Solving Discontinuous Collocation Equations for a Class of Brain Tumor Models on GPUs

I.E. Athanasakis, N.D. Vilanakis and E.N. Mathioudakis

***Abstract*—Brain tumor modeling has been a challenging research subject over the past decades. Incorporation of brain's tissue heterogeneity (gray-white matter) in the models, has been realized by the introduction of a discontinuous diffusion coefficient in the core PDE, as tumor cells migrate with different rates in brain's white and gray matter. For the numerical treatment of these models, recently, a fourth order Discontinuous Hermite Collocation (DHC) method coupled with high order semi implicit and strongly stable Runge-Kutta (RK) time discretization schemes, has been successfully developed. And as, at each time step, large linear systems has to be solved, significant computational cost emerges. For their efficient solution the incomplete LU factorization preconditioned BiCG stabilized iterative method, suggested by the eigenvalue topology, is adopted. The implementation of the numerical method in Matlab environment takes place in a multicore CPU-only computational architecture. Here, we present the design and development of a CPU-GPU parallel algorithm to carry out the whole computation. Its mapping and implementation on a GPU-CPU architecture necessitates the use of CUDA development tools. Several numerical experiments are presented to demonstrate the efficiency of the parallel algorithm.**

***Index Terms*—Discontinuous Hermite Collocation, DIRK methods, Matlab, CUDA, CPU-GPU computations.**

I. The numerical brain tumor model

Simulations of the progress of untreated diffusive brain tumors are based on classical reaction-diffusion mathematical models, such as in [1],[2] and [3]. Recently, Swanson ([4],[5]) introduced an appropriately discontinuous diffusion coefficient and generalized these models incorporating the heterogeneity of the brain tissue (white-grey matter). The basic linear version of the model is then expressed with the following reaction-diffusion equation:

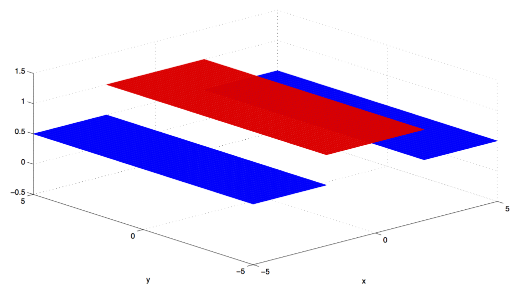


Fig. 1. Discontinuous coefficient D for a two-value problem.

Manuscript received March 23, 2015 ; revised April 6, 2015.

This work was supported by EU (European Social Fund ESF) and Greek funds through the operational program *Education and Lifelong Learning* of the National Strategic Reference Framework (NSRF) - Research Funding Program: THALIS.

All authors are with the Applied Mathematics and Computers Laboratory, Technical University of Crete, University Campus, 73132 Chania, Crete, Greece – Corresponding author's e-mail: manolis@amcl.tuc.gr.

$$\frac{\partial \bar{c}}{\partial \bar{t}} = \nabla \cdot \left(\bar{D}(\bar{\mathbf{x}}) \nabla \bar{c} \right) + \rho \bar{c} \quad , \tag{1}$$

where $\bar{c}(\bar{\mathbf{x}}, \bar{t})$ denotes the tumour cell density, ρ denotes the net proliferation rate, and $\bar{D}(\bar{\mathbf{x}})$ is the diffusion coefficient representing the active motility of malignant cells satisfying

$$\bar{D}(\bar{\mathbf{x}}) = \begin{cases} D_g & , \quad \bar{\mathbf{x}} \text{ in Grey Matter} \\ D_w & , \quad \bar{\mathbf{x}} \text{ in White Matter} \end{cases} , \tag{2}$$

with D_g and D_w scalars and $D_w > D_g$. Using the dimensionless variables :

$$\mathbf{x} = \sqrt{\frac{\rho}{D_w}}\bar{\mathbf{x}} \quad , \quad t = \rho \bar{t} \quad , \quad f(\mathbf{x}) = \bar{f}\left(\sqrt{\frac{\rho}{D_w}}\bar{\mathbf{x}} \right),$$

$$\text{and} \quad c(\mathbf{x}, t) = \bar{c}\left(\sqrt{\frac{\rho}{D_w}}\bar{\mathbf{x}}, \rho \bar{t} \right) \frac{D_w}{\rho N_0}$$

with $N_0 = \int \bar{f}(\bar{\mathbf{x}}) d\bar{\mathbf{x}}$ to denote the initial number of tumour cells in the brain at $\bar{t} = 0$, as well as the transformation

$$c(\mathbf{x}, t) = e^t u(\mathbf{x}, t) \quad ,$$

the model in $2+1$ dimensions reduces to

$$\begin{cases} u_t = (Du_x)_x + (Du_y)_y \quad , \quad (x, y, t) \in [a, b]^2 \times \mathbb{R}^+ \\ \frac{\partial u}{\partial \eta} = 0, \ u(x, y, 0) = f(x, y), \ D = \begin{cases} \gamma, x \notin [w_1, w_2) \\ 1, x \in [w_1, w_2) \end{cases} \end{cases} \tag{3}$$

