Accelerate numerical diffusion solver of 2D multi-scale and multi-resolution agent-based brain cancer model by employing graphics processing unit technology

[BIOCOMP]

Beini Jiang[1]
[1]Department of Mathematical Sciences
Michigan Tech University
Houghton, MI, USA
beinij@mtu.edu

Le Zhang[1*]
[1]Department of Mathematical Sciences
Michigan Tech University
Houghton, MI, USA
zhangle@mtu.edu

Wen Zhang[1]
[1]Department of Mathematical Sciences
Michigan Tech University
Houghton, MI, USA

Allan Struthers[1]
[1]Department of Mathematical Sciences
Michigan Tech University
Houghton, MI, USA

Michael E Berens[2]
[2]Cancer and Cell Biology Division
Translational Genomics Research Institute, TGen
Phoenix, AZ, USA

Xiaobo Zhou[3]
[3]Center for Bioinformatics and Department of Pathology
The Methodist Hospital Research Institute & Weill Cornell Medical College
Houston, Texas, USA

***Abstract*—Diffusion model is increasingly employed to simulate diffusion of biological compounds including nutrient, oxygen and chemoattractants in the agent-based model (*ABM*). However, it takes long compute time to employ conventional numerical methods such as alternating direction implicit (*ADI*) method to approximate the exact solution of the diffusion processed by sequential computing algorithm. To overcome this limitation, our study employs cutting-edge graphics processing unit (*GPU*) technology to speed up the conventional sequential numerical solver for diffusion and incorporates our proposed parallel computing algorithms into our well developed 2D multi-scale and multi-resolution agent-based brain cancer model to break through the bottleneck of the *ABM* that it is hard to simulate the large system restricted to the limited compute resource and memory. Our simulation outputs demonstrate that *ABM* model can be used to simulate real-time actual cancer progression with relative fine grids by using *GPU* based parallel computing algorithm.**

Keywords: graphics processing unit; agent-based model; alternating direction implicit method; domain decomposition; parallel computing

I. INTRODUCTION

Agent-based model (*ABM*) has become a popular method to describe the complex dynamic, adaptive and self-organizing cancer system. For example, Mansury and Deisboeck [1, 2] employed the *ABM* to simulate the expansion of brain tumor in micro-macro environments. And Zhang et al. [3-6] developed multi-scale *ABMs* to model the growth of glioma and investigate incoherent relations of the tumor expansion among macroscopic environment, microscopic environment and molecular environments. A diffusion module is employed to simulate the diffusion of the chemoattractants on the macroscopic scale environment.

Though conventional finite difference numerical methods such as *ADI*, Gauss–Seidel and Jocobi methods [7-9] for diffusion module already have been used to simulate diffusion of biological compounds such as nutrients, oxygen and chemoattractants [3, 10-15] for years, they all depend on the grid size so much that a relative fine grids can better mimic the diffusion process but significantly increase the compute time. Therefore, previous studies such as the work done by Athale et al. [10, 11] and Wang et al.[16] have to employ relatively coarse grids to reduce the compute time and the work done by Dai et al.[17] and Zhang et al.[18-20] employed special numerical scheme such as preconditioned Richardson method [21, 22] to sacrifice the compute accuracy in some dimensions of coordinates to reduce the compute resource request due to the specific aim of these biomedical projects. Nonetheless, our well developed 2D multi-scale and multi-resolution *ABM* model needs such a fast diffusion module that not only can accurately model the diffusion process but also costs less compute resource. For this reason, using parallel computing algorithm to speed up the conventional numerical

solver [23, 24] is the best promising solution. Quite a few previous parallel computing algorithms employed Message Passing Interface (*MPI*) [25], a parallel computing scheme based on multiple instruction multiple data infrastructure, to parallel the sequential numerical diffusion solver. However, *MPI* is not only too expensive to be routinely used for light computing project, but also its compute speed is limited by the communication rate [26]. Since 2007, *NVDIA* keeps releasing its graphics processing unit (*GPU*) and the novel Compute Unified Device Architecture (*CUDA*) based on single instruction multiple data infrastructure (*SIMD*). Until now, *GPU* of *NVDIA* has been evolved into a highly parallel, multithreaded, many core processor, with dramatic compute ability and high memory bandwidth [27] , especially for the recent *Fermi GPU* [28, 29]. Compared to *MPI*, *GPU* computing is more affordable, portable and suitable for the *ABM* simulation.

