The Advanced Computing Hub at BSC: improving fusion codes following modern software engineering standards

OPEN ACCESS
IOP Publishing
Plasma Physics and Controlled Fusion
Plasma Phys. Control. Fusion **66** (2024) 075014 (11pp)
https://doi.org/10.1088/1361-6587/ad4589

The Advanced Computing Hub at BSC: improving fusion codes following modern software engineering standards

X Sáez[1,*], A Soba[1,2], J V Ylla Català[1], G Saxena[1], M Garcia-Gasulla[1], C Morales[1], D V Dorca[1], M Komm[3], A Podolnik[3], J Romazanov[4], E Sánchez[5], J L Velasco[5] and M J Mantsinen[1,6]

[1] BSC—Barcelona Supercomputing Center, Barcelona, Spain
[2] CONICET—Consejo Nacional de Investigaciones Científicas y Técnicas, Buenos Aires, Argentina
[3] Institute of Plasma Physics of the Czech Academy of Sciences, Prague, Czech Republic
[4] IEK-4-Institute of Energy and Climate Research, Forschungszentrum Jülich, Jülich, Germany
[5] Laboratorio Nacional de Fusión—CIEMAT, Madrid, Spain
[6] ICREA—Institució Catalana de Recerca i Estudis Avançats, Barcelona, Spain

E-mail: xavier.saez@bsc.es, soba@cnea.gov.ar, joan.vinyals@bsc.es, gaurav.saxena@bsc.es, marta.garcia@bsc.es, cristian.morales@bsc.es, david.vicente@bsc.es, komm@ipp.cas.cz, podolnik@ipp.cas.cz, j.romazanov@fz-juelich.de, edi.sanchez@ciemat.es, joseluis.velasco@ciemat.es and mervi.mantsinen@bsc.es

Received 30 June 2023, revised 15 April 2024
Accepted for publication 30 April 2024
Published 30 May 2024

Abstract

Several dedicated High-Performance Computing (HPC) centers provide essential expertise and support in developing a suitable portfolio of EUROfusion standard codes. Barcelona Supercomputing Center (BSC) is one of these HPC hubs involved in this complex task. Several fusion codes were selected, installed and analyzed to meet the developers' requirements, ranging from portability to GPU, improving the performance, getting better data management, extending the capacity of coupling with other codes, etc. In this paper, we will describe the work developed by BSC and some of the tasks carried out in this project. We will explain briefly how the project is faced and the work required to create good quality codes, i.e. robust and trustable software capable of running efficiently in modern HPC systems.

Keywords: ACH, HPC, GPU, optimization, ERO2, SPICE2, KNOSOS

1. Introduction

In the European Research Roadmap to the realisation of fusion energy [1], there is a need to accelerate the understanding and predictive capabilities of simulation models to guide ITER [2] operation and DEMO [3] design. A key factor to achieve this milestone is the production of a high-quality suite of research codes called EUROfusion-standard software to model data from existing EUROfusion facilities and to reliably extrapolate to these future devices. EUROfusion-standard software [4] is basically a common development platform following modern software engineering standards that will benefit EUROfusion users with free availability of up-to-date release versions of the research source codes to be used for production runs.

To enable large-scale numerical simulations, several dedicated High-Performance Computing (HPC) centres provide essential expertise and support in developing a suitable portfolio of EUROfusion standard software codes. These centres are called Advanced Computing Hubs (ACHs) and will provide expert help to users under three categories: (1) HPC

* Author to whom any correspondence should be addressed.

working in scalable algorithms, code parallelization, performance optimization, code refactoring, and GPU-enabling, among others; (2) Integrated modelling and control working in code adaptation to IMAS framework; and finally, (3) Data management developing open access to data and data analysis tools.

Barcelona Supercomputing Center (BSC) is one of the HPC hubs involved in this complex task. Several fusion codes were selected, installed and analysed in order to meet the developers' requirements, ranging from portability to GPU, improving the performance, getting better data management or extending the capacity of coupling with other codes. Three different groups from BSC are involved in this project: Operations, Best Practices for Performance and Programmability group (BePPP) and Fusion. The operations group is in charge of maintenance, deployment and performance knowledge of the clusters available at BSC. This group is fundamental in the installation, performance testing and acceleration of software using vectorization and hybridization techniques on heterogeneous HPC architectures. The BePPP is expert in activities such as developing programming models, dynamic load balancing of HPC code or scalability on many architectures. The researchers at BePPP have a long track record in performance assessment as well as in software engineering, development of tools, methodologies and techniques for the analysis and performance enhancement of the performance of HPC codes. Finally, the fusion group, has experts in the optimization of codes, GPU porting and expertise in the simulated physics in which each code is involved.

This paper describes the work developed by BSC in this project, and it is structured as follows. Section 2 introduces the role of the ACHs and their objectives within the EUROfusion framework. In section 3, the methodology and tools used to develop the work are presented. Section 4 explains the work required to improve an example set of codes to be capable of running efficiently in modern HPC systems. We included in this paper the work on three codes: SPICE2, ERO2.0 and KNOSOS. Finally, some concluding remarks are highlighted.

2. EUROfusion ACHs

The European Research Roadmap provides a clear and structured way forward to commercial energy from fusion, and it can be considered the basis for the programs of EUROfusion and Fusion for Energy (F4E). The roadmap outlines steps to push fusion from being laboratory-based and science-driven towards an industry- and technology-driven venture.

The most remarkable milestones in this roadmap are the building of two machines: DEMO and ITER. DEMOnstration fusion power plant project is being conducted in Europe as part of the European Research Roadmap. DEMO follows ITER project and its goal is to design a commercial fusion power plant. This is a crucial step since ITER is an experimental machine not connected to the grid to deliver electrical energy. Conversely, DEMO should deliver 300–500 MW of electrical energy to the grid (MWe). The design, construction and operation of DEMO require the full involvement of industry and this will ensure that after a successful DEMO operation industry can take on the responsibility for commercial fusion power.

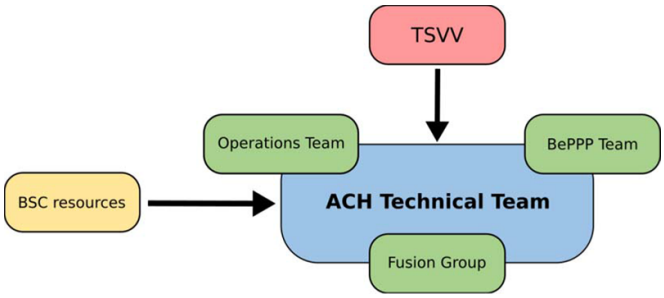


Figure 1. BSC-ACH organization.