where the diffusion coefficient D is discontinuous along two interface lines $x = w_1$ and $x = w_2$, as it is depicted in Fig. 1. The discontinuous diffusion coefficient D, directly implies discontinuity of u_x, hence continuity of Du_x, across each interface line. In fact, the linear parabolic nature of the initial-boundary value problem implies continuity of u across each interface, that is

$$[u] := \lim_{x \to w_k^+} u(x, y_0) - \lim_{x \to w_k^-} u(x, y_0) = 0 \tag{4}$$

and

$$[Du_x] := \lim_{x \to w_k^+} Du_x(x, y_0) - \lim_{x \to w_k^-} Du_x(x, y_0) = 0 \tag{5}$$

where $k = 1, 2$ and $y = y_0$ is a fixed point in $[a, b]$. Taking, now, into consideration the above two continuity constrains an alternative way to state the model can be described by

$$\begin{cases} u_t = Du_{xx} + Du_{yy} \quad , \quad (x, y) \in R_\ell \\ \frac{\partial u}{\partial \eta} = 0 \quad , \quad u(x, y, 0) = f(x, y) \\ [u] = 0 \quad , \quad [Du_x] = 0 \quad \text{at} \quad x = w_{1,2} \end{cases} \tag{6}$$

where $R_1 := [a, w_1) \times [a, b]$, $R_2 := [w_1, w_2) \times [a, b]$ and $R_3 := [w_1, b) \times [a, b]$.

A new fourth order tensor product Discontinuous Hermite Collocation (DHC) method has been successfully developed in [6] (see also [7] for the 1D analog model) and coupled with high order Runge-Kutta schemes for the effective solution of the model problem described above. The approximate solution, DHC method seeks, is in the form

$$U(x,y,t) = \sum_{i=1}^{2N_y+2} \sum_{j=1}^{2N_x+2} [\alpha_{i,j}(t)\phi_i(x)\phi_j(y)] \qquad (7)$$

where $N_y = N_x = N_{x_1} + N_{x_2} + N_{x_3}$ denotes the number of subintervals of R_ℓ and ϕ the Hermite cubic basis functions. For the determination of the $\alpha_{i,j}$ unknowns in (7), the DHC leads to the system of ODEs (cf. [6]):

$$\left(C_x^0 \otimes C_y^0\right)\dot{\mathbf{a}} = \left(D_x C_x^2 \otimes C_y^0\right)\mathbf{a} + \left(D_x C_x^0 \otimes C_y^2\right)\mathbf{a} \qquad (8)$$

where C_x^* and C_y^* denote the $1d$ Discontinuous and Continuous Collocation matrices respectively (cf. [7]), D_x is the diagonal matrix $D_x = diag(\gamma,\ldots,\gamma,1,\ldots,1,\gamma,\ldots,\gamma)$, and $\otimes$ denotes the Kronecker (tensor) matrix product. Setting, now, $A_0 := C_x^0 \otimes C_y^0$ and $B := D_x C_x^2 \otimes C_y^0 + D_x C_x^0 \otimes C_y^2$ of order $N = 4N_xN_y$ equation (8) can be written as:

$$A_0\dot{\mathbf{a}} = B\mathbf{a} \quad \text{or} \quad \dot{\mathbf{a}} = C(t,\mathbf{a}) \quad \text{for} \quad C(t,\mathbf{a}) = A_0^{-1}B\mathbf{a}. \qquad (9)$$

Finally, the DHC method is coupled (cf. [7],[6]) with an optimal two-step and third-order Diagonally implicit Runge-Kutta scheme [8] for the time discretization, yielding the system of linear equations:

$$\begin{aligned}
\mathbf{a}^{(n,1)} &= \mathbf{a}^{(n)} + \tau\ \lambda\ C(t_{n,1},\mathbf{a}^{(n,1)})\\
\mathbf{a}^{(n,2)} &= \mathbf{a}^{(n)} + \tau\left[(1-2\lambda)C(t_{n,1},\mathbf{a}^{(n,1)}) + \lambda C(t_{n,2},\mathbf{a}^{(n,2)})\right]\\
\mathbf{a}^{(n+1)} &= \mathbf{a}^{(n)} + \frac{\tau}{2}\left[C(t_{n,1},\mathbf{a}^{(n,1)}) + C(t_{n,2},\mathbf{a}^{(n,2)})\right] \qquad (10)
\end{aligned}$$

or, equivalently,

$$\left.\begin{aligned}
A\ \mathbf{a}^{(n,1)} &= A_0\ \mathbf{a}^{(n)}\\
A\ \mathbf{a}^{(n,2)} &= A_0\ \mathbf{a}^{(n)} + \tau(1-2\lambda)B\ \mathbf{a}^{(n,1)}\\
A_0\ \mathbf{a}^{(n+1)} &= A_0\ \mathbf{a}^{(n)} + \frac{\tau}{2}\left[B\ \mathbf{a}^{(n,1)} + B\ \mathbf{a}^{(n,2)}\right]
\end{aligned}\right. \qquad (11)$$

where $A := A_0 - \tau\lambda B$, τ is the time spacing and $\lambda = \frac{1}{2} + \frac{\sqrt{3}}{2}$.

