frontiers | Frontiers in Neuroinformatics

TYPE Original Research
PUBLISHED 26 October 2022
DOI 10.3389/fninf.2022.883742

OPEN ACCESS

EDITED BY
Sharon Crook,
Arizona State University, United States

REVIEWED BY
Patrick K. Quoika,
Technical University of Munich, Germany
Hans Ekkehard Plesser,
Norwegian University of Life Sciences, Norway
Neslihan Serap Sengor,
Istanbul Technical University, Turkey

*CORRESPONDENCE
Erik De Schutter
erik@oist.jp

†These authors share first authorship
‡These authors share senior authorship

RECEIVED 25 February 2022
ACCEPTED 30 September 2022
PUBLISHED 26 October 2022

CITATION
Chen W, Carel T, Awile O, Cantarutti N, Castiglioni G, Cattabiani A, Del Marmol B, Hepburn I, King JG, Kotsalos C, Kumbhar P, Lallouette J, Melchior S, Schürmann F and De Schutter E (2022) STEPS 4.0: Fast and memory-efficient molecular simulations of neurons at the nanoscale.
Front. Neuroinform. 16:883742.
doi: 10.3389/fninf.2022.883742

STEPS 4.0: Fast and memory-efficient molecular simulations of neurons at the nanoscale

Weiliang Chen[1†], Tristan Carel[2†], Omar Awile[2], Nicola Cantarutti[2], Giacomo Castiglioni[2], Alessandro Cattabiani[2], Baudouin Del Marmol[2], Iain Hepburn[1], James G. King[2], Christos Kotsalos[2], Pramod Kumbhar[2], Jules Lallouette[1], Samuel Melchior[2], Felix Schürmann[2‡] and Erik De Schutter[1*‡]

[1]Okinawa Institute of Science and Technology Graduate University (OIST), Okinawa, Japan, [2]Blue Brain Project, École Polytechnique Fédérale de Lausanne (EPFL), Geneva, Switzerland

Recent advances in computational neuroscience have demonstrated the usefulness and importance of stochastic, spatial reaction-diffusion simulations. However, ever increasing model complexity renders traditional serial solvers, as well as naive parallel implementations, inadequate. This paper introduces a new generation of the STochastic Engine for Pathway Simulation (STEPS) project (http://steps.sourceforge.net/), denominated STEPS 4.0, and its core components which have been designed for improved scalability, performance, and memory efficiency. STEPS 4.0 aims to enable novel scientific studies of macroscopic systems such as whole cells while capturing their nanoscale details. This class of models is out of reach for serial solvers due to the vast quantity of computation in such detailed models, and also out of reach for naive parallel solvers due to the large memory footprint. Based on a distributed mesh solution, we introduce a new parallel stochastic reaction-diffusion solver and a deterministic membrane potential solver in STEPS 4.0. The distributed mesh, together with improved data layout and algorithm designs, significantly reduces the memory footprint of parallel simulations in STEPS 4.0. This enables massively parallel simulations on modern HPC clusters and overcomes the limitations of the previous parallel STEPS implementation. Current and future improvements to the solver are not sustainable without following proper software engineering principles. For this reason, we also give an overview of how the STEPS codebase and the development environment have been updated to follow modern software development practices. We benchmark performance improvement and memory footprint on three published models with different complexities, from a simple spatial stochastic reaction-diffusion model, to a more complex one that is coupled to a deterministic membrane potential solver to simulate the calcium burst activity of a Purkinje neuron. Simulation results of these models suggest that the new solution dramatically reduces the

per-core memory consumption by more than a factor of 30, while maintaining similar or better performance and scalability.

KEYWORDS

STEPS, stochastic simulation, molecular neuroscience, HPC, supercomputing

1. Introduction

For several decades computational modeling has progressively proven its importance in neuroscience research, covering a wide range of research domains and disciplines: from sub-cellular molecular reaction-diffusion dynamics to whole-brain neural network simulations. Breakthroughs in experimental methods and community-driven data sharing portals have significantly increased the amount of available experimental data, enabling the advance of complex data-driven modeling and analysis. These efforts are further enhanced by large collaborative projects such as the US BRAIN initiative (Insel et al., 2013), and the EU Human Brain Project (Markram et al., 2011; Amunts et al., 2016, 2019), where complex computational modeling plays an essential role. The rapid progress of neuroscience modeling brings critical advances to our understanding of neuronal systems, yet unprecedented challenges to simulator software development have emerged from two primary directions: first, the need to simulate neuronal functionalities across multiple spatio-temporal scales, and second, the requirement of simulating such systems with extraordinary efficiency.

1.1. The STEPS project and its applications

The STochastic Engine for Pathway Simulation (STEPS) project has evolved following the above trends over the years. The STEPS project started as a mesoscopic scale stochastic reaction-diffusion solution (Hepburn et al., 2012) driven by a spatial variant of the well-known Gillespie Stochastic Simulation Algorithm (SSA) method (Gillespie, 1977). Over the years, serial STEPS has contributed to a wide range of research domains, such as studies on long-term depression in cerebellar Purkinje cells (Antunes and De Schutter, 2012; Zamora Chimal and De Schutter, 2018), viral RNA degradation and diffusion (Schelker et al., 2016), longitudinal anomalous diffusion in neuron dendrites (Mohapatra et al., 2016), and calcium signaling in astrocytes (Denizot et al., 2019). We gradually expanded STEPS to support electrical potential calculation on tetrahedral meshes with the EField solver (Hepburn et al., 2013), allowing combined simulations of reaction-diffusion and membrane potential dynamics on a single mesh reconstruction of neuronal morphology. This solution was important for research that showed that stochastic activation of ion channels, in particular calcium-activated potassium channels, produces significant variability in Purkinje cell dendritic calcium spike shape (Anwar et al., 2013). However, it was soon clear to us that the serial nature of STEPS was the major bottleneck for simulating such complicated models; even a sub-branch of a Purkinje neuron often took weeks to complete one realization of 500 ms biological time. This issue was partially addressed in STEPS 3.0 by introducing the parallel operator splitting method to the reaction-diffusion solution (Hepburn et al., 2016; Chen and De Schutter, 2017), which aided research such as platform development for automatic cancer treatment discovery (Stillman et al., 2021). A parallel EField implementation supported by the PETSc library (Abhyankar et al., 2018) was added to STEPS 3.1. The parallel solution dramatically improved performance by thousand folds compared to the serial counterpart, making it possible to model a complete neuron with detailed morphology and channel mechanisms (Chen et al., 2022).

1.2. The need of a new parallel solver

Moving to parallel STEPS has greatly improved performance compared to the serial solution. However, as the hardware and software of high-performance computing have advanced in recent years, noticeable bottlenecks have been observed in modeling applications with STEPS. The main objective of this article is to identify these bottlenecks and address them with a new parallel implementation.

For many scientific applications, the memory capacity of High-Performance Computing (HPC) systems is one of the main constraints for running simulations at scale. A large number of today's HPC systems have about 2~3GB of main memory per core (Zivanovic et al., 2017). This is an improvement compared to previous BlueGene-like systems where memory capacity is typically ~1GB per core. Current systems are increasingly heterogeneous with the use of accelerators such as GPUs. The memory capacity of such a system is significantly lower compared to what is commonly available on host CPUs. In the case of Intel Knights Landing processors (Sodani et al., 2016), the total capacity is approximately 0.2GB per core. The next generation of processors such as Intel Sapphire Rapids will most likely have

a per-core memory capacity similar to the current generation. This poses a significant challenge to application developers: on the one hand the raw computing power is significantly increasing with architectures like GPUs, while on the other hand maintaining a low memory footprint becomes increasingly important to achieve better performance.

One major limitation of the existing parallel implementation in STEPS comes from the mesh data architecture inherited from the serial solution. While bridging the gap between serial and parallel STEPS and making many non-parallel components reusable, the serial nature of the design requires the complete data of the whole mesh and the molecule state of each mesh element to be stored in every computing core. This poses a hard limit on the maximum model size determined by the per-core memory availability, the model complexity, and the mesh size. Thanks to support from the parallel solver, realistic simulations with a large number of chemical reactions for a great period of biological time can now be accomplished in a reasonable computing time. However, this in turn raises research interests in even more complicated models and more realistic morphologies, reaching the limits of the implementation. The memory constraints in modern HPC systems further amplify such limitations.

The solution to this issue is a new parallel implementation constructed on the foundation of a sophisticated distributed mesh library, Omega_h (Ibanez and Roberts, 2018). Thanks to the distributed nature of the mesh library and the redesigns of other STEPS components, we are able to dramatically reduce the memory footprint of the simulation while maintaining similar or better performance and scalability.

1.3. Other solutions for spatial reaction-diffusion simulations

Traditionally, spatial reaction-diffusion simulation solutions are divided into two major categories, voxel-based and off-voxel particle-based. Voxel-based simulators divide the geometry into small voxels, where the Reaction-Diffusion Master Equation is solved by variants of the Gillespie SSA method (Gillespie, 1977). Example simulators in this category include STEPS (Hepburn et al., 2012), MesoRD (Hattne et al., 2005), and NeuroRD (Oliveira et al., 2010). Off-voxel particle-based solutions represent each molecule in the system individually as sphere-like physical entities, track the Brownian motion of each molecule in a continuum space, and simulate molecular reactions caused by collisions. Example simulators of this category include Smoldyn (Andrews and Bray, 2004), MCell (Kerr et al., 2008), and ReaDDy (Schöneberg and Noé, 2013). Solutions between these two major categories also exist, for instance, Spatiocyte (Arjunan and Tomita, 2010), which simulates individual molecule particle movement with reactions on a hexagonal close-packed lattice.

Some early attempts of parallel spatial reaction-diffusion simulation solutions have been reviewed in Chen and De Schutter (2017). Here we report the latest developments in the field since then. In the voxel-based simulator domain, apart from STEPS 3.x in our previous report, Patoary et al. (2019) further optimized the multi-threading Neuron Time Warp solution, and achieved 5.5x speedup with 7 logical processors, comparing to the single logical processor simulation. In the off-voxel particle-based domain, the ReaDDy 2 simulator reported an approximately sixfold speedup with 11 threads, using single thread simulation as the baseline (Hoffmann et al., 2019). The parallel implementation of Spatiocyte, pSpatiocyte (Arjunan et al., 2020), reported a 7,686x speedup with 663,552 cores on the RIKEN K computer, compared to the 64 core baseline simulation. It is worth noting that direct performance comparisons of these simulators are often challenging, as different theoretical solutions and model abstractions are applied in the implementations.

1.4. Naming conventions and the structure of the article

To avoid confusion, we will hereby call the non-parallel, spatial STEPS solver "serial STEPS," the existing parallel implementation reported in Chen and De Schutter (2017) "STEPS 3," and the new parallel implementation supported by Omega_h that we introduce in this paper "STEPS 4." Note that serial STEPS, STEPS 3 and STEPS 4 are all integrated solutions of the STEPS 4.0 release, and the users are free to choose any of them for their simulations based on the research requirements.

In Section 2, we first describe our design principles and the implementation details of STEPS 4, and then introduce some software engineering techniques applied to the overall STEPS project for maintainability and efficiency improvements. In Section 3, we present the validations of the implementation with a series of well-established models, followed by performance and scalability analysis of results. In Section 4, we further discuss the achievements, limitations and potential solutions of this study, as well as the future development plans for STEPS 4 and the STEPS project in general.

2. Methods

The STEPS development project follows three major methodological principles. First, it aims toward the researchers. STEPS attempts to provide a user-friendly modeling interface, and to progressively reduce the need for manual coding efforts with implementations of auxiliary supports. Second, we focus on improving its performance, as this determines if the simulations can be completed within the expected research time frame.

Third, it aims to be future-proof. Since the first public release, the STEPS project has more than 10 years of history. Over the years, many new standards and solutions in programming and software engineering have been established and become the new standard in software development. Some of them have been adopted in previous STEPS development, but more work is still required to ensure that the software development infrastructure is ready for future project expansions. The following sections detail how these principles are practically applied in the project.

2.1. Code modernization and future-proofing

Although STEPS introduced many new features and additions in the following years since its first release in 2012 the core coding components and style remained relatively unchanged. With this in mind, in this work we have implemented various changes in STEPS in general and adopted modern software design principles to STEPS 4 in particular. All these changes have the aim to reduce bugs, improve maintainability and usability of the code and increase the performance of time-critical data structures and routines.

First, we have adopted the C++17 standard for STEPS. This allowed us to take advantage of modern programming language features, increasing code expressiveness and compactness through meta-programming techniques such as SFINAE (Substitution Failure Is Not An Error). We have also removed raw pointers in favor of references and other safer data-passing and access strategies provided by the C++ standard and the Guidelines Support Library[1]. Second, we have reduced code branching and indirections using meta-programming techniques, which streamline code execution. Third, when choosing container data structures we avoid C++ Standard Template Library (STL) associative containers, which are known to be very inefficient in terms of memory management and performance. The intrinsic arborescent memory layout of `std::map` brings very poor data locality that makes it unusable in computational kernels. Using `std::unordered_map` is a better choice since it uses a contiguous arrays to store the hash values, but its implementation relies on `std::list` to store the values for backward compatibility reasons of the API, which brings back a data locality issue. Because the dataspace of the keys in STEPS 4 is made contiguous, the best data structure based on the STL is `std::vector<std::vector<>>` because the data access is O(1) and data locality, though still flawed, is a bit better than `std::unordered_map` since the values of a key are stored contiguously. Instead, we have designed a new optimized data structure to maximize both access and data locality, the `flat-multimap`.

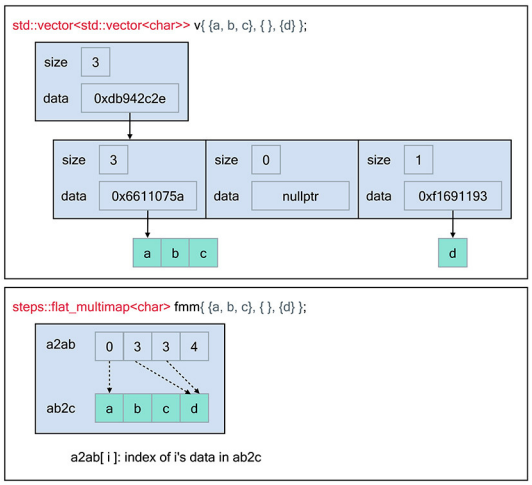

FIGURE 1
Memory layout of the `flat-multimap` container in comparison with a `vector of vectors` container constructed using the Standard Template Library to store the following key-values: `0 → [a, b, c], 2 → [d]`. The `flat-multimap` class relies on 2 member variables, `a2ab` and `ab2c`. *a* is the top element index, *ab* is an index to retrieve the data of *a* in `ab2c`. Data are stored contiguously in `flat-multimap` to reduce heap fragmentation and increase data locality. In contrast, data stored in the STL container are more fragmented. With `flat-multimap`, the values of key `a` are stored in the range `ab2c[a2ab[a]]` and `ab2c[a2ab[a + 1]] - 1`. In this example, values of key 0 are in ab2c[0, 2] i.e [a, b ,c], key 1 has no value since a2ab[1] == a2ab[2], finally values of key 3 are in ab2c[3, 3] i.e [d].