However, the step from ITER to DEMO is challenging. DEMO design is complex regarding the number of systems necessary to produce and control the plasma. Experimental data from ITER and IFMIF-DONES [5] are essential but insufficient to design DEMO with confidence inside an unexplored environment to predict plasma and materials performance. Consequently, there is a need to create a high-quality suite of research codes (EUROfusion-standard software) to model data from existing EUROfusion facilities and extrapolate to future devices reliably.

Therefore, the goal is to bring together fusion physicists, materials scientists and engineers with a new generation of mathematicians and computer scientists within the same organizational framework to use high-performance computers (HPC) and accelerate the development of fusion energy.

As part of this approach, EUROfusion has initiated coordination among theory and advanced simulation by creating the Theory, Simulation, Validation and Verification tasks (TSVV) and the ACHs. The TSVVs will perform fundamental research and channel science, and the ACHs will be computing of excellence in specific fields of scientific modelling and simulation that will provide their expertise and knowledge to TSVVs.

2.1. BSC-ACH

In 2021, EUROfusion, through the Centro de Investigaciones Energéticas, Medioambientales y Tecnológicas (CIEMAT) [6], entrusted BSC to create one of these ACHs that offers advanced simulation to TSVVs. This hub is named BSC-ACH and provides expert support to users regarding HPC: scalable algorithms, code parallelization, performance optimization, code refactoring, and GPU-enabling, among others.

BSC-ACH is organized as figure 1 shows. Three research groups at BSC work collaboratively in each project: Fusion, Operations and BePPP. The Fusion group provides the knowledge required by the fusion scientific codes. The Operations team contributes the knowledge and tools to cover the different computer architectures. Finally, the BePPP team brings the

Table 1. Codes assigned to BSC-ACH. The codes with a grey background have been selected for this paper.

Code	Work required	ACH groups in charge
SOLPS	1. Check the correctness of the OpenMP multi-threading 2. Improve compiler options and/or job submission script 3. Consider vectorization of the code profitable for use on Cray platform	Operations, BePPP
KNOSOS	Optimization	Operations
XTOR-K	GPU porting	Fusion, Operations
BIT1	GPU porting	Fusion, Operations
SPICE2	Implementing parallel Poisson Solver and parallel electric field calculations	Fusion, Operations
GENE-X	1. Implementing the ability to access the unstructured computational grid in arbitrary order 2. Testing the performance of different reordering strategies	Fusion, Operations
ERO 2.0	GPU porting	Fusion, BePPP
STELLA	Optimization	BePPP
JOREK	Reduce memory consumption and improve performance	Fusion, Operations

experience to modify and update the codes following the best practices to get high-quality codes.

Following figure 1, TSVVs are responsible for making the necessary requests to ACHs to improve the performance and efficiency of the codes they need for their research. These requests may include optimizing existing codes, implementing new functionalities, porting the code to new architectures or adopting new tools and technologies that can help them achieve their goals by reducing the computing time spent in the simulations or improving the quality of the results achieved. In other words, the ACHs' goal is to ensure that the codes are as efficient and effective as possible, allowing TSVVs to advance their research more quickly and efficiently. In particular, as an example, table 1 depicts the work requests done by TSVVs to BSC-ACH on different fusion codes.

3. Methodology

3.1. Working procedure

The working diagram starts with a kick-off meeting (KoM) between the developers and the ACH staff. This meeting is the first contact between both teams to answer basic questions to start working: description of the code, libraries used, parallelization applied, performance, requests from developers to perform in the code, etc. Regarding these requests, there is a discussion to establish the priority among them depending on their feasibility and required resources, since the available time and resources to achieve the results are limited.

The next step is installing the code in one of the BSC clusters. Any issue that appears in this process is discussed with the developers. Once the code is installed, the Operations and BePPP teams do a performance analysis to evaluate if there is room for improvement, e.g. detecting bottlenecks among memory accesses, unbalance among processes, or a low number of arithmetic instructions executed. This analysis serves as a basis for deciding on the next steps.

The priority is to solve any performance issue detected. If this is not possible because there is not enough time and resources to carry out such a task, a new following meeting is scheduled to discuss with the developers how to continue. For example, improving efficiency could require the change of specific data structures that would involve modifying much of the source code. Finally, for each code is written a report of the work developed every year.

3.2. Tools

This section introduces some of the tools used for the analysis in the example codes described later.

3.2.1. Profiling

- **Extrae** [7] is a tool to trace the performance of a code by collecting hardware performance indicators and software events to generate trace files.

  Extrae is compatible with several programming models, such as OpenMP [8], MPI [9], OpenCL [10], CUDA [11], and their combined usage with MPI. To analyze a code with Extrae, it is necessary to instrument that code by the submission of a script defining certain environmental variables. The user can customize the instrumentation in an XML file, which specifies hardware counters to record, detail level of the tracing, and more. Once the execution is completed, each process generates a file encapsulating the collected performance data. These files are unified in a coherent way to one final file using another tool.

  The fundamental strategy used by Extrae for tracing is based on the early loading of a cloned MPI library with the same routines, using LD_PRELOAD environment variable in Linux systems. This cloned library contains extra code for tracing the execution and then calling the true routines in the original library. Extrae also offers the option of a more detailed instrumentation by allowing the user to add manually specific events inside the code.
- **POP standard metrics** [12] is a methodology for analyzing parallel codes to provide a quantitative way of measuring the relative impact of the different factors inherent in parallelization. This methodology was defined by the Performance Optimisation and Productivity Center of Excellence in HPC (POP CoE) [13].

  From the trace data of an execution provided by Extrae, the tool calculates a set of metrics, organized in a hierarchy, so that each metric reflects a common cause of inefficiency

in parallel programs (such as for example figure 5, which is presented later in this paper).

These metrics are then calculated as efficiencies between 0% and 100% and, in general, efficiencies above 80% are considered acceptable, whereas lower values indicate performance issues that need to be explored in detail.

3.2.2. Tracing

- **Paraver** [14], also known as Parallel Visualization and Events Representation, is a utility designed at BSC to visualize and examine trace files.

  From trace files produced by Extrae of an execution, Paraver can present a timeline (such as figure 4, which is discussed further on) from a great number of available metrics (e.g. floating-point operations per second (flops), instructions per second (IPS), instructions per cycle (IPC), MPI calls, user functions, bandwidths and more). Analyzing these timelines, we can identify patterns or regions with performance issues as bottlenecks.
- **NVIDIA Nsight Systems** [15] is a system-wide performance analysis tool designed at NVIDIA Corporation to help identify regions that present bottlenecks across the CPU and GPU. The tool supports various programming languages, APIs, and standards, including CUDA, OpenACC [16], OpenMP, and more.

