D2.3 Final Report on Multiscale Computing Patterns including their Performance

Due Date	Month 36
Delivery	Month 36
Lead Partner	UvA
Dissemination Level	Public
Status	Final
Approved	Internal review
Version	1.0

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

DOCUMENT INFO

Date and version number	Author	Comments
20.08.2018 v0.1	Alfons Hoekstra	Skeleton of report, first intro text.
02.09.2018 v0.2	Alowayyed, Luk, Hoenen	First full version
04.09.2018 v0.3	Alfons Hoekstra	Edited first full version, addition of missing paragraphs.
04.09.2018 v0.3	Alowayyed, Hoenen, Bosak, Vassaux, Luk	Add missing paragraphs and check text coherency
06.09.2018 v0.4	Saad Alowayyed	Merge to the new version maintain equations and add to the conclusion.
13.09.2018 v0.5	Hoekstra, Alowayyed	Finalise first draft
20.09.2018 v0.6	Saad Alowayyed	Update structure and reviewers feedback
24.09.2018 v0.7	Alowayyed, Vassaux, Luk	HMC results and fusion example feedback
25.09.2018 v1.0	Hoekstra, Alowayyed	Final version

CONTRIBUTORS

Contributor	Role
Alfons Hoekstra (UvA)	Editor, PI and WP2 leader
Saad Alowayyed (UvA)	contributor
Onnie Luk (MPG)	contributor
Oliver Hoenen (MPG)	contributor
Bartosz Bosak (PSNC)	contributor
Maxime Vassaux (UCL)	contributor
David Coster (MPG)	Internal reviewer
Tomasz Piontek (PSNC)	Internal reviewer

TABLE OF CONTENTS

1 Executive summary

After a short review of the Multiscale Computing Patterns and related software, this deliverable reports the main developments and the workflow of the Multiscale Computing Patterns and Algorithms software. The Multiscale Computing Patterns Algorithms and software has been extended with new capabilities, most notably the performance database with is filled using tools from WP4 and is used by the Pattern-Driven planner to propose candidate deployment and execution plans, based on measurements of the single scale components, to the QCG pattern aware planner. Second, automating the benchmark process, to run on all EEE HPC resources, to populate the database and make the performance data available when requested. An example of using the Multiscale Computing Patterns and Algorithms software is illustrated, showing that a wide range of variables for Multi-objective Optimisation algorithms can be chosen.

The third and final pattern, the Heterogeneous Multiscale Computing pattern, has been implemented, and first benchmarks that demonstrate how to deal with the inherently dynamic nature at runtime of applications that utilise this pattern have been performed. This mainly involves an adaptive and dynamic load-balancing scheme that orchestrates micro-scale simulation. Two examples of different HMM application are illustrated.

2 Report

2.1 Introduction

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

This deliverable D2.3 will, quoting again from the DoA, "report on the performance of all selected multiscale applications as measured on the Experimental Execution Environment and selected production environments. It will give a detailed account of various metrics of resource usage, including wall clock time, data throughput, scalability and energy consumption. It will also provide a detailed account on the performance prediction models." Most of the application specific measurements will be reported in deliverable D3.3 and performance profiling is part of deliverable D4.3. Here we will describe new results, building upon deliverables D2.1 and D2.2, that are generic to the Multiscale Computing Patterns. It is mainly based on work performed in task 2.3 (Development of the multiscale computing

patterns, which was extended with 6 months to M30) but also on 2.4 (profiling and performance measurement) and task 2.5 (Performance prediction modelling).

In deliverable D2.1 we wrote that “converting MML for ComPat, task 2.2, has partly been addressed” and “we will therefore continue to put effort in this task, slightly deviating from the original DoA and extending this task into the second year of the project”, followed by deliverable D2.2 where we wrote that “we have now reached a point, given the current design of the Multiscale Computing Patterns (MCPs), that the information needed by them should not be embedded in MML, but provided as separate component. This specifically relates to performance data of the single scale models, as described in section 2.3.1 and illustrated in section 2.4. We have, however, decided to not close task 2.2 yet, as further development of the MCPs may warrant updates to MML.” We have now reached a final conclusion that updates of MML are not needed for the MCPs, closing task 2.2.

The further development of the actual patterns, task 2.3, has been the major focus of WP2 during M18 to M36 of the project. In collaboration with WP3, WP4, and WP5 and building upon the conceptual design of MCPs as reported in deliverable D2.1 and the Multiscale Computing Patterns and Algorithms software that was described in detail in deliverable D2.2, as demonstrated for specific applications in deliverable D3.2, we have now finalised the design for the specific architecture for the Multiscale Computing Patterns and Algorithms software, in line with the overall ComPat architecture described in deliverable D5.1. We have implemented all its main features, and worked out examples to illustrate some of these concepts.

The main developments, for the period of M24 - M36, were on the performance measurements database and how to integrate this component with the Optimisation component of the MCP and Algorithms software to predict the performance of the multiscale application based on measurements of the single scale components. Another major addition was to automate the benchmark process to populate the database and make the performance data available when requested. In addition to time-to-completion metrics, energy consumption was profiled and used as a criterion in the MCP and Algorithms software. Finally, the HMC pattern has been realised, implemented, and first tests have been performed.

This deliverable will first quickly review the MCPs (as described in detail in deliverable D2.1) and the design of the Multiscale Computing Patterns and Algorithms software (as described in detail in deliverable D2.2), and then continue to describe and illustrate further developments and implementations of the Multiscale Computing Patterns and Algorithms software, including realisation of the third MCP, the Heterogeneous Multiscale Computing.

2.2 Multiscale Computing Patterns and related Algorithms and Software

As a courtesy to the reader, and to keep this deliverable self-contained, In this section we provide a short summary of Multiscale Computing Patterns, as described in detail in deliverable D2.1 and the Multiscale

Computing Patterns and Algorithms software, as described in detail in deliverable D2.2. We do refer the reader to deliverables D2.1 and D2.2 and [1,18] for all details, including worked out examples.
We define Multiscale Computing Patterns (MCP) as "high-level call sequences that exploit the functional decomposition of multiscale models in terms of single scale models" [1]. We have identified three computing patterns that we believe are most relevant for high performance multiscale computing, namely: Extreme Scaling (ES), Replica Computing (RC) and Heterogeneous Multiscale Computing (HMC).

The *Extreme Scaling* computing pattern represents a specific class of multi-scale applications where one (or perhaps a few) of the single scale models in the overall multiscale model dominates all others, in terms of computational and/or energy cost, by far. Such a dominating *primary model* is expected to scale to very large systems (i.e., multi-petascale or above) and the efficiency of the primary model largely determines the efficiency of the multiscale application. Consequently, one of our goals is to ensure minimal interference by the other single scale models, so-called *auxiliary models*. These typically have a much lower computational and/or energy cost, and could even be sequential codes [1].

