Research

Cite this article: Luk OO *et al.* 2019

Application of the extreme scaling computing pattern on multiscale fusion plasma modelling. *Phil. Trans. R. Soc. A* **377**: 20180152.
<http://dx.doi.org/10.1098/rsta.2018.0152>

Accepted: 23 October 2018

One contribution of 11 to a theme issue
'Multiscale modelling, simulation and
computing: from the desktop to the exascale'.

Subject Areas:

computational physics, computer modelling
and simulation, plasma physics

Keywords:

multiscale modelling, fusion plasma, high
performance computing, performance
metrics, profiling

Author for correspondence:

O. O. Luk
e-mail: onnie.luk@ipp.mpg.de

Application of the extreme scaling computing pattern on multiscale fusion plasma modelling

O. O. Luk¹, O. Hoenen¹, O. Perks², K. Brabazon²,
T. Piontek³, P. Kopta³, B. Bosak³, A. Bottino¹,
B. D. Scott¹ and D. P. Coster¹

¹Max-Planck-Institut für Plasmaphysik, Garching, Germany

²ARM Ltd, Warwick, UK

³Poznań Supercomputing and Networking Center, Poznań, Poland

00L, 0000-0003-0560-4797

The extreme scaling pattern of the ComPat project is applied to a multi-scale workflow relevant to the magnetically confined fusion problem. This workflow combines transport, turbulence and equilibrium codes (together with additional auxiliaries such as initial conditions and numerical module), which aims at calculating the behaviour of a fusion plasma on long (transport) time scales based on information from much faster (turbulence) time scales. Initial findings of profile measurements are reported in this paper and indicate that, depending on the chosen performance metric for defining 'cost', such as time to completion, efficiency and total energy consumption of the multiscale workflow, different choices on the number of cores would be made when determining the optimal execution configuration. A variant of the workflow which increases the inherent parallelism is presented, and shown to produce equivalent results at (typically) lower cost compared with the original workflow.

This article is part of the theme issue 'Multiscale modelling, simulation and computing: from the desktop to the exascale'.

1. Introduction

Nuclear fusion provides the source, directly or indirectly, for almost all of the energy used on earth: directly

Figure 1. Plot showing the multiscale nature of the dynamics in a fusion plasma. Colour coding is used to identify the dimensionality of the problem. In this paper, we focus on the turbulence and core transport problems.

via the use of sunlight which is provided by nuclear fusion occurring in the sun, not only by photovoltaics or solar-thermal systems, but also by the production of organic carbon products via photosynthesis or wind driven by thermal inputs from the sun; and indirectly via the use of fossil fuels (which were derived by photosynthesis in the past) and nuclear energy, where the fissile material arose from fusion-driven processes in stars.

Since the middle of the last century, scientists and engineers have explored ways of harnessing nuclear fusion in a controlled way to produce energy. In the intervening 70 or so years, our understanding of the problem has deepened, and there is currently an effort between seven parties (the European Union, China, India, Japan, Russia, South Korea and the USA) to build ITER¹ in southern France. However, there is still substantial ongoing research to understand the behaviour of a fusion plasma.

Indeed, the dynamics of a thermonuclear fusion plasma is very complex. The complexity arises from the instabilities driven by plasma turbulence, which in turn can destroy plasma confinement. Fully simulating the impact of turbulence on the performance of a fusion device such as ITER is challenging. Doing so solely with a turbulence code is projected to exceed even the next generation of exascale computers, so a multiscale approach will be necessary. Fusion plasmas exhibit highly disparate spatio-temporal scales: turbulence spans roughly microseconds in time and millimetres in space, and overall transport spans in the order of few seconds in time and a few metres in space, while other involved phenomena stand at a different scale, as shown in figure 1.

When scientists want to study complex multi-scale phenomena by using numerical simulations on a computer, an interesting idea is to follow a component-based approach, where a multi-scale model is described through a set of coupled single-scale submodels. Following such idea, each submodel is easier to design compared to the full multi-scale one, and very often such single-scale submodel have already been extensively tested, verified and validated as a standalone model. This approach also has several advantages in terms of software engineering: it leads to a set of simpler algorithms and code base for each submodel; and, it allows for faster and better-optimized implementations that are easier to debug and maintain, especially when

¹<https://www.iter.org/>.

developed by several people. However, despite the benefits at the submodel level, this approach implies added complexity in binding all submodels within a coupled multiscale application and can generate runtime overheads. To alleviate the efforts required from the application developers, many generic and domain specific projects are developing coupling libraries, frameworks and execution platforms. An overview of these efforts is given in [1].