  Nsight Systems provides a unified timeline (such as figure 7, which can be found later in the next section) that presents how an application utilizes the system's hardware and software resources. As Paraver, this timeline allows us to identify opportunities for optimization and avoid potential performance pitfalls, such as inefficient resource usage or load unbalancing among threads.

3.3. Machines

This section gives an overview of the different machines used for this paper.

- **MareNostrum 4 (MN4)** [17] is a general-purpose system located at BSC. It is composed of 3456 nodes and each node has two Intel Xeon Platinum 8160 chips, each with 24 processors, amounting to a total of 165888 processors and a main memory of 390 Terabytes. Its peak power is 11.15 Petaflops.
- **CTE-POWER** [17] is a cluster based on IBM Power9 processors hosted at BSC, with a Linux Operating System and an InfiniBand interconnection network. Its main characteristic is the availability of 4 NVIDIA Volta GPUs per node, making it an ideal cluster for GPU-accelerated applications. It has a computing power of over 1.5 Petaflops.
- **Marconi** [18] is a supercomputer located at CINECA with 3188 computing nodes. Each node contains 2 Intel Xeon 8160 (24-processors) with 192 GBytes of RAM. The nodes are connected using an Intel Omnipath network. The peak performance is 10 Petaflops.
- **Marconi100 (M100)** [18] is an accelerated cluster based on Power9 chips and Volta NVIDIA GPUs acquired by CINECA. It has 980 nodes and each node contains 2 IBM POWER9 AC922 with 16 processors and 256 GBytes. The nodes are connected using an InfiniBand network. The peak performance is about 32 Petaflops.

4. Example codes

This section addresses our work on a selection of codes (SPICE2, ERO2 and KNOSOS) applying the methodology and the tools detailed in the above section. For each code, there is a brief description of that code, the task requested by the developers, the work and tools employed, and the results obtained in relation to the goals established initially.

The aim of this section is to share this knowledge, since it is difficult to find this kind of real experience of porting or improving the performance of production fusion codes in HPC architectures due to its complexity.

4.1. SPICE2

The Sheath Particle In CEll (SPICE) package [19–21] includes two codes: SPICE2 (2D3V) and SPICE3 (3D3V). These codes are dedicated to performing simulations of particles in a fixed magnetic and self-consistent electric field and have been successfully used for the study of plasma deposition near castellated Plasma Facing Components (PFCs).

SPICE2 is written in Fortran 90 and parallelized following the domain decomposition principle and message-passing interface (MPI). All internal routines are parallel, except for the Poisson solver. The Poisson solver is sequential and takes 3% of the overall calculation time. It operates with global matrices of potential and charge density.

The solvers available in SPICE2 before this ACH work were a direct solver built on UMFPACK libraries [22], using a sparse matrix strategy to manage the matrix information, and one parallel solver that present limitations in scalability. The better solver, the serial one, is very fast, although it suffers from the limitations of any direct solver when it reaches the memory limit of the system.

After the KoM, the SPICE2 developers required the following main goals. Firstly, implementing a 2D parallel Poisson solver to run simulations by increasing grid size to allow the code to treat bigger domains than the UMFPACK library allows using (≈4000 cells in each dimension). And secondly, the implementation of a parallel routine for E-field calculation.

To achieve these goals, the first task was to include a solver from PETSc library [23] in the code. The solver selected was the KSP linear solver context, and several options were explored for solving the Poisson equation. Thirteen KSP solvers were tested, and the fastest ones were KSPBCGS (preconditioned biconjugate stabilized method), KSPFBCGSR (a mathematically equivalent variant of flexible bi-CG-stab), KSPPIPEBCGS (pipelined BiCGStab method) and KSPCG (preconditioned conjugate method). Among them, the best result in terms of CPU time consumption was the KSPCG solver. The preconditioner of KSP solver can be chosen from a big

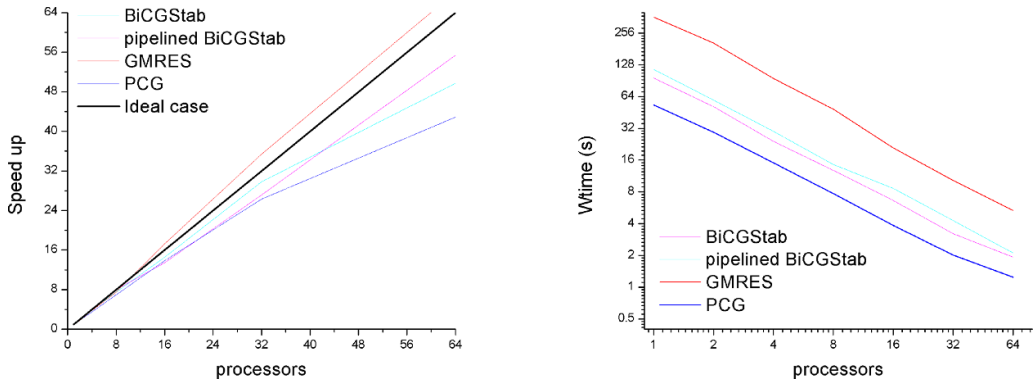


Figure 2. Scalability of selected KSP solvers for SPICE2 (left) and the wall-time consumed per each solver (right).

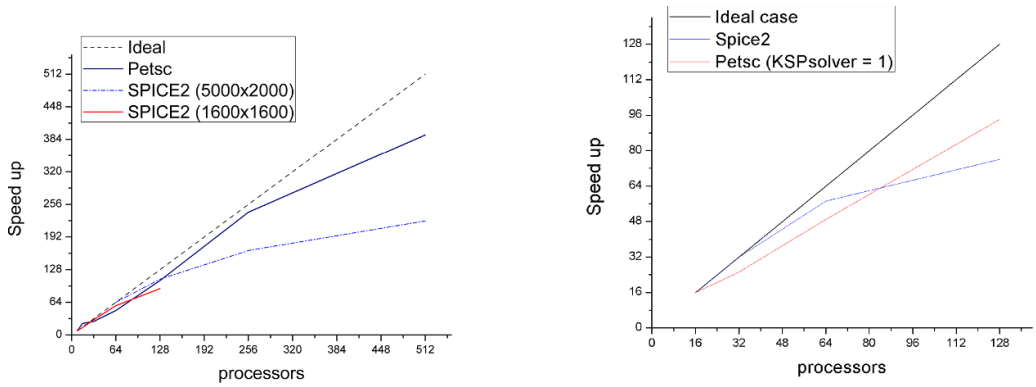


Figure 3. Scalability of SPICE2 on Marconi (left) and MN4 (right).

list of option in the PETSc library. We use the Jacobi's preconditioner to the conjugated gradient method. In general, the PETSc library allows the use of different preconditioners for each solver and it was not possible to assess the performance of the solver and the preconditioner separately [24]. But as we deal with a very simple Poisson equation, the usual solvers reach convergence easily, and better results have always been observed using preconditioners, regardless of the number of nodes used (see figure 2) [25, 26].

