ARTICLE IN PRESS

Computer Physics Communications ▮ (▮▮▮▮) ▮▮▮–▮▮▮

Contents lists available at ScienceDirect

Computer Physics Communications

journal homepage: www.elsevier.com/locate/cpc

# FabSim: Facilitating computational research through automation on large-scale and distributed e-infrastructures☆

Derek Groen [a,b,*], Agastya P. Bhati [a], James Suter [a], James Hetherington [c], Stefan J. Zasada [a], Peter V. Coveney [a,**]

[a] Centre for Computational Science, University College London, 20 Gordon street, London, WC1H 0AJ, United Kingdom
[b] Department of Computer Science, Brunel University, St John's Building, Kingston Lane, Uxbridge, UB8 3PH, United Kingdom
[c] Research Software Development Group, University College London, Podium Building, 1 Eversholt Street, London, NW1 2DN, United Kingdom

ARTICLE INFO

*Article history:*
Received 14 December 2015
Received in revised form
16 May 2016
Accepted 19 May 2016
Available online xxxx

*Keywords:*
Automation
Workflows
Distributed computing
Software
Bloodflow modelling
Molecular dynamics
Multiscale modelling
Clay-polymer nanocomposites

ABSTRACT

We present FabSim, a toolkit developed to simplify a range of computational tasks for researchers in diverse disciplines. FabSim is flexible, adaptable, and allows users to perform a wide range of tasks with ease. It also provides a systematic way to automate the use of resources, including HPC and distributed machines, and to make tasks easier to repeat by recording contextual information. To demonstrate this, we present three use cases where FabSim has enhanced our research productivity. These include simulating cerebrovascular bloodflow, modelling clay-polymer nanocomposites across multiple scales, and calculating ligand–protein binding affinities.

**Program summary**

*Program title:* FabSim

*Catalogue identifier:* AFAO_v1_0

*Program summary URL:* http://cpc.cs.qub.ac.uk/summaries/AFAO_v1_0.html

*Program obtainable from:* CPC Programme Library, Queen's University, Belfast, N. Ireland

*Licensing provisions:* BSD 3-Clause

*No. of lines in distributed program, including test data, etc.:* 268282

*No. of bytes in distributed program, including test data, etc.:* 2791792

*Distribution format:* tar.gz

*Programming language:* Python.

*Computer:* PC or Mac.

*Operating system:* Unix, OSX.

*RAM:* 1 Gbytes

*Classification:* 3, 4, 6.5.

*External routines:* NumPy, SciPy, Fabric (1.5 or newer), PyYaml

*Nature of problem:*

Running advanced computations using remote resources is an activity that requires considerable time and human attention. These activities, such as organizing data, configuring software and setting up individual runs often vary slightly each time they are performed. To lighten this burden, we required an approach that introduced little burden of its own to set up and adapt, beyond which very substantial productivity ensues.

☆ This paper and its associated computer program are available via the Computer Physics Communication homepage on ScienceDirect (http://www.sciencedirect.com/science/journal/00104655).

* Corresponding author at: Centre for Computational Science, University College London, 20 Gordon street, London, WC1H 0AJ, United Kingdom.
** Corresponding author.
*E-mail addresses:* Derek.Groen@brunel.ac.uk (D. Groen), p.v.coveney@ucl.ac.uk (P.V. Coveney).

http://dx.doi.org/10.1016/j.cpc.2016.05.020

*Solution method:*

We present a toolkit which helps to simplify and automate the activities which surround computational science research. FabSim is aimed squarely at the experienced computational scientist, who can use the command line interface and a system of modifiable content to quickly automate sets of research tasks.

*Restrictions:*

FabSim relies on a command-line interface, and assumes some level of scripting knowledge from the user.

*Unusual features:*

FabSim has a proven track record of being easy to adapt. It has already been extensively adapted to facilitate leading research in the modelling of bloodflow, nanomaterials, and ligand–protein binding.

*Running time:*

FabSim can be used interactively, typically requiring a few seconds to perform a basic task.



## 1. Introduction

Research based on computational science and technology continues to advance at a rapid pace, driven in part by the continual evolution, and recent diversification, of computing infrastructures. At the top end, supercomputers are both highly parallel (at time of writing, the number one supercomputer has 3.1 million cores) and highly heterogeneous (four of the top ten supercomputers feature accelerators). Because computing infrastructures are growing in parallelism and becoming more diverse, we require sophisticated computational techniques to take full advantage of the power available.

Recent advances in modelling methods, such as ensemble and multiscale computing, allow us to solve complex problems more efficiently using high-end infrastructures [1–4]. These techniques tend to require users to construct, execute, validate, analyse and curate a number of different models (and model executions) for each computation. Care must be taken when performing these tasks, as a simple mistake can render the full computation useless. For example, computations may produce incorrect results, or produce too little information to allow for reproduction or replication.

In an era where compute resources are arguably easier to obtain than human resources, this requirement for continued human attention can become a bottleneck, limiting the pace of computational research to the number of person hours invested in it. Indeed, within our own research group we have realized that much of our daily work was spent on the attention-requiring activities that accompany the need to run ensemble or multiscale computations. To lighten this burden, we sought an approach that introduced little burden of its own to set up and adapt, beyond which very substantial productivity benefits ensue by "automating away" routine activities.

Here we present our approach, based on FabSim, for managing computations and automating the research tasks that accompany them. FabSim includes a software toolset, as well as a set of best practices which serve to aid researchers in maintaining a computational environment which is simple to use, navigate, interpret and modify. FabSim allows researchers to reduce the complexity of administrative tasks to that of issuing single-line commands, and saves time by introducing a systematic structure for curating input and output files, user and machine configurations, as well as application execution instances. The tool is straightforward to use "out of the box", and as simple to customize.

In Section 2, we present a range of related research and development activities. In Section 3 we present FabSim, describe its architecture as well as its key features for users and developers. In Section 4 we present three research activities where FabSim is currently in use, and describe how FabSim is being deployed and adapted in these contexts. We provide concluding remarks in Section 5.

## 2. Related work

To some extent, the functionality offered by FabSim is similar to that provided by manifold grid middleware projects, developed over the last decade or more. By allowing a computational scientist to run workloads on remote high performance computing resources, FabSim shares functionality with middleware toolkits such as Globus [5], Unicore [6] and gLite [7]. However, the focus of FabSim is very different to these monolithic toolkits. Firstly, FabSim is aimed squarely at the experienced computational scientist, presenting a command line interface that is easy for the developer/scientist to extend. In this respect, FabSim shares similarities with the Growl Toolkit [8], an example of a lightweight middleware system designed to address some of the shortcomings of other grid middlewares, using both a combination of Web service components and wrapper scripts to automate tasks performed by Globus grid middleware client tools.

Additionally, FabSim is built on SSH, found on practically every Unix- or Linux-based system, and does not mandate the installation of an additional heavyweight middleware stack on the resources being accessed. This means that FabSim can be used widely on any HPC resource that supports SSH or GSISSH.

FabSim is more directly comparable with tools such as Longbow [9], which is used for molecular dynamics pertaining to computational biology, and also aims to provide shorthand commands to run applications (including ensembles) on distributed machines. Longbow is, however, more limited in scope, as it provides no explicit support for multi-step workflows or for tasks associated with code compilation and deployment. Ruffus [10] is a light-weight Python tool which automates complicated analysis activities, with less concern for computations on distributed resources. Snakemake [11] offers a workflow definition language, and provides an execution environment. Snakemake workflows can be run remotely, although the tool does not provide further facilities for using distributed resources (e.g., curating information for its users as to how to access different machines).

Many other tools provide functionalities that complement FabSim. Coupling environments such as MUSCLE 2 [12], DataSpaces [13] and MPWide [14] allow codes to efficiently exchange data at runtime, and can be used to speed up the remote execution of coupled tasks in FabSim.

