PHILOSOPHICAL TRANSACTIONS A

royalsocietypublishing.org/journal/rsta

Review

Cite this article: Groen D, Knap J, Neumann P, Suleimenova D, Veen L, Leiter K. 2019 Mastering the scales: a survey on the benefits of multiscale computing software. *Phil. Trans. R. Soc. A* **377**: 20180147.
http://dx.doi.org/10.1098/rsta.2018.0147

Accepted: 6 November 2018

One contribution of 11 to a theme issue 'Multiscale modelling, simulation and computing: from the desktop to the exascale'.

Subject Areas:
computer modelling and simulation, mathematical modelling, software

Keywords:
multiscale computing, multiscale modelling, multiscale simulation, high-performance computing, usability

Author for correspondence:
Derek Groen
e-mail: derek.groen@brunel.ac.uk

Electronic supplementary material is available online at https://dx.doi.org/10.6084/m9.figshare.c.4352660.

THE ROYAL SOCIETY PUBLISHING

Mastering the scales: a survey on the benefits of multiscale computing software

Derek Groen[1], Jaroslaw Knap[2], Philipp Neumann[3], Diana Suleimenova[1], Lourens Veen[4] and Kenneth Leiter[2]

[1]Department of Computer Science, Brunel University London, Uxbridge, UK
[2]US Army Research Laboratory, Aberdeen Proving Ground, Aberdeen, MD USA
[3]Department of Scientific Computing, University of Hamburg, Hamburg, Germany
[4]Netherlands eScience Center, Amsterdam, The Netherlands

DG, 0000-0001-7463-3765

In the last few decades, multiscale modelling has emerged as one of the dominant modelling paradigms in many areas of science and engineering. Its rise to dominance is primarily driven by advancements in computing power and the need to model systems of increasing complexity. The multiscale modelling paradigm is now accompanied by a vibrant ecosystem of multiscale computing software (MCS) which promises to address many challenges in the development of multiscale applications. In this paper, we define the common steps in the multiscale application development process and investigate to what degree a set of 21 representative MCS tools enhance each development step. We observe several gaps in the features provided by MCS tools, especially for application deployment and the preparation and management of production runs. In addition, we find that many MCS tools are tailored to a particular multiscale computing pattern, even though they are otherwise application agnostic. We conclude that the gaps we identify are characteristic of a field that is still maturing and features that enhance the deployment and production steps of multiscale

application development are desirable for the long-term success of MCS in its application fields.

This article is part of the theme issue 'Multiscale modelling, simulation and computing: from the desktop to the exascale'.

1. Introduction

Many phenomena in science and engineering are amenable to multiscale modelling. Multiscale modelling is a divide-and-conquer paradigm in which multiscale models are built as assemblies of individual unit processes, often also referred to as at-scale models, operating at distinct spatial or temporal scales. With the inclusion of relevant unit processes, multiscale models are capable of accurately characterizing phenomena in regimes not easily observed *in vivo* or *in vitro*. Multiscale modelling is primarily a computational endeavour and, over the last two decades, a range of supporting software has emerged for building computational multiscale models, for example, facilitating the *coupling* of existing at-scale models, enabling the use of (remote) high-performance computing resources, or simplifying the management of multiscale simulation runs through automation. Although current *multiscale computing software* (MCS) has been shown to provide benefits, as evidenced by their uptake [1,2], we seek to more clearly analyse their current added value to the multiscale application development process, and find previously under-prioritized areas in which software could provide further support.

In this article, we define MCS as software that provides added value during one or more stages of the multiscale application development process, and has an explicitly formulated orientation towards multiscale, multiphysics, multimodel or other coupled applications. Using this definition, we then analyse a representative set of existing MCS in order to establish the current state of the art in MCS, identify the main obstacles preventing a widespread adoption of MCS in science and engineering, and chart a path forward for development of the next-generation MCS. To that end, we start by summarizing the recent developments in MCS in §2, review the common steps in the process of developing a multiscale application in §3, and reflect on the scope, advantages and drawbacks of adopting generic MCS in §4. In §5, we present our analysis approach, followed by an overview of key results from our analysis in §6, and a discussion with conclusion in §7.

2. Recent developments in multiscale computing software

Following the formulation of mathematical foundations of multiscale modelling (cf. [3,4] for an overview), computational aspects of multiscale modelling have only recently become the focus of the scientific community. This interest has yielded a number of MCS aiming to ease creation of multiscale models, especially those relying on modern high-performance computing architectures. In particular, emerging exascale computing architectures present both a challenge and an opportunity for MCS development [5]. On the one hand, exascale computers promise to provide an unprecedented compute capacity, most probably required for multiscale modelling. On the other hand, in order to fully harness this capacity, significant algorithmic advances are necessary to handle fault tolerance and robustness, heterogeneity of processors and memory and energy-efficiency, to name a few.

Multiscale modelling is a divide-and-conquer endeavour. Relevant scales, both temporal and spatial, are identified and models developed at each individual scale. These at-scale models are then combined to form a multiscale model. The description of a multiscale model can be formally handled by means of the scale-separation map which defines the individual scales in a multiscale model along with the interactions between scales [6]. The scale separation map is often encoded in the multiscale modelling language (MML), a descriptive language for multiscale model development [7]. More recently, multiscale computing patterns (MCP), higher-level abstractions

serving as a basis for more generic MCS software, have been introduced [8]. MCP are categories of multiscale models that exhibit common scale-separation maps and coupling topologies between model components. Example MCP include the Extreme Scaling (ES) pattern where a single at-scale model dominates computational cost within a multiscale model, the Heterogeneous Multiscale Computing (HMC) pattern based on the heterogeneous multiscale method (HMM) [4] where many microscale models are coupled to a macroscale model and launched on-demand, and the Replica Computing (RC) pattern where a large number of individual model ensembles are evaluated under a range of initial conditions.

By their nature, multiscale models are composed of individual at-scale (or single scale) model components. Each at-scale component is frequently a complex parallel software developed over many years. This fact has motivated a shift away from monolithic approaches to multiscale model development and towards more heterogeneous component-based approaches, capable of incorporating existing at-scale models with minimal software modifications. One such approach, the cooperative parallelism programming model, is a task-based multiple-program multiple-data approach to parallel programming [9]. In cooperative parallelism, single unit computation tasks named symponents (a portmanteau of simulation and component) are executed by a runtime system. Symponents are able to interact dynamically with the runtime system to launch, communicate with, and destroy additional symponent calculations. The Co-op MCS implements the cooperative parallelism programming model and leverages the Babel software [10] to integrate symponents together that are written in different programming languages [11,12]. The cooperative parallelism programming model is well-suited for development of multiscale models [13] and the Co-op MCS has been successfully employed for multiscale modelling of materials [14].