In general, the aim of this study is to incorporate the parallel diffusion numerical solver based on latest released *Fermi GPU* technology into our previous well developed multi-scale and multi-resolution *ABM* model [5] to resolve its compute capability shortage problem. The methods section introduces the conventional numerical scheme, alternating direction implicit (*ADI*) method [7, 30] and the *GPU* implementation [31]. And then, we show that our parallel algorithms significantly increase the performance when applied to the 2D multi-scale and multi-resolution *ABM* [5].

II. METHODS

This section gives a brief introduction to *ADI* scheme [7] with the standard domain decomposition strategy [32, 33] followed by the description of *GPU* implementations.

A. Numerical diffusion solver:ADI Scheme

The diffusion of the chemical cues is described by (1.a), where the *D* is the diffusivity for glucose (D_G=6.7×10^{-7} cm^2s^{-1}) and TGF_α (D_T=5.18×10^{-7} cm^2 s^{-1}), respectively.

$$\frac{\partial u}{\partial t} = D\nabla^2 u = D\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right) = D\left(u_{xx} + u_{yy}\right). \quad (1.a)$$

The Crank–Nicolson method approximates (1.a) by (1.b)

$$\frac{u_{ij}^{n+1} - u_{ij}^{n}}{\Delta t} = \frac{D}{2}\left(\frac{\delta_x^2}{\Delta x^2} + \frac{\delta_y^2}{\Delta y^2}\right)\left(u_{ij}^{n+1} + u_{ij}^{n}\right). \quad (1.b)$$

where u_{ij}^n is the numerical approximation of $u(x_i, y_j, t_n)$ and $x_i = i\Delta x, y_j = j\Delta y, t_n = n\Delta t$. δ_x and δ_y denote the central difference operators [7].

Introducing an intermediate level $u_{ij}^{n+1/2}$, the *ADI* method modifies (1.b) into two separate difference equations with implicit scheme, given by (2):

$$\frac{u_{ij}^{n+1/2} - u_{ij}^{n}}{\Delta t/2} = D\left(\frac{\delta_x^2}{\Delta x^2} u_{ij}^{n+1/2} + \frac{\delta_y^2}{\Delta y^2} u_{ij}^{n}\right). \quad (2.a)$$

$$\frac{u_{ij}^{n+1} - u_{ij}^{n+1/2}}{\Delta t/2} = D\left(\frac{\delta_x^2}{\Delta x^2} u_{ij}^{n+1/2} + \frac{\delta_y^2}{\Delta y^2} u_{ij}^{n+1}\right). \quad (2.b)$$

Writing $\mu_x = D\frac{\Delta t}{\Delta x^2}$ and $\mu_y = D\frac{\Delta t}{\Delta y^2}$ reduces (2) into the Peaceman-Rachford *ADI* scheme [7], shown as (3)

$$-\frac{\mu_x}{2} u_{i-1,j}^{n+1/2} + (1+\mu_x) u_{ij}^{n+1/2} - \frac{\mu_x}{2} u_{i+1,j}^{n+1/2} = \frac{\mu_y}{2} u_{i,j-1}^{n} + \left(1-\mu_y\right) u_{ij}^{n} + \frac{\mu_y}{2} u_{i,j+1}^{n}. \quad (3.a)$$

$$-\frac{\mu_y}{2} u_{i,j-1}^{n+1} + \left(1+\mu_y\right) u_{ij}^{n+1} - \frac{\mu_y}{2} u_{i,j+1}^{n+1} = \frac{\mu_x}{2} u_{i-1,j}^{n+1/2} + (1-\mu_x) u_{ij}^{n+1/2} + \frac{\mu_x}{2} u_{i+1,j}^{n+1/2}. \quad (3.b)$$

The right part of both equations of (3) is explicit formula and easily parallelized, while the left part is a symmetric and tridiagonal system of equations $Ax = b$ to be solved with the Thomas algorithm [7, 34].