The solution, now, of the above system of linear equations is computationally described by means of the following algorithm:

```
Create matrices $A, A_b, A_0, B$ and $\boldsymbol{a_{old}}$
for $t = dt$ to $t_{max}$ with time step $dt$
    Compute $a_0 = A_0\boldsymbol{a_{old}}$
    if $t = dt$ then
        Solve $A_b\boldsymbol{a_{new}} = \boldsymbol{a_0}$
    else
        Solve $A_0\,\boldsymbol{a_1} = \boldsymbol{a_0}$
        Compute $\boldsymbol{a_0} = \boldsymbol{a_0} - dt\frac{\sqrt{3}}{3}B\boldsymbol{a_1}$
        Solve $A\,\boldsymbol{a_2} = \boldsymbol{a_0}$
        Compute $\boldsymbol{a_2} = \boldsymbol{a_0} + \frac{dt}{2}B(\boldsymbol{a_1} + \boldsymbol{a_2})$
        Solve $A_0\,\boldsymbol{a_{new}} = \boldsymbol{a_2}$
    endif
    Compute $\boldsymbol{a_{old}} = \boldsymbol{a_{new}}$
endfor
```

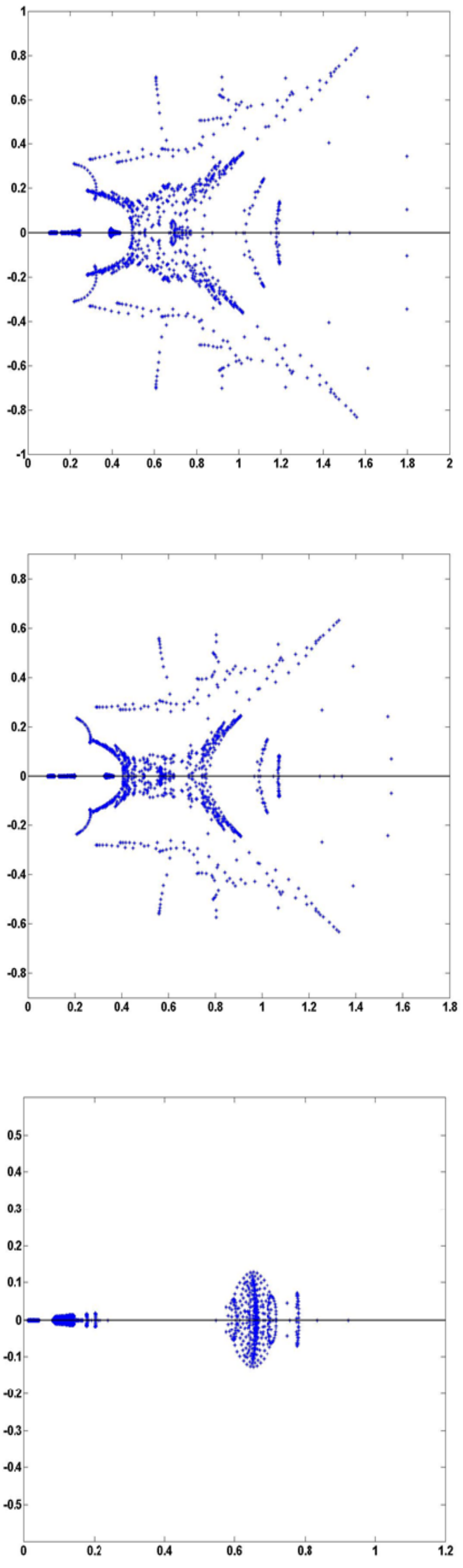


Fig. 2. Eigenvalue charts of matrices A_b, A_0 and A, respctively.