Replica Computing is a multiscale computing pattern that combines a potentially very large number of terascale and petascale simulations (also known as 'replicas') to produce scientifically important and statistically robust outcomes. The replicas are not part of a larger spatial structure (as is the case in Heterogeneous Multiscale Computing), but they are applied to explore a system under a broad range of conditions. Replica Computing is set up through an initialisation stage, which determines the simulations required to explore or incorporate a given parameter space. This initialisation is then followed by one or more sequences of simulation and data processing [1].

Heterogeneous Multiscale Computing pattern will be presented and discussed in depth in section 2.4.2.

The key idea is that we define generic task graphs for each pattern, such that application specific task graphs can be embedded in the generic task graphs. We use the generic task graph to obtain an optimised mapping of the application to an HPC resource, and try to find generic algorithms for this. An MCP therefore is a tuple of a generic task graph plus data or models on the performance of single scale models, a specification of a specific multiscale application in terms of the xMML and a set of algorithms and heuristics that combine this into detailed input/configuration files for the execution environment [1].

We have designed, implemented, and tested a first version of the Multiscale Computing Patterns and Algorithm software [18]. This software consists of Description, Optimisation and Execution components.

The Description component translates the task graph, representing a multiscale simulation, to a particular type of multiscale computing pattern. Second, the Optimisation component selects and

applies dedicated algorithms to find the most suitable mapping between submodels and available HPC resources. Third, the Execution component involves a middleware layer that maps submodels to the number and type of physical resources based on the suggestions proposed by the Optimisation part together with infrastructure-specific metrics such as (expected) queueing time and resource availability.

The main purpose of the Multiscale Computing Patterns and Algorithm software is to leverage the multiscale computing patterns to simplify and automate the execution of complex multiscale simulations on high performance computers, and to provide both application-specific and pattern-specific performance optimisation [18].

2.3 The Multiscale Computing Patterns and Algorithms software

The development of the Multiscale Computing Pattern and Algorithms software is shown in Fig. 1. This figure is similar to the one shown in D2.2 and [18] with more technical details.

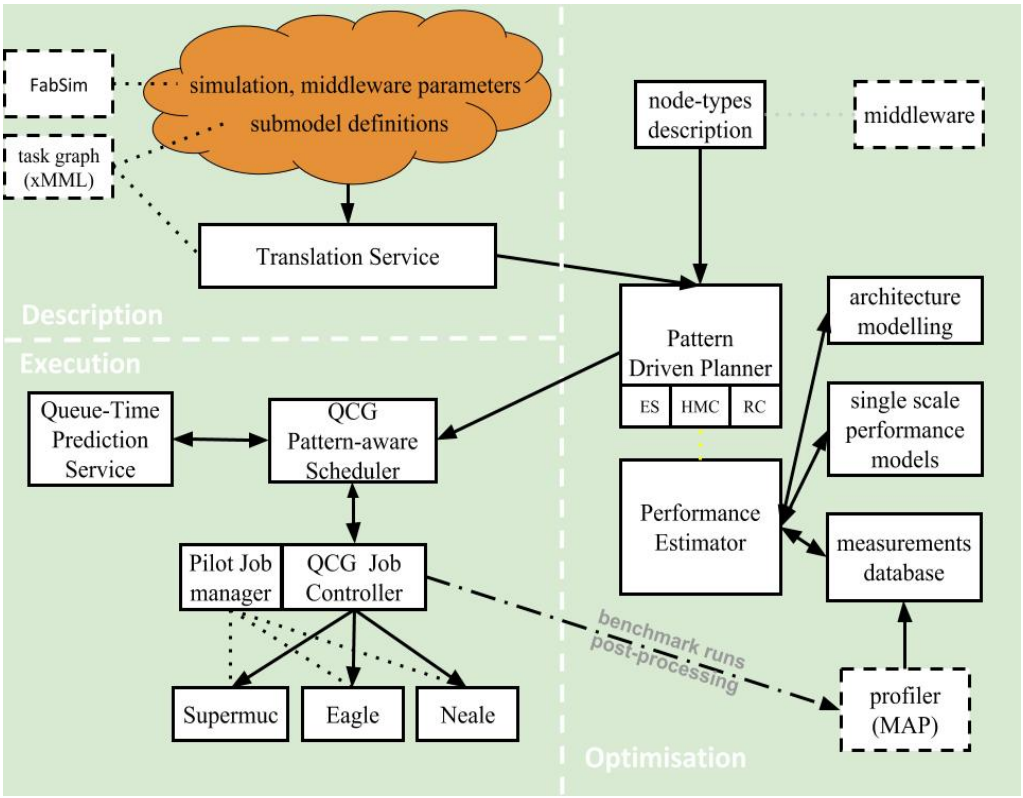

Figure 1: ComPat stack developement status and plans

Figure shows the main components that comprise the MCP and Algorithms software (i.e., Description, Optimisation and Execution). In the next subsections, we will report the developments and the workflow of the MCP and Algorithms software on two mode; the normal “run” and the “benchmark” mode.

2.3.1 Design and Implementation

The next subsections show the new and integrated components of the MCP and Algorithm software on the three different parts; the Description, Optimisation and Execution.

Description part

The translation service from the Description part has been redesigned and now takes user input to allow a seamless integration with the Fabsim tool. In terms of the embedded algorithms, the only modification is the addition of a module to parse the application task graph (obtained from xMML) and determine from it a formula to estimate the overall time of the application depending on the time spent in each submodel involved. This formula is stored within the inputs of the Pattern-Driven Planner (as ***modeltime***) and will later be used by the Optimisation part to predict the makespan for a given configuration. An example of this is shown in section 2.4 .

Optimisation part

We have implemented the **performance database** to store, retrieve, query and interpolate performance data of the single scale components take make up a multiscale simulation in a faster and semi-automated form. The database is deployed in the LRZ SuperMUC facility and is accessible from the QCG head node. This database contains information of multiscale runs, different kernels with their main parameters, the detailed kernel performance / energy matrices and systems involved node types. The kernel performance includes all the information needed to do full performance measurements and predictions such as the overall run time, muscle time with communication between kernels, mpi time represent the communication within a kernel, IO time and energy profiling data (for more in-depth information we refer to deliverables D4.3 and D6.3).

The **node types** are detected automatically from the middleware. Currently, this is done by requesting the available nodes number of cores per node and number of nodes.

