Distributed multiscale computing

Borgdorff, J.

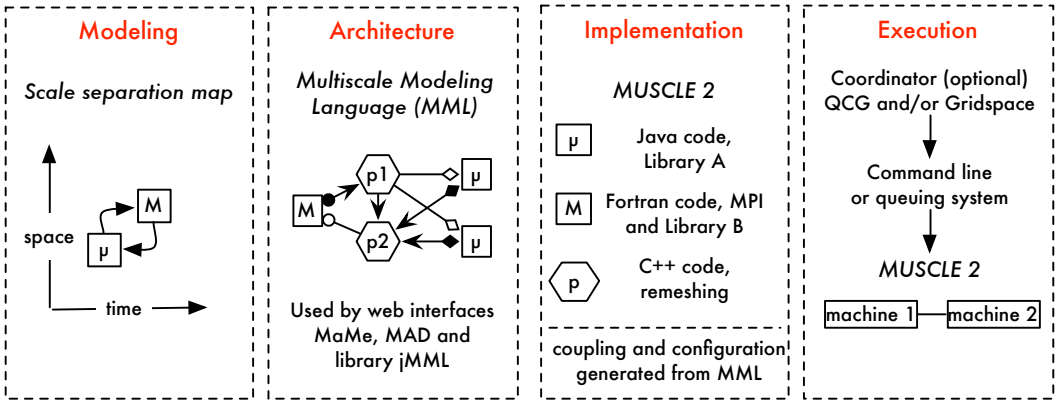
6

Conclusions and discussion

In this thesis, distributed multiscale computing has been treated from multiple angles: devising a conceptual basis; providing a coupling library and runtime environment; setting up a toolchain and infrastructure; implementing and executing a multiscale model of in-stent restenosis; and testing the performance of distributed multiscale computing. Each part has evolved due to the varying needs of different multiscale models and limitations of the available hardware.

The conceptual and computational framework described in this thesis, dubbed the multiscale modelling and simulation framework (MMSF) [17, 36], is summarised in Figure 6.1.

The conceptual foundations in Chapter 2 allows describing, specifying, and categorising multiscale models. It elucidates how a scale may be interpreted, as a granularity or resolution and a total size or scope. It shows how scales apply to modelling, by modelling phenomena as a set of interacting phenomena, each with a definite scale. Correspondingly, a multiscale model is a set of coupled single scale models. Ideally, each single scale model is independent and each instance of it represents a definite part



(a) Conceptual framework **(b)** Computational Framework

Figure 6.1: Overview of the multiscale modelling and simulation framework described in this thesis.

of the domain. A single scale model is then described by a fixed submodel execution loop and by coupling templates based on the scales of the participating submodels. These constructs allow modelling a multiscale simulation, a meta-model as it were, for example by constructing a task graph of a multiscale model. The coupling topology, the way that submodel instances are coupled, has a number of properties that determine its complexity: being cyclic or not, having a dynamic number of instances and a dynamic number of iterations. At later stages, this thesis focusses on cyclic coupling topologies, with a static number of instances and a dynamic number of iterations. This is the most complex case that was handled.

The architecture of a multiscale model is described with the Multiscale Modelling Language (MML). The XML format of MML, xMML, can be used as an input for a task graph, a simulation, a basic model implementation, or a graphical representation of the model. An XML schema and output tools were created as part of this thesis (see Appendix D), and web interfaces Mapper Memory (MAME) and Multiscale Application Designer (MAD) were created by ACK Cyfronet AGH, Kraków, Poland in collaboration.

The theoretical framework provides a specification for an implementation, but multiple implementations are possible. In this thesis, MUSCLE has been used as the library for implementation and as the runtime environment. Its main focus is local and

distributed computing of cyclic multiscale models. This idea was first implemented in the COAST project [143], originally by the name DSCL [65]. Putting DSCL to the test in realistic scenarios gave ample room for improvement, such as: portable installation; a succinct API; extensive documentation; support for MPI and OpenMP; support for heterogeneous clusters, supercomputers and their firewalls; well-performant communication; removal of deadlock conditions; asynchronous submodel execution; support for common programming languages like Fortran and Python; and complete and understandable logging and error reporting. The Java API was first slightly simplified and then slowly updated with the latest theoretical concepts. The C and C++ APIs were overhauled at the start of the development of MUSCLE because of their previous tight integration with the MUSCLE runtime system, and then kept stable. Because of the subsequent loose coupling between the API and the runtime system, the runtime system could be thoroughly improved without affecting users of the MUSCLE API. These improvements have been implemented in the duration of this thesis, and the performance of the current library has been thoroughly tested (see Chapter 3). The performance of MUSCLE is shown to be superior to file-based communication, GridFTP and DSCL, but slower than MPI.

