could be improved by changing data size (coalescing reads/writes).

Table B2. Description of performance analysis metrics pertaining to I/O, currently supported by our RCF.

MPI Performance Metric	Description
num_p2p_calls[nb]	Average number of point-to-point MPI operations per MPI rank. Can indicate inefficiency of point-to-point communication pattern.
p2p_comm_time[s]	Average time spent in point-to-point MPI operations per MPI rank. Can indicate inefficiency of point-to-point communication pattern.
p2p_message_size[kB]	Average size of point-to-point MPI messages per MPI rank. Can indicate whether performance could be improved by changing message size.
num_coll_calls[nb]	Average number of collective MPI operations per MPI rank. Can indicate inefficiency of collective communication pattern.
coll_comm_time[s]	Average time spent in collective MPI operations per MPI rank. Can indicate inefficiency of collective communication pattern.
delay_mpi[s]	Total amount of MPI time spent in waiting caused by inefficient communication patterns. If this value is large, it signifies that the application has a significant amount of delays that cause wait states in MPI operations.
delay_mpi_ratio	Ratio of waiting time caused by MPI to total time spent in MPI. If this value is large, it signifies that the application has a significant amount of delays that cause wait states in MPI operations.

Table B3. Description of performance analysis metrics pertaining to MPI communication, currently supported by our RCF.

Memory Performance Metric	Description
memory_footprint[kB]	Average memory footprint per MPI rank. This metric helps to estimate total amount of main memory that program uses while running.
cache_usage_intensity	Ratio of total number of cache hits to the total number of cache accesses. If this value is small, the application uses the cache inefficiently.

Table B4. Description of performance analysis metrics pertaining to memory use, currently supported by our RCF.

Competing interests. The authors declare that there are no conflicts of interest.

Acknowledgements. The authors gratefully acknowledge the reviewers thoughtful suggestions and comments which led to a greatly improved paper. We also acknowledge computing time granted by the JARA-HPC partition for the project: "Fractal Scaling of Hydrodynamics at the Catchment Scale" on the supercomputer JUQUEEN at Forschungszentrum Jülich. This work was supported by the Energy oriented Centre of Excellence (EoCoE), grant agreement number 76629, funded within the Horizon2020 framework of the European Union and POP (676553), the Centre of Excellence Performance Optimisation and Productivity. In addition we acknowledge the supercomputing support provided to us by the Simulation Laboratory Terrestrial Systems (www.hpsc-terrsys.de/simlab) of the Centre for High Performance Scientific Computing in Terrestrial Systems (Geoverbund ABC/J) and the Jülich Supercomputing Centre (JSC).

Node Performance Metric	Description
time_omp[s]	Total time spent in OpenMP parallel regions. Can indicate load imbalances with regards to inter- and intra-node operations.
omp_ratio[%]	Ratio of the time spent in OpenMP parallel region over the total computation time. Can indicate load imbalances with regards to inter- and intra-node operations.
delay_omp[s]	Total amount of OpenMP synchronization overhead. If this value is large, it signifies that the application has a significant amount of delays that cause wait states in OpenMP constructs.
delay_omp_ratio	Ratio of synchronization overhead time in OpenMP to total time spent in OpenMP. If this value is large, it signifies that the application has a significant amount of delays that cause wait states in OpenMP constructs.

Table B5. Description of performance analysis metrics pertaining to node-level performance currently supported by our RCF.

Core Performance Metric	Description
IPC	Ratio of total instructions executed to the total number of CPU cycles. This metric shows the workload of the CPU. Low values usually indicate the presence of pipeline bubbles and/or cache misses and/or mispredicted branches.
time_no_vec[s]	Wall clock time without compiler vectorization.
vec_eff	Ratio of total wall time of the reference run to the total wall time without vectorization. If this value is low, the application is not vectorized or poorly vectorized.
time_no_fma	Total wall time with disabled “fused multiply/add” (FMA) instructions.
fma_eff	Ratio of total wall time of the reference run to the total wall time without fused-multiply-add operations (FMA). If this value is low, the application does not or use FMA or poorly use FMA operations.

