Patterns for High Performance Multiscale Computing

Alowayyed, S.; Piontek, T.; Suter, J.L.; Hoenen, O.; Groen, D.; Luk, O.; Bosak, B.; Kopta, P.; Kurowski, K.; Perks, O.; Brabazon, K.; Jancauskas, V.; Coster, D.; Coveney, P.V.; Hoekstra, A.G.

DOI
10.1016/j.future.2018.08.045

Publication date
2019

Document Version
Final published version

Published in
Future Generation Computer Systems

Future Generation Computer Systems 91 (2019) 335–346

Contents lists available at ScienceDirect

Future Generation Computer Systems

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

Patterns for High Performance Multiscale Computing

S. Alowayyed [a,b,*], T. Piontek [c], J.L. Suter [d], O. Hoenen [e], D. Groen [f,d], O. Luk [e], B. Bosak [c], P. Kopta [c], K. Kurowski [c], O. Perks [g], K. Brabazon [g], V. Jancauskas [h], D. Coster [e], P.V. Coveney [d], A.G. Hoekstra [a,i]

[a] Computational Science Lab, Institute for Informatics, Faculty of Science, University of Amsterdam, The Netherlands
[b] King Abdulaziz City for Science and Technology (KACST), Riyadh, Saudi Arabia
[c] Poznań Supercomputing and Networking Center, Poznań, Poland
[d] Centre for Computational Science, University College London, United Kingdom
[e] Max-Planck-Institut für Plasmaphysik, Garching, Germany
[f] Department of Computer Science, Brunel University London, United Kingdom
[g] ARM Ltd., Warwick, United Kingdom
[h] Leibniz-Rechenzentrum der Bayerischen Akademie der Wissenschaften, Garching, Germany
[i] ITMO University, Saint-Petersburg, Russian Federation

HIGHLIGHTS

- We introduce the idea of the Multiscale Computing Patterns (MCP).
- We present the MCP software for High Performance Multiscale Computing.
- To simplify and automate the execution of complex multiscale simulations on HPC.
- Also to provide both application-specific and pattern-specific performance optimisation.
- We test the performance and the resource usage for three multiscale models (two MCPs).
- We demonstrate how the software automates resource selection and load balancing.

ARTICLE INFO

Article history:
Received 9 January 2018
Received in revised form 18 July 2018
Accepted 27 August 2018
Available online 10 September 2018

Keywords:
Multiscale computing
High performance computing
Modelling methodology
Distributed computing
Model coupling

ABSTRACT

We describe our Multiscale Computing Patterns software for High Performance Multiscale Computing. Following a short review of Multiscale Computing Patterns, this paper introduces the Multiscale Computing Patterns Software, which consists of description, optimisation and execution components. First, the description component translates the task graph, representing a multiscale simulation, to a particular type of multiscale computing pattern. Second, the optimisation component selects and applies algorithms to find the most suitable mapping between submodels and available HPC resources. Third, the execution component which a middleware layer maps submodels to the number and type of physical resources based on the suggestions emanating from the optimisation part together with infrastructure-specific metrics such as queueing time and resource availability. The main purpose of the Multiscale Computing Patterns software is to leverage the Multiscale Computing Patterns to simplify and automate the execution of complex multiscale simulations on high performance computers, and to provide both application-specific and pattern-specific performance optimisation. We test the performance and the resource usage for three multiscale models, which are expressed in terms of two Multiscale Computing Patterns. In doing so, we demonstrate how the software automates resource selection and load balancing, and delivers performance benefits from both the end-user and the HPC system level perspectives.

1. Introduction

Multiscale modelling & simulation has become a well-established way to study complex phenomena that encompass multiple space and time scales [1]. In this approach, a multiscale model is constructed by combining, or *coupling*, a collection of single-scale submodels, each of which captures processes on a

* Corresponding author at: Computational Science Lab, Institute for Informatics, Faculty of Science, University of Amsterdam, The Netherlands.
E-mail addresses: S.A.Alowayyed@uva.nl (S. Alowayyed), A.G.Hoekstra@uva.nl (A.G. Hoekstra).

https://doi.org/10.1016/j.future.2018.08.045

distinct space and time scale; see e.g. [2–4]. Multiscale modelling is widely used in most areas of science and engineering [5], such as biomedicine [6–8], fusion [9,10], material science [10,11], energy [12] and engineering [10,13]. It is self-evident that any high-fidelity multiscale model must employ substantial high performance computing resources, since the individual single scale models comprising it themselves have to run on such machines.

In addition to specific multiscale applications, a number of tools and frameworks which assist in multiscale computing have been established. These range from domain-specific frameworks such as AMUSE [14] and OASIS-MCT [15] to solver-specific frameworks such as the MOOSE framework for finite-element codes [16] and fully generic frameworks [1,3,17–19] encompassing related coupling tools such as the Multiscale Coupling Library and Environment 2 (MUSCLE2) [20].

We have previously developed the Multiscale Modelling and Simulation Framework (MMSF) [3,17–19], which provides a theoretical and methodological framework for constructing multiscale simulations in four main stages. First, we model multiscale phenomena as collections of single-scale submodels then decide on which models interact with each other and how. Single scale models and couplings are presented within a Scale Separation Map, allowing us to describe and compare multiscale models on a conceptual level. Second, we specify the single scale models, their couplings and interactions using the Multiscale Modelling Language [1,18]. Third, we convert these definition to a fully implemented multiscale model, currently relying on MUSCLE2 [20] (although the concepts of the MMSF can also be applied to other coupling environments such as AMUSE [14]). An important property of MUSCLE2 is the separation of concerns that it affords. Submodels are not aware of any other components. Moreover, required adaptations are minimal on the level of a submodel in order for it to be incorporated into a multiscale model implemented with MUSCLE2. Fourth, we deploy and execute the multiscale application on a set of computational resources. Developers and users can run different submodels on different machines [4], using for example the QCG middleware [21], a paradigm that we call Distributed Multiscale Computing [4].

Knap et al. [22] have previously proposed a distributed multiscale computing framework that supports the on-demand execution of microscale models coupled to a macroscale model (very similar to one of the computing patterns we proposed in [23]), using large scale supercomputing resources. Although their framework has, to our knowledge, not yet been applied outside the domain of materials science for which it was originally created, the authors do propose a general conceptual framework that could be adopted for use in other disciplines. This resonates with our vision of generic multiscale computing environments, where a separation of concerns is achieved between multiscale modelling & simulation on the one hand, and deploying and executing a multiscale simulation in a given HPC environment on the other.

Our "Lego-based" philosophy for the construction and execution of multiscale applications relies on single scale submodels and their interactions, and results in more degrees of freedom for both programming and executing a multiscale simulation. To efficiently execute multiscale applications on high-end HPC machines, a number of challenges have to be addressed, such as load balance (providing resources to each of the single scale models), fault tolerance (sometimes instantiations of single scale models may fail) and energy awareness (depending on properties of single scale models, potentially also in combination with load balancing, energy aware optimisation). Our intention is that these challenges are handled in a generic way, as far as possible avoiding the imposition of that burden on the developers of multiscale applications. Those developers should take care of the scale bridging mechanisms and the efficiency of the single scale models, while the challenges of execution within a High Performance Computing (HPC) environment should be addressed through a generic layer added to MMSF that we call *Multiscale Computing Patterns* (MCPs) [23].

We defined MCPs as "high-level call sequences that exploit the functional decomposition of multiscale models in terms of single scale models" [23], and distinguished three patterns: Extreme Scaling, Heterogeneous Multiscale Computing and Replica Computing. Each of these patterns is described using a generic task graph that aids in understanding how to best map these patterns to HPC resources. In addition to the generic task graph, an MCP contains performance information about single scale models, an XML-based specification of the multiscale application named xMML [20], and a set of algorithms and heuristics used to combine this into input files for the execution environment. In this paper, we report on the design and implementation of the MCP software, and present the first results of executing multiscale simulations using MCPs, including discussions on the added value of using such solutions for High Performance Multiscale Computing. Here, we mainly integrate these MCPs with MMSF to increase the effectiveness by means of which we can develop, deploy and execute multiscale simulations on existing petascale and emerging exascale resources [23].

The MCP software architecture consists of a *description component*, an *optimisation component* and an *execution component*. In the description component the software uses the task graph of the specific multiscale model, in combination with auxiliary information (e.g., definitions of single-scale models), to identify the type of pattern and create input definitions for the optimisation component. In the optimisation component, the software selects and applies a set of optimisation algorithms to identify a range of efficient mappings of the submodels in the application to specific HPC resources. Lastly, the execution component is a middleware layer which identifies the optimal mapping of submodels to the available resources, taking additionally into account queueing times and resource occupancy. Moreover, the execution component deploys and executes the application, with all of its submodels on the target resources. Three examples of using Multiscale Computing Patterns software are illustrated and examples of cost functions are worked out, showing that a wide range of variables for Multi-Objective Optimisation algorithms can be chosen. The idea is that Multiscale Computing Patterns software will automatically detect which cost functions and algorithms to select based on the type of pattern and user requirements.

