Multiscale Modeling: recent progress and open questions

Chopard, B.; Falcone, J.-L.; Kunzli, P.; Veen, L.; Hoekstra, A.G.

DOI
10.1007/s41939-017-0006-4
Publication date
2018
Document Version
Final published version
Published in
Multiscale and Multidisciplinary Modeling, Experiments and Design
License
Article 25fa Dutch Copyright Act (https://www.openaccess.nl/en/policies/open-access-in-dutch-copyright-law-taverne-amendment)
Link to publication

Citation for published version (APA):
Chopard, B., Falcone, J.-L., Kunzli, P., Veen, L., & Hoekstra, A. G. (2018). Multiscale Modeling: recent progress and open questions. *Multiscale and Multidisciplinary Modeling, Experiments and Design*, *1*(1), 57–68. https://doi.org/10.1007/s41939-017-0006-4

Multiscale and Multidisciplinary Modeling, Experiments and Design (2018) 1:57–68
https://doi.org/10.1007/s41939-017-0006-4

ORIGINAL PAPER

Multiscale modeling: recent progress and open questions

Bastien Chopard[1] · Jean-Luc Falcone[1] · Pierre Kunzli[1] · Lourens Veen[2] · Alfons Hoekstra[3,4]

Received: 13 December 2017 / Accepted: 23 December 2017 / Published online: 26 January 2018

Abstract
Many important scientific problems are inherently multi scale. This is, for instance, the case in models in material science or environmental science. A big challenge is to formulate generic frameworks for multiscale modeling and simulation. Despite its importance, the scientific community still lacks a well-accepted generic methodology to address multiscale computating. We review a recent theoretical framework which aims at filling this gap. We also present new results and extension in relation with scale bridging methods and execution multiscale simulation on HPC systems, and discuss open questions related to this topic.

Keywords Mutiscale modeling · High Perfomance multiscale computing · Theoretical framework · Coupling middleware (MUSCLE) · Scale bridging techniques

1 Introduction

Problems in science and engineering often contain several interacting phenomena, and involve different temporal and spatial scales. As an example among many, we can mention biomedical applications (Evans et al. 2008; Tahir et al. 2011, 2013; Groen et al. 2013; Hoekstra et al. 2016), in which slow biological processes are coupled to fast fluid mechanics. Multiscale and multiscience problems are often very hard to address even numerically, due to the difficulty to include a very large range of scales in the same solver. An explicit processing of all scales may be beyond the power of even the fastest supercomputers.

Although one can find in the literature a lot of references pertaining to “multiscale modeling”, only a few methodological papers (Ingram et al. 2004; Dada and Mendes 2011; Yang and Marquardt 2009; Weinan et al. 2007) propose a conceptual framework, based on solid and general theoretical grounds. As a matter of fact, for many multiscale applications, methodology is entangled with the specificity of the problem, preventing generalization to other problems. Furthermore, similar multiscale strategies are often proposed under different names. We refer the reader to Hoekstra et al. (2014) for a more detailed discussion of the status of multiscale modeling in several scientific communities.

Solving interdisciplinary multiscale and multiscience problems involve more and more frequently scientists with different background and possibly working in different spatial locations. Therefore, a well-established methodology is essential to build and maintain a computer code solving the problem. Proposing a framework including theoretical concepts, a multiscale modeling language and an execution environment will allow scientists to separate the problem-specific components from the strategy to bridge the scales and the processes.

There exist many computer solvers addressing given physical processes at chosen scales. Once coupled according to a general strategy, these solvers can be the components of a complex multiscale, multiscience applications, whose architecture is described at a high level of abstraction, allowing incremental development and long-term sustainability.

A step towards the solution to the above identified requirements can be found in the so-called multiscale modeling and simulation framework (MMSF) that some of us have been developing over the last few years (Chopard et al. 2014; Borgdorf et al. 2013b). This framework encourage multiscale modelers to express problems composed of a wide range of scales as a set of separate “single scale” models, and to dis-

✉ Bastien Chopard
Bastien.Chopard@unige.ch

1 Computer Science Department, University of Geneva, Geneva, Switzerland

2 Netherlands eScience Center, Amsterdam, The Netherlands

3 Computational Science Lab, University of Amsterdam, Amsterdam, The Netherlands

4 ITMO University, Saint-Petersbourg, Russia

tinguish these single scale models from the "scale bridging" methods that need to be developed to couple them. Over the past years, MMSF has been successfully applied and evaluated on several different applications (Borgdorff et al. 2014).

In the present paper, we will review the MMSF framework and introduce new developments and examples. In particular, we will present the muscleHPC programming library that significantly enhance the performance of MMSF applications. We will discuss and evaluate a scale bridging technique coined *amplification* in Hoekstra et al. (2010), which is well adapted to combine slow biophysical phenomena with fast haemodynamic processes. We also discuss a way to help users partition and schedule MMSF applications, using a discrete event simulator. Last but not least, we introduce the new concept of "multiscale computational patterns", reflecting complex multiscale couplings that are recurrently present in many applications.

2 The MMSF framework

The multiscale modeling and simulation framework (MMSF) is a theoretical and operational approach to describe and simulate multiscale, multiscience phenomena. By adhering to a single framework, not tied to a specific discipline, groups of researchers ensure that their respective contribution may seamlessly integrate with others. MMSF is described in detail in Borgdorf et al. (2013b) and references therein. Here, we review the main ideas of the formalism.

Figure 1 presents the different steps of MMSF. First, the relevant scales and the relevant processes to be modeled need to be identified. Each process (for instance, fluid flow) at the desired scales is referred to as submodels, typically implemented in a monolithic computer program.

In MMSF, we propose to interpret the submodels at an abstract level with a generic time loop structure, as described in Fig. 2. We claim that even though the implementation of a submodel may not follow exactly this construction, the operations that we identified are present in many scientific solvers. They correspond to the initialization of the variables, followed by a time loop. Within this time loop, there is an operation S for *solver* which is repeated to advance the simulation to the next time point. To do so, boundary conditions need to be specified at the limits of the domain. This is the role of B to provide such data. In a multiscale system, the boundaries are often provided by another submodel. For other problems, it may also be S that need information computed by another submodel. Finally, we formally define operation O_i (for *intermediate observation* and O_f (for *final observation*). They are placed in the execution loop where the physical quantities of interest have been computed at the corresponding time.