Equation (3) could be written into a general form as (4.a) with $x_0 = 0$ and $x_{N+1} = 0$.

$$a_i x_{i-1} + b_i x_i + c_i x_{i+1} = d_i, i = 1,2,\dots,N. \quad (4.a)$$

The corresponding matrix form of this tridiagonal system is represented by (4.b)

$$\begin{bmatrix} b_1 & c_1 & 0 & \cdots & \cdots & 0 \\ a_2 & b_2 & c_2 & \ddots & & \\ 0 & a_3 & b_3 & \ddots & \ddots & \\ \vdots & \ddots & \ddots & \ddots & \ddots & 0 \\ \vdots & & \ddots & \ddots & \ddots & c_{N-1} \\ 0 & \cdots & \cdots & 0 & a_N & b_N \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ \vdots \\ x_N \end{bmatrix} = \begin{bmatrix} d_1 \\ d_2 \\ d_3 \\ \vdots \\ \vdots \\ d_N \end{bmatrix}. \quad (4.b)$$

B. Thomas Algorithm

The Thomas algorithm is employed to solve (4.a). It has two major steps. First is computing coefficients β_k (5.a) and ν_k (5.b) known as forward sweep. Second is using backward substitution to get solutions as (5.c).

$$\beta_k = \begin{cases} \dfrac{c_1}{b_1}; & k = 1 \\ \dfrac{c_k}{b_k - \beta_{k-1} a_k}; & k = 2,3,\dots,N-1 \end{cases}. \quad (5.a)$$

$$\nu_k = \begin{cases} \dfrac{d_1}{b_1}; & k = 1 \\ \dfrac{d_k - \nu_{k-1} a_k}{b_k - \beta_{k-1} a_k}; & k = 2,3,\dots,N \end{cases}. \quad (5.b)$$

$$\begin{cases} x_N = \nu_N \\ x_k = \nu_k - \beta_k x_{k+1}; & k = N-1, n-2, \dots, 1 \end{cases}. \quad (5.c)$$

The details of the deduction of (5) are described in Morton's book [7].

C. Domain Decomposition

For the boundary value problem on a large domain, the domain decomposition method decomposes the problem into smaller independent boundary value problems on smaller subdomains and then employ iterative method to resolve differences between the solutions on adjacent subdomains [32, 33]. We develop such a *GPU* based parallel computing

algorithm with classical alternating Schwarz method [33, 35] that can benefit from the advantages of *GPU* technology. Fig. 1(b) [31] exhibits the decomposition of a 10 by 10 array with an 8 by 8 inner array (green) and four vectors of boundary points (red) (Fig .1(a) [31]) into 4 overlapping 6 by 6 sub-arrays, each of which consists of a 4 by 4 inner array (green) and four artificial internal boundaries (red). Each sub-array is iteratively solved to make the artificial boundaries converge [7, 32, 33, 35, 36]. Here, we use the data transfer between sub-matrix 1 and sub-matrix 2 as an example. The values of the four inner elements on the rightmost side in sub-matrix 1 are sent to sub-matrix 2 as the new left artificial boundary as well as the values of the four inner elements on the leftmost side in sub-matrix 2 are sent to sub-matrix 1 as the new right artificial boundary until both artificial boundaries converge.

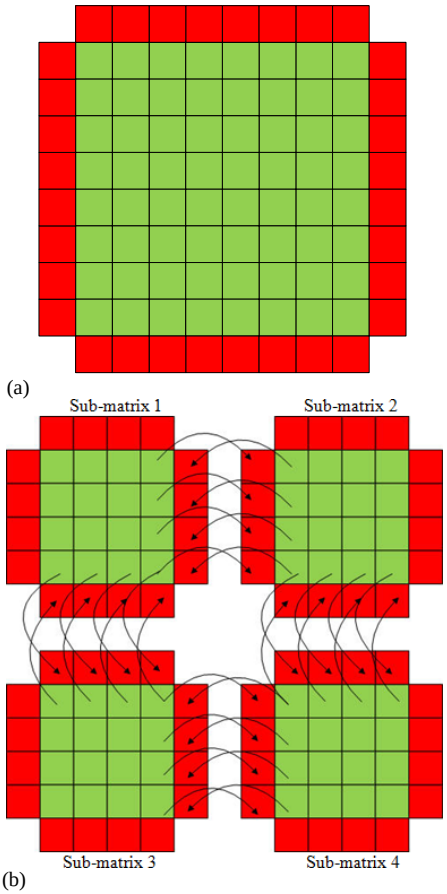

Figure 1 [31] (a) A 10 by 10 solution matrix with red to indicate boundary elements and blue to indicate inner elements and (b) Decomposition of a 10 by 10 array into 4 overlapping 6 by 6 sub-arrays with red to indicate boundary elements, green to indicate inner elements and the arrows to show how to update the boundary data.