The structure of our paper is as follows. We describe the MCPs in Section 2, and introduce the Multiscale Computing Patterns software and its components in Section 3. In Section 4, we provide by way of proof of concept three examples of the use of the Multiscale Computing Patterns software. Finally, we provide a discussion and conclusion in Section 5.

2. Multiscale computing patterns and high performance multiscale computing

In this section, we discuss the concept of Multiscale Computing Patterns and express the MCPs as generic task graphs. For full details, we refer to Alowayyed et al. [23]. Fig. 1 shows the generic task graphs for all three computing patterns.

The Extreme Scaling (ES) pattern represents a type of multiscale model where one *primary* single-scale model is coupled to a set of serial and/or parallel *auxiliary* models on any scale as shown in Fig. 1((a) and (b)). The primary model[1] is compute intensive, energy hungry, and highly scalable, whereas the auxiliary models are not. Therefore, the efficiency of this type of multiscale models is highly dependent on the efficiency of the primary model and

[1] We assume one primary model here. In practice, ES could consist of more than one primary model.

the primary–auxiliary interactions. Assuming that developers have implemented the primary model efficiently, the main aim is to reach a minimal interference between primary and auxiliaries. This can be done using load balancing while ensuring minimal communication between primary and auxiliaries. The serial auxiliary model can give rise to strong underutilisation of available resources (e.g. if it does not scale to a large number of cores), and special mechanisms to handle such situation are required.

The Heterogeneous Multiscale Computing (HMC) pattern (Fig. 1(c)) represents the typical form of macro-micro coupling, where the numerical solver at the macro-scale level requires input from multiple micro scale model instantiations (for instance to compute a spatially varying quantity, such as for example a constitutive equation, say a viscosity in a flow problem). Thus, the number of micro-scale models is dynamic and largely dependent on the dynamic evolution of the macro-model. The HMC manager has some control over the number of micro-scale models, by preventing redundant calculations (by storing results of previous microscale simulations in a HMC database and where possible extracting results from the database, e.g. by interpolations between results obtained earlier), and spawning extra micro-scale models, when necessary. Typically, the number of micro-scale models will be very large and a single microscale run may require substantial computing resources and, hence, dominate computing and energy cost.

In the Replica Computing (RC) pattern a large number of copies of tera- and/or peta-scale simulations are needed to produce statistically robust results. These replicas are not part of an overarching structure like HMC, but are spawned in the initial step. In this step, the parameter space for parameter sweeping is set. Then, simulations and data processing per replica take place. Both static and dynamic flavours of RC are considered in Fig. 1((d), (e)). All replicas execute independently of each other. If a replica (i.e. a simulation) fails, the RC patterns affords some level of fault tolerance, taking into account maintaining the overall statistics. This is the main difference with HMC. On the other hand, HMC and RC are similar in terms of load balance issues.

3. Design of multiscale computing patterns software

The Multiscale Computing Patterns software consists of three parts: (1) the Description Part, where the user describes the multiscale application, (2) the Optimisation Part, where the software predicts and optimises the application performance, (3) and the Execution Part, where the application is deployed using an underlying resource allocation service (in our case the QCG middleware). We present the components of the Multiscale Computing Patterns software, and their interrelations, in Fig. 2.

The logical description and the complete set of requirements of a multiscale application is collected in the description component. This part relies on concepts from the Multiscale Modelling and Simulation Framework. It is helpful to facilitate the work of the end user and provide a single input mechanism for all multiscale applications, as well as detecting the type of MCP. The MMSF xMML description file was extended for Replica Computing to accommodate the notation of the number of replicas. The optimisation component determines which MCP optimisation applies, collects required performance figures, and calculates the relevant metrics (e.g. parallel efficiency, throughput, energy usage). Based on these results, a constrained optimisation is performed resulting in a small set of the most suitable execution scenarios which are passed to the execution component. The role of the execution component is to select the best allocation plan, based on the availability of the requested resources and cost criteria (time to complete, energy usage), and to start and monitor the execution.

3.1. Description component

The Description component (top layer in Fig. 2) contains an architecture-agnostic definition of the multiscale application, and its main requirements. It builds on concepts from the Multiscale Modelling and Simulation Framework. The description component consists of a task graph, submodel definitions, simulation and middleware parameters and user information, all feeding into the Translation Service.

The task graph is expressed in the form of a highly adaptable textual description (xMML, see [20]) which is used to detect motifs (repetitive submodels and dependencies). The task graph is also needed to observe workflow related issues such as the expected frequency of communication between submodels.

Submodel definitions contain all the information required for a single-scale model to run. This includes information on submodel-specific dependencies, and the resource requirements for each submodel (e.g., mandatory use of GPU-architectures, or a minimal memory requirement per core). The description component may rely on previously developed tools such as MAD/MaMe [24], and can already leverage existing configuration information from the FabSim automation environment [25].

All the simulation and middleware parameters are collected in a separate component. This includes input parameters, the required environment modules and resource limits for the overall simulation (all submodels and coupling library). Also, this component holds all information needed to compose the multiscale simulation (e.g. using MUSCLE2) and to execute the simulation (e.g. using QCG Middleware [26]) using (distributed) HPC infrastructures such as the Experiment Execution Environment (EEE). This component is designed such that existing known machine configurations from FabSim (machines.yml) can be directly reused in the context. Similarly, user-specific information can be directly reused from existing FabSim configurations (machines_user.yml). We present an example of the reuse of FabSim information within this context as part of the Binding Affinity application described in Section 4.2.

These three pieces of information are then supplied to a translation service, which merges and converts them into a format suitable for the optimisation component. Currently, the translation tool is application specific, and produces two xml files as output. One file, matrix.xml, is shown in Listing 1 in Appendix A and contains templated information from the submodel definitions. The other file, multiscale.xml shown in Listing 2 (Appendix A), has information from the simulation and middleware parameters.

3.2. Optimisation component

The main software tool within the Optimisation component is the Pattern-Driven Planner. This tool requires input from both the Translation service as well as the Node Description List. The node description list is updated regularly to reflect the current status of available nodes in the targeted supercomputers, and contains information of node types. A single node type represents a set of nearly identical nodes in terms of hardware configuration (e.g. processor type). The node types should be defined based on knowledge gathered a priori by the middleware from the infrastructure provider. Table 1 shows an example of node types.

The second layer contains the Pattern-Driven Planner and the Performance Estimator components. The Pattern-Driven Planner component collects measurements and/or predictions of performance of submodels, under various execution scenarios, from the Performance Estimator. Then, it uses this information to compute required cost functions (e.g efficiency, throughput, energy usage, or a combination) on available resources. Given the specific MCP and all other available information, the Pattern-Driven Planner performs a constrained optimisation against these cost functions,

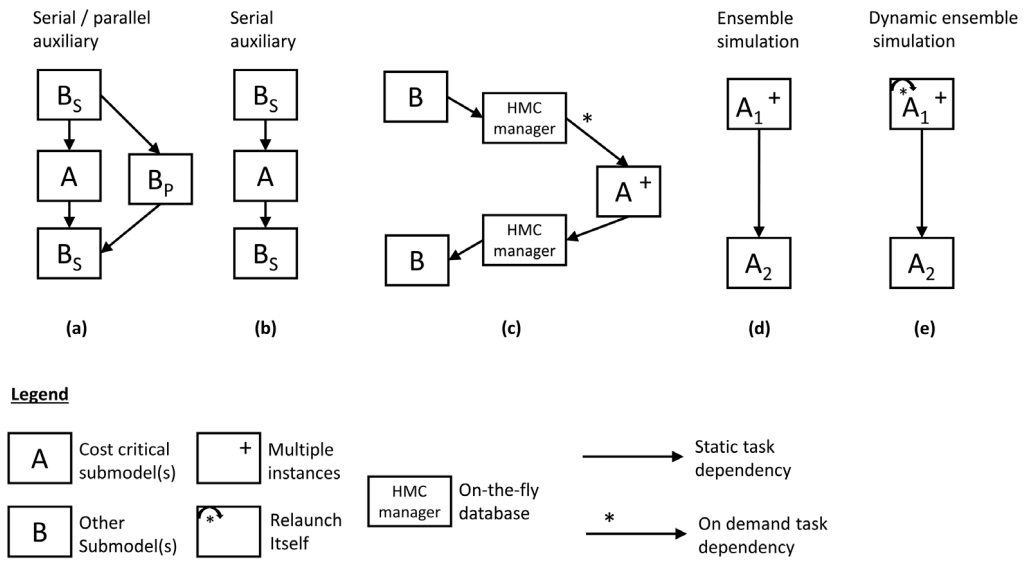


Fig. 1. Generic task graphs for the Extreme Scaling computing pattern (a,b), the Heterogeneous Multiscale Computing pattern (c) and the Replica Computing pattern (d,e). (a) shows the case where the auxiliaries B_P are running in parallel with the primary model A, while in both (a) and (b) auxiliaries B_S are in series with the primary model. In (c) multiple instances of the cost critical micro submodel (A) are called on-demand by the macro submodel (B). The macro-scale solver requires input from micro-scale solvers at every time step. (d) shows the case where multiple instances of submodels interact in phases, while in (e) the same operation is shown with addition to a self-relaunch mechanism.