Figure 1 illustrates the memory layout of the `flat-multimap` container in comparison with a naive STL implementation by employing a vector of vectors data structure. The STL implementation exhibits poor data locality as the number of heap allocations required is O(n) whereas `flat-multimap` is O(1) as it always requires 2 allocations. This gives `flat-multimap` several advantages over the STL counterpart. First, it reduces heap fragmentation in the memory. In addition, as the data are stored contiguously in `flat-multimap`, data locality is greatly improved and the solution is more cache-friendly. In exchange, the `flat-multimap` container requires a fixed size and shape upon creation, which can not be changed throughout the simulation. However, this restriction is mostly irrelevant to STEPS 4, as the sizes of the majority of data are determined and fixed by the model.

With the increased complexity of a software, there is a growing concern about introducing bugs in the code that remain undetected. In the best case, these bugs will lead to crashes during runtime. In the worst case, they may silently introduce

1 https://github.com/microsoft/GSL

erroneous results and non-reproducible behavior. Although STEPS runs an extensive validation set to try and ensure this doesn't happen, it is difficult to make sure that every base is covered by such efforts. In an attempt to address this issue at least partially, we introduced C++ vocabulary types meant to indicate to the compiler the different entities used in a STEPS simulation (e.g., `species`, `membrane`, `channel`, `patch`, etc., but also `tetrahedron`, `triangle`, etc.).

A vocabulary type is a type whose name carries a specific meaning in addition to its data. For example, an instance of a class `Width` made of a floating-point value carries both the value and the nature of this value, in opposition to fundamental types like integers or floating-points. Usually fundamental types don't tell much about the meaning of their instances. Vocabulary types can be used to create interfaces comprehensible, expressive, and robust. For instance, vocabulary types can improve functions like below:

```
void process_local_tetrahedron(int index);
```

In the signature of this function, most of the information about the parameter is carried by the variable name and function name, which the compiler cannot use. For the compiler, `process_local_tetrahedron` is only a function that takes a 32 bits integer in parameter. For the developer, this integer is an index of a tetrahedron, local to the current process. Vocabulary types allow us to transfer information traditionally held by the name of the symbols to the typing system by rewriting the signature of the function like this:

```
struct local_tetrahedron_id {
    int value{};
};
void process(local_tetrahedron_id entity);
```

Thus, the compiler is now able to report an issue when the index of one type is erroneously passed to a function expecting another type.

Furthermore, by ensuring the code compiles with GCC, clang, AppleClang and Intel OneAPI, we ensure that language and system compatibility is maintained, further increasing code safety. Numerous compilation flags have been added into our build system, which allow us to spot and fix potential issues in the code early in the development process. We have also moved to a more modular build design where features can be enabled *via* build configuration flags, which also benefits overall software architecture.

Finally, we have tackled software sustainability beyond code modernization. To improve developer confidence and bug detection we have added continuous integration (CI) pipelines into the review process. Proposed patches are automatically built and tested before they can be merged into the development trunk. We have also created a STEPS package for the Spack (Gamblin et al., 2015) package manager. This not only adds a software distribution channel for HPC systems but also provides the developers with a comprehensive build environment that allows them to conveniently test STEPS with various dependency versions and build options. The choice of the underlying libraries (see Section 2.2.2) plays an important role in ensuring that STEPS remains well maintainable, and easily extensible toward new features and use-cases while continuing to support the latest hardware architectures and parallel programming paradigms.

2.2. Implementing a parallel solver with distributed mesh backend

2.2.1. Implementation criteria

To be able to make informed choices about the STEPS 4 implementation, we set early in the development a number of criteria by which to make decisions. Clearly the first and most important criterion is simulation runtime. The goal of the STEPS 4 implementation is to develop a new efficient solution for large-scale modeling with complex geometries. From a user's perspective, the most straightforward and important concern is time-to-solution, how fast a simulation reaches a desired stopping time. For parallel simulations, another important concern is scalability. In high performance computing, parallel scalability is commonly described by two notions, strong scaling and weak scaling. The former describes runtime performance at increasing number of cores and a fixed problem size, while the latter scales the problem size with the number of cores. In practice, the problem size of a STEPS production simulation is often determined by the source materials. Thus, we focus on strong scaling as our parallel performance criterion. STEPS 4 is designed mainly for simulations that run on high performance computing clusters. As mentioned previously, one key characteristic of modern clusters is the large amount of computing cores together with the limited amount of per-core memory, thus memory footprint management is essential to support large scale simulations. We regard it as our third implementation criterion.

These criteria often affect each other in a simulation. For instance, the reduction of memory footprint could substantially improve the efficiency of memory caching, and further improve scalability. Therefore, we do not focus on an individual criterion, but consider them as a whole when making implementation decisions.

2.2.2. Prototyping STEPS 4

Choosing the distributed mesh library with the most suitable abstractions and best performance properties is vital for the success of STEPS 4. This library is the backbone of the whole implementation, providing fundamental data layout and access functionalities, which tightly associates with the criteria

discussed above. Besides performance considerations, from a developer's perspective, the mesh library should also provide a rich and extendable API that can be connected with other STEPS components with ease. Furthermore, while STEPS 4 mainly targets CPUs, the algorithms themselves could in principle be implemented on other hardware architectures and the right abstraction layer should allow a relatively smooth transition toward supporting shared-memory parallelism or GPUs.

To investigate the advantages and drawbacks of different distributed mesh libraries, we used them to implement a series of stand-alone mini-applications to cover a wide range of STEPS functionalities, from simple mesh importing and exporting in a distributed manner, to a functional reaction-diffusion solution integrated with various validations and use case models. These mini-applications were gathered in a library named Zee. Using the Zee library we were able to investigate how different components of STEPS, for example, the operator splitting method, can be implemented on top of different distributed mesh libraries, and to investigate the coding flexibility as well as the performance of our implementations. These investigations provided us essential insight for the choice of a suitable distributed mesh library for STEPS 4, and prototypes for the actual implementation.

We put our evaluation focus on two distributed mesh library candidates, Omega_h (Ibanez and Roberts, 2018) and the DMPlex module from the PETSc library (Abhyankar et al., 2018). Both libraries provide very well-suited features and showed promising performance. The choice of library, however, depends on factors beyond pure technical considerations. On the one hand, PETSc seemed a natural choice since STEPS 3's parallel EField solver already uses PETSc as a backend. Choosing PETSc's DMPlex would eliminate the need for an extra library, as well as the associated data conversions and transfers between libraries. Additionally, PETSc is an extremely well-known and supported library with a large active community. On the other hand, DMPlex is a minor component in the PETSc framework, supported only by few developers and with a small user community. Since the Zee mini-applications revealed that not all functionalities required in STEPS 4 are currently present in DMPlex, and some of which have considerably low priority on the PETSc development roadmap, our choice had to fall on Omega_h.

Omega_h is a C++14 library providing highly-scalable distributed adaptive meshing primitives. Distributed-memory parallelism is natively supported through Message Passing Interface (MPI), while on-node shared-memory parallelism is supported *via* Kokkos (Trott et al., 2022), a C++ library that provides abstractions for parallel execution with OpenMP on CPU and CUDA on GPU. Omega_h ensures a fully deterministic execution. Given the same mesh, global numbering and size field, mesh operations produce the exact same results regardless of parallel partitioning and ordering. This does, however, not extend to changing compilers or hardware. Omega_h is being actively developed and is used for a number of ongoing projects. Moreover, its codebase being much smaller than PETSc, it allowed us to have a comprehensive overview of its capabilities. Despite the lack of documentation, the source code is concise and self-explanatory. Contributing to Omega_h has been much easier than it would have been with PETSc. We were for instance able to add support to the MSH multi-part file format version 4 into Omega_h quite easily.

Omega_h's modern C++ interface was a significant advantage over PETSc as its ease of use allowed us to implement compact yet expressive mini-applications very quickly. We found that the C-oriented API of PETSc makes the library hard to comprehend and is much more error prone than Omega_h's. Additionally, the data management policies of DMPlex are quite complex and require a deep knowledge of PETSc internals as entity data is not directly exposed to the user as it is in Omega_h. This leads to the code being more cluttered and difficult to maintain.

2.2.3. Solver components and the simulation core loop

Fundamentally, STEPS 4 adopts the same operator splitting solution for reaction-diffusion simulation as in STEPS 3 (Hepburn et al., 2016), but with significant differences in the implementation details due to its distributed nature and other optimization goals.

In STEPS 3, the data and operators are intermixed in the solver, and data that are associated may be stored sparsely due to the data structures inherited from previous STEPS implementations. For instance, the molecule state of a tetrahedron and the states of its neighboring tetrahedrons may be stored far away from each other in memory. This is because the molecule state is stored sparsely in individual tetrahedrons together with other data such as mesh connectivity and kinetic processes. This means operator visits to the molecule state often require significant address jumps across memory, decreasing cache efficiency. The bundle of operators and data also make their optimization cumbersome, as new operator solutions or new data structures can not be implemented directly as independent alternatives.

In STEPS 4 one critical implementation change is the separation and encapsulation of different solver components. The two major components are: SimulationData, the data that represents the current state of the simulation, and the operator collection, which are applied to the data so that the simulation evolves to the next state. The simulation state consists of the molecule state M, where the distribution of molecule species is stored and updated, the kinetic process state K, which stores and maintains all kinetic processes such as reactions and surface reactions in the simulation and the information of each kinetic process, including the propensity and update dependencies, and finally the voltage state V of the mesh if voltage-dependent

surface reactions and channels are expressed in the model. The voltage state contains the electrical potential at each vertex of the mesh, as described in Hepburn et al. (2013). The operator collection consists of the operators needed for each step of the simulation core loop, mainly, the reaction SSA operator, the diffusion operator and the PETSc EField operator. As the state data is encapsulated and accessed by operators *via* a unified interface, new operators can be easily developed and provided to the solver as alternative solutions. The encapsulation of simulation states M, K and V also allows the state data to be stored contiguously in memory space, thus improving caching efficiency of the solution.

As mentioned in Section 1, while the kinetic processes and their dependency graphs are partitioned and distributed among computing cores in STEPS 3, all mesh elements and their molecule states are duplicated, leading to high memory consumption and communication overhead when dealing with large scale models. In STEPS 4, the mesh itself is partitioned and distributed, thus each computing core only operates on the data for the sub-domain problem of its associated partition. Ghost layers were implemented for partition boundaries so that simulation states of the boundaries can be synchronized through regular data exchange. This solution ensures a relatively consistent memory footprint for any given sub-domain problem with a fixed partition size, regardless of the size of the overall problem.

Figure 2 schematically illustrates the simulation core loop. When the simulation enters the core loop that advances the simulation state from time T_{start} to $T_{end} = T_{start} + \Delta T$, the simulation period is divided into multiple time windows, whose period is either determined by a user-defined EField period ΔT_{EField} if the EField operator is involved, or equals ΔT otherwise. We call this the EField time window. Each EField time window is then further subdivided by a period of ΔT_{RD}, where ΔT_{RD} is determined by the mesh and the diffusion constants of the simulated model. This is the reaction-diffusion (RD) time window.

At the beginning of each RD time window, the reaction SSA operator is applied to the simulation data repeatedly. Each time, the SSA operator first randomly selects a kinetic process event kp from the kinetic process state K and the event time Δt according to the SSA solution described by the operator and the propensities of the kinetic processes. It then applies the molecule changes caused by the event to the molecule state M, updates the propensities of all kinetic processes that depend on kp in K, and advances the simulation state time for Δt. The SSA iteration stops when the state time reaches the end of the RD time window. As explained in Hepburn et al. (2016) and Chen and De Schutter (2017), the SSA operator is executed independently by each MPI rank without the need for any communication.

At the end of the RD time window, the diffusion operator computes the number of molecules that should diffuse out of each tetrahedron for the time window period ΔT_{RD}. For this calculation the diffusion rates of each diffusive molecule species must be taken into account. The operator then removes them from their original tetrahedrons and redistributes them to their target tetrahedrons. The redistribution is stored in a delta molecule state ΔM, which is then synchronized by Omega_h across all simulation ranks. After the synchronization, each rank applies the changes in ΔM to M for the tetrahedrons it owns, and updates the propensities that are affected by the changes. This completes the operations in a single RD time window.

The solver then repeats this process until the state time reaches the end of the EField time window, at which point the EField operator evolves the voltage state V for the period of ΔT_{EField}, based on the electric currents computed from M and K. This concludes the operations in a EField time window.

If $\Delta T_{EField} < \Delta T$ the EField time window process is repeated, otherwise the simulation core loop is completed and the user regains the simulation control for data inquiry.

2.2.4. Optimization on kinetic process dependency graph

A kinetic process dependency graph describes the update dependency of each kinetic process in the system. Technically, it returns a list of kinetic processes whose propensities must be updated when a certain kinetic process is selected and applied by the SSA operator. Under the operator splitting framework, the reactions in each tetrahedron are independent until the diffusion operator is applied. Therefore, it is possible to divide the dependency graph into independent subgraphs and apply the SSA operator to them separately. This independent graph optimization further compresses the targeting domain of the SSA operator, providing potentially substantial gains in simulation performance.

An example of the optimization for a small model is depicted in Figure 3. This model has two tetrahedrons, each with three volume reactions. One tetrahedron also contains four surface reactions. Each colored node in the figure represents a kinetic process. An arrow goes from one node to the other if the occurrence of an event of the first entails a change in propensity of the second. The whole dependency graph of the model can therefore be subdivided into two independent subgraphs, in red and blue as shown in the figure. Each subgraph can be evolved freely by a SSA operator without the other's interference in a RD time window period.

