7-2019

Simulation of a continuum tumor model using distributed computing.

Dylan A Goodin
University of Louisville

Simulation of a Continuum Tumor Model using Distributed Computing

by

Dylan Goodin

B.S., University of Louisville, July 2019

A Thesis

Submitted to the Faculty of the

University of Louisville

J.B. Speed School of Engineering

as Partial Fulfillment of the Requirements

for the Professional Degree

MASTER OF ENGINEERING

Department of Bioengineering

July 2019

SIMULATION OF A CONTINUUM TUMOR MODEL USING DISTRIBUTED COMPUTING

Submitted by:_____

Dylan Goodin

A Thesis Approved on

_____

07/15/2019

By the Following Reading and Examination Committee

_____

Dr. Hermann Frieboes, Thesis Director

_____

Dr. Jill Steinbach-Rankins

_____

Dr. Eric Rouchka

ACKNOWLEDGEMENTS

I thank Dr. Frieboes for his patience and the opportunity to work with this model and improve it as much as I have. His confidence in me to do this work has made a world of a difference.

Special thanks to Dr. Jill Steinbach-Rankins and Dr. Eric Rouchka for taking the time to focus on this work, as it has been an intense focus of mine for a couple of years now. Thanks also to the University of Louisville's Cardinal Research Cluster team who helped me learn how the cluster operates and for maintaining the cluster for use for all researches at the University of Louisville.

I also thank the BE department of the University of Louisville for the education they have given to me over my bachelor's and master's degrees. They are fortunate to have a strong roster of knowledgeable, accessible, and friendly faculty.

ABSTRACT

Mathematical modeling aims to provide a theoretical framework for understanding tissue dynamics and for establishing treatment response for diseased tissues, such as tumors. Previously published continuum models have successfully represented idealized two-dimensional and three-dimensional tissue for short periods of time. A recently published continuum model of cancer increases model complexity and describes three-dimensional tissue that, due to the required complexity of the geometric multigrid solver, can only be feasibly applied to millimeter-scale simulations. Furthermore, the computational cost for such models has hindered their application in the laboratory and in the clinic. With computational demands greatly outpacing current openMP-based approaches on single-CPU-socket machines, higher performance solvers for large-scale tissue models remain a critical need. In this thesis, preliminary results of a CUDA and CUDA-MPI based parallelization applied to a tissue model are presented, with significant speedups seen in solution calculation for an initial time step. With further access to larger distributed computing, these parallel frameworks could potentially scale to simulate large-scale tissues.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

I. INTRODUCTION

A. Rationale for Cancer Mathematical Modeling

Cancer remains a significant disease after centuries of treatment and medical study. In the US, for instance, the number of diagnoses in 2019 is projected to be over 1.7 million, over 33% of which will die of the disease. Its effect is tantamount to heart disease over the same one-year period (Institute, 2019; Prevention, 2018).

Cancer's characteristics have been outlined thoroughly by the works of Hanahan and Weinberg as they describe mutated variants of a host cell that demands and consumes host resources to fuel an abnormal and continuous mitotic behavior that extends into the foreseeable future (Douglas Hanahan & Robert, 2011; D. Hanahan & Weinberg, 2000). Their works define a depth of knowledge that belays a desire to find a treatment method for the disease. Three overarching categories of treatment have emerged from this continued research: surgery, chemotherapy, and radiation (K. D. Miller et al., 2016; Napier, Scheerer, & Misra, 2014; Weinberg, 2013). Surgery, in the case of cancer, is a debulking process, in which a portion of a tumor mass is removed. In many cases, surgery can be combined with other treatment results with varying degrees of effectiveness (Felip et al., 2010; Rydzewska, Tierney, Vale, & Symonds, 2010; Sasako et al., 2011). However, surgery can lead to a resection of the organ in question, such as a

mastectomy or esophagectomy, for example (K. D. Miller et al., 2016; Napier et al., 2014).

While it can be used in conjunction with the other two, radiation, on its own, has benefits to the patient by causing tissue damage at the site of the injury (Weinberg, 2013). It is commonly used in addition to either surgery, chemotherapy, or, more recently, immunotherapy to improve survivability (Kang, Demaria, & Formenti, 2016; Leibovich et al., 2000; T. P. Miller et al., 1998; Ragaz et al., 1997; Weinberg, 2013). Additionally, radiation can target the cancer more locally than chemotherapy and is less intrusive than surgery. However, its role remains more preventative than curative, leaving clinicians to turn to chemotherapy as the mainstay of treatment (K. D. Miller et al., 2016).

The mechanism of action for chemotherapy varies from drug-to-drug, but the most common pathway involves the disruption of cellular replication. For example vinblastine inhibits microtubule assembly, stopping mitosis at prometaphase (Weinberg, 2013). Consequently, any cell in the patient that commonly divides will be affected, leading to hair loss and nausea. Even in cases where the patient braves the chemotherapeutic process, drug resistance may form in remaining tumors (Holohan, Van Schaeybroeck, Longley, & Johnston, 2013). If adjuvant therapies fail to remove the growth, any realized tumor resistance to previously effective drugs will complicate any future treatment plans.

It is worth noting an up and coming option for cancer patients: immunotherapy. Immunotherapy aspires to harness T-cells in the host body to reevaluate cancer cells histocompatibility by either disrupting immunoediting in tumors (i.e. disrupting CTLA-4 or PD-1/PD-L1 pathways) or performing an adoptive cell transfer (Du, Herbst, & Morgensztern, 2017; Pardoll, 2012; Rosenberg, 2014; Schumacher & Schreiber, 2015; Topalian et al., 2015; Yang, 2015). However, few people directly benefit from immunotherapy in its current form since its efficacy is limited to specific patients with certain types of cancer, e.g. lung, lymphomas, leukemias, or melanoma (Du et al., 2017; Topalian et al., 2015). As an example of the negative, metastatic epithelial cancers cannot currently be treated using immunotherapy (Topalian et al., 2015).

With an increasing number of chemotherapeutic and immunotherapeutic tools at a clinician's disposal, not to mention the ability to add treatments in conjunction to one another, the importance of creating a treatment plan is paramount. Thus, a consistent and impartial testing apparatus is critical to ensure the best patient outcome. Mathematical modeling aims to provide the consistency necessary to test treatment plans, as well as make predictions to better treatment paths (Michor, Liphardt, Ferrari, & Widom, 2011).

B. Background to Cancer Mathematical Modeling

Cancer has a rich history of mathematical modeling that extends from the middle of the 20th century, enlightening clinicians to better treatment regimens for tumor patients (Coldman & Goldie, 1986; Norton & Simon, 1977). Tumor complexity, however, makes creating a model that can accurately simulate tumors challenging, leaving researchers with room to create a swathe of cancer mathematical models (Frank, Iwasa, & Nowak, 2003; Sanga et al., 2006). More recently, models strive to increase realism by enlarging the number of phenomena considered, including thermodynamics, discrete tumor types, and vasculature constraints (Anderson & Chaplain, 1998; Frieboes et al., 2010; Wise, Lowengrub, Frieboes, & Cristini, 2008). By accounting for these additional phenomena, newer models with these compensatory equations exchange the numerical and underlying simplicity of their equations with wider applicability at the cost of mathematical complexity (Altrock, Liu, & Michor, 2015; Michor et al., 2011). The most recent of these advents is the Ng-Frieboes model that supports multispecies environments, Helmholtz energy calculations, metabolite concentrations, and vasculature in an effort to detail tumor growth in a clinically relevant manner (Ng & Frieboes, 2017, 2018). Construction of such a model yields a promising framework upon which the efficacy of drug delivery can be tested in a consistent and objective manner.

C. Background and Simulation Description of the Ng-Frieboes Tumor Model

The Ng-Frieboes tumor model as presented in Ng and Frieboes (2017) simulates the evolution of a single living phenotype of tumor, represented with a volume fraction ϕ_V, in a healthy environment filled with host cells and extracellular matrix (ECM), represented by the volume fractions ϕ_H and ϕ_E, respectively. The tumor cells vie for resources against the healthy tumor cells while balancing their need for metabolites, including oxygen, carbon dioxide, lactate, bicarbonate, sodium and chloride ions, and H^+ ions. Crowding in a limited tissue space is abstracted into solid mass pressure and pressure from surrounding fluids, represented as p and q, respectively. These pressures create velocity in the solid tissue mass u_α and create buildup of elastic energy $\mathcal{W}$ on the surrounding ECM. Matrix degrading enzymes and myofibroblast concentrations increase due to remodeling of surrounding ECM to compensate for increased strain from tumor growth.

When tumor growth factors have led to a sufficiently large tumor mass, certain parts of the tumor, such as cells surrounded by thick layers of tumor cells, can be deprived of resources. As such, the tumor use angiogenic factors to encourage vasculature growth from surrounding vessels towards the cells. Increased vessel leakiness has been well-documented from these relatively quick changes to local vasculature; the body compensates for the resulting edematous environment by increasing lymphatic

growth (Swartz & Lund, 2012). Therefore, the model simulates lymphatic growth with independent terms to the vasculature, although both are closely related both mathematically and physiologically. However, even with growth towards the tumor, the effectiveness of the vasculature is limited physiologically by oxygen's diffusion rate. Thus, interior hypoxic regions in sufficiently large tumors will operate in varying levels of anerobic glycolysis, building up lactic acid in the process. In a sufficiently hypoxic state, the tumor cells become apoptotic/necrotic, represented as the dead cell volume fraction ϕ_D.

The model's key equations exhibited in Ng and Frieboes (2018) and derived in Ng and Frieboes (2017) are presented in their nondimensionalized forms below:

$$\frac{\partial \phi_V}{\partial t} + \nabla \cdot (\phi_V u_\alpha) = M \cdot \nabla \cdot (\phi_V \nabla \mu_T) + S_V \tag{1}$$

$$\mu_E = \frac{\partial F_b}{\partial \phi_E} + \frac{\partial \mathcal{W}}{\partial \phi_E} - \epsilon_E^2 \cdot \nabla^2 \phi_E - \epsilon_{TE}^2 \cdot \nabla^2 \phi_E \tag{2}$$

$$\frac{\partial \mathcal{W}}{\partial \phi_E} = \epsilon_e \cdot [6 \cdot \phi_E (1 - \phi_E)] \cdot \sum_{i,j=1}^{3} \left[\frac{1}{2} \cdot \left(\tilde{\mathcal{E}}_T\right)_{ij} \cdot \mathbb{T}_{ij}^* - \left(\tilde{\mathcal{E}}_T^*\right)_{ij} \cdot \mathbb{T}_{ij} \right] \tag{3}$$

$$\tilde{\mathbb{T}}_{mn} = 2 \cdot L_2 \left(\tilde{\mathcal{E}}_T\right)_{mn} + L_1 \cdot \delta_{mn} \cdot \sum_{s=1}^{3} \left(\tilde{\mathcal{E}}_T\right)_{ss} \tag{4}$$

$$\nabla \cdot \left[k_\alpha \cdot (\nabla p - \frac{\gamma_T}{\epsilon_T} \mu_T \nabla \phi_T - \frac{\gamma_E}{\epsilon_E} \mu_E \nabla \phi_E \right] = -(S_V + S_D + S_E) \tag{5}$$

$$u_\alpha = -k_\alpha \cdot \left[\nabla p - \frac{\gamma_T}{\epsilon_T} \mu_T \nabla \phi_T - \frac{\gamma_E}{\epsilon_E} \mu_E \nabla \phi_E \right] \tag{6}$$

$$\nabla \cdot (D_n \nabla n) + k_{n1} n_C - (k_{n1} + k_{n2}) \cdot n = 0 \tag{7}$$

$$\nabla \cdot \left(D_{tgf} \nabla (tgf) \right) + \lambda_{tgf} - \left(\lambda_{tgf} + \lambda_{de,tgf} + \lambda_{U,tgf} \right) \cdot tgf = 0 \tag{8}$$

$$\frac{\partial B_n^E}{\partial t} + \nabla \cdot (B_n^E u_E) = -\nabla \cdot \boldsymbol{J}_{BnE} + S_{BnE} \tag{9}$$

where term values in equations 1 through 9 are given in Table I and $\delta_{mn} = \begin{cases} 0, m \neq n \\ 1, m = n \end{cases}$.

