D2.1 Report on Existing Software Suitability and Adaptation

Due Date:	Month 12
Delivery:	September 30th, 2016
Lead Partner:	UvA
Dissemination Level:	Public
Status:	Final
Approved:	Executive Board: yes
Version:	1.0

This project has received funding from the *European Union's Horizon 2020 research and innovation programme* under grant agreement No 671564.

DOCUMENT INFO

Date and version number	Author	Comments
24.08.2016 v0.1	Alfons Hoekstra	Skeleton of report, first intro text.
31.08.2016 v0.2	Alfons Hoekstra, Saad Allowayd, Derek Groen	MMSF, MCP, and generic task graphs added.
08.09.2016 v0.3	Alfons Hoekstra, Saad Allowayd, Derek Groen	Update and streamlining of texts, adding improved pictures
12.09.2016 v0.4	Alfons Hoekstra, Peter Coveney, Derek Groen	Further updates and sharpening of text, new version of figures.
13.09.2016 v0.5	Derek Groen, Alfons Hoekstra	Adding news figures for MCPs, additional text for MCPs.
15.09.2016 v0.6	Oliver Hoenen, Saad Allowayed	Clarifications of texts, addition of paragraphs on RC.
25.09.2016 v1.0	Alfons Hoekstra	Updating and improving after feedback from internal reviewers.

CONTRIBUTORS

The contributors to this deliverable are:

Contributor	Role
Alfons Hoekstra (UvA)	PI and WP leader
Saad Allowayed (UvA)	contributor
Derek Groen (Brunel)	contributor
Peter Coveney (UCL)	contributor

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1 Executive summary

After a short review of the Multiscale Modelling and Simulation Framework and the vision of Multiscale Computing Patterns, this deliverable demonstrates how Multiscale Computing Patterns are expressed on the level of task graphs. A pattern is a generic task graph plus a set of rules on how to execute the task graph, based on performance of the single scale simulations, plus pieces of software that result in sufficient information (input files, configuration files) for the ComPat middleware. Generic task graphs for Extreme Scaling and Heterogeneous Multiscale Computing, and Replica Computing are introduced, and an example of an Extreme Scaling application is worked out, in terms of performance, showing several limiting execution paths. The idea is that the pattern automatically decides which computing path to pick, based on information available to the pattern.

2 Main body of the report

2.1 Introduction

The goal of Work Package 2 is, quoting from Part A of the DoA, to "create a general mapping from a multiscale model to a multiscale computing pattern. Reusable software components will be created for each of the three computing patterns. MML specifications of multiscale models will be modified to include these components, and the modified MML will be converted to input for the high-level tools (WP4) and middleware (WP5). The performance of the patterns will be tested and predicted."

This deliverable should then "Report on what software will be used in the project, how it compares to other software packages, and outline a detailed plan for their adaptation and integration, as well as report on MML implementation and status report on implementation of the multiscale computing patterns," and will be based on work performed in tasks 2.1 (Establishing existing software suitability) and 2.2 (Implementing a method for converting MML for ComPat) and 2.3 (Development of the multiscale computing patterns).

Establishing software suitability, task 2.1, has been performed in collaboration with WP5 and the results and conclusions are described in deliverable D5.1. We note however that the development of the patterns is in its initial phase, and that as we continue in their design and implementation, we may want to add additional software or require adaptation of existing software. However, at this stage we conclude that the software stack as described in D5.1 is suitable to meet the needs of the patterns.

Converting MML for ComPat, task 2.2, has partly been addressed, as will be described below. However, we have not yet reached definite conclusions as to if and how MML needs to be enhanced to express patterns, and their load balancing. We will therefore continue to put effort in this task, slightly deviating from the original DoA and extending this task into the second year of the project.

The development of the actual patterns, task 2.3, has been a major focus of WP2 in the first year of the project. Although the vision of Multiscale Computing Patterns (MCP) to enable high performance multiscale computing is now well established, actually defining where and how to express the patterns in the context of the Multiscale Modelling & Simulation Framework (MMSF) turned out to be less trivial than anticipated. Partners in WP2 have organized numerous brain storming meetings (over Skype) and many face-to-face meetings, resulting in the proposal, during the first All Hands Meeting in Month 6 of the project, to base the patterns on generic task graphs. The second half of the first year was then used to further work out this proposal, and this deliverable will mainly report on the current set of propositions and (partial) results. Our next step will be, in collaboration with WP3, to work out specific examples to test our ideas, and create first implementations of the MCPs.

Milestone2 (Month12): *Software suitability and adaptations determined, MML adapted* has been partly reached, in the sense that we have determined software suitability as discussed above, but so far we have not yet adapted MML. We believe that doing this is too early in the project. As will be discussed in the next sections, we do foresee adaptations needed for MML, but we expect to be able to identify and realise these required changes in the next year of the project.

This deliverable will first quickly review the MMSF, using the description that is also available in part B of the DoA, extended with some information on the concept of the task graphs. Next, for the sake of completeness, we will introduce the basic ideas behind the MCPs (again, mainly summarizing and updating what has been written in part B of the DoA, but sharpening the definitions and further clarifying their distinctions). Next, we will describe in some detail the notion of generic multiscale computing task graphs for MCPs, and some examples of this concept. Finally, we will discuss the next steps in WP2.

2.2 The Multiscale Modelling and Simulation Framework

The Multiscale Modelling and Simulation Framework (MMSF) is a theoretical and practical way to model, characterise and simulate multiscale phenomena. We have been developing the MMSF over the past years within the European projects COAST[1] and MAPPER[2]. MMSF currently comprises a 4-

1 www.complex-automata.org

2 www.mapper-project.eu

stage pipeline, going from developing a multiscale model to executing a multiscale simulation, see Figure 1. We describe MMSF detail in [1,2] and references therein.

First, we model phenomena by identifying relevant processes (single scale models) and their relevant scales, as well as their mutual couplings. This part of multiscale modelling is not addressed in this project (for this see e.g. [3]). The ComPat applications portfolio is already fully developed as multiscale models. Some of them are also realised and implemented in the MMSF, others are currently brought under the framework (for details we refer to deliverable D3.1).

The single scale models and their coupling are specified with the Multiscale Modelling Language (MML) [1,4], thereby forming the architecture of a multiscale model. It describes the scales and computational requirements of submodels and any scale bridging components needed. The applications in ComPat can already be described with MML, but we will further enhance the definitions in MML to cater for the intricacies of executing in HPC environments, and providing the multiscale computing patterns with sufficient details to be able to configure efficient load balancing and data handling strategies.