One of these efforts, developed over almost a decade within different European projects, leads to the concept of the multiscale modelling and simulation framework (MMSF). It is currently used in different scientific communities that are dealing with multi-scale or multi-physics problems, such as thermonuclear fusion, astrophysics, material science and biomedicine. MMSF, described in detail in [2], offers both a theoretical and a computational framework. When they are taken together as a pipeline, MMSF can guide scientists through all the steps, from first conceptual models to comprehensive simulation of the multi-scale phenomena on targeted set of computing resources. This multi-stage pipeline organization allows for more flexibility: scientists can decide to either use the entire pipeline if they are developing their application from scratch, or only a subset of different stages if existing models or legacy codes are available.

There are also fusion domain specific efforts toward multiscale modelling of fusion plasma. Nishimura *et al.* [3], for example, used a nonlinear drift wave turbulence simulation to calculate the transport coefficients for tokamak edge plasmas. LoDestro *et al.* [4] combined a gyrokinetic particle simulation that studies instabilities of drift-wave time scale with a numerical solution that provides closure approximations to the fluid equations to study low-frequency plasma turbulence. Shestakov *et al.* [5] created a self-consistent model by coupling a turbulence simulation, which solves two-dimensional Hasegawa–Wakatani equations, to a one-dimensional transport model with implicit temporal discretization scheme to advance the equilibrium fields. Some of the more recent efforts on integrated modelling include building a transport solver that takes steady-state turbulent fluxes from a turbulence code [6] and a transport manager that can run turbulent transport and neoclassical codes concurrently [7]. There are other more extensive modelling frameworks as well. For example, an integrated modelling Python framework is established to take users' feedback and inputs to continually improve and broaden the multiscale physics model [8]. In particular, the European Integrated Tokamak Modelling [9] effort (EFDA-ITM) designed a generic platform for tokamak modelling and framework to aid the construction of complex workflows [10] that follow the component-based approach. This platform is currently sustained through the EUROfusion consortium,² which contributes extensively to the ITER Integrated Modelling and Analysis Suite (IMAS) [11].

In this paper, we present the initial results from the use of the ComPat framework and software to study multiscale fusion plasmas under one of the multiscale computing patterns. In particular, we study the profiles of every submodel in two workflow designs measured from several computer clusters. Specifics of the ComPat project and its software are discussed in §2. The implementation of the fusion multiscale application using the ComPat framework along with preliminary results are presented in §3. Section 4 presents the computational performance of the implemented application, followed by a discussion of two methods to improve the performance. Finally, conclusion and future plans are discussed in §5.

2. The ComPat Project

With the performance of supercomputers on the rise, there is a growing demand for heavy HPC resources to allow for higher resolution within multiscale applications. However, as HPC systems have been designed historically for monolithic codes, additional effort is required to efficiently map multiscale applications following the MMSF approach to top tier supercomputers. The ComPat project³ aims at filling the gap between MMSF and traditional HPC systems by

²<https://www.euro-fusion.org/>.

³<http://www.compat-project.eu/>.

focusing on two major objectives: §2a to provide a collection of methods and software that ease the development of component-based, multiscale simulations following the MMSF or similar approach, and §2b to create generic transformation and optimization methods that improve the simulation's runtime and/or other performance metric on a targeted set of execution platforms, despite the coupling overheads. Here we present a part of the software and algorithmic solutions proposed in ComPat, which we use in our implementation of a fusion application. Additional solutions from ComPat are described in [12,13] and out of scope in this work, as they fit in other categories of applications.

(a) Simplify build of multiscale applications, from concept to execution

The ComPat software stack is made of several components that are at the core of MMSF. First the multiscale modelling language (MML) and its representation in XML format (xMML) allow developers to describe, at a conceptual level, submodels and their interactions [14] within the targeted multiscale application. Given such representation, a small tool (jMML⁴) generates graphical representations of the topology, task-graph and skeleton configuration file for the coupling framework.

This framework is called the Multiscale Coupling Library and Environment (MUSCLE2⁵). It is composed of two parts: a library handling data exchanges between submodels, and a runtime environment to orchestrate the execution of each submodel on possibly distributed computing resources [15]. The library exposes a simple API in Java, C, C++, Python and Fortran, allowing developers of submodels written in any of these programming languages to query parameters, send and receive data (arrays, strings and raw bytes), log and stage in and out files. As MUSCLE2 follows a component-based approach, the implementation of a submodel (called a *kernel*) does not impact the other submodels present in the coupled application. One only has to ensure that the data types between directly coupled submodels are consistent with each other. When this is not the case, MUSCLE2 provides several mechanisms to transform these data in transit through light-weighted filters or combine data from several sources through the so-called mappers. Such a design for implementing submodels guarantees that a kernel can be re-used in a different application context, or exchanged with another kernel without impacting the rest of the components present in a given application. A coupled application in MUSCLE2 is defined by a single configuration file, which can be partially generated from the MML description of the application. Implemented in Ruby, this configuration file contains declaration of all kernels with their coupling, simulation data, and parameters that are either global or local to a kernel. This file is then used by the runtime environment to execute the simulation with different resource settings: one can decide to run all components within a single runtime environment, or distribute the kernels among several runtime environments (potentially on different computers) without changing a single line in the configuration file.

