Contents lists available at ScienceDirect

Computers in Biology and Medicine

journal homepage: www.elsevier.com/locate/compbiomed

Simulation of 3D centimeter-scale continuum tumor growth at sub-millimeter resolution via distributed computing

Dylan A. Goodin [a], Hermann B. Frieboes [a,b,c,*]

[a] Department of Bioengineering, University of Louisville, KY, USA
[b] James Graham Brown Cancer Center, University of Louisville, KY, USA
[c] Center for Predictive Medicine, University of Louisville, KY, USA

ARTICLE INFO

Keywords:
3D tumor model
Cancer simulation
Distributed computing
Parallelized computing
Continuum models
Mixture models
CUDA
MPI
openMP

ABSTRACT

Simulation of cm-scale tumor growth has generally been constrained by the computational cost to numerically solve the associated equations, with models limited to representing mm-scale or smaller tumors. While the work has proven useful to the study of small tumors and micro-metastases, a biologically-relevant simulation of cm-scale masses as would be typically detected and treated in patients has remained an elusive goal. This study presents a distributed computing (parallelized) implementation of a mixture model of tumor growth to simulate 3D cm-scale vascularized tissue at sub-mm resolution. The numerical solving scheme utilizes a two-stage parallelization framework. The solution is written for GPU computation using the CUDA framework, which handles all Multigrid-related computations. Message Passing Interface (MPI) handles distribution of information across multiple processes, freeing the program from RAM and the processing limitations found on single systems. On each system, Nvidia's CUDA library allows for fast processing of model data using GPU-bound computing on fewer systems. The results show that a combined MPI-CUDA implementation enables the continuum modeling of cm-scale tumors at reasonable computational cost. Further work to calibrate model parameters to particular tumor conditions could enable simulation of patient-specific tumors for clinical application.

1. Introduction

Representation of tumor growth in clinically-relevant contexts has generally been explored via three main types of models: continuum models that simulate tissue-scale behavior, discrete models that define individual cells and their interactions, and hybrid models utilizing a combination of both approaches. These efforts have been traditionally constrained by the computational cost to numerically solve the associated equations, with the results limited to representing mm-sized or smaller tumors. For discrete models the challenge has been to simulate billions of cells and their interactions, while for continuum models the cost of representing cm-scale domains becomes computationally prohibitive. In particular, models based on continuum mixture theory to simulate tumor growth have been developed [1–5] and analyzed [6–8], building upon earlier work to represent tumor tissue as different phases of a mixture [9–29]. However, more complex continuum models have struggled to achieve high performance simulations at patient-scale (cm) resolution.

Lorenzo et al. used a continuum two-phase model to simulate a prostate tumor with 2.66 cm^3 volume from CT-scan [30]. Antonopoulos et al. represented a 4.2 cm^3 domain for 3 simulated months with 2.2 mm^3 resolution [31]. While both models reached cm scale, multi-species representation and vascularization were not incorporated. Wise et al. developed an adaptive multigrid framework for simulating a continuum multispecies tumor model using a single-core computer process, finding that time required to simulate a single day of tumor evolution at $1*10^{-2}$ days per time step increases from ~12 min during early time steps to ~400 min by end of simulation [32]. In Ref. [33] the model of [1] was coupled with a lattice-free random walk angiogenesis model [34–37]. Recently, a mixture model with continuum 3D representation of tumor, vasculature, and extracellular matrix (ECM) was presented in Refs. [38,39]. Open Multi-Processing (OpenMP) parallelization benefits were offset in Ref. [38] by increased model complexity: early model performance was 156 min per simulated day to ~280 min per simulated day for $1*10^{-2}$ days per time step. In these models, coupling of tumor and vasculature in a biologically realistic 3D representation to simulate clinically-relevant tumor growth incurs a high computational cost. Consequently, the numerical implementation to solve the coupled equations has hindered these models from reaching

* Corresponding author. Department of Bioengineering, Lutz Hall 419, University of Louisville, KY, 40292, USA
E-mail address: hbfrie01@louisville.edu (H.B. Frieboes).

https://doi.org/10.1016/j.compbiomed.2021.104507
Received 5 April 2021; Received in revised form 15 May 2021; Accepted 16 May 2021
Available online 21 May 2021

Abbreviations

3D	Three-dimensional
AdP	Administrative process
CPU	Central processing unit
CT	Computed tomography
CUDA	Compute unified device architecture
ECM	Extracellular matrix
GCP	General computation process
GPU	Graphics processing unit
MDE	Matrix degrading enzyme
MPI	Message passing interface
openMP	Open multi-processing
PC	Personal computer
RAM	Random access memory
TAF	Tumor angiogenic factors
TGF	Tumor growth factors

practical application, especially in terms of simulating patient tumor response to potential courses of treatment in a timely manner to drive clinical decision-making.

Outside of the context of continuum models, several parallelized implementations have been developed over the past decade to improve performance. In Ref. [40], a tumor model parallelization saw a 5.2x performance increase over a single-process approach using eight processors. OpenMP implementations have improved tumor modeling performance, as shown in Refs. [41,42]. An early effort at parallelizing a Cellular Potts model used Message Passing Interface (MPI) but remained a 2D simulation [43]. Models have benefitted from multiple approaches, including an MPI-based parallel solver named NAStJA [44–46] and Compute Unified Device Architecture (CUDA) based solvers [47,48]. A near 30x uplift over CPU-based implementations using a CUDA-based solver was seen in Ref. [47]. Likewise, cellular automata tumor modeling has benefited from CUDA and CPU-based parallelization approaches [49–51]. A tumor simulation using finite element methods leveraged an MPI framework to attain ~4x performance improvement by spanning the simulation across 16 processes [52]. Performance gains for a finite-element method were also realized using Galois, a software package that employs an amorphous data-parallelism model [53]. Recently, a hybrid model was parallelized using the framework to simulate ~1 cm^3 melanoma evolution [54]. Of note, Antonopoulos et al.'s continuum model in MATLAB emphasized macroscopic tumor phenomena, simulating a cubic 4.2 cm length domain at 2 mm^3 resolution. By simulating fewer equations at a lower resolution than in Ref. [38], the model was capable of simulating ~3 months of tumor evolution in 10–12min [31].

Complementing these previous efforts, this study presents a distributed computing implementation of the mixture model in Refs. [38,39] via a combined MPI-CUDA implementation to simulate cm-scale vascularized 3D tumor growth tissue at sub-millimeter resolution.

2. Materials and methods

2.1. Model of tumor growth

We fully parallelize the continuum 3D model presented in Refs. [38,39], which used openMP. Briefly, the model simulates evolution of a single tumor cell phenotype in an environment with host cells and ECM. Tumor tissue vies for resources against healthy tissue while balancing the need for nutrients, metabolites, and ionic species, including oxygen, carbon dioxide, lactate, bicarbonate, sodium ions, chloride, and H^+ ions. Crowding in a limited tissue space is abstracted into solid mass pressure and pressure from surrounding fluids. These pressures drive velocity in the solid tissue mass and create buildup of elastic energy on the surrounding ECM. Matrix degrading enzymes and myofibroblast concentrations increase due to remodeling of surrounding ECM to compensate for strain induced by tumor growth.

During tumor growth, tissue distal from vasculature can be deprived of resources. The tumor releases angiogenic factors to encourage growth of surrounding vasculature towards hypoxic tissue. Increased vessel leakiness has been well-documented from such relatively quick vasculature changes; the body compensates for edema by increasing lymphatic growth [55]. Therefore, the model simulates lymphatic growth with independent terms to the vasculature, although both are closely related mathematically and physiologically. However, vasculature effectiveness is limited physiologically by the diffusion rate of oxygen. Thus, interior hypoxic regions in sufficiently large tumors will operate at varying levels of anaerobic glycolysis, building up lactic acid. In a sufficiently hypoxic state, tumor cells become apoptotic or necrotic, represented as dead cell volume fraction.

Numerically, this model is solved using a geometric multigrid solver. At its finest multigrid level, the solver uses evenly-spaced points to define model solution resolution. By increasing the number of points per side of the cubic domain and using a point-to-point distance $<100\ \mu m$, sub-mm precision is retained while increasing the domain size beyond a centimeter on a side. At each point on a cubic domain, a solution for model variables is generated, with solution generation occurring after θ units of simulated time elapse. Key equations in non-dimensionalized form and the numerical solver are summarized further in **Supplement**.

2.2. Limitations of openMP-based solver

Three limitations of openMP-based solver in Refs. [38,39] include:

1. When tested using 128^3 grids, maximum performance was obtained using only 8 cores out of 32 on a 32-core processor on the University of Louisville Cardinal Research Cluster (CRC), potentially due to insufficient memory bandwidth. Further testing on an AMD 2990WX exhibited more promising results, indicating that nascent CPUs may fare better from openMP. However, limitations to core counts would further constrain gains.
2. openMP is a shared-memory architecture that runs on non-distributed systems, limiting performance gains to a single PC, workstation, or High Performance Computing (HPC) node.
3. Many PCs have insufficient RAM to hold larger tumor model spaces. Table 1 summarizes expected RAM footprint for varying model sizes.

2.3. Distributed computing solution

To simulate tumors at patient tissue cm-scales, model in Refs. [38,39] requires sufficient computational resources to function at a 512^3 sized domain and, according to Table 1, over 100 GB RAM are necessary. Because single-socket computers do not typically possess this much RAM, a new solution generator is required for long-term parallel computing.

For this purpose, this study implements the numerical solving

Table 1
Memory footprint for varying model sizes using model in Refs. [38,39].

