COMPAT
Computing Patterns for High Performance Multiscale Computing

Deliverable 3.2: Report on Instantiating Computing Patterns for HPMC Applications

Due Date	M18
Delivery	M18+1 week, PO agreed
Lead Partner	UCL
Dissemination Level	PU
Status	Final
Approved	Internal review yes
Version	V1.0

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
24.03.2017 v0.1	James Suter	First draft
28.03.2017v0.2	Alfons Hoekstra / Saad Alowayyed	Changes to integrate with D2.2
29.03.2017v0.3	Oliver Hoenen	Additional input to fusion application
30.03.2017v0.4	Derek Groen	Internal Review
31.03.2017v0.5	Oliver Perks	Internal Review

CONTRIBUTORS

The contributors to this deliverable are:

Contributor	Role
James Suter (UCL)	Author of the deliverable
Saad Alowayyed (UvA)	Contributor
Derek Groen (Brunel)	Contributor and Internal Reviewer
Alfons Hoekstra (UvA)	Contributor
Olivier Hoenen (IPP)	Contributor
Paval Sun (ITMO)	Contributor
Oliver Perks (Arm)	Internal Reviewer

TABLE OF CONTENTS

LIST OF TABLES AND FIGURES

LIST OF ABREVIATIONS

AMUSE	Astrophysical Multipurpose Software Environment
CPU	Central Processing Unit

EEE	Experimental Execution Environment [Project Testbed]
ES	Extreme Scaling [Pattern]
FEM	Finite Element Methods
gMML	Graphical Multiscale Modelling Language
GPGPU	General-Purpose Graphics Processing Units
HMC	Heterogeneous Multiscale Computing [Pattern]
HMM	Heterogeneous Multiscale Manager
HPC	High Performance Computing
HPMC	High Performance Multiscale Computing
IBLB	Immersed Boundary Lattice Boltzmann
ISR	In-Stent Restenosis
MAPPER	Multiscale Applications on European e-Infrastructures [Project]
MML	Multiscale Modelling Language
MPI	Message Passing Interface
MUSCLE	Multiscale Coupling Library and Environment
QCG	Quality in Cloud and Grid
RBC	Red Blood Cells
RC	Replica Computing [pattern]

1 Executive summary

This document summarises the work done so far in instantiating computing patterns for High Performance Multiscale Computing (HPMC) applications using the ComPat tools, software and infrastructure.

We have successfully instantiated three fast-track applications (in the areas of nuclear fusion, drug binding efficiency and blood-flow modelling) using the ComPat components on the Experimental Execution Environment (EEE). The fast-track applications were identified in Deliverable 3.1 as those where the single-scale models are well optimised and scale-bridging components (to couple different single-scale models) are already developed. These applications cover two MCPs (Extreme Scaling and Replica Computing) and require two different coupling environments (MUSCLE2 for ES and file-based data exchange/FabSim for RC). As per the Description of Actions (DoA), this work is part of Task 3.2 "Instantiation and validation of multiscale computing patterns (M13-24)".

The fast-track applications have, as planned, proved to be useful test cases for the Multiscale Computing Pattern Software developed in WP 2, which enables an application to be executed on the EEE starting from a Multiscale Modelling Language (MML) description of the application (Deliverable 2.2). As described in Deliverable 2.2, once instantiated as a computing pattern, the Multiscale Computing Pattern Software orchestrates the HPMC application to run as efficiently as possible on the EEE using the ComPat technology described in Deliverables 5.1 and 5.2. In this document, we present the chain of actions and configuration files required to achieve this instantiation from an end-user perspective (Section 2.3 and the Annex). This document therefore provides a guide for future instantiations of other HPMC applications.

In the next phase of the project, Tasks 3.2 and 3.3 will be active. For the former task, we will instantiate the remainder of the deep-track applications; the process is described in this document, including the scientific challenges this will allow us to study (Section 3). This includes the Heterogeneous Multiscale Computing (HMC) pattern. The later task, in close collaboration with WP4, will provide us with automatic collection of the performance of the applications to feed back into Multiscale Computing Pattern Software (WP2). The next phase of the project requires close collaboration between WP2, WP3 and WP4, continuing our recent work in developing the Multiscale Computing Pattern Software in WP2 and instantiating HPMC applications in WP3.

2 Report on instantiating Computing Patterns for HPMC applications

2.1 Introduction

The objective of Work Package 3 “concerns the instantiation of multiscale computing patterns with the selected grand challenges”. This includes “enabling application software to interface with high-level tools and middleware services”. The aim of this document is to report on our progress in instantiating applications as Multiscale Computing Patterns (MCPs) using ComPat technology components, including the very recent work on the multiscale computing pattern software, described in Deliverable 2.2.

As described in Deliverable 3.1, assessing the readiness of our grand-challenge applications, it was clear we should divide the applications into “fast” and “deep”-track. The fast-track applications were identified as those where the single-scale models are well optimised and scale-bridging components (to couple different single-scale models) are already developed. The deep-track applications required more sophisticated multiscale computing scenarios and/or development of scientific scale-bridging methods before they can be instantiated. In this phase of the project we have instantiated the fast-track applications, and we describe the process of instantiating these computing patterns using the current ComPat technology stack (WP 5) on the Experimental Execution Environment (EEE) (WP 6), and for developing tools in WP 2 that convert Multiscale Modelling Language (MML) descriptions of applications into task graphs and subsequently execution recipes (referred to as the multiscale computing pattern software in the rest of this report). Please note that this Deliverable 3.2 was originally scheduled in M24, but because of change of the reporting period from 12 months to 18 months during the grant preparation phase, was brought forward to M18. This report is a snapshot of current work in progress, and hence many of the specifications described here may be subject to change in subsequent reports. We also briefly describe forthcoming work with the deep-track applications, which will take place in later reporting periods.

We have so far instantiated the three fast-track applications. These applications cover two MCPs (Extreme Scaling and Replica Computing) and require two different coupling environments (MUSCLE2 for ES and file-based data exchange/Fabsim for RC). The fast-track applications are: calculating ligand-protein binding affinities, global turbulence simulations and Red Blood Cells (RBCs) with platelet transport. The latter two applications use the MUSCLE coupling environment as they require communication between submodels at each iteration of the outer submodel. These applications are examples of the Extreme Scaling (ES) pattern. The ligand-protein binding affinities simulations (which we refer to as the Binding Affinity Calculator (BAC) in the rest of this document) can be described as a loosely-coupled application, where the results of one complete submodel simulation is used as input into the next submodel (*i.e.* a workflow). As described in Deliverable 3.1, the BAC requires many identical replicas of each submodels to produce a binding affinity with reproducible error, and hence this application is an example of a Replica Computing (RC) pattern.

The work in this reporting period is highly aligned with ongoing Tasks 2.3, 4.3, 5.2 and 5.3, which are described in detail in Deliverables 2.2, 4.2 and 5.3. Therefore, in this report, we focus on the instantiation of computing patterns from an *end-user* perspective. The technical details regarding other work-packages is addressed in their respective deliverables. We will also not discuss the scientific details of each grand-challenge application in this report, as this was previously described in detail in Deliverable 3.1. Instead, we report the status of each fast-track application and subsequently the procedure for an end user to instantiate their multiscale application using the ComPat technology as it currently stands, using the fast-track application as examples. In the Annex we provide the specific configuration files etc. required for the fast-track applications.

2.2 Description of fast-track applications and their requirements

2.2.1 Global turbulence application (Extreme Scaling)

The global turbulence application aims at simulating the evolution of temperature profiles in the plasma core at transport time scales (seconds) while taking into account the influence of turbulence (micro seconds), to predicting the performance of future fusion devices. Turbulence is calculated with a gyrofluid turbulence model (3D) describing the plasma from a set of so-called flux tubes [4]. Heat fluxes are then averaged in time and converted into transport coefficient for the transport code (1D) with regular update of the plasma equilibrium (2D). All submodels are coupled together by means of a standardized data structure through the MUSCLE2 library [1-3] on the EEE (PSNC, LRZ). In addition, an adaptive time stepping mechanism has been added to the transport solver, which allows us to limit evolution of gradients to keep the turbulence model in a well-defined range. The multiscale model consists of a 3D gyrofluid submodel (GEM), and two other codes (ETS for 1D transport, CHEASE for 2D equilibrium) and one numerical tool (IMP4DV, to convert heat fluxes coming from the turbulence submodel into coefficients taken into account by the transport submodel to evolve the temperature profiles). There are also two mappers [1] (EQUIL) and (COREP), which are “fan-out”: *i.e.* they take the data outputted from one submodel and communicate it to more than one model (three in both cases). The primary model (highest computational cost) is GEM, while the other submodels are auxiliary models. The graphical representation of the MML (gMML) for the global turbulence model is shown in Figure 1, created using the jMML programme developed during the MAPPER project [3].

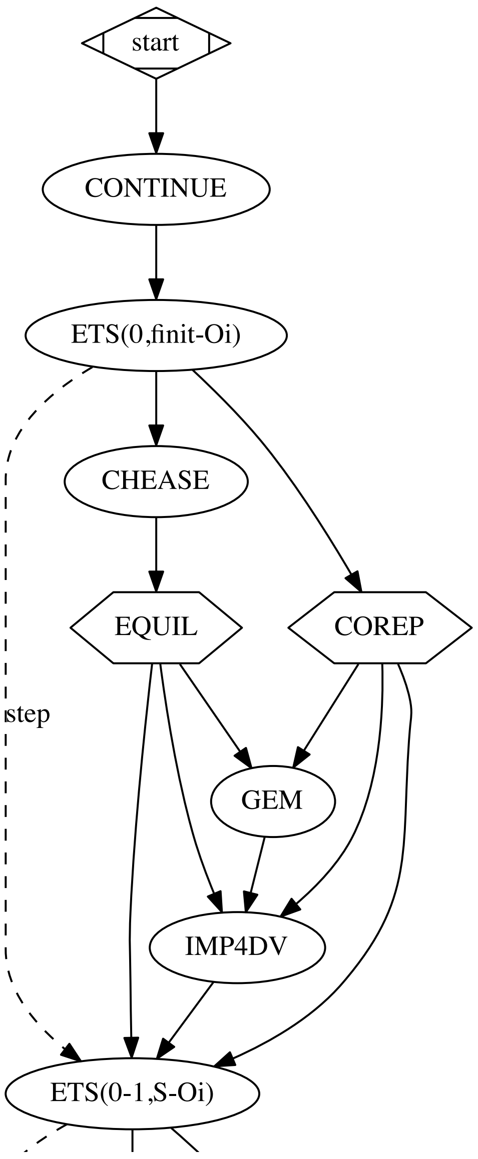


Figure 1: gMML representation of the global turbulence application, showing the first iteration of the ETS submodel. The models surrounded by an ellipse are submodels, those surrounded by hexagons are mappers, while arrows correspond to data conduits (see Borgorff *et al.* [1] for more details).