To populate the database, the MCP and Algorithms software has an option to run in "**benchmark** mode". In this mode, the problem size is smaller or the duration is shorter than the original run. The link between the benchmark problem and the production run problem have to be dynamically interpreted in the database as a set of parameters. The scale factor, between different parameters, is an input to the Pattern-Driven Planner part of the MCP and Algorithms software. Figure 2 illustrates the benchmark run mode.

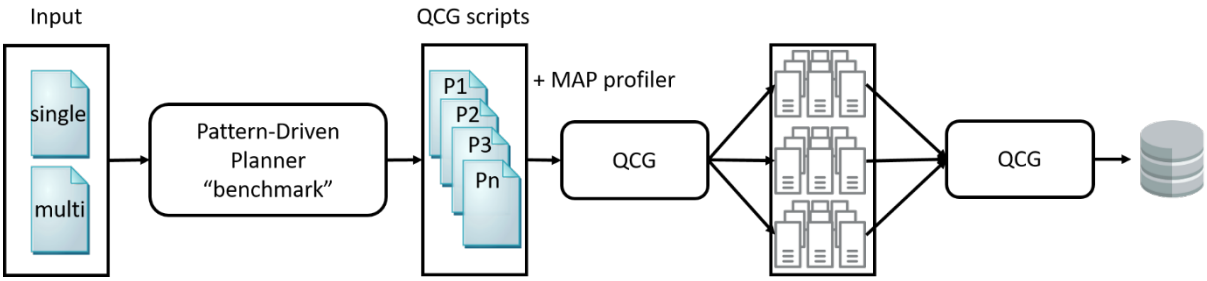

Figure 2: Multiscale Computing Pattern and Algorithms software workflow in benchmark mode.

First, the Pattern-Driven Planner takes the description of the single-scale and multi-scale as input. Next, the Pattern-Driven Planner generates different execution plans -in QCG scripts box- to ensure running the benchmark problem in all node types and hosts where each single-scale model is available, and on each possible set of core count determined from information provided by each single-scale model. This information can be the minimum and maximum number of cores, and possibly an expression to calculate all valid numbers within this range. The information on the amount of memory required per core is not implemented in the system, however, the user can decide to make a submodel non-available on a given node type if this node type has less than the minimum required memory to run the code.

After all valid execution plans have been set up, each execution plan is modified to call the MAP profiling tool (see deliverable D4.1 and D4.2) for each single-scale model, and an additional stage-out phase is added to collect all the performance profiles. These additions can be done automatically or necessitate the user to set up specific inputs for the MCP and Algorithms software, depending on the use-case. Staged-out performance profiles are then automatically uploaded to the database, from which they can be further queried in order to predict the performances of further runs that can be compared or extrapolated from these benchmarks (see Figure). This is the way to do automated benchmarking of multiscale applications on the MCP and Algorithms software to seamlessly schedule and deploy a multiscale simulation in the most efficient way to a supercomputing environment, meeting the constraints of both the user and the owners of the supercomputing facility.

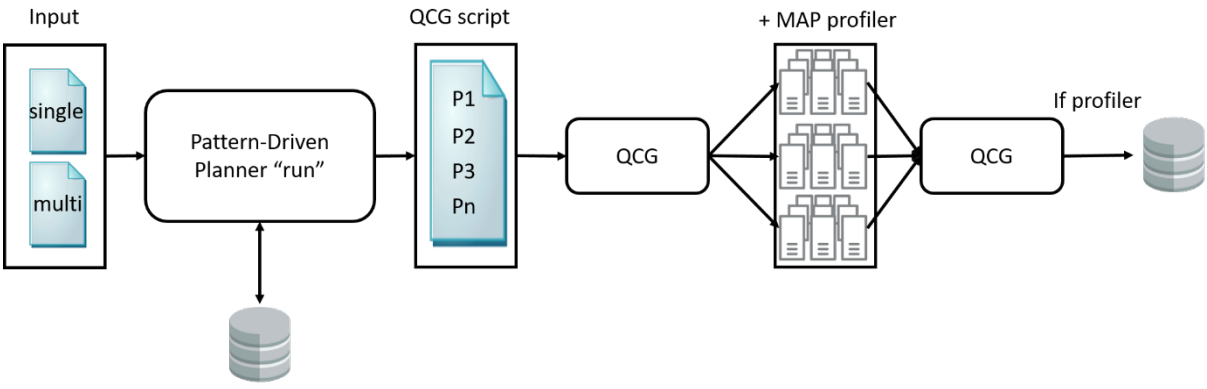

Figure 3: Multiscale Computing Patterns and Algorithms software workflow in run mode.

Figure shows the workflow of the **normal run**. In this mode, the Pattern-Driven Planner generates, using the same mechanism as in benchmark, a number of execution plans. The different is that for each execution plan we had queried the database to predict the performances (time, energy, etc), which are compared or extrapolated from the benchmarks runs we launched. Each execution plan is filled with time, energy and efficiency numbers. Then, they are ordered based on the criteria chosen time-to-completion, energy consumption, and efficient usage of resources (see section 2.4 for example) in one QCG script. QCG will then calculate the queue time using the Queue Time Prediction Service (see D6.3) and re-order these plans based on the new information (for further details we refer to D5.3 section 2.3). If the profile option is on "–P option", then a MAP profiler will be used like what we did for the benchmark and the database is updated automatically.

We originally planned to include a simulation of the resources on which the multiscale models would be executed. Such component would allow estimating the performance of the single-scale models on some architecture, and with that information, the behaviour of a multiscale application could be estimated. This is certainly an interesting option to have to explore the behaviour of MCPs and multiscale applications expressed in terms of MCPs on non-existing architectures. Our MCP and Algorithms software is completely ready for this option. Unfortunately, we have not been able to deliver this option. Having the choice on spending our resources on hardening and finalising key components in our software stack, or adding an additional new component, we opted for the first. In this way, we have a fully functional software stack that can be used for existing Tier0 machines. Unfortunately, we cannot predict (yet) the behaviour on future machines. This we consider future work.

Execution part

The new QCG schema expose a set of new capabilities provided on a middleware layer and tailored to enable multi-criteria brokering as well as to improve the support for energy-efficient execution of High Performance Multiscale Computing application on computing resources. The newly implemented pattern-aware brokering plugin for QCG-Broker allows selecting execution resources based among others on time to completion, energy-usage and CPU-Hours-usage metrics. In order to support the first metric, i.e. time-to-completion, the new brokering plugin has been integrated with an additional new component implemented based on the cooperation between WP5 and WP6, the Queue Time Prediction Service, which provides statistical estimations for queueing time. For more information, please refer to the deliverables D5.3 and D6.3.

A new component was implemented to support HMC pattern, the pilot job. It is a normal job submitted to supercomputers to reserve resources and use the resources as a unified unit. The QCG Pilot Job Manager system has a similar mechanism by allowing the management of the resources on the application level. The pilot job is akin to a traditional job array, but it allows the scheduling and

execution of small and dynamic jobs with different resource requirements. For more details, we refer to D5.3.