TABLE I

PARTIAL LIST OF NG FRIEBOES MODEL VARIABLES & PARAMETERS

Variable	Biological Representation	Term Definition
ϕ_V	Viable Tumor cell volume fraction	(Ng & Frieboes, 2017)
ϕ_E	Extracellular matrix volume fraction	
ϕ_H	Healthy host cells volume fraction	
u_α	Solid Cell velocity	
u_E	Extracellular Matrix velocity	
M	Mobility of cell species	
μ_T	Tumor cell potential	
μ_E	Extracellular Matrix Potential	
S_V	Viable Tumor Cell Source	
S_D	Dead/Necrotic Tumor Cell Source Term	
S_E	Extracellular Matrix Source Term	
S_{BnE}	Blood cell Source Term	
$\mathcal{W}$	Elastic Energy	
$\mathcal{E}_T$	Elastic stiffness of tumor component	
$\tilde{\mathcal{E}}_T^*$	Eigenstrain	
$\mathbb{T}$ & $\mathbb{T}^*$	Extracellular matrix stresses	(Ng & Frieboes, 2018)
L_1 & L_2	Lamé constants for cell components	(Ng & Frieboes, 2017)
ϵ_E	Interaction strength of the Extracellular Matrix	
ϵ_T	Interaction strength for Tumor Cells	
ϵ_e	Strain energy coefficient	
k_a	Motility of the solid phase	
p	Solid phase tumor cell Pressure	
γ_T	Tumor cell adhesion parameter	
γ_E	Extracellular matrix adhesion parameter	
n	Concentration of oxygen	
D_n	Diffusivity of oxygen in Tumor	
k_{n1}	Rate constants	(Ng & Frieboes, 2018)
k_{n2}		
tgf	Tumor growth factor concentration	(Ng & Frieboes, 2017)

D_{tgf}	Diffusivity of tumor growth factor in tumor	
λ_{tgf}	Tumor growth factor rate constant	
$\lambda_{de,tgf}$	Degradation rate constant for Tumor growth factor	
$\lambda_{U,tgf}$	Total uptake rate constant for tumor growth factor	
B_n^E	New blood vessels	
$\boldsymbol{J}_{BnE}$	Blood vessel diffusive flux	

In its current form, the Ng-Frieboes model can simulate a globular tumor, such as lung cancer or carcinoma along with surrounding tissue. With additional functionality and future work, this model could generate tumor volumes from The Cancer Genome Atlas (TCGA) project and predict treatment method effects.

D. Multigrid Model

The coupled nature of the Ng-Frieboes model and previous models has led to numerical solution methods (Frieboes et al., 2010; Ng & Frieboes, 2018). The numerical solution for the model stems from Multigrid based on previous work by Lowengrub and coworkers (Lowengrub et al., 2010). The algorithm is given in Ng and Frieboes (2018):

For each level $\ell = \ell_{min}\ to\ \ell_{max}$

If $\ell = \ell_{min}$

$\bar{\psi}_{\ell_{min}}^{t,0,v_0} = \text{SMOOTH}\left(v_0, \psi_{\ell_{min}}^{t,r=0}, L_\ell, R_\ell\right)$

Else

If $\ell < \ell_{global}$

r = r + 1

$\psi_\ell^{t,r} =$

$\text{ADAPTFAS}\left(\ell, \gamma_\ell, \tau_\ell, v_0, v_1, v_2, \psi_\ell^{t,r-1}, \psi_{\ell-1}^{t,r-1}, L_\ell, R_\ell\right)$

Else

Do

r = r + 1

$\psi_\ell^{t,r} =$

$\text{ADAPTFAS}\left(\ell, \gamma_\ell, \tau_\ell, v_0, v_1, v_2, \psi_\ell^{t,r-1}, \psi_{\ell-1}^{t,r-1}, L_\ell, R_\ell\right)$

While $\left(\left\|R_\ell - L_\ell\left(\psi_\ell^{t,r}\right)\right\| > \tau_\ell\right)$

End If

End If

If $\ell < \ell_{global}$

Find prolongate solution $\psi_{\ell+1}^{t,r-1} = \text{PROLONGATE}\left(\psi_\ell^{t,r-1}\right)$

Else If $\ell_{global} \leq \ell < \ell_{max}$

$F_\ell^{t,r-1}$ =FLAG($\psi_\ell^{t,r-1}$)

If $F_\ell^{t,r-1} \neq \emptyset$

Create block $B_{\ell+1} \subseteq \Omega_{\ell+1}$:

$B_{\ell+1}$

$= \text{BLOCKGEN}\left(F_\ell^{t,r-1}, \eta_{threshold}, threshold_{size}, \eta_{min}\right)$

Find prolongate solution $\psi_{\ell+1}^{t,r-1} = \text{PROLONGATE}\left(\psi_\ell^{t,r-1}\right)$

Else

Break

End If

End If

End For

Where ADAPTFAS, PROLONGATE, BLOCKGEN, and SMOOTH are defined in Ng and Frieboes 2018 and the parameters are defined in table II.

In each section, openMP improved algorithm performance by parallelizing operations performed on Ω_ℓ.

TABLE II

DEFINITIONS OF NG FRIEBOES ALGORITHM PARAMETERS

Parameter	Description	Term Definition
Ω_ℓ	Model domain at level ℓ	(Ng & Frieboes, 2018)
ℓ	Level index	
ℓ_{global}	Finest level that always spans Ω_ℓ	
γ_ℓ	Cycle Index for Level ℓ	
r	Multigrid iteration number	
t	Time step index	
ψ	Solution on level ℓ	
$\bar{\psi}^t$	Initial solution estimate for time step t	
L_ℓ	Left-hand side equation terms for level ℓ	
R_ℓ	Right-hand side equation terms for level ℓ	
τ_ℓ	Solution Tolerance for level ℓ	
v_0, v_1, v_2	Preset, arbitrary number of smoothing steps	
$\eta_{min}, \eta_{threshold}$	Minimum and threshold efficiencies, respectively	

II. PROPOSED TUMOR MODEL NUMERICAL SOLVER

A. openMP Shortcomings and Overall Contribution to the Model

As noted in the in Ng and Frieboes (2018), only openMP was used to parallelize the implemented framework. There are three main limitations that are imposed by parallelizing using openMP alone:

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 Cardinal Research Cluster (CRC). Results are shown in table VI. These findings are indicative of a memory transfer bottleneck. Hence, openMP-only implementations will not scale well locally for sufficiently large grid sizes.

2. openMP is a shared-memory architecture that runs on non-distributed systems, limiting performance gains to what a single independent computer can accomplish (i.e. single PC or workstation).

3. Many PCs will not possess enough RAM to hold larger tumor model spaces. For example, a 256^3 grid is expected to use over 12 GB of space; this is well out of reach of many PCs at the time of writing. Future grid sizes for application use

could reach or exceed 1024^3, putting total RAM usage upwards of 850 GB. Table III summarizes the expected RAM footprint for varying model sizes. Appendix 1 covers how these values were obtained.

TABLE III

MEMORY REQUIREMENTS FOR VARYING TUMOR MODEL SIZES

	Max 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 nodes on the finest level (GB)	3.3	25.5
RAM Required for single process on the finest level (GB)	13.6	107.6
Maximum spherical tumor diameter to simulate (mm)	5.1	10.2

To simulate tumors on physiologically realistic scales, the Ng-Frieboes model must have enough computational resources to function on at least a 512^3 sized domain and, according to table III, over 100 GB of RAM is required for such a task. Because many modern computers do not contain nearly this much RAM, a new solution generator is required.

This thesis describes a scalable framework designed to alleviate the shortcomings found with the openMP-based implementation of the Ng Frieboes model. MPI handles distribution of information across multiple processes, freeing the program from the RAM and processing constraints of a single system. On each system, Nvidia's GPU CUDA library allows for faster processing of model data. Thus, the model framework is a two-part model: an MPI-CUDA model.

Finally, other distributed computing frameworks will briefly be covered here. The type of simulation being done here, generally known as Big Compute, requires consistent communication between multiple data repositories. As such, HPC architectures configured for Big Data, in which repositories are assumed to contain independent pieces of data, are not designed for Big Compute Tasks. Additionally, these models run on Java whereas MPI is compatible with C thus giving MPI a small performance advantage (Byun et al., 2012; Taboada, Ramos, Expósito, Touriño, & Doallo, 2013). Therefore, Hadoop and Spark, specializing in Big Data, were not optimal for tumor simulation (Byun et al., 2012; Zaharia et al., 2016).

B. MPI DESIGN

In MPI, there are two classes of processes (i.e. nodes) in the program:

1. The administrative node (AdN). Its responsibilities include saving the model, ensuring synchronization of the model at certain points in execution, such as calculation of the residual, and designation of synching properties and node adjacency. There is only one node designated the AdN.
2. General Computation nodes (GCN). GCNs take up a non-overlapping cubic region in Ω. Each one can operate on more than one level as designated by the AdN at the start of the model's execution.

At the beginning of the model, the single AdN is designated. It then starts to define node boundaries:

1. Collect statistics on node characteristics. Determine the minimum amount of RAM possessed by a single node and by a graphics card.
2. Determine the maximum sized domain that each GCN can handle. To agree with the domain Ω, the cubic domain Ω_D has a side length 2^k where $k \leq \ell_0 + \ell_{index}$. The resulting size is the fundamental size for the node. A

corollary to the definition is that the coarsest level ℓ_{index} may define a domain Ω_0 that is larger than a single node.