Note that this optimization heavily relies on the hit rate of drawing SSA events that take place within the time window in each subgraph. Its advantage diminishes and eventually becomes a burden if most of the drawn events happen beyond the time window and are discarded. This hit rate positively correlates to the duration of the time window, the molecule concentrations and the reaction rates. Therefore, this optimization favors simulations with a large RD time window, high molecule concentrations and highly active reactions, but

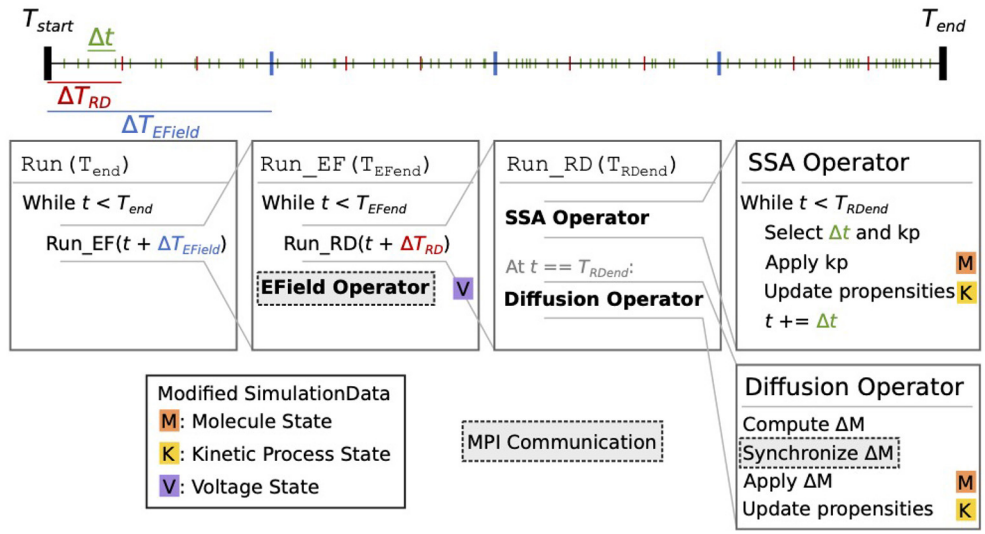

FIGURE 2
Schematic representation of the STEPS 4 simulation core loop. In this example, when running the simulation from T_{start} to T_{end}, the simulation time is first split into ΔT_{EField} time windows (blue ticks). Each ΔT_{EField} time window is further subdivided into ΔT_{RD} time windows (red ticks). Kinetic process events are represented as green ticks, their number in each time window depends on the propensities of the reactions. Current state time is denoted t. The leftmost Run(T_{end}) box is the entry point into the core loop, it splits the time in ΔT_{EField} time windows. The second box (Run_EF(T_{EFend})) runs a full EField time step until T_{EFend}, the end-time that was passed from the first loop. It first subdivides the EField time window in ΔT_{RD} time windows and calls Run_RD for each one. When state time t reaches T_{EFend}, it runs the EField operator. The third box (Run_RD(T_{RDend})) represents one RD time window, it is composed of the SSA operator and the diffusion operator. The SSA operator first selects and applies kinetic processes until the state time reaches T_{RDend}; it's in this loop that the state time is updated. The diffusion operator is then applied: it computes the changes ΔM to the molecule state and applies them. Each of these steps can involve the modification of the simulation data. When it does, a letter with a colored background is present to its right. The letter M with an orange background signifies that this operation modifies the molecule state; the letter K with a yellow background signifies that it modifies the kinetic process state; and the letter V on a purple background signifies that it modifies the voltage state. Finally, steps with a darker background and a dashed outline involve MPI communication between processes.

disfavors simulations with a small time window, low molecule concentrations and less active reactions.

In the STEPS 4 parallel scheme, a simulation core loop is completed after every MPI process finishes its operations, therefore the overall performance of the solution is determined by the slowest computing core. Due to concentration gradients as well as spatial variations of channel density in the model, large scale simulations with complex morphology may exhibit high variability of event drawing hit rate among computing cores. In this case, switching off the independent graph optimization is preferred.

2.2.5. EField solver improvements

Generally, in order to obtain the most accurate results and the best performance, the solver and preconditioner in PETSc need to be tailored to the particular simulation. Previously STEPS 3 used by default the Conjugate Gradient iterative solver (CG) and the Geometric Algebraic Multigrid (AMG) preconditioner. However, performance tests have consistently shown that they do not scale well for large problems. Thus, for STEPS 4 we replaced solver and preconditioner with the widely used Pipelined Conjugate Gradient method (KSPPIPECG) and the Point Block Jacobi preconditioner (PCPBJACOBI), respectively. The same configuration was also applied to STEPS 3 as the new default option. We have not performed a thorough investigation on solvers and preconditioners as it was out of the scope of the present paper.

Another improvement is the distribution of PETSc vectors and matrices for the EField computation. STEPS 3 distributes them equally among computing cores without considering if the mesh elements represented by the matrix partition are owned by the same core. This causes owner mismatches between the EField solution data and the reaction-diffusion solution data, which need to be resolved by expensive cross process data exchanges. In order to avoid this issue, STEPS 4 assembles the vectors and matrices so that each processor only takes care of the degrees of freedom corresponding to the sub-part of the mesh that is owned locally on this processor. This greatly increases data locality and performances since reaction-diffusion and the EField solvers exchange data only locally.

2.2.6. Coupling with other STEPS components

Setting up a simulation in STEPS 4 is mostly done in the same way as in STEPS 3: it involves the declaration of

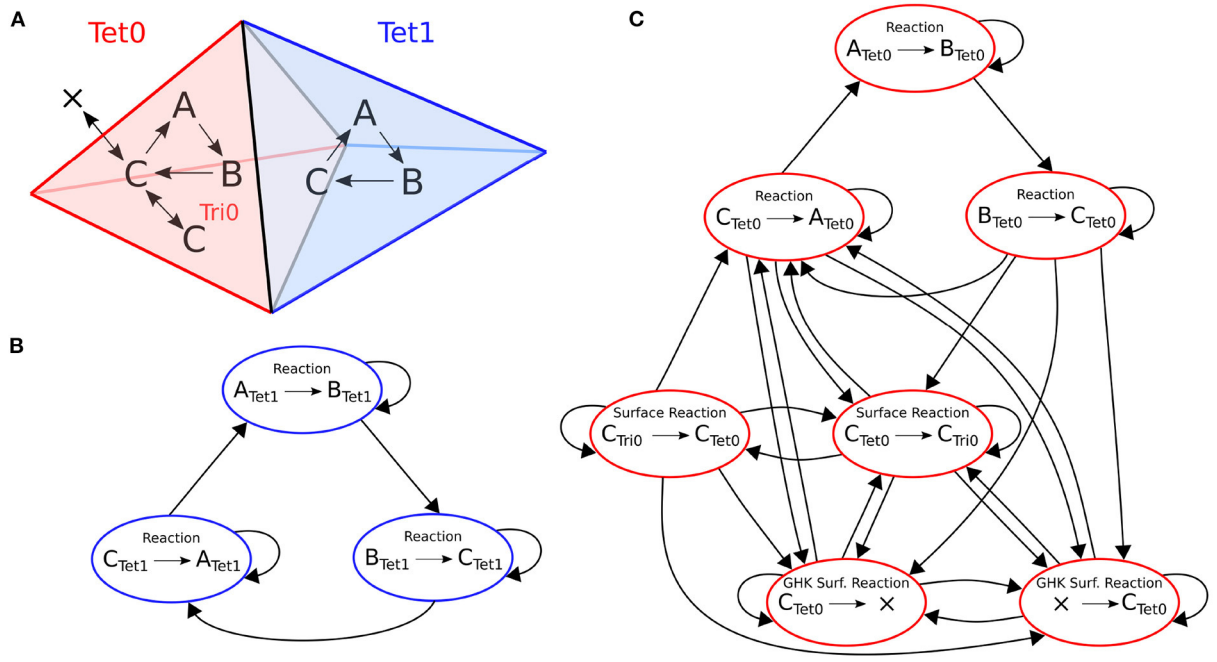

FIGURE 3
Structure of the reaction dependencies graph on a mesh with two connected tetrahedrons labeled 0 and 1. **(A)** The two tetrahedrons and the reactions they contain. Both tetrahedrons contain reactions $A \rightarrow B$, $B \rightarrow C$ and $C \rightarrow A$. Tetrahedron 0 also contains four surface reactions: Species C can be transferred back and forth to a triangle ($C_{tet} \rightarrow C_{tri}$ and $C_{tri} \rightarrow C_{tet}$); and species C can cross the membrane back and forth as a GHK current (since the amount of C outside of tetrahedron 0 is not modeled, it is equivalent to creating and removing species C). **(B,C)** The corresponding reaction dependencies graphs. Each colored node represents a kinetic process. An arrow goes from one node to the other if the occurrence of an event of the first entails a change in propensity of the second. In blue and red are the extracted connected components of the graph.

a biochemical model and a description of the geometry in which the model will be simulated. The biochemical model is composed of species, channels, reactions, diffusion rules and currents that are grouped by volume or surface systems. Although most of the biochemical modeling features available in STEPS 3 are also available in STEPS 4, surface diffusion rules are not yet supported. Internally, the same classes are used for declaring a biochemical model in STEPS 3 and in STEPS 4. While in STEPS 3 tetrahedral meshes were managed with the `TetMesh` class, a different class (`DistMesh`) was added for distributed meshes in STEPS 4. This class inherits from the same `Geom` base class as `TetMesh` but acts as a wrapper around the `Omega_h::Mesh` distributed mesh class. Classes related to the declaration of compartments (`DistComp`), patches (`DistPatch`) and membranes (`DistMemb`) in a distributed mesh are also different from the ones used in STEPS 3. Most notably, as explained in the previous section, while tetrahedral compartments in STEPS 3 are usually built from a list of tetrahedron identifiers, STEPS 4 makes use of physical tags in distributed meshes to create distributed compartment and distributed patches. On solver creation, the `DistTetOpSplit` distributed solver class in STEPS 4 initializes the relevant data structures from the biochemical model and geometry description classes. Although this type of initialization through the python API corresponds to the most frequent use case, the distributed solver can also be used and initialized directly in C++, without requiring the creation of STEPS biochemical model and geometry classes.

2.3. Validation strategy

In order to ensure accurate results, STEPS 4 is validated on a series of published models. We extend the validation pack described in Hepburn et al. (2012, 2016) to validate the reaction-diffusion solver and the basic functionalities of other data structures introduced in the new implementation. The faster validations are integrated into the STEPS release and used in continuous integration while the others are available in the STEPS validation repository [2].

The package also contains fast validations with the EField solution. However, since these models are stochastic models designed to run in a reasonable amount of time, they each contain a small tolerance that could mask minor numerical

2 https://github.com/CNS-OIST/STEPS_Validation

inaccuracies. So as to rigorously test our new methods and implementations in STEPS 4 and ensure even no small loss of numerical accuracy, we go further in this study and investigate STEPS 4 in a series of models, comparing either to STEPS 3 results or analytical solutions to a high degree of accuracy.

Validating stochastic simulation solutions presents several challenges. Often, analytical solutions exist only for a few trivial problems and, even in those cases, the stochastic nature of the simulator makes results fluctuate around the analytical solution depending on the particular seed provided to the random number generator (RNG). Unfortunately, fixing the seeds and numerically comparing STEPS 3 and STEPS 4 results is not a meaningful strategy since the two simulators use RNG streams in different ways. Thus, we validate STEPS 4 in a statistical sense.

2.3.1. Statistical analysis

We extract meaningful statistical data from multiple realizations with different RNG seeds and compare either with STEPS 3 results or the analytical solution when available.

The general steps are:

- Record relevant trace results such as the voltage traces in a particular location in the mesh from multiple realizations of STEPS 3 and STEPS 4 simulations.
- Refine traces to extract key features of the simulation, e.g., the frequency of a spike train.
- Collect refined features among the various simulation runs and statistically compare STEPS 3 and STEPS 4.

The choice of what must be recorded and what are the relevant features depends on the particular model at hand.

In literature, many goodness of fit tests exist. One of the most used is the Kolmogorov-Smirnov test (KS test) (Massey Jr, 1951). It is demonstrated that it produces conservative results in case of discrete distributions (Noether, 1963). Since our analysis also consider peak time stamps which are inherently discretized, we decide to use the Cramér-von Mises test (CVM test) (Cramér, 1928; Von Mises, 1928) for our statistical comparisons between STEPS 3 and STEPS 4, utilizing the Scientific Python (SciPy) library. The null hypothesis is that the two samples come from the same distribution. Perhaps a common misconception is that a p-value below a chosen level such as 0.01 means that the null hypothesis must be rejected and, therefore, the distributions are different. In fact, when comparing two identical distributions the p-value is expected to be uniformly distributed on [0,1], and so if this test is repeated many times one would expect to see a p-value below 0.01 1% of the time. In our tests, where multiple distributions are compared within one model, we reject the null hypothesis only if there is strong evidence that p-values are consistently low, evidenced by significantly more than 1% of the p-values generated being below the 0.01 level.

Conversely, when traces are relatively smooth and the features are few, we study directly the confidence intervals at a 99% confidence level. In this case, we reject the null hypothesis if the mean of the STEPS 3 traces does not lie in the confidence interval of the STEPS 4 traces or vice versa.

3. Results

3.1. Validations

As STEPS 4 contains multiple operator components targeting different sub-systems, such as molecular reaction-diffusion and EField, we carefully select the models and independently validate each component before testing the whole implementation on a complex, real case scenario.

3.1.1. Validations of the reaction-diffusion solver

As mentioned in Section 2.3, the reaction-diffusion validations have been discussed in previous publications and are included in the STEPS validation package. STEPS 4 passes all the validations in the package. For the sake of brevity we do not provide detailed analysis of these validations here.

3.1.2. Validations of the EField solver

To validate the EField solver we use the Rallpack models described in Bhalla et al. (1992), focusing on Rallpack 1 as a basic validation of our solution, and we introduce a new statistical analysis of a stochastic implementation of Rallpack 3.

- Rallpack 1 simulates a simple uniform unbranched passive cable. No randomness is involved in this validation and STEPS 4 results are compared directly to the analytic solution.
- Rallpack 2 model solution is equivalent to Rallpack 1 but based on branching morphology. This mathematical morphological description is in practice very difficult to capture realistically in a mesh (Hepburn et al., 2013), and since Rallpack 1 already provides a basic passive validation we do not provide a Rallpack 2 solution here.
- Rallpack 3 examines the interaction between the EField system and the stochastic channel activities of the well-known Hodgkin-Huxley model (Hodgkin and Huxley, 1952). No analytical solution is available for this test, thus we compare STEPS 3 and STEPS 4 solutions using the statistical validation framework illustrated in Section 2.3.1.

3.1.2.1. Rallpack 1

Rallpack 1 (Bhalla et al., 1992) focuses on the validation of the EField solver in a passive model, with no active properties. It consists of a leaking, sealed straight cable with a current injection

TABLE 1 Parameters for rallpack 1.

Parameters	Value
Leak conductance	0.25 S/m^2
Reversal potential	−65 mV
Resistance	1 Ω
Current	0.1 nA
Cable length	1 mm
Membrane capacitance	0.01 F/m^2
EField time step (ΔT_{EField})	5 µs
Number of tetrahedrons	1,135

(J) at z_{min}. Rallpack 1 setup is depicted in Figure 4A. Current is injected in a leaking cable with sealed ends. Table 1 provides the parameters. A leak channel is introduced on every surface triangle. This is slightly different from the analytic solution setup where the leak is uniformly distributed along the cable. However, the effects should be negligible if the mesh is sufficiently refined.

Without loss of generality, we can focus on the voltage traces at the extremes of the cable, where the voltage taps (V taps) are located. This is because the equations are linear and all the intermediate solutions are super-positions of the results at the extremities.

Figure 4 visually compares STEPS 4 results with the analytic solution. As expected, there is close agreement with mean square errors (mse) $mse_{V_{z_{min}}} = 0.069\text{mV}^2$ and $mse_{V_{z_{max}}} = 0.019\text{mV}^2$. STEPS 3 presents almost exactly the same results. When comparing STEPS 3 with STEPS 4 on the same mesh, the mse is $< 10^{-15}\text{mV}^2$ for both $V_{z_{min}}$ and $V_{z_{max}}$ and is due to numerical precision (results not shown).

Convergence to the analytical solution through mesh refinement proceeds as expected with an initial steep drop followed by a plateau at numerical precision (Supplementary Section S2.1).

3.1.2.2. Rallpack 3

Rallpack 3 is an active model that builds on Rallpack 1 by adding Hodgkin-Huxley sodium and potassium channels, and is simulated on the same simple, uniform, unbranched cable geometry. The model tests ion channel activation as well as spike propagation. Rallpack 3, when run stochastically, presents sources of randomness and the problem cannot be solved analytically. A statistical analysis is employed to study this simulation and validate the code.