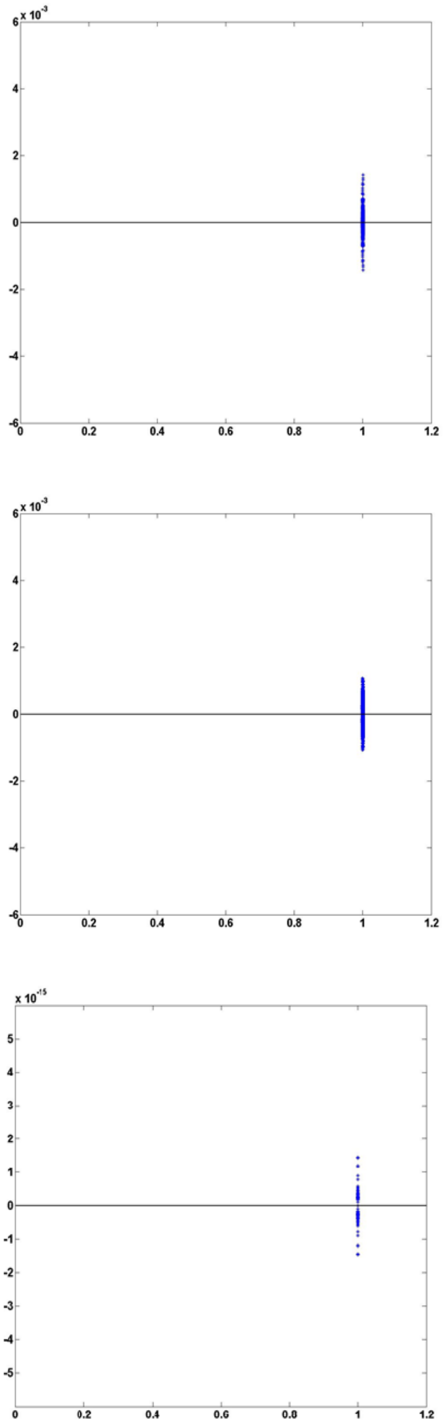


Fig. 3. Eigenvalue charts of matrices A_b, A_0 and A, respctively, with the application of the preconditioning technique.

We remark that the first step of the algorithm utilizes the matrix A_b, where $A_b := A_0 - \tau B$. The use of this matrix refers to the application of Backward Euler method to avoid possible oscillations. For every other step from there on, matrices A, A_0 and B involved in the calculations, arise from the DIRK scheme. And as all of them are large and sparse, iterative methods are preferable for the solution of the corresponding linear systems. The iterative method could be a stationary one or a Krylov subspace method. A stationary method would demand a matrix splitting in the form of $A = D - L - U$, but the nature and the properties of the matrices indicate that would not be an appropriate approach. On the other hand, Krylov subspace methods, where matrix-vector products are the dominant operations, are well suited for parallel execution. Therefore, the BiCGSTAB method [9] is preferred for the solution of the three linear systems in each step of the algorithm. The selection of this particular method is justified by the eigenvalue properties of the matrices involved in the linear systems described above, as well as effective preconditioning techniques.

To be more specific, calculation of the eigenvalues (see Fig. 2) indicates that a suitable preconditioning technique would have a positive effect on the convergence rate of the Krylov iterative method. Since the matrices are stored in a sparse format, the incomplete LU factorization of each matrix A_b, A_0 and A, that is $M_{A_b} :=\mathrm{i}LU(A_b)$, $M_{A_0} :=\mathrm{i}LU(A_0)$ and $M_A :=\mathrm{i}LU(A)$, is a very convenient choice as a preconditioner. An essential fact we should emphasize on, is that all three matrices are time independent, therefore their $\mathrm{i}LU$ factorization computation is performed once before the time stepping process.

Following, now, $\mathrm{i}LU(A)$ preconditioning, it becomes clear (inspect Fig. 3) that the eigenvalues of each one of the matrices are clustered around unity and notably close to the real semi-axis, indicating the proper selection of the preconditioning technique and encouraging the use of the BiCGSTAB method instead of GMRES [10].

II. The parallel algorithm

During each time step, the proposed numerical scheme requires the solution of three linear systems, with time-independent coefficient matrices, by an iterative method (BiCGSTAB) in which basic linear algebra operations dominate. The decoding of this sentence indicates the development of an application that will utilize both the CPU and the GPU; sequential parts, like backward and forward substitutions, will be executed on the CPU, while the demanding floating-point operations of the compute-intensive parts will be transferred and performed on the GPU. It should be recalled that a Graphics Processing Unit possesses a great number of cores which are heavily multithreaded, working in a single instruction-multiple data mode. That makes it an excellent numeric computing engine, in case of matrix-vector multiplication operations (see for instance [11] and [12]). Therefore, in the parallel algorithm the creation of the matrices A_b, A_0, A, B and vector $\boldsymbol{a_{old}}$ along with the computation of the $\mathrm{i}LU$ factorization of each matrix take place on the CPU. The data are being copied to the GPU's memory before the beginning of the time stepping process and from there on the vast majority of the floating-point operations is performed on the GPU. The communication cost is related to the length of the vectors needed to be transferred between CPU and GPU for the execution of the preconditioned system's backward and forward substitutions with coefficient matrices M_{A_b}, M_{A_0} and M_A.

