D7.2 Second report on adaptation of applications

Project acronym: *MAPPER*
Project full title: Multiscale Applications on European e-Infrastructures
Grant agreement no.: 261507

Due-Date:	M24
Delivery:	M24
Lead Partner:	UNIGE
Dissemination Level:	PU
Status:	Draft
Approved:	
Version:	1.3

DOCUMENT INFO

Date and version number	Author	Details
09-07-2012	B. Chopard	call for contributions
23-08-2011 v0.1	M. Ben Belgacem	first contribution from UNIGE
27-08-2011 v0.2	W. Dubitzky	contribution from UU
04-09-2012 v0.3	D. Groen	contribution from UCL
07-09-2012 v0.4	O. Hoenen & L. Fazendeiro	contribution for FUSION
11-09-2012 v0.5	M. Ben Belgacem	second contribution from UNIGE
18-09-2012 v1.0	B. Chopard	integration of all contributions
20-09-2012 v1.0	J. Borgdorff	contribution from UvA
21-09-2012 v1.1	B. Chopard	integration and editing
22-09-2012 v1.2	WP7 members	proof reading
24-09-2012 v1.3	Stefan	Quality board Reviewing

Contents

List of Tables

List of Figures

Part I
Executive summary

This deliverable provides a review of the work done in WP7 during the second year of the MAPPER project.

The main goal of WP7 is to adapt or develop a set of selected multiscale applications to the MAPPER framework. This amounts to expressing these applications in a Multiscale Modeling Language (MML), and to adapt the submodels in order to implement their mutual couplings. From that stage, the application can be run on a distributed computing infrastructure, such the European grid platforms.

Deliverable D7.2 discusses the status of adaptation of the selected applications, whether tightly or loosely coupled, as specified in task 7.1 and 7.2. Each application is described with respect to its level of integration as a “multiscale distributed application”.

In short, all applications have now been able to use the proposed multiscale methodology and have been able to reformulate the existing code (or develop new one) to integrate the coupling middleware. The tools provided by WP8 have been successfully used for achieving this goal.

Most applications could be executed on parts of the MAPPER infrastructure (a set distributed clusters managed by a grid environment). Therefore milestone 17 can be considered as attained. Production runs can be now envisaged and the computing infrastructure available for MAPPER will then be needed.

Another aspect of this deliverable concerns performance evaluation, namely task 7.3 that started at M18. This WP is specifically concerned with application performance rather than service or infrastructure performance. Our aim is to evaluate the cost/gain of the coupling methodology proposed by MAPPER at the user level. This typically includes a performance comparison of the monolithic code versus the ”MAPPERIZED” one, and an estimate of the time spent in coupling componants such as conduits and mappers. These measurements should be done on several hardware configurations like, for instance, a sequential processeur, a university cluster and, finally, a grid infrstructure.

The ultimate goal is to define a performance model which can predict the performance of a MAPPER application, knowing the performance of its submodels and the type of couplings between them.

This deliverable is the result of contributions from all partners involved in the applications development. UNIGE, as workpackage leader has supervised the integration of all parts of this document.

Part II
Report on the adaptation of applications

1 Simulation of Clay-polymer Nanocomposites (Nanomaterials)

1.1 Description

Layered mineral composites have a substantial potential impact in areas such as energy applications (oil industry additives), materials applications (nano-composite materials) and biomedical applications (drug delivery) [14]. The microscopic structure and mechanisms of layered nanomaterials operate over many different length scales, ranging from nanometers to microns. One of the key challenges in the simulation of such systems is efficiently sampling these scales to understand how the microscopic structure affects the macroscopic properties of the composite. To overcome this problem of multiple length scales, we develop a multiscale scheme where separate simulations at one length scale pass input parameters to higher length scales, starting from the electronic structure through classical atomistic molecular dynamics to coarse-grained models [39, 38]. In this study, we address the challenge of creating, deploying and executing this hierarchical multiscale modelling scheme to study the behavior of clay-polymer nanocomposites.

There are many factors, operating over various length scales, that can affect the mechanical properties of clay-polymer nanocomposites. For example, the orientation of the clay tactoids in the polymer matrix will affect the mechanical resistance of the composite, such as the transfer of stress, while at the shortest scale the molecular arrangement and the adhesion energy of the polymer molecules in the vicinity of the clay-polymer interface affect the overall mechanical properties of the system.

We have developed a hierarchical scheme which uses an acyclically coupled multiscale simulation mechanism that allows us to study and design of layered mineral composites. Our simulation is initiated at the quantum mechanical level, followed by fine-grained molecular dynamics simulations, in turn providing the input for course-grained molecular dynamics simulations (see Fig. 1). Several post-processing scripts are run between each level of the simulation to perform data conversion. These scripts may be preprogrammed, but can also (in some cases) be modified by the user between simulations. Because the simulations do not run concurrently, and the frequency (and required performance) of data exchange between subcodes is limited we use GridSpace to perform the acyclic coupling between the submodels [8].

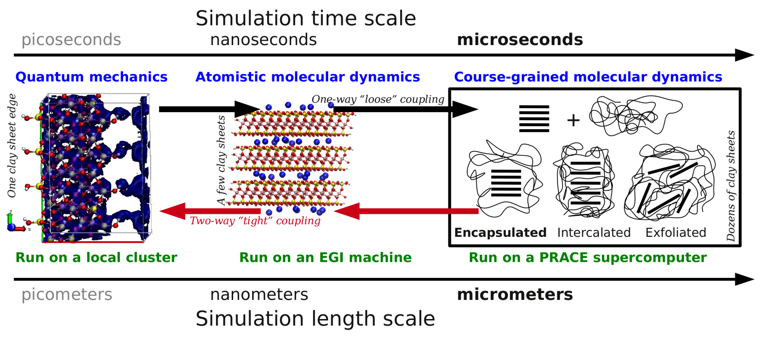

Figure 1: Brief graphical overview of the coupling within the nanomaterials application.

1.2 Status

We have developed a distributed acyclically coupled multiscale simulation using CPMD and LAMMPS, the latter used to model both atomistic and coarse-grained molecular dynamics. We demonstrated our scenario as part of the MAPPER Year 1 Review using the GridSpace, QosCosGrid and AHE tools from the project. Our efforts in the second year were primarily concentrated on bolstering the scientific strength of our multiscale simulation. We are working on more sophisticated quantum mechanical models, incorporating larger sections of clay sheets and more surrounding water for the calculation of the electrostatic potential. We also compare different approaches (e.g., constant volume vs. constant pressure) to obtain a more reliable energy profile from these calculations. On the larger scales we have run several large production simulations, using existing CPMD results, on the HECToR Supercomputer at EPCC in Edinburgh. Additionally, we have developed an improved polymer growth routine that we use between the quantum mechanical and atomistic simulation. Initializing a stable multiscale clay-polymer system is a scientifically challenging endeavour, as initial biases in the polymer distribution can have an adverse effect on the accuracy of the scientific result. Our work resulted in several publications in conferences [20, 38] and a magazine article [12], although the main body of our scientific output is currently being written up.

On the computational side we have made additional progress as well. Our scenario now works in conjunction with GridSpace and AHE, where the two tools are more reliably and robustly integrated than before. In addition, we have performed a number of performance tests (described in the next section) and are currently testing the codes for accuracy.

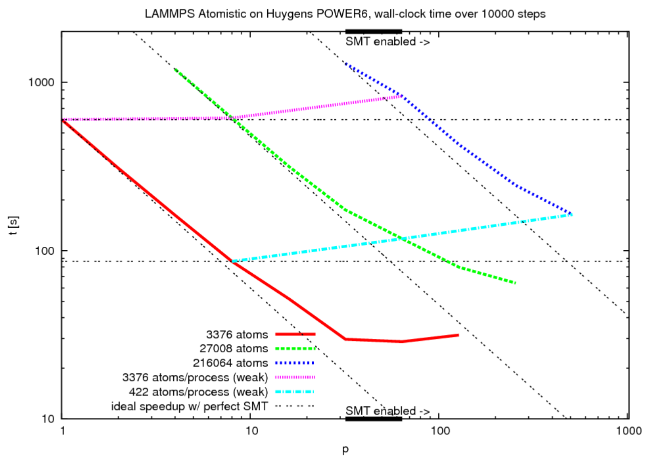

Figure 2: Benchmark results of running LAMMPS on Huygens in atomistic mode.

1.3 Performance Benchmarks

We have not performed scalability tests of CPMD, as we currently only calculate the potentials of a single clay sheet edge within our multiscale application. However, a report on several scalability tests can be found at: `http://www.hpcadvisorycouncil.com/pdf/CPMD_Performance_Profiling.pdf`.

We have benchmarked a range of atomistic simulations of nanocomposites on the Huygens supercomputer at SARA in Amsterdam. This 65 TFLOP/s machine is equipped with around 3456 IBM POWER6 processors. We provide the wall-clock time spent to run atomistic simulations for 10000 steps as a function of the number of processes in Fig 2.

In our application we use SMT to run 2 processes on a single core. Although this somewhat worsens the overall scalability in the plot below, it also allows us to save 50% on our consumption of compute resources. We have enabled SMT in all our tests that use more than 32 cores. The idealized speedup lines do not take the inefficiency introduced by using SMT into account.

We have also benchmarked a range of coarse-grained simulations of nanocomposites on the Huygens supercomputer. We provide the wall-clock time spent to run coarse-grained simulations for 10000 steps as a function of the number of processes in Fig 3. Here, too, we have used SMT in all our tests that use more than 32 cores.

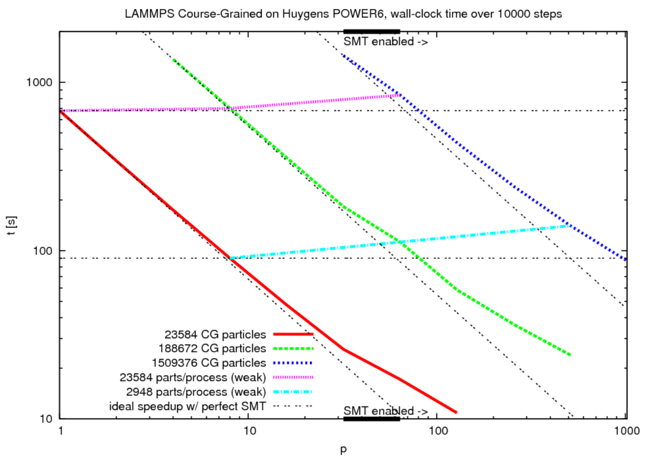

Figure 3: Benchmark results of running LAMMPS on Huygens in coarse-grained mode mode.

2 In-stent Restenosis 3D (Physiology)

2.1 Description

The In-stent Restenosis 3D (ISR3D) has been described in detail in delivrable D7.1. In short it models what may occur after the stenting of a coronary artery. The main submodels are: smooth muscle cell proliferation (SMC) which causes the restenosis; blood flow (BF) which affects the proliferation speed; and drug diffusion of a drug diffusing stent. This runs in a loop, with one iteration of SMC triggering a blood flow and drug diffusion simulation. Afterwards, SMC continues with their information. The model continues as long as SMC has more iterations.

2.2 Implementation

The implementation details of In-stent Restenosis 3D have not changed significantly since D7.1. There has been a conference publication of how ISR3D runs on the MAPPER infrastructure [13].

The SMC code is under active development and now has more realistic tissue modeling. It is also in the process of being parallelized, so that we can make better use of the resources at hand.