Table B6. Description of performance analysis metrics pertaining core-level performance currently supported by our RCF.

References

Alonso, P., Badia, R. M., Labarta, J., Barreda, M., Dolz, M. F., Mayo, R., Quintana-Orti, E. S., and Reyes, R.: Tools for Power-Energy Modelling and Analysis of Parallel Scientific Applications, in: 2012 41st International Conference on Parallel Processing, pp. 420–429, IEEE, https://doi.org/10.1109/ICPP.2012.57, http://ieeexplore.ieee.org/document/6337603/, 2012.

Ashby, S. F. and Falgout, R. D.: A parallel multigrid preconditioned conjugate gradient algorithm for groundwater flow simulations, 124, 145–159, 1996.

Attig, N., Gibbon, P., and Lippert, T.: Trends in supercomputing: The European path to exascale, Computer Physics Communications, 182, 2041–2046, https://doi.org/10.1016/j.cpc.2010.11.011, http://linkinghub.elsevier.com/retrieve/pii/S0010465510004571, 2011.

Bahra, A.: Managing work flows with ecFlow, ECMWF Newsletter,. Tech. Rep., 129, 30 – 32, 2011.

Balaji, P., Gupta, R., Vishnu, A., and Beckman, P.: Mapping communication layouts to network hardware characteristics on massive-scale Blue Gene systems, Computer Science - Research and Development, 26, 247–256, https://doi.org/10.1007/s00450-011-0168-y, http://link.springer.com/10.1007/s00450-011-0168-y, 2011.

Beanato, G., Loi, I., De Micheli, G., Leblebici, Y., and Benini, L.: Configurable Low-Latency Interconnect for Multi-core Clusters, pp. 107–124, Springer, Berlin, Heidelberg, https://doi.org/10.1007/978-3-642-45073-0_6, https://link.springer.com/chapter/10.1007/978-3-642-45073-0_6, 2013.

Bierkens, M. F. P., Bell, V. A., Burek, P., Chaney, N., Condon, L. E., David, C. H., de Roo, A., Döll, P., Drost, N., Famiglietti, J. S., Flörke, M., Gochis, D. J., Houser, P., Hut, R., Keune, J., Kollet, S., Maxwell, R. M., Reager, J. T., Samaniego, L., Sudicky, E., Sutanudjaja, E. H., van de Giesen, N., Winsemius, H., and Wood, E. F.: Hyper-resolution global hydrological modelling: what is next?, Hydrological Processes, 29, 310–320, https://doi.org/10.1002/hyp.10391, http://doi.wiley.com/10.1002/hyp.10391, 2015.

Breß, S. and Saake, G.: Why it is time for a HyPE, Proceedings of the VLDB Endowment, 6, 1398–1403, https://doi.org/10.14778/2536274.2536325, http://dl.acm.org/citation.cfm?doid=2536274.2536325, 2013.

Brodtkorb, A. R., Dyken, C., Hagen, T. R., Hjelmervik, J. M., and Storaasli, O. O.: State-of-the-art in heterogeneous computing, Scientific Programming, 18, 1–33, https://doi.org/10.3233/SPR-2009-0296, http://dx.doi.org/10.3233/SPR-2009-0296, 2010.

Burstedde, C., Fonseca, J. A., and Kollet, S.: Enhancing speed and scalability of the ParFlow simulation code, arxiv.org, http://arxiv.org/abs/1702.06898, 2017.

Carns, P., Harms, K., Allcock, W., Bacon, C., Lang, S., Latham, R., and Ross, R.: Understanding and Improving Computational Science Storage Access through Continuous Characterization, ACM Transactions on Storage, 7, 1–26, https://doi.org/10.1145/2027066.2027068, http://dl.acm.org/citation.cfm?doid=2027066.2027068, 2011.