The primary algorithm is being described hereupon. As the time procedure is in progress, the BiCGSTAB solver takes place three times in each iterative step in a CPU-GPU fashion way, therefore this particular part of the parallel algorithm is being described apart.

```
Create matrices on CPU  $A_b, A_0, A, B$  and  $\boldsymbol{a_{old}}$
Compute on CPU iLU factorizations for matrices $A_b, A_0$ and $A$
Send from CPU to GPU matrices $A_b, A_0, A, B$ and $\boldsymbol{a_{old}}$
for  $t = dt$  to  $t_{max}$ with time step  $dt$
    Compute in parallel on GPU   $\boldsymbol{a_0} = A_0\boldsymbol{a_{old}}$
    if  $t = dt$  then
        Solve in parallel on GPU with BiCGSTAB $A_b\boldsymbol{a_{new}} = \boldsymbol{a_0}$
    else
        Solve in parallel on GPU with BiCGSTAB $A\boldsymbol{a_1} = \boldsymbol{a_0}$
        Compute in parallel on GPU  $\boldsymbol{a_0} = \boldsymbol{a_0} - dt\frac{\sqrt{3}}{3}B\boldsymbol{a_1}$
        Solve in parallel on GPU with BiCGSTAB  $A\boldsymbol{a_2} = \boldsymbol{a_0}$
        Compute in parallel on GPU $\boldsymbol{a_2} = \boldsymbol{a_0} + \frac{dt}{2}B(\boldsymbol{a_1} + \boldsymbol{a_2})$
        Solve in parallel on GPU with BiCGSTAB $A_0\boldsymbol{a_{new}} = \boldsymbol{a_2}$
    endif
    Compute in parallel on GPU   $\boldsymbol{a_{old}} = \boldsymbol{a_{new}}$
endfor
Send from GPU to CPU solution vector  $\boldsymbol{a_{new}}$
```

The iLU preconditioned GPU - BiCGSTAB iterative method is described with the following parallel algorithm in case of solving the $A\boldsymbol{x} = \boldsymbol{b}$ linear system, with LU been the incomplete factorization of matrix A :

```
Choose initial approximation $\boldsymbol{x}^{(0)}$ of the solution $\boldsymbol{x}$
Compute on GPU   $r^{(0)} = \boldsymbol{b} - A\boldsymbol{x}^{(0)}$
Choose $\hat{r}$ (usually $\hat{r} = r^{(0)}$)
for $i = 1, 2, ...$
    Compute on GPU   $\rho_{i-1} = \hat{r}^T\, r^{(i-1)}$
    if $\rho_{i-1} = 0$  method fails
    if  $i = 1$
        Compute on GPU   $p^{(1)} = r^{(0)}$
    else
        $\beta_{i-1} = \frac{\rho_{i-1}}{\rho_{i-2}}\;\frac{\alpha_{i-1}}{\omega_{i-1}}$
        Compute on GPU $p^{(i)} = r^{(i-1)} + \beta_{i-1}(p^{(i-1)} - \omega_{i-1}v^{(i-1)})$
    endif
    Send from GPU to CPU  $p^{(i)}$
    Solve on CPU  $L\,\boldsymbol{y} = p^{(i)}$
    Solve on CPU  $U\,\hat{p} = \boldsymbol{y}$
    Send from CPU to GPU  $\hat{p}$
    Compute on GPU   $v^{(i)} = A\,\hat{p}$
    Compute on GPU   $\alpha_i = \frac{\rho_{i-1}}{\hat{r}^T v^{(i)}}$
    Compute on GPU   $s = r^{(i-1)} - \alpha_i\, v^{(i)}$
    if  $\|\, s\,\|$  is small enough then
        Compute on GPU   $\boldsymbol{x}^{(i)} = \boldsymbol{x}^{(i-1)} + \alpha_i\,\hat{p}$  stop
    Send from GPU to CPU  $s$
    Solve on CPU  $L\,\boldsymbol{y} = s$
    Solve on CPU  $U\,z = \boldsymbol{y}$
    Send from CPU to GPU  $z$
    Compute on GPU   $t = A\,z$
```