2.2.2 BAC application (Replica Exchange)

The aim of the Binding Affinity Calculator (BAC) is to calculate the strength of macromolecular binding free energies using computationally based molecular modelling for applications in drug discovery and personalized medicine [5]. The underlying computational method is based on classical molecular dynamics (MD). These are coupled to the molecular mechanics Poisson−Boltzmann surface area (MMPBSA) method to calculate binding free energies. In order to obtain statistically converged and reproducible results, ensembles of MD calculations are performed with each simulation differing only in their initial starting conditions. A typical BAC run for a single protein-ligand complex consists of a two-step workflow (MD-MMPBSA) performed for 25 independently initialized replicas. It used two different codes (NAMD [6] and AmberTools [7]) which are coupled through the output files of NAMD being read as input by AmberTools. This workflow requires the working directory to be persistent, with the AmberTools submodel to be executed in the same directory as the NAMD submodel. These requirements have been added to the QCG submission tools (see Deliverable 5.2). The gMML representation of the BAC workflow is shown in Figure 2 for a single replica and with the replicas explicitly represented.

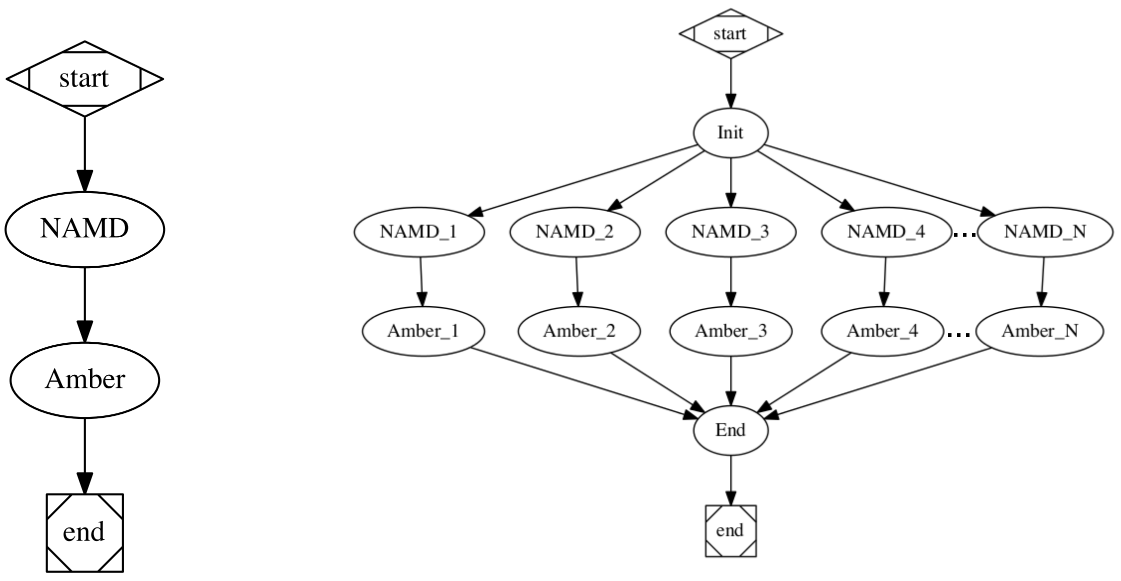


Figure 2: gMML single replica workflow for the BAC (left). There are *n* replicas created of this workflow; this is explicitly represented in the gMML on the right.

2.2.3 Red Blood Cells (RBC) application (Extreme Scaling)

This fast-track application is a sandbox application to couple Palabos [8] (a fully parallelised open source Lattice Boltzmann Model) with the Hemocell (called Ficsion in earlier reports) suspension simulation framework (an Immersed Boundary Lattice Boltzmann Model (IB-LBM)) [9]. This coupling is shown in Figure 3, demonstrating that the continuous fluid field represented in LBM is coupled to the in- and outlets of the dense suspension fluid. Note that, the scientific scale bridging model is still missing in this work, and is under development. Nevertheless, while the scale bridging algorithms are being developed, we can already use the sandbox implementation to test all issues with respect to the ES pattern. The sandbox application has been coupled using MUSCLE on EEE resources.

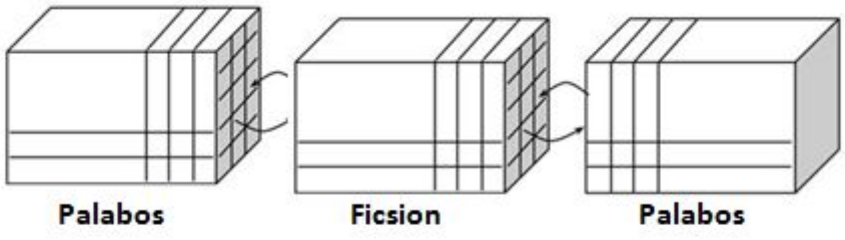


Figure 3: An example of an ES application, a blood suspension model (Ficsion, the primary model) coupled to two continuous blood flow models (auxiliary models, Palabos) that provide the inflow and outflow conditions for the suspension model.

The gMML representation of the RBC is shown below in Figure 4.

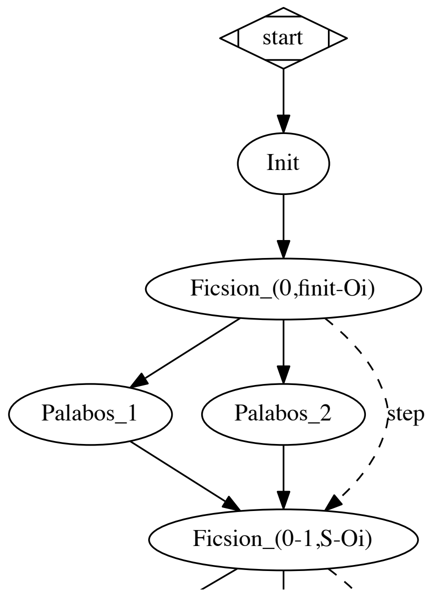


Figure 4: gMML representation showing the first iteration of the RBC model. The primary model (Ficsion/Hemocell) communicates with two auxiliary models (Palabos_1 and Palabos_2).

2.3 Progress in employing multiscale computing patterns in scientific applications

In Deliverable 2.1, we made a major design decision for ComPat: The MCPs will be expressed on the level of a task graph, generated from the XML format of MML (xMML) [1-3]. The xMML description is therefore the starting point of the instantiation for the end-user. The aim of ComPat is for control flow tools to be directed by execution recipes created by the multiscale computing pattern software developed in WP 2, starting from this xMML description of the application. In the following sections we will describe the full chain of actions for an application to be instantiated as a computing pattern by the end-user, using an xMML description as the starting point.

The chain of actions for instantiating a multiscale simulation pattern as a MCP is as follows (referring to Deliverable 2.2 for all the details of the software parts that are used):

1. Ensure the submodels are correctly installed on the required EEE resources / compatible with ComPat stack components on the EEE.
2. Describe the full multiscale simulation as an xMML description.
3. [Optional] Create an YaML input file specifying input files, submodel restrictions (available environment modules, specific libraries, minimum or maximum number of cores, maximum number of nodes, allowed resources, CPU / GPGPU cores etc). This currently also includes the total number of cores for the whole multiscale simulation to be run on.
4. If submodel performance data is not available, the end-user must create the required performance data (filling the “performance matrix”) by performing benchmark runs.
5. Use the “description part” of the WP 2 multiscale computing pattern software to combine user-supplied input (if present) and the xMML description to create configuration files for the multiscale simulation (matrix.xml and multiscale.xml); these are used as input (with the performance data) for the WP 2 performance model software (see Deliverable 2.2, referred to as the “optimisation part” of the multiscale computing pattern software). The end-user can also enter simulation specific information by completing template matrix.xml and multiscale.xml files (created by the description part of the multiscale computing pattern software) if the user has not supplied a YaML file.
6. The “optimisation part” of the multiscale computing pattern software produces several options with optimal performance for QCG to submit to EEE (see Deliverable 2.2).
7. The QCG-broker selects the most appropriate resource from those chosen by considering the load on that machine (see Deliverable 5.2), referred to as the “services part” of the multiscale computing pattern software.

Table 1: Table showing the status of each application (fast and deep-track) with the instantiation steps.

	Step 1 Installed on EEE	Step 2 xMML description	Step 3 YaML file user-input	Step 4 Performance data	Step 5 Multiscale computing pattern software	Step 6 Input to QCG	Step 7 Submission by QCG-Broker.
Fast track applications (M12-18)							
Global turbulence	✓	✓		✓	✓	✓	✓
RBC	✓	✓		✓	✓	✓	✓
BAC	✓	✓	✓	✓	✓	✓	✓
Deep track applications (M19-36)							
HemeLB	✓		✓				
ISR	✓	✓					
AMUSE Astrophysics	✓						
Fusion Deep-track	✓						
BAC Deep-track	✓						

In Table 1, we show which steps are currently enabled for each of the three fast-track applications. In step 5, the multiscale computing pattern software converts the xMML description into an acyclic directed (task) graph, and patterns in the graph are identified. The multiscale computing pattern software combines this result with execution recipes specific for the pattern, performance models or data for the single scale models and scale bridging algorithms, to determine the actual execution, which is specified in configuration files for ComPat middleware. xMML has been described in Deliverable 2.1 (and references therein). In the Annex 5.1 we show the xMML description for all three fast-track applications, along with the associated task-graphs. Details of how the multiscale computing pattern software decides on the best execution scenarios are described in Deliverable 2.2.