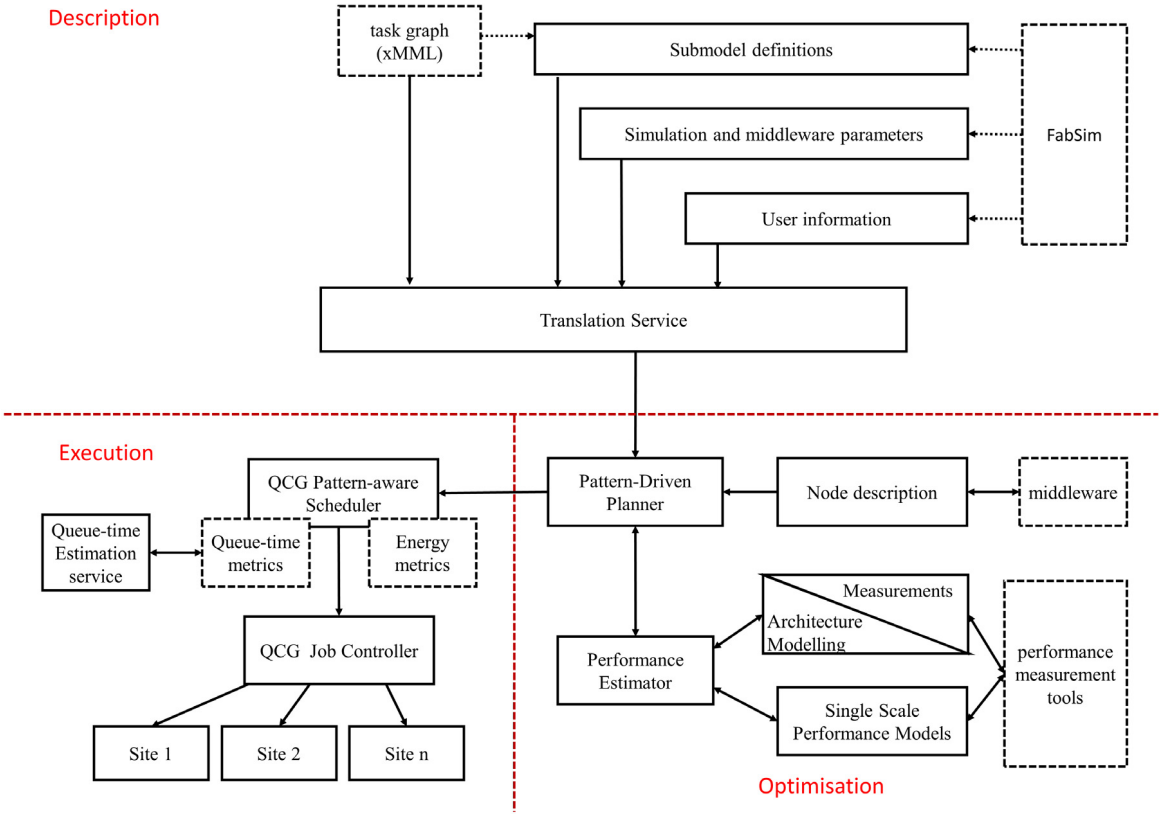


Fig. 2. Architecture of the Multiscale Computing Patterns software. The dashed-line boxes represent external components (which either exist separately or are under development).

and provides a selection of particularly suitable execution scenarios to the Execution component. The Execution component will then select the optimal execution plan based on chosen specified cost criteria (time to completion, energy consumption), by taking into account additional information only available to the middleware (e.g. estimated queueing time, live information on availability of resources, etc.).

The Measurements and Architecture Modelling components respectively store and calculate submodel performance information as a function of the chosen number of cores/nodes and problem size. The Single Scale Performance Model captures the scalability of submodels with respect to problem sizes and number of processors. The Performance Estimator, in turn, relies among others on the Single Scale Performance Model to obtain, interpolate and/or

Table 1
Example of node types, an input to the Pattern-Driven Planner. Note that RAM per node is in Giga bytes. Processor type **1** Intel(R) Xeon(R) CPU E5-2697 v3 @ 2.60 GHz, **2** Intel(R) Xeon(R) CPU E5-2680 @ 2.70 GHz, **3** Intel(R) Xeon(R) CPU E7-4870 @ 2.40 GHz and **4** Intel(R) Xeon(R) CPU E5-2650 v2 @ 2.60 GHz.

Type name		# of nodes	Processors per node	Cores per node	RAM per node	Processor type
Host	Type					
Eagle	haswell_64	492	2	28	64	**(1)**
	haswell_128	460			128	
	haswell_256	52			256	
Supermuc	Thin	9216	2	16	32	**(2)**
	Fat	205	4	40	256	**(3)**
Stfc	Default	118	2	16	64	**(4)**

calculate performance results for the multiscale model. For example, this could be achieved by interpolating between performance results of adjacent problem sizes in the multiscale model and/or, relying on performance models, to predict the performance using a core count for which no measured values have yet been obtained.

In the Measurements component, the overall cost of the submodel as a function of problem size is measured for 1 to n cores on the first node in a specific node type, and for 2 to N nodes assuming full occupancy on each node for a given number of iterations. In the baseline case, the cost is represented as execution time, but note that these calculations can also be done for other metrics of cost such as energy. The actual measurements can be obtained from tools designed specifically for performance profiling tools, such as Allinea MAP [27], the tool of choice in our research. A template of measurements is shown in Listing 3 in Appendix A; this measurement listing might contain specially marked values (i.e NA), for node types where a specific single scale model is not supported.

The Architecture Modelling software is established to provide predictions for existing machines, but also for non-existing emerging exascale configurations. This allows users to assess how MCPs could optimally benefit from such hypothetical machines, or contribute in co-evolution of such new architectures.

Based on performance results from the Performance Estimator, the Pattern-Driven Planner groups types of nodes into classes depending on the similarity of performance figures, type of computing pattern and cost criteria (e.g efficiency, makespan time, energy usage, resource usage, ...) computed as cost function. Then, using multi-objective optimisation, the tool will generate a small number of alternative execution scenarios. The importance of the alternatives here is to give the Execution component the freedom to choose from a set of resources with comparable performance per submodel depending on the availability of these resources as well as on the variation in queue times. We will enhance this component to extend the patterns with capabilities to also consider issues related to energy awareness and fault-tolerance. All in all, for each pattern we will formulate constrained optimisation problems that as output will deliver alternative execution profiles to the Execution component.

The exact output of the Pattern-Driven Planner to the Execution component will be several allocation plans and other requirements, as described in the next section, to run multiscale application. Here, the output file holds information about the kernels and corresponding helpers, the classes of node types and a set of allocation plan. The allocation plan is a specific mapping of the multiscale model to resources. Listing 4 in Appendix A shows the template of the node classes and allocation plans parts.

3.3. Resource allocation and execution component

The responsibility for the execution component is twofold. First, it needs to select the best allocation plan from the plans provided by the optimisation component. The selection pertains to the mapping of computational kernels to a specific set of physical resources of defined types, taking into account the (sometimes conflicting) requirements of users and resource providers. Second, once an optimal plan has been selected, this component needs to ensure the efficient and reliable execution of the application within the distributed heterogeneous infrastructure.

The execution component is mainly provisioned using the QCG environment[2] [24], a mature middleware system deployed in many HPC centres across Europe. QCG delivers a set of ready to use components that can be installed and managed at each site, irrespective of the internal policies or local queueing systems. To fulfil expectations and objectives of both users and resource providers, QCG features and extendable brokering service which allows for customised brokering algorithms and strategies. In addition, QCG provides support for advance reservation, co-allocation and workflows, enabling the execution management of multi-kernel applications, with both cyclic and acyclic dependencies, on a distributed e-Infrastructure [20,21].

Deploying multiscale simulations on production e-infrastructures gives rise to a number of challenges that are difficult to anticipate prior to the execution component. For example, the user objective for an immediate job start, e.g. through means of advance reservation, needs to be harmonised with the provider's objective for high resource utilisation. In addition, the Pattern Driven Planner provides plans that are likely to be optimal from a user perspective, but have not yet incorporated the constraints imposed by the presence of other workloads in the e-infrastructure environment.

The *QCG Pattern-aware Scheduler* (which is part of QCG Broker) calculates which of the plans provided by the Pattern Driven Planner is optimal with respect to the objectives of all involved stakeholders. In doing so it takes into account the current and historical load on e-infrastructure resources, including both the occupancy of the actual resources and the queue lengths. The Pattern-aware Scheduler can perform this optimisation with respect to required cost criteria, either a single time to completion criterion or a combination of two criteria, total energy expenditure and time to completion. Here, the time to completion is calculated by adding the predicted queueing time (predicted by Queue Time Prediction Service to QCG) and execution time (provided by the optimisation component). In the energy optimisation case, QCG Scheduler firstly selects a set of candidate plans which finish according to the QCG time to completion prediction in the requested period of time and then it selects an optimal plan with the minimal total energy expenditure (calculated and given by the optimisation component). Through its direct integration with the QCG environment, the Pattern-aware Scheduler accounts for the multi-kernel nature of multiscale application and the fact that each kernel may behave differently in the context of performance and energy-usage when executed on different resource types [26,28,29].