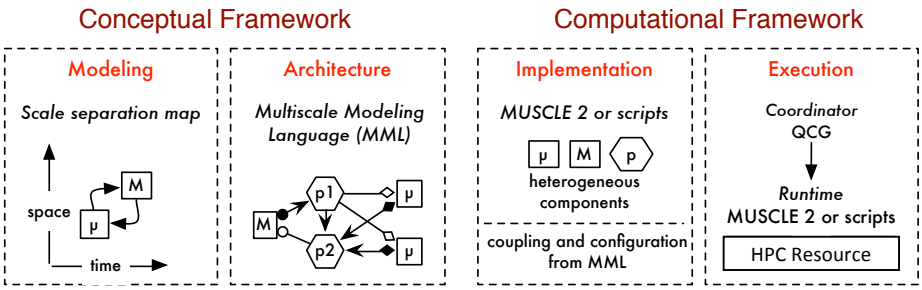

Figure 1: The MMSF pipeline consists of a conceptual part, where the model is coupled and specified, and a computational part, where it is implemented and executed.

A coupling library like MUSCLE2 [5] ensures that communication between heterogeneous components is possible, with minimal and local changes to the single scale code. In the last step of the pipeline, submodels are executed on a computing infrastructure. Each submodel may require different computing resources. Some may be massively parallel while others may require special hardware and software. In MMSF, the submodels can be distributed on several computers, without additional software development [6–9].

In an exascale environment, the way models are coupled will to a large extent determine the strategies to obtain highly efficient, fault tolerant, and energy-aware execution of a multiscale simulation. We will develop these strategies on the abstract level of multiscale computing patterns and include them in the stage-4 execution environment of the MMSF, ready for use by multiscale simulation executing in

'HPMC mode'.

An important ingredient of the MMSF that so far has not been fully exploited is the notion of task graphs for multiscale computing. Borgdorff et al. already introduced task graphs when specifying the foundations of the MMSF [1]. As shown in Figure 2, a task graph can be derived from an xMML specification of a multiscale application, and the task graph in turn can be used as input for scheduling software. We have demonstrated that task graphs can automatically be derived from xMML [1] and demonstrated the use of task graphs for one specific application [6]. For ComPat, the task graph will become a very important concept, as it will be used as the basis to create Multiscale Computing Patterns.

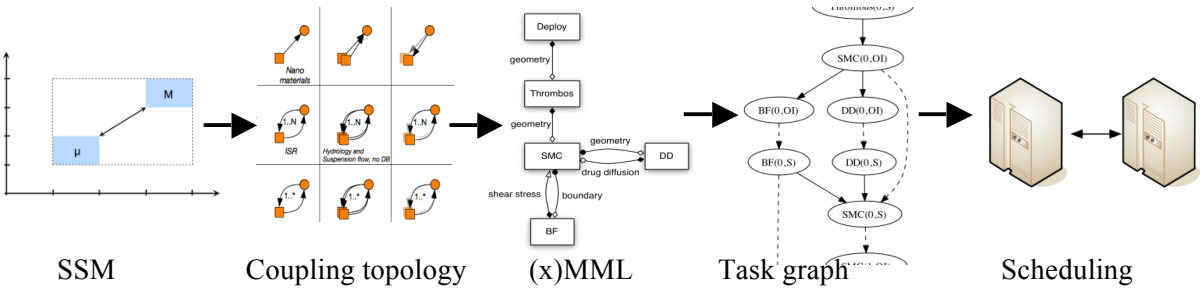

Figure 2: Several stages of description of a multiscale model in the MMSF, starting from the Scale Separation Map, details of the Coupling Topology are added, followed by a full specification in terms of xMML, from which a Task Graph is derived that can then be used as input to scheduling software.

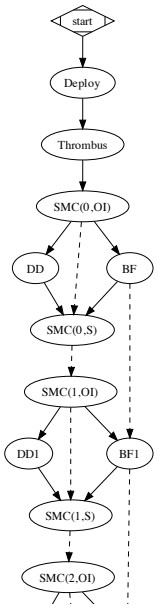

Figure 3: example of a task graph, showing the initialisation and first two cycles in the ISR3D model. Note that in the ISR application typically a few thousands of full cycles are performed.

Task graphs were introduced in the MMSF for deadlock detection, validity checking, and for estimating computational costs and scheduling. An example of a task graph for the In-Stent Restenosis application (see deliverable D3.1) is shown in Figure 3.

A task graph is a directed acyclic graph of tasks (the nodes) and their dependencies or data flows (the edges). It can be used for scheduling on parallel and/or distributed computing resources [10]. It can also be seen as a serialized or unfolded graph of the MML description, which may be cyclic. Although task graphs themselves are well-known, converting a problem to a task graph is problem-specific, and we have developed an algorithm that allows for the extraction of task graphs from the xMML description [1].

Task graphs can get extremely large, growing exponentially in the number of temporally scale separated submodels, exacerbated by submodels that have a lot of iterations. A methods to reduce the number of nodes is collapsing redundant nodes, which is also demonstrated in [1]. In fact, Figure 3 shows such a reduced graph.

2.3 Multiscale Computing Patterns

We define Multiscale Computing Patterns (MCP) as high-level call sequences that exploit the functional decomposition of multiscale models in terms of single scale models. The task graph introduced in the previous section will be the main ingredients to express MCP in terms of objects within the MMSF.

Based on our previous experience with multiscale modelling we have identified three computing patterns that we believe are most relevant for high performance multiscale computing, namely

- Extreme Scaling,
- Heterogeneous Multiscale Computing, and
- Replica Computing.

When we wrote the project proposal we stated: "This is by no means an exhaustive list, but we believe that these three patterns, and combinations thereof, cover a large range of possible call sequences in high performance multiscale computing." After one year in the project, and having had many discussions with experts in multiscale modelling (within ComPat and with external collaborators) we still believe that this statement is true, and therefore choose to retain the original three patterns. We did realise that we need to be more specific about the computational distinctions between the three patterns, and here we will sharpen the definitions of the patterns as compared to earlier versions. We will discuss hybrids of multiscale computing patterns in D.2.2.

We anticipate a number of key challenges when using today's multi-petascale platforms and future exascale resources in relation to developing and implementing the MCPs. These challenges include unprecedented extreme parallelism, unprecedented number of hardware failures during exascale execution, unprecedented data volumes, and unprecedented costs. In all our MCPs these exascale challenges are of central importance. In particular, unprecedented costs in exascale computing have multiple aspects, as running applications on such scales is accompanied with an exceptional cost requirement in terms of computer time (measured e.g. in core or node hours), energy consumption, monetary cost, or time to completion (particularly in urgent computing). Throughout this section we will discuss cost from a generalized perspective, and describe sections in each of the MCPs that are particularly *cost-critical*. Each MCP has different cost-critical sections, and these differences help us to more clearly separate the definitions for each of the patterns. All the exascale challenges dictate to a large extent our choice of algorithms, as we will discuss below.