The essential feature of the xMML description is that it is a human readable text file that can easily be written by the application user to describe the coupling topology of the multiscale application and to give information on the nature of each of the submodels. For example, in the xMML description of the BAC application, the "multiplicity" tag indicates that the Replica Computing should be invoked, with the number of replicas corresponding to the "multiplicity" attribute. Currently, for the fast-track applications, the user supplies the xMML for their application, and the tools developed in WP2 automatically determine the corresponding MCP; however, we plan to provide xMML templates to assist the design of new multiscale simulations.

```
<instance id="NAMD " submodel="NAMD " multiplicity="16"/>
<instance id="Amber" submodel="Amber" multiplicity="20"/>
<coupling from="NAMD.toAmber" to="Amber.toAmber"/>
```

Figure 5: The "multiplicity" tag in the xMML description of the BAC, indicating that the replica computing (RC) pattern will be invoked. This is recognised by the "translation" part of the multiscale computing pattern software.

In Step 5, the multiscale computing pattern software requires information from the end user about both the overall multiscale simulation (number of cores, restrictions etc.) and submodel specific information (such as whether there is a restriction on the number cores, or type of nodes (GPU / CPU etc)). There can be multiple instances of a submodel; for example, a submodel could be defined in the xMML language as "Molecular Dynamics", and the instance could be different molecular dynamics codes such as LAMMPS or NAMD. The user must therefore specify which instance is required, along with any restrictions that apply to that particular instance. An illustration is in the xMML for the BAC application, which although specifying NAMD, the actual code (instance) used is specified by the end user.

This distinction between "submodel" and "instance" is part of our desire that xMML scripts provide templates that can be reused for other multiscale simulations, or indeed that codes can be used in a plug-and-play manner. In this manner, the topology and interaction between submodels are separated from their implementation on the resources, allowing the details of the instance to be defined by the end-user, without altering the xMML description of the multiscale application.

This information can be supplied in a variety of ways. The first way, currently implemented for the BAC application, is in the form of a YaML file, which provides a compact, intuitive and human-readable data format. The file is based upon the machines.yml file used by FabSim for the configuration settings of submodels on remote resources [10]. Additional tags to indicate restrictions on the use of the code, etc are included. An example of the user specified YaML files for the BAC application can be found in the Annex 5.2. The software developed in WP2 takes the xMML file and the end-user supplied information, and converts it into a form readable by the performance prediction software, developed in WP2. This consists of two files: matrix.xml, which specifies submodel information, and multiscale.xml, which specifies the coupling between the submodels. The multiscale computing pattern software can produce template matrix.xml and multiscale.xml files direct from the xMML description of the application if no information is supplied by the user when the multiscale computing pattern software is executed. This is therefore the second option for the user to supply the required information; the template files can be completed by the user when all the information is available and this can be used as input to the next stage of the multiscale computing pattern software. Examples of matrix.xml and multiscale.xml for the fast-track applications is given in Deliverable 2.2.

If the application uses the MUSCLE coupling environment (such as the transport-turbulence and RBC application), the developer specifies MUSCLE either in the YaML file (to be developed) or in matrix.xml / multiscale.xml (current implementation). To assist the end-user when using the MUSCLE coupling environment, the software developed in WP2 creates a template cxa configuration file for MUSCLE from the xMML description of the multiscale model. This template is then completed by the end user. An example of a template cxa file is supplied in Annex 5.4.

For the performance service to work, it requires performance data for each submodel on different resources / types of computational nodes. A description of how nodes-types are defined is given in Deliverable 2.2. In the next reporting period, using the performance data collection tools developed in WP4, we will develop methods to automatically generate this data. This is the focus of Task 3.3 (M19-M36) "Task 3.3: Profiling and performance measurement". Deliverable 4.2 reports on the current status of the performance analysis tools to be used in ComPat with the fast-track applications. We envisage this being possible in two ways: either by use of a benchmark flag when invoking the performance service, which submits a ready-made benchmark file for a submodel of which timings are scaled to give estimates of runtime behaviour, or via monitoring of previous full multiscale simulations. Performance data is currently collected for the global turbulence application using the tools developed in WP 4 (Deliverable 4.2). We will shortly be developing methods for this data to be transferred to a location where it can be accessed by the performance modelling part of the multiscale computing pattern software. However, in the current instantiations of fast-track applications, the performance data is supplied by the end-user in the YaML configuration file, or manually entered into matrix.xml. Examples of the performance data we have collected for the fast track application is given in Deliverable 2.2. Note that the end-user does not need to define a primary or auxiliary model in a ES pattern in the information supplied to the multiscale computing pattern software; it is automatically determined by the multiscale computing pattern software from the topology and the performance data.

The outputs of the performance service are configuration files and execution scripts for QCG-Broker. These configuration scripts give several options for the multiscale simulations, which are decided by the multiscale computing pattern software to be have the highest performance. Currently performance is defined as a combination of time to solution and parallel efficiency, although in the future this will include other factors, such as energy consumption. The QCG configuration files are then submitted, with the QCG-Broker deciding on the resource on to submit the multiscale simulation to. In the case of the BAC (replica computing pattern), the QCG-Broker can send replicas to be run over multiple resources if there is a limit in the number of jobs that can be concurrently run on the selected resource.

3 Plans for phase 2 (M18-M36) of the project

In the previous section, we have detailed the instantiation of the fast-track applications. The deep-track applications will be the focus for M18-36, especially those in the Heterogeneous Multiscale Computing (HMC) pattern, which requires a Heterogeneous Multiscale Manager (HMM) manager and/or an on-the-fly database to orchestrate the pattern.
The deep-track applications have been described in Deliverable 3.1. In the following we give an update and the plans for the next reporting period:

- Nanomaterials (HMC pattern)

Within the realm of nanomaterials, we are currently developing the scientific coupling between atomistic and finite-element simulations. This will allow us to simulate nano-composite materials up to macroscopic length and time scales. It requires the finite-element simulations to be parameterized by atomistic simulations as and when they are required (through the polling of a database of parameters). This will be a test-case for the HMC pattern, as the number of atomistic simulations will be unknown until execution time (*i.e.* dynamic). We envisage the finite-element code will be paused or check-pointed when the atomistic simulations are created, allowing the scheduling and submission of the atomistic simulations to be determined by the multiscale computing pattern software.

- Astrophysics (ES / HMC pattern)

The Astrophysical simulations performed in the ComPat project by project partners at University of Leiden are currently coupled using the AMUSE coupling library, which has been demonstrated to efficiently couple up to 7 different submodels [11]. The astrophysics application therefore is a test-case for the QCG tools to use coupling libraries other than MUSCLE. Work has commenced on executing AMUSE applications on the EEE during the current reporting period. We can currently submit AMUSE jobs on the PSNC machines using the QCG submission tools (WP4). The multiscale simulations that utilize AMUSE use MPI_comm_spawn to create new MPI processes when required by the coupled codes. This means the number of MPI processes is not known at start up. It is therefore a challenge for a performance model to predict this behaviour; there are additional difficulties in scheduling such a setup on a HPC resource when a dynamic number of instances are required, within iterations of running submodels (normally called using MPI_comm_spawn). The Astrophysics applications using AMUSE will therefore be a test of our scheduling / coordination tools.

- Aneurysm Flow Dynamics (RC / ES pattern)

Computational Fluid Dynamics (CFD) has been proposed as a tool for informing clinical treatment decisions for aneurysms. We use the Lattice Boltzmann code HemeLB [12], and in the upcoming period we will focus on the introduction of flow diverters within blood vessels to explore the long range effects of their introduction, such as likely candidate sites for new aneurysm formation. This will involve the running of many replicas to deal with uncertainties in medical input data, and to span the physiological range of heart rates in the patient post-intervention. This work would be difficult to schedule efficiently without the ComPat tools being developed for the RC pattern. We also plan to use the ES pattern, with HemeLB as the primary model simulating the stresses of the blood flow on the inserted stent, coupled to the 'stent builder' (developed during this project) as the auxiliary model, modifying the shape of the stent. The efficient running of a stent deployment simulation greatly benefits from use of the ComPat tools, particularly due to the very different computational requirements of the primary and auxiliary models in this case, and naturally fits into the ComPat paradigm.

- In-Stent Restenosis (ES / RC)

Restenosis is governed by several interconnected processes, and the model developed by ITMO studies the cell growth in the vessel wall, drug diffusion from the stent struts and the effects of the surrounding geometry and blood flow on its cell growth. In Deliverable 3.1 we have described in some detail our model and its application as an Extreme Scaling HPMC pattern.
This model contains a multitude of parameters, most of which are selected based on experimental data. To study the sensitivity of the growth process on these parameters, we have performed a simulation in spirit of the Replica Computing pattern. We have not yet actually used RC pattern services as described in Deliverable 2.2, this we will do in Phase 2 of the project. More details about the in-stent restenosis is given in Annex 5.5.

- Ligand – Drug Binding Affinities (RC / HMC)

The BAC has been used as a fast-track application, but it is part of our future plans that we extend the BAC methodology to a range of new use-cases and create more complicated workflows. We envisage re-simulation and chemical modification of promising drug candidates, which will require multiscale coupling to quantum simulations for ligand parameterization. We have identified ChemShell [13] as a promising candidate code for quantum simulation and is hosted at the ComPat partner STFC. ChemShell has also tested for integration with the profiling tools described in Deliverable 4.2. This will be an example of the HMC pattern, with the possibility of two different databases: for the parameterisation of the classical molecular dynamics atoms and for the chemical modification itself.

- Global Turbulence application

In the context of deep-track applications, very minimalistic runs with a gyrokinetic turbulence model (5D) have been demonstrated with a fluxtube local code implemented as a single program (still following the ES pattern and implemented in MUSCLE2). The next steps, after linking to the pattern services and the new QCG, will be dedicated to bring deep-track applications to production. This will consist of modifying the fluxtube code implementation to run fluxtubes independently as several instances of the program, in order to fit in with the HMC pattern (we understand what are the required changes, if no issues are discovered we should be able to have a first prototype ready in a few weeks). The other parallel task is to complete the addition of the global gyrokinetic code into our MUSCLE2 framework (a prototype is almost ready but we foresee more testing/debugging then). Finally, to be able to run these deep-track application on resolutions where it makes sense, substantial computing allocations should be purchased (on and outside EEE).

4 Conclusions

In the document, we report on the successful instantiation of the Extreme Scaling and Replica Computing patterns using fast-track applications on the EEE, and describe the chain of actions required by the end-user to achieve this. It provides a snapshot of current progress in Task 3.2: “Instantiation and validation of multiscale computing patterns (M12-M24)” and offers a guide for instantiation new applications.

Combined with Deliverable 2.2, we have detailed the process of transforming an application from an abstract description at the level of xMML / task graph into configuration files / execution recipes with optimal performance.

