Journal of
open research software

Cohen, J et al 2014 Simplifying the Development, Use and Sustainability of HPC Software. *Journal of Open Research Software,* 2(1): e16, pp. 1-6, DOI: http://dx.doi.org/10.5334/jors.az

ISSUES IN RESEARCH SOFTWARE

Simplifying the Development, Use and Sustainability of HPC Software

Jeremy Cohen[*], Chris Cantwell[†], Neil Chue Hong[‡], David Moxey[§], Malcolm Illingworth[‖], Andrew Turner[‖], John Darlington[*] and Spencer Sherwin[§]

Developing software to undertake complex, compute-intensive scientific processes requires a challenging combination of both specialist domain knowledge and software development skills to convert this knowledge into efficient code. As computational platforms become increasingly heterogeneous and newer types of platform such as Infrastructure-as-a-Service (IaaS) cloud computing become more widely accepted for high-performance computing (HPC), scientists require more support from computer scientists and resource providers to develop efficient code that offers long-term sustainability and makes optimal use of the resources available to them. As part of the libhpc stage 1 and 2 projects we are developing a framework to provide a richer means of job specification and efficient execution of complex scientific software on heterogeneous infrastructure. In this updated version of our submission to the WSSSPE13 workshop at SuperComputing 2013 we set out our approach to simplifying access to HPC applications and resources for end-users through the use of flexible and interchangeable software components and associated high-level functional-style operations. We believe this approach can support sustainability of scientific software and help to widen access to it.

Keywords: HPC; software deployment; workflows; software components; component metadata

1 Introduction

Building scientific software for use on HPC platforms can be a complex process bringing together the specialist domain knowledge of the scientists who are likely to be the end-users of the resulting software, method developers, computer scientists and resource providers. By releasing the tight coupling that often exists between the different entities in the development of HPC software and replacing this with a set of less interdependent processes, each entity is able to focus on their specific area of expertise. Our approach uses metadata and software components combined with functional constructs – coordination forms [1] – for specifying component orchestration. Within components, software contains wrappers with clearly defined interfaces and supporting metadata structures against which external code can be developed by third parties, reducing the need for the detailed interactions that can often be required between users and developers to produce efficient code.

In this paper we set out our views on the challenges of ensuring ease of access to and sustainability of scientific HPC software. Our approach is based on work carried out in the libhpc project [2], which is developing a framework to provide a richer means of job specification and efficient execution of complex scientific software on heterogeneous infrastructure, and on previous material presented in [3]. Our motivation, supported through the development of simplified, application or domain-specific user interfaces, is the desire to make it easier for end-users to both describe the tasks they want to undertake and to make use of a wider range of computational infrastructure, in a more streamlined manner. The user interfaces provide a means for users to leverage the framework in order to run applications on different types of hardware, without the need to have detailed knowledge of how this hardware operates.

We consider that software development can be simplified by allowing developers to focus on working within their domain, with fewer cross-domain interactions. This is supported by the use of software components that aid encapsulation of software processes with clearly-defined interfaces. We believe that a flexible model using software components and high-level functional constructs for component orchestration can help to incentivise developers to extend their code with new features and support for new

[*] Department of Computing, Imperial College London, London, UK
jeremy.cohen@imperial.ac.uk

[†] National Heart & Lung Institute, Imperial College London, London, UK

[‡] Software Sustainability Institute, University of Edinburgh, Edinburgh, UK

[§] Department of Aeronautics, Imperial College London, London, UK

[‖] EPCC, University of Edinburgh, Edinburgh, UK

Corresponding author: Jeremy Cohen

hardware platforms. In turn, this should help to support long-term sustainability of software.

Section 2 provides an overview of related work and Section 3 then sets out our position on the challenges of improving scientific software with respect to development, user accessibility and flexibility. Section 4 discusses sustainability of HPC applications and Section 5 looks at libhpc's use of abstractions and metadata with conclusions provided in Section 6.

2 Related work

Extensive research has been undertaken to make software easier to develop and use, particularly as new computing hardware and infrastructure patterns emerge. There are research programmes that have focused on linking application scientists with computer scientists, funding a range of projects to assist with optimising specific codes, supporting frameworks or the underlying infrastructure on which these codes are run.

