Distributed multiscale computing

Borgdorff, J.

1 Introduction

In daily life, we experience a limited part of the world. We cannot distinguish anything smaller than 0.1 millimetre or compare our length with an object of more than a few metres. Likewise, reaction times start at 0.1 seconds and our lifespan is bounded to 80 years - although our attention and reasoning spans are considerably less. This is *our* scale, the human scale. Even at these scales, the human race needs instruments like rulers and clocks to have a common understanding of length and time.

Imagine the shock when Pasteur discovered that organisms a lot smaller than us were causing our diseases. We were simply not able to reason intuitively about these new bacteria - but after the discovery a lot of epidemics started to make sense. On an even smaller scale, Ancient Greek philosophers imagined something like an atom should exist, but their inability to do any measurements on that scale lead to the concept of an elemental atom. And although modern atomic and even quantum theory have been accepted for a while, the scales below that (string theory? cause of gravity?) are completely abstract and not at all certain. Distances over 10,000 kilometres and forces larger than experienced in a roller-coaster become unimaginable and hard to

evaluate - but they are essential to any meaningful astrophysics.

We have the measuring instruments and theory to reason about scales far from our experience and the computational resources to do the arithmetic. For many phenomena in physics, a mathematical model was around far before it could be simulated. Since the computer was invented, these mathematical models have been discretised so they can be simulated on our binary computers. Current computers are powerful enough to calculate many of these models with arbitrary precision – often more precision than the accuracy of the available data justifies.

The converse phenomenon appears when we understand, more or less, what rules govern a system on a fine scale, but to execute those rules for a interesting time and space scale would require far too much computing power. Take molecular systems, where we have an exact model for the forces between parts of molecules. For example, although H_2O molecules can be simulated in small quantities, a realistic volume and composition of flowing water simply cannot be computed by its H_2O molecules. For most circumstances, however, more efficient hydrodynamics models are used, taking into account only the flow of the water and not the H_2O molecules.

By combining models on different scales, a coarse scale model gains accuracy by informing certain detailed dynamics with a fine scale model. The coarse and fine scale models can be validated at their own scale, usually with data that is collected with instruments catered for that scale. By excluding certain fine dynamics from a coarse-scale model, it can be computed faster, and by judicious use of fine-scale models, additional computations are done only where they give more accuracy.

Now as simple as it may sound to just use different scale models together, combining or coupling models in a physically realistic way is a complex challenge in itself. Although the models might be validated individually, their combination is not; and even though they can be computed in combination, this requires a suitable software and hardware ecosystem.

Europe has a large and fast network of computer centres, some aimed at performance, others at quantity, and several European projects strive for interoperation and connectivity between those computers. In this infrastructure it makes sense to choose the most suitable computational resource (a supercomputer, cluster, or single computer) to run a simulation on. And since a multiscale model is composed of models at different scales, it is worthwhile to explore whether those models should be run at a single location or distributed over multiple resources.

At this point we arrive at the thesis' title: distributed multiscale computing. *Can* we distribute simulations of multiscale models, and more importantly, *should* we? Of course, this dissertation also covers the *how* part of the question. The fact that this dissertation was defended, and taking into account a positive results bias, obviously means the answer to the first part of the question is yes. Whether we should do distributed multiscale computing turns out to be more subtle.

In this dissertation I will start by clearly defining what a multiscale model is and what type of couplings can be used between scales. Having a clear definition, the computational requirements of different parts of the multiscale model will be specified in an unambiguous way, a necessity for any kind of general software ecosystem. I will show that this specification can be used for modelling the performance and execution of a simulation, and for actually running the simulation. Running a simulation is aided by a user interface, middleware for executing software on distributed computer systems, and a runtime environment for the model to be implemented in. A multiscale model of in-stent restenosis is taken through this process as a proof of concept. The performance of other applications that took this approach is measured, giving an indication of when distributed multiscale computing is beneficial.

1.1 Multiscale applications

Multiscale modelling has seen a surge in recent years, measured in involved projects [62], and e.g., papers from mathematics [56], biology [41, 125, 127] physics [44, 50, 82, 118], engineering [50, 75, 76]. Listing all papers would be unproductive, just enumerating them is a feat in itself [62]. Taking out application-specific details, these papers have similar concerns: how to couple scales without introducing too much error and how to compute the resulting multiscale models efficiently.