4.1.1. PETSc inclusion. The new solvers were separately. Figure 2 shows the scalability (left) and the wall-time consumed (right) of the new solvers. The figure indicates the best performance in the PETSc case: super-scalability for GMRES and lowest wall-time for KPSPCG. These tests were done using a medium matrix size, generated with a square of 1612×1612 domain.

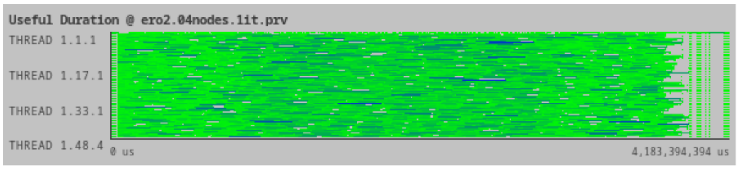
Despite the good scalability of the solver based on the PETSc library, the computation times, even for 128 processors, do not exceed the computation speed of the solver based on the UMFPACK library, which is a direct, sparse and serial solver. The new solver, based on the PETSc library, is able to run without present limitations regarding the domain size. We obtain a good performance up to 128 processors. Figure 3 (left) shows the results for the whole code using PETSc solver on Marconi. In figure 3 (right), PETSc solver standalone and SPICE2 code are shown running up to 128 processors on MareNostrum 4.

We can remark that in both supercomputers (Marconi and MareNostrum4), a loss of scalability for 256 and 512 processors is observed. Given the local domain is so small, the number of communications makes the problem inefficient. If the domain is increased, our expectation is that the efficiency will improve.

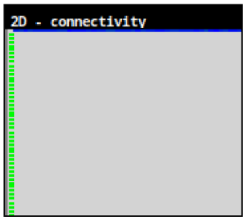
A comparison of the use of the serial UMFPACK solver vs. the PETSc solver is presented in table 2 for a 1600×1600 domain in SPICE2. The computing time of the code (excluding IO functions to avoid overhead) shows that while the computing time using the serial solver increases slightly deb to the relative weight of the time used to make initial matrix inversion, the computing time using the PETSc solver reaches better values for 256 processors.

Table 2. Comparison of the SPICE2 time using the serial UMFPACK solver vs. the PETSc solver.

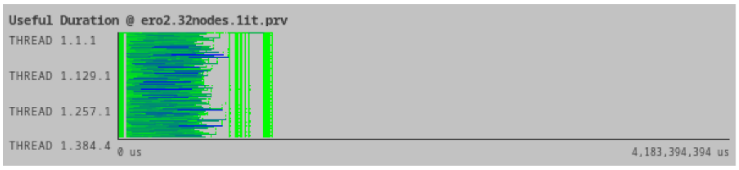
processors	UMFPACK (s)	PETSc (s)
16	108.716	976.339
32	105.991	441.469
64	117.479	247.903
128	124.301	175.839
256	160.317	153.775



(a) ERO2.0 Execution with 4 nodes - Useful duration Paraver timeline



(b) ERO2.0 Execution with 4 nodes - MPI Connectivity Paraver 2D histogram.



(c) ERO2.0 Execution with 32 nodes - Useful duration Paraver timeline

Figure 4. ERO2.0 Extrae trace view in Paraver.

To summarize the results of the scalability analysis, the code has been tested and can run up to 512 processors with acceptable scalability. The accuracy of the results with both parallel solvers is acceptable, and the results were compared with those from the direct solver when was possible, concretely in small and medium domains. Using bigger domains (more than 4000 cells per side), the comparison is not possible as we explained earlier, and only the new PETSc solver can be used.

4.2. ERO2

ERO2.0 [27] is a parallel Monte-Carlo simulation code for particle transport in fusion plasma devices (e.g. tokamaks) written in C++ and parallelized using MPI and OpenMP.

Following the initial work plan established with the ERO2.0 team, we worked on two lines of improvements: firstly, the performance analysis and, secondly, the introduction of GPU support.

4.2.1. Performance assessment of ERO2.0 and proposal for optimizations. Figure 4(b) plots the communications matrix. Given a row i and column j, there is a communication between process i and process j when there is a colour point in the intersection (i,j). The colour gradient between green and blue indicates the number of messages, where blue is the maximum value. Therefore, the figure shows there are only communications between the MPI rank 0 and any other rank because row 0 and column 0 have colour points, which means the parallel scheme of ERO2.0 is Master/Slave.

Looking at a different aspect, figure 4(a) plots a colour gradient timeline (horizontal axis) for all the processes (in the vertical axis). In this figure, green means a short burst of computation, whereas blue means more significant useful computation times. Hence from the random distribution of colours in the timeline, we deduce a variability in the duration of chunk computations, which we could later map to a variability in particle computation time. We can also observe that due to this difference in duration in particle simulation, a load imbalance is induced at the end of the simulation.

This load imbalance scales to become the main bottleneck in the execution. For instance, the run shown in figure 4(c) (in the same time scale as figure 4(a)) finishes much faster and, consequently, the load imbalanced part of the whole elapsed time is roughly half of the total execution time.

For each process, the execution consists of fetching a chunk of particles, opening an OpenMP parallel region to compute the particles using all of its threads, and finally, when all particles have been computed, closing the parallel region to start the following communication. We have observed that due to particle execution time variability, load imbalance occurs each time the parallel region is closed. The main reason for using OpenMP, as expressed by the developers, is to reduce memory usage. However, when we increase the threads per process ratio to improve memory usage, the application suffers more from this load imbalance.

The next step was to characterize the performance of ERO2.0 code by measuring the efficiency of the code by computing the POP metrics for executions with different amounts of nodes. Figure 5 depicts the loss of Parallel Efficiency is correlated with both the load balance efficiency and the serialization efficiency. This means that the load imbalance is a problem at both levels, MPI and OpenMP. Moreover, the figure also shows a serialization problem which we have found is caused by a lack of parallelization at the data-gathering phase.

As a result, the work was focused on solving the load balance issues following three proposals: two to solve the MPI load imbalance and one for the OpenMP load imbalance.

The first proposal is to start with larger chunks of particles and gradually reduce their size. This proposal aims to increase the probability that a particle that will take long falls within a chunk of the beginning of the execution.