Please cite this article in press as: D. Groen, et al., FabSim: Facilitating computational research through automation on large-scale and distributed e-infrastructures, Computer Physics Communications (2016), http://dx.doi.org/10.1016/j.cpc.2016.05.020

In addition, there are a number of simulation environments which serve to combine functionalities from existing codes to construct and run multiscale simulations [1], such as AMUSE [15], CouPe [16], MOOSE [17] and OASIS [18]. In particular, the Application Hosting Environment [19] provides an easy-to-use environment by centralizing the application deployment in the hands of one expert-user, though it provides no automation for those users who deploy new software. Both GEL [20] and Swift [21] assist in the coordination and efficient execution of large numbers of scripts, while workflow engines such as Kepler [22,23], Taverna [24] and GridSpace [25] provide a graphical environment to help users perform complex simulation workflows. Ludascher et al. [26] provide a comprehensive overview of key challenges and advances in workflow management and scientific task automation. A major strength of FabSim, compared to the aforementioned tools, is its strong focus on accelerating and simplifying development activities. Its aim is not only to simplify the execution of workflows that have been previously defined, but also to simplify the creation of new workflows, and indeed of new computational approaches in general.

## 3. Overview of FabSim

The central purpose of FabSim is to save time for computational researchers by simplifying key tasks when performing computation-driven research. With FabSim we achieve this by offering a set of useful functionalities in a highly transparent and modifiable program structure. By pairing customizable software with a set of best practices we provide researchers with a methodological framework which helps them to perform a range of tasks in a simpler, quicker and more systematic way.

FabSim has a range of built-in functionalities. It supports the execution of jobs on remote resources (through SSH or GSISSH), automates the transfer of data to and from those resources, and provides an administrative framework for curating job information, including its inputs, outputs and environmental parameters. Moreover, FabSim allows for defining complex computations by combining job executions on remote resources with (local or remote) data processing steps and data conversion routines. It is possible for these computations to be cyclic, or to have a dynamic number of steps (e.g., determined at run-time by FabSim based on convergence criteria), or to encompass an ensemble of tasks covering a large parameter space.

The predecessor of FabSim (FabricHemeLB) was developed in 2011 to automate the deployment and execution of the HemeLB bloodflow simulator [27]. We then repurposed the toolkit in 2013 to provide the same convenience for materials molecular dynamics simulations. At this stage, we acknowledged the potential of the software and proceeded to develop a general-purpose version of the toolkit, FabSim. FabSim is now actively maintained and enhanced by academic researchers at both University College London and Brunel University London. At time of writing, the active contributors to the FabSim codebase are Derek Groen, Agastya Bhati and James Suter. James Hetherington has done extensive work on the initial development phases of the toolkit, while Stefan Zasada has made key enhancements in terms of security and usability. Peter Coveney has aligned FabSim with several leading research efforts, and helped capture the key concepts of FabSim in this publication. The extended versions of FabSim are maintained by several research groups, based at UCL, Brunel, as well as at the University of Edinburgh.

FabSim is written in Python and requires no administrative privileges to install. It relies on the Fabric [28] library to embed convenient, light-weight and non-invasive mechanisms for accessing and managing remote machines. In addition, FabSim uses the YaML [29] library, which provides features to work with a compact, intuitive and human-readable data format. The code is freely available at the Computer Physics Communications Program Library, as well as on Github (www.github.com/UCL-CCS/FabSim).

### 3.1. Architecture

We provide an overview of the FabSim architecture in Fig. 1. FabSim contains seven modules, six of which are developed to be easy to modify by users. Key low-level functionalities are defined within the FabSim function library, and are variously general-purpose, domain-specific, problem-specific or for single use. Here, functions with a different scope are placed in different files. Both the Construction and Automation Module (CAM) and the Data Processing Module (DPM) are used to define one-liner commands for higher-level functions and automated workflows. These three modules are commonly extended by defining new Python functions, which combine existing features in FabSim with domain-specific scripts or other resources. Any command defined in the CAM or the DPM which involves the use of remote resources requires the use of the FabSim Security module, which is indicated using a red dashed line. The CAM and DPM modules also rely on three other modules to obtain machine-specific configuration information (from the Machine Configs module), application-specific configuration information (Application Configs module), and key data processing functionalities (Data Processing Scripts Repository). Content in both the Machine Configs and the Application Configs module can be superseded by user-specific configuration files.

The security module is the only module which is not intended to be modified by users. It enables the use of FabSim commands on remote resources using either SSH or GSISSH access mechanisms which are supported by the vast majority of Top 500 supercomputers.

### 3.2. Using FabSim

Once installed, all FabSim features are called from the terminal, using commands that adhere to the following structure:

```
fab <hostname> <command>:<configuration_name>,
<parameter1=x1>,(...),<parameterN=xn>
```

We describe a number of commands commonly used in FabSim in Table 1. Each function can be given parameters if convenient, or uses machine-specific defaults if no parameters are specified by the user. We have also defined a range of commands which are specific for FabSim in different domains. We discuss these commands separately in the examples provided in Section 4, and describe how these functions can be used as part of more complex user-made commands.

### 3.3. Remote execution

FabSim allows users to perform computations on remote resources using one-line commands. For example, to launch an instance of the LAMMPS molecular dynamics code on nodes on a supercomputer named “exa”, one could use:

```
fab exa lammps
```

As part of this command, FabSim stages in a directory with input files from the local machine to the remote resource, and then uses SSH or GSISSH to submit the job using the remote job scheduling system. One can use (or define) similar commands to perform tasks directly on the head node of the remote resource. For example, to locate all modules containing the name “lammps” on the “exa” machine, one could use:

```
fab exa probe:lammps
```

Please cite this article in press as: D. Groen, et al., FabSim: Facilitating computational research through automation on large-scale and distributed e-infrastructures, Computer Physics Communications (2016), http://dx.doi.org/10.1016/j.cpc.2016.05.020

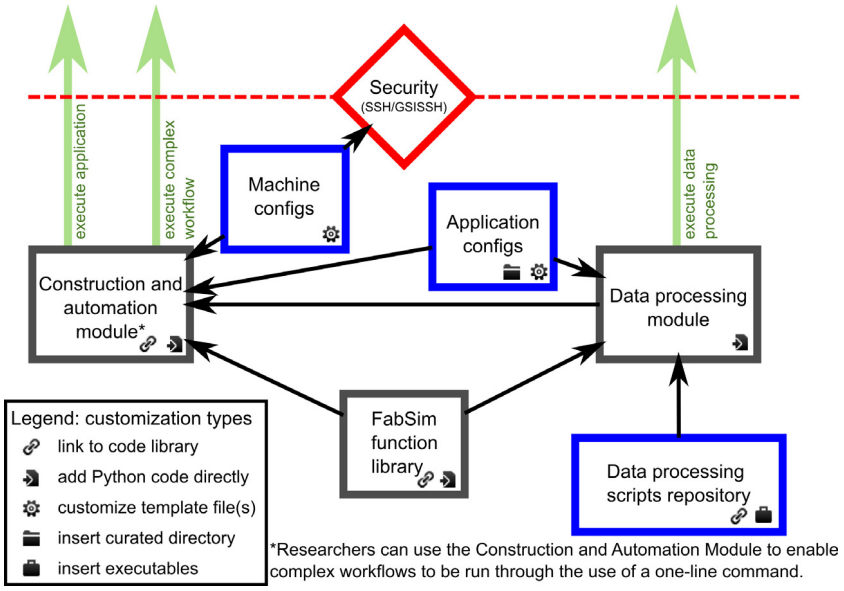


**Fig. 1.** Diagrammatic overview of the modules present in FabSim. Information dependencies are indicated by black arrows (e.g., the data processing module makes use of information provided by the application configs module). Security is managed using Paramiko (www.paramiko.org), which in turn relies on SSH. All modules are installed on a user's local workstation. User activities involving remote resources are indicated by light green arrows passing through the security layer (which is indicated by the red dotted line). Small icons are provided by gentleface.com. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

**Table 1**
List of commands commonly used in FabSim.

| Command name | Brief description |
|---|---|
| `probe` | Probes target host for presence of a module with a given name |
| `stat` | Provides a status report of jobs running on the target host |
| `monitor` | Does a `stat` on the target machine once every two minutes, and displays the output |
| `fetch_results` | Fetches all the application results from the target host and saves them on the local host |
| `cancel` | Cancels a job on the target machine which is in the queue or running |
| `analyse` | Performs shallow data analysis of a results file |
| `run_job` | Runs an application job remotely |
| `run_ensemble` | Runs an ensemble of application jobs remotely (packed into one job if supported) |
| `blackbox` | Runs a script through FabSim on the local machine |
| `archive` | Archives a directory containing application results |
| `cold` | Copies a source code to a remote resource, compiles and builds everything (application-specific) |
| `<application name>` | Runs an application job using the `<application name>` code specifically (application-specific) |

It is also possible to combine the execution of multiple jobs on remote resources in a single FabSim command (see Section 4.3), or to construct a cyclic scheme of interdependent jobs interspersed with local data processing tasks (see Section 4.2).