In a second step, one has to indicate how the submodels are coupled, by specifying when and what data must be exchanged. The transformation of data from one submodel to the other is often refereed to as scale bridging techniques. In MMSF, coupling is implemented as components called *filter*

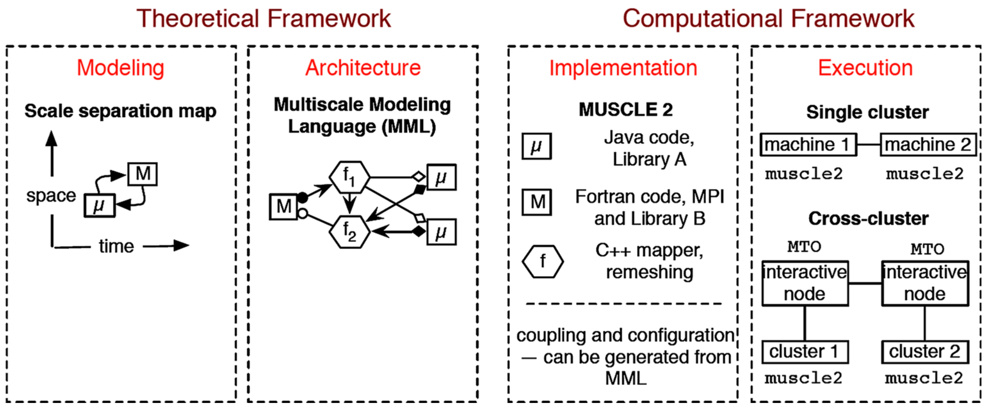

Fig. 1 Pipeline of actions needed to develop and run a multiscale application within MMSF

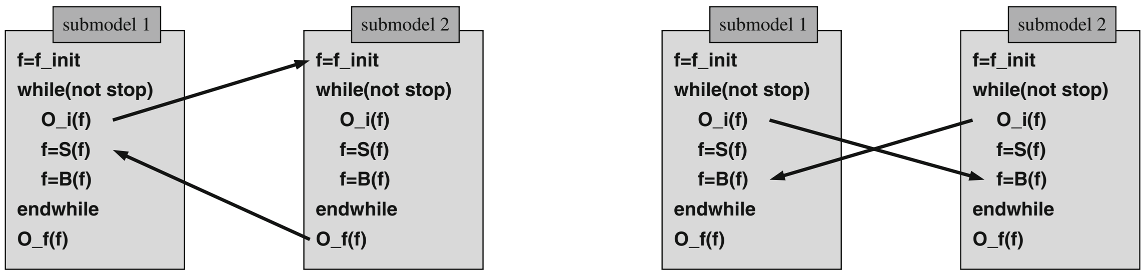

Fig. 2 Generic submodel execution loop and two examples of coupling templates

Table 1 Relation between coupling templates and position in the scale separation map for two submodels X and Y sharing the same computational domain

Name	Coupling	Temporal scale relation
Interact	$O_i^X \to S^Y$	Overlap
Call	$O_i^X \to f_{\text{init}}^Y$	X larger than Y
Release	$O_f^Y \to S^X$	Y smaller than X
Dispatch	$O_f^X \to f_{\text{init}}^Y$	Any

or *mapper*. They transform the output of a submodel into the input of another one, according to the need of the selected scale bridging. For instance, a scale bridging technique could transform the interaction between small-scale suspensions in a fluid as an effective viscosity at a larger scale (Lorenz and Hoekstra 2011).

Structurally, the coupling between submodel can be described by the concept of *coupling templates*, or at a the level of the entire application as *computational patterns*. Figure 2 shows an example of coupling between submodels. As a result of the above generic structure of the time loop, only a few coupling is possible. The sender of information is either O_i or O_f. In addition, the receiving operators can only be S, B or f_{init}.

An interesting observation is that coupling templates reflect the type of scale separation between the interconnected submodels. This is explained in Table 1.

The software specification of the submodels and their coupling can be formulated with a multiscale modeling language (MML) (Falcone et al. 2010), which describes the architecture of a multiscale model. It describes the scales and computational requirements of submodels and scale bridging components, each with pre-defined output and input ports used for communicating data. These ports are associated with a data type and a communication rate tied to the submodel scales, and an output port can be coupled to an input port if these data types and rates match.

The software MUSCLE 2 (Borgdorff et al. 2013b, c) provides a middleware to build such a multiscale application, by connecting submodels together, in a data-driven way. Legacy code, written in several standard programming language can be coupled. It simply requires to add in each submodel, send, and receive calls to local output and input ports. These ports, as well as their connections, are defined in a configuration file and reflect the desired coupling architecture.

MUSCLE 2 allows distributed multiscale computating, namely, runs, where the different submodels can be executed on remote machines, whether supercomputers or not. As a consequence, data transfer is implemented in a very generic way, and fast communication cannot be guaranteed. Recently, the muscleHCP C++ library has been developed (Belgacem and Chopard 2016) to alleviate this problem. This library assumes C++ code tightly complying with the MMSF concepts and provides MPI interconnection between submodels that partition the cores of a given parallel machines.

3 Example: transport of volcanic ashes

The overview of the MMSF approach, its associated tools and software given in the previous section, is a summary of the full framework. We refer the reader to Chopard et al. (2014), Borgdorff et al. (2013c, b), Borgdorff et al. (2014) for in-depth descriptions. To illustrate the main concepts we introduce an example of an application, we developed within this formalism. We refer to a multiscale model to describe the atmospheric transport of volcanic ashes (usually called tephra transport in the community), from its source (the volcano plume), to the local region dominated by sedimentation, to far regions, where atmospheric particles can have severe effects on commercial aviation.

The particle transport and deposition solver is called TETRAS. This is a parallel, scalable code described in detail in Künzli et al. (2016). The targeted global architecture of the application, corresponding to a distributed computation on several machines, is illustrated in Fig. 3, using the MML description (Falcone et al. 2010). Boxes are the submodel and the arrows indicate the type of coupling. The results obtained with a simplified version of the application, namely, a two-domain computation, in which the local one the volcanic plume is resolved is computed with a high spatial and temporal resolution, whereas on the regional scale, it has a coarse resolution and contains no plume model. A meteorological wind field is assumed to be known, which is used to compute the advection of the volcanic ashes. On top of the TETRAS transport model, an aggregation process can be turned on to simulate the important fact that small tephra particles might stick together to form bigger suspensions that will sediment faster than the original smaller particles.

This example illustrates what in MMSF is called a multidomain coupling between the so-called “plume” and “region” submodels. It also corresponds to a time-scale overlap, hence O_i to B coupling. On the other hand, the aggregation submodel has a “single-domain” relation with “plume”. In addition, we assumed here a time-scale separation reflecting the hypotheses that aggregation is a faster process that advection–diffusion, considering the spatial resolution we used near the volcano. Therefore, we see here an example of a $O_i \to f_{\text{init}}$ coupling template.

4 DES approach to scheduling

As suggested in the previous section, an important step to execute a MMSF application is to schedule each of its sub-