D. Parallel Computing Algorithms to Speed up the diffusion solver

The first step of *ADI* is to set up the explicit scheme, shown as the right part of (3). Since each element could be computed independently, the explicit scheme is easy to be parallelized with single-instruction, multiple-thread (*SIMT*) infrastructure of *CUDA*. The second step is to solve the implicit scheme of *ADI* by Thomas algorithm. As we discussed in our previous research [31], Thomas algorithm is the bottleneck to speed up the conventional numerical diffusion solver.

CUDA programming has two major steps. The first is preparing such data that can be paralleled in the host side (*CPU*). The second is processing these data in the device side (*GPU*) by kernel. *CUDA* organizes the threads into a two-level hierarchy (Fig. 2-1 of *NVIDIA CUDA* Programming Guide [27]). As shown by Fig. 2-2 of *NVIDIA CUDA* Programming Guide [27], a thread executing on the device has access to the device's (*GPU*) *DRAM* and on-chip memory through 6 different memory spaces such as registers, local memory, shared memory, global memory, constant memory, and texture memory [27, 37-40]. As a very important memory of *GPU*, global memory is in charge of exchanging the data between the host (*CPU*) and the device (*GPU*). Moreover, it plays such a role that passes the messages between the threads from different blocks, since current *GPU* infrastructure prohibits the communication of threads from different blocks [27, 29, 41]. However, as an off-chip memory, the latency of global memory is very high. As on-chip memory, shared memory, registers, and constant-memory caches are much faster with much lower latency. Nonetheless, shared memory is very limited and it is only allocated to each block. For example, the capacity of the latest version of *GPU* (Fermi) is only 64KB [28, 42]. Moreover, another on-chip memory, constant memory, is disallowed to be written to during the computation [27, 43] though it is cashed.

CUDA uses a new architecture called *SIMT* to mange threads running different programs. The multiprocessor *SIMT* unit creates, manages, schedules, and executes threads in groups of 32 parallel threads we call warps [27]. We have developed three parallel computing algorithms to accelerate the numerical diffusion solver based on the new features of *GPU* technology [31]. The first is parallel computing algorithm with global memory (*PGM*), which employs only global memory to carry out parallel computing. The second is parallel computing algorithm with shared memory, global memory and *CPU* synchronization [27, 29, 41, 44] (*PSGMC*) and the third is parallel computing algorithm using shared memory, global memory and *GPU* synchronization [29, 41, 45] (*PSGMG*). *PSGMC* and *PSGMG* employ "tiles" strategy to partition the data and take advantages of both global memory and shared memory with the classical alternating Schwarz domain decomposition method [7, 32, 33, 35, 36]. The details of these three implementation methods are presented in our recent publication [31]. Here, we incorporate our fastest parallel diffusion solver into 2D multi-scale and multi-resolution *ABM* [5] to speed up the computation of *ABM*.

III. RESULTS

Our source code is implemented by C [46, 47] and *NVCC* [48] programming language and running on the recent Fermi

GPU card (GeForce GTX 480) [42, 49, 50] with *CUDA* standard.

In the beginning, let us briefly show how to use parallel computing algorithms [31] based on *GPU* technology to accelerate the numerical diffusion solver as following.

First, we employ *PGM* to compute the diffusion on the lattice with different number of grid points and compare the computing time with the sequential computing. Fig. 2 shows *PGM* computing time is not always faster than sequential algorithm for the lattice with small point number but dramatically faster than sequential algorithm for the lattice with large point number [31].

