Coupled simulations in plasma physics with the Integrated Plasma Simulator platform

O. Hoenen¹, D. Coster¹, S. Petruczynik², and M. Plociennik²

¹ Max-Planck-Institut fuer Plasmaphysik, Boltzmannstr. 2, D-85748 Garching, Germany
{olivier.hoenen, david.coster}@ipp.mpg.de
² Poznan Supercomputing and Networking Center, IBCh PAS, Noskowskiego 12/14 Pozna, Poland
{marcinp, seba}@man.poznan.pl

Abstract
Simulations of fusion plasma obtained in a Tokamak device can involve a wide range of physics phenomena occurring at different scales. Programming such a simulation is challenging and tends to increase the complexity of the code and its maintenance. Many approaches are trying to alleviate this issue by coupling several single scale components, the complexity being moved from the physics code to the coupling and execution platform. In this paper we are presenting the Integrated Plasma Simulator (IPS) platform, its advantages for running efficiently coupled simulations for different plasma physics use cases and are briefly comparing it to other platforms used in the fusion community.

Keywords: simulation platform, code coupling, plasma physics

1 Introduction
Understanding and controlling instabilities in a magnetically confined plasma is of major importance on the road to the production of electricity through nuclear fusion. One major difficulty lies in the computational cost associated to the kinetic study of the micro-turbulence (in microseconds and millimeters) at the global confinement time (in seconds and meters), which exceeds the capabilities of current top-Tier computers. Solutions based on multiscale modelling or using a set of reduced models (using fluid, MHD description) can be used in order to alleviate the computational load. This comes nonetheless at the cost of more complex software development and maintenance.

To build such multiscale or multi-models application, an interesting approach [1] consists in coupling single scale components (where a scale can be either spatial, temporal or refer to a different physics or numeric model), where each single component is easier to develop, validate and maintain. Such coupling can rely on the usage of a data structure common to all
components, and possibly using a dedicated I/O library to access the data, as proposed by the EFDA ITM-TF task force [2, 3]. The coupled application is then described using a script based approach or controlled by a scientific workflow manager.

In such workflows, coupled components can be legacy codes, possibly running in parallel, at different core count and runtime, making its optimal execution on the targeted environment more complex. A solution to simplify the execution of such workflow consists in running the simulation platform within a regular parallel allocation in a single computer. The Integrated Plasma Simulator [4] is based on such principle: it runs in a single (possibly very large) computing allocation and handles internally the scheduling of load balancing of the tasks associated with each component, considering several layers of abstraction for the parallelism.

We have implemented within the IPS platform two fusion workflows with different computational needs: an acyclic (loose-coupling) chain composed of a high-resolution equilibrium reconstruction and its stability study, and a cyclic (tight-coupling) turbulence transport time evolution. The acyclic case involves a parameter scan for which the runtime of each run can differ a lot, whereas the cyclic case is composed of codes which have to be executed in sequence with different level of parallelism and computational cost. This contribution presents briefly the characteristics of the IPS platform and compares them to other platforms used by the fusion community in Europe. The layered implementation required to embed legacy codes coming from the ITM community into the IPS platform is presented for generic components. Finally, targeted cyclic and acyclic workflows are presented and their characteristics are discussed to highlight the benefits of using one or another platform in the different configurations.

2 Coupling and simulation platforms

Through the Integrated Tokamak Modelling activity (EFDA-ITM [2]), the European fusion research community spent a substantial amount of effort in order to develop a generic data structure [5] for the targeted integrated simulation platform. This data structure is organized as a set of Consistent Physical Objects (CPO) storing information and quantities for different parts of the physics. Code coupling is done through this common data structure: each code has to provide an interface using only CPO as input and output data, and a separate XML file containing code specific parameters. As a result, codes which share the same purpose are also sharing the same interface, so they are interchangeable and easier to compare in benchmarks and validation efforts. Two codes with different interfaces can be coupled together if at least one output CPO of one code is an input for the other code. Codes can use CPO directly, but they can also provide a wrapper which converts CPO data into the internal data representation of the code, which is especially interesting when dealing with legacy codes.