The degree of randomness strongly depends on the single-channel conductance and resulting density of channels, which are parameters that must be introduced when running the model stochastically. Using biologically-plausible values for single-channel conductance, with 20 pS the Rallpack 3 model demonstrates a significant number of failed spikes as illustrated in Figure 5A. Even if this behavior is an interesting stochastic effect, it strongly hinders statistical analysis. For this reason, we chose single-channel conductance of both sodium and potassium channels to be 4 pS. This almost entirely extinguishes failed spikes whilst maintaining biological plausibility. Figures 5B,C and the additional studies in Supplementary Section S2.1.1 were produced using single-channel conductance of 4 pS.

The two sample sets consist of 10,000 simulation runs each performed with STEPS 3 and STEPS 4 respectively. As for Rallpack 1, we record voltages at the extremes of the cable ($V_{z_{min}}$ $V_{z_{max}}$) (the raw traces).

The voltage trace at z_{min} presents a high peak of ~40 mV followed by a regular spike train with peaks just surpassing 20 mV. The spike train at z_{max} has no bigger spike at the beginning and spike peaks are above 40 mV. For both traces valleys are at ~-65 mV and frequencies are ~69 Hz. The simulated time span is 250 ms.

Given that traces are spike trains with, possibly, a single greater initial peak, the key features extracted and statistically analyzed are:

- peak heights;
- peak timestamps.

The null hypothesis is that STEPS 3 and STEPS 4 simulation results come from the same population, in other words, the simulations are identical. We use the CVM test to refute it with a 99% confidence level. In order to study uncorrelated events we divide the two sample sets into 100 batches each with 100 samples and we compare each STEPS 3 batch with each STEPS 4 batch, producing a set of p-values. Thus, for each key feature (e.g., time stamp of peak number 3) we obtain 10,000 p-values. If the two initial samples are taken from the same population, p-value distributions are expected to be uniform (Murdoch et al., 2008). If the number of p-values below 0.01 is higher than would be expected from a uniform distribution, we refute the null hypothesis. Figures 5B,C present the p-value distributions for peak heights and time stamps as boxplots. For the sake of clarity and brevity here we show only the results for the traces at z_{max}. At z_{min} the results are qualitatively identical. We briefly recall here that the boxplot of a uniform distribution of p-values is centered around 0.5, the median is at 0.5, min and max are at 0 and 1 and Q1 and Q3 quartiles are at 0.25 and 0.75, respectively. All the boxplots follow this trend.

For these reasons, we cannot refute the null hypothesis and we accept that the two samples are taken from the same population.

Supplementary Section S2.1.1 offers a thorough overview of the peak statistics (distributions, means, and standard deviations) while Supplementary Section S2.2 reports all the p-value distributions in detail.

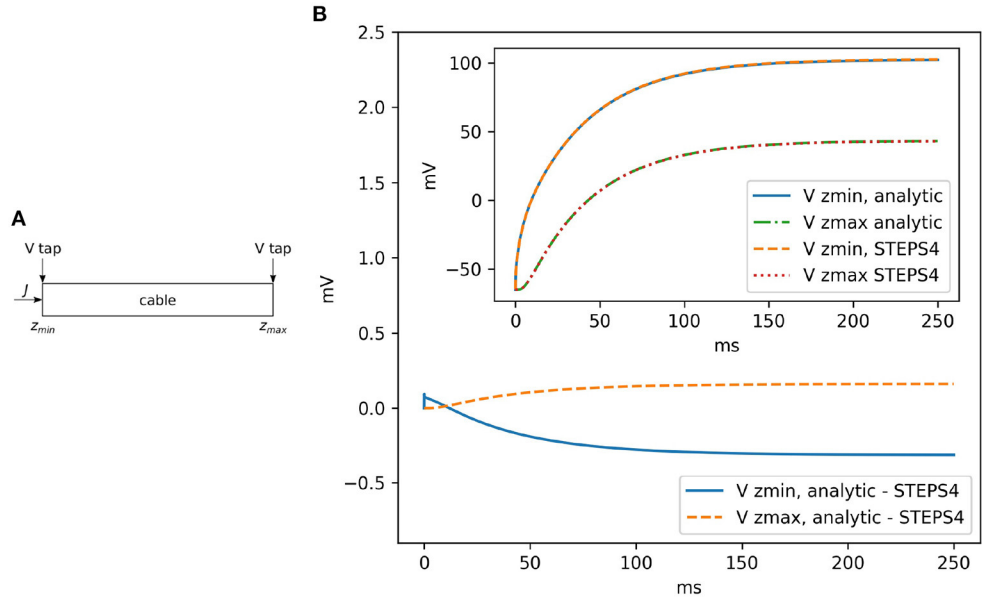

FIGURE 4
(A) The general setup for Rallpack 1. Current is introduced into a leaking cable with sealed ends. Voltage is recorded at the extremities (V taps). **(B)** The voltage difference between the analytic solution and STEPS 4 at z_{min} and z_{max}. The inset shows the overlapping curves.

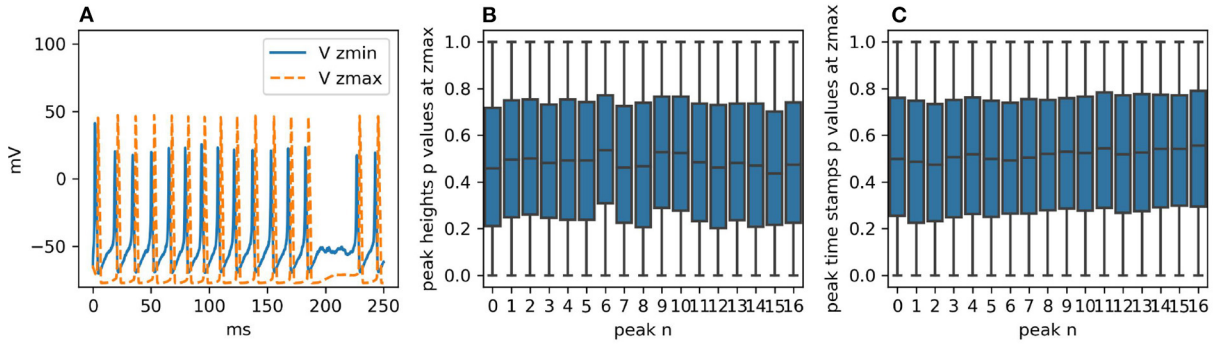

FIGURE 5
(A) Voltage traces at z_{min} and z_{max} for one realization of the simulation with STEPS 4. Failed spikes can occur with single-channel sodium and potassium conductances of 20 pS, as shown, but are eliminated with 4pS. **(B)** With single-channel conductance of 4pS, boxplots for each peak height at z_{max} of the p-values generated by dividing the two samples in 100 batches of 100 runs and then comparing them with the CVM test. As expected, the distributions are uniform. **(C)** The same analysis for the peak heights. Their intrinsic discretization does not affect the p-value distributions.

3.1.3. Validation of the reaction-diffusion and EField combined solution

Finally, we validate all components of the STEPS 4 simulator together by combining reaction-diffusion and EField features and their possible interactions.

3.1.3.1. The calcium burst model

A previously published calcium burst model (Anwar et al., 2013) is selected for the full validation. It contains most of the modeling features supported by STEPS 4, such as regular molecule reaction-diffusion events, ligand-based channel activation and electric potential dynamics. Thus, it contains all the mechanics required to validate STEPS 4 as a whole. Minor modifications are applied to the original model in Anwar et al. (2013) in order to run on a full dendritic mesh, as opposed to the sub-branch mesh used in previous studies. Figure 6A illustrates the full dendritic morphology. The full dendritic mesh was created from reconstruction retrieved from

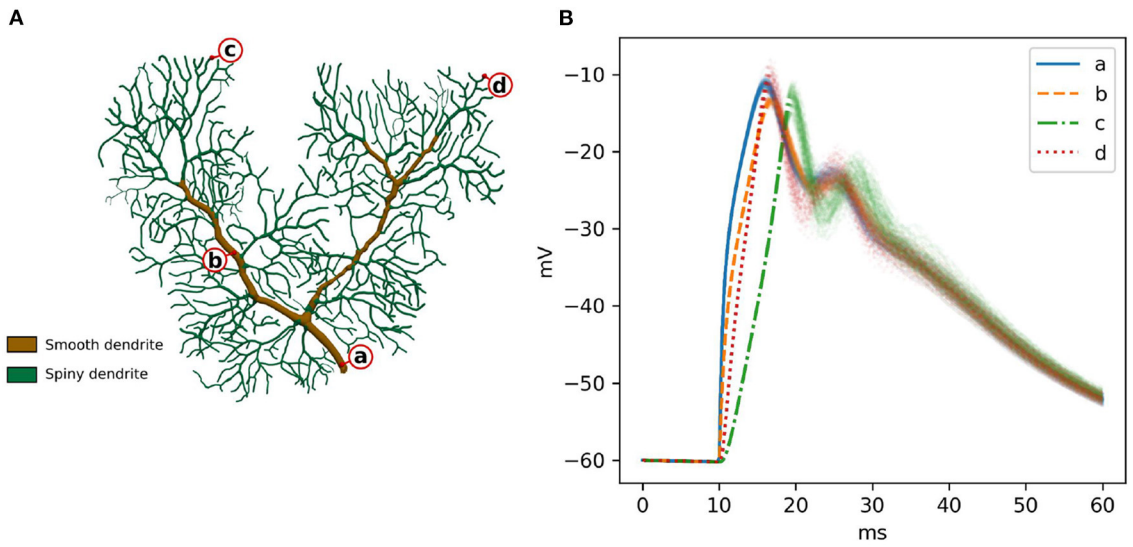

FIGURE 6
(A) The Purkinje dendrite mesh reconstruction for both calcium burst models. The mesh consists of 853,193 tetrahedrons. Dendrite elements are further classified and annotated into two components, representing smooth dendrite and spiny dendrite. **(B)** Raw voltage traces for 100 runs of STEPS 4 at the four different spatial locations indicated by a–d in **(A)**. After the first depolarization to ~18 ms the systems starts to behave stochastically coinciding with calcium-activation of potassium channels.

NeuroMorpho.Org (Ascoli et al., 2007), data ID: NMO_35058 (Anwar et al., 2014)[3]. The calcium burst model is also used to analyze the performance of the implementation in Section 3.2.

In order to sample a good representation of the dendritic tree we recorded voltage at the four disparate points shown in Figure 6A. Two points (a and b) were recorded from the smooth part of the dendrite, characterized by high diameter and low capacitance, and two (c and d) in separate regions of the spiny dendrite, characterized by low diameter and high capacitance. Figure 6B presents these traces for 100 runs of STEPS 3. In brief and as described in Anwar et al. (2013), AMPAR channel activation by a simulated glutamate burst beginning at 10 ms gives a strong depolarization, and corresponding activation of $Ca_v2.1$ P-type calcium channels gives rise to a peaks at ~18 ms and ~28 ms. Calcium activity activates mslo BK and SK2 calcium-activated potassium channels, producing the repolarization.

As for Section 3.1.2.2, our null hypothesis is that the two simulators run the same simulation and results are picked from the same population. We try to refute this statement, computing the confidence intervals of the averages of the traces at 99% probability. By definition, the confidence intervals mark a region where the trace average lies with 99% probability. Thus, if the average of the traces of the STEPS 3 set does not lie in between the confidence intervals of the STEPS 4 set or vice versa we reject the hypothesis. Figure 7 presents average and confidence intervals for all the four traces. Since confidence bands are extremely narrow, each picture is also shown with the average of the averages removed. This greatly enhances the small differences that exist between the two simulation results. STEPS 4 averages almost always lie in the confidence intervals of the STEPS 3 simulation set and vice versa. For these reasons we cannot reject the null hypothesis and we consider STEPS 4 validated even in this complex scenario.

3.2. Performance

We evaluated the performance of the implementation using three models with gradually increased complexity to cover the use cases from a wide range of research interests. The first one is a simple reaction-diffusion model on a simple cuboid mesh (we term the "simple" model). In the second model, we simulate the background activities of the calcium burst model to investigate the performance of the reaction-diffusion solution on complex Purkinje cell morphology with resting calcium activity (the "background" model). Finally, in the third model we simulate the complete calcium burst model by adding calcium channels, potassium channels and AMPAR activation (see Figure 6) to study how the combined solution performs with a real world model (the "complete" model). The simple model and the background model have previously been used to study performance and scalability of the reaction-diffusion operator splitting solution in STEPS 3

3 http://neuromorpho.org/neuron_info.jsp?neuron_name=10-2012-02-09-001

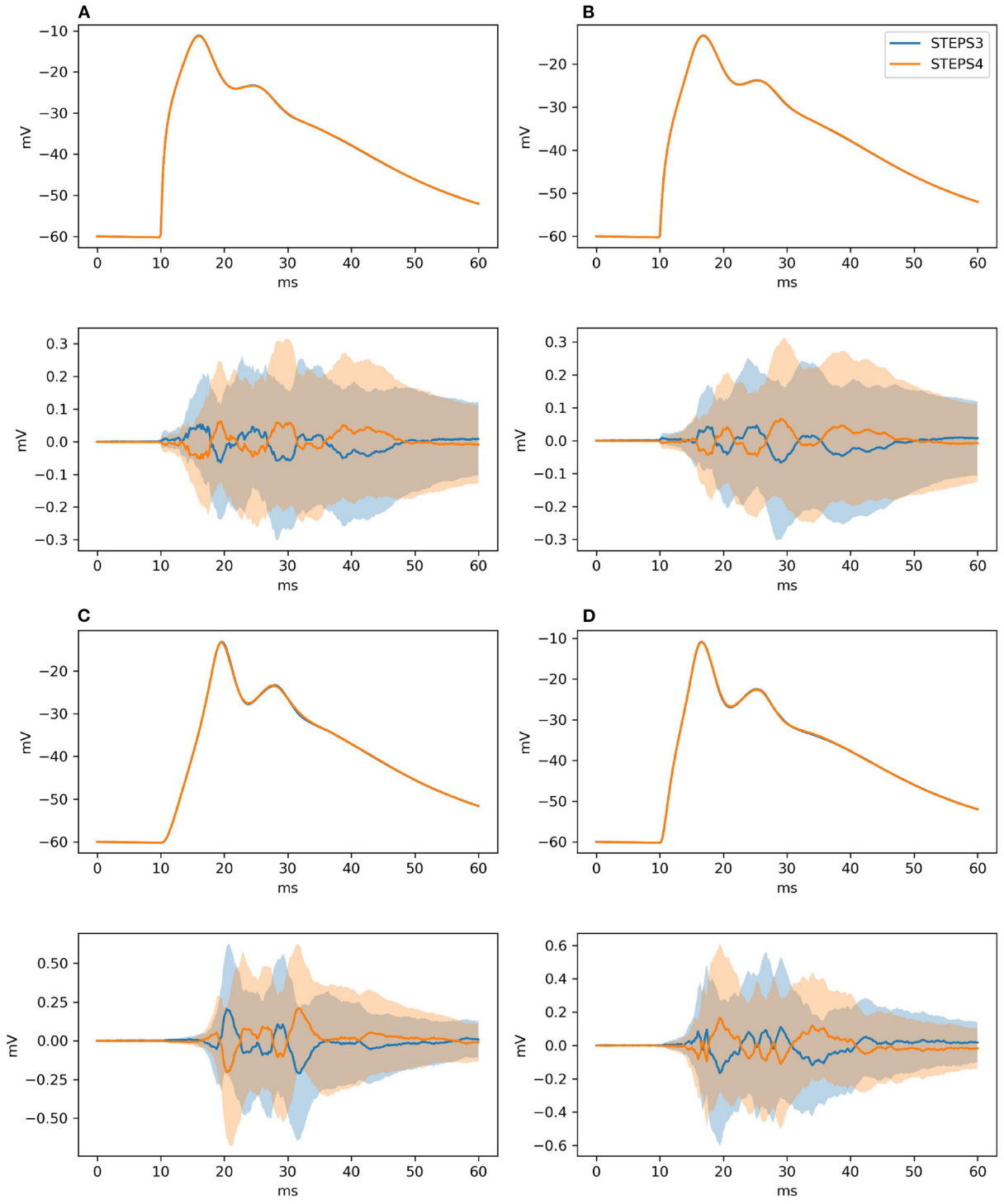

FIGURE 7
The panels illustrate average and confidence intervals (at 99% probability) of STEPS 3 and STEPS 4 simulation sets for the voltages measured at: **(A)** root and **(B)** middle point on the spiny membrane, and on **(C)** left and **(D)** right tip on the smooth membrane. Since confidence intervals are extremely narrow, the lower subplot in each panel presents the results relative to the average of all traces so that the confidence intervals can be seen clearly. Each sample consists of 100 runs. Since averages lie everywhere in each other confidence intervals we cannot refute the null hypothesis.

