BioFVM-X: An MPI+OpenMP 3-D Simulator for Biological Systems

Gaurav Saxena[1(✉)], Miguel Ponce-de-Leon[1(✉)], Arnau Montagud[1(✉)], David Vicente Dorca[1(✉)], and Alfonso Valencia[1,2(✉)]

[1] Barcelona Supercomputing Center (BSC), Barcelona, Spain
{gaurav.saxena,miguel.ponce,arnau.montagud,david.vicente,alfonso.valencia}@bsc.es
[2] Institució Catalana de Recerca i Estudis Avançats (ICREA), Passeig de Lluís Companys 23, 08010 Barcelona, Spain

Abstract. Multi-scale simulations require parallelization to address large-scale problems, such as real-sized tumor simulations. BioFVM is a software package that solves diffusive transport Partial Differential Equations for 3-D biological simulations successfully applied to tissue and cancer biology problems. Currently, BioFVM is only shared-memory parallelized using OpenMP, greatly limiting the execution of large-scale jobs in HPC clusters. We present BioFVM-X: an enhanced version of BioFVM capable of running on multiple nodes. BioFVM-X uses MPI+OpenMP to parallelize the generic core kernels of BioFVM and shows promising scalability in large 3-D problems with several hundreds diffusible substrates and ≈0.5 billion voxels. The BioFVM-X source code, examples and documentation, are available under the BSD 3-Clause license at https://gitlab.bsc.es/gsaxena/biofvm_x.

Keywords: Multi-scale modeling · Lattice-free modeling · OpenMP · MPI · Shared-memory · Distributed-memory · Parallelization

1 Introduction

Advances in understanding complex biological systems such as tumors require multi-scale simulations that integrate intracellular processes, cellular dynamics, and their interaction with the environment. Computational biologists use a wide range of approaches to simulate how single cells affect multi-cellular systems' dynamics [17,24]. Nevertheless, large-scale multi-scale modeling still needs tools to accurately simulate the environment in an efficient manner.

BioFVM [8] is a Finite Volume Method (FVM) [20] based simulation software for solving Partial Differential Equations (PDEs) [29] that model complex processes like the uptake, release and diffusion of substrates for multi-cellular systems such as tissues, tumors or microbial communities. Apart from being a self-contained callable library that can be used to implement and simulate biological models, BioFVM forms the core component of PhysiCell [9] - a flexible, lattice-free, agent-based multi-cellular framework capable of simulating cell

E. Cinquemani and L. Paulevé (Eds.): CMSB 2021, LNBI 12881, pp. 266–279, 2021.
https://doi.org/10.1007/978-3-030-85633-5_18

mechanics, such as cell movement, cell-cell interaction and different cell phenotypes, as well as the micro-environment consisting of diffusing substrates, signaling factors, drugs, etc. BioFVM is capable of handling multiple substrates and can simulate chemical and biological processes using both cell and bulk sources. The following diffusive PDE on a computational domain Ω (and boundary $\partial\Omega$) is solved for a substrate density vector ρ:

$$\frac{\partial \rho}{\partial t} = \nabla \cdot (\mathbf{D} \circ \nabla \rho) - \lambda \circ \rho + \mathbf{f}, \tag{1}$$

with the boundary condition $(\mathbf{D} \circ \nabla \rho) \cdot n = 0$ on $\partial\Omega$ and the initial condition $\rho(\mathbf{x}, t_0) = g$ in Ω. In (1) above, $\mathbf{D}$ is the matrix of (constant) diffusion coefficients, λ is the decay rate, $\mathbf{f}$ is the net source term and $\circ$ is the term-wise product of vectors [8]. Without loss of generality, the substrate density ρ can represent any kind of molecule such as a nutrient, a by-product, a signal molecule or a drug. As a consequence, modeling complex environments requires simulating many densities, posing a challenging scaling problem. Simulating the environment requires the numerical solution of the linear system obtained by a Finite Volume Discretization of the PDE given by Eq. (1), which BioFVM solves using the Thomas algorithm [31] - a fast, direct solver for tridiagonal systems. BioFVM's biggest scalability limitation is that it cannot execute on multiple nodes of an HPC cluster to solve a single, coherent problem and thus the problem *must* fit into the memory of a single node.

We present BioFVM-X[1]: an enhanced distributed version that uses MPI (Message-Passing Interface [21]) to parallelize the core kernels of BioFVM - enabling one to solve very large problems which were not previously solvable using the shared-memory only version. This contribution represents the first and the most critical step on the road to a distributed implementation of PhysiCell.

2 Related Work

Different agent-based approaches have been proposed to model and simulate multi-cellular systems, including on-lattice cellular automata, the Cellular-Potts model [10] and overlapping spheres, among others [23]. BioFVM [8,9] was created with the goal of achieving simplicity of usage, flexibility in expressing cell models, and optimizing execution speed while minimising dependencies on external libraries but is only shared-memory parallelized using OpenMP [22].