The type of models ranges from single small and integrated mathematical models to large model repositories maintained by multidisciplinary consortia. In the former, the multiscale model can follow from mathematical functional decomposition, by scaling part of the function up or down. In the best-case scenario, the gains and errors introduced by making a multiscale model are then analytically solvable. Frequently though, models on different scales already exist and need to be coupled rather than decomposed. Neither the mathematical nor the computational methods to do this are straightforward, as the models may use different formalisms, such as continuum mod-

elling versus grid-based modelling, and the codes may be implemented with different programming languages, libraries, or tools. This type of model often uses an ad-hoc solution for the mathematical and computational couplings. In large projects, consortia, or communities, usually the partners involved agree in advance on the scientific, mathematical and computational methods that will be used. This allows interoperation of models from different partners, but may hinder progress if the methods are ill-fitted or not thought out.

An example of purely mathematical multiscale model is the reaction-diffusion analysed in Caiazzo et al. [31]. This model consists of a pre-defined large scale reaction and of local diffusion, and it could model, for example, the effect of large external heat sources and sinks on a poorly conducting homogeneous material. When considering that the reaction (heating up the material locally) is much faster than the diffusion (conducting the heat nearby in the material), it turns out that these processes can be computed independently, after which the result is applied to a the same domain. And since the reaction time is lower and the time scale of the reaction is thus finer, this saves many calculations for the diffusion process, since it only needs to be applied every so many reaction time steps. The error that is introduced by reducing the number of diffusion time steps can then fairly accurately be estimated.

Another example is the heterogeneous multiscale method used for advection-diffusion problems with highly oscillatory coefficients [3]. Advection-diffusion processes are found in any moving liquid or gas, and are of interest for example in building aerodynamic cars or efficient engines, in understanding blood flow or magma movement, and in processing food, oil, paint, or concrete. Without going into details, the model uses the framework of the heterogeneous multiscale method (HMM) to give accurate results and error estimates. HMM in general focuses on well-defined macro-micro couplings, by computing variables needing little detail on a coarse scale and variables that (perhaps locally) need more detail on a micro-scale [4, 44, 94].

The previous models have a regular description and their properties are fairly well understood mathematically and physically. In current multiscale blood flow models that take into account red blood cells or platelets, for example, these kinds of properties are more experimental. In the simplest case (and this is not simple at all), cells in the blood are modelled as point particles with a mass and velocity. To do this fairly accurately, for example the following three scale model was proposed [48]: on the coarsest scale, blood is treated as a simple fluid and simulated with the Navier-Stokes

equations, which generates a velocity field over the domain; on the finest scale, the particles exert some repulsive and attractive forces on each other and on the boundary; and in between, a mesoscopic scale calculates the interaction between the particles and the fluid. The error introduced compared to the blood flow in a living blood vessel cannot easily be established, but it can be compared with certain mathematical models that model some of the same properties, such as Pouseuille flow and Couette flow.

Blood modelling is also being done using the immersed boundary method [85, 107, 108], which models boundary of a cell with certain forces and lets it interact with a fluid dynamics model [116]. However, these methods so far require both cells and fluid to be calculated throughout the entire domain, limiting the size of blood vessels that can be simulated. Indeed, this is a good reason for decomposing a model into multiple submodels: being able to define where to use a certain accuracy by providing a more complex cell model, and using coarser and faster submodels elsewhere.

In the European MAPPER project, six medium to large multiscale applications instigated the need for distributed multiscale computing. The models came from the disciplines nanomaterials [132], fusion [53], biomedicine [60], hydrology [114], and systems biology [135]. These are described in Chapter 5 and in Appendix C, and one of these, a multiscale model of in-stent restenosis (ISR3D) will be handled with more detail in Chapter 4.

