International Journal of Engineering, Mathematical and Physical Sciences
ISSN: 2517-9934
Vol:3, No:12, 2009

Semi-Lagrangian Method for Advection Equation on GPU in Unstructured $\Re^3$ Mesh for Fluid Dynamics Application

Irakli V. Gugushvili, Nickolay M. Evstigneev

***Abstract*—**Numerical integration of initial boundary problem for advection equation in $\Re^3$ is considered. The method used is conditionally stable semi-Lagrangian advection scheme with high order interpolation on unstructured mesh. In order to increase time step integration the BFECC method with limiter TVD correction is used. The method is adopted on parallel graphic processor unit environment using NVIDIA CUDA and applied in Navier-Stokes solver. It is shown that the calculation on NVIDIA GeForce 8800 GPU is 184 times faster than on one processor AMDX2 4800+ CPU. The method is extended to the incompressible fluid dynamics solver. Flow over a Cylinder for 3D case is compared to the experimental data.

***Keywords*—**Advection equations, CUDA technology, Flow over the 3D Cylinder, Incompressible Pressure Projection Solver, Parallel computation.

I. Governing equations

EQUATIONS of type

$$\frac{\partial F}{\partial t} + \nabla \cdot \vec{G} = 0, \qquad (1)$$

describe special conservative advection of the scalar property F. These equations are commonly used in incompressible and compressible fluid dynamics, free surface flows, shallow water models etc. Here $\vec{G}$ is the flux vector of the scalar function found as the product of the scalar function and the advection velocity vector function. In terms of scalar function – velocity function (1) can be rewritten as:

$$\frac{\partial F}{\partial t} + \vec{V} \cdot \nabla F - F \nabla \cdot \vec{V} = 0, \qquad (2)$$

here $\vec{V}$ - velocity vector-function. Only some analytical solutions exist for the simplest initial-boundary value problems in three dimensional space [1, p242]. In order to solve general type initial boundary value problem one must apply numerical methods for (2), especially in case of nonlinear relations (i.e. advection term in Navier-Stokes equations where F stands for conservative momentum vector). Using regular finite volume methods takes great usage of CPU time and memory, thus leading to parallel or distributed computational methods.

Irakli V. Gugushvili is the PhD student in the Institute for Hydraulics and Hydrodynamics, Moscow Russia and working in the dam breach department. (gugushvili_i@mail.ru)

Nickolay M. Evstigneev is the Doctor in Applied Mathematics and working as a leading scientist in the Institute for System Analysis, Russian Academy Of Science in the Nonlinear and Chaotic Dynamics Laboratory. (evstigneevnm@yandex.ru)

The work is supported by RFFI grant 08-07-00074a and 09-07-00078a alongside with program ONIT RAN (project 1.9).

II. Semi-Lagrangian method

The method is widely used in meteorology and detail description can be found, i.e. in [2, 4].

Let $\Omega \in \Re^3$ be time invariant (steady) arbitrary domain, bounded by $\partial\Omega \in \Re^2$. And let $\vec{V} : \Omega[0..t] \to \Re^3$ be initial velocity vector-function distribution. One must solve numerically the following initial-boundary value problem for (2):

$$F(0,\vec{x}) = F_0(\vec{x});\ F(t,\vec{x}|_{\partial\Omega 1}) = F1;\ \left.\frac{\partial F}{\partial \vec{n}}\right|_{\partial\Omega 2} = F2, \qquad (3)$$

Here F1 and F2 – arbitrary constants.

Let the domain Ω be meshed with 3-simplexes (tetrahedron) of N total number of elements. The problem (3) will be solved for the following form of equations (2):

$$\frac{dF}{dt} = 0;\ \frac{d\vec{x}}{dt} = \vec{V}(\vec{x},t);\ \vec{x} \in \Omega[0..N] \qquad (4)$$

The solution algorithm for (4) is written as follows:

1. Lagrange step. Every point in Ω, of the tetrahedron mass center with coordinates $\vec{x}_i$, is transferred to coordinates $\vec{x}^*{}_i$ for time moment t (previous time moment) in Ω by integrating the trajectory:

$$\vec{x}^*{}_i = \vec{x}_i - \int_t^{t+\Delta t} V(\tau) d\tau. \qquad (5)$$

2. Interpolation step. Determination of the element (localization in Ω) that contains the point with $\vec{x}^*{}_i$ coordinates and calculation of scalar function value F by interpolation between K-order neighbor elements:

$$F(\vec{x}^*{}_i, t) = \mathrm{I}(F\{\hat{\vec{x}}_{iK}\}, t), \qquad (6)$$

here I – interpolation operator; $\vec{\tilde{x}}_{iK}$ - neighbor coordinates up to the neighbors of K-th order.

3. Regularization step. Scalar field update on t+Δt time step:

$$F(\vec{x}_i, t+\Delta t) = F(\vec{x}^*_i, t). \quad (7)$$

The following procedure is monotonic (if TVD [9, 10] condition is applied), conservative (if local mass conservation is applied for every element [9]) and unconditionally stable because solution characteristic lines are always in the physical dependence domain. Various constructions of these steps exist, see [4], but here some modifications are used.

A. Trajectory integration

In order to determine the trajectory integral (5) the velocity vector-function values of either previous time step values or both time steps are used [2 ,5]. The explicit formula to define velocity function values on t and t+Δt time steps is used. So, one of the coordinate vectors (i.e. in x direction) can be found with the second order time integration as:

$$x^*_i - x_i = \int_t^{t+\Delta t} V_x(\tau)d\tau \approx \Delta t \frac{V_x(x^*_i, t) + V_x(x_i, (t+\Delta t))}{2}, \quad (8)$$

here V_x - velocity vector-function value in 'x' direction.

It is obvious, that x^*_i from (8) can be found only by solving system of algebraic linear equations. One should avoid solving SLAE for every element in Ω for it is too computationally expensive. To do so one can use Taylor series expansion near point x_i:

$$V_x(x^*_i, t) = V_x(x_i, t) + \nabla V_x(x_i, t)\cdot(x^*_i - x_i) + O(x^2), \quad (9)$$

and by doing so derive explicit formula for x^*_i:

$$x^*_i = x_i - \frac{\frac{\Delta t}{2}\left(V_x(x_i, t+\Delta t) + V_x(x_i, t) - \nabla V_x(x_i, t)\cdot x_i\right)}{\left[1 + \frac{\Delta t}{2}\nabla V_x(x_i, t)\right]}, \quad (10)$$

here: $\nabla V_x(x_i, t)$ - gradient of velocity in 'x' direction. All values in (10) are known on 't' time step, so it is possible to derive x^*_i explicitly.

The following procedure is second order time accurate in case of second or higher order gradient operator for velocity vector-function.

B. Element localization

The procedure of element localization is conducted very easy on structured meshes [4]. However it is very difficult to find a proper element on unstructured meshes for every element verification algorithm is $O(N^2)$ computationally expensive and it becomes merely impossible to use it on large element meshes. There exist numbers of advanced element localization methods that simply return element number on unstructured topology with coordinates $\{x^*_i; y^*_i; z^*_i\}$. Very good survey on these methods for pure Lagrangian integration is given in [4] where it is proved that the most efficient localization method is proposed in [3]. However, method [3] suffers from dead lock cycling if the trajectory intersects lower topological dimension primitives, i.e. planes and points; and the denser the mesh is, the higher is dead lock cycling probability. Leaving obvious topological explanations, authors made some changes to the algorithm [3], so that it doesn't suffer from the dead cycles for lower dimensional topological primitives.

The procedure contains the directional search algorithm that determents the simplex boundaries (with lower topological indexes) that trajectory intersects. It is done by locating the point of intersection, first between planes (triangles of a tetrahedron) and trajectory line, then between side lines (triangle sides) and trajectory line, and then between points (tetrahedron vertexes) and trajectory line. The algorithm is straightforward and uses geometrical relations, which can be found i.e. [4]. Further the algorithm travels through all elements that have trajectory line intersections and terminates when the traced element has no more intersections with the trajectory. The algorithm was found steady and robust with no dead lock cycling. Its computational cost can be estimated as O(A·C·N) , here C-number of elements that the trajectory intersects, A – number of simplex boundaries for intersection, which for the worst case of intersection in a tetrahedron equals 4(for triangles)+6(for sides)+4 (for vertexes). C can be estimated from the Courant–Friedrichs–Lewy condition for (4) as $C \approx V \cdot \Delta t / \Delta x$.

C. Interpolation

The Voronoi diagram in the system of tetrahedra is used for scalar-function interpolation shown on fig.1.

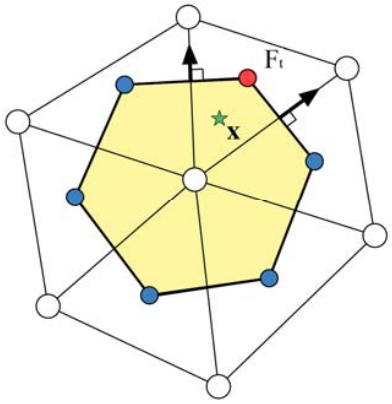

Fig. 1 Voronoi cell with six tetrahedra (two dimensional projection)

Voronoi cell vertexes are constructed using centers of mass in neighbor tetrahedra. Value of the scalar-function F_x in the 't' tetrahedron is estimated as the weighted sum of all Vronoi cell vertexes. Vertex weight with value F_t (fig.1) can be found as:

$$w_t(x) = \frac{|N_t|}{\prod_{f \in \sigma_t} \vec{n}_f x + d_f}, \quad (11)$$

with $$\sum_{t=1}^{m} w(x)_t = 1. \quad (12)$$

Here: σ_t - set of Voronoi cell faces, adjoined to the 't' vertex; $\vec{n}_f$ - face outward normal; x – distance from the point to the face; d_f - face plane shift; m – number of faces for a Voronoi cell; $|N_t|$ - determinant of the matrix, formed by all unit normals to σ_t.

If the tetrahedral grid is constructed using Delaunay algorithm [11] than the formula (11) can be simplified due to some spatial properties: 1) Vertex vectors of Delaunay mesh are always perpendicular to the Voronoi faces; 2) A tetrahedron volume is defined as 1/6det|E_t|, where E_t is the matrix, formed by three vectors of vertexes $\vec{e}_1, \vec{e}_2, \vec{e}_3$ with a base point among the tetrahedron vertexes. Then (11) can be rewritten as:

$$w_t(x) = \frac{|N_t|}{\prod_{f \in \sigma_t} \vec{n}_f (c_t - x)} = \frac{|\vec{e}_1 \| \vec{e}_2 \| \vec{e}_3 \| N_t|}{\prod_{f \in \sigma_t} |\vec{e}_f| \cdot \vec{n}_f (c_t - x)} = \frac{6Vol(t)}{\prod_{f \in \sigma_t} \vec{e}_f (c_t - x)}, \quad (13)$$

here Vol(t) is the volume of the tetrahedron 't'; $\vec{e}_f$ is the vector coming from the Voronoi cell to the other faces 't'; c_t is the mass center of the tetrahedron 't'; x is the point coordinate for interpolation.

After calculating $w_t(x)$ using (13) and, since (12) is true, F_x is defined as:

$$F(x_i) = \sum_{t=1}^{m} w_t(x_i) \cdot F_t, \quad (14)$$

which is used on regularization step. Boundary conditions (3) are taken into consideration on this step as well. For Dirichlet boundary conditions a virtual cell with a common topology is considered and a value of F1 (3) is set explicitly. For Neumann boundary conditions the face value in the denominator in (13) is changed for F2, see (3).

D. BFECC method with limiter

In order to have a high order monotonic solution a BFECC method is used, see [12]. The trajectory integration procedure as in (10) with values of $V_x(x_i,t)$ on a time step t is considered. And the solution procedure of the general problem (3) for equations (4) on tetrahedral mesh in Ω domain that consists of formulas (10), (14) and (7) as the operator $\Im$ is written as:

$$F^{t+\Delta t} = \Im(\vec{V}^t, F^t). \quad (15)$$

The operator finds value of scalar-function F on the time step t+Δt with the first order of accuracy in time. The interpolation operator (14) does not violate monotonic condition,, see [10, sec.4.4 p.257]. Thus, the procedure (15) is absolutely stable and monotonous and has a first order of accuracy in time.

The classical BFECC method [12] is not monotonous in time and can create spurious oscillations. In order to restrain monotonic condition a three stage TVD method is applied:

$$F^{t+n1} = \Im(\vec{V}^t, F^t), \quad (16)$$

$$F^{t+n2} = \Im((-\vec{V})^t, F^{t+n1}), \quad (17)$$

$$F^{t+\Delta t} = \Im\left(\vec{V}^t, [F^t + \frac{r(F^t - F^{t+n2})}{2}]\right), \quad (18)$$

here r is the limiting function slope [10,p.258], defined by the TVD condition.

III. Program adaptation on the Graphic Processor Unit

For dense meshes the listed above algorithm should be optimized for parallel computations. In this case one can expect to use lesser wall time for challenging problems with more than 10 millions elements. In this paper the method of using graphic processor unit as a parallel computational environment is used. NVIDIA CUDA [13] developers environment is used in this paper to solve generally posed problem (3) for equation (4) using algorithm (16)-(18). Procedure and implementation in the CUDA C++ language are similar to the procedure, listed in [8]. In accordance with the NVIDIA CUDA [13] programmer's guide, the program was written that uses algorithms (16)-(18) as kernels in GPU with utilization of the device shared memory. All pre- and postprocessor instructions are using CPU.

Algorithm, responsible for the operator projection (15), uses maximum time saving procedures. For that purpose a shared memory block is created and all calculations are conducted in the shared memory; this ought to speed up the GPU calculations greatly as stated in the programmer's guide [13, p.55]. In order to minimize the ratio of global memory/shared memory reloads, all nested tetrahedra are supposed to be close to each other. For that purpose all elements are regrouped in global memory of GPU before the time loop, thus grouping neighboring elements close to each other in the global device memory. This procedure is $O(N^{1,5})$ computationally expensive and can be used while a mesh is loading. The indexing of the kernel uses single unsigned integer over the whole domain that represents the element number in the mesh geometric relation array. This index is parallelized on maximum number of threads for the specific device used. So all the cycle equations (16)-(18) are calculated in the GPU with no GPU-CPU memory reloads during the calculation in the time loop. The results are loaded from the GPU to CPU memory only when the last timestep is finished. After that all memory in the device is released and all post processing work is conducted on the CPU. The GPU adapter,

which is used in this work is NVIDIA GeForce 8800 GTX with 768MB of device RAM that is installed on the AMD Athlon 64X2 4800+ with 4GB of RAM.

IV. Test Cases and Efficiency Estimations

For the test case of the advection equation on GPU we are considering the problem (3) for equation (4) where the scalar function is given as an initial condition and a vector function is given as a constant value. Two problems are considered – step function and sinus and elliptic half periods in $\Omega \in \Re^3$. The results are presented on fig2. for the single step function and on fig.3 for the sinus and elliptic half period. The used CFL number for all test cases is 3, number of tetrahedral elements in the mesh is 40 000, the grid is not adaptive and the ratio of mean element volume to it's mean verge plane square is 0.23mm. All tests are using 30 timesteps for the positive value of the velocity vector function distribution and 30 timesteps for the negative velocity vector-function value distribution to advect the initial given value of the F scalar function.

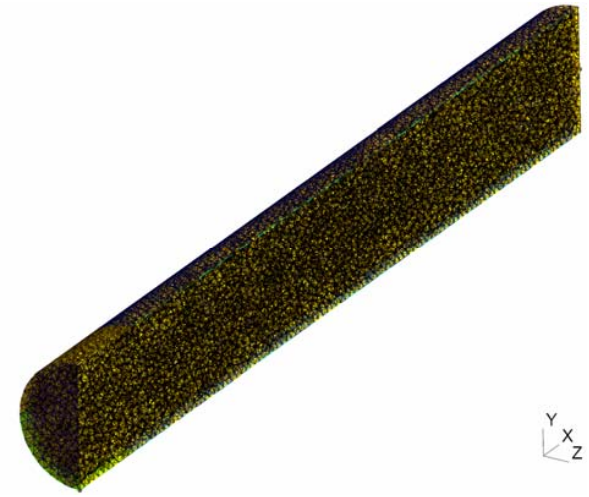

Fig. 2. Mesh for the test case with the plane section.

The results show that the method has small amount of the artificial viscosity. Comparison of the initial condition with the step function gave a blur of 3.2% for 60 timesteps. The results for the sinus and ellipse cases show some decay of the sinus wave amplitude – this can be seen in the fig5 with the cross plane section data plot. It happens due to the limiter operation that imposes the TVD monotonic condition. Other results show good correlation with the initial conditions, especially for ellipse case and the step case.

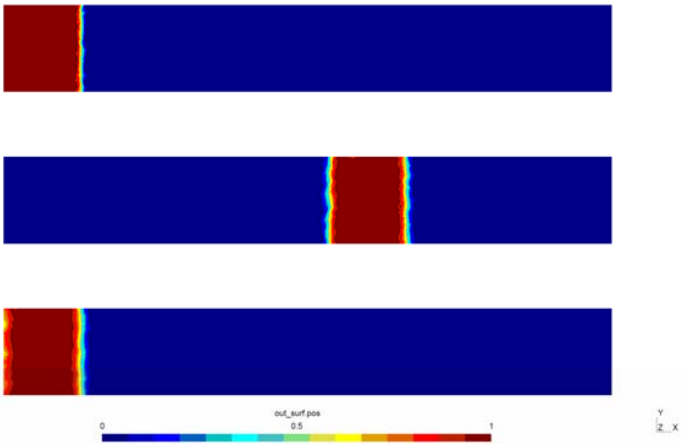

Fig. 3. Step problem test case. Results are shown on the middle plane. Top – initial condition; middle – 30 timeseps with positive velocity; buttom – 30 timesteps with negative velocity.

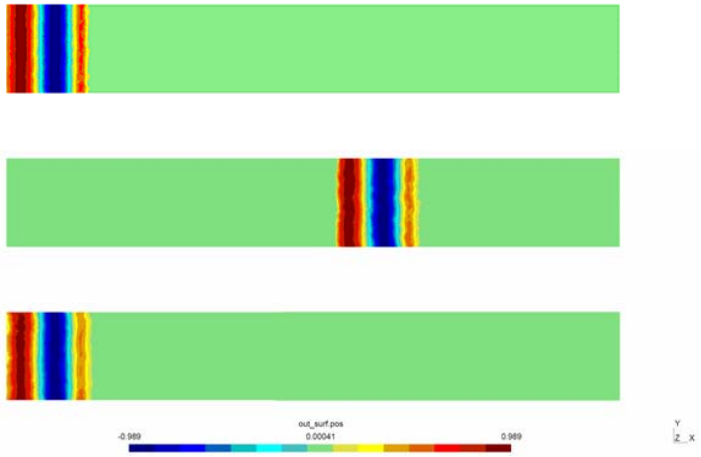

Fig. 4. Sinus and ellipse test case. Results are shown on the middle plane. Top – initial condition; middle – 30 timeseps with positive velocity; buttom – 30 timesteps with negative *velocity.*

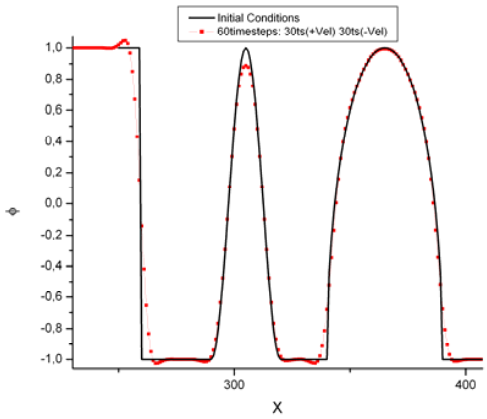

Fig. 5. Sinus and ellipse test case-initial conditions vs. time integration for 60 timesteps. {X;0.5;0.5}-plane section.

Comparison of efficiency was conducted for these test cases on GPU and CPU algorithms. CPU version of the code was executed on the serial mode with one core operating on the CPU. The results are close to the ones, obtained in the other work for Poisson equation in [8] – the computer speed increases to 184 times as fast as the CPU version from the mesh of $1 \cdot 10^6$ elements and more.

V. Fluid Dynamics Extension

The method, described in the paper was extended to incompressible fluid dynamics solver, using pressure projection algorithm on the unstructured tetrahedral mesh. The governing equations are conservation PDEs of mass and momentum – the Navier-Stokes equations:

$$\begin{cases} \nabla \cdot \vec{V} = 0 \\ \partial \vec{V} / \partial t + (\vec{V} \cdot \nabla)\vec{V} + \nabla P = R^{-1} \nabla^2 \vec{V} \end{cases} \quad (19)$$

here P is a pressure scalar function in $\Omega x(0..t)$.

The solution procedure of (19) consists of fractural step method on physical processes as follows:

1. Advection stage, using (16)-(18) equations:

$$v_i^{t+n1} = \Im(\vec{V}^t, v_i^t), i = \{1...3\}, \quad (20)$$

$$v_i^{t+n2} = \Im((-\vec{V})^t, v_i^{t+n1}), \ i = \{1...3\}, \quad (21)$$

$$v_i^{t+a} = \Im\left(\vec{V}^t, [v_i^t + \frac{r(v_i^t - v_i^{t+n2})}{2}]\right) i = \{1...3\}, \quad (22)$$

where v_i is the velocity vector-function cartesian component and $\Im()$ is the advection operator, described in (8)-(14). Internal fractural time steps are: 't' for the previous timestep; 't+a' – advection timestep; 't+d' – diffusion timestep; 't+p' – pressure timestep and 't+Δt' for the final timestep.

2. Diffusion step, using finite volume method with the second order of accuracy:

$$v_i^{t+d} = v_i^{t+a} + \Delta t \cdot R^{-1} \cdot \Theta(v_i^{t+\sigma}), \quad (23)$$

where R is the Reynolds number and $\Theta()$ is the diffusion operator, see [9]. σ is a implicit (d) or explicit (a) method switch. For R>100 it is better to use an explicit scheme. Equation (23) represents the diffusion equation of the form $v_i^{t+d} = v_i^{t+a} + \Delta t \cdot R^{-1} \cdot \nabla^2 v_i^{t+\sigma}$.

3. Pressure step that consists of the Poisson equation $\nabla^2 P = -\nabla \cdot \vec{V}^{t+d}$, using finite volume method and conjugate gradient method to solve the system of algebraic equations:

$$\Theta(P) = -\nabla \cdot \vec{V}^{t+d}, \quad (24)$$

where divergence term on the right evaluates using staged finite volume formulation.

4. Pressure projection step that corrects velocity vector-function using solenoidal pressure solution in the form $\partial \vec{V} / \partial t = \nabla P$ is written as:

$$\vec{V}^{t+\Delta t} = \vec{V}^{t+d} - \Delta t \cdot \nabla P. \quad (25)$$

All 1-4 steps are unconditionally stable if d stands for σ in (23).

System of (20)-(25) is added with the proper class of boundary and initial conditions for a general type of the problem, see [8] and [9] for more details[1].

The classical problem of 3D flow past the cylinder was considered to verify the numerical method and to estimate accuracy.

The adaptive mesh consists of 121 000 elements is shown on fig.6 with literal plane cut.

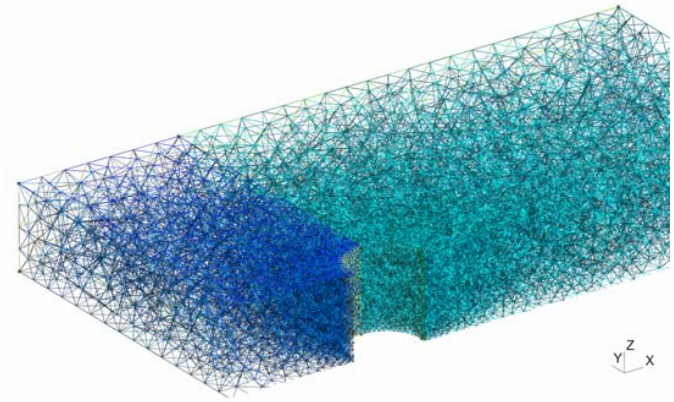

Fig. 6. 3D Cylinder flow problem mesh – literal plane section.

Some calculations were concluded to get the laminar, transient and turbulent flow regimes. To take turbulent regime into account, the large eddy simulation strategy was incorporated. The LES model was developed by one of the authors can be classified as the dynamic eddy viscosity subgrid model with implicit filtering. More details on LES models can be found in [9].

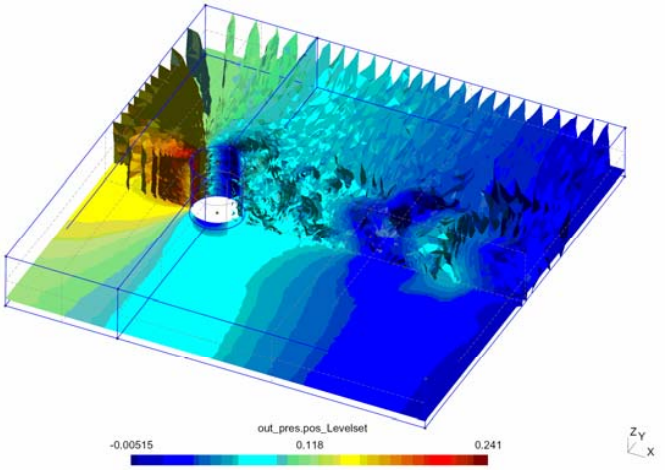

Fig.7 Pressure instantaneous isasurfaces for R=5000

[1] Some care must be taken for the pressure wall boundary conditions because those are not physical. See [6,9] for details.

Fig. 7 shows instantaneous pressure isasurfaces plots and fig.8 presents instantaneous velocity vectors in a close-up view for R equals 5000 in (19).

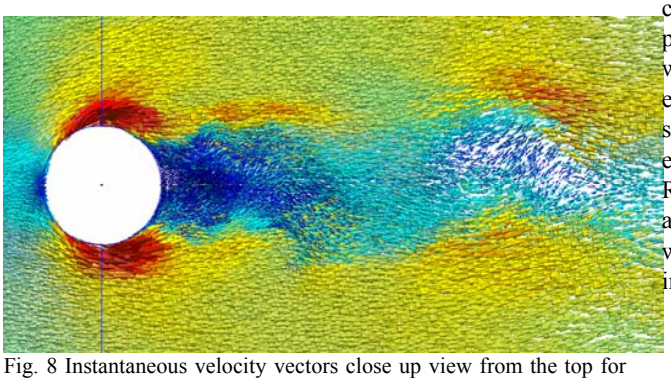

Fig. 8 Instantaneous velocity vectors close up view from the top for R=5000. Scale starts from 2.256 for the red color.

Averaged over the Z axis variations of the time mean drag coefficient C_d are plotted on fig.9 to verify the solver. For that purpose several test cases with different Reynolds numbers were conducted for the flow over the cylinder. The experimental results are taken from [14]. It can be clearly seen that the solver predicts drag coefficient very good with detail representation of the drag crisis regimes for relatively high Reynolds numbers and drag partial recovery for even greater R numbers.

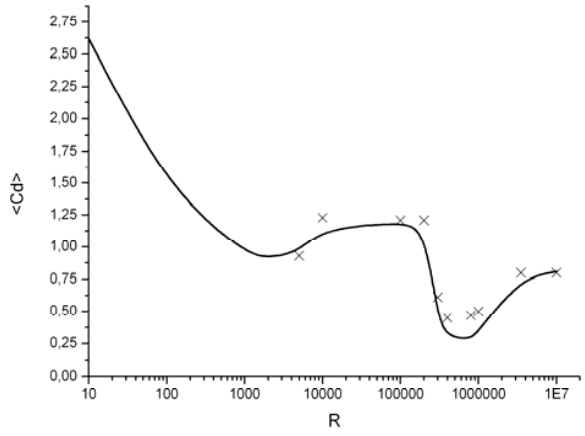

Fig. 9 Averaged over the Z axis variations of the time mean drag coefficient C_d. Line – experiment from [14], Crosses – numerical results.

VI. Conclusion

A semi Lagrangian method for advection equations in three dimensional Euclidian space is presented in the paper. The method uses some novel interpolation and limiting techniques. This guaranties it's monotonic and conservation properties for the advecting scalar function. The method is developed as the parallel environment for the graphic processor units using NVIDIA CUDA programming language. Advection property test cases confirmed that the method is conservative, monotonic and has a high order of approximation in time and space. GPU implementation guarantees a significant speed up of overall 184 times compared to the single processor calculations on the CPU. This allows solving complex problems on cheap and freely available personal computers with the proper GPU video card installed. The method is extended to incompressible pressure projection fluid dynamics solver. It is used in the advection part of the Navier-Stokes equations. The test case of the flow past a cylinder for high Reynolds numbers shown that the method is correct and accurate. To model the turbulent flow a LES methodology was incorporated. In future the described method will be used in the computational complex, developed by the author.

Acknowledgment

The second author wishes to thanks his wife, Marina, for valuable corrections introduced.

References

[1] A D Polyanin; V F Zaitsev. Handbook of Nonlinear Partial Differential Equations. – Chapman & Hall/CRC Press, Boca Raton, 2003.
[2] Douglas Enright, Frank Losasso, Ronald Fedkiw. A Fast and Accurate Semi-Lagrangian Particle Level Set Method. // Proceedings of the 4th ASME-JSME Joint Fluids Engineering Conference, number FEDSM2003, 45144. ASME, 2003.
[3] Chorda R, Blasco J.A., Fueyo N. An efficient particle-locating algorithm for application in arbitrary 2D and 3D grids// Int. J. of Multiphase Flow, 28, 2002 N9, 1565-1580.
[4] Volkov K.N., Emelyanov V.N. Implementation of the Lagrangian approach to the description of gas-particle flows on unstructured meshes.// J. Numerical methods and programming. Vol9, pp. 19-33, 2008.
[5] Paoliy R. , Poinsotz T., Shari K. Testing semi-Lagrangian schemes for two-phase flow applications.// Proceedings of the Summer Program 2006, pp213-222. Center for Turbulence Research, Toulouse, France.
[6] Chunlei Liang, Evstigneev N., A study of kinetic energy conserving scheme using finite volume collocated grid for LES of a channel flow. // Proceedings of the international conference on numerical methods in fluid dynamics. King's College London, Strand, WC2R 2LS, 2006, pp.61-79.
[7] Evstigneev N.M., Magnitskii N.A., Sidorov S.V. On the nature of turbulence flow in the backward face step problem // J. Differential equations, Vol.45, 2009, pp.69-73.
[8] Evstigneev N.M. Numerical integration of Poisson's equation using a graphics processing unit with CUDA-technology // J. Numerical methods and programming., Vol10, pp. 268-274, 2009.
[9] Evstigