DOI: 10.14529/jsfi190402

Towards Heterogeneous Multi-scale Computing on Large Scale Parallel Supercomputers

Saad Alowayyed[1,2], *Maxime Vassaux*[3], *Ben Czaja*[2], *Peter V. Coveney*[2,3], *Alfons G. Hoekstra*[2]

New applications that can exploit emerging exascale computing resources efficiently, while providing meaningful scientific results, are eagerly anticipated. Multi-scale models, especially multi-scale applications, will assuredly run at the exascale. We have established that a class of multi-scale applications implementing the heterogeneous multi-scale model follows, a heterogeneous multi-scale computing (HMC) pattern, which typically features a macroscopic model synchronising numerous independent microscopic model simulations. Consequently, communication between microscopic simulations is limited. Furthermore, a surrogate model can often be introduced between macro-scale and micro-scale models to interpolate required data from previously computed micro-scale simulations, thereby substantially reducing the number of micro-scale simulations. Nonetheless, HMC applications, though versatile, remain constrained by load balancing issues. We discuss two main issues: the *a priori* unknown and variable execution time of microscopic simulations, and the dynamic number of micro-scale simulations required. We tackle execution time variability using a pilot job mechanism to handle internal queuing and multiple sub-model execution on large-scale supercomputers, together with a data-informed execution time prediction model. To dynamically select the number of micro-scale simulations, the HMC pattern automatically detects and identifies three surrogate model phases that help control the available and used core amount. After relevant phase detection and micro-scale simulation scheduling, any idle cores can be used for surrogate model update or for processor release back to the system. We demonstrate HMC performance by testing it on two representative multi-scale applications. We conclude that, considering the subtle interplay between the macroscale model, surrogate models and micro-scale simulations, HMC provides a promising path towards exascale for many multi-scale applications.

Keywords: multi-scale modelling, surrogate model, computational science, heterogeneous multi-scale computing, high performance computing, exascale.

Introduction

Science has the ability to describe phenomena and often to predict their occurrence and outcome in an accurate and rapid manner. These phenomena naturally, and computationally, are multi-scale both in time and space [4, 13, 15, 16, 18, 28, 32]. Multi-scale computing [3, 5, 10, 11, 14, 17–19] depends on scale separation and invokes set of single-scale models, each representing a process in time and space, coupled together to describe a phenomenon ranging over temporal and spatial scales. From a computational point of view, the single-scale models, which by themselves could be massively parallel simulations, could run independently of each other, which provide new opportunities in terms of scalability and performance tuning for the emerging exascale [5, 17]. By investigating a broad range of multi-scale applications from many different domains, we have identified a set of generic patterns that are common to these applications. These patterns can be seen as a "high-level call sequences that exploit the functional decomposition of multi-scale models in terms of single scale models" [5].We believe

[1] King Abdulaziz City for Science and Technology (KACST), Riyadh, Saudi Arabia
[2] Computational Science Lab, Institute for Informatics, Faculty of Science, University of Amsterdam, The Netherlands
[3] Centre for Computational Science, University College London, United Kingdom

there are three generic multi-scale computing patterns to be most relevant for high performance multi-scale computing, namely extreme scaling (ES), replica computing (RC) and heterogeneous multi-scale computing (HMC) [5].

The extreme scaling pattern is where one sub-model (the primary model) dominates the computation, while the others are auxiliary models. The main target is to reduce communication between them and prevent serialisation due to auxiliary models. The replica computing pattern represents a set of multi-scale applications where a large number of single-scale simulations are combined to obtain statically robust outcomes. Distributing these replicas on different supercomputers could lead to an increase in overall performance. For further discussions on these two patterns we refer to [3–5, 25].

The heterogeneous multi-scale computing pattern is based on, and inspired by, the heterogeneous multi-scale method [33]. In a heterogeneous multi-scale method, 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 is coupled with a large and typically dynamic number of micro-scale models [4, 30] (Fig. 1).

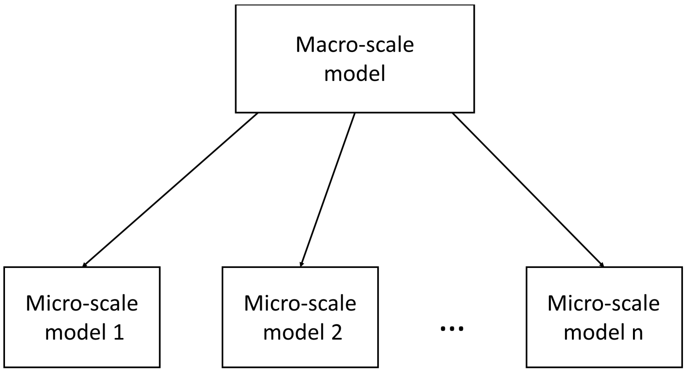


Figure 1. Computational structure of hierarchical multi-scale applications

The HMC pattern implements a family of multi-scale models which, using the multi-scale modelling and simulation framework (MMSF) terminology (see [5, 9, 10]), are single domain with multiple and dynamic instantiations of the micro-scale dynamics [9]. The heterogeneous multi-scale method [1, 33] represents the most obvious multi-scale model that fits the HMC pattern. Other examples could include uncertainty quantification on extreme scaling applications using the so-called semi-intrusive algorithms [26].

The primary potential and advantage of heterogeneous multi-scale modelling is capturing the dynamics at the macro-scale level by explicitly considering some microscopic details of the problem. Thus, HMM is a modelling approach used to numerically solve single-domain multi-scale problems by coupling multiple micro-scale submodels together with a macro-scale model, and each of the micro-scale simulations solves a macroscopic property concurrently. The overall macro-scale behaviour then emerges when the separate submodels are combined. Micro-scale models thus are employed to resolve each unknown component of the problem separately and return the result to the macro-scale model. Frameworks schema taking into account the computational aspects of HMM, certainly in relation to HPC, are rare [5, 20]. For this reason we propose heterogeneous multi-scale computing as a way to efficiently execute HMM models on state of the art HPC infrastructure.

For the purpose of illustration, consider the case of a flowing suspension where the macro-scale constitutive equations may not be known [23]. A lattice Boltzmann model of local fluid velocity u and an advection–diffusion model of local particle density H, representing the macro-scale models, are coupled with a set of fully resolved 3-D lattice Boltzmann suspension model(s). These micro-scale models compute the local viscosity ν and the diffusivity tensor D from a complete run on each, or many lattice point(s) in the macro-scale model to be used in the next time step [23]. This example will be used as proof of concept and is shown in Fig. 2.

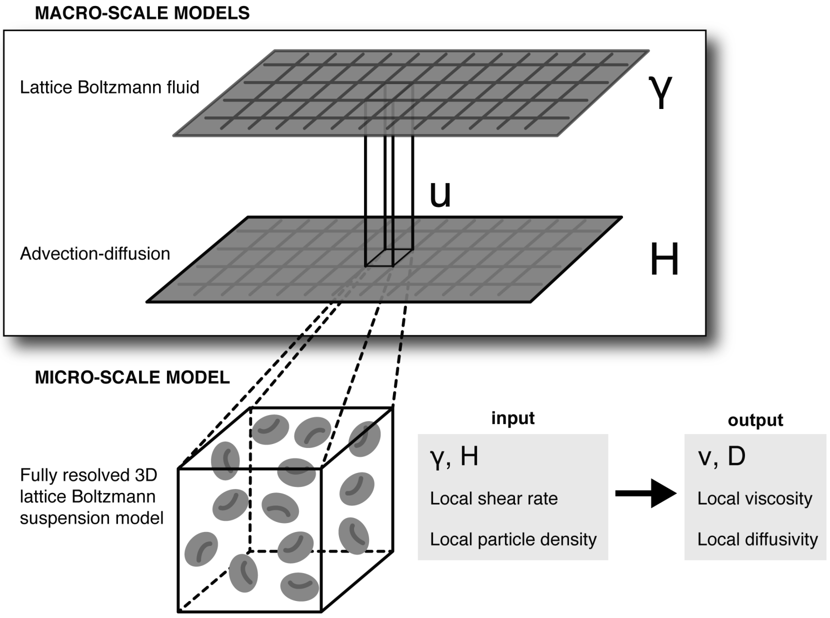


Figure 2. Lattice boltzmann fluid model of local velocity u and advection–diffusion model of local particle density H, representing the macro-scale model, are coupled with a set of fully resolved 3-D lattice boltzmann suspension model(s). These micro-scale models compute the local viscosity ν and the diffusivity tensor D

Load Balancing in HMC

Micro-scale models can be and usually are computationally intensive, such as in the 3-D example considered above. The number of micro-scale models also depends on the spatial properties of the macro-scale model [23]. From a computational point of view, the potentially very large number of micro-scale simulations, the dynamic nature of the amount of required micro-scale models, and their execution time, can become a bottleneck in production runs [5]. This leads to challenging large scale dynamic simulations that are far from trivial to efficiently execute in HPC environments.

To deal with the issue of variable number of executing micro-scale models and their variable execution time in a high performance computing environment, we rely on the pilot job mechanism in combination with a layer of dedicated optimisation and scheduling functionality. The pilot job, for example RADICAL-Pilot [29], is a normal job submitted to supercomputers to reserve resources and use them as an integrated whole. The QCG pilot job manager system [21] has a similar mechanism which allows management of the resources on the application level. A 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.

A possible approach to reduce the number of micro-scale simulations is to utilise a surrogate model as an intermediate layer between the scales to prevent re-computing micro-scale simulations for parameters that are already known, and to use the already computed quantities at the micro-scale, stored in a database, to build a surrogate model. The database and surrogate model, in conjunction with an HMC manager, restores previously computed data, interpolates them where necessary using the surrogate model, and provides input to the macro-scale model(s). Depending on the state of the surrogate model and the history of the data required, this mechanism can reduce the number of micro-scale simulations significantly, potentially by orders of magnitude.

Generally, this approach is practical and has already been demonstrated in multiple fields [22, 31]. The exchange of data should not be a bottleneck in this approach, because the exchange between scales usually involves a few floating-point numbers that represent specific properties. However, the number of micro-scale models should be mapped efficiently to available hardware resources to avoid potential load-balancing required at runtime issues [5].

