HPC/Exascale Centre of Excellence in Personalised Medicine

D1.2 Software best practices and optimisation interim report

Version 1.0

Contract Number	951773
Project Website	http://www.permedcoe.eu/
Contractual Deadline	M18, March 2022
Dissemination Level	PU
Nature	R
Author(s)	Jesse Harrison (CSC), Arnau Montagud (BSC), Vincent Noël (IC), Pablo Rodríguez-Mier (UKHD), Miroslav Kratochvíl (UNILU)
Contributor(s)	-
Reviewer(s)	Jose Carbonell (BSC), Javier Nieto (ATOS)
Keywords	Best practices, software optimisation, benchmarking

Notice: *The research leading to these results has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No "951773".*

1.1. Change Log

Version	Author	Date	Description of Change
V0.1	Jesse Harrison	01 March 2022	Initial draft
V0.2	Jesse Harrison	22 March 2022	Version incorporating reviewer feedback
V1.0	Alba Jené	31 March 2022	Minor editorial changes
			(Final Change Log entries reserved for releases to the EC)

Table of contents

Table of contents 3
1. Executive Summary 4
2. Introduction 5
2.1 Deliverable background and aims 5
2.2 Relation to other Deliverables and Milestones 5
3. Application optimisation, best practices and scalability roadmap updates 6
3.1 PhysiCell-X 6
3.2 MaBoSS 9
3.3 CellNOpt / CARNIVAL 13
3.4 COBREXA 16
4. Preliminary core application efficiencies and development areas 20
4.1 PhysiCell-X 20
4.2 MaBoSS 21
4.3 CellNOpt / CARNIVAL 23
4.4 COBREXA 25
5. Conclusions and future tasks 28
6. Annex I: Software best practices follow-up questionnaire template 29
Acronyms and Abbreviations 32
References 33

1. Executive Summary

This Deliverable provides an update on software best practices applied to the upscaling of PerMedCoE core applications for use on HPC platforms, including updated development steps with reference to software scalability and extension processes. Further to outlining optimisation plans and scalability roadmap revisions for each core application, current benchmarking and PoP collaboration developments (since the submission of Deliverable D1.1) are summarised. The overview of core tool adherence to the software best-practice guidelines established in Deliverable D1.1 is based on a follow-up survey circulated among PerMedCoE tool developers.

2. Introduction

2.1 Deliverable background and aims

A key objective of PerMedCoE is to upscale several existing cell-level simulation software tools for use on diverse HPC platforms. Activities in relation to software tool optimisation have focused on four core applications (PhysiCell, MaBoSS, CellNOpt / CARNIVAL and COBRA). As part of the PerMedCoE mid-term review, this Deliverable provides tool-specific updates on core application optimisation steps to date, adherence of the core applications to software best-practice guidelines, scalability roadmap revisions, and performance benchmarks performed either individually by the core tool development teams or in collaboration with PoP.

Adherence to best-practice guidelines was evaluated using a follow-up survey circulated among PerMedCoE tool developers (Annex I). Specifically, the questionnaire was used to:

- *Assess tool adherence to FAIR principles.*
- *Collect information on the requirements underlying core tool development activities*, as part of which core tool developers were requested to provide a brief description of steps taken to identify needs, lacks and desired functionalities of a given core tool in the context of PerMedCoE.
- *Identify potential commonalities between tools*, i.e. features, functionalities or general sets of code that could be reused across the PerMedCoE tool collection (or a subset of it).
- *Obtain details on steps taken to validate tool functionality*, particularly with reference to cross-comparisons between tool versions and via potential end-user testing.

2.2 Relation to other Deliverables and Milestones

Deliverable 1.2 serves as a continuation to Deliverable D1.1 [1] and Milestone MS05 [2]. For best-practice guidelines on PerMedCoE software containerisation and workflow development, see Deliverable D2.2 [3]. Official new developments and a public code release are also available in Deliverable D1.3 [4].

Topics concerning software best practices described in Deliverable D1.1 and not included in the follow-up survey in the present report are addressed by other PerMedCoE deliverables. The use of optimised core applications as part of different use cases is described in Deliverable D3.4 [5]. Recommendations for building block and workflow design, building block and workflow adherence to software best practices, and dependencies on system libraries introduced by core tool upscaling are discussed in Deliverable D2.2 [3].

3. Application optimisation, best practices and scalability roadmap updates

This Section summarises key development steps in relation to the refactoring of PerMedCoE core applications for use on HPC platforms, and adherence of the refactored software to software best-practices described in Deliverable D1.1 [1]. Updates are also provided with reference to the scalability roadmap described in Deliverable D1.1 [1].

3.1 PhysiCell-X

3.1.1 Application optimisation

PhysiCell-X (https://gitlab.bsc.es/gsaxena/physicell_x) is a version of PhysiCell (https://github.com/MathCancer/PhysiCell) developed for use on HPC platforms as part of PerMedCoE, incorporating support for parallel computation using OpenMP and MPI. While prior PhysiCell versions include support for shared-memory parallelization using OpenMP, hybrid OpenMP-MPI parallelism can be used to execute large-scale cell-level models involving millions of cells, with such models requiring the use of thousands of cores distributed over several compute nodes. A key development update to enable MPI support in PhysiCell-X has involved using 1-D domain decomposition to allocate voxels to individual MPI processes published as BioFVM-X [6], addressing the challenge of subdomain partitioning identified in Deliverable D1.1 [1].

3.1.2 Adherence to software best practices

FAIR principles

Details on the adherence of PhysiCell-X to FAIR principles are provided in Table 1.

Principle	FAIR principle requirements	Answer	Further details
F (Findable)	Formal tool release made public on GitHub?	Yes (PhysiCell)	Public repository: https://gitlab.bsc.es/gsaxena/physicell_x
	Dissemination of formal tool releases	Yes	Dissemination actions: BioFVM-X presented at the 19th International Conference on Computational Methods in Systems Biology, 22nd-24th September 2021, Bordeaux, France and Online BioFVM-X presented at PerMedCoE's booth at ISMB/ECCB 2021 on July 28 2021 PhysiCell-X used in PerMedCoE-BioExcel PATC course in January-February 2022

A (Accessible)	Is the core tool fully open-source?	Yes	Tool licence: BSD 3-Clause Licence
I (Interoperable)	Singularity (Apptainer) container available?	Ongoing	Not available yet for PhysiCell-X Definition file for PhysiCell/PhysiBoSS (employed by Use Case 5): https://github.com/PerMedCoE/BuildingBlocks/blob/main/Resources/images/PhysiCell-COVID19.singularity
	If a Singularity container is available, are dependencies and auxiliary installations version-controlled?	Not yet	In progress (versions to be defined, cross-compared and harmonised between building blocks)
	Are tool versions systematically versioned (e.g. Major.Minor.Patch) and documented using a changelog?	Yes	Core software releases are semantically versioned and documented using a changelog
R (Reusable)	Test suite (e.g. brief test code) available for each release?	Yes	Two experiments as test code (heterogeneity and predator-prey)
	Open-licence user documentation available?	Yes	Available in core tool GitHub repository: https://gitlab.bsc.es/gsaxena/physicell_x/-/blob/master/documentation/PhysiCell-X_UserGuide.pdf
	Benchmarking activities completed or underway as part of PerMedCoE Task 3.1	Ongoing	Initial community benchmarks are available in Deliverable 3.3 [7] Scalability benchmarks for PhysiCell are available in section 4.1.2 of the present document

Table 1. PhysiCell-X adherence to FAIR principles.