2.3 Performance evaluation

ISR3D runtimes have been measured in several settings. First of all, each of the submodels are used and tested locally. However, to do a coupling between the submodels a more advanced machine is needed, since then multiple submodels will run at once. The goal is to know both the runtime and the efficiency of the setup used.

2.3.1 Test setup

ISR3D is executed in four different scenarios: locally, on a NGI cluster, on a PRACE machine. Since the runtime behavior ISR3D is cyclic, determined by the number of smooth muscle cell iterations, we measured the runtime of a single cycle for each. Since reservation would only be done once per total simulation, and not once per cycle, the total runtime is going to be dominated by execution and not reservation time. We did therefore not include it in our runtime statistics.

Local machine The local setup consists of an iMac with Mac OS X 10.6 installed, with a dual-core Intel i7 3.6 GHz processor and 4 GB of RAM installed. It can run smooth muscle cell (SMC) and drug diffusion submodels without much trouble.

NGI resources The entire model was also run on NGI resources, the Polish PL-Grid site Zeus from Cyfronet, Krakow, to be precise. In this instance, we used one node with 8 cores of an Intel 2.4 Ghz processor.

PRACE machine Next, the entire model was run on a PRACE resource, the Dutch Huygens site of Sara, Amsterdam. In this case, all submodels were run on two cores, except BF which was run on 64 cores of the IBM Power6 4.7 Ghz processor. This scenario required within site communication, since the two cores running most of the submodels were run on a different node than the BF code.

NGI-PRACE combined Finally, we spread the model into two parts: BF running on Huygens with 64 cores and all other submodels running on Zeus on two cores.

To optimize efficiency on Huygens, we created a setup which alternates the blood flow calculations of one simulation with the blood flow calculations of another, both running in the same reservation, thus doubling the efficiency of the model on resource usage. This is enforced using a classical wait/notify loop treating the two blood flow calculations as mutually exclusive, and implemented as two light-weight wait/notify mappers with no busy wait.

2.3.2 Results

In Table 1 the runtimes of the scenarios above are given. The efficiency is calculated as the CPU time taken, divided by the cores reserved times the

Table 1: Runtimes with different scenarios. The first column is the scenario, the final column the efficiency and the others are runtimes in minutes.

Scenario	BF	Other submodels	Coupling	Total	Efficiency
Local	40	29	1	70	80%
NGI	35	19	1	55	80%
PRACE	8	21	1	30	27%
PRACE-double	8	22	2	40	45%
NGI-PRACE	8	16	2	26	32%
NGI-PRACE-double	8	18	3	29	56%

time taken per cycle.

By using the PRACE resource the runtime went down drastically, but the efficiency was also severely decreased. This was somewhat ameliorated by using a scheme where two simulations alternated on the same resource. Interestingly, the distributed simulation between NGI and PRACE was faster and more efficient than running on the PRACE machine alone. The reason is that the Blob submodel, which is a serial Fortran code, only compiled with the GNU gfortran compiler, which is not fully optimized for the IBM Power6 processor of the PRACE machine. The NGI resource, however, has an Intel processor and since gfortran is well optimized for this Blob ran 3 times faster (from 12 to 4 minutes).

2.3.3 Conclusions

For ISR3D, doing distributed multiscale computing is a viable option. Compared to running locally, distributed multiscale computing cuts the runtime cost by more than a factor 2, although it is only 70% as efficient. Coupling cost played a small role in this, taking about 1/10th of the total time. The resource management cost is larger though, so another viable option is to do only NGI computation, where we could launch many jobs at once and have a very decent efficiency. Once the SMC code is parallelized, these options should be reevaluated.

3 Equilibrium Stability Workflow (Fusion)

3.1 Description

In fusion devices, such as tokamaks, the study of equilibrium and stability is a crucial issue. Not only should the plasma inside the tokamak be in a state of magnetohydrodynamics (MHD) equilibrium but we must also assure that it lies in a stable configuration on a long enough timescale. In this context, the equilibrium stability workflow [25] application is one of the tools used to simulate important aspects of nuclear fusion processes by measuring the competition between stabilizing and destabilizing force fields. This workflow consists of two loosely coupled subcodes: an MHD equilibrium code (HELENA) [21], which is a high resolution fixed-boundary Grad-Shafranov solver, and a linear MHD stability code (ILSA), working in the so-called MISHKA mode [22]. This was one of the first modules to be developed by the EFDA Integrated Tokamak Modeling Task Force [6] and is therefore also of the better established ones.

The equilibrium stability application is a loosely coupled workflow where the data can be exchanged via files (such as in the present status of the application) or via structured objects defined by the EFDA ITM task-force (see [16, 23] and references therein). Within this application several alternative workflows are possible which vary the profiles from the equilibrium code, recompute the equilibrium for each case and then calculate the MHD stability. One such example, which we have been using for testing purposes, is the $j - \alpha$ stability workflow [25]. In this case different profiles are computed, taking into account parameters such as the maximum edge current pressure gradient and edge current density of the plasma, and then fed again into HELENA, which in turn passes the new modified equilibrium to the ILSA code for stability calculations.

3.2 Implementation

A huge effort towards code coupling through the definition of a common datastructure has been made by the ITM task-force over the past few years. This unified datastructure is composed of complex objects called CPOs (Consistent Physical Objects) which regroup quantities relevant for a given part of the physics. As each simulation code involved in the ITM works on such CPO objects, all codes solving the same physics have the same interface. Thus they are interchangeable and code coupling is simplified. In the scope of this work, we have decided to use codes coming from the ITM in order to take benefits from the CPO datastructure and the code coupling experience. Note that even if the datastructure is evolving each year, we chose to stay in version 4.09a, which was the latest stable release when we started working on adaptation of fusion applications.

The loosely-coupled fusion application is composed of two Fortran codes, HELENA and ILSA, which are working on CPOs. HELENA is generating an *equilibrium* CPO, given a set of code specific parameters stored in a XML file and a set of input files. ILSA is taking this equilibrium CPO plus its own XML parameters' file as input and generates an *mhd* CPO. CPO exchange between the codes is done through files, each program being composed of the following steps:

1. read input CPO from file
2. read XML parameters

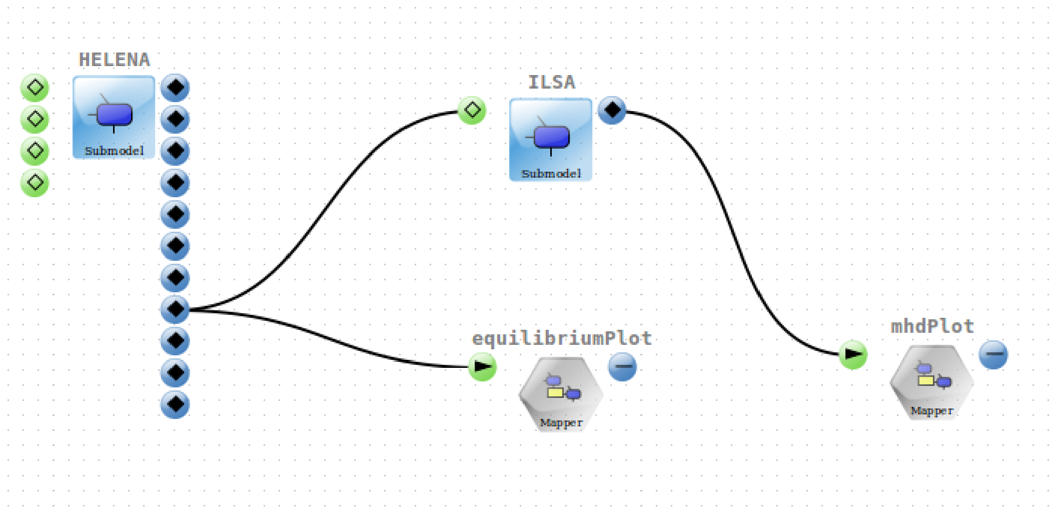

Figure 4: Helena Ilsa application built with MAD tool.

3. call the main routine of the solver
4. write output CPO in file

HELENA and ILSA codes have been updated to version 4.09a of the CPO and were tested on different MAPPER resources (Zeus at Cyfronet and the linux cluster at LRZ). Both executables are installed as executors in GridSpace2, where their executable snippet consists of specific parameters given in XML format. Once this step was done by WP8, we used the MaMe tool to register both codes in the *Models* repository, and build the application through the MAD tool's graphical user interface, as shown in Figure 4. An XMML description of the experiment is then automatically generated and the simulation can be run from Experimental Workbench. This figure shows also two post-processing and visualization components. They are composed of a Python module reading CPO from files and storing it into a Python object, and a script using the Matplotlib library to draw the relevant quantities. Figures 5 and 6 show the plots resulting from the HELENA equilibrium and ILSA *mhd* CPO on a small test case, using real Tokamak geometry (ASDEX Upgrade).

Note that until now, this experiment has been run only on the same site (Zeus), but adding a second site (LRZ cluster) is currently being investigated. Minor modifications should also be added to the application in order to improve its ease of use, especially in the case of a parameter scan, but this workflow is already close to production state.

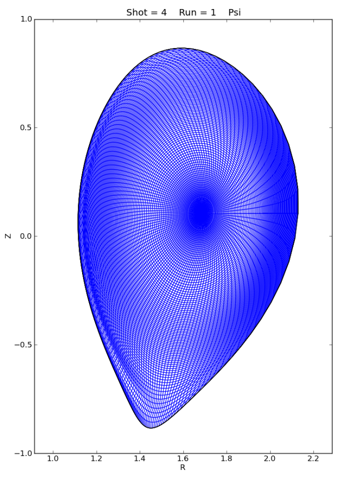

Figure 5: Shape of equilibrium coming from Helena.

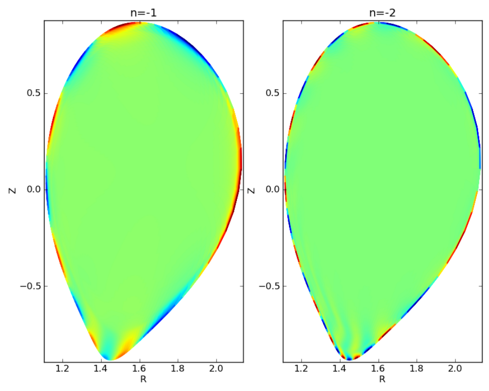

Figure 6: Two different poloidal modes coming from Ilsa.

4 Transport Turbulence Equilibrium (Fusion)

4.1 Description

The multiscale physical processes occurring inside nuclear fusion devices such as tokamaks are illustrated in Figure 7.

The transport turbulence equilibrium application [16, 27] is a simplified version of a simulation of the full fusion core in a nuclear fusion reactor. The three main subcodes involved are:

1. HELENA: 2D equilibrium solver (elliptic, no explicit time, but equilibrium time dependent) [21].
2. GEM: 3D gyrofluid turbulence code [36], calculates transport coefficients.
3. ETS: 1D transport code [16] that calculates new profiles.

These are the main components. However, both for HELENA and GEM, a number of modules can be substituted, with differing trade-offs of speed and accuracy/complexity, which are quite useful for testing, debugging and general implementation purposes. These include BDSEQ, which computes a simple equilibrium, BOHMGB, related to neo-classical transport in tokamak devices, as well as a few others. There are also some simple service modules in addition to these physics modules.