	Level Size	
	256^3	512^3
Points on a Side	130	258
Maximum Level size Simulated (#Points on a side)	256	512
Upper Bound RAM Required per process with eight processes on the finest level with an equal distribution of level data (GB)	3.3	25.5
Total RAM Required for single process on the finest level (GB)	13.6	107.6
Maximum spherical tumor diameter that could be simulated with 50 μm point resolution (mm)	12.8	25.6

scheme of [39] using a two-stage parallelization framework. First, numerical computations were rewritten for GPU computation using CUDA. This framework handles all Multigrid-related computations, including Gauss-Seidel red-black smoothing, restriction, prolongation, and error correction. MPI handles distribution of information across multiple processes, freeing the program from the RAM and processing limitations found on single-system parallelization frameworks. On each system, Nvidia's CUDA library allows for faster processing of model data using GPU-bound computing on fewer systems. Thus, the new model framework is a two-part MPI-CUDA model.

The type of simulation being considered, generally known as HPC, requires consistent communication between multiple data processors Architectures configured for Big Data, in which data processors are designed to perform tasks at a coarser resolution, are unideal for datapoint-level communication [56]. Further, common Big Data platforms, such as Hadoop and Apache Spark, rely on either disk-based queries or exhibit possess significantly more overhead than comparable MPI-based HPC implementations, respectively, making MPI a more viable distributed computing framework for our purposes [56–59].

Overall algorithm in MPI-CUDA tumor model is identical to model in Refs. [38,39], save that the conditions for block generation have changed. Under the previous framework efficiency was defined as $\eta = \frac{\#Points\ in\ F_{\ell+1}^{t,r-1}}{\#Points\ in\ B_{\ell+1}}$ where the set of all flagged points in level ℓ at time step t and solver iteration $r-1$ is represented as $F_{\ell}^{t,r-1}$and the set of all points within blocks in level ℓ is represented by B_{ℓ}. To prolongate to a new level, η had to be lower than a pre-defined cutoff efficiency. In the new framework, the decision process is simplified to an all-or-nothing behavior where a single flagged point on ℓ will cause the solver to operate over the entirety of the domain on level $\ell+1$ (i.e., $\Omega_{\ell+1}$). This behavior can be interpreted as creating a block $B_{\ell+1}$ whose size is determined by prolongating block $B_{\ell} = \Omega_{\ell}$ using the prolongation operator function $P_{\ell+1}^{\ell}(X_{\ell})$ for some set of points X on ℓ. This decision can be summarized as $F_{\ell}^{t,r-1} \neq \emptyset \Rightarrow B_{\ell+1} = \mathrm{P}_{\ell+1}^{\ell}(B_{\ell}) = \Omega_{\ell+1}$. Memory management is thus greatly simplified, since the solver either finishes at level ℓ or processes level ℓ for a given time step. Consequently, this decision also increases workload on levels where only a subset of Ω_{ℓ} requires smoothing.

While this method simplifies memory management, it can sacrifice solution accuracy. Residual error is calculated as:

$$\left\| R_{\ell} - L_{\ell}\left(\psi_{\ell}^{t,r}\right) \right\|_{B_{\ell}} = \sqrt{\frac{1}{|B_{\ell}|} \cdot \sum_{p \in B_{\ell}} \sum_{v \in V} \left(R_{p,v} - L_{p,v}\right)^2} \tag{1}$$

where RHS and LHS solutions are $R_{p,v}$ and $L_{p,v}$, respectively, for all points p in block $B_{\ell+1}$ and variables v in the set of all tumor model variables V. R_{ℓ} and L_{ℓ} are the RHS and LHS model terms on ℓ, respectively and $\psi_{\ell}^{t,r}$ is the variable solution on ℓ at time step t and solver iteration r. When size of $B_{\ell+1}$ is not fit to the flagged points, sensitivity to local error is decreased. Thus, model error will be artificially high. This was corrected by redefining p to fit the set of all flagged point $F_{\ell}^{t,r-1}$:

$$\left\| R_{\ell} - L_{\ell}\left(\psi_{\ell}^{t,r}\right) \right\|_{F_{\ell}^{t,r-1}} = \sqrt{\frac{1}{\left|F_{\ell}^{t,r-1}\right|} \cdot \sum_{p \in F_{\ell}} \sum_{v \in V} \left(R_{p,v} - L_{p,v}\right)^2} \tag{2}$$

This method allows for easy memory transfers from CPU to GPU while retaining solution accuracy.

2.4. Model architecture

Flow of information during execution differs from the previous architecture. MPI implementation has two classes of processes:

1. Administrative process (AdP). Responsibilities include construction of model domain and decisions pertaining to solution convergence. There is only one process designated as AdP within MPI-CUDA runtime.
2. General Computation Process (GCP). GCPs take up a non-overlapping cubic region in Ω. Each can operate on more than one level as designated by AdP at start of model execution.

Algorithm 1 summarizes the process for any computation function X that is neither restriction nor prolongation. Before synchronization, each GCP must unload its corresponding GPU $\mathrm{P_m}$ containing unsynchronized data before executing X on GPU $\mathrm{P_n}$. Preceding execution of function X on level ℓ, all data across GCPs is synchronized to avoid race conditions. Of note, in Algorithm 1 the binding rules for GPUs $\mathrm{P_m}$ and $\mathrm{P_n}$ are left to the implementer. Ideally, processes are bound in a non-overlapping fashion to a single GPU. That is, two GCPs g and h are the same if and only if $\mathrm{m_g = n_g = m_h = n_h}$, but hardware limitations may require an overlapping allocation in which multiple MPI processes share GPU resources.

```
RunFunction(X, g, $\ell$, m, n) {

    Select GPU $\mathrm{P_m}$

    If GPU $\mathrm{P_m}$ contains unloaded data addressed to GCP g {
            Unload $\Omega_\ell$ data from $\mathrm{P_m}$
            Synchronize $\Omega_\ell$ with all GCPs on level $\ell$
    }

    Select GPU $\mathrm{P_n}$

    Load level $\ell$ data associated with GCP g onto GPU $\mathrm{P_n}$

    Run X on $\mathrm{P_n}$
}
```

Processes are applied to level ℓ sequentially filling a single region of the model in a manner depicted by Fig. 1, in which level ℓ, level $\ell+1$, and level $\ell+2$ operate over the same domain object, represented by the triangle. Level ℓ contains a single process. Adapting a method of hierarchical process filling proposed by Ref. [60], on level $\ell+1$ three additional processes are required to process level $\ell+1$. All four processes, including region 1 on level $\ell+1$, restrict to region 1. Same relationship exists between levels $\ell+2$ and $\ell+1$. One-eighth of domain covered by a single GCP unit in level ℓ is retained locally while other 7 parts of $\Omega_{\ell+1}$ are sent to seven other GCPs. Thus, amount of work increases linearly with number of levels, since processes on each level after and including level ℓ have same domain size [60]. This also means that each GCP on a previous level must operate on the final level ℓ_{max}. Scaling this approach for 3D, total amount of processes required is:

$$Processes\ Required = \begin{cases} 1, \ n_0 > m_0 \\ 8^{m_0 - n_0}, \ n_0 \leq m_0 \end{cases} \tag{3}$$

where n_0, $m_0 \in \mathbb{N}$, $n_0 < m_0$ the finest level ℓ_{max} has 2^{m_0} points on a side, and each process holds 2^{n_0} points per side per level with maximum RAM usage. Thus, for $n_0 = m_0 - 1 \Rightarrow \#Processes = 8^{m_0-(m_0-1)} = 8\ processes$. Because a portion of computational work remains on every finer level after a process is first introduced, processes are utilized to a greater degree over a non-hierarchical filling method.

At the beginning of model execution, a single AdP is designated. AdP starts by defining process boundaries determined by the maximum sized domain that each GCP can contain. To agree with domain Ω, cubic domain Ω_D for each GCP has side length 2^k, where $k \leq \ell_0 + \ell_{index}$. Value of k can be specified at runtime or be empirically derived by hardware availability. The resulting size is the fundamental size for each GCP. Consequently, coarsest level ℓ_0 may define a domain Ω_0 that is larger than a single GCP.

For process n operating over a subset of Ω_{ℓ}, denoted Ω_{ℓ}^{n}, a set of GPUs is paired with process n to process Ω_{ℓ}^{n}. For this study, we assume Ω_{ℓ}^{n} is

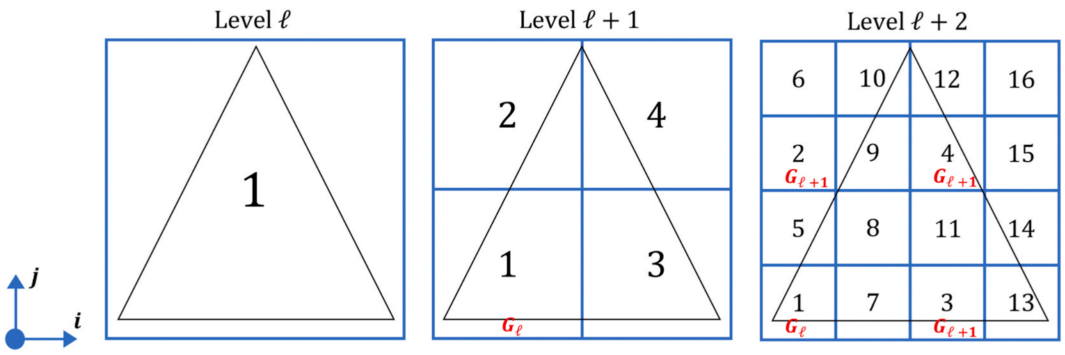

Fig. 1. Multilevel Nodal Geometry on sequential levels ℓ, $\ell+1$, and $\ell+2$. Processes on level ℓ, $\ell+1$, and $\ell+2$ operate over equally sized datasets regardless if operating on Ω_ℓ, $\Omega_{\ell+1}$, and $\Omega_{\ell+2}$, respectively. This is because the simulated distance between points is halved on level $\ell+1$ and halved again on level $\ell+$ 2, thus the density of information keeps pace with the addition of more GCPs. This approach is extended in this work to a three-dimensional simulation domain. During restriction, GCPs locally restrict their domain data and consolidate their information in the -*i* and -*j* direction to nodes marked with G_ℓ or $G_{\ell+1}$. Prolongation reverses this process by transferring G_ℓ data along the $+i$ and $+j$ direction.

cubic. If required by hardware constraints, Ω_ℓ^n is subdivided into subdomains ω_j^ℓ that are sufficiently reduced to fit in GPU RAM. Subdomains have following properties for *m* subdomains on level ℓ:

1. $\omega_j^\ell \subseteq \Omega_\ell^n,\ j \in \{1, \ldots, m\}$
2. $\omega_1^\ell \cap \omega_2^\ell \cap \ldots \cap \omega_m^\ell = \varnothing$
3. $\omega_1^\ell \cup \omega_2^\ell \cup \ldots \cup \omega_m^\ell = \Omega_\ell^n$
4. $\omega_j^\ell \neq \varnothing,\ j \in \{1, \ldots, m\}$

If a single GPU has enough RAM to hold Ω_ℓ^n, then $m = 1$. Because of stencil operations, a one-point shell layer around each subdomain is required. Next, GPUs receive relevant constant terms from the model, e.g., point spacing on level ℓ and domain dimensions. Finally, function X is called on all GPUs. After computation, data are unloaded as required to allow data to synchronize between all Ω_ℓ^n on Ω_ℓ. Due to memory transfers from GPU to CPU, this process constitutes the bulk of this method's overhead.