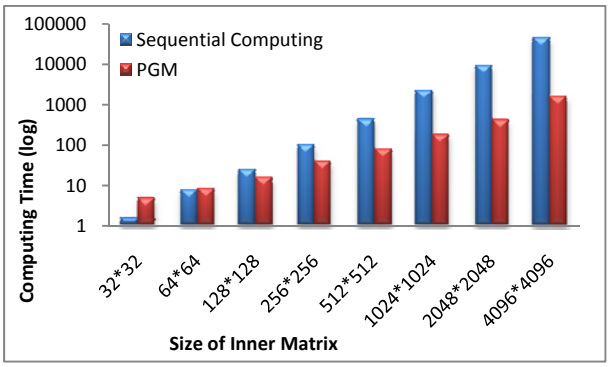

Figure 2 [31]. Computing time of *PGM* and sequential computing by logarithmic scale. The *x* axis represents the inner matrix size (number of inner grid points) and *y* axis represents the computing time (logarithmic scale with base 10) in millisecond. The blue bar represents the computing time of sequential computing and the red bar represents the computing time of *PGM.*

Second, we compare the compute time between *PSGMC* and *PGM*, when simulating the diffusion on a 4098 by 4098 lattice. Fig. 3 shows *PSGMC* improves the performance by 58% compared with *PGM* [31].

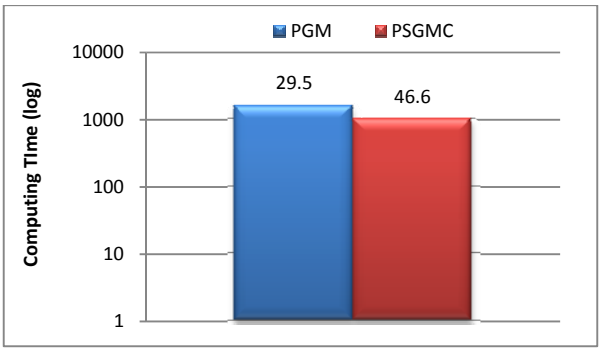

Figure 3 [31]. Computing time of *PSGMC* and *PGM* by logarithmic scale. The *y* axis represents the computing time (logarithmic scale with base 10) in millisecond. The blue bar represents the computing time of *PGM* and the red bar represents the optimal computing time of *PSGMC*. The number on each bar indicates the multiple of acceleration to the sequential computing.

Third, we compare the performance of *PSGMC* and *PSGMG*. Fig. 4 exhibits *PSGMG* improves the performance by 11% compared with *PSGMC*, when processing the diffusion on a 4098 by 4098 lattice [31].

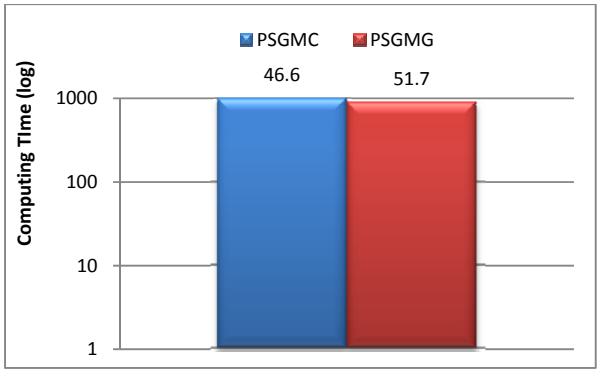

Figure 4 [31]. Computing time of *PSGMG* and *PSGMC* by logarithmic scale. The *y* axis represents the computing time (logarithmic scale with base 10) in millisecond. The blue bar represents the computing time of *PSGMC* and the red bar represents the computing time of *PSGMG*. The number on each bar indicates the multiple of acceleration to the sequential computing.

Next, we incorporate the fastest parallel computing method (*PSGMG*) into the well developed multi-scale and multi-resolution *ABM* model [5]. The multi-resolution model is designed based on two different resolution lattices, namely low-resolution lattice and high-resolution lattice. The low-resolution lattice is set up with a grid size of about 62.5 μm, on each grid point of which, a 6 by 6 high-resolution lattice with a grid size of approximately 10 μm is superimposed, described by Fig. 5 [5]. To demonstrate the advantages of the parallel computing algorithm, we scale up the lattice size of the previous multi-scale and multi-resolution *ABM* model [5]. Current low-resolution lattice is changed from 100 by 100 to 683 by 683 and high-resolution lattice is upgraded from 600 by 600 to 4098 by 4098.

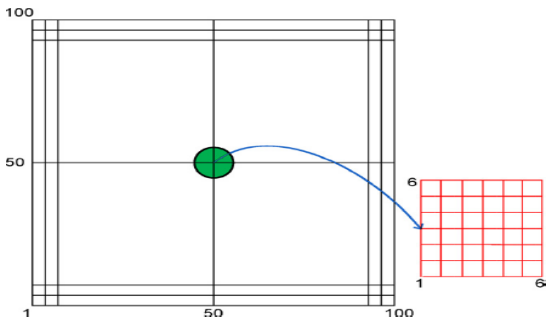

Figure 5 [5] Configuration of multi-resolution lattice.