The division into fast-track and deep-track applications has allowed us to develop and refine not only the ComPat infrastructure and technology efficiently. For example, as described in Deliverable 5.2, we can now run replicas in the RC pattern over multiple resources. It has also allowed us to consider new and novel multiscale scenarios for grand challenge applications (for deep-track applications) and develop the scientific advances required. In this document, we describe how we envisage the developments and instantiations of the deep-track applications.

Alongside instantiating the deep-track applications, the next reporting period will involve collaborations with WP2 to refine the multiscale computing pattern software, including automating the performance data collection (with WP4), and testing different metrics within the performance model (such as energy consumption).

5 Annexes

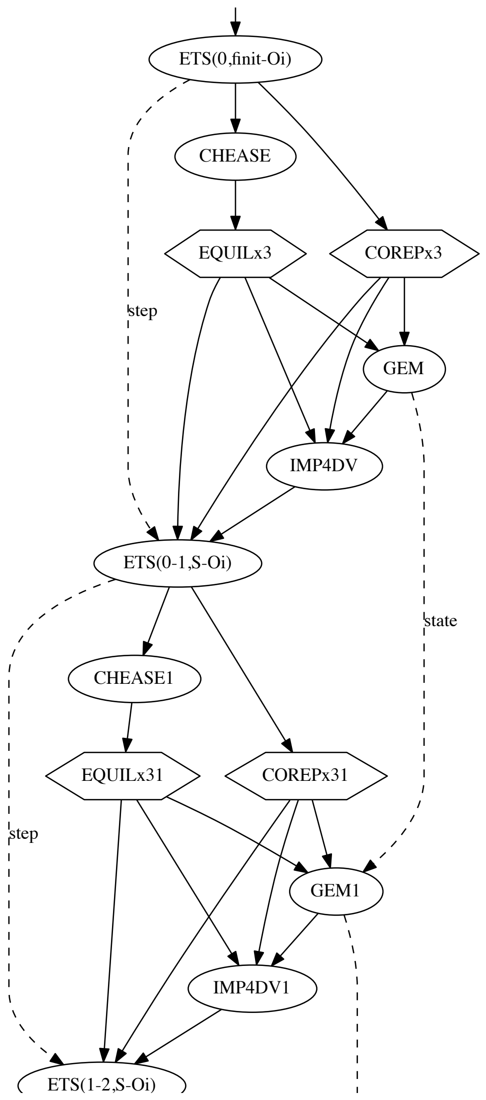
5.1 xMML descriptions and task graphs

The xMML description for all three fast-track applications, along with the associated task-graphs (created using the jMML programme developed during the MAPPER project https://github.com/blootsvoets/jmml/).

1. Global turbulence:

```
<?xml version='1.0' encoding='UTF-8'?>
<model id="TTE" name="turbulence-transport-equilibrium workflow" xmml_version="0.4" xmlns="http://www.mapper-project.eu/xmml">
 <!--
    xMML description of Fusion coupled application composed of
    turbulence-transport-equilibrium submodels.
    @version: 0.2
    @author: olivier.hoenen@ipp.mpg.de
 -->
 <description>
  The application simulates the time evolution of electron and ion
  temperature profiles (macro) in the core of the plasma and the effects
  of turbulence at micro scales. The application consists of 3 submodels
  (transport, equilibrium and turbulence) and one numerical tool (to convert
  heat fluxes coming from the turbulence submodel into coefficients taken
  into account by the transport submodel to evolve the temperature profiles).
  The turbulence code (GEM, 3D gyrofluid fluxtube approximation) is our
  primary submodel and all other codes (ETS for 1D transport, CHEASE for
  2D equilibrium and imp4dv to convert fluxes) are auxiliaries in this
    extreme scaling scenario.
 </description>

 <definitions>

  <mapper id="dupEquil" type="fan-out">
    <ports>
  <in id="equilibrium_in" />
  <out id="equilibrium_out1"/>
  <out id="equilibrium_out2"/>
  <out id="equilibrium_out3"/>
    </ports>
  </mapper>

  <mapper id="dupCorep" type="fan-out">
    <ports>
  <in id="coreprof_in"/>
  <out id="coreprof_out1"/>
  <out id="coreprof_out2"/>
  <out id="coreprof_out3"/>
    </ports>
  </mapper>

  <submodel id="init">
    <timescale delta="0" total="0"/>
    <spacescale delta="0" total="0"/>
    <ports>
  <out id="inputCPOs" operator="Of"/>
    </ports>
  </submodel>

  <submodel id="transp">
    <timescale delta="1E-0" total="1E+1"/>
    <spacescale delta="1E-2" total="1E+1"/>
    <ports>
  <in id="inputCPOs" operator="finit"/>
  <in id="coreprof_in" operator="S"/>
  <in id="coretransp_in" operator="S"/>
  <in id="equilibrium_in" operator="S"/>
  <out id="coreprof_out" operator="Oi"/>
  <out id="equilibrium_out" operator="Oi"/>
    </ports>
  </submodel>

  <submodel id="turb" stateful="yes">
    <timescale delta="1E-6" total="1E-4"/>
    <spacescale delta="1E-3" total="1E-2"/>
    <ports>
  <in id="coreprof_in" operator="finit"/>
  <in id="equilibrium_in" operator="finit"/>
  <out id="coretransp_out" operator="Of"/>
    </ports>
  </submodel>

  <submodel id="equil">
    <timescale delta="0" total="0"/>
    <ports>
  <in id="equilibrium_in" operator="finit"/>
  <out id="equilibrium_out" operator="Of"/>
    </ports>
  </submodel>

  <submodel id="f2dv">
    <timescale delta="0" total="0"/>
    <ports>
  <in id="coreprof_in" operator="finit"/>
  <in id="equilibrium_in" operator="finit"/>
  <in id="coretransp_in" operator="finit"/>
  <out id="coretransp_out" operator="Of"/>
    </ports>
  </submodel>

</definitions>

<topology>
  <instance id="CONTINUE" submodel="init"/>
  <instance id="ETS" submodel="transp"/>
  <instance id="GEM" submodel="turb"/>
  <instance id="CHEASE" submodel="equil"/>
  <instance id="EQUILx3" mapper="dupEquil"/>
  <instance id="COREPx3" mapper="dupCorep"/>
  <instance id="IMP4DV" submodel="f2dv"/>

  <coupling from="CONTINUE.inputCPOs" to="ETS.inputCPOs"/>

  <coupling from="ETS.coreprof_out" to="COREPx3.coreprof_in"/>
  <coupling from="ETS.equilibrium_out" to="CHEASE.equilibrium_in"/>

  <coupling from="COREPx3.coreprof_out1" to="GEM.coreprof_in"/>
  <coupling from="COREPx3.coreprof_out2" to="IMP4DV.coreprof_in"/>
  <coupling from="COREPx3.coreprof_out3" to="ETS.coreprof_in"/>

  <coupling from="CHEASE.equilibrium_out" to="EQUILx3.equilibrium_in"/>

  <coupling from="EQUILx3.equilibrium_out1" to="GEM.equilibrium_in"/>
  <coupling from="EQUILx3.equilibrium_out2" to="IMP4DV.equilibrium_in"/>
  <coupling from="EQUILx3.equilibrium_out3" to="ETS.equilibrium_in"/>

  <coupling from="GEM.coretransp_out" to="IMP4DV.coretransp_in"/>
```

```
    <coupling from="IMP4DV.coretransp_out" to="ETS.coretransp_in"/>
  </topology>

</model>
```



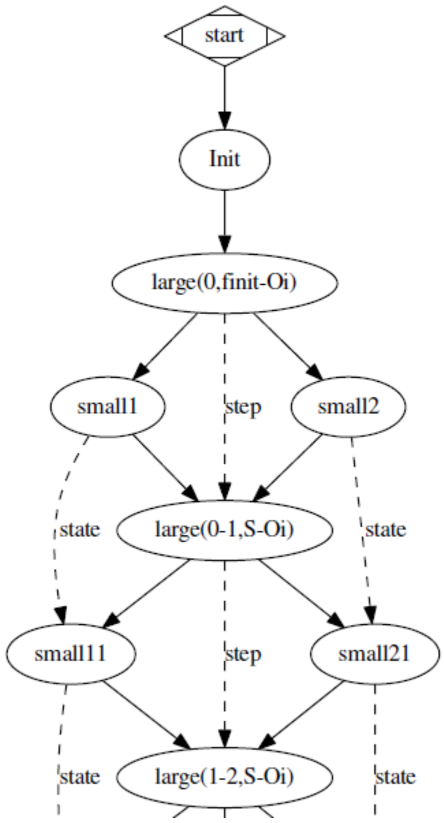
2. BAC:

```
<?xml version="1.0"?>
<model id="BAC" name="BAC model" version="1.0" xmml_version="0.4" xmlns="http://www.mapper-project.eu/xmml">
   <description>
BAC- 16 replicas, specified using the “multiplicity” attribute. Each replica contains a workflow of NAMD simulation -> Amber simulation
     </description>

     <definitions>

<submodel id="NAMD " name="NAMD ">
      <timescale delta="1E-6" total="1E-4"/>
      <spacescale delta="1E-3" total="1E-2"/>
<ports>
     <out id="toAmber" operator="Of" />
</ports>
</submodel>
<submodel id="Amber" name="Amber" >
     <timescale delta="1E-6" total="1E-4"/>
    <spacescale delta="1E-3" total="1E-2"/>
    <ports>
```

```
  <in id="toAmber" operator="finit" />
</ports>
    </submodel>
</definitions>

<topology>
  <instance id="NAMD " submodel="NAMD " multiplicity="16"/>
  <instance id="Amber" submodel="Amber" multiplicity="20"/>
  <coupling from="NAMD.toAmber" to="Amber.toAmber"/>

</topology>
</model>
```