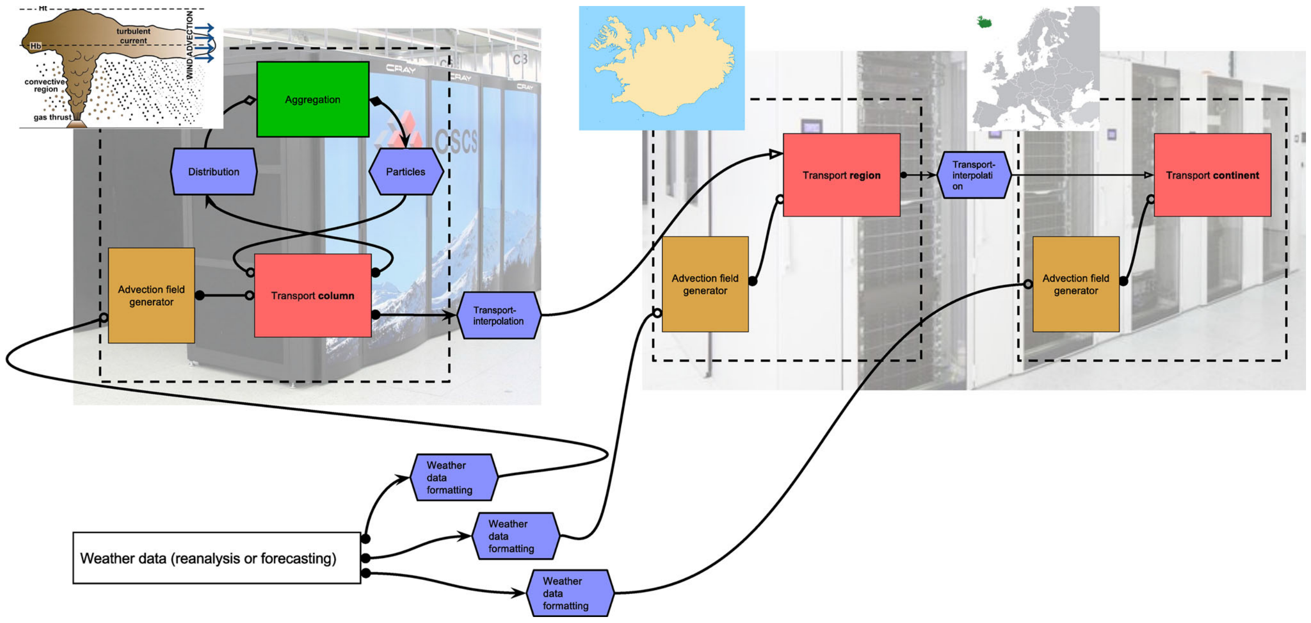

Fig. 3 Deployment of the tephra transport application, according to the MMSF approach, on several super-computer

models on available computing resources. Indeed, finding an optimal scheduling and placement for any given configuration is a difficult task.

Let us consider two tightly coupled submodels with overlapping temporal scales (corresponding to the *interact* coupling template). Each iteration of the first submodel will require the completion of the corresponding iteration of the second submodel (and vice versa). If one model is faster than the other one in terms of computation, then it will have to wait on the other one to start the next iteration. However, if both submodels are parallelizable, we can assign an appropriate number of computing cores to each submodel such as to match their respective execution times and avoid idle CPUs.

Another opportunity for scheduling optimization appears with the *call-release* coupling template, where the temporal scales are separated. The execution of an iteration of the model with the coarser time scale will depend on a complete execution of the model with the finer time scale. The execution of the coarser model will, therefore, be suspended, while the finer model runs. To maximize resource usage, both sub models could run on the same computing cores, because they will never be active on the same time.

The above examples could be optimized analytically, if performance models are available for both submodels. However, finding an optimal scheduling becomes difficult for applications with more than two submodels, like the transport application presented in Figs. 3 and 4. The difficulty also increases when submodels are deployed on several machines and the communication times become significant.

To optimize such placements, we have designed a discrete event simulator (DES) able to evaluate the effect of a given scheduling without the need to run the full application. The goal is to allow the user to quickly explore different scheduling options and have a feedback of its expected quality. Moreover, the DES can be combined with multiscale computing patterns introduced in Sect. 6, to facilitate automated placements.

In our approach, every submodel is simply represented as a state machine, whose current state models the progression of each simulation. To match the MMSF, this state is expressed in terms of position inside the generic time loop, as shown in Fig. 2. In other words, we only track the current interaction and the current operation. We can predict the time to advance to the next operation using a performance model depending on the model resolution (spatial and temporal scales) and the number cores assigned to each submodel. Of course, performance models must be established and experimentally validated for each submodel, but such studies are routinely performed on HPC architectures and most submodel implementations are published with speed-up curves. To include the delays caused by the communications, we also use a performance model to estimate the amount of information to transmit. The hardware itself is modeled using a relative speed for the CPU cores and the communication speed of the interconnection network. Again, those values are routinely measured for most HPC infrastructure.

With the above hypotheses, we can define the *discrete events* modeled in our system: `submodel started`, `submodel stopped`, `computation done`, `communication received`. These events represent the out-

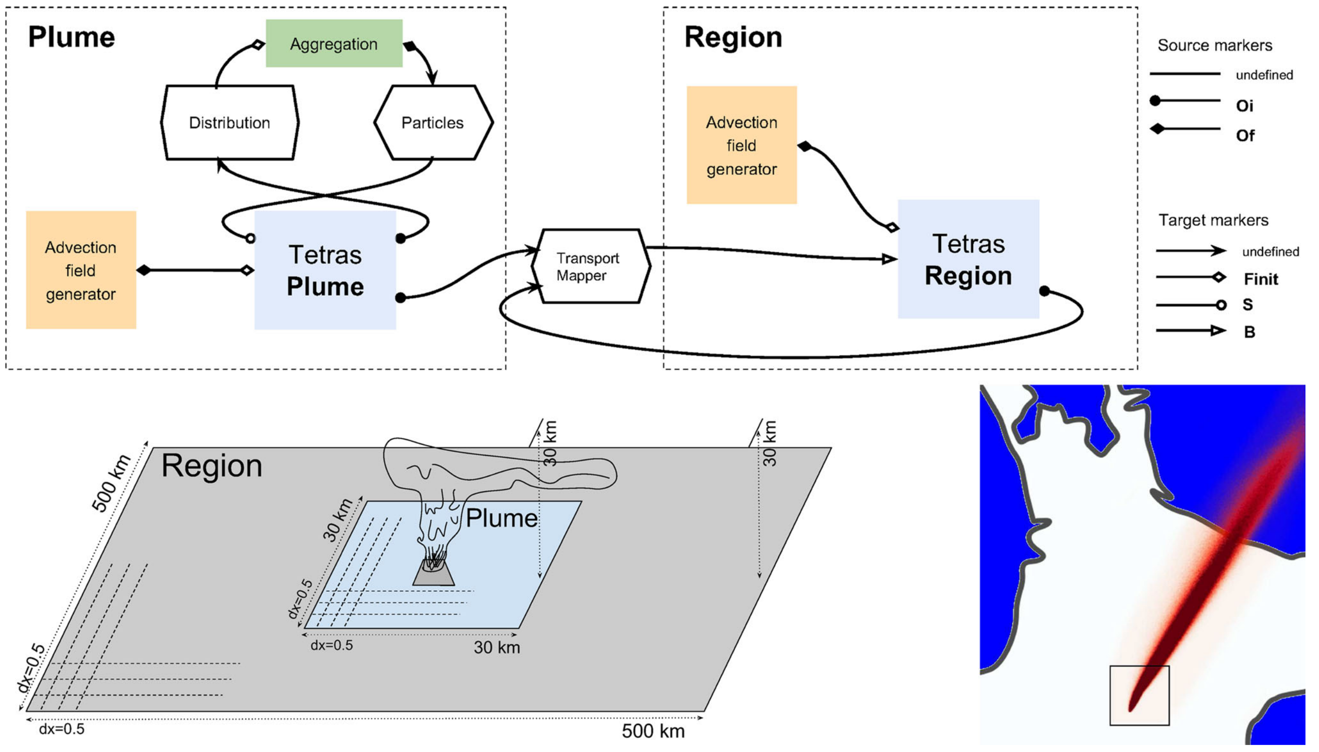

Fig. 4 Results of a simplified, two-domain multiscale tephra transport code for the eruption of volcano Ruapehu, 1996 NZ (upper panel) and corresponding MML representation (lower panels)

come of actions and lead to state transitions. For instance, let us consider the multiscale coupling, as shown in Fig. 2, left. When the event `submodel 1 started` occurs, we know that the operators f_{init} will be performed before observing the state and sending the necessary information to *submodel 2*. Using performance models and hardware description, we can predict the time to perform these operations and we can add the event `communication received` in the future for *submodel2*. This event in turn will yield a full execution of *submodel 2* which again could be estimated, as well as the time needed to send back the results to *submodel 1*. Then, we can issue a `communication received` event for *submodel 1*. This new action will allow *submodel 1* to continue its computation and thus finish the iteration. This will produce a new event `computation done` that will represent the end of the current iteration, and so on.