2.5. *Data synchronization*

When syncing data across GCPs, there are three vectors that must be defined: (1) a syncing vector, $\overrightarrow{S}$, (2) a process vector, $\overrightarrow{N}$, that points from self to an adjacent GCP, and (3) a data vector, $\overrightarrow{D}$, for directing synchronized information to the correct cubic feature (i.e., face, edge, or corner). Because processes are arranged as cubes in a Cartesian grid, there are 26 possible syncing directions for each GCP. Described by graph theory, each GCP forms a star graph S_{26} with its neighbors. Any MPI send-and-receive operation is a two-step process, in which any link (u, v_m) for $m \in \{1, 2, \ldots, 26\}$ from center node of star graph *u* to vertex v_m must be traversed in both directions. For maximum performance, perfect matching is desirable, meaning that on level ℓ, half of GCPs are sending data and half of the GCPs are receiving data during the synchronization command. In addition, at any given moment of synchronization, any chain of successive links on ℓ must be acyclic to prevent hanging. Synchronization process in this model, therefore, has two objectives: (1) creation of a unified timing structure that ensures synchronization across all nodes on level ℓ without program hanging and (2) derivation of $\overrightarrow{D}$ and $\overrightarrow{N}$ at each link in star graph.

On each GCP every value in a 3x3x3 syncing stencil is cycled through in a preset order. With the center process of the stencil as the center of a GCP's domain, each stencil cell represents a cubic feature. A syncing vector $\overrightarrow{S}$ points from the origin to the cubic feature represented by an index of the stencil, representing a link on the star graph. MPI synchronization commands used in this framework do not resume execution until sending and receiving operation is completed. Thus, by cycling through all possible syncing vectors in a set order on all GCPs, every $\overrightarrow{S}$ at a given step of the syncing process will be parallel, ensuring that the vector field of all syncing vectors has zero curl and, hence, fulfilling objective 1. For a given $\overrightarrow{S}$, the GCPs send data in a checkerboard pattern, with one half of the GCPs acting as senders and the other half as receivers. For a sending GCP *s* operating in Ω_ℓ^s and a receiving GCP *r* operating in Ω_ℓ^r, *s* sends the cubic feature indicated by $\overrightarrow{S} = \overrightarrow{D}_s = \overrightarrow{N}_s$ to the receiver whose $\overrightarrow{D}_r = \overrightarrow{N}_r = -\overrightarrow{S}$. Then, the sending/receiving roles are reversed so that a cubic feature Ω_ℓ^r is sent to Ω_ℓ^s, giving both *s* and *r* the data required to update their respective cubic feature. This process is repeated for all $\overrightarrow{S}$ such that any GCP *n* on ℓ can perform stencil operations anywhere in Ω_ℓ^n.

While $\overrightarrow{D}$ and $\overrightarrow{N}$ are parallel to $\overrightarrow{S}$ for interior synchronization events, syncing events on the border of Ω_ℓ involve cubic features that do not correspond to the syncing stencil. In these situations, vectors $\overrightarrow{N}$ and $\overrightarrow{D}$ are derived from projections of $\overrightarrow{S}$, thus linking objective 2 to objective 1. This allows the model to consistently synchronize information across all GCPs on ℓ without interaction from AdN and without forming cyclic subgraphs.

In the case of restriction, information must be consolidated from GCPs that exist on levels greater than or equal to $\ell+1$ to GCPs that operate on both level ℓ and level $\ell+1$. As represented in Fig. 1, the filling method creates 2x2 squares of GCP domains. Each square contains a single GCP (G_ℓ) whose operating domain spans partitions of both ℓ and $\ell+1$. G_ℓ's domain is at the minimum (*i*,*j*) corner of the 2x2 square. Restriction is performed locally on each GCP on $\ell+1$, and the results are consolidated along the *j*-axis first followed by the *i*-axis at the corresponding G_ℓ. For each 2x2 square, this process moves all restriction information to each G_ℓ while parallelizing the restriction process. Likewise, prolongation involves distributing level ℓ data to all the corresponding GCPs on level $\ell+1$. Distribution process reverses the consolidation process by distributing first from the G_ℓ along the *i*-axis and then the *j*-axis. Prolongation calculations are then done locally on all GCPs on level $\ell+1$. On level $\ell+2$ the restriction and prolongation processes scale to include nodes from both level ℓ (G_ℓ) and nodes on level $\ell+1$ ($G_{\ell+1}$). For this 3D model the preceding restriction and prolongation processes were scaled to a 2x2x2 cube region for each G_ℓ.

2.6. *Performance timing*

All timing results for openMP vs. CUDA test and MPI tests were obtained using time.h clock statements and operated on a reference homogenous tumor shape with heterogeneous vasculature created for this model runtime. Computer used for comparing openMP to MPI-CUDA framework has AMD 2990WX 32-core processor, two Titan RTX GPUs with computation load placed allocated to the non-display GPU, and 128 GB of DDR4 RAM at 2666 MHz. Both GPUs were set to WDDM mode. CUDA test case consists of two MPI processes: one AdP and one GCP. When running on a single PC, negligible overhead occurs from process communication thus a two process MPI-CUDA runtime is akin to a single process CUDA task, differing only in slight overhead due convergence decisions and process initialization steps. Furthermore, MPI-CUDA runtime was configured to run on a single, non-display Titan RTX GPU. Parameters other than time step size and tolerance were same as [39]. Time step size and tolerance for openMP and CUDA-only tests was $\theta = 5*10^{-3}$, and $\tau_{\ell_{max}} = 1*10^{-3}$, respectively. Lastly, each of the University of Kentucky's Lipscomb Compute Cluster (LCC) nodes used for MPI tests comprises two 20-core Intel Xeon 6230 processors, 192 GB of RAM, and four Nvidia V100 32 GB GPUs.

2.7. *Simulation of cm-scale tumors with sub-mm resolution*

Two large tumors were simulated: (1) ~1 cm tumor in a 256^3 domain, and (2) ~2 cm tumor in a 512^3 domain. The two simulated tumors are identical in shape to the performance tests, being homogeneously defined with an initial volume fraction $\tilde{\phi}_V = 0.65$. The shape was defined using a combination of sinusoidal functions and bivariate normal distributions (as shown in Graphical Abstract). The initial conditions are included in **Supplement**. This domain size was derived from the diffusivity of oxygen (10^{-5} cm^2/s [61,62]). Both 256^3 and 512^3 simulations operated with resolution of 50 μm lengths ($1.25 * 10^{-4}$ mm^3). Eight nodes on the LCC with eight CUDA-solving processes per node were used for both simulations. Screenshots were taken for the initial state, 167 time steps into the simulation (5 simulated days) and 267 time steps into the simulation (8 simulated days). Table 2 lists computational solver parameters used for both domains.

To increase model metabolite stability in the charge-balance equation, sodium concentrations were introduced throughout the model domain. A small increase in concentration of carbon dioxide, lactate, bicarbonate, and H^+ at the borders was applied to ensure convergence. As shown in Fig. 7, the distance traveled by molecules before being absorbed by blood vasculature is significantly smaller than the distance between the tumor and the domain borders. This change, then, does not significantly affect the outcome by later points in the simulation. Before vasculature formation, the molecular species travel closer to the model borders, as noted in the smaller 256^3 case. To more accurately simulate a larger tumor mass and to provide sufficient oxygen and glucose to the model domain, more mature vascularization was created at the model borders and within the domain. Biological variables and parameters for long term 256^3 and 512^3 simulations are in **Supplement**.

Table 2
Computational parameters from Ref. [39] used for tumor simulations. Initial values are set pre-model runtime. Values not listed are same as [39].

Parameter	Description	Value Assigned for cm-Scale Runs	
		256^3	512^3
ℓ_{global}	Finest level that always spans Ω	2	
ℓ_{max}	Finest grid used for Ω	4	
σ	Tolerance reduction factor from level ℓ to $\ell + 1$	1.10	
θ	Time step size (days)	$3*10^{-2}$	
γ	Cycle Index (1 for V-cycle, 2 for W-cycle)	1	
$\tau_{\ell_{max}}$	Solution Tolerance for level ℓ_{max}	$2*10^{-3}$	
$\nu_0, \nu_1, \nu_2, \nu_b$	Preset number of smoothing steps	4,2,2,2	
r_{max}	Maximum number of Smoothing Steps before Divergence Exception is raised	15	45
C_ℓ	Maximum gradient difference allowed for Universal Gradient Test for the FLAG routine on ϕ_V.	0.05	
d_ℓ	Number of cells inward from the bound to include in near-boundary extra smoothing steps (for zero-indexed level ℓ)	2^ℓ	

2.8. *Statistics*

All timing results were obtained for at least n = 3 simulation runs, with error bars representing 95% confidence interval (CI) of average value.