3. The nodes are arranged sequentially with each node filling a single region of the model in a manner depicted by figure 1:

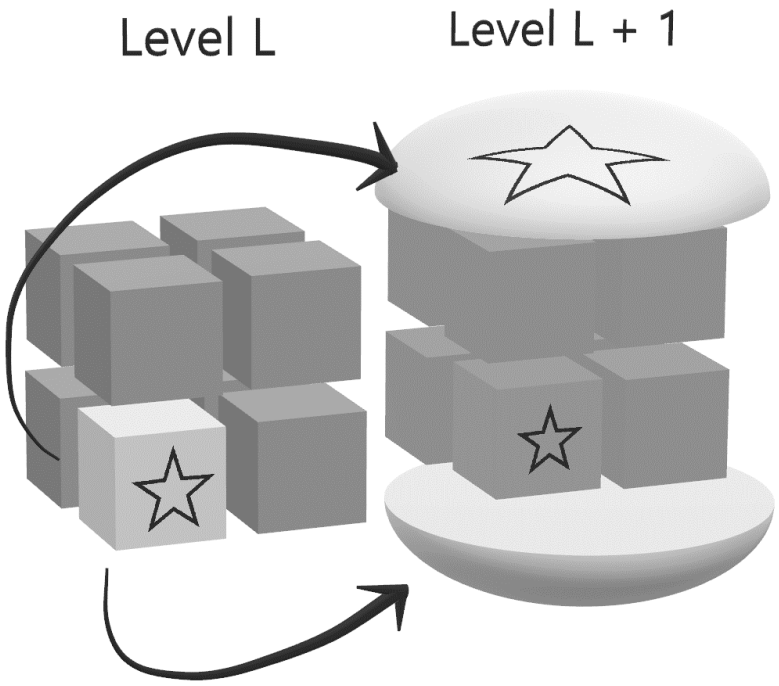

FIGURE 1 – NODAL DISTRIBUTION ON ARBITRARY LEVEL L

In figure 1, level L contains eight nodes, all of which are at the maximum capacity per node for a single level. Adapting a method of hierarchical node filling proposed by Reiter et al. 2013, on level L + 1 eight times the number of nodes will be required to fill the domain since memory occupation is maximized on level L. The light grey node is expanded on level L + 1, revealing seven new nodes. One-eighth of the

domain covered by the star node on level L is retained locally while the other 7 parts of the domain are sent to 7 other GCNs. Thus, the amount of work increases linearly with the number of levels, since nodes on each level after and including level L would have the same domain size (Reiter, Vogel, Heppner, Rupp, & Wittum, 2013). This also means that every node on a previous level must operate on the final level L_{max}. Overall, then, the total amount of nodes required is described in equation 10:

$$Nodes\ Required = 8^{n_0 - m_0 + L} \tag{10}$$

Where $n_0, m \in \mathbb{Z}, n_0 < m_0$, the coarsest level L_0 has 2^{m_0} points on a side, each node holds 2^{n_0} points per side per level with maximum RAM usage, and L is equal to the number of levels in the model. Thus, in figure 1 $L = 2\ \&\ n_0 = m - 1 \Rightarrow \#Nodes = 8^{2-1} = 8$ nodes. RAM usage is explored in greater detail in Appendix I. Because a portion of the computational work remains on every finer level after a node is first introduced, nodes are utilized to a greater degree over a non-hierarchical filling method with lowered node-to-node communication.

C. OVERALL ALGORITHM

The overall algorithm in the MPI-CUDA tumor model is identical to that of the Ng-Frieboes Model, save that the conditions for block generation have changed. In the old model, efficiency was defined as $\eta = \frac{\#Flagged\ Points\ in\ B_{\ell+1}}{Total\ \#Points\ in\ B_{\ell+1}}$ where the set of all flagged points $F_{\ell}^{t,r-1} \subseteq \Omega_{\ell+1}$. Also, the block $B_{\ell+1} \subseteq \Omega_{\ell+1}$. To prolongate to a new level, η had to be lower than a pre-defined cutoff efficiency. In the new model, the decision process is simplified to an all-or-nothing behavior: $F_{\ell}^{t,r-1} \neq \emptyset \Rightarrow B_{\ell+1} = \mathrm{P}_{\ell+1}^{\ell}(B_{\ell}) = \Omega_{\ell+1}$. By doing so, memory management is greatly simplified, since a level is either processed or ignored for a given time step. However, this decision also increases workload on levels where only a subset of Ω_{ℓ} requires smoothing.

Other than the key difference outlined above, the Multigrid algorithm remains identical to the Ng-Frieboes model outlined in Ng-Frieboes 2018. However, the flow of information during execution greatly differs from the Ng-Frieboes method. Figure 2 summarizes the process for any computation function X that is neither restriction nor prolongation.

FIGURE 2 – EXECUTION OF ARBITRARY FUNCTION X

D. FUNCTION X CALLED ON LEVEL ℓ

For node n operating over a subset of Ω_ℓ, denoted Ω_ℓ^n, GPUs on node n are selected in round-robin fashion to process Ω_ℓ^n. Ω_ℓ^n is subdivided into subdomains ω_j^ℓ that are sufficiently reduced to fit in GPU RAM. The subdomains have the following properties for m subdomains on level ℓ:

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

If a single GPU has enough RAM to hold the entire domain, then $m = 1$. To prevent GPUs from mixing old and new data, each function call is preceded with an unloading of processed function data and a reloading of new function data. Before running the next function, the nodes synchronize level ℓ data. Next, the GPUs receive relevant constant terms from the model including vasculature parameters and the dimensions of their respective domains. Finally, function X is called on all GPUs.

E. FUNCTION PROCESSING USING PARALLEL ARCHITECTURES

To improve CUDA performance, stencil formation was updated significantly. Because the original openMP architecture used host memory, no formal stencil variables were required. However, the CUDA architecture requires further optimization to minimize use of slower request to so-called global memory. The solution was to use a smaller programmer-controllable cache known as shared memory. The algorithm is laid out below:

Input: thread t positioned at point (i,j,k) on ω_j^{ℓ}

(i,j,k) is loaded into register memory on t from global memory.

Shared memory s is created for all threads in a thread block B_t.

(i,j,k) is copied into s.

Synchronize all threads on B.

For each adjacent point P:

If $P \in s$:

Copy P from shared memory to register memory on t.

Else

Copy P from global memory to register memory on t.

End If

End For

The above process is done on a per-block basis on the CUDA GPU. Each thread on the block runs the above function simultaneously, thus the block B_t becomes a repository for point data for all threads in the block. Threads within each block are organized geometrically on Ω_ℓ^n in a cube. If a thread looks up a given point value (i,j,k) and it resides in shared memory, the thread will avoid a more time-consuming global memory retrieval. Since CUDA does not permit information exchange between shared memory blocks, if a given block does not contain the information requested by t, t must defer to global memory.

Stencils in this model do not exceed a 3x3x3, thus most stencils formed by threads in the cubical block B_t will avoid duplicating global memory searches. The only drawback to this approach is that in cases where GPU functions require more than one stencil simultaneously register pressure can occur, a phenomenon where insufficient registers exist to handle the number of variables requested. In such a scenario, the GPU defers resources to a repository of slower global memory denoted as local memory. After using a Nvidia profiler, it was confirmed that registers were not overloaded for almost all GPU-based functions. The effect of removing any other register pressure has not been tested and is a subject for future study.

When establishing GCN communication, a variety of methods were used:

1. A 2x2x2 node domain was constructed with each node corresponding to one octant of the nodal lattice. Each node communicates with 7 other adjacent nodes.
2. Each node has a specific variable index set to the MPI rank of the node. This value, by nature of the MPI initialization process, is unique to each node.
3. At the end of the syncing process, the number of ghost points with a given MPI rank is determined. That value, given a correct syncing operation, corresponds to the number of points that are adjacent to each other in the tumor domain.

When syncing data across GCNs, there are three operating directions to consider relative to the node GCN in question. These directions can be represented with three separate vectors:

1. A unit syncing vector, $\vec{S}$.
2. A nodal vector for the data $\vec{N}$.
3. The unit storage vector for data, $\vec{D}$.

Because nodes are arranged in a Cartesian grid, rules are easily established for points not on the boundary of the tumor domain Ω_ℓ. First, relative to each node, in graph theory terms, each GCN forms a star graph S_{26} with its neighbors. Any MPI send and receive operation is a one-step process, in which any link (u, v_m) for $m \in \{1,2, \dots ,26\}$ must be traversed. For maximum performance, perfect matching is desirable, meaning that on

level ℓ, half of the GCNs are sending data and half of the GCNs are receiving data during the synchronization command. Finally, any subgraph created by tracing the synchronization path comprising consecutive syncing vectors on the overall graph G_S must be acyclic to prevent the program from halting.

Ensuring synchronization between points is a multistep process accomplished through three steps:

1. Establishment of a residing map on each GCN that informs it of adjacent point data.
2. Development of the synchronization matrix. Every value in a 3x3x3 nodal grid, representing G_S, is filled with a value corresponding to how synchronization should take place, including send or receive and sides to select in border operations.
3. Derivation of the storage vector $\vec{D}$ for a given syncing vector $\vec{S}$.

Every value in the 3x3x3 nodal grid is cycled through in a preset order. With the center node of the nodal grid as the origin, any point selected, excluding the origin, will form a unit vector $\vec{S}$ starting from the origin and moving away towards the selected point. MPI does not resume execution until the package is successfully sent and received. Thus, for any MPI send/receive event, all subgraphs must be acyclic. Put another way, the vector

sum formed by consecutive vectors $\vec{S}$ cannot equal the zero vector. Although sufficient but not necessary, a guideline for ensuring that no cycles form is by looking at the vector field of $\vec{S}$. If $\nabla \times \vec{S} = 0$ for all vectors $\vec{S}$ then the resulting syncing cycle will not form a cycle. By cycling through all possible $\vec{S}$ values in a known order on all GCNs and with the implicit synchronization required for a successful MPI send/receive operation, the summation of any number of $\vec{S}$ vectors will be non-zero. In addition, the curl of the resulting vector space is also zero. Figure 3 describes one possible $\vec{S}$ field that qualifies as a field. Each arrow in the figure represents a pair of nodes, one sending and the other receiving.

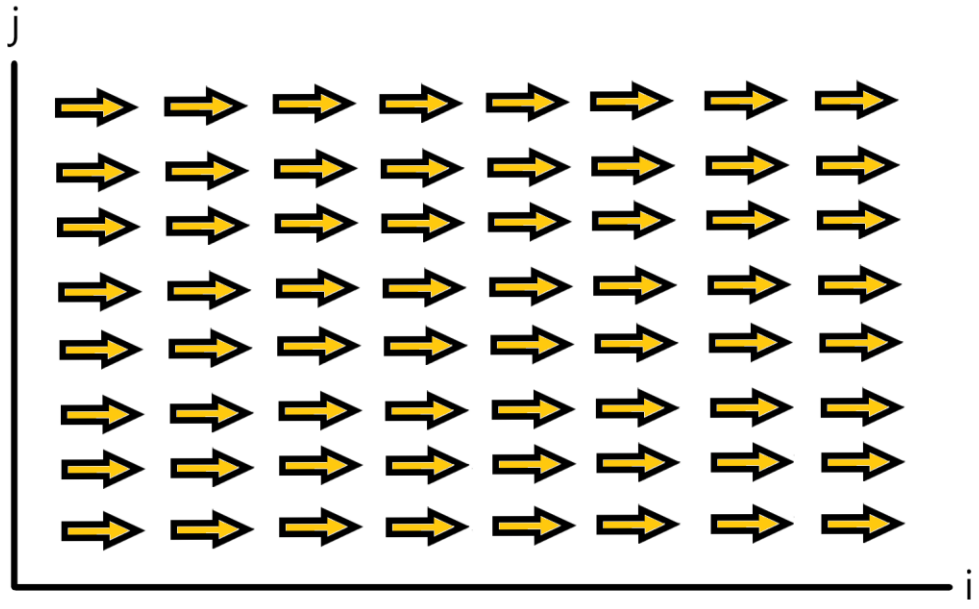

FIGURE 3 – A ZERO CURL VECTOR FIELD CANDIDATE FOR THE SYNC VECTOR

For the MPI model, a syncing vector field with zero curl was found for each syncing direction, thus ensuring no unresolvable communication errors could form during function execution.