It is important to realise that the MCPs, while being prepared for the challenges listed above, primarily catch the computational structure of the multiscale models, independent of the details of the underlying exascale machines. The latter is hidden in the tools, services and middleware, which the multiscale computing patterns rely on (for their description, see deliverable D5.1). Such separation of concerns is a main goal of ComPat, and if successful, can have a strong impact on High Performance Multiscale Computing.

The *Extreme Scaling* computing pattern represents a specific class of multi-scale applications where one (or perhaps a few) of the single scale models in the overall multiscale model dominates all others, in terms of computational and/or energy cost, by far. Such a dominating *primary model* is expected to scale to very large systems (i.e., multi-petascale or above) and the efficiency of the primary model largely determines the efficiency of the full application. Consequently, one of our goals is to ensure minimal interference by the other single scale models, so-called *auxiliary models*. These typically have a much lower computational and/or energy cost, and could even be sequential codes. Load-balancing, decentralized communication, and computation overlapping are some of the techniques we can use here, depending on the relation between the primary and auxiliary models.

The Extreme Scaling pattern applies when, for instance, in addition to the time and space scale differences, one submodel exposes a strong increase in dimensionality or resolution compared to the others, and therefore becomes *cost-critical*. The cost difference between the primary model and the auxiliary models introduces a set of load balancing challenges and possible bottlenecks. The challenge is to efficiently execute the primary model as it is coupled to the auxiliary models, minimizing additional overhead incurred by the coupling and load imbalance, and to propose new variations of the pattern, which include mechanisms for fault-tolerant, energy-efficient, and data-intensive computing.

In the *Heterogeneous Multiscale Computing* pattern, we couple a macroscopic model to a large and dynamic number of microscopic models. The pattern is based on Heterogeneous Multiscale Methods (HMM) [11,12], which are a class of modelling approaches where the constitutive equations of a local state in the macroscale system of interest are not known in closed form, mainly as a consequence of the sheer complexity of the processes at the microscale. The basic philosophy of HMMs is to apply a numerical solver to the macroscale equations and to provide the missing macroscale data using an appropriate microscale model. HMMs hold the promise to simulate very complex phenomena, directly coupling detailed microscopic dynamics to emerging macroscopic behaviour. Instead of relying on heuristic constitutive relations or on idealised theory containing simplifying assumptions, HMMs allow all microscopic details to be taken into account while at the same time being able to simulate macroscale dynamics. Partner UvA has previously developed an HMM model for suspension flow [13], and partner UL has developed a HMM model for Galaxy merger simulations [14]. HMM-type

high performance computational frameworks are still rare; we seek to establish one by defining a multiscale computing pattern for HMM, which is HMC.

The number of microscale models required in HMC depends on spatial properties of the macroscale model, and can in some cases easily be in the order of 10^7 or more (e.g., for suspension flow, where a microscale simulation is required for each lattice point in the macroscale simulation). In addition, each microscale model can be a detailed 3D simulation, e.g., a suspension simulation, requiring substantial parallel computing resources. The large number and size of the microscale models causes them to dominate the computational and energy cost of the application as a whole, and are therefore cost-critical. Consequently, it is essential that these models are efficiently mapped and executed on (exascale) HPC resources, and that any overheads incurred by interactions with other components (such as the HMM manager, described below) are minimized. Like ES applications, HMC applications will require exascale resources to fully meet the ambitions for our applications, primarily due to these cost-critical microscale simulations. Additionally, the cost of microscale models in HMC, especially when mapped to exascale resources, does result in unprecedented data challenges, which we will address by introducing a scalable data management software architecture.

As part of this architecture, we will construct an on-the-fly database to limit the number of required microscale simulations [15,16]. This database serves to store previously computed data and, where desirable, interpolate between already computed values to provide input to the macroscale model. This is feasible because the amount of data passed up to the macroscale model is usually not large, perhaps a few floating-point numbers representing quantities of interest (such as the viscosity for fluid problems) for each microscale model.

The on-the-fly database will be managed by an HMC manager, which will use an asynchronous request-response mechanism. Here the coarse-scale model first sends a request to an HMC manager for the properties that it requires more information on. The manager then consults the database for the needed information, using a user-supplied coordinate system. A user-defined algorithm will decide whether the cached information is sufficiently accurate, and whether interpolated values may be used. If the available information is insufficient, the manager will start a new job to get more accurate fine-scale information. For simulations where the evaluation of cached results is compute intensive, this evaluation will be delegated to a separate computing resource in order not to overload the HMC manager. By using asynchronous I/O and offloading intensive calculations, the HMC manager is able to handle many requests simultaneously.

To ensure the quality of an interpolated result, the HMC manager may start a job simultaneously to verify that the result is indeed sufficiently accurate. If this is not the case, the coarse-scale model will

receive a correction and will need to roll back some calculations. This technique is only feasible if the error rate of interpolated results is sufficiently small and a rollback of calculations can be efficiently performed. Algorithms for this method already exist for smaller resources, and in ComPat we will optimize these methods for exascale resources, where rollback will be considerably more expensive.

Replica Computing is a multiscale computing pattern that combines a potentially very large number of terascale and petascale simulations (also known as 'replicas') to produce scientifically important and statistically robust outcomes. The replicas are not part of a larger spatial structure (as is the case in, for example, Heterogeneous Multiscale Computing), but they are applied to explore a system under a broad range of conditions. Replica Computing is set up through an initialization stage, which determines the simulations required to explore or incorporate a given parameter space. This initialisation is then followed by one or more sequences of simulation and data processing. In general, within Replica Computing we distinguish three scenarios:

1. *Ensemble simulations*. Here a large set of models (replicas) is initiated, run and analysed. The results of these models, which can either be large scale computing jobs themselves or require small amounts of computing resources, are then provided as part of the initial conditions of one or more secondary models. These models may operate on a much larger space and time scale and in that case are usually critical to the overall application in terms of computational and energy cost, or may need just a fraction of resources as compared to the replica computations.
2. *Dynamic ensemble simulations*. Similar to ensemble simulations, here the approach relies on a set of small-scale models that are initiated, run, and analysed. However, in dynamic ensemble simulations, the analysed data is used to rerun a new, better calibrated or complementary set of small-scale models, allowing characterizing system behaviour in complicated parameter landscapes. This landscape in itself can be multiscale. The results of these model executions are then used as a basis for one or more secondary models, similar to the ensemble simulation scenario.
3. *Replica-exchange simulations*. This scenario introduces a set of models with varying parameters (e.g., temperature or spatial characteristics), which are run concurrently. Simulation data is exchanged between these single-scale models at runtime, for example to allow individual particles to migrate from one model to another. These exchanging replicas may be multiscale in their own right, or they can provide statistically robust data, which is then used for more coarse-grained models, which operate on larger time and length scales. Replica exchange simulations are already used in a variety of fields on smaller scales, including e.g. materials science [17], climate sciences [18], biomedicine [19], and origin of life studies [20].