Owing to the modularity of the cooperative parallelism approach, developers can easily mix-and-match various at-scale models and incorporate surrogate models to reduce computational cost. For example, adaptive sampling algorithms have been developed within the Co-op system to automatically construct surrogate models during multiscale model evaluation [15]. In adaptive sampling, input and output data obtained from evaluation of at-scale model components are used to construct surrogate models that are stored in a metric-tree database. The surrogate models are much cheaper to compute and can often be evaluated in place of at-scale models with manageable errors. The use of adaptive sampling techniques in a multiscale model can reduce computational cost by several orders of magnitude [14,16]. Moreover, the modular nature of the Co-op system allows for the use of adaptive sampling techniques in any multiscale model developed within the framework. In addition to its implementation in Co-op, the adaptive sampling method has been released in software as the Adaptive Sampling Proxy Application (ASPA) [17].

A modular component-based approach to multiscale modelling is also fundamental to the Multiscale Coupling Library and Environment (MUSCLE) [18]. The MUSCLE software has matured over many years and several different versions have been released. The original MUSCLE is tailored to complex automata modelling and multi-agent computing [19–21]. A subsequent version, MUSCLE 2, is designed for distributed multiscale computation where at-scale model components execute across disparate and potentially geographically separate computers [22]. MUSCLE 2 is able to incorporate at-scale model components written in a variety of programming languages including Java, C, C++, Python and Fortran and is able to directly generate runtime configurations using the MML specification of a multiscale model. Among other things it has been embedded in the VPH Hypermodelling Framework [23]. The newest version, MUSCLE 3, aims to more tightly integrate an extended version of the MML, with better support for dynamic submodel instantiation, surrogate modelling and uncertainty quantification and sensitivity analysis.

Another computational framework for scale-bridging in multiscale modelling is the Hierarchical Multiscale Simulation (HMS) framework [24]. The HMS framework closely follows the HMM for multiscale model construction. HMS combines hierarchies of at-scale model components together and implements a runtime system to schedule and execute at-scale models

on available computational resources. Each at-scale model component is taken to be a standalone executable written in any programming language to ease incorporation of existing complex at-scale models into a multiscale model. The HMS framework has been extended to allow for execution of at-scale model components across multiple high-performance computers [25]. In addition, an adaptive sampling algorithm has been introduced into the framework to reduce computational expense [16].

MCS has also arisen within a number of scientific areas, including astrophysics, climate modelling, materials modelling, plasma physics and systems biology [2]. An exhaustive bibliography of multiphysics and multiscale software frameworks through 2015 has been provided in [1]. These MCS are frequently tailored to a particular phenomenon under consideration by each community. Yet, they are often sufficiently generic to be adapted to other areas with minimal effort. One example in astrophysics, the Astrophysical Multipurpose Software Environment (AMUSE), is a Python-based software framework to combine simulation codes together for astrophysical simulations [26]. AMUSE includes a large number of community astrophysics simulation codes to handle gravitational dynamics, stellar evolution, hydrodynamics, and radiative transfer and implements user-friendly features including a unit algebra model to simplify unit-conversions between models in the framework. The AMUSE approach has been proven successful and it now serves as the basis for the Oceanographic Multipurpose Software Environment (OMUSE) for ocean modelling [27]. Other MCS for atmosphere and ocean modelling includes The Earth System Modelling Framework (ESMF), a component-based software platform under development since the early 2000s [28].

For materials science applications, the Exascale Co-design Center for Materials in Extreme Environments (ExMatEx) has developed a number of MCS [29]. ExMatEx has placed a particular emphasis on new exascale computing architectures as an enabler for new approaches to multiscale modelling including task-based computation and adaptive fault-tolerant algorithms [30]. In order to facilitate the creation of new multiscale computing algorithms, ExMatEx has developed a number of proxy apps: simplified at-scale models which mimic the computational workload of more complex models. The proxy apps are designed to be simpler to work with than more complex models for the development of new multiscale computing algorithms. Through the use of the proxy apps, ExMatEx has created and released several software packages for multiscale computing. The Task-based Scale-bridging Code (TaBaSCo) uses Charm++ to execute an adaptive and asynchronous task-based computation of an embedded viscoplasticity model (CoEVP) for the constitutive response of a continuum model of Lagrangian hydrodynamics [31,32]. The software is written to evaluate multiscale computing approaches on the Trinity Advanced Technology System supercomputer, a pre-exascale system at Los Alamos National Laboratory. Another software, the Distributed Database Kriging for Adaptive Sampling (D^2KAS) implements a redis in-memory data store in combination with locally aware hashing to construct and evaluate kriging surrogate models on-the-fly from at-scale model data [33]. A twist on the adaptive sampling approach is a method which avoids use of a data store entirely, but samples an at-scale model at a set of spatial points at each timestep to construct a surrogate model using Akima splines [34].

In systems biology, the ENteric Immunity Simulator Multi-scale Modelling (ENISI MSM) is a Java-based system for multiscale modelling of immunological processes [35]. ENISI MSM combines together agent-based models, ordinary differential equation-based models, and partial differential equation-based models along with a visualization interface to control the simulation. A recent version of ENISI MSM has been released for high-performance computing environments.

In addition to the task-based integrative frameworks for multiscale modelling described above, there exist a number of coupling frameworks for multiphysics and multiscale modelling. Coupling frameworks are mainly designed to facilitate the exchange of data between different models. Such data exchange occurs at an interface or handshake region between models, as is often the case in partitioned-domain multiscale methods [36]. Coupling frameworks typically implement methods for the interpolation of data between different meshes and parallel data

exchange between individual processes in each model. Software implementing this type of coupling includes the Model Coupling Toolkit (MCT) [37] and Multiscale Universal Interface [38]. MCT has been employed in OASIS-MCT to couple two-dimensional fields for climate system modelling [39]. The Macro-Micro-Coupling Tool (MaMiCo) enables coupling molecular dynamics and computational fluid dynamics codes and includes capabilities to perform ensemble sampling of molecular dynamics trajectories to obtain statistically converged flow field quantities [40,41].

Since multiscale models are inherently complex software with individual components which are themselves large-scale parallel applications, tools to aid developers and users of multiscale models are required for multiscale modelling approaches to become widely adopted. For example, the deployment of a multiscale model across multiple high-performance computers where each system has a different compiler suite, MPI version, node configuration, etc., still presents a formidable challenge. Fortunately, these needs are not unique to multiscale modelling, and it is likely that software tools developed in other areas, for example in cloud-based distributed systems, can be easily adopted to form a complete MCS stack.