The diffusion of the chemical cues is observed on the high-resolution lattice, with a grid size of approximately $10\mu m$, namely both Δx and Δy in the *ADI* scheme (2) are equal to $10\mu m$. Δt is set to $1s$ to make $\max\{\mu_x, \mu_y\} \leq 1$ regarding to the maximum principle [7], thus the *ADI* scheme needs to be computed 3600 times for each time step, which is equivalent to $1h$.

And then, Fig. 6 exhibits that parallel computing can significantly increase the performance of the compute time 37.5 folders than sequential computing for multi-scale and multi-resolution *ABM* model [5].

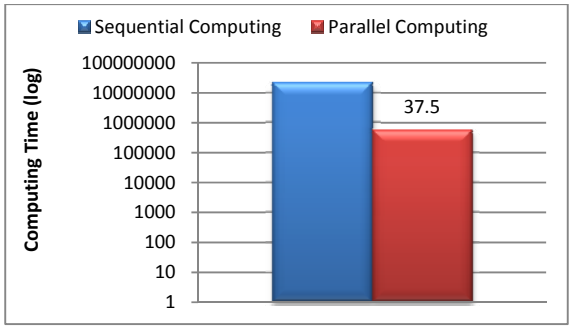

Figure 6. Computing time of parallel and sequential computing by logarithmic scale. The *y* axis represents the computing time (logarithmic scale with base 10) in millisecond. The blue bar represents the computing time of sequential computing and the red bar represents the optimal computing time of parallel computing. The number on the red bar indicates the multiple of acceleration to the sequential computing.

IV. CONCLUSIONS

This study demonstrates that it is possible to simulate the real-time actual tumor progression in a 2D lattice with relative fine grids by using *GPU* based parallel computing algorithms. Our extension research will develop a *GPU* based parallel *ODE* solver to speed up the molecular pathway module of our well developed multi-scale and multi-resolution agent-based model [5].

References

[1] Y. Mansury, M. Kimura, J. Lobo, and T. S. Deisboeck, "Emerging patterns in tumor systems: simulating the dynamics of multicellular clusters with an agent-based spatial agglomeration model," *J Theor Biol* vol. 219, pp. 343-370 2002.

[2] Y. Mansury and T. S. Deisboeck, "The impact of "search precision" in an agent-based tumor model," *J Theor Biol* vol. 224, pp. 325-337, 2003.

[3] L. Zhang, C. A. Athale, and T. S. Deisboeck, "Development of a three-dimensional multiscale agent-based tumor model: simulating gene-protein interaction profiles, cell phenotypes and multicellular patterns in brain cancer," *J Theor Biol,* vol. 244, pp. 96-107, Jan 7 2007.

[4] L. Zhang, Z. Wang, J. A. Sagotsky, and T. S. Deisboeck, "Multiscale agent-based cancer modeling," *J Math Biol,* vol. 58, pp. 545-59, Apr 2009.

[5] L. Zhang, L. L. Chen, and T. S. Deisboeck, "Multi-scale, multi-resolution brain cancer modeling," *Math Comput Simul,* vol. 79, pp. 2021-2035, Mar 2009.

[6] L. Zhang, C. Strouthos, Z. Wang, and T. S. Deisboeck, "Simulating brain tumor heterogeneity with a multiscale agent-based model: Linking molecular signatures, phenotypes and expansion rate," *Mathematical and Computer Modelling,* vol. 49, pp. 307-319, 2009.

[7] k. Q. Morton and D. F. Mayers, *Numerical solution of partial differential equations*, 2nd ed. New York: Cambridge University Press, 2008.

[8] J. C. Strikwerda, *Finite Difference Schemes and Partial Differential Equations*, 2nd ed. Philadelphia, PA: SIAM: Society for Industrial and Applied Mathematics, 2004.

[9] R. L. Burden and J. D. Faires, *Numerical analysis*, 8th ed. Belmont, CA: Thomson Higher Education, 2008.