### 3.4. Limitations

Although FabSim can be integrated and extended for a wide range of purposes, the scope of the core software has some important constraints. FabSim is intended primarily for user-developers without administrative rights, and therefore does not contain features that require these privileges to set up.

For example, FabSim itself does not have resource brokering capabilities, as job scheduling is considered to be an important administrative responsibility. Instead, FabSim relies on remotely installed schedulers such as PBS Pro or LoadLeveller to manage jobs on remote machines. Support for newly released schedulers can be added by adjusting the machine-specific configurations and creating a template file for the job submission script. Likewise, FabSim does not enable users to create accounts on remote machines, as this too requires administrative privileges.

### 3.5. Customizing FabSim

A key strength of FabSim is its ease of customization. FabSim uses a template/variable substitution system to enable users to easily introduce customized scripts, and relies on the ease of use of Python to allow users to define custom functionalities. The substitution system relies on a set of YaML files (see Section 3.6 for an example), which specify the default values for all the important variables in FabSim. Customized values can be assigned to these variables on a machine-specific basis, on a user-specific basis, or both. When the user invokes a FabSim command, the relevant variables are provided to the context of that command, eliminating the need for users to specify these on the command line.

FabSim also relies on a number of templates, for example to identify the formatting required to make a correct header for a batch job script, or to flexibly insert commands for executing a specific MPI implementation. When templates are used, FabSim uses the variables within its context to determine which values to insert into the templates, and which templates to combine. For example, we can combine templates for executing specific applications with those for specific scheduling systems, without the need to define new templates for each combination of the two.

Please cite this article in press as: D. Groen, et al., FabSim: Facilitating computational research through automation on large-scale and distributed e-infrastructures, Computer Physics Communications (2016), http://dx.doi.org/10.1016/j.cpc.2016.05.020

**Table 2**
List of customizable components in FabSim. The name of the component is given in the first column, the module where it is located (see Fig. 1 for a diagrammatic overview of the modules) in the second column. The scope within which the component is applied and customized is given in the third column. Possible scopes include general-purpose (general), domain-specific (domain), problem-specific (problem) and machine-specific (machine). A short summary of the contents of the component is given in the fourth column.

| Name | Location | Scope | Summary of contents |
|---|---|---|---|
| config_files | Application configs | Problem | Input data for specific computations. |
| fab.py | Constr. and auto. module | General | General Fabsim commands. |
| data_proc/dataXX.py | Data processing module | Domain | Data processing commands. |
| data_proc/dataYY.py | Data processing module | Problem | Data processing commands. |
| data_proc/data.py | Data processing module | General | Data processing commands. |
| blackbox/ | Data processing scripts repo | Domain | Plug-in scripts, binaries. |
| python/ | Data processing scripts repo | Domain | Linkable Python plug-ins. |
| fabXX.py | Function library | Domain | FabSim commands. |
| fabXX/fabYY.py | Function library | Problem | FabSim commands. |
| machines.yml | Machine configs | Machine | Default machine settings. |
| machines_user.yml | Machine configs | Machine | User machine settings. |

In Table 2 we present the configuration and software hooks that FabSim provides, each of which can be modified by the user. Within FabSim we define a number of different scopes, as the features in FabSim differ in their range of applicability. The base scope of FabSim is general-purpose, which includes features that are deemed to be of value for any user installing FabSim (e.g., job submission, file transfer and so on). Features can be added on this level by adding functions to fab.py. Machine-specific customizations can be added in one of two separate scopes (“machine-specific” and “machine and user-specific”). Perhaps the scope most important to the user is the domain-specific scope. The script files that reside in this scope contain features that are specific to the user domain (e.g., multiscale materials modelling, blood flow simulation, or molecular ligand–protein binding calculations), but which can be reused for existing and future research problems. In some projects we have been working on, the amount of customization in this scope has become so large that the community adopted a modified name for the tool altogether (e.g., FabHemeLB and FabMD). Last, there is the scope which is problem-specific, which includes bespoke scripts and features that are used for highly-specific research purposes.

### 3.6. Best practices

FabSim is accompanied with a number of best practices which aid the user in maintaining a simple and consistent environment. Here we describe several examples of best practices that are key to keeping FabSim simple to use and modify.

Machine-specific configurations, which are applicable to all users of that machine, are defined in machines.yml, User-specific information for each machine is stored locally in machines_user.yml. An entry in the machines.yml YaML configuration could for example look as follows:

```
petaflop_machine:
job_dispatch: "qsub"
run_command: "aprun -n $cores"
batch_header: pbs
remote: "login.petaflop.ac.uk"
home_path_template: "/home/$project/$username"
runtime_path_template: "/work/$project/$username"
modules: ["load lammps", "load namd"]
queue: "standard"
corespernode: 32
```

As best practice, we keep general-purpose features in fab.py (or libraries that are included in fab.py), and domain-specific features in separate python files (e.g., fabNanoMD.py), distinct from any source files which contain problem-specific features.

### 3.7. Provenance, curation, reproducibility

When a user performs a computation remotely with FabSim, a number of extra actions are executed to help improve the repeatability and reproducibility of the computation. First, FabSim stores the full internal context in YML format in the results directory of the executed computation, allowing users to identify FabSim variables that may have been wrongly set. Second, FabSim stores the environment variables used at the remote resource in another text file, allowing users to spot changes in the configuration of the remote resource between jobs. Third, FabSim retains the generated job submission script for future reference. And fourth, when FabSim creates results directories for submitted computations, it allows users to modify the name of these directories using variables (e.g., code version number, time stamp, configuration file used, or the number of cores used). In particular, by using time stamps in naming results directories one can prevent new computations from overwriting results that were generated by previous computations.

In our experience, we have frequently been able to repeat our runs using the FabSim logging infrastructure. However, it may still occur that repeated computations lead to different results. We have experienced this occasionally when existing modules residing on remote machines are recompiled with different settings or using a different compiler, or when the computations are repeated using a different set of resources.

### 3.8. Security

Access to grid resources is usually secured through authentication and authorization mechanisms based on X.509 certificates, a security credential used to authenticate the user when accessing a grid. To access the resources on a particular distributed e-infrastructure, the user needs a certificate recognized by that infrastructure. Certificates are generally issued on a national basis, by a national research certificate authority (CA). The Interoperable Grid Trust Federation [30] exists to ensure mutual trust between different national certificate issuing bodies. This means that certificates issued in one country will be accepted by e-infrastructure resources based in a different country.

Efforts to address the usability of e-infrastructures have long been hampered by existing security mechanisms imposed on users. Typically, these require a user to obtain one or more digital certificates from a certificate authority, as well as to maintain and renew these certificates as necessary. The difficulty in doing this leads to widespread certificate sharing and misuse and a substantial reduction in the number of potential users [31], and has caused many HPC resource providers and users to abandon grid middleware tools. This has in turn led to secure certificate sharing

Please cite this article in press as: D. Groen, et al., FabSim: Facilitating computational research through automation on large-scale and distributed e-infrastructures, Computer Physics Communications (2016), http://dx.doi.org/10.1016/j.cpc.2016.05.020

mechanisms to be promoted which seek to ameliorate some of the worst aspects of certificate misuse [32].

With FabSim, we provide support using either grid certificates (through GSISSH) or using the SSH protocol (using the Paramiko library, www.paramiko.org). We explicitly enable SSH support because almost all HPC resources support this protocol to allow remote users to access the machine. FabSim uses the Fabric and Paramiko libraries to allow users to perform remote operations using basic SSH as the transport middleware. As such, FabSim's security model is essentially the SSH security model; public/private keys are used to authenticate FabSim operations on target resources, and the `~/.ssh/known_hosts` file is used to allow users to configure mutual authentication.

Unlike grid based X.509 authentication (which is also supported through GSISSH), the SSH setup is entirely controlled by the end user. Typically, on a grid using X.509 certificates, an administrator must set up details of a user's certificate on their resources while SSH based authentication requires that the user sets up their own keys on a target resource. Although key management does potentially increase the management overhead of setting up FabSim security, the ubiquity of SSH means that most FabSim users will have already taken steps to set up SSH keys for the resources that they wish to access.