Requirements underlying software development

PhysiCell-X development activities have been motivated by the requirement for a version of PhysiCell that is:

- Efficient, with simulations performed using low-level code (and with no scripting languages employed)
- Lightweight, cross-platform and based on minimal dependencies, for improved ease of installation, use and version control
- Sufficiently flexible to accommodate diverse modelling scenarios

- Based on compartmentalised code for improved reusability

Commonalities and validation of functionality

PhysiCell-X is based on C++ and shares most of its code with PhysiCell. The focus of PhysiCell-X and PhysiCell is to model populations of cells by using off-lattice agent-based modelling.

This tool has a high capability of being connected to the rest of PerMedCoE core tools that focus on intracellular processes. One such connection that has already been done is PhysiBoSS, where MaBoSS is embedded into PhysiCell, to provide stochastic simulations of Boolean models inside each agent.

Functionality cross-checks for new versions of PhysiCell have been performed for v1.6.1 - v1.9. Systematic comparisons of results produced by PhysiCell-X versus PhysiCell versions are being used as test cases and are planned as part of future PerMedCoE activities.

3.1.3 Scalability roadmap revisions

Updates and further tasks with reference to the scalability roadmap for PhysiCell-X include:

i) Employing MPI for large-scale simulations. Large-scale simulations addressed using PhysiCell-X could involve, for example, analyses with a finer resolution (more voxels), more substrates (within the cell microenvironment) and more cells. PhysiCell-X enables the simulations of larger-scale models than those possible using the OpenMP-only version of PhysiCell. These simulations will be used to identify performance bottlenecks and development opportunities while employing PhysiCell-X as part of PerMedCoE workflows, as in the Milestone MS13 part of Deliverable D3.4 [5].

ii) Replacing the current serial Thomas solver with a modified parallel solver. Limitations to scalability caused by the use of a serial Thomas solver were identified in Deliverable D1.1. Development work is currently underway to replace the serial Thomas solver employed by PhysiCell with a modified hybrid Thomas solver [8]. This new solver will be able to use openMP and MPI to solve the tridiagonal system of equations of the Thomas solver in parallel. Following Amdhal's law, this change is expected to allow for much better scalability.

iii) Evaluating inhomogeneous 1-D partitioning as a method to improve cell load balancing. For example, in the case of a spheroid, corner processes have a much smaller number of cells and this can be improved by shifting subdomain boundaries. Because this may adversely affect the diffusion solver as the number of voxels in the process increases, a heuristic needs to be created to balance the work required to compute voxels and cells.

iv) Considering alternative domain partitioning methods to increase the scalability of the code. We are studying methods that may improve the scalability of PhysiCell-X,

such as work-aware domain partitions, memory-aware domain partitions and 3D domain partitions. The use of these methods could circumvent some optimisation drains that we have detected with current code.

v) Exploring the incorporation of GPU support for solving diffusion equations. Support for different GPU architectures is a matter under discussion as a long-term development goal for PhysiCell-X. To meet this goal, it would be necessary to evaluate the performance advantages of having selected functions running on GPUs. As an example of a function that could be refactored for GPUs, the diffusion solver is launched 60 times before each evaluation of a cell-specific phenotype. An in-depth exploration would initially be required to assess current performance issues before committing to refactoring the code to confirm that GPUs would help tackle bottlenecks that are otherwise impossible to solve. An objective of PerMedCoE is to obtain a better sense of the performance drains of PhysiCell-X and the usefulness of GPU refactoring during the second half of the project. Similar projects in the field will be evaluated to check whether it is possible to adapt existing code or, at least, be inspired by previously published research (see e.g. [9]).

vi) Integrating PhysiCell-X with MaBoSS. One of the expansions of PhysiCell-X involves combining it with the MaBoSS library (PhysiBoSS-X). This has already been achieved and PhysiBoSS-X is currently under testing using a spheroid-tumour necrosis factor experiment (details in [10]).

3.2 MaBoSS

3.2.1 Application optimisation

i) Improvement of POSIX threads implementation. MaBoSS simulations work by computing probabilities over a large number of individual simulations, making it an embarrassingly parallel problem. Support for parallel computation using POSIX threads was already implemented in prior versions. However, the final aggregation of results from individual threads was done sequentially, resulting in worse performances for a large number of threads. A new aggregation method has been implemented which allows results to be gathered in parallel, allowing the aggregation to scale logarithmically with the number of threads.

ii) Development of MPI parallelisation. A new level of parallelism has been implemented, allowing MaBoSS to run simulations on multiple HPC nodes, using MPI. It allows very efficient distribution of the computation load, without suffering from the memory bottleneck. The size of results which is passed between nodes is relatively small, and is not dependent on the number of individual simulations. With this new functionality, MaBoSS can scale to a very large number of individual simulations, which is crucially needed for getting precise results on very large models.

iii) Memory improvements. While improving the support for parallel computation in MaBoSS, inefficiencies were identified that resulted in the production of large data structures that appeared unnecessary. These methods were optimised to only

generate the data structure when needed, vastly reducing the memory usage of most MaBoSS simulations.

iv) Compatibility with community standards. MaBoSS uses a proprietary format to describe Boolean models. While a specialised package of the SBML modelling language (SBML-qual) has been developed to describe boolean models, this standard is not yet fully compatible with MaBoSS models due to the lack of description of (in)activation rates. SBML-qual can, however, still be used to represent simple MaBoSS models, with default values for rates. Previously, external software (GINsim, BioLQM) has been required to convert these models to the MaBoSS format. To remove this dependency on external software, direct support has been implemented for SBML-qual models in MaBoSS, simplifying the simulation of large numbers of published models. Support has also been implemented for another standard (Bnet format) to further facilitate the interoperability between tools within the CoLoMoTo community (http://colomoto.org).

v) WebMaBoSS: a web interface for MaBoSS modelling. MaBoSS simulations were initially performed using the command line, which generates CSV files for simulation results. While very robust, this was a limiting step for users. To simplify the use of MaBoSS, simple Python bindings were developed to facilitate the simulations via Python code, providing results as standard Pandas dataframes. The only drawback is that a certain knowledge of Python is required. Even though it remains a very common language, we perceived it was still difficult for non-computer scientists to get introduced to MaBoSS modelling.To further improve the accessibility of MaBoSS to users with limited programming experience,, a web interface (WebMaBoSS) was developed [11]. This web interface allows users to easily import models from databases (leveraging the new SBML-qual compatibility) and store them in a database. Users can then easily modify them, simulate them, and obtain interactive figures to browse results. The web interface is already used in courses on MaBoSS modelling and has received positive feedback from students (including biologists, bioinformaticians, or modellers). Another challenge that the web interface tackles involves simulations of sensitivity analyses of MaBoSS models, gathering the outputs of a large number of possible mutants. These analyses can potentially take a long time to run and need special methods for the large resulting data sets to be analysed. To address this challenge, a user-friendly interface was developed to create such analyses, which can then run on the server of the web interface. Additionally, a filter is provided to search for potentially interesting mutants. Access to WebMaBoSS could potentially be extended to include computations performed on HPC. The web interface is available on the MaBoSS website (https://maboss.curie.fr/webmaboss) and its source code is available on GitHub (https://github.com/sysbio-curie/WebMaBoSS).

3.2.2 Adherence to software best practices

FAIR principles

Details on the adherence of MaBoSS to FAIR principles are provided in Table 2.