One effort to address the relative lack of supportive tools for deployment of multiscale models is FabSim [42]. FabSim aims for reproducible execution of complex workflows across multiple high-performance computers and has been successfully applied to multiscale models requiring ensembles of molecular dynamics simulations [43], as well as blood flow [44]. A larger framework like Automated Interactive Infrastructure and Database for computational science (AiiDA) [45] can also be used for this purpose, while tools such as Longbow [46] are suitable alternatives for running single jobs remotely using quick one-liner commands. Workflow packages which can be used to help execute multiscale applications include the Kepler Project [47], the Swift scripting language [48], the Ensemble Toolkit [49] and Parsl [50] which allow for development and execution of parallel workflows involving many individual programmes across clouds and supercomputers.

3. Multiscale computing applications: the development process

Before assessing in what ways MCS can benefit the developer of multiscale computing applications, we review a number of common steps we recognize in the development process for multiscale applications.

We identify the typical steps required when developing a multiscale computing application in figure 1. Development starts with the design of the conceptual models to address a scientific challenge of interest (Design Step). This includes selecting necessary single-scale models and determining which of these need to be coupled directly. Next, the computational models are adapted, and coupling mechanisms are implemented to facilitate the transport of data between the submodels (Implementation Step). This can, for example, be done using coupling libraries or workflow tools. Once the single-scale models and coupling mechanisms have been established, the implementation can be applied to the specific scientific problem of interest (Instantiation). This includes adding relevant data and parameters, e.g. force field definitions and initial particle configurations for a multiscale molecular application.

After Instantiation, the application is made operational at the target platform (e.g. cluster, cloud or supercomputer, Deployment Step), upon which it is (initially) run (Execution Step). Deployment is complicated due to the fact that various single-scale models and their coupling have to be orchestrated, potentially on a heterogeneous platform (CPU/GPU supercomputer) or even multiple platforms (combining clusters or working in a cloud). After the initial run, the application is usually subject to a cycle of further optimizations (Optimization Step) and executions (or repetitions of earlier steps as needed). Optimization in the context of this paper refers to bolstering the scientific and technical quality of the application such that it becomes suitable for use in production runs. This may involve fixing verification or validation issues that arose during execution, rerunning the application multiple times to test the sensitivity of key parameters, or to check the propagation of uncertainties in the model. Once the application has been sufficiently optimized, researchers proceed with performing the main runs (Production

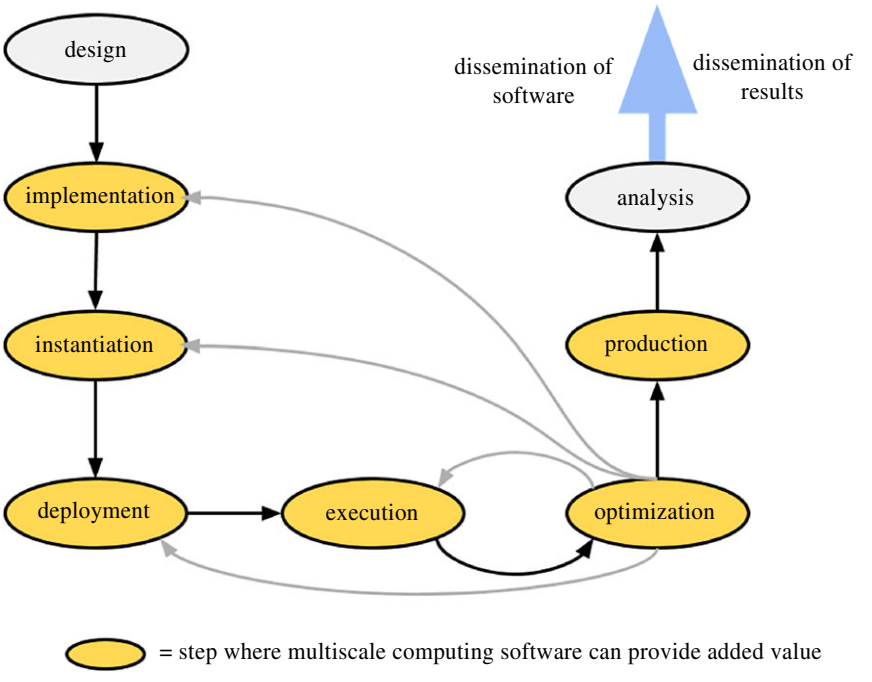

Figure 1. Overview of a typical process for developing multiscale computing applications. (Online version in colour.)

Step) and analyse its output data (Analysis Step). Lastly, researchers disseminate their work by publishing key results, and/or the software approach that they have developed to obtain these results (Dissemination Step).

4. The role of multiscale computing software

We define MCS as software which adds value during one or more stages of the multiscale application development process, and has an explicitly formulated orientation towards multiscale, multiphysics, multimodel or coupled applications. Arguably, there are six steps in the multiscale application development process where MCS can provide added value: (1) Implementation, (2) Instantiation, (3) Deployment, (4) Execution, (5) Optimization and (6) Production.

In this work, we investigate the added value of a range of MCS, attempting to include an example of each type that is commonly used. Because the number of MCS packages is very large, our analysis is not exhaustive, but focuses on major examples of each specific type that we are aware of, and that are publicly available. For example, our analysis of the potential added value of the OpenFOAM multiphysics code [51] will apply to a large extent to other multiphysics codes, such as Elmer or LAMMPS [52]. Likewise, analysis concerning the widely used OASIS-MCT coupler similarly helps to determine the added value of other couplers, such as C-Coupler1 [53] or YAC [54].

(i) Scope

An important aspect of MCS is the intended scope of the software. Here we briefly reflect on a few relevant scopes of applicability for these tools, from more specific to more generic. Software can be instance-specific (e.g. written ad-hoc for a single run or typed into an interactive terminal), problem-specific (e.g. custom-made for a clay-polymer MD simulation), system-specific (e.g. tailored for MD simulations), discipline-specific (e.g. intended for materials science applications)

Table 1. Overview of drawbacks when using generic MCS

type	description of drawback	example means of mitigation
adoption overhead	it takes time to understand and set up software which is written by others	good and simple build systems, clear documentation, tutorials
application overhead	it takes time to introduce domain- or problem-specific in a generic setting, and to modify generic code to facilitate an unexpected situation	make MCS non-intrusive, limit the range of features provided by the MCS, clear documentation, tutorials
search overhead	it can take time to find the right software (or it might not exist at all)	create search directories, pursue good citation practices of directly used and closely related MCS in scientific articles
increased support requirements	more support effort is necessary due to a larger community size, leading to less support per user	establish a self-supporting user community
lack of control and/or ownership	development is frequently managed by others, reduced academic credit due to not developing own tools, no control over the software installation in the case of software as a service	make code open-source, support branching developments and spin-offs, avoid centralized installations, make a clear case against reinventing the wheel

or generic. More generic software tends to have a stronger focus on ease of reuse, serves a larger community and tends to get scrutiny from people with a wider range of academic backgrounds. However, a major drawback is the need to engineer the software for a wider range of possible use cases, which may increase the effort required to develop more generic MCS.