For realistic, complex simulations, the need is to simulate billions of cells and dynamic, complex 3-D environments, only achievable by optimal, full scale utilization of parallel systems [12,14]. Biocellion [14] is a flexible, discrete agent-based simulation framework that uses MPI for inter-node communication, as well as other dependencies, such as PNNL Global Arrays [25], CHOMBO [3], the Intel TBB [11] and the iterative Multigrid solver [2,32]. Nevertheless, Biocellion has fixed routines to describe system behaviors, is dependent on external libraries

[1] Available at: https://gitlab.bsc.es/gsaxena/biofvm_x under BSD 3-Clause license.

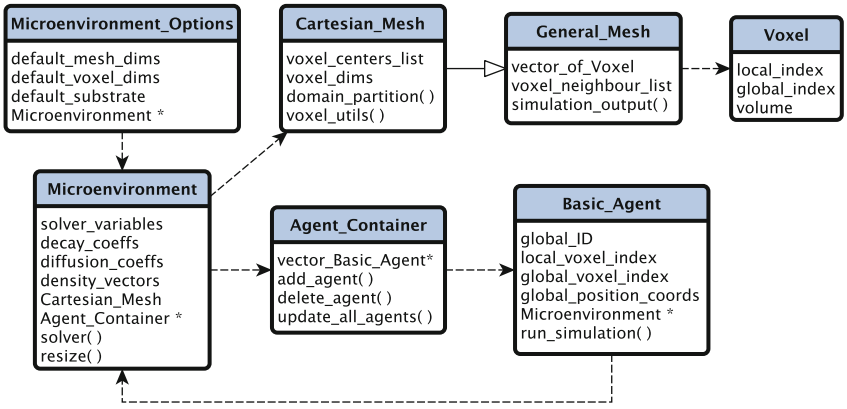


Fig. 1. Key classes in BioFVM, along with their member data and functions. Functions are distinguishable by a leading parenthesis i.e. `( )`. Names are arbitrary but convey semantic information. Solid, thick arrow with an un-shaded triangle represents inheritance and dashed arrows denote a pointer or class relationship - the class (or its pointer) being pointed to by the arrow is a data member of the class from which the arrow originates.

and is closed source, which might deter potential users. Chaste is an open-source, general purpose simulation package for modeling soft tissues and discrete cell populations [18] that can be used with MPI using PETSc [1] but which itself suffers from multiple dependencies. Timothy [4,5] is another open-source, MPI based tool but with several dependencies, such as Zoltan [6], Hypre [7] and SPRNG [19].

3 Internal Design and Domain Partitioning

The simplicity, flexibility, minimal dependence on external libraries, execution speed and openness of BioFVM make it an ideal experimental candidate for distributed parallelization. In BioFVM, the 3-D simulation domain is divided into Voxels (Volumetric pixels). The principal classes depicting the internal architecture and their relationship in BioFVM is shown in Fig. 1.

The top-level biological entities along with related classes (see Fig. 1) are: (1) Biological Environment (`Microenvironment` and `Microenvironment_Options`), (2) Physical Domain represented as 2-D/3-D Mesh (`General_Mesh`, `Cartesian_Mesh` and `Voxel`), and (3) Cells (`Basic_Agent` and `Agent_Container`). The data members of some classes are either the objects or the pointers of another class type (see dashed arrows in Fig. 1). The `Microenvironment` class sets the micro-environment name, the diffusion/decay rates of substrates, defines constants for the Thomas algorithm, contains an object of `Cartesian_Mesh`, a pointer to the `Agent_Container` class and performs I/O.

A group of resizing functions that determine the global/local voxels are members of the `Cartesian_Mesh` class. The `Microenvironment_Options` class helps to set oxygen as the first default substrate and the default dimensions of the domain/voxel. The `Cartesian_Mesh` class is publicly derived from `General_Mesh` (thick arrow in Fig. 1). The `Basic_Agent` class forms an abstraction of a cell. An object of the `Basic_Agent` class can either act as a source or sink of/for substrates. Each agent has a unique ID, a type, and maintains the local/global index of its current voxel.

We initialize MPI with the `MPI_THREAD_FUNNELED` thread support level and after domain partitioning [27,28], assign the sub-domains to individual MPI processes. Our implementation as of now supports only a 1-D x-decomposition (see Appendix A). The randomly generated positions of basic agents are mapped to respective processes (see Appendix B) after which they are created individually and in parallel on the MPI processes. Each MPI process initializes an object of the `Microenvironment` class, maintains the local and global number of voxels, local (`mesh_index`) and global voxel indices (`global_mesh_index`) and the center of each local voxel's global coordinates. A 1-D x-decomposition permits us to employ the optimal *serial* Thomas algorithm [30,31] in the undivided y and z dimensions. This enables all threads within a node to simultaneously act on elements belonging to different linear systems.