For the node vector, each node creates a stencil containing the node IDs of adjacent nodes. With a given sync vector and data stored locally on each GCN, a node vector can be correctly defined. Finally, the node can use the sync vector to derive the node vector. Figure 4 depicts the vectors for two nodes in communication at the border.

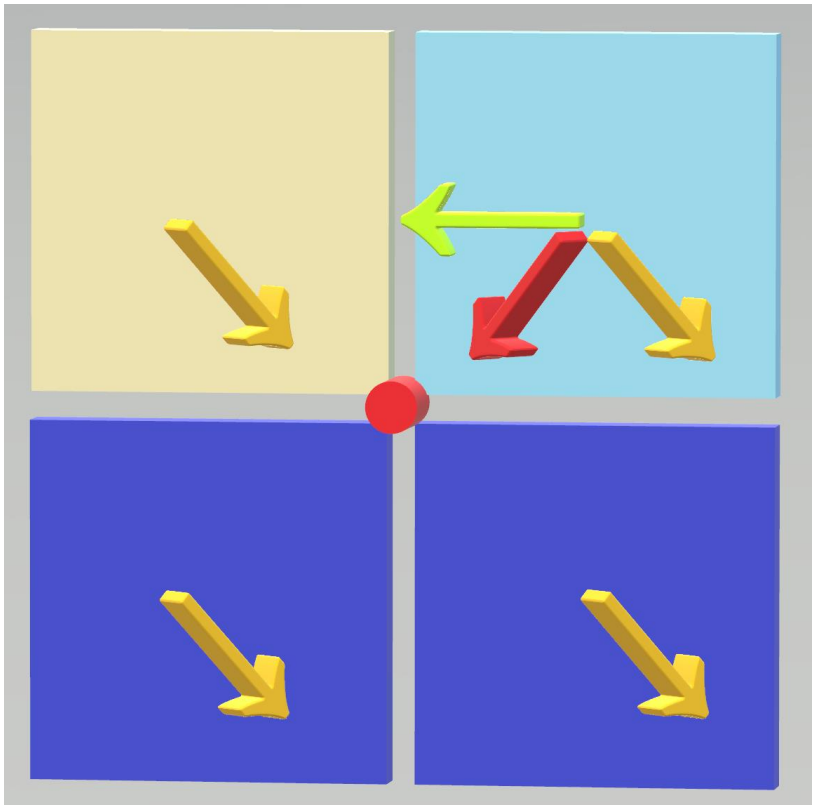

FIGURE 4 – MPI SYNCING VECTORS AT DOMAIN BORDERS

In figure 4, the red point is a corner points required on all four nodes for computation. The sync vectors $\vec{S}$ (orange) are parallel thus ensuring there will be no communication hanging. The tan node, represented at the border as a tan square, is sending the red point to the light blue node. Thus, the node vector $\vec{N}$ (green) is pointing west towards the tan sending node. The point data obtained by the tan node will be placed in a corner of the light blue node's domain, thus the storage vector $\vec{D}$ (red) points towards red corner point.

F. RESTRICTION & PROLONGATION/ERROR CORRECTION

Restriction and prolongation/error correction require additional steps, since multiple levels and, therefore, multiple node groups, must interact. Restriction in the MPI-CUDA model is a three-step process:

1. For nodes on level ℓ Restrict approximate numerical solution u_ℓ using the restriction function, denoted as Γ

$$\Gamma(u_\ell) = u_{\ell-1} \tag{10}$$

2. Sync $u_{\ell-1}$ values across all nodes on level u_ℓ using the system laid out in part A.
3. Restrict approximate right-hand-side (RHS) solution f_ℓ using Γ

$$f_{\ell-1} = \Gamma(f_\ell) = \Gamma(\mathrm{f}_\ell - N_\ell[u_\ell]) + N_{\ell-1}[\Gamma(u_\ell)] \tag{11}$$

Where the analytical solution takes the form $N_\ell[v_\ell] = g_\ell$ after complete convergence of $u_\ell \to v_\ell$ and $f_\ell \to g_\ell$.

4. Sync $f_{\ell-1}$ across all nodes on level u_ℓ using the system laid out in part A.
5. Send $u_{\ell-1}$ and $f_{\ell-1}$ to nodes containing level $\ell - 1$ data.

For the highest performance gain, minimizing the number of transfers between external nodal communication and internal GPU communication is ideal. Thus, in step 1, $u_{\ell-1}$ and $f_{\ell-1}$ are produced on nodes on ℓ. After syncing $u_{\ell-1}$ across all nodes on ℓ, $f_{\ell-1}$ is computed. After unloading both $u_{\ell-1}$ and $f_{\ell-1}$, the solution is sent to level $\ell - 1$ by collapsing a 2^3 worth of $u_{\ell-1}$ & $f_{\ell-1}$ node data into a single node on $\ell - 1$. Sequential collapse along each axis distributes the work across all nodes on level ℓ. Because of standard coursing depicted in figure 1, every restriction operation results in seven nodes sending restricted information to a single node.

For error correction, from level ℓ to $\ell + 1$ the process requires little adaption from the Ng-Frieboes method:

1. Transfer level $u_{\ell-1}$ to nodes on u_ℓ.
2. Determine error on each point on level ℓ.

$$e_{\ell-1} = u_{\ell-1} - \Gamma(\mathrm{u}_\ell) \tag{12}$$

3. Sync error e_ℓ across all nodes on u_ℓ.

4. Apply correction to obtain corrected solution v_ℓ.

$$v_\ell = u_\ell + P(e_\ell)$$

III. MATERIALS AND METHODS

A. OVERALL METHODS

Model accuracy was ensured by comparing model input to the Ng-Frieboes openMP numerical solutions. However, the flux of metabolites was changed from the original openMP code because its logic was found to inaccurately represent the underlying model. As such, old and new mathematical model inputs do not output equivalent model values. However, there are two reasons why this discrepancy does not dissuade using the openMP model as a guide for verifying the new model's physiological relevance:

1. Model consistency. Due to the effects of floating-point arithmetic evaluation on results, the compilation process, such as optimization of debug code, can affect the final values outputted by the model (Collingbourne, Cadar, & Kelly, 2014). This was observed on early CUDA builds when comparing CRC solutions compiled using the Linux-based g++ compiler verses Windows compiled code. Consecutive runs with consistent results ensured that solution variance was not due to race conditions.

2. MPI-CUDA solution error reached the tolerance in fewer solver iterations than the older model, supporting the conclusion that the original

openMP code incorrectly calculated flux values and, therefore, slowed down the convergence process. Because metabolite variables are affected by the flux function, the solution difference between the two models is more significant than the MPI-CUDA framework, as documented in table VII. The effects of this change are explored in greater detail in the discussion section.

Once the model had been debugged and tested, consistency was ensured from the original completed CUDA-only code to the present MPI-CUDA build by printing out all volume fractions, pressures, metabolites, growth factors, and other miscellaneous tumor variable data from the first-time step. Those values were compared to archived output from the older build using a SHA-256 hash. Matching hashes implied that the integrity of the solving process was not impacted by the code.

All timing results were obtained using time.h clock statements and used the test scenario described in Ng and Frieboes (2017). The computer used for testing has an AMD 2990WX processor with one Titan RTX GPU. A CUDA only simulation is emulated using two MPI instances of the program: one AdN and one GCN. Because of the minimal communication between these processes and only the GCN operates on the domain, the two MPI instance program will behave similarly to a single process. MPI-CUDA was performed using nine processes, one AdN and eight GCNs. One GCN operated on all coarser levels as a single node, then MPI was used to divide the finest

level into octants, with each GCN operating exclusively in one of eight octants. For both CUDA and MPI-CUDA, the program was tested using a coarsest grid of 8^3. Standard coursing was used to reach one of two different finest grids: 128^3 and 256^3. Two iterations of the CUDA and CUDA-MPI models were performed to ensure consistent model results. All run data are given in table VIII in appendix II. Finally, execution times were also obtained from the CRC for the openMP model with varying thread counts.

B. EXAMPLE TESTING METHODOLOGY

When testing the prolongation routine outlined in part 2.C, four tests were performed on the MPI process that transferred $u_{\ell-1}$ to eight nodes on level ℓ, with each node sitting in its own octant:

1. At each point (i, j, k) on level ℓ, manually set each point to a value equal to the unique MPI ID for the expected destination node. After $u_{\ell-1}$ is transferred, each node reports the total number of each possible node ID it finds on its subdomain on level ℓ. For a passed test, a node on level ℓ will only find its MPI ID on its subdomain.

2. Because each node's domain is cubic, there can be at most 8 destination nodes for a single point of data on level $\ell - 1$. The second test sets one of the eight possible corner points and ensures that each node (a) receives the point and (b) places it in the correct position in memory. Each node reports if the point value is found in its subdomain and, if so, where that value resides. To pass, each node finds the value only once and displays the correct location.
3. The first memory value and final memory value are marked on level $\ell - 1$. For a passed test, only the first and eighth node find the first and final value, respectively on their subdomains.
4. A few random points were selected and marked on level $\ell - 1$ by the programmer before function execution to test different cases during the syncing process. Each node reports if it received the marked point and, if so, where it resides on the grid. To pass, node output will match the predictions made by the programmer.