The QCG Pattern-aware Scheduler relies on a plugin-like architecture to gather all required information (see dashed boxes in the Execution component of Fig. 2). For example, the scheduler uses the *Queue time metrics plug-in* to get precise knowledge about the expected queue time on available resources. This plug-in is integrated with external resource-level components, in this case the *Queue-time Estimation service*. Similarly, we are planning to implement an Energy metrics plug-in and combine it with the QCG Pattern-aware scheduler.

As the new brokering module uses new types of input parameters to specify the requirements of the MCPs, we have extended the job description interface and revised several internal schemas used to exchange information between the components in QCG-Broker service. We present several key fragments of the this extended

[2] www.qoscosgrid.org.

Table 2
Resources used for the measurements in Sections 4.1 and 4.2.

Resource	CPU architecture	Cores
SuperMuc	Intel(R) Xeon(R) CPU E5–2680	147 456
	Intel(R) Xeon(R) CPU E7–4870	8 200
Eagle	Intel(R) Xeon(R) CPU E5-2697 v3	2 408

description in Appendix F. Here, all jobs described using a *patternTopology* element will be processed using the new scheduling engine.

Based on the result of the QCG Pattern-aware Scheduler, the QCG Job Controller module prepares the execution environment by transferring input data and starting the job submission to one or more distributed resources. The resources in our e-infrastructure are made accessible to QCG Job Controller using services implementing the Basic Execution Service (BES) interface [30].

QCG Job Controller contains a set of specific adaptations to address the requirements for efficiently executing high performance multiscale simulations using high-end e-Infrastructures. Both the QCG-Broker interface and the core capabilities of the service components have been extended to support a range of pattern-based multiscale jobs. Specifically, to allow efficient execution of the Replica Computing Pattern applications, we have incorporated two additional QCG mechanisms: workflows and job arrays. We incorporated a modified version of existing workflow mechanisms [31], eliminating the need to transfer data between subsequent tasks executed on the same resource, and simplifying the execution of workflows in parameter sweep tasks. The job arrays functionality allows a set of independent tasks to be run on a resource and be considered as a single QCG task. In the Replica Computing Pattern these sets of subtasks can be scheduled by the middleware to be executed on various clusters, thereby balancing the overall load on the infrastructure. Job arrays not only help to reduce the management complexity of all tasks executed separately, but increase the overall throughput of the system and decreases the total time to completion of a simulation.

4. Applications of the multiscale computing patterns software

In this section, we present three exemplar applications from different scientific domains (one from Fusion research and two from biomedicine, being cell based blood flow modelling and the Binding Affinity Calculator BAC) to demonstrate the capabilities practical usage of the MCP software, and the benefits in terms of application performance. Our applications are mapped to two different computing patterns, with Fusion and cell based blood flow modelling mapped to the Extreme Scaling (ES) pattern and BAC to the Replica Computing (RC) pattern. Applications for HMC are currently under development. In addition, details of the required steps and various code snippets at each level of the software stack are presented from the perspective of the application developer. All performance figures presented are measured at two supercomputers that participated in the studies, namely SuperMUC [32] (Tier-0 HPC from Leibniz-Rechenzentrum, Germany), and Eagle [33] (Polish national grid clusters from Poznan Supercomputing and Network Center, Poland). Further details are listed in Table 2.

4.1. Extreme scaling

In ES, the ultimate goal is to ensure minimal interference between the primary model and the auxiliaries. It may happen (as in the example of the cell based blood flow simulation) that the auxiliary models induce large waiting times for the primary model, thus potentially wasting resources and reducing resource usage. The Multiscale Computing Patterns software detects this situation automatically, and then interleaves two multiscale simulations, executing both at the same time [23]. This mechanism would increase the resource usage efficiency. For ES, the efficiency of the multiscale model ϵ_M can be calculated as [23]:

$$\epsilon_M = \frac{\epsilon_P}{\frac{T_{aux}(P)}{T_Pr(P)} + 1}, \tag{1}$$

and the resource usage efficiency (R) as:

$$R = \frac{\sum_i T_i P_i}{T \sum_i P_i}. \tag{2}$$

where P_i is the number of cores used for submodel i, T_i is the execution time on submodel i excluding waiting times, T is the total execution time including waiting times, and ϵ_P the efficiency of the primary model.

Fusion application

Nuclear fusion has the potential to produce clean and carbon-free energy, as physicists hope to demonstrate with ITER, which is a tokamak device that uses magnetic fields to confine plasma. However, the grand challenge of long-term plasma confinement requires the understanding of interactions between very small-scale turbulence and large scale plasma behaviour [9,34]. Therefore, having a robust multiscale computing scheme to study this interaction has become a vital goal in the fusion community. Our targeted fusion application simulates the time evolution of a plasma's 1D profiles (for instance electron temperature) in the tokamak core with a transport code, while under the influence of anomalous transport coefficients computed by a 3D turbulence code and periodic 2D equilibrium reconstruction [34]. The transport, turbulence, and equilibrium codes are submodels developed separately and are well-benchmarked. These submodels share a common data interface and are embedded into MUSCLE2 as kernels, which allows for straightforward coupling through a simple and configurable script as described in [9]. Such simulation is essentially multiscale in time, and corresponds to the ES computing pattern depicted in Fig. 1(b). The turbulence code is the primary submodel in this application because it requires the vast majority of the computational power compared to the other submodels.

The starting point in the description component of the software is to compose a task graph in xMML format. This text file (shown in Appendix B) contains the list of submodels involved, time and space scales and input/output data of each submodel, and coupling between submodel pairs through their respective input/output data. If desired, the user can deploy the jMML tool [20] to generate the task graph from the xMML [2] (displayed in Appendix D). Besides visual representations, the jMML tool can turn the content from a task graph into a skeleton configuration for MUSCLE2. The designer of the coupled application can implement submodels as MUSCLE2 kernels and other simulation parameters (either global or specific to a kernel) into the MUSCLE2 configuration file [20]. An example of the fusion application's configuration file, which is written as a ruby script, is displayed in Appendix C. Note that at this stage, the user can directly connect to a cluster where all executables, libraries and input data are present, and write an ad-hoc script to be submitted to the local batch queue system. However, the burden of manually adjusting the configuration and selecting the optimal cluster lies on the user every time wants to run a simulation. The MCP software has features that relieve these burdens from the user by automatically selecting the best configuration for a given performance metric, as described in further detail in the remainder of this subsection.

The task graph is submitted and parsed by the Translation service along with other specifics provided by the developer, such

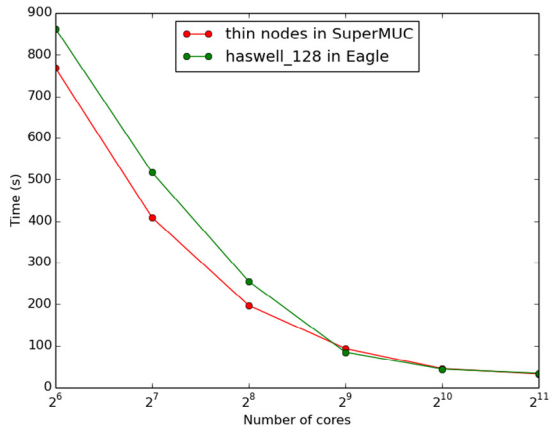


Fig. 3. Runtime on different resources for one iteration of the primary submodel in the fusion application.

Table 3
Performance for ES applications, namely Fusion and cell based blood flow modelling (RBC). T is the execution time (excluding waiting times) for primary (Pr) and auxiliaries (aux) on P_{Pr} and P_{aux} number of cores in seconds, ϵ is the efficiency for the primary (Pr) and the multiscale model (M) and R is the resource usage.

Simulation	Host	P_{Pr}	T_{Pr} (S)	P_{aux}	T_{aux} (S)	ϵ_{Pr}	ϵ_M	R
Fusion	SuperMuc	1024	49017	1	780	1.0618	0.9774	0.91
RBC	Eagle	1036	1936.7	168	1531.6	0.7066	0.3946	0.367
RBC_{alt}	Eagle	1036	3081.34	168	3081.34	0.7066	0.3954	0.746

as additional submodel definitions, details on middleware, simulation parameters, and user information. In the current implementation, the Translation service is a python script which, as a result, creates two template xml files: matrix.xml and multiscale.xml. Matrix.xml contains information related to single scale submodels, such as measurements of their performance. An example of benchmark data on scaling of the primary submodel for two types of nodes is shown in Fig. 3. Multiscale.xml contains information related to the coupled application. These two templates are prefilled with information from the xMML file and can be completed by the application designer. An example is given in Appendix E.

Next, the outputs of the Translation service (matrix.xml and multiscale.xml) are passed on to the Pattern-Driven Planner, which in turn generates an XML job script for the selected middleware (the QCG). Currently, the Pattern-Driven Planner proposes three optimal plans that minimise the cost, and an example of such plans is shown in Appendix F. Currently, these plans are drafted based on the measurements of runs performed manually by the application designer. The next stage will be to enhance the Performance Estimator such that it can benchmark on-the-fly and interpolate on settings for which no performance data is available.