3. Results

3.1. *Comparison of openMP and single MPI-CUDA process: CUDA contribution to model speedup*

Tumor simulated for openMP vs. CUDA-only test was a 128^3 domain with initial condition in Supplementary Fig. 1A. Domain size was cubic with 4 mm side length; thus, resolution of simulation was 31.25 μm (from diffusivity of oxygen, 10^{-5} cm^2/s [61,62]). Parameter values were unchanged from Refs. [38,39]. Each coarser level $\ell - 1$ has twice the distance between points as its corresponding finer level ℓ. From Fig. 2A, a 14.7x performance increase of CUDA over openMP was seen for first-time step. In the second time step, our CUDA framework was 7.9x times faster than openMP. Due to corrections made in MPI-CUDA framework, this approach converged with fewer cycles than original openMP implementation. It is probable that convergence would be improved in openMP-based code if flux term changes were applied. However, we verified performance improvement by evaluating time per smoothing step. Fig. 2B shows 10.7x improvement over original openMP implementation. Further, because of adaptive grid methods described in Ref. [39] that were used in openMP Multigrid algorithm, only a subset of the domain was solved over on the finest two levels of simulation; thus, openMP spends more time doing less computational work than our CUDA framework. Finally, because AMD 2990WX possess 32 cores, it is difficult to find current single socket computers capable of equivalent performance. Although it is possible that AMD 2990WX memory bandwidth may not fully utilize all CPU cores as effectively on a multi-die CPU system, the distinction is unlikely to close the performance gap. Consequently, it is reasonable to expect that performance improvements of CUDA over openMP will scale across other platforms.

Model accuracy was ensured by comparing model input to openMP numerical solutions in Refs. [38,39]. To ensure model consistency across varying numbers of MPI thread counts, initial conditions and end-state after two time steps were compared between separate runs using SHA-256 hash. All algorithm behavior is represented in first two-time steps; thus, comparing first two time steps is sufficient to confirm solution integrity. This hash was created with printouts of volume fractions, pressures, metabolites, growth factors, and other model variables. Matching hashes implied that integrity of solving process was not impacted during development or by varying thread counts. This analysis also confirmed that MPI synchronization produced equivalent results for 1, 8, and 64 GCPs.

3.2. *MPI contribution to model speedup*

To confirm that MPI increases performance over a single GCP MPI-CUDA instance, the LCC was used. Using same initial condition (Supplementary Fig. 1B) and resolution of 31.25 μm, domain size was increased to a cube with 256^3 interior points (258^3 including border points). Thus, domain size was cubic with 8 mm side length. All computational parameter values differing from Refs. [38,39] are in Table 3.

Triplicate tests were performed at three different numbers of MPI processes:

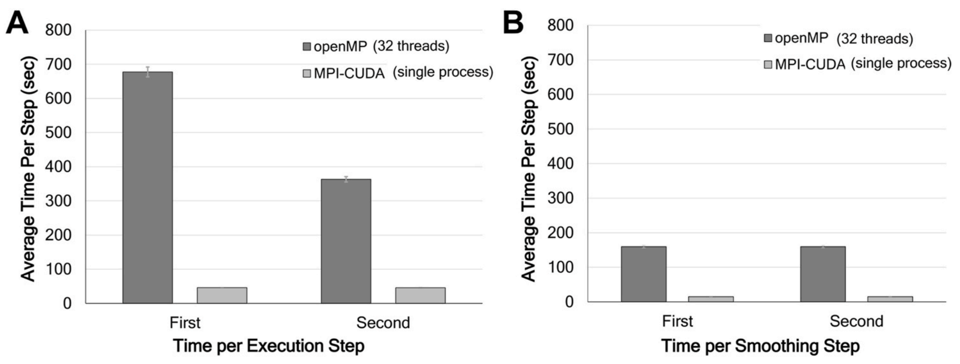

Fig. 2. Performance comparison for first two time steps between openMP and single-process MPI-CUDA instance in 128^3 domain. (A) Execution time; (B) V-cycle solvers. Error bars are 95%.

Table 3
Computational parameter values for CUDA-MPI tests. All other parameters were retained from Refs. [38,39].

Computational Parameter	Description	openMP vs. MPI-CUDA Single Process	MPI-CUDA LCC
$n_{\ell_{max}}$	Edge length of finest cubic domain $\Omega_{\ell_{max}}$	128	256
Domain Size	Side length of $\Omega_{\ell_{max}}$ (mm)	0.4	0.8
σ	Tolerance reduction factor from level ℓ to $\ell+1$	4.0	
$\tau_{\ell_{max}}$	Solution Tolerance for level ℓ_{max}	0.001	

1. Two process MPI-CUDA case: one AdP and one GCP. This test was akin to that used on AMD 2990WX test and baseline for cases 2 & 3.
2. Nine process MPI-CUDA case: one AdP and eight GCPs. For an eight GCP setup the finest model domain was split into octants, with each process operating over a single octant. In this setup, each LCC Restriction operation from ℓ_{max} to $\ell_{max}-1$ maps eight GCPs to a single GCP for processing. One LCC node was used with two MPI-CUDA processes assigned to two of the four available cluster GPUs.
3. Sixty-five process MPI-CUDA case: one AdP and 64 GCPs. For this setup, an extra layer of 64 nodes is added at the finest level. Restricted information is sent to layer $\ell-1$ containing eight of the 64 GCPs. Further restrictions to levels $\ell-2$ and coarser behave identically to case 2. Four LCC nodes were used with four MPI-CUDA processes assigned to each of the 16 available GPUs.

Single process case used same test case in openMP vs. CUDA test but scaled to larger domain. Eight-GCP and 64-GCP cases were distributed to two and four nodes, respectively. To measure effect of process density of each GPU on performance, for two groups, (1) 8 GCPs and one AdP and (2) 64 GCPs and one AdP, the number of nodes available was doubled; thus, 8-GCP-Low-Density test had two nodes with four GCPs per node while 64-GCP-Low-Density test had eight nodes with eight GCPs per node.

Averaging the ratios in timing results when moving from 8 GCPs to 64 GCPs for time steps 2 and 3 in Fig. 3, a total improvement of 5.3x is observed. Multi-process allocation can bottleneck due to competition for memory bandwidth and simulation speeds. In eight-GCP case, four GPUs held two GCPs each; 16 GPUs held four processes each in the 64-GCPs case. To quantify performance lift by redistributing processes across more GPUs, two extra runs were performed with eight GPUs running one process apiece, increasing performance by 1.2x over the original case. This decreased the performance impact of switching to 64 processes from 2.8x in four GPU case to 2.4x in the eight GPU case. Similar to 8-process case, moving to 64 processes at 2 processes per GPU (for a total of 32 GPUs) increased performance 1.3x over 64-process runtime with 4 processes per GPU (16 GPUs). In total 64-GCP-Low-Density distribution outperforms CUDA-only distribution (1-GCP) by 6.7x. Combined with gains with CUDA over openMP, MPI-CUDA framework has capacity to simulate larger tumor masses in a distributed manner at speeds not possible under the previous framework. Furthermore, larger scale simulations benefit from increased resource availability, making 64 GCPs the selected distribution method for 256^3 domain and 512^3 domain simulations. However, there are diminished gains for scaling across more nodes, suggesting that this approach may weakly scale to project size.

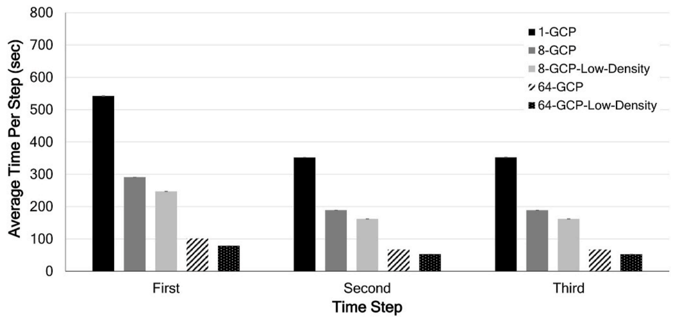

Fig. 3. Mean performances per time step for MPI-CUDA processing a 256^3 domain. Error bars are 95%.

3.3. *Simulation of cm-scale tumors*

The 256^3 simulation that ran on 32 V100 GPUs took about 2 h real time to simulate 5 simulated days with an average time per time step of 43.2 s. An additional 93.8 min were required to reach 8 simulated days. For 512^3 simulation, same 32 V100 GPU setup took ~31.5 h to reach 5 simulated days at average rate of 11.3 min per time step. Additional 26.5 h were required to reach 8 simulated days. Here, average rate per time step increased to 15.9 min per time step.