Software efforts, such as in the area of workflows, aim to simplify the use of distributed resources to run multiple codes or tools that may previously have been scripted locally. Environments such as Taverna [4] provide a means of executing workflows consisting of multiple components that may be available locally or as remote Web Services. In addition to generic workflow systems, many systems have been developed to assist users in specific domains. For example, in Bioinformatics, systems such as Galaxy [5] or VisTrails [6] provide domain-specific features to improve the user experience.

With the emergence of cloud computing, including IaaS public cloud platforms such as Amazon EC2 [7] or RackSpace [8] and private cloud frameworks such as OpenStack [9], access to large-scale, remote, distributed infrastructure has become much easier. Other types of architectures such as GPUs and FPGAs provide many opportunities for improving code performance but at the cost of the complexity of porting or building new code. Heterogeneous hardware can also require learning different development approaches so frameworks such as OpenCL™ [10], which provides a C-based language for developing cross-platform code, and OpenACC [11], which uses compiler directives to specify code that should be executed on alternative hardware, have emerged to provide a common approach to developing code that can be executed on different platforms. Similarly, OP2 [12] is a framework for running unstructured grid applications on multiple cores of either GPUs or CPUs. Compile-time auto-tuning and runtime optimisation offer the potential of supporting a much wider range of developers. Auto-tuning can be applied in the context of libraries that generate optimised code at build time to undertake their specific functionality, for example, the Optimised Sparse Kernel Interface (OSKI) Library [13] that optimises code for sparse matrix operations. Auto-tuning compilers such as Milepost GCC [14] offer a more general option using advanced functionality to produce compiled code that is optimised for the platform that they are building on.

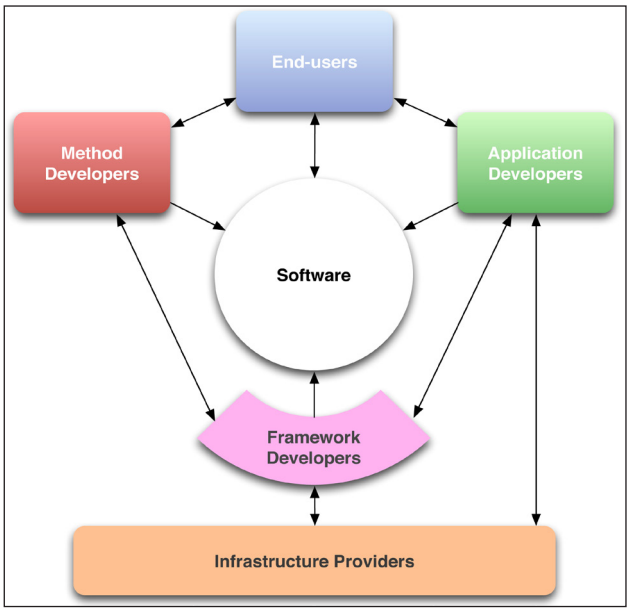

Figure 1: An example of the complex relationships between the different groups involved in HPC software development and use.

3 Improving scientific software

3.1 Overview and Challenges

What constitutes improvement in computer software? For some individuals it is likely to be better performance, for others it may be ease of use while others may be interested in additional features, greater extensibility or options for customisation. For scientific software, improved accuracy, more realistic models or more advanced algorithms may be of importance. What constitutes improvement is also likely to differ depending on the role of the individual in question. **Figure 1** illustrates the different roles involved in software development and use that we consider in our work. The diagram provides an example of the interactions between the entities and their close relationships when building scientific applications.

An aim of libhpc is to improve the experience for end-users who want to run scientific codes but may not have extensive knowledge of high performance computing platforms or be familiar/comfortable with command-line interaction with computer systems. At present, it may be necessary for a new user of an open source HPC application to build the code from source, install this code on a remote hardware platform and then run the code via a command line interface. Users may well need to obtain support from computer scientists and from the providers of their selected target resources in order to get code running, especially when they intend to use external computational platforms operated by third parties. This can be addressed using high-level graphical interfaces built on middleware which abstracts away the complexity of a heterogeneous fabric of resources from the end-user. An example of this is provided in the development of Nekkloud [15], a web-based environment for running finite element jobs using the Nektar++ [16] finite element framework.