Finally, the job script from the Pattern-Driven Planner is submitted to the QCG. QCG selects one of three plans and starts the simulation on the system(s) involved. For the fusion application, and in general for most ES applications, it is more sensible to select a plan in which all submodels run on a single site, for auxiliaries do not require much resources. In that case, we should only care about serialisation due to serial auxiliary models and how that could impact the execution [23]. Also note that the speedup of the primary model is super-linear because the efficiency was calculated with 64 cores instead of one core, which may lead to latency hiding. For fusion, the time of the primary model spent in waiting for the auxiliaries is not that large, as shown in Table 3, so no additional actions are required, and, therefore, the resource usage is high.

In particular, QCG selects the thin nodes in SuperMUC to run the fusion simulation (see Table 3). A production run with 1000 iterations using 2048 cores was completed successfully using the software scheme described earlier. The entire run was completed in approximately 11.1 h, or 22733 core hours. Among the three submodels, the primary submodel (a turbulence code based on gyrofluid theory) took about 17 s per iteration, while the transport and equilibrium auxiliary submodels took 1 and 3 s, respectively. However, the fusion plasma in this particular example needs approximately 4000 iterations to reach equilibrium state. Therefore, improving efficiency becomes essential as future simulations require more computing time. The current simulation couples the submodels in series. One way to speed-up the simulation is to run auxiliary submodels in parallel when possible, which is theoretically the case for the timescale-less equilibrium submodel. This idea is preliminary and its validation is necessary before such transformation is added as a possible optimisation technique in the Pattern-Driven Planner.

The ultimate goal for the fusion application is to use a more sophisticated turbulence model, namely replacing the gyrofluid model with a gyrokinetic model, to simulate plasma in the core of a tokamak. In addition, the ability to simulate plasma of a much larger volume would be necessary to understand possible instabilities that could destroy plasma confinement in the ITER tokamak. These goals require an extensive amount of computing resources, as well as intelligent and highly optimised coupling approaches. The Multiscale Computing Patterns software have demonstrated initial success with a smaller-scale problem. With further improvements, we envision that these patterns can efficiently handle future exascale calculations involving ITER and gyrokinetic simulations.

Cell based blood flow simulation

In this application we couple continuous blood flow simulations implemented in Palabos [35] (a fully parallelised open source Lattice Boltzmann Model) to cell based blood flow simulations implemented in the Hemocell suspension simulation framework (an Immersed Boundary Lattice Boltzmann Model (IB-LBM)) [36–39]. Specifically, we couple two continuous fluid fields (C_L and C_R, which are serial auxiliary models) to the inlet and outlet of HemoCell, in order to provide the correct in- and outflow conditions to the more expensive suspension simulation (P, the primary model in this application), and to keep the domain for the cell based blood flow simulation as small as possible. This application has also been coupled using MUSCLE2. The performance measurements are shown in Fig. 4. As is clear, in this case the auxiliary models (C_L and C_R) require a small amount of computing and only execute on a small core count. However, the primary model, HemoCell, is compute hungry, but at the same time scales very well to a much larger core count, even so that the execution time of the primary becomes comparable to the execution time of the auxiliary submodels. This situation was analysed in [23] and calls for a more advanced scheduling of the pattern, basically interleaving two instantiations in order to make best use of the available computing resources. The MCP software is able to orchestrate such more advanced scheduling of multiscale applications.

Table 3 shows that the resource usage for running this application is 0.35. This is due to the large waiting times of both primary and serial auxiliaries in the naive scheduling, which means wasting 1122 cores hours (1.08 h per core) for primary and 1755 (12.5 h per core) for auxiliary models by doing nothing but waiting. To solve this, we interleave two different instantiations with each other, as proposed in [23]. This mechanism was coordinated using wait/notify semantics [10]. By doing so, we doubled the resource usage efficiency by reducing the wasted cores to 887 and 152 core hours for primary and auxiliaries models respectively. By implementing more advanced load balancing algorithms and selecting the right number of cores for the primary and auxiliary models, we can increase the resource usage efficiency even more. We are currently realising such more advanced features of the MCP software.

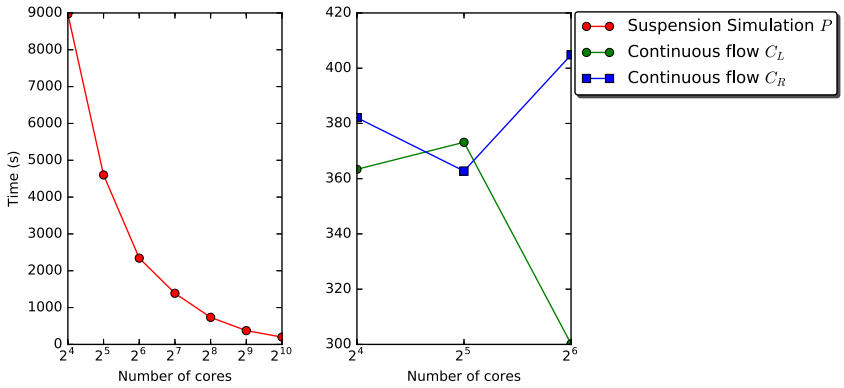


Fig. 4. Total runtime of cell based blood flow modelling submodels on Eagle [33] *haswell*_128 nodes. Note the difference in scale between the primary suspension model (left) and the auxiliaries continuous flow models (right).

4.2. Replica computing (binding affinity calculator)

The procedure for replica computing is similar to that for Extreme Scaling (Section 4.1). The starting point for all RC pattern applications is the task graph, via an xMML textual description. A "multiplicity" tag in the "instance" node of the xMML description indicates that multiple instances (replicas) are required for that submodel. The Translation service detects the presence of this tag, identifies that the RC pattern is required and that the associated cost function in the Multiscale Computing Patterns software should be invoked.

The Translation service uses submodel definitions in separate files. To illustrate this, we describe the process for the Binding Affinity Calculator (BAC) [40], an automated molecular simulation based free energy calculation workflow tool, which we use to calculate ligand-protein binding affinities. Rapid and accurate calculation of binding free energies is of major concern in drug discovery and personalised medicine. The underlying computational method is based on classical molecular dynamics (MD). These MD simulations are coupled to the molecular mechanics Poisson–Boltzmann surface area (MMPBSA) method to calculate the binding free energies. For purposes of reliability, ensembles of replica MD calculations are performed for each method, and we have found that about 25 of these are required per MD simulation in order to guarantee reproducibility of predictions. This is due to the intrinsic sensitivity of MD to the initial conditions, since the dynamics is chaotic. Therefore, BAC is an ideal example of the replica computing pattern. BAC consists of a workflow where, within each replica, the output from one submodel (NAMD) is used as input to the next submodel (AmberTools). For more information, we refer to [40,41].

BAC previously used the FabSim [25] tool extensively to perform simulation runs and, therefore, we have added an option to the Translation service to allow the matrix.xml and multiscale.xml files to be completed (as much as possible) through reading of FabSim configuration files. This demonstrates the potential versatility of our MCP approach, which should enable relatively straightforward integration with existing multiscale execution environment as, in this case, FabSim. For example, it uses the machines.yml configuration file from FabSim, which lists the configuration settings of submodels on remote resources (e.g., location of libraries and required execution flags). Additional information specific to for the Translation service (and not required by FabSim) can also be added to this file, including restrictions on the submodel (GPU/CPU compatibility, max/min number of cores, etc.). This allows submodel information to be reused if it is required for different multiscale applications. Then, FabSim compatible YaML file (shown in Appendix G) are used to assist the completion of matrix.xml and multiscale.xml. BAC currently does not use a coupling library (such as MUSCLE), so no additional files are required. However, in the future we foresee hybrid MCPs, where each replica could for instance be a full-blown ES by itself, and then such additional information would be needed.

Following the procedure outlined for the ES pattern, the user passes matrix.xml and multiscale.xml to the Optimisation component. Unlike the ES pattern, the user does not need to specify the required number of cores for the overall simulation. This is decided by the Performance Estimator by calculating the cost function.

Finding a cost function for RC that will generate resource allocation plans is different to that for ES. First, there is an obvious trade-off between the number of replicas that must be executed, the minimum number of cores that one single replica needs, and the total number of cores available for the overall job. The performance data for RC uses the minimum time per replica for different node types in different hosts, as shown in Fig. 5 for a single BAC replica. This data is collected in the Single Scale Performance Model. In the simplest cost-function, where we consider only time to solution, all replicas would be run concurrently on the node with the shortest running time per replica. However, there are several constraints that the Performance Estimator must also consider such as queue constraints (number of concurrent jobs, time limitations node availability and queueing time).

Most supercomputers have a limit on the number of jobs that can be run or queued at any moment in time per user. For example, on SuperMUC machine, the maximum number of jobs that can be run concurrently on the thin nodes in the "general" queue is 8, while there are no restrictions on the Eagle machine.