Fig. 4 shows 512^3 domain simulation of ~2 cm diameter tumor at 5 and 8 simulated days. Viable and dead tissues are evident. Pronounced release of tumor angiogenic factors (TAF) is triggered by hypoxia, which leads to angiogenesis and growth of blood vasculature (Fig. 5). However, both blood and lymphatic vasculature concentrations decreased overall. Cellular respiration leveraged increased oxygen supply, thereby raising carbon dioxide concentration. ECM concentration remained relatively stable (Fig. 6). As tumor mass compressed internally, matrix degrading enzymes (MDE) concentration shifted away from periphery, explaining local ECM loss at $i = 1.27$ cm plane. Decline can also be attributed to lower concentration of myofibroblasts in inner tumor. Because

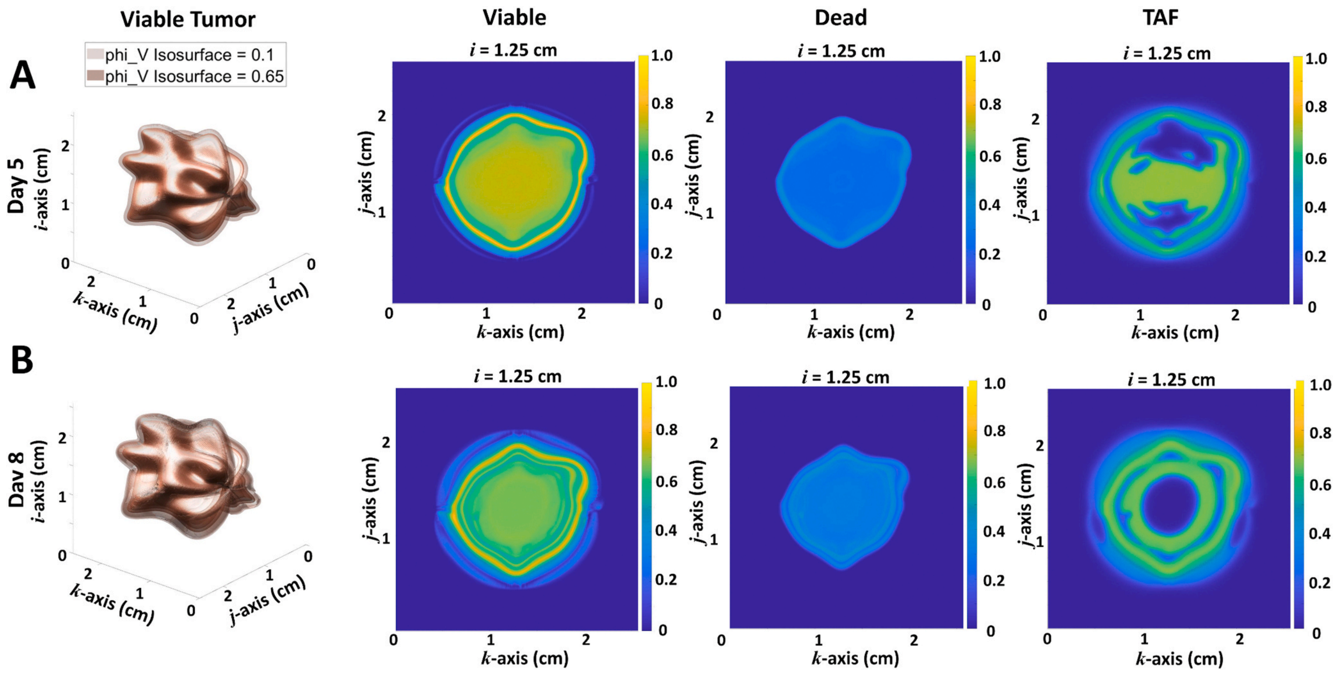

Fig. 4. Evolution of ~2 cm diameter tumor in 512^3 domain at simulated (A) 5 days and (B) 8 days. Viable, dead, and tumor angiogenic factors (TAF) are shown (plane *jk*).

myofibroblasts are created by the model within ECM and become necrotic at low oxygen levels, their concentration remained relatively stable from 5 to 8 simulated days. Meanwhile, a layer of higher viability tumor mass formed near vasculature in the peritumoral space. Negative pressure from tumor and ECM necrosis shifted the viable tumor layer towards interior regions, distancing this viable tissue from blood vasculature. This layer became necrotic and is present at both 5 simulated days and 8 simulated days. While tumor growth factor (TGF) concentration rose in peritumoral range over the 3 simulated day period, encouraging increased tumor proliferation at periphery, MDE concentration decreased locally at $i = 1.27$ cm plane.

The highly hypoxic nature of the tumor resulted in a persistently high H^+ and lactic acid concentration from bicarbonate buffer and anaerobic glycolysis (Fig. 7). Glucose, being necessary for both aerobic and anaerobic glycolysis, is scarcer in the internal tumor portions and continues to decrease in peritumoral region over time. Carbon dioxide, being formed by aerobic glycolysis, was consumed, in part, by the bicarbonate buffer, increasing bicarbonate prevalence.

Simulations of ~1 cm tumor in 256^3 domain generally yielded similar results as the larger tumor in 512^3 domain using same parameter values (Fig. 8). After an initial drop in mass due to lag in angiogenic response, both tumors assumed a growth pattern by 8 simulated days. Despite an overall decrease in density in the interior portions of the tumor, the 512^3 $\tilde{\phi}_V = 0.1$isosurface exhibited a growth rate of 2.6% volume per day. The more robust (blood and lymphatic) vascularization of the smaller tumor is evident at this timepoint earlier than in larger tumor, as are higher TAF and MDE concentrations.

4. Discussion

This study implements a distributed computing (parallelized) implementation of the mixture model in Refs. [38,39] to simulate 3D continuum tumor growth at cm-scale at sub-mm resolution. Compared to previous work, the model here accounts for a richer set of biological phenomena, simulating ECM-tumor interaction, blood and lymphatic vasculature evolution, metabolic consequences of anaerobic respiration, acidity induced via the bicarbonate buffer, and secretion of diffusible factors in response to hypoxic conditions. These results highlight how identical parameter sets perform at in different domain sizes, 256^3 and 512^3, suggesting future work required to fine-tune parameter sets best suited for large-scale growth.

The CUDA-MPI approach improves the model performance over the previous openMP approach [39] by ~50x. This value comes from accumulating the benefits seen across the smoothing test from openMP vs. CUDA (Fig. 2B) and the step time duration between 1-GCP vs. 64-GCP test (Fig. 3). Using the 1-GCP vs. 64-GCP-Low-Density increases total performance improvement to ~70x that of the previous openMP approach. Because of differing testing approaches and non-equivalent hardware, these performance improvements cannot be directly compared to other tumor modeling approaches. Nevertheless, cumulative improvements demonstrated here are in similar league to those in Refs. [47,48,52], demonstrating immediate benefits of using CUDA-MPI over openMP. To our knowledge, these results mark the first time a multigrid 3D continuum tumor model has been fully parallelized. Based on the performance metrics obtained, we anticipate future work to simulate tumor sizes as in Ref. [54]. These improvements are also consistent with Navier-Stokes Multigrid simulations where CPU-side parallelization across 64 processors saw a 50x improvement in speed [63]. A different Navier-Stokes solver using an MPI-CUDA framework achieved a 21x performance uplift over an 8-core Intel Xeon baseline using 2 GPUs and a 130x performance uplift using 128 GPUs [64]. Huang et al. created an MPI-CUDA framework to implement a Sparse Equations and Least Squares method for use in seismic tomography. Their results report a 37x performance uplift using 60 CPU cores relative to a single core baseline; using 60 GPUs nearly doubled performance over their 60 CPU results [65]. These values suggest that our model's performance benefits are on the same order of magnitude as similarly parallelized problems.

Continuum tumor mixture models, due to the numerous interwoven phenomena simulated, have many guiding equations, leading to multiple variables and quantities to evaluate and compute. As such, the memory required per point in model domain at level ℓ (Ω_ℓ) may be significantly higher than the raw variable count suggested in the case of Navier-Stokes equations. RAM constraints on GPUs become increasingly difficult to navigate as biological precision and generalizability are pursued by more specialized model equations. While MPI can involve

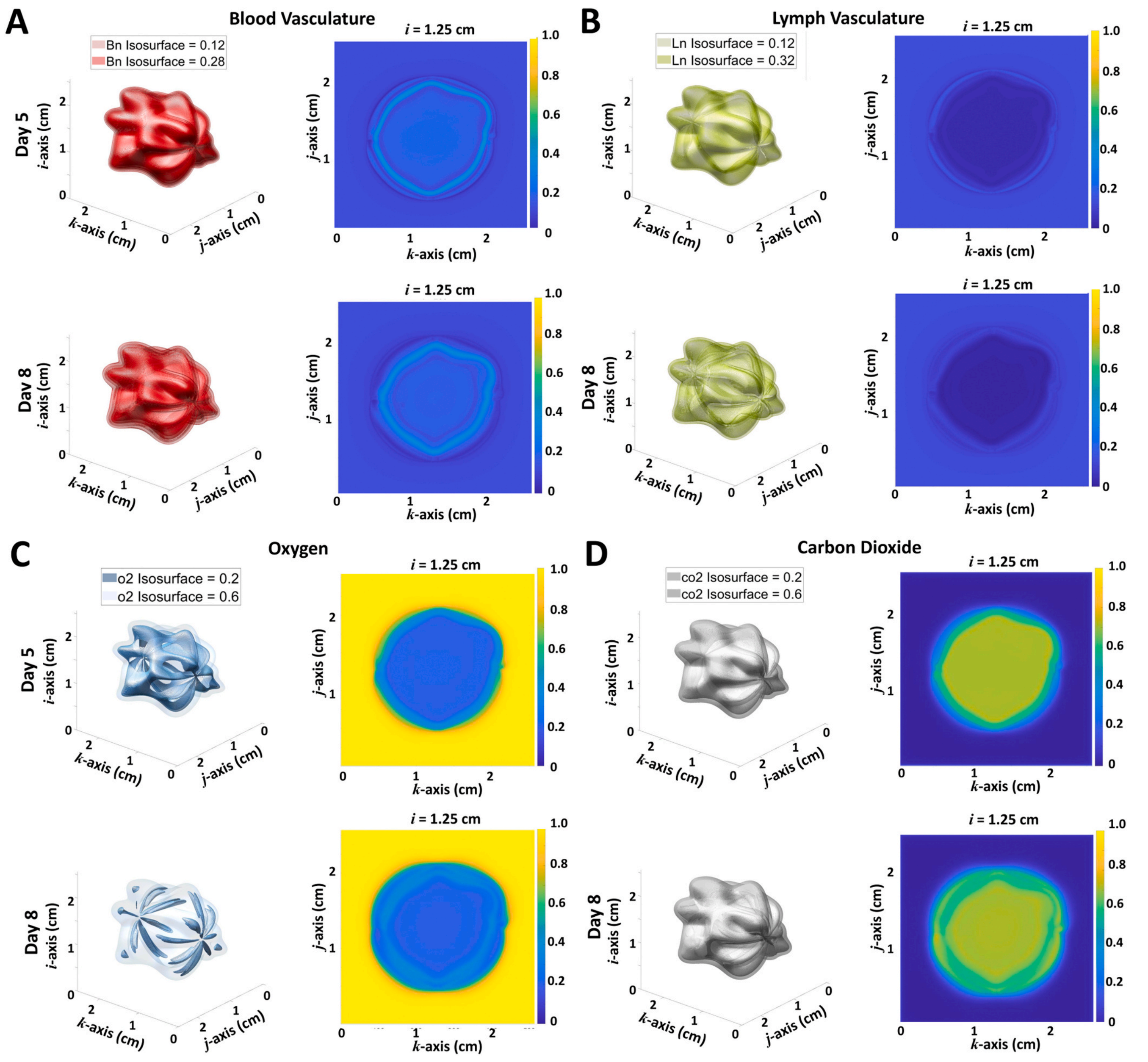

Fig. 5. Tumor vessel evolution for ~2 cm diameter tumor in 512^3 domain. (A) Blood vasculature. (B) Lymphatic vasculature. (C) Oxygen (D) Carbon dioxide.

more GPUs and lower the per-GPU RAM requirements, we recognize that this study used Titan RTX and V100 GPUs, both of which possess over 20 GB of RAM. GPUs with lower RAM capacity individually and in aggregate would with current technology have difficulty running highly-detailed continuum tumor models, potentially relegating such models to high-end desktop PCs and, in the case of cm-scale modeling, to larger compute clusters.