2.4 Using the Multiscale Computing Patterns and Algorithms software

In this section, we will first report the usage of the MCP and Algorithms software and then we will discuss the third and last pattern, the Heterogeneous Multiscale Computing pattern, in more details.

2.4.1 Example -- Fusion

As reported in deliverables D2.2 and D3.2, the global turbulence (fusion) application fits into the Extreme Scaling computing pattern, for one of the single scale submodels (turbulence model) is scalable and requires the majority of the computing resources when compared to the rest of the submodels (equilibrium, flux-to-transport-coefficient module, and transport models). The workflow topology is illustrated in Figure 4. The turbulence (TUR) is simulated by the GEM code, equilibrium (EQU) by CHEASE, transport (TRA) by ETS, and module (FDV) by IMP4DV. Each one of these submodels are wrapped around by a MUSCLE2 kernel. For details of fusion software we refer to deliverable D3.1.

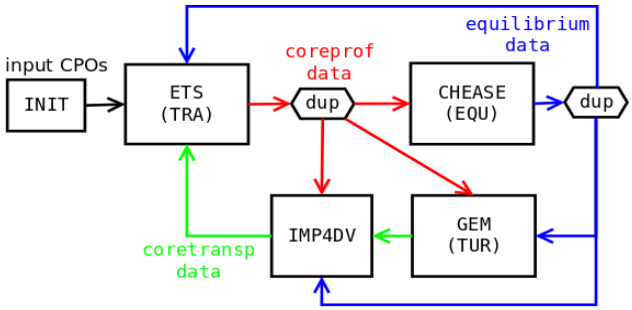

Figure 4: The topology of the fusion application.

This topology is an input to the coupling tools used (MUSCLE2) in form of connection schema represented in xMML. The MCP and Algorithms have the ability to automatically detect the topology and "propose" a multi-scale performance model as discussed in section 2.3.1. This model can be reviewed by the user and changed accordingly. In Figure 4, the multi-scale performance model is the sum of all components as one "motif". Other workflow topology (not presented in this deliverable) is assembled and tested. In that workflow, the GEM and CHEASE can be initiated at the same time and the multiscale performance model is the sum of the ETS, IMP4DV, and the maximum time between GEM and CHEASE.

The MCP and Algorithms software provides the ability to generate and possibly submit a set of benchmark cases to all HPC resources available in the EEE. These resources include LRZ SuperMUC's Thin, Fat, and Haswell node types; PSNC Eagle's Haswell-64, Haswell-128, and Haswell-

256 node types; STFC Neale's IvyBridge node type. A set of benchmarks contains five cases, where each case uses a different number of cores required by the kernels.

Each time we submit a set of benchmarks, the MAP integrates into the MUSCLE2 environment to monitor the performance of kernels and obtains performance profiles as shown in section 2.3.1. The performance numbers include runtime, time spent in MUSCLE2 operation, time spent in MPI, and energy consumption of the kernel. These profiles are then recorded into a database, and in the future the data can be extracted to produce estimated performance metrics for the multiscale application. There are three performance metrics that are taken into account: time to completion *T*, efficiency ε and total energy E_{tot}. While time to completion is straightforward, the efficiency and total energy consumption are not. The efficiency is defined as

$$\varepsilon = (N_P T_P + N_a T_a) / (N T),$$

with N_P, N_a, and N as the number of cores for primary, auxiliary, and multiscale models, respectively; and, T_P and T_a as the time spent in the primary and auxiliary models, respectively. The total energy consumed by the multiscale application is the sum of energy consumed by all MUSCLE2 kernels. This can contribute towards the next generation exascale computers, especially when energy consumption is heavily discussed as a measure of resource usage.

For the rest of this subsection, we present several examples. We request from the MCP and Algorithms software the best execution plans based on multiple criteria: time-to-completion, efficiency and energy usage. In this run, we ask for all the set of available executing configuration for the multiscale application running on minimum of 128 and maximum of 4099 cores. In addition, this instance of Fusion consists of two "kernels" ETS and GEM. We configure GEM to run on 128 – 2048 cores, while ETS runs on one core. We set the MCP and Algorithms software, in the following runs, to show the best three plans (-M 3 option).

Time-to-completion

In this example, we run the default behaviour of the MCP and Algorithms software. Here we request the minimum time-to-completion of fusion multiscale application on all the node types available on EEE. This small run, a benchmark run, consists of 4 multiscale iterations (motifs). The Pattern-Driven Planner will generate the following plans:

Table 1: Execution plans generated from the Pattern-Driven-planner for a small run with 4 motifs. These plans are ordered by the time-to-completion.

Plan name	Plan time (seconds)	Cores / sub-model	Energy (Joules)	Node type	Efficiency
Plan0	1960.56	(1,2048)	4155313.31	Haswell_64	0.91
Plan1	1972.79	(1,2048)	4720715.21	Haswell_128	0.92
Plan2	1997.92	(1,1024)	2605986.11	Haswell_64	0.93

We notice that each one of the plans listed in Table 1 are local to a node type. This is true in ES, unless there is a restriction such as running one code on one machine and other codes on another machine. The long communication time between different nodes, which is represented in the database under MUSCLE time, is the main reason to avoid the options of different node types. In addition, all plans are listed in the ascending order of time-to-completion. In the plan time, we added a safety limit time to the real run time. The safety limit time, which is set to a default value of 1800 seconds, is used to avoid the hard wall-clock limit that might occur due to the external factors (e.g., wake-up node times). The first plan, for example, has a real run time of 160.56 seconds to run 4 motifs.

Next, we show a "production" run, with a large number of iterations. In this run, we increase the number of iterations (motifs) from 4 to 300. By using the same setting as before, the Pattern-Driven Planner generates plans listed in Table 2:

Table 2: Execution plans generated from the Pattern-Driven-planner for a large run with 300 motifs. These plans are ordered by the time-to-completion.

Plan name	Plan time (seconds)	Cores / sub-model	Energy (Joules)	Node type	Efficiency
Plan0	13882.18	(1,2048)	312687325.6	Haswell_64	0.91
Plan1	14802.42	(1,2048)	355233820.1	Haswell_128	0.92
Plan2	16693.63	(1,1024)	196100453.9	Haswell_64	0.93

We notice that the generated plans are scaled with the change of this global parameter (number of motifs). In the database, the performance values are normalised with this value. The current problem is ~ 75 times larger than the previous example. The real run time of the first plan, after taking out the safety limit time, is 12080 seconds (3.3 hours), which is ~75 times longer. This scaling exercise can be done for the local parameter of the single-scale models as well. The same mechanism applies to the energy consumption.

Efficiency

In this example, we demonstrate the efficiency criterion (-E option) in the same way as the previous example. However, we only choose from the nodes available in SuperMUC. This is done using "-H

HOSTNAME". By choosing the efficiency, the execution plans are chosen based on the shortest time first, then, we order the first *n* plans based on efficiency.

Table 3: Execution plans generated from the Pattern-Driven-planner for a small run with 4 motifs. These plans are ordered by the efficiency.

Plan name	Plan time	Cores / sub-model	Energy	Node type	Efficiency
Plan0	70403.3	(1, 128)	226717053.7	Thin	0.96
Plan1	44501.06	(1, 256)	256033981.7	Thin	0.94
Plan2	31549.94	(1, 512)	338506140.6	Thin	0.92

The resulting execution plans are, as expected, request less core counts compared to the previous examples. The efficiency can be seen as the effect of the auxiliary single-scale models on the multi-scale execution time.

After defining, discussing, and providing proofs of concept for two computing patterns (namely Extreme Scaling and Replica Computing) in deliverables D2.2 and D3.2, we will now report on the third and most dynamic computing pattern, heterogeneous multiscale computing (HMC) [1]. Moreover, the results of two HMM applications using the Optimisation part are presented.

2.4.2 Heterogeneous Multiscale Computing

The HMC pattern implements a family of multiscale models which, using the MMSF terminology, are single domain with multiple and dynamic instantiations of the micro-scale dynamics, utilising a call/release coupling template [2]. The heterogeneous multiscale method [3,4] represents the most obvious multiscale model that fits the HMC patterns, but other examples would include running uncertainty quantification on extreme scaling applications using so-called semi-intrusive algorithms [5]. The later will be further explored and developed in the recently started FET-HPC project VECMA.

In this set of multiscale applications, a complex phenomenon is modelled by employing a numerical solver for the macro-scale equations and obtaining missing properties (e.g., constitutive equations) from suitable micro-scale simulations. Hence, the macro-scale model(s) are coupled, usually for each iteration, with a large and dynamic number of micro-scale models (Figure 5).

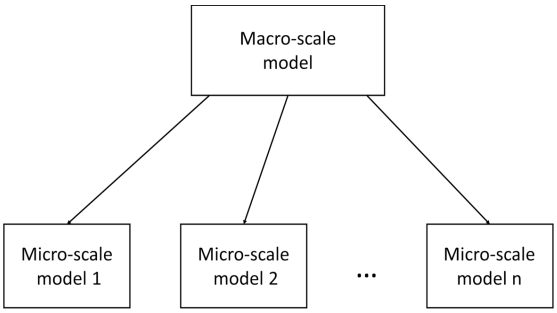

Figure 5: The computational structure of hierarchical multiscale applications.

The HMC pattern is based on, and inspired by, the hierarchical multiscale method (HMM) [4]. The primary potential and advantage of HMM is capturing of the dynamics at the macro-scale level by considering some microscopic details of the problem. Thus, HMM is a modelling technique used to numerically solve multiscale problems by coupling multiple submodels together, each of which solves a component separately. An overall macro-scale model then emerges when the separate submodels are combined. Micro-scale models are employed to resolve each unknown component of the problem separately and return the result to the macro-scale model. In this case, heterogeneous suggests that the problem is multi-physical in nature [6]. Frameworks taking into account the computational aspects of HMM, certainly in relation to HPC, are rare [1,7]. For this reason, we propose heterogeneous multiscale computing.

From a computational point of view, the potentially large number and dynamic nature of the number of required micro-scale models, which are determined at runtime, can become a bottleneck in production runs [1]. The number of micro-scale models also depends on the spatial properties of the macro-scale model [8]. Moreover, micro-scale models can be computationally intensive. Figure 6 shows the execution time of the micro-scale model as a function of the number of processors. In the micro-scale, we simulated the red blood cells and platelets suspensions in plasma using HemoCell [14]. In this specific benchmark, it is clear that the minimal execution time is obtained using 16 processors. Increasing the number of processors to more than 16 actually increases the execution time. This is a well-known phenomenon in strong scaling of parallel applications, and is due to increasing overhead time as the number of processors increases. Thus, in this example, the boundary limits for the number of processors for the micro-scale models are P_{min}= 1 and P_{max} = 16.

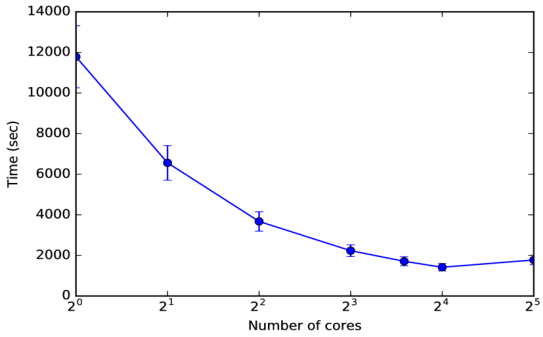

Figure 6: Average performance of the cell-suspension LBM solver, representing micro-scale models.

To reduce the large number of micro-scale models, a surrogate model should be utilised in between the micro- and macro-scales. The surrogate model contains a database that is used to store data to avoid duplicating calculations and to build a surrogate model (e.g., based on a Gaussian processes) to conduct interpolations for similar parameters. This process decreases the required number of micro-scale models requested, which in turn decreases computation time. Generally, this solution is practical and has been used in multiple fields [9,10]. In this example, we replaced this process with performance figures, which mimic the operation.

Surrogate model performance analysis

The number of micro-scale models changes with every macro-scale model iteration and is highly dependent on the state of the surrogate model. For this, we need to analyse the performance of the surrogate model per time step (*g(t)*) as follows:

$$g(t) = \eta_D(t) \,/\, DoF,$$

where *DoF* is the total number of degrees of freedom of the macro-scale model and $\eta_D(t)$ is the number of successful calls to the database per marco-scale time step, which replaces the need to generate micro-scale jobs.

The performance of the surrogate model will vary the number of micro-scale models needed for each macro-scale iteration significantly, which can lead to load imbalance and low utilisation of the available resources. Thus, the main target of the runtime optimisation part of MCP and Algorithms software is to schedule this dynamically varying load of micro-scale models in an optimal way on the available resources.

The value of g can vary for cases where the user is building the surrogate from scratch ($g \to 0$), where the surrogate model is replacing the micro-scale models efficiently ($g \to 1$) or for in-between scenarios. To deal with the distribution of the number of processors in all these cases, we define

three different phases in HMC and the corresponding processor distribution mechanism is shown in Algorithm 1.

Algorithm 1: HMC phases

```
1:  procedure HMC(g(t), DoF, P_min , P_max , P_μ )
2:      η(t) = DoF (1 − g(t))
3:      if η(t)P_min > P_μ then                    . Phase 1
4:          run(η(t), P_min )
5:      else if η(t)P_min < P_μ < η(t) P_max then   . Phase 2
6:          run(η(t), P_μ / η(t) )
7:      else                                       . Phase 3
8:          run(η(t), P_max )
9:      end if
10: end procedure
```

The important elements in Algorithm 1 are the number of processors reserved for all micro-scale models P_μ, and $\eta(t)$, the number of micro-scale models in time step t. If $P_\mu < P_{min}\,\eta(t)$, then the most appropriate action to take is to do farming by running each micro-scale model with a minimal number of processors P_{min}. On the other hand, if $P_\mu < P_{max}\,\eta(t)$, then running with P_{max} is the most suitable choice. Otherwise, for $P_{min}\,\eta(t) < P_\mu < P_{max}\,\eta(t)$, all micro-scale models must be run on $\lceil P_\mu/\eta(t) \rceil$, limiting it to not more than P_{max}, which is the maximum number of processes that the model can benefit from before the performance decreases. Also, if the performance values of running the micro-scale models using different parameters are available or can be estimated, as shown in HMM_materials, the number of processors per micro-scale model can be changed accordingly in the second phase. We will apply this concept to a case for the surrogate model starting from scratch.

Figure 7 shows the performance of a surrogate model (top), expressed by g(t) and the corresponding number of micro-scale jobs (bottom) of the surrogate model under investigation. In this example DoF = 48818, P_{min} = 1 and P_{max} = 16. The colours show the three phases, as introduced above. Phase one, where the farming of jobs is done using P_{min} processors per micro-scale model, is represented in green. Phase two is shown in blue and the third and final phase is shown in red. The dashed lines in the figures are the micro-scale model iterations. Figure 7 shows a surrogate model with good performance. This performance figure is based on results from a simulation in which this "surrogate model" was actually implemented [10] with modification at the first few macro-scale iterations to mimic the case of a new surrogate model.

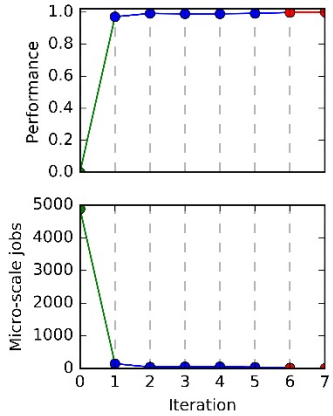

Figure 7: Performance of a surrogate model generated from scratch. Top graph shows the performance values of the surrogate model and lower graph shows the corresponding numbers of micro-scale jobs. The colours refer to the phases, where green is the first phase.

As shown in Figure 7, the first phase of the new surrogate model requires more jobs at the beginning to build the surrogate model. This phase also takes more macro-scale iterations to complete. Note that the number of micro-scale jobs per iteration is calculated as $\eta(t) = DoF(1 - g(t))$. However, in the first phase we do not run the total number of 48818 jobs, but we run batches from which we can then train the surrogate model. In the second phase, the number of micro-scale jobs is less than the number of micro-scale jobs in the first phase. Knowing that it is not beneficial to run a micro-scale job utilising more than P_{max} processors, and the time to run a micro-scale job varies with the number of processors and the input parameters, we might end up having a number of idle processors. We can use these free processors in the second and third phases to further explore the parameter space of the micro-scale model for better performance of the surrogate model, or even change the number of processors per micro-scale model based on different input parameters for each micro-scale model. In the third phase, it is preferable to release unused processors back to the system, to save on the budget. Generally, switching between phases will be totally dynamic and the runtime part of the MCP and Algorithms software should handle this process, as will be illustrated in the next sections. We also note that it might even happen that during the runtime of the HMM we switch back and forth between the different phases, completely depending on how we traverse the parameter space of the microscale jobs. However, the hope is that, by using advanced surrogate modelling techniques and proactively computing in unexplored parts of parameter space, we can keep training the surrogate model to perform very well. This however remains to be investigated, and was not part of the work done in ComPat.

Our proposed solution is to use the resources dynamically supported by the architecture, with the assistance of tailored scheduling controlled by the pattern. For the architecture, we will use the pilot job mechanism.

We executed our experiments on Eagle [16], a supercomputer at the Poznan Supercomputing and Networking Center (PSNC) in Poznan, Poland. The QCG pilot job used in this

work was a normal job submitted to Eagle to reserve a set of resources. After reservation, these resources were then managed using a python script, the Pilot job manager. In this script the user can launch, request and kill jobs dynamically. In our benchmarks, the Pilot job reserves a number of processors first. Then, in the Pilot job manager, we submit the macro-scale model and the HMC manager. The macro-scale, after an iteration, requests a number of parameters to the surrogate model. The surrogate model looks in the database, interpolates the missing quantity and requests to run a number of micro-scale models for the missing quantities. The number of micro-scale models and the available resources are then sent to the HMC manager to suggest the right distribution of the resources to the Pilot job manager. In addition, it acts to different phases of the HMM application accordingly. The pilot job manager then executes the submodels utilising an internal queue on the required resources, gathers the values from the micro-scale jobs and sends them back to the surrogate model. For more information on the QCG pilot job, we refer to D5.3.

Preliminary results – RBC application

We measured the runtime and utilisation for the performance shown in Figure 7. Resource utilisation U is defined as U = R / C, where R is the actual used number of processors and C is the capacity, which is the total number of processors available for the job. The utilisation was measured as a function of wall clock time during the execution.

In this experiment, the number of processors allocated for the macro-scale model was $P_M = 8$, for the surrogate model $P_D = 1$ and for the HMC manager $P_H = 1$. The total number of processors available for the micro-scale models was $P_\mu = 206$. The degree of freedom selected, for simplicity, was DoF = 48818.

Figure 8 (top panel) presents the utilisation of the system, using our strategy, for the surrogate model presented in Figure 7.

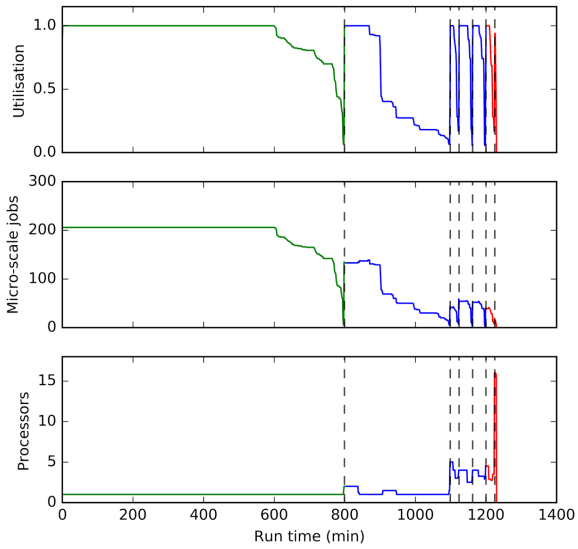

Figure 8: Utilisation of the system (top panel) by using the surrogate model, as shown in Fig 4. The corresponding number of micro-scale jobs (middle panel) and number of processors per micro-scale job (lower panel) were decided by the HMC manager.

In Figure 8 (top panel), in the first macro-scale iteration, a large number of micro-scale jobs are executed, each executing with $P_{min} = 1$ in a first-in, first-out queue (in our method, we use sub-queues in the pilot job rather than in batches). The utilisation in this phase is high because we simply exploit all available resources. At the beginning, the utilisation is one, which means that all 206 processors are used as shown in Figure 8 (middle panel). After a while, ~ 18-32 jobs are completed, shown as the first few small decreases of the green line.

The blue line in Figure 8 (top panel), for the second to fifth macro-scale iterations, shows $P_{min}\ \eta(t) < P_{\mu} < P_{max}\ \eta(t)$ phase. The utilisation at the beginning of these iterations is one, because all the processors reserved for micro-scale models are used by running the micro-scale models with $\eta(t)$, $\lceil P_{\mu} / \eta(t) \rceil$. For example, in the second macro-scale iteration (which lasted from 800 to 1100 minutes of the runtime), 142 jobs were run with ~ 2-1 processors, each shown in Figure 8 the middle and lower panels, respectively. As a result of different micro-scale model execution times with different numbers of processors per micro-scale model invocation, we notice a gradual decrease in the utilisation. In this situation, an internal mechanism could be implemented to use the available processors to refine the surrogate model by proactively (so not informed by the macro-scale model) executing micro-scale models in yet unexplored regions of the micro-scale input parameter space. The gradual decrease in the second macro-scale iteration does not occur in the next iterations (macro-scale iterations 3 to 5), because the number of micro-scale model jobs in iteration 4, for example, is smaller (54 running with ~ 4 processors shown in blue in Fig. 8 (middle and lower panels)).

The last macro-scale iteration in Figure 8 (top panel) falls in the third phase, where the number of micro-scale jobs is only 12, and the number of processors per micro-scale job run is P_{max} = 16. As is shown clearly by the red line in the graph and also the red lines in Figure 8 (middle panel), we run all the micro-scale models in one fast run. This phase is fast, but the utilisation remains low. In this state, as discussed, we could release the unused processors, as the surrogate is mature enough to replace the need to generate new micro-scale jobs.

Preliminary results -- HMM-Material application

In this application, the Optimisation part investigate the main issue related to the execution of the HMM-Material. In this application, the number of micro-scale models is static, because of the lack of surrogate model. However, the variable execution time of the single micro-scale jobs would case a load imbalance. The left panel of Figure 9 shows the strong scaling of one strains tensor setup and the right panel is the execution time of the micro-scale models with different strains tensors normalisation. For more information on the HMM-Material application, we refer to D3.1.

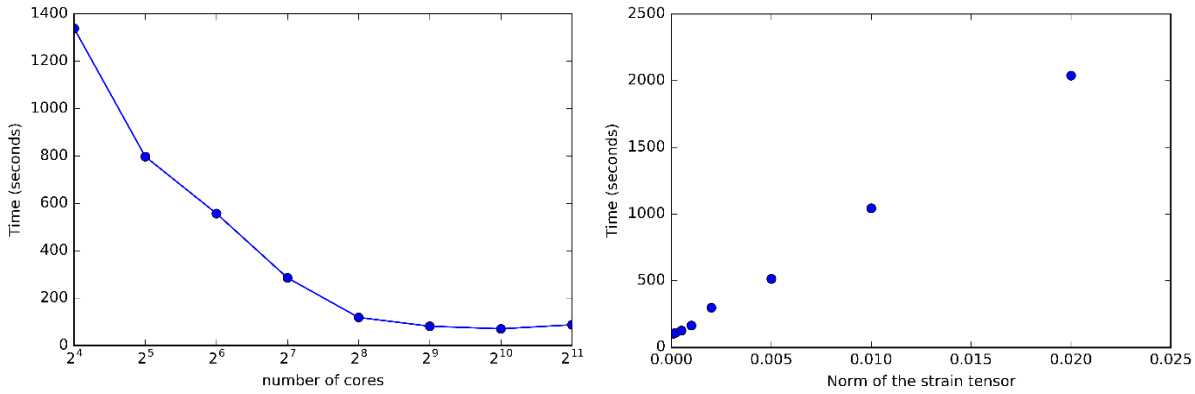

Figure 9: The execution time of micro-scale md simulations using the same configuration on different number of cores (left panel), and on the same number of cores with different stain tensor configurations (right panel).

The micro-scale jobs are executed in the order they figure in the input file until the global resource allocation given to the PilotJob is filled, the following jobs are set in a single queue. In the Optimisation part, we developed an optimisation routine for the execution of the list of micro-scale jobs with the QCG PilotJob system: time-to-completion and efficiency are the two potential optimised criteria. The optimisation is performed controlling the combination of resources allocated to each micro-scale job and the order it appears in the micro-scale jobs list.

Resources allocation for individual micro-scale jobs are determined differently depending on the optimised criteria. Although in both situations, the allocated resource is determined using performance data of the micro-scale scale model, and more specifically the speedup. In the efficiency approach, the resources provided to the micro-scale jobs correspond to the limit of linear scaling. In the

time-to-completion approach, conversely, the resources allocated correspond to the point where the speedup becomes null with an increase of resources. The speedup data of the micro-scale model is extracted from the database and fitted to Amdahl's law, as a function of the resource size. In both cases, the fitted curve is used to determine the desired allocation. A remaining issue, specific to the HMC, is that such optimisation is performed at every iteration of the macroscale model, hence at runtime. The database, which is located on SuperMUC, is fetched from anywhere in the EEE to predict the performance of the micro-scale jobs. Then, the micro-scale jobs are placed in the list in decreasing order of allocated resources. Making using of the 'tetris-like' approach of the PilotJob to fill its available resources, the total allocation is optimally employed. The routine finally dumps the micro-scale jobs list, containing execution order and allocated resources, which in turn is pared by the QCG PilotJob system for execution. Figure 10 shows the utilisation of the resources for the material application using our optimisation strategy for five macro-scale iterations (separated with dashed lines).

The top part of Figure 10 shows the utilisation of the resources, all the resources are continuously used until no more jobs are to be submitted, and then utilisation decreases just before the end of the iteration. The main observation is that the utilisations are consistent at every iteration, which shows that the optimisation method works quite well systematically. In the first iteration, for example, the resources are not entirely used only 15% of the time (i.e after 300 seconds).

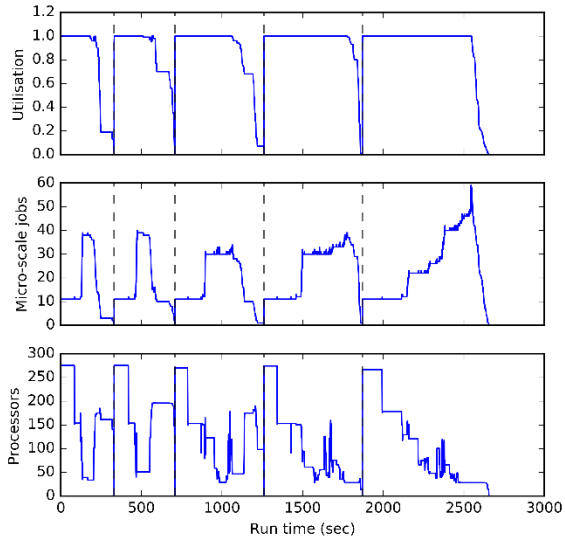

Figure 10: Utilisation of the system (top panel). The corresponding number of micro-scale jobs (middle panel) and number of processors per micro-scale job (lower panel).

At the beginning, the bigger micro-scale jobs are submitted (on average 250 cores per micro-scale jobs), which lasts for 100 seconds (see the bottom part of Figure 10). Then, the size of the submitted micro-

jobs almost continuously decreases, with an always increasing number of micro-jobs being executed simultaneously (see the middle part of Figure 10).

It is interesting to note that some large micro-scale jobs can not be submitted at some point due to lack of resources and are only executed in between 100 and 180 seconds (see the bottom part of Figure 10 for the first iteration for example). Instead of waiting for the availability of these resources to submit them, and leaving some resources idle, the PJM fills these resources with smaller micro-scale jobs, thus the entirety of the resources remain used.

3 Conclusions

We have described in some detail the activity in WP2 in M24 - M36 of the ComPat project, showing the last developments on the three parts (Description, Optimisation and Execution) of the Multiscale Computing Patterns and Algorithms software. The developments were mainly on

- Developing performance database measurements,
- Optimising execution plan selections to predict the performance of the multiscale application based on measurements of the single scale components,
- Automating the benchmark process, to run on all EEE HPC resources, to populate the database and make the performance data available when requested.
- Use the profiled energy consumption as a criterion in the MCP and Algorithms software

We showed the mechanism of running HMM applications using MCP and Algorithms software. We proposed that the execution of the micro-scale models in the HMM application should be viewed as three distinct phases. Then showed the utilisation resulted from applying different number of cores for different phase. We also illustrated the on-fly optimisation on another example (HMM case), where there it is running only on one phase.

4 References

[1] S. Alowayyed, D. Groen, P. V. Coveney, and A. G. Hoekstra, “Multiscale computing in the exascale era,” Journal of Computational Science, vol. 22, pp. 15–25, 2017.

[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: Mathematical, Physical and Engineering Sciences, vol. 372, p. 20130407, 2014.

[3] A. Abdulle, E. Weinan, B. Engquist, and E. Vanden-Eijnden, “The heterogeneous multiscale method,” Acta Numerica, vol. 21, pp. 1–87, 2012.

[4] W. E, B. Engquist, and Z. Huang, “Heterogeneous multiscale method: A general methodology for multiscale modeling,” Phys. Rev. B, vol. 67, no. 9, p. 92101, 2003.

[5] A. Nikishova, L. Veen, P. Zun, and A. Hoekstra, “Uncertainty Quantification of a multiscale model for In-Stent Restenosis,” Cardiovascular Engineering and Technology, 2018.

[6] L.-T. Cheng and E. Weinan, “The Heterogeneous Multi-Scale Method for Interface Dynamics,” Contemporary mathematics., vol. 330, pp. 43–54, 2003.

[7] J. Knap, C. E. Spear, O. Borodin, and K. W. Leiter, “Advancing a distributed multi-scale computing framework for large-scale high-throughput discovery in materials science,” Nanotechnology, vol. 26, no. 43, p. 434004, 2015.

[8] E. Lorenz and A. Hoekstra, “Heterogeneous multiscale simulations of suspension flow,” Multiscale Model. Simul. Multiscale Modeling and Simulation, vol. 9, no. 4, pp. 1301–1326, 2011.

[9] C. Wang, Q. Duan, W. Gong, A. Ye, Z. Di, and C. Miao, “An evaluation of adaptive surrogate modeling based optimization with two benchmark problems,” ENSO Environmental Modelling and Software, vol. 60, pp. 167–179, 2014.

[10] K. W. Leiter, B. C. Barnes, R. Becker, and J. Knap, “Accelerated scale-bridging through adaptive surrogate model evaluation,” Journal of Computational Science Journal of Computational Science, vol. 27, pp. 91–106, 2018.

[11] M. Turilli, M. Santcroos, and S. Jha, “A comprehensive perspective on pilot-job systems,” ACM Comput. Surv., vol. 51, pp. 43:1–43:32, Apr. 2018.

[12] 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, vol. 227, pp. 4895–4911, May 2008.

[13] G. Zavodszky, B. van Rooij, V. Azizi, S. Alowayyed, and A. Hoekstra, “Hemocell: a high-performance microscopic cellular library,” Procedia Computer Science, vol. 108, pp. 159–165, 2017.

[14] G. Závodszky, B. van Rooij, V. Azizi, and A. G. Hoekstra, “Cellular Level In-silico Modeling of Blood Rheology with An Improved Material Model for Red Blood Cells,” Frontiers in physiology, vol. 8, 2017.

[15] A. J. Wagner and I. Pagonabarraga, “Lees–edwards boundary conditions for lattice boltzmann,” Journal of Statistical Physics, vol. 107, pp. 521–537, Apr 2002.

[16] www.wiki.man.poznan.pl/hpc/index.php/Eagle.

[17] O. O. Luk, O. Hoenen, O. Perks, K. Brabazon, T. Piontek, P. Kopta, B. Bosak, A. Bottino, B. D. Scott and D. P. Coster, “Application of the Extreme Scaling Computing Pattern on Multiscale Fusion Plasma Modelling,” Philosophical Transactions of the Royal Society A. -- submitted.

[18] S. Alowayyed, et. al, “Patterns for High Performance Multiscale Computing,” Future Generation Computer Systems. Accepted, 2018.