



Grant agreement no. 823712

## CompBioMed2

**Research and Innovation Action**

H2020-INFRAEDI-2018-1

Topic: Centres of Excellence in computing applications

### D5.2 Advanced Report on Biomedical Applications

Work Package: 5

Due date of deliverable: Month 26

Actual submission date: 30th November 2021

Start date of project: 01 October 2019 Duration: 48 months

Lead beneficiary for this deliverable: BSC

Contributors: USFD, UVA, BSC, UNIBO, UCL, UPF, CBK, UOXF

#### Disclaimer

This document's contents are not intended to replace consultation of any applicable legal sources or the necessary advice of a legal expert, where appropriate. All information in this document is provided "as is" and no guarantee or warranty is given that the information is fit for any particular purpose. The user, therefore, uses the information at its sole risk and liability.

For the avoidance of all doubts, the European Commission has no liability in respect of this document, which is merely representing the authors' view.

Project co-funded by the European Commission within the H2020 Programme (2014-2020)		
Dissemination Level		
PU	Public	YES
CO	Confidential, only for members of the consortium (including the Commission Services)	
CI	Classified, as referred to in Commission Decision 2001/844/EC	

PU	Page 1	Version 1.1
This project has received funding from the European Union's Horizon 2020 research and innovation programme under the Grant Agreement No 823712		



## Table of Contents

1	Version Log	3
2	Contributors	3
3	Definition and Acronyms	4
4	Public Summary	6
5	Introduction	6
6	Update on the Incubation Plan components	7
6.1	Innovation Management	7
6.1.1	IP Register	7
6.1.2	Incubator/Accelerator Register	7
6.1.3	External Expert Advisory Board (EEAB)	7
6.2	Stakeholder engagement	8
6.3	Enhancement of software/service offering	8
6.4	Technical aspects	8
	BSC – UOXF – UNIBO: Alya	8
	UPF: TorchMD	13
	ACELLERA: PlayMolecule	13
	USFD: pFIRE	15
	USFD: openBF	17
	UvA: HemoCell	18
	UCL: TIES	19
7	Conclusions	23

## 1 Version Log

Version	Date	Released by	Nature of Change
V0.1	30/09/2021	Mariano Vázquez	BSC Outline document, overall sections defined
V0.2	22/11/2021	Mariano Vazquez	Corrected comments from the internal reviewers
V1.0	30/11/2021	Emily Lumley	Final Draft, submitted to the EC
V1.1	17/02/2023	Hugh Martin	Updated version following M36 review

## 2 Contributors

Name	Institution	Role
Mariano Vázquez	BSC	Principal Author
Damien Dosimont	BSC	Contributor
Nick Laver	CBK	Contributor
Daniele Tartarini	USFD	Contributor
Ivan Benemerito	USFD	Contributor
Mateusz K Bieniek	UCL	Contributor
Alexander D Wade	UCL	Contributor
Max van der Kolk	UvA	Contributor
Raimondas Galvelis	Acellera	Contributor
Adrià Pérez	UPF	Contributor
Roberta de Michele	UNIBO	Reviewer
Vicente Grau	UOXF	Reviewer
Peter Coveney	UCL	Reviewer
Emily Lumley	UCL	Reviewer

### 3 Definition and Acronyms

Acronyms	Definitions
API	Application Programming Interface
BAC	Binding Affinity Calculator
BSC	Barcelona Supercomputing Centre
CI/CD	Continuous Integration/Continuous Development
CLI	Command Line Interface
CoE	Centre of Excellence
CPU	Central Processing Unit
DICE	Data Infrastructure Capacity for EOSC
EEAB	External Expert Advisory board
EU	European Union
GNN	Graph Neural Network
GPU	Graphics Processing Unit
GUI	Graphical User Interface
HPC	High Performance Computing
HPDA	High Performance Data Analytics
HTTPS	Hypertext Transfer Protocol Secure
IAB	Innovation Advisory Board
IP	Intellectual Property
LEXIS	Large-scale Execution for Industry and Society
MD	Molecular Dynamic
MPI	Message Passing Interface
pFIRE	parallel Framework for Image Registration
RBFE	Relative Binding Free Energy
SaaS	Software as a Service
SME	Small and Medium Enterprise
TIES	Thermodynamic Integration with Enhanced Sampling
UCL	University College London
UOXF	University of Oxford
UPF	Universitat Pompeu Fabra
USFD	University of Sheffield

UvA	University of Amsterdam
VCS	Version Control System
VPN	Virtual Private Network
WP	Work Package



## 4 Public Summary

---

CompBioMed is concerned with the development of computational techniques and research software applications in the computational biomedicine domain, with particular focus on the use of High Performance Computing (HPC) to support this research. The Centre of Excellence (CoE) acts as a hub of best practice for the community, facilitated by a programme of dissemination and engagement with a broad range of stakeholders from academia, clinical practice and industry.

WP5, Incubator Applications, deals with promoting application usage and integration with modern e-infrastructures, such as current petascale computers and high-performance clouds, but always focusing on hardening and polishing codes which have the potential to be used by the community in future exascale computers. It is worth to remark that WP5 is a work package in which CompBioMed2 reports enhancement on documentation, development pipelines, deployment and easiness-to-use even in a way that it will also cover the future exascale world.

The sustainability of the software and services developed by the CoE is demonstrated by how the users exploit the software generated. Improving both access to applications and their usability can be achieved through a process of software incubation, informed by end-user needs. Dissemination of best practice in software incubation also has the potential to provide benefit to code developers beyond the CoE partners, for application to other community software tools. The actions carried out in this WP are aligned with the objective of preparing our codes for extended use in the exascale era.

This deliverable reports the mid-term advances in incubation strategies for the CompBioMed2 codes.

## 5 Introduction

---

Software Incubation has a strong impact on the sustainability of any Centre of Excellence. Incubation activity is focussed on broadening the impact of the computational solutions developed by the CoE and promoting best practices within the biomedical research community. In particular, WP5 deals with everything necessary to make our software more reliable, robust, and usable, with a special focus on large-scale systems, towards exascale. Our strategy of Software Incubation is designed to enhance the potential of the CoE to provide services to stakeholders with interests in biomedical HPC applications. In D5.1 we defined an incubation plan for CompBioMed2, which is implemented transversally through all the work packages. In WP5 we deal with the technical aspects which cut through the different tasks, doing all that is needed to ensure: code usability, application hardening, preparing content for external use and commercialization, high performance data analytics (HPDA), model integration and containers for cloud computing.

## 6 Update on the Incubation Plan components

This section describes what we have done to update the different aspects of the Incubation Plan. Firstly, Innovation Management is addressed regarding the registry and the status of the EEAB. Next, there follows a short update of the Stakeholder Engagement status. We conclude the section with a brief description of the advances on the Enhancement of Software/Service offering of CompBioMed2.

### 6.1 Innovation Management

#### 6.1.1 IP Register

In the first phase of the CompBioMed CoE (2016-2019), a formal Intellectual Property (IP) register was created, which recorded background, foreground (results), and related third party intellectual property across the project. The IP Register continues in CompBioMed 2 and is available centrally on the CompBioMed intranet. It supports IP owners to record results, applications and outputs for incubation and exploitation, and promotes the consideration of the IP status as these results evolve. In CompBioMed2 the register is maintained and updated within Task 5.3 in WP5 by UCL with the support of BSC.