3. RBC – Sandbox:

```
<!--
    xMML description of ISR coupled application composed of 3 models.
    @version: 0.1
    @author: Saad Alowayyed

-->
<model id="SandBox" name="SandBox Application" xmml_version="0.4" xmlns="http://www.mapper-project.eu/xmml">
  <description>
    The sandbox simulates couples Palabos with Ficsion.
  </description>

  <definitions>

    <submodel id="Init">
      <timescale delta="0" total="0"/>
      <spacescale delta="0" total="0"/>
      <ports>
    <out id="Init2large" operator="Of"/>
      </ports>
    </submodel>

    <submodel id="large">
      <timescale delta="1E-0" total="1E+1"/>
      <spacescale delta="1E-2" total="1E+1"/>
      <ports>
    <in id="Init2large" operator="finit"/>
    <out id="largeOi" operator="Oi"/>
    <in  id="largeS" operator="S"/>
      </ports>
    </submodel>
    <submodel id="small1" stateful="yes">
      <timescale delta="1E-6" total="1E-4"/>
      <spacescale delta="1E-3" total="1E-2"/>
      <ports>
    <in  id="small1finit" operator="finit"/>
    <out id="small1Of" operator="Of"/>
      </ports>
    </submodel>
    <submodel id="small2" stateful="yes">
      <timescale delta="1E-6" total="1E-4"/>
      <spacescale delta="1E-3" total="1E-2"/>
      <ports>
    <in  id="small2finit" operator="finit"/>
    <out id="small2Of" operator="Of"/>
      </ports>
    </submodel>
  </definitions>

  <topology>
    <instance id="Init" submodel="Init"/>
    <instance id="large" submodel="large"/>
    <instance id="small1" submodel="small1"/>
    <instance id="small2" submodel="small2"/>
    <coupling from="Init.Init2large" to="large.Init2large"/>
    <coupling from="large.largeOi" to="small2.small2finit"/>
    <coupling from="large.largeOi" to="small1.small1finit"/>
    <coupling from="small1.small1Of" to="large.largeS"/>
    <coupling from="small2.small2Of" to="large.largeS"/>


  </topology>

</model>
```



5.2 YaML files for the BAC application

```
production:
  local_configs: "/net/dirac/mnt/store6/dave/janssen/out-stage/"
  model: "NAMD"
  name : namd
  file_type: URL
  benchmark_scaling: 1
  iterations_scaling: 1
  dependencies: namd
  restrictions:
    cpu:
      min: 1
      max: 512
  execution:
    arguments:
      script:
        "cd input/mineq_confs\n
        namd_exec eq0-rep${PS_rep}.conf > ../replicas/rep${PS_rep}/equilibration/eq0.log\n
        namd_exec eq1-rep${PS_rep}.conf > ../replicas/rep${PS_rep}/equilibration/eq1.log\n
        namd_exec eq2-rep${PS_rep}.conf > ../replicas/rep${PS_rep}/equilibration/eq2.log\n
        cd ../../sim_confs\n
    namd_exec sim1-rep${PS_rep}.conf > ../../input/replicas/rep${PS_rep}/simulation/sim${PS_rep}.log"
   value: text
production2:
 local_results: "~/work/fabsim/FabSim/results"
 model: "Amber"
 name: mmpbsa
 restrictions:
  cpu:
   min: 1
   max: 1
   number: 1
 dependencies: amber
 execution:
  arguments:
   script:
    "cd rep${PS_rep}/fe-calc
mpirun -n ${QCG_PROCS} MMPBSA.py.MPI -i ../../nmode.in -sp ../../build/complex.top -cp ../../build/com.top -rp
../../build/rec.top -lp ../../build/lig.top -y ../simulation/sim1.dc/ > amber.log"
```

5.3 Matrix.xml and Multiscale.xml

Note, templates for matrix.xml and multiscale.xml are also reported in Deliverable 2.2

1. Fusion application:

Matrix.xml:

```
<singlescale>
  <submodels>
    <submodel name="init" class="NativeKernel">
      <instance name="CONTINUE">
        <restrictions>
          <cpu number="1"/>
        </restrictions>

        <available_resources>
          <resource name="supermuc" nodeType="thin"/>
                                                  <resource name="eagle" nodeType="haswell_128"/>
        </available_resources>

      </instance>
    </submodel>

    <submodel name="f2dv" class="NativeKernel">
      <instance name="IMP4DV">
        <restrictions>
          <restrictions>
          <cpu number="1"/>
        </restrictions>

        <available_resources>
          <resource name="supermuc" nodeType="thin"/>
                                                  <resource name="eagle" nodeType="haswell_128"/>
        </available_resources>

      </instance>
    </submodel>

                  <submodel name="dupEquil" class="DuplicationMapper">
      <instance name="EQUILx3">
        <restrictions>
          <restrictions>
          <cpu number="1"/>
        </restrictions>

        <available_resources>
```

```
            <resource name="supermuc" nodeType="thin"/>
                                                        <resource name="eagle" nodeType="haswell_128"/>
        </available_resources>

    </instance>
</submodel>

                    <submodel name="dupCorep" class="DuplicationMapper">
    <instance name="COREPx3">
        <restrictions>
            <restrictions>
            <cpu number="1"/>
        </restrictions>

        <available_resources>
            <resource name="supermuc" nodeType="thin"/>
                                                        <resource name="eagle" nodeType="haswell_128"/>
        </available_resources>

    </instance>
</submodel>

                    <submodel name="transp" class="NativeKernel">
    <instance name="ETS">
        <restrictions>
            <cpu>
                <number> 1 </number>
                <min_cores> 1 </min_cores>
                <max_cores> 1 </max_cores>
            </cpu>
        </restrictions>

        <benchmark_input>
            <file type="URL">gsiftp://input_to_ETS.xml</file>
            <iterations> 1000 </iterations>
        </benchmark_input>
        <scalability_formula> NY </scalability_formula>

        <available_resources>
            <resource name="supermuc" nodeType="thin"/>
                                                        <resource name="eagle" nodeType="haswell_128"/>
        </available_resources>

    </instance>
</submodel>

                    <submodel name="equil" class="NativeKernel">
    <instance name="CHEASE">
        <restrictions>
            <cpu>
                <number> 1 </number>
                <min_cores> 1 </min_cores>
                <max_cores> 1 </max_cores>
            </cpu>
        </restrictions>

        <benchmark_input>
            <file type="URL">gsiftp://input_to_CHEASE.xml</file>
            <iterations> 1000 </iterations>
        </benchmark_input>
        <scalability_formula> NY </scalability_formula>

        <available_resources>
            <resource name="supermuc" nodeType="thin"/>
                                                        <resource name="eagle" nodeType="haswell_128"/>
        </available_resources>
        </instance>
    </submodel>

                        <submodel name="turb" class="MPIKernel">
        <instance name="GEM">
            <restrictions>
                <cpu>
                    <number> (2^x)*k:k=8 </number>
                    <min_cores> 64 </min_cores>
                    <max_cores> 2048 </max_cores>
                </cpu>
            </restrictions>

            <benchmark_input>
                <file type="URL">gsiftp://input_to_GEM.xml</file>
                <iterations> 1000 </iterations>
            </benchmark_input>
            <scalability_formula> NY </scalability_formula>

            <available_resources>
                <resource name="supermuc" nodeType="thin"/>
                                                        <resource name="eagle" nodeType="haswell_128"/>
            </available_resources>

        </instance>
    </submodel>

<performance>
    <submodel name="transp">
        <instance name="ETS">
            <resources name="supermuc">
                <nodeType> thin </nodeType>
                <numberOfCores> 1</numberOfCores>
                <wallClockTime> 1.3</wallClockTime>
                                                        <numberOfNodes> 0 </numberOfNodes> <!-- because it is a helper -->

            </resources>

            <resources name="eagle">
                <nodeType> haswell_128 </nodeType>
                <numberOfCores> 1 </numberOfCores>
                <wallClockTime> 1.0 </wallClockTime>
                                                        <numberOfNodes> 0 </numberOfNodes>
            </resources>

        </instance>
    </submodel>

                        <submodel name="equil">
        <instance name="CHEASE">
            <resources name="supermuc">
                <nodeType> thin </nodeType>
                <numberOfCores> 1 </numberOfCores>
                <wallClockTime> 3.9 </wallClockTime>
                                                        <numberOfNodes> 0 </numberOfNodes>

            </resources>

            <resources name="eagle">
                <nodeType> haswell_128 </nodeType>
                <numberOfCores> 1 </numberOfCores>
                <wallClockTime> 2.6 </wallClockTime>
                                                        <numberOfNodes> 0 </numberOfNodes>
            </resources>

        </instance>
    </submodel>
```

```
                    <submodel name="turb">
          <instance name="GEM">
            <resources name="supermuc">
              <nodeType> thin </nodeType>
              <numberOfCores> 64;128;256;512;1024;2048</numberOfCores>
                                                  <wallClockTime> 767.9; 408.5; 197.2; 93.7; 45.2; 32.4</wallClockTime>
                                                  <numberOfNodes> 4;8;16;32;64;128</numberOfNodes>

            </resources>

            <resources name="eagle">
              <numberOfCores> 64;128;256;512;1024;2048</numberOfCores>
              <wallClockTime> 863; 517; 255; 85; 44; 34</wallClockTime>
                                                  <numberOfNodes> 3;5;10;19;37;74</numberOfNodes>
            </resources>

          </instance>
        </submodel>

    </performance>
</singlescale>
```