If codes are developed in the same programming language, they can exchange the CPO directly when they are coupled within the same program unit. When different programming languages are used, or when coupled codes are not executed within the same process, CPO data storage and retrieval are granted through the Universal Access Layer library (UAL [6]). The UAL can store CPO in different databases and file formats. Its API is provided in Fortran, C++, Java, Python and Matlab, increasing the range of codes that can be coupled within the same simulation. Describing and executing such coupled simulations, possibly involving parallel sections, MPI codes and codes written in different languages is not an easy task: it requires to develop complex scripts or to use a higher level simulation platform. Such platforms can be general purpose like the Swift [7] toolkit for workflow multiscale simulations, or domain specific. Interested reader can refer [8] which surveys general purpose multiscale simulation platforms used in astrophysics, biology, engineering, material sciences. Within the scope of this paper we
will focus on platforms used by the fusion community in Europe, as IPS, Kepler, MUSCLE and GridSpace. In the following we will describe briefly these tools and their features, summarized in Table 1.

### 2.1 IPS: Integrated Plasma Simulator

The Integrated Plasma Simulator (IPS\(^1\) [4]) is a Python framework for component coupling developed for the Center for Simulation of RF Wave Interactions with Magnetohydrodynamics (CSWIM) in the USA. The IPS framework provides an environment to couple multiple components concurrently or serially. The framework is used for multiphysics simulation of fusion plasmas, in the context of many-task computing. The IPS core framework consists of several services facilitating the management of the tasks (task manager, dispatching tasks) and re-

---

\(^1\)http://cswim.org/ips
sources (resource manager), communication (event service), data handling (data manager) and coupling of the components. It also provides fusion specific services for plasma evolution and state manager. A simulation in IPS is usually orchestrated by a special type of component called **driver**. The driver is interacting with core services, invoking components responsible for computational tasks named **worker**. The internal scheduling is done through a simple algorithm where the tasks are run in the order of appearance (first-hit filling rule). A worker component can launch multiple tasks executed concurrently, using blocking calls in order to synchronize all tasks. The driver can run the worker components concurrently, with blocking or non-blocking calls. All IPS components are then embedded within a single batch allocation.

Adding a new physics component in IPS consists in implementing a **worker** in Python to wrap the native code and a **driver** to pilot the simulation. More details are given in section 3.

### 2.2 Kepler scientific workflow manager

Kepler\(^2\) is a Java-based platform for helping scientists to design and execute complex workflows [9]. It supports foreign language codes through the Java Native Interface. It provides a rich graphical user interface to help designing the workflow, which can be stored and exchanged in XML format, and is then executed following a dataflow-oriented approach. It is extensible and provides a large range of generic and domain specific components, some of which are granting access to distributed computational resources.

Adding a new component in Kepler (called **actor**) corresponding to a physics code is done by the FC2K [10] semi-automated code generator: given a physics code compiled as a library, and the description of its inputs and outputs, FC2K generates the sources for a wrapper (Fortran or C++) to fetch data through the UAL library, and for the actor (Java) to be included in Kepler. Such actor can be executed directly from the JVM or as a separate program in batch, either interactively, using a batch queuing system or on remote HPC or Grid infrastructure [11, 12].

As Kepler is the standard tool to design and run integrated simulation in the ITM community, a large number of different physics and benchmark/validation workflows have been already developed using this platform [3]. The scheduling of the different actors is managed by a component called **director**, which is provided in Kepler with different versions in order to pilot from the simplest sequential and deterministic workflow to a complex workflow with branches and parallel sections.

### 2.3 The Multiscale Coupling Library and Environment

MUSCLE\(^3\) is a coupling library dedicated to multiscale applications [13] following a component-based approach. It provides its simple API in Java, C/C++, Fortran, Python, MATLAB and Scala, which supports developers while implementing their own components. Such a component can be parallel, using either MPI, OpenMP or threads. Coupling is described in an extensible way as a script in Ruby. The described coupled simulation is then executed in a runtime environment which deals transparently with inter-component communications, and can use TCP/IP to bridge several runtime environments in order to access distributed computing resources.