Distributed multiscale computing, the rightmost box in Fig. 6.1, is showcased in Chapter 4 by example of ISR3D, a model of in-stent restenosis. It involves not only a runtime environment and a model, but also an infrastructure, execution coordinator, and user interface. This thesis shows how these elements can be combined – and that the elements do require some integration. The execution coordinator needs to know about the policies, firewalls and available services of distributed hardware, a user interface needs to expose available resources to a user, and the runtime environment must be able to accept commands from the coordinator. In the MAPPER project, these elements have successfully been brought together, truly enabling distributed multiscale computing.

The final test, the most important question, *should* we do distributed multiscale computing is addressed in Chapter 5. That multiscale simulation as proposed in this thesis is most useful is demonstrated by the number of models that adopted it: the separation of concerns that it provides is essential for any complex project. *Distributed* multiscale computing, however, depends on the characteristics of the models coupling topology, scale relations, submodel implementation and available infrastructure. The answer provided by Chapter 5 is: yes – situationally. We *should* do distributed multi-

scale computing if: 1) we have too few computational resources available locally and we have additional resources elsewhere; 2) submodels are sufficiently heterogeneous in their hardware or software requirements; or 3) computational needs differ sufficiently within a simulation. An exact equation for "sufficiency" is expressed in Ch. 5 for pairs of submodels.

6.1 Limitations

The models described in this thesis all have a fixed temporal granularity, regular spatial scales, and a static number of submodel instances. This means that limited experience has been acquired with reactive submodel interaction, dynamic domain decomposition, and dynamic work distribution.

First, reactive submodel interaction is useful in distributed discrete event modelling, where interaction between submodels is not governed by a time step but by direct reactions to each others events. In this type of modelling, heuristics have to be employed to avoid or revert temporal inconsistencies [54, 77, 121]. Not only does this introduce theoretical difficulties, such as the need for discrete events with infinitely fine temporal granularity, submodels that are completely dependent on each others events invalidate the assumption that submodels are largely independent – which is the basis for the conceptual work and multiscale modelling at large. Multiple solutions to fit these event-based models exist, each consisting of a weakening of the discrete event paradigm. The first solution is to set a minimal time difference between events. This way, the time scale granularity is bounded. In practice, the minimum time difference means that the submodel execution loop can rely on taking small steps to accommodate the timing of the events. The second is to limit the impact of events to within submodels, and exposing a regular temporal granularity to other submodels. In practice, this means applying a temporal conduit filters to all conduits of an event-based model. These filters may aggregate and interpolate between messages to provide a steady stream of messages for other submodels. Finally, if the number of event-based submodels is limited to two, the time of the next event can be communicated between them. This means that a third submodel cannot inject an additional event that will change the times between two messages, invalidating the submodel execution loop.

Second, regular spatial scales mean that the domain size of a model is going to be the same during a simulation. For many submodels, the problem size increases with

the domain size. Having regular scales means that their problem size stays about the same, and in turn the machines that are able to run the submodel will be sufficient during the entire simulation. In this work there was no need to change a machine during simulation, or start and stop a submodel to use more processor cores, or in general, to do dynamic domain decomposition. Domain decomposition, splitting a domain into multiple submodel instances, is possible in theory and in practice, but in practice it must be performed in advance and remain static during a simulation.

Finally, in dynamic modelling approaches like stochastic computing and heterogeneous multiscale modelling (HMM) [44], the number of problems to be solved may depend on the specifics of a macro-scale model, like having an irregular geometry. The parts of the geometry of interest may be modelled in micro-scale models. The number of micro-scale model instances depends on the specific geometry, which may depend on the course of the simulation. Although these types of models are accounted for in the theoretical framework, none of the software solutions mentioned in this thesis do, except by disregarding the assumption that a submodel instance has a fixed domain. Concretely, a submodel instance may simulate multiple disparate domains, thereby acting as multiple submodel instances. A mapper may assign a variable number of domains to a fixed number of submodels, thereby alleviating the need for a variable number of submodel instances.