The use of SSH keys and Fabric also means that FabSim can make use of Fabric extensions for key management, such as the keymanager add-on [33], which allows users to manage keys on multiple servers very easily, using Fabric itself.

## 4. Exemplar FabSim use cases

In this section we present exemplar FabSim use cases in three scientific domains where FabSim has so far been applied. These include simulations of cerebrovascular bloodflow, multiscale simulations of clay-polymer nanocomposites, and ensemble molecular dynamics simulations used to calculate ligand–protein binding affinities.

### 4.1. Cerebrovascular bloodflow

FabSim is widely used in combination with the HemeLB bloodflow simulator [34,27], which we use to investigate the flow properties of blood in arterial networks (e.g., a segment of the middle cerebral artery or a network of vessels in the retina). HemeLB is specifically optimized to efficiently model flow in sparse networks, and scales to up to 49,152 cores [35,27]. Using both HemeLB and FabSim, we have been able to make reliable predictions of the blood flow properties in cerebral arteries [36] and in mouse retinas [37,38]. Accurate predictions of the blood flow under realistic conditions are essential to gain a further understanding of important medical conditions, such as aneurysm formation and tumour growth. For example, the risk of bleeding in the brain appears to correlate with abnormal flow properties in the vicinity of brain aneurysms, such as a high wall shear stress [39].

HemeLB is in use across a range of supercomputer platforms, and many of its applications require complex workflows consisting of multiple simulations. Here, the HemeLB-adapted version of FabSim (named FabHemeLB) allows users to systematically run and curate results of sets of simulations (see the workflow on the right side of Fig. 2 for an example). This makes it considerably easier to perform scalability and accuracy studies which require a large number of simulation runs [27,40].

The HemeLB version of FabSim features three major adaptations:

First, FabHemeLB provides a range of domain-specific features, primarily to enable automated execution of ensemble multiscale simulations. In the workflow depicted on the left in Fig. 3, we first collect patient-specific heart parameters (e.g., heart rate, cardiac output volume) at different levels of physical activity. We then use a 1D model (The Python Network Solver [41]) to calculate the expected inflow profile in a cerebral artery, which is used with HemeLB to model a patient-specific cerebral arterial network in 3D. We then collect the results of our ensemble of simulations using a one-line FabSim command.

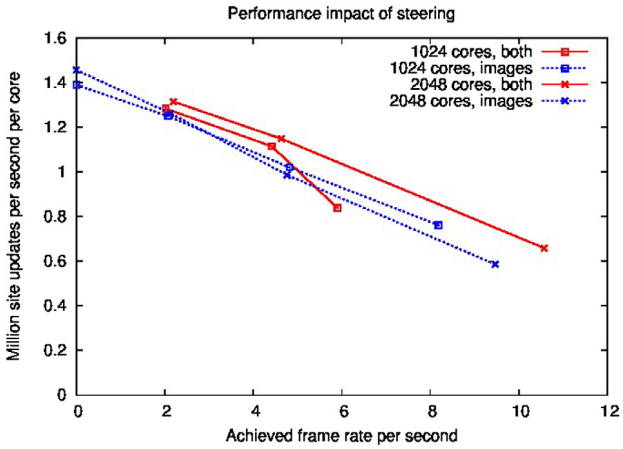


**Fig. 2.** Example of a benchmark which was systematically acquired using FabSim on the HECToR supercomputer at EPCC in Edinburgh, United Kingdom. We show the performance impact of running HemeLB with a connected steering client [34]. We show results for 1024 and 2048 cores without steering client (plotted at frame-rate zero), with the client used only for image streaming (images, dotted lines) and with the client used both for image streaming and steering the HemeLB simulation (both, solid lines). This figure is reproduced from Groen et al. [27], in which the significance of this data is discussed in detail.

Second, FabSim uses the FabSim templating system to automate the installation of HemeLB on new supercomputers. This process was previously labour-intensive as the existing CMake system only permits a limited amount of automation, and lacks an intuitive way to store machine-specific information about the required configuration flags and environment settings. We present the added value of FabSim in the remote installation of HemeLB in Fig. 4. Here, FabSim saves time by allowing researchers to provision installation details in a compact and readable way, thereby automating the installation process for future users.

Third, we extended FabSim to enable a new kind of simulation analysis with HemeLB, in which we can quickly run a set of simulations on a large supercomputer to compare wall shear stresses in arterial bloodflow as a function of exercise intensity. We present a detailed description of this workflow, used on the 2.6 PFLOPs ARCHER supercomputer in Edinburgh, UK, in Itani et al. [36].

### 4.2. Multiscale modelling of clay-polymer nanocomposite materials

Multiscale modelling approaches offer large advantages in the domain of materials modelling. Here, a key challenge in the field is to predict large scale materials properties, whilst taking into account the chemical specificity of the constituent atoms and molecules, as well as processing conditions.

We have created an adapted version of FabSim (named FabMD) to construct and apply a multiscale modelling methodology for the study of clay-polymer nanocomposite materials [42,43]. These composite materials, due to a combination of their low density with superior materials properties, have already been applied in industries such as packaging, automotive, aviation, and drug transport [44]. However, it is costly, time-consuming and labour-intensive to search for new materials using experimental approaches, whereas multiscale simulations are relatively fast and

Please cite this article in press as: D. Groen, et al., FabSim: Facilitating computational research through automation on large-scale and distributed e-infrastructures, Computer Physics Communications (2016), http://dx.doi.org/10.1016/j.cpc.2016.05.020

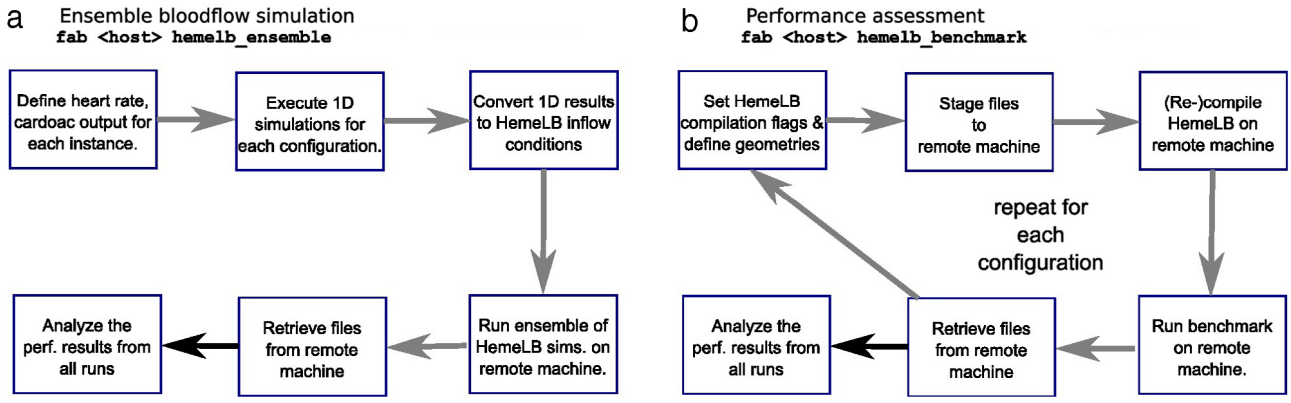


**Fig. 3.** Workflows which have been automated using FabHemeLB. (a) Diagram of the workflow used to perform ensemble multiscale simulations of blood flow in middle cerebral arteries (presented in detail in [36]). (b) A commonly applied workflow to do systematic performance tests (applied for example in Groen et al. [1]).