Adding a new component in MUSCLE (called **kernel**) requires to implement a simple source code which involves a time loop and calls to MUSCLE’s API. The basic API usage is very similar to MPI: init – send – receive – finalize and reading the value of different constants or variables shared in the simulation. Each component is then compiled into its own executable and linked.

\(^2\)https://kepler-project.org

\(^3\)http://www.qoscosgrid.org/trac/muscle
against the MUSCLE library. Send and receive operations in MUSCLE are involving simple
data types only, thus the physics codes are called from a wrapper which serialize/deserialize
the CPO data, which is then transferred by MUSCLE as a byte buffer [14].

2.4 GridSpace virtual laboratory

GridSpace\(^4\) is a web-based platform for running and monitoring executable code snippets [15]
using different interpreters (to run the code) and executors (to access distributed computing
resources). It has been extended to support complex control flow by allowing nested snippets
(sub-snippets). Additional tools can be used to help the developer to register semantic information about components (MaMe) and compose the coupled application through a web-based
graphical application (MAD).

For each added C++ or Fortran codes, an executable wrapper is created for reading input
CPO, XML input parameters and writing output CPO with files. In GridSpace, such programs
are declared as interpreters, taking the code specific parameter XML file as the snippet to
be executed. The master snippet (embedding one or more sub-snippets) can be implemented
as a simple shell script and interpreted in bash for instance. The user can then decide on
which machine each snippet will be run by associating the executor accordingly. This has been
demonstrated using resources of the PL-Grid Infrastructure [16].

3 Layered generic component implementation

Building a complex simulation from a set of coupled codes has numerous advantages, such
as code reusability, ease of validation against others, extensibility and ease of maintenance.
Nevertheless, it can also be considered as a burden if it requires developing a specific version
of the physics code, tied to a specific framework without any guarantee about its long term
maintenance. In order to reduce the amount of extra work on the developer, we followed a non-
intrusive approach based on layers of wrapper codes as shown in Figure 1. Following this onion
peel design, changing the common datastructure to transfer data between components requires
modifying only the \textit{Data wrapper}, changing the coupling and execution platform requires to
modify the \textit{Coupling wrapper} and to adapt the implementation of the \textit{Simulation workflow}. Of course, modifying the internals of the physics solver is always possible and only requires
to modify the \textit{Native code}, as long as its signature remains identical. Once all wrappers are
implemented, several use cases can be investigated by modifying only the \textit{Simulation workflow},
its parameters and its logic.

In our specific case, native codes correspond to the implementation in Fortran of different
physics models, while the data wrappers are converting the native codes internal data represent-
ation into the common CPO data structures (derived types in Fortran). These two layers
are under the responsibility of the physics code developers, as deep knowledge of the physics
codes is required to understand and translate CPO data into another representation. The
implementation of the coupling wrapper necessitates a good knowledge of the coupling platform
and its API, but it usually requires only an overview of the underlying physics layers (e.g the
type of code and its signature). Consequently, implementation of this layer can be generated
automatically or semi-automatically. At last, the simulation workflow is build by a modeller
with good understanding of all underlying physics as well as knowledge of the coupling platform
capabilities. Usage of high-level graphical tools, as offered by Kepler or MAD tools, tends to
reduce the level of expertise required for designing a new simulation from scratch.

\(^4\)http://dice.cyfronet.pl/products/gridspace
In the context of the IPS platform, the last two external layers (colored in blue in Figure 1) are IPS specific. The coupling wrapper is described by a worker component: it is made of a Python script which calls an executable version of the data wrapper developed in Fortran and storing CPO to files. The worker is responsible for the execution of the codes (using task manager), and data handling (using data manager). The worker methods to be implemented are usually initialisation, step evolution, finalization. The simulation workflow is implemented as a driver component, which defines in Python the configuration files, shared variables, and uses the core services resource manager to map the computing resources and pilot the worker components by activating their step methods.

4 Loosely coupled use-case

A coupled simulation is said loosely-coupled when it can be expressed as an acyclic graph or a workflow without feedback loops. The absence of feedback loop puts less emphasis on performance of data exchange between coupled components. The so-called stability chain is a multi-physics loosely-coupled case which consists in the following two steps:

1. reconstruction of a high-resolution equilibrium from a low-resolution equilibrium coming from experimental data. This is done with the fixed-boundary equilibrium code HELENA, which is a serial code implemented in Fortran 90.