We illustrate an application of our MMSF discrete event simulator for the simplified tephra transport application, as detailed in Fig. 4. In this example, the *Plume* submodel sends its particle field to the *Aggregation* submodel and to the *Region* submodel at each iteration. To proceed, it must receive the new particle distribution from the *Aggregation* submodel. There is a tight coupling between *Aggregation* and *Plume*, but not between *Plume* and *Region*. However, *Region* cannot proceed without receiving data from *Plume*, and thus, it depends indirectly on *Aggregation*. Moreover, *Plume* will exchange particles distribution across all its domain with *Aggregation*, while *Plume* only sends to *Region* the particles leaving its domain (advected by the wind). Therefore, communication between *Plume* and *Aggregation* is more demanding than between *Plume* and *Region*. Figure 5 shows two different placements of the application simulated by the DES. In the left panel, we show a setting, where the *Plume* submodel was executed on a 50-core machine, while both *Aggregation* and *Region/submodels*, were executed on a different similar machine. In the right panel, the *Plume* and *Aggregation* submodels were executed on the same machine, while the *Region* model was executed alone. As expected, because of the importance of communications, the second configuration will compute the solution faster. Such setting also rationally saves shared computing resources by avoiding idle CPU during the reservation (dark red vs. dark green). For example, we can see that increasing the number of cores assigned to the *Region* submodel will not reduce the time to solution, because the rest of the application will be the limiting factor.

Currently, we have implemented all the basic components of such simulator and we have performed several qualitative analyses to validate its meaningfulness. In the future, we plan to validate our approach quantitatively and integrate the current simulator with the rest of the MMSF tool chain.

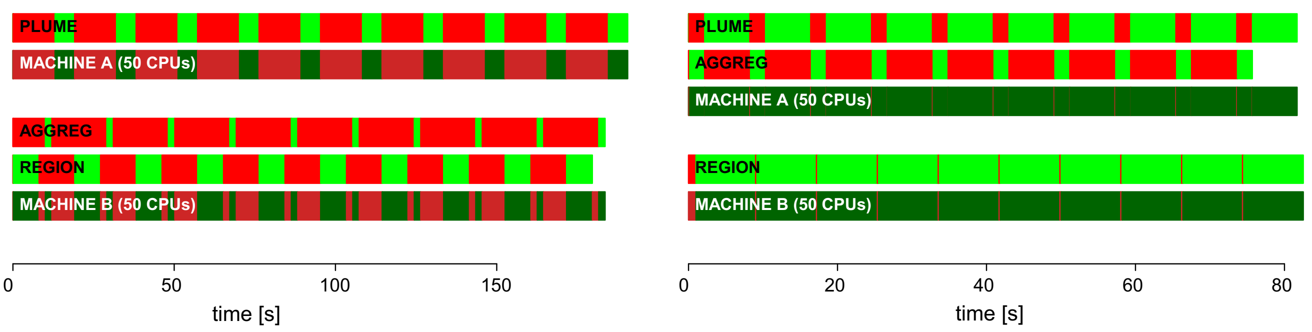

Fig. 5 Discrete event simulation of the processing activity of two different placements of the tephra transport submodels. The submodel activity is shown with light colors: light green for computation and light red for waiting. The machine activity is shown with similar dark colors: dark green for CPU activity and dark red for waiting. Time is represented on the horizontal axis, and the scale is different for both examples, since the left case is twice slower

5 Scale bridging technique: time splitting and amplification

This section illustrates one example of a scale bridging technique coined "amplification" in Hoekstra et al. (2010), and compares it with the more classical "time-splitting" technique.

A situation which often appears in multiscale modeling is the coupling between a fast and a slow process, with a large time-scale separation.

As an example, consider the problem of coral growth described in Merks et al. (2003), where the geometry of the coral changes by a small amount in 1 year, while the fluid flow, which delivers the nutrients, adapts almost immediately to the new boundary condition given by the coral shape. In terms of modeling, it is of course useless to simulate the fluid for 1 year, for each of the growing steps of the coral. It is much better to split the two process, and deal with them at their own time scale. In practice, this means to compute the flow until a steady state is reached, with a fixed coral structure. Then, using this fixed fluid state, the flow of nutrients can computed, leading to the growth of the coral. This process can be iterated as many times as needed, to be obtained the coral structure after the desired number of years. Using the MMSF formalism, this coupling of the two submodels (fluid and growth) is expressed as

$$O_i^{(\text{growth})} \rightarrow F_{\text{init}}^{(\text{fluid})} \quad O_{\text{f}}^{(\text{fluid})} \rightarrow S^{(\text{growth})}.$$

This time separation scale bridging technique is often call *time-splitting*. Its accuracy as a function of the scale separation has been studied in Caiazzo et al. (2009).

Although time-splitting is very simple and natural, it is less appropriate when the fast process does not converge to a constant value, but keeps varying in time. We are, for instance, thinking of thrombus formation in a cerebral aneurysm. The growth of the thrombus is driven by a pulsatile blood flow that continuously varies over a cardiac cycle, while the thrombus needs many heart beats to substantially grow. To avoid simulating the actual number of cardiac cycles (each of which may require a large amount of CPU time, even on a large parallel computer), the idea is to *amplify* the growth rate, so that, in one cardiac cycle, the growth of the thrombus is already noticeable. Such a scale bridging technique is called *amplification* or *acceleration*. It has been used, for instance, in Malaspinas et al. (2016) to produce the thrombus illustrated in Fig. 6 in only 2000 cardiac cycles.

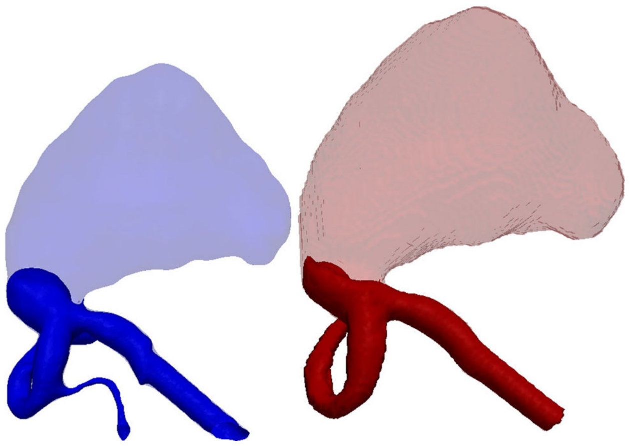

Fig. 6 *Left* simulated thrombus growth (light blue) in a patient specific geometry. *Right* the corresponding clinical observation (light red). The *amplification* scale bridging technique was used to coupled the blood flow time scale with that of the thrombus growth (color figure online)

This technique, when formulated as a coupling template in MMSF, corresponds to

$$O_i^{(\text{blood})} \rightarrow S^{(\text{thrombus})} \quad O_i^{(\text{thrombus})} \rightarrow B^{(\text{fluid})}$$

which means, according to Table 1, that the time separation has been absorbed by the amplification.