The Thomas algorithm is used to solve a tridiagonal system of linear equations in serial and consists of two steps, namely, Forward Elimination (FE) step followed by a Backward Substitution (BS) step. Unfortunately, both the steps involve serial and dependent operations and thus, the solver is inherently serial and cannot be fully (trivially) parallelized. Although we decompose data in the $x-$direction, the solver still runs *serially* i.e. MPI process rank i must finish the FE before this step can begin on MPI process rank $i+1$. Thus, the performance of this multi-node but serial Thomas solver is expected to be worse than a single-node Thomas solver due to the overhead of communication. The performance penalty is least in the $x-$direction as the data is contiguous in the memory as compared to the y and z direction where the data in the voxels' vector is non-contiguous. Thus, we decompose data only in the $x-$direction and avoid decomposition in the other directions. We expect to replace this non-optimized implementation by a modified, MPI+OpenMP version of the modified Thomas algorithm [15] in future versions.

4 Experiments

We used the MareNostrum 4 (MN4) supercomputer at the Barcelona Supercomputing Center (BSC) for all our experiments. Each node has two 24-core Intel Xeon Platinum 8160 processors and a total memory of 96 GB. BioFVM-X only requires a C++ compiler and an MPI implementation for compilation. We used `GCC 8.1` and `OpenMPI 3.1.1` running atop the SUSE Linux Enterprise Server 12 SP2 OS. The parallel file system is the IBM General Parallel File System and the compute nodes are interconnected with the Intel Omni-Path technology

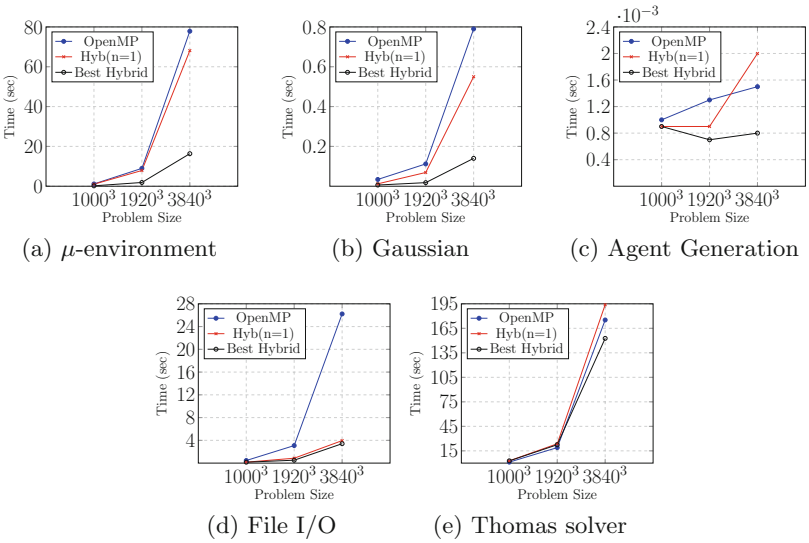
with a bandwidth of 100 Gbits/s. We pinned the threads to individual cores and bind each MPI process to a single processor (socket). We set the OpenMP environment variables `OMP_PROC_BIND=spread`, `OMP_PLACES=threads` [26] and used the `--map-by ppr:1:socket:pe=24` notation to allocate resources (see https://gitlab.bsc.es/gsaxena/biofvm_x).

We used a cubic physical domain and cubic voxels for all our tests. Our implementation assumed that the total number of voxels in the BioFVM's x-direction are completely divisible by the total number of MPI processes. The example that we used to demonstrate the benefits of Hybrid parallelism is *tutorial1* in the *BioFVM/examples* directory. This example: (1) Initializes and resizes the micro-environment (μ-environment, MC kernel) (2) Creates a Gaussian profile (GPG kernel) of the substrate concentration (3) Writes the initial and final concentrations to a `.mat` file (I/O kernel) (4) Creates Basic Agents (Sources and Sinks, BAG kernel) and (5) Simulates Sources/Sinks and Diffusion (Solver kernel).

Figure 2 presents timing results for the MC, GPG, BAG, I/O and Solver kernels on physical domains of sizes $1000^3, 1920^3$ and 3840^3. Cubic voxels had a volume of 10^3 with 5×10^2 sources and 5×10^2 sinks in this example. We denote the Hybrid implementation as "Hyb (n = a)", where "a" denotes the total number of nodes. For example, with Hyb (n = 2), we obtain a total of 2 (nodes) $\times$ 2 (MPI processes) $\times$ 24 (OpenMP threads) = 96 OpenMP threads, as we always run 2 MPI processes per node and 24 OpenMP threads per MPI process. Instead of 8 MPI processes for the domain of size 1000^3, we used 10 MPI processes due to a divisibility problem. Figure 3 shows the initial and final concentration of the diffusing substrate (oxygen) for a domain of size 1000^3. The simulation plots were obtained with Hyb (n = 1) by executing the `cross_section_surface.m` Matlab script bundled with BioFVM.

In summary for Hyb (n = 1), both MC and BAG kernels took advantage of the multiple MPI processes as initialization of the `Microenvironment` and `Basic_Agent` class objects were simultaneously carried out on separate processes in BioFVM-X as opposed to a single thread in BioFVM. The (MPI) I/O kernel showed significant performance gains over serial I/O for the tests considered (Fig. 2). Nevertheless, the Solver kernel execution run-times did not reflect a significant gain in the Hybrid version. An extended analysis of these results can be found in Appendix C. Note that it is generally very difficult for an MPI+OpenMP implementation to outperform the pure OpenMP implementation on a *single* node, as is the case of Fig. 2, due to the additional memory footprint of MPI and the cost of message-passing/synchronization. Our aim in the current work was to tackle very large problems that cannot fit into the memory a single node and to reduce their time to solution in a multi-node scenario.