IV. RESULTS

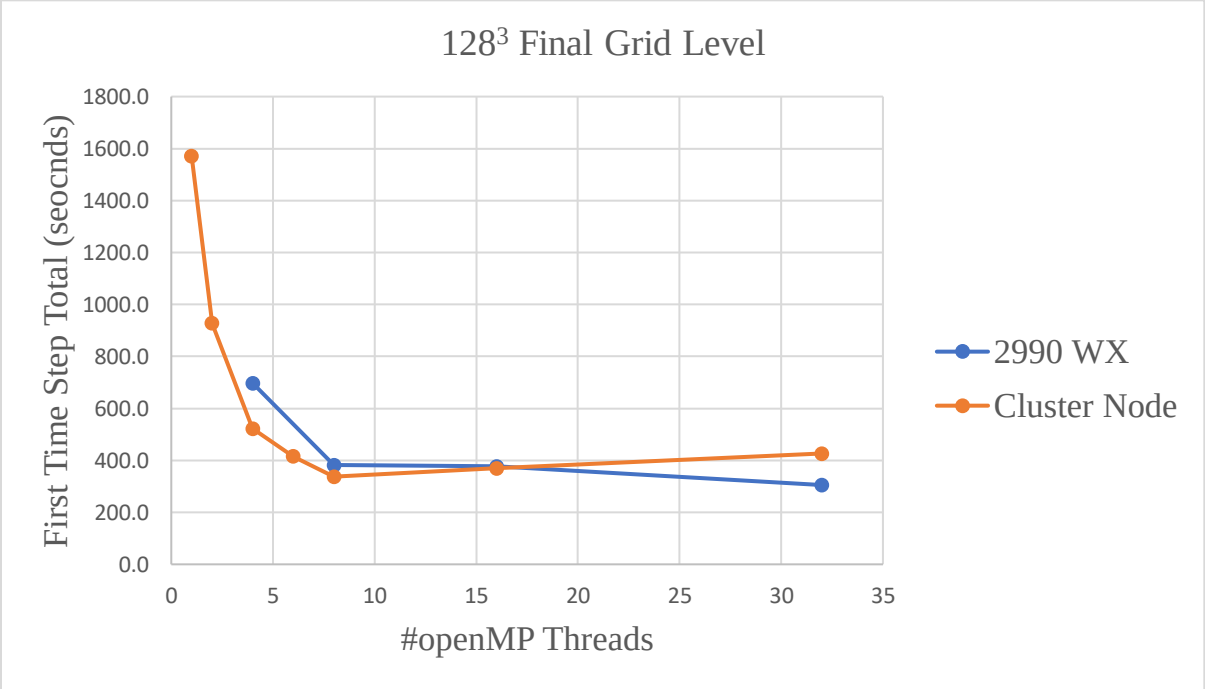

FIGURE 5 – OPEN-MP VS MPI-CUDA PROCESSES AND PERFORMANCE

TABLE IV

TIMING RESULTS FOR OPEN-MP MODEL ON 2990WX

Finest Interior Level Size (#points)	Number of Threads Enabled	Data Export Time (seconds)	Total Execution Time (seconds)	Total time for single time step (seconds)
128^3	4	45.7	651.6	697.3
	8	46.7	335.7	382.0
	16	49.1	327.2	376.3
	32	45.4	259.7	305.1

TABLE V

TIMING RESULTS FOR FIRST TIME STEP USING OPEN-MP

	# Threads						
Computer Type	1	2	4	6	8	16	32
Desktop Time (seconds)	-	-	697.3	-	382.0	376.3	305.1
CRC Cluster Node (seconds)	1571.7	927.1	521.8	414.7	337.4	369.0	426.3

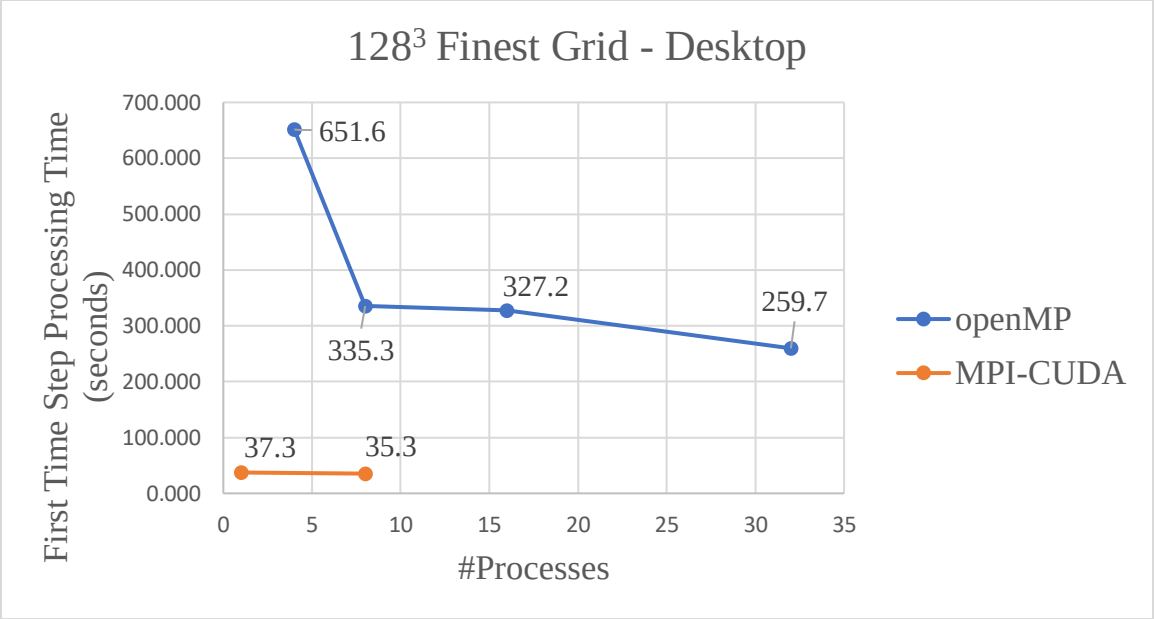

FIGURE 6 – OPEN-MP PARALLELIZATION CLUSTER VS DESKTOP

TABLE VI

TIMING RESULTS FOR MPI-CUDA MODEL

			AMD 2990WX		
			Total Execution Time for First Time Step (seconds)		
Finest Level Size (#points)	Tumor Domain Side Length (mm)	Framework Type	Trial #1	Trial #2	Average
128	2.56	CUDA	37.1	37.6	37.3
256	5.12		295.4	291.1	293.2
128	2.56	MPI-CUDA	35.2	35.4	35.3
256	5.12		257.4	259.2	258.3