Even though the MUSCLE2 can distribute a simulation over remote computing resources, the deployment and set-up for such a run is tedious and can be simplified through the use of a computing resource broker. The QCG middleware⁶ provides such functionality, and its overall architecture is summarized in figure 2. For the end-user, interaction is done through a client in the grid domain, where the user is identified through a X.509 certificate. The QCG client can provide either a simple interface through batch scripts similar to most common local resource management systems (LRMS), or an advanced interface through a XML document for more complex scenario. Providing that the targeted HPC systems grant access to part of their resources through some Grid or Cloud environment, using QCG to broker remains advantageous even for simulations that do not require cross-cluster execution: the end-user can maintain a single QCG

⁴<https://github.com/blootsvoets/jmml>.

⁵<https://github.com/psnc-apps/muscle2>.

⁶<http://www.qoscosgrid.org/trac/qcg>.

Figure 2. Overview of the middleware architecture: QCG components and their relationship. (Online version in colour.)

script for all LRMS and local configurations, and monitor runs on a single server, laptop or web service portal. More details can be found in [16,17].

(b) Estimate the performance of multiscale applications

Once a set of MUSCLE2 kernels is implemented into the multiscale application, the next step is to study the amount of computing resources required by the kernels. If the multiscale application contains a highly scalable parallel kernel, and/or a large number of kernels or instances of a given kernel, then the simulation has to run on a set of computing resources from large HPC systems. In such case, optimizing the performance becomes a necessity. Compared to a traditional monolithic code, a coupled application generates overheads (typically the time spent in MUSCLE2 in our case) that decrease the overall performance. However, it is possible to run each kernel on computing resources (general purpose, accelerators or even FPGA) adapted to its dominant algorithm. This introduces an additional level of parallelism at the coupling level, which brings many opportunities to hide the overheads and in some cases out-perform a more traditional implementation.

In that context, the underlying idea in ComPat is that performance of the entire multiscale application can be predicted by combining two pieces of information together: (1) computational costs of every single-scale submodel and (2) interactions among submodels. These interactions are described by the submodel coupling topology and other parameters, such as the number and dynamicity of their instances. In fact, the claim is that, with only three configurations and any of their combinations, one can describe all kinds of multiscale applications and predict their performances on any HPC where the submodels have been profiled. These configurations are called the multiscale computing patterns (MCPs), and they are the *extreme scaling* (ES), *heterogeneous multiscale computing* (HMC) and *replica computing* (RC) patterns. In particular, a multiscale application follows the ES pattern if one submodel (the *primary*) dominates by far in terms of application computing costs compared to the other submodels (the *auxiliaries*), and both the number and creation of all submodels are static. A detailed description of each MCP can be

Figure 3. The ARM MAP tool provides full traces of performance metrics throughout the simulation run. In this particular example, MAP measures the profiles of a parallelized (with MPI) Fortran kernel running on 512 cores. The profiles in display include ratio compute/communications/I/O, CPU floating point rate, power usage and MUSCLE2 receive duration. Finer details can be made available through profiles at the subroutine/function level or even at a given line in the source code.

found in [12]. For the rest of the paper, we focus on the ES MCP, for it fits the description of our targeted fusion application. After such parametric prediction, for a given application that fits in a given pattern, its result can then be used to select the best scenario corresponding to a given problem depending on available computing resources.

To profile the performance of each submodel *in situ* (as a MUSCLE2 kernel within the context of the coupled multiscale application), the MAP tool from ARM⁷ has been integrated into the ComPat software stack. MAP is a non-intrusive tool that records the time spent on each line of code from a C, C++ or Fortran program. In order to activate the MAP tool, the program needs to compile with debug flag so it can show the report along with source code. In addition to the traditional metrics such as runtime, I/O, memory usage and MPI, MAP can monitor energy consumption at the node level and highlight the time spent in the MUSCLE2 API. Full traces of metrics shown in figure 3 can provide very deep insight into the performance of a kernel, allowing developers to optimize the kernel's implementation regardless of the performance of the coupled application. To estimate the performance of the coupled application, a smaller set of metrics has to be extracted from such comprehensive profiles in order to combine them with the coupling topology description. The Optimization part [13] of the ComPat software stack is a service that allows benchmarks to run automatically on all available resource types, gathers profiling information into a performance database, allows end-user to select the metric to be optimized (cpu-hours, time to completion, energy), and chooses the proper configuration to minimize cost of the selected metric.

3. Example fusion application, implementation and first results

To study the dynamics of fusion plasmas, we want to build a multiscale simulation that can incorporate the effects of microturbulence into the overall plasma transport. Besides connecting the transport (TRA) submodel to the flux-tube local turbulence (TUR) submodel, we need to update plasma profiles and boundary geometry using an equilibrium (EQU) submodel. These three submodels, including their respective scales, dimensionality and interactions, are summarized in figure 4. To bridge the micro time scales of the turbulence to the macro time scales

⁷<https://developer.arm.com>.