Principle	FAIR principle requirements	Answer	Further details
F (Findable)	Formal tool release made public on GitHub?	Yes	Public repository: https://github.com/sysbio-curie/MaBoSS-env-2.0 Further MPI version pending
	Dissemination of formal tool releases	Not yet	Further releases in the context of PerMedCoE to be advertised on e.g. Twitter
A (Accessible)	Is the core tool fully open-source?	Yes	Tool licence: BSD 3-Clause Licence
I (Interoperable)	Singularity (Apptainer) container available?	Yes	Link to definition file: https://github.com/PerMedCoE/MaBoSS_BB/blob/main/container/maboss.def
	If a Singularity container is available, are dependencies and auxiliary installations version-controlled?	Ongoing	In progress (versions to be defined, cross-compared and harmonised between building blocks)
	Are tool versions systematically versioned (e.g. Major.Minor.Patch) and documented using a changelog?	Yes	Core software releases are semantically versioned
R (Reusable)	Test suite (e.g. brief test code) available for each release?	Yes	Available in core tool GitHub repository
	Open-licence user documentation available?	Yes	https://maboss.curie.fr https://pymaboss.readthedocs.io
	Benchmarking activities completed or underway as part of PerMedCoE Task 3.1	In progress	Initial benchmarks available in Deliverable 3.3 [7]

Table 2. MaBoSS adherence to FAIR principles.

Requirements underlying software development

The MaBoSS development team is part of CoLoMoTo, a consortium for logical modelling tools (http://www.colomoto.org), as well as SysMod (https://sysmod.info), the Computational Modelling of Biological Systems community. The development team also organises meetings every two years in Basel to discuss community-driven guidelines for standards, annotations and curation of mathematical models, with an emphasis on logical models. These meetings and conferences highlighted major needs

of the community to reproduce, exchange and exploit models. Among these needs for novel functionalities, the following three priority areas were identified:

- *Standardisation of the models.* This was selected as a priority development target because MaBoSS was not directly compatible with the widely used standard SBML-qual format.
- *Optimisation of the performances of the model simulations.* For this issue, methods were sought to scale up simulations with models involving 50–100 nodes, leading to the first parallel implementation of MaBoSS using POSIX threads.
- *User-friendly web interface for users.* This development target focused on simplifying the use of Boolean models with MaBoSS without knowing the details of the mathematical framework. An interface was developed where models in SBML-qual format can be imported, run with MaBoSS and make use of all the functionalities of MaBoSS (sensitivity analysis, drug simulations, etc.). This interface is and will be used for teaching and as a support for any published models, in support of activities to improve core tool usability and access as part of PerMedCoE Work Package 2.

Commonalities and validation of functionality

A common element between development work focusing on MaBoSS versus other PerMedCoE core software tools has involved the provision of MPI support. Moreover, MaBoSS was initially developed as a standalone application, but its usage is moving toward being a C++ library, used by other software. The first software to do so is PhysiCell (Section 3.1), which now includes support for performing MaBoSS simulations for intracellular models in the form of PhysiBoSS, an add-on of PhysiCell. Recently, CompuCell3D (another agent-based framework) also started using the library for simulating intracellular models.

While systematic cross-comparisons between the development version of MaBoSS (with MPI compatibility) and prior versions of the software are yet to be completed, initial functionality tests have demonstrated an ability to perform large simulations on multiple nodes (with >90% parallel efficiency on eight nodes). MaBoSS is also being evaluated by PoP, with results currently pending (see Section 5).

3.2.3 Scalability roadmap revisions

Updates and further tasks with reference to the scalability roadmap for MaBoSS include:

i) Investigating the feasibility of exact simulations using GPUs. MaBoSS algorithm simulates multiple continuous time Markov chains to compute approximate time-dependent probabilities of Boolean states. The main advantage is that Boolean models can be simulated without storing the full state transition graph (STG) in memory, and thus permitting the simulation of large models. One application of MaBoSS, ExastoLog,

was developed to compute exact steady state probabilities using the full state transition graph [12]. This method relies on linear algebra to compute the steady state probabilities using the model STG, and its main bottleneck is the inversion of very large matrices. The current method uses CPU to compute the inverse matrices, and in theory could be improved by performing this operation on GPU. However, despite using sparse matrices, the STG uses a very large amount of memory which, in practice, limits the size of models to approximately 20 nodes (using a few GB of memory). For each addition of a node in the model, the memory footprint of the STG is expected to double. With such drastic constraints, we expect to quickly hit the limit of the GPU memory, even for moderate-size models. An ongoing area of investigation involves comparing the relative performance benefits (and limitations) of implementing this new simulation algorithm on GPU versus those gained through improved support for parallelism in MaBoSS.

ii) Further HPC compatibility improvements based on analytical bottleneck identification. This work will be completed in collaboration with PoP (Section 4.2.2).

iii) Development of new Python bindings using Python C/C++ extensions. The present version of Python bindings for MaBoSS is a wrapper around the MaBoSS binary executable, which lets us easily modify the model, and run simulations. However, this comes with two important issues. The first one is that a new parser is required to load the model in memory, modify it, and then produce a new version of MaBoSS model files. This leads to numerous incompatibilities between the MaBoSS C++ parser and the Python parser. The other challenge is that, to load results into Python data structures, CSV files produced by the MaBoSS binary must be parsed, which is costly on large datasets. To tackle both of these issues, work is underway to develop a new version of Python bindings using Python C/C++ extensions, enabling direct interaction with the MaBoSS C++ library both for editing the model, and for producing results directly into Python data structures (NumPy arrays and Pandas dataframes). We expect to finalise the development of this new version of the Python bindings with the extension as an internal module of the existing Python bindings, making it transparent for the user.

iv) Using compressed data formats for storing MaBoSS simulation results. Currently, simulation results are stored in CSV, a format which is simple to read, but costly to store and to parse. A future goal is to investigate the use of compressed data formats, including HDF5, Parquet or Feather, to efficiently store the simulation results. Using formats such as these could facilitate the execution of large-scale models by saving disk space. Further, because loading compressed data formats requires minimal post-processing, their use is expected to afford a computational speed-up compared to alternative data storage methods.

3.3 CellNOpt / CARNIVAL

3.3.1 Application optimisation

A key development step taken to upscale CellNOpt and CARNIVAL for use on HPC platforms has involved employing an Ant Colony Optimisation (ACO) simulator with MPI for async parameter fitting and OpenMP for inner parallelization of the simulator needed to evaluate the solutions. The MPI-compatible ACO simulator can be used by CellNOpt and CARNIVAL, with an ACO-compatible version of CARNIVAL having recently been developed:

https://github.com/saezlab/permedcoe/tree/master/containers/parallel-solvers

One of the main issues of the tools was the time needed to train the models to data, which compounds also with the number of samples to analyse. Also, since the models are usually not identifiable, there are many optimal solutions that can explain the observed data in the same way. The advantage of the new solver is twofold: first, it splits the parameter fitting across nodes that evolve in parallel, so they can find alternative solutions as they start for different random initializations. These tasks need to run the simulator to evaluate how good a proposed solution is and to change it accordingly. Thanks to the OpenMP version of the simulator, the evaluation of candidate solutions is also done in parallel exploiting shared-memory parallelism.

Further, we extended the functionality of the old CARNIVAL R by adding bindings for integration with the commercial solver Gurobi (https://www.gurobi.com), which is bound with MPI. This enables the exploitation of distributed memory parallelism for the faster analysis of single/small samples whenever a valid licence of Gurobi is available. This was integrated in the main stable branch for the R version of CARNIVAL (https://github.com/saezlab/CARNIVAL).

3.3.2 Adherence to software best practices

FAIR principles

Details on the adherence of CellNOpt / CARNIVAL to FAIR principles are provided in Table 3.