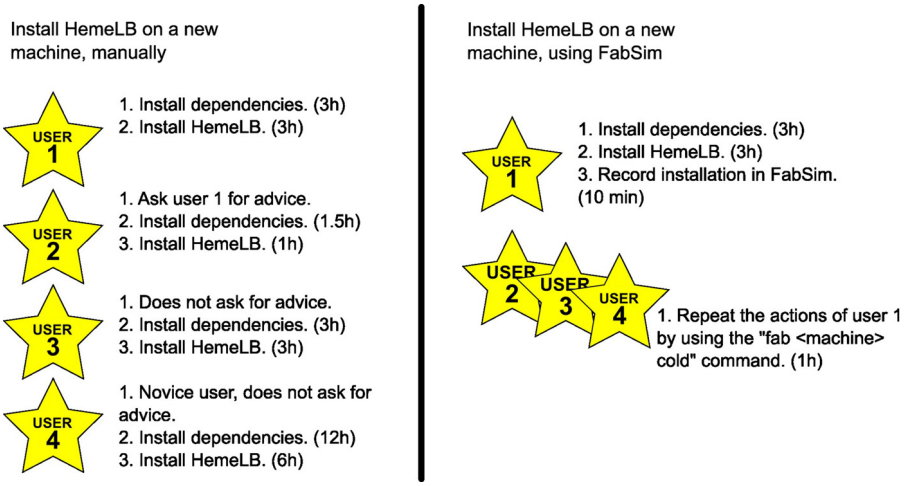


**Fig. 4.** Example of the speed-up obtained through the use of FabSim. We present typical tasks performed by a team of researchers when installing HemeLB manually (on the left), or using FabSim (right).

cheap. As a result, using these approaches we can identify systems that are likely to possess superior materials properties, focusing the laboratory searches for much greater efficiency and hence accelerated discovery.

We present a diagrammatic description of our multiscale materials modelling workflows in Fig. 5. We use FabSim to coarse-grain our clay-polymer systems, and we extended FabSim to automate the two iterative techniques used to determine accurate course-grained potential parameterizations: Iterative Boltzmann Inversion (IBI) and Potential of Mean Force (PMF) calculations.

In the IBI procedure we combine remote job execution with data processing on the local host to iteratively adjust pair potentials. We launch a single simulation per iteration to determine the radial distribution function which results from using these potentials, until they match the distribution functions from all-atom molecular dynamics simulations [42] within a desired tolerance. IBI is computationally efficient, but can be inaccurate, particularly when the number of particles is small (e.g., solutes dissolved in solution) or when the interaction potentials are particularly attractive.

Whenever IBI is ineffective we instead apply PMF, which is computationally more expensive but more robust. In the PMF procedure we combine an ensemble of remote jobs with data processing on the local host. Here each simulation job in the ensemble has its particles constrained at a given distance; this is done to calculate the mean force between two particle types at a given distance. For each PMF iteration, we then perform a set of ∼20–40 simulations with the particles constrained at various distances.

As an example, we summarize the input and output of several IBI iterations in Fig. 6. Here we performed four iterations to parameterize a suitable coarse-grained potential between two polymer particles [42]. In our case the full procedure of creating a new and chemically specific coarse-grained system required over 1000 submitted simulation jobs. FabMD was essential in enabling us to perform these procedures in a quick, automated fashion, providing adaptable quick-hand commands to generate suitable potentials for new systems as our study proceeded.

### *4.3. Calculating ligand–protein binding affinities from ensemble molecular dynamics*

The ability to calculate the free energy of binding a lead compound with a target protein, also known as the binding affinity, is of great importance in the field of personalized medicine and drug discovery as, in most cases, it forms the basis of ranking drugs/lead molecules based on their potency. Several *in silico* methods are available to calculate binding affinities, but the field has gained a degree of notoriety since, frequently, results reported in the literature have not been repeatable by others [2,45]. This lack of reproducibility is due to the insufficient sampling of phase space, arising from the extreme sensitivities of calculated properties to the initial conditions of a molecular dynamics system. Such

Please cite this article in press as: D. Groen, et al., FabSim: Facilitating computational research through automation on large-scale and distributed e-infrastructures, Computer Physics Communications (2016), http://dx.doi.org/10.1016/j.cpc.2016.05.020

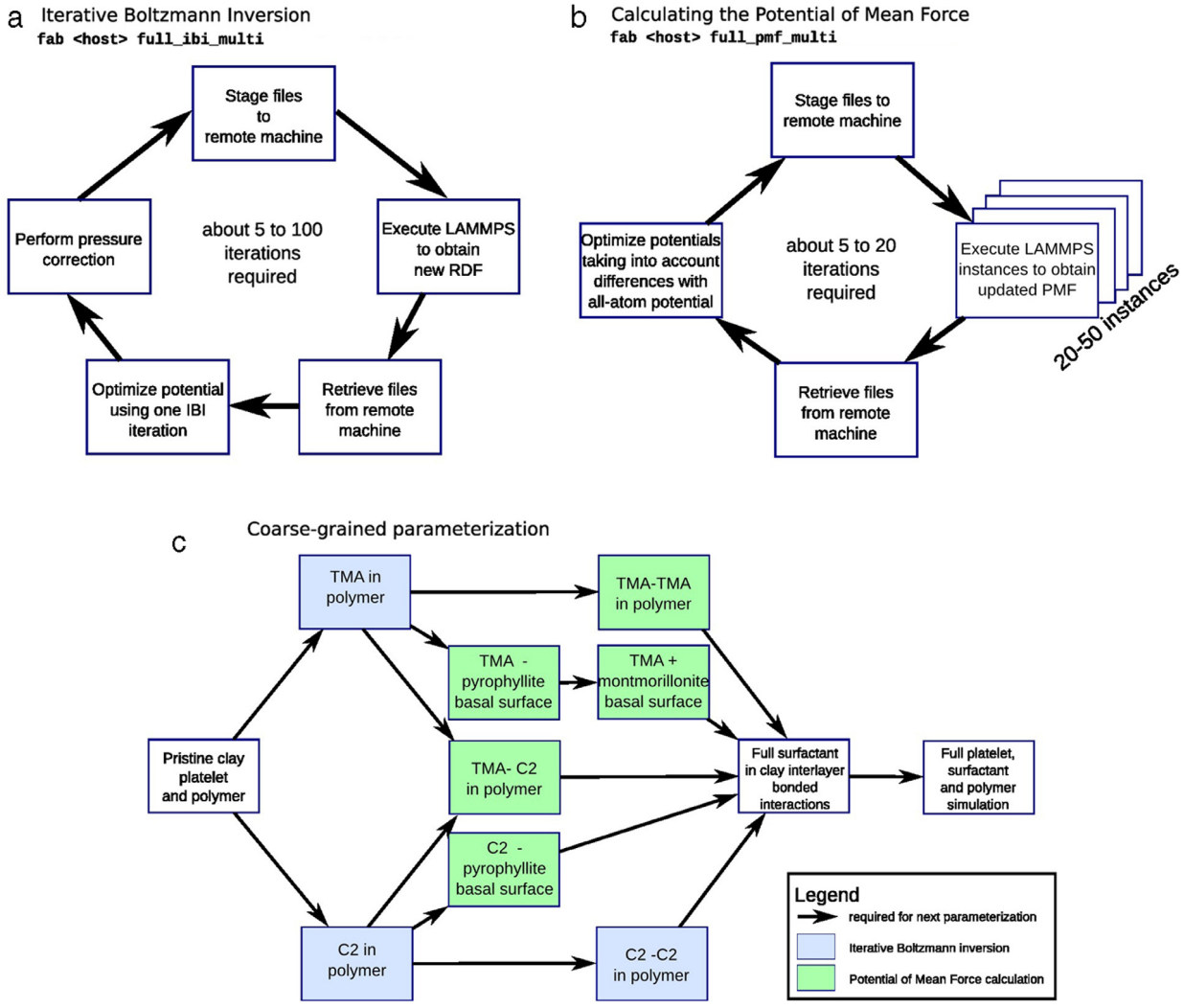


**Fig. 5.** Workflows which have been automated using FabMD to create a coarse-grained model of a mixture of montmorillonite clay and polymers. (a) Workflow used to perform Iterative Boltzmann Inversions. (b) Workflow used to calculate the Potential of Mean Force. (c) Diagrammatic overview of the steps involved to do a full coarse-grained parameterization (applied e.g. by Suter et al. [42]). Here the molecule names have been abbreviated: TMA for tetra-methyl ammonium and C2 for ethane particles.

unreproducible binding affinities are clearly unreliable for medical or industrial applications.

Using an ensemble simulation approach we have shown that, if the entire protocol of binding affinity calculation is repeated a sufficient number of times for a biomolecular system, then the computed binding affinities have a Gaussian frequency distribution and their ensemble average is the theoretical estimate of the binding affinity for that biomolecular system, with bootstrapping providing tight error bounds. Such precise and reproducible binding affinity estimates are expected to be useful in drug discovery and drug selection in personalized medicine.

Our ensemble simulation approach to control errors and ensure reproducibility, which uses the Binding Affinity Calculator (BAC [46,47,2]) in combination with an adapted version of FabSim (FabBioMD), is shown in Fig. 7.