The model of in-stent restenosis, ISR3D, regards a porcine coronary artery applied with stent-assisted balloon angioplasty [32, 46]. The main goal of the model is to understand how a blood vessel responds to a stent, and most importantly why its response results in a stenosis in certain cases. Since this process is not completely understood, and even the key drivers of the process are not irrefutably identified [79], a model of the process will test hypotheses of likely key factors [137, 139]. In ISR3D, the growth of tissue by proliferating smooth muscle cells is taken as a driving force of the stenosis. The growth is assumed to be inhibited by the shear stress exerted by blood flow and by the drugs the stent is coated with, if any. The multiple scales are found on the temporal axis, since the proliferation of smooth muscle cells is perceived at a much coarser time scale than changes in shear stress from the blood flow: in the order of hours to months, and in the order of microseconds to a second, respectively. To make the model truly multiscale, we assume that the stresses from the blood do not change as long as the blood vessel has the same geometry and assuming that the only changes in geometry are caused by proliferating smooth muscle cells. The result is that

the blood flow needs to be recalculated only when the smooth muscle cells proliferate, rather than continuously during the runtime of the model. Since the assumptions are simplifications of the actual process, we introduce some error, and since the model is complicated and there is no reference model to compare it with, this error can only be roughly estimated. This is a problem for modelling of complex problems in general, and exacerbated by linking multiple complex models together. Understanding what error is actually propagated through such a multiscale system will be a key problem in the coming years.

Large-scale modelling initiatives try to specify in advance what the error of certain models is, given certain circumstances. Examples are earth systems modelling by the ESMF[1] for weather and climate prediction [8, 11], heart and organ modelling by CHASTE[2] for a deeper understanding the human body [115], astrophysics modelling by AMUSE[3] [118] and modelling tokamak reactors for fusion as organised by the EUFORIA project[4] [53].

Each of these projects first need a common understanding of how they want to model different parts of an overall system. Without such an understanding, partners will not be able to combine their efforts, or worse, they will combine models but with incorrect results. When their models can theoretically be combined, the implementations of the models still need a shared computational technique for communicating. First, the data types and units need to match, and second, the transport media needs to be compatible. The projects listed above solve this by implementing their own computational framework, which is their uniform computational mechanism for communication. Often, this framework is designed specifically for the problem it means to solve, and combines only with the tools that their community is currently using. In the context of one project that may be efficient and clear, however, it is obvious that this practice introduces redundancies between projects. Moreover, as a project progresses, a framework can become too specific, raising the bar for other partners to join. There is no doubt that having such a framework does solve computer science issues that domain experts have no interest nor expertise in, resulting in a clear division of labour. In the smaller multiscale models outlined above, such a division is usually not present and modellers need to possess very good computational skills in addition

[1] URL: `http://www.earthsystemmodelling.org/`
[2] URL: `http://www.cs.ox.ac.uk/chaste/`
[3] URL: `http://amusecode.org/`
[4] URL: `http://www.euforia-project.eu/`

to their domain expertise.

Communities like those in the EUFORIA project have been doing simulations on supercomputers for decades and as a result, that is the age of some simulation codes [53]. These codes are well-tested, efficient – and very complex. To get new results out of them, they may be coupled to other codes, while running them on a computing architecture compatible with the code. To avoid introducing errors in the code, it should be modified as little as possible. In the past decades, the ease of use of programming languages and libraries has increased, as well as automatic optimisation techniques of code, but these do not integrate well with old codes. Also, new codes can be optimised for different computer hardware architectures. These type of applications put specific demands on any multiscale framework: to be able to reuse and integrate old codes, and to be able to use multiple hardware architectures in a single multiscale model.