Davis, N. E., Robey, R. W., Ferenbaugh, C. R., Nicholaeff, D., and Trujillo, D. P.: Paradigmatic shifts for exascale supercomputing, The Journal of Supercomputing, 62, 1023–1044, https://doi.org/10.1007/s11227-012-0789-3, http://link.springer.com/10.1007/s11227-012-0789-3, 2012.

Eaton, B., Gregory, J., Drach, B., Taylor, K., Hankin, S., Caron, J., Signell, R., Bentley, P., Rappa, G., Höck, H., et al.: NetCDF Climate and Forecast (CF) metadata conventions, 2003.

Eyring, V., Bony, S., Meehl, G. A., Senior, C., Stevens, B., Stouffer, R. J., and Taylor, K. E.: Overview of the Coupled Model Intercomparison Project Phase 6 (CMIP6) experimental design and organisation, Geoscientific Model Development Discussions, 8, 10 539–10 583, https://doi.org/10.5194/gmdd-8-10539-2015, http://www.geosci-model-dev-discuss.net/8/10539/2015/, 2015.

Gasper, F., Goergen, K., Shrestha, P., Sulis, M., Rihani, J., Geimer, M., and Kollet, S.: Implementation and scaling of the fully coupled Terrestrial Systems Modeling Platform (TerrSysMP v1.0) in a massively parallel supercomputing environment - A case study on JUQUEEN (IBM Blue Gene/Q), Geoscientific Model Development, 7, 2531–2543, https://doi.org/10.5194/gmd-7-2531-2014, http://www.geosci-model-dev.net/7/2531/2014/, 2014.

Geimer, M., Wolf, F., Wylie, B. J. N., Ábrahám, E., Becker, D., and Mohr, B.: The Scalasca performance toolset architecture, Concurrency and Computation: Practice and Experience, 22, 702–719, https://doi.org/10.1002/cpe.1556, http://doi.wiley.com/10.1002/cpe.1556, 2010.

Hammond, G. E., Lichtner, P. C., and Mills, R. T.: Evaluating the performance of parallel subsurface simulators: An illustrative example with PFLOTRAN, Water Resources Research, 50, 208–228, https://doi.org/10.1002/2012WR013483, http://doi.wiley.com/10.1002/2012WR013483, 2014.

Han, X., Hendricks Franssen, H.-J., Jiménez Bello, M. Á., Rosolem, R., Bogena, H., Alzamora, F. M., Chanzy, A., and Vereecken, H.: Simultaneous soil moisture and properties estimation for a drip irrigated field by assimilating cosmic-ray neutron intensity, Journal of Hydrology, 539, 611–624, https://doi.org/10.1016/j.jhydrol.2016.05.050, http://www.sciencedirect.com/science/article/pii/S0022169416303171, 2016.

Heinzeller, D., Duda, M. G., and Kunstmann, H.: Towards convection-resolving, global atmospheric simulations with the Model for Prediction Across Scales (MPAS) v3.1: an extreme scaling experiment, Geosci. Model Dev, 9, 77–110, https://doi.org/10.5194/gmd-9-77-2016, www.geosci-model-dev.net/9/77/2016/, 2016.

Hethey, J. M.: GitLab repository management : delve into managing your projects with GitLab, while tailoring it to fit your environment, Packt Pub, http://cds.cern.ch/record/1633720, 2013.

Hindmarsh, A. C., Brown, P. N., Grant, K. E., Lee, S. L., Serban, R., Shumaker, D. E., and Woodward, C. S.: SUNDIALS, ACM Transactions on Mathematical Software, 31, 363–396, https://doi.org/10.1145/1089014.1089020, http://portal.acm.org/citation.cfm?doid=1089014.1089020, 2005.