The second idea tries to solve the load imbalance caused by particles that take much more time to compute and are distributed with one of the last chunks. The proposed solution is to dynamically compute the average computation time of particles and then use this information, the number of remaining particles, and the number of resources to predict when most

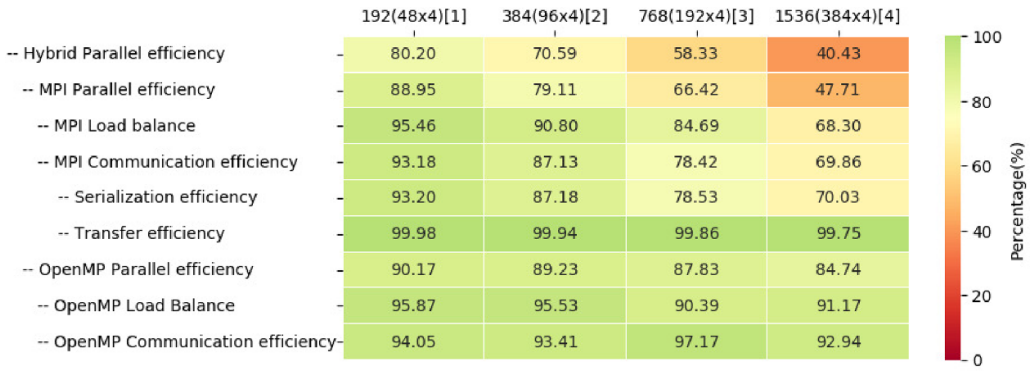


Figure 5. ERO2.0 POP metrics table for 4, 8, 16, and 32 nodes.

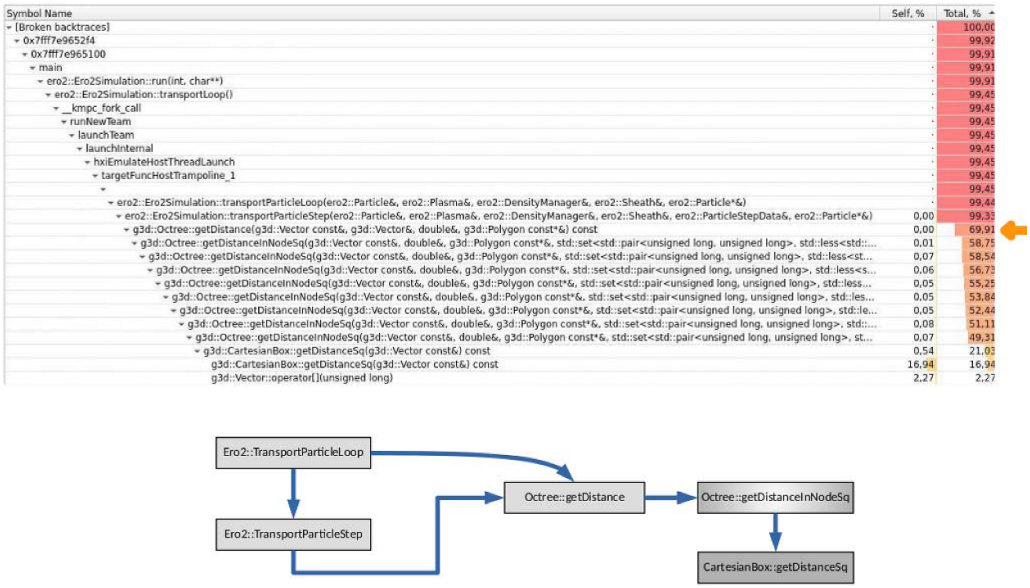


Figure 6. Top time-consuming routines in ERO2 code.

particles will have finished. Finally, this prediction is used to terminate the execution of particles past that point.

The last suggestion to solve the OpenMP load imbalance involves including the MPI communication (needed to fetch chunks) inside the OpenMP parallelization. More specifically, each thread would perform its communication and manages its particles. This parallelization scheme allows to have more threads sharing some memory, reducing the overall memory usage per node without the need for synchronization with each other.

We have implemented proofs of concept for the three suggestions and measured their performance in the MN4 machine, obtaining some preliminary results. For the first case, we observed up to 34% speedups in most cases. On the contrary, a slight improvement was observed from the second suggestion. Finally, the measures realized using all the suggestions applied reached a 50% improvement in the best-case scenario. This answer allows us to continue to the next level of work, which includes adding all these improvements to the production version of the ERO2.0 code.

4.2.2. Introduction of GPU support. The starting point for porting to GPU was the introduction of OpenACC standardization in the most time-consuming part, which was identified using NVIDIA Nsight tool (figure 6). The routine getDistance was selected with nearly 70% of the elapsed time, which performs a loop over the polygons that are inside the leaf nodes of an octree, checking the distance to each of the polygons.

Many difficulties came up during the process due to the OpenACC was not yet mature for the C++ language, such as the C++-style loops and polymorphism were not supported by OpenACC. In spite of this, a version of the routine running on GPU was achieved. Some functions of the code from the NVTX library from NVIDIA were embedded into this version to aid in performance profiling and optimization. Using the NVIDIA Nsight tool, the execution was traced and the resulting timeline is showed in figure 7.

However, the result of this first GPU implementation was not satisfactory, since the execution time increased from 8 s to 46 s. The next step was to attempt manual management of data transfers for two fields in the octree class: children and

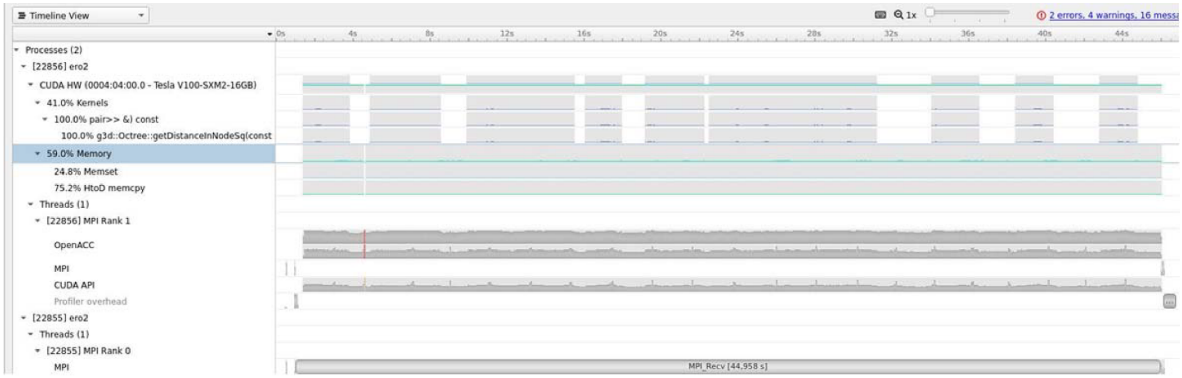


Figure 7. Trace of ERO2.0 using Nsight from NVIDIA that shows how implemented kernels are executed on GPUs.