This application is a tightly-coupled one, with data flowing (see Fig. 8): from HELENA to GEM and ETS; from GEM to ETS; and from ETS to

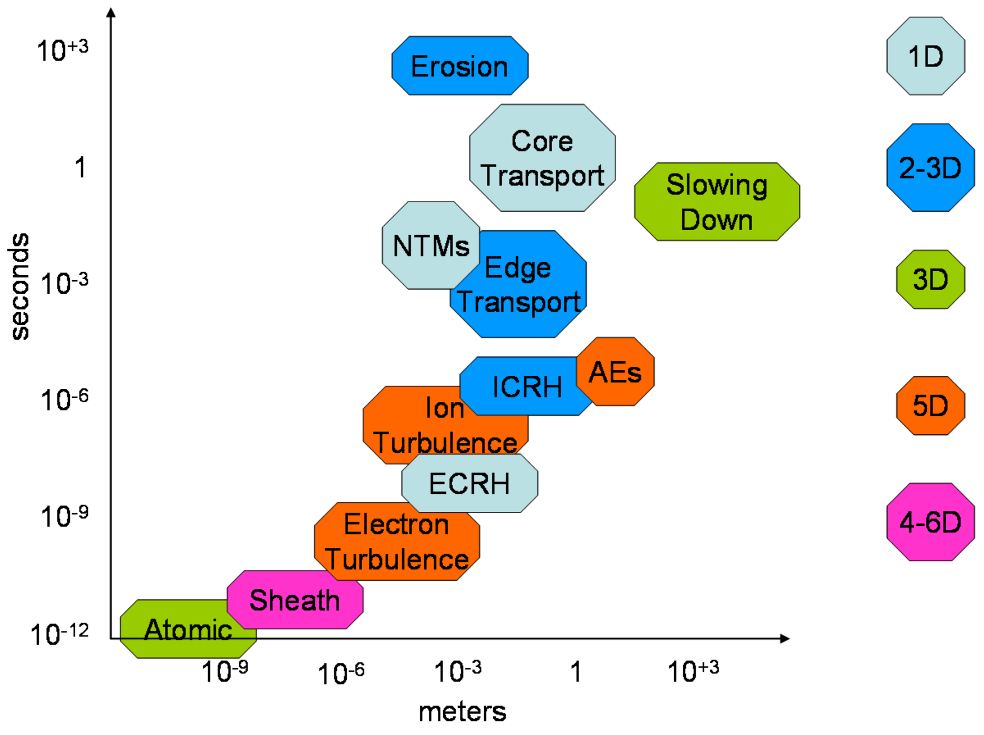

Figure 7: Space and time scales involved in Fusion physics.

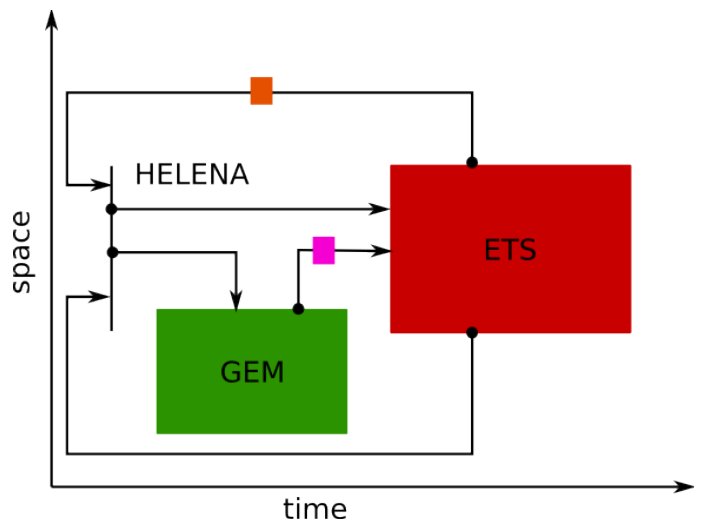

Figure 8: Transport Turbulence Equilibrium dataflow.

HELENA and GEM. Most of the codes are serial and/or do not require much CPU time. The exception is GEM, in which most of the computing time in production runs will be spent. For this code, 8 to 16 instances need to run in parallel, each requiring between 64 to 256 cores, each instance corresponding to a flux-tube, i.e., a *local* gyrofluid simulation covering a given region of the core of the tokamak. This methodology of using several flux tubes (as opposed to a very large *global* simulation) has a great potential for scaling well up to the simulation of larger nuclear fusion devices, such as ITER [7]. At the moment, however, we are still using some of the simpler modules for testing purposes, instead of the proper turbulence code GEM.

4.2 Implementation

The tightly-coupled fusion application is composed of three submodels: Turbulence, Equilibrium and Transport. As for the loosely-coupled application, each submodel defines a set of CPO objects for organizing data calculated by a model, and data exchanged between different models. Several codes developed within the ITM, which correspond to the same or to equivalent submodels share the same interface composed of a set of input and output CPO. Such codes are interchangeable, allowing us to build different flavours of the same experiment. Amongst the different codes shown in Table 2, we have chosen the simplest ones for each submodel in order to test the adaptation of this type of application in the MAPPER framework: BDSEQ for Equilibrium in circular geometry, BOHMGB for Turbulence, and ETS for Transport.

Submodel	*Input*	*Output*	*Code*
Turbulence	equilibrium coreprof	coretransp	BOHMGB ETAIGB GEM
Equilibrium	equilibrium	equilibrium	BDSEQ HELENA
Transport	equilibrium coreprof coretransp coresource coreimpur toroidfield	equilibrium coreprof	ETS

Table 2: Uniformized input/output and interchangeable codes for each submodels.

Each code is written in Fortran (most codes in the field of Plasma Physics are), the main procedure takes CPO derived types as arguments and is compiled as a static library. As MUSCLE is developed in Java, a set of wrappers has to be implemented around each native code (see Fig. 9), with two different goals. The first goal is to make the main Fortran procedure available from Java, and the second one is to provide a way to convert CPO data objects made of complex Fortran derived types into some interoperable data which can be handled in other programming languages. So for each submodel implemented in Fortran, the three following pieces of source code have to be provided:

1. Fortran wrapper: defines a routine which takes the interoperable data format and converts it into a Fortran derived type for each input CPO of the target procedure. Performs the inverse steps for each output CPO. Moreover, this routine should use C-type variables and C name binding capabilities (coming from the `iso_c_binding` module) in order to be fully compatible with C.
2. C wrapper: defines a C function receiving Java objects, converts them into C-type variables and passes them to the C-bound Fortran routine. Then, it converts C-type returned values into Java-type objects. This function interface and native type conversion are performed by JNI, the programming framework for Java Native Interface.
3. MUSCLE Java kernel: instantiate the MUSCLE `CAController` class. Declare the method corresponding to the Fortran procedure as `native` and load the dynamic library which contains the JNI C function. Overwrite `addPortals` and `execute` methods of this class.

All three submodels are registered in the repository using the MaMe

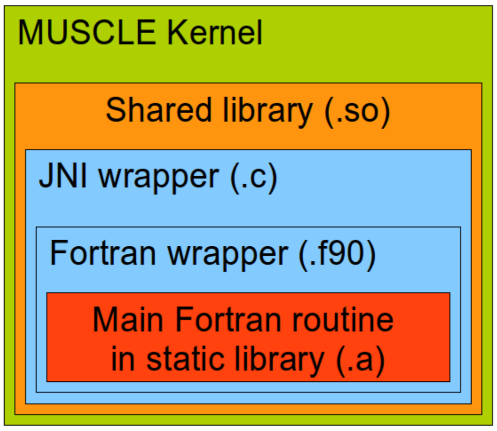

Figure 9: Wrappers around Fortran native code used in MUSCLE.

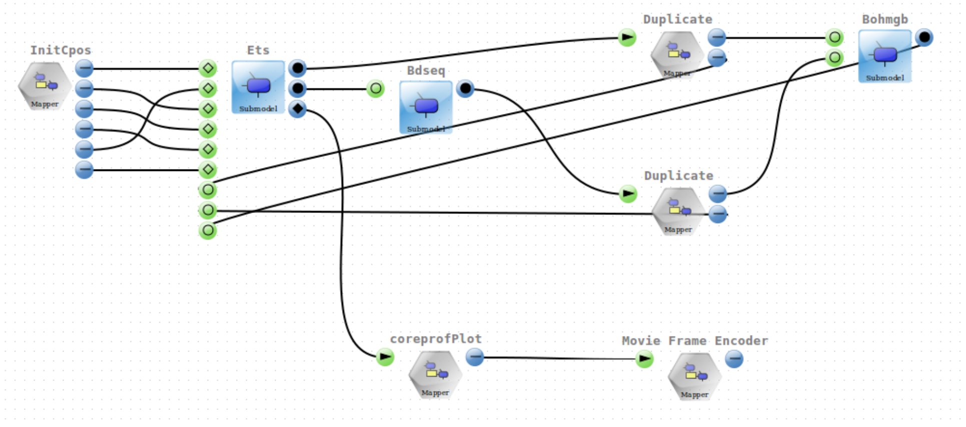

Figure 10: Turbulence Equilibrium Transport built with MAD tool.

tool, given the Jar file containing the kernel bytecode and a parameter to specify the path to the shared library loaded by the kernel. In addition, some utility kernels have been added: one for duplicating a conduit when a submodel output is an input for more than one submodel, and a second one for initializing the CPO when the experiment starts. Post-processing is performed outside MUSCLE, through a visualization component based on Python/Matplotlib, and by re-using the movie-frame-encoder component initially defined for the canals experiment, where only the mencoder command line stored in its snippet under Experimental Workbench is modified. Then, XMML description of the application is done directly by MAD tool when the application is designed graphically as shown in Figure 10.

Data conversion between Fortran complex derived types and some interoperable format is still in proof of concept phase: at this moment we are

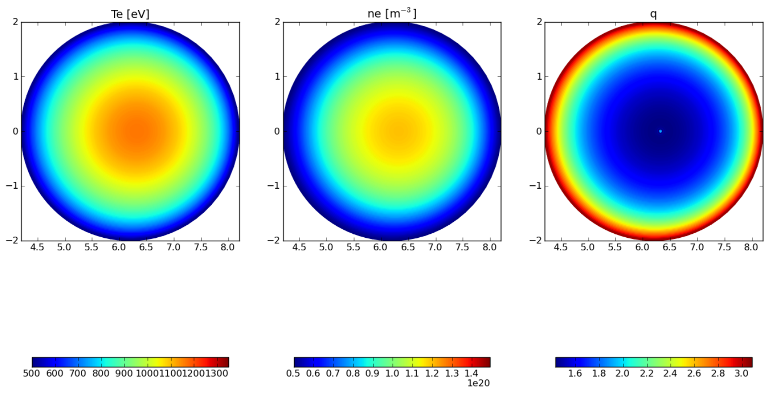

Figure 11: Movie of some quantities mapped to a circular equilibrium and evolving in time.

using file storage in order to pass CPO through the different components of the MUSCLE experiment. Wrapper functions and kernel portals send and receive simple strings giving the name of the file which stores the CPO object. Concerning the implementation of our kernels, we are not using MUSCLE `willStop` method to automatically perform the time evolution as it is not fully compatible with our needs. As time evolution is handled manually, the cxa script has to be slightly modified in order to define the required global variables.