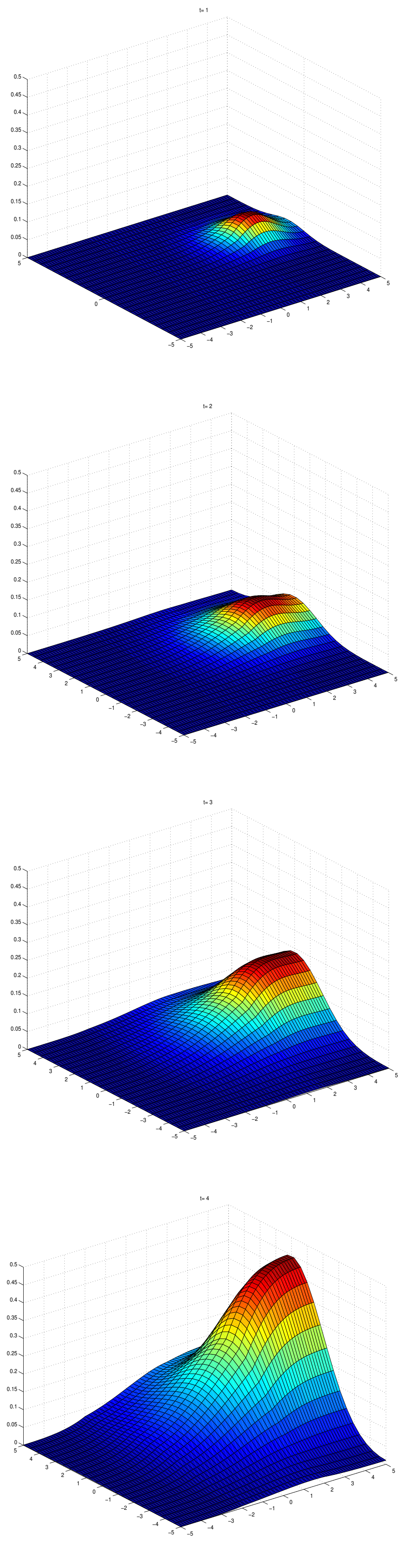


Fig. 4. Model problem's approximate solutions for 2D DHC-DIRK numerical scheme at time steps $t = 1, 2, 3$ and 4 respectively.

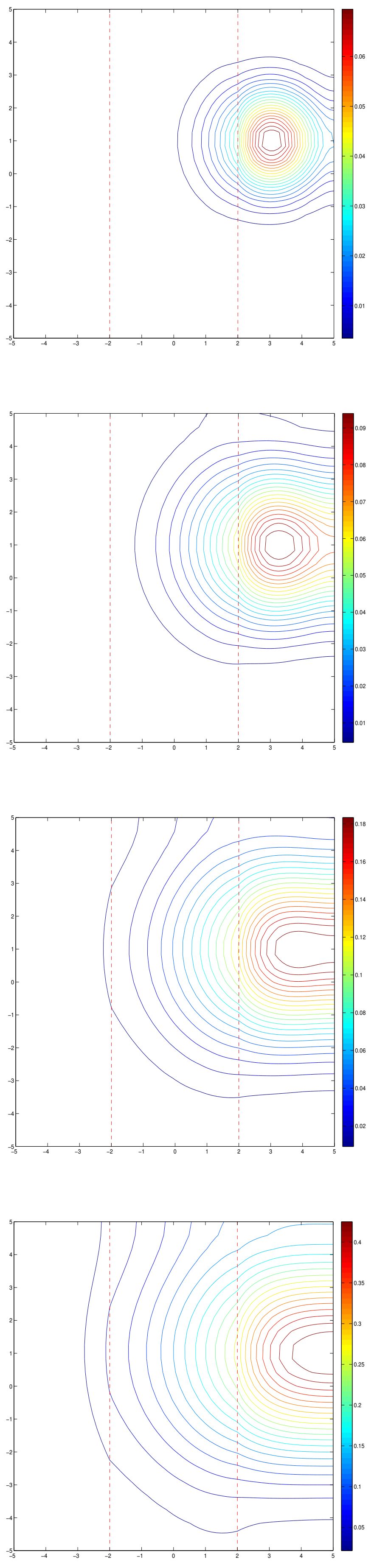


Fig. 5. Model problem's contour plots of approximate solutions at time steps $t = 1, 2, 3$ and 4 respectively.

```
    Compute on GPU    $\omega_i = \frac{s^T t}{t^T t}$
    Compute on GPU    $\boldsymbol{x}^{(i)} = \boldsymbol{x}^{(i-1)} + \alpha_i \hat{p} + \omega_i z$
    Check for Convergence
    if  $\omega_i = 0$  stop
    Compute on GPU    $r^{(i)} = s - \omega_i t$
endfor
```

III. CPU-GPU IMPLEMENTATION

The implementations of the developed algorithm were carried out on a shared memory HP SL390s G7, consisting of two 6-core Xeon X5600@2.8 GHz type processors with 12 MB Level 3 cache memory each. The total memory is 24 GB and the operating system is Oracle's Linux version 6.2. The machine is also equipped with a Fermi edition Tesla M2070 GPU, with 6 GB of memory and 448 cores on 14 multiprocessors. The time measurements comparison is between 2 different applications that were developed. The first one was developed in Matlab R2014b [13] and runs on a multicore CPU-only environment, while the second one was developed in Matlab R2014b and in PGI's 15.3 CUDA Fortran [14] and runs on a CPU-GPU environment. In the development of the CPU-GPU application, subroutines from cuBLAS and cuSPARSE libraries [15] (in GPU operations) and from SparseKit (in CPU operations) were used for CUDA 6.0 compiler suite [16]. We have to mention that Matlab software does not support yet GPU computations for sparse matrices. This is the main reason that the CPU-GPU part of the algorithm is implemented in CUDA Fortran. The Matlab Compiler toolbox from 2014b software version is used for the compilation of the standalone multicore Matlab application for the CPU-only implementations.

Time evolution of the brain's tumor model numerical solution is depicted in Figures Fig. 4 and 5, for the case of $t_{max} = 4$ and $dt = 0.05$, and the numerical scheme's successful treatment of the problem is being demonstrated. The regions (stripes) with different rates of cell motility as well as the discontinuity interfaces are visible and handled effectively by the numerical scheme.