multiscale.xml

```
<multiscale>
  <info>
    <job appID="compat-test" project="compat">
      <computing> ES </computing>
      <modeltime> ETS + CHEASE + GEM </modeltime>
      <task persistent="true" taskId="ES-task-1">

        <numberofcores>
          <min> 64 </min>
          <max> 2048 </max>
        </numberofcores>
      </task>
    </job>
  </info>

 <topology>
  <instance id="CONTINUE" helper="init"/>
  <instance id="ETS"      helper="transp"/>
  <instance id="GEM"      submodel="turb"/>
  <instance id="CHEASE"   helper="equil"/>
  <instance id="EQUILx3"  helper="dupEquil"/>
  <instance id="COREPx3"  helper="dupCorep"/>
  <instance id="IMP4DV"   helper="f2dv"/>

  <coupling from="CONTINUE" to="ETS"/>

  <coupling from="ETS"      to="COREPx3"/>
  <coupling from="ETS"      to="CHEASE"/>

  <coupling from="COREPx3"  to="GEM"/>
  <coupling from="COREPx3"  to="IMP4DV"/>
  <coupling from="COREPx3"  to="ETS"/>

  <coupling from="CHEASE"   to="EQUILx3"/>

  <coupling from="EQUILx3"  to="GEM"/>
  <coupling from="EQUILx3"  to="IMP4DV"/>
  <coupling from="EQUILx3"  to="ETS"/>

  <coupling from="GEM"      to="IMP4DV"/>
  <coupling from="IMP4DV"   to="ETS"/>

  </topology>

  <middleware name="QCG">

     <execution type="compat">
        <executable>
           <application name="muscle2" version="compat-1.2"/>
        </executable>

        <arguments>
           <value>short.cxa.rb</value>
        </arguments>

        <stdout>
           <directory>
              <location         type="URL">gsiftp://qcg.man.poznan.pl//home/plgrid-groups/plggcompat/Fusion/FastTrack/qcg-
test/results</location>
           </directory>
        </stdout>

        <stderr>
           <directory>
              <location         type="URL">gsiftp://qcg.man.poznan.pl//home/plgrid-groups/plggcompat/Fusion/FastTrack/qcg-
test/results</location>
           </directory>
        </stderr>

        <stageInOut>
           <file name="test.cxa.rb" type="in">
              <location         type="URL">gsiftp://qcg.man.poznan.pl//home/plgrid-groups/plggcompat/Fusion/FastTrack/qcg-
test/test.cxa.rb</location>
           </file>

           <file name="inputs.tgz" type="in">
              <location         type="URL">gsiftp://qcg.man.poznan.pl//home/plgrid-groups/plggcompat/Fusion/FastTrack/qcg-
test/inputs.tgz</location>
           </file>

           <file name="extract_inputs.sh" type="in">
              <location         type="URL">gsiftp://qcg.man.poznan.pl//home/plgrid-groups/plggcompat/Fusion/FastTrack/qcg-
test/extract_inputs.sh</location>
           </file>

           <directory name="outputs" type="out">
              <location         type="URL">gsiftp://qcg.man.poznan.pl//home/plgrid-groups/plggcompat/Fusion/FastTrack/qcg-
test/results</location>
           </directory>
        </stageInOut>

        <environment>
           <variable name="QCG_MODULES_LIST">compat/app/fusion</variable>
           <variable name="QCG_PREPROCESS">extract_inputs.sh</variable>
        </environment>

     </execution>
  </middleware>
</multiscale>
```

2. BAC

Matrix.xml:

```
<singlescale>
  <submodels>
     <submodel name="namd_kernel" class="MPIKernel">
        <instance name="namd">
```

```
            <restrictions>
              <cpu>
                <number> NA </number>
                <min_cores> 1 </min_cores>
                <max_cores> 2048 </max_cores>
              </cpu>
            </restrictions>

            <benchmark_input>
              <file type="URL">gsiftp://qcg.man.poznan.pl:~/bac/input</file>
                                                <script>
                                                        <![CDATA[ cd input/mineq_confs
                                                        qcg_app          namd          eq0-benchmark.conf          >
../replicas/rep0/equilibration/eq0.log
                                                        ]]>
                                                </script>
                                                <iterations> 100 </iterations>
            </benchmark_input>
            <scalability_formula> NA </scalability_formula>

            <available_resources>
              <resource name="supermuc" nodeType="thin"/>
              <resource name="supermuc" nodeType="fat"/>
              <resource name="supermuc" nodeType="haswell"/>
            </available_resources>

          </instance>
        </submodel>

                        <submodel name="amber_kernel" class="MPIKernel">
          <instance name="amber">
            <restrictions>
              <cpu>
                <number> 1 </number>
                <min_cores> 1 </min_cores>
                <max_cores> 1 </max_cores>
              </cpu>
            </restrictions>

            <benchmark_input>
            </benchmark_input>
            <scalability_formula> NA </scalability_formula>

            <available_resources>
              <resource name="supermuc" nodeType="thin"/>
              <resource name="supermuc" nodeType="fat"/>
              <resource name="supermuc" nodeType="haswell"/>
            </available_resources>

          </instance>
        </submodel>

      </submodels>

              <performance>
        <submodel name="namd">
          <instance name="namd">
            <resources name="supermuc">
              <nodeType> thin </nodeType>
                                                <numberOfCores>
1;2;4;8;16;32;64;128;256;512;1024;2048</numberOfCores>
                                                <wallClockTime>
2265408;1187136;606528;290304;150336;77760;42508.8;21254.4;12960;7257.6;7776;8294.4</wallClockTime>
                                                <numberOfNodes> 1;1;1;1;1;2;4;8;16;32;64;128</numberOfNodes>

              <nodeType> fat </nodeType>
              <numberOfCores> 1;2;4;8;16;32;40;80;160;320;640;1280</numberOfCores>
```

```
                                        <wallClockTime>
2336947.2;1164844.8;583718.4;297043.2;153446.4;88128;71539.2;38880;20736;13478.4;10886.4;13996.8</wallClockTime
>
                                        <numberOfNodes> 1;1;1;1;1;1;1;2;4;8;16;32</numberOfNodes>

                                        <nodeType> haswell </nodeType>
          <numberOfCores> 1;2;4;8;16;28;56;112;224;448</numberOfCores>
                                        <wallClockTime>
2105222.4;1071532.8;539136;274752;141523.2;83462.4;42508.8;23328;12960;7257.6</wallClockTime>
                                        <numberOfNodes> 1;1;1;1;1;1;2;4;8;16</numberOfNodes>

        </resources>
      </instance>
    </submodel>

                <submodel name="amber">
      <instance name="amber">
        <resources name="supermuc">
          <nodeType> thin </nodeType>
                                        <numberOfCores> 1 </numberOfCores>
                                        <wallClockTime> 1200 </wallClockTime>
                                        <numberOfNodes> 1 </numberOfNodes>

          <nodeType> fat </nodeType>
          <numberOfCores> 1 </numberOfCores>
                                        <wallClockTime> 1200 </wallClockTime>
                                        <numberOfNodes> 1 </numberOfNodes>

                                        <nodeType> haswell </nodeType>
          <numberOfCores> 1 </numberOfCores>
                                        <wallClockTime> 1200 </wallClockTime>
                                        <numberOfNodes> 1 </numberOfNodes>
        </resources>
      </instance>
    </submodel>
        </performance>
</singlescale>
```

Multiscale.xml

```
<multiscale>
  <info>
    <job appID="bac-compat" project="compat">
      <computing> RC </computing>
      <modeltime> </modeltime>

            <task taskId="namd" persistent="true">
        <numberofcores>
          <min> 1 </min>
          <max> 1024 </max>
        </numberofcores>
      </task>

                <task taskId="amber" extension="namd_PSit${PS_it}">
        <numberofcores>
          <min> 1 </min>
          <max> 1 </max>
        </numberofcores>
      </task>

    </job>
  </info>

        <topology>
                <instance id="namd" submodel="namd"/>
                <instance id="amber" submodel="amber"/>
        </topology>
    <middleware name="QCG">

        <task taskId="namd">
                <execution type="compat">
                        <executable>
                                <application name="bash"/>
                        </executable>
                        <arguments>
                                <value>bac-namd.sh</value>
                                <value>${PS_rep}</value>
                        </arguments>
                        <stageInOut>
                                <file name="bac-namd.sh" type="in">
                                        <location      type="URL">gsiftp://qcg.man.poznan.pl/compat/bac-xml/muc-
bac-namd.sh</location>
                                </file>
                                <directory name="input" type="in">
                                        <location                  type="URL">gsiftp://qcg.man.poznan.pl/compat/bac-
xml/input</location>
                                </directory>
                                <file name="qcg.debug" type="out">
                                        <location                  type="URL">gsiftp://qcg.man.poznan.pl/compat/bac-
xml/output/output-muc-${JOB_ID}-namd-rep${PS_rep}.qcg.debug</location>
                                </file>
                                <file name="_stdouterr" type="out">
                                        <location                  type="URL">gsiftp://qcg.man.poznan.pl/compat/bac-
xml/output/output-muc-${JOB_ID}-namd-rep${PS_rep}.stdouterr</location>
                                </file>
                        </stageInOut>
                </execution>
                <executionTime useReservation="false">
                        <!-- walltime 1h for this task -->
                        <executionDuration>P0Y0M0DT3H0M</executionDuration>
                </executionTime>
                <parametersSweep>
                        <!-- this task will be replicated (parametrized) according to the 'rep' parameter
         (in this case 2 times) -->
                        <parameter>
                                <name>rep</name>
                                <value>
                                        <loop>
                                                <start>1</start>
                                                <end>2</end>
                                                <step>1</step>
                                                <decimalPlaces>0</decimalPlaces>
                                        </loop>
                                </value>
                        </parameter>
                </parametersSweep>
        </task>

        <task taskId="amber">
        <requirements>
                        <resourceRequirements>
                                <computingResource>
                                        <hostParameter name="queue">
                                                <stringValue value="test"/>
                                        </hostParameter>
                                </computingResource>
                        </resourceRequirements>
                        <patternTopology>
                                <kernels>
                                        <kernel id="amber_kernel"/>
                                </kernels>
                                <classes>
```

```
                                                <class id="c1">
                                                        <node host="supermuc">thin</node>
                                                </class>
                                        </classes>
                                        <plans>
                                                <plan id="plan1">
                                                        <criteria>
                                                                <time>P0Y0M0DT3H0M</time>
                                                        </criteria>
                                                        <group>
                                                                <kernel refid="amber_kernel">
                                                                        <class refid="c1">
                                                                                <cores>2</cores>
                                                                        </class>
                                                                </kernel>
                                                        </group>
                                                </plan>
                                        </plans>
                                </patternTopology>
                        </requirements>
                        <execution type="compat">
                                <executable>
                                        <application name="bash"/>
                                </executable>
                                <arguments>
                                        <value>bac-amber.sh</value>
                                        <value>${PS_rep}</value>
                                </arguments>
                                <stageInOut>
                                        <file name="bac-amber.sh" type="in">
                                                <location      type="URL">gsiftp://qcg.man.poznan.pl/compat/bac-xml/muc-
bac-amber.sh</location>
                                        </file>
                                        <directory name="input/replicas/rep${PS_rep}" type="out">
                                                <location                  type="URL">gsiftp://qcg.man.poznan.pl/compat/bac-
xml/output/output-muc-${JOB_ID}-rep${PS_rep}</location>
                                        </directory>
                                        <file name="qcg.debug" type="out">
                                                <location                  type="URL">gsiftp://qcg.man.poznan.pl/compat/bac-
xml/output/output-muc-${JOB_ID}-amber-rep${PS_rep}.qcg.debug</location>
                                        </file>
                                        <file name="amber.log" type="out">
                                                <location                  type="URL">gsiftp://qcg.man.poznan.pl/compat/bac-
xml/output/output-muc-${JOB_ID}-amber-rep${PS_rep}.log</location>
                                        </file>
                                        <file name="_stdouterr" type="out">
                                                <location                  type="URL">gsiftp://qcg.man.poznan.pl/compat/bac-
xml/output/output-muc-${JOB_ID}-amber-rep${PS_rep}.stdouterr</location>
                                        </file>
                                </stageInOut>
                        </execution>
                        <executionTime useReservation="false">
                                <executionDuration>P0Y0M0DT3H0M</executionDuration>
                        </executionTime>
                        <workflow>
                                <!-- the 'amber' task will start after 'namd' successfully finish;
                each 'amber' parameter sweep task is started after coresponding 'namd'
                task ('rep' variable) -->
                                <parent triggerState="FINISHED">namd_PSit${PS_it}</parent>
                        </workflow>
                        <parametersSweep>
                                <parameter>
                                        <name>it</name>
                                        <value>
                                                <loop>
                                                        <start>0</start>
                                                        <end>1</end>
                                                        <step>1</step>
                                                        <decimalPlaces>0</decimalPlaces>
                                                </loop>
                                        </value>
                                </parameter>
                                <parameter>
                                        <name>rep</name>
                                        <value>
                                                <loop>
                                                        <start>1</start>
                                                        <end>2</end>
                                                        <step>1</step>
                                                        <decimalPlaces>0</decimalPlaces>
                                                </loop>
                                        </value>
                                </parameter>
                        </parametersSweep>
                </task>

           </middleware>
        </multiscale>
```