The full experiment composed of a MUSCLE snippet followed by two post-processing steps runs on a single site at the moment (Zeus). Figure 11 shows a frame of the movie generated from data of equilibrium and coreprof CPO produced by ETS and evolving in time. This is a test experiment used to demonstrate all the steps involved in the adaptation of such Fortran codes to the MAPPER infrastructure. We are currently working on three aspects to transform this test case into a production run which can be used to simulate real physics problems:

1. implementing a dedicated serialization library to transfer CPO from Fortran to Java via byte streams
2. adapting more complex and parallel codes such as GEM
3. avoid XMML manual modifications for handling time evolution by converting our kernels to MUSCLE 2.0

5 HemeLB (Physiology)

5.1 Description

Recent progress in imaging and computing technologies has resulted in important advances in physiology. Using modern imaging methods, we are now able to scan the geometry of individual vessels within patients and map out potential sites for vascular malformations such as intracranial aneurysms. Likewise, recent increases in computational capacity and algorithmic improvements in simulation environments allow us to simulate blood flow in great detail. The HemeLB lattice-Boltzmann application [28] aims to combine these two developments, thereby allowing medical scans to be used as input for blood flow simulations. It also enables clinicians to run such simulations in real-time, providing runtime visualization feedback as well as the ability to steer the simulation and its visualization [29]. One principal goal for HemeLB is to act as a production toolkit that provides both timely and clinically relevant assistance to surgeons. To achieve this we must not only perform extensive validation and testing for accuracy, reliability, usability and performance, but also ensure that the legal environment and the medical and computational infrastructure are made ready for such use cases [34].

Within MAPPER we are constructing multiscale bloodflow simulations, featuring both the HemeLB application and the Python Navier-Stokes (PyNS) full-body 1 dimensional bloodflow solver. The aim of coupling these codes is to incorporate a full-body bloodflow, while maintaining a high spatial and temporal resolution in the area of specific (clinical or scientific) interest. The submodels are all tightly coupled, each providing boundary conditions to the concurrently-executing, adjacent submodels. The frequent communication between models places demanding requirements on the latency of the coupling library, although in the first instance the volume of data to be exchanged is low. The coupling from the network model to HemeLB will require construction of a flow profile (typically a parabolic Poiseuille flow profile) and the reverse coupling will require computation of the the average pressure and velocity. Coupling of the different resolution HemeLB simulations will require resampling of the underlying lattice-Boltzmann distribution functions between resolutions. The number of submodels and the coupling between them is determined during simulation setup. In our multiscale implementation we have chosen to ensure good performance from Day 1, establishing the coupling between the submodels using MPWide [19] in the first instance, and then incorporating additional coupling mechanisms such as MUSCLE or MPI where needed or convenient.

5.2 Status

We have performed an in-depth investigation of the simulation, steering and visualization performance of HemeLB on the HECToR Cray-XE6 machine in Edinburgh. We have also developed a single-scale performance model that allows us to predict the runtime of HemeLB in advance, something which is a necessity on the pathway towards clinical use. Both these advances have been written up in a paper and submitted to the Journal of Computational Science [18].

We have also successfully implemented a Multiscale Intercommunicator which allows HemeLB to publish shared values and to connect to any intercommunication or coupling library by defining a template class. As a first step we have defined an Intercommunicator class to connect HemeLB with MPWide. The Multiscale Intercommunicator code is part of the HemeLB code base and is subjected to unit tests, functional tests and integration checks using Jenkins.

Using our multiscale code extensions in conjunction to MPWide, we have made a preliminary coupling of HemeLB to HemeLB. In addition, we have made a preliminary coupling of PyNS to PyNS using a newly developed Python interface to MPWide. At time of writing the values we pass between the simulations are still very simplistic, although we have identified the correct parameters to exchange during the simulation. We have established a coupling between HemeLB and PyNS, and incorporated velocity-aware boundary conditions to enable a more realistic method of coupling between the codes.

5.3 Performance Benchmarks

We have run a number of performance tests on the HECToR Cray-XE6 machine at EPCC in Edinburgh. The tests were carried out using three different simulation domains, a Cylinder domain consisting of 15,607,040 lattice sites, a Bifurcation domain consisting of 19,808,107 lattice sites, and a Large Bifurcation domain consisting of 81,132,544 lattice sites. The predictions from our performance model are generally in agreement with our measurements, especially for the larger simulation domains. However, the model does not predict the superlinear speedup measured in the results. This is mainly due to the relatively large calculation and communication load imbalances we find at these low core counts.

6 Irrigation Canals (Hydrology)

The goal of this application is to develop hydrodynamic submodels in order to simulate a network of canals or any water course that need to be controlled and managed to produce irrigation water, electricity, etc. We refer the reader to deliverable D7.1 for more details.

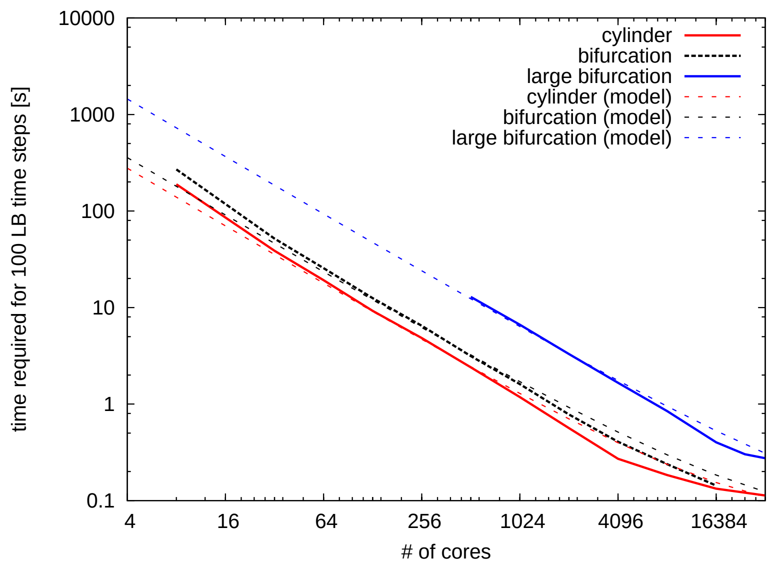

Figure 12: Wall-clock time spent to simulate 100 time steps as a function of the number of cores used for the Cylinder, Bifurcation and Large Bifurcation simulation domains. Predictions by the performance model we have developed are indicated by the dashed lines.

6.1 Adaptation to the MAPPER approach

Due to the size of an irrigation network and the large variation in the flow complexity across different sections, some parts of the canal section can be described with 1D shallow water based models, whereas other sections need a 3D, free-surface hydrodynamic model (FS3D) to properly capture the flow properties. Our future plan is to construct the whole canal "on the fly" based on these 1D and 3D models.

To do so, we performed a simulation to validate the 1D and 3D models following the line proposed by the MAPPER project.

6.1.1 1D submodels

The 1D submodels are based on the D1Q3 lattice Boltzmann (LB) shallow water equation and describe a long canal section where the water flow is mostly in the direction of the canal. 1D submodels could be connected (coupled) together directly or through water junctions. The implementation of many 1D junctions (canal section, gate, spillway, pumping station, etc.) is finished and validated through a prototype.

For validation, we constructed a canal section composed of three submodels and two mappers (junctions): a Gate and Spillway. In first step, the coupling schema is done according to the MML specification and designed using the MAME/MAD tools. In second step, the simulation is performed over the GridSpace framework and presented in the last summer school (London 2011) as a prototype: http://www.mapper-project.eu/web/guest/mad-mame-ew.

The coupling algorithm for two submodels connected with a junction is described as follows. Each submodel obeys the CxA based formalism, as depicted in Fig. 13. In this formalism, a submodel:

1. Initializes its parameters from the CxA coupling file, send its maximum number of iterations to the junction, and then starts execution.
2. Performs one step of computation (e.g collide and stream for a LB model)
3. Updates boundary conditions. This is done by sending data to the junction and waiting for updated information.
4. Makes an observation if needed.
5. Increments the iteration counter.
6. Repeats back to step 2 until the maximum number of iterations is reached.
7. Sends *End message* to the junction and finish the execution.

In the present case, the junction is implemented as a daemon program that keeps listening to data from the two corresponding canal sections. It is worth noting that a junction could also be modeled as a full-fledged submodel, with its generic CxA execution loop, solving the boundary condition problem at each iteration and running until the whole execution ends.

The communication process requires synchronization in the following way: the junction uses the *receive()* method, a blocking point-to-point operation of the MUSCLE framework, to receive data from each canal section. From the other side, canals send data to the junction and call the *receive()* method to wait for updated boundary information. Once data has arrived from both canal sections, the junction performs the boundary computation and sends back the updated information to the recipient submodels. This process keeps running until the maximum iteration number is reached for each submodel. The submodel will then end their execution after sending an *End* message to the junction.

Besides, in the case where the upstream and downstream canals have a different spatial and temporal resolutions, grid refinement techniques must be used for the coupling where the connecting junction must be programmed to handle two different frequencies of send/receive operations for each side. We have implemented such a coupling using the grid refinement algorithm presented in [26]. The temporal resolution Δt_c of the coarse grid was twice greater than the Δt_f of the fine grid. The junction component was running on a separate computer node, thus illustrating the distributed multiscale nature of the simulation.

The above results and a new coupling algorithm to implement a *Gate* submodel are published in [11]

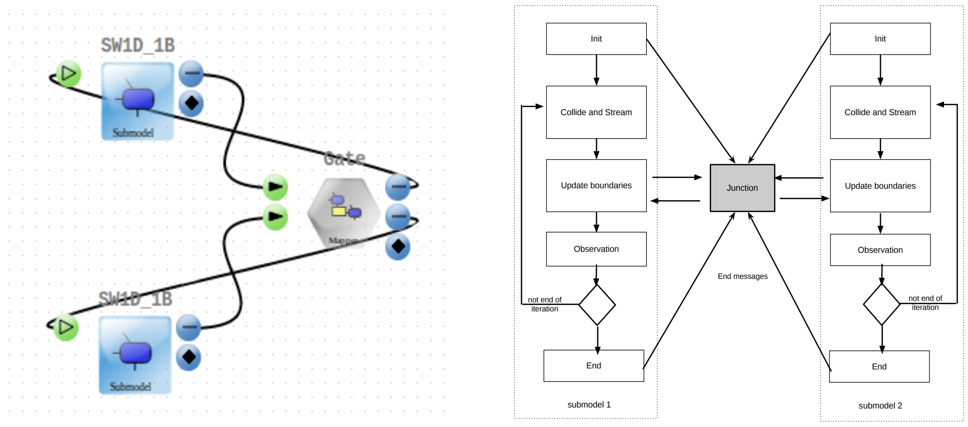

Figure 13: (Left:) The MAD [1] tool, a software part of GridSpace platform, generates automatically the XMML coupling description. (Right:) Template of the coupling algorithm for two submodels and a junction.

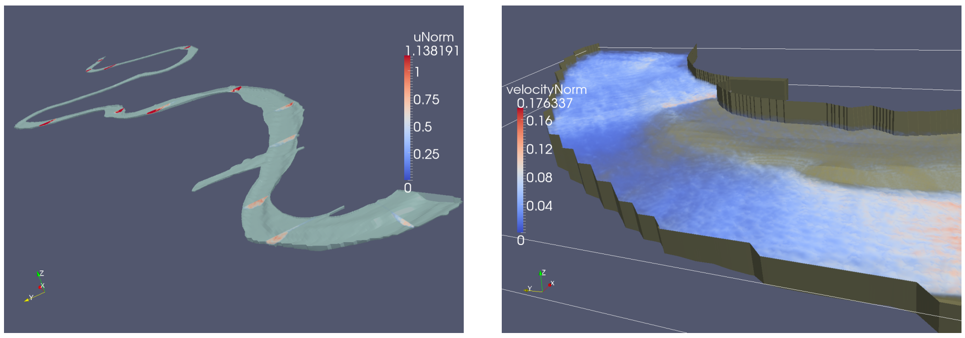

Figure 14: Simulation of the flow in the Rhone river in Geneva (left). Example of a 3D, free surface simulation of a draining event at the dam in Verbois (right).