1.2 Theory of multiscale modelling and computing

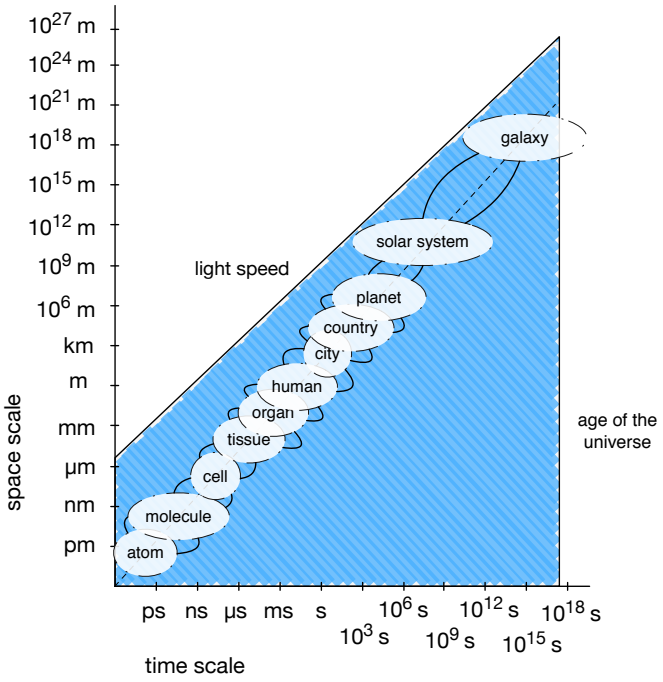
Given the output from these large projects, as well as resulting from dedicated efforts, there are a number of computational frameworks to do multiscale simulations. Because of the limited overlap in their theoretical underpinnings, these frameworks are generally not compatible. A number of theoretical frameworks have been created to fill this gap. Some are categorisations of existing models [75, 76, 90, 141]. These categorisations observe at typical macro-micro couplings, with a model with fine spatial and temporal scales coupled to a model with coarser scales. They then focus on how these models tend to be computed, in parallel, as separate programs, sequentially, or a combination thereof.

Others contain more fundamental multiscale modelling theory [22, 28, 44, 68, 70, 151], based on possible multiscale phenomena and their interactions in nature, and how these could be implemented as multiscale models. These theories each separate phenomena by their scale, and assign special importance to the interactions between the scales. Hoekstra et al. [68] treat scales as properties of abstract Cellular Automata, and their combination as so-called Complex Automata (CxA). The grid size and fixed time steps are used as the scales of the system. Complex Automata are the basis for work in Borgdorff et al. [22], in Chapter 2, which generalises the models of the scales beyond cellular automata while maintaining a fixed computational structure. In both

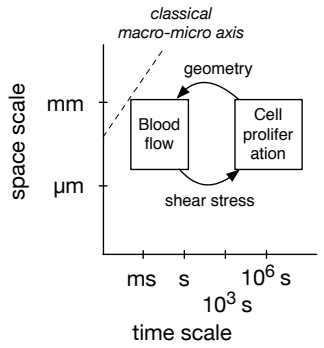
theories, the interaction is captured in coupling templates. Yang and Marquardt [151] approach multiscale models from a ontological perspective, yielding hierarchical multiscale models with pre-defined laws to which interaction must adhere. Brandt [28] proposes a method for deducing the scales from mathematical equations that are used in a large system, and therewith gives an approach to make a model multiscale, rather than directly modelling a multiscale system. Finally, E et al. [44] focus on macro-micro models, combining a model with macroscopic temporal and spatial scales with a microscopic one. The approach is quite specific and therefor open for thorough analysis [4], but it cannot be used for all systems.

All of the approaches above deal with the concepts and practicalities of modelling and simulations, not with the question of whether scales are inherent to a system or purely the modellers choice. For physics and philosophy, it is an essential question. If scales are inherent to nature, that would mean that atoms, life, and the universe are necessarily confined to certain sizes and timespans – the mathematical, continuous, infinitely scalable universe that many physics equations describe could not exist. Interpretations of quantum mechanics and the speed of light hint that this might indeed be true, and of course there are enough detractors to such a controversial theory. This is the domain of digital physics [156]; see [128] for a philosophical review. Although I am deeply interested in the subject, it is not the subject of this thesis, and I do not think the issue can ever be proved either way.