Hutton, C., Wagener, T., Freer, J., Han, D., Duffy, C., and Arheimer, B.: Most computational hydrology is not reproducible, so is it really science?, Water Resources Research, 52, 7548–7555, https://doi.org/10.1002/2016WR019285, http://doi.wiley.com/10.1002/2016WR019285, 2016.

Hwu, W.-m.: What is ahead for parallel computing, Journal of Parallel and Distributed Computing, 74, 2574–2581, https://doi.org/10.1016/j.jpdc.2014.02.005, http://linkinghub.elsevier.com/retrieve/pii/S0743731514000331, 2014.

IBM: IBM XL C/C++ for Blue Gene/Q: Compiler Reference, http://www-01.ibm.com/support/docview.wss?uid=swg27027065{&}aid=1, 2012.

Irving, D.: A Minimum Standard for Publishing Computational Results in the Weather and Climate Sciences, Bulletin of the American Meteorological Society, 97, 1149–1158, https://doi.org/10.1175/BAMS-D-15-00010.1, http://journals.ametsoc.org/doi/10.1175/BAMS-D-15-00010.1, 2016.

January, C., Byrd, J., Oró, X., and O'Connor, M.: Allinea MAP: Adding Energy and OpenMP Profiling Without Increasing Overhead, in: Tools for High Performance Computing 2014, pp. 25–35, Springer International Publishing, Cham, https://doi.org/10.1007/978-3-319-16012-2_2, http://link.springer.com/10.1007/978-3-319-16012-2{_}2, 2015.

Jin, H., Jespersen, D., Mehrotra, P., Biswas, R., Huang, L., and Chapman, B.: High performance computing using MPI and OpenMP on multi-core parallel systems, Parallel Computing, 37, 562–575, https://doi.org/10.1016/j.parco.2011.02.002, 2011.

John, E. and Rubio, J.: Unique chips and systems, CRC Press, 2008.

Jones, J. E. and Woodward, C. S.: Newton–Krylov-multigrid solvers for large-scale, highly heterogeneous, variably saturated flow problems, Advances in Water Resources, 24, 763–774, https://doi.org/10.1016/S0309-1708(00)00075-0, http://linkinghub.elsevier.com/retrieve/pii/S0309170800000750, 2001.

Keune, J., Kollet, S., Sulis, M., Shresta, P., Görgen, K., and Ohlwein, C.: Implementation of a coupled soil-vegetation-atmosphere system over the European CORDEX domain Hans-Ertel-Centre for Weather Research -Climate Monitoring Branch Centre for High-Performance Scientific Computing in Terrestrial Systems.

Keyes, D. E.: Exaflop/s: The why and the how, Comptes Rendus Mecanique, 339, 70–77, https://doi.org/10.1016/j.crme.2010.11.002, http://linkinghub.elsevier.com/retrieve/pii/S1631072110002032, 2011.

Knüpfer, A., Rössel, C., an Mey, D., Biersdorff, S., Diethelm, K., Eschweiler, D., Geimer, M., Gerndt, M., Lorenz, D., Malony, A. D., Nagel, W. E., Oleynik, Y., Philippen, P., Saviankou, P., Schmidl, D., Shende, S. S., Tschüter, R., Wagner, M., Wesarg, B., and Wolf, F.: Score-P – A Joint Performance Measurement Run-Time Infrastructure for Periscope, Scalasca, TAU, and Vampir, in: Tools for High Performance Computing 2011: Proceedings of the 5th International Workshop on Parallel Tools for High Performance Computing, September 2011, ZIH, Dresden, edited by Brunst, H., Müller, M. S., Nagel, W. E., and Resch, M. M., pp. 79–91, Springer, https://doi.org/10.1007/978-3-642-31476-6_7, 2012.

Kollet, S. J. and Maxwell, R. M.: Integrated surface–groundwater flow modeling: A free-surface overland flow boundary condition in a parallel groundwater flow model, Advances in Water Resources, 29, 945–958, https://doi.org/10.1016/j.advwatres.2005.08.006, http://www.sciencedirect.com/science/article/pii/S0309170805002101, 2006.