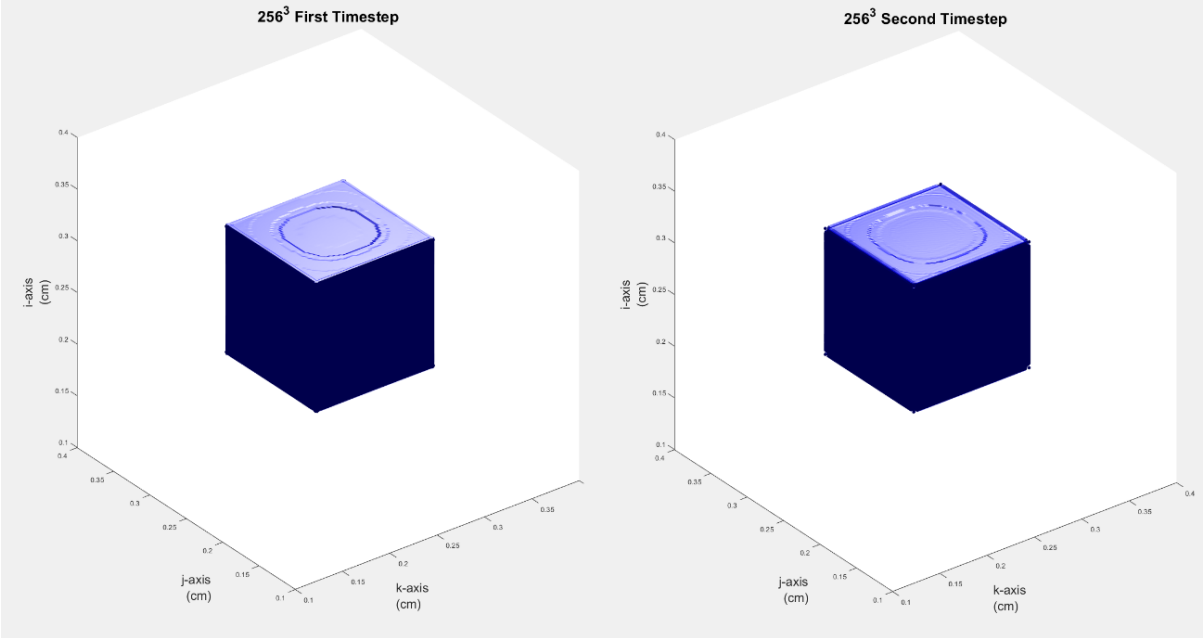

FIGURE 7 – INITIAL TUMOR VOLUME IN 5.1 MILLIMETER DOMAIN AT THE END OF TIME STEP 1 & 2

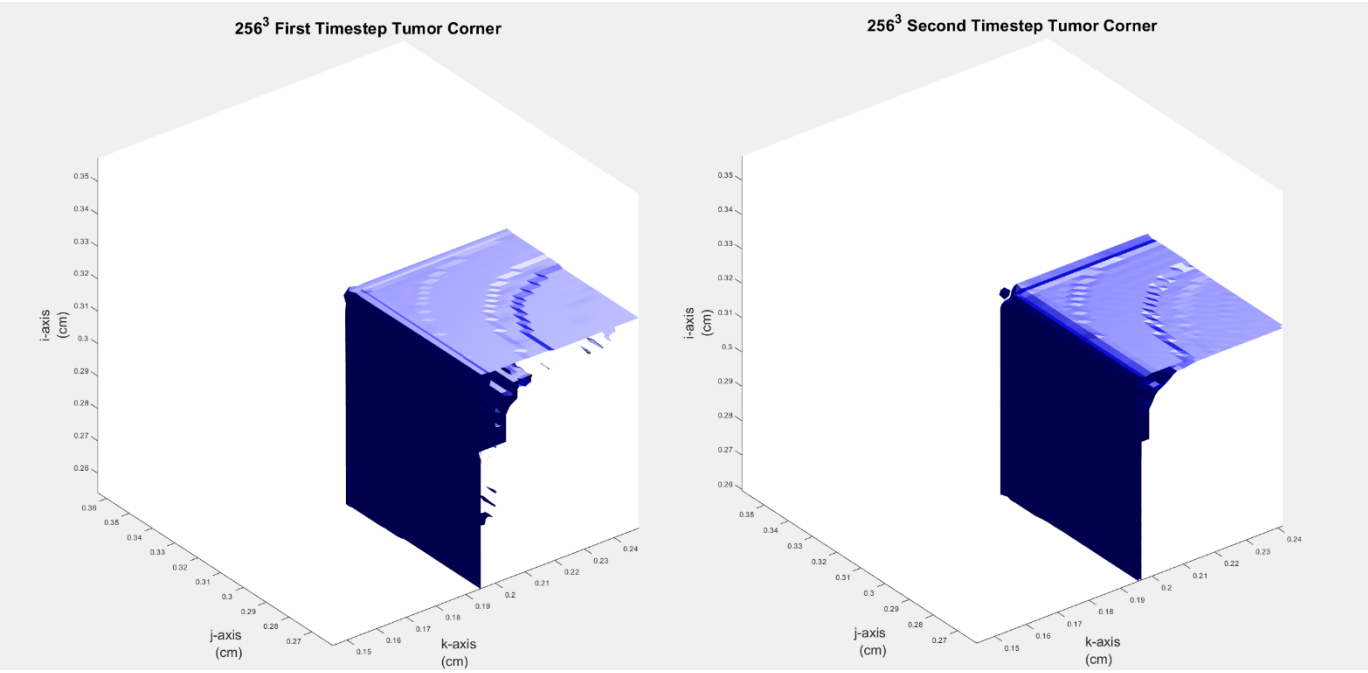

FIGURE 8 – TIME STEP 2 CORNER

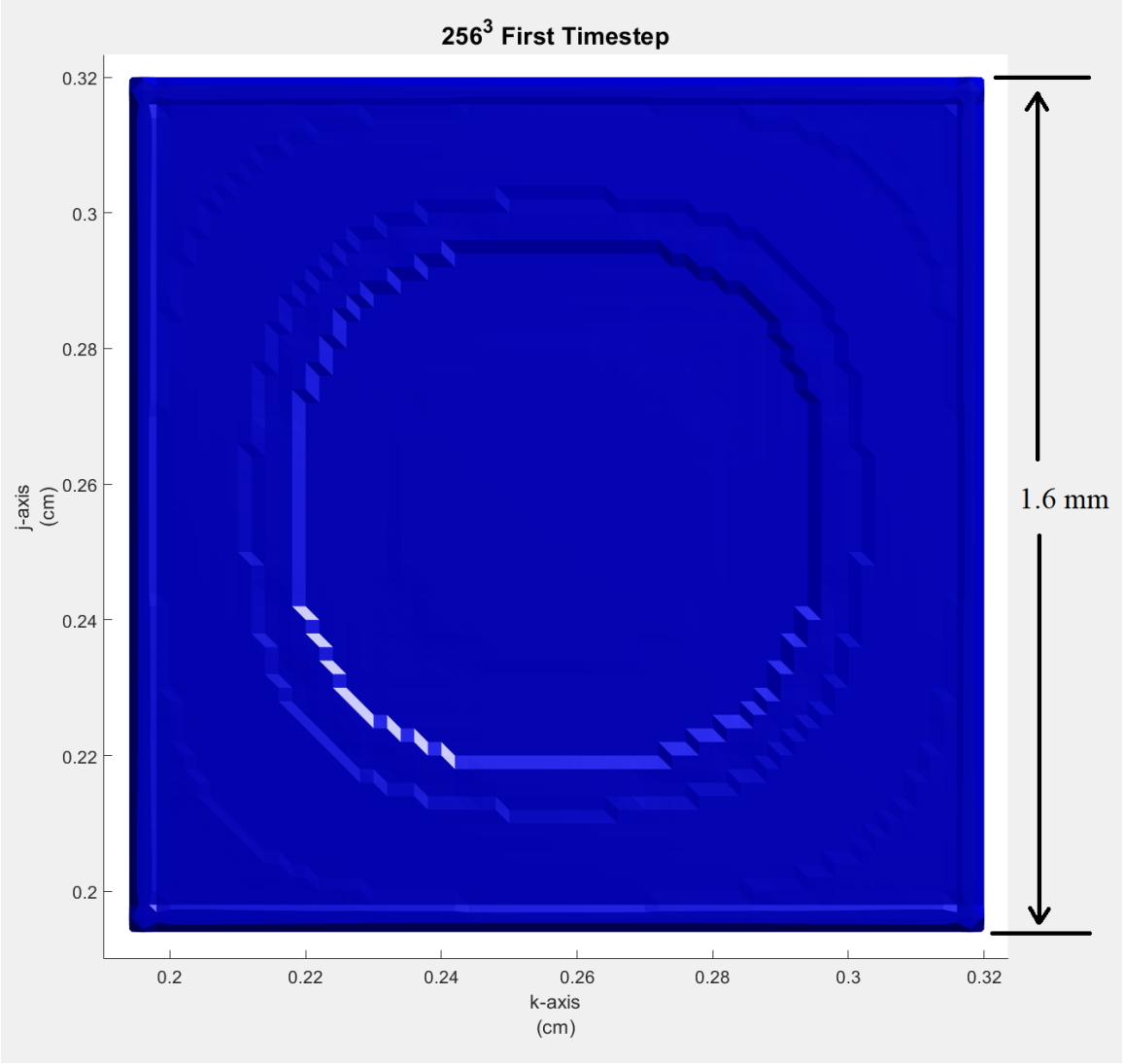

FIGURE 9 – FIRST TIME STEP CUBIC TUMOR DIMENSIONS

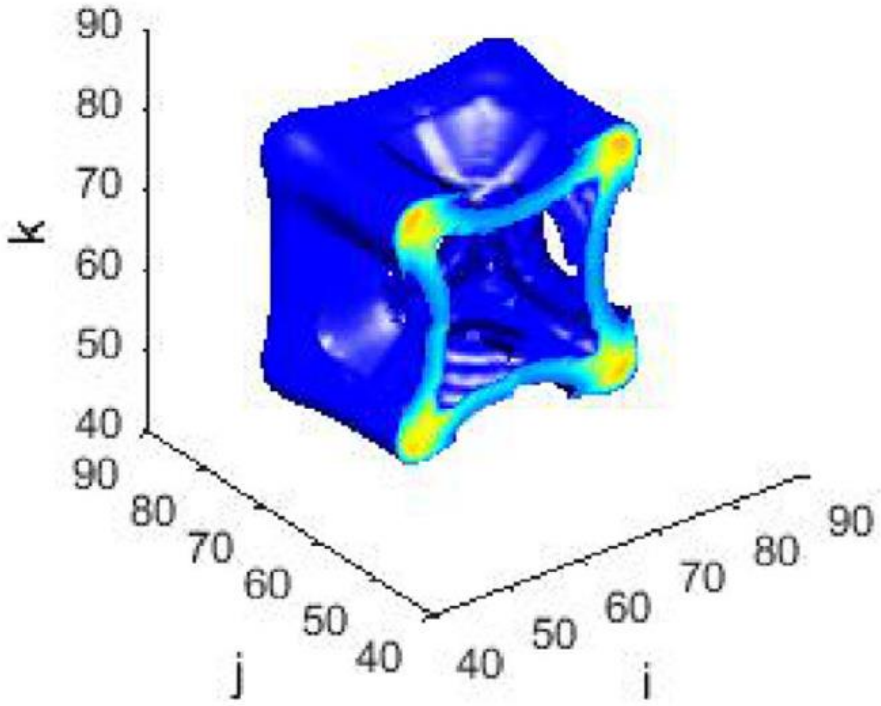

FIGURE 10 – OPENMP OUTPUT FROM NG-FRIEBOES 2017 AFTER 500 TIME STEPS

TABLE VII

OPENMP VS. CUDA BICARBONATE AND PRESSURE SOLUTION COMPARISON

	Level 2 – Same cycle volumes		Level 4 – Different cycle volumes	
openMP vs. CUDA Property	P-Alpha	Bicarbonate	P-Alpha	Bicarbonate
Absolute Mean Difference	3.0E-04	4.8E-03	3.5E-03	1.0E-02
Variance of Model Difference	3.7E-07	3.3E-05	1.5E-05	1.1E-04
Relative Error Difference (%)	-0.6%	-9.6%	-15.4%	-30.3%
Relative Error Standard Deviation (%)	0.2%	1.0%	21.5%	18.6%

V. DISCUSSION

Figure 5 reveals the performance differences for a 2990WX vs a single node of the CRC. While both systems possess equal core counts, there is a decrease in performance on the CRC node with greater than eight CPU cores in parallel use. Interestingly, the 2990WX shows improved performance, even to 32 cores. This difference in results is believed to be caused by memory bandwidth limitations on the CRC node. In figure 6 we see that increasing the number of cores using open-MP gave diminished gains for program instances with more than 8 threads, a find consistent with model performance on the CRC. However, performance continued to improve as more cores were added in the case of the 2990WX. With desktop performance of open-MP being the lowest recorded time, it is used as the baseline for testing the MPI-CUDA implementation.

For the CUDA only run on 128^3, a sizeable performance increase is observed, from 4.31 minutes on openMP to 0.622 minutes. The CUDA version, therefore, is 6.96x faster than the openMP version it replaces on the same grid size. When increased to 256^3 on a side, the time to process the first-time step increased 7.86x, indicating that time is scaling linearly with problem size. However, it will be difficult for many current GPUs to exceed the RAM and computing of a Titan RTX solution, making these numbers a best-case scenario.

MPI-CUDA appears more promising than either option. While its 128^3 time was akin to the CUDA model, its processing time increased 7.37x for an 8x increase in problem size. The most likely explanation for the performance increase comes from optimized memory movement from CPU to GPU under MPI loads. For an equivalently sized dataset, the MPI processes can parallelize memory manipulation on the CPU, thus increasing memory bandwidth usage. Another implication of this result is that MPI communication operations on the AMD 2990WX were more than offset by the increased memory bandwidth usage. With multiple independent nodes of equivalent power to the test desktop, it is conceivable that larger tumors than this could be simulated.

Because of changes in flux terms made in the MPI-CUDA code, the MPI-CUDA code converges in one Multigrid cycle, whereas the openMP model requires 12 iterations. It is possible that convergence would also be improved in openMP implementations were applied. Because the openMP code remains unpatched, this scenario can be simulated by timing the first iteration of the Multigrid solver. The results are given in table IX in Appendix II and are comparable to the results obtained for CUDA and MPI-CUDA implementations. However, because of adaptive grid technologies built into the openMP model version, only a small portion of the final 128^3 grid is solved over, thus openMP performs computationally less work than the CUDA model but taking more time to do so. Finally, because both the cluster node and the AMD 2990WX possess 32 cores, well

above most current CPUs, it will be difficult to increase openMP performance beyond the numbers presented in Table V and VI, whereas the MPI-CUDA will potentially scale to many more machines for similar results.

Figure 7 depicts the initial cubic tumor at the initial time step and the second time step. While the domain could contain a tumor with a radius of 5 millimeters, the model begins with a 1.6 mm tumor situated in the center of the domain. The tumor is more closely visualized in figure 9. While this simulated tumor is not realistic in shape, it is optimized to verify model integrity, as the observations in Ng and Frieboes 2017 demonstrate. Like the openMP model they documented, the cubic tumor evolved by smoothing the corners of the cube into expanding bulbs. Although extremely early in the tumor model's execution, evolution of corner behavior is visible as early as the second time step in the CUDA model, as shown in figure 8. The phenomenon is more easily seen in figure 10 taken from Ng and Frieboes 2017, after the tumor model had executed for 500 timesteps. Future CUDA modeling with more timesteps will give greater insight into how these corners evolve over time and ensure consistency between the two frameworks.

Finally, the two methods solution differences are categorized in table VII. Because the final solution for a given timestep resides on level ℓ_{max}, the openMP framework must prolongate all data from coarser levels at the end of a timestep. Since the CUDA model processes all data points on ℓ_{max}, solution accuracy is increased on

regions that the openMP model would exclude from its $B_{\ell+1} \subseteq \Omega_{\ell+1}$. The openMP framework would decrease local error relative to the MPI-CUDA framework within its $B_{\ell+1}$ due to repeated smoothing efforts, although erroneous flux terms may have increased the number of smoothing cycles required to reach convergence. Nevertheless, in the case of MPI-CUDA, the inclusion of more low-error points decrease the normalized mean error, thereby reaching the tolerance on level $\ell + 1$ in fewer iterations than the openMP counterpart. In table VII, both models process the entirety of level 2. In the case of level 4, openMP only processes a subset of the domain Ω_4, leading to an increase in the magnitude of the relative solution difference.

VI. FUTURE WORK

While the parallelization performed on the tumor model is significant, more development is required before being clinically relevant. First, this node structure has no fault tolerance. That is, if a single GCN were to fail to respond, the program, as it stands, would exit without completing the model. Possible solutions include redundant data on different nodes and time-step backups. Many of these features have made their way onto Big Data cluster libraries, such as Hadoop or Apache Spark, with recent advancements being set in stencil processing (Jie et al., 2014; Yuan et al., 2016). Thus, a future implementation may draw from one of these cluster libraries. Second, while the focus of this thesis has been model performance, ensuring solution stability over multiple time steps remains a development goal.