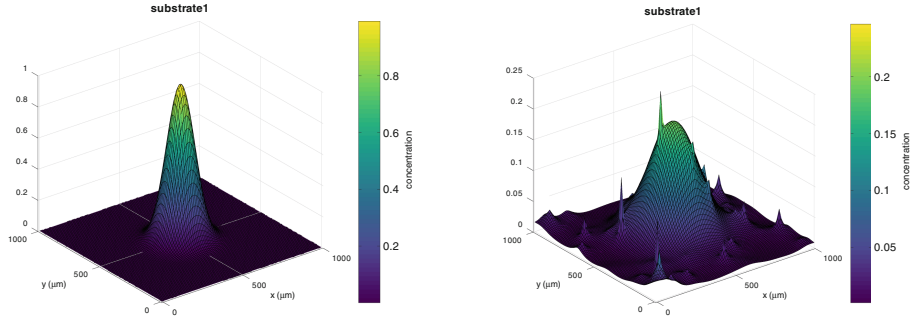
After testing with increased voxels and basic agents, we run a performance test to evaluate the scalability in the number of substrates. We found that the pure OpenMP BioFVM version is incapable of executing a simulation of 400 substrates on a domain of 1500^3 due to memory limitations. Nonetheless, we successfully run a Hybrid simulation using 400, and even 800 substrates, on a domain of 1500^3 by distributing the computation between 2 nodes.



(a) μ-environment (b) Gaussian (c) Agent Generation

(d) File I/O (e) Thomas solver

Fig. 2. Pure OpenMP Vs Hybrid MPI execution times for increasing problem sizes. Hyb (n = 1) represents the time when a single node with 2 MPI processes and 24 threads is used. The Best Hybrid represents the least time for that kernel for *any* number of experimental nodes considered.



(a) Initial Concentration (b) Final Concentration

Fig. 3. 3-D concentration density of oxygen simulated using Hyb (n = 1) for a domain of size 1000^3 and 1000 Basic Agents.

To further showcase BioFVM-X capabilities, we run a parallelized version of the model of tumor growth in a heterogeneous micro-environment from BioFVM [8]. We verified that the BioFVM-X distributed-memory 3-D tumor example yielded the exact same results as the shared-memory one (see Appendix D and Fig. 8). This is further proof that BioFVM-X correctly distributes the original BioFVM models with a boost in performance due to the load distribution and the potential of scaling simulations to a cluster of nodes, thus enabling researchers

Table 1. Time (in seconds) of execution for the pure OpenMP and the Hybrid version for a problem of size $7680 \times 7680 \times 7680$ ($\approx$0.5 billion voxels). The pure OpenMP version terminates while throwing Out Of Memory error.

$7680 \times 7680 \times 7680$	OpenMP	Hyb (n = 4)	Hyb (n = 8)
Build μ-environment	-	141.98	67.81
Gaussian profile	-	0.916	0.448
Initial file write	-	2.56	4.1
Agent generation	-	0.1060	0.0023
Source/sink/diffusion	-	1109.69	1210.41
Final file write	-	4.83	3.32
Total time	-	1260	1286.1

to address bigger, more complex problems. In addition, with a problem of size 7680^3, the memory consumption of the pure OpenMP version reaches $\approx$97% of the total memory of the node (96 GB) and the simulation terminates with a bus error. For the same problem size, the Hybrid code on 4 (with 192 threads) and 8 nodes (with 384 threads) executes successfully (Table 1).

5 Conclusion and Future Work

Multi-scale modeling has already proven its usefulness in a diversity of large-scale biological projects [9,16,24], but these efforts have been hampered by a scarcity of parallelization examples [4,12,14]. We present BioFVM-X - an enhanced MPI+OpenMP Hybrid parallel version of BioFVM capable of running on multiple nodes of an HPC cluster. We demonstrate that BioFVM-X solves very large problems that are infeasible using BioFVM as the latter's execution is limited to a single node. This allows BioFVM-X to simulate bigger, more realistic *in-silico* experiments. Further, despite the fact that our solver is only partially parallelized, we see performance gains in multiple execution kernels. In the future, we aim to replace the solver in the x-direction with a parallel modified Thomas algorithm [15].

BioFVM-X is open source under the BSD 3-Clause license and freely available at https://gitlab.bsc.es/gsaxena/biofvm_x. Even though it can be used to easily implement and simulate biological models in a self-contained manner, BioFVM-X also forms the lower layer of our ongoing efforts to have a parallel large-scale and multi-scale modeling framework termed PhysiCell-X, based on PhysiCell [9] - a framework that is under active development and has multiple stable releases.