TABLE 2 Hardware and software configurations of the Blue Brain 5 (Phase 2) supercomputer.

	System	HPE SGI 8600
Hardware	Compute Node (880×)	2× Intel Xeon Gold 6248 Cascadelake @2.5*GHz* (20 physical cores per CPU)
	Memory	384*GB* of main memory (12 × 32*GB* DDR4-2933 DIMMS)
	Network	InfiniBand EDR 100*Gbps* / Fat-tree topology
	Accelerator	GPFS/ IBM Spectrum Scale Filsystem (6.2*PB*)
Software	Compiler	GCC C++ compiler 9.3.0
	Operating System	Red Hat Enterprise Linux Server 7.9
	MPI	HPE MPI (SGI MPT) 2.25
	Python	3.8.3
	Linked Libraries	PETSc 3.14.1, Omega_h 9.34.6, Intel MKL 2018.3, Eigen 3.3.8, SUNDIALS 2.7.0, mpi4py 3.1.3, NumPy 1.21.4, GMSH 4.9.0

Supercomputer.

(Chen and De Schutter, 2017). As the implementation has been improved since the initial implementation, and the hardware used in the previous research is now outdated, new simulation series of these two models are performed to acquire up-to-date results for comparison. The parallel performance of the combined solution with the complete calcium burst model has not been reported previously. We also investigate the effect of the independent graph optimization on the simple model with different molecule concentration setups. We disable this optimization for the calcium burst background and complete models as the complex morphology of Purkinje cell could lead to poor SSA event hit rates in some partitions, and worsens the overall performance of these simulations if this optimization is enabled.

3.2.1. Benchmarking setup

All simulation benchmarks were run on the Blue Brain 5 (BB5) supercomputer hosted at the Swiss National Computing Center (CSCS) in Lugano, Switzerland. A complete description of the hardware and software configuration details of the BB5 system are provided in Table 2. All benchmarks were executed in pure MPI mode by pinning one MPI rank per core. As the number of cores used for simulation needs to be a power of 2 (see Supplementary Section S1.1), for each series of benchmark we first choose an initial core count as a baseline and then double the core count.

The code instrumentation for the performance measurement in STEPS is performed through an *Instrumentor* interface. This is a light wrapper that allows for marking/profiling code regions of interests either by calling a start/stop method or by *C++* Resource Acquisition Is Initialization (RAII) style. Various backends are used by this interface, in particular in this work we use Caliper 2.6 (Boehme et al., 2016), and LIKWID 5.2.0 (Treibig et al., 2010).

For each benchmark configuration, we repeat the simulation 30 times, and show the average results in the figures. The standard deviations of the results are reported as the error bars for each data point in the figures. Per-core memory consumption of each simulation is also measured using the *psutil* Python module (Rodola, 2020) and reported. The comparisons are mainly conducted between STEPS 3 and STEPS 4. For the scalability studies, we also compare the results with the theoretical ideal speedup scenarios. We further investigate the contribution and scaling properties of operator components in STEPS 4, namely, the SSA operator, the diffusion operator and the EField operator, by measuring their individual speedup as well as the proportion in the overall simulation time cost.

3.2.2. The simple model

We reuse the simple model in Chen and De Schutter (2017) which consists of 10 diffusing species with different initial molecule counts within simple cuboid geometry with 13,009 tetrahedrons. These species interact with each other through 4 different reversible reactions with different rate constants. The details of the model can be found in Table 3. We choose 2 cores as the performance baseline and increase the core count to $2^{11} = 2,048$ as the maximum. Note each core has less than 10 tetrahedrons with this maximum, at which point it is unlikely that the simulations remain scalable. However, the result is still interesting as it illustrates the behavior of our solution under extreme scaling scenarios.

The effect of the independent graph optimization is also investigated using the simple model with different initial molecule counts. We first simulate the model in Table 3 without the optimization and use it as the baseline configuration. We then modify the baseline model with four new settings, the first two reduce the initial count of each molecular species by 10x and 100x, and the other two increase molecule counts by 10x and 100x. We name these simulation series "0.01x," "0.1x,"

TABLE 3 Species and reactions as well as the initial configuration of the simple model.

Species	Diffusion coefficient (μm^2 / s)	Initial count
A	100	1,000
B	90	2,000
C	80	3,000
D	70	4,000
E	60	5,000
F	50	6,000
G	40	7,000
H	30	8,000
I	20	9,000
J	10	10,000
Reaction	**Rate Constant**	
$A + B \rightleftharpoons C$	$k_f : 1,000(\mu M \cdot s)^{-1}, k_b : 100s^{-1}$	
$C + D \rightleftharpoons E$	$k_f : 100(\mu M \cdot s)^{-1}, k_b : 10s^{-1}$	
$F + G \rightleftharpoons H$	$k_f : 10(\mu M \cdot s)^{-1}, k_b : 1s^{-1}$	
$H + I \rightleftharpoons J$	$k_f : 1(\mu M \cdot s)^{-1}, k_b : 1s^{-1}$	

"1x," "10x," and "100x" respectively. We also repeat these series with independent graph optimization enabled and record the results for comparison. As this optimization solely targets the SSA operator, a single core is used to run the simulation series, and the time cost of the SSA operator instead of the overall simulation time cost is measured.

Simulation results of the simple model are summarized in Figure 8. Both STEPS 3 and STEPS 4 implementations demonstrate a steady decrease of simulation time early on until $2^6 = 64$ cores, and maintain roughly the same time cost for the rest of the configurations. The memory footprint improvement from STEPS 4 is significant. In the baseline simulations, STEPS 4 consumes 45.6MB of memory per core, about 60% of the required memory for STEPS 3. When simulating the model with thousands of cores, the memory consumption of STEPS 4 further decreases to about 4.5MB per core, 10% of the baseline simulation consumption, thanks to the completely distributed nature of the solution. While the memory footprint of STEPS 3 simulations also decreases with high core counts, the number stabilizes at 16MB, 2.6 times more than STEPS 4 requires. The strong scaling speedup for both STEPS 3 and STEPS 4 in Figure 8C suggests that the STEPS 4 achieves close-to-ideal speedup until $2^6 = 64$ cores, reflecting the time cost result in Figure 8A. In fact, the SSA component further maintains a linear speedup until $2^9 = 512$ cores according to the component scalability analysis in Figure 8D. However, due to the high scalability, its proportion in the overall time cost reduces significantly in high core count simulations. For these simulations, the diffusion operator and other background maintenance routines become the two major proportions of the simulation time cost.

The performance difference caused by the independent graph optimization is illustrated in Figure 8F by the ratio between enabling and disabling the optimization. In the baseline 1x simulations and other series with reduced molecule counts, enabling the optimization results in a slight performance decrease as the SSA time cost ratios in these series are all above 1.0, ranging from 1.15 in the 0.01x series, to 1.09 in the 1x series. The benefit of the optimization is noticeable in the 10x series with a ratio of 0.97, and becomes significant in the 100x case, which shortens more than half of the simulation time. These results agree with our analysis in Section 2.2.4.

3.2.3. The calcium burst background model

We extend our investigation on the reaction-diffusion component with the calcium burst background model with complex Purkinje cell morphology as described in Section 3.1.3.1. There is no voltage component nor any ion channels in this model, only background buffering reaction and diffusion. In total, the model consists of 15 molecule species, 8 of which are diffusive, and 22 reactions. The simulated mesh consists of 853,193 tetrahedrons. To eliminate any difference caused by partitioning, we pre-partition the mesh in Gmsh then import the partitioned mesh to the simulations, therefore the partitioning is always the same for each benchmark configuration. We start the simulation series from $2^5 = 32$ cores within a single node, then double the core count each time until the maximum of 512 nodes with $2^{14} = 16,384$ cores is reached.

Figure 9 presents the key results of the simulation series. In general, STEPS 4 performs slightly worse than STEPS 3 in low core count configurations, but eventually achieves similar performance as the core count increases. This is because currently STEPS 4 implements the widely accepted Gibson and Bruck (Gibson and Bruck, 2000) next reaction method as the default SSA operator. This method provides logarithmic computational complexity with simple data structures that we find suitable for the distributed solution. On the other hand, STEPS 3 inherits the serial implementation of the Composition and Rejection method (Slepoy et al., 2008), which requires a more complex data structure but takes advantage of its constant time complexity, particularly when dealing with large number of reactions in low core count simulations. It is worth noting that the compartmental design in STEPS 4 supports multiple operator implementations, therefore more efficient operators can be easily integrated to the solution in the future.

Dramatic improvement in memory consumption can be observed for STEPS 4 in Figure 9B. All STEPS 3 simulations require no less than 2GB of memory per core; on the other hand, the highest per-core memory footprint for STEPS 4 is about 630 MB with $2^5 = 32$ cores, and drops down to about 67MB with $2^{10} = 1,024$ cores and above, roughly 3% of what is required by STEPS 3.

Both STEPS 4 and STEPS 3 demonstrate linear to super-linear speedup until $2^{12} = 4,096$ cores in Figure 9C. Component scaling analysis in Figure 9D suggests that both

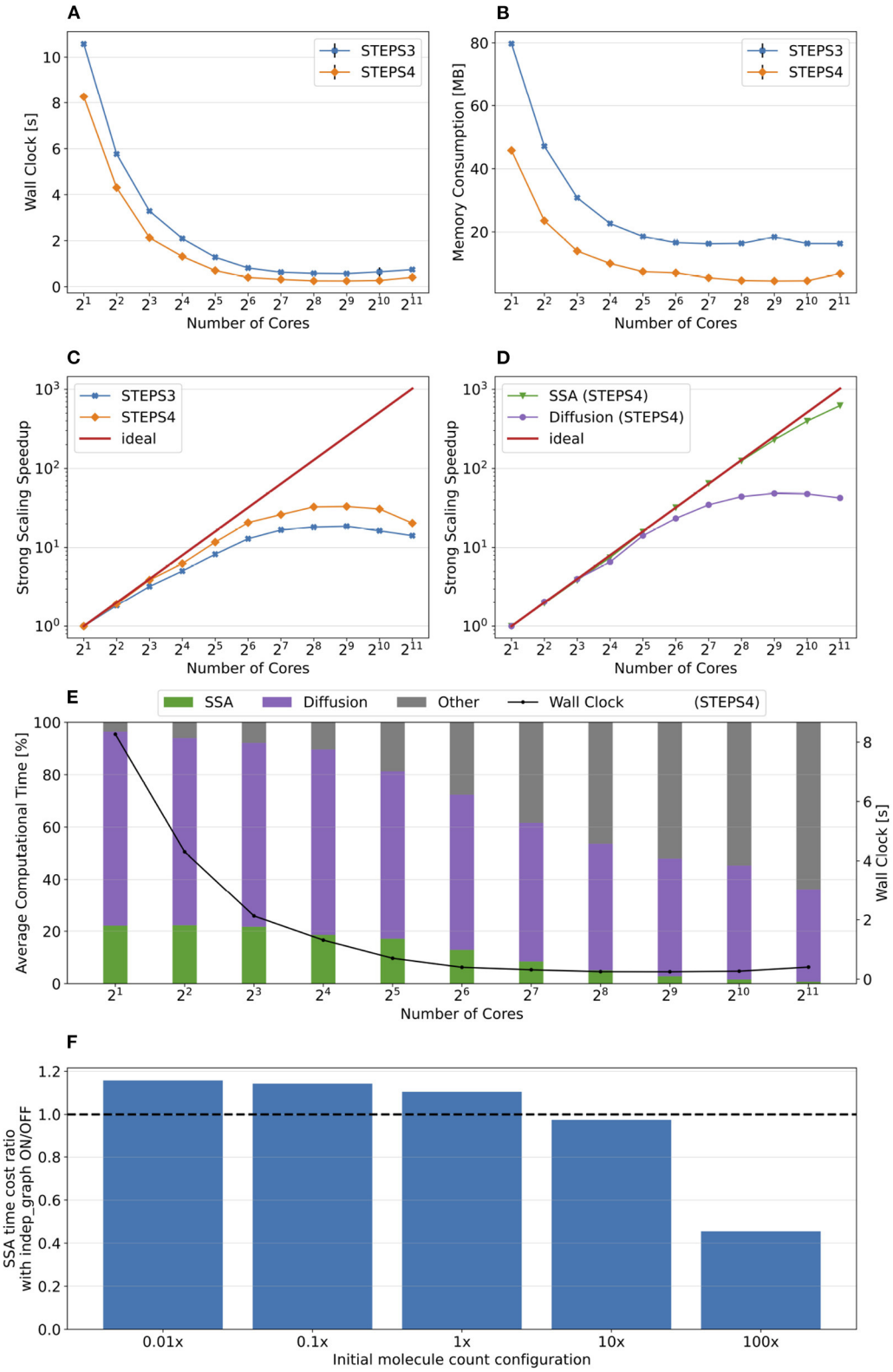

FIGURE 8
The performance results and scalability of the simple model. **(A)** Both STEPS 3 and STEPS 4 implementations demonstrate a steady decrease of time cost early on, then maintain similar time cost beyond $2^6 = 64$ cores. **(B)** STEPS 4 consumes significantly less per-core memory than STEPS 3, ranging from 60% in the baseline simulation, to approximately 30% in high core count simulations. **(C)** STEPS 4 achieves close-to-ideal

(Continued)

FIGURE 8 (Continued)
speedup until $2^6 = 64$ cores, but has poor scalability afterward. Similar but slightly worse scalability can be observed for STEPS 3. **(D)** Component scalability analysis of STEPS 4. The SSA operator shows a linear speedup until $2^9 = 512$ cores. **(E)** Component proportion analysis of STEPS 4. Due to the highly scalable SSA operator, the time cost of high core-count simulations is dominated by the diffusion operator and other non-scalable bookkeeping routines, resulting in poor scalability in high core count scenarios. **(F)** The SSA operator time cost ratio with and without independent graph optimization in different initial molecule count setups. Enabling the optimization results in performance decrease in low molecule density simulations, but provides significant speedup in simulations with high molecule density.

the SSA and the diffusion operators contribute to this result. The diffusion operator maintains close-to-linear speedup until $2^{12} = 4,096$, while the SSA operator demonstrates super-linear speedup throughout the series. We investigated this scaling behavior and further profiling on the SSA operator indicates that the super-linear speedup mainly comes from the update routine of the operator, including the propensity calculations and the priority queue updates. This suggests that the improvement on memory caching may play an important role here.