#### 6.1.2 Incubator/Accelerator Register

In the first phase of the CoE (CompBioMed1), a Central Incubator Register which lists EU innovation incubators and accelerators was compiled and is publicly available at <https://www.compbiomed.eu/services/central-incubator-registry/>. Incubators allow academic and industrial partners to collaborate to exploit HPC and associated e-infrastructure by raising awareness in industry, especially in SMEs, by making available and providing support for the use of cutting edge HPC facilities. The incubator register is particularly useful in the final stage of the innovation process, in supporting the exploitation planning and subsequently, where appropriate, coordinating the entry of innovation candidates into incubator environments that are suited to their specific exploitation needs. These resources are harnessed as part of the activities in Task 5.3.

#### 6.1.3 External Expert Advisory Board (EEAB)

The External Expert Advisory Board (EEAB) is a group of individuals representing a selection of our industry partners, academic and clinical practitioners for the purpose of offering advice and support on a wide range of issues relevant to all activities in the project. It is chaired by the Project Coordinator with members appointed by the General Assembly. It is a reconstruction of the Innovation Advisory Board (IAB), which was created during in the first iteration of the CoE, which includes clinicians and other relevant experts who joined since the project began.

A major role of the EEAB, which has persisted through both iterations of the CoE, is its role as an advisory resource for issues that arise in innovation, collaboration, dissemination and exploitation. This board advises on planned incubation activities, offering valuable perspectives from the variety of industry sectors involved. New members have been appointed as the project has evolved, ensuring a cohort of innovation focused members within the EEAB along with other expertise throughout the Centre. Task 1.7 has overseen the nomination of new members, who were then formally voted onto the board by the General Assembly.

## 6.2 Stakeholder engagement

The Incubation activities undertaken focus on broadening the impact of the computational solutions developed by the CoE and promoting best practices within the biomedical research community. CompBioMed2 has facilitated these interactions through the innovation and community activities in WP6 (see D6.1 published in M18), along with other EU funded projects in this space, within the *In Silico* World initiative, such as the Scalability<sup>1</sup> or the Good Simulation Practice<sup>2</sup> Channels in Slack. Additional stakeholder engagement has been achieved through the continued dissemination of training materials through a variety of channels, including the CoE's Training Portal and Repository, the compbiomed.eu website, the e-Seminar series, which has grown throughout the pandemic, and open-source repositories such as GitHub, and the established Helpdesk, to track all support requests from potential clients, associate partners, etc., and the creation of general compbiomed.eu e-mail addresses (see Deliverable D4.1). Additionally, the CoE held an online conference 15-17<sup>th</sup> September 2021 which attracted around 200 participants and 50 speakers across a range of session and topics.

## 6.3 Enhancement of software/service offering

There are several new products and services developed within CompBioMed2. This includes new data transfer infrastructures, for instance those developed in collaboration with DICE (Data Infrastructure Capacity for EOSC), or with LEXIS (Large-scale Execution for Industry and Society), for the facilitation of intense data transfer across HPC centres and during data staging. We developed an automated benchmark environment for the testing and execution of code, analysis and postprocessing and code scalability support as a service to increase the quality of code output. On training and dissemination, we have new course materials for the introduction of student and professional medics to high performance computing and *in silico* research; and an e-Seminar series with 18 instalments to date (as of October 6 2021) covering a wide range of topics in computational biomedicine<sup>3</sup>.

## 6.4 Technical aspects

We will go through the technical aspects of the different partners' efforts reported in this Work Package.

### BSC – UOXF – UNIBO: Alya

One of the most important tasks that BSC is performing in CompBioMed2 facing application hardening and usage enhancement was to radically transform the Alya development strategy. Currently, Alya is hosted on *gitlab.com* under the form of a git repository. Git is a free and open-source distributed version control system, being available since 2006 when it was created by Linus Torvalds, the same person behind Linux. Between 2006 and the present, Git has become the dominant version control system (VCS) in software development, used today by millions of developers. On top of that, quoting from its webpage "GitLab is a complete DevOps platform, delivered as a single application, fundamentally changing the way Development, Security, and Ops teams collaborate and build software". Besides hosting the repository, GitLab proposes a web-based graphical interface and all the DevOps features that enable efficient teamwork.

<sup>1</sup> <https://www.compbiomed.eu/compbiomed-scalability-channel/>

<sup>2</sup> <https://insilico.world/community/good-simulation-practice/>

<sup>3</sup> <https://www.compbiomed.eu/events-2/>



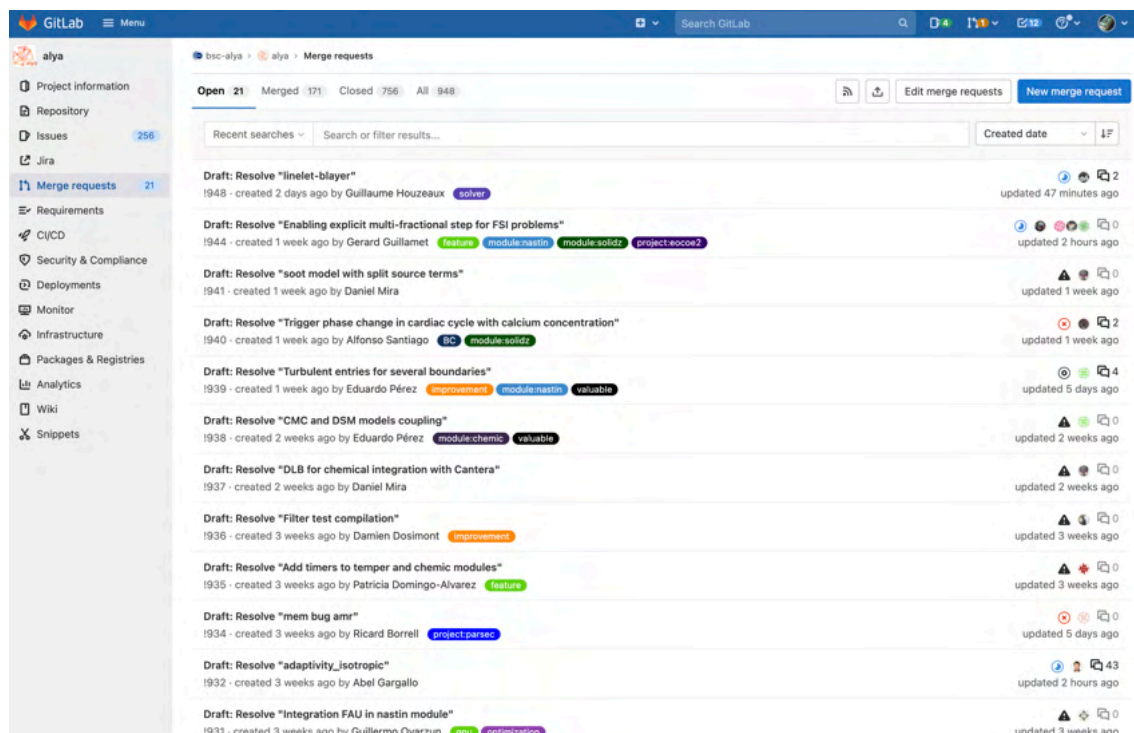


Figure 1. Screenshot of the Alya GitLab CI/CD pipeline.