Here we use FabSim to automate the execution of ensemble simulations and analysis procedures. We launch multiple instances of a given molecular model, each of which has different initial atomic velocities and is called a "replica". For each replica we then perform equilibration, production molecular dynamics, followed by post-processing of the MD trajectories to determine the free energy of binding based on our ESMACS protocol [45]. As a last step, we perform a statistical analysis on the collected results to report binding affinities with error estimates. Without the automation provided by FabBioMD, this activity would have a very substantial manual overhead, reducing the rate of progress and being prone to human errors. On a sizeable HPC resource, the approach has the considerable advantage that all replicas can be run concurrently: in the time it takes to run one, we can run all of them.

Specifically for FabBioMD we included specific functions to run ensemble molecular dynamics simulations and data analytics in the form of free energy calculations. For example, we implemented one-liner commands to execute ensembles of simulation and analysis steps, and mechanisms to integrate FabBioMD with support mechanisms for so-called "block jobs". Schedulers with block job support allow users to group a number of simulations within a single supercomputer job, allowing these simulations to be run simultaneously and reducing the number of jobs that need to be submitted to the scheduler. We have used our approach on major supercomputers such as ARCHER (EPCC, UK), BlueWonder (STFC, UK) and SuperMuc (LRZ, Germany). FabBioMD currently supports NAMD for ensemble MD simulation and AMBER [48] for the calculation of free energy contributions to the binding affinity based on MMPBSA [49] and normal mode methods respectively.

A number of related tools introduce automated workflows into free energy calculations. These tools include for example the

Please cite this article in press as: D. Groen, et al., FabSim: Facilitating computational research through automation on large-scale and distributed e-infrastructures, Computer Physics Communications (2016), http://dx.doi.org/10.1016/j.cpc.2016.05.020

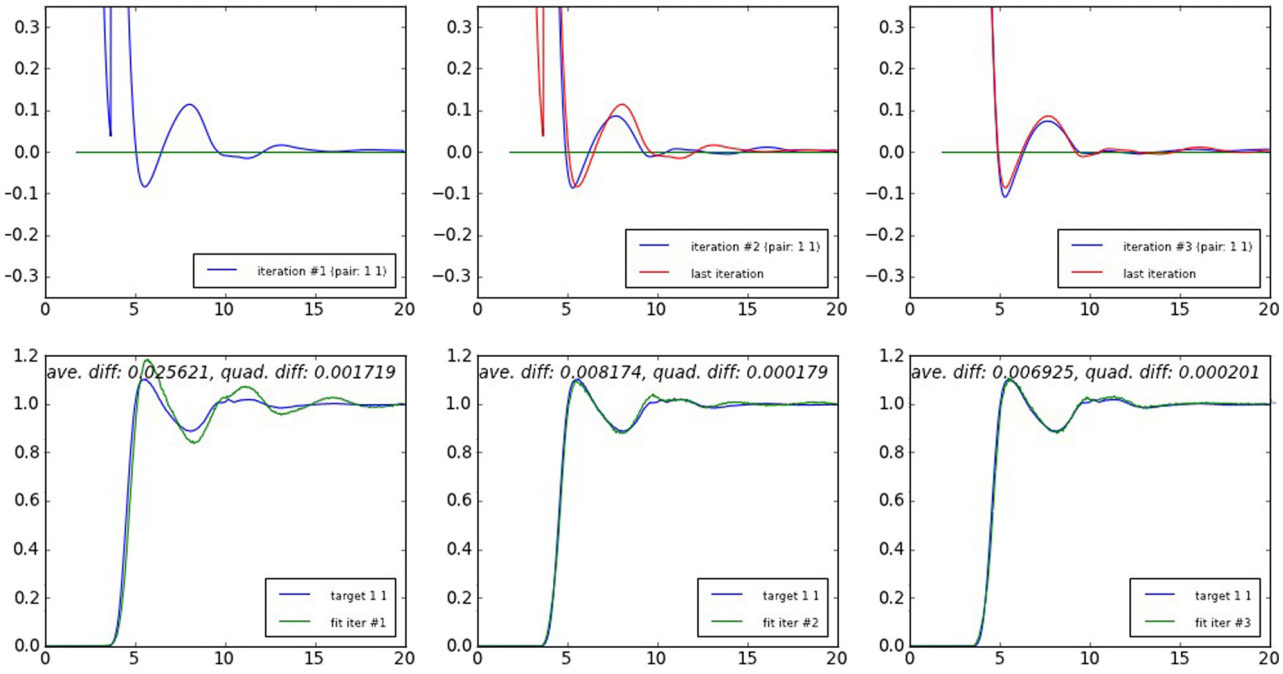


**Fig. 6.** Example output of the first three Iterative Boltzmann Inversions (IBI), used to parameterize polymer–polymer interactions. The changes in potential obtained through the procedure are shown in the top row for iterations 1–3 (left to right). The resulting changes in the radial distribution functions (RDF) for each potential are provided in the bottom row for iterations 1–3 (left to right). Here we also plot the desired RDF, which we obtained from an atomistic simulation of the same system (''target''), as well as two error measures (the mean absolute difference and the mean squared difference) which can be used as convergence criteria. For reasons of simplicity we have omitted information on the pressure correction, which is simultaneously performed in these IBI iterations [42].

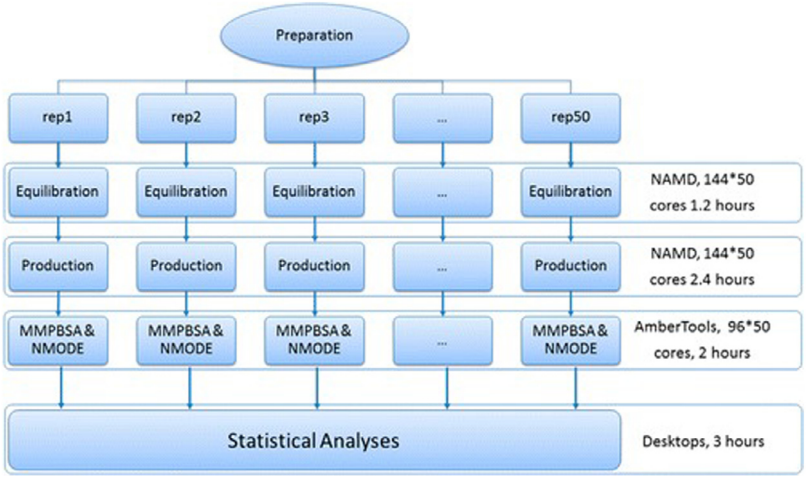


**Fig. 7.** Example of an ESMACS workflow, automated using FabSim, as it is employed in the pMHC binding affinity prediction, reproduced from Wan et al. [45]. Here, 50 trajectory calculations (known as replicas) are performed concurrently in three phases (equilibration, production, MMPBSA&NMODE), each replica using up to 144 cores. After these calculations have completed, the peptide–protein binding affinity is obtained through statistical analysis. We show the ensemble simulations required in a single trajectory calculation for which we need to have access to 7,200 cores for simulations and to 4,800 cores for free energy calculations. For a concurrent three-trajectory case we need 19,200 cores for simulations and 14,400 cores for free energy calculations. The workflow requires approximately 9 hours to complete, given that sufficient resources are available. To compute more than one binding affinity concurrently, one needs to multiply the requirement by the number of peptides of interest.

Amber suite [50], FESetup [51], free energy workflow (FEW) [52], free energy perturbation (FEP) workflow [53], as well as an automation approach using Copernicus [54] in combination with Gromacs [55]. In general, these tools are limited to the traditional one-off MD simulation approach, which is limited by unreliable free energy predictions, and do not support the ensemble simulation approach we described here.

As an example result from our ensemble approach using FabBioMD, we present a set of predictions of peptide-MHC (MHC is the Major Histocompatibility Complex protein) binding affinities. The binding between a peptide and MHC is central to the human immune response. We briefly summarize this study here but it is described in full by Wan et al. [45], a set of 12 different peptide sequences bound to `HLA-A*02:01 MHC` allele were selected, and their calculated binding affinities (using ensembles of 50 instances) were compared with those determined experimentally. In Fig. 8(a) and (b) we present an example normalized frequency distribution of the ensemble of binding affinities of two of the 12 peptides.