Reusability may be limited not only to the extent that MCS is generic from a scientific perspective. Other limitations, such as restrictions on supported codes, programming languages, user types, operating systems or resource platforms can further limit both the reusability of MCS, and other aspects such as the maximum attainable size of its user community.

(ii) Advantages

Many different kinds of added value can be provided by MCS, but for the purposes of this work we place any added value advertised by these tools within four categories, each of which may apply to the aforementioned six steps in the development process.

Software may help *Curate* multiscale applications, e.g. by making activities more reproducible, more organized, more transparent and/or easier to scrutinize. Software may help to *Accelerate* multiscale applications by speeding up the progress in a process step, or to *Simplify* by reducing the amount of skills and knowledge needed to perform that step. Lastly, MCS may *Expand* the range of possibilities for the developer by introducing alternative approaches, or by providing more flexible use of existing ones.

(iii) Drawbacks

Adopting generic software for multiscale computing provides clear benefits, but choosing more generic tools over more specific ones comes with a range of drawbacks as well. These drawbacks are described in detail in table 1.

We argue that the first four of these five drawbacks apply less frequently when choosing domain- or system-specific MCS, while the fifth drawback applies to any type of externally owned or controlled software. Though a detailed drawback analysis is beyond the scope of this work, we do recommend that application developers consider these possible drawbacks prior to adopting

new MCS, and that MCS developers attempt to identify and mitigate the most serious drawbacks in their software.

5. Analysis approach

We have collectively gathered data on a range of MCS, allowing all authors to submit information about specific tools into a database using a Google Form. Each tool was examined by at least two of the authors. An empty example form is provided in the electronic supplementary information as a reference. As a starting point, we investigated a subset of the software presented by Groen *et al*. [2], upon which we then manually searched for more recent MCS. We recorded 26 tools in total, and chose to analyse 21 of them. Of the analysed tools, 20 of them are freely available to the public, while 1 tool (HMS) was freely available to the authors, and is expected to be released freely to the public in early 2019. The other five tools were omitted either because we could not access their public website or because they had been superseded with newer tools.

6. Results

(a) High-level overview

We provide a brief overview of the scope and supported platforms and patterns in table 2. Here, we find that C++ is the most widely supported language, although Python is also quite prevalent. In terms of MCPs, we see a clear segregation of tools, with a large number of tools providing support for one specific MCP. This is interesting, because the MCPs were introduced well after many of these tools were established [8].

We provide an overview of the added values from the tools in figure 2, using the approach we introduced earlier. This list includes 11 generic toolkits, 7 discipline-specific toolkits and 3 system-specific toolkits. In this figure, we can quickly distinguish several things. Firstly, tools that serve more development steps are shown with more filled boxes in the figure. Entries that contain all (or nearly all) filled boxes provide support throughout the development process, while entries with fewer filled boxes are more specific in their purpose. Using more specific tools can mitigate the adoption overhead, as there are fewer development steps that need to be incorporated. Second, the number of arrows inside each box helps indicate the completeness of added values a tool provides in that step. For example, MUSCLE 2 and MPWide both provide added value in the implementation step, but whereas MPWide only expands the range of options in this step, MUSCLE 2 also delivers curation and acceleration benefits due to it providing a more structured framework. This does not necessarily mean that MUSCLE 2 is the better option in all cases; the choice between the two may partially depend on the application need for curation and acceleration in the implementation step.

We provide summary statistics for the added values in table 3. Although our review is far from exhaustive, and many tools we examine have counterparts that are somewhat similar (e.g. OpenFOAM to Elmer, AiiDA to FabSim), it does give a general impression of which areas of added value are targeted to which extent. Based on the results, we find that the tools in our study particularly focus on the Implementation step, and less on the Instantiation and Execution steps. In terms of added values, we recognize a strong focus on simplification and curation, with many of the more recently emerged tools particularly targeting the latter.

The table also exposes a range of clear added value gaps in our examined tools. We did not record any added value towards accelerating the optimization step (and relatively little added value overall), and tools provide even fewer features that help users during the production step. The strong focus of tools on earlier phases of the development process, and relative lack of focus on later phases, could be seen as an indicator that multiscale computing as a discipline has not yet fully matured.

Table 2. Summary of MCS scope and platforms. The scope is given in the second column, supported programming languages in the third column, and supported multiscale computing patterns (Extreme Scaling (ES), Heterogeneous Multiscale Computing (HMC) or Replica Computing (RC)) in the fourth column.

name	scope	supported languages	supported patterns
ASPA	generic	C++	HMC
Amuse	discipline-specific	Python	ES,HMC,RC
Cactus	generic	Custom (language of choice)	HMC
CoHMM/D2KAS	generic	C++	HMC
CouPE	generic	C++, FORTRAN, Wrappers for C/C++/Fortran modules exist (explains my answer here)	ES,HMC
ELMER	system-specific	C++, FORTRAN, C	ES
ENISI MSM	discipline-specific	Java	ES
ESMF	discipline-specific	C++, FORTRAN	ES,HMC,RC
FLASH	discipline-specific	FORTRAN	ES
FabSim	generic	Python	ES,HMC,RC
HMS	generic	Python, C++, FORTRAN	RC
MOOSE	system-specific	C++	ES,HMC
MPWide	generic	Python, C++, C	ES
MUI	generic	C++	ES
MUSCLE2	generic	Python, C++, Java, FORTRAN, Ruby, C, Matlab	ES
MaMiCo	system-specific	C++, command-line (e.g. bash), SCons for compiling	ES,HMC
OASIS3-MCT_3.0	discipline-specific	FORTRAN, C	ES
OMFIT	discipline-specific	Python	ES,HMC,RC
Parsl	generic	Python	RC
Swift	generic	Domain-specific or bespoke language	ES,HMC,RC
TabaSCo	discipline-specific	C++, CHARM++	HMC

Table 3. Summation of added values from MCS in each development step

step	no. curate	no. accelerate	no. simplify	no. expand
implementation	10	5	14	12
instantiation	5	2	9	2
deployment	2	6	8	1
execution	4	8	8	2
optimization	4	1	6	1
production	2	7	1	2
total	27	29	46	20