Acknowledgements. The research leading to these results has received funding from EU H2020 Programme under the PerMedCoE project, grant agreement number 951773 and the INFORE project, grant agreement number 825070. The authors would like to thank Paul Macklin and Randy Heiland from Indiana University for their constant support and advice regarding BioFVM.

Appendix A 1-D Pure x-Domain Decomposition

Figure 4 shows a 1-D x-direction domain partition of a 3-D domain (x-direction is the unit-stride dimension).

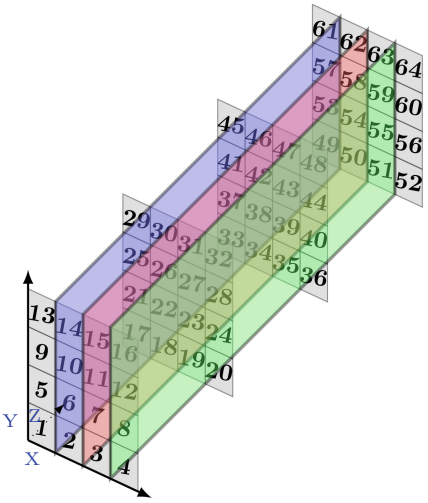


Fig. 4. A 3-D domain of dimensions $4 \times 4 \times 4$ visualized as four 2-D plates (shaded gray) of dimension 4×4 arranged one after the other. A 1-D domain partition (shown with blue, red and green planes) of 4 MPI processes in the x-direction divides the voxels numbered 1 to 64 into 4 parts. Rank 0, Rank 1, Rank 2, and Rank 3 processes contain voxel IDs numbered $4n + 1$, $4n + 2$, $4n + 3$, and $4n + 4$ respectively, where $n = 0, 1, 2, \ldots, 15$. Data is contiguous in the x-direction and the distance between 2 consecutive elements in the y and z directions is 4 and 16, respectively. (Color figure online)

Figure 5 shows the algorithm for domain partitioning where voxels are assigned to each MPI process. First, the domain dimensions (e.g. $xmin, xmax$) and the voxel dimensions (Δx) are used to decide the total number of global voxels (g_x_nodes). Given the total number of MPI processes (P), the voxels per MPI process (l_x_nodes) in the x-direction are computed next. This is followed by the computation of the global coordinates for the centers of voxels (for brevity, lines 1–6 in Fig. 5 show this for the $x-$direction *only*, with the treatment of remaining directions being analogous to the $x-$direction). Further, since each MPI process must maintain the local and corresponding global voxel index, the global mesh index of the first voxel ($l\_strt\_g\_index$) is computed on each process - used subsequently to assign the global mesh index to each voxel on that process (see the triply nested loop in Fig. 5). In addition to the assignment of a local/global voxel index on each process, a list of the immediate directional-neighbours of each voxel is also maintained (*not* shown in Fig. 5). In parallel,

Require: $xmin, xmax$, $d[]$ (Topology Dimensions), $c[]$ (Process Coordinates), Δx (Voxel x-length)

$g\_x\_nodes \leftarrow \frac{(xmax-xmin)}{\Delta x}$ ▷ y and z analogous
$l\_x\_nodes \leftarrow \frac{g\_x\_nodes}{d[1]}$
$l\_x\_start \leftarrow xmin + (c[1] \times l\_x\_nodes \times \Delta x)$
$i \leftarrow 0$
while $i{+}{+} \leq l\_x\_nodes - 1$ **do**
    $x\_c[i] \leftarrow l\_x\_start + (i + 0.5) * \Delta x$
$z_l \leftarrow c[2] \times g\_x\_nodes \times g\_y\_nodes \times l\_z\_nodes$
$y_l \leftarrow (d[0] - c[0] - 1) \times g\_x\_nodes \times l\_y\_nodes$
$x_l \leftarrow c[1] \times l\_x\_nodes$
$l\_strt\_g\_index \leftarrow x_l + y_l + z_l$
$n, i, j, k \leftarrow 0$
while $k{+}{+} < l\_z\_nodes$ **do**
    $z_k \leftarrow k \times g\_x\_nodes \times g\_y\_nodes$
    while $j{+}{+} < l\_y\_nodes$ **do**
        $y_j \leftarrow j \times g\_x\_nodes$
        while $i{+}{+} < l\_x\_nodes$ **do**
            $vxl[n].cntr[0] \leftarrow x\_c[i]$
            $vxl[n].cntr[1] \leftarrow y\_c[j]$
            $vxl[n].cntr[2] \leftarrow z\_c[k]$
            $vxl[n].g\_indx \leftarrow l\_strt\_g\_index + z_k + y_j + i$
            $n \leftarrow n + 1$

Fig. 5. Assignment of voxels to MPI processes in 1-D x-Domain Decomposition. Only partitioning of x-dimension is shown (same for y and z-directions). Prefixes $l\_$ and $g\_$ stand for "local" and "global", respectively. Array `d[]` contains the topology dimensions and array `c[]` contains MPI process coordinates [21]. The triply nested loop sets the global voxel (vxl) centers ($cntr$) and the global voxel index ($indx$).

such a scheme must accommodate for the cases when there is no local x, y or z neighbour but a global neighbour exists on the neighbouring process or when the process is aligned to the physical boundary of the domain. In BioFVM, a list for the Moore neighbourhood is also built for each voxel. The Moore neighbourhood equates to a 9-pt stencil in 2-D and a 27-pt stencil in 3-D [13].