5.1 A simple example

In this section, we will analyze quantitatively, using a simple dynamical problem, the accuracy of the *amplification* scale bridging and compare it to time-splitting.

The following system can be seen as an overly simplified version of the thrombus formation problem:

$$\begin{cases} \dot{h} = \alpha(u_{\max} - u) \\ \dot{u} = -\gamma u + F(t) + \beta h \end{cases} \tag{1}$$

where α, β, γ, and $u_{\max}$ are parameters and $F(t)$ is a given function of time. The link with our motivation problem is the following: $h(t)$ is the thickness of the clot that forms on the vessel wall. It starts with $h(t = 0) = 0$. Clot grows if the wall shear rate is abnormally small, and stops when a threshold value, h_∞ is reached. Here, we simply abstract this process by saying that the growth is proportional to the distance to this threshold. Above the threshold, we can imagine an erosion process takes place, thus reducing h.

The quantity u represents the speed of the blood. There is a friction parameter γ and an external force $F(t)$, mimicking the role of the heart. As the clot grows, the lumen decreases and the speed u increases to keep the same flow. Here, this is abstracted by adding h as a pseudo force term.

5.2 Constant driving force

We first assume that $F = F_0$ is constant over time. The steady state of this system is easily obtained by setting $\dot{h} = \dot{u} = 0$. We then obtain

$$u_\infty = u_{\max} \quad h_\infty = \frac{\gamma u_{\max}}{\beta} - \frac{F_0}{\beta}.$$

It is interesting to note that the steady state does not depend on α, and in a numerical solver, we can use a large value to quickly reach the final solution. This is the essence of the amplification method.

Let us assume that the growth of h is slow and that we are interested to know its evolution at time intervals Δt large compared to the time needed for u to reach a steady state. For a given value of h_0 assumed to be constant over Δt, the steady-state value u_0 that u will reach is

$$u_0 = \frac{\beta}{\gamma} h_0 + \frac{F_0}{\gamma}.$$

We can now advance h assuming that $u = u_0$ during Δt, and we get

$$h_1 = h_0 + \alpha(u_{\max} - u_0)\Delta t.$$

This procedure can be repeated to give the iterative system

$$\begin{cases} u_i = \frac{\beta}{\gamma} h_i + \frac{F_0}{\gamma} \\ h_{i+1} = h_i + \alpha(u_{\max} - u_i)\Delta t \end{cases} \tag{2}$$

with $u_i = u(i\Delta t)$ and $h_i = h(i\Delta t)$. This can also be written as

$$\begin{cases} u_i = \frac{\beta}{\gamma} h_i + \frac{F_0}{\gamma} \\ h_{i+1} = h_i + \alpha \left(u_{\max} - \frac{\beta}{\gamma} h_i - \frac{F_0}{\gamma} \right) \Delta t \end{cases} \tag{3}$$

whose solution at time $t = \infty$ is, as before

$$u_{\max} - \frac{\beta}{\gamma} h_\infty - \frac{F_0}{\gamma} = 0.$$

Figure 7 (left) shows the solution of Eq. (1) using a standard Euler scheme (continuous lines). The final value of $u = u_{\max}$ and $h = h_\infty$ is indicated with the horizontal dashed lines. The figure also shows the quality of the time-splitting solution that consists of reaching a steady state for u, assuming h constant, then advancing h for a time Δt using the value of u, and so on. We can see that if h grows slowly compare to the speed at which u reaches a steady state, the approximation is very good. In addition, if one compares the time steps used for the accurate Euler scheme, namely, $\delta t = 0.1$ and $\Delta t = 20$, there is a speedup of 200 when using such an approximation. We also observe (not shown here) that if the time separation decreases (larger α), the approximation becomes less and less good for small times, but reaches anyway the correct solution in the long-time regime.

Alternative to the above time-splitting approach, we can consider an approximation based on the idea of *amplification*. In this case, one artificially increases the growth rate α of the slow process. Figure 7 (right) shows the quality of this approximation for an amplification by a factor $\mu = 10$. In this case, we solve Eq. (1) with $\alpha \to \mu\alpha$, and for a time interval $s \in [0, t_{\max}/\mu]$:

$$\begin{cases} \frac{dH}{ds} = \mu\alpha(u_{\max} - U) \\ \frac{dU}{ds} = -\gamma U + F + \beta H \end{cases} \quad s \in [0, t_{\max}/\mu]. \tag{4}$$

Here, we used the same time steps $\delta t = 0.1$ for both the accurate Euler scheme and the *amplified* Euler scheme. In Fig. 7 (right), the rescaled quantities $U(\mu s) \approx u(t)$ and $H(\mu s) \approx h(t)$ are shown. A good approximation is obtained. In terms of speedup, one gets a gain of a factor $\mu = 10$, because the computation has to be performed only to a shorter time interval, of length t_{max}/μ.

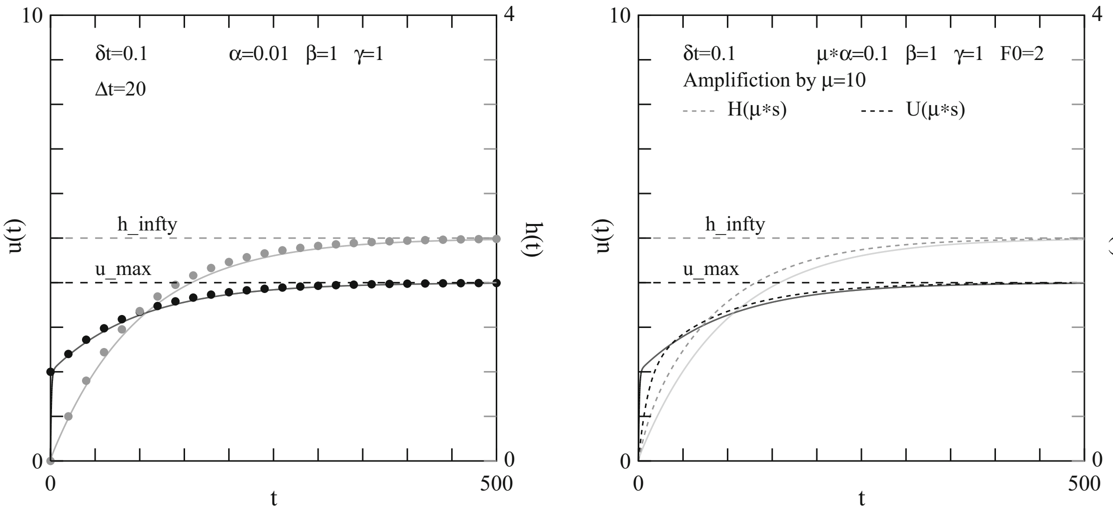

Fig. 7 *Left panel:* Accurate numerical solution of Eq. (1), displayed with the continuous lines. Approximation given by Eq. (3): black and gray dots. *Right panel:* solution using amplification