In terms of program readability, significant improvement could be made with memory management on the GPU. In order to be usable on the CRC, the program was built to run on a minimum compute capability of 2.0 (Nvidia). Thus, many CUDA advents were passed over in the program's creation, such as a unified memory architecture. Dropping support for older CUDA standards could streamline programming. General performance improvements for memory transfer could also be implemented. With the addition of increased Multigrid technologies such as adaptive grid meshes would reduce computational workload and further increase model performance. For some

problem sizes, a CUDA-MPI framework may not be optimal due to overhead of passing data to GPUs for processing. Indeed, an openMP/MPI framework has outperformed CUDA-MPI tasks when testing a smaller mathematical model with greater efficiency (Lončar et al., 2016). Future evaluation of openMP-MPI vs. CUDA-MPI may lead to further optimizations of the Ng-Frieboes model for mixed grid sizes.

Currently, the model inputs a hard-coded scenario ideal for testing, however, a real tumor is globular and would require input imaging data. For testing purposes, possible input could come from The Cancer Imaging Archive (TCIA) created by TCGA. Using patient imaging data, real tumor evolution could be simulated to compare to known samples, thereby validating these equations' versatility and physiological accuracy. Because of the adaptability of the Multigrid method, the MPI-CUDA framework could be modified to run mathematical models on more general tissue ailments, such as drug delivery for microbial infection in certain organs. The capability to model sufficiently large tissues would permit accurate simulations of tissue-scale proportions.

From the standpoint of the model, additional equations and/or terms will be required to describe present and future treatment methods. For example, immunotherapy's reliance on T-cells restricts its efficacy to cancers with the correct antigens on their cellular membrane; future mathematical modeling would need to account for this phenomenon (Hall, 2016).

Finally, the burgeoning field of -omics data analysis, in which tissues are categorized using tissue profiling, provides a unique opportunity for model validation and application. For example in the case of transcriptomics, from a single instant in time, the change in gene expression in time, denoted as “pseudotime,” can be generated by simultaneously sampling different cells in different stages of development from the same tissue source to reconstruct the tissue’s overall gene expression (Reid & Wernisch, 2016). Elucidating drug effects using this method could spur the creation of mathematical models that accurately describe cellular models and aggregate behavior in cell cultures and, consequently, increase model realism. With sufficient accuracy, future mathematical models with increased predictive power may even return to benefit the underlying biological analyses by informing molecular expression research.

APPENDIX I

A. DERIVATION OF MODEL MEMORY FOOTPRINT

If the length between points on the Multigrid model is 20 µm per point on the finest level, then for a domain of 2 cm:

$$\#points\ per\ side = \frac{length}{\frac{length}{point}} = \frac{2 \cdot 10^{-2}}{20 \cdot 10^{-6}} = \frac{1}{10} \cdot 10^{4}$$

$$= 1000 \text{ points per side}$$

Rounding up to the nearest power of two, we obtain 1024 points per side. Using an approach where each process resides on a single domain (i.e. no process operates on more than one level), if a process can run 2^n points on a side and the domain contains 2^m points on a side (where $n, m \in \mathbb{Z}\ \&\ n \leq m$), then the number of processes will be:

$$\#Processes\ Total = P = \frac{Volume\ of\ Domain}{Volume\ Per\ Process} = \frac{(2^m)^3}{(2^n)^3} = 8^{m-n} \qquad (A.1)$$

For the above example:

$$P = 8^{10-7} = 8^3 = 512 \text{ Processes}$$

If each computer can perform 256 points per side rather than 128 in the previous example, then:

$$P = 8^{10-8} = 8^2 = 64 \text{ Processes}$$

For this example, we will use 4096 points on a side:

$$4096 \cdot 20 \cdot 10^{-6} \cdot 10^2 = 8192 \cdot 10^{-3} = 8.192 \text{ cm}$$

Thus, we could comfortably simulate an eight cm tumor with a 4096^3 simulation. Also, we consider how many independent process (i.e. 1 process per node) would be required for a 256^3 computation volume per nodes:

$$P = 8^{12-8} = 8^4 = 4096 \text{ nodes}$$

If we want there to be L levels total along with the finest level, then the following formula can be used to determine how much memory will be required:

$$Memory\ Needed = M = (Memory\ Per\ Point) \cdot (\#Points) \qquad \text{(A.2)}$$

$$= k \cdot \left(\sum_{j=1}^{L} \left(2^{c-j+1}\right)^3 \right) = k \cdot \left(\sum_{j=1}^{L} 8^{c-j+1} \right)$$

Where the number of points per side on finest level is 2^c, $c - L + 1 > 1 \Rightarrow L < c$, and k is the memory per point. For this model, $k = 692\ \frac{Bytes}{Point}$. To be viable computationally, the model should run on at least an 8^3 grid, thus $L < c - 1$.

Assuming five levels ($L = 5$) are required and the finest level has 4096 points on a side (thus $c = 12$), then

$$M = 692 \cdot \left(\sum_{j=1}^{5} 8^{12-j+1} \right) = 54.3\ \text{TB}$$

Border points can be added easily by adding two to the side length at each level:

$$M_T = 692 \cdot \sum_{j=1}^{L} \left(2^{c-j+1} + 2\right)^3 \tag{A.3}$$

$$L = 5, c = 12 \Rightarrow M_T = 692 \cdot \sum_{j=1}^{L} \left(2^{12-j+1} + 2\right)^3 = 54.4 \text{ TB}$$

Clearly, by the ratio test, the effect of border points diminishes as $c \to \infty$, thus for subsequent calculations, they will be assumed negligible.

According to the naïve process filling approach, where processes cannot span more than one level:

$$P = \sum_{j=1}^{L} p_j \text{ where } p_j = \begin{cases} 1, j \geq c - n + 1 \\ 8^{c-n-j}, j < c - n + 1 \end{cases} \tag{A.3}$$

When each node occupies 256 points per process, $n = 8$. Thus, for L = 5 and c = 12:

$$p_j = 8^{12-8-j+1} = 8^{5-j} \tag{A.4}$$

$$\Rightarrow P = 8^4 + 8^3 + 8^2 + 8 + 1 = 4681 \text{ Nodes}$$

Under this system, the memory use per process is $\frac{54.34}{4681} = 11.6$ GB, not including boundary/ghost points on each node.

The MPI setup uses an approach that attempts to remove some of the memory transfers and reduce idling by having nodes span more than one level. Overall, this leads to the following formula:

$$\#\text{Total Processes Total} = P_{Total} = 8^{n_0 - m_0} \cdot 8^L = 8^{n_0 - m_0 + L}, m_0 > n_0 \quad \text{(A.5)}$$

Where each node can process a cubic region containing 8^{n_0} points, the coarsest cubic domain contains 8^{m_0} points, and the total number of levels is L. In cases where the node can process a larger domain than the coarsest level, one node will handle all domains from L_0 to L_t where L_t is the #levels where the domain volume is less than or equal to than the node's processing volume (i.e. $2^m \leq 2^n$). In that case, formula A.5 can be adapted to a more general form:

$$\#\text{Total Processes Total} = 8^{n_0 - m_0 + L - L_t} \quad \text{(A.6)}$$

For the preceding example $L_t = 1, n_0 = 8, m_0 = 8$, and $L = 5$, thus:

$$8^{8-8+5-1} = 8^4 = 4096 \text{ nodes}$$

The maximum amount of RAM per process will be the first node allocated, since it will span all five levels, starting at 256^3 and ending at 4096^3, using a constant 258^3 points per level. In this scenario, the total RAM usage would be equal to:

$$692 \cdot (2^8 + 2)^3 \cdot 5 = 59.4 \text{ GB}$$

In this design the number of nodes is constant with a constant RAM requirement per node. Overall efficiency also increases relative to naïve grid filling, since multigrid runs on a single level at any given time, meaning nodes that span a single level will idle while computations are performed elsewhere.

Because one node will occupy all the levels whose side length is less than m and the computational work decreases exponentially with decrementing levels, there is little reason to use fewer levels.

As an example, we can calculate how much RAM is required for a single node existing on 5 levels with the coarsest level being 8 points on a side and the finest level being 128 on a side:

$$692 \cdot \left(\sum_{j=1}^{5} \left(2^{7-j+1} + 2 \right)^3 \right) = 1.8\ GB$$

In practice, the MPI framework requires more RAM than this amount due to temporary storage between certain computation steps, but that amount is not significant for this example. Because many computers used in either personal or cluster settings possess more than eight GB of RAM at the time of writing, there is no reason to remove coarser levels from a RAM usage perspective. For instance, removing the two coarsest level barely decreases the RAM requirement for a single node model:

$$692 \cdot \left(\sum_{j=1}^{3} \left(2^{7-j+1} + 2 \right)^3 \right) = 1.7\ GB$$

Finally, because Multigrid relies on multiple levels to converge to a solution, excluding these levels could, for some cases where error is sufficiently close to the solution tolerance, spell the difference between a successful convergence and another costly

Multigrid cycle. Therefore, it is reasonable to keep smaller coarser grids, even with ever-increasing grid sizes.

B. FULL DATA OUTPUT

TABLE VIII

FULL MPI-CUDA & CUDA DATA TIMING

TRIAL 1				
Final Level Size	Model Type	Data Saving Time (seconds)	Processing Time (seconds)	Total Execution Time (seconds)
128	CUDA	34.1	37.1	71.2
256		265.7	295.4	561.1
128	MPI-CUDA	5.4	35.2	40.6
256		40.0	257.4	297.3
TRIAL 2				
Final Level Size	Single or Multiple Processes	Data Saving Time (seconds)	Processing Time (seconds)	Total Execution Time (seconds)
128	CUDA	34.7	37.6	72.2
256		274.0	291.1	565.1
128	MPI-CUDA	5.4	35.4	40.9
256		40.2	259.2	299.4

TABLE IX

OPEN-MP SINGLE ITERATION 128^3 FINEST GRID LEVEL

Number of Threads Enabled	Data Export Time (seconds)	Total Execution Time (seconds)	Total time for single time step (seconds)
32	49.3	51.8	101.2

REFERENCES

Altrock, P. M., Liu, L. L., & Michor, F. (2015). The Mathematics of Cancer: Integrating Quantitative Models. *Nature reviews. Cancer, 15*(12), 730-745. doi:10.1038/nrc4029

Anderson, A. R., & Chaplain, M. (1998). Continuous and discrete mathematical models of tumor-induced angiogenesis. *Bulletin of mathematical biology, 60*(5), 857-899.

Byun, C., Arcand, W., Bestor, D., Bergeron, B., Hubbell, M., Kepner, J., . . . O'Gwynn, D. (2012). *Driving big data with big compute.* Paper presented at the 2012 IEEE Conference on High Performance Extreme Computing.