As an example, if we have two RC applications which require 40 and 80 replicas respectively, the Pattern-Driven Planner needs to calculate which is faster: running all replicas at one supercomputer SuperMUC (while taking into account the constraint of concurrently running 8 jobs per user) or distributing the jobs among different hosts, for example, across SuperMUC and Eagle, using the functionality in QCG to run across multiple resources. To illustrate how this could be coordinated, let us take the hypothetical situation that there is also a 12 job limit on concurrent jobs running on Eagle to mimic the workload. In Fig. 6, we show the time to completion as a function of the number of "batches" running on SuperMUC, where a "batch" is defined as a set of 8 concurrent running jobs on SuperMUC. The remainder of the replicas are run on Eagle (again in "batches" of up to 12 jobs).

Fig. 6 shows we estimate that for 40 replicas, the shortest time to completion is for 2 "batches" to be run on SuperMUC, while for 80 replicas, the minimum time to completion is for 4 "batches" to be run on SuperMUC. This assumes that all replicas take the same time (the shortest time-to-completion from our benchmarking), that all the replicas are independent (no communication) and that each "batch" runs directly after the other. It is clear there is a

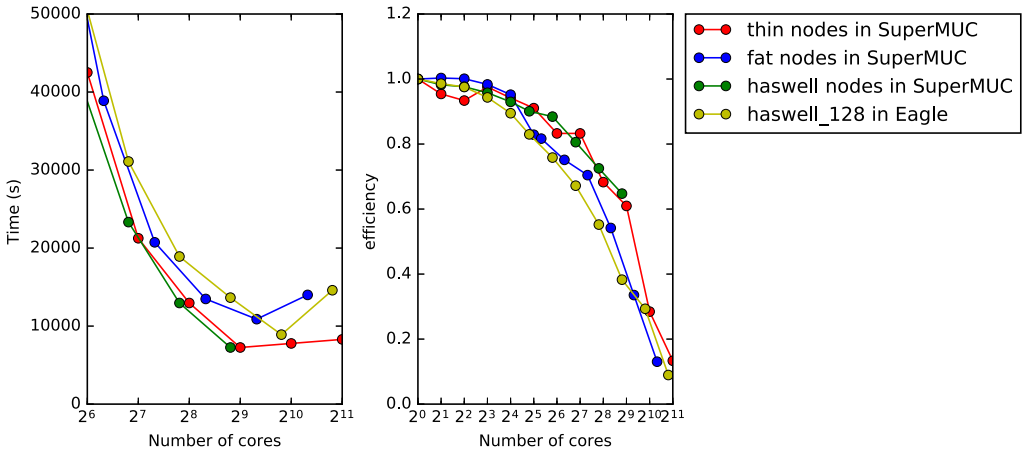


Fig. 5. Time and efficiency per replica (NAMD kernel) on different number of cores on different node types.

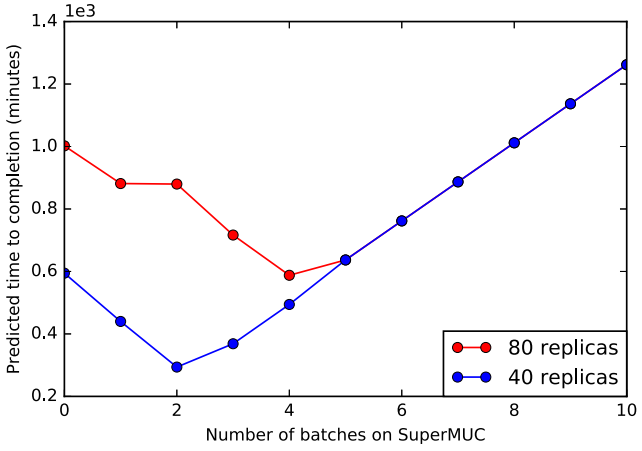


Fig. 6. Theoretical time of running multi-replicas simulations across 2 resources (SuperMUC and Eagle), as a function of number of "batches" (i.e. sets of 8 concurrent jobs) on SuperMUC. The remainder of the replicas are run as batches of up to 12 concurrent jobs on eagle.

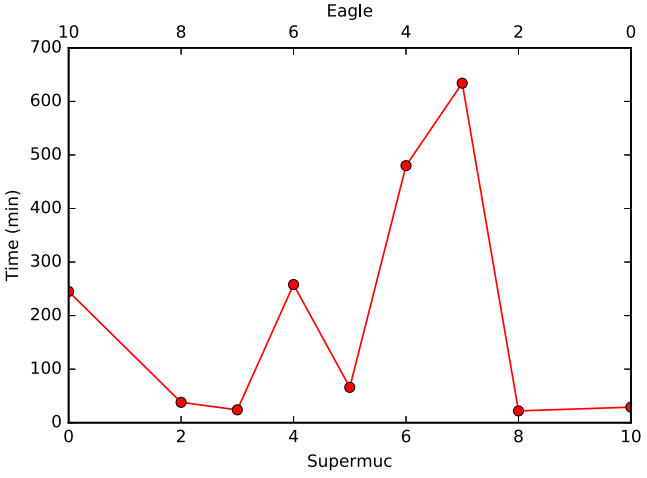


Fig. 7. Time to completion of multi-replicas simulations across 2 resources (SuperMUC and Eagle), as a function of number of replicas, ten in total, distributed on SuperMUC and Eagle.

limitation to this model; it will only be realistic if the time spent in the queue is very short. Otherwise, the time to completion could be very different to that predicted in Fig. 6, and we could envisage the most efficient split in replicas across resources being completely changed if the queueing times are very different across the resources. The estimation of queueing time will be investigated and to incorporated into the middleware in the future (as described in Section 3.3).

The output file description to the execution component is unified among all computing patterns as described in Section 3.3. QCG also have the ability to distribute replicas to the intended machines and gather the results in one place. Fig. 7 shows timings of test BAC runs. In these studies, we run 10 replicas across two supercomputers, SuperMUC and Eagle. By running 8 replicas on SuperMUC and the rest on Eagle we reach the least time-to-completion.

To quantify this speedup [42], we would compare the best timing of distributing replicas T_{distr} with the best timing of running them on SuperMUC (with batch time) T_{local} . The speedup is calculated as:

$$S = \frac{T_{local}}{T_{distr}},$$

and the speedup is 1.2 for our set of studies. This means that at the moment of running this set of replicas, we would gain a speedup due to the varied queuing time. The queueing time will be predicted and hosts will be automatically selected by QCG based on the knowledge of run time and queueing time as stated before. Table 4 summarises the results from the BAC application for time-to-completion runs in Fig. 7.

Table 4
Performance model for RC application, namely BAC. N is the total number of replicas, P_R is the number of cores per replica, T_R is the Time per replica in seconds, T_{local} and T_{distr} are the shortest total simulation time (including queueing times) for a local and a distributed runs and S is the Speedup.

Simulation	N	P_R	T_R (S)	T_{local} (S)	T_{distr} (S)	S
BAC	10	10	20	29	24	1.208

5. Discussion and conclusions

We have introduced and described the Multiscale Computing Patterns software, which extends the Multiscale Modelling and Simulation Framework to enable high performance multiscale computing based on three generic patterns. We demonstrated its usage and added-value for three different types of multiscale applications: fusion and cell based blood flow simulation, both as examples of Extreme Scaling, and binding affinity calculation as an example for Replica Computing. In addition, these multi-scale models are based on different coupling approaches, including

MUSCLE2, as well as coupling using scripts and the FabSim automation toolkit.

We implemented and demonstrated the Extreme Scaling and Replica Computing computing patterns. The third computing pattern, Heterogeneous Multiscale Computing, will be implemented, discussed and demonstrated in future work. In the current implementation, each of the demonstrated applications highlights specific strengths of our software approach. For the fusion application, the software abstracts the complication of HPC and chooses the most appropriate number of cores to obtain the required cost criteria (i.e. time to completion). For blood flow, our approach enabled the use of double the resources otherwise accessible. Lastly, for binding affinity calculations, our approach serves to abstract away the choice as to whether the replicas should all run on one and the same computer or be distributed across multiple computers. This automated scheduling approach, which recommends execution across two resources, delivers a time-to-completion speedup of 1.2 compared to the scheduling of all replicas on a single resource.

The Multiscale Computing Patterns software maintains a separation of concerns in three areas. The top layer, the Description component, represents the logical description of the multiscale model. This is the part that is most 'visible' to the application users and developers. The Optimisation component is focused on performance aspects, and provides a number of optimisation criterion based on the type of the multiscale computing pattern and the required criteria. Finally, the Execution component integrates a range of functionalities from the underlying e-infrastructure, and uses the information from the Description and Optimisation components to create and run execution scenarios, each optimised either for minimal time to completion, or minimal energy consumption (given a fixed time to completion requirement). This modular implementation helps multiscale model developers to concentrate on optimising the single scale models of which the application is comprised, without needing to go into details about the HPC machines. The developer can choose the optimisation criteria required.

In this work, we have assembled a range of powerful functionalities for optimising and deploying multiscale applications on large scale HPC infrastructures operating at the multi-petascale, and presented an application-agnostic approach which reduces the development effort required for these purposes. We plan to release the software described here shortly. Generic approaches to High Performance Multiscale Computing are highly sought after across scientific disciplines, and indeed we have already begun propagating the first elements of our approach to other application domains such as astrophysics and materials modelling.