In GitLab, Alya follows a development workflow based on continuous integration - continuous development/deployment (CI/CD). This practice is widely used in the software industry and the open-source DevOps community<sup>4,5</sup>, and has been recently adopted by the HPC community<sup>6</sup>. It allows the programmers to structure and speed up the development, automate the testing to prevent bugs from being introduced, guarantee code reliability, promote its portability, ensure the stability of the code on various platforms, as well as keep track of the code performance.

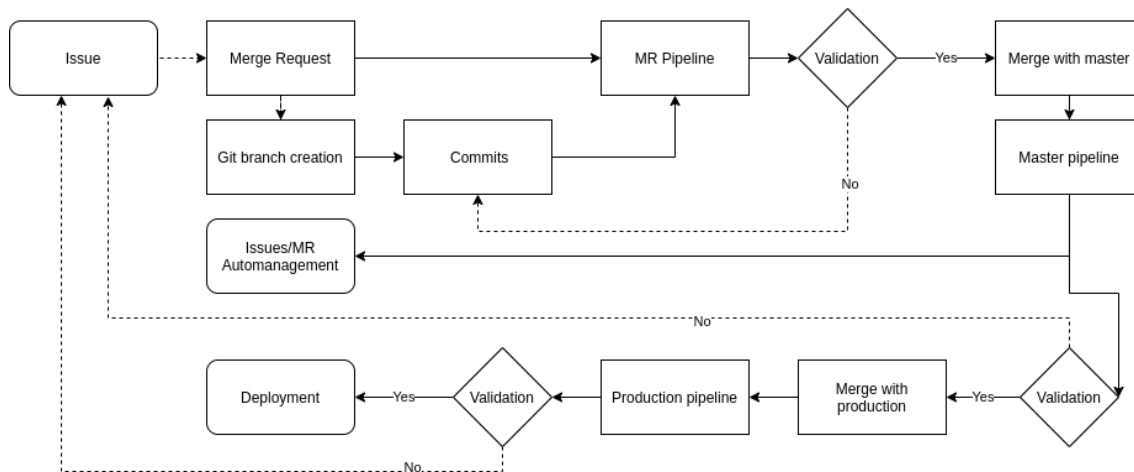


Figure 2. Scheme of the Alya's CI/CD pipeline.

<sup>4</sup> [Shahin, Ali Babar, et Zhu, « Continuous Integration, Delivery and Deployment »](#)

<sup>5</sup> [Sampedro, Holt, et Hauser, « Continuous Integration and Delivery for HPC ».](#)

<sup>6</sup> [Sharif, Janto, et Lueckemeyer, « COaaS ».](#)

Being one of the modelling codes which is clearly on the road to the exascale realm, Alya must have a specifically designed CICD pipeline. The Alya CICD workflow has currently 60-80 users/developers at different institutions, which has created more than 1100 issues. Notably, CompBioMed2's partner UOXF has been its first beta-tester, helping BSC to improve the user experience. Currently, it has more than 400 regression and unit tests and more than 20 benchmark tests. So far, we have more than 14,000 commits to the Master pipeline and the 12 open and available source forks we have created.

**Test Suite.** The testsuite is a program written in Fortran that builds Alya with several compilers (INTEL, GNU, PGI and XL) and various options on MareNostrum IV and power9 (29 builds), as well as 353 regression tests for each build, for which the outputs are compared with reference data. The testsuite guarantees the stability of the code for each build, as well as the validity of the output, and thus the physical equations written in the code. We want to remark that one of the builds of the test suite is an Alya version *containerized using Singularity*, to ensure the sustainability of the containerized version. It is executed each time a merge request is about to be validated.

**Benchmark Suite.** The benchmark suite which is run in the master pipeline is also managed by the testsuite program. In this case, the test cases executed are monitored and the duration of the different parts of the code, as well as various metrics such as the memory consumption or the computation and communication efficiency, are retrieved and stored in a database. This can provide important information not only to track time changes in the code efficiency, but also with *co-design* purposes, by monitoring code performance under the different operational conditions of the cluster. A web interface, *rooster*, enables such data to be visualized and the variation of such metrics at each commit to be analyzed, as well as comparing the build performance between them. The Benchmark Suite is critical to ensure performance stability, progressively more critical as the size of the system grows, with its culmen at the exascale.

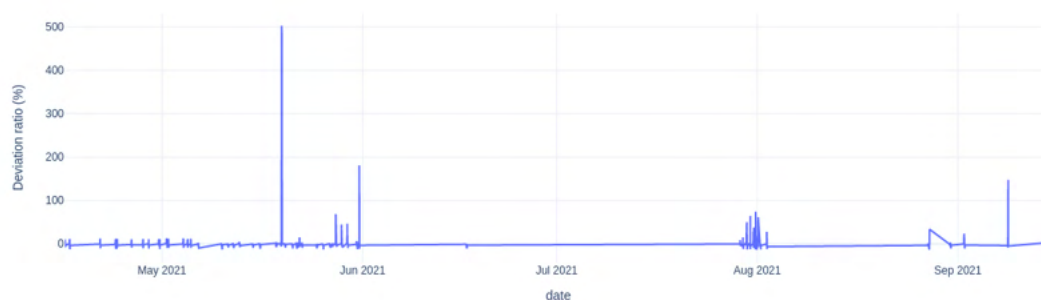
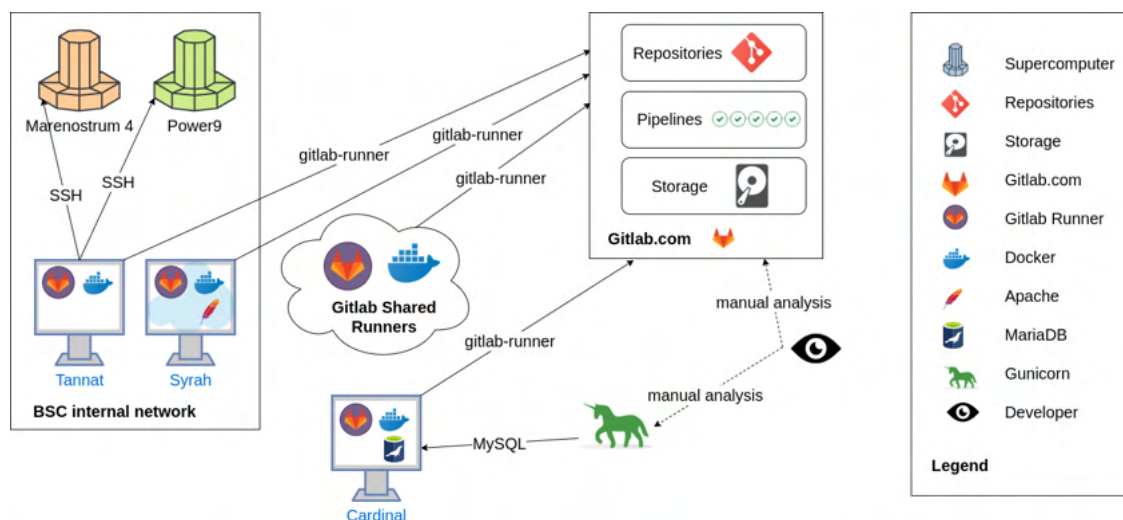


Figure 3. Graph produced by rooster highlighting the test cases execution time variation. The outliers are related to issues with MareNostrum IV (temperature or gpfs issues).