Figure 4. Generic view of the main submodels involved in our fusion multiscale application, as first introduced in [18]. (Online version in colour.)

of transport solver, the heat-fluxes from the TUR submodels are averaged. Once this multiscale method is well understood and proven to give reasonable physics results, we can move forward and include more physics (e.g. plasma heating) and new transport channels into the multiscale simulation.

The turbulence component of the target application requires large amount of HPC resources. Therefore, in order to have better control over the performance of the coupled simulation, we adapt the ComPat methodology and software stack presented in §2. Following the complete MMSF approach, we start this integration work with submodels already implemented into each of the targeted components, as these were previously developed within the EFDA-ITM project. In the span of over a decade, this project delivered several essential components towards the integrated simulation of complex phenomena occurring in tokamak devices.

- (i) The design of standardized code interface through a common data ontology (made of *Consistent Physical Objects* or CPO [19]) is implemented in a code-and-language-agnostic manner.
- (ii) A large list of codes and modules adapted to the CPO, covering a wide range of aspects such as transport solvers for interpretive and predictive scenario, heating and current drives, equilibrium, impurities, and various neoclassical and turbulence transport modules.
- (iii) Several applications (or workflows) designed and orchestrated by the open-source Kepler⁸ tool, which allows for benchmarking and verifying interchangeable physics modules, and coupling different physics modules into a more complex simulation [10].

We decide to re-use (i) and (ii) to benefit from the extensively tested and validated modules that are ready for coupling. For TRA submodel the selected code is ETS [20], which can solve equations describing poloidal flux, density equation for every ion species, temperature equations for both ions and electrons, and the toroidal rotation equation for ions. The EQU submodel we use is CHEASE [21], which is a high-resolution fixed-boundary Grad-Shafranov solver. For TUR submodel, we have chosen GEM [22], which is a gyrofluid electromagnetic flux-tube model that determines heat and particle fluxes at each flux-tube in a field-aligned shifted-metric coordinate system. The calculations at each flux-tube are done concurrently. An additional component (IMP4DV module based on dynamical alignment [23]), is added to the application in order to transform the turbulence code output (fluxes) into transport coefficients expected by ETS. The topology of this application is summarized in figure 5.

Since we are more interested in high performance computing rather than the ease of design (lego blocks, graphical interface, Java), we decide to replace (iii) with MUSCLE2. Implementing

⁸<https://kepler-project.org/>.

```

MUSCLE_Init(argc , argv)
p=MUSCLE_get_property("ref")
while !MUSCLE_will_stop() {
  MUSCLE_Receive("Pin",Din)
  compute(Din,p,Dout)
  MUSCLE_Send("Pout",Dout)
}
MUSCLE_Finalize()

```

Listing 1. MUSCLE2 kernel structure

Figure 5. Initial topology (WF_{sync}) of the fusion application. (Online version in colour.)

MUSCLE2 components from existing codes that are previously called from a component in Kepler is made easier by strictly following the generic approach detailed in [18], for which a modular and non-intrusive implementation is based on a hierarchy of interchangeable wrappers. The structure of a standard MUSCLE2 kernel is given in Listing 1, where Din/Dout are the input and output data, and Pin/Pout are the names of the input and output ports, respectively. This snippet of code shows that the API of MUSCLE2 has some similarities with MPI, which could reduce the learning curve and improve the uptake for developers of HPC applications. One major difference is the notion of *ports*, which identify input and output conduits through which the data are received and sent. Once all the kernels are implemented and each is compiled into an executable, including the INIT component that imports initial data into the system from various sources (files, databases, etc.), the coupled application sketched in figure 5 is described in a Ruby script. The connections between kernels can be specified within the script by pairing output port from one kernel to the input port of another kernel. The special *dup* components are provided by the MUSCLE2 standard library: they are generic duplication data mappers that scatter data when needed by several consumers.

To test this application, an ASDEX Upgrade tokamak-like scenario is simulated until a pseudo-convergence criteria is met. The time evolution of the q_e and T_e at eight flux-tubes are shown in figure 6. Every time GEM is called within the workflow, it runs 5000 internal time steps. Each one of these steps is $0.002 L_\perp/c_s$, in which $L_\perp$ is the background profile length scale and c_s is the speed of sound. The vorticity scale is approximately $c_s/L_\perp$. Each call to GEM is equivalent to one time step of ETS (Δt). Owing to high temporal disparity between the turbulence and transport calculations, a time bridging method is constructed to ensure the T_e and its radial gradient $\nabla_\rho T_e$ (both determined within ETS) do not evolve too abruptly and cause GEM to become unstable. Hence, an adaptive Δt is used to control the T_e and $\nabla_\rho T_e$ values. Another challenge with such application is to find the quasi-steady state of the plasma. One method is to take the time average of $T_e(\langle T_e \rangle)$ for each flux-tube, at every t -time iterations, and the $\langle T_e \rangle$ from the previous time groups are compared to the current one. If the previous values are within one standard deviation (s.d.) of the current $\langle T_e \rangle$, then we claim that the quasi-steady state has been reached. Further details of this work can be found in [24].