Acknowledgements

We acknowledge partial funding from the European Union Horizon 2020 research and innovation programme under grant agreement No 671564 for the ComPat project (http://www.compat-project.eu/). SA acknowledges funding from King Abdulaziz City for Science and Technology (KACST), Saudi Arabia. AGH acknowledges partial financial support from the Russian Scientific Foundation via grant #14-11-00826. P.V.C. thanks the MRC Medical Bioinformatics project (MR/L016311/1), the EU H2020 CompBioMed grant (http://www.compbiomed.eu, Grant No. 675451) and funding from the UCL Provost. This research was also supported in part by the PLGrid Infrastructure including dedicated HPC resources at the Poznan Supercomputing and Networking Center.

Appendix A. Supplementary data

Supplementary material related to this article can be found online at https://doi.org/10.1016/j.future.2018.08.045.

References

[1] A.G. Hoekstra, B. Chopard, P.V. Coveney, Multiscale modelling and simulation: a position paper, Philos. Trans. Ser. A Math. Phys. Eng. Sci. 372 (2014) 20130377.
[2] A.G. Hoekstra, E. Lorenz, J.-L. Falcone, B. Chopard, Toward a complex automata formalism for multiscale modeling, Int. J. Multiscale Comput. Eng. 5 (6) (2007) 491–502.
[3] B. Chopard, J. Borgdorff, A.G. Hoekstra, A framework for multi-scale modelling, Phil. Trans. R. Soc. A 372 (2014) 20130378.
[4] J. Borgdorff, J.L. Falcone, E. Lorenz, C. Bona-Casas, B. Chopard, A.G. Hoekstra, Foundations of distributed multiscale computing: Formalization, specification, and analysis, J. Parallel Distrib. Comput. 73 (4) (2013) 465–483.
[5] D. Groen, S.J. Zasada, P.V. Coveney, Survey of multiscale and multiphysics applications and communities, Comput. Sci. Eng. 16 (2) (2014) 34–43.
[6] H. Tahir, C. Bona-Casas, A.J. Narracott, J. Iqbal, J. Gunn, P. Lawford, A.G. Hoekstra, Endothelial repair process and its relevance to longitudinal neointimal tissue patterns: comparing histology with in silico modelling, J. R. Soc. Interface / R. Soc. 11 (94) (2014) 20140022.
[7] D. Groen, J. Borgdorff, C. Bona-Casas, J. Hetherington, R.W. Nash, S.J. Zasada, I. Saverchenko, M. Mamonski, K. Kurowski, M.O. Bernabeu, a. G. Hoekstra, P.V. Coveney, Flexible composition and execution of high performance, high fidelity multiscale biomedical simulations, Interface Focus 3 (2) (2013) 20120087.
[8] A.G. Hoekstra, S. Alowayyed, E. Lorenz, N. Melnikova, L. Mountrakis, B. van Rooij, A. Svitenkov, G. Závodszky, P. Zun, Towards the virtual artery: a multiscale model for vascular physiology at the physics-chemistry-biology interface, Phil. Trans. R. Soc. A 374 (2080) (2016) 20160146.
[9] O. Hoenen, L. Fazendeiro, B.D. Scott, J. Borgdorff, A.G. Hoekstra, P. Strand, D.P. Coster, Designing and running turbulence transport simulations using a distributed multiscale computing approach, in: EPS 2013, Europhysics Conference Abstracts, Vol. 37D, no. 37, 2013, pp. P4.155.
[10] 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, Performance of distributed multiscale simulations, Phil. Trans. R. Soc. A 372 (2014) 20130407.
[11] J.L. Suter, D. Groen, P.V. Coveney, Chemically specific multiscale modeling of clay-polymer nanocomposites reveals intercalation dynamics, tactoid self-assembly and emergent materials properties, Adv. Mater. 27 (6) (2015) 966–984.
[12] D.J. Hill, Nuclear energy for the future, Nature Mater. 7 (9) (2008) 680.
[13] R. Delgado-Buscalioni, P.V. Coveney, Continuum-particle hybrid coupling for mass, momentum, and energy transfers in unsteady fluid flow, Phys. Rev. E 67 (4) (2003) 46704.
[14] S.P. Zwart, S. McMillan, A. van Elteren, I. Pelupessy, N. de Vries, Multi-physics simulations using a hierarchical interchangeable software interface, Comput. Phys. Comm. 184 (3) (2013) 456–468.
[15] S. Valcke, The OASIS3 coupler: a European climate modelling community software, Geosci. Model Dev. 6 (2) (2013) 373.
[16] D. Gaston, C. Newman, G. Hansen, D. Lebrun-Grandie, MOOSE: A parallel computational framework for coupled systems of nonlinear equations, Nucl. Eng. Des. 239 (10) (2009) 1768–1778.
[17] P.M.A. Sloot, A.G. Hoekstra, Multi-scale modelling in computational biomedicine, Brief. Bioinform. 11 (1) (2009) 142–152.
[18] A.G. Hoekstra, E. Lorenz, J.-L. Falcone, B. Chopard, Towards a complex automata framework for multi-scale modeling: Formalism and the scale separation map, in: Y. Shi, G.D. van Albada, J. Dongarra, P.M.A. Sloot (Eds.), Computational Science–ICCS 2007, Springer Berlin Heidelberg, Berlin, Heidelberg, 2007, pp. 922–930.
[19] A.G. Hoekstra, A. Caiazzo, E. Lorenz, J.-L. Falcone, B. Chopard, Complex automata: multi-scale modeling with coupled cellular automata, Simul. Complex Syst. Cell. Autom. (2010) 29–57.
[20] J. Borgdorff, M. Mamonski, B. Bosak, K. Kurowski, M. Ben Belgacem, B. Chopard, D. Groen, P.V. Coveney, A.G. Hoekstra, Distributed multiscale computing with MUSCLE 2, the multiscale coupling library and environment, J. Comput. Sci. 5 (5) (2014) 719–731.
[21] T. Piontek, B. Bosak, M. Ciznicki, P. Grabowski, P. Kopta, M. Kulczewski, D. Szejnfeld, K. Kurowski, Development of science gateways using QCG-lessons learned from the deployment on large scale distributed and HPC infrastructures, J. Grid Comput. 14 (4) (2016) 559–573.
[22] J. Knap, C.E. Spear, O. Borodin, K.W. Leiter, Advancing a distributed multi-scale computing framework for large-scale high-throughput discovery in materials science, Nanotechnology 26 (43) (2015) 434004.
[23] S. Alowayyed, D. Groen, P.V. Coveney, A.G. Hoekstra, Multiscale computing in the exascale era, J. Comput. Sci. 22 (2017) 15–25.
[24] B. Bosak, P. Kopta, K. Kurowski, T. Piontek, M. Mamoński, New QosCosGrid middleware capabilities and its integration with European e-infrastructure, in: eScience on Distributed Computing Infrastructure, Springer, 2014, pp. 34–53.