While the parallelization performed on the tumor model is significant, further development is required before deployment in a clinical environment. First, a more finely-tuned parameter set could help achieve persistent intra-tumoral increasing concentrations, such as ECM, required to simulate large-scale particular cancer types. Second, this process structure has no fault tolerance. If a single GCP were to fail to respond, the program would exit without completing the model. Fault tolerance has already been implemented in Big Data cluster libraries, such as Hadoop and Apache Spark [56,58]. Thus, future implementation may draw from techniques used by these Big Data frameworks.

With the addition of Multigrid technologies such as adaptive grid meshes, computational workload would be reduced and would increase model performance. For some problem sizes, a CUDA-MPI framework may not be optimal due to the overhead of passing data for processing to GPUs. Indeed, an openMP/MPI framework has outperformed CUDA-MPI tasks when operating on a smaller mathematical model [66]. It is suspected that a tradeoff point exists, which could be a subject for future research. Evaluation of openMP-MPI vs. CUDA-MPI at varied grid levels may lead to further optimizations of mixed grid sizes.

Additionally, because of parallel synchronization constraints, the adaptive grid mesh method previously used in Refs. [38,39] was discarded in favor of a modified residual calculation procedure as detailed in Methods. An adaptive grid mesh implementation would require adaptive process assignments to subsets of non-global domains and, if implemented at MPI level, may better utilize processing resources. In many cases only a single V-cycle was required to converge to selected tolerance. A minority of time steps, especially time steps directly after and including initial time step, required more smoothing iterations to achieve tolerance; it is likely that using a different multigrid cycle, such

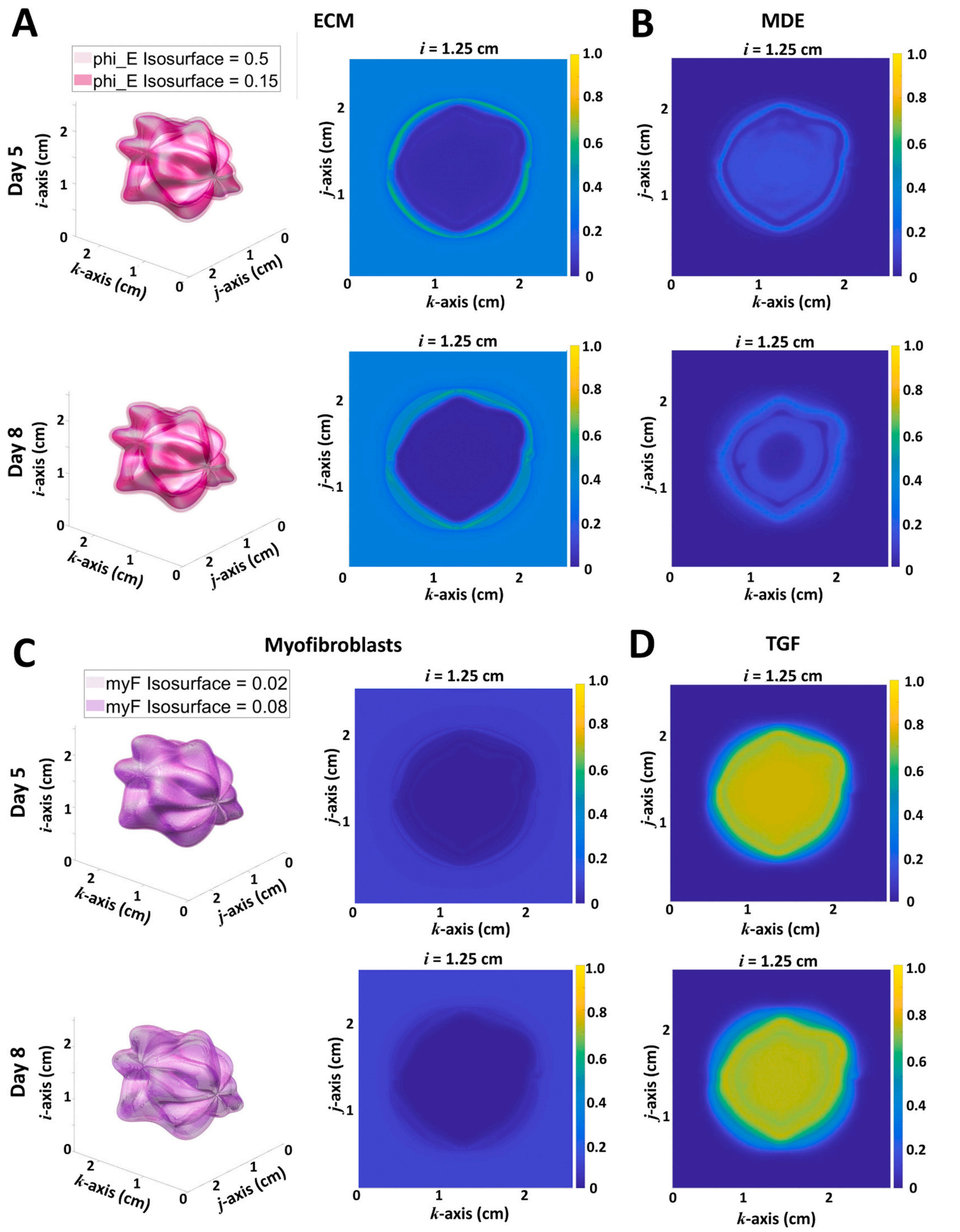

Fig. 6. Tumor matrix evolution for ~2 cm diameter tumor in 512^3 domain. (A) Extracellular matrix (ECM). (B) Matrix degrading enzymes (MDE). (C) Myofibroblasts. (D) Tumor growth factors (TGF).

as the F-cycle, would improve convergence performance in those cases [67]. Some other minor performance improvements can be made, such as consolidating the AdP with a GCP to reduce thread count by one. From a GPU standpoint, with RAM counts on GPUs increasing significantly over the past half-decade, 256^3 currently and 512^3 in the future will likely become entirely GPU-side computations within a couple GPU generations. Further refinement could thus reduce MPI's contribution by removing most memory transfers.

Because of its low cost of failure and ideal reproducibility, *in silico* simulation of clinically-relevant tumor-sized growth could help to analyze patient treatment, especially when coupled to tumor-specific parameters. Flexibility afforded by parameters leveraged in this model may yield a platform for accommodating a wide range of characteristics, anticipating tumor evolution and forecasting on patient potential outcomes. Further, discovering which model parameters influence positive clinical outcome could posit opportunities for novel clinical approaches and provide a basis for further exploration. A faster turnaround would offer a more responsive methodology of engaging with oncological hypothesis testing and to focus *in vitro* and *in vivo* experimental effort.

With the complexity and scale of the model, the number of parameters makes assumptions inevitable. Akin to the reliance of machine learning on high data acquisition for training sets, determining patient-