4. Improvement of the performance for multiscale simulations

By implementing our multiscale fusion plasma simulation within the ComPat methodology and software stack, we obtain a description of the coupled application topology and performance profiles for each of the involved submodels. These profiles from each submodel include several performance metrics measured from multiple types of HPC hardware where simulation was benchmarked. We can use that added knowledge in two different ways in order to further optimize our multiscale simulation introduced in the previous section.

Figure 6. Time history of the electron heat flux q_e (top panel) and electron temperature T_e (bottom panel) of every flux-tube, and time step size Δt (middle panel).

(a) Increasing the level of parallelism of coupled applications

After a simple analysis on data dependencies from the topology described by WF_{sync} in figure 5, we determine that all components have to be executed synchronously one after the other in a sequence. Every code within its component has different computational requirements (GEM is parallel while others are serial), and this leads to a decrease in parallel efficiency. The reason behind it is that GEM remains idle as it waits for other codes to finish their calculation step and send updated input data. This limits the speedup of the application even with an ideal implementation of TUR submodel that scales perfectly. Nevertheless, a careful analysis of the time scales associated with each submodel shows that EQU is scaleless: there is no requirement (as far as time evolution is concerned) to keep it synchronized with TRA and TUR. In addition, the geometry information from the EQU submodel is expected to barely change within one time step, since the profiles are evolving slowly. Therefore, a new topology for the application is constructed, where TUR and FDV submodels use equilibrium data from the previous iteration. This alternate topology, WF_{async} , is illustrated in figure 7; it eliminates the direct data dependency between TUR and EQU, thus allowing GEM and CHEASE codes to run in parallel. To demonstrate whether the delayed equilibrium CPOs would affect the physics results, the same test case presented in §3 is executed again. The results from both workflows show that the core plasma reaches quasi-steady state at a comparable time, and the electron temperature profile obtained from the asynchronous application matches well within 1 s.d. compared to the profile from the synchronous workflow, as shown in figure 8. More details can be found in [24].

In order to quantify the gain in performance from the change in topology, we simulate 4000 iterations using GEM on 1024 cores of the Marconi-Fusion⁹ supercomputer, which is a part of the PRACE Tier 0 system from CINECA dedicated to support fusion research within the EUROfusion consortium. WF_{async} is measured to have a 5.7% shorter wallclock time compared with WF_{sync} . This gain in time is correlated to the intrinsic parallel scalability of the turbulence code. However, GEM is not designed for high core-count scalability, so its parallel efficiency is expected to decrease significantly beyond 128 cores per flux-tube. In other words, since there are a total of eight flux-tubes, GEM's scalability is far from linear once the core count goes beyond 1024. To study further into the potential impact of such change in topology, we calculate the theoretical

⁹<http://www.hpc.cineca.it/hardware/marconi>.

Figure 7. New topology (WF_{async}) of the fusion application. The concurrent execution section (box with dotted border) is where submodels can run in parallel. (Online version in colour.)

Figure 8. Impact of the asynchronous workflow, in comparison with the synchronous one, on electron temperature profile at quasi-steady state.

speedup limits of both configurations using Amdahl's Law [25]. Here we assume that GEM scales linearly from 1 to 16 cores per flux tube and an iteration of the turbulence code is always longer than an iteration of the equilibrium code. As shown in figure 9, WF_{async} can potentially lead to a large increase in speedup to the component-based multiscale application. However, the measured speed-up values clearly diverge from the theoretical values starting at 1024 cores, which can be explained by the scalability limitations of GEM code on such type of problem/grid size. For different types of problems, especially when a more sophisticated and scalable gyrokinetic turbulence code is implemented as the TUR submodel, we expect to see the measured speed-up to follow much more closely to the theoretical values at a higher core count.