Figure 4. Other outputs produced by rooster, showing the evolution of the execution time for intel i4 vs the date, as well as the details of the duration of each code part, and a comparison with other builds.

**Architecture.** To execute the whole pipeline, we have architected our system as follows:



- Gitlab.com is responsible for the storage of the repositories, the execution of the pipelines, and the storage of the artifacts, which are files generated by the job execution that can be transferred between jobs or between pipelines.
- Pipeline jobs are executed by gitlab-runner, which is a service. Gitlab.com provides free runners, but we also have three machines running the jobs: two of them are located within the BSC network, and another one, which is outside.
- The testsuite, benchmark and deployment jobs run mandatorily on one of our systems within the BSC network, because they require access to MareNostrum IV and Power9.



the two supercomputers we use in production. As time evolves, we can setup the tests on a different architecture, of different size and features.

- The testsuite results are stored , being accessible through Apache as web pages.
- The database containing the benchmark results is hosted on a different system within the BSC network.
- Rooster, the graphical interface to visualize the benchmark results, runs through a guinicorn server hosted on the SaaS heroku.

In parallel to this effort, BSC and UOXF have been working on testing the new Alya MultiMesh capacity for coupling the electromechanical problems, reporting in WP5 the effort of improving the input format files and the restarting strategy. The MultiMesh strategy is critical for using the proper amount of resources according to the physics of the individual part on a coupled run, specially in the largest systems.

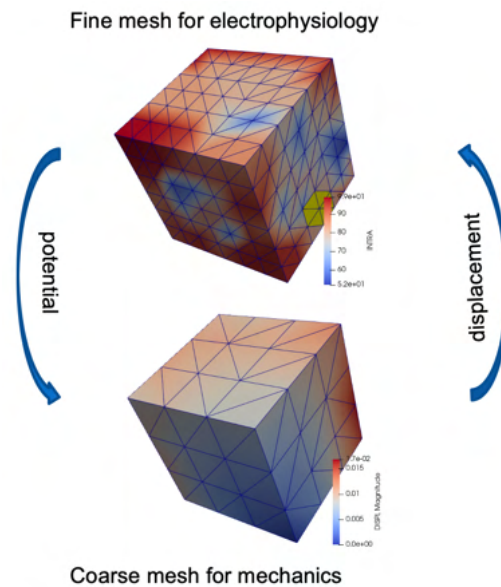


Figure 5. Alya MultiMesh coupling strategy.

Finally, BSC and UNIBO are refactoring BoneStrength application to integrate Alya following this strategy:

- UNIBO will use it deployed in MareNostrum IV
- Virtual Population runs will be managed using either Dakota or VECMA Tools, both of them already tested

Deployment required to compile and install the different libraries in MareNostrum, also creating several Python scripts to manage the patient's definition and data gathering.

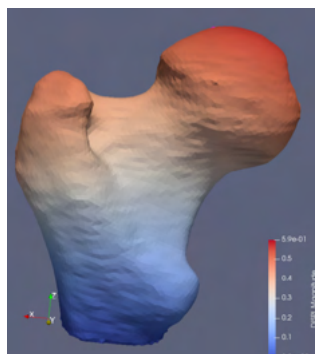


Figure 6. BSC and UNIBO are refactoring UNIBO's BoneStrength.

### UPF: TorchMD

UPF has been focused on the continuous development of TorchMD, our software for the development and usage of machine learning derived potentials for molecular simulations. TorchMD is divided into three different parts/applications, with their own code and repositories, all hosted in Github and freely available for everyone at <https://github.com/torchmd>.

The first, TorchMD itself, is an end-to-end differentiable molecular dynamics code. TorchMD-NET is a software for training and creating the neural network potentials. TorchMD-CG is a specific application to learn coarse-grained potentials, currently specialized in protein folding.

At the moment, our efforts for application hardening are focused on TorchMD-NET, and trying to optimize both the speed of neural network training and evaluation in a production environment. Regarding the first problem, we have rewritten our current code and network architectures using Pytorch Geometric, a PyTorch-based library with optimized code to easily write, implement and run Graph Neural Networks (GNNs). Implementing TorchMD-NET with PyTorch Geometric provided us with a ~3x speed up in training, which is especially significant if you are dealing with large training datasets.

Another issue we were encountering is the simulation speed of TorchMD with neural network computed potentials, particularly in our application for coarse-graining simulations. We have been using TorchScript in order to compile our models and switch from a pure Python code to a TorchScript program, optimizing the code and making it independent of Python, making it possible to run the scripts in a production environment where Python speed might be disadvantageous. By using TorchScript with our trained neural network potentials, we have achieved a ~1.8x speed-up in our coarse-grained simulations using TorchMD

### ACELLERA: PlayMolecule

PlayMolecule is a drug discovery web-service. It contains a variety of applications which allow users to accelerate and improve their drug discovery workflows either through the use of novel machine learning methods (such as binding affinity predictors) or through molecular dynamics simulations to elucidate biological structures and binding modes. PlayMolecule is used daily by academic institutions and industry. Students as well as experienced researchers are using PlayMolecule to evaluate machine learning methods or simplify their molecular dynamics workflows. PlayMolecule already counts close to two thousand registered users with around 80 new users registering every month. PlayMolecule is available at <https://playmolecule.com/>. The free access has restrictions imposed on atom count of molecular systems due to constraints in computing resources which are available.



PlayMolecule consists of various separate components. The web server, the backend server and the individual applications. The web server is written in Python, React and Angular, the backend is written in GoLang and the applications are written mostly in Python with a few in C++. The web server and backend server have a total memory requirement of around 12 GB RAM. The individual application runs usually require 16 GB or more RAM, at least 2 CPU cores and a few of them, specially those related to molecular dynamics and neural networks, require NVIDIA GPUs. The code is deployed with a custom Python installation script which handles all the system setup and data shipping from Google Cloud Storage.

The architecture of PlayMolecule can be seen in Figure 7 The web server communicates with the computation backend (written in GoLang) which in turn sends job executions through RabbitMQ to the Queue app which communicates with the queueing system. The queueing system then distributes the jobs to the workers and once completed, the jobs send their results to the backend again, which stores them in the MinIO server. The backend also employs a MySQL database to manage users, job executions, MinIO file tags and other information necessary.