The diffusion operator is the dominating component in this series, as shown in Figure 9E. Its proportion in the overall time cost increases from 65 to 95%. The proportion of other non-scaling routines also rises but is still less than 10% with the maximum $2^{14} = 16,384$ cores. Overall the performance profile of the background model is very similar to the simple model profile. This is not surprising as they both involve the same operators but the background model has more tetrahedrons and reactions per core compared to the simple model.

3.2.4. Complete calcium burst model

The complete calcium burst model as described in Section 3.1.3.1 extends the background model by coupling molecular reaction-diffusion updates with voltage-dependent channel activation as well as membrane potential changes. Different channel density parameters are assigned to the smooth and the spiny sections of the mesh to approximate the effect caused by regional spine density difference. The model consists of 15 regular species, 8 of which are diffusive, 5 types of channels with in total 27 different channel states, 59 regular reactions and 16 voltage-dependent reactions. Compared to the previous two models, the complete model produces a simulation with extremely complex dynamics and imbalanced computational load, both spatially and temporally. We consider it as an excellent demonstration of STEPS 4 performance in realistic research projects.

Figure 10 summarizes the key results of the simulation series. While STEPS 4 performs slightly worse than STEPS 3 initially, it reaches similar performance with $2^9 = 512$ cores, and outperforms STEPS 3 for the rest of the series. As expected, STEPS 4 continues its advantage on per-core memory footprint management, starting from 1.5GB for $2^5 = 32$ core simulations, to approximately 500MB for $2^9 = 512$ cores and above. The minimum memory requirement for STEPS 3 is 5GB, 10 times what is needed with STEPS 4. While the BB5 cluster has high memory capacity per compute node and is able to provide 12GB of memory per core for simulations (given the 32 active processes per node), many HPC clusters commonly have the memory capacity restriction of about 4GB per core (Zivanovic et al., 2017), therefore only STEPS 4 simulations can be run on those clusters.

Overall, STEPS 4 achieves a better scalability compared to STEPS 3, with linear speedup from the diffusion operator, and the super-linear speedup from the SSA operator. However, the EField operator has limited scalability, reaching maximum 10x speedup relative to the baseline. This results in a great increase of EField operator time cost in proportion to the total computation time, from 10% in the baseline simulations to 76% in the $2^{14} = 16,384$ core simulations, making it the major performance bottleneck of the series, as shown in Figure 10E.

3.2.5. Memory footprint with refined mesh

As shown in the above results, the significantly reduced memory footprint is one of the major advantages of STEPS 4. To further investigate the memory consumption difference between STEPS 4 and STEPS 3, we refine the Purkinje cell mesh and rerun both the calcium burst background model and the complete calcium burst model with the new mesh. The refined mesh consists of 3,176,768 tetrahedrons. For simplicity, we name the original mesh as the "1M" mesh, and the refined mesh as the "3M" mesh accordingly. As the 3M simulations exhibit similar performance profiles as the 1M versions, we focus on the memory footprint of the simulations. Performance and scalability results of the 3M simulations can be found in the Supplementary Section S2.3.

Figures 11A,B provide an overall view of the results. For all simulation series, the baseline configuration, i.e., the one with the lowest core count, has the highest memory footprint, then progressively reduces to a consistent minimum. This is essential as any cluster with per-core memory capacity below the minimum can not execute the simulation regardless how many cores are available. Thus, we hereby use the minimum memory consumption from each series for comparison. Figure 11A presents the memory consumption of the background model, for both STEPS 3 and STEPS 4, and for both the 1M and 3M meshes. For the 1M mesh simulations, STEPS 4 requires 67MB memory per core, while STEPS 3 requires approximately 2GB, 30x of the STEPS 4 requirement. For the 3M mesh simulations,

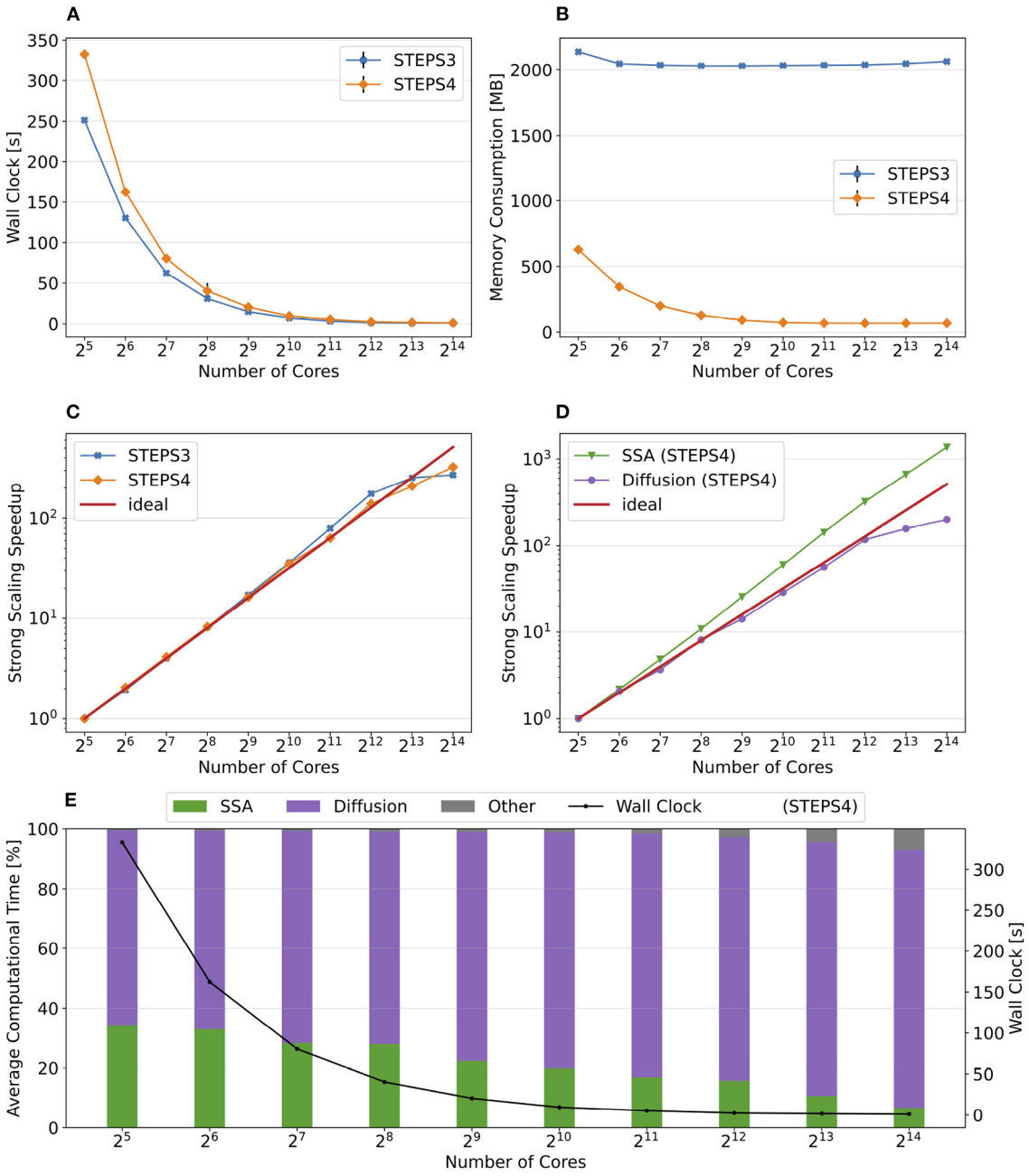

FIGURE 9
The performance results and scalability of the calcium burst background model. **(A)** Steady decrease of simulation time cost can be observed in both STEPS 4 and STEPS 3 simulations. STEPS 4 performs slightly worse than STEPS 3 in low core count simulations, but both eventually achieve similar performance as core count increases. **(B)** The memory footprint of STEPS 4 is superior compared to the STEPS 3 counterparts, requiring about 630MB for $2^5 = 32$ core simulations, and 67MB for $2^{10} = 1,024$ core and above simulations. STEPS 3 consumes more than 2GB of memory per core for the whole series. **(C)** Both STEPS 4 and STEPS 3 demonstrate linear to super-linear scaling speedup. **(D)** Component scalability analysis of STEPS 4 suggests that the diffusion operator in STEPS 4 exhibits linear speedup until $2^{10} = 1,024$ cores, while the SSA operator shows a remarkable super-linear speedup throughout the series. **(E)** Component proportion analysis of STEPS 4. The diffusion operator is the dominating component, taking from 65 to 95% of the overall computational time.

200MB memory per core is required by STEPS 4, while 6.6GB is required by STEPS 3, about 33x of the STEPS 4 requirement. Results of the complete model are shown in Figure 11B. For the 1M series, STEPS 4 requires about 500MB of memory, while STEPS 3 requires approximately 5.1GB, resulting in a 10x difference. For the 3M series, the memory footprint of

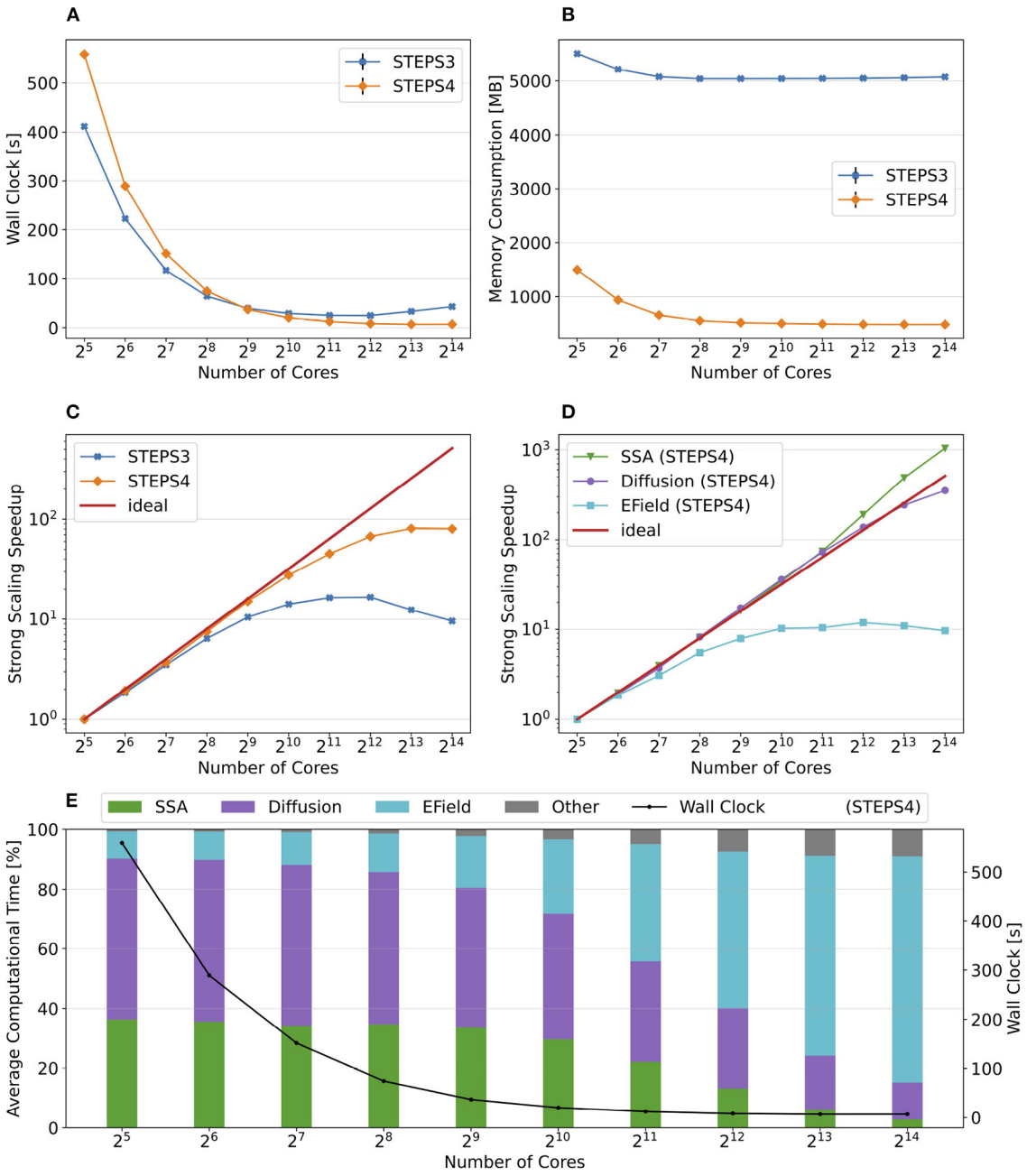

FIGURE 10
The performance results and scalability of the calcium burst complete model. **(A)** STEPS 4 performs slightly worse than STEPS 3 in low core count simulations, but reaches similar performance with $2^9 = 512$ cores, and outperforms STEPS 3 afterward. **(B)** STEPS 4 requires about 1.5GB for the $2^5 = 32$ core baseline simulations. Its memory footprint quickly decreases to 500MB for $2^9 = 512$ cores and above. STEPS 3 consumes more than 5GB of memory per core for the whole series. **(C)** STEPS 4 achieves a better scalability compared to STEPS 3. **(D)** Component scalability analysis of STEPS 4. The SSA operator shows super-linear speedup throughout the series. The diffusion operator also exhibits linear speedup until $2^{13} = 8,192$ cores. However, the EField operator shows limited scalability with maximum 10x speedup with $2^{10} = 1,024$ cores and above. **(E)** Component proportion analysis of STEPS 4. The EField operator progressively dominates the computational time, from 10% in the baseline simulations to 76% in the $2^{14} = 16,384$ core simulations, due to its limited scalability compared to the other operator components.

STEPS 4 increases to 770MB. We are unable to simulate the 3M complete model in STEPS 3 with 12GB of memory per core.