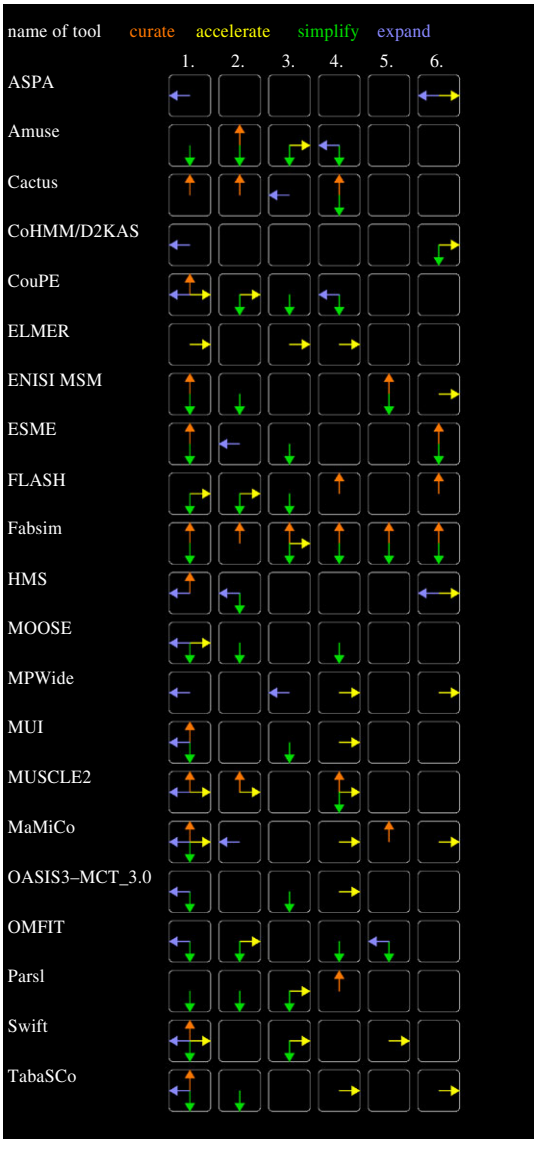

Figure 2. Overview of added value of the software tools. Here, the tools are provided one per row and the number of each relevant development step in each column (respectively (1) Implementation, (2) Instantiation, (3) Deployment, (4) Execution, (5) Optimization and (6) Production). Tools are sorted alphabetically. (Online version in colour.)

(b) Detailed analysis of selected tools

(i) Adaptive Sampling Proxy Application

The Adaptive Sampling Proxy Application (ASPA) is a toolkit for automated construction of surrogate models within a multiscale model hierarchy. It allows developers across disciplines to construct kriging surrogate models on-the-fly using data obtained from the evaluation of at-scale model components of a multiscale model. ASPA uses a local kriging strategy to limit the amount of data incorporated in an individual surrogate model and contains a metric tree database to store the collection of surrogate models and allow for their quick retrieval. The adaptive sampling method is intended for use in applications that fit the HMC pattern, specifically for cases where a surrogate model is able to approximate the microscale model well for particular model inputs.

Implementation. Expand. Expands the concept of multiscale simulation software. In addition to software consisting solely of coupled at-scale models, ASPA introduces a database to store surrogate models constructed using model output data and allows the surrogate models to be incrementally updated and quickly evaluated, including an error estimate.

Instantiation, Deployment, Execution and Optimization. To the best of our knowledge, ASPA does not directly add value on these steps of the development process.

Production. Accelerate and Expand. Accelerates evaluation of computationally demanding multiscale models and enables using surrogates in production.

(ii) MUSCLE 2

The MUltiScale Coupling Library and Environment 2 establishes couplings between at-scale model components in a systematic and discipline-agnostic manner. It takes a description of the model in terms of components and conduits between them, and executes the simulation accordingly by starting processes and opening TCP connections. Components must be linked with the MUSCLE 2 library, available in a range of languages, to be usable. Although it is not impossible to dynamically instantiate model components separately, MUSCLE 2 provides no support for this, and is mostly geared towards ES applications.

Implementation. Curate, Accelerate and Expand. Requires the model structure to be clearly described, takes care of network communications and enables coupling of very diverse models.

Instantiation. Curate and Accelerate. Unifies multiscale application definition and parameter values in a single archivable file.

Execution Curate, Accelerate, Simplify. Model description includes directions for starting the full application. Can start up all components locally in a single command, automatically establishes network connections, and logs what was done.

Deployment, Optimization and Production. To the best of our knowledge, MUSCLE 2 does not directly add value on these steps of the development process.

(iii) OASIS3-MCT

OASIS3-MCT is a so-called *coupler* which enables the coupling of models with a focus on climate model components. It originates from the Centre Européen de Recherche et Formation Avancée en Calcul Scientifique (CERFACS). OASIS3-MCT provides a high level of parallelism, and is particularly optimized for efficient interpolation and regridding as well as data exchange in coupled mesh-based applications.

Implementation. Expand. OASIS3-MCT supports one-to-many concurrent couplings, a feature which is relatively rare in other toolkits.

Deployment. Simplify. Provides a wrapper which makes deployment of all models easier.

Execution. Accelerate. OASIS3-MCT supports parallel coupling channels using MPI, clearly improving performance compared to single MPI channels, or TCP/file I/O communications. The toolkit as a whole is also heavily optimized to be fast.

Instantiation, Optimization and Production. To the best of our knowledge, OASIS3-MCT does not directly add value on these steps of the development process.

(iv) OMFIT

OMFIT is a model coupling framework which features a GUI, and is used extensively in the Fusion community. Its implementation is generic, and supports the use of parallel codes on HPC resources. It has a large range of supported modules built-in, which provide both physics solvers as well as other functionalities such a integrations with data sources and visualization tools.

Implementation. Simplify, Expand. Provides a wide range of modules that can be easily coupled, and a GUI to simplify the process of making couplings.

Instantiation. Accelerate, Simplify. Integrates with a range of experimental databases, which makes instantiation simpler and faster in a range of cases.

Execution. Simplify. OMFIT allows predefining coupling schemes, and simplify doing test runs in that way.

Optimization. Simplify, Expand. Supports a range of analysis and visualization techniques to make this step more flexible and simpler.

Deployment and Production. To the best of our knowledge, OMFIT does not directly add value on these steps of the development process.

(v) Parsl

Parsl is a Python-based parallel scripting library that supports development and execution of asynchronous and parallel data-oriented workflows (dataflows). These workflows glue together existing executables (called Apps) and Python functions with control logic written in Python. Parsl brings implicit parallel execution to standard Python scripts.

Implementation. Curate, Simplify, Expand. Provides a dependency-driven workflow model. Allows creation of complex workflow using any infrastructure (from laptop to supercomputer) through one script. Enables the creation of interactive data-intensive workflows.