Given the different user profiles shown in **Figure 1**, this work demonstrates how a web-based graphical interface

can offer end-users an easier approach to running applications and a more effective means to target different types of hardware resource. The complexity of preparing the hardware and software and packaging this into an easily accessible format is handled by individuals with expertise in these tasks and is hidden from the end-user. The middleware is built by framework developers who work closely with infrastructure providers and developers to automate the software deployment process. Users then simply select their requirements in the web-based interface, provide their input data and then submit the job. The current target for this work is scientific applications and the associated scientists who will have knowledge of a particular domain and may have experience of using domain-specific software packages. By providing an abstraction layer above the computing platforms we aim to ensure that scientists retain some control and flexibility over how they execute their software while also gaining the ability to much more easily target different computational platforms.

The way that code is built is key in ensuring efficiency, long-term sustainability and maintainability of software. Our approach differs from existing component-based frameworks in the way that we use metadata describing both software and hardware and the way that this metadata is used as part of a mapping process to identify the most effective hardware to use to run a given component-based application on a per-run basis. This approach also allows management of data across components and resources ensuring that application data can be handled according to requirements specified in component metadata and through the dynamic orchestration of the execution process to ensure performance and efficiency.

Challenges in building a framework for efficient deployment of software include the complexity of many scientific methods and algorithms and the need to maintain scientists' and method developers' understanding of these to allow selection of suitable computational platforms. Method developers can produce high-quality code but their detailed understanding of the science that is mapped into this code is often lost and can be almost impossible to recreate. This results in code that can be difficult to maintain and extend and that lacks portability. We strongly believe that by wrapping code in components and augmenting these components with metadata that provides details of how and why code is designed and built in a particular way, it is possible to provide long-term benefits to both developers and users. Software components promote code re-use and can simplify the optimisation of individual elements of complex scientific code. The use of high-level functional constructs to orchestrate these components further supports optimisation and efficient execution of applications making it easy to dynamically select optimal components shortly before application run-time.

3.2 Software communities

While frameworks such as libhpc can provide ways to more easily describe complex computational jobs and target a range of infrastructure, the code providing the scientific methods, and the code of the framework itself, still needs to be maintained. One of the most practical ways to build a critical mass of interest and support for a code and, hence, the potential for longevity, is to encourage the establishment of *communities* around particular codebases or projects. The members of the community contribute to the development and maintenance of the software in a distributed, yet coordinated, fashion. Such an approach distributes knowledge regarding all aspects of the code, spanning the methods, optimisation and deployment across the community such that the loss of any one member is far less likely to hamper the process of maintaining knowledge and understanding of the code in the long-term.

Communities can work well where a large number of people have a vested interest in a particular tool. There are many such examples amongst open source projects, hosted on systems such as SourceForge and GitHub, which rely on communities of developers and users. In the case of large and high-profile projects, communities can be very powerful, often taking on extensive project management and development tasks and providing a means for discussion spanning all stakeholders. In contrast, community-building around small-scale scientific projects in a narrow application domain can be challenging due to the comparatively small size of the total user community, meaning it is harder to attain the critical mass of interest to seed the development of a supporting community. In niche areas, maintaining the interest and engagement of the community is key, especially when community members can choose to devote their time to another project without warning.

The complexity of HPC codes means that they often require more experienced developers and a greater investment of time. Support of these applications is often funded through members of related research projects or users in industry contributing time to a project, but this means that smaller-scale users are then reliant on these groups to keep the application up to date and to fix bugs. In general, the most successful open source projects tend to be those that appeal to the widest range of potential users, for example the Mozilla Foundation [17] projects such as the Firefox web browser. There are examples of domain-specific scientific open source tools that have built a community to sustain and extend them, for example OpenFOAM [18], however, this process often relies on the availability of funding to seed the development process until a supporting community has been formed.