Appendix B Mapping Basic Agents to a Voxel

A mapping that relates the position coordinates of the Basic Agent to the local index of a process-specific voxel is illustrated with the help of an algorithm in Fig. 6. Given the positions vector (denoted by $p[]$ in Fig. 6) of a Basic Agent, first the MPI Cartesian coordinates of the MPI process that contains the Basic Agent are computed (denoted by x_p, y_p and z_p). This is followed by the computation of the global x, y and z index (denoted by $first_x, first_y$ and $first_z$) of the first voxel of the MPI process that contains the Basic Agent. After calculating the directional i.e. x, y and z global indices of the voxel (denoted by vox_x, vox_y and

Require: $xmin, ymin, zmin$, $d[]$ (Topology Dimensions), $d[]$ (MPI Cartesian dimensions),$p[]$ (Agent position coordinates), $\Delta x, \Delta y, \Delta z$ (Voxel x/y/z-length)

$x_p \leftarrow d[0] - 1 - \lfloor (p[1] - ymin)/(l\_y\_nodes * \Delta y) \rfloor$
$y_p \leftarrow (p[0] - xmin)/(l\_x\_nodes * \Delta x)$
$z_p \leftarrow (p[2] - zmin)/(l\_z\_nodes * \Delta z)$
$first_x \leftarrow y_p * l\_x\_nodes$
$first_y \leftarrow (d[0] - 1 - x_p) * l\_y\_nodes$
$first_z \leftarrow z_p * l\_z\_nodes$
$vox_x \leftarrow \lfloor (p[0] - xmin)/\Delta x \rfloor$
$vox_y \leftarrow \lfloor (p[1] - ymin)/\Delta y \rfloor$
$vox_z \leftarrow \lfloor (p[2] - zmin)/\Delta z \rfloor$
$d_x \leftarrow vox_x - first_x$
$d_y \leftarrow vox_y - first_y$
$d_z \leftarrow vox_z - first_z$
$l\_index \leftarrow (d_z * l\_y\_nodes + d_y) * l\_x\_nodes + d_x$

Fig. 6. Mapping the position of a Basic Agent (in array $p[]$) to the process-local index (l_index) of a voxel that contains $p[]$. Prefix $l\_$ stands for "local". Array $d[]$ contains the MPI Cartesian topology dimensions. $l\_x/y/z\_nodes$ give the number of process local voxels and $\Delta x, \Delta y, \Delta z$ denote voxel dimensions (generally $\Delta x = \Delta y = \Delta z$).

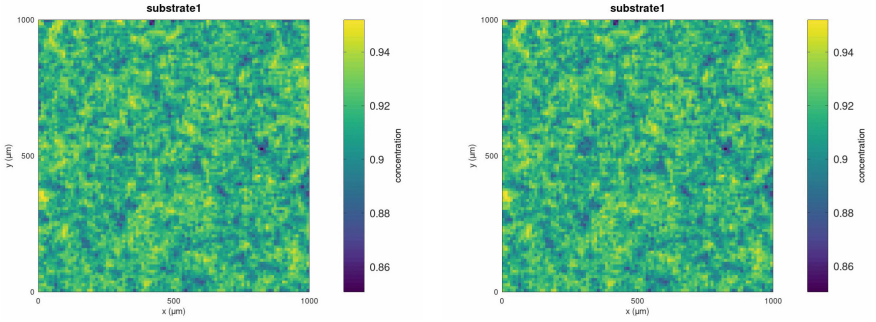
vox_z) that contains the Basic Agent, indices of the "first" voxel of the MPI process computed above is subtracted from the directional indices to obtain a local offset (denoted by d_x, d_y and d_z) of voxel indices in each direction. Subsequently, to obtain the local index of the process-specific voxel (l_index), the directional local offsets are appropriately multiplied by the number of process-local voxels.

Appendix C Extended Results

For an 8x increase in the number of voxels, the OpenMP MC, GPG, BAG and I/O kernels show a $7.86 - 8.67$x, $3.29 - 7.05$x, $1.15 - 1.3$x and $6.78 - 8.51$x increase, respectively (Fig. 2). The increase in the corresponding kernels for the best overall Hybrid version are: $8.7-9.4$x, $3-7.78$x, $0.77-1.14$x and $3.14-6.68$x, respectively (Fig. 2). Both MC and BAG kernels can take advantage of the multiple MPI processes as initialization of the `Microenvironment` and `Basic_Agent` class objects are simultaneously carried out on separate processes in BioFVM-X as opposed to a single thread in BioFVM. The (MPI) I/O kernel shows significant performance gains over serial I/O for the tests considered. For an 8x increase in the mesh resolution, the $6.78 - 8.11$x increase for Hybrid version in the Solver kernel looks promising as compared to the $9.24-15.93$x pure OpenMP increase, but the Hybrid version's absolute execution run-times do not reflect a significant gain. To help solve this, future versions of BioFVM-X will use the parallel modified Thomas solver [15] in the x-direction.