Principle	FAIR principle requirements	Answer	Further details
F (Findable)	Formal tool release made public on GitHub?	Yes	Public repositories: https://github.com/saezlab/cellnopt (stable) https://github.com/saezlab/permedcoe/blob/master/containers/parallel-solvers/cellnopt.tar.gz (experimental developments for WP1)
	Dissemination of formal tool releases	Not yet	Will be announced on Twitter once the tool is stable enough for end-users

A (Accessible)	Is the core tool fully open-source?	Yes	Tool licence: GPLv3 (except commercial solvers, see Section 3.3.1)
I (Interoperable)	Singularity (Apptainer) container available?	Yes	Links to definition file: https://github.com/saezlab/permedcoe/blob/master/containers/saez-tools/saeztools.singularity (CellNopt R stable) https://github.com/saezlab/permedcoe/blob/master/containers/parallel-solvers/signaling-solvers.singularity (Parallel CellNopt, experimental)
	If a Singularity container is available, are dependencies and auxiliary installations version-controlled?	Ongoing	In progress (versions to be defined, cross-compared and harmonised between building blocks)
	Are tool versions systematically versioned (e.g. Major.Minor.Patch) and documented using a changelog?	Yes / in progress	Core software releases are semantically versioned and documented using a changelog: https://www.bioconductor.org/packages/release/bioc/html/CellNOptR.html A similar approach to versioning and documentation will be applied to the new development versions for CellNopt and CARNIVAL
R (Reusable)	Test suite (e.g. brief test code) available for each release?	In progress	N/A
	Open-licence user documentation available?	Yes	Available in core tool GitHub repository (for stable version): https://bioconductor.org/packages/release/bioc/vignettes/CARNIVAL/inst/doc/CARNIVAL.html https://saezlab.github.io/CellNOptR/
	Benchmarking activities completed or underway as part of PerMedCoE Task 3.1	Yes	Initial benchmarks available in Deliverable 3.3 [7]

Table 3. CellNOpt / CARNIVAL adherence to FAIR principles.

Requirements underlying software development

CellNOpt / CARNIVAL development activities have been motivated by the following requirements and desired features:

- *Improved scalability for analyses of large networks.* Development targets with reference to this topic have included exploitation of shared-memory and distributed memory strategies.
- *Improved portability and ease of compilation.* Having a simulator implemented in C++ instead of an R version would make it possible to compile CellNOpt / CARNIVAL for specific architectures. Distribution of the software would also be improved through the introduction of fewer dependencies and decreased binary sizes.

Commonalities and validation of functionality

CellNOpt can be integrated with MaBoSS to perform continuous simulations with Boolean formalism. Similar to other PerMedCoE tools, support for parallel computation has been introduced via OpenMP and MPI support.

To validate results obtained using new versions of the CellNOpt simulator featuring a parallel solver, comparisons have been performed with the previous version with different test cases as a benchmark.

3.3.3 Scalability roadmap revisions

Updates and further tasks with reference to the scalability roadmap for CellNOpt/CARNIVAL include implementing:

i) A lightweight Python-based version of CARNIVAL including an out of the box open-source solver (COIN-OR branch and cut solver). Support for several non-proprietary solvers (including e.g. SCIP and GLPK) has also been implemented to facilitate the deployment of CARNIVAL on multiple HPC environments with different configuration requirements and licences:

https://github.com/saezlab/permedcoe/tree/master/carnivalpy

ii) Extensions for parallel knockout analysis for generation edge essentiality, and parallel permutation analysis with CARNIVAL. These extensions can directly benefit from new parallelised strategies. The idea of the former is to parallelise the knockout of selected nodes and edges to recover alternative solutions that are still optimal, identifying which parts of the contextualised network are more variable. The latter extension will work on reconstructions in parallel of permuted data in order to estimate the distribution of the null hypothesis for statistics of interest. Both steps can be performed in parallel.

3.4 COBREXA

3.4.1 Application optimisation

Application development steps related to COBREXA [13] have included implementing a flexible parallelisation pipeline for small subtasks of the analysis process and developing approaches to minimise the task distribution overhead and latency. The

implementation of the parallelization framework directly into the logic of the analysis in the package distinguishes COBREXA from other implementations of the constraint-based modelling methodology. High-level design of data-structures and analysis functions that are open for composition enable users to create analysis functions that are "parallel by default" and HPC-enabled from the beginning without any extra effort. This is a stark contrast to the current state of art, where the parallelisation is usually added to the packages as a challenge-driven afterthought, and only few methods are parallelised (typically, flux variability analysis). Despite the general, simple-looking design, the performance of the constructed analyses is competitive with or better than the performance of other highly optimised single-purpose specialised software packages. Initial benchmarking results for this work are discussed in Section 4.4.1. A further development step has involved implementing support for preloading precompiled Julia code to reduce the time required during HPC task startup.

3.4.2 Adherence to software best practices

FAIR principles

Details on the adherence of COBREXA to FAIR principles are provided in Table 4.

Principle	FAIR principle requirements	Answer	Further details
F (Findable)	Formal tool release made public on GitHub?	Yes	Public repository: https://github.com/LCSB-BioCore/COBREXA.jl
	Dissemination of formal tool releases	Yes	Dissemination activities: Julia packaging https://juliahub.com/ui/Packages/COBREXA/Uq4VT Bio.tools https://bio.tools/cobrexa.jl
A (Accessible)	Is the core tool fully open-source?	Yes	Tool licence: Apache 2.0
I (Interoperable)	Singularity (Apptainer) container available?	Yes	Link to definition file: https://github.com/LCSB-BioCore/COBREXA.jl/blob/master/cobrexa.def The container building and distribution is fully automated using CI/CD.
	If a Singularity container is available, are dependencies and auxiliary installations version-controlled?	Yes	External dependencies are controlled using standard version bounds. Additional testing is required to discover potential additional incompatibilities among current HPC facilities.

	Further steps taken relation to ensuring interoperability as part of PerMedCoE workflows (see D2.2 [3] for details)	Yes	Automated CI checks for compatibility with multiple environments, including some possible PerMedCoE target environments.
	Are tool versions systematically versioned (e.g. Major.Minor.Patch) and documented using a changelog?	Yes	Semantic versioning is required for Julia packages.
R (Reusable)	Test suite (e.g. brief test code) available for each release?	Yes	Available, package is continuously tested, code coverage is above 90%
	Open-licence user documentation available?	Yes	Reference documentation is built and published automatically from code docstrings Additional tutorials (currently 6 available) and Jupyter notebooks (currently 9) are available within the documentation: https://lcsb-biocore.github.io/COBREXA.jl/stable/
	Benchmarking activities completed or underway as part of PerMedCoE Task 3.1	Ongoing	COBREXA has been scrutinized against other constraint-based analysis toolkits in Kratochvíl et al [13]. Systematic benchmarking activities are underway.

Table 4. COBREXA adherence to FAIR principles.

Requirements underlying software development

A key development requirement introduced by PerMedCoE has involved the need for easy organisation of analysis of ensembles of large numbers of models of a specified size. Further COBREXA development work concerning the design of the extensible model type and modification system has been driven by an analysis of deficiencies in current COBRA software toolboxes. Additional format compatibility requirements have been identified with Heinrich-Heine Universität (Quantitative and Theoretical Biology Institute, Ebenhöh group).

Commonalities and validation of functionality

The model modification and parallelization framework of COBREXA might be useful in other model-centric analysis tools implemented in Julia. While older versions of COBREXA do not exist, it has been shown to outperform previous software packages in the COBRA ecosystem in terms of scalability [13]. Performance optimisation using conventional profiling and performance measurement tools was a part of the COBREXA development process, with the main results described in Kratochvíl *et al*. [13].