Please cite this article in press as: D. Groen, et al., FabSim: Facilitating computational research through automation on large-scale and distributed e-infrastructures, Computer Physics Communications (2016), http://dx.doi.org/10.1016/j.cpc.2016.05.020

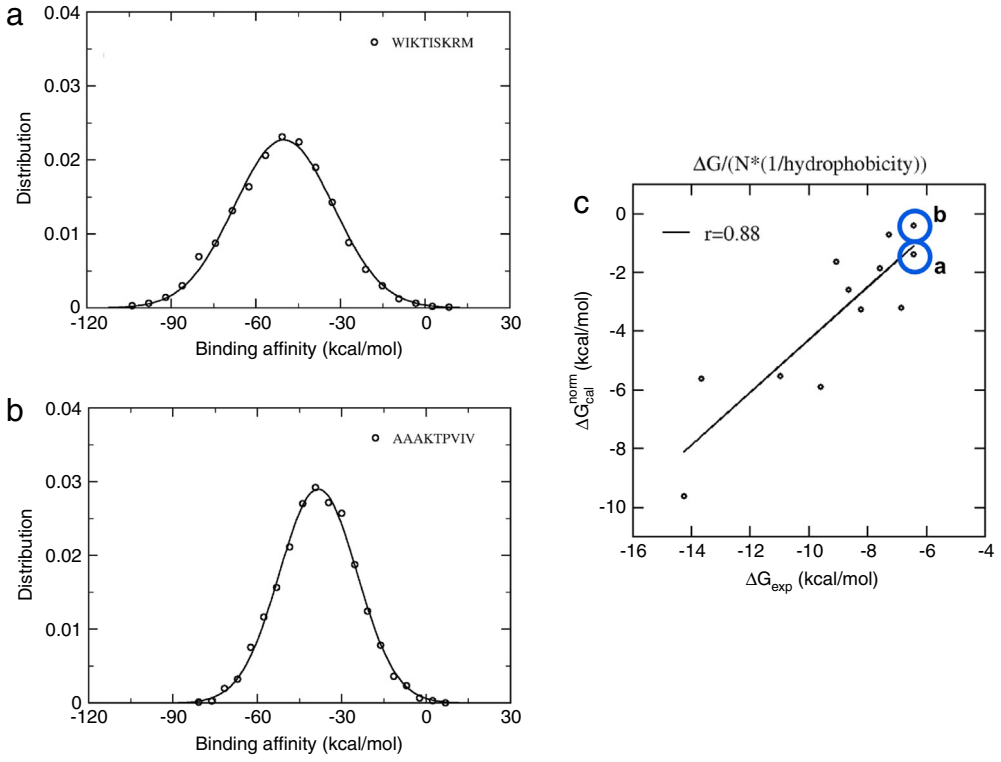


**Fig. 8.** (a) and (b) Example outcomes of two peptide binding affinity calculations (WIKTISKRM and AAAKTPVIV) using ESMACS within BAC (from Wan et al. [45]). Each simulation performed resulted in one data point; 50 simulations were run in total leading to the Gaussian frequency distribution shown. (c): Comparison of the peptide binding affinity calculation results with experimental results. Each data point consists of one binding affinity calculation, with the two points in the top right corresponding respectively, from top to bottom, to the outcomes shown in (b) and (a). These results are discussed in detail in Wan et al. [45].

Each distribution is a Gaussian and corresponds to a single point in Fig. 8(c), where we provide the correlation between experimental binding affinities and our predictions.

## 5. Conclusion

In the present work, we have described and made available FabSim, a new approach to reduce the complexity of tasks associated with computational research. We illustrate the scope and flexibility of FabSim by presenting its application in three diverse domains where it has been, and continues to be, used to simplify computational tasks and to improve their reproducibility. Our use cases in bloodflow modelling, materials modelling and binding affinity calculation provide evidence that FabSim benefits computational research on a generic level.

In the area of brain bloodflow, we have described how FabSim can be used to do systematic benchmarking, to execute an ensemble of multiscale simulations, and to simplify the deployment of HemeLB on remote machines. In the nanomaterials area, we have shown how FabSim automates iterative parameterization of coarse-grained potentials, and allows us to systematically model the self-assembly of layered composite materials with chemical specificity. FabSim has also been applied to streamline the calculation of ligand–protein binding affinities through our Binding Affinity Calculator, allowing users to automatically launch ensemble computations, and thereby controlling uncertainties and producing reproducible results. In all cases, FabSim assists in the curation of run data by furnishing information about the job specification and the environment variables. The extent to which FabSim has been applied and adapted in these three domains serves to demonstrate its flexibility and ease of adoption. Indeed, using FabSim we have been able to publish our research findings in leading scientific journals in each domain.

As part of this software paper, we provide the FabSim code base to the scientific community under a permissive BSD 3-clause licence. As part of so doing, we hope to encourage computational researchers to begin "automating away" some of their more tedious tasks, and to free up more human effort for advancing science.

## Acknowledgements

We thank Dr. Shunzhou Wan for his help in constructing the Binding Affinity Calculation section of this article, and Miguel Bernabeu, Rupert Nash, Sebastian Schmieschek, Mohamed Itani and Hywel Carver for their contributions to FabHemeLB. This work was funded in part by the EU FP7 MAPPER, CRESTA, P-medicine and VPH-SHARE project (Grant Nos. 261507, 287703, 270089, 269978), by the EU H2020 ComPat project (Grant No. 671564) by EPSRC via the 2020 Science Programme (EP/I017909/1), the Qatar National Research Fund (Grant No. 09–260–1–048), MRC Bioinformatics project (MR/L016311/1) and the UCL Provost. AB is funded by an INLAKS Foundation Scholarship and a UCL Overseas Research Studentship Award (2014–2017). Supercomputing time was provided by the Hartree Centre (Daresbury Laboratory) on BlueJoule and BlueWonder via the CGCLAY project, and on HECToR and ARCHER, the UK national supercomputing facility at the University of Edinburgh, via EPSRC through grants EP/F00521/1, EP/E045111/1, EP/I017763/1 and the UK Consortium on Mesoscopic Engineering Sciences (EP/L00030X/1).

## References

[1] D. Groen, S.J. Zasada, P.V. Coveney, IEEE Comput. Sci. Eng. 16 (2) (2014) 34–43.
[2] D.W. Wright, B.A. Hall, O.A. Kenway, S. Jha, P.V. Coveney, J. Chem. Theory Comput. 10 (3) (2014) 1228–1241.
[3] A. Hoekstra, B. Chopard, P.V. Coveney, Phil. Trans. R. Soc. A 372 (2021) (2014).