6.1.2 3D submodels

This type of submodel resolves the details of the water flow around gates and junctions. The implementation of the LB based 3D submodels is mostly done using PALABOS and MPI environments and a free surface flow model (see D7.1 for more detail). As a specific case, the bed profile of the Rhone river in the Geneva area has been provided by the *Services Industriels de Genève*, the electricity company in Geneva. It is described by an STL file [4]. The simulations are illustrated in Fig. 14 and will be described in an article in preparation [31]. Such 3D LB submodel requires off-lattice boundary conditions and considerable computing resources. Following the MAPPER guidelaine, and for validation purposes, we divided a big 3D section into two sections (or more) and connect them using MUSCLE2 API (C++ version). Our objective is to simulate 13 km of the Rhne river over distributed clusters rather than a local simulation.

The simulation is conducted using qcg-broker grid-middleware and based on scripts and job description files. The workflow composing the jobs is described on a XML file and sent to the QCG broker for execution. Then, the QCG broker will starts all the jobs once the remote candidate clusters are available at the same time. An example of XML based worker flow is described in appendix A.2.

Simulating 13 km of a river course with a 3D free-surface hydrodynamical model using a 40 cm spatial resolution is a numerical challenge. A few hours of real time can be simulated with a several thousands cores in about two weeks. By coupling several petaFlops supercomputers, as possible in the MAPPER vision, a few weeks of real time can be considered. However, such a detailed simulation may not be required along the full water course and the coupling of the 3D section with 1D shallow water section is the most promising way to reach the time scale that operators need to consider when managing the river.

6.2 Mapper performance prediction model

Regarding the performance measurement, we consider the time overhead induced by the MAPPER approach when performing simulations of real hydrodynamical problems. The considered MAPPER tools are: MUSCLE2 API, QCG broker, and PALABOS toolkit [3].

To do so, a PALABOS based lattice Boltzmann 3D submodel (cavity3D) is considered to perform tests and simulations. The pseudo algorithm of the code is shown in the listing 1

Listing 1: pseudo-code of the caviy3d example

```
Finit
While (it++<2000){
 CollideAndStream()
 GetBoundaryData()
 SendReceiveBoundaryData()
 UpdateBoundaryData()
}
end
```

- The *CollideAndStream*() operation: consists in a LB computing operations.
- The *GetBoundaryData*() operation: retrieves boundary data scattered over the same cluster nodes.
- The *SendReceiveBoundaryData*() operation: sends/receives the selected boundary data to/from other submodels.
- The *UpdateBoundaryData*() operation: updates the boundary data scattered over the same cluster nodes.

The simulation scenario consists in:

1. Running a monolithic simulation over one cluster.
2. Splitting the cavity into two equal sections (left and right) and coupling them using MUSCLE API. Simulation will be first performed on the same cluster and, then, over two distributed clusters (Galeera and Reef) [2] using QCG grid middleware.
3. Comparing the execution clock-time and data transfer impact of the distributed simulation to the monolithic one.

Inspired by the work of [9], we elaborated a performance prediction model that approximates the execution time based on the computing resources and the coupling strategy.

We define $T_{muscle}(p, N)$, the execution time of a "mapperized" LB based submodel over p cores and with problem size N as

$$\begin{aligned} T_{muscle}(p, N) &= \frac{T_{serial}(N)}{p} + T_{com} \\ &= T_{mono}(p, N) + T_{com} \end{aligned} \tag{1}$$

where T_{com} is the communication time incurred by MUSCLE, T_{mono} is the monolithic execution time of the submodel (kernel) on p cores (same cluster) without using MUSCLE. We assume that the execution time of one kernel is equal to $T_{mono}(p, N) = T_{serial}(N)/p$ and the communication speed and time between the same cluster nodes are similar. Hence, we can deduce the submodel "MUSCLING" efficiency $\varepsilon(p, N)$ with:

$$\varepsilon^{muscle}(p, N) \quad = \frac{\text{monolithic execution time}}{\text{muscle execution time}} \quad = \frac{1}{1+\theta} \tag{2}$$

where θ is the communication overhead fraction of a submodel defined as

$$\theta(p, N) = \frac{T_{com}(N)}{T_{mono}(p, N)} \tag{3}$$

In what follows, we develop a performance model for irregular canal sectioning of a tightly coupled scenario with a problem of size $L_x \times L_y \times L_z$, where L_x (resp. L_y and L_z) stands for the actual size along x-axis (resp. y-axis and z-axis))). "Mapperization" is achieved by dividing the whole LB application, along the x-axis, into several submodels s_j, $j \in \{1..k\}$, where each of them has a LB grid size of $(n_{x,j}, n_y, n_z)$, where $n_{x,j}$ (resp. n_y and n_z) stands for the number of lattice sites of submodel s_j along the x-axis (resp. y-axis and z-axis). Note that $\sum_j^k n_{x,j} = n_x$ but the submodels do not have necessarily the same grid size.

The total execution time per each submodel is determined by the computation time required for its domain and the time required to exchange and update its boundary data B_k. Let t_k be the execution time of the submodel s_k over p_k cores. We write t_k as

$$t_k = T_{cpu} + T_{com}(B_k) \tag{4}$$

where T_{cpu} is the computation time required for the submodel s_k over p_k cores and $T_{com}(B_k)$ is the time required for the boundary condition operation, defined as (see details in A.1):

$$\begin{aligned} T_{cpu}(p_k, \Delta x) &= A_k/\Delta x^3 \\ T_{com}(B_k) &= B_k/\Delta x^2 \end{aligned} \tag{5}$$

It is oblivious that T_{muscle} will be determined by the slowest submodel, i.e:

$$T_{muscle} = T_{muscle}(p_k, N) = \max_{1\leq j\leq k}\{t_j\} \tag{6}$$

and, hence, the total communication fraction overhead of the application can be written as

$$\begin{aligned} \varepsilon^{muscle}(k, N) &= \tfrac{T_{serial}(N)}{p\times\max_j\{t_j\}} \\ &= \tfrac{1}{1+\theta^{muscle}} \end{aligned} \tag{7}$$

where p=$\sum_{j=1}^{k} p_k$ and θ^{muscle} is the total "muscling" overhead defined as

$$\theta^{muscle}(p_k, N) = \frac{\max_j\{t_j\}}{T_{mono}(p_k, N)} - 1 \tag{8}$$

By joining the eq (5) to eq (4), we finally obtain an equation for t_k

$$\begin{aligned} t_k &= A_k/\Delta x^3 + B_k/\Delta x^2 \\ &= A_k/\Delta x^3 \times (1 + \tfrac{B_k}{A_k} \times \Delta x) \\ &= \alpha(\Delta x) \times (1 + \beta(\Delta x)) \end{aligned} \tag{9}$$

Consequently, we obtain from eq (5) and eq (6):

$$\begin{aligned} &T_{mono}(\Delta x) = \alpha(\Delta x) \\ &T_{muscle}(\Delta x) = \alpha(\Delta x) \times (1 + \beta(\Delta x)) \end{aligned} \tag{10}$$

In the next section, we will measure T_{mono} and T_{muscle} for both the local and grid simulation case. Then, we will validate empirically the relation between T_{mono} and T_{muscle} (in eq 12) and measure the muscle efficiency (ε^{muscle}).

6.3 Performance Measurements

The cavity section has a length L_x of 4500 meters, a width L_y of 100 meters and a depth L_z of 25 meters. The Lattice Boltzmann (LB) implementation is based on MPI and simulation is carried out with different value of Δx ranging from 0.4(m) to 2(m) with a step of 0.2(m). Regarding the LB parameters, each lattice cell contains 19 distribution functions with DOUBLE precision data type (8 bytes). The size of one lattice cell is equal to $19 \times 8 = 152$ Bytes. To treat the off-lattice boundary condition, we need for each submodel the $e = 3$ boundary cells along the x-axis, including all the corresponding y-axis and z-axis cells.

Table 3 shows the different lattice grid sizes and the size of the boundary data used for the simulations as a function of Δx. The quantities n_x, n_y and n_z represent the number of lattice sites along tghe x-axis, y-axis and z-axis of the cavity3D section, respectively.

Table 3: LB grid size based on Δx

Δx(m)	0.4	0.6	0.8	1	1.2	1.4	1.6	1.8	2
n_x	11250	7500	5625	4500	3750	3215	2813	2500	2250
n_y	250	167	125	100	84	72	63	65	50
n_z	63	42	32	25	21	18	16	14	13
$B_k(Kb)$	7182	3198	1824	1140	804	590	459	414	269

The monolithic execution is carried out over 20 cores on the same Gordias cluster (hepia-unige), having the following configuration:

- 28 nodes with12 Gb and 8 cores(intel 64 bits) each.
- OpenMPI environment
- InfiniBand (IB) based network interface communication with a speed of 832 Mb/s.
- Ethernet network interface with a speed of 112 Mb/s

For the "mapperized" simulation, we performed a local simulation on the Gordias cluster and a distributed one over two geographically distributed clusters (Galera and Reef).

6.3.1 Local execution

In order to run the local simulation on the Gordias cluster, we firstly run the muscle manager in a separate node. Then, we run the left and the right sections over 10 cores each using Sun Grid Engine (SGE) [5] middleware.

The total number of iterations we performed is equal to 2000 for both the monolithic and muscle local simulations. For each iteration, we measure the computation clocktime of the "boundary" and "collideAndStream" operations. We repeated the simulation with different Δx values as depicted in the figure 15.

The difference of the execution clocktime between the monolithic and MUSCLE implementation for the case Δx=0.4 was 106 seconds over a total elapsed time of 5498 seconds. For the case Δx = 1.4, it was 141 seconds over a total elapsed time of 370 seconds. This suggests that the time overhead of MUSCLE on the same cluster (collideAndStream and data transfer operations) vary slowly with Δx and can be averaged to 120 seconds, for all values of Δx.

The measurements in figure 15 show that the execution time T_{muscle} and T_{mono} depend linearly on the number of iterations (iT):

$$\begin{aligned} T_{mono}(\Delta x) &= a \times iT \\ T_{muscle}(\Delta x) &= b \times iT \end{aligned} \tag{11}$$

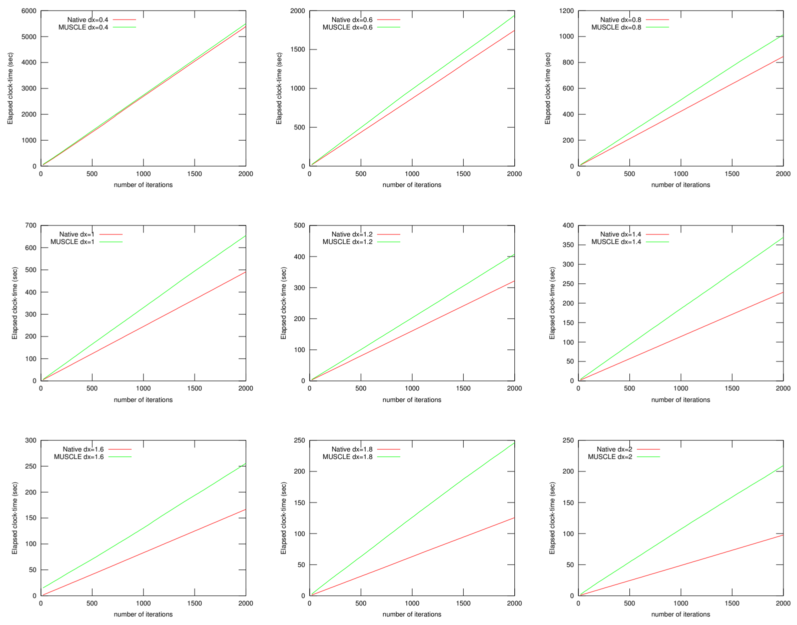

Figure 15: Execution time (wallclock time) versus iteration number for the monolithic code (red line) and the multiscale approach (green). The sameon cluster(Gordias)has been used in both case, with the same total number of processors.