2. study the stability of the considered configuration of the plasma using the linear Magnetohydrodynamics (MHD) code ILSA, which is implemented in Fortran 90 and executed sequentially.

In order to optimize the stability of the plasma configuration, the \( j - \alpha \) scan workflow performs several stability chains, where each chain considers a modified equilibrium with pressure \( (p) \) and current \( (j) \) profiles scaled from an initial one. By exploring for instance 10 different scaling for each \( p \) and \( j \) around the initial value, we obtain a total of 121 independent stability chains that can be executed in parallel as shown in Figure 2. This is a typical use-case for a coupled simulation made of serial components where all the parallelism lies at the workflow level.

Implementation of such workflow is straightforward, using for instance a simple script (bash, IDL or any other language) to automatize the submission of each chain as a separate job to the queuing system of a parallel computing environment. Nevertheless, such direct implementation has a major drawback: depending and the batch queuing system configuration (limitation of the number of jobs executed at the same time, priorities) and the load of the computer, some of the jobs can spend a significant amount of time waiting in the queue, delaying the final results.
In order to reduce queuing time or increase the number of serial jobs allowed at the same time, such jobs can be submitted to a wider distributed environment by using for instance a Grid infrastructure through middleware capabilities embedded in either Kepler or GridSpace platforms. The drawback of such approach is an increasing complexity for administrating the platform (installing and updating the required tools, configuring firewalls, managing credentials) and the need for users to obtain and renew periodically the associated grid certificates.

Another solution consists in submitting the workflow to a single parallel computing allocation. The drawback of such approach is linked to potentially very heterogeneous computing costs associated with ILSA runs: depending on the stability of the studied configuration and on chosen numerical parameters, ILSA’s runtime can be from short (unstable cases) to very long (more then $10 \times$ slower for the most stable cases). Such discrepancies will result in wasting computing time, as allocated processors will remain in idle state after the most unstable cases have finished.

IPS is submitted as a single parallel computing allocation and therefore usually reduces the overall queuing time. In addition it exploits more efficiently the available resources by having its own internal scheduler, which distributes and balances dynamically the computing resources to the pool of tasks present in the simulation. Of course, minimal wall-clock time can always be obtain by using a 121 cores allocation, but this will waste some computing time if the $j - \alpha$ scan has heterogeneous stability. Given a posteriori runtime for each chain, the minimum size of the computing allocation can be computed, as well as the optimal scheduling of tasks. Sub-optimal implementation can use heuristics to choose the size of the computing allocation (for instance given a user’s estimate for ratio and position of stable and unstable regions) and schedule the tasks (for instance following a stable case first approach).

A last point to be taken into consideration is the memory usage of each component, as ILSA runs can have an important memory footprint (some cases can require up to 12 GB by run). This can be an issue in many systems, where the amount of memory by node is not sufficient to use all available cores. Unfortunately IPS does not provide at the moment a mechanism to schedule and balance tasks by taking into account their foreseen memory consumption.

5 Tightly coupled use-case

A coupled simulation which can be expressed as a cyclic graph or a workflow with presence of feedback loops is called tightly-coupled. Typical examples are simulations with time evolution, convergence loops or any kind of loop where consecutive iterations are not independent. Due to
the high frequency of data exchange, such simulations are usually well suited to platforms which favor fast data transfer. The Turbulence–Transport–Equilibrium workflow (TTE) corresponds to such use-case: it simulates the time evolution of 1D profiles (electron and ion temperature and density) in the core plasma under the influence of anomalous transport coefficients computed by a 3D turbulence code, with periodic 2D equilibrium reconstruction. This is a multiscale simulation with temporal scales varying from the macro level ($\sim s$) for the transport code to the micro level for the turbulence code ($\sim \mu s$), and CPO data are exchanged back and forth as shown in Figure 3.