2.4 Multiscale Computing Patterns and the Multiscale Modelling and Simulation Framework

A first step is to investigate how the MCPs map to the components in the MMSF. To do so we will first quickly review some more concepts of the MMSF; for full details we refer to [1,2] and references therein.

The first notion is the *Scale Separation Map* (SSM) and the associated *interaction regions* between two processes placed on the SSM, see Figure 4. Another relevant notion is the relation between the computational domains of two processes. These can either overlap (single domain) or be multi domain, where both computational domains exchange information through a boundary or small overlap region. Note that single-domain vs multi-domain is a property that is additional to the notion of interaction regions, that is for all interaction regions in Figure 4 one can find examples of single-domain or multi-domain multiscale applications. This has immediate consequences on how scale bridging information is exchanged between single scale models. The notion of the interaction regions in combination with the relation between the computational domains leads to a powerful classification of multiscale systems.

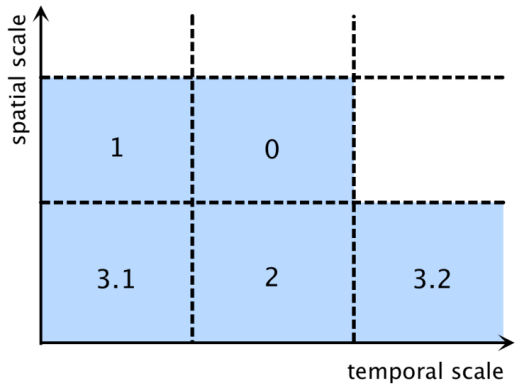

Figure 4: The interaction regions on the scale map, process A resides in region 0, and process B can then reside in 5 regions, leading to 5 types of interactions. 0 – scale overlap, Multiphysics; 1 – time scale separation; 2- spatial scale separation; 3.1 – classical micro-macro coupling; 3.2 – micro-macro coupling where a fast process on a large spatial scale is coupled to a slow process on a small spatial scale.

Next we define a generic *Submodel Execution Loop* (SEL) that abstracts the computations in all single scale models as a while loop over three abstract operators (initialisation, a 'solver', and a boundary condition operator) and two operators that can observe the state of a single scale model (one inside the while loop, and a second upon termination of a single scale model). We find that in coupling together single scale models in a multiscale model, only four *coupling templates*, defined as

directed communication from an observation operator of one single scale model to a computing operator of another single scale model. Figure 5 shows the SEL of two processes, and an example of a coupling template in case the two processes would be time scale separated (so interaction region 1, 3.1, or 3.2). This coupling template is the *call-release* template.

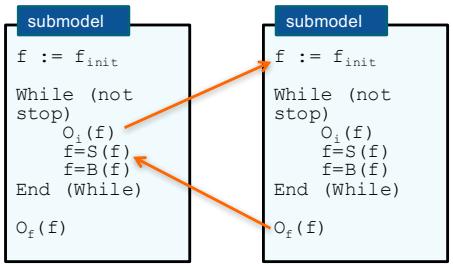

Figure 5: Example of two processes, interaction region 1, 3.1, or 3.2, showing the SEL and the coupling templates (in this case the call – release pair).

Now we introduce the notion of multiscale computing, and the two main multiscale computing paradigms, acyclic (or loosely coupled, or workflows) and cyclic (or tightly coupled), see Figure 6. In acyclic multiscale computing one single scale model provides input to another, and single scale models are executed once. This can be seen as a traditional workflow, with the difference that the scale bridging, the arrow between the single scale models could entail a quite complicated hand-shake. In general, the fact that these models operate on different scales means that the very nature of the models may be radically different – for example, one might be particulate/stochastic, the other continuum based and deterministic. Getting a "handshake" between both models is frequently complicated and requires multiple steps and computations. In cyclic multiscale computing, single scale models call each other in an iterative loop, and therefore single scale models can execute many times. For such cyclic computing dedicated coupling libraries are required.

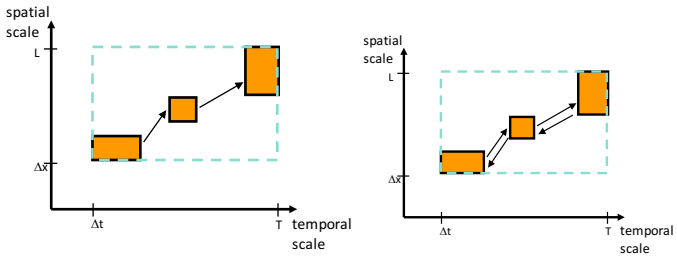

Figure 6: Acyclic (left) and cyclic (right) multiscale applications

Next we need to specify how many instances of single scale models are executed, if this number is fixed or dynamic, and in case of cyclic applications, how many synchronization points are required (so how many cycles are passed in the cyclic application) and if the number of synchronization points are static or dynamic. All possible combinations lead to 9 different *coupling topologies*, see Figure 7. In

ComPat we will limit ourselves, and not implement all possible coupling topologies. Our choice will be determined by the application portfolio.

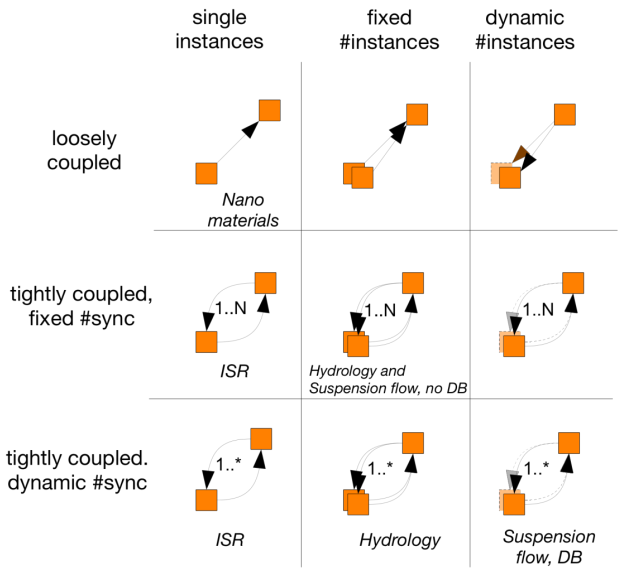

Figure 7: Coupling topologies

The *Multiscale Modelling Language* (MML) translates all these concepts into a graphical (gMML) and machine readable (xMML) specification of the multiscale model that contains in principle sufficient information for execution of the multiscale model in any type of computing environment. In the MAPPER project we have demonstrated all these capabilities in the context of *Distributed Multiscale Computing* (DMC).