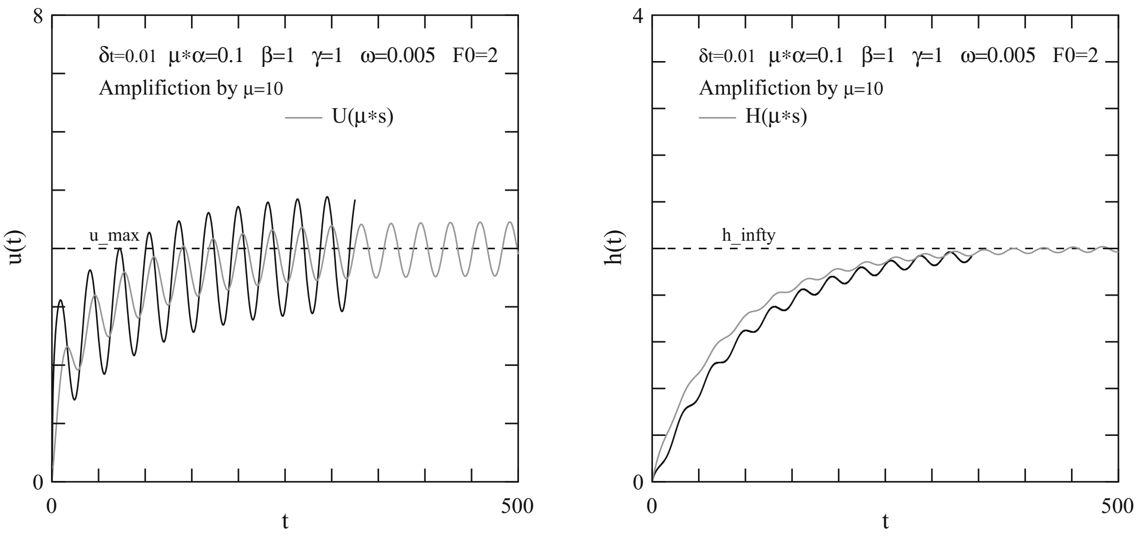

Fig. 8 Accurate numerical solution of Eq. (1), displayed with the continuous black lines. Solutions with *amplification* given by increasing the growth rate α by a factor μ are shown with the gray curves. The left panel shows the values of the velocity and the right panel the values of the growth

5.3 Time-dependent driving force

When the driving force F in Eq. (1) keeps varying in time, as is the case for plusatile flows, the time-splitting approach is not as easily applied as u does not quickly reach a quasi constant value. On the other hand, the amplification technique is obvious.

Here, we consider the case, where

$$F(t) = \sin \omega t + F_0.$$

As in the previous section, we consider an amplification factor of $\mu = 10$ when solving Eq. (1), meaning that we replace α by $\mu\alpha$. We also focus on the accelerated fields H and U, as defined in the previous section, leading to Eq. (4).

Figure 8 shows the result of the accurate Euler solution without amplification (black continuous lines), as well as the result of the amplification case (gray lines), with again $\mu = 10$. We can observe a satisfactory agreement with the accurate solution. We see a slight phase delay for $u(t)$ and $h(t)$, which, however, looks of little importance at a large time scale. The average time values of h and u are not affected by the amplification. However, the amplitude of u is significantly under-estimated. Although this is not critical if one is interested in the growth process, further investigations of this feature would be desirable. An intuitive explanation is that the faster growth adds more resistance to the flow, preventing it to reach its peak values. If we were interested in the correct value of u, we could rescale the friction of the force amplitude but this would require further investigation.

From the results presented in the previous two sections, we observe that time-splitting offers a very high potential to save CPU time when the problem amounts to coupling a process that quickly reaches a steady state with a slow process that mostly depends on this steady state only. Amplification requires no software modification and is straightforwardly implemented. It is also applicable when the fast process is time dependent. The examples shown here suggests that the amplification factor (and the speedup of the computation) should not be taken too big, at the risk of reducing too much the accuracy in the earlier regime. However, if one is interested in the long-time regime, the early time inaccuracy is corrected as time progresses. It is also interesting to note that the time at which the growth process reaches a given fraction, close to its final value is correctly captured by the accelerated field U.

Of course, the present example is very simple and its behavior cannot be generalized to other processes without proper investigations. However, our main goal here is to illustrate the principle of *amplification* as simple bridging techniques that we intend to further investigate.

6 Multiscale high-performance computing patterns

We expect that multiscale simulations will be one of the main workloads on high-performance computing systems. As individual clock speeds are no longer increasing, HPC systems can only achieve larger computational speeds by deploying more processors. The parallelism in a model is usually achieved by spatial domain decomposition. In a weak scaling sense, adding more processors results in simulating increasingly large systems. Yet, we are usually also interested in simulating larger time scales, and as the system becomes larger, we typically need to longer integrate the dynamical variables, to see the phenomena of interest. This means that it will become increasingly more difficult to study large space and long-time behavior with monolithic codes. Multiscale computing is able to circumvent this problem by deploying single scale components on HPC architectures, to produce optimal performance and to bridge both time and space scales.

In the spirit of the MMSF, we have developed *multiscale computing patterns* (MCP) Alowayyed et al. (2017) as a generic vehicle to realize optimized, that is load balanced, fault tolerant, and energy aware high-performance multiscale computing. MCPs should lead to further separation of concerns. The application developers compose multiscale models and execute multiscale simulations in the MMSF. Pattern software, maybe employing the discrete event simulations presented in Sect. 4, then realizes optimized, fault tolerant, and energy aware multiscale computing.

The first step is to identify generic computing patterns that allow the development of algorithms for common multiscale computing scenarios. We define multiscale computing patterns as high-level call sequences that exploit the functional decomposition of multiscale models in terms of single scale models. We have identified three MCPs (Alowayyed et al. 2017):

1. Extreme scaling, where one (or a few) single scale models require HPC, which are coupled to other, less costly single scale models.
2. Heterogeneous multiscale computing, where a very large number of microscale models are coupled to a macroscale model.
3. Replica computing, where a large number copies (replicas) are executed, that may or may not exchange information.

MCPs can be expressed at the level of the task graph, which is the directed acyclic graph used to determine the execution order of submodels, to schedule submodel dependencies, and to estimate runtime and communication cost (Borgdorf et al. 2013b). We formulate generic task graphs for each MCP and use them to obtain an optimized mapping of the multiscale simulation on HPC resources. Figure 9 summarizes the approach. An MCP is a generic task graph combined with data on the performance of single scale models, a specification of a given multiscale application in terms of the MMSF and a set of algorithms and heuristics that combine this into detailed input/configuration files for the execution environment in which the multiscale simulation will be executed.

In Alowayyed et al. (2017), we discuss a few examples of MCPs, for the case of an extreme scaling scenario, where cell-resolved blood flow simulations are coupled to continuous flow simulation that serve as inlet and outlet regions. We have recently implemented pattern software, where all functionality shown in Fig. 9 has been realized for the extreme scaling and the replica computing patterns, and applied to a range of multiscale models, from multiscale modeling of fusion in the ITER reactor to multiscale modeling of binding affinities between drugs and target molecules (Alowayyed et al. 2017).

7 Outlook and open questions

The MMSF was conceived almost a decade ago and its theoretical underpinning is described in detail by Borgdorf et al.