Selected codes, all written in Fortran, are ETS (transport solver), BDSEQ (circular equilibrium approximation) and GEM [17] (electromagnetic gyro-fluid turbulence). The turbulence code is expected to be parallel. In the case of GEM, it is developed with MPI and uses a flux tube approximation: for small test cases, 8 flux tubes are positioned around the magnetic field lines, with each flux tube running on 64 cores. In such configuration, GEM uses a total of 256 cores. As the data flow imposes synchronized execution of components in the workflow, all the parallelism in this simulation lies at the component level, within the GEM code.

For such type of tightly-coupled workflow, execution in a distributed environment will usually penalize performance, unless the total amount of computing power for all components exceeds the capability of one supercomputer, or a component is optimized for very specific hardware. On the contrary, a platform such as MUSCLE, which allows inter-component data transfer using fast communications, can be well-suited. But in that case, a potential issue is raised in our application due to the complexity of the CPO objects exchanged between components: sending a CPO object made of hundreds of different simple type fields requires either many simple data type messages, or an additional step for serializing and deserializing the object. Both solutions generate overheads which reduce the performance gap between these platforms (such as MUSCLE) and platforms using files for data exchange (such as IPS or Kepler).

![Figure 3: Time dependent Turbulence–Transport–Equilibrium workflow.](image)

![Figure 4: Timings for $j - \alpha$ parameter scan](image)

### 6 Preliminary results

**Loosely-coupled case** The $j - \alpha$ simulation presented in section 4 has been implemented in IPS using a single allocation of cores and run on Chimera, a SGI Altix UV1000 Tier-1 supercomputer (PSNC, Poland) with 16 TB of shared RAM (ccNUMA). Validation runs were using 121 cores allocations resulting in obtaining the minimum wall-clock time for the simulation,
which in our test case was almost 13 hours. As expected, this time corresponds to most stable regions whereas the most unstable regions are computed in a bit more than 1 hour, as shown in Figure 4. In case physicists can predict which region and fraction of this diagram will be stable for a given equilibrium, we can then order the runs from longest to shortest to reduce considerably the amount of cores required to obtain the result within the same overall time.

**Tightly-coupled case**  The TTE simulation presented in section 5 has been implemented for both IPS and MUSCLE platforms. In each case we are using the exact same native Code and data wrapper as introduced in Figure 1. In IPS, the coupling wrapper handles directly CPO stored in files whereas in MUSCLE it sends and receives (non-optimally) serialized CPO. In both cases, the simulation workflow has a macro scale time step of 0.01 and evolves 1 second of physical time, thus calling 100 times each component.

Both simulation were executed on Helios, a Tier-0 supercomputer (IFERC, Japan) dedicated to the fusion community, using 16 nodes connected with InfiniBand and made of 2 Xeon E5 processors (Sandy-Bridge, 16 cores and 64 GB of RAM). The wallclock obtained with IPS and MUSCLE is very similar (10s difference on a 5.5 hours run [18]), which exhibits in such simulation the importance of the serialization overhead for complex objects.

7 Conclusion

In this paper we have proposed an overview of the different component coupling and execution platforms used by the Integrated Modelling community for fusion research in Europe and presented a layered approach for component implementation which is non-intrusive, allows code reuse, and is platform-agnostic.

We have implemented two cases representing different parallel paradigm (workflow parallelism and component parallelism) within IPS. Results obtained in different computers demonstrates ease of deployment (even on Tier-0 computers) and reduced overhead allowing to reach performances equivalent to tightly-coupled platforms in some cases. The rich features of IPS concerning the parallelization and its internal scheduler are clearly of advantage and we have shown how it could be used in principle to minimize the size of a computing allocation without affecting wall-clock time. Unfortunately the scheduling algorithm is rather simple and at the moment can not take into account the memory consumptions, which is critical for some codes.

Future works will focus on finding semi-automatically optimal allocation size for $j - \alpha$ like use-cases, on the development of hybrid use-cases (parallelization at component and workflow levels) as well as making a more in-depth benchmark comparing the performance of the different platforms. Other interesting platforms such as EFFIS [19] or OMFIT [20], developed within similar efforts in the USA, might also be investigated.