[10] C. Athale, Y. Mansury, and T. S. Deisboeck, "Simulating the impact of a molecular 'decision-process' on cellular phenotype and multicellular patterns in brain tumors," *J Theor Biol,* vol. 233, pp. 469-81, Apr 21 2005.

[11] C. A. Athale and T. S. Deisboeck, "The effects of EGF-receptor density on multiscale tumor growth patterns," *J Theor Biol,* vol. 238, pp. 771-9, Feb 21 2006.

[12] K. R. Swanson, E. C. Alvord, Jr., and J. D. Murray, "A quantitative model for differential motility of gliomas in grey and white matter," *Cell Prolif,* vol. 33, pp. 317-29, Oct 2000.

[13] K. R. Swanson, E. C. Alvord, Jr., and J. D. Murray, "Virtual brain tumours (gliomas) enhance the reality of medical imaging and highlight inadequacies of current therapy," *Br J Cancer,* vol. 86, pp. 14-8, Jan 7 2002.

[14] K. R. Swanson, C. Bridge, J. D. Murray, and E. C. Alvord, Jr., "Virtual and real brain tumors: using mathematical modeling to quantify glioma growth and invasion," *J Neurol Sci,* vol. 216, pp. 1-10, Dec 15 2003.

[15] K. R. Swanson, R. C. Rostomily, and E. C. Alvord, Jr., "A mathematical modelling tool for predicting survival of individual patients following resection of glioblastoma: a proof of principle," *Br J Cancer,* vol. 98, pp. 113-9, Jan 15 2008.

[16] A. X. Cong, H. O. Shen, W. X. Cong, and G. Wang, "Improving the Accuracy of the Diffusion Model in Highly Absorbing Media," *International Journal of Biomedical Imaging,* vol. 2007, 2007.

[17] W. Dai, A. Bejan, X. Tang, L. Zhang, and R. Nassar, "Optimal temperature distribution in a three dimensional triple-layered skin structure with embedded vasculature," *Journal of Applied Physics,* vol. 99, 2006.

[18] L. Zhang, W. Dai, and R. Nassar, "A Numerical Method for Optimizing Laser Power in the Irradiation of a 3-D Triple-Layered Cylindrical Skin Structure," *Numerical Heat Transfer,* vol. 48, pp. 21 - 41, 2005.

[19] L. Zhang, W. Dai, and R. Nassar, "A Numerical Method for Obtaining an Optimal Temperature Distribution in a 3-D Triple-Layered Cylindrical Skin Structure Embedded with a Blood Vessel " *Numerical Heat Transfer,* vol. 49, pp. 765 - 784, 2006.

[20] L. Zhang, W. Dai, and R. Nassar, "A numerical algorithm for obtaining an optimal temperature distribution in a 3D triple-layered cylindrical skin structure," *Computer Assisted Mechanics and Engineering Sciences,* vol. 14, pp. 107-125, 2007a.

[21] B. Bialecki, "Preconditioned Richardson and Minimal Residual Iterative Methods for Piecewise Hermite Bicubic Orthogonal Spline Collocation Equations," *Siam Journal on Scientific Computing,* vol. 15, pp. 668-680, May 1994.

[22] W. H. Dai and R. Nassar, "A preconditioned Richardson method for solving three-dimensional thin film problems with first order derivatives and variable coefficients," *International Journal of Numerical Methods for Heat & Fluid Flow,* vol. 10, pp. 477-487, 2000.

[23] B. Barney, "Introduction to Parallel Computing," 2010.

[24] K. Asanovic, R. Bodik, B. C. Catanzaro, and J. J. Gebis, "The Landscape of Parallel Computing Research:A View from Berkeley," 2006.

[25] Y. Aoyama and J. Nakano, "RS/6000 SP: Practical MPI Programming," IBM, 1999.

[26] C. Rosul, "Message Passing Interface (MPI) Advantages and Disadvantages for applicability in the NoC Environment," 2005.

[27] NVIDIA, "NVIDIA CUDA Programming Guide," NVIDIA, 2009a.

[28] NVIDIA, "NVIDIA's Next Generation CUDA Compute Architecture: Fermi": NVIDIA, 2009b.

[29] W. C. Feng and S. C. Xiao, "To GPU Synchronize or Not GPU Synchronize?," in *International Symposium on Circuits and Systems* Paris, France, 2010.