4 Sustaining HPC applications

Managing the long-term sustainability of software is a particular challenge in the case of open source software that is made available to users at no cost. While models such as paid-for support are already widely used and can help to provide funding to ensure a core team of developers maintain an application, this is likely to be more of a challenge for applications that have a small user base. Where an application relies on being highly optimised to specific hardware, there is a further need to

ensure reliable, ongoing maintenance programmes exist in order to take advantage of the latest technology. For example, GPGPUs provide potential for massively parallel computation but implementing code for them is challenging and can be time consuming. This can result in a situation where specialist applications with the greatest need for ongoing maintenance are often those that have the least chance of drawing in the necessary support to achieve this.

As new computational models, such as IaaS clouds, and novel hardware, such as FPGAs, become more widely used, we believe that there is an increasingly urgent need for more advanced approaches to managing and maintaining software to ensure sustainability. Many of the aims and approaches described in this paper for improving access to scientific HPC software and for simplifying the use of heterogenous hardware are brought together in the libhpc framework [2].

Unlike systems such as OP2 [12] or FEn-iCS [19], the libhpc approach does not seek to generate optimised code for applications. Instead it still relies on platform-specific code being built by experienced developers. However, middleware is used to intelligently determine the most suitable resources in a heterogeneous environment to be used for running a user's job and then select the most appropriate code implementation to ensure efficient job execution. Libhpc therefore takes a higher-level approach than that used by code generation systems and these systems may still be used to help develop the underlying code used by libhpc. We do not claim that our approach reduces the amount of work that developers need to carry out but it is considered that the de-coupling of entities in the development chain imposes fewer dependencies on the development process for individual developers. This allows them to concentrate on their core areas of expertise and should, ultimately, make the overall process of developing optimised elements of an application more straightforward.

The efficient targeting of code to resources and increasing availability of remote infrastructure cloud platforms provides users with much greater flexibility than if they were restricted to their own local resources. This in turn incentivises users to work with software that supports the middleware and should further motivate developers to extend code to target new platforms because there is a much greater chance of user uptake of the latest code implementations.

5 Abstractions and Metadata

The libhpc framework is designed to use metadata and abstract software components to enable the specification of applications without the requirement for defining a specific, concrete, code implementation at the time of application definition. Abstract software components define the capabilities of a component without providing a specific code implementation. Specialisations of the abstract component may exist for different hardware platforms containing a code implementation optimised to the specific platform.

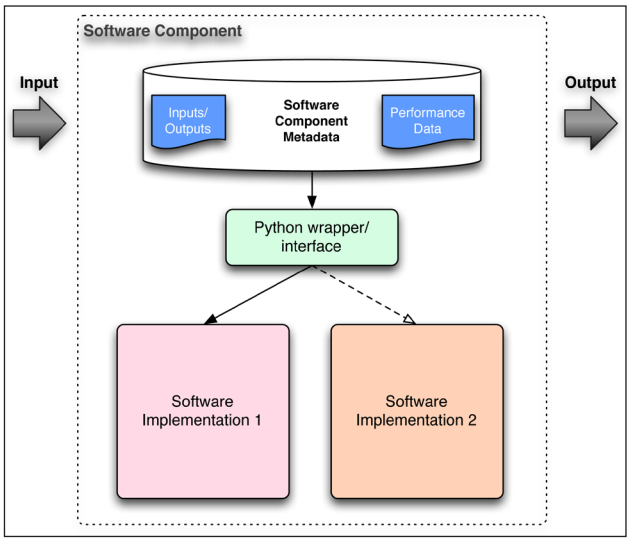

Figure 2: Structure of a libhpc component.