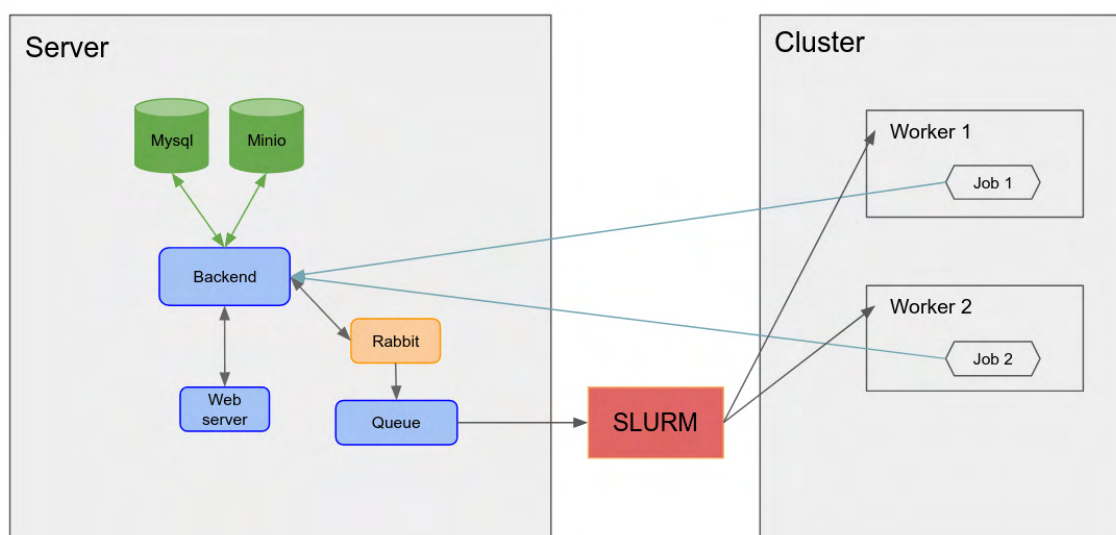


Figure 7. Conceptual architecture and key components of PlayMolecule

All PlayMolecule applications are containerized with Singularity to ensure easy deployment and reproducibility. For visualisation of the user interfaces and the results, PlayMolecule offers a web server which integrates a state-of-the-art 3D molecular viewer. HPC resources can be leveraged by PlayMolecule both to serve a larger number of users as well as to provide faster computation. Applications such as AdaptiveSampling depend on multiple GPUs being available so that multiple simulations can be run in a high-throughput parallel manner to explore faster protein conformational space, or protein-ligand interactions and thus can fully leverage large computing clusters.

PlayMolecule applications mostly run in an embarrassingly parallel manner and thus scale linearly to the number of resources which are available and jobs which are run. Network usage is limited to the transfer of input and output files for jobs which execute on cluster nodes. The data transfer is done from the applications to the PlayMolecule backend which in turn stores the files in the MinIO server. Output files are typically in the order of a few megabytes, however

applications such as SimpleRun and AdaptiveSampling generate molecular dynamics trajectories of multiple gigabytes and thus can increase network traffic. Computation time varies between applications ranging from a few seconds in ProteinPrepare to multiple weeks in AdaptiveSampling.

Ongoing work in PlayMolecule is focused on reducing data duplication between cluster nodes, the MinIO server and the web server to also reduce network load as well as storage requirements. All PlayMolecule singularity applications are built using internal Python libraries which we regularly evaluate for performance and algorithms.

### USFD: pFIRE

pFIRE (parallel Framework for Image Registration) is a software suite implementing a medical image elastic registration algorithm. The original algorithm has been developed and validated for a specific set of anatomical regions and modalities. The current work is focused on extending and validating the algorithm on different biomedical domains. From the technological perspective, the initial code needed a refactoring to improve sustainability, reliability, interoperability, and better computational performances on HPC systems (with special attention on readiness for exascale architectures).

Following the FAIR software<sup>7</sup> principles the following actions have been implemented:

- **Findable:** the software is open source and freely available from <http://github.com/INSIGNEO/pFIRE> both for humans and machines. Each version was initially identified by sequential numbering (version xx.y.z) while in the future a more meaningful convention based on the year of release will be used (version YEAR.y.z). A journal paper with DOI will be published for next release and a set of domain metadata will be defined to allow indexing.
- **Accessible:** the software is freely retrievable via standard protocols as source code, compiled binary and library, and containerised Docker/Singularity images ([hub.docker.com/r/insigneopfire/pfire](https://hub.docker.com/r/insigneopfire/pfire))
- **Interoperable:** standard data formats are used for images to be registered (e.g. DICOM, png) and to store the registration data to reproduce results (HDF5)
- **Reusable:** it is open source and built against standard C++ FAIR dependencies. License is Apache 2.0 and credit is acknowledged to both developers and those formulating the underlying theory. Thorough and accessible documentation for end-user and developers, and tutorials lower the learning curve for wider adoption of the software by the community.

### Reliability

pFIRE is co-developed with research groups focusing on clinical applications of imaging registration to satisfy the highest possible specifications in terms of reliability and accuracy. It is undergoing a comprehensive refactoring process introducing extensive testing (unit tests, regression tests) and validation benchmarks. The unit tests are implemented through the Boost Unit Test Framework while regression tests and benchmark are implemented via an in-house ad hoc framework implemented in Python and Matlab.

The benchmark suite is designed to extract quality measures to assess the registration quality to be used as a comparative measure with respect to other registration algorithms. A section of

<sup>7</sup> <https://www.fairsfair.eu/fair-practices-semantic-interoperability-and-services>

the benchmark is used to profile the computational performance of the software on parallel architectures and large datasets.

### pFIRE Containers

pFIRE is made available as Docker and Singularity images respectively for desktop users and HPC systems. The generation of images is a time-consuming process especially when several library dependencies have to be tested and supported on few operating systems (e.g. CentoOS and Ubuntu). The solution adopted for pFIRE is to modularize the container image in three components (Figure 8): Operating system, pFIRE dependencies and pFIRE release. Each component is a container image built starting from the one immediately below. Once the pFIRE dependencies image is generated, any new pFIRE releases can be generated without re-building the components below, saving time.

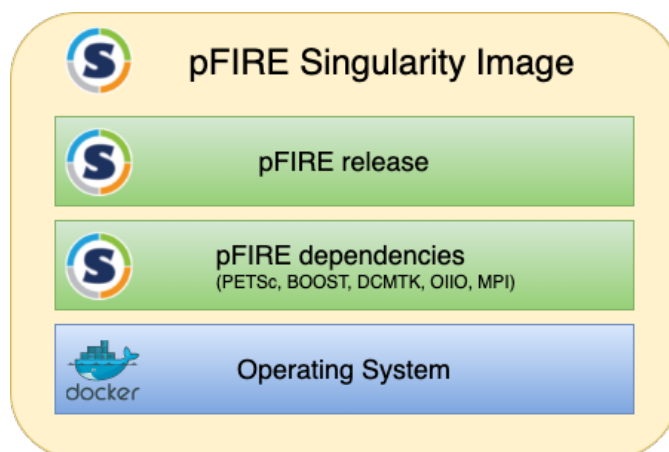


Figure 8. pFIRE Singularity image structure

### Containerisation for Cloud-HPC

Like most of the software with complex library dependencies, pFIRE installation from source is a time consuming task that may prove challenging for end-users not familiar with Linux. For that reason, pFIRE is provided as public Docker image ready to be used on any workstation operating system.

It is expected that users would like to register complex and large datasets exploiting HPC systems, with exascale systems as the final goal. pFIRE supports parallel computing via MPI and has been installed on clusters systems of the University of Sheffield (ShARC, Bessemer) and ARCHER at the EPCC. HPC system administrators usually maintain restrictive policies on installation of libraries and configurations allowed on their systems. It may prove challenging to compile pFIRE and its dependencies on any possible HPC systems. The solution that has been adopted to simplify deployment on HPC is to distribute pFIRE as Singularity image. This solution allows the use of optimized MPI installation on the destination HPC system with negligible impact on performance and deployment time.

### Continuous integration and Delivery (CI/CD)

Since the start of pFIRE development a CI/CD model has been used where code updates committed to Github triggered the execution of a testing framework on Travis-CI online; software artifacts were made available on Github. The refactoring work conducted in the CompBioMed project revealed the limitations of this setup and led to the adoption of the more flexible and customizable system based on Jenkins CI/DI. In the new system developers continue



to use Github as concurrent versioning system and upon each code commit the testing framework is run by Jenkins. It allows the CI/CD pipeline to be run on different computational nodes, Docker images and HPC systems.