Kollet, S. J. and Maxwell, R. M.: Capturing the influence of groundwater dynamics on land surface processes using an integrated, distributed watershed model, Water Resources Research, 44, n/a–n/a, https://doi.org/10.1029/2007WR006004, http://doi.wiley.com/10.1029/2007WR006004, 2008.

Kraus, J., Förster, M., Brandes, T., and Soddemann, T.: Using LAMA for efficient AMG on hybrid clusters, Computer Science - Research and Development, 28, 211–220, https://doi.org/10.1007/s00450-012-0223-3, http://link.springer.com/10.1007/s00450-012-0223-3, 2013.

Kurtz, W., He, G., Kollet, S. J., Maxwell, R. M., Vereecken, H., and Hendricks Franssen, H.-J.: TerrSysMP–PDAF (version 1.0): a modular high-performance data assimilation framework for an integrated land surface–subsurface model, Geoscientific Model Development, 9, 1341–1360, https://doi.org/10.5194/gmd-9-1341-2016, http://www.geosci-model-dev-discuss.net/8/9617/2015/http://www.geosci-model-dev.net/9/1341/2016/, 2016.

Labarta, J., Gi Enez, J., Martínez, E., Gon, P., Servat, H., Llort, G., and Aguilar, X.: Scalability of Visualization and Tracing Tools, 33, 3–0, http://www.fz-juelich.de/nic-series/volume33, 2006.

Leutwyler, D., Fuhrer, O., Lapillonne, X., Lüthi, D., and Schär, C.: Towards European-scale convection-resolving climate simulations with GPUs: a study with COSMO 4.19, Geoscientific Model Development, 9, 3393–3412, https://doi.org/10.5194/GMD-9-3393-2016, http://www.geosci-model-dev.net/9/3393/2016/gmd-9-3393-2016-discussion.html, 2016.

Liu, B., Zydek, D., Selvaraj, H., and Gewali, L.: Accelerating High Performance Computing Applications: Using CPUs, GPUs, Hybrid CPU/GPU, and FPGAs, in: 2012 13th International Conference on Parallel and Distributed Computing, Applications and Technologies, pp. 337–342, IEEE, https://doi.org/10.1109/PDCAT.2012.34, http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6589302, 2012.

Lührs, S., Rohe, D., Frings, W., Thust, K., and Schnurpfeil, A.: Flexible and Generic Workflow Management, International Conference on Parallel Computing 2015, https://doi.org/10.3233/978-1-61499-621-7-431, http://juser.fz-juelich.de/record/808798, 2016.

Manubens-Gil, D., Vegas-Regidor, J., Prodhomme, C., Mula-Valls, O., and Doblas-Reyes, F. J.: Seamless management of ensemble climate prediction experiments on hpc platforms, in: High Performance Computing & Simulation (HPCS), 2016 International Conference on, pp. 895–900, IEEE, 2016.

Maxwell, R. M.: A terrain-following grid transform and preconditioner for parallel, large-scale, integrated hydrologic modeling, Advances in Water Resources, 53, 109–117, https://doi.org/10.1016/j.advwatres.2012.10.001, 2013.

Maxwell, R. M. and Miller, N. L.: Development of a Coupled Land Surface and Groundwater Model, Journal of Hydrometeorology, 6, 233–247, https://doi.org/10.1175/JHM422.1, http://journals.ametsoc.org/doi/abs/10.1175/JHM422.1, 2005.

Maxwell, R. M., Lundquist, J. K., Mirocha, J. D., Smith, S. G., Woodward, C. S., and Tompson, A. F. B.: Development of a Coupled Groundwater–Atmosphere Model, Monthly Weather Review, 139, 96–116, https://doi.org/10.1175/2010MWR3392.1, http://journals.ametsoc.org/doi/abs/10.1175/2010MWR3392.1, 2011.