In the conceptual framework, a dynamic number of submodel instances can only be coupled to a mapper that knows about the number of couplings. This is severely limiting to submodel instances that form a network. The conceptual problem with coupling a dynamic number of submodel instances between submodels, is that the framework does not assign a way to couple conduits between ports without being predefined. Network and graph models have ways of solving this, either by adding a centralised authority deciding on which nodes should be connected, or by adding some way for submodels to explicitly recognise each other. These methods are deemed out of the scope of the multiscale modelling and simulation framework and this thesis.

Those are the main limitations of the theory and the runtime environment. Still, both theory and software are applicable to the multiscale models described in Chapter 5, and to a large class of multiscale models that originated from a single scale or bi-scale model.

A more practical limitation to this work is the infrastructure. The European computing infrastructure, eInfrastructure, is a complex federation of clusters and super-

computers, consisting of heterogeneous software, hardware, and services, and governed by a variety of access and security policies. Moreover, the infrastructure is dynamic, limiting the lifetime of any given framework to a few years. The only way to keep using a framework, is by co-evolving it with the infrastructure. Moreover, in a set of interacting software components, the components must keep in line with both the infrastructure and each other. It is therefor imperative that software developed in an infrastructure project, such as MAPPER, can also operate independently, so that it is still usable if one of the interacting components fails to keep up. And, a framework as a whole that aims to outlive its project will need dedicated efforts for each participating component. The solution of distributed multiscale computing proposed in this thesis therefor also has a limited lifetime. The theoretical basis is steady, but xMML, user interface tools, the runtime environment, the coordinator and user interface form a fragile assortment. Fortunately, the QCG software stack, MUSCLE, and xMML are usable for multiscale modelling and simulation in isolation. In fact, having only MUSCLE already makes it possible (but not necessarily easy) to do distributed multiscale simulation. Also, the framework as it is Chapter 4 will be reproducible for only a while, but the setup in terms of functional components is generally applicable.

6.2 Directions for further research

Distributed multiscale computing and multiscale modelling are progressing fields, and many aspects are not yet well developed or understood [71]. Two theoretical problems about multiscale modelling strike me as most urgent: can the effect of single scale models on each other be described in a general way based in their scale relation, and can scales be extracted from a phenomenon in an objective and accurate way.

First, single scale models may have different kinds of interactions, like reinforcing or diminishing each others dynamics. Moreover, a fine and coarse scale model with the same domain must be correlated in some manner. In the dynamics between single scale models, phenomena like error propagation or emergence from fine to coarse scales are not generally understood. A current effort to describe the effect that single scale models have on each other is by using information propagation and processing as proposed by Quax et al. [120]. It remains unclear whether a general solution can be found for this question or that each solution will be largely based on domain-specific and ad-hoc methods.

Second, there are a limited number of explanations of why we observe scales. One of these explanations takes the complexity profile as a measure Bar-Yam [13]. It divides a domain into a number of same-sized subdomains, and calculates the entropy of the domain with the subdomains as elements. It repeats this process for a range of subdomain sizes, creating a complexity profile. According to this method, the subdomain sizes where the profile peaks, and entropy is highest, are the characteristic scales of the system. In other words, the scales of a system are the subdomain sizes that explain most of the system. This algorithm scales factorially, however, so it is not a tractable method. A successor to this method scales polynomially but it does not detect strongly emergent scales [14]. In addition, variants of these methods are only descriptive so they do not explain *why* a certain size explains a lot of the system. A clear physical understanding of this question would be of great benefit to multiscale modelling. Methods developed to this effect may yield good advances in understanding the nature of scales, and how objective they are.

Computational questions include how well the task graph is suitable in practice for the scheduling of distributed multiscale simulations, given the time that it takes a node in the task graph to start. Moreover, this question could be explored using stochastic times of execution and queuing.

A few requests have been made to make MUSCLE suitable for dynamic work distribution. Since this requires a significant interaction with a resource manager, it would need to cooperate with a component that knows how to interact with different resource managers, including cloud infrastructure. A dedicated effort in that direction would be necessary for success.

With this work, the theory on how to characterise multiscale models is well-developed, and distributed simulation software is available, leading to potentially large performance gains; a complex approach for demanding multiscale models. As long as research groups and infrastructure providers continue to develop and support distributed multiscale computing, it will be an attractive paradigm.