In the new system, Figure 9, Jenkins executes the testing framework on a number of Docker/Singularity images representing the environments (operating systems and library dependencies) currently supported by pFIRE. Execution of testing framework on HPC systems is currently under development along with a regression and profiling benchmark.

When a new release of pFIRE successfully completes the testing, new software artifacts are released and the containers are updated.

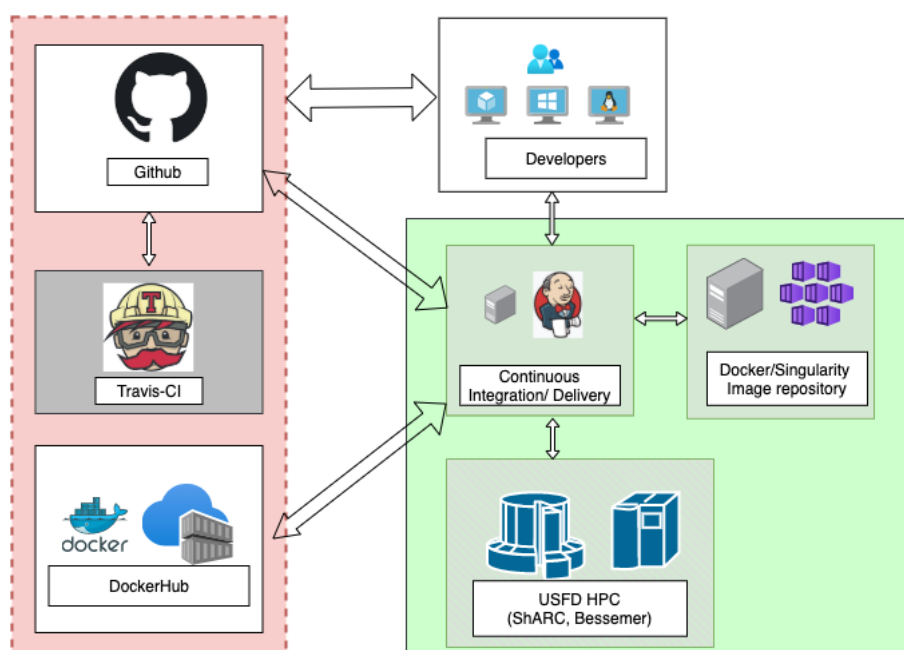


Figure 9. pFIRE continuous integration and delivery framework

Regarding co-design aspects of application usage enhancement, the distribution of pFIRE as an out-of-the-box Docker container enabled us to lower the usability barrier with users and actively engage with research groups within USFD and other European institutions to capture their feedback.

This model of co-design has been instrumental in collaborating with researchers in anatomical domains where the pFIRE algorithm was not validated before. This close interaction with end-users is enabling the algorithm to be tweaked to address for example registration of pulmonary hypertension [DT\_5], cardiac cycle [DT\_4] and digital volume correlation for bone strength analysis [DT\_6]. In particular, a benchmark framework has been developed to evaluate the accuracy of the registration algorithm on images of interest to the user. It extracts a set of quality measures used to refine the algorithm and add functionalities to the software.

### USFD: openBF

openBF is a software that allows the wave propagation problem in 1D models of the human cardiovascular system to be solved, and the prediction of flow rate, pressure and other haemodynamic variables across the network. The software has been validated against the literature in case of physiological flow in models of full body and brain circulation. It is currently

used for identification of biomarkers for cardiovascular pathologies such as vasospasm and ischaemic stroke (manuscript in preparation).

openBF is open-source and available through GitHub at <https://github.com/INSIGNEO/openBF> and through Figshare at <https://doi.org/10.15131/shef.data.7166183>. It has been licensed with Apache 2.0. The GitHub repository provides documentation and tutorials for the usage on exemplar networks. Continuous integration testing is run with Travis-CI after push to any GitHub branch.

openBF requires the installation of the Julia programming language (<https://julialang.org>), and can then be added through the package manager. Besides the Julia Revise package, which is needed for development, no further dependencies are required for research applications. Additionally, openBF is also available through Julia notebook at <https://mybinder.org>.

openBF operates with textual input and output files. To increase usability and user uptake, we are developing a GUI using the Python library Tkinter. This will allow users to create networks and define their connectivity and parameters using a drag and drop interface, and to run simulations directly from the GUI.

### UvA: HemoCell

The enhancements introduced in HemoCell focus on application hardening by introducing a variety of updates considering the build system, the continuous integration / continuous development (CI/CD), and the user-facing documentation pages. These updates simplified the build process, reduced repeated code, and streamlined the CI/CD pipelines. Combined with improved documentation, they make HemoCell more approachable for first-time users and more robust for changes during development.

Previously the build system used CMake to configure and compile HemoCell. Specifically, a CMake configuration was provided with each individual example or case study. These files included compilation information, such as compiler flags, dependencies, and other settings, to ensure the examples are properly linked and compiled. The downside of this approach was the large number of repeated configuration information within each example file. Thus, changing a compiler flag would require repeated, and error-prone, changes throughout all CMake files.

To reduce this repetition, the build system has been largely rewritten. The new approach still uses CMake., however, now HemoCell is compiled as a library first, providing multiple versions of the library with different physics enabled, such as interior viscosity or solidification mechanics. Afterwards, the individual examples only require linking with any of these precompiled libraries. This has the advantage of combining all configuration information into a single file and reducing example configuration to a simple instruction to link with HemoCell. Furthermore, recompilation of examples does not (necessarily) require recompiling the complete library and only requires compiling and linking the user's code. This significantly accelerates the "edit-and-compile" cycles during development.

In addition to rewriting the build system itself, a set of validation tests have been included. During compilation, the configuration provides an optional dependency with the Google Test framework, enabling the compilation of multiple validations tests. These tests are based on previously published numerical experiments illustrating validated cell mechanics and they ensure that previously validated behavior remains true during future development.

Finally, the CI/CD pipelines have been updated to reflect the changes in the build system and introduction of validation tests. The pipeline contains four distinct stages. First, all libraries, examples, and tests are compiled. Secondly, any (unit) tests are evaluated to ensure the tests code is still functional. Then, we evaluate the detailed validations tests. If those pass, the documentation is compiled and provided on Gitlab for the latest version of the library. These pipelines are enabled for all merge requests onto the library's main branch, providing a distinct moment for code inspection and review.

## UCL: TIES

### TIES

Thermodynamic Integration with Enhanced Sampling, or TIES, is a protocol for the calculation of the Relative Binding Free Energies (RBFE) with the potential applications in different stages of drug development. TIES is part of the broader suite of programs for binding affinity calculation (BAC). To create TIES we have written a Python application using the latest software practices that include Unit Testing and Continuous Integration, Python API and CLI that is based on it, which we further then validated<sup>8</sup>. We have now released a software suite ([ties-service.org](https://ties-service.org)) consisting of a web interface called WebTIES to generate input for RBFE calculations and an open source package TIES MD to run RBFE calculations. BAC programs are parallel applications and as such can be scaled up to consume large core count. In recent block operations on SuperMUC-NG applications of BAC used 311,040 cores to calculate 3200 binding affinities in 11 hours of wall time.