To visualise the scales of a system, the scale separation map was introduced on several occasions [41, 68, 76], and informally used in numerous applied papers. For example, Figure 1.1 shows the classical micro to macro axis, scaling from the atomic scale to our galaxy. For many processes, the temporal and spatial scale *are* positively correlated, for instance, in many systems atoms have much faster dynamics than molecules. However, for instance when modelling smooth muscle cell proliferation in an arterial wall, coupled to the velocity and shear stresses of pulsatile blood flow that it receives from the human body, scales are diametrically opposed. The blood flow, modelled using the forces that the heart produces and propagates through the system of blood vessels, has a fine time scale (microseconds up to a second) but a coarse spatial scale (centimetres up to a metre) whereas smooth muscle proliferation has a fine spatial scale (micrometres up to a millimetre) and a coarse temporal scale (hours up to months). In an example from astrophysics, models of two bodies with a large mass and some velocity that exert gravity on each other, like a planet and a moon, go through stable



(a) Classical micro to macro axis



(b) Micro to macro axis

Figure 1.1: In (a) is the classical example of the universe as a micro to macro system; it is not accurate and does not distinguish between the granularity and total size of a system, and it represents objects rather than concrete processes with a time scale. In (b) is a simplified model of a segment of a coronary artery, combining cell proliferation in the vessel wall and the shear stress that the blood flow induces, covered in more detail in Chapter 4; it does not match this classical axis.

motions and do not require a fine time scale to extrapolate their movements. However, when increasing the problem size to three bodies, suddenly the model becomes chaotic and a very fine time scale is needed. For both models, choosing the correct time scale is necessary to efficiently calculate an accurate result. A scale separation map visualises which scales are relevant for a given model, and whether these scales are significantly different, i.e., separated. Edges between the phenomena on different single scale models indicate what kind of interaction the scales have, making the model truly multiscale.

Closely related to the term multiscale, multilevel assumes that each level is built up from the level below it [151]. 'Multilevel' distinguishes the coarse and fine representations of a system but it does not necessarily associate them with a unit. The idea behind the term multilevel is clearer than that of multiscale: a level is made of a number of systems, and the combined dynamics of elements on a level is different than the dynamics of the individual elements, yielding one element on a higher level. Using multilevel modelling it clearly delineates the subsystems and restricts the possibility for interaction, but it hampers interoperability between other systems (including real-world data) since there are no physical units to compare two systems with. This view may be contrasted with the one posed by Li et al. [90], which describes a multilevel model as composing a number of submodels with a scale (and thus with a unit) into structural levels. This makes the system comparable with others, and using levels provides a way of bringing structure into a set of models. Moreover, both approaches to multilevel modelling assume a hierarchical view of the world that will not be found in many natural systems. Finally, Ryan [122] links scale, level and emergence by posing that the level is not the distinguishing factor for having emergence, the scale is (combined to the resolution and scope). If the inherent property of a scale would be that it has emergent properties different from the scales finer than it, this would certainly be an interesting argument for using the term multiscale.

To model the interaction between scales, researchers in many fields try to define standard scale bridging methods: mathematical descriptions of how to transform information on one scale and project it to another scale. However, by focussing on a single discipline, some of these methods only reinvent or rename existing methods from another field. Through dialog, and based on analysis of the function of these scale bridging methods, the multiscale modelling community would benefit from creating a consensus on their naming and purpose.

Table 1.1: Terms used for scale bridging.

fine-/coarse-graining	filtering	time-splitting
up-/downscaling	homogenisation	amplification
projection	splitting	condensation
lifting	micro-macro methods	constitutive models
interpolation	refinement	restriction
prolongation	averaging	injection

An informal survey held at a Multiscale Modelling and Computing workshop[5] among specialists from different disciplines yielded a range of coupling operators, listed in Table 1.1. For example 'upscaling' and 'downscaling' are used by Brandt [28] while their equivalents 'coarse-graining' and 'fine-graining' are used in Complex Automata theory [68, 70]. This thesis does not intend to clear the matter up, but it will restrict itself to the terms coarse- and fine-graining. Coarse-graining means translating data on a fine scale, with a fine resolution, to a coarse scale, with a coarser resolution, fine-graining means the opposite. The terms 'small-' or 'large-scale' are avoided, since these refer to the total size of a system and not to its resolution – the usual interpretation of modelling at a scale. The relation of up and down between scales is ambiguous; on the other hand, so is the notion of a grain in continuum models. I will use the latter nonetheless, since without a resolution (grain), the term 'scale' becomes rather meaningless[6]. Other terms in the table, such as 'interpolation' or 'averaging' have a scale bridging function besides describing a relation between scales. Since their meaning is generally accepted, the application to multiscale models is less ambiguous.