Maxwell, R. M., Condon, L. E., and Kollet, S. J.: A high-resolution simulation of groundwater and surface water over most of the continental US with the integrated hydrologic model ParFlow v3, Geoscientific Model Development, 8, 923–937, https://doi.org/10.5194/gmd-8-923-2015, http://www.geosci-model-dev.net/8/923/2015/, 2015.

Meadows, L.: Experiments with WRF on Intel many integrated core (Intel MIC) architecture, in: OpenMP in a Heterogeneous World, pp. 130–139, Springer, 2012.

Mills, R. T., Lu, C., Lichtner, P. C., and Hammond, G. E.: Simulating subsurface flow and transport on ultrascale computers using PFLOTRAN, Journal of Physics: Conference Series, 78, 012 051, https://doi.org/10.1088/1742-6596/78/1/012051, http://stacks.iop.org/1742-6596/78/i=1/a=012051?key=crossref.44db1960f3a116e6c15b5393a40ec03d, 2007.

Minden, V., Smith, B., and Knepley, M. G.: Preliminary Implementation of PETSc Using GPUs, pp. 131–140, Springer Berlin Heidelberg, https://doi.org/10.1007/978-3-642-16405-7_7, https://link.springer.com/chapter/10.1007/978-3-642-16405-7_7, 2013.

Moore, S., Terpstra, D., London, K., Mucci, P., Teller, P., Salayandia, L., Bayona, A., and Nieto, M.: PAPI deployment, evaluation, and extensions, in: 2003 User Group Conference. Proceedings, pp. 349–353, IEEE, https://doi.org/10.1109/DODUGC.2003.1253415, http://ieeexplore.ieee.org/document/1253415/.

Navarra, A., Kinter, J. L., and Tribbia, J.: Crucial Experiments in Climate Science, Bulletin of the American Meteorological Society, 91, 343–352, https://doi.org/10.1175/2009BAMS2712.1, http://journals.ametsoc.org/doi/abs/10.1175/2009BAMS2712.1, 2010.

Nerger, L. and Hiller, W.: Software for ensemble-based data assimilation systems—Implementation strategies and scalability, Computers {&} Geosciences, 55, 110–118, https://doi.org/10.1016/j.cageo.2012.03.026, http://www.sciencedirect.com/science/article/pii/S0098300412001215, 2013.

Oliver, H. J., Shin, M., Fitzpatrick, B., Clark, A., Sanders, O., M214089, Smout-Day, K., Matthews, D., Wales, S., Osprey, A., Reinecke, A., Williams, J., Kinoshita, B. P., Pulo, K., and Valters, D.: Cylc/Cylc: Cylc-7.3.0, https://doi.org/10.5281/ZENODO.545663, https://zenodo.org/record/545663#.WRLKLjclHCI, 2017.

Petersen, L., Orchard, D., and Glew, N.: Automatic SIMD vectorization for Haskell, ACM SIGPLAN Notices, 48, 25–36, 2013.

Prein, A. F., Langhans, W., Fosser, G., Ferrone, A., Ban, N., Goergen, K., Keller, M., Tölle, M., Gutjahr, O., Feser, F., Brisson, E., Kollet, S., Schmidli, J., van Lipzig, N. P. M., and Leung, R.: A review on regional convection-permitting climate modeling: Demonstrations, prospects, and challenges, Reviews of Geophysics, 53, 323–361, https://doi.org/10.1002/2014RG000475, http://doi.wiley.com/10.1002/2014RG000475, 2015.

Rane, A., Krishnaiyer, R., Newburn, C. J., Browne, J., Fialho, L., and Matveev, Z.: Unification of Static and Dynamic Analyses to Enable Vectorization, pp. 367–381, Springer, Cham, https://doi.org/10.1007/978-3-319-17473-0_24, http://link.springer.com/10.1007/978-3-319-17473-0_24, 2015.