3.4.3 Scalability roadmap revisions

Updates and further tasks with reference to the scalability roadmap for COBREXA include:

i) Identifying options to solve non-linear models using constraint-free solvers. We have tested that an approximate solution using smooth optimisation and error-minimisation methods (available with arbitrary precision bound, with a tradeoff between precision and performance) will be able to compete with traditional algebraic solvers in terms of performance if massive parallelisation is available. At the same time, linearity requirements are vastly relaxed for typical error-minimisation methods, which opens ways for solving more complicated and interesting metabolic problems, such as ones involving metabolic adjustments and thermodynamic laws.

ii) Exploring opportunities to solve large-scale models using GPU acceleration. The error minimisation methods rely on well explored, easily parallelisable operations that can be described e.g. with Sparse BLAS routines. The GPU implementation will be accelerated either using cuBLAS or hipBLAS, and possibly further optimised by customising the BLAS routines to exploit the specifics of metabolic data. If successful, interesting results may be obtained from comparisons with parallel and GPU-based versions of the traditional linear solvers, such as cuOSQP.

iii) Improving model storage and loading efficiency. The current implementation of COBREXA does not allow for memory mapping using e.g. mmap for memory-mapped file support. The approximate methods, on the other hand, allow for highly efficient storage of the optimisation state, with reasonable cache efficiency expectations and minimal serialisation overhead. This may be utilised to load the models and optimisation states very quickly, and efficiently mirror the data from main storage to main memory, parallel accelerator memory (such as GPU main memory), or to local caches (register sets, shared memory of symmetric multiprocessors). Special support might be needed for efficient storage of sparse stoichiometric matrices that is suitable for the cache hierarchy and execution model of the parallel accelerators, to optimise the occupancy of the individual compute units and minimise the amount of necessary cacheline transfers.

iv) Model exchange and long-term archival methods. The currently utilised model formats (e.g. SBML, JSON and MAT) do not easily store huge models (>1M reactions), do not possess functionality for sharing model components or modifications of base data, nor follow FAIR (Findable, Accessible, Interoperable, Reusable) guidelines for data. On the contrary, we have already observed that alternative and specialised storage and exchange methods provide significant improvement of loading performance and decrease the model storage size. We will investigate alternative storage and exchange formats (possibly based on ideas from linked data and RDF), and possibly implement the appropriate serialisation functionality and format documentation in case the investigation shows that some of the problematic areas can be improved upon.

4. Preliminary core application efficiencies and development areas

This section reports on updates concerning PerMedCoE core tool benchmarking following the submission of Deliverable D1.1 [1], and core tool upscaling development areas identified by PoP for PhysiCell. Collaborations with PoP in relation to the other core tools are currently in progress (Section 5).

4.1 PhysiCell-X

4.1.1 Preliminary benchmarking results

Preliminary benchmarking tests using PhysiCell-X have demonstrated the potential to apply the software to models comprising 2.5 million initial cells and over 1500 compute cores, representing a significant advance in scalability compared to OpenMP-only implementations of PhysiCell.

A comparison of PhysiCell and PhysiCell-X on a single computing node showed an overhead of up to 17% in PhysiCell-X. Even so, rather than having a faster tool for executing single-node jobs, the aim of developing PhysiCell-X was to enable jobs using as many nodes as possible to simulate set-ups that were not possible using PhysiCell.

4.1.2 Areas for development identified by PoP

While a PoP performance analysis has not been completed for PhysiCell-X, it has been conducted for PhysiCell. The code used for PhysiCell only differs from that used by PhysiCell-X in the domain decomposition of the diffusion solver. Thus, even though performance analysis specific to PhysiCell-X are being planned, we are confident that the lessons learned with PhysiCell (see Deliverable D1.1 [1]) will be translatable to PhysiCell-X.

The analytical bottlenecks for PhysiCell identified in collaboration with PoP concern three areas: memory allocation, load imbalance and instructions per cycle (IPC). The memory allocation problem stemmed from the allocation and deallocation of memory whenever accessing a cell's memory vector. This caused a drop in frequency that was solved when changing the memory allocation library from malloc to jemalloc (Fig. 1), a library that emphasises fragmentation avoidance and scalable concurrency support.

The load imbalance was caused by an uneven distribution of the work among threads. To contain this drain, we increased the amount of work so that idle workers could be used to perform tasks. Technically, this was achieved by adding collapse clauses to different nested for loops that used OpenMP. This change further improved the speedup of the code (Fig. 1).

The IPC drop was due to the way in which PhysiCell stores the information of new cells being generated (at the end of a vector) and the need to read this vector each time the cell needs to integrate the forces from its neighbouring cells. Several ways to implement memory-aware executions were explored, including using worksharing

methods, dynamic scheduling and executions with information on which voxels are not empty. This improved the results of the original code, but not of the jemalloc + collapse changes (Fig. 1).

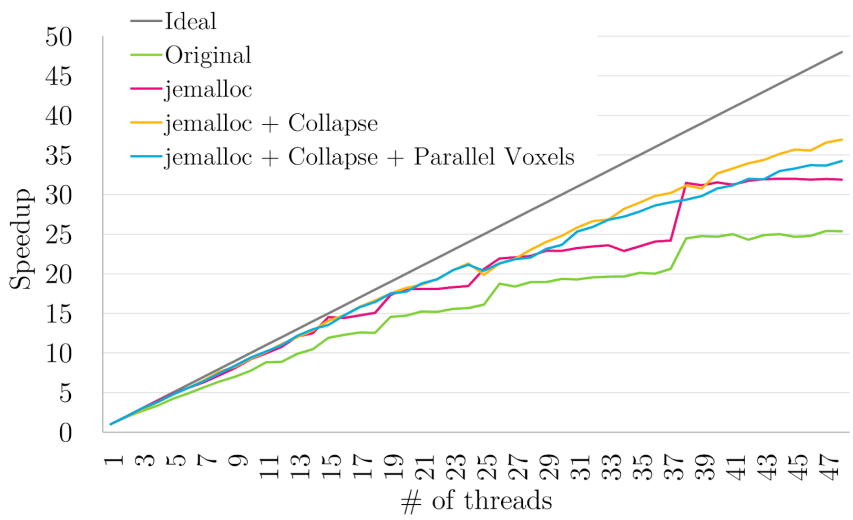

Figure 1. PhysiCell's speed-up according to the number of threads and the alteration done on the code after PoP performance analysis.

Further to these results, PoP performance analyses of PhysiCell and PhysiCell-X are being run in another computer cluster with Kunpeng 920 CPUs (ARM v8.1), with 64 cores each @ 2.6GHz. A comparative analysis of the performance results on MareNostrum 4 and Kunpeng will be provided in the second half of the project.

4.2 MaBoSS

4.2.1 Preliminary benchmarking results

Preliminary benchmarking results have been obtained following improvements to:

- The existing version of MaBoSS (v2.4.0), in relation to POSIX thread parallelisation and memory usage (see items i and v in Section 3.2.1). These improvements are implemented in a new MaBoSS version (v2.5.0).
- A version of MaBoSS with support for parallel computing using MPI (see item ii in Section 3.2.1). MPI support is also included in MaBoSS v.2.5.0 as an optional feature specified when compiling the software.

i) Execution times of MaBoSS v2.5.0 versus v2.4.0. A comparison of execution times of MaBoSS v2.5.0 and v2.4.0 using 1-32 cores and one million individual simulations showed a considerable (5×) speed-up between the pre- and post-optimisation versions, which is mostly due to optimised memory usage (Fig. 2).

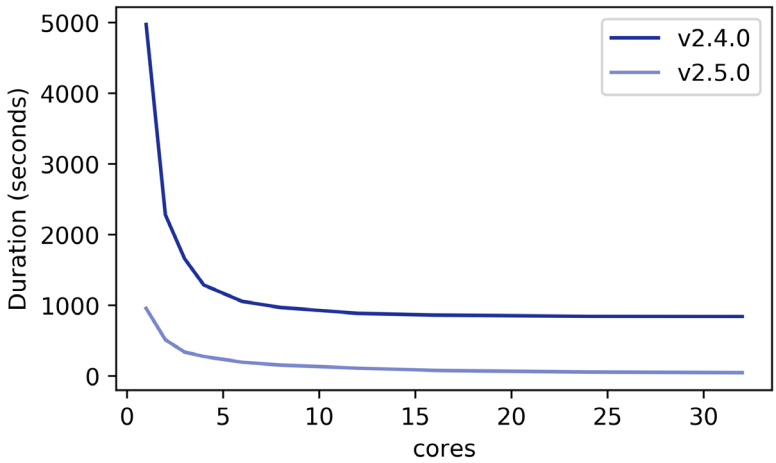

Figure 2. Comparison of execution times between MaBoSS v2.4.0 (pre-optimisation) and v2.5.0 (post-optimisation).

ii) Execution times for the MPI-compatible version of MaBoSS. A comparison of execution times using 1-114 cores and a version of MaBoSS using a combination of POSIX thread and MPI parallelism versus POSIX thread parallelism only is shown in Fig. 3. Compared to the POSIX-only version, using MPI afforded a further reduction in execution times. Using 19 cores on six nodes, it was possible to reduce the simulation time from 10000 seconds to 150 seconds (66× speedup, 58% parallel efficiency).