Component metadata is stored within a component repository. In the current approach, component metadata has a pointer to a Python wrapper which is used to execute the component's underlying code which may itself be Python code or be provided within a library or executable that has been written in some other language (see **Figure 2**). This allows developers to produce new implementations of the functionality of a given component that may be optimised to a different type of hardware platform. The new implementations can then be registered with the component repository along with a specialisation of the original component containing metadata specific to the new implementation. Components form trees with a separate tree for each component type. The more abstract instances of a component are higher up the tree with the lower components in the tree having more detailed functionality specified by their metadata. Leaf nodes in the component tree contain a specific code implementation or a pointer to the code. While it is accepted that the provision of metadata and code wrappers imposes additional requirements on developers, we consider that the cost of these additional requirements is acceptable in the context of the end-user benefits and the flexibility provided by the framework.

For example, the storage of metadata and specialisation information for components provides a derivation history that lets us understand how components have been extended and where functionality has been added or changed. This is particularly important in recording the developer knowledge that is invested in the code development process. As discussed in [3], maintaining this information is important in ensuring the long-term sustainability of software.

Co-ordination forms, the functional constructs used to specify component orchestration, can also have multiple implementations. The ability to select between a set of alternative implementations for both software components, and the control structures that combine them, offers significant flexibility in how applications are built, maintained and extended. The selection of these

alternatives is handled by an intelligent mapper that can identify the most suitable software implementation(s) to address a user's requirements. This provides users with the ability to undertake computations that may previously not have been possible without extensive communication with developers and resource providers.

6 Conclusions

We have described an approach for improving the usability and sustainability of scientific software for a range of different stakeholders based on our experiences in the libhpc project and related work. The focus has been on managing sustainability through decoupling the dependencies that exist between entities in the traditional development lifecycle for scientific HPC applications. By capturing as much metadata as possible about a user's requirements and the capabilities of hardware and software, advanced middleware can be provided to compose components and identify suitable target hardware platforms for running code. This provides end-users with much more flexibility in the types of hardware that are accessible to them and the overall experience that they have in developing code themselves.

We summarise our position and the key points and lessons learned from our work as follows:

- Complexity in HPC codes often stems from a distributed development process and the interactions between different entities.
- Logically separating the tasks undertaken by the different entities can reduce complexity and allow a more structured and sustainable development process.
- Scientific code is generally an unsustainable way to provide long-term preservation of the clearly structured processes and concepts it is used to represent.
- Well-defined, higher-level representations of scientific processes should be stored as metadata alongside code in order to simplify software maintenance and extension.
- Domain-specific user interfaces enhance the usability of scientific software and can be developed to provide transparent access to a range of computational platforms.

As we continue our work in the libhpc stage 2 project we are implementing more of the framework and developing demonstrators to show how the approaches discussed here can be realised in different scientific domains. It is hoped that this work will serve to support users in a range of fields who are part of the project and to provide us with valuable feedback to assist in optimising our approaches to improving scientific software for those who build and use it.

7 Acknowledgements

The authors would like to thank the UK Engineering and Physical Sciences Research Council (EPSRC) for funding the *libhpc: Intelligent Component-based Development of HPC Applications* stage 1 (EP/I030239/1) and stage 2 (EP/K038788/1) projects.