Where a and b stands for the slope coefficients of the execution lines in the graph. After evaluating a and b for each Δx case, we can use them with the eq. (10) to fit empirically the relation between T_{muscle} and T_{mono}. This gives:

$$\begin{aligned} T_{mono}(\Delta x) &= 0.1748/\Delta x^3 \\ T_{muscle}(\Delta x) &= T_{mono}(\Delta x)(1 + 0.43 \times \Delta x) \end{aligned} \tag{12}$$

From eq. (7), we can plot the efficiency of ”mapperizing” the LB test application” in figure 16

It is clear that the benefit of a distributed execution becomes evident when the size of such problem exceeds the capability of one computer or cluster in term of computing resources. The benefit disappears when the overhead of the distributed computation becomes larger than the gains of faster processing, particularly, when the problem size is not large enough.

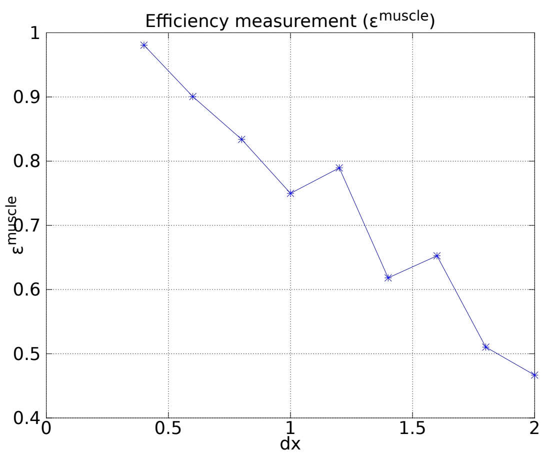

Figure 16: Execution time efficiency as a function of Δx - local cluster case

6.3.2 Distributed execution

For the distributed execution, we kept the same configuration (number of cores, same executable and input data, etc.), but used two separate clusters: Galera and Reef. The measurements shown in figure 17 show that the distributed execution time T_{grid} (stands for T_{muscle} over the grid) depends also linearly on the number of iterations (iT). So we can write: $T_{grid}(\Delta x) = c \times iT$, where c is the slope coefficient of the distributed execution line in the graph.

Compared to the local execution (one cluster), the distributed execution is mainly influenced by the network speed connection between clusters, in addition to the size of the boundary data (fig 17). This also can be observed on the distributed efficiency values $\epsilon^{grid} = T_{mono}/T_{grid}$ in table 4 and on the comparison between ϵ^{muscle} and ϵ^{grid} depicted in Fig 18. Besides, we can observe that the muscle overhead is inversely proportional to the simulation size.

Thus, as a conclusion, it's clear that the time overhead of "mapperizing" a big application is negligible comparing to the procured benefits in time and resource computation usage.

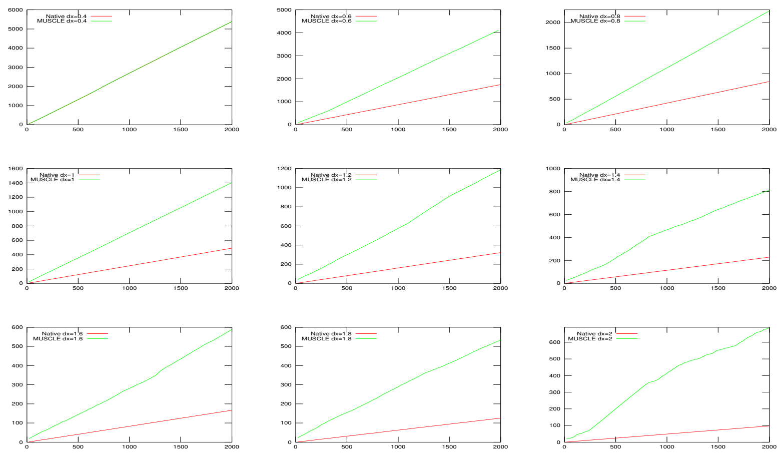

Figure 17: Execution time on local cluster (Gordias) and on two separated clusters (Galera and Reef). The number of performed iterations is equal to 2000. The data communication between clusters is handled by the muscle MTO layer with a network speed of 22 MB/s between the two clusters.

7 Reverse-engineering of gene-regulatory networks (Computational Biology)

7.1 Description

Regulation of gene expression (or gene regulation) refers to processes that cells use to create functional gene products (RNA, proteins) from the information stored in genes (DNA). These processes range from DNA-RNA transcription to the post-translational modification of proteins. Gene regulation is essential for life as it increases the versatility and adaptability of an organism by allowing it to express protein when needed. While some aspects of gene regulation are well understood, many open research questions still remain (Davidson & Levin, 2005 [17]). Due to the wide availability of well-characterized components from biological gene networks, the stage has been set for mathematical modelling and computational simulation of gene regulatory networks. The dynamic behaviour and regulatory interactions of genes can be revealed by time-series experiments, that is, experiments that measure the expression of multiple genes over time (Spellman et al., 1998 [37]). As this type of experimental data becomes more readily available, mathematical modelling and computational simulation become an important tool for investigating the structure and time-dependent behaviour of gene regulation networks.

Besides logical (Kauffman, 1993 [33]) and stochastic (Perrin et al., 2003 [32])

Table 4: ϵ^{grid}: efficiency of distributed execution (2000 iterations)

Δx	T_{mono}(s)	T_{grid}(s)	ϵ^{muscle}	ϵ^{grid}
0.4	5392.54	5498.85	0.980666867	0.583040329
0.6	1746.9	1939.85	0.900533546	0.424933106
0.8	846.53	1015.09	0.833945759	0.37978017
1	490.94	654.78	0.749778552	0.349712218
1.2	321.81	407.62	0.789485305	0.271171444
1.4	228.34	369.37	0.618187725	0.28150886
1.6	166.85	255.68	0.652573529	0.283036472
1.8	125.69	246.26	0.510395517	0.235763055
2	97.76	209.47	0.466701676	0.142777859

modelling approaches, various continuous modelling methods capable of capturing such complex behaviour deterministically are commonly used. A range of mathematical methods facilitating the reverse-engineering (automated inferences or construction) of quantitative, dynamic GRN models from time-series expression data have been reported in the literature (Cho et al., 2007 [15]). Typical methods based on differential equations include the s-system method (Savageau, 1976 [35]), artificial neural networks method (Vohradský, 2001 [40]) and general rate law of transcription method (Mendes et al., 2003 [30]). These and similar methods are the basis of our Computational Biology application within MAPPER.

To-date the majority of the gene regulation models address GRN systems containing less than 20 genes. The reason for this is that the computing requirements for reverse-engineering such models from data grows exponentially with the number of genes in the underlying GRN system. The computing requirements are also dependent on the connectivity topology of the underlying GRN system, the number of parameters in the chosen GRN rate law, the given time resolution, the numerical integration method chosen, and other factors.

Currently, there is a trend in biology to investigate increasingly larger GRN systems; ultimately many important life processes are regulated by more than 20 genes. The study by Davidson & Levin (2005) [17] is an example for such an investigation. Such large GRN systems are characterized by an inherent biological organization composed of sub-networks, each regulating a separate biological function or process while still interacting with each other. The sub-networks in such a multi-network structure may operate on distinct temporal scales. Applying current GRN modelling and simulation approaches to such large, multi-network GRN systems is problematic, both conceptually and computationally. A promising approach to address these challenges is MMS. In the MAPPER Computational Biology application, we develop an MMS approach to GRN modelling and simulation.

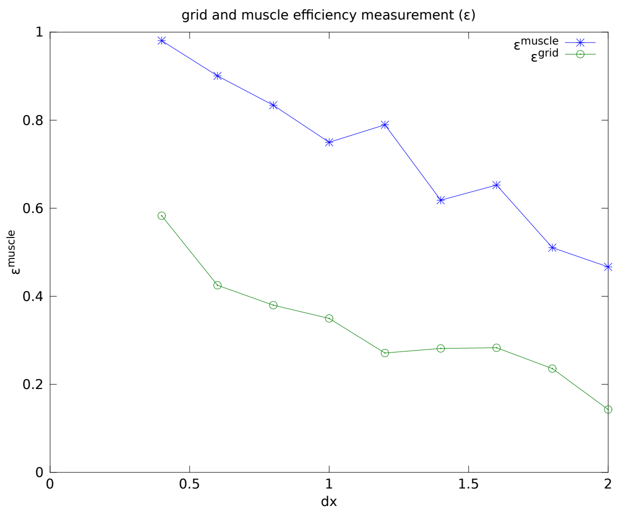

Figure 18: Execution time efficiency as a function of Δx - The upper curve shhows ϵ^{muscle} and the lower one shows ϵ^{grid}.

In a first step, we have developed GRN modelling and simulation solution in Java and integrated it with MAPPER to enable distributed multiscale execution. Currently, we adopt an MMS approach based on particle swarm optimization (PSO) (Kennedy & Eberhart, 1995 [24]) where each sub-model consist of a PSO island[1]. We have deployed and tested this version of the GRN application and are now engaged in larger-scale experiments in which we explore various novel elements in GRN modelling and simulation (rate law, multi-condition reverse-engineering, validation of structure and behaviour).

In the next step, we aim to cast the reverse-engineering algorithm into the MMS framework by representing a single or a group of genes as individual-scale models.

7.2 Summary of achievements

Based on the developments in Year 1 of the MAPPER project, we have redesigned some aspects of the MAPPER Computational Biology application. In particular, in Year 2 we have developed and refined the following MultiGrain[2] features:

[1]Essentially, a PSO islands represents a confined evolutionary environment where a population of individuals (PSO particles) is allowed to evolve over a number of generations. The number of generations may be pre-determined, based on time constraints, or derived from a pre-determined condition.

[2]MultiGrain is the name given to the present GRN modelling framework

- Various components that allow the switching between GRN rate laws in a flexible way.
- Incorporation of a new ODE solver with improved performance characteristics.
- A component to write GRN models in systems biology mark-up language (SBML)[3] format for the supported rate laws.
- A component that facilitates the separation of the reverse-engineering process into two steps: one dealing with the determination of the network topology of the model, and one which in which the parameters to be optimized can be selected (typically this would be used to optimize all parameters except those that have been determined in a previous step). This component is crucial to the envisaged novel approach to reverse-engineer GRN models where single-gene GRN models or multi-gene GRN models are treated as single-scale models in the MMS framework.
- A component to visualize the network topology of GRN models.
- A component facilitating different validation strategies (multiple validation sets, model structure and behaviour)

The software modules have been implemented and tested, using the MUSCLE[4] library, on production hardware of the MAPPER e-infrastructure. While more implementation and tests are necessary, the milestones for Year 2 have been accomplished. Year 3 will focus on additional implementation and testing, and, most importantly, on a number of studies demonstrating the advantages we gain from the MMS to gene regulation modelling and simulation.