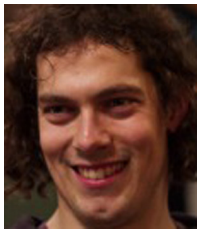
[25] D. Groen, A.P. Bhati, J. Suter, J. Hetherington, S.J. Zasada, P.V. Coveney, FabSim: facilitating computational research through automation on large-scale and distributed e-infrastructures, Comput. Phys. Comm. 207 (2016) 375–385.
[26] K. Kurowski, J. Nabrzyski, A. Oleksiak, J. Weglarz, Multicriteria Aspects of Grid Resource Management, in: International series in Operations Research and Managment Science, vol. 64, 2003, pp. 271–294.
[27] C. January, J. Byrd, X. Oró, M. O'Connor, Allinea MAP: Adding energy and OpenMP profiling without increasing overhead, in: Tools for High Performance Computing 2014, Springer, 2015, pp. 25–35.
[28] K. Kurowski, J. Nabrzyski, A. Oleksiak, J. Weglarz, A multicriteria approach to two-level hierarchy scheduling in grids, J. Sched. 11 (5) (2008) 371–379.
[29] K. Kurowski, A. Oleksiak, W. Piątek, T. Piontek, A. Przybyszewski, J. Węglarz, DCworms-A tool for simulation of energy efficiency in distributed computing infrastructures, Simul. Model. Pract. Theory 39 (2013) 135–151.
[30] I. Foster, A. Grimshaw, P. Lane, W. Lee, M. Morgan, S. Newhouse, S. Pickles, D. Pulsipher, C. Smith, M. Theimer, OGSA Basic Execution Service version 1.0, 2007.
[31] B. Bosak, J. Komasa, P. Kopta, K. Kurowski, M. Mamoński, T. Piontek, New capabilities in QosCosGrid middleware for advanced job management, advance reservation and co-allocation of computing resources-quantum chemistry application use case, in: Building a National Distributed e-Infrastructure-PL-Grid, Springer, 2012, pp. 40–55.
[32] Leibniz-Rechenzentrum, SuperMUC Petascale System. URL https://www.lrz.de/services/compute/supermuc/.
[33] Poznań-Supercomputing-Networking-Center, Eagle. URL https://wiki.man.poznan.pl/hpc/index.php/Eagle.
[34] G.L. Falchetto, D. Coster, R. Coelho, B.D. Scott, L. Figini, D. Kalupin, E. Nardon, S. Nowak, L.L. Alves, J.F. Artaud, Corrigendum: The European integrated tokamak modelling (ITM) effort: achievements and first physics results (2014 Nucl. Fusion 54 043018), Nucl. Fusion 54 (9) (2014) 99501.
[35] FlowKit-Ltd, Palabos. URL www.palabos.org.
[36] L. Mountrakis, E. Lorenz, O. Malaspinas, S. Alowayyed, B. Chopard, A.G. Hoekstra, Parallel performance of an IB-LBM suspension simulation framework, in: International Conference on Computational Science, Elsevier, Reykjavík, Iceland, 2015, p. 10.
[37] G. Závodszky, B. van Rooij, V. Azizi, A.G. Hoekstra, Cellular level in-silico modeling of blood rheology with an improved material model for red blood cells, Front. Phys. 8 (2017) 563.
[38] HEMOCELL A high-performance framework for dense cellular suspension flows. URL https://www.hemocell.eu/.
[39] G. Zavodszky, B. van Rooij, V. Azizi, S. Alowayyed, A.G. Hoekstra, Hemocell: a high-performance microscopic cellular library, Procedia Comput. Sci. 108 (2017) 159–165.
[40] S.K. Sadiq, D. Wright, S.J. Watson, S.J. Zasada, I. Stoica, P.V. Coveney, Automated molecular simulation based binding affinity calculator for ligand-bound HIV-1 proteases, 2008.
[41] A.P. Bhati, S. Wan, D.W. Wright, P.V. Coveney, Rapid, accurate, precise, and reliable relative free energy prediction using ensemble based thermodynamic integration, J. Chem. Theory Comput. 13 (1) (2017) 210–222.
[42] A.G. Hoekstra, P.M.A. Sloot, Introducing Grid speedup Γ : A scalability metric for parallel applications on the grid, in: P.M.A. Sloot, A.G. Hoekstra, T. Priol, A. Reinefeld, M. Bubak (Eds.), Advances in Grid Computing - EGC 2005, Springer Berlin Heidelberg, Berlin, Heidelberg, 2005, pp. 245–254.

Saad A. Alowayyed is a Ph.D. candidate in Computational Science Lab (CSL) at the University of Amsterdam. His thesis is concerned with Multiscale computing patterns. He received his Master in high performance computing from the University of Edinburgh. Saad also works as a researcher in King Abdulaziz for Science Technology (KACST).

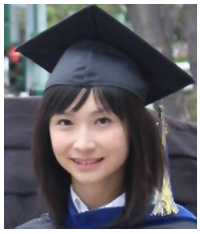
Tomasz Piontek graduated from Poznań University of Technology in Computer Science. He is a member of Applications Department at Poznań Supercomputing and Networking Center, Poland and head of the Large Scale Applications and Services Department. He has been involved in many EU-funded R&D projects in the areas of distributed and parallel computing, including QosCosGrid and MAPPER. His research activities are focused on the modelling of advanced applications for modern hybrid HPC architectures, and online services scalability and availability.

James L. Suter received his M.Chem. from the University of Oxford and Ph.D. from the University of Cambridge under the supervision of Professor Michiel Sprik. His research addresses multiscale materials modeling and analysis of clay and graphene nanocomposites using high-performance computing.

Olivier Hoenen is a Post Doctoral Research Associate at the Max-Planck-Institut fuer Plasmaphysik (IPP) in the Numerical Methods for Plasma Physics Division, specialized in parallel computing and adaptive methods. He was involved in the Infrastructure Project for the ITM-TF, and in the EU FP7 project EUFORIA. He participated in the MAPPER project where he was interested in distributed multiscale simulations for plasma physics. He is involved in the EUROfusion Infrastructure and Support Activities where he maintains the integrated modelling and simulation platform and participates to further developments of the ITER platform IMAS.



Derek Groen is a Lecturer in Simulation and Modelling at Brunel University London, and a visiting Lecturer at University College London. He specializes in multiscale simulation, high performance computing and automation, and has published 45 peer-reviewed papers. He has created multiscale models and performed validation studies using the HemeLB bloodflow simulation environment, and won the ARCHER Early Career Impact Award in 2015 as a result. He obtained his PhD from the University of Amsterdam in 2010, where he ran large cosmological simulations geographically distributed across up to four supercomputers.

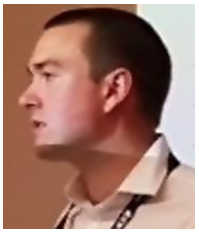


Onnie Luk is a Ph.D. graduate from the University of California at Irvine, where she conducted her research on the role of convective cell in nonlinear interaction of kinetic Alfven waves. She also has experience in analyzing magnetometer data obtained by the Galileo spacecraft. Currently, she is a post doctoral researcher at the Max-Planck-Institut fÃijr Plasmaphysik. She is part of the ComPat project, in which she explores time bridging methods to connect turbulence and transport models in a component-based multiscale fusion plasma simulation.

Bartosz Bosak received his master's degree in computer science in 2007 from Poznań University of Technology in Poland (Laboaratory of IT Systems in Management). Since 2006 he has been working at the Application Department of Poznan Supercomputing and Networking Center as a systems analyst and developer. His research interests include widely undrestood support for large scale computations on Grid, Cloud and HPC infrastructures, multiscale computing as well as system integration. He was a participant of a variety of European and national projects including BREIN, MAPPER, ComPat, Geant and PLGrid.

Piotr Kopta received his M.Sc. degree in Computer Science from the Technical University of Częstochowa in 2002. Currently he is an systems analyst at the Poznań Supercomputing and Networking Center. His research interests concern high performance computing in particulary new computational architectures.

Krzysztof Kurowski holds a Ph.D. degree in Computer Science. He graduated from Poznan University of Technology. He has been leading Applications Department at Poznan Supercomputing and Networking Center in Poland since 2008. He has been actively involved in many international R&D projects in the area of computer science, HPC, and ICT, including GridLab and ComPat. He was a research visitor at University of Queensland, Argonne National Lab, and at CCT Louisiana University. His research activities are focused on advanced applications, computing simulations, scheduling and resource management in networked environments.



Oliver Perks is a Field Application Engineer at Arm, providing application porting and optimisation support to customers, through the use of the Arm HPC tools. Oliver obtained his Ph.D. from Warwick University, in profiling of HPC applications, and subsequently moved into industry to practice performance optimisation for large scale production workloads. Having joined Allinea, Oliver began working on a number of H2020 projects, including ComPat, and continues that involvement as a representative of Arm.

Keeran Brabazon has a Ph.D. in Scientific Computation from the University of Leeds. In his current role at Arm he specialises in the performance analysis of high-performance computing (HPC) simulations, with a focus on the identification of specific application performance bottlenecks through targeted sparse data collection. Keeran works in the team developing the Arm Forge and Performance Reports HPC tools.

Vytautas Jančauskas got his Ph.D. from Vilnius University in 2016. The subject of the thesis was evaluating the performance of multi-objective optimisation methods. He worked as a lecturer and assistant lecturer for more than 3 years. Supervised undergraduate computer networks course at the Faculty of Mathematics and Informatics at Vilnius University. Vytautas contributed extensively to various open-source projects, via the Google's Summer of Code program. He has joined the ComPat project at LRZ in 2016.

David Coster is a senior researcher at the Max-Planck-Institut für Plasmaphysik, where he leads the Edge Physics group in the Division of Tokamak Physics, one of the two theory divisions at IPP Garching. He was the Project Leader for Integrated Modelling Project 3 (Core and Edge Transport) within the EFDA ITM Task Force. As well as being a Deputy Task Force Leader, Dr Coster was the deputy coordinator of the seventh European Framework Programme (FP7) project EUFORIA and was also involved in the FP7 project MAPPER. His own research has concentrated on understanding the behaviour of the edge and scrape-off layer regions of the tokamak plasma. He is also a member of the EUROfusion IT committee and deputy coordinator of the sub-committee developing an approach to Open Data within EUROfusion.

Peter Coveney holds a Chair in Physical Chemistry, is Director of the Centre for Computational Science, and is an Honorary Professor in Computer Science at UCL. He is also Professor Adjunct at Yale University, he is active in a broad area of interdisciplinary theoretical research including condensed matter physics and chemistry, materials science, life and medical sciences. He has published over 400 scientific papers, edited 16 journal "theme issues", and authored three books, including two best-selling popular science books.

Alfons Hoekstra holds a Ph.D. in Computational Science from the University of Amsterdam and currently is a professor in Computational Science at the University of Amsterdam and the national research university ITMO, St Petersburg, Russia. His research focuses on multi-scale multi-science modelling, large-scale simulations, and high performance computing, mainly in the biomedical domain and complex systems science. He has a long-standing expertise in Computational Biomedicine, Complex Systems simulations, and high performance parallel and distributed computing. He has published over 250 research papers. He currently leads the Computational Science Lab at the University of Amsterdam.