Coldman, A. J., & Goldie, J. H. (1986). A stochastic model for the origin and treatment of tumors containing drug-resistant cells. *Bulletin of mathematical biology, 48*(3-4), 279-292. doi:10.1007/BF02459682

Collingbourne, P., Cadar, C., & Kelly, P. H. J. (2014). Symbolic Crosschecking of Data-Parallel Floating-Point Code. *IEEE Transactions on Software Engineering, 40*(7). doi:10.1109/TSE.2013.2297120

Du, L., Herbst, R. S., & Morgensztern, D. T. O. R. P. Y. C. C. C. Y. S. o. M. C. S. W. W. W. N. H. C. T. U. S. A. (2017). Immunotherapy in Lung Cancer. *Hematology/Oncology Clinics of North America, 31*(1), 131-141. doi:10.1016/j.hoc.2016.08.004

Felip, E., Rosell, R., Maestre, J. A., Rodríguez-Paniagua, J. M., Morán, T., Astudillo, J., . . . Spanish Lung Cancer, G. (2010). Preoperative chemotherapy plus surgery versus surgery plus adjuvant chemotherapy versus surgery alone in early-stage non-small-cell lung cancer. *Journal of clinical oncology : official journal of the American Society of Clinical Oncology, 28*(19), 3138-3145. doi:10.1200/JCO.2009.27.6204

Frank, S. A., Iwasa, Y., & Nowak, M. A. (2003). Patterns of cell division and the risk of cancer. *Genetics, 163*(4), 1527-1532.

Frieboes, H. B., Jin, F., Chuang, Y.-L., Wise, S. M., Lowengrub, J. S., & Cristini, V. (2010). Three-dimensional multispecies nonlinear tumor growth—II: tumor invasion and angiogenesis. *Journal of Theoretical Biology, 264*(4), 1254-1278.

Hall, J. E. (2016). *Guyton and Hall textbook of medical physiology* [1 online resource : illustrations (some color)](13th edition. ed.). Retrieved from https://www.clinicalkey.com/dura/browse/bookChapter/3-s2.0-C20120065131
https://elsevierelibrary.co.uk/product/guyton-hall-textbook-medical-physiology70313
https://www.clinicalkey.com.au/dura/browse/bookChapter/3-s2.0-C20120065131
http://ezsecureaccess.balamand.edu.lb/login?url=https://www.clinicalkey.com/dura/browse/bookChapter/3-s2.0-C20120065131

https://nls.ldls.org.uk/welcome.html?ark:/81055/vdc_100034764868.0x000001
http://liverpool.idm.oclc.org/login?url=https://www.clinicalkey.com/meded/content/toc/3-s2.0-C20120065131
http://libanswers.liverpool.ac.uk/faq/182775
Hanahan, D., & Robert. (2011). Hallmarks of Cancer: The Next Generation. *Cell, 144*(5), 646-674. doi:10.1016/j.cell.2011.02.013
Hanahan, D., & Weinberg, R. A. (2000). The hallmarks of cancer. *Cell, 100*(1), 57-70.
Holohan, C., Van Schaeybroeck, S., Longley, D. B., & Johnston, P. G. (2013). Cancer drug resistance: an evolving paradigm. *Nature Reviews Cancer, 13*(10), 714-726. doi:10.1038/nrc3599
Institute, N. C. (2019). Cancer of Any Site - Cancer Stat Facts. Retrieved from https://seer.cancer.gov/statfacts/html/all.html
Jie, Z., Juanjuan, L., Hardesty, E., Hai, J., Kuan-Ching, L., Computer, I. A. t. I. C. o., & Information Science Taiyuan, C. J. J. (2014). GPU-in-Hadoop: Enabling MapReduce across distributed heterogeneous platforms. In *2014 IEEE/ACIS 13th International Conference on Computer and Information Science (ICIS)* (pp. 321-326): IEEE.
Kang, J., Demaria, S., & Formenti, S. (2016). Current clinical trials testing the combination of immunotherapy with radiotherapy. *Journal for ImmunoTherapy of Cancer, 4*(1). doi:10.1186/s40425-016-0156-7
Leibovich, B. C., Engen, D. E., Patterson, D. E., Pisansky, T. M., Alexander, E. E., Blute, M. L., . . . Zincke, H. (2000). Benefit Of Adjuvant Radiation Therapy for Localized Prostate cancer with a Positive Surgical Margin. *The Journal of Urology, 163*(4), 1178-1182. doi:10.1016/S0022-5347(05)67717-8
Lončar, V., Young-S, L. E., Škrbić, S., Muruganandam, P., Adhikari, S. K., & Balaž, A. (2016). OpenMP, OpenMP/MPI, and CUDA/MPI C programs for solving the time-dependent dipolar Gross-Pitaevskii equation. *Computer physics communications, 209*, 190-196.
Lowengrub, J. S., Frieboes, H. B., Jin, F., Chuang, Y. L., Li, X., Macklin, P., . . . Cristini, V. (2010). Nonlinear modelling of cancer: bridging the gap between cells and tumours. *Nonlinearity, 23*(1), R1-R91. doi:10.1088/0951-7715/23/1/R01
Michor, F., Liphardt, J., Ferrari, M., & Widom, J. (2011). What does physics have to do with cancer? *Nature Reviews Cancer, 11*(9), 657-670. doi:10.1038/nrc3092
Miller, K. D., Siegel, R. L., Lin, C. C., Mariotto, A. B., Kramer, J. L., Rowland, J. H., . . . Jemal, A. (2016). Cancer treatment and survivorship statistics, 2016. *CA: A Cancer Journal for Clinicians, 66*(4), 271-289. doi:10.3322/caac.21349
Miller, T. P., Dahlberg, S., Cassady, J. R., Adelstein, D. J., Spier, C. M., Grogan, T. M., . . . Fisher, R. I. (1998). Chemotherapy alone compared with chemotherapy plus radiotherapy for localized intermediate- and high-grade non-Hodgkin's lymphoma. *The New England journal of medicine, 339*(1), 21-26.

Napier, K. J., Scheerer, M., & Misra, S. (2014). Esophageal cancer: A Review of epidemiology, pathogenesis, staging workup and treatment modalities. *World journal of gastrointestinal oncology*, *6*(5), 112.

Ng, C. F., & Frieboes, H. B. (2017). Model of vascular desmoplastic multispecies tumor growth. *Journal of Theoretical Biology, 430*, 245-282. doi:10.1016/j.jtbi.2017.05.013

Ng, C. F., & Frieboes, H. B. (2018). Simulation of Multispecies Desmoplastic Cancer Growth via a Fully Adaptive Non-linear Full Multigrid Algorithm. *Frontiers in Physiology, 9*. doi:10.3389/fphys.2018.00821

Norton, L., & Simon, R. (1977). Tumor size, sensitivity to therapy, and design of treatment schedules. *Cancer treatment reports*, *61*(7), 1307-1317.

Nvidia. (2010). Tesla C2050/C2070 GPU Computing Processor Supercomputing at 1/10th the Cost. Retrieved from https://www.nvidia.com/docs/IO/43395/NV_DS_Tesla_C2050_C2070_jul10_lores.pdf

Pardoll, D. M. (2012). The blockade of immune checkpoints in cancer immunotherapy. *Nature reviews. Cancer*, *12*(4), 252-264. doi:10.1038/nrc3239

Prevention, C. f. D. C. a. (2018). Heart Disease Facts. Retrieved from https://www.cdc.gov/heartdisease/facts.htm

Ragaz, J., Jackson, S. M., Le, N., Plenderleith, I. H., Spinelli, J. J., Basco, V. E., . . . Olivotto, I. A. (1997). Adjuvant radiotherapy and chemotherapy in node-positive premenopausal women with breast cancer. *The New England journal of medicine*, *337*(14), 956-962.

Reid, J. E., & Wernisch, L. (2016). Pseudotime estimation: deconfounding single cell time series. *Bioinformatics (Oxford, England)*, *32*(19), 2973-2980. doi:10.1093/bioinformatics/btw372

Reiter, S., Vogel, A., Heppner, I., Rupp, M., & Wittum, G. (2013). A massively parallel geometric multigrid solver on hierarchically distributed grids. *Computing and Visualization in Science*, *16*(4), 151-164. doi:10.1007/s00791-014-0231-x

Rosenberg, S. A. (2014). Decade in review-cancer immunotherapy: entering the mainstream of cancer treatment. *Nature reviews. Clinical oncology*, *11*(11), 630-632. doi:10.1038/nrclinonc.2014.174

Rydzewska, L., Tierney, J., Vale, C. L., & Symonds, P. R. (2010). Neoadjuvant chemotherapy plus surgery versus surgery for cervical cancer. *The Cochrane database of systematic reviews*(1), CD007406. doi:10.1002/14651858.CD007406.pub2

Sanga, S., Sinek, J. P., Frieboes, H. B., Ferrari, M., Fruehauf, J. P., & Cristini, V. (2006). Mathematical modeling of cancer progression and response to chemotherapy. *Expert Review of Anticancer Therapy*, *6*(10), 1361-1376. doi:10.1586/14737140.6.10.1361

Sasako, M., Sakuramoto, S., Katai, H., Kinoshita, T., Furukawa, H., Yamaguchi, T., . . . Ohashi, Y. (2011). Five-year outcomes of a randomized phase III trial comparing adjuvant chemotherapy with S-1 versus surgery alone in stage II or III gastric cancer. *Journal of clinical oncology : official journal of the American Society of Clinical Oncology, 29*(33), 4387-4393. doi:10.1200/JCO.2011.36.5908

Schumacher, T. N., & Schreiber, R. D. (2015). Neoantigens in cancer immunotherapy. *Science, 348*(6230), 69-74. doi:10.1126/science.aaa4971

Swartz, M. A., & Lund, A. W. (2012). Lymphatic and interstitial flow in the tumour microenvironment: linking mechanobiology with immunity. *Nature reviews. Cancer, 12*(3), 210-219. doi:10.1038/nrc3186

Taboada, G. L., Ramos, S., Expósito, R. R., Touriño, J., & Doallo, R. (2013). Java in the High Performance Computing arena: Research, practice and experience. *Science of Computer Programming, 78*(5), 425-444.

Topalian, S. L., Wolchok, J. D., Chan, T. A., Mellman, I., Palucka, K., Banchereau, J., . . . Wittrup, K. D. (2015). Immunotherapy: the path to win the war on cancer? *Cell, 161*(2), 185.

Weinberg, R. A. (2013). *The Biology of Cancer*. New York, NY: W. W. Norton & Company.

Wise, S. M., Lowengrub, J. S., Frieboes, H. B., & Cristini, V. (2008). Three-dimensional multispecies nonlinear tumor growth—I: model and numerical method. *Journal of Theoretical Biology, 253*(3), 524-543.

Yang, Y. (2015). Cancer immunotherapy: harnessing the immune system to battle cancer. *Journal of Clinical Investigation, 125*(9), 3335-3337. doi:10.1172/jci83871

Yuan, Y., Salmi, M. F., Huai, Y., Wang, K., Lee, R., Zhang, X., & Ieee International Conference on Big Data Washington Dc, U. S. A. D. D. (2016). Spark-GPU: An accelerated in-memory data processing engine on clusters. In *2016 IEEE International Conference on Big Data (Big Data)* (pp. 273-283): IEEE.

Zaharia, M., Xin, R. S., Wendell, P., Das, T., Armbrust, M., Dave, A., . . . Franklin, M. J. (2016). Apache spark: a unified engine for big data processing. *Communications of the ACM, 59*(11), 56-65.