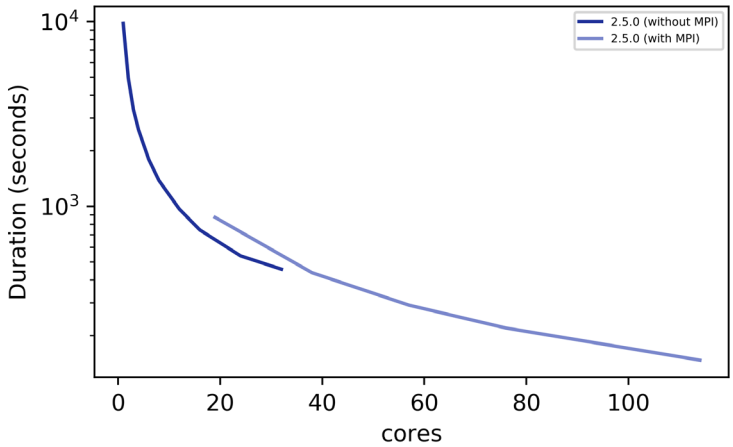

Figure 3. MaBoSS wall times using v2.5.0 without MPI support, (1-32 cores) or with MPI support (19-114 cores, 1-6 nodes with 19 cores per node).

While the POSIX threads implementation had a large parallel cost due to memory bandwidth (67% parallel efficiency on 32 cores), the MPI implementation showed a

very moderate reduction in parallel efficiency when using several nodes (99% on six nodes while reserving 19 cores per node) (Fig. 4).

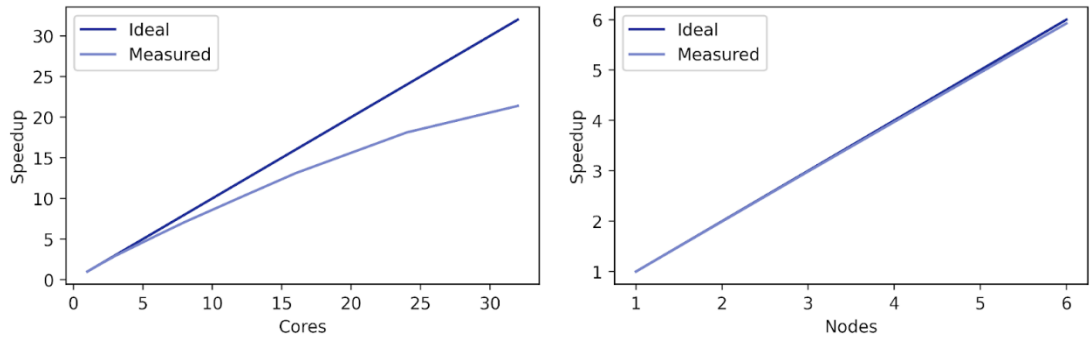

Figure 4. Parallel speed-up of MaBoSS. Results are shown for POSIX threads parallelism (1-32 cores, left) and for MPI parallelism (1-6 nodes, right).

The MaBoSS benchmarking measurements described in this Deliverable were performed on Intel Xeon Gold 6148 processors (27,5 MB cache, 2,40 GHz) with 192GB Shared Memory, using Intel Omni-path Architecture and a BEEGFS file system, and CentOS 7.

4.3 CellNOpt / CARNIVAL

4.3.1 Preliminary benchmarking results

The new CellNOpt C++ implementation with the ACO C++ solver with MPI/OpenMP parallelisation support has been benchmarked using the LiverDREAM model available on the Saez research group website:

https://saezlab.github.io/CellNOptR/5_Models%20and%20Documentation

In order to demonstrate the advantages of the new strategy in comparison with the old genetic algorithm (GA) implemented in the original CellNOpt version, the cumulative probability of obtaining the optimal solution for the LiverDREAM model was determined in relation to the required search duration (Fig. 5).

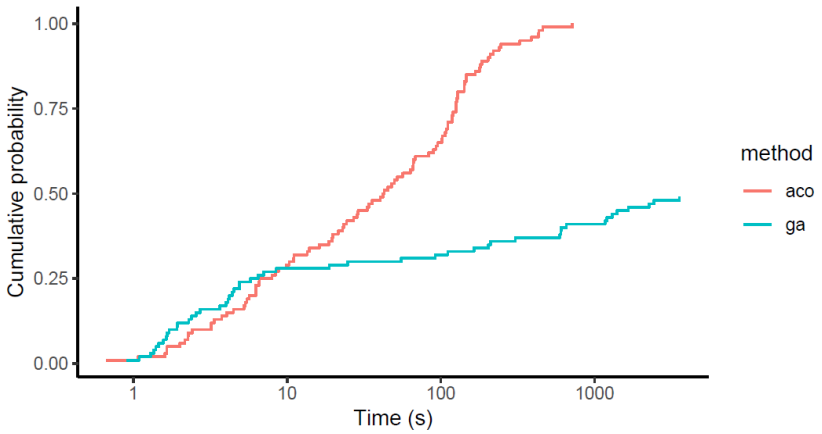

Figure 5. Average cumulative probability distribution (100 runs) of obtaining the optimal solution in the LiverDREAM challenge using CellNOpt, with a 1h max time limit.

The benchmark was designed to measure the improvements of the new tool for both shared-memory and distributed-memory parallelism. It should be noted that CellNopt and CARNIVAL do not directly benefit from more simulations per unit of time, since the simulation results do not change. However, faster simulations allow more evaluations of different solutions and thus the ACO solver may be able to discover the optimal solution faster, which is the ultimate purpose of the tools. For this reason, the benchmark measures how fast this optimal solution can be reached, on average, under different scalability settings.

The performance of the genetic algorithm and ACO were compared under different settings in using the FinisTerrae-II HPC (Galicia Supercomputing Center, CESGA). Each node has two Intel Haswell E5-2680v3 CPUs at 2.50 GHz, 12 cores per processor (24 cores per node), and 128 GB of RAM. The nodes follow a fat-tree topology connection using InfiniBand FDR 56 Gbps. Jobs involving 1-24 nodes and 1-8 OpenMP threads were used for benchmarking. For this benchmark, the GA algorithm was additionally adapted to exploit OpenMP/MPI parallelism. The LiverDREAM model was also used to evaluate the performance of CellNOpt over different combinations of node and thread reservations. The new ACO algorithm with the refactored CellNopt C++ simulator showed superior results even after adapting the old GA algorithm to be compared using the same conditions (Fig. 6).