References

1. **Darlington, J, Guo, Y, To, H W** and **Yang, J** 1995 Functional Skeletons for Parallel Coordination. In: EURO-PAR'95 Parallel Processing. Springer-Verlag, pp. 55– 69. DOI: http://dx.doi.org/10.1007/BFb0020455
2. libhpc: Intelligent Component-based Development of HPC Applications. Available at: http://www.imperial.ac.uk/lesc/projects/libhpc [Last accessed 03 April 2014].
3. **Cohen, J, Darlington, J, Fuchs, B, Moxey, D, Cantwell, C, Burovskiy, P,** et al 2012 libHPC: Software sustainability and reuse through metadata preservation. In: First Workshop on Maintainable Software Practices in e-Science. Chicago, IL, USA. Available at: http://www.software.ac.uk/sites/default/files/softwarepractice2012_submission_8.pdf
4. **Wolstencroft, K,** et al. 2013 The Taverna workflow suite: designing and executing workflows of Web Services on the desktop, web or in the cloud. *Nucleic Acids Research*, 41(W1): W557-W561. DOI: http://dx.doi.org/10.1093/nar/gkt328
5. **Giardine, B,** et al 2005 Galaxy: a platform for interactive large-scale genome analysis. *Genome Research,* 15(10): 1451–1455. DOI: http://dx.doi.org/10.1101/gr.4086505
6. **Callahan, S P, Freire, J, Santos, E, Scheidegger, C E, Silva, C T** and **Vo, H T** 2006 VisTrails: visualization meets data management. In: Proceedings of the 2006 ACM SIGMOD International Conference on Management of Data. SIGMOD '06. New York, NY, USA: ACM, pp. 745–747. DOI: http://doi.acm.org/10.1145/1142473.1142574.
7. **Amazon Web Services, Inc** 2013 Amazon Elastic Compute Cloud (Amazon EC2). Available at: http://aws.amazon.com/ec2 [Last accessed 01 September 2013].
8. **Rackspace Limited** 2013 Cloud Overview — Rackspace Hosting. Available at: http://www.rackspace.co.uk/cloud/ [Last accessed 01 September 2013].
9. **OpenStack Open Source Cloud Computing Software** 2013. Available at: http://www.openstack.org/ [Last accessed 01 September 2013].
10. **Khronos Group** 2013 OpenCL™– The open standard for parallel programming of heterogeneous systems. Available at: http://www.khronos.org/opencl/ [Last accessed 01 September 2013].
11. **OpenACC Home** 2013 Available at: http://www.openacc-standard.org/ [Last accessed 01 September 2013].
12. **Mudalige, G R, Giles, M B, Reguly, I, Bertolli, C** and **Kelly, P H J** 2012 OP2: An active library framework for solving unstructured mesh-based applications on multi-core and many-core architectures. In: Innovative Parallel Computing (InPar), San Jose, CA on 13-14 May 2012, pp. 1–12. DOI: http://dx.doi.org/10.1109/InPar.2012.6339594.
13. **Vuduc, R, Demmel, J W** and **Yelick K A** 2005 OSKI: A library of automatically tuned sparse matrix kernels. *Journal of Physics: Conference Series*, 16(1): 521–530. Available at: http://stacks.iop.org/1742-6596/16/i=1/a=071

14. **Fursin, G,** et al 2011 Milepost GCC: Machine Learning Enabled Self-tuning Compiler. *International Journal of Parallel Programming,* 39(3): 296–327. DOI: http://dx.doi.org/10.1007/s10766-010-0161-2
15. **Cohen, J, Moxey, D, Cantwell, C, Burovskiy, P, Darlington, J** and **Sherwin, S J** 2013 Nekkloud: A software environment for high-order finite element analysis on clusters and clouds. In: IEEE International Conference on Cluster Computing, IEEE'13. DOI: http://dx.doi.org/10.1109/CLUSTER.2013.6702616.
16. **Nektar**++ 2013 Nektar++: An efficient h to p finite element framework. Available at: http://www.nektar.info/ [Last accessed 07 April 2014].
17. **Home of the Mozilla Project** 2014 Available at: http://www.mozilla.org/ [Last accessed 03 April 2014].
18. **OpenFOAM®Foundation** 2014 The OpenFOAM Foundation. Available at: http://www.openfoam.org/ [Last accessed 03 April 2014].
19. **Logg, A, Mardal, K A** and **Wells, G N** 2012 Finite Element Assembly. In: Logg, A, Mardal, K A and Wells, G N (eds.) *Automated Solution of Differential Equations by the Finite Element Method.* Lecture Notes in Computational Science and Engineering, vol. 84 Springer. pp. 141–146. DOI: http://dx.doi.org/10.1007/978-3-642-23099-8_6.

How to cite this article: Cohen, J, Cantwell, C, Chue Hong, N, Moxey, D, Illingworth, M, Turner, A, Darlington, J and Sherwin, S 2014 Simplifying the Development, Use and Sustainability of HPC Software. *Journal of Open Research Software,* 2(1): e16, pp. 1-6, DOI: http://dx.doi.org/10.5334/jors.az

Published: 9 July 2014

]u[*Journal of Open Research Software* is a peer-reviewed open access journal published by Ubiquity Press.

OPEN ACCESS