(b) Automatic selection of best configurations for a given simulation

A multiscale application implemented with the ComPat software stack benefits from the automated performance profiling on all single-scale submodels, thanks to the ARM MAP tool. As presented in §2, the curated profiled data containing meta-data describes the parameters of the

Figure 9. Theoretical speed-up for the fusion multiscale application depending on the selected topology, given a TUR submodel implementation that scales linearly, and measured speed-up with the GEM gyrofluid code as TUR submodel.

submodels and its execution environment. The data also include runtime, time spent in MUSCLE2 operations, time spent in MPI operations and energy consumption on the node. This information can be used along with the coupling topology to predict the overall performance of simulation for relevant costs. From the end-user point of view, the categories of main interests are:

- Time to completion (T) of the simulation, which is currently equivalent to the wall clock time. In particular, T is minimized in order to obtain results as quickly as possible. T is especially useful when the queuing time is incorporated into its definition. However, such information is not yet available.
- Efficiency (ϵ), which is defined in equation (4.1), with N_p , N_a and N as number of cores for primary, auxiliary and multiscale models, respectively; and, T_p , T_a as time spent on the primary model and auxiliary models, respectively. This metric should be optimized in order to produced the most out of a given computing allocation, and can be combined with a deadline for any instance associated with the computing allocation or specific deliverable.

$$\epsilon = \frac{N_p T_p + N_a T_a}{NT}. \quad (4.1)$$

- Total energy (E_{tot}), which is the sum of energy consumed by each node that the involved MUSCLE2 kernels are running on. While it is not yet clear how such metric will be used in the future, energy is becoming a very important topic in the roadmap for the next generation of exascale computers. Thus, developers will certainly benefit from learning how their applications behave in that aspect.

As a demonstration, the fusion application in both topologies WF_{sync} and WF_{async} are simulated as a short 4-iteration benchmark via the QCG client, which then submits runs in all possible distributions of resources on available HPC systems, monitor the performance of each MUSCLE2 kernel with MAP, and gather these performance profiles into a database for future usage and analysis. All performance data collected in the past are then extracted from this database and combined together to produce estimates on performance metrics for the entire

Table 1. Overall Time to Completion (T), Efficiency (ϵ) and Total Energy (E_{tot}) consumed by four iterations of the fusion application described by WF_{sync} , using various numbers of cores.

no. cores	128	256	512	1024	2048
T (s)	837.6	454.0	232.7	142.7	138.3
ϵ	0.968	0.943	0.899	0.827	0.790
E_{tot} (MJ)	3.885	3.606	3.174	3.460	6.201

Table 2. Overall time to completion (T), efficiency (ϵ) and total energy (E_{tot}) consumed by 4 iterations of the fusion application described by WF_{async} , using various numbers of cores.

no. cores	130	258	514	1026	2050
T (s)	828	440	219	128	111
ϵ	0.972	0.970	0.955	0.926	0.900
E_{tot} (MJ)	3.929	3.436	3.105	3.158	5.219

multiscale application. For this example, we apply constraint such that we are only looking into data coming from runs executed on the so-called thin-nodes from SuperMUC phase 1 in LRZ.¹⁰ Each one of these thin-nodes has 16 cores and 32 GB of memory. Table 1 shows how the chosen performance metrics evolve for topology WF_{sync} , as N_p for GEM (the primary model) ranges from 128 to 2048 cores, and other auxiliary codes (CHEASE, IMP4DV and ETS) share the idle resources.

The same measurements are also provided in table 2 for the asynchronous application WF_{async} . Here, as long as the runtime for a step of CHEASE is less than the runtime for a step of GEM (a fair assumption unless in a very peculiar setting), time from CHEASE is not taken into account when estimating T . Here, we add two more cores compared to the runs presented in table 1) such that there is enough resources to allow both GEM and CHEASE to run concurrently.

Figure 10 shows a comparison of the normalized overall costs T , NT (which is related to ϵ) and E_{tot} , where the normalization for each metric is carried out by dividing all of its measured values by its minimum cost measured in WF_{sync} . We see that, depending on which metric is being considered, a different resource configuration provides the optimal setting: in SuperMUC 128-, 512- and 2048-core runs cost the least in terms of NT , E_{tot} and T , respectively. The measure of E_{tot} may tie into the efficiency of the submodels, especially the parallelized submodels (e.g. GEM). We also suspect that NT plays a strong role in the measurement of E_{tot} , especially at higher core count (e.g. above 512 cores). However, this would require further investigation in order to confirm. When similar costs are calculated for different class of nodes (generation of processors, accelerators, etc.) within the same site or different HPC systems, it provides a very detailed picture which allows the QCG middleware to easily select the optimal configuration based on the end-user's needs at the moment. The end-user can even combine and weigh the metrics in order to fine-tune the optimization process.

Going further, the predictions of such system could be refined if any theoretical performance model exists for some of the codes or submodels involved in the multiscale application. It is also worth mentioning that estimation of overall performance from submodel profiles is more straightforward in our case (ES pattern where all submodels are statically defined) compared to other patterns where the number of instances of a submodel could be dynamic (typically in the HMC pattern). Finally, as time to completion could depend heavily on the time spent by a given job waiting in the queue at a given location, an estimation of such queuing time would be very valuable in our system. As there are very few HPC centres that provide such service, a generic

¹⁰<http://www.lrz.de/services/compute/supermuc/>.

Figure 10. Normalized costs for the synchronous (solid line) and asynchronous (dotted line) workflows.

queue time prediction service [26] is being experimented as an addition to the ComPat stack. Information from this service would then be combined with the ideal T estimation to provide a more accurate and dynamical prediction for the time to completion.