7.3 Improved approach to ODE solver

After initially supporting only the artificial neural network (ANN) rate law (Vohradský, 2001 [40]), we expanded the application and added support for four additional rate laws: Hill, s-system and general mass action kinetics (Voit & Schwacke, 2007 [41]). We replaced the previous ODE solver XPPAUT[5] because of undesirable performance overheads associated with its file-based approach to invoke its functions from Java. The latest version of

[3]`http://sbml.org/`

[4]MUSCLE Multiscale Coupling Library and Environment: `http://apps.man.poznan.pl/trac/muscle`

[5]XPPAUT, a tool for solving stochastic, differential, and difference equations: `http://www.math.pitt.edu/~bard/xpp/xpp.html`

our GRN tool implementation uses Flanagans scientific library[6]. This library is implemented in Java. This provides a considerable advantage (both in development and performance) since our GRN tool is also implemented in Java.

7.4 Analysis and visualization of GRN network topology

The current implementation of the GRN tool employs the JGraph[7] Java library to add support for visualizing the network topology or structure of the reverse-engineered GRN models. This component represents GRN graphs in the simple interaction format (SIF); which is supported by Cytoscape[8], a widely used computational biology software tool for analysing and visualizing biochemical pathways and networks. Graph construction from a SIF file is straightforward. The SIF is convenient for combining multiple sets of interacting genes into a larger gene network, or add new gene-gene interactions to an existing SIF file. Lines in a SIF file define a source node, a relationship or interaction type, and one or more target nodes. In the GRN modelling and simulation application we are dealing with only a single node type (gene) and two types of interactions (activates, represses). The following excerpt illustrates the SIF based on a 5-gene GRN:

```
geneA supresses geneB
geneC activates geneA
geneD activates geneE geneF
geneE
geneF represses geneC
```

The first line identifies two network nodes, `geneA` and `geneB`, and a repressing influence of `geneA` on `geneB`. The second line is similar, except that the relationship from `geneC` to `geneA` is an activating relationship. Line three specifies an activating relationship of the node that represents `geneD` on the nodes representing genes `geneE` and `geneF`. The fourth line shows a node `(geneE)` that does not influence any other nodes but may be influenced by other nodes `(geneD)`. Finally, line five is similar to line one, except that it involves `geneF` and `geneC`.

7.5 Flexible validation of GRN models

We are aiming to explore comprehensive validation scenarios currently not covered in state of the art modelling of GRN systems. In particular, our aim is to investigate:

[6]Michael Thomas Flanagans scientific library: `http://www.ee.ucl.ac.uk/~mflanaga/java/`

[7]The JGraph library: `http://www.jgraph.com/`

[8]Cytoscape: computational biology software tool for analysing and visualizing biochemical pathways and networks: `http://www.cytoscape.org/`

- **Model behaviour validation.** While this is the *de facto* standard in evaluating GRN models, we are keen to compare two (possibly more) error scoring methods: mean absolute error (MAE), root mean square error (RMSE) (Willmott & Matsuura, 2005 [42]).
- **Structure validation.** Validating the network structure representing the interactions of a dynamic system is typically more important than validating the dynamic behaviour against the measurements (Barlas, 1994 [10]).
- **Multi-data validation.** Flexible support for validating GRN models against one or more data sets selected from a pool of training and validation data sets. This feature allows us to explore scenarios involving multiple training and validation sets, including multi-condition experiments (repeated-measures design).
- **Multi-model validation.** Usually, an MMS study starts out with linking or coupling multiple single-scale models that have already been validated individually. Since we are planning to investigate multi-scale reverse-engineering of large modular GRN systems, we will need to investigate novel ways of validating such multi-scale models.

The current implementation of MultiGrain allows us to assign training and validation sets in a flexible manner. As we proceed with our implementation in Year 3, we will enhance MultiGrain to cover more of validation features listed above.

7.6 Performance analysis

We evaluate the performance of some of the MultiGrain components/features; here we focus mainly on the performance of fitness calculations. We have executed the application under different hardware configurations, varying the number of PSO steps.

We reverse-engineered a five-gene GRN model from a gene expression data set containing 40 time-points. Our test scenario included two PSO islands, each populated with 100 PSO particles. The islands exchanged particles every 100 steps and we varied the number of such particle exchanges (10, 50 and 100 respectively, for each hardware scenario). Thus, we had 100,000, 500,000 and 1,000,000 fitness calculations respectively (see Table 5).

The two hardware configurations we considered are quite distinct. The first consists of an Acer Aspire laptop computer with a dual-core processor clocking at 2.2 GHz and 4 GB of RAM, running Arch Linux in a virtual machine on Windows 7. The virtual machine was configured to use at most 1 GB of RAM and only one virtual processor. The second hardware configuration consisted of an HPC computer cluster (HP Cluster Platform 3000

Hardware	Fitness calculation	Computation Time [s]
Laptop	100,000	82
	500,000	340
	1,000,000	669
Cluster	100,000	29
	500,000	67
	1,000,000	134

Table 5: Performance analysis of our application

BL 2 × 220), consisting of 984 nodes with 20 TB of memory and capable of 105 Tflops. The operating system used here was Scientific Linux.

As expected, the application scales almost linearly on a sequential processor (the laptop). On a distributed system, however, computation times grow sub-linearly with an increasing number of fitness computations, showing that the applications benefits a coupled insular (multi-model) PSO setup when the number of fitness computations increases.

Part III
Conclusion

In conclusion the results obtained in WP7 after this second year of the MAPPER project correspond rather well to the description of work. All applications have been adapted to the proposed formalism, although with different levels of maturity. For some applications, further developments and testings are still needed whereas, for other ones the effort is now on running a production codes in order to obtain new scientific results.

This inhomogeneity in the development and in the presentation of the applications reflects the different needs, pre-exposure to the approach, and historical backgrounds of each group. But, at this stage, we can say that all applications are amenable to the MAPPER framework, and that a convergence of the methodology is emerging, thanks to the tools offered by WP8.

Several articles relating either the approach or advances in specific scientific domains have been published or submitted. We can also notice that the MAPPER project, with its proposed computational framework has stimulated the development of better versions of the existing codes, with a larger scales and more ambitious scientific goals.

Performance evaluation is still at an early phase for some groups but rather elaborated for others. Execution times have been measured in various situations but there is still a need for a uniform presentation and simple metrics to assess the cost/benefit of the Distributed Multiscale Computation proposed in MAPPER. However, some analysis already show that, for large enough problems, the overhed of the MAPPER approach is acceptable and even negligible.

References

[1] MAD software. `https://gs2.mapper-project.eu/mad/`.

[2] MAPPER pre-Production Infrastructure. `http://www.mapper-project.eu/web/guest/wiki/-/wiki/Main/MAPPER+pre-Production+Infrastructure`.

[3] PALABOS toolkit. `http://www.palabos.org/`.

[4] STL. `http://en.wikipedia.org/wiki/STL_(file_format)`.

[5] Sun Grid Engine. `http://www.oracle.com/us/sun/index.htm`.

[6] EFDA Task Force on Integrated Tokamak Modelling: *https://www.efda-itm.eu.*

[7] ITER organization: *http://www.iter.org.*

[8] Distibuted computing environments - gridspace technology, 2011.

[9] L. Axner, J. Bernsdorf, T. Zeiser, P. Lammers, J. Linxweiler, and A.G. Hoekstra. Performance evaluation of a parallel sparse lattice boltzmann solver. *Journal of Computational Physics*, 227(10):4895 – 4911, 2008.

[10] Y. Barlas. Model validation in systems dynamics. *International Systems Dynamics Conference*, 1994.

[11] Mohamed Ben Belgacem, Bastien Chopard, and Andrea Parmigiani. Coupling method for building a network of irrigation canals on a distributed computing environment. In G.C. Sirakoulis and S. Bandini, editors, *ACRI 2012*, volume 7495 of *LNCS*, page 309.318. Springer-Verlag Berlin Heidelberg, 2012.

[12] J. Borgdorff, D. Groen, S. Ferlin, I. Saverchenko, J. Suter, A. Hoekstra, and P. Coveney. Multiscale simulations on distributed european e-infrastructures. 10(1):72–77, 2012.

[13] Joris Borgdorff, Carles Bona-Casas, Mariusz Mamonski, Krzysztof Kurowski, Tomasz Piontek, Bartosz Bosak, Katarzyna Rycerz, Eryk Ciepiela, Tomasz Gubała, Daniel Harezlak, M Bubak, Eric Lorenz, and Alfons G Hoekstra. A Distributed Multiscale Computation of a Tightly Coupled Model Using the Multiscale Modeling Language. *Procedia Computer Science*, 9:596–605, 2012.

[14] B. Chen, J. R. G. Evans, H. C. Greenwell, P. Boulet, P. V. Coveney, A. A. Bowden, and A. Whiting. A critical appraisal of polymer–clay nanocomposites. *Chem. Soc. Rev.*, 37(3):568–594, 2008.

[15] K.H. Cho, S.M. Choo, S.H. Jung, J.R. Kim, H.S. Choi, and J. Kim. Reverse engineering of gene regulatory networks. *IET Systems Biology*, pages 149–163, 2007.

[16] D.P. Coster, V. Basiuk, G. Pereverzev, D. Kalupin, R. Zagórksi, R. Stankiewicz, F. Imbeaux P. Huynh, and Members of the Task Force on Integrated Tokamak Modelling. The european transport solver. *IEEE Transactions on Plasma Science*, 38(9):2085–2092, 2010.

[17] M. Davidson, E. amd Levin. Gene regulatory networks. *Proceedings of the National Academy of Sciences of the United States of America*, 102(14):4935, 2005.

[18] D. Groen, J. Hetherington, H. B. Carver, R. W. Nash, M. O. Bernabeu, and P. V. Coveney. Analyzing and modeling the performance of the HemeLB lattice-boltzmann simulation environment. *submitted to JOCS*, 2012. arXiv:1209.3972.

[19] D. Groen, S. Rieder, P. Grosso, C. de Laat, and P. Portegies Zwart. A light-weight communication library for distributed computing. *Computational Science and Discovery*, 3(015002), August 2010.

[20] D. Groen, J. Suter, and P. V. Coveney. Constructing distributed multiscale simulations of clay-polymer nanocomposites. In *Seventh IEEE international conference on e-Science and Grid computing: Stockholm, Sweden*, pages 105–111, Piscataway, NJ, December 2011. IEEE Computer Society.

[21] G. Huysmans, J. Goedbloed, and W. Kerner. Isoparametric bicubic hermite elements for the solution of the Grad-Shafranov equation. *CP90 Conference on Comp. Physics, Word Scientific Publ. Co.*, 371, 1991.

[22] G. T. A. Huysmans, S. E. Sharapov, A. B. Mikhailovskii, and W. Kerner. Modeling of diamagnetic stabilization of ideal magnetohydrodynamic instabilities associated with the transport barrier. *Phys. Plasmas*, 8(10):4292, 2001.

[23] F. Imbeaux, J.B. Lister, G.T.A. Huysmans, W. Zwingmann, M. Airaj, L. Appel, V. Basiuk, D. Coster, L.-G. Eriksson, B. Guillerminet, D. Kalupin, C. Konz, G. Manduchi, M. Ottaviani, G. Pereverzev, Y. Peysson, O. Sauter, J. Signoret, and P. Strand. A generic data structure for integrated modelling of tokamak physics and subsystems. *Computer Physics Communications*, 181(6):987–998, 2010.

[24] J. Kennedy and R Eberhart. Particle swarm optimization. In *Proceedings of IEEE International Conference on Neural Networks*, volume IV, pages 1942–1948, 1995.