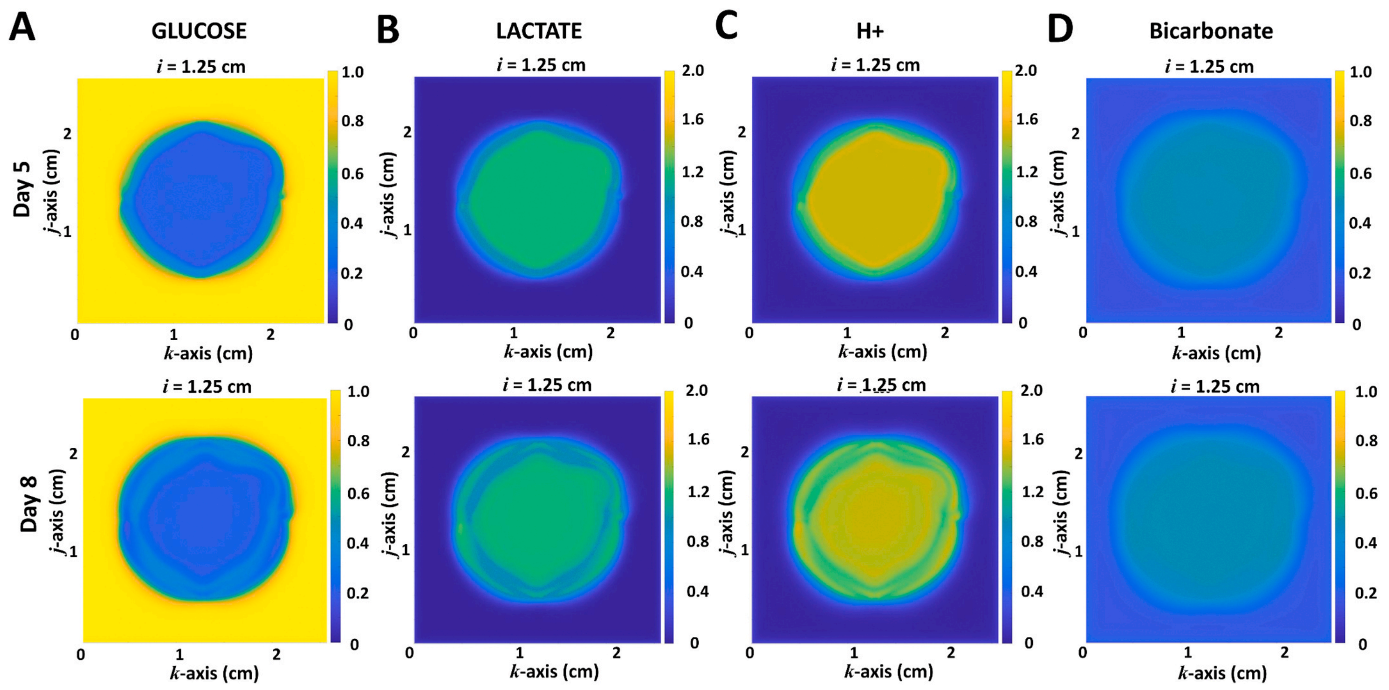

Fig. 7. Evolution of metabolism-related variables for ~2 cm diameter tumor in 512^3 domain. (A) Glucose. (B) Lactate. (C) Hydrogen ion (H^+). (D) Bicarbonate.

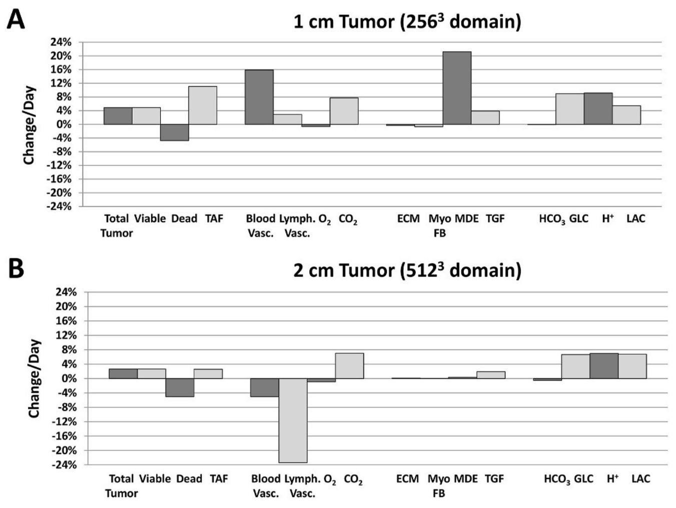

Fig. 8. Rate of change of tumor variables (%change/day) at 8 simulated days for (A) 256^3 domain and (B) 512^3 domain. From left to right: total tumor; viable tumor; dead tumor; TAF (tumor angiogenic factors); blood vasculature; lymphatic vasculature; O_2 (oxygen); CO_2 (carbon dioxide); ECM (extracellular matrix); myoFB (myofibroblasts); MDE (matrix degrading enzymes); TGF (tumor growth factors); GLC (glucose); LAC (lactate); H^+ (hydrogen ion); HCO_3 (bicarbonate).

specific parameter values will require integrating *in silico* evaluation with relevant clinical data. This requirement is exacerbated as more detailed biological phenomena are considered. For example, introducing immunotherapies and immuno-onco interactions will require additional parameters, meaning that balancing performance with model complexity will continue to affect larger-scale continuum tumor modeling. Despite limitations, this study presents a first step in achieving centimeter-scale 3D continuum tumor simulations with sub-millimeter resolution, with future work envisioned to move this approach closer to clinical application.

Conflict of interests statement

The authors declare no known conflicts of interest.

Declaration of competing interest

The authors have no competing interests to disclose.

Acknowledgements

HBF acknowledges partial support by National Institutes of Health/ National Cancer Institute grant R15CA203605 and Department of Defense/U.S. Army Medical Research grant W81XWH2110012. This work was conducted in part using resources of the Research Computing group and Cardinal Research Cluster (CRC) at University of Louisville. Authors thank for assistance Harrison Simrall with the CRC and Steven Goodin with matrix unrolling techniques. Authors acknowledge University of Kentucky Center for Computational Sciences and Information Technology Services Research Computing for support and use of Lipscomb Compute Cluster (LCC) and associated computing resources; this material is based upon work supported by National Science Foundation grant 1925687.

Appendix A. Supplementary data

Supplementary data to this article can be found online at https://doi.org/10.1016/j.compbiomed.2021.104507.

References

[1] S.M. Wise, J.S. Lowengrub, H.B. Frieboes, V. Cristini, Three-dimensional multispecies nonlinear tumor growth—I: model and numerical method, J. Theor. Biol. 253 (2008) 524–543.
[2] J.T. Oden, A. Hawkings, S. Prudhomme, General diffuse-interface theories and an approach to predictive tumor growth modeling, Math. Model Methods Appl. Sci. 2 (20) (2010) 477–517.
[3] V. Cristini, X. Li, J.S. Lowengrub, S.M. Wise, Nonlinear simulations of solid tumor growth using a mixture model: invasion and branching, J. Math. Biol. 58 (2009).
[4] A. Hawkins-Daarud, S. Prudhomme, K.G. van der Zee, J.T. Oden, Bayesian calibration, validation, and uncertainty quantification of diffuse interface models of tumor growth, J. Math. Biol. 67 (2013) 1457–1485.