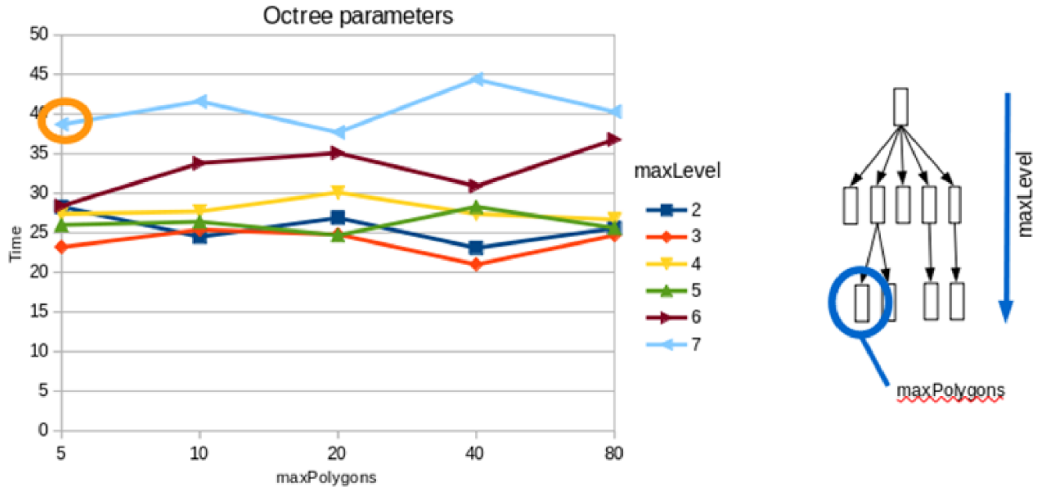


Figure 8. Test to verify how affects the depth of the octree to the code performance. *MaxPolygons* is the maximum number of polygons inside the octree leaf in order to stop subdividing, and *maxLevel* is the maximum level of subdivision after which to stop further subdivision of the octree.

polygons. The result was an improvement from 46 s to 38 s. Therefore, the works are still in progress in order to increase this manual data management to other structures in the code.

With respect to the machines used for this work, we used the CTE-POWER and M100 machines. Both machines are based on two Power9 and four GPUs, NVIDIA V100. Although the porting of the code was expected to perform without problems, due to the different software stacks, the same working code on CTE-POWER could not compile on M100 and necessary modifications were needed. After those changes, the code is able to run on both kinds of pieces of hardware.

After the results using OpenACC standard were obtained, our proposal to continue the development was to include CUDA in the code and take profit of the Dynamic Parallelism and a better C++ implementation in the recursive search in the octree. However, as the octree depth can be switched in the ERO2.0 configuration file, before any programming effort to introduce CUDA in the code, we did a simple test to verify that reducing the depth of the octree could improve the code performance. Figure 8 shows that there is a significant improvement when we decrease this parameter from the default value, and therefore, this gives further motivation to continue the work along the lines discussed above.

4.3. KNOSOS

KNOSOS (KiNetic Orbit-averaging Solver for Optimizing Stellarators) [28, 29] is a gyrokinetic code that simulates multi-species plasma in a 3D magnetic confinement. Each species is solved on its own surface. KNOSOS is completely written in Fortran90 and parallelized using MPI. Further, each surface/species is independently handled by a single MPI process. The work on KNOSOS has focused on understanding the code and analysing the possible solutions to improve its scalability and performance. The starting point was profiling and tracing the code using inputs provided by the developers, such as the test case which consisted of 22 surfaces and 22 MPI processes. One of the main issues detected was the existence of an important load imbalance between MPI ranks for some inputs. Such load imbalance seemed to be the product of the execution of different amounts of iterations in one of the subroutines (Sb1) between MPI ranks, which could oscillate between 1 and 2800 iterations.

After this discovery, we focused on and improving a sequential execution of the KNOSOS code. Since the MPI implementation of this code relies on launching 'independent' calculations, improving a sequential version

would translate to an overall improvement in the MPI version.

4.3.1. Sequential profiling. To identify the most time-consuming parts of the code, we adapted the input files received (which consisted of 22 'independent' calculations) to a new input that only contained a single calculation (concretely, the first one). Once a working version with a single MPI rank and input was obtained, basic profiling was performed using Valgrind [30].

The profiling indicated that the Sb1 subroutine takes up almost the totality of the execution time for this case. And into this subroutine, the main part of its execution time is used by the following subroutines: FILL_MATRIX_PETSC (40.55%), INTEGRATE_G (27.51%) and COEFFICIENTS_DKE (24.86%). Therefore, the next step was focused on these functions (and their direct callers/callees) to have an impact on the total execution time. Although it is possible to study how to optimize a strictly sequential version of this code (it will be done in the following steps of the optimization process), as a first step, the decision was to try to parallelize the most intensive parts using OpenMP.

4.3.2. Parallelization using OpenMP. Our first focus was the relationship between the subroutines COEFFICIENTS_DKE and BOUNCES, since BOUNCES was inheriting practically all of the execution time of its caller and it takes up about 40% of the whole execution. As different instances of the BOUNCES subroutine were identified as independents, i.e. they could be performed in parallel without producing data conflicts between OpenMP threads, the below explained implementation of OpenMP parallelization in COEFFICIENTS_DKE inside the file `low_collisionality.f90` was applied.

The addition of OpenMP in the main time-consuming Sb1 subroutines reduced the execution time from 15 s (sequential) to 10 s (using 4 OpenMP threads). Seeing that some benefits were achieved, the same approach was repeated with the rest of the subroutines of Sb1. Since LAGRANGE does not inherit all the burden of INTEGRATE_G and its contribution to the total execution time is just about 8%, the OpenMP overhead makes this improvement negligible. Therefore, a possible continuation path would be to identify and parallelize other routines to achieve a better balance between MPI and OpenMP parallelization.

4.3.3. Parallel profiling. The next step was to re-investigate the problem by building a parallel profile with Scalasca [31]. We noted that in the test executed, the MPI process with rank 15 executed 2800 iterations of the `CALC_LOW_COLLISIONALITY` subroutine while the average iterations for the remaining processes for the same routine was well below 100. Moreover, this process took 73.91 s to execute the `SOLVE_DKE_QNB_AMB` subroutine as opposed to the next maximum time of 18.14 s taken by MPI process with rank 5.