3. Red Blood Cells (SandBox)

Matrix.xml:

```
<singlescale>
   <submodels>
      <submodel name="large" class="MPIKernel">
         <instance name="large">
            <restrictions>
               <cpu>
                  <number> x^2 </number>
                  <min_cores> 1 </min_cores>
                  <max_cores> 1024 </max_cores>
               </cpu>
            </restrictions>

            <benchmark_input>
               <file type="URL">gsiftp://config_large.xml</file>
               <iterations> 30 </iterations>
            </benchmark_input>
            <scalability_formula> x=2n </scalability_formula>

            <available_resources>
               <resource name="supermuc" nodetype="thin"/>
               <resource name="supermuc" nodetype="fat"/>
               <resource name="eagle" nodetype="haswell_128"/>
            </available_resources>

         </instance>
      </submodel>

                  <submodel name="small1" class="MPIKernel">
         <instance name="small1">
            <restrictions>
               <cpu>
                  <number> x^2 </number>
                  <min_cores> 1 </min_cores>
                  <max_cores> 128 </max_cores>
               </cpu>
            </restrictions>

            <benchmark_input>
               <file type="URL">gsiftp://config_small1.xml</file>
               <iterations> 30 </iterations>
            </benchmark_input>
```

```
    <scalability_formula> x=2n </scalability_formula>

    <available_resources>
     <resource name="supermuc" nodetype="thin"/>
     <resource name="supermuc" nodetype="fat"/>
     <resource name="eagle" nodetype="haswell_128"/>
    </available_resources>

   </instance>
  </submodel>

        <submodel name="small2" class="MPIKernel">
   <instance name="small2">
    <restrictions>
     <cpu>
      <number> x^2 </number>
      <min_cores> 1 </min_cores>
      <max_cores> 128 </max_cores>
     </cpu>
    </restrictions>

    <benchmark_input>
     <file type="URL">gsiftp://config_small2.xml</file>
     <iterations> 30 </iterations>
    </benchmark_input>
    <scalability_formula> x=2n </scalability_formula>

    <available_resources>
     <resource name="supermuc" nodetype="thin"/>
     <resource name="supermuc" nodetype="fat"/>
     <resource name="eagle" nodetype="huawei_128"/>
    </available_resources>

   </instance>
  </submodel>

 </submodels>



 <performance>
  <submodel name="large">
   <instance name="large">
    <resources name="supermuc">
     <nodeType> thin </nodeType>
                    <numberOfCores> 1;2;4;8;16;32;64;128;256;512;1024</numberOfCores>
                    <wallClockTime>
30565.2;19289.4;11043.5;6990.88;4080.57;2816.754;1719.19;1013.66;760.079;643.465;572.581</wallClockTime>
                    <numberOfNodes> 1;1;1;1;1;2;4;8;16;32;64</numberOfNodes>

     <nodeType> fat </nodeType>
     <numberOfCores> 1;2;4;8;16;32;64;128;256;512;1024</numberOfCores>
     <wallClockTime>
30565.2;19289.4;11043.5;6990.88;4442.67;4051.89;2798.78;1758.49;1509.41;1397.31;1280.1</wallClockTime>
                    <numberOfNodes> 1;1;1;1;1;1;2;4;7;13;26</numberOfNodes>
    </resources>

    <resources name="eagle">
     <numberOfCores> 1;2;4;8;16;32;64;128;256;512;1024</numberOfCores>
     <wallClockTime>
32840.4;11566.5;27399.93;15593.55;3485.49;2262.52;1370.95;1194.03;977.379;723.304;484.175</wallClockTime>
                    <numberOfNodes> 1;1;1;1;1;2;3;5;10;19;37</numberOfNodes>
    </resources>

   </instance>
  </submodel>
```

```
                    <submodel name="small1">
          <instance name="small1">
             <resources name="supermuc">
                <nodeType> thin </nodeType>
                                                   <numberOfCores> 1;2;4;8;16;32;64;128</numberOfCores>
                                                   <wallClockTime>
622.303;367.12;326.559;255.832;241.497;179.632;132.234;114.808 </wallClockTime>
                                                   <numberOfNodes> 1;1;1;1;1;2;4;8</numberOfNodes>

                <nodeType> fat </nodeType>
                <numberOfCores> 1;2;4;8;16;32;64;128</numberOfCores>
                <wallClockTime> 649.852;484.22;370.641;258.458;205.878;270.77;247.144;200.654</wallClockTime>
                                                   <numberOfNodes> 1;1;1;1;1;1;2;4</numberOfNodes>
             </resources>

             <resources name="eagle">
                <numberOfCores> 1;2;4;8;16;32;64;128</numberOfCores>
                <wallClockTime> 524.020;932.215;877.707;319.447;193.384;203.176;190.204;170.430</wallClockTime>
                                                   <numberOfNodes> 1;1;1;1;1;2;3;5</numberOfNodes>
             </resources>

          </instance>
       </submodel>

                    <submodel name="small2">
          <instance name="small2">
             <resources name="supermuc">
                <nodeType> thin </nodeType>
                                                   <numberOfCores> 1;2;4;8;16;32;64;128</numberOfCores>
                                                   <wallClockTime>                                   572.104
             368.71;328.52;259.108;242.092;180.597;133.321;115.15 </wallClockTime>
                                                   <numberOfNodes> 1;1;1;1;1;2;4;8</numberOfNodes>

                <nodeType> fat </nodeType>
                <numberOfCores> 1;2;4;8;16;32;64;128</numberOfCores>
                <wallClockTime> 700.051;482.63;368.68;255.182;205.283;269.805;246.057;200.312 </wallClockTime>
                                                   <numberOfNodes> 1;1;1;1;1;1;2;4</numberOfNodes>
             </resources>

             <resources name="eagle">
                <numberOfCores> 1;2;4;8;16;32;64;128</numberOfCores>
                <wallClockTime>
637.1855649063;414.5110108849;370.7904806239;322.1749814805;196.2847365702;212.0826437462;192.7695409192;23
4.8149867718 </wallClockTime>
                                                   <numberOfNodes> 1;1;1;1;1;2;3;5</numberOfNodes>
             </resources>

          </instance>
       </submodel>

    </performance>
</singlescale>
```

Multiscale.xml

```
<multiscale>
    <info>
       <job appID="compat-test" project="compat">
          <computing> ES </computing>
          <modeltime> large+max(small1,small2) </modeltime>
          <task persistent="true" taskId="ES-task-1">

             <numberofcores>
                <min> 1 </min>
                <max> 1024 </max>
             </numberofcores>
          </task>
       </job>
```

```
   </info>

  <topology>

            <instance id="Init" submodel="Init"/>
   <instance id="large" submodel="large"/>
   <instance id="small1" submodel="small1"/>
   <instance id="small2" submodel="small2"/>

            <coupling from="Init" to="large"/>
   <coupling from="large" to="small2"/>
   <coupling from="large" to="small1"/>
   <coupling from="small1" to="large"/>
   <coupling from="small2" to="large"/>

   </topology>

   <middleware name="QCG">

      <execution type="compat">
         <executable>
            <application name="muscle2" version="compat-1.1"/>
         </executable>

         <arguments>
            <value>sb.cxa.rb</value>
         </arguments>

         <stdout>
            <directory>
               <location type="URL">gsiftp://results/stdout/</location>
            </directory>
         </stdout>

         <stderr>
            <directory>
               <location type="URL">gsiftp://results/stderr/</location>
            </directory>
         </stderr>

         <stageInOut>
            <file name="sb.cxa.rb" type="in">
               <location type="URL">gsiftp:/./sb.cxa.rb</location>
            </file>

            <file name="inputs.tgz" type="in">
               <location type="URL">gsiftp:///inputs.tgz</location>
            </file>

            <file name="extract_inputs.sh" type="in">
               <location type="URL">gsiftp:///extract_inputs.sh</location>
            </file>

            <directory name="outputs" type="out">
               <location type="URL">gsiftp:/</location>
            </directory>
         </stageInOut>

         <environment>
            <variable name="QCG_MODULES_LIST">compat/apps/bio</variable>
            <variable name="QCG_PREPROCESS">extract_inputs.sh</variable>
         </environment>

      </execution>
   </middleware>
</multiscale>
```

5.4 Template cxa file

For the fusion / global turbulence application:

```
# Set class and library paths

# total simulation time
$env["max_timesteps"] = "1.0"

# declare kernels
init = Instance.new('init')
ETS = Instance.new('ETS')
GEM = Instance.new('GEM')
CHEASE = Instance.new('CHEASE')
equildup = Instance.new('equildup')
corepdup = Instance.new('corepdup')
coreprof2equilibrium = Instance.new('coreprof2equilibrium')

# parameters
init['dt']="0.0e0"
init['T']="0.0e0"
init['dx']="0.0e0"
init['X']="0.0e0"
ETS['dt']="1.0e0"
ETS['T']="1.0"
ETS['dx']="1.0e-2"
ETS['X']="1.0"
GEM['dt']="1.0e-6"
GEM['T']="1.0"
GEM['dx']="1.0e-3"
GEM['X']="1.0e-2"
CHEASE['dt']="0.0e0"
CHEASE['T']="1.0"
coreprof2equilibrium['dt']="0.0e0"
coreprof2equilibrium['T']="1.0"

# configure coupling
init.couple(ETS, 'initCPOs')
ETS.couple(corepdup, 'coreprof' => 'corep_in')
corepdup.couple(coreprof2equilibrium, 'corep_out1' => 'coreprof')
corepdup.couple(GEM, 'corep_out2' => 'coreprof')
coreprof2equilibrium.couple(CHEASE, 'equilibrium' => 'equilibrium_in')
CHEASE.couple(equildup, 'equilibrium_out' => 'equil_in')
equildup.couple(GEM, 'equil_out1' => 'equilibrium')
equildup.couple(ETS, 'equil_out2' => 'equilibrium')
GEM.couple(ETS, 'coretransp')
```

5.5 Additional Information on the ISR application (ITMO University)

Cardiovascular diseases are one of the leading causes of death in the world. Diseases of coronary arteries are an important subset of these diseases. They are often corrected by stenting the affected artery. One of the important complications of stenting is a development of a repeated narrowing, or restenosis. Restenosis is governed by several interconnected processes, and this model studies the cell growth in the vessel wall, drug diffusion from the stent struts and the effects of the surrounding geometry and blood flow on tis cell growth. In deliverable D3.1 we have described in some detail our model and its application as an Extreme Scaling multiscale application.