Having identified the need for models with different scales and for scale-bridging methods, there are two solutions to making this possible. The first is integrating the models with the scale-bridging method into a single methodology and model. The second is making models and scale-bridging methods into separate communicating entities. This approach amounts to component-based modelling [8, 55, 92, 146]. The frameworks mentioned in the previous section all use this approach, as do many soft-

[5] Multiscale Modelling and Computing Workshop at the Lorentz Center, Apr. 8–12, 2013, Leiden, The Netherlands. Related special issue *Multiscale systems in fluids and soft matter: approaches, numerics, and applications* pending at R. Soc. Phil. Trans. A. URL: `http://www.lorentzcenter.nl/lc/web/2013/569/info.php3?wsid=569`

[6] I *could* introduce the terms 'resolving'/'zooming in' and 'expanding'/'zooming out' to complicate matters further.

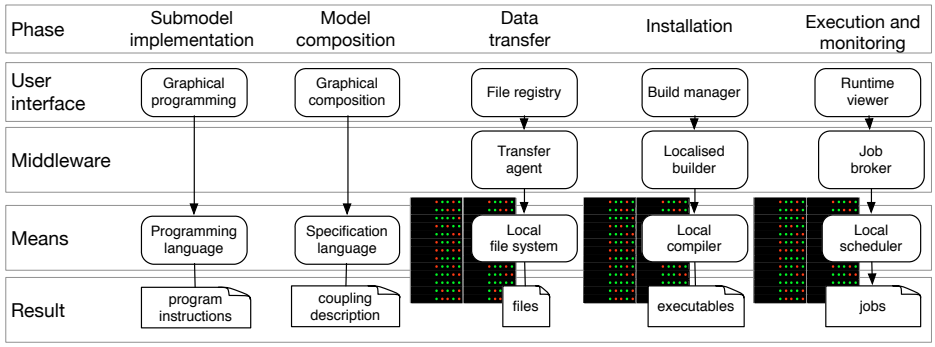


Figure 1.2: Ecosystem for high performance simulation. A user may interact with a graphical user interface, the middleware or directly with the means of doing a task, in order of increasing complexity.

ware in the next section. This thesis uses the same approach, and a multiscale model is therefor defined as a set of coupled single scale models.

Naively, a component-based model can be translated directly from a scale map. However, the scale map does not include information on how to convert data from different scales, nor how many processes should be simulated on each scale. Finally, no information about the computational aspects of each model is available. Of course, these aspects can be manually implemented in the code and externally documented. Another possibility is to include scale information in an existing computational description of the model. An integrated approach is proposed through the Multiscale Modelling Language, described in Section 2.3. In this language, components with their scales are well-defined, as well as the type of data to be transferred and the scale bridging components used. Using the description, models can be analysed and executed, as described in Chapter 2.

1.3 Multiscale computing

To do a simulation of a mathematical model on a high-performance computer, a few steps have to be taken, as per Figure 1.2. First, the submodels for each scale must be implemented, and then coupled. Then any input data and files for the simulation need to be transferred. The model code also needs to be compiled, taking into account the target architecture. Finally, a job that that runs the simulation will be started and

afterwards the output data can be collected.

Some types of single scale models, like specific types of physics equations or protein interaction networks, can be defined in a graphical user interface created for this purpose. This tool will generate program instructions. When a tool is not available or it does not cover all aspects of a problem, the user has to choose a programming language to implement the model with. This choice may be determined by the programming experience of the users and their peers, the languages expressiveness, the availability of domain-specific libraries, and its performance. Although it is practical to use the same language for all single scale models, some models may strongly benefit from using a different programming language.

The coupling of the submodels may be done within the submodel code, or defined externally through handover points. A frequently used method is simple file-based communication, since files are ubiquitous and it places no demands on the programming language that is used, or on the time intervals that codes are running; and the files can be viewed in any file browser. Communication via files is, however, relatively slow. Another method is tight and direct integration, for example using shared memory or MPI communication. This requires all submodels to use a compatible language, and all submodels need to be active simultaneously for communication to occur. Moreover, it binds a multiscale model to a single cluster or machine. A slightly more loose coupling is one using TCP/IP, the protocol used over the internet, since it places no requirements on the programming language and can communicate to the outside world. This method is slightly slower than MPI or shared memory, it also requires simultaneously active submodels, and it is potentially insecure since the communication can be intercepted. Because of this last caveat, many supercomputers and cluster place firewall restrictions on this type of communication.