Mapping these basic MMSF notions to the three MCPs defined in section 2.3 leads to a number of relevant questions:

1. Can we think of examples of MCPs for all five interaction regions, or, would interaction regions be excluded for specific MCPs?
2. Which MCPs are possible for single domain or multi domain applications?
3. How can MCPs make use of knowledge of the coupling templates, or would this be too application specific?
4. Cyclic versus acyclic, what does this mean for the MCPs?
5. How would MCPs map on the space of possible coupling topologies, and should we limit ourselves?

Answering these questions is an ongoing effort within ComPat. Below we present our current understanding; however, we expect to see updates of this in next deliverables.

In Table 1 we present a summary of our discussions in answering the five questions posed above. This is based on our experience with an extended set of multiscale applications that have been realised in the MMSF, and based on our current understanding of the MCPs. Indeed, the patterns add a further dimension to the MMSF, as they do not seem to be tied to very specific combinations in the MMSF. It is clear that HMM is restrictive in the sense that the underlying HMC application is always a single domain application, in interaction region 3.1. ES and RC are not restricted along those two dimensions. Although ES could be single domain, we exclude that for now, as our current applications do not have one. We expect however that this choice will not exclude single domain ES applications, but this remains to be confirmed. For HMM and RC we will assume dynamic coupling topologies. We realize that this is an ambitious statement. Having said that, currently, the most full fledged and demanding example in ComPat of RC, the Binding Affinity Calculator (see deliverable D3.1) has no dynamic coupling. In the MAPPER project we have not tried such dynamic coupling topologies, and we therefore need to test in some detail the MMSF execution models for this coupling topology. As we move on in the project we intend to update this table when needed.

Table 1: Mapping MCPs to the MMSF

	Extreme Scaling (ES)	*Hierarchical Multiscale Method (HMM)*	*Replica Computing (RC)*
Interaction regions	all are possible	always 3.1, classical micro-macro	all are possible
Relation between computational domains	Usually multidomain, single domain would be possible, but we exclude it for now.	always single domain	all are possible
Coupling templates	all	Always the call/release template	all
Multiscale computing paradigm	Both are possible, but all our applications are cyclic.	Always cyclic	both
Coupling topologies	For now we will limit ourselves to fixed # of instances of single scale models, and fixed # of synchronization points.	Certainly dynamic # of instances, and both fixed and dynamic # of synchronization points are possible.	Certainly dynamic # of instances, and both fixed and dynamic # of synchronization points are possible.

2.5 Generic task graphs and Multiscale Computing Patterns

An important result for WP2, and a major design decision for ComPat, is the realisation that the MCPs in some form will be expressed on the level of the task graph. The task graph, as explained in section 2.2, is a directed acyclic graph used to determine the execution order of submodels, to schedule submodel dependencies and to estimate runtime and communication cost.

The key idea is that we define generic task graphs for each MCP, such that application specific task graphs can be embedded in the generic task graphs. This embedding should of course be automated.

Next, we use the generic task graph to obtain an optimized mapping of the application to an HPC resource, and try to find generic algorithms for this. What exactly is meant by an 'optimal' mapping needs to be defined, or can be made application specific. In any case, it should be optimized with respect to several dimensions (efficient use of resources, power consumption, wall clock time, load balancing, and fault tolerance). The way we proceed is that for each generic task graph we will specify sets of optimal execution profiles, or define constrained optimization problems that should be easily solvable when fed with details of the specific applications. Figure 8 summarizes the approach. An MCP is a tuple of a generic task graph plus data or models on the performance of single scale models (left in Figure 8), a specification of a specific multiscale application in terms of the MMSF (right in Figure 8) and a set of algorithms and heuristics that combine this into detailed input/configuration files for the execution environment (the plus-sign in Figure 8).

The approach in ComPat will therefore be that a task graph is generated from the application specific xMML description. This is then taken together with execution recipes specific for a pattern and performance models, or data, for the single scale models, and with scale bridging algorithms to determine the actual execution. This will finally be specified together with the xMML and the execution recipe, in configuration files for the ComPat middleware. This immediately leads to a number of key research questions:

1. Can we find such generic task graphs for each pattern?
2. How can we map specific applications to such task graphs?
3. How can we use the generic task graphs to set up optimized executions on HPC machines?
4. What information is needed and on which level to make this work?

We intend to answer these questions in the next few months of the project. Some partial answers are formulated in the next sections of this deliverable. Our approach is to work from examples, that is, to create a few early examples that are fully worked out, and then generalise to create full-fledged MCPs.

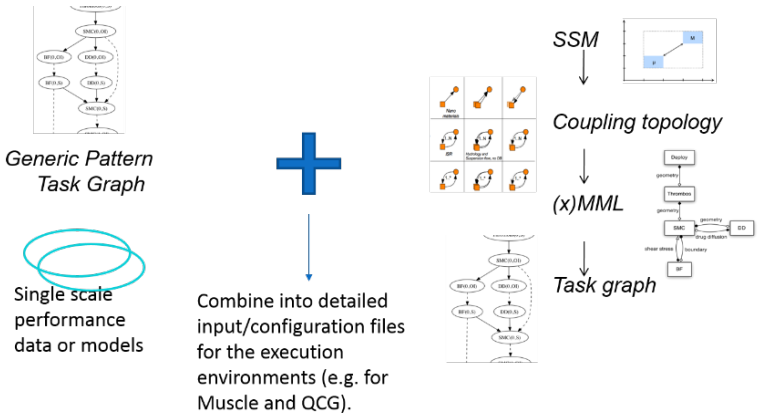

Figure 8: Multiscale Computing Patterns implemented as generic task graphs and algorithms to generate sufficient information for the execution engines.

We have worked out generic task graphs for the *Extreme Scaling* (ES), the *Hierarchical Multiscale Computing* (HMC), and *Replica Computing* patterns, see Figure 9 to Figure 11. The meaning of the boxes and symbols in these figures is shown in the legend in Figure 12. For ES any type of interaction region is still possible, whereas for HMC we assume only type 3.1 (pure micro-macro coupling). For coupling topologies in principle all are possible, but for now we will assume for ES a fixed number of instances and a fixed number of synchronization points. For HMC and RC on the other hand we allow this to be completely dynamic.