Appendix D Correctness Checking

To verify the correctness of the simulation, we run a simulation on a domain of size 1000^3 but increase the number of Basic Agents to 2×10^6 (Fig. 7 and Table 2).



(a) OpenMP (48 threads) (b) Hybrid(n=1)

Fig. 7. 2-D cross-section of the final concentration density of a given substrate with 2×10^6 agents on a domain of size 1000^3 using (a) Pure OpenMP (b) Hybrid MPI + OpenMP

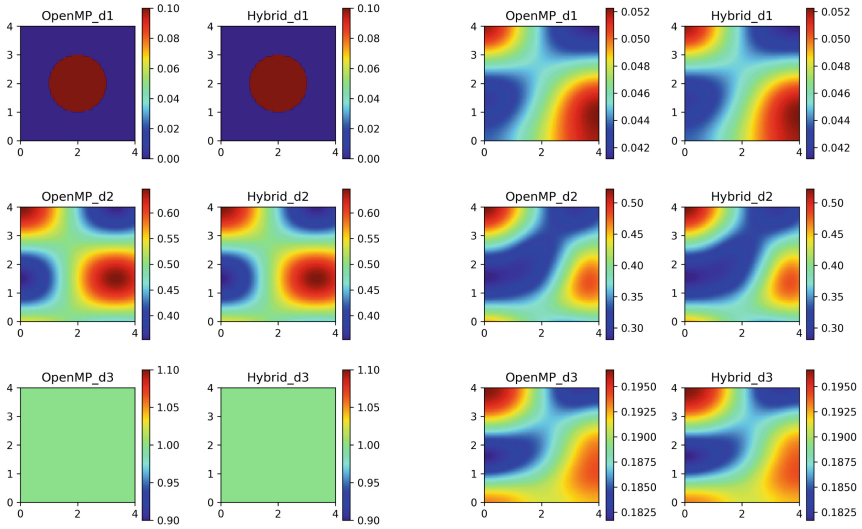
Table 2. Time (in seconds) of execution of simulation for the OpenMP version and the Hybrid version in a domain of size $1000 \times 1000 \times 1000$ with 2×10^6 Basic Agents.

$1000 \times 1000 \times 1000$	OpenMP	Hyb (n = 1)
Build μ-environment	1.14	1.03
Gaussian profile	0.0157	0.0117
Initial file write	0.219	0.084
Agent generation	2.46	1.45
Source/sink/diffusion	7.48	5.88
Final file write	0.22	0.063
Total time	11.56	8.54

To further underline the correctness of BioFVM-X, we compared the results of a tumor growth model in a heterogeneous environment from BioFVM [8] available at this link. In this model a 2-D tumor growth is driven by a substrate supplied by a continuum vascular system and cells die when it is insufficient. Additionally, the tumor cells have motility and can degrade the vascular system. We first expanded this example to a 3-D example (instead of the original 2-D) and specified the domain as $80 \times 80 \times 80$ voxels for a total of 512 000 voxels. We choose two different configurations:

– a shared-memory configuration (OpenMP) of 48 threads
– a hybrid shared and distributed-memory configuration (MPI+OpenMP) of 2 MPI processes running 24 threads each on a single node.

The comparison of the shared-memory and distributed-memory simulations yields identical results as shown in Fig. 8, further confirming that BioFVM-X provides the same results as BioFVM. The code to reproduce the figure is available on the BioFVM-X code repository.



(a) Initial densities at z=2.025 (b) Final densities at z=2.025

Fig. 8. 2-D cross-section of the (a) initial and (b) final concentration densities of three substrates from the 3-D tumor growth model on a domain of size $80 \times 80 \times 80$ voxels using shared-memory (`OpenMP_d*`) and distributed-memory with MPI+OpenMP (`Hybrid_d*`).

References

1. Balay, S., et al.: PETSc Users Manual. No. ANL-95/11-Revision 3.15 (2021). https://www.mcs.anl.gov/petsc
2. Briggs, W.L., McCormick, S.F., et al.: A Multigrid Tutorial, vol. 72. SIAM (2000)
3. Adams, M., et al.: Chombo Software Package for AMR Applications - Design Document. Lawrence Berkeley National Laboratory Technical Report LBNL-6616E
4. Cytowski, M., Szymanska, Z.: Large-scale parallel simulations of 3d cell colony dynamics. Comput. Sci. Eng. **16**(5), 86–95 (2014). https://doi.org/10.1109/MCSE.2014.2