Instantiation. Simplify. Simplifies the passage of data between models.

Deployment. Accelerate, Simplify. Single scripts map directly to a range of resource platforms.

Execution. Curate. Provides a range of sophisticated data handling and error management features.

Optimization and Production. To the best of our knowledge, Parsl does not directly add value on these steps of the development process.

(vi) Macro-Micro-Coupling

The Macro-Micro-Coupling Tool (MaMiCo) [40,41] attempts to ease the development of and share existing coupling algorithms for particle-mesh, in particular for molecular-continuum, flow simulations and is therefore a system-specific MCS. Separating continuum and molecular dynamics (MD) solvers from the actual coupling algorithm via strict interfacing and also separating coupling steps in a modular way within MaMiCo, arbitrary solvers can be plugged together. The software supports execution on distributed-memory platforms using MPI, is written in C++ and uses SCons for compiling.

Implementation. Curate, Accelerate, Simplify and Expand. MaMiCo features well-defined interfaces to support among others debugging and unit/integration testing. After a certain accustomization phase, this also accelerates and simplifies code development. Accordingly, new algorithms to couple MD and continuum solvers can be easily incorporated as demonstrated in [40] (expansion).

Instantiation. Expand. Once a new coupling algorithm for a particular flow problem has been incorporated, this coupling can be evaluated using any interfaced particle/continuum package immediately.

Deployment. None.

Execution. Accelerate. Through the multi-instance sampling in MaMiCo [40], faster time-to-solution is reached, although at higher compute cost. Incorporation of noise filters is a work in progress and is meant to further accelerate sampling/noise reduction in MD.

Optimization. Curate. Standardized coarse-grained output is provided through MaMiCo (csv and vtk formats), allowing to compare results for different couplings.

Production. See acceleration aspect for multiscale software execution above.

(vii) FabSim

FabSim is an automation environment which is optimized for curating complex multiscale workflows and providing one-liner access to perform simulations on remote machines. Its base implementation is generic, and has been used in disciplines ranging from materials to blood flow

and migration. It is designed to be easy to modify, and has resulted in several domain-specific spin-off tools (e.g. FabMD [43] and FabFlee [55]) over the years.

Implementation. Curate and Simplify. Easily combine execution patterns. Curate workflow building blocks.

Instantiation. Curate. Curate collections of simulation input.

Deployment. Curate and Accelerate. Automate deployment. Reuse working configs from other users.

Execution. Curate and Simplify. Curate simulation output and environment. Simplify execution on remote resources.

Optimization. Curate and Simplify. Curate output and environment. One-liner commands for parameter explorations.

Production. Curate. Curate output and environment. Curate multi-machine workflows in single commands.

7. Conclusion

Several conclusions can be drawn from our analysis. First, the availability of MCS has become considerably broader since 2014 [2], with many of the newer tools aiming explicitly to simplify application development. Second, Python has now become one of the leading platforms to help facilitate multiscale applications (e.g. see FabSim, OMFIT and Parsl for recent examples). Third, in terms of generality, we now find generic MCS being applied in all major computational research disciplines. However, the tools remain quite specific in other aspects: most MCS are far from language-agnostic and 16 of the 21 tools are intended for a subset of applications that fit one or two particular MCPs [8].

Until now, the majority of the MCS development has focused on integrating frameworks to combine at-scale models together to form a multiscale model. As the multiscale computing field remains still quite immature, there has been a corresponding lack of development of tools to ease deployment, configuration, debugging, profiling, optimization and visualization of multiscale models. As a consequence, we find in our analysis that relatively few tools provide added value in the later steps in the development process (especially deployment, optimization and production). These steps are both labour-intensive and crucial for the long-term success of multiscale applications, and the introduction of mature MCS there may drive the research impact in the field as a whole.

Within our work we also briefly reflect on the (frequently under-documented) drawbacks associated with MCS. A systematic analysis of drawbacks, describing the trade-offs expected when adopting the software, does not solely serve the community as a whole. It can also give a much clearer justification to the existence of individual tools, particularly when two tools with similar added values provide these benefits with substantially different kinds of drawbacks.

Major progress has been made towards providing discipline-agnostic MCS. Now, our next targets should be to address the previously overlooked parts of application development, and to more clearly present and curate the adoption drawbacks and benefits to the users.

Data accessibility. This article has no additional data.

Authors' contributions. D.G. coordinated the project, wrote the main content on the analysis of added values, designed the analysis survey form and developed the scripts to convert the analysis data into figures and tables. K.L. wrote and refined §2, while D.S. helped collect data by performing an independent literature search for multiscale computing tools. L.V. wrote the section on MUSCLE 2, and P.N. wrote the section on MaMiCo. All authors helped collect data for the survey, provided revisions for the manuscript, read and approved the manuscript.

Competing interests. We declare we have no competing interests.

Funding. This project has received funding from the European Union's Horizon 2020 Research and Innovation Programme under grant agreement nos. 800925 and 671564. P.N. acknowledges financial support through the project 'Task-based load balancing and auto-tuning in particle simulations' (TaLPas), grant no. 01IH16008B.

Acknowledgements. We are grateful to the organizers and participants of the Lorentz Center workshop 'Multiscale Computing: From the Desktop to the Exascale', for providing an excellent discussion platform, which substantially helped us in defining the paper concept. We are also grateful to David Coster for providing information on the OMFIT and AiiDA toolkits.