		Average best value				% hits			
	Colonies	Threads/OMP				Threads/OMP			
	MPI	1	2	4	8	1	2	4	8
LiverDREAM aco	1	0,029524	0,028441	0,026396	0,026276	19%	28%	45%	46%
	2	0,026877	0,025794	0,025554	0,023990	38%	48%	52%	64%
	4	0,025313	0,026997	0,024471		52%	40%	61%	
	6	0,025554	0,024832	0,022305		51%	58%	78%	
	8	0,025674	0,022907			51%	73%		
	12	0,024832	0,021704			58%	84%		
	24	0,022185				80%			
LiverDREAM ga	1	0,029404	0,029524	0,028923	0,029043	20%	19%	24%	23%
	2	0,029043	0,028923	0,028321	0,028321	23%	24%	29%	29%
	4	0,027840	0,027960	0,027960		33%	32%	32%	
	6	0,027719	0,027960	0,026516		34%	31%	44%	
	8	0,027960	0,027479			32%	36%		
	12	0,027599	0,028080			35%	31%		
	24	0,026757				42%			

Figure 6. Preliminary benchmarks for CellNOpt with different threads/nodes. Results on the left show the average objective value obtained for a given configuration of threads/nodes for the ACO and GA algorithms. On the right, values correspond to the proportion of times (%) over 100 runs where the algorithm obtained an optimal solution.

4.4 COBREXA

4.4.1 Preliminary benchmarking results

Benchmarking results for the COBREXA.jl Julia package are available in Kratochvíl *et al.* [13] and the associated supplementary material. Briefly, the performance of COBREXA.jl was compared to that of COBRApy and the COBRA Toolbox on the multi-node University of Luxembourg Iris cluster (https://hpc.uni.lu), using human microbiota models available via the AGORA database [14]. Functionalities used for benchmarking included flux variability analysis and the computation of production envelopes created for 3-D grids in the flux space. Compared to COBRApy and the COBRA Toolbox, flux variability analysis using COBREXA.jl was able to utilise more computation resources, resulting in significant speed-ups gathered from distributed computation (despite the introduced distributed processing overhead, we measured over 4× speed-up on clusters as small as 256 CPU cores; see Fig. 7). Speed-ups in excess of 10× were also observed for the production envelope functionality using multicore parallelism, with multi-node parallelism providing further arbitrary speed-ups (Fig. 8). Based on preliminary observations of the scaling trend (Fig. 7), we expect the approach to work reliably for much larger models and model ensembles. A surprising amount of computational overhead was removed simply by choice of the high-performance computational environment, as seen e.g. in the JSON model loading benchmark (Fig. 9).

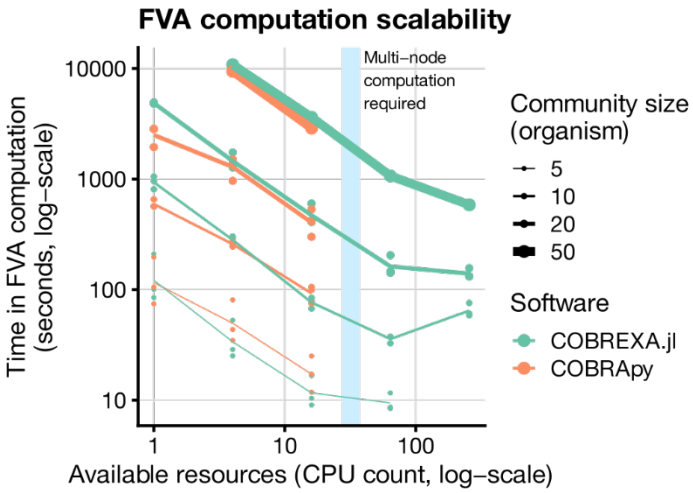

Figure 7. Performance of the distributed flux variability analysis (FVA) implementation in COBREXA compared to maximal scalability achievable with COBRApy (reported as computation time, lower is better.) Community size is measured in organisms, each organism contains on average around 2000 reactions.

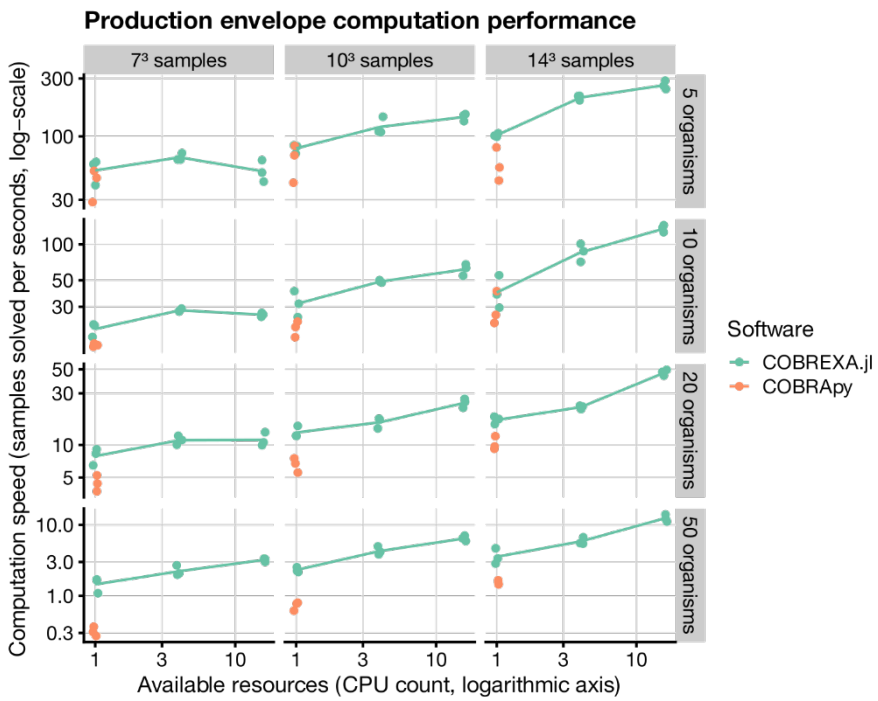

Figure 8. Performance and scalability of production envelope computation (in values computed per time, higher is better). The result shows mainly the easy applicability of the parallel-first design of COBREXA to many types of analyses, which would otherwise need to be parallelised manually with custom code (as in the case of COBRApy).

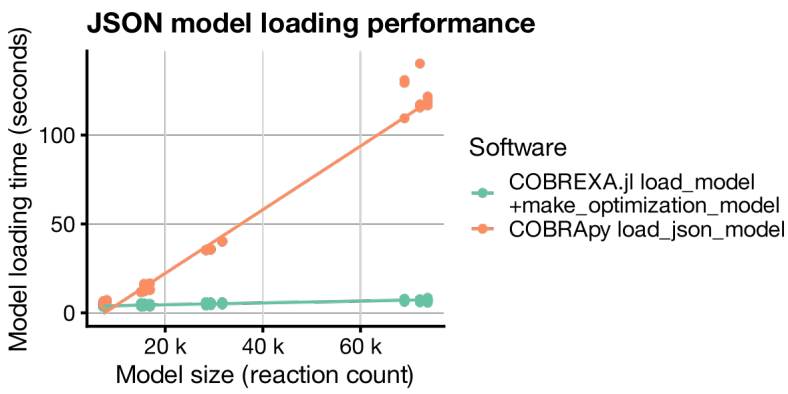

Figure 9. Performance of JSON model loading in COBREXA compared to COBRApy (reported as loading time, lower is better). The improvement is partly a result of utilising the high–performance computing environment of Julia, and partly an outcome of efficiency optimisations that avoid unnecessary internal conversions of model formats.

COBREXA.jl was found to be robust against parallelisation overhead when performing multinode analyses involving mid-size models (50 organisms / 100k reactions). The methods scale to larger models (we have successfully tested solving the models of the complete AGORA microbiome data set of > 1.5M reactions). Benchmarking of the very large models is currently planned. Future benchmarking will focus mainly on technical

aspects of the data processing (especially the storage and communication overhead, as described in Section 3.4.3. iii & iv), and on alternative approaches to model optimisation (Section 3.4.3. i).