### WebTIES

The web portal facilitates the preparation of input files for the RBFE calculation with the TIES protocol. In other words, it is an interface to the TIES 20 software. It is developed using the Django framework in Python and the overall ecosystem is presented in the Figure below. It consists of four main elements: a web server for interacting with the users, a queueing system for any computational tasks that span more than 0.1 of a second, workers that process the queue, and the database.

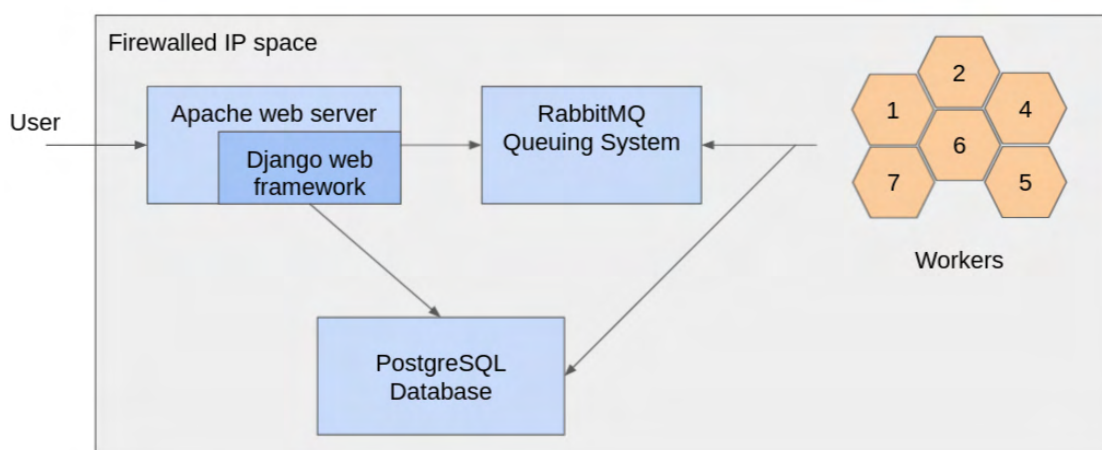
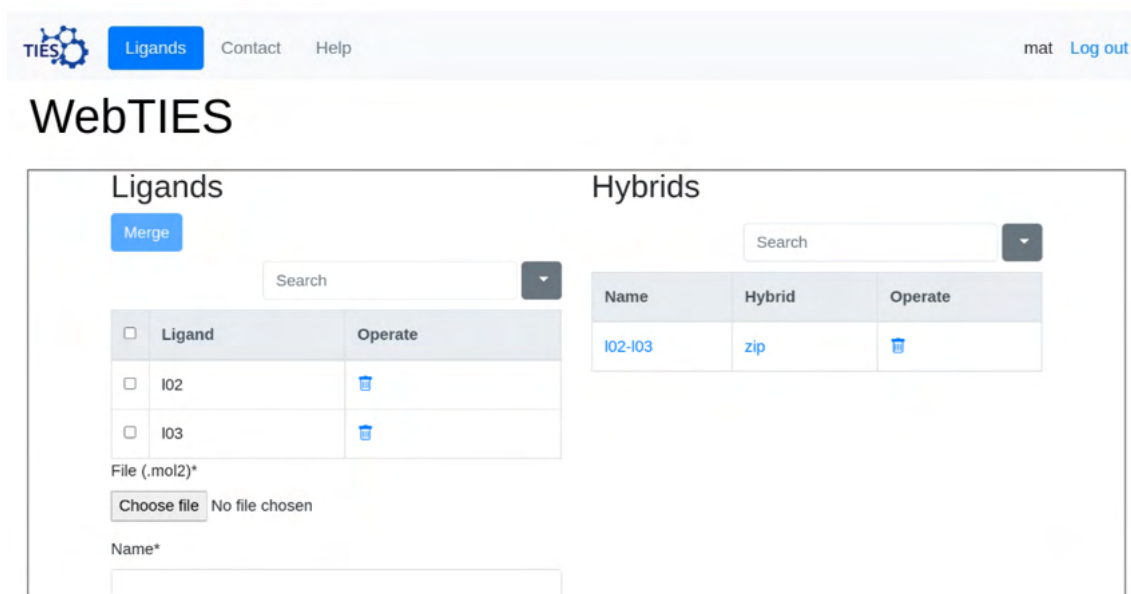


Figure 10. WebTIES Architecture. The arrows represent the direction in which the connection and interaction is initiated.

WebTIES is configured to use a reverse proxy with the encrypted HTTPS protocol in the public space. Internally, the requests only on the specific port are forwarded to the correct Apache server. Furthermore, any administrative work (/admin), including staff user accounts in WebTIES, are required to be in the administrative virtual private network (VPN). Additionally, any connection made is password controlled and open only to specific hosts.

Each of the major components is hosted in either a container or a Virtual Machine, with the communications taking place over the network, thus ensuring that the components can be replaced easily.

The system was designed in order to avoid being overwhelmed with work by employing the queueing system. Users are allowed to submit a number of jobs where each can take anywhere from 2 seconds up to 30 minutes in extreme cases on a single core. This work is saved and processed by the workers as they become free. Our initial set up uses 50 workers with the full capacity in our network of around 300 workers. This ensures appropriate scaling in the future, as the container can be easily deployed on additional hosts to increase the processing capabilities.



The screenshot shows the WebTIES web interface. At the top, there is a navigation bar with the TIES logo, links for 'Ligands', 'Contact', and 'Help', and a user profile section with the name 'mat' and a 'Log out' link. Below the navigation bar, the main content area is divided into two sections: 'Ligands' and 'Hybrids'. The 'Ligands' section has a 'Merge' button, a search bar, and a table with columns 'Ligand' and 'Operate'. The table contains two rows: 'I02' and 'I03', each with a trash icon in the 'Operate' column. Below the table, there is a file upload section with a 'Choose file' button and a 'No file chosen' message. The 'Hybrids' section has a search bar and a table with columns 'Name', 'Hybrid', and 'Operate'. The table contains one row: 'I02-I03' with 'zip' in the 'Hybrid' column and a trash icon in the 'Operate' column.

Figure 11. The initial minimal web interface. Users can upload ligands, process them, and then download the generated files.

The initial release of the website is minimal as shown in the web interface Figure 11 above. User creation is automatic for everyone with an academic email, whereas private users are asked to answer additional questions. User emails are verified with a standard token generation and automated emails, and any user input is sanitised. Once the user generates the input files, the next stage is to run the computationally expensive calculations by employing TIES MD.

### TIES MD

TIES MD is an open source Python based implementation of the TIES protocol. TIES MD can set up and run RBE calculations, a task which is normally complex and time consuming. This

complexity arises from two sources 1) because running many replica simulations is critical to control the aleatoric error in Molecular Dynamic (MD) simulations which are inherently chaotic and 2) because RBFE can be run with an alchemical methodology which requires many independent simulations to be run for different lambda windows<sup>8</sup>. As such a typical RBFE calculation might involve setting up and running 65 independent simulations, TIES MD expedites the setup of these simulations by preparing all required inputs and scripts automatically. TIES MD can create input for the commonly used MD engines OpenMM<sup>9</sup> and NAMD<sup>10</sup> and could be extended to support others. The Figure below shows all the individual simulations that TIES MD sets up for one RBFE calculation explicitly. These ensembles of calculations cannot be run without large core counts over which to distribute the workload. Each replica and lambda window is independent thus these simulations are embarrassingly parallel and so can scale to tens of thousands of cores easily and can be deployed on emerging exascale machines.