In the following Table I the time measurements (in seconds) are presented, comparing the Matlab multithread CPU implementation and the CUDA Fortran CPU-GPU implementation, in different cases of discretization, namely n = 400, 1600, 6400 and 25600 finite elements. The size of the linear systems involved in each test case is equal to the number of unknowns $N = 4n$ to be evaluated. Also the total degrees of freedom (dof) are mentioned for every finite element problem size. The observed acceleration is graphically presented in Fig. 6 for all test cases between the multicore CPU Matlab and the CPU-GPU Matlab-CUDA Fortran realization of the parallel algorithm.

Table I : Execution time measurements for Matlab multithread CPU and Matlab-CUDA Fortran CPU-GPU implementations

Number of elements	Number of unknowns	dof	CPU Matlab Time	CPU - GPU Time
400	1600	6400	0.83	0.31
1600	6400	25600	2.35	0.94
6400	25600	102400	11.5	4.55
25600	102400	409600	202	81.8

Beside the time results comparing the two applications, there are also a few comments to be made about the implementations using the GPU. Regarding the communication cost between the CPU and the GPU memories, the duration of the transfers when copying from the GPU to the CPU and backwards was the same for the same amount of data, in case of 25600 finite elements, and was less than 1 second. The same behavior was noticed in cases with less finite elements. In addition, when having a closer look at the GPU's computations, it was confirmed that the matrix-vector multiplication procedure is the most time consuming one. For instance, in case of 25600 finite elements, the matrix-vector operation was performed 5008 times and consumed nearly 64% of the GPU implementation

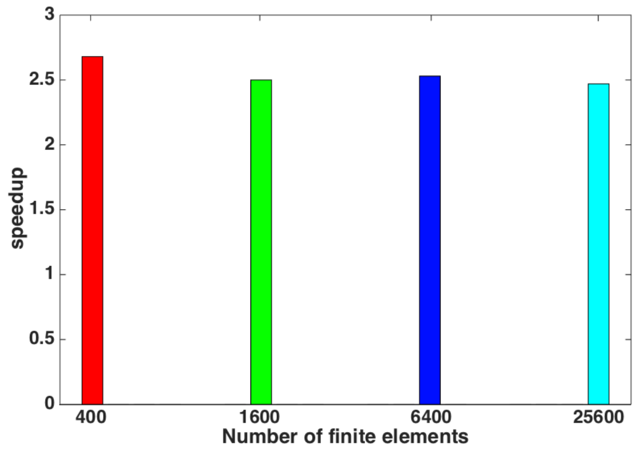
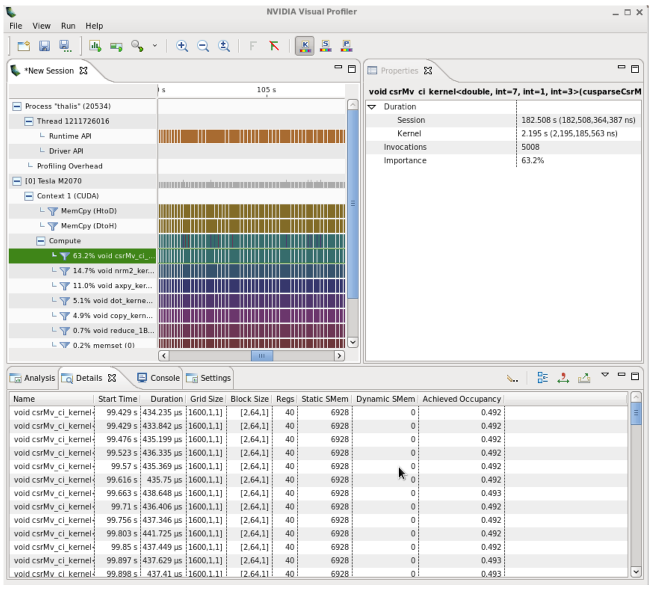


Fig. 6. Speedup measurements for CPU Matlab and CPU-GPU Matlab-CUDA Fortran implementations.

time, while the vector addition operation which was performed 13450 times consumed 11%.



void csrMv ci kernel<double, int=7, int=1, int=3>(cusparseCsrM

Duration	
Session	182.508 s (182,508,364,387 ns)
Kernel	2.195 s (2,195,185,563 ns)
Invocations	5008
Importance	63.2%