5. Conclusion

The dynamics of fusion plasmas is a multiscale physics problem, for it involves a wide range of scales in time and space. While running a micro-scale turbulence code long enough to understand the large-scale overall transport is impractical, building a multiscale simulation has become the only viable approach. The ComPat project uses the existing MMSF and MUSCLE2 to construct a component-based multiscale model that falls into one of the three defined MCPs. In addition, software is developed to help users run their simulation optimally, according to their performance metrics of choice.

The fusion multiscale application presented in this paper takes the ComPat approach. Its current topology is a simple configuration yet produces reasonable physics results: it couples an equilibrium model (CHEASE), a gyrofluid flux-tube turbulence model (GEM), a module that translates fluxes into transport coefficients (IMP4DV) and a transport model (ETS) together. Such a configuration falls under the ES pattern, with GEM as the primary model and the rest as auxiliary models. The primary model requires the majority of computing resource and consumes the most energy, therefore having understanding on how the primary model performance affects the overall performance of the multiscale model can help in terms of building a more optimal workflow. The MAP profiling tools can integrate into a MUSCLE2 kernel to take measurements such as runtime, time spent in MUSCLE2 operation, time spent in MPI communications and energy consumption. The performances of the two topologies presented, one running all components synchronously and the other running the GEM and CHEASE in parallel, were studied using MAP on the SuperMUC. In particular, the cost functions of core numbers multiplied by time to completion, total energy consumed and time to completion (wall clock time) are compared between the synchronous and asynchronous versions of the application.

Overall, the asynchronous workflow is demonstrated to perform better than the synchronous workflow, while its theoretical gains would be even more interesting with a turbulence code that scales to a higher core count.

While these are initial results presented in this paper, this work will be continued on other machines and other node types to provide a broader picture on the performance of the different simulation configurations. These runs will then populate the performance database and help ComPat's Optimization service to calculate more accurately its cost functions and determine more optimal settings depending on the end-users constraints. In the future, the GEM model will be replaced by other more accurate (and considerably more expensive) gyrokinetic models (local and global). We will then explore the performance of these models and how that affects the workflow performance.

Data accessibility. To access the ComPat software, please visit <https://github.com/compat-project>.

Authors' contributions. O.O.L., O.H. and D.P.C. drafted the manuscript. O.O.L. carried out simulations discussed in this paper. O.O.L. and O.H. performed the data analysis. O.P. and K.B. provided their expertise in the MAP profiling tool and designed the benchmarks. T.P., P.K. and B.B. provided their expertise in the QCG middleware and MUSCLE2. A.B., B.D.S. and D.P.C. provided their expertise in the single scale models within the fusion multiscale framework.

Competing interests. We declare that we have no competing interests.

Funding. This project has received funding from the European Union's Horizon 2020 research and innovation programme for the ComPat project, under grant agreement no. 671564. This project is part of the FET-Future Emerging Technologies funding schema. Lastly, part of this work was supported by the National Science Centre (NCN) in Poland under the MAESTRO grant no. DEC-2013/08/A/ST6/00296.

Acknowledgements. The authors thank all the collaborators from the ComPat project. The authors also want to thank EUROfusion and the EUROfusion High Performance Computer (Marconi-Fusion) for the computing allocation, which made part of the results presented in this paper possible.

References

1. Groen D, Zasada SJ, Coveney PV. 2014 Survey of multiscale and multiphysics applications and communities. *Comput. Sci. Eng.* **16**, 34–43. (doi:10.1109/MCSE.2013.47)
2. Chopard B, Borgdorff J, Hoekstra AG. 2014 A framework for multi-scale modelling. *Phil. Trans. R. Soc. A* **372**, 20130378. (doi:10.1098/rsta.2013.0378)
3. Nishimura Y, Coster D, Scott B. 2004 Characterization of electrostatic turbulent fluxes in tokamak edge plasmas. *Phys. Plasmas* **11**, 115–124. (doi:10.1063/1.1631811)
4. LoDestro LL, Cohen BI, Cohen RH. 1991 Comparison of simulations and theory of low-frequency plasma turbulence. *Plasma Phys. Controlled Fusion* **2**, 31.
5. Shestakov AI, Cohen RH, Crotinger JA, LoDestro LL, Tarditi A, Xu XQ. 2003 Self-consistent modeling of turbulence and transport. *J. Comput. Phys.* **185**, 399–426. (doi:10.1016/S0021-9991(02)00063-3)
6. Barnes M, Abel IG, Dorland W, Görler T, Hammett GW, Jenko F. 2010 Direct multiscale coupling of a transport code to gyrokinetic turbulence codes. *Phys. Plasmas* **17**, 056109. (doi:10.1063/1.3323082)
7. Candy J, Holland C, Waltz RE, Fahey MR, Belli E. 2009 Tokamak profile prediction using direct gyrokinetic and neoclassical simulation. *Phys. Plasmas* **16**, 060704. (doi:10.1063/1.3167820)
8. Meneghini O *et al.* 2015 Integrated modeling applications for tokamak experiments with OMFIT. *Nucl. Fusion* **55**, 083008. (doi:10.1088/0029-5515/55/8/083008)
9. Becoulet A, Strand P, Wilson H, Romanelli M, Eriksson L, The Contributors to the European. 2007 The way towards thermonuclear fusion simulators. *Comput. Phys. Commun.* **177**, 55–59. (doi:10.1016/j.cpc.2007.02.051)
10. Falchetto GL *et al.* 2014 The European Integrated Tokamak Modelling (ITM) effort: achievements and first physics results. *Nucl. Fusion* **54**, 043018. (doi:10.1088/0029-5515/54/4/043018)
11. Imbeaux F *et al.* 2015 Design and first applications of the ITER integrated modelling and analysis suite. *Nucl. Fusion* **55**, 123006. (doi:10.1088/0029-5515/55/12/123006)
12. Alowayyed S, Groen D, Coveney PV, Hoekstra AG. 2017 Multiscale computing in the exascale era. *J. Comput. Sci.* **22**, 15–25 (doi:10.1016/j.jocs.2017.07.004)