[30] R. McOwen, *Partial Differential Equations: Methods and Applications*, 2nd ed. Upper Saddle River, New Jersey: Prentice Hall, 2002.

[31] B. Jiang, A. Struthers, L. Zhang, Z. Sun, Z. Feng, X. Zhao, W. Dai, K. Zhao, X. Zhou, and M. Berens, "Employing graphics processing unit technology, alternating direction implicit method and domain decomposition to speed up the numerical diffusion solver for the biomedical engineering research," *International Journal for Numerical Methods in Biomedical Engineering,* vol. (in press), 2011.

[32] B. Smith, P. Biqrstad, and W. Gropp, *Domain Decomposition: Parallel multilevel methods for elliptic partial differential equation*, 1st ed. New York: Cambridge University Press, 2004.

[33] A. St-Cyr, M. J. Gander, and S. J. Thomas, "Optimized Restricted Additive Schwarz Methods," in *16th International Conference on Domain Decomposition Methods*, New York 2005.

[34] W. Dai, "A Parallel Algorithm for Direct Solution of Large Scale Five-Diagonal Linear Systems," in *Proceedings of the Seventh SIAM Conference on Parallel Processing for Scientific Computing*, San Francisco, CA, 1995, p. 875.

[35] X. C. Cai and M. Sarkis, "A restricted additive Schwarz preconditioner for general sparse linear systems," *Siam Journal on Scientific Computing,* vol. 21, pp. 792-797, Oct 26 1999.

[36] J. P. Zhu, *Solving Partial Differential Equations On Parallel Computers*. London: World Scientific Publishing Co. Pte. Ltd., 1994.

[37] V. Volkov and J. Demmel, "Benchmarking GPUs to Tune Dense Linear Algebra," in *Conference on High Performance Networking and Computing archive Proceedings of the 2008 ACM/IEEE conference on Supercomputing* Austin, TX: IEEE Press Piscataway, NJ, USA 2008.

[38] J. Nickolls, I. Buck, M. Garland, and K. Skadron, "Scalable Parallel Programming with CUDA," in *ACM Queue*. vol. , 2008, pp. 42-53.

[39] M. Guevara, C. Gregg, K. hazelwood, and K. Skadron, "Enabling Task parallelism in the CUDA Scheduler," in *Proceedings of the Workshop on Programming Models for Emerging Architectures (PMEA)* Raleigh, NC, 2009.

[40] S. Che, M. Boyer, J. Y. Meng, D. Tarjan, J. W. Sheaffer, and K. Skadron, "A performance study of general-purpose applications on graphics processors using CUDA," *Journal of Parallel and Distributed Computing,* vol. 68, pp. 1370-1380, Oct 2008.

[41] S. C. Xiao, A. M. Aji, and W. C. Feng, "On the Robust Mapping of Dynamic Programming onto a Graphics Processing Unit," in *International Conference on Parallel and Distributed Systems* Shenzhen, China, 2009.

[42] NVIDIA, "Tuning CUDA Applications for Fermi," NVIDIA, 2010.

[43] D. Kirk and W. M. Hwu, *Programming Massively Parallel Processors*, 1st ed. Burlington, MA: Morgan Kaufmann, 2010.

[44] M. Boyer, M. Sarkis, and W. Weimer, "Automated Dynamic Analysis of CUDA Programs," in *Third Workshop on Software Tools for MultiCore Systems in conjunction with the IEEE/ACM International Symposium on Code Generation and Optimization (CGO)* Boston, MA: , 2008.

[45] S. C. Xiao and W. C. Feng, "Inter-Block GPU Communication via Fast Barrier Synchronization," in *In Proc. of the IEEE International Parallel and Distributed Processing Symposium* Atlanta, GA 2010.

[46] B. W. Kernighan and D. M. Ritchie, *The C Programming Language*, 2nd ed. Englewood Cliffs, New Jersey: Prentice Hall, 1988.

[47] S. G. Kochan, *Programming in C*, 3rd ed. Indianapolis, Indiana: Sams, 2004.

[48] NVIDIA, "The CUDA Compiler Driver NVCC," NVIDIA, 2007.

[49] P. N. Glaskowsky, "NVIDIA's Fermi: The First Complete GPU Computing Architecture " 2009.

[50] T. R. Halfhill, "Looking Beyond Graphics," 2009.