To further explore the capability of STEPS 4 in supporting super-large scale models, we refine the Purkinje cell mesh using Gmsh, then simulate the complete model with the refined

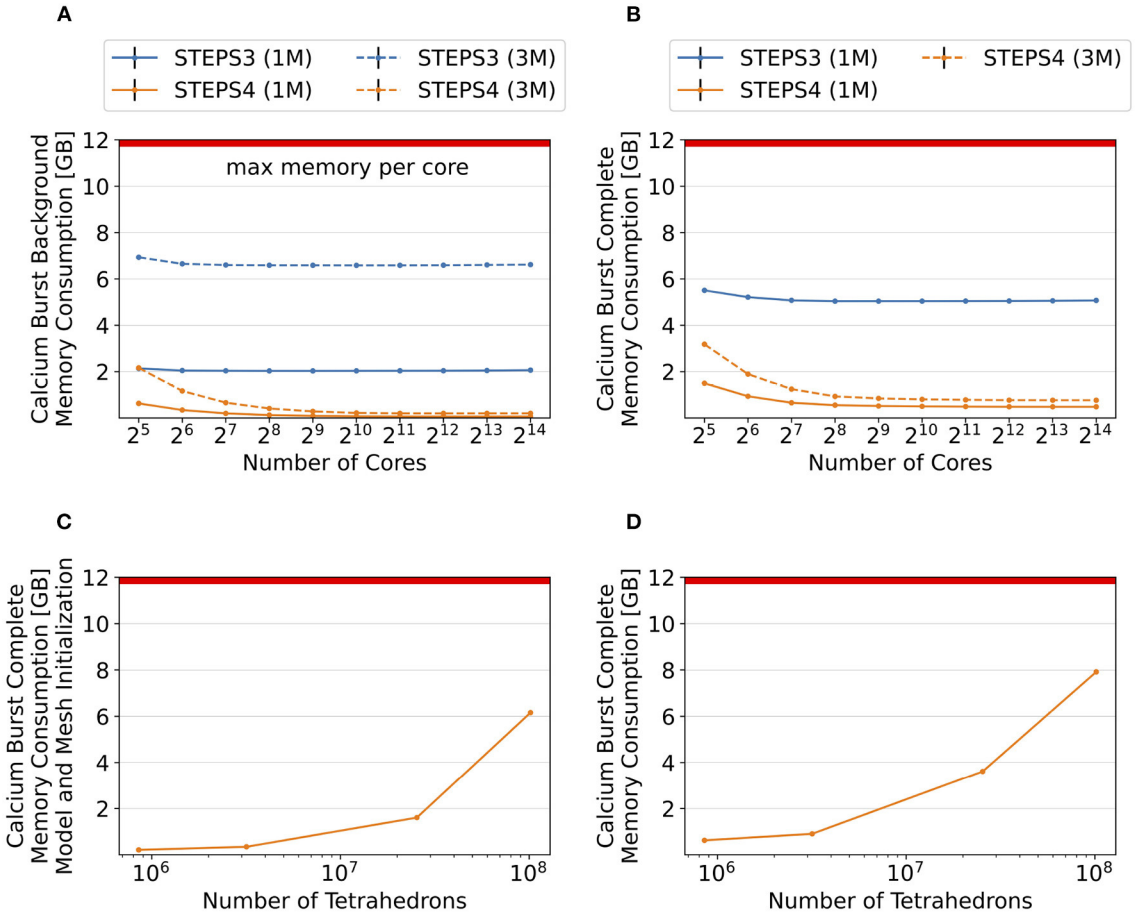

FIGURE 11
Memory footprint analysis and exploration of super-large scale models. In general, per-core memory consumption decreases as the core count increases, until a stabilized minimum consumption is reached. **(A)** Results of the background model simulations. The memory required per core hardly changes for STEPS 3 from 2.1GB to 2.0GB, while it declines rapidly for STEPS 4 from 626MB to 67MB. Similar results can be observed in the 3M series, where per-core memory consumption declines from 6.9GB to 6.6GB for STEPS 3, and from 2.1GB to 200MB for STEPS 4. **(B)** Results of the complete model simulations. In the 1M series results, memory consumption decreases from 5.5GB to 5.1GB for STEPS 3, and from 1.5GB to 500MB for STEPS 4. STEPS 4 in the 3M series consumes 3.2GB to 770MB of memory per core as core count increases. The 12GB memory capacity of the cluster per core is inadequate for the 3M complete model simulations with STEPS 3. **(C)** Memory consumption in GB at the initialization stage for the 1, 3, 25, and 100M meshes. **(D)** Total memory consumption in GB of STEPS 4 for the 1, 3, 25, and 100M mesh models. From our estimation, the 200M mesh requires a little over the 12GB memory capacity per core in the current setup using 16,384 MPI tasks.

meshes on $2^{14} = 16,384$ cores, and record the memory consumption at both the initialization and execution stages. The refined meshes have 25.4 million, 101.6 million and 203.3 million tetrahedrons, and are named "25M", "100M", and "200M" meshes respectively. Due to the large scale and consequently long execution time of these models, we do not run the full simulations but stop them after the first time point when memory consumption is stabilized. As shown in Figure 11C, memory consumption at the initialization stage increases from 213MB for the 1M mesh to 6.16GB for the 100M mesh. Slightly more memory is required for the simulation stage (Figure 11D), varying from 480MB for the 1M mesh, to 7.77GB for the 100M mesh. We are unable to initialize and execute simulations with the 200M mesh as the 12GB memory capacity is reached. From our estimation based on curve fitting of the results, a successful execution of the 200M mesh simulation would require approximately 13GB of memory on each core.

3.2.6. Single node roofline analysis of STEPS 4

In general, STEPS 4 demonstrates similar or better performance compared to STEPS 3 in high core count simulations, but has lower performance in small core count simulations. As discussed previously, one of the reasons is the different SSA operator implementations, but other factors may also be involved. As the performance with small core

count simulations is also important for STEPS 4 usage, a detailed performance analysis of current simulations is necessary to determine the direction of future optimizations. We choose the complete model as the profiling target since all major operators are included in the simulation. Note that in low core count configuration, the SSA and the diffusion operators are the dominating components in the simulation, thus they are the main focus of the analysis here. This is different from the optimization of high core count simulations, where the EField operator dominates the computation.

The analysis is based on the Roofline model (Williams et al., 2009), evaluating the scaling trajectory (Ibrahim et al., 2018) of the most computationally expensive routines, in our case, the SSA reaction operator, the Diffusion operator, and the EField operator. The Roofline model is one of the simplest tools to apply hardware/software co-design, enabling investigation on the interaction between hardware characteristics like memory bandwidth and peak performance, and the software characteristics such as memory locality and arithmetic intensity. Thus, it provides essential information on whether the investigated components are memory bandwidth or compute bound, and consequently vital suggestions on optimization strategies.

The Roofline model shown in Figure 12 for the Cascade Lake node on BB5 is constructed from a measured memory bandwidth ($\approx 197GB/s$) and a measured peak core performance ($\approx 78Gflop/s$, where *flop* stands for floating-point operations). Both metrics are measured with the likwid-bench utility (Treibig et al., 2010). In the Roofline graph, the x-axis is the arithmetic (or computational) intensity, computed as the ratio of floating point operations to transferred bytes from the main memory (DRAM traffic), and the y-axis is the observed performance. To obtain a scaling trajectory (Ibrahim et al., 2018) the measures are taken for varying core counts. Additionally, we run simulations with hyper-threading ($2^6 = 64$ processes) in order to utilize maximum resources.

For the measurements of the routine with LIKWID, a MPI synchronization barrier is added before and after each measured kernel. This is done to ensure that the measured metrics (e.g., hardware counters) indeed belong to the respective routines.

From Figure 12, it can be seen that all routines have a low arithmetic intensity. Each routine is represented by a different symbol and each data point is labeled by the number of processes. As described in Ibrahim et al. (2018), for ideal scaling a doubling of concurrency corresponds to a change in Δy (observed Flop/s) of $\approx 2\times$ without a corresponding change in Δx (arithmetic intensity), a behavior observed in our experiments. The SSA kernel is the one with the lowest arithmetic intensity and it is well into the arithmetic intensity regime where we expect the kernel to be memory bound.

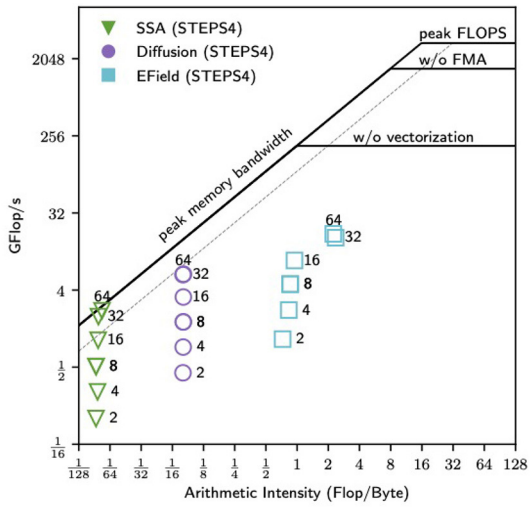

FIGURE 12
Roofline single-node scaling trajectories. The solid black lines are the full node hardware limits and the dashed gray line is the peak memory bandwidth for one socket. Each data point is labeled by the number of processes. All the computational kernels present a low arithmetic intensity mainly due to not ideal data locality (not optimal cache utilization). Nevertheless, the scaling is close to ideal (especially for the SSA and Diffusion operators) given that the doubling of concurrency leads to a corresponding $\Delta y > 0$. Hyper-threading at 64 cores does not give any substantial performance increase.

The Diffusion kernel presents similar behavior but with higher arithmetic intensity. Both kernels reach a saturation point as they approach the peak memory bandwidth. This observation suggests that there would be little to no gain to be had by vectorizing these kernels, instead possible improvements would have to come from algorithmic changes and/or cache blocking strategies in order to either increase the arithmetic intensity or fit the working memory set into the last level cache (LLC). For the EField kernel, we observe both $\Delta y > 0$ and $\Delta x > 0$ as we perform the strong scaling. This transition indicates that the number of floating-point operations has remained constant, so data movement must have decreased (Ibrahim et al., 2018). Finally, for all the computational kernels hyper-threading does not lead to any substantial performance increase.

To reach the maximum performance of a compute node, we need to efficiently utilize the cache memory hierarchy. In the Roofline graph, the higher the cache efficiency the higher the computational intensity. In our case, the low arithmetic intensity could be explained by the use of data structures that do not favor data locality (e.g., maps/dictionaries over vectors). Thus, a substantial improvement in the computational intensity of STEPS 4 can be achieved by favoring data locality and thus higher cache utilization, such as by a more extensive use of the flat-multimap.

4. Discussion

4.1. Achievements

With this continuous development and modernization of STEPS, we achieved several major goals:

- We modernized the existing code base of the entire framework adopting modern programming standards and practices such as C++17 and continuous integration. Particular care was posed on safety features such as vocabulary types. These improvements provided a solid modern foundation for STEPS 4 development.
- We developed a distributed solution that addressed the bottlenecks of STEPS 3.

STEPS 4 achieves similar performance and scalability as STEPS 3 while dramatically reducing the memory footprint. This is a key feature for future realistic modeling using STEPS. The Purkinje dendrite morphology simulated in the calcium burst model was reconstructed from light microscopic imaging. The spines were ignored and only the skeleton of the dendrite was preserved. It is possible to reconstruct a highly realistic Purkinje neuron containing all visible spines from high resolution electron microscopic imaging, however, the mesh generated from such morphology is expected to have 10 to 100 times more tetrahedrons than those used in current simulations. Such large models are completely out-of-reach for STEPS 3 since even the relatively small 3M calcium burst model already exceeds the 12GB per-core memory capacity on a state-of-the-art cluster like BB5. Conversely, STEPS 4 showed its potential on supporting simulations with such scale in the refined mesh simulations.

4.2. Limitations and solutions

STEPS 4 is not a complete replacement for STEPS 3. It is a highly specialized version of the operator-splitting solution specifically tailored for cluster-based, super-large scale simulations. Thus, we paid particular attention to performance optimizations whilst maintaining accuracy.

Even if STEPS 4 covers most features available in STEPS 3, some remain missing. For instance, the diffusion of species on surfaces (i.e., between patch triangles), and the associated surface diffusion boundaries, are not yet available. Patches between compartments are in principle supported but meshes have to be partitioned in such a way that tetrahedrons on both sides of patch triangles are owned by the same process. In STEPS 3, this constraint is enforced by *ad-hoc* partitioning adjustment; in STEPS 4 since the mesh is handled by Omega_h, this constraint is not enforced and it is up to the modelers to generate suitable partitioned meshes for their simulations, which is a limitation of this approach. We plan to support automatic partitioning adjustment with constraints in STEPS 4 in the future, however this requires further collaboration with the Gmsh and Omega_h developers as these libraries need further development to support such functionality. Finally, some auxiliary features in STEPS 3 such as the Region of Interest (ROI) functionality and visualization are not yet supported as implementations of new STEPS modeling toolkits are required to adapt the new distributed mesh formats and protocols.

4.3. Potential enhancements for STEPS 4

The distributed mesh backend of STEPS 4, Omega_h, not only supports traditional MPI based distributed-memory parallelism, but also shared-memory parallelism through OpenMP, and GPU parallelization *via* the CUDA framework. It also provides unique features such as mesh adaptation suitable for GPUs using flat array data structures and bulk transformations. These advanced features are currently not utilized in STEPS 4 as it relies solely on CPU based MPI parallelism. With the importance of GPU based fat compute nodes in modern HPC clusters, such features will play important roles when STEPS 4 is transitioned to other parallelism schemes.

In addition, the scalability analysis in Section 3.2 suggests two major axes for future development. Firstly, the EField operator is shown to be the major bottleneck in high core count simulations due to its poor scalability. Detailed profiling is required in the future to investigate the fundamental cause of this bottleneck, and to address it. Secondly, the Roofline analysis shows low computational intensity for all major kernels (SSA, Diffusion, EField). This behavior points to unsatisfactory use of cache memory, mainly caused by containers/data structures with poor data locality. A more extensive use of the `flat-multimap` could greatly improve cache utilization and increase the arithmetic intensity of these computational kernels.

4.4. Choosing between STEPS 3 and STEPS 4 in research projects

It is difficult to provide a solid guideline for choosing between STEPS 3 and STEPS 4 in a research project as different factors need to be considered. At the current stage, because not all the features in STEPS 3 are supported by STEPS 4, we recommend the researcher to firstly check if the features required in the model are supported by STEPS 4. If the model is supported by both implementations, then the researcher needs to consider what platform the model will be simulated on. Due to the efficiency difference of the current SSA operator, simulations on multi-core desktop workstation may be in favor of STEPS 3, while simulations on large scale clusters with limited memory resource may prefer STEPS 4 thanks to its memory footprint optimization. It is also worth mentioning that converting a

STEPS 3 model to STEPS 4 is a relatively trivial task, often only involving several lines of code changes in the modeling script. Therefore, the researcher can conduct a pilot benchmark with both solutions, then choose the suitable one for later simulation tasks based on the benchmark results.

4.5. Other current developments and future directions

4.5.1. Vesicle modeling

Currently STEPS, as all SSA methods in general, models molecules as points that do not occupy a significant volume of the space in which they reside. This is an obvious limitation if one wants to model certain types of structures in the cell such as vesicles. Vesicles are relatively large structures (∼40 nm diameter in the case of synaptic vesicles for example) that play many important roles in biology, and their complex structure and diverse functionality mean they cannot be realistically simulated by the point-molecule approach. Vesicles undergo processes such as endocytosis and exocytosis, interact with cytosolic and surface-bound molecules, and can be spatially organized into clusters such as in the presynaptic readily-retrievable pool. In an upcoming release, STEPS aims to support all of these features in an initial parallel implementation.