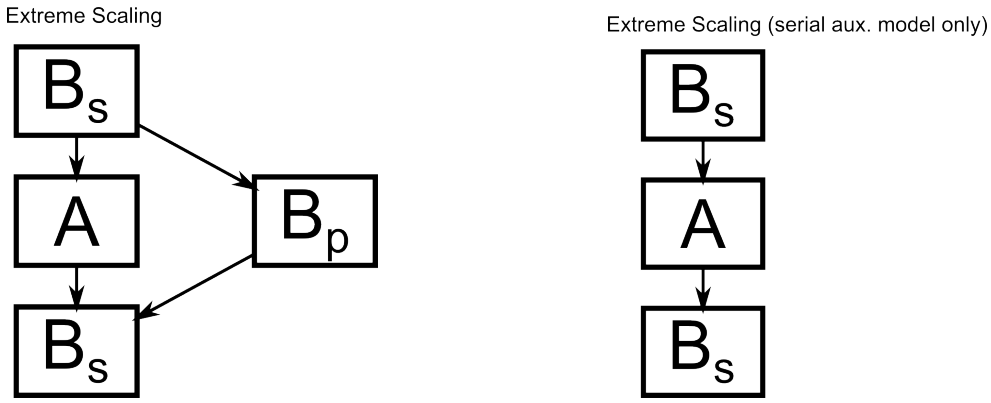

Figure 9: Generic task graph for ES. The version on the left shows the graph with both Serial and Parallel auxiliary models, the version on the right only has a Serial Auxiliary model.

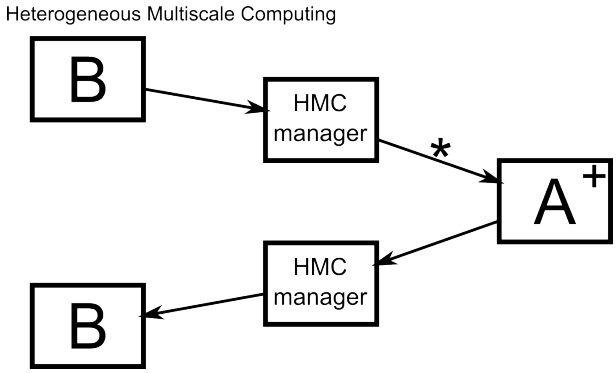

Figure 10: Generic task graphs for HMC.

Figure 11: Generic task graphs for RC. The version on the left shows the static variant for ensemble simulations and replica exchange, while the version on the right is for dynamic ensemble simulation.

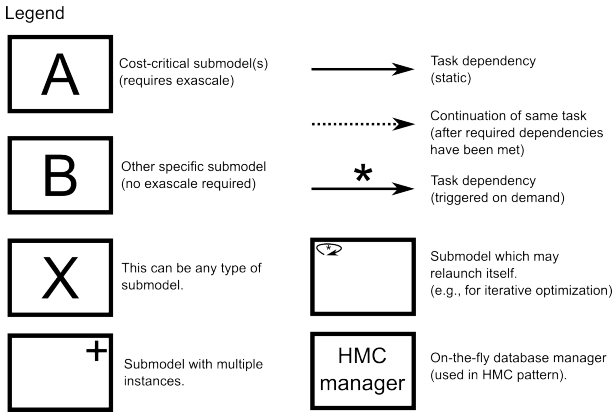

Figure 12: Legend used in the generic task graphs.

Figure 9 (left) shows the generic task graph for ES, where a collection of auxiliary models can either be executed in parallel with the primary model, or in series with the primary model. Figure 9 (right) shows the version where only a serial auxiliary model is present. The graphs should be understood that they are repeating elements in the overall task graph. So, considering the task graph for the In-Stent Restenosis application in Figure 3, and knowing that the fluid flow process is the most time consuming, we can identify the BF (blood flow) process with A in the generic task graph, the DD (drug diffusion) process with the parallel auxiliary model B_p in the generic task graph, and SMC (Smooth Muscle Cells) iterations together as the serial auxiliary model B_s in the generic task graph. We find many repeating units of the generic task graph for ES in the task graph from Figure 3, typically a few 1000 (depending on the time step taken in the macroscale SMC process, and the total time to be simulated – typically a timestep of 1 hour and total simulated time of a few weeks). We also find that the application in Figure 3 has an initialisation phase (the Deploy and Thrombus processes) as well as a post processing phase (not shown in Figure 3), which are currently not captured by the generic task graphs. We intend next to work out more examples of the task graph and create required software. Next we will return to the definition of the generic task graph and further update and refine them, based on experience gained in working with them.

Depending on the execution behaviour of the primary and auxiliary models on HPC machines, a specific execution of the ES graph is considered. The main aim is to align the auxiliary models to produce their data before the primary model requires it. This would be achieved by either pre-computing the values of non-scaling models or interleaving simulations as suggested by the performance model, see also the discussion in section 2.3.

For HMC, Figure 10, a large and dynamic number of microscale simulations is coupled with one macroscale model, with a HMM database in between. The role of the database is to prevent computing

of previously computed results, to interpolate between earlier computed results, and to submit microscale simulation jobs when needed. The latter could be done in a pro-active way, if resources allow it, to start precomputing quantities in anticipation of request from the macroscale solver. A specific execution graph for this pattern depends on the execution behaviour of the macro- and micro-scale models on HPC machines.

For RC, see Figure 11, we find two variants, which capture the behaviour of the three types of replica computing that we defined. In both cases a potentially large set of replicas *A1+* are executed independently and then fed into a second master process A_2. In RC, if a replica fails, a restart is not immediately needed, as long as the overall statistical quality of the ensemble that is computed by the RC application is maintained. This is a main distinction with HMM, where if a microscale simulation fails it must be restarted, as the database requested output from this microscale simulation. This will be further investigated in the next 6 months of the project.

With the generic task graphs defined, we can now continue to specify in more detail issues related to fault tolerance, load balancing, and energy awareness for each pattern, and try to push as much as possible on the level of the generic MCPs.

2.6 Example of Extreme Scaling

For the ES case we have worked out in more detail a scenario for one of the applications in ComPat, related to Biomedicine. We consider the case of a cell-based blood suspension model as the primary model, coupled to continuous blood flow models as the auxiliary models, see Figure 13.

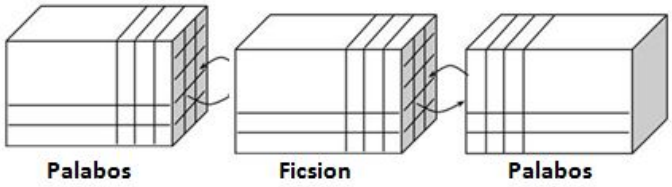

Figure 13: example of an ES application, a blood suspension model (Ficsion, the primary model) coupled to two continuous blood flow models (auxiliary models, Palabos) that provide the in- and outflow conditions for the suspension model (more details for this application are described in deliverable D3.1).

The resulting SSM, gMML and task graph are shown in Figure 14. Note that we assume two instantiations of the auxiliary model, which in itself is a full-fledged parallel application, representing the inlet and outlet regions of the suspension flow domain. Also note that the embedding of this application task graph into the generic ES task graph (Figure 9, left) is straightforward, the two

instantiations of Palabos together form the serial auxiliary model, and the parallel auxiliary model is empty.

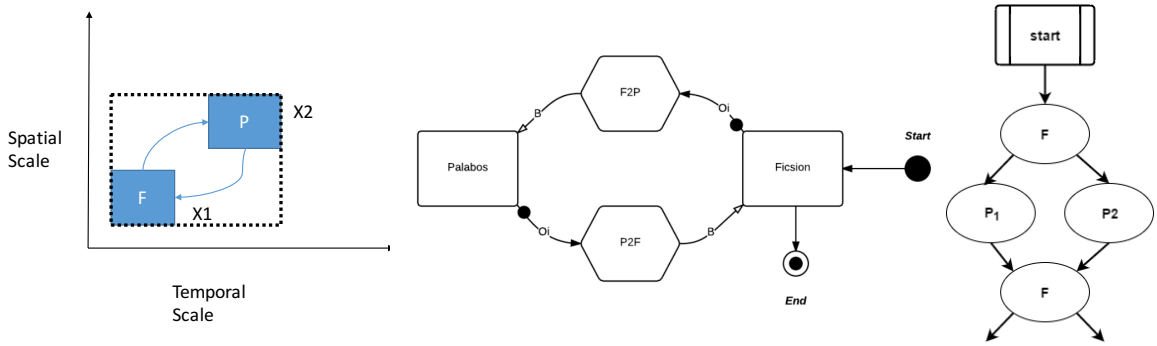

Figure 14: The SSM (left), gMML (middle), and task graph (right) for the ES case coupling a primary model for blood suspensions (Ficsion, F) to two instantiations of the auxiliary model (Palabos, P).

Next we formulate a (preliminary) performance model. Call $Tpr(P,Npr)$ the execution time of the primary model in this example as a function of the number of processors P and the problem size of the primary model Npr. Likewise, call $Taux(P,Naux)$ the execution time for the serial auxiliary model as a function of P and the problem size of the auxiliary model $Naux$. A defining feature of the ES patterns is that the primary model is very compute intensive and needs petascale or even exascale performance. In terms of the performance model this means that $Tpr(1,Npr) \gg Taux(1,Naux)$. Both the primary and auxiliary models can run in parallel, but their scalability can be completely different. For the execution time of the ES scenario we can now write

$$TES=TprP,Npr+Taux(P,Naux) \quad (1)$$

and for the resulting efficiency of the ES pattern we can write

$$\varepsilon = 1P1+1P+P \approx 1P1P+P = 1P\ 1P1 + P1 = \varepsilon PP+1 \quad (2)$$

with ε the efficiency of the primary model. We assume that the primary model scales very well, which seems like a reasonable assumption, given that in the ES case the primary model is the one that requires almost all computing resources and has been optimized sufficiently to run very efficiently on Petascale or emerging Exascale resources.

We consider two limiting cases. First assume that the auxiliary model does also scale well enough on the available resources, at least so that even on P processors, $Tpr(P,Npr) \gg Taux(P,Naux)$. In this case we find that $\varepsilon=\varepsilon$ and we can simply execute the ES task graph, as drawn in Fig. 3 (left).

The other extreme would be that the auxiliary model does not scale at all, even to the point that $Tpr(P,Npr) \ll Taux(P,Naux)$, leading to $\varepsilon \to 0$. This requires a completely different execution of the MCP. Suppose that we could find a pair of processor numbers such that $P=P1+P2$ and $Tpr(P1,Npr) \sim Taux(P2,Naux)$, then we could interleave two instantiations of the ES patterns, combining the execution of a primary model in the first instance with execution of the auxiliary model in the second instance, see Figure 15 (right).

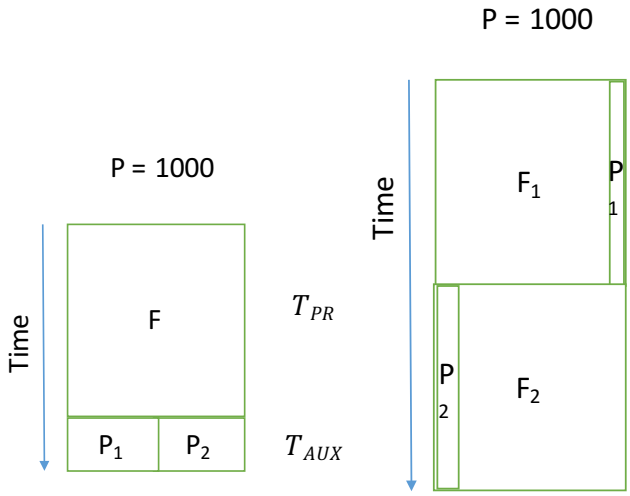

Figure 15: Two limiting possibilities of executing the ES task graph from Figure 14. In the left figure, F denotes Ficsion (the primary model) and P_1 and P_2 the two serial auxiliary models (Palabos). In the right figure F_1 and F_2 denote two instantiations of the primary model, and now P_1 and P_2 denote the auxiliary models coupled to F_1 and F_2 respectively (so lumping together the two Palabos instantiations).

In reality most cases will be somewhere between these two limiting cases, and we need to decide what to do. This will lead to an algorithm that will form the heart of the ES MCP, and will be addressed in the next few months of the project. Here interesting possibilities appear to optimize both throughput of an ES pattern, to have a well-balanced load, and to optimize energy usage, by allowing adjustment of the clock speeds of the set of processors P_1 or P_2. In that case $Tpr(P1,Npr) \sim Taux(P2,Naux)$ leads to a whole series of potential solutions of the load balancing problem, and on that manifold we can then optimize either throughput, and/or energy usage of the execution. The ES MCP algorithm will also

take into consideration threshold values, which indicate when to switch from one type of execution to another.

2.7 Plans for Y2

With the MCP now firmly defined in terms of generic task graphs, MMSF constructs and additional performance estimations, the next step will be to implement a few examples (in close collaboration with WP3). We intend to select example applications from each MCP, and to have them fully operational by M18 of the project, so that we can demonstrate MCPs in action during the first project review.