[25] C. Konz, G.T.A. Huysmans, W. Zwingmann, M. Ottaviani, B. Guillerminet, F. Imbeaux, P. Huynh, V. Drozdov, P. Strand, JET-EFDA, and ITM-TF contributors. Scientific workflows for the linear MHD stability analysis chain. *Proc. of the 37th EPS Conference on Plasma Physics*, 2010.

[26] D. Lagrava, O. Malaspinas, J. Latt, and B. Chopard. Advances in multi-domain lattice Boltzmann grid refinement. *Journal of Computational Physics*, 231(14):4808 – 4822, 2012.

[27] C.F. Maggi, R.J. Groebner, C. Angioni, T. Hein, L.D. Horton, C. Konz, A.W. Leonard, C.C. Petty, A.C.C. Sips, P.B. Snyder, J. Candy, R.E. Waltz, ASDEX Upgrade, and DIII-D Teams. Pedestal and core confinement of hybrid scenario in ASDEX Upgrade and DIII-D. *Nucl. Fusion*, 50(2):025023, 2010.

[28] Marco D Mazzeo and Peter Coveney. HemeLB: A high performance parallel lattice-Boltzmann code for large scale fluid flow in complex geometries. *Comput. Phys. Commun.*, 178(12):894–914, 2008.

[29] Marco D Mazzeo, Steven Manos, and Peter Coveney. In situ ray tracing and computational steering for interactive blood flow simulation . *Comput. Phys. Commun.*, 181:355–370, 2010.

[30] P. Mendes, W. Sha, and K. Ye. Artificial gene networks for objective comparison of analysis algorithms. *Bioinformatics*, 19(2):122–129, 2003.

[31] A. Parmigiani, J. Latt, M. Ben Belgacem, and B. Chopard. A Lattice Boltzmann simulation of the Rhone river. *Int. J. Mod. Phys. C*, In preparation.

[32] B.E. Perrin, L. Ralaivola, A. Mazurie, S. Bottani, J. Mallet, and F. d'Alche Buc. Gene networks inference using dynamic bayesian networks. *Bioinformatics*, 19(2):138–148, 2003.

[33] Kauffman S. *The origins of order: Self-organization and selection in evolution.* Oxford University Press, 1993.

[34] S. K. Sadiq, M. D. Mazzeo, S. J. Zasada, S. Manos, I. Stoica, C. V. Gale, S. J. Watson, P. Kellam, S. Brew, and P. V. Coveney. Patient-specific simulation as a basis for clinical decision-making. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 366(1878):3199–3219, 2008.

[35] M.A. Savageau. *Biochemical systems analysis: A study of function and design in molecular biology.* Addison-Wesley, Reading, Mass., 1976.

[36] B. D. Scott. Computation of electromagnetic turbulence and anomalous transport mechanisms in tokamak plasmas. *Plasma Phys. Control. Fusion*, 45:A385, 2003.

[37] P.T. Spellman, G. Sherlock, M.Q. Zhang, V.R. Iyer, K. Anders, M.B. Eisen, P.O. Brown, D. Botstein, and B. Futcher. Comprehensive identification of cell cycle-regulated genes of the yeast saccharomyces cerevisiae by microarray hybridization. *Molecular Biology of the Cell*, 9:3273–3297, 1998.

[38] J. Suter, D. Groen, L. Kabalan, and P. V. Coveney. Distributed multiscale simulations of clay-polymer nanocomposites. In *Materials Research Society Spring Meeting*, number 1470, San Francisco, CA, April 2012. MRS Online Proceedings Library.

[39] J. L. Suter and P. V. Coveney. Computer simulation study of the materials properties of intercalated and exfoliated poly (ethylene) glycol clay nanocomposites. *Soft Matter*, 5(11):2239–2251, 2009.

[40] J. Vohradský. Neural network model of gene expression. *The FASEB Journal*, 15:846, 2001.

[41] E.O. Voit and J.H. Schwacke. Understanding through modeling: A historical perspective and review of biochemical systems theory as a powerful tool for systems biology. In A.K. Konopka, editor, *Systems biology: Principles, methods and concepts*, pages 28–77, 2007.

[42] C.J. Willmott and K. Matsuura. Advantages of the mean absolute error (mae) over the root mean square error (rmse) in assessing average model performance. *Climate Research*, 40:79–82, 2005.

A Hydrology

A.1 T_{cpu} and T_{com} equations

In this appendix, we will develop a detailed equations of T_{cpu} and $T_{com}(B_k)$.

Let F_k be the theoretical floating-point computing speed (FLOPS) of a cluster node, and x the number of floating point operations required to treat one lattice grid. Theoretically, we can determine T_{cpu} as

$$\begin{aligned} T_{cpu}(p_k, \Delta x) &= c_0 \times \frac{L_x.Ly.L_z}{\Delta x^3} \times \frac{x}{F_k \cdot p_k} \\ &= c_0 \times (n_x.n_y.n_z) \times \frac{x}{F_k \cdot p_k} \\ &= c_0 \times (n_x.n_y.n_z) \times \sigma_{cpu} \\ &= A_k/\Delta x^3 \end{aligned} \tag{13}$$

Where c_0 is a constant and σ_{cpu} the time required to compute one lattice node.

As described in listing 1, during each iteration, the communication time $T_{com}(B_k)$ is determined by T_{get}, $T_{exchange}$ and T_{up}, the duration times taken by the GetBoundaryData(), SendReceiveBoundaryData() andUpdateBoundaryData() operations, respectively. We can write:

$$T_{com}(B_k) = T_{get} + T_{exchange} + T_{up} \tag{14}$$

For communication between submodels (kernels), we consider a synchronous blocking point-to-point communication model where a submodel sends and receives data sequentially and not in parallel. Besides, each submodel sends and receives data to and from s_j other submodels depending on the coupling schema and the number of MAPPER conduits d_k that a submodel has. In addition, during each iteration, a boundary operation requires the processing of $e \times n_y.n_z$ (e stands for the number of required lattice along the x-axis) lattice cells, each having f variables (distribution functions) and a given data type T. Now, we can say that $T_{com}(B_k)$ depends also on the conduits network speed S. With all these definitions, we can now write

$$\begin{array}{ll} T_{get}(p_k, \Delta x) & = \frac{c_1}{p_k} \times d_k.e.n_y.n_z \times f \times sizeof(T) \div S \\ T_{exchange}(\Delta x) & = c_2 \times e.n_y.n_z \times f \times sizeof(T) \div S \\ T_{up}(p_k, \Delta x) & = \frac{c_3}{p_k} \times e.n_y.n_z \times f \times sizeof(T) \div S \end{array} \tag{15}$$

And finally we obtain:

$$\begin{array}{ll} T_{com}(B_k) & = (\frac{c_1+c_3}{p_k} + c_2) \times e.n_y.n_z \times f \times sizeof(T) \div S \\ & = B_k/\Delta x^2 \end{array} \tag{16}$$

A.2 XML based Jobs profiler for QCG broker

Listing 2: QCG based workerflow to connect two 3D canal sections over two distributed clusters

```
<qcgJob appId="MAPPER" xmlns:jxb="http://java.sun.com/xml/ns/jaxb" xmlns:xsi="
    http://www.w3.org/2001/XMLSchema−instance">
 <task persistent="true" taskId="task">
           <requirements>
                 <topology>
                       <processes processesId="LeftSection">
                          <processesCount>
                             <value>10</value>
                          </processesCount>
                          <candidateHosts>
                                   <hostName>galera.task.gda.pl</
                                       hostName>
                           </candidateHosts>

     <reservation type="LOCAL">NO_RESERVATION</reservation>
     <resourceRequirements>
```

```
                <computingResource>
            <hostParameter name="memory">
                <value>20000</value>
            </hostParameter>
                <computingResource>
                <resourceRequirements>
                                                <processes>
                                                <processes processesId="RightSection">
                                                    <processesCount>
                                                        <value>10</value>
                                                    </processesCount>
                                                    <candidateHosts>
                                                        <hostName>reef.man.poznan.pl</hostName>
                                                    </candidateHosts>
                                                    <reservation type="LOCAL">NO_RESERVATION
                                                        </reservation>
              <resourceRequirements>
                <computingResource>
            <hostParameter name="memory">
                <value>20000</value>
            </hostParameter>
                </computingResource>
                </resourceRequirements>
                                                </processes>
                                        </topology>
                                </requirements>
        <execution type="mapper">
         <executable>
          <application name="muscle2"/>
         </executable>
         <arguments>
          <value>canals.qcg.cxa.rb</value>
         </arguments>
         <stdout>
          <directory>
           <location type="URL">gsiftp://qcg.man.poznan.pl/~/CANALS/results/${JOB_ID
                }.output</location>
          </directory>
         </stdout>
         <stderr>
          <directory>
           <location type="URL">gsiftp://qcg.man.poznan.pl/~/CANALS/results/${JOB_ID
                }.error</location>
          </directory>
         </stderr>
         <stageInOut>
          <file name="canals.qcg.cxa.rb" type="in">
           <location type="URL">gsiftp://qcg.man.poznan.pl/~/CANALS/inputs/link.cxa</
                location>
          </file>
          <file name="config.xml" type="in">
           <location type="URL">gsiftp://qcg.man.poznan.pl/~/CANALS/inputs/config.xml
                </location>
          </file>
          <file name="canals.pre.sh" type="in">
           <location type="URL">gsiftp://qcg.man.poznan.pl/~/CANALS/inputs/canals.pre.
                sh</location>
          </file>
          <file name="canals.post.sh" type="in">
           <location type="URL">gsiftp://qcg.man.poznan.pl/~/CANALS/inputs/canals.post
                .sh</location>
          </file>
```

```
    <directory name="stats" type="out">
     <location type="URL">gsiftp://qcg.man.poznan.pl/~/CANALS/results/${JOB_ID
          }.stats</location>
    </directory>
   </stageInOut>
   <environment>
    <variable name="QCG_MODULES_LIST">canals/MM</variable>
                                    <variable name="QCG_PREPROCESS">./canals.pre.sh
                                          </variable>
                                    <variable name="QCG_POSTPROCESS">./canals.post
                                          .sh</variable>
                                    <variable name="MUSCLE_ARGS">--reverse</
                                          variable>
   </environment>
  </execution>
  <executionTime>
   <executionDuration>P0Y0M0DT3H30M</executionDuration>
  </executionTime>
 </task>
</qcgJob>
```

B Computational Biology

B.1 Abbreviations and Glossary of Terms

COPASI	Complex pathway simulator A software tool facilitating the modelling, simulation and analysis of biochemical networks and their dynamics.
DNA	Deoxyribonucleic acid Macromolecule storing genetic information used to produce RNA and proteins.
GRN	Gene-regulatory network A biochemical network of genes that controls biological processes by regulating the production of RNA and protein in response to external and internal stimuli.
GRN model	A GRN model is a mathematical representation of a GRN.
GRN system	Same as GRN.
MAE	Mean absolute error The average of the absolute errors.
MMS	Multiscale modelling and simulation An approach to model and simulate systems which involve features on multiple scales (typically, time and space). A key issue involves the linking or coupling of single scale models.
ODE	Ordinary differential equation An equation containing a function of one independent variable and its derivatives.
PSO	Particle swarm optimization Evolutionary optimization technique inspired by swarm behaviour.
RMSE	Root mean squared error The square root of the sum of the squared errors divided by the number of observations.
RNA	Ribonucleic acid Macromolecule storing genetic information transcribed from DNA.
SBML	Systems Biology Mark-up Language A standard XML-based format to represent, store and share biochemical models.
SIF	Simple interaction format A simple, standard file format supporting the storing and sharing graphs and networks.