While the vesicle modeling development has been a separate project from STEPS 4, one tantalizing prospect is to marry many of the novel features of STEPS 4 with the vesicle modeling to allow bigger, more detailed simulations that can be run for longer biological times. This will, however, require substantial development on STEPS 4.

4.5.2. Coupling of STEPS with other simulator software

As part of the BBP mission to create a large scale reconstruction of brain tissue, a multi-scale approach for simulation is deemed necessary to capture elements at various temporal and spatial scales: one time scale for rapidly changing neuron voltages, a different, slower time scale for changing ion concentrations. Likewise, neuron morphologies can be distributed among computing ranks irrespective of geometric boundaries whereas bulk ion concentrations and metabolism use a coarse grain division of the spatial scale. For this purpose different simulators are used to leverage their specialized capabilities. NEURON (Carnevale and Hines, 2009) is used to solve relevant equations for membrane voltage and communication between neurons in addition to calcium in astrocyte morphologies. Meanwhile STEPS is employed to compute concentrations of diffusing ions in the extracellular space. A more memory efficient STEPS enables better sharing of computing resources between the two simulators.

5. Conclusion

The STEPS 4.0 project development reported in this article addresses several issues in previous STEPS releases, improving the user modeling experience, as well as modernizing the existing code base in order to aid future development. The main contribution of this research is a new parallel stochastic reaction-diffusion solver supported by a sophisticated distributed mesh library. While maintaining similar performance and scalability, the new solver dramatically reduces the memory footprint of simulations, resolving the major bottleneck in previous solutions. This breakthrough empowers future neuroscience research by enabling super-large scale molecular reaction-diffusion simulations with biologically realistic models.

Data availability statement

The STEPS simulator is available at http://steps.sourceforge.net/. Models for validation and performance investigation, as well as simulation data presented in this publication are available at https://github.com/CNS-OIST/STEPS4ModelRelease/tree/Frontiers2022.

Author contributions

ED and FS conceptualized and led this study. TC and WC led the overall software development of STEPS 4. SM and TC added in Omega_h the support to Gmsh file format 4 and features required in STEPS 4. BD, SM, TC, and WC contributed to the Zee library development and evaluations. AC, BD, CK, GC, JL, SM, TC, and WC contributed to the software development of STEPS 4. AC, CK, IH, JL, TC, and WC contributed to the pre-release testing, debugging and optimization of STEPS 4. JL contributed to the python interface development for STEPS 4 and model conversions from STEPS 3 to STEPS 4. AC and IH designed and conducted the validation benchmarks. CK, GC, and WC designed and conducted the performance benchmarks. NC checked statistical soundness of the tests and contributed in CI. OA, JK, and PK contributed to technical discussions and supervised the BBP team. WC coordinated the writing of the paper. All authors gave feedback and contributed to the article and approved the submitted version.

Funding

Research reported in this publication was supported by the Okinawa Institute of Science and Technology Graduate

University (OIST) and funding to the Blue Brain Project, a research center of the École polytechnique fédérale de Lausanne (EPFL), from the Swiss government's ETH Board of the Swiss Federal Institutes of Technology and the European Union's Horizon 2020 Framework Programme for Research and Innovation under the Specific Grant Agreement No. 785907 (Human Brain Project SGA2).

Conflict of interest

The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

Publisher's note

Supplementary material

The Supplementary Material for this article can be found online at: https://www.frontiersin.org/articles/10.3389/fninf.2022.883742/full#supplementary-material

References

Abhyankar, S., Brown, J., Constantinescu, E. M., Ghosh, D., Smith, B. F., and Zhang, H. (2018). Petsc/ts: a modern scalable ode/dae solver library. *arXiv preprint arXiv*:1806.01437. doi: 10.48550/arXiv.1806.01437

Amunts, K., Ebell, C., Muller, J., Telefont, M., Knoll, A., and Lippert, T. (2016). The human brain project: creating a european research infrastructure to decode the human brain. *Neuron* 92, 574–581. doi: 10.1016/j.neuron.2016.10.046

Amunts, K., Knoll, A. C., Lippert, T., Pennartz, C. M., Ryvlin, P., Destexhe, A., et al. (2019). The human brain project–synergy between neuroscience, computing, informatics, and brain-inspired technologies. *PLoS Biol.* 17, e3000344. doi: 10.1371/journal.pbio.3000344

Andrews, S. S., and Bray, D. (2004). Stochastic simulation of chemical reactions with spatial resolution and single molecule detail. *Phys. Biol.* 1, 137–151. doi: 10.1088/1478-3967/1/3/001

Antunes, G., and De Schutter, E. (2012). A stochastic signaling network mediates the probabilistic induction of cerebellar long-term depression. *J. Neurosci.* 32, 9288–9300. doi: 10.1523/JNEUROSCI.5976-11.2012

Anwar, H., Hepburn, I., Nedelescu, H., Chen, W., and De Schutter, E. (2013). Stochastic calcium mechanisms cause dendritic calcium spike variability. *J. Neurosci.* 33, 15848–15867. doi: 10.1523/JNEUROSCI.1722-13.2013

Anwar, H., Roome, C. J., Nedelescu, H., Chen, W., Kuhn, B., and De Schutter, E. (2014). Dendritic diameters affect the spatial variability of intracellular calcium dynamics in computer models. *Front. Cell Neurosci.* 8, 168. doi: 10.3389/fncel.2014.00168

Arjunan, S., and Tomita, M. (2010). A new multicompartmental reaction-diffusion modeling method links transient membrane attachment of *E. coli* MinE to E-ring formation. *Syst. Synth. Biol.* 4, 35–53. doi: 10.1007/s11693-009-9047-2

Arjunan, S. N., Miyauchi, A., Iwamoto, K., and Takahashi, K. (2020). pspatiocyte: a high-performance simulator for intracellular reaction-diffusion systems. *BMC Bioinform.* 21, 33. doi: 10.1186/s12859-019-3338-8

Ascoli, G. A., Donohue, D. E., and Halavi, M. (2007). NeuroMorpho.Org: a central resource for neuronal morphologies. *J. Neurosci.* 27, 9247–9251. doi: 10.1523/JNEUROSCI.2055-07.2007

Bhalla, U., Bilitch, D., and Bower, J. (1992). Rallpacks: a set of benchmarks for neuronal simulators. *Trends Neurosci.* 15, 453–458. doi: 10.1016/0166-2236(92)90009-W

Boehme, D., Gamblin, T., Beckingsale, D., Bremer, P.-T., Gimenez, A., LeGendre, M., et al. (2016). "Caliper: performance introspection for HPC software stacks," in *SC'16: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis* (Salt Lake City, UT: IEEE), 550–560.

Carnevale, N. T., and Hines, M. L. (2009). *The NEURON Book, 1st Edn*. Cambridge: Cambridge University Press.

Chen, W., and De Schutter, E. (2017). Parallel STEPS: Large scale stochastic spatial reaction-diffusion simulation with high performance computers. *Front. Neuroinform.* 11, 13. doi: 10.3389/fninf.2017.00013

Chen, W., Hepburn, I., Martyushev, A., and De Schutter, E. (2022). "Modeling neurons in 3d at the nanoscale," in *Computational Modelling of the Brain* (Cham: Springer), 3–24.

Cramér, H. (1928). On the composition of elementary errors: II, Statistical applications. *Scand. Actuar. J.* 11, 141–180. doi: 10.1080/03461238.1928.10416872

Denizot, A., Arizono, M., Nägerl, U. V., Soula, H., and Berry, H. (2019). Simulation of calcium signaling in fine astrocytic processes: effect of spatial properties on spontaneous activity. *PLoS Comput. Biol.* 15, e1006795-e1006795. doi: 10.1371/journal.pcbi.1006795

Gamblin, T., LeGendre, M., Collette, M. R., Lee, G. L., Moody, A., de Supinski, B. R., et al. (2015). "The spack package manager: bringing order to HPC software chaos," in *Supercomputing 2015 (SC'15)*. (Austin, TX: LLNL-CONF-669890).

Gibson, M. A., and Bruck, J. (2000). Efficient exact stochastic simulation of chemical systems with many species and many channels. *J. Phys. Chem. A* 9, 104. doi: 10.1021/jp993732q

Gillespie, D. T. (1977). Exact stochastic simulation of coupled chemical reactions. *J. Phys. Chem.* 81, 2340–2361. doi: 10.1021/j100540a008

Hattne, J., Fange, D., and Elf, J. (2005). Stochastic reaction-diffusion simulation with MesoRD. *Bioinformatics* 21, 2923–2924. doi: 10.1093/bioinformatics/bti431

Hepburn, I., Cannon, R., and De Schutter, E. (2013). Efficient calculation of the quasi-static electrical potential on a tetrahedral mesh and its implementation in steps. *Front. Comput. Neurosci.* 7, 129. doi: 10.3389/fncom.2013.00129

Hepburn, I., Chen, W., and De Schutter, E. (2016). Accurate reaction-diffusion operator splitting on tetrahedral meshes for parallel stochastic molecular simulations. *J. Chem. Phys.* 145, 054118. doi: 10.1063/1.4960034

Hepburn, I., Chen, W., Wils, S., and De Schutter, E. (2012). STEPS: efficient simulation of stochastic reaction-diffusion models in realistic morphologies. *BMC Syst. Biol.* 6, 36. doi: 10.1186/1752-0509-6-36

Hodgkin, A. L., and Huxley, A. F. (1952). A quantitative description of membrane current and its application to conduction and excitation in nerve. *J. Physiol.* 117, 500–544. doi: 10.1113/jphysiol.1952.sp004764

Hoffmann, M., Fröhner, C., and No,é, F. (2019). Readdy 2: fast and flexible software framework for interacting-particle reaction dynamics. *PLoS Comput. Biol.* 15, e1006830. doi: 10.1371/journal.pcbi.1006830

Ibanez, D., and Roberts, N. (2018). *Omega_h*. [*Software*] Available online at: https://github.com/sandialabs/omega_h.

Ibrahim, K., Williams, S., and Oliker, L. (2018). "Roofline scaling trajectories: a method for parallel application and architectural performance analysis," in *2018 International Conference on High Performance Computing and Simulation (HPCS)* (Orleans: IEEE), 350–358.

Insel, T. R., Landis, S. C., and Collins, F. S. (2013). The nih brain initiative. *Science* 340, 687–688. doi: 10.1126/science.1239276

Kerr, R. A., Bartol, T. M., Kaminsky, B., Dittrich, M., Chang, J. C., Baden, S. B., et al. (2008). Fast monte carlo simulation methods for biological reaction-diffusion systems in solution and on surfaces. *SIAM J. Sci. Comput.* 30, 3126. doi: 10.1137/070692017

Markram, H., Meier, K., Lippert, T., Grillner, S., Frackowiak, R., Dehaene, S., et al. (2011). Introducing the human brain project. *Procedia Comput. Sci.* 7, 39–42. doi: 10.1016/j.procs.2011.12.015

Massey Jr, F. J. (1951). The kolmogorov-smirnov test for goodness of fit. *J. Am. Stat. Assoc.* 46, 68–78. doi: 10.1080/01621459.1951.10500769

Mohapatra, N., Tønnesen, J., Vlachos, A., Kuner, T., Deller, T., Nägerl, U. V., et al. (2016). Spines slow down dendritic chloride diffusion and affect short-term ionic plasticity of gabaergic inhibition. *Scientific Rep.* 6, 23196–23196. doi: 10.1038/srep23196

Murdoch, D. J., Tsai, Y.-L., and Adcock, J. (2008). *P*-values are random variables. *Am. Stat.* 62, 242–245. doi: 10.1198/000313008X332421

Noether, G. E. (1963). Note on the kolmogorov statistic in the discrete case. *Metrika* 7, 115–116. doi: 10.1007/BF02613966

Oliveira, R., Terrin, A., di benedetto, G., Cannon, R., Koh, W., Kim, M., et al. (2010). The role of type 4 phosphodiesterases in generating microdomains of camp: large scale stochastic simulations. *PLoS ONE* 5, e11725. doi: 10.1371/journal.pone.0011725

Patoary, M. N. I., Tropper, C., McDougal, R. A., Lin, Z., and Lytton, W. W. (2019). Parallel stochastic discrete event simulation of calcium dynamics in neuron. *IEEE/ACM Trans. Comput. Biol. Bioinform.* 16, 1007–1019. doi: 10.1109/TCBB.2017.2756930

Rodola, G. (2020). *psutil*. Available online at: https://github.com/giampaolo/psutil.v5.8.0.

Schelker, M., Mair, C. M., Jolmes, F., Welke, R.-W., Klipp, E., Herrmann, A., et al. (2016). Viral rna degradation and diffusion act as a bottleneck for the influenza a virus infection efficiency. *PLoS Comput. Biol.* 12, e1005075. doi: 10.1371/journal.pcbi.1005075

Schöneberg, J., and Noé, F. (2013). Readdy-a software for particle-based reaction-diffusion dynamics in crowded cellular environments. *PLoS ONE* 8, e74261. doi: 10.1371/journal.pone.0074261

Slepoy, A., Thompson, A. P., and Plimpton, S. J. (2008). A constant-time kinetic Monte Carlo algorithm for simulation of large biochemical reaction networks. *J. Chem. Phys.* 128, 205101. doi: 10.1063/1.2919546

Sodani, A., Gramunt, R., Corbal, J., Kim, H.-S., Vinod, K., Chinthamani, S., et al. (2016). Knights landing: second-generation intel xeon phi product. *IEEE Micro* 36, 34–46. doi: 10.1109/MM.2016.25

Stillman, N. R., Balaz, I., Tsompanas, M.-A., Kovacevic, M., Azimi, S., Lafond, S., et al. (2021). Evolutionary computational platform for the automatic discovery of nanocarriers for cancer treatment. *NPJ Comput. Mater.* 7, 1–12. doi: 10.1038/s41524-021-00614-5

Treibig, J., Hager, G., and Wellein, G. (2010). "LIKWID: a lightweight performance-oriented tool suite for x86 multicore environments," in *Proceedings of PSTI2010, the First International Workshop on Parallel Software Tools and Tool Infrastructures* (San Diego, CA).

Trott, C. R., Lebrun-Grandi,é, D., Arndt, D., Ciesko, J., Dang, V., Ellingwood, N., et al. (2022). Kokkos 3: programming model extensions for the exascale era. *IEEE Trans. Parallel Distribut. Syst.* 33, 805–817. doi: 10.1109/TPDS.2021.3097283

Von Mises, R. (1928). *Wahrscheinlichkeit Statistik und Wahrheit*. (Berlin:Springer).

Williams, S., Waterman, A., and Patterson, D. (2009). Roofline: an insightful visual performance model for floating-point programs and multicore architectures. *ACM Commun.* 52, 65–76 doi: 10.1145/1498765.1498785

Zamora Chimal, C. G., and De Schutter, E. (2018). Ca2+ requirements for long-term depression are frequency sensitive in purkinje cells. *Front. Mol. Neurosci.* 11, 438. doi: 10.3389/fnmol.2018.00438

Zivanovic, D., Pavlovic, M., Radulovic, M., Shin, H., Son, J., Mckee, S. A., et al. (2017). Main memory in hpc: do we need more or could we live with less? *ACM Trans. Archit. Code Optim.* 14, 1–26. doi: 10.1145/3023362