References

1. Babur Ö, Smilauer V, Verhoeff T, van den Brand M. 2015 A survey of open source multiphysics frameworks in engineering. *Procedia Comput. Sci.* **51**, 1088–1097. (doi:10.1016/j.procs.2015.05.273)
2. Groen D, Zasada SJ, Coveney PV. 2014 Survey of multiscale and multiphysics applications and communities. *IEEE Comput. Sci. Eng.* **16**, 34–43. (doi:10.1109/MCSE.2013.47)
3. Kevrekidis IG, Samaey G. 2009 Equation-free multiscale computation: algorithms and applications. *Annu. Rev. Phys. Chem.* **60**, 321–344. (doi:10.1146/annurev.physchem.59.032607.093610)
4. Weinan E, Engquist B, Li X, Ren W, Vanden-Eijnden E. 2007 Heterogeneous Multiscale Methods: a review. *Commun. Comput. Phys.* **2**, 367–450.
5. Ashby S *et al.* 2010 The opportunities and challenges of exascale computing. *Summary Report of the Advanced Scientific Computing Advisory Committee (ASCAC) Subcommittee*, pp. 1–77.
6. Hoekstra A, Lorenz E, Falcone J-L, Chopard B. 2007 Towards a complex automata framework for multi-scale modeling: formalism and the scale separation map. In *Computational science ICCS 2007*, vol. 4487 (eds Y. Shi, G. van Albada, J. Dongarra and P. Sloot). Lecture Notes in Computer Science, pp. 922–930. Berlin, Germany: Springer.
7. Falcone J-L, Chopard B, Hoekstra A. 2010 MML: towards a multiscale modeling language. *Procedia Comput. Sci.* **1**, 819–826. (doi:10.1016/j.procs.2010.04.089)
8. Alowayyed S, Groen D, Coveney PV, Hoekstra AG. 2017 Multiscale computing in the exascale era. *J. Comput. Sci.* **22**, 15–25. (doi:10.1016/j.jocs.2017.07.004)
9. May JM, Ashby SF. 2007 Multiphysics simulations and petascale computing. In *Petascale computing* (ed. DA Bader), pp. 96–115. London, UK: Chapman and Hall.
10. Epperly TG, Kumfert G, Dahlgren T, Ebner D, Leek J, Prantl A, Kohn S. 2012 High-performance language interoperability for scientific computing through Babel. *Int. J. High Perform. Comput. Appl.* **26**, 260–274. (doi:10.1177/1094342011414036)
11. Barton N *et al.* 2007 Co-op, version 00. Tech. Rep. UCRL-CODE-232892, Lawrence Livermore National Laboratory.
12. Jefferson D. 2006 Relationship between Co-op and MPI-2. Tech. Rep. UCRL-TR-225783, Lawrence Livermore National Laboratory.
13. Barton NR, Bernier JV, Knap J, Sunwoo AJ, Cerreta EK, Turner TJ. 2011 A call to arms for task parallelism in multi-scale materials modeling. *Int. J. Numer. Methods Eng.* **86**, 744–764. (doi:10.1002/nme.3071)
14. Barton NR, Knap J, Arsenlis A, Becker R, Hornung RD, Jefferson DR. 2008 Embedded polycrystal plasticity and adaptive sampling. *Int. J. Plast.* **24**, 242–266. (doi:10.1016/j.ijplas.2007.03.004)
15. Knap J, Barton NR, Hornung RD, Arsenlis A, Becker R, Jefferson DR. 2008 Adaptive sampling in hierarchical simulation. *Int. J. Numer. Methods Eng.* **76**, 572–600. (doi:10.1002/nme.2339)
16. Leiter KW, Barnes BC, Becker R, Knap J. 2018 Accelerated scale-bridging through adaptive surrogate model evaluation. *J. Comput. Sci.* **27**, 91–106. (doi:10.1016/j.jocs.2018.04.010)
17. Dorr MR. 2012 ASPA - adaptive sampling proxy application. Tech. Rep. LLNL-SM-595112, Lawrence Livermore National Laboratory.
18. Borgdorff J, Mamonski M, Bosak B, Kurowski K, Ben Belgacem M, Chopard B, Groen D, Coveney PV, Hoekstra AG. 2014 Distributed multiscale computing with MUSCLE 2, the multiscale coupling library and environment. *J. Comput. Sci.* **5**, 719–731. (doi:10.1016/j.jocs.2014.04.004)
19. Hegewald J, Krafczyk M, Tölke J, Hoekstra A, Chopard B. 2008 An agent-based coupling platform for complex automata. In *Proc. of the 8th Int. Conf. on computational science, part II, ICCS '08*, pp. 227–233. Berlin, Germany: Springer.
20. Hoekstra A, Caiazzo A, Lorenz E, Falcone J-L, Chopard B. 2010 *Complex automata: multi-scale modeling with coupled cellular automata*, pp. 29–57. Berlin, Germany: Springer.

21. Hoekstra AG, Lorenz E, Falcone J-L, Chopard B. 2007 Toward a complex automata formalism for multiscale modeling. *Int. J. Multiscale Comput. Eng.* **5**, 491–502. (doi:10.1615/IntJMultCompEng.v5.i6)
22. Borgdorff J, Falcone J-L, Lorenz E, Bona-Casas C, Chopard B, Hoekstra AG. 2013 Foundations of distributed multiscale computing: formalization, specification, and analysis. *J. Parallel Distrib. Comput.* **73**, 465–483. (doi:10.1016/j.jpdc.2012.12.011)
23. Viceconti M, Bnà S, Tartarini D, Sfakianakis S, Grogan J, Walker D, Gamble S, Testi D. 2018 Vph-hf: A software framework for the execution of complex subject-specific physiology modelling workflows. *J. Comput. Sci.* **25**, 101–114. (doi:10.1016/j.jocs.2018.02.009)
24. Knap J, Spear C, Leiter K, Becker R, Powell D. 2016 A computational framework for scale-bridging in multi-scale simulations. *Int. J. Numer. Methods Eng.* **108**, 1649–1666. (doi:10.1002/nme.5270)
25. Knap J, Spear CE, Borodin O, Leiter KW. 2015 Advancing a distributed multi-scale computing framework for large-scale high-throughput discovery in materials science. *Nanotechnology* **26**, 434004. (doi:10.1088/0957-4484/26/43/434004)
26. Pelupessy F, van Elteren A, de Vries N, McMillan S, Drost N, Zwart SP. 2013 The astrophysical multipurpose software environment. *Astron. Astrophys.* **557**, A84. (doi:10.1051/0004-6361/201321252)
27. Pelupessy I, van Werkhoven B, van Elteren A, Viebahn J, Candy A, Zwart SP, Dijkstra H. 2016 OMUSE: Oceanographic multipurpose software environment. In *2016 IEEE 12th Int. Conf. on e-science (e-science)*, pp. 399–399.
28. Hill C, DeLuca C, Balaji Suarez M, Silva Ad. 2004 The architecture of the earth system modeling framework. *Comput. Sci. Eng.* **6**, 18–28. (doi:10.1109/MCISE.2004.1255817)
29. Germann TC, McPherson AL, Belak JF, Richards DF. 2013 Exascale co-design center for materials in extreme environments (ExMatEx) annual report - year 2. Tech. Rep. LLNL-SR-647437, Lawrence Livermore National Laboratory.
30. Pavel RS, McPherson AL, Germann TC, Junghans C. 2015 Database assisted distribution to improve fault tolerance for multiphysics applications. In *Proc. of the 2nd Int. workshop on hardware-software co-design for high performance computing*, *Co-HPC '15*, pp. 4:1–4:8. New York, NY, USA: ACM.
31. Dorr M, Barton N, Keasler J, Li F. 2014 CoEVP: A co-design embedded viscoplasticity scale-bridging proxy app for ExMatEx. *Lawrence Livermore National Laboratory, Technical Report LLNL-SM-655180*.
32. Pavel R, Junghans C, Mniszewski SM, Germann TC. 2017 Using Charm++ to support multiscale multiphysics on the Trinity supercomputer. 15th Annual Workshop on Charm++ and its Applications.
33. Roehm D, Pavel RS, Barros K, Rouet-Leduc B, McPherson AL, Germann TC, Junghans C. 2015 Distributed database kriging for adaptive sampling (D^2KAS). *Comput. Phys. Commun.* **192**, 138–147. (doi:10.1016/j.cpc.2015.03.006)
34. Rouet-Leduc B *et al.* 2014 Spatial adaptive sampling in multiscale simulation. *Comput. Phys. Commun.* **185**, 1857–1864. (doi:10.1016/j.cpc.2014.03.011)
35. Mei Y, Carbo A, Hontecillas R, Hoops S, Liles N, Lu P, Philipson C, Bassaganya-Riera J. 2014 ENISI MSM: A novel multi-scale modeling platform for computational immunology. In *2014 IEEE Int. Conf. on bioinformatics and biomedicine (BIBM)*, pp. 391–396. (10.1109/BIBM.2014.6999190).
36. Miller RE, Tadmor EB. 2009 A unified framework and performance benchmark of fourteen multiscale atomistic/continuum coupling methods. *Modell. Simul. Mater. Sci. Eng.* **17**, 053001. (doi:10.1088/0965-0393/17/5/053001)
37. Larson J, Jacob R, Ong EFall. 2005 The model coupling toolkit: a new Fortran90 toolkit for building multiphysics parallel coupled models. *Int. J. High Perform. Comput. Appl.* **19**, 277–292. (doi:10.1177/1094342005056115)
38. Tang Y-H, Kudo S, Bian X, Li Z, Karniadakis GE. 2015 Multiscale universal interface: A concurrent framework for coupling heterogeneous solvers. *J. Comput. Phys.* **297**, 13–31. (doi:10.1016/j.jcp.2015.05.004)
39. Craig A, Valcke S, Coquart L. 2017 Development and performance of a new version of the OASIS coupler, OASIS3-MCT_3.0. *Geosci. Model Dev.* **10**, 3297–3308. (doi:10.5194/gmd-10-3297-2017)