KNOSOS solves a system of linear equations of the form `Ax=b`, and after thoroughly scrutinizing the values in vector `b` for MPI process with rank 15, we detected the presence of a `NaN` at the third position of the vector. This error propagated to the vector `x` when a standard built-in solver of PETSc was called. After replacing the `NaN` value by a small random value (chosen by consulting the values in vector `b` of other MPI processes), we found that the total runtime of the application reduced from 1941.66 s to 685.77 s, a 2.83x decrease in time, and the abnormal iterations observed previously on MPI process with rank 15 disappeared. The NaN error was traced to a division by zero arithmetic exception, which was detected when the PETSc specific `fp_trap` flag was passed to the program through the command line. As a result, we strongly recommended to the developers that routines be added in KNOSOS to detect `NaN` and `Inf` values, especially in the vectors of PETSc.

4.3.4. Vectorization. One of the most time-consuming routines identified in the KNOSOS application was `CALCB_DEL`, which is executed multiple times under top-level subroutines. We found a time-intensive loop in this routine which the compiler refused to vectorize due to a forward dependency. We broke the forward dependency to achieve vectorization after introducing several temporary variables and restructuring the loop, and the outcome was a reduction of the time for a single instance of the subroutine `CALCB_DEL` from 126.51 s to 47.28 s, while the total run-time of KNOSOS decreased from 685.77 s to 551.22 s, i.e. a speed-up achieved of 2.67x and 1.24x respectively.

In our analysis, we also noticed that the compiler (Intel) was using the `XMM/YMM` registers for vectorization. Since MN4 supports AVX-512 operations, we set the compile time flag `-qopt-zmm-usage=high` to use the 512-bit registers for vectorization. Further, as KNOSOS is written in Fortran, we added the compilation flag `-align array64byte` to align the data to a 64-byte boundary (except for the data in `COMMON` block). Along with the compile time flags, we used the Intel specific `vector aligned` directive at appropriate positions in the code. All these additions further reduced the run-time of `CALCB_DEL` from 47.28 s to 35.16 s, and the total run-time of KNOSOS dropped from 551.22 s to 514.02 s, i.e. an accumulated speed-up achieved through vectorization of 3.59x and 1.33x respectively.

4.3.5. MPI parallelization. As mentioned earlier, KNOSOS involves solving multiple species on their respective surfaces using independent MPI processes. Thus, if we execute KNOSOS with 22 surfaces and more than 22 MPI processes, only the first 22 MPI processes participate in the execution while the remaining processes remain idle. To divide a surface among multiple MPI processes, we created a mapping `rank(numproc,ns)` such that if `rank(i,j)=1`, then MPI rank `i-1` participates in solving the surface number `j`. Further, an MPI process can only be assigned to a single surface, while a surface can be associated with multiple MPI processes. We created routines to replicate the work of a particular surface on

Table 3. Average application timing (per process) when multiple processes are assigned to surfaces using a Round–Robin and Compact scheme.

Processes	Round–Robin (s)	Compact (s)
22	25.98	24.71
44	24.48	21.48
48	22.31	23.35
66	19.68	21.95
88	19.13	19.05
96	19.47	21.14

all the MPI processes assigned to that surface. The MPI processes are assigned to surfaces using two different schemes:

- Round-Robin (RR): We assign the first process to the first surface, the second process to the second surface and so on. If processes are remaining, we restart the assignment with the first surface.
- Compact (CO): Assuming that the total processes (`tp`) are greater than the total surfaces (`ts`), we assign $\frac{\mathrm{tp}}{\mathrm{ts}}$ MPI processes to each surface and if processes are remaining i.e. $\mathrm{tp} \bmod \mathrm{ts} \neq 0$, we again assign one process each to the surfaces in order till the processes are exhausted.

To identify the processes associated with a surface, the processes were grouped in a separate `SURFACE_COMM_WORLD` MPI communicator corresponding to each surface. It can be noted that the MPI processes associated with a particular surface did not distribute but replicated the entire work associated with that surface. In consultation with the developers, it was decided to use these MPI processes to divide the work done in a subroutine named `COEFFICIENTS_DKE`, which took about 33% in the vectorized version. A new version `COEFFICIENTS_DKE_NEW` of the aforementioned subroutine was written, allowing the MPI processes associated with a particular surface to work on different chunks of data (portions of arrays). The decision to call the new version `COEFFICIENTS_DKE_NEW` or the old version `COEFFICIENTS_DKE` was based on whether the number of MPI processes in the `SURFACE_COMM_WORLD` was greater than one or not, respectively. After the processing of data, an `MPI_Allgatherv` involving 12 arrays was needed to gather all portions of the arrays on the root process of each of the `SURFACE_COMM_WORLD` communicators.

Table 3 shows that the best KNOSOS runtime using both distribution schemes was obtained with 88 MPI processes. A reduction of 26.37% in the case of RR and 22.90% in the case of CO can be seen using 88 processes (compared to the baseline of 22 processes). At a subroutine level, the subroutine `COEFFICIENTS_DKE` took approximately 5.43 s per process in the 22 MPI process case and 1.57 s per process in the 88 process case. This amounts to a speed-up of ≈3.45 - a value close to the ideal value of 4. In summary, we were able to reduce the total run-time of the application by approximately 51% with Vectorization, additional compiler options and our MPI parallelization scheme.

5. Conclusions and discussion

In this brief contribution, some of the procedures and activities of BSC-ACH have been presented. We describe the main tools used to perform the analysis tasks and different strategies adopted to optimize the codes of the fusion community.

We chose three codes in different stages of the work to show three typical examples of the interaction between the ACH and code developers. For one of the cases, KNOSOS, the final obtained results were good enough for the code authors at that stage of the development. In the case of the SPICE2 code, the improvement introduced in the code was in line with the developers' requirements. These results are not as good as we expected, but this leads to a new round of discussions between the ACH and SPICE developers, in order to try new improvement possibilities in a new cycle of work. Finally, the ERO2.0 case is an ongoing work, and we present some details on how bottlenecks were detected. However, it is necessary to clarify that similar work was done over the KNOSOS and SPICE2 codes to detect bottlenecks and workflow. Furthermore, ERO2.0 demands efforts for porting it to GPUs, a task that we also need to do over other codes in the ACH project, and this is why we selected this code as an example since it is in a more advanced stage in our ACH.

It is necessary to clarify that many of the codes we are working with have a long history of development, and hundreds of hours of work by their respective developers. This ensures that the results obtained with these codes are relevant from the point of view of the physics they simulate. However, it also implies considerable prior optimization work. As a consequence, it is not easy to obtain drastic changes in its computational behavior, and many times working for months on a code results 'only' an appreciable but not decisive increase in its performance. However, the way to obtain a code with a professional character is to combine the work of different specialists, who test the code under various conditions and on different machines with different configurations. This ensures portability, good behavior, and debugging of bugs that are only discovered during these tasks.