Rosas, C., Giménez, J., and Labarta, J.: Scalability prediction for fundamental performance factors, Supercomputing Frontiers and Innovations, 1, http://superfri.org/superfri/article/view/7, 2014.

Ruti, P. M., Somot, S., Giorgi, F., Dubois, C., Flaounas, E., Obermann, A., Dell'Aquila, A., Pisacane, G., Harzallah, A., Lombardi, E., Ahrens, B., Akhtar, N., Alias, A., Arsouze, T., Aznar, R., Bastin, S., Bartholy, J., Béranger, K., Beuvier, J., Bouffies-Cloché, S., Brauch, J., Cabos, W., Calmanti, S., Calvet, J.-C., Carillo, A., Conte, D., Coppola, E., Djurdjevic, V., Drobinski, P., Elizalde-Arellano, A., Gaertner, M., Galàn, P., Gallardo, C., Gualdi, S., Goncalves, M., Jorba, O., Jordà, G., L'Heveder, B., Lebeaupin-Brossier, C., Li, L., Liguori, G., Lionello, P., Macià̀s, D., Nabat, P., Önol, B., Raikovic, B., Ramage, K., Sevault, F., Sannino, G., Struglia, M. V., Sanna, A., Torma, C., and Vervatis, V.: Med-CORDEX Initiative for Mediterranean Climate Studies, Bulletin of the American Meteorological Society, 97, 1187–1208, https://doi.org/10.1175/BAMS-D-14-00176.1, http://journals.ametsoc.org/doi/10.1175/BAMS-D-14-00176.1, 2016.

Saviankou, P., Knobloch, M., Visser, A., and Mohr, B.: Cube v4: From Performance Report Explorer to Performance Analysis Tool, Procedia Computer Science, 51, 1343–1352, https://doi.org/10.1016/j.procs.2015.05.320, 2015.

Schwitalla, T., Bauer, H.-S., Wulfmeyer, V., and Warrach-Sagi, K.: Continuous high resolution mid-latitude belt simulations for July{&}amp;ndash;August 2013 with WRF, Geoscientific Model Development Discussions, pp. 1–46, https://doi.org/10.5194/gmd-2016-195, http://www.geosci-model-dev-discuss.net/gmd-2016-195/, 2016.

Shrestha, P., Sulis, M., Masbou, M., Kollet, S., and Simmer, C.: A scale-consistent Terrestrial Systems Modeling Platform based on COSMO, CLM and ParFlow., Monthly Weather Review, 142, 3466–3483, https://doi.org/10.1175/MWR-D-14-00029.1, http://dx.doi.org/10.1175/MWR-D-14-00029.1, 2014.

Smari, W. W., Bakhouya, M., Fiore, S., and Aloisio, G.: New advances in High Performance Computing and simulation: parallel and distributed systems, algorithms, and applications, Concurrency and Computation: Practice and Experience, 28, 2024–2030, https://doi.org/10.1002/cpe.3774, http://doi.wiley.com/10.1002/cpe.3774, 2016.

Stodden, V., McNutt, M., Bailey, D. H., Deelman, E., Gil, Y., Hanson, B., Heroux, M. A., Ioannidis, J. P. A., and Taufer, M.: Enhancing reproducibility for computational methods, Science, 354, 1240–1241, https://doi.org/10.1126/science.aah6168, http://www.sciencemag.org/cgi/doi/10.1126/science.aah6168, 2016.

Zhukov, I., Feld, C., Geimer, M., Knobloch, M., Mohr, B., and Saviankou, P.: Scalasca v2: Back to the Future, in: Tools for High Performance Computing 2014, pp. 1–24, Springer International Publishing, Cham, https://doi.org/10.1007/978-3-319-16012-2_1, http://link.springer.com/10.1007/978-3-319-16012-2_1, 2015.