A user may benefit from choosing a runtime environment for coupling a multiscale model, since many component-based runtime environments already exist and they solve many known communication issues (see Sec. 3.1). Usually, the runtime environment determines the type of communication between submodels that will take place, and it will have a internal representation of the couplings. In Chapter 3 the Multiscale Coupling Library and Environment 2 (MUSCLE 2) is presented, which has support for a number of programming languages and uses TCP/IP and shared memory communication for the coupling.

Many countries in Europe have an organisation with a supercomputer, and most

universities and some research groups additionally have some clusters of computers. Although each supercomputer and cluster is maintained by a local organisation, many supercomputers in Europe and the fast networks between them are federated in the PRACE project, and clusters are federated in the EGI project. Although the federated organisations and their computers differ, these projects mandate a minimum level of compatibility, connectivity, and service.

Scientists can apply for computing resources directly at an affiliated organisation, usually their own university or research institute. Generally this is an application requiring little effort and reporting, and the awarded computing resources will have limited performance. Alternatively, with some more effort, they can apply for any European computing resources through the PRACE or EGI projects. Computing resources in these project are abundant and some yield high performance, and moreover, the computers are connected with fast networks. Because these computers are semi-public, the security is tight and resource scheduling policies are in place to provide a certain service level. For demanding simulations these projects are useful - but individual users need to estimate if the resources are worth the cost.

All computer systems that are used in this thesis have a scheduler to divide computing resources between users. Modellers are required to describe their simulation by the program that it runs in and the amount of resources that it needs; this is called a job. The scheduler decides on which computer to actually run the program within a cluster or supercomputer. If too many jobs are already running, the job is put into a queue for later processing. The scheduler assumes that the program and any input data is locally available, that the output data will be collected after the job finishes, and it will stop the job if the program generates an error. Computers and clusters that are linked together, have similar software environments, and can be used by the same scheduler, are also known as grids [52].

To make use of the grid in a high-level way, several tasks can be composed into a directed graph, called a scientific workflow [15], and these tasks are then scheduled to wherever on the grid computers are available. Coupling descriptions made with the Multiscale Modelling Language can be converted in such a graph, to make use of the scheduling heuristics and software already available. Without going into the details, scheduling jobs is a hard problem [38, 45, 87, 99, 119, 152, 153], and an optimal schedule takes exponential time to find[7].

[7] Unless, of course, P = NP.

Interacting with the scheduler and passing the security measures can be tedious so several so-called middleware have been created to aid users in these steps. Middleware can usually distribute and collect data and programs and authenticate themselves on behalf of the user. Some also act as meta-schedulers by directing schedulers on multiple systems, or perform advance reservation of computing resources. These possibilities are limited by the policies and functionalities offered by the actual clusters.

Globus [51] middleware is installed on all clusters from EGI and PRACE, and it is a solid and secure basis for users to rely on. Because of its widespread installation base and many stakeholders, it needs to compromise on many aspects, and can be slow to get new functionality implemented. It is nevertheless a solid base that other middleware and user interfaces make firm use of. It can transfer data and run jobs, and it is actively supported by system administrators. A venerable low-level middleware Condor [91, 142] is used to schedule jobs on grids and is also the basis for some workflow systems, but is not a standard in EGI, and will not be used in this thesis. Smaller middleware may provide additional functionalities like flexible co-allocation and advance reservation, in the QCG software stack [6, 84], but it needs to be installed on a site-by-site basis. The Application Hosting Environment [154] can upload an application to a designated site and run it. It is designed to make middleware more user friendly rather than expanding on their functionality, and it includes a graphical user interface.