In more details, this requires:

1. Elaboration of the ES, HMM, and RC pattern in terms of generic task graphs and associated execution paths and generic performance models;
2. For all applications a full xMML specification (WP3);
3. Software to create task graphs from xMML. We have an implementation available from the algorithm proposed in [1], this will be reengineered to fit into the ComPat software stack;
4. Software to embed the application task graph into one of the generic task graph;
5. Performance data or models for all single scale models, scale bridging methods, etc, that are part of the full multiscale model. This can be done analytically, and/or by relying on tools from WP4;
6. Software that will take as input (generic) task graphs, execution models, optimization goals, together perhaps with information on the target HPC machine, and produces as output configuration files and execution scripts for the ComPat middleware so that the application can be executed as efficiently as possible on the EEE (for a description of the EEE, see deliverable D6.1).

Our goal is to have first implementations of the whole tool chain available by M18 of the project. Once this goal has been achieved, the next step will be to add functionality for fault tolerance, advanced load balancing, and energy awareness. Also, we will then be able to clearly indicate if and how (x)MML should be updated, and to what extent updates of coupling libraries such as MUSCLE or AMUSE would be needed to be able to execute the MCPs.

3 Conclusions

Multiscale Computing Patterns are expressed on the level of task graphs. A Multiscale Computing Pattern is a generic task graph plus a set of rules on how to execute the task graph, based on performance of the single scale simulations, plus pieces of software that result in sufficient

information (input files, configuration files) for the ComPat middleware. This definition is one of the main results of the work package 2 in the first year of the project. We have confirmed the suitability of the current Multiscale Modelling & Simulation Framework and related software for Multiscale Computing Patterns, but we have not yet a clear understanding of the (x)MML definitions which need to be updated, and if so, to what extent that would lead to changes in the software chain that generates xMML files, and the software that generates task graphs from xMML. This will be investigated in the second year of the project. Our approach, as discussed above, will be to study specific instances of the Multiscale Computing Patterns, relying on the application portfolio from work package 3, and then generalising these.

4 References

[1] J. Borgdorff, J.-L. Falcone, E. Lorenz, C. Bona-Casas, B. Chopard, A.G. Hoekstra, Foundations of distributed multiscale computing: Formalization, specification, and analysis, J. Parallel Distrib. Comput. 73 (2013) 465–483. doi:http://dx.doi.org/10.1016/j.jpdc.2012.12.011.

[2] B. Chopard, J. Borgdorff, A.G. Hoekstra, A framework for multi-scale modelling, Philos. Trans. R. Soc. A. 372 (2014) 20130378. doi:10.1098/rsta.2013.0378.

[3] S. Karabasov, D. Nerukh, A. Hoekstra, B. Chopard, P. V Coveney, Multiscale modelling: approaches and challenges, Philos. Trans. R. Soc. A. 372 (2014) 20130390. doi:10.1098/rsta.2013.0390.

[4] J.-L. Falcone, B. Chopard, A. Hoekstra, MML: towards a Multiscale Modeling Language, Procedia Comput. Sci. 1 (2010) 819–826. doi:DOI: 10.1016/j.procs.2010.04.089.

[5] J. Borgdorff, M. Mamonski, B. Bosak, K. Kurowski, M. Ben Belgacem, B. Chopard, et al., Distributed multiscale computing with MUSCLE 2, the Multiscale Coupling Library and Environment, J. Comput. Sci. 5 (2014) 719–731. doi:http://dx.doi.org/10.1016/j.jocs.2014.04.004.

[6] J. Borgdorff, C. Bona-Casas, M. Mamonski, K. Kurowski, T. Piontek, B. Bosak, et al., A Distributed Multiscale Computation of a Tightly Coupled Model Using the Multiscale Modeling Language, Procedia Comput. Sci. 9 (2012) 596–605. doi:10.1016/j.procs.2012.04.064.

[7] J. Borgdorff, M. Ben Belgacem, C. Bona-Casas, L. Fazendeiro, D. Groen, O. Hoenen, et al., Performance of distributed multiscale simulations, Philos. Trans. R. Soc. A Math. Phys. Eng. Sci. 372 (2014) 20130407. doi:10.1098/rsta.2013.0407.

[8] D. Groen, J. Borgdorff, C. Bona-Casas, J. Hetherington, R.W. Nash, S.J. Zasada, et al., Flexible composition and execution of high performance, high fidelity multiscale biomedical simulations, Interface Focus. 3 (2013) 20120087. doi:10.1098/rsfs.2012.0087.

[9] M.B. Belgacem, B. Chopard, J. Borgdorff, M. Mamonski, K. Rycerz, D. Harezlak, Distributed multiscale computations using the MAPPER framework, Procedia Comput. Sci. 18 (2013) 1106–1115.

[10] Y.-K. Kwok, I. Ahmad, Benchmarking and Comparison of the Task Graph Scheduling Algorithms, J. Parallel Distrib. Comput. 59 (1999) 381–422. doi:10.1006/jpdc.1999.1578.

[11] E. Weinan, Principles of Multiscale Modelling, Cambridge University Press, 2011.

[12] E. Weinan, B. Engquist, X. Li, Weiqing Ren, E. Vanden-Eijnden, Heterogeneous Multiscale Methods: A Review, Commun. Comput. Phys. 2 (2007) 367–450.

[13] E. Lorenz, A.G. Hoekstra, Heterogeneous Multiscale Simulations of Suspension Flow, Multiscale Model. Simul. 9 (2011) 1301–1326.

[14] S.P. Zwart, J. Bédorf, Computational Gravitational Dynamics with Modern Numerical Accelerators, ArXiv E-Prints. (2014) 1409.5474.

[15] W. E, B. Engquist, The Heterogeneous Multiscale Methods, COMM. MATH. SCI. 1 (2003) 87–132.

[16] E. Lorenz, Multi-scale simulations with complex automata: in-stent restenosis and suspension flow, Thesis, PhD thesis, University of Amsterdam, 2011.

[17] E. Martínez, B.P. Uberuaga,, A.F. Voter, Sublattice parallel replica dynamics, Phys. Rev. E. 89 (2014) 63308.

[18] D. Stevenson, F. Dentener, M. Schultz, K. Ellingsen, T.V. Noije, O. Wild, et al., Multimodel ensemble simulations of present-day and near-future tropospheric ozone, J. Geophys. Res. 111 (2006) D08301.

[19] G. Bowman, X. Huang, V. Pande, Network models for molecular kinetics and their initial applications to human health, Cell Res. 20 (2010) 622–630.

[20] P. Coveney, J. Swadling, J. Wattis, H. Greenwell, Theory, modelling and simulation in origins of life studies, Chem. Soc. Rev. (2012).