[4] J. Borgdorff, M. Ben Belgacem, C. Bona-Casas, L. Fazendeiro, D. Groen, O. Hoenen, A. Mizeranschi, J.L. Suter, D. Coster, P.V. Coveney, W. Dubitzky, A.G. Hoekstra, P. Strand, B. Chopard, Phil. Trans. R. Soc. A 372 (2021) (2014).
[5] I. Foster, J. Comput. Sci. Tech. 21 (4) (2006) 513–520.
[6] The UNICORE Project, 2016, http://www.unicore.org.
[7] gLite Middleware, 2016, http://glite.web.cern.ch/glite/.
[8] M. Hayes, L. Morris, R. Crouchley, D. Grose, T. Van Ark, R. Allan, J. Kewley, in: 4th UK e-Science All Hands Meeting, Nottingham, UK, 2005.
[9] Longbow, 2015. http://www.hecbiosim.ac.uk/wikis/index.php/Longbow.
[10] L. Goodstadt, Bioinformatics 26 (21) (2010) 2778–2779.
[11] J. Koester, S. Rahmann, Bioinformatics 28 (19) (2012) 2520–2522.
[12] J. Borgdorff, M. Mamonski, B. Bosak, K. Kurowski, M. Ben Belgacem, B. Chopard, D. Groen, P.V. Coveney, A.G. Hoekstra, J. Comput. Sci. 5 (5) (2014) 719–731.
[13] C. Docan, M. Parashar, S. Klasky, Clust. Comput. 15 (2) (2012) 163–181.
[14] D. Groen, S. Rieder, S.P. Zwart, J. Open Res. Softw. 1 (1) (2013) e9.
[15] S.F. Portegies Zwart, S.L. McMillan, A. van Elteren, F.I. Pelupessy, N. de Vries, Comput. Phys. Comm. 184 (3) (2013) 456–468.
[16] CouPE, 2015. sites.google.com/site/coupempf/.
[17] D. Gaston, C. Newman, G. Hansen, D. Lebrun-Grandié, Nucl. Eng. Des. 239 (10) (2009) 1768–1778.
[18] J.B. Gregersen, P.J.A. Gijsbers, S.J.P. Westen, J. Hydroinform. 9 (3) (2007) 175–191.
[19] S.J. Zasada, P.V. Coveney, Comput. Phys. Comm. 180 (12) (2009) 2513–2525.
[20] C.C. Lian, F. Tang, P. Issac, A. Krishnan, J. Parallel Distrib. Comput. 65 (7) (2005) 857–869.
[21] M. Wilde, M. Hategan, J.M. Wozniak, B. Clifford, D.S. Katz, I. Foster, Parallel Comput. 39 (9) (2011) 633–652.
[22] I.J. Taylor, E. Deelman, D.B. Gannon, M. Shields, Workflows for e-Science: Scientific Workflows for Grids, Springer Publishing Company, Incorporated, 2014.
[23] B. Ludäscher, I. Altintas, C. Berkley, D. Higgins, E. Jaeger, M.B. Jones, E.A. Lee, J. Tao, Y. Zhao, Concurr. Comput.: Pract. Exp. 18 (10) (2006) 1039–1065.
[24] K. Wolstencroft, R. Haines, D. Fellows, A. Williams, D. Withers, S. Owen, S. Soiland-Reyes, I. Dunlop, A. Nenadic, P. Fisher, et al., Nucleic Acids Res. (2013) 328.
[25] K. Rycerz, M. Bubak, E. Ciepiela, D. Harelak, T. Gubaa, J. Meizner, M. Pawlik, B. Wilk, Future Gener. Comput. Syst. 53 (0) (2015) 77–87.
[26] B. Ludäscher, I. Altintas, S. Bowers, J. Cummings, T. Critchlow, E. Deelman, D.D. Roure, J. Freire, C. Goble, M. Jones, S. Klasky, T. McPhillips, N. Podhorszki, C. Silva, I. Taylor, M. Vouk, in: Arie Shoshani, Doron Rotem (Eds.), Scientific Data Management, Challenges, Technology and Deployment, in: Computational Science Series, 230, Chapman and Hall/CRC, ISBN: 978-1-4200-6981-5, 2009, pp. 476–508.
[27] D. Groen, J. Hetherington, H.B. Carver, R.W. Nash, M.O. Bernabeu, P.V. Coveney, J. Comput. Sci. 4 (5) (2013) 412–422.
[28] J. Forcier, Fabric, 2014. http://www.fabfile.org.
[29] YAML, 2015. www.yaml.org.
[30] IGTF: Interoperable Global Trust Federation, https://www.igtf.net/.
[31] B. Beckles, V. Welch, J. Basney, Int. J. Hum. Factors Manuf. 63 (1/2) (2005) 74–101.
[32] S.J. Zasada, A.N. Haidar, P.V. Coveney, Phil. Trans. R. Soc. A 369 (1949) (2011) 3413–3428.
[33] Fabric SSH Keymanager, 2015, https://github.com/farridav/keymanager.
[34] M.D. Mazzeo, P.V. Coveney, Comput. Phys. Comm. 178 (12) (2008) 894–914.
[35] Neurological simulation milestone reached after UCL embraces Allinea's tools on UK's largest supercomputer, 2014, http://www.allinea.com/news/201406/neurological-simulation-milestone-reached-after-ucl-embraces-allinea%E2%80%99s-tools-uk%E2%80%99s.
[36] M.A. Itani, U.D. Schiller, S. Schmieschek, J. Hetherington, M.O. Bernabeu, H. Chandrashekar, F. Robertson, P.V. Coveney, D. Groen, J. Comput. Sci. 9 (2015) 150–155.
[37] M.O. Bernabeu, M.L. Jones, J.H. Nielsen, T. Krüger, R.W. Nash, D. Groen, S. Schmieschek, J. Hetherington, H. Gerhardt, C.A. Franco, P.V. Coveney, J. R. Soc. Interface 11 (99) (2014).
[38] C.A. Franco, M.L. Jones, M.O. Bernabeu, I. Geudens, T. Mathivet, A. Rosa, F.M. Lopes, A.P. Lima, A. Ragab, R.T. Collins, L.-K. Phng, P.V. Coveney, H. Gerhardt, PLoS Biol. 13 (4) (2015) e1002125.
[39] J.R. Cebral, F. Mut, J. Weir, C. Putman, Am. J. Neuroradiol. 32 (1) (2011) 145–151.
[40] R.W. Nash, H.B. Carver, M.O. Bernabeu, J. Hetherington, D. Groen, T. Krüger, P.V. Coveney, Phys. Rev. E 89 (2014) 023303.
[41] S. Manini, L. Antiga, L. Botti, A. Remuzzi, Ann. Biomed. Eng. (2014) 1–13.
[42] J.L. Suter, D. Groen, P.V. Coveney, Adv. Mater. 27 (6) (2015) 966–984.
[43] J.L. Suter, D. Groen, P.V. Coveney, Nano Lett. 15 (12) (2015) 8108–8113.
[44] S.S. Ray, Macromol. Chem. Phys. 215 (12) (2014) 1162–1179.
[45] S. Wan, B. Knapp, D.W. Wright, C.M. Deane, P.V. Coveney, J. Chem. Theory Comput. 11 (7) (2015) 3346–3356.
[46] S.K. Sadiq, D. Wright, S.J. Watson, S.J. Zasada, I. Stoica, P.V. Coveney, J. Chem. Inf. Model. 48 (9) (2008) 1909–1919.
[47] S.K. Sadiq, D.W. Wright, O.A. Kenway, P.V. Coveney, J. Chem. Inf. Model. 50 (5) (2010) 890–905.
[48] D.A. Pearlman, D.A. Case, J.W. Caldwell, W.S. Ross, T.E. Cheatham, S. DeBolt, D. Ferguson, G. Seibel, P. Kollman, Comput. Phys. Comm. 91 (1) (1995) 1–41.
[49] P.A. Kollman, I. Massova, C. Reyes, B. Kuhn, S. Huo, L. Chong, M. Lee, T. Lee, Y. Duan, W. Wang, O. Donini, P. Cieplak, J. Srinivasan, D.A. Case, T.E. Cheatham, Acc. Chem. Res. 33 (12) (2000) 889–897.
[50] D.A. Case, T.E. Cheatham, T. Darden, H. Gohlke, R. Luo, K.M. Merz, A. Onufriev, C. Simmerling, B. Wang, R.J. Woods, J. Comput. Chem. 26 (16) (2005) 1668–1688.
[51] FESetup, 2015. http://www.hecbiosim.ac.uk/fesetup/download/0-/3-fesetup.
[52] N. Homeyer, H. Gohlke, J. Comput. Chem. 34 (11) (2013) 965–973.
[53] L. Wang, Y. Wu, Y. Deng, B. Kim, L. Pierce, G. Krilov, D. Lupyan, S. Robinson, M.K. Dahlgren, J. Greenwood, D.L. Romero, C. Masse, J.L. Knight, T. Steinbrecher, T. Beuming, W. Damm, E. Harder, W. Sherman, M. Brewer, R. Wester, M. Murcko, L. Frye, R. Farid, T. Lin, D.L. Mobley, W.L. Jorgensen, B.J. Berne, R.A. Friesner, R. Abel, J. Am. Chem. Soc. 137 (7) (2015) 2695–2703.
[54] S. Pronk, I. Pouya, M. Lundborg, G. Rotskoff, B. Wesén, P.M. Kasson, E. Lindahl, J. Chem. Theory Comput. 11 (6) (2015) 2600–2608.
[55] S. Pronk, S. Pall, R. Schulz, P. Larsson, P. Bjelkmar, R. Apostolov, M.R. Shirts, J.C. Smith, P.M. Kasson, D. van der Spoel, B. Hess, E. Lindahl, Bioinformatics (2010).

Please cite this article in press as: D. Groen, et al., FabSim: Facilitating computational research through automation on large-scale and distributed e-infrastructures, Computer Physics Communications (2016), http://dx.doi.org/10.1016/j.cpc.2016.05.020