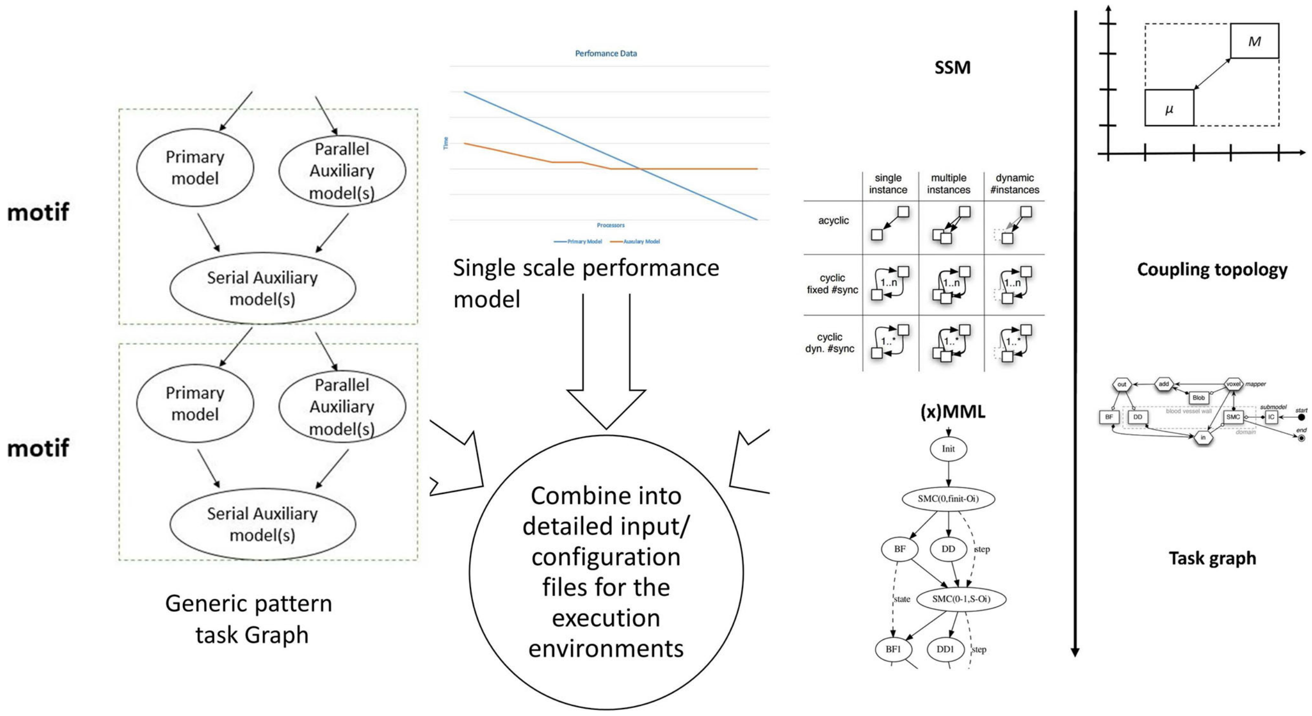
motif
Primary model
Parallel Auxiliary model(s)
Serial Auxiliary model(s)
motif
Primary model
Parallel Auxiliary model(s)
Serial Auxiliary model(s)
Generic pattern task Graph
Perfomance Data
Single scale performance model
Combine into detailed input/ configuration files for the execution environments
SSM
single instance
multiple instances
dynamic #instances
acyclic
cyclic fixed #sync
cyclic dyn. #sync
(x)MML
Init
SMC(0,finit-Oi)
BF
DD
step
state
SMC(0-1,S-Oi)
BF1
DD1
M
μ
Coupling topology
Task graph

Fig. 9 Multiscale computing patterns implemented as generic task graphs and algorithms to generate sufficient information for the execution engines

(2013b). Since then a growing number of multiscale models have been designed and implemented using the MMSF (Belgacem et al. 2013a; Borgdorff et al. 2013b, 2014. Most of them relied on MUSCLE as an API and execution environment to couple together single scale codes and execute the overall simulation. However, recently, also other examples implementing MMSF have been demonstrated, relying on MuscleHPC (Belgacem and Chopard 2016) or FAB-SIM (Groen et al. 2016).

The theoretical foundation of the MMSF seems well established. A major open issue is a classification of scale bridging methods. The amplification, as discussed in this paper, seems to be one important generic class, but in our opinion a deeper understanding of scale bridging methods would be an important and needed addition to the MMSF.

Another very relevant issue in relation with multiscale modeling and simulation is that of sensitivity analysis and uncertainty quantification. In our view, we can formulate multiscale uncertainty quantification in terms of the MMSF, relying on the replica computing pattern. We have recently proposed a generic family of semi-intrusive uncertainty quantification algorithms for time-scale separated multiscale model (Nikishovay and Hoekstra 2017). The embedding of such multiscale uncertainty quantification in the MMSF, and algorithms for the different classes of multiscale models in the MMSF is currently under active investigation, as well as mapping of such algorithms to HPC, using MCP software.

Over the years the MMSF has been used to realize distributed multiscale computing (Belgacem et al. 2013a; Borgdorff et al. 2013c) and high-performance multiscale computing (Borgdorff et al. 2014; Belgacem and Chopard 2016) relying on a plethora of middleware and runtime support systems (Blegacem et al. 2015). The lesson learned form this experience shows that a multiscale simulation requires a planning phase, in which a user specifies *what* is needed (single scale codes and related tools), *how* to execute (providing performance information and using patterns to produce execution plans), and *where* to execute [finding the best resources available to a user, using, e.g., the QCG middleware Piontek et al. (2016)].

The multiscale computing job can then be *launched* by the chosen middleware. A *Coordinator* will be launched first, which in turn starts up the submodels and mappers that perform the *Computation* (relying on, e.g., information from the MCPs), and facilitates their *Communication* by propagating information about which component runs where. The Coordinator is different depending of the ‘glue’ software that is used (e.g. Muscle, MuscleHPC, …), but in all cases, its role in the multiscale computing is comparable, and from the perspective of MMSF based multiscale computing, is a generic component. We intend to formalize this generic architecture, so that it becomes possible to create MMSF compliant software, which in the end may even lead to a well-established API for multiscale computing.

Although the MMSF is in our view now well established, one important aspect seems to be still missing in both the MML and in implementations such as MUSCLE or MuscleHPC. And that is the notion of a dynamic number of instantiations of single scale models. Borgdorf et al. (2013b) already identified coupling templates where the number of instantiations of a single scale model can be dynamic and unknown at compile time. Important classes of mutliscale models that fall under this category are heterogeneous multiscale.

dynamics, a varying number of microscale simulations have to be started in order to simulate unknown properties at the macroscale. In addition, in replica computing applications, such dynamic features are required. We are currently in the process of updating the Multiscale Modeling Language to be able to express this in a natural way. Moreover, we are also in the process of upgrading MUSCLE2 to MUSCLE3, where these dynamic instantiations of single scale models during runtime will be available.

Another aspect of multiscale computing that has not yet received sufficient attention is that of data handling, that is optimizing the transport of data between submodels. Certainly in HPC environments, and if single scale models need to exchange large volumes of data, dedicated data pattern software would be needed.