[5] V. Cristini, H.B. Frieboes, X. Li, J. Lowengrub, P. Macklin, S. Sanga, S.M. Wise, X. Zheng, Nonlinear Modeling and Simulation of Tumor Growth, Selected Topics in Cancer Modeling, Birkhäuser, Boston, 2008, pp. 1–69.
[6] S. Frigeri, M. Grasselli, E. Rocca, On a diffuse interface model of tumour growth, Eur. J. Appl. Math. 26 (2015) 215–243.
[7] C. Cavaterra, E. Rocca, H. Wu, Long-Time Dynamics and Optimal Control of a Diffuse Interface Model for Tumor Growth, Applied Mathematics & Optimization, 2019.
[8] P. Colli, G. Gilardi, E. Rocca, J. Sprekels, Optimal distributed control of a diffuse interface model of tumor growth*, Nonlinearity 30 (2017) 2518.
[9] J.P. Ward, J.R. King, Mathematical modelling of avascular-tumour growth, IMA J. Math. Appl. Med. Biol. 14 (1997) 39–69.
[10] C. Please, G. Pettet, D. McElwain, A new approach to modeling the formation of necrotic regions in tumors, Appl. Math. Lett. 11 (1998) 89–94.
[11] J.P. Ward, J.R. King, Mathematical modelling of avascular-tumour growth. II: modelling growth saturation, IMA J. Math. Appl. Med. Biol. 16 (1999) 171–211.
[12] C. Please, G. Pettet, D. McElwain, Avascular tumour dynamics and necrosis, Math. Methods Appl. Sci. 9 (1999) 569–579.
[13] C.J. Breward, H.M. Byrne, C.E. Lewis, The role of cell-cell interactions in a two-phase model for avascular tumour growth, J. Math. Biol. 45 (2002) 125–152.
[14] D. Ambrosi, L. Preziosi, On the closure of mass balance models for tumor growth, Math. Model Methods Appl. Sci. 12 (2002) 737–754.
[15] C.J. Breward, H.M. Byrne, C.E. Lewis, A multiphase model describing vascular tumour growth, Bull. Math. Biol. 65 (2003) 609–640.
[16] H. Byrne, J. King, D. McElwain, L. Preziosi, A two-phase model of solid tumour growth, Appl. Math. Lett. 16 (2003) 567–573.
[17] H. Byrne, L. Preziosi, Modelling solid tumour growth using the theory of mixtures, Math. Med. Biol. 20 (2003) 341–366.
[18] S.J. Franks, H.M. Byrne, J.R. King, J.C. Underwood, C.E. Lewis, Modelling the early growth of ductal carcinoma in situ of the breast, J. Math. Biol. 47 (2003) 424–452.
[19] S.J. Franks, H.M. Byrne, H.S. Mudhar, J.C. Underwood, C.E. Lewis, Mathematical modelling of comedo ductal carcinoma in situ of the breast, Math. Med. Biol. 20 (2003) 277–308.
[20] T. Roose, P.A. Netti, L.L. Munn, Y. Boucher, R.K. Jain, Solid stress generated by spheroid growth estimated using a linear poroelasticity model, Microvasc. Res. 66 (2003) 204–212.
[21] R. Araujo, D. McElwain, A mixture theory for the genesis of residual stresses in growing tissues I: a general formulation, SIAM J. Appl. Math. 65 (2005) 1261–1284.
[22] R. Araujo, D. McElwain, A mixture theory for the genesis of residual stresses in growing tissues II: solutions to the biphasic equations for a multicell spheroid, SIAM J. Appl. Math. 66 (2005) 447–467.
[23] M.A. Chaplain, L. Graziano, L. Preziosi, Mathematical modelling of the loss of tissue compression responsiveness and its role in solid tumour development, Math. Med. Biol. 23 (2006) 197–229.
[24] A. Tosin, Multiphase modeling and qualitative analysis of the growth of tumor cords, Netw. Heterogeneous Media 3 (2008) 43–84.
[25] D. Ambrosi, L. Preziosi, Cell adhesion mechanisms and stress relaxation in the mechanics of tumours, Biomech. Model. Mechanobiol. 8 (2009) 397–413.
[26] D. Ambrosi, A. Duperray, V. Peschetola, C. Verdier, Traction patterns of tumor cells, J. Math. Biol. 58 (2009) 163–181.
[27] L. Preziosi, A. Tosin, Multiphase modelling of tumour growth and extracellular matrix interaction: mathematical tools and applications, J. Math. Biol. 58 (2009) 625–656.
[28] L. Preziosi, A. Tosin, Multiphase and multiscale trends in cancer modelling, Math. Model Nat. Phenom. 4 (2009) 1–11.
[29] P. Tracqui, Biophysical models of tumor growth, Rep. Prog. Phys. 72 (2009), 056701.
[30] G. Lorenzo, M.A. Scott, K. Tew, T.J.R. Hughes, Y.J. Zhang, L. Liu, G. Vilanova, H. Gomez, Tissue-scale, personalized modeling and simulation of prostate cancer growth, Proc. Natl. Acad. Sci. U. S. A 113 (2016) E7663–E7671.
[31] M. Antonopoulos, D. Dionysiou, G. Stamatakos, N. Uzunoglu, Three-dimensional tumor growth in time-varying chemical fields: a modeling framework and theoretical study, BMC Bioinf. 20 (2019) 442.
[32] S.M. Wise, J.S. Lowengrub, V. Cristini, An adaptive multigrid algorithm for simulating solid tumor growth using mixture models, Math. Comput. Model. 53 (2011) 1–20.
[33] H.B. Frieboes, F. Jin, Y.-L. Chuang, S.M. Wise, J.S. Lowengrub, V. Cristini, Three-dimensional multispecies nonlinear tumor growth—II: tumor invasion and angiogenesis, J. Theor. Biol. 264 (2010) 1254–1278.
[34] A.R. Anderson, M. Chaplain, Continuous and discrete mathematical models of tumor-induced angiogenesis, Bull. Math. Biol. 60 (1998) 857–899.
[35] S.R. McDougall, A.R. Anderson, M.A. Chaplain, J.A. Sherratt, Mathematical modelling of flow through vascular networks: implications for tumour-induced angiogenesis and chemotherapy strategies, Bull. Math. Biol. 64 (2002) 673–702.
[36] M.J. Plank, B.D. Sleeman, A reinforced random walk model of tumour angiogenesis and anti-angiogenic strategies, Math. Med. Biol. 20 (2003) 135–181.
[37] M.J. Plank, B.D. Sleeman, Lattice and non-lattice models of tumour angiogenesis, Bull. Math. Biol. 66 (2004) 1785–1819.
[38] C.F. Ng, H.B. Frieboes, Model of vascular desmoplastic multispecies tumor growth, J. Theor. Biol. 430 (2017) 245–282.
[39] C.F. Ng, H.B. Frieboes, Simulation of multispecies desmoplastic cancer growth via a fully adaptive non-linear full multigrid algorithm, Front. Physiol. 9 (2018).
[40] R. Wcisło, W. Dzwinel, Particle Model of Tumor Growth and its Parallel Implementation, Parallel Processing and Applied Mathematics : 8th International Conference, PPAM 2009, Wroclaw, Poland, September 13-16, 2009. Revised Selected Papers, Part I, Springer Berlin Heidelberg, Berlin, Heidelberg, 2010, pp. 322–331.
[41] R. Wcisło, P. Gosztyła, W. Dzwinel, N-body parallel model of tumor proliferation, in: Proceedings of the 2010 Summer Computer Simulation Conference, 2010, pp. 160–167.
[42] A. Ghaffarizadeh, S.H. Friedman, P. Macklin, BioFVM: an efficient, parallelized diffusive transport solver for 3-D biological simulations, Bioinformatics 32 (2016) 1256–1258.
[43] N. Chen, J.A. Glazier, J.A. Izaguirre, M.S. Alber, A parallel implementation of the Cellular Potts Model for simulation of cell-based morphogenesis, Comput. Phys. Commun. 176 (2007) 670–681.
[44] M. Berghoff, J. Rosenbauer, A. Schug, Massively Parallel Large-Scale Multi-Model Simulation of Tumor Development, 2019.
[45] M. Berghoff, J. Rosenbauer, A. Schug, Massively Parallel Large-Scale Multi-Model Simulation of Tumour Development Including Treatments, John von Neumann-Institut für Computing, 2020. NIC Symposium 2020.
[46] M. Berghoff, I. Kondov, J. Hotzer, Massively parallel stencil code solver with autonomous adaptive block distribution, IEEE Trans. Parallel Distr. Syst. 29 (2018).
[47] J.J. Tapia, R. D'Souza, M. Ieee, International Conference on Systems, S.M.C.S.A.T. X.U.S.A. Cybernetics, Data-parallel algorithms for large-scale real-time simulation of the cellular potts model on graphics processing units, in: Conference Proceedings - IEEE International Conference on Systems, Man and Cybernetics, 2009, pp. 1411–1418.
[48] J.J. Tapia, R.M. D'Souza, Parallelizing the Cellular Potts Model on graphics processing units, Comput. Phys. Commun. 182 (2011) 857–865.
[49] A.G. Salguero, A.J. Tomeu, M.I. Capel, B. th International Conference on Practical Applications of Computational, P.t. Bioinformatics, Parallel cellular automaton tumor growth model, Adv. Intell. Syst.Comput 803 (2019) 175–182.
[50] A.G. Salguero, A.J. Tomeu-Hardasmal, M.I. Capel, Dynamic Load Balancing Strategy for Parallel Tumor Growth Simulations, J. Integr. Bioinf. 16 (2019).
[51] A.J. Tomeu, A.G. Salguero, M.I. Capel, Speeding Up Tumor Growth Simulations Using Parallel Programming and Cellular Automata, IEEE Latin America Transactions 14 (2016).
[52] S. Dong, Y. Yan, L. Tang, J. Meng, Y. Jiang, Simulation of 3D tumor cell growth using nonlinear finite element method, Comput. Methods Biomech. Biomed. Eng. 19 (2016) 807–818.
[53] M. Łoś, A. Kłusek, M.A. Hassaan, K. Pingali, W. Dzwinel, M. Paszyński, Parallel fast isogeometric L2 projection solver with GALOIS system for 3D tumor growth simulations, Comput. Methods Appl. Mech. Eng. 343 (2019) 1–22.
[54] A. Klusek, M. Los, M. Paszynski, W. Dzwinel, Efficient model of tumor dynamics simulated in multi-GPU environment, Int. J. High Perform. Comput. Appl. 33 (2019) 489–506.
[55] M.A. Swartz, A.W. Lund, Lymphatic and interstitial flow in the tumour microenvironment: linking mechanobiology with immunity, Nat. Rev. Canc. 12 (2012) 210–219.
[56] H. Asaadi, D. Khaldi, B. Chapman, C. Ieee International Conference on Cluster Computing, A comparative survey of the HPC and big data paradigms: Analysis and experiments, Proceedings - IEEE International Conference on Cluster Computing, ICCC (2016) 423–432.
[57] J. Dongarra, B. Tourancheau, S. Kamburugamuve, P. Wickramasinghe, S. Ekanayake, G.C. Fox, Anatomy of machine learning algorithm implementations in MPI, Spark, and Flink, Int. J. High Perform. Comput. Appl. 32 (2018) 61–73.
[58] J.L. Reyes-Ortiz, L. Oneto, D. Anguita, Big Data Analytics in the Cloud: Spark on Hadoop vs MPI/OpenMP on Beowulf, Procedia Computer Science 53 (2015) 121–130.
[59] S. Canon, A. Gittens, E. Racah, M. Ringenburg, L. Gerhardt, J. Kottalam, J. Liu, K. Maschhoff, A. Devarakonda, J. Chhugani, P. Sharma, J. Yang, J. Demmel, J. Harrell, V. Krishnamurthy, M.W. Mahoney, U.S.A.D.D. Prabhat, Ieee International Conference on Big Data Washington Dc, Matrix factorizations at scale: A comparison of scientific data analytics in spark and C+MPI using three case studies, in: 2016 IEEE International Conference on Big Data (Big Data), 2016, pp. 204–213.
[60] S. Reiter, A. Vogel, I. Heppner, M. Rupp, G. Wittum, A massively parallel geometric multigrid solver on hierarchically distributed grids, Comput. Visual Sci. 16 (2013) 151–164.
[61] L.J. Nugent, R.K. Jain, Extravascular diffusion in normal and neoplastic tissues, Canc. Res. 44 (1984) 238–244.
[62] H.B. Frieboes, M.E. Edgerton, J.P. Fruehauf, F.R. Rose, L.K. Worrall, R.A. Gatenby, M. Ferrari, V. Cristini, Prediction of drug response in breast cancer using integrative experimental/computational modeling, Canc. Res. 69 (2009) 4484–4492.
[63] P. Benedusi, D. Hupp, P. Arbenz, R. Krause, A Parallel Multigrid Solver for Time-Periodic Incompressible Navier–Stokes Equations in 3D, in: B. Karasözen, M. Manguoğlu, M. Tezer-Sezgin, S. Göktepe, Ö. Uğur (Eds.), Numerical Mathematics and Advanced Applications ENUMATH 2015, Springer, Cham, 2015, pp. 265–273.
[64] D. Jacobsen, J. Thibault, I. Senocak, An MPI-CUDA Implementation for Massively Parallel Incompressible Flow Computations on Multi-GPU Clusters, 48th AIAA

Aerospace Sciences Meeting Including the New Horizons Forum and Aerospace ExpositionOrlando, 2010. FL.

[65] H. Huang, L. Wang, E.J. Lee, P. Chen, An MPI-CUDA Implementation and Optimization for Parallel Sparse Equations and Least Squares (LSQR), Procedia Computer Science 9 (2012) 76–85.

[66] V. Lončar, L.E. Young-S, S. Škrbić, P. Muruganandam, S.K. Adhikari, A. Balaž, OpenMP, OpenMP/MPI, and CUDA/MPI C programs for solving the time-dependent dipolar Gross-Pitaevskii equation, Comput. Phys. Commun. 209 (2016) 190–196.

[67] U. Trottenberg, C.W. Oosterlee, A. Schuller, Multigrid, Elsevier, 2000.