This model contains a multitude of parameters, most of which are selected based on experimental data. To study the sensitivity of the growth process on these parameters, we have performed a simulation in spirit of the Replica Computing pattern. We have not yet actually used RC pattern services as described in Deliverable 2.2, this we will do in phase 2 of the project.

The important parameters which were tested in this experiment are the percentage of pores (fenestrations) in the internal membrane and the threshold weight of the cell's neighbours *CI*, exceeding which stops the cell growth. The pore percentage is based upon the data about the concentration and the size of pores in experimental samples, and the *CI* is selected based on the cell configuration in the model and on the experimental end configuration. We have performed all these simulations on the Lomonosov supercomputer located at Moscow State University.

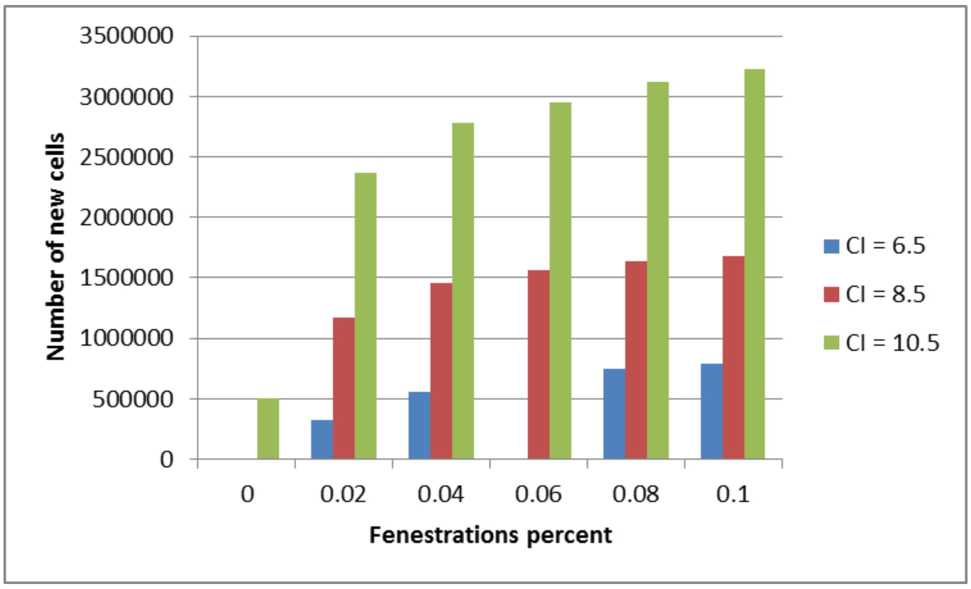


Figure 1 — endpoint number of new cells for different modelling parameters. No data for parameters (0.06, 6.5).

Figure 1 shows the results of modelling for different pairs of these parameters. The cell growth is estimated based on the amount of new cells in the vessel lumen. Typical stent deployment parameters are used: strut width of 0.2 mm and injury score (IS) 2, i.e. the membrane is stretched but not ruptured. The data for the fenestration percentage of 0.06 and CI = 6.5 is absent due to a technical error during the simulation.

The results show that for the pore percentage, the biggest difference in growth shows up at the moment of qualitative transition from the absence of pores to their presence, while increasing the number of pores has a much smaller impact. Without the pores, the cells can only move to the lumen through the ruptures in the IEL (which aren't present in this deployment configuration), so there is almost no growth for CI 6.5 and 8.5 (12 new cells for CI = 6.5 and 8025 for CI = 8.5). For CI = 10.5 a minor growth inside the vessel wall is observed. In the presence of the fenestrations, the CI value is much more important than the exact percentage of the fenestrations. For further modelling, a value of 4% fenestrations and CI = 10.5 were chosen, since they agree with the experiment the best.

It is commonly thought that the development of restenosis is greatly affected by the stent configuration (mainly strut width) as well as the stent deployment depth and vessel damage. To test the effect of these parameters in our model we have performed a trial using the replica computing pattern. We have tested three stent geometries and three injury scores that corresponded to three different deployment depths. At IS 1 the internal lamina is stretched at the angle less than 45°, at IS 2 this angle is between 45° and 90°, and at IS 3 the membrane ruptures.

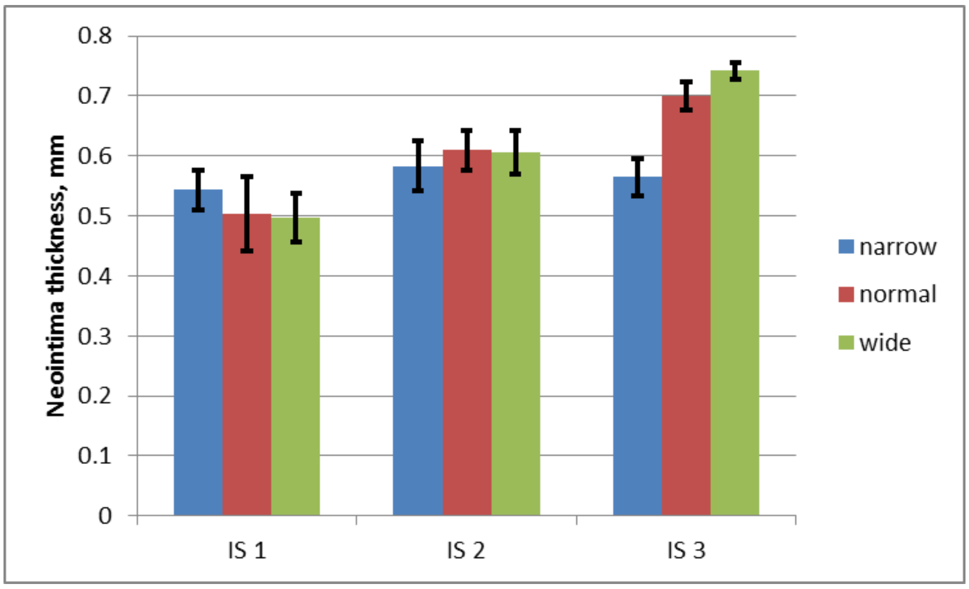


Figure 2 — the average end thickness of neointima over struts for different stenting conditions.

The resulting neointima thickness over stent struts is shown in Figure 2. A very narrow stent penetrates deeply into the tissue, and the dividing cells close up over it quickly, so the end result is almost the same for all cases. Wider stents are more sensitive to deployment depth, but the difference between the normal and the wide stent is only notable for severely damaged vessels.

In these experiments we have clarified the model parameters and have demonstrated the effects of stent configuration and deployment of neointima growth in the model.

6 References

[1] J. Borgdorff, M. Mamonski, B. Bosak, K. Kurowski, M. B. Belgacem, B. Chopard, D. Groen, P. V Coveney, A. G. Hoekstra, "Distributed Multiscale Computing with MUSCLE 2, the Multiscale Coupling Library and Environment", *Journal of Computational Science* 5 (2014) 719–731

[2] J. Borgdorff, M. Ben Belgacem, C. Bona-Casas, L. Fazendeiro, D. Groen, O. Hoenen, A. Mizeranschi, J. L. Suter, D. Coster, P. V. Coveney, W. Dubitzky, A. G. Hoekstra, P. Strand, and B. Chopard, "Performance of distributed multiscale simulations", *Philosophical Transactions of the Royal Society A*, 372, 20130407 (2014)

[3] https://github.com/blootsvoets/jmml/

[4] G. L. Falchetto, et al. "The European Integrated Tokamak Modelling (ITM) effort: achievements and first physics results." *Nuclear Fusion* 54.4 (2014) 043018.

[5] D. Wright, B. Hall, O. Kenway, S. Jha, P. V. Coveney, "Computing clinically relevant binding free energies of HIV-1 protease inhibitors" *Journal of Chemical Theory and Computation* 10 (2014) 1228-1241

[6] www.ks.uiuc.edu/Research/namd/, last accessed 28 Mar 2017

[7] ambermd.org, last accessed 28 Mar 2017

[8] www.palabos.org, last accessed 28 Mar 2017

[9] L. Mountrakis, E. Lorenz, O. Malaspinas, S. Alowayyed, B. Chopard, A. G. Hoekstra, "Parallel performance of an IB-LBM suspension simulation framework", *Journal of Computational Science* 9 (2015) 45–50.

[10] D. Groen, A. Bhati, J. Suter, J. Hetherington, S. Zasada, P. V. Coveney, "FabSim: facilitating computational research through automation on large-scale and distributed e-infrastructures", *Computer Physics Communications*, 207 (2016) 375–385

[11] F. L. Pelupessy, *et al.* "The astrophysical multipurpose software environment." *Astronomy & Astrophysics* 557 (2013) A84.

[12] D. Groen, J. Hetherington, H. B. Carver, R. W. Nash, M. O. Bernabeu, P. V. Coveney, "Analyzing and Modeling the Performance of the HemeLB Lattice-Boltzmann Simulation Environment", *Journal of Computational Science*, 4 (2013), 412–422

[13] chemshell.org, last accessed 29 Mar 2017