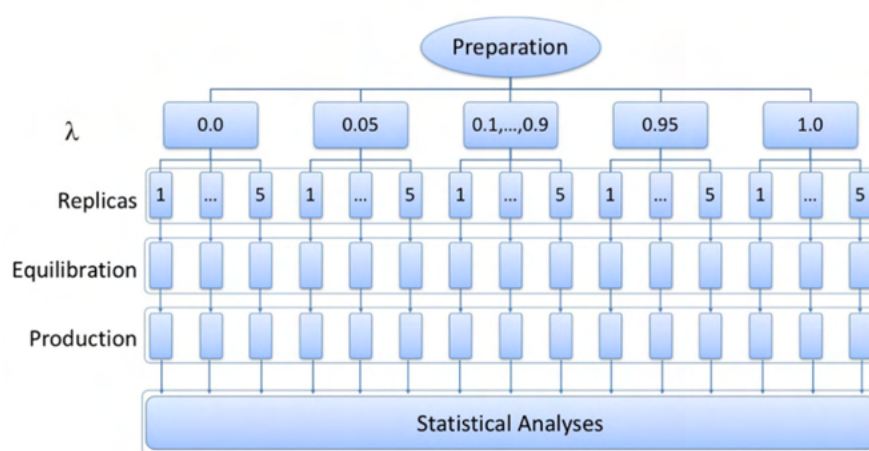


Figure 12. All simulation prepared by TIES MD for a RBFE calculation. There are 65 simulations total coming from 13 Lambda windows each run for 5 replicas. For each simulation there are multiple stages: equilibration, production and analysis TIES MD will also perform all these stages, as necessary.

TIES MD is supported by documentation which is provided here: ([https://ucl-ccs.github.io/TIES\\_MD/](https://ucl-ccs.github.io/TIES_MD/)) This documentation is built automatically using Sphinx<sup>11</sup> and documents both the code and provides tutorials for how to run the simulations in High Performance Computing (HPC) environments.

<sup>8</sup> Jorgensen, W.L. and Thomas, L.L., 2008. Perspective on free-energy perturbation calculations for chemical equilibria. *Journal of chemical theory and computation*, 4(6), pp.869-876.

<sup>9</sup> Eastman, P., Swails, J., Chodera, J.D., McGibbon, R.T., Zhao, Y., Beauchamp, K.A., Wang, L.P., Simonett, A.C., Harrigan, M.P., Stern, C.D. and Wiewiora, R.P., 2017. OpenMM 7: Rapid development of high performance algorithms for molecular dynamics. *PLoS computational biology*, 13(7), p.e1005659.

<sup>10</sup> Phillips, J.C., Braun, R., Wang, W., Gumbart, J., Tajkhorshid, E., Villa, E., Chipot, C., Skeel, R.D., Kale, L. and Schulten, K., 2005. Scalable molecular dynamics with NAMD. *Journal of computational chemistry*, 26(16), pp.1781-1802.

<sup>11</sup> Sphinx Homepage. Available at: <https://www.sphinx-doc.org/en/master/index.html> (29th September 2021).

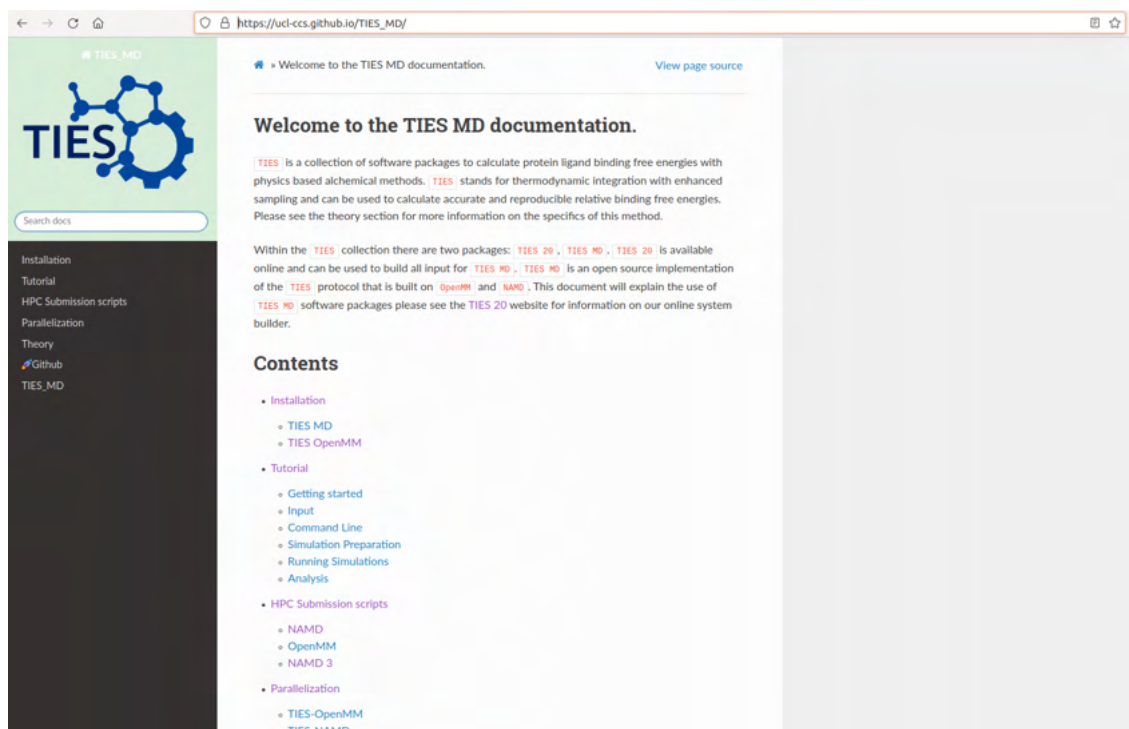


Figure 13. Presentation of the home page for TIES MD documentation.

The performance of TIES MD is critically linked to the performance of the MD engines which run the MD simulation. Performing the MD simulation is the bottleneck to this code and as such any performance improvement to these MD engines is inherited by TIES MD. To provide context for the performance of these simulations when using OpenMM with 65 V100 Nvidia Graphic Processing Units (GPU) one RBEF calculation would take around 2 hours of wall time. Since all the simulations are independent, performance can be scaled linearly adding more RBEF calculations, lambda windows or replica simulation with little or no loss in performance.



## 7 Conclusions

This document has defined the nature of the activities that provide the basis for incubation of software applications within CompBioMed2, with a special focus on the aspects that will be useful as the systems grow towards exascale. Progress to date has been reported for tasks which have already commenced, along with future plans for activities which will be reported in later WP5 deliverables. Links to activity in other Work Packages has been described, particularly in the context of the contribution of the incubation activities to the overall sustainability plan for the CoE.

In this deliverable we have updated the different Incubation Plan components, as set out in D5.1. We have updated the Innovation Management, regarding registry and status of the EEAB and the Stakeholder Engagement status. Finally, we provided a brief description of the different advances on the Enhancement of Software/Service offering of CompBioMed2 from the technical point of view for the different applications of the Center of Excellence.