While the middleware interacts with clusters and supercomputers, a user interacts with a possibly separate program, the user interface. The user interface may provide an easy means to direct middleware and to configure and track simulations. It may be web-based, like GridSpace 2 [39], a desktop program like Kepler [95] and Taverna [150], or complete virtual machines, like SimFlowny [10], the Application Hosting Environment and GridSpace 1 [39]. They can be completely integrated, including middleware, simulation configuration and tracking, like Kepler or Taverna, or be a tool to just start a simulation, by cooperating with other tools, like GridSpace. Integrated user interfaces, that handle composition of simulations, data transfers and executions are also called scientific workflow suites [15]. The benefit for users is that all information about a simulation is centralised into a single tool. However, it may also lead to lock-in, in the sense that it can be hard to move away from such a tool if it misses certain features or is no longer supported.

1.4 Distributed multiscale computing

Clusters and supercomputers have different specialities. Most clusters focus on computing many jobs on relatively cheap hardware, some have a focus on general purpose GPUs, others on large amounts of memory or very fast data retrieval. Supercomputers focus on computing jobs as fast as possible, by providing a very large amount of fast multi-core CPUs with fast connections between them, in combination with fast data retrieval and storage, and large amounts of memory. The number of CPU-cores, or simply cores, determines how good programs must be at parallel processing. Programs that use supercomputers need to prove that they are capable of scaling to the number of cores that these computers offer. Indeed, with a multiscale model, it would be ideal to compute the submodels that scale well on a supercomputer, and others on a small cluster. Also, a single cluster may not provide enough computing power to calculate an entire multiscale model. These observations form the basis for distributed multiscale computing.

A limiting factor for doing distributed computing is the ratio between communication time and compute time. Fortunately, networks between large compute centres are generally fast, but if communication happens very frequently or in very large volumes, communication time will dominate and a distributed simulation will actually run slower than a local one. In fact, this is a general limitation of component-based modelling: since the communication between components happens at pre-defined handover points, there are fewer opportunities to tightly integrate the data of two components, thus potentially increasing communication time.

Even if distributed computing decreases the time to finish a simulation (its makespan), access to a cluster or supercomputer is usually associated with a limited allocation of compute time per user or project. This compute time is usually measured in time used per compute core, using the unit core-hours or core-years. So a job using 32 cores during one hour costs the same as using one core during 32 hours: 32 core-hours. Decreasing the makespan of a simulation by using more cores locally or by using distributed computing, usually increases the communication time and consequently the resource usage, measured in core-hours. On the other hand, choosing the appropriate machine for each submodel may actually decrease both makespan and resource usage, since one submodel may benefit only from fast data access, and another only from a large number of cores.

This part of distributed computing is not new for component-based systems [142].

However, scale map and the multiscale modelling language provide a limited set of coupling types, with which we can predict the runtime properties of a multiscale simulation. Using the concepts scale overlap and scale separation, parallelism within a multiscale model can be exploited and a runtime system that minimises the resource usage can be chosen. This exercise was done for six multiscale applications in the MAPPER project in Chapter 5.

1.5 Outline of this thesis

This thesis investigates key aspects related to distributed multiscale computing.

First, what does the community mean by multiscale modelling and how can we accurately describe concepts in multiscale modelling. This description must take into consideration how a multiscale model will be computed. These topics are addressed in Chapter 2, by defining multiscale modelling concepts, applying them to the Multiscale Modelling Language (MML), and creating a runtime model of multiscale simulations using a task graph.

In Chapter 3, a runtime environment for distributed multiscale simulations, MUSCLE 2, is described, and the performance of MUSCLE 2 is tested. It deals with three questions: what obstacles does European e-Infrastructure present to portability and distributed computing, how can a runtime environment couple very heterogeneous submodels, and is the communication overhead low enough so that it feasible to do distributed simulations?

To run a distributed multiscale simulation, an ecosystem of theory, software, and hardware is necessary. In Chapter 4, a multiscale model of in-stent restenosis, ISR3D, is described and taken through that ecosystem, consisting of a model specification and implementation, user interface, middleware, reservation and execution. A preliminary test of its distributed performance is also presented.

Whether distributed multiscale computing is a viable paradigm, will depend on its performance. A full test of the performance of distributed multiscale simulations is done in Chapter 5. In it, six multiscale simulations are executed on local and distributed infrastructure, measuring their execution speed and the amount of resources consumed.

Software that was developed as part of this work is listed in Appendix D.