5. Cytowski, M., Szymanska, Z.: Large-scale parallel simulations of 3d cell colony dynamics: the cellular environment. Comput. Sci. Eng. **17**(5), 44–48 (2015). https://doi.org/10.1109/MCSE.2015.66
6. Devine, K.D., Boman, E.G., Leung, V.J., Riesen, L.A., Catalyurek, U.V.: Dynamic load balancing and partitioning using the Zoltan toolkit (2007). https://www.osti.gov/biblio/1147186
7. Falgout, R.D., Yang, U.M.: *hypre*: a library of high performance preconditioners. In: Sloot, P.M.A., Hoekstra, A.G., Tan, C.J.K., Dongarra, J.J. (eds.) ICCS 2002. LNCS, vol. 2331, pp. 632–641. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-47789-6_66
8. Ghaffarizadeh, A., Friedman, S.H., Macklin, P.: Biofvm: an efficient, parallelized diffusive transport solver for 3-d biological simulations. Bioinformatics **32**(8), 1256–1258 (2015)
9. Ghaffarizadeh, A., Heiland, R., Friedman, S.H., Mumenthaler, S.M., Macklin, P.: PhysiCell: an open source physics-based cell simulator for 3-d multicellular systems. PLOS Computat. Biol. **14**(2), e1005991 (2018)
10. Graner, F., Glazier, J.A.: Simulation of biological cell sorting using a two-dimensional extended Potts model. Phys. Rev. Lett. **69**(13), 2013 (1992)
11. Intel: Intel®Thread Building Blocks | Intel®Software. https://software.intel.com/en-us/tbb
12. Jiao, Y., Torquato, S.: Emergent behaviors from a cellular automaton model for invasive tumor growth in heterogeneous microenvironments. PLOS Comput. Biol. **7**(12), e1002314 (2011)
13. Kamil, S., Chan, C., Oliker, L., Shalf, J., Williams, S.: An auto-tuning framework for parallel multicore stencil computations, pp. 1–12. IEEE (2010)
14. Kang, S., Kahan, S., McDermott, J., Flann, N., Shmulevich, I.: Biocellion: accelerating computer simulation of multicellular biological system models. Bioinformatics **30**(21), 3101–3108 (2014)
15. Kim, K.H., Kang, J.H., Pan, X., Choi, J.I.: PaScaL_TDMA: a library of parallel and scalable solvers for massive tridiagonal systems. Comput. Phys. Commun. **260**, 107722 (2021)
16. Letort, G., et al.: PhysiBoSS: a multi-scale agent-based modelling framework integrating physical dimension and cell signalling. Bioinformatics **35**, 1188–1196 (2019). https://doi.org/10.1093/bioinformatics/bty766
17. Macklin, P.: Key challenges facing data-driven multicellular systems biology. GigaScience **8**(10), giz127 (2019)
18. Maini, P., et al.: Chaste: cancer, heart and soft tissue environment. J. Open Source Softw. **5**(47), 1848 (2020)
19. Mascagni, M., Srinivasan, A.: Algorithm 806: SPRNG: a scalable library for pseudorandom number generation. ACM Trans. Math. Softw. (TOMS) **26**(3), 436–461 (2000)
20. Mazumder, S.: Numerical Methods for Partial Differential Equations: Finite Difference and Finite Volume Methods. Academic Press (2015)
21. Message Passing Interface Forum: MPI: A message-passing interface standard version 3.1 (June 2015). https://www.mpi-forum.org/docs/mpi-3.1/mpi31-report.pdf
22. OpenMP Architecture Review Board: OpenMP application program interface version 5.0 (November 2018). https://www.openmp.org/wp-content/uploads/OpenMP-API-Specification-5.0.pdf
23. Osborne, J.M., Fletcher, A.G., Pitt-Francis, J.M., Maini, P.K., Gavaghan, D.J.: Comparing individual-based approaches to modelling the self-organization of multicellular tissues. PLOS Comput. Biol. **13**(2), e1005387 (2017)

24. Ozik, J., Collier, N., Heiland, R., An, G., Macklin, P.: Learning-accelerated discovery of immune-tumour interactions. Mol. Syst. Des. Eng. **4**(4), 747–760 (2019)
25. Pacific Northwest National Laboratory: PNNL: Global Arrays Toolkit. https://hpc.pnl.gov/globalarrays/
26. Van der Pas, R., Stotzer, E., Terboven, C.: Using OpenMP–The Next Step: Affinity, Accelerators, Tasking, and SIMD. MIT Press (2017)
27. Saxena, G., Jimack, P.K., Walkley, M.A.: A cache-aware approach to domain decomposition for stencil-based codes, pp. 875–885. IEEE (2016)
28. Saxena, G., Jimack, P.K., Walkley, M.A.: A quasi-cache-aware model for optimal domain partitioning in parallel geometric multigrid. Concurrency Comput. Pract. Exp. **30**(9), e4328 (2018)
29. Strauss, W.A.: Partial Differential Equations: An Introduction. Wiley (2007)
30. Süli, E., Mayers, D.F.: An Introduction to Numerical Analysis. Cambridge University Press (2003)
31. Thomas, L.: Elliptic Problems in Linear Differential Equations Over a Network. Watson Scientific Computing Laboratory. Columbia University, NY (1949)
32. Trottenberg, U., Oosterlee, C.W., Schuller, A.: Multigrid. Elsevier (2000)