In this sense, the ACHs carry out a task that is not only beneficial for the codes that are involved, but also necessary to bring these codes to standardization that ensures their continuity in current machines and, above all, also to that the fusion community will have access to in the future.

Data availability statement

The data cannot be made publicly available upon publication because the cost of preparing, depositing and hosting the data would be prohibitive within the terms of this research project. The data that support the findings of this study are available upon reasonable request from the authors.

Acknowledgments

This work has been carried out within the framework of the EUROfusion Consortium, funded by the European Union via the Euratom Research and Training Program (Grant Agreement No 101052200 - EUROfusion). Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Commission. Neither the European Union nor the European Commission can be held responsible for them. The research group Fusion Group at BSC is grateful for the support received from the Departament de Recerca i Universitats de la Generalitat de Catalunya with code 2021 SGR 00908, and from the Spanish National R&D Project PID2019-110854RB-I00 funded through MCIN/AEI/10.13039/501100011033. This work was supported by the Ministry of Education, Youth and Sports of the Czech Republic through the e-INFRA CZ (ID:90254).

ORCID iDs

X Sáez https://orcid.org/0000-0003-1989-9485
A Soba https://orcid.org/0000-0003-1147-508X
J V Ylla Català https://orcid.org/0000-0002-4711-6815
G Saxena https://orcid.org/0000-0001-7667-5053
M Garcia-Gasulla https://orcid.org/0000-0003-3682-9905
M Komm https://orcid.org/0000-0001-8895-5802
A Podolnik https://orcid.org/0000-0003-1237-8812
J Romazanov https://orcid.org/0000-0001-9439-786X
E Sánchez https://orcid.org/0000-0003-1062-7870
J L Velasco https://orcid.org/0000-0001-8510-1422
M J Mantsinen https://orcid.org/0000-0001-9927-835X

References

[1] EUROfusion Consortium 2023 EUROfusion-realising fusion energy (available at: www.euro-fusion.org)
[2] ITER Organization 2023 ITER-the way to new energy (available at: www.iter.org)
[3] Ciattaglia S *et al* 2017 The European DEMO fusion reactor: design status and challenges from balance of plant point of view *EEEIC / I&CPS Europe*
[4] Litaudon X *et al* 2022 EUROfusion-theory and advanced simulation coordination (E-TASC): programme and the role of high performance computing *Plasma Phys. Control. Fusion* **64** 034005
[5] IFMIF-DONES 2023 The key for the future (available at: https://ifmif-dones.es/)
[6] CIEMAT 2023 National fusion laboratory (LNF) (available at: www.fusion.ciemat.es/home/)
[7] Barcelona Supercomputing Center Extrae user guide 2023 (available at: www.bsc.es/computer-sciences/extrae)
[8] OpenMP ARB 2023 The OpenMP API specification for parallel programming (available at: www.openmp.org)
[9] Message Passing Interface Forum 2021 MPI: a message-passing interface standard version 4.0 (available at: www.mpi-forum.org/docs/mpi-4.0/mpi40-report.pdf)
[10] The Khronos Group Inc 2023 OpenCL overview (available at: www.khronos.org/api/opencl)
[11] NVIDIA Corporation 2023 CUDA zone-library of resources | NVIDIA developer (available at: https://developer.nvidia.com/cuda-zone)
[12] POP CoE 2023 POP standard metrics for performance analysis of hybrid parallel applications (available at: https://pop-coe.eu/further-information/learning-material)
[13] Barcelona Supercomputing Center 2023 Performance optimisation and productivity. a centre of excellence in HPC (available at: https://pop-coe.eu)
[14] Pillet V, Labarta J, Cortes T and Girona S 1995 PARAVER: a tool to visualize and analyze parallel code *Proc. WoTUG-18: Transputer and Occam Developments* ed P Nixon pp 17–31
[15] NVIDIA Corporation 2023 NVIDIA Nsight systems (available at: https://developer.nvidia.com/nsight-systems)
[16] OpenACC-standard.org 2021 OpenACC programming and best practices guide (available at: www.openacc.org/sites/default/files/inline-files/OpenACC_Programming_Guide_0_0.pdf)
[17] Barcelona Supercomputing Center 2023 Technical information | BSC-CNS (available at: www.bsc.es/marenostrum/marenostrum/technical-information)
[18] CINECA 2023 UG3.0: system specific guides (available at: https://wiki.u-gov.it/confluence/display/SCAIUS/UG3.0%3A+System+Specific+Guides)
[19] Komm M, Ratynskaia S, Tolias P, Cavalier J, Dejarnac R, Gunn J P and Podolnik A 2017 On thermionic emission from plasma-facing components in tokamak-relevant conditions *Plasma Phys. Control. Fusion* **59** 9
[20] Komm M, Gunn J P, Dejarnac R, Pánek R, Pitts R A and Podolník A 2017 Particle-in-cell simulations of the plasma interaction with poloidal gaps in the ITER divertor outer vertical target *Nucl. Fusion* **57** 126047
[21] Gunn J *et al* 2017 Surface heat loads on the ITER divertor vertical targets *Nucl. Fusion* **57** 046025
[22] Davis T 2004 Algorithm 832: UMFPACK, an unsymmetric-pattern multifrontal method *ACM Trans. Math. Softw.* **30** 2
[23] UChicago Argonne, LLC and the PETSc Development Team 2023 Petsc users manual (available at: https://petsc4py.readthedocs.io/en/stable/manual/)
[24] Balay S *et al* 2015 Anl-95/11 rev 3.6, argonne national laboratory, PETSc users manual, revision 3.6
[25] Eisenstat S 1981 Efficient implementation of a class of CG methods *J. Sci. Stat. Comput.* **2** 1–4
[26] Freund R, Golub G H and Nachtigal N 1992 *Iterative Solution of Linear Systems, Acta Numerica* (Cambridge University Press)
[27] Romazanov J *et al* 2019 Beryllium global erosion and deposition at JET-ILW simulated with ERO2.0 *Nucl. Mater. Energy* **18** 331–8
[28] Velasco J L, Calvo I, Parra F and García-Regaña J 2020 KNOSOS: a fast orbit-averaging neoclassical code for stellarator geometry *J. Comput. Phys.* **412** 109512
[29] Velasco J L, Calvo I, Parra F I, d'Herbemont V, Smith H M, Carralero D and Estrada T (W7-X Team) 2021 Fast simulations for large aspect ratio stellarators with the neoclassical code KNOSOS *Nucl. Fusion* **61** 11603
[30] Valgrind™ Developers 2023 Valgrind (available at: https://valgrind.org)
[31] Geimer M, Wolf F, Wylie B J, Ábrahám E, Becker D and Mohr B 2010 The scalasca performance toolset architecture, concurrency and computation *Pract. Exp.* **22** 702–19