5. Conclusions and future tasks

The software optimisation reports and benchmarking results presented in this Deliverable, along with scalability roadmap revisions and a follow-up survey of best-practice guideline implementation, provide an up-to-date overview of core software development activities within PerMedCoE. Based on work undertaken to date, the following conclusions can be drawn:

- Scalability improvements have been achieved for all PerMedCoE core applications. Further to tool-specific solutions, MPI integration has been critical to improving the HPC compatibility of the modelling tools.
- Steps have been taken to ensure FAIR principle adherence for all core applications. Where proprietary extensions are used (e.g. as in the case of the Gurobi solver used by CARNIVAL), the use of non-proprietary alternatives has been implemented.
- There is demonstrable potential for integrating different core applications and examples of this have successfully been completed as part of PerMedCoE (e.g. through the development of PhysiBoSS).
- Additional scalability roadmap targets have been identified for each core application, to further improve their HPC readiness.
- Performance and scalability benchmarking, including collaboration with PoP, is ongoing. During the second half of PerMedCoE, efforts will be directed to establish a common testing framework for tool benchmarking, validation and comparisons.

Future tasks related to software optimisation, best-practice implementation and benchmarking include:

- Exploring and implementing further features as outlined in the scalability roadmap updates reported in this Deliverable.
- Further stress testing to identify the scalability limits of HPC-optimised core software tools.
- Functionality cross-checks of development versions against previously established versions (e.g. MPI implementation of MaBoSS vs a non-MPI implementation, and PhysiCell-X vs PhysiCell).
- PoP collaborations for CellNOpt / CARNIVAL, COBREXA and MaBoSS, e.g. for the identification of key analytical bottlenecks.
- Ensuring that core software tool versioning and configuration steps are transparently implemented and harmonised between all PerMedCoE building blocks and workflows.

6. Annex I: Software best practices follow-up questionnaire template

Name of core tool: *Tool name*

Questionnaire completed by: *Name + date*

1. Adherence to FAIR principles

Principle	FAIR principle requirements	Answer	Further details
F (Findable)	Formal tool release made public on GitHub?	Yes / No	Public repository link: Link here (or delete)
	Dissemination of formal tool releases	Yes / No	Dissemination activities: List in bullet points (or delete)
A (Accessible)	Is the core tool fully open-source?	Yes / No	Tool license: License here If No, details on why not, otherwise delete
I (Interoperable)	Singularity (Apptainer) container available?	Yes / In progress	Link to definition file: Link here (or delete) Can be core tool repository, ultimately should also be discoverable under PerMedCoE organisation in building block repository
	If a Singularity container is available, are dependencies and auxiliary installations version-controlled?	Yes / No	
	Further steps taken relation to ensuring interoperability as part of PerMedCoE workflows (see D2.2 [3] for details)	N/A	Details here, e.g. cross-comparisons of Singularity definition files to ensure common library versions
	Are tool versions systematically versioned (e.g. Major.Minor.Patch)	Yes / No	

	and documented using a changelog?		
R (Reusable)	Test suite (e.g. brief test code) available for each release?	Yes / In progress	E.g. "Available in core tool GitHub repository"
	Open-licence user documentation available?	Yes / In progress	E.g. "Available in core tool GitHub repository"
	Benchmarking activities completed or underway as part of PerMedCoE Task 3.1	Yes / No	Details where possible, links to relevant info, e.g. "New benchmarks available in D1.2" (and other Milestones / Deliverables where reported or planned

2. Requirement analysis

Brief description of steps taken to identify needs, lacks and desired functionalities of the core tool in the context of PerMedCoE:

Answer here, can use bullet points

(Note: Technicalities related to these can be described in D1.2, this section should instead provide a short summary of how those needs / required functionalities were identified)

3. Identification of commonalities

Does the core tool employ features, functionalities or general sets of code that could be reused across the entire PerMedCoE tool collection (or a subset of it)?

Answer here (+ links where relevant)

Has the tool already employed solutions (e.g. for parallelisation) that are, to your knowledge, already used as part of other PerMedCoE core tools?

Answer here (+ links where relevant)

4. Validation of tool functionality

Further to performance benchmarking (details in e.g. D1.1 and D1.2), have results obtained using the latest core tool version been compared with those produced using older versions?

Answer here

Have users external to the PerMedCoE project used or evaluated the latest core tool version? If not, have relevant third-party groups been identified who could provide feedback on newly developed features?

Acronyms and Abbreviations

- ACO: Ant colony optimisation
- CI/CD: Continuous Integration/Continuous Delivery
- D: Deliverable
- FVA: Flux variability analysis
- GA: Genetic algorithm
- GPU: Graphics processing unit
- IPC: Instructions per cycle
- JSON: JavaScript Object Notation
- MS: Milestone
- PoP: Performance Optimisation and Productivity
- RDF: Resource Description Framework
- SBML: Systems biology markup language

References

1. PerMedCoE Deliverable 1.1: Roadmap of software scalability to pre-exascale, extension processes and best practices for software development
2. PerMedCoE Milestone 05: Roadmap on core applications' needs for pre-exascale optimisation
3. PerMedCoE Deliverable 2.2: Midterm code release
4. PerMedCoE Deliverable 1.3: Midterm tools release
5. PerMedCoE Deliverable 3.4: Use Case progress reports
6. Saxena G, Ponce-de-Leon M, Montagud A, Vicente Dorca D, Valencia A. (2021) BioFVM-X: An MPI+OpenMP 3-D Simulator for Biological Systems. In: Cinquemani E, Paulevé L (eds) Computational Methods in Systems Biology. CMSB 2021. Lecture Notes in Computer Science, vol 12881. Springer, Cham. DOI:10.1007/978-3-030-85633-5_18
7. PerMedCoE Deliverable 3.3: First PerMed technology observatory release and benchmark report
8. Kim KH, Kang JH, Pan X, Choi JI. (2021) PaScaL_TDMA: A library of parallel and scalable solvers for massive tridiagonal systems. Computer Physics Communications 260: [107722]. DOI:10.1016/j.cpc.2020.107722
9. Stack M, Macklin P, Searles R, Chandrasekaran S. (2021) OpenACC acceleration of an agent-based biological simulation framework. DOI: 10.48550/arXiv.2110.13368
10. Ponce-de-Leon M, Montagud A, Akasiadis C, Schreiber J, Ntiniakou T, Valencia A. Optimizing dosage-specific treatments in a multi-scale model of a tumor growth. DOI: 10.1101/2021.12.17.473136
11. Noël, V., Ruscone, M., Stoll, G., Viara, E., Zinovyev, A., Barillot, E., & Calzone, L. (2021). WebMaBoSS: a web interface for simulating Boolean models stochastically. Frontiers in Molecular Biosciences 8: 754444. DOI:10.3389/fmolb.2021.754444

12. Koltai M, Noël V, Zinovyev A, Calzone L, Barillot E. (2020). Exact solving and sensitivity analysis of stochastic continuous time Boolean models. BMC Bioinformatics 21(1): 1-22. DOI:10.1186/s12859-020-03548-9

13. Kratochvíl M, Heirendt L, Wilken SE, Pusa T, Arreckx S, Noronha A, van Aalst M, Satagopam VP, Ebenhöh O, Schneider R, Trefois C, Gu W. (2022) COBREXA.jl: constraint-based reconstruction and exascale analysis. Bioinformatics 38(4): 1171-1172.DOI:10.1093/bioinformatics/btab782

14. Magnúsdóttir S, Heinken A, Kutt L, Ravcheev DA, Bauer E, Noronha A, Greenhalgh K, Jäger C, Baginska J, Wilmes P, Fleming RMT, Thiele I. (2017) Generation of genome-scale metabolic reconstructions for 773 members of the human gut microbiota. Nature Biotechnology 35: 81–89. DOI:10.1038/nbt.3703