Acknowledgments

This work has been carried out within the framework of the EUROPshen Consortium and has received funding from the Euratom research and training programme 2014-2018 under grant agreement No 633053. The views and opinions expressed herein do not necessarily reflect those of the European Commission. The research presented in this contribution was partially supported by the MAPPER project, which receives funding from EUs FP7 (2007-2013) under grant agreement No RI-261507. A part of this work was carried out using the HELIOS supercomputer system at Computational Simulation Centre of International Fusion Energy Research Centre.
(IFERC-CSC), Aomori, Japan, under the Broader Approach collaboration between Euratom and Japan, implemented by Fusion for Energy and JAEA. The authors would like to thank the IPS developer team, especially Wael R. Elwasif for providing support throughout this project.

References

[1] J. Borgdorff et al. Foundations of distributed multiscale computing: Formalization, specification, and analysis. *Journal of Parallel and Distributed Computing*, 73(4):465–483, 2013.

[2] EFDA-ITM Task Force. [http://www.efda-itm.eu](http://www.efda-itm.eu).

[3] G.L. Falchetto et al. The european integrated tokamak modelling (ITM) effort: achievements and first physics results. *Nuclear Fusion*, 54(4):043018, 2014.

[4] W.R. Elwasif et al. The design and implementation of the swim integrated plasma simulator. In *Parallel, Distributed and Network-Based Processing (PDP)*, 2010, pages 419–427, Feb 2010.

[5] F. Imbeaux et al. A generic data structure for integrated modelling of tokamak physics and subsystems. *Computer Physics Communications*, 181(6):987–998, 2010.

[6] G. Manduchi et al. A universal access layer for the integrated tokamak modelling task force. *Fusion Engineering and Design*, 83(23):462–466, 2008.

[7] M. Wilde et al.

[8] D. Groen et al. Survey of multiscale and multiphysics applications and communities. *Computing in Science Engineering*, 16(2):34–43, Mar 2014.

[9] I. Altintas et al. Kepler: an extensible system for design and execution of scientific workflows. In *Scientific and Statistical Database Management, 2004. Proceedings.*, pages 423–424, June 2004.

[10] M. Plociennik et al. Tools, methods and services enhancing the usage of the kepler-based scientific workflow framework. In *International Conference on Computational Science*, volume 29, pages 1733–1744, 2014.

[11] M. Plociennik et al. Approaches to distributed execution of scientific workflows in kepler. *Fundamenta Informaticae*, 128(3):281–302, 2013.

[12] M. Plociennik, et al. High level tools for fusion simulations workflows in distributed computing environment. In *High Performance Computing and Simulation (HPCS)*, pages 602–608, July 2012.

[13] J. Borgdorff et al. Distributed multiscale computing with MUSCLE 2, the multiscale coupling library and environment. *Journal of Computational Science*, 5(5):719–731, 2014.

[14] O. Hoenen et al. Designing and running turbulence transport simulations using a distributed multiscale computing approach. In *40th European Physical Society Conference on Plasma Physics*, 2013.

[15] E. Ciepiela et al. Managing entire lifecycles of e-science applications in the gridspace2 virtual laboratory from motivation through idea to operable web-accessible environment built on top of pl-grid e-infrastructure. In *Building a National Distributed e-Infrastructure PL-Grid*, volume 7136 of *Lecture Notes in Computer Science*, pages 228–239. 2012.

[16] K. Rycerz et al. Enabling multiscale fusion simulations on distributed computing resources. In *eScience on Distributed Computing Infrastructure*, volume 8500 of *LNCS*, pages 195–210. 2014.

[17] Bruce D. Scott. Free-energy conservation in local gyrofluid models. *Physics of Plasmas (1994-present)*, 12(10), 2005.

[18] S. Petrucznik et al. Comparison of IPS and muscle simulation platforms in the context of selected nuclear fusion coupled-application. In *CGW Workshop 2014*, 2014.

[19] J. Cummings et al. Effis: An end-to-end framework for fusion integrated simulation. In *Parallel, Distributed and Network-Based Processing (PDP)*, 2010, pages 428–434, Feb 2010.

[20] O. Meneghini and L. Lao. Integrated modeling of tokamak experiments with omfit. *Plasma and Fusion Research*, 8:2403009, 2003.