The role of the HMC manager is to build the surrogate model and evaluate it when needed. In this process, at every time-step the macro-scale model sends a request to the HMC manager for required quantities. The manager then consults the surrogate model for the required information. For this purpose, a user-defined script is implemented to decide whether the cached or interpolated data is sufficient. If not, the manager will start new micro-scale models to obtain more accurate results. To prevent the manager itself from becoming the bottleneck, so as we will assume asynchronous I/O and separate computing resources for the HMC manager. A schematic of this process is shown in Fig. 3.

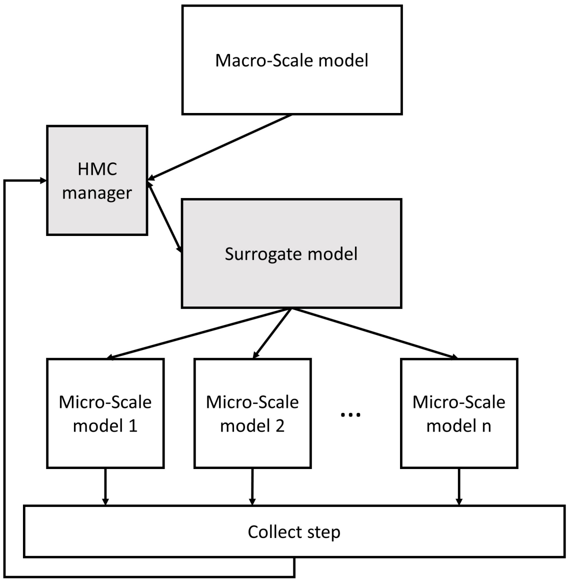


Figure 3. Main workflow of the heterogeneous multi-scale computing and different components (macro, micro, surrogate and HMC manager). The main components are highlighted

Although the main benefit of the dynamic nature of the surrogate model is to decrease the computational cost, it still may cause load imbalance because the number of single-scale models, at each macro-scale iteration, is dynamic [22]. The purpose of the HMC pattern is to mitigate these issues and provide a level of control over the execution that aims to minimise load imbalance and maximise resource utilisation over the complete HMC run.

Our approach is to use the resources dynamically supported by the architecture, with the assistance of tailored scheduling which is controlled by the pattern. In the next section, we will focus on load balancing in micro-scale models in HMC.

1. Methods

A typical iteration of an HMC application consists of simulating one time-step of a macro-model, followed by a large number of simulations of micro-models. The outputs of the micro-model simulations are synchronised before moving to the next iteration of the macro-model. While the single macro-model simulation usually is straightforward to perform, efficient use of resources during simulation of the micro-model requires potentially complex scheduling. A first issue arises from the submission of the micro-model simulations, as their large number makes it impractical to submit them individually to the supercomputer job manager. The relatively small size of each of these jobs would generally grant them a low execution priority in turn, the variable queuing time may cause significant bottlenecks when synchronising the micro-model simulations results.

A simple solution is to batch the micro-model simulations and perform them in a single, large resource allocation. This large allocation can either be requested at the beginning of the multi-scale simulation, or at every iteration of the workflow. The former avoids repeated queuing periods of time, while the later releases the large allocation during the execution of the macro-model simulation. Independently of the chosen method, the micro-model simulations have to be internally scheduled within the requested resource allocation. The Internal scheduling algorithm, set by the user in QCG, is First Come First Serve, while the large allocation can be subdivided into a separated number of sub-allocations of fixed size. An identical number of cores is allocated to each sub-allocation. Naively, before starting the first micro-model simulation, each of these are arbitrarily assigned to a sub-allocation, such that each will perform a similar number of simulations (see Fig. 4a and b).

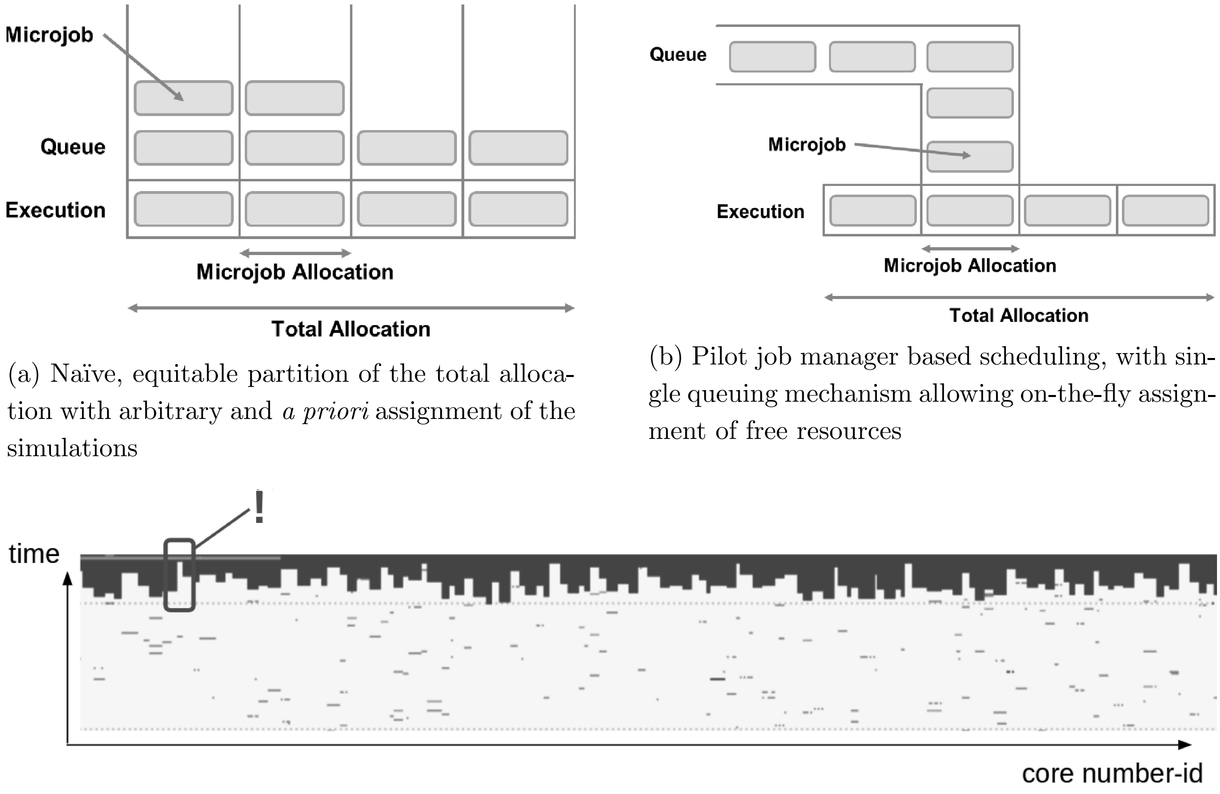
However, this simplistic sub-allocation assignment method can rapidly become inefficient. Even though each sub-allocation is similar in terms of resource size and number of simulations to perform, nothing guarantees that each of the simulations have comparable execution time. Due to the lack of *a priori* knowledge about this execution time, arbitrary assignment can cause some sub-allocations to complete their simulations far quicker than others by, for example performing only short ones. Consequently, a significant load imbalance can result in leaving sub-allocations idle for long periods of time (see Fig. 4c).

1.1. Optimisation of Resources

We propose to address the load balancing issue induced by arbitrary *a priori* resource assignment and variable execution time of the micro-model simulations using two mechanisms: (i) pilot job manager (PJM) internal scheduler, and (ii) optimisation of sub-allocation size.

The PJM mechanism essentially consists of an internal job scheduler for the large allocation provisioned for the whole set of micro-model simulations of a given iteration. The execution order of the jobs is specified in a First In First Out (FIFO) manner. The main advantage of the PJM is that all the jobs are gathered in a single queue, allowing execution on any subset of cores from the large allocation, as soon as the required resource size is available. Moreover, the size of this sub-allocation can be easily specified independently for each job, which helps when size adjustment is needed to reduce execution time disparities.

In order to reduce the variability of the micro-model simulations execution time, the sub-allocation size can be adjusted individually. Acknowledging that micro-model simulations ought to scale strongly up to certain number of cores, this can be achieved without any loss of effi-



(a) Naïve, equitable partition of the total allocation with arbitrary and *a priori* assignment of the simulations

(b) Pilot job manager based scheduling, with single queuing mechanism allowing on-the-fly assignment of free resources

(c) Consequences of the variable execution time of the micro-model simulations on idle time of resources. In light the portion of actual computation, and in dark, idle time. Few sub-allocations (e.g., Exclamation mark) can keep all other sub-allocation idle for a significant amount of time causing load imbalance

Figure 4. Internal scheduling of micro-model simulations inside a large allocation

ciency, as long as the sub-allocation size remains in the strong scaling domain. The upper limit of the strong scaling domain can easily be obtained from benchmarks scaling (see Fig. 8b). Subsequently, the simulations allocation size can be re-scaled proportionally to their estimated execution time. The longest simulations are performed with the number of cores associated with upper boundary of the strong scaling domain, while the shortest are performed with the lower boundary, that is one core. However, in less variable micro-model simulations execution time, in job farming it is always better to run with the lower bound as shown next.

Predicting execution time can be done in various ways which either rely on benchmarking data or a reference model function of input parameters. The former can be queried directly to obtain an estimation of an execution time (see Fig. 8a).

1.2. Incorporating Surrogate Model

Now we want to consolidate the multi-scale model with a surrogate model, let us assume that we will utilise a pilot job running on P_{total} processors for a time T, where P_{total} and T are decided by the user at launch time. Let the macro-scale model (M) execute on P_M processors, the surrogate model (S) execute on P_S processors, the HMC manager (HMC) on P_H processor and η micro-scale models execute on P_μ processors. For the macro-scale model, the HMC manager and the surrogate model, the processor allocation is static while for the micro-scale models it is dynamic.

Ultimately, the number of micro-scale models η to be executed is dynamic and changes at every macro-scale time step, depending on how effective the surrogate model is in predicting the missing quantities. Thus, $\eta(t)$ is the number of micro-scale models at time t. Its range is $(0 \leq \eta \leq DoF)$, where DoF is the total number of degrees of freedom of the macro-scale model that are coupled to the micro-scale simulations. This could, for example, be the number of lattice points in a flow simulation where, in each lattice point, the viscosity is unknown and needs to be obtained from a micro-scale simulation.

The number of successful calls to the surrogate model per time step, which replaces the need to generate micro-scale jobs, is $(\eta_D(t))$. This will reduce the number of micro-scale models to

$$\eta(t) = DoF - \eta_D(t). \tag{1}$$

We define $g(t)$ as the performance of the surrogate model,

$$g(t) = \frac{\eta_D(t)}{DoF}, \tag{2}$$

and then write the actual number of micro-scale models that need to be executed at a macro-scale time step as

$$\eta(t) = DoF(1 - g(t)). \tag{3}$$

To complete the process, the number of processors allocated to run *one* micro-scale model i, $P_{\mu i}$, is determined at *runtime* by the HMC manager. These numbers are stored in an array $(P_\mu[])$ to represent the number of processors to run the $\eta(t)$ micro-scale model(s) for one macro-scale iteration t. Also, the computing resources for the HMC manager P_H are determined separately.

At launch time, the HMC pattern software obtains the number of processors for each main component (i.e. P_M, P_S, P_μ and P_H) and the macro-model degrees of freedom (DoF) from the user. Also, the initial performance measurements of the micro-scale model are benchmarked to retrieve the minimum $P_{\min}$ and maximum $P_{\max}$ number of processors to run *one* micro-scale model. $P_{\min}$ is determined by memory requirements and $P_{\max}$ by strong scaling behaviour where $P_{\max}$ denotes the number of processors required, where the execution time is minimised.

Depending on the budget of the user, the requested time to completion and the expected number of micro-scale models, the user would estimate the resources to be used and the time required for execution. Next, the variables used during runtime, namely the number of processors per micro-scale model $P_{\mu i}$, must be determined as discussed previously.

Generally, the total number of processors for micro-scale models P_μ can be used either in *stateful* or *stateless* mode. If the micro-scale models are stateful, then the micro-scale models reside in memory and the HMC manager feeds the micro-scale models with appropriate input and initial conditions. Although this method is easy to implement, it assumes that all micro-scale models will be the same, requiring the same time and computational power, which is a main source of a load-imbalance situation.

In the *stateless* mode, the number of processors per micro-scale model $P_{\mu i}$ is totally dynamic. In this case, the pattern software decides how to utilise P_μ for each macro-scale iteration. The main advantage of this method is that when the computation of the micro-scale models is complete, their nodes/cores can be used for pre-calculating some of the database properties of the surrogate model, or simply release them to the system.

The distribution of the number of nodes/cores per micro-scale model depends on the number of micro-scale models requested and the number of available processors for these micro-scale models. In the following subsections, we will discuss a mathematical model for the dynamic number of micro-scale models, and we will provide an example of the performance of the surrogate model.

1.2.1. Execution time model

In this section, we discuss optimal manners in which to run the micro-scale jobs. First, the computing time for one and for multiple micro-scale models is analysed, then the performance of the surrogate model $g(t)$ per time step during the simulation is evaluated.

The computing time to run *one* micro-scale model using $P_{\mu i}$ processors can be calculated utilising the concept of fractional overhead [5, 7], with

$$T^{\mu i}(P_{\mu i}) = \frac{T^{\mu i}(1)}{P_{\mu i}} + To(P_{\mu i}) = \frac{T^{\mu i}(1)}{P_{\mu i}}(1 + fo(P_{\mu i})), \tag{4}$$

where $T^{\mu i}(1)$ is the time to run *one* micro-scale model using a single processor and $To(P_{\mu i})$ is the overhead time for running *one* micro-scale model using $P_{\mu i}$ processors. The fractional overhead fo for running *one* micro-scale model using $P_{\mu i}$ processors can be expressed as

$$fo(P_{\mu i}) = \begin{cases} 0, & \text{if } P_{\mu i} = 1 \\ P_{\mu i} To(P_{\mu i})/T^{\mu i}(1) > 0, & \text{otherwise.} \end{cases}. \tag{5}$$

The total time T to run *each* micro-scale model on the same number of $P_{\mu i}$ processors (so, independent of i) using a total of P_μ processors for all micro-scale models will be

$$T = \frac{\eta(t)}{\lceil {}^{P_\mu}/{}_{P_{\mu i}} \rceil} \left(\frac{T^{\mu i}(1)}{P_{\mu i}} \big(1 + fo(P_{\mu i})\big) \right) \sim \frac{\eta(t)}{P_\mu} \big(T^{\mu i}(1)(1 + fo(P_{\mu i}))\big), \tag{6}$$

where $\eta(t)$ is the number of micro-scale models in time step t. The target is to minimise T, then

$$P_{\mu i} = 1; fo(1) = 0, \tag{7}$$

$$T = \frac{\eta(t) T^{\mu i}(1)}{P_\mu}. \tag{8}$$

Otherwise,

$$T = \frac{\eta(t) T^{\mu i}(1)}{P_\mu}(1 + fo(P_{\mu i})), \tag{9}$$

which means that the fewer processors we use per micro-scale model, the lower will be the overhead. One should note the difference with Section 1.1, where the surrogate model is not used. In the second step we assume that $T^m u_i(1)$ is equal for all i, so very little or no variability in the runtime for the microscale simulations comparing to the surrogate model.

1.2.2. Surrogate model performance

The number of micro-scale models $\eta(t)$ 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 ($g(t)$). The value of g can vary for cases where the user is building the surrogate from scratch ($g \longrightarrow 0$), to cases where the surrogate model replaces the micro-scale models efficiently ($g \longrightarrow 1$), or for in-between situations. To deal with the distribution of the number of nodes/cores in these cases, we defined three different phases, and the corresponding processor distribution mechanism is shown in Alg. 1.

The important elements in Alg. 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 perform 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 no more than P_{max}, which is the maximum number of processes that the model can benefit from before performance decreases. Also, if the performance values of running the micro-scale models using different parameters are available or can be estimated, the number of processors per micro-scale model can be changed accordingly in the second phase. We will apply this concept to two different cases for the surrogate model, starting from scratch and from a developed surrogate model.

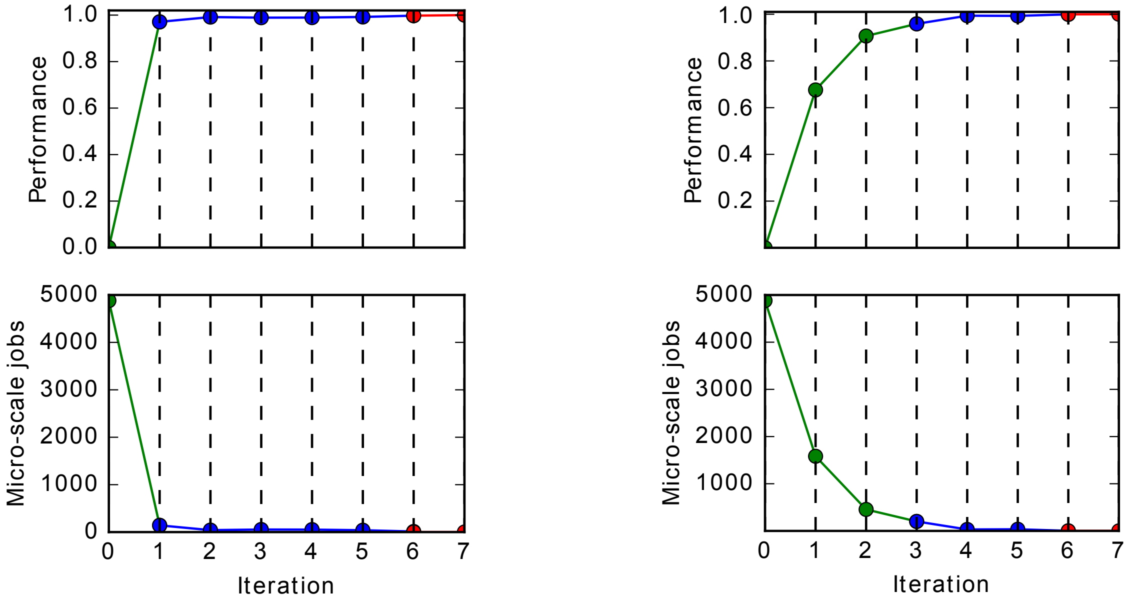
Algorithm 1 HMC phases

```
procedure HMC($g(t), DoF, P_{min}, P_{max}, P_\mu$)
    $\eta(t) = DoF(1 - g(t))$
    if $\eta(t)P_{min} > P_\mu$ then                                  ▷ Phase 1
        $run(\eta(t), P_{min})$
    else if $\eta(t)P_{min} < P_\mu < \eta(t)P_{max}$ then            ▷ Phase 2
        $run(\eta(t), \lceil P_\mu/\eta(t) \rceil)$
    else                                                              ▷ Phase 3
        $run(\eta(t), P_{max})$
```

Case (a): Surrogate model from scratch

If the user starts building the surrogate from scratch, i.e. for $g \longrightarrow 0$, then it is meaningless to run DoF micro-scale models to fill the database at the beginning. This will be inefficient, since the degree of freedom can easily reach 10^6 or more. What can be done is cluster input parameters into an input subsample. In this case, the initial batch of micro-scale models (η_{init}), from which a surrogate model is built, is first executed, then run the next batch (either look it up in the database or run the micro-scale model), and so on.

Figure 5 shows the performance of a surrogate model (top graphs), expressed by $g(t)$, and the corresponding number of micro-scale jobs (lower graphs) for two different performance levels of the surrogate model. Both examples had a ($DoF = 48818$). 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 macro-scale model iterations. The first example, Fig. 5a, shows a surrogate model with good performance (model = scratch, good performance), while the second example; Fig. 5b, demonstrates poor performance (model = scratch, poor performance). These performance figures are based on the results from a simulation in which this surrogate model was actually implemented [22] with modification at the first few macro-scale iterations in order to mimic the case of a new surrogate model.



(a)

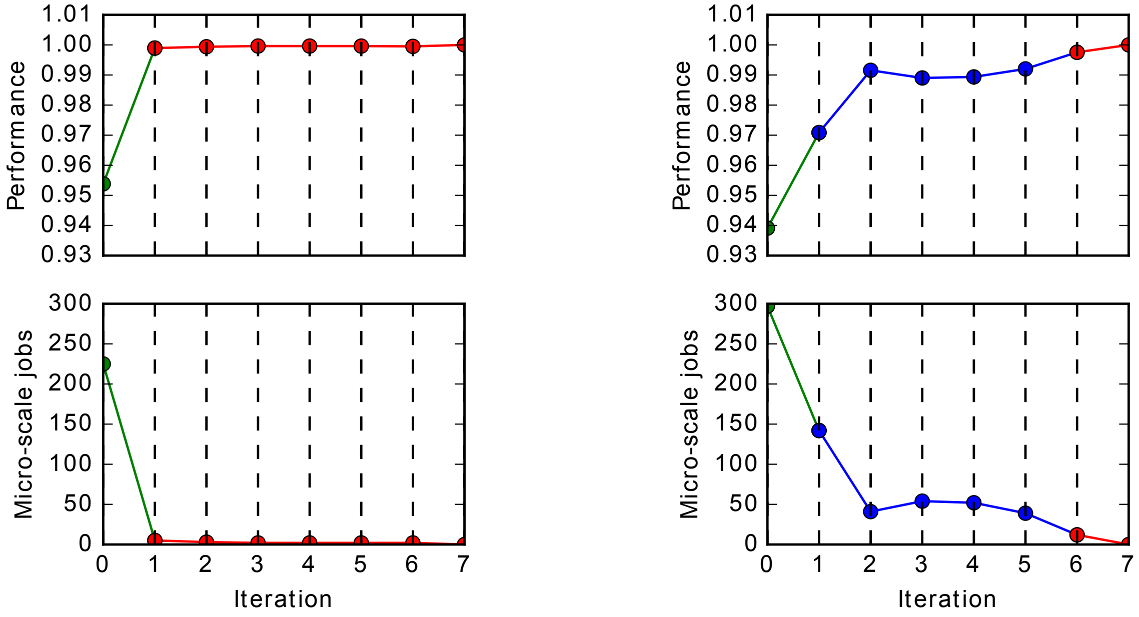
(b)

Figure 5. Behaviour of a surrogate model generated from scratch for two different performance levels. A good performance level is on the left (model = scratch, good performance), a poor performance level is on the right (model = scratch, poor performance). The upper two panels show the performance values of the surrogate model and lower two panels show the corresponding numbers of micro-scale jobs. In this example, $P_{min} = 1$, $P_{max} = 16$ and $P_{\mu} = 206$. The colours refer to the phases, where green is the first phase, blue the second, and red the third. The dashed lines represent the macro-scale model iterations

Case (b): Developed surrogate model

A HMM simulation can also be executed using a previously constructed surrogate model. Figure 6 (top) shows the performance of a well-established surrogate model and the corresponding micro-scale models (Fig. 6 (lower)) for two different performance levels of the surrogate model. As for case (a), both simulations had a $DoF = 48818$. The performance figures are based on the results from a simulation in which this surrogate model was actually implemented [22].

As shown in Fig. 5 and 6, the first phase of the new surrogate model, case (a), 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 degree of freedom (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 have a number of idle cores/nodes. For the two study cases, we can use the free cores/nodes 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 a number of unused processors back to the system to save on the cycle budget. Generally, switching between phases will be totally dynamic and the runtime part of the HMC pattern software should handle this process, as will be illustrated in the next sections.



(a)

(b)

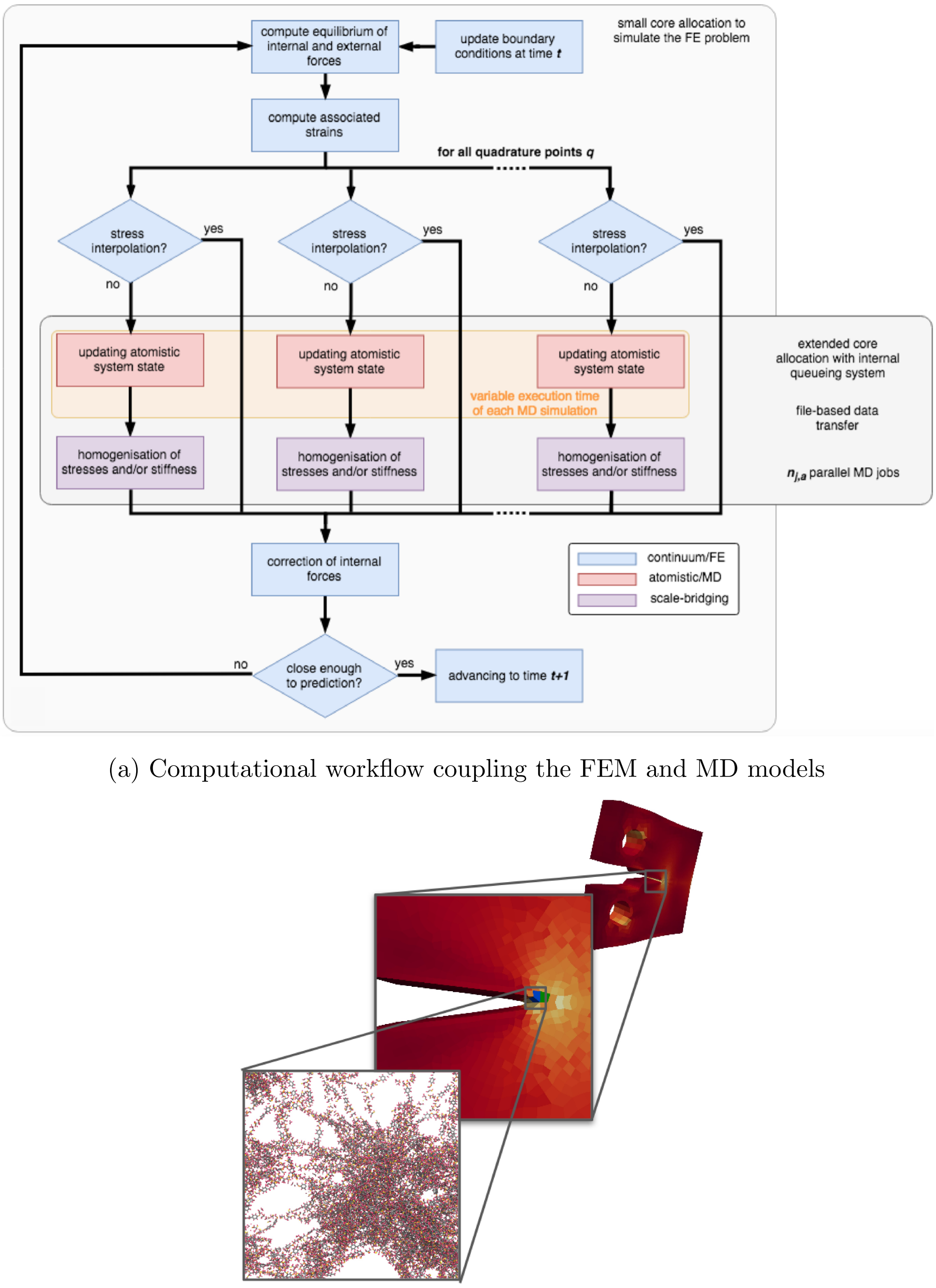
Figure 6. Behaviour of a well-developed surrogate model for two different performance levels. The good performance level is shown on the left (model = developed, good performance), the poor performance level on the right (model = developed, poor performance). Upper panels show the performance values, and lower panels show the corresponding numbers of micro-scale jobs. In this example, $P_{min} = 1$, $P_{max} = 16$ and $P_{\mu} = 206$. The colours refer to the phases, where green is the first phase, blue the second, and red the third. The dashed lines represent the macro-scale model iterations

2. Results

In this section, we will discuss our results for addressing the issues of load balancing both due to variable execution time and to a dynamic number of micro-model using two representative applications, which are a nano-materials system and a red blood cell suspension system, respectively.

2.1. Nano-materials Application

Ab initio physical models are unfeasible beyond the smallest scales (i.e. the nano-scale), although density functional theory and molecular dynamics are the models of choice of material scientists, these models overlook the geometrical and structural complexity of the engineering scale, which induces a heterogeneous conglomeration of local mechanical states. Correctly capturing these is essential to observe the emergence of macroscopic materials properties. Current attempts to explore the properties of a system of atoms when bridging from the atomistic to the continuum scale most often involve ad hoc assumptions [12], in the form of constitutive modeling, most certainly missing out substantial peculiarities of the newly formulated material. Our HMC application [30] computes the dynamic equilibrium of mechanical forces in a continuum structure using the finite element method (FEM). In a classical FEM approach the local constitutive relation between stresses and strain comprises a series of phenomenological mathematical equations, but in the present case it is replaced by a molecular dynamics (MD) simulation (see Fig. 7b). An MD simulation, with nanoscale detail of the material structure, is


compute equilibrium of internal and external forces
update boundary conditions at time *t*
small core allocation to simulate the FE problem
compute associated strains
for all quadrature points *q*
stress interpolation?
yes
no
stress interpolation?
yes
no
stress interpolation?
yes
no
updating atomistic system state
updating atomistic system state
updating atomistic system state
variable execution time of each MD simulation
homogenisation of stresses and/or stiffness
homogenisation of stresses and/or stiffness
homogenisation of stresses and/or stiffness
extended core allocation with internal queueing system
file-based data transfer
$n_{j,a}$ parallel MD jobs
correction of internal forces
continuum/FE
atomistic/MD
scale-bridging
no
close enough to prediction?
yes
advancing to time *t+1*

(a) Computational workflow coupling the FEM and MD models

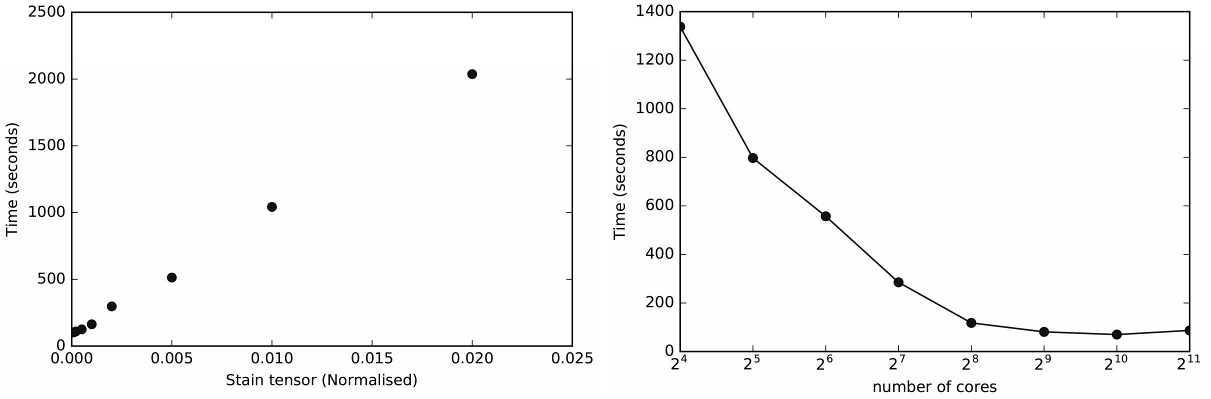
(b) Zoom across the multiple scales of the workflow, from which the continuum description reproduces realistic boundary conditions of a standard compact-tension test for fracture toughness estimation, embedding at atomistic description of the polymer, capturing the chemical specificity of the crosslinked chain network

Figure 7. Prediction of mechanical properties using an HMM workflow

performed whenever the stress resulting from an applied strain history is required. In a nutshell, the application consists of a macroscopic FEM model which synchronises the simulation of a large number of microscopic MD models iteratively as time advances (see Fig. 7a).

In more detail, the macroscopic model feeds a microscopic model with a given strain tensor; the microscopic model evolves following Newton's equation of motion to reach the given strain.

The resulting evolution is dependent on the strain rate applied on the microscopic system. In turn, the execution time of the microscopic simulation is highly correlated with the amplitude of the applied strain (see Fig. 8a). Any heterogeneous macroscopic field at a certain iteration in the macroscopic model will inevitably result in varying execution time of the microscopic model simulations. However, as mentioned in Section 1.1, this variability can be toned down by exploiting the known strong scaling curves of the microscopic model (see Fig. 8b). Note that since the microscopic system evolves out of equilibrium with the applied strain, each finite element cell is necessarily more associated with its own atomistic structure. This makes the use of surrogate modelling difficult with the nano-materials application, but this will be addressed with the red blood cell suspension application in Section 2.2.



(a) Linear dependence of the strain amplitude on the execution time

(b) Strong scaling of the simulation of the model of the 40000 atoms structure strained at 0.1% of epoxy resin

Figure 8. Benchmarking of the molecular dynamics model simulation

The microscopic MD systems describe epoxy resin and graphene nanocomposites, involving approximately 40000 atoms. Each microscale system represents a 10 nm wide cubic box. The microscopic simulations are performed with a constant time step of 2 fs and a constant strain rate of 10^{-4} s^{-1}. The MD simulations are performed using LAMMPS [27] and the ReaxFF force field [2] that captures on-the-fly bonding and debonding of atoms. The macroscopic system models the structure of a compact-tension test following ASTM International Standards [6] (ASTM/E1820). The geometry of the test is specifically designed to trigger the appearance of a single crack in the structure. The dimensions of the compact-tension test specimen are typically of the order of a few centimetres, while the load is applied over a few seconds (see Fig. 7b). Assuming time scale separation, the macroscopic simulations are performed with a much larger time step, that is 0.5 μs, as compared to the microscale simulations. Similarly, assuming space scale separation, the FEM cells are 1 mm wide. The FEM simulations are performed using the Deal.II library [8].

2.1.1. Results

We perform a benchmark of the nano-materials application on the five first time steps of the simulation described in Section 2.1. Internal scheduling of the micro-scale model simulations is executed with three different configurations: (i) a *naïve*, (ii) a *PJM*, and (iii) a *PJM+opt* configuration. The first (*naïve*) corresponds to the configuration described in Fig. 4a, with arbitrary and *a priori* assignment of the simulations. The second (*PJM*) refers to the pilot job manager based scheduling (see Fig. 4b), with single queuing mechanism allowing on-the-fly assignment to free resources. The third (*PJM+opt*) is equivalent to the *PJM* configuration, with the additional optimisation layer compensating for the variability of the micro-scale model simulation execution time. This benchmark is performed for two sizes of allocation: (a) 20 nodes and (b) 100 nodes each computing a total of 28 cores on the Eagle supercomputer[4] in Poznan, Poland. In this test case, the strong scaling limit of the micro-scale model is 4 nodes due to the limited number of particles. In the *naïve* and *PJM*, we assume that the micro-scale simulations are all performed on the same sub-allocation size, hence 4 nodes. From practice and the in-depth knowledge discussed in 1.2.1, in case of large number of replicas, running on one node will reduce the communication and would provide a better performance. In the *PJM+opt* configuration, the sub-allocation is varied from 1 to 4 nodes following the predicted execution time of the micro-scale simulation. The reason for this choice is because one replica takes the whole node, i.e. all the 28 cores.

The total runtime for each of the three internal scheduling configurations with the two sizes of allocation is shown in Fig. 9a, b. Independently of the resource allocation size, we observe an important speedup, up to a 70% runtime reduction, in configurations featuring the flexible PJM mechanism. Nonetheless, the relative speedup in presence of the PJM is reduced on larger allocations to approximately 35%. As the allocation size grows, with a constant amount of micro-scale simulations, internal queuing is reduced as is the predominance of load balance induced by rigid *a priori* plan. Conversely, flexible internal queuing dominates when the total allocation size becomes comparable to the micro-scale simulation allocation, respectively 20 and 4.

A more detailed analysis using the performance reports provided by the PJM, helps us to determine why the optimisation layer actually has an effect on the total runtime with a 100 nodes application. In Fig. 9c, d, the PJM analytics indicate the evolution over the course of the simulation of the utilisation percentage of the resources, the number of concurrent micro-scale simulations, and the instantaneous average of cores per micro-scale simulation. In the first iteration, we can then observe that for the simulation with the *PJM+opt* configuration with 100 nodes (see Fig. 9d), 62 out of 100 nodes are not used due to misleading scheduling decisions and currently working on improving this. In turn, the first iteration is slightly longer with than without the optimisation layer. Nonetheless, when we focus on the last three iterations the *PJM+opt* configuration is actually faster than the *PJM* configuration by 20% due to the size of single-scale model. These three iterations demonstrate the benefit of reducing the execution time variability between micro-scale simulations. Clearly, this gain of performance would be way beneficial when we run the simulation for a large number of iterations. The small peaks in Fig. 9d (middle) represent the small and fast micro-scale models being executed by the end of each macro-scale iteration.

The detailed analytics reported by the PJM also illustrate generally the benefits of having a flexible internal queuing system. Consistently for both configurations shown in Fig. 9c, d, we observe optimal usage of the total allocation up to a certain, late, point in time. This corresponds to the point in time when the queue of simulations is finally emptied. From then onward, resource usage slowly decreases until all simulations are completed.

[4] https://wiki.man.poznan.pl/hpc/index.php/Eagle

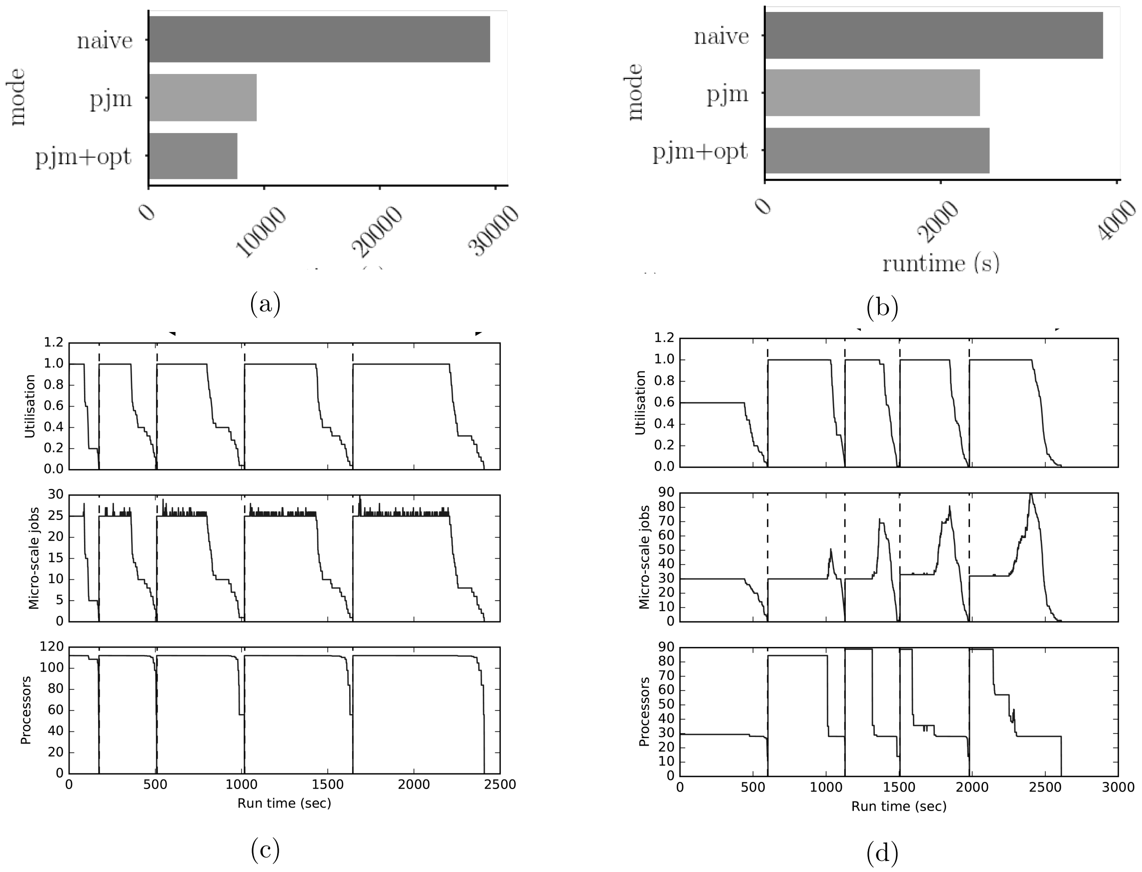


Figure 9. Benchmarking of the full nano-materials with various internal scheduling methods. Comparison between *naïve* (*a priori* scheduling assignment with multiple internal queues), *PJM* (single queue internal scheduler) and *pjm+opt* (single queue and variable resource allocation) configurations. The application is run for 5 iterations of the HMC workflow. (a, b) Influence of the internal scheduling method on the total runtime of the Nano-materials application. Two allocation sizes are tested: 20 nodes (left) and 100 nodes (right). (c, d) Detailed analytics from the PJM for the 100 node allocations with *pjm* (left) and *pjm+opt* (right) configurations only. That is time evolution of total allocation usage (top), number of running microscopic model simulations (middle) and average number of processes per running micro-scale simulation (bottom)

2.2. Red Blood Cell Suspension Application

In this example, we present the benefit of our scheduling method for HMM applications with a surrogate model. Resolving the properties of whole blood on the multiple spatial scales present in the human body is a significant challenge for a single blood-flow model. Whole blood is a suspension of deformable red blood cells (RBCs) and as a result has many non-Newtonian flow properties, for example shear thinning. On spatial scales $\leq 300\mu$m phenomena like the Fåhraeus-Lindqvist effect, the RBC free layer and platelet margination require cell resolved blood flow models which take into account the material properties of the RBCs that influence the emergent rheology of blood. On scales $> 300\mu$m cell resolved models quickly become too expensive and are not viable options, so continuous models are employed. The ultimate goal of developing an HMM blood flow model is to simulate blood on scales where only continuous models have been applied, informed by the cell resolved nature of whole blood. In this section we present a place holder 3D HMM blood flow application to highlight the benefit of our scheduling method.