40. Neumann P, Bian X. 2017 MaMiCo: Transient multi-instance molecular-continuum flow simulation on supercomputers. *Comput. Phys. Commun.* **220**, 390–402. (doi:10.1016/j.cpc.2017.06.026)
41. Neumann P, Flohr H, Arora R, Jarmatz P, Tchipev N, Bungartz H-J. 2016 MaMiCo: Software design for parallel molecular-continuum flow simulations. *Comput. Phys. Commun.* **200**, 324–335. (doi:10.1016/j.cpc.2015.10.029)
42. Groen D, Bhati AP, Suter J, Hetherington J, Zasada SJ, Coveney PV. 2016 FabSim: facilitating computational research through automation on large-scale and distributed e-infrastructures. *Comput. Phys. Commun.* **207**, 375–385. (doi:10.1016/j.cpc.2016.05.020)
43. Suter JL, Groen D, Coveney PV. 2015 Chemically specific multiscale modeling of clay-polymer nanocomposites reveals intercalation dynamics, tactoid self-assembly and emergent materials properties. *Adv. Mater.* **27**, 966–984. (doi:10.1002/adma.201403361)
44. Itani MA, Schiller UD, Schmieschek S, Hetherington J, Bernabeu MO, Chandrashekar H, Robertson F, Coveney PV, Groen D. 2015 An automated multiscale ensemble simulation approach for vascular blood flow. *J. Comput. Sci.* **9**, 150–155. (doi:10.1016/j.jocs.2015.04.008)
45. Pizzi G, Cepellotti A, Sabatini R, Marzari N, Kozinsky B. 2016 AiiDA: automated interactive infrastructure and database for computational science. *Comput. Mater. Sci.* **111**, 218–230. (doi:10.1016/j.commatsci.2015.09.013)
46. Gebbie-Rayet J, Shannon G, Loeffler HH, Laughton CA. 2016 Longbow: a lightweight remote job submission tool. *J. open Res. Software* **4**.
47. Ludäscher B, Altintas I, Berkley C, Higgins D, Jaeger E, Jones MB, Lee EA, Tao J, Zhao Y. 2006 Scientific workflow management and the Kepler system. *Concurr. Comput. Pract. Exp.* **18**, 1039–1065. (doi:10.1002/(ISSN)1532-0634)
48. Wilde M, Hategan M, Wozniak JM, Clifford B, Katz DS, Foster I. 2011 Swift: a language for distributed parallel scripting. *Parallel Comput.* **39**, 633–652. (doi:10.1016/j.parco.2011.05.005)
49. Balasubramanian V, Treikalis A, Weidner O, Jha S. 2016 Ensemble toolkit: Scalable and flexible execution of ensembles of tasks. In *2016 45th Int. Conf. on parallel processing (ICPP), Philadelphia, PA, 16–19 August*, pp. 458–463. IEEE.
50. Babuji Y, Brizius A, Chard K, Foster I, Katz DS, Wilde M, Wozniak J. 2017 Introducing parsl: A Python Parallel Scripting Library. (doi:10.5281/zenodo.891533).
51. Jasak H, Jemcov A, Tukovic Z. 2007 OpenFOAM: A C++ library for complex physics simulations. In *Int. workshop on coupled methods in numerical dynamics, Dubrovnik, Croatia, 19–21 September*, vol. 1000, pp. 1–20. IUC Dubrovnik, Croatia. Zagreb, Croatia: Faculty of Mechanical Engineering and Naval Architecture, University of Zagreb.
52. Plimpton SJ. 1995 Fast parallel algorithms for short-range molecular-dynamics. *J. Comp. Phys.* **117**, 1. (doi:10.1006/jcph.1995.1039)
53. Liu L, Yang G, Wang B, Zhang C, Li R, Zhang Z, Ji Y, Wang L. 2014 C-Coupler1: a chinese community coupler for earth system modeling. *Geosci. Model Dev.* **7**, 2281–2302. (doi:10.5194/gmd-7-2281-2014)
54. Hanke M, Redler R, Holfeld T, Yastremsky M. 2016 YAC 1.2.0: new aspects for coupling software in Earth system modelling. *Geosci. Model Dev.* **9**, 2755–2769. (doi:10.5194/gmd-9-2755-2016)
55. Suleimenova D, Bell D, Groen D. 2017 Towards an automated framework for agent-based simulation of refugee movements. In *2017 winter simulation Conf, (WSC), Las Vegas, NV, 3–6 December*, pp. 1240–1251. IEEE.