The currently running EU funded project COMPAT (www.compat-project.eu) is implementing the MCP software, and benchmarking high-performance multiscale computing applications. The MCP software should automate as much as possible load balancing of multiscale simulations, and should also help in choosing execution plans that are optimal in terms of energy usage.

Acknowledgements The authors acknowledge financial support from the Swiss Initiative PASC, from CADMOS, and from the COMPAT EU Project. BC, JLF, and PK thank Constanza Bonadonna for the collaboration on the model for volcanic ashes. BC thanks Alireza Yasdani for stimulating discussions on the amplification scale bridging techniques. On behalf of all authors, the corresponding author states that there is no conflict of interest.

References

Alowayyed S, Groen D, Coveney PV, Hoekstra A (2017) Multiscale computing in the exascale era. J Comput Sci 22:15–25. https://doi.org/10.1016/j.jocs.2017.07.004

Alowayyed S, Piontek T, Suter JL, Hoenen O, Groen D, Luk OO, Bosak B, Kopta P, Kurowski K, Perks O, Brabazon K, Jancauskas V, Coster D, Coveney PV, Hoekstra AG (2017) Patterns for high performance multiscale computing. Future Gener Comput Syst

Blegacem MB, Chopard B (2015) A hybrid HPC/cloud distributed infrastructure: coupling EC2 cloud resources with HPC clusters to run large tightly coupled multiscale applications. Future Gener Comput Syst. https://doi.org/10.1016/j.future.2014.08.003

Belgacem MB, Chopard B (2016) Muscle-hpc: a new high performance api to couple multiscale parallel applications. Future Gener Comput Syst 67:72–82. https://doi.org/10.1016/j.future.2016.08.009

Belgacem M Ben, Chopard B, Borgdorff J, Mamonski M, Rycerz K, Harezlak D (2013a) Distributed multiscale computations using the MAPPER framework. Procedia Comput Sci 18:1106–1115. https://doi.org/10.1016/j.procs.2013.05.276

Borgdorf J, Falcone JL, Lorenz E, Bona-Casas C, Chopard B, Hoekstra AG (2013b) Foundations of distributed multiscale computing: formalization, specification, analysis and execution. J Parallel Distrib Comput 73:465–483

Borgdorff J, Mamonski M, Bosak B, Groen D, Belgacem MB, Kurowski K, Hoekstra AG (2013c) Distributed multiscale computing with the multiscale modeling library and runtime environment. Procedia Comput Sci 18:1097–1105

Borgdorff J, Mamonski M, Bosak B, Groen D, Belgacem MB, Kurowski K, Hoekstra AG (2013) Multiscale computing with the multiscale modeling library and runtime environment. Procedia Comput Sci 18(0):1097–1105. https://doi.org/10.1016/j.procs.2013.05.275. http://www.sciencedirect.com/science/article/pii/S1877050913004183

Borgdorff J, Belgacem MB, Bona-Casas C, Fazendeiro L, Groen D, Hoenen O, Mizeranschi A, Suter JL, Coster D, Coveney PV, Dubitzky W, Hoekstra AG, Strand P, Chopard B (2014) Performance of distributed multiscale simulations. Philos Trans A 372(2021):20130407

Caiazzo A, Falcone JL, Chopard B, Hoekstra AG (2009) Asymptotic analysis of complex automata models for reaction-diffusion systems. Appl Numer Math 59(8):2023–2034

Chopard B, Borgdorff J, Hoekstra AG (2014) A framework for multiscale modeling. Philos Trans A 372:20130,376

Dada JO, Mendes P (2011) Multi-scale modelling and simulation in systems biology. Integr Biol 3(2):86–96

Evans D, Lawford PV, Gunn J, Walker D, Hose DR, Smallwood R, Chopard B, Krafczyk M, Bernsdorf J, Hoekstra A (2008) The application of multi-scale modelling to the process of development and prevention of stenosis in a stented coronary artery. Philos Trans R Soc 366:3343–3360

Falcone JL, Chopard B, Hoekstra A (2010) MML: towards a multiscale modeling language. Procedia Comput Sci 1(11):819–826

Groen D, Borgdorff J, Bona-Casas C, Hetherington J, Nash RW, Zasada SJ, Saverchenko I, Mamonski M, Kurowski K, Bernabeu MO, Hoekstra AG, Coveney PV (2013) Flexible composition and execution of high performance, high fidelity multiscale biomedical simulations. Interface Focus 3(2):20120087

Groen D, nad James Suter APB, Hetherington J, Zasada SJ, Coveney PV (2016) Fabsim: facilitating computational research through automation on large-scale and distributed e-infrastructures. Comput Phys Commun. https://doi.org/10.1016/j.cpc.2016.05.020

Hoekstra AG et al (2016) Towards the virtual artery: a multiscale model for vascular physiology at the pcb interface. Philos Trans R Soc A 374(0160):146. https://doi.org/10.1098/rsta.2016.0146

Hoekstra AG, Caiazzo A, Lorenz E, Falcone JL, Chopard B (2010) Modelling complex systems by cellular automata, chap. 3. Springer, Berlin

Hoekstra AG, Coveney P, Chopard B (2014) Position a paper on multiscale modeling and computing. Philos Trans A 372:20130377

Ingram G, Cameron I, Hangos K (2004) Classification and analysis of integrating frameworks in multiscale modelling. Chem Eng Sci 59:2171–2187

Künzli P, Tsunematsu K, Albuquerque P, Falcone JL, Chopard B, Bonadonna C (2016) Parallel simulation of particle transport in an advection field applied to tephra dispersal. Comput GeoSci 89:174–185

Lorenz E, Hoekstra A (2011) Heterogeneous multiscale simulations of suspension flow. Multiscale Model Simul 9:1301–1326

Malaspinas O, Turjman A, de Souza DR, Garcia-Cardena G, Raes M, Nguyen PTT, Zhang Y, Courbebaisse G, Lelubre C, Boudjelti KZ, Chopard B (2016) A spatio-temporal model for spontaneous thrombus formation in cerebral aneurysms. J Theor Biol 394:68–76

Merks RMH, Hoekstra AG, Kaandorp JA, Sloot PMA (2003) Models of coral growth: spontaneous branching, compactification and the laplacian growth assumption. J Theor Biol 224:153–166

Nikishovay A, Hoekstra A (2017) Semi-intrusive uncertainty quantification for multiscale models. SIAM J. Uncertain Quantif

Piontek T, Bosak B, Cinicki M, Grabowski P, Kopta P, Kulczewski M, Szejnfeld D, Kurowski K (2016) Development of science gateways using qcglessons learned from the deployment on large scale distributed and hpc infrastructures. J Grid Comput 14:559–573

Tahir H, Hoekstra A, Lorenz E, Lawford P, Hose D, Gunn J, Evans D (2011) Multiscale simulations of the dynamics of in-stent restenosis: impact of stent deployment and design. Interface Focus 1:365–367

Tahir H, Casas CB, Hoekstra A (2013) Modelling the effect of a functional endothelium on the development of in-stent restenosis. PLoS ONE 8(e66):138

Weinan E, Li X, Ren W, Vanden-Eijnden E (2007) Heterogeneous multiscale methods. A review. Commun Comput Phys 2:367–450

Yang A, Marquardt W (2009) An ontological conceptualizatin of multiscale models. Comput. Chem. Eng. 33:822–837