Name	Start Time	Duration	Grid Size	Block Size	Regs	Static SMem	Dynamic SMem	Achieved Occupancy
void csrMv_ci_kernel‹	99.429 s	434.235 µs	1600,1,1]	[2,64,1]	40	6928	0	0.492
void csrMv_ci_kernel‹	99.429 s	433.842 µs	1600,1,1]	[2,64,1]	40	6928	0	0.492
void csrMv_ci_kernel‹	99.476 s	435.199 µs	1600,1,1]	[2,64,1]	40	6928	0	0.492
void csrMv_ci_kernel‹	99.523 s	436.335 µs	1600,1,1]	[2,64,1]	40	6928	0	0.492
void csrMv_ci_kernel‹	99.57 s	435.369 µs	1600,1,1]	[2,64,1]	40	6928	0	0.492
void csrMv_ci_kernel‹	99.616 s	435.75 µs	1600,1,1]	[2,64,1]	40	6928	0	0.492
void csrMv_ci_kernel‹	99.663 s	438.648 µs	1600,1,1]	[2,64,1]	40	6928	0	0.493
void csrMv_ci_kernel‹	99.71 s	436.406 µs	1600,1,1]	[2,64,1]	40	6928	0	0.492
void csrMv_ci_kernel‹	99.756 s	437.346 µs	1600,1,1]	[2,64,1]	40	6928	0	0.492
void csrMv_ci_kernel‹	99.803 s	441.725 µs	1600,1,1]	[2,64,1]	40	6928	0	0.492
void csrMv_ci_kernel‹	99.85 s	437.449 µs	1600,1,1]	[2,64,1]	40	6928	0	0.492
void csrMv_ci_kernel‹	99.897 s	437.629 µs	1600,1,1]	[2,64,1]	40	6928	0	0.493
void csrMv_ci_kernel‹	99.898 s	437.41 µs	1600,1,1]	[2,64,1]	40	6928	0	0.493

Fig. 7. NVIDIA's Visual Profiler graphical performance application tool.

The above observations were made using the NVIDIA's Visual Profiler as shown in Fig. 7.

IV. Conclusions

A new parallel algorithm for computing architectures with accelerators, implementing the Discontinuous Hermite Collocation method, has been developed. The algorithm was realized on machines with Graphics Processing Units and the time measurements were compared to Matlab multicore implementations. The results reveal the highest efficiency of the CPU-GPU implementation since the performance acceleration that was observed reached 2.5x.

Acknowledgment

The present research work has been co-financed by the European Union (European Social Fund ESF) and Greek national funds through the Operational Program *Education and Lifelong Learning* of the National Strategic Reference Framework (NSRF) - Research Funding Program: THALIS. Investing in knowledge society through the European Social Fund.

References

[1] G.C. Cruywagen, D.E. Woodward, P. Tracqui, G.T. Bartoo, J.D. Murray and E.C. Alvord Jr., "The modeling of diffusive tumours, journal of biological systems," *Journal of Biological Systems*, vol. 3, pp. 937–945, 1995.

[2] P.K. Burgess, P.M. Kulesa, J.D. Murray and E.C. Alvord Jr., "The interaction of growth rates and diffusion coefficients in a three-dimensional mathematical model of gliomas," *Journal of Neuropathology and Experimental Neurology*, vol. 14(5), pp. 704–713, 1997.

[3] D.E. Woodward, J.Cook, P.Tracqui, G.C. Cruywagen, J.D. Murray and E.C. Alvord Jr., "A mathematical model of glioma growth: the effect of extent of surgical resection, journal of biological systems," *Cell Proliferation*, vol. 29, pp. 269–288, 1996.

[4] K.R. Swanson, E.C. Alvord Jr. and J.D. Murray, "A quantitive model for differential motility of gliomas in grey and white matter," *Cell Proliferation*, vol. 33, pp. 317–329, 2000.

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

[6] I.E. Athanasakis, E.P. Papadopoulou and Y.G. Saridakis, "Discontinuous Tensor Product Collocation and Runge-Kutta schemes for heterogeneous brain tumor invasion models," *Submitted*, 2015.

[7] I.E. Athanasakis, M.G. Papadomanolaki, E.P. Papadopoulou and Y.G. Saridakis, "Discontinuous Hermite Collocation and Diagonally Implicit RK3 for a brain tumour invasion model," *Proceedings of WCE 2013 Vol I, IAENG*, pp. 241–246, 2013.

[8] R. Alexander, "Diagonally implicit runge-kutta methods for stiff odes," *SIAM Num. Anal.*, vol. 14(5), pp. 1006–1021, 1977.

[9] H.A. van der Vorst, "Bi-CGSTAB : A fast and smoothly converging variant of bi-cg for the solution of nonsymmetric linear systems," *SIAM J. Sci.Stat.Comp.*, vol. 13, pp. 631–644, 1992.

[10] Y. Saad, *Iterative methods for sparse linear systems*. SIAM, 2003.

[11] E.N. Mathioudakis, N.D. Vilanakis, E.P. Papadopoulou and Y.G. Saridakis, "Parallel iterative solution of the hermite collocation equations on GPUs," *Proceedings of WCE 2013 Vol II, IAENG*, pp. 1281–1286, 2013.

[12] N.D. Vilanakis and E.N. Mathioudakis, "Parallel iterative solution of the hermite collocation equations on GPUs II," *Journal of Physics: Conference Series*, vol. 490(1), 2014.

[13] *http://www.mathworks.com*.

[14] *http://www.pgroup.com*.

[15] *http://www.nvidia.com/cuda*.

[16] Shane Cook, *CUDA Programming*. MK, 2013.