13. Alowayyed S *et al.* 2018 Patterns for high performance multiscale computing. *Future Gener. Comput. Syst.* **91**, 335–346. (doi:10.1016/j.future.2018.08.045)
14. Falcone JL, Chopard B, Hoekstra AG. 2010 MML: towards a multiscale modeling language. *Procedia Comput. Sci.* **1**, 819–826. (doi:10.1016/j.procs.2010.04.089)
15. Borgdorff J, Mamonski M, Bosak B, Kurowski K, Belgacem MB, Chopard B, Groen D, Coveney PV, Hoekstra AG. 2014 Distributed multiscale computing with MUSCLE 2, the multiscale coupling library and environment. *J. Comput. Sci.* **5**, 719–731. (doi:10.1016/j.jocs.2014.04.004)
16. Bosak B, Kopta B, Kurowski K, Piontek T, Mamonski M. 2014 New QosCosGrid middleware capabilities and its integration with European e-infrastructure. In *eScience on distributed computing infrastructure*, pp. 34–53. Switzerland: Springer International Publishing.
17. Piontek T, Bosak B, Ciznicki M, Grabowski P, Kopta P, Kulczewski M, Szejnfeld D, Kurowski K. 2016 Development of science gateways using QCG - lessons learned from the deployment on large scale distributed and HPC infrastructures. *J. Grid Comput.* **14**, 559–573. (doi:10.1007/s10723-016-9384-9)
18. Hoenen O, Coster D, Petruczynik S, Plociennik M. 2015 Coupled simulations in plasma physics with the integrated plasma simulator platform. *Procedia Comput. Sci.* **51**, 1138–1147. (doi:10.1016/j.procs.2015.05.283)
19. Imbeaux F *et al.* 2010 A generic data structure for integrated modelling of tokamak physics and subsystems. *Comput. Phys. Commun.* **181**, 987–998. (doi:10.1016/j.cpc.2010.02.001)
20. Coster DP, Basiuk V, Pereverzev G, Kalupin D, Zagorksi R, Stankiewicz R, Huynh P, Imbeaux F. 2010 The European transport solver. *IEEE Trans. Plasma Sci.* **38**, 2085–2092. (doi:10.1109/TPS.2010.2056707)
21. Lütjens H, Bondeson A, Sauter O. 1996 The CHEASE code for toroidal MHD equilibria. *Comput. Phys. Commun.* **97**, 219–260. (doi:10.1016/0010-4655(96)00046-X)
22. Scott BD. 2005 Free-energy conservation in local gyrofluid models. *Phys. Plasmas* **12**, 102307. (doi:10.1063/1.2064968)
23. Scott BD. 2005 Dynamical alignment in three species tokamak edge turbulence. *Phys. Plasmas* **12**, 082305. (doi:10.1063/1.1993507)
24. Luk OO, Hoenen O, Bottino A, Scott BD, Coster DP. In press. ComPat framework for multiscale simulations applied to fusion plasmas. *Comput. Phys. Commun.* (doi:10.1016/j.cpc.2018.12.021)
25. Amdahl GM. 1967 Validity of the single processor approach to achieving large scale computing capabilities. In *Proc. of the Spring Joint Computer Conference, Atlantic City, NJ, 18–20 April*, pp. 483–485. New York, NY: ACM.
26. Jancauskas V, Piontek T, Kopta P, Bosak B. 2019 Predicting queue wait time probabilities for multi-scale computing. *Phil. Trans. R. Soc. A* **377**, 20180151. (doi:10.1098/rsta.2018.0151)