In this application, blood-flow is modelled on the macro-scale as a continuous fluid using the lattice Boltzmann method (LBM). Also on the macro scale is an advection-diffusion solver which models the evolution of red blood cell volume fraction (i.e. the haematocrit) profiles. The micro scale is modelled with the cell resolved blood flow model HemoCell, in which plasma is modelled by the LBM, the mechanical model of the RBCs are described by a discrete element method and are couple to the plasma via the immersed boundary method [34, 35]. From each local haematocrit and shear rate combination on the macro scale, cell resolved micro scale will simulate perfect sheared environments using Lees-Edwards boundary conditions [24]. Local viscosity and diffusion coefficients will be measured on the micro scale and passed up to the macro scale. The local viscosity will be passed to the LBM fluid solver, and the diffusion coefficient will be passed to the advection-diffusion solver. On the macro scale the LBM fluid solver will take the local viscosities, and the advection-diffusion solver the difusion coefficients, and both will step the macro simulation through the next time step calculating new shear rates and haematocrit profiles for the next iteration. The micro-scale models will resolve blood flow on spatial scales of $\sim 100\mu m$ and temporal scales of $\sim 10ms$, while the macro-scale models will resolve spatial scales of centimeters and temporal scales of seconds. A benefit of such an HMM model is that we will have on the largest scales a continuous blood flow solver which is informed by a micro scale cell resolved blood flow solver. This should lead to a better resolution of the transport of blood cells on the largest vessels found in the body. To avoid duplicating shear rate and local haematocrit microscale simulations, we aim to build a surrogate model (e.g., based on a Gaussian process) to conduct interpolations for similar parameters. This process decreases the required number of micro-scale models requested, which in turn decreases computation time. A schematic of this HMM blood flow model is shown in Fig. 10.

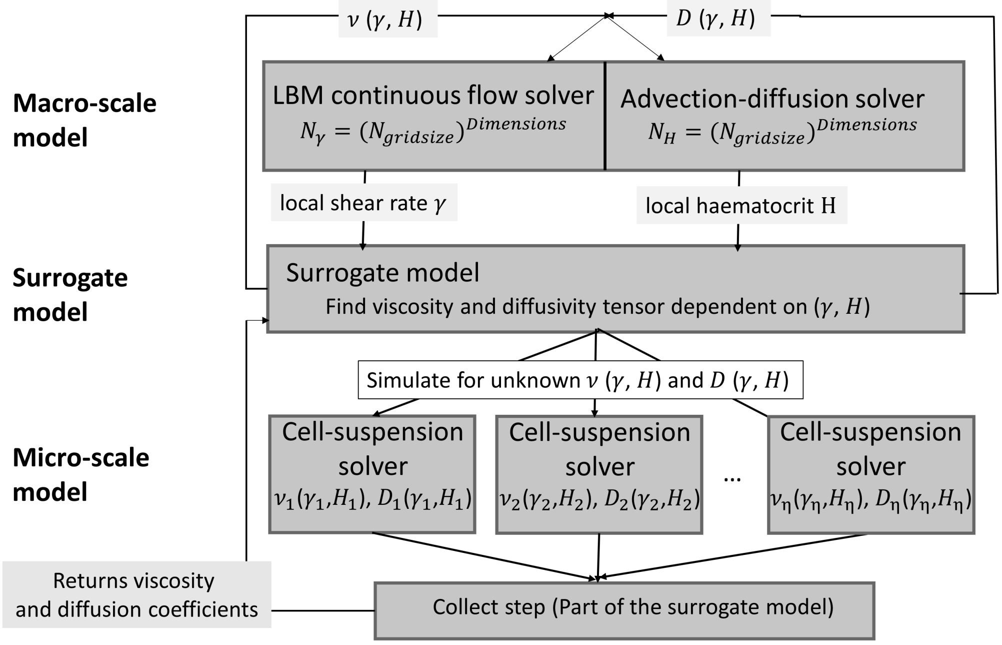

$\nu\,(\gamma, H)$
$D\,(\gamma, H)$
Macro-scale model
LBM continuous flow solver
$N_\gamma = (N_{gridsize})^{Dimensions}$
Advection-diffusion solver
$N_H = (N_{gridsize})^{Dimensions}$
local shear rate γ
local haematocrit H
Surrogate model
Surrogate model
Find viscosity and diffusivity tensor dependent on (γ, H)
Simulate for unknown $\nu\,(\gamma, H)$ and $D\,(\gamma, H)$
Micro-scale model
Cell-suspension solver $\nu_1(\gamma_1, H_1)$, $D_1(\gamma_1, H_1)$
Cell-suspension solver $\nu_2(\gamma_2, H_2)$, $D_2(\gamma_2, H_2)$
…
Cell-suspension solver $\nu_\eta(\gamma_\eta, H_\eta)$, $D_\eta(\gamma_\eta, H_\eta)$
Returns viscosity and diffusion coefficients
Collect step (Part of the surrogate model)

Figure 10. Main structure of the red blood cells HMM example used in this section exhibiting the different scales (macro and micro)

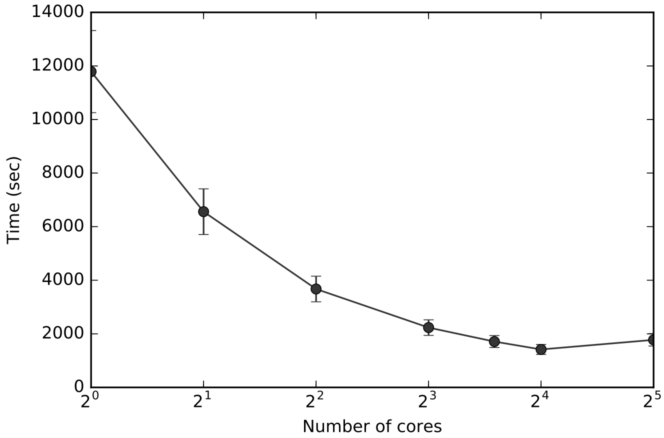


Figure 11. Average performance of the cell-suspension LBM solver, representing micro-scale models. In this example, the system size was 256^3 lattice units, and the total number of simulated cells was 16277

In this study, we replaced the actual surrogate model with the performance graphs in Fig. 5 and 6. 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 HMC pattern software is to schedule this dynamically varying load of micro-scale models in an optimal way on the available resources.

To understand and improve the scheduling of these micro-scale jobs, we need to profile the average performance of the micro-scale models first. Figure 11 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 [35]. Total system was 256^3 lattice units, and the simulation contained 16277 red blood cells. 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 the 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 $P_{max} = 16$.

To confirm the choice of farming a large number of micro-scale models using the least number of processors, in addition to the discussion presented in Section 1.2.1, consider the following example. Assume that $P_\mu = 1024$ processors are reserved for micro-scale models and that in one macro-scale iteration it is required to run $\eta(t) = 4096$ jobs. This is the first phase of the surrogate model performance (i.e. $\eta(t)P_{\min} > P_\mu$). By running the simulation using one core/node per micro-scale model, four batches are needed to finish, where each batch will take $\sim$ 3 core hours. This means that a total of $\sim$ 12 core hours of computing is required. On the other hand, if 16 processors per micro-scale model were to be used, then 64 batches would be needed at 30 min each, resulting in a total of $\sim$ 32 core hours of computational time.

2.2.1. Results

In the following computer simulations the runtime and utilisation for the four cases of performance shown in Fig. 5 and 6 are measured. 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.

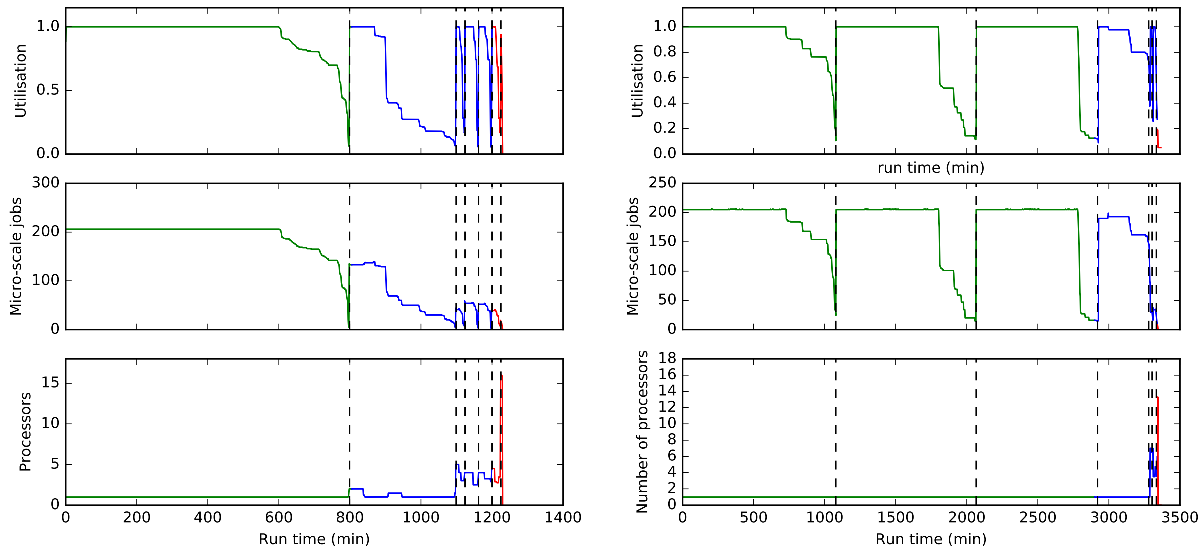
In all these simulations, the number of cores allocated for the macro-scale model was $P_M = 8$, for the surrogate model $P_S = 1$ and for the HMC manager $P_H = 1$. The total number of cores available for the micro-scale models was $P_\mu = 206$. The degree of freedom selected was $DoF = 48818$.

The QCG pilot job used in the experiments 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, the macro-scale model and the HMC manager are submitted. The macro-scale, after an iteration requests a number of parameters from 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. In the benchmarks, we mimic this operation by using the performance of the surrogate model. 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. Also, 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.

Figure 12 (top panels) presents the utilisation of the system, with assistance from the HMC manager and the surrogate model that starts from scratch. The utilisation of the performance of the surrogate model presented in Fig. 5a (model = scratch, good performance) is shown here in Fig. 12a (top panel), while the utilisation of the performance of the surrogate model presented in Fig. 5b (model = scratch, poor performance) is shown in Fig. 12b (top panel).

In Fig. 12a (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 batches). The utilisation in this phase is high because we simply exploit all the available resources. At the outset the utilisation is one, which means that all 206 processors are used as shown in Fig. 12a (middle panel). After a while, between 18–32 jobs are completed, shown as the first few small decreases of the green line.

The blue line in Fig. 12a (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 outset 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 $\sim 1 - 2$ cores, each shown in Fig. 12a 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 cores to refine the surrogate model by proactively (i.e. 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 following iterations (macro-scale iterations 3 to 5), because the number of micro-scale



(a) (b)

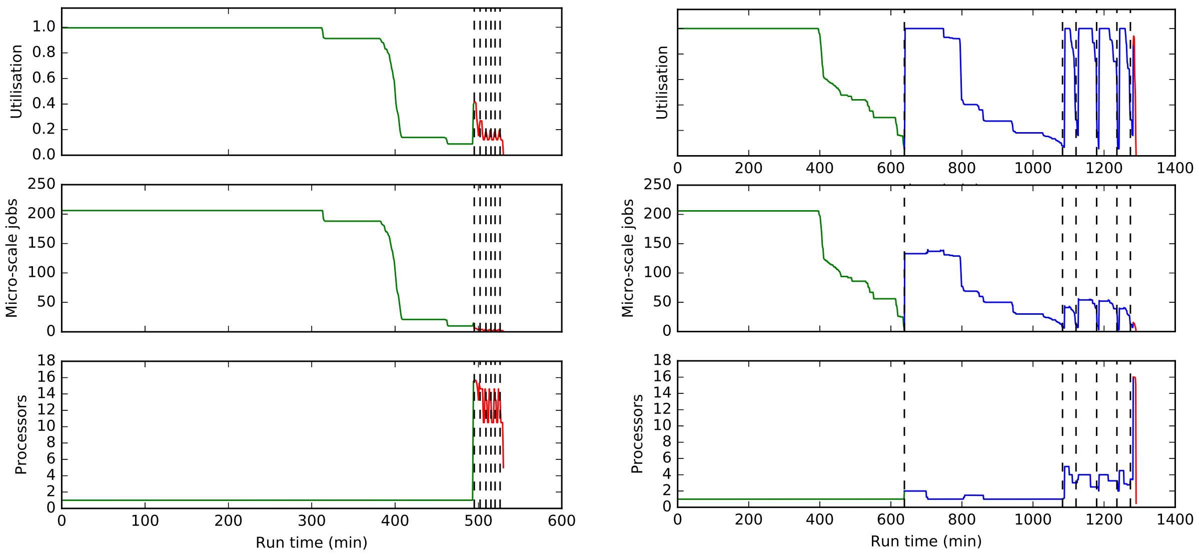
Figure 12. Utilisation of the red blood cell HMM application using PJM (top panels) using the surrogate model from scratch, as shown in case (a) in Section 1.2.2. Good performance level manifest on the left (model = scratch, good performance), while poor performance level is displayed on the right (model = scratch, poor performance). The corresponding number of micro-scale jobs (middle panels) and number of cores per micro-scale job (lower panels) were decided by the HMC manager. The colours represent the phases, where green is the first phase, blue – the second, and red is the third. The dashed lines represent the macro-scale iterations

model jobs in iteration 4, for example, is smaller (54 running with ∼ 4 processors shown in blue in Fig. 12a (middle and lower panels)).

The last macro-scale iteration in Fig. 12a (top panel) falls in the third phase, where the number of micro-scale jobs is only 12, and the number of cores per micro-scale job run is $P_{\max} = 16$. As is shown by the red line in the graph and also the red lines in Fig. 12a (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.

While the second case, Fig. 12b (top panel), shows similar behaviour, the surrogate model is not as effective as for the first case. Figure 12b (middle panel) illustrates the corresponding number of micro-scale jobs and the average number of processors per micro-scale jobs. Here, we need to generate more micro-scale jobs to sample parameter space at the outset. This example also shows that after running a large number of micro-scale jobs for the first three macro-scale iterations, we reached a steady level where we ran 35 micro-scale jobs in the second phase and ∼ 2 micro-scale jobs in the third phase utilising a dynamic number of cores/nodes.

When a previously developed surrogate model is used, the resulting utilisation of the system is as shown in Fig. 13 (top panel), and the corresponding number of micro-scale jobs and the average number of processors per micro-scale jobs are shown in Fig. 13 (middle and lower panels, respectively). This case shows the situation where the surrogate model must initially run a large number of micro-scale models at the outset to fill the database rapidly and then becomes sufficiently effective to replace the need for micro-scale models. The utilisation of the performance of the surrogate model presented in Fig. 6a (model = developed, good performance)



(a) (b)

Figure 13. Utilisation of the red blood cell HMM application using PJM (top panels) by using a previously developed model, as shown in study case (b) in Section 1.2.2. Good performance level manifest on the left (model = developed, good performance), while poor performance level is displayed on the right (model = developed, poor performance). The corresponding number of micro-scale jobs is shown in the middle panel. The lower panel shows the corresponding number of cores per micro-scale job decided by the HMC manager. The colours represent the phases, where green is the first phase, blue – the second, and red is the third. The dashed lines represent the macro-scale iterations

is shown here in Fig. 13a (top panel), while the utilisation of the performance of the surrogate model presented in Fig. 6b (model = developed, poor performance) is shown in Fig. 13b (top panel).

Figure 13 (top panels) appears similar to the previous case, but we notice that because the difference in the number of micro-scale jobs requested for each macro-scale step is high, there are more idle cores in the first phase than for the previous case. Here, we might consider returning the cores back to the system as expected, reaching the steady state in this case is faster than in the previous case, and in the subsequent runs the surrogate model replaces the need for generating new micro-scale jobs, up to where the first few macro-scale iterations are completed as shown in Fig. 13 (middle panels).

Finally, note that in production runs, the number of iterations on the macro-scale will be much larger, and if the surrogate model is performing very well, the overall utilisation can be very small (as, e.g., reported in ref [22]). Once the HMC run is running steadily in phase 3, the HMC manager should return resources to the system.

Conclusion

We have illustrated the mechanism of running HMM applications using pattern software. In this paper, two sources of load imbalance in HMC, namely variable execution time and dynamic number of micro-models, are investigated and a number of solutions are proposed.

First, the large number of micro-scale model simulations at each iteration of the HMC workflow requires them to be performed inside a single large resource allocation to avoid significant and asynchronous queuing time. In turn, the variable execution time of the micro-scale simulation induced significant idle periods on part of the large allocation. A pilot job manager mechanism was introduced to handle internal queuing and execution of these simulations. The flexibility of how the PJM mechanism in comparison to a pre-assigned rigid batching system enables drastic improvements of core utilisation lead to reducing of overall ideal time by up to 70%. The improvement was particularly significant on smaller allocations. In addition, facilitated by the use of a PJM, an optimisation layer has also been used to tackle directly the variability of execution time. Scaling the individual resource sub-allocation of each micro-scale simulation, based on predicted execution time, load balancing was even further reduced with 20% shorter runtime. The adaptive resource allocation was maintained within the range of strong scaling of the model in order to preserve efficiency.

Second, we propose that the execution of the micro-scale models in the HMM application should be viewed as three distinct phases. The first phase arises when a very large number of micro-scale models ($P_\mu < \eta(t)P_{\max}$) needs to be executed. In this phase, the appropriate action is to conduct farming with $P_{\min}$ processors per micro-scale model. This reduces the overhead, as demonstrated mathematically and by means of computational simulations. The second phase occurs when ($P_{\max}\eta(t) < P_\mu < P_{\max}\eta(t)$). In this phase, the required micro-scale jobs are executed with $\lceil P_\mu/\eta(t) \rceil$ cores/nodes. During the last phase, when the number of micro-scale jobs requested is less than the total number of the available processors for micro-scale models, the most appropriate action is to run all the remaining jobs using $P_{\max}$ processors. The surrogate model is nearly or fully developed at this stage and can replace the need to generate micro-scale jobs efficiently. At this stage, certainly when many iterations of the macro-scale need to be simulated, releasing unused processors will result in an increase in utilisation and a reduction of the computational cost. An HMC assuming four different scenarios of surrogate models was executed. In each scenario, the number of micro-scale jobs generated at each macro-scale iteration is shown, the utilisation of the system using a pilot job and the HMC manager, with the focus on running the micro-scale jobs on the right resources.

Although farming independent jobs is a well-known method, the act of dynamically changing from one farming phase to another under the control of a dynamically evolving surrogate model, with its corresponding actions and decisions, is new. In our simulations, we substituted the actual implementation of Gaussian process regression of the surrogate model with the performance of a surrogate model, as observed in an HMM for predicting material properties [22]. This performance figure provides the required number of micro-scale jobs that are needed at each macro-scale iteration. Finally we note that combining the dynamically changing runtime of the micro-scale models and the dynamically changing number of micro-scale models to be executed has not yet been addressed in the paper. However, the combination of the scheduling mechanisms discussed in Section 1, and the overall HMC phases in Section 1.2.2, should be able to handle this situation as well.

Acknowledgments

We acknowledge partial funding from the European Union Horizon 2020 research and innovation programme under grant agreement No 671564 for the ComPat project (http://www.compat-project.eu/). SA acknowledges funding from King Abdulaziz City for Science and Technology (KACST), Saudi Arabia. P.V.C. thanks the MRC Medical Bioinformatics project (MR/L016311/1), the EU H2020 CompBioMed grant (http://www.compbiomed.eu, Grant

No. 675451) and funding from the UCL Provost. This research was also supported in part by the PLGrid Infrastructure including dedicated HPC resources at the Poznan Supercomputing and Networking Center.

References

1. Abdulle, A., Weinan, E., Engquist, B., Vanden-Eijnden, E.: The heterogeneous multiscale method. Acta Numerica 21, 1–87 (2012), DOI: 10.1017/S0962492912000025

2. Aktulga, H.M., Fogarty, J.C., Pandit, S.A., Grama, A.Y.: Parallel reactive molecular dynamics: Numerical methods and algorithmic techniques. Parallel Computing 38(4), 245–259 (Apr 2012), DOI: 10.1016/j.parco.2011.08.005

3. Alowayyed, S., Piontek, T., Suter, J.L., Hoenen, O., Groen, D., Luk, O., Bosak, B., Kopta, P., Kurowski, K., Perks, O., Brabazon, K., Jancauskas, V., Coster, D., Coveney, P.V., Hoekstra, A.G.: Patterns for high performance multiscale computing. Future generation computer systems 91, 335–346 (2019), DOI: 10.1016/j.future.2018.08.045

4. Alowayyed, S.: Patterns for multiscale computing. Ph.D. thesis, University of Amsterdam (2018), `http://hdl.handle.net/11245.1/9cf904f8-fcc7-4b8a-a105-622a865359d8`

5. Alowayyed, S., Groen, D., Coveney, P.V., Hoekstra, A.G.: Multiscale computing in the exascale era. Journal of Computational Science 22, 15–25 (2017), DOI: 10.1016/j.jocs.2017.07.004

6. ASTM: Test method for short-beam strength of polymer matrix composite materials and their laminates, DOI: 10.1520/d2344_d2344m-16, accessed: 2019-11-29

7. Axner, L., Bernsdorf, J., Zeiser, T., Lammers, P., Linxweiler, J., Hoekstra, A.G.: Performance evaluation of a parallel sparse lattice Boltzmann solver. Journal of Computational Physics 227(10), 4895–4911 (2008), DOI: 10.1016/j.jcp.2008.01.013

8. Bangerth, W., Hartmann, R., Kanschat, G.: Deal.II–A general-purpose object-oriented finite element library. ACM Trans. Math. Softw. 33(4) (Aug 2007), DOI: 10.1145/1268776.1268779

9. Borgdorff, J., Ben Belgacem, M., Bona-Casas, C., Fazendeiro, L., Groen, D., Hoenen, O., Mizeranschi, A., Suter, J.L., Coster, D., Coveney, P.V., Dubitzky, W., Hoekstra, A.G., Strand, P., Chopard, B.: Performance of distributed multiscale simulations. Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences 372 (2014), DOI: 10.1098/rsta.2013.0407

10. Borgdorff, J., Lorenz, E., Bona-Casas, C., Hoekstra, A.G., Falcone, J.L., Chopard, B.: Foundations of distributed multiscale computing: Formalization, specification, and analysis. J. Parallel Distrib. Comput. Journal of Parallel and Distributed Computing 73(4), 465–483 (2013), DOI: 10.1016/j.jpdc.2012.12.011

11. Borgdorff, J., Mamonski, M., Bosak, B., Kurowski, K., Belgacem, M.B., Chopard, B., Groen, D., Coveney, P., Hoekstra, A.: Distributed multiscale computing with MUSCLE 2, the Multiscale Coupling Library and Environment. Journal of Computational Science 5(5), 719–731 (2014), DOI: 10.1016/j.jocs.2014.04.004

12. Bouvard, J.L., Ward, D.K., Hossain, D., Nouranian, S., Marin, E.B., Horstemeyer, M.F.: Review of hierarchical multiscale modeling to describe the mechanical behavior of amorphous polymers. Journal of Engineering Materials and Technology 131(4) (Sep 2009), DOI: 10.1115/1.3183779

13. Cheng, L.T., Weinan, E.: The heterogeneous multi-scale method for interface dynamics. Contemporary mathematics. 330, 43–54 (2003), DOI: 10.1007/978-94-007-0412-1_18

14. Chopard, B., Falcone, J.L., Kunzli, P., Veen, L., Hoekstra, A.: Multiscale modeling: recent progress and open questions. Multiscale and Multidiscip. Model. Exp. and Des. Multiscale and Multidisciplinary Modeling, Experiments and Design 1(1), 57–68 (2018), DOI: 10.1007/s41939-017-0006-4

15. Coveney, P.V., Boon, J.P., Succi, S.: Bridging the gaps at the physics–chemistry–biology interface. Philosophical Transactions of the Royal Society of London A: Mathematical, Physical and Engineering Sciences 374(2080), 335–339 (2016), DOI: 10.1098/rsta.2016.0335

16. Engquist, B., Lötstedt, P., Runborg, O.: Multiscale modeling and simulation in science. Springer Science & Business Media, Heidelberg (2009), DOI: 10.1007/978-3-540-88857-4

17. Hoekstra, A.G., Chopard, B., Coster, D., Portegies Zwart, S., Coveney, P.V.: Multiscale computing for science and engineering in the era of exascale performance. Phil. Trans. R. Soc. A Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences 377(2142), 20180144 (2019), DOI: 10.1098/rsta.2018.0144

18. Hoekstra, A.G., Chopard, B., Coveney, P.V.: Multiscale modelling and simulation: a position paper. Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences 372, 30377–30385 (2014), DOI: 10.1098/rsta.2013.0377

19. Hoekstra, A.G., Portegies Zwart, S., Coveney, P.V.: Multiscale modelling, simulation and computing: from the desktop to the exascale. Phil. Trans. R. Soc. A Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences 377(2142), 20180355 (2019), DOI: 10.1098/rsta.2018.0355

20. Knap, J., Spear, C.E., Borodin, O., Leiter, K.W.: Advancing a distributed multi-scale computing framework for large-scale high-throughput discovery in materials science. Nanotechnology 26(43), 434004–434016 (2015), DOI: 10.1088/0957-4484/26/43/434004

21. Kopta, P., Bosak, B.: QCG-PilotJob. https://github.com/compat-project/QCG-PilotJob (2018)

22. Leiter, K.W., Barnes, B.C., Becker, R., Knap, J.: Accelerated scale-bridging through adaptive surrogate model evaluation. Journal of Computational Science 27, 91–106 (2018), DOI: 10.1016/j.jocs.2018.04.010

23. Lorenz, E., Hoekstra, A.G.: Heterogeneous multiscale simulations of suspension flow. Multiscale Modeling and Simulation 9(4), 1301–1326 (2011), DOI: 10.1137/100818522

24. Lorenz, E., Hoekstra, A.G., Caiazzo, A.: Lees-edwards boundary conditions for lattice Boltzmann suspension simulations. Phys. Rev. E Physical Review E 79(3) (2009), DOI: 10.1103/PhysRevE.79.036706

25. Luk, O., Hoenen, O., Perks, O., Brabazon, K., Piontek, T., Kopta, P., Bosak, B., Bottino, A., Scott, B.D., Coster, D.P.: Application of the extreme scaling computing pattern on multiscale fusion plasma modelling. Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences 377(2142), 20180152 (2019), DOI: 10.1098/rsta.2018.0152

26. Nikishova, A., Veen, L., Zun, P., Hoekstra, A.G.: Uncertainty quantification of a multiscale model for in-stent restenosis. Cardiovascular Engineering and Technology (2018), DOI: 10.1007/s13239-018-00372-4

27. Plimpton, S.: Fast parallel algorithms for short-range molecular dynamics. Journal of Computational Physics 117(1), 1–19 (Mar 1995), DOI: 10.1006/jcph.1995.1039

28. Sloot, P.M.A., Hoekstra, A.G.: Multi-scale modelling in computational biomedicine. Briefings in bioinformatics 11(1), 142–152 (2009), DOI: 10.1093/bib/bbp038

29. Turilli, M., Santcroos, M., Jha, S.: A comprehensive perspective on Pilot-Job systems. ACM Computing Surveys 51(2), 1–32 (2018), DOI: 10.1145/3177851

30. Vassaux, M., Richardson, R., Coveney, P.V.: The heterogeneous multiscale method applied to inelastic polymer mechanics. Philos. Trans. R. Soc. A Math. Phys. Eng. Sci. Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences 377(2142) (2019), DOI: 10.1098/rsta.2018.0150

31. Wang, C., Duan, Q., Gong, W., Ye, A., Di, Z., Miao, C.: An evaluation of adaptive surrogate modeling based optimization with two benchmark problems. Environmental Modelling and Software 60, 167–179 (2014), DOI: 10.1016/j.envsoft.2014.05.026

32. Weinan, E.: Principles of multiscale modeling. Cambridge University Press, Cambridge, New York (2011), DOI: 10.1063/PT.3.1609

33. Weinan, E., Engquist, B., Huang, Z.: Heterogeneous multiscale method: a general methodology for multiscale modeling. Physical Review B 67(9), 092101 (2003), DOI: 10.1103/PhysRevB.67.092101

34. Závodszky, G., van Rooij, B., Azizi, V., Hoekstra, A.G.: Cellular level in-silico modeling of blood rheology with an improved material model for red blood cells. Frontiers in physiology 8 (2017), DOI: 10.3389/fphys.2017.00563

35. Závodszky, G., van Rooij, B., Azizi, V., Alowayyed, S., Hoekstra, A.G.: Hemocell: a high-performance microscopic cellular library. Procedia Computer Science 108, 159–165 (2017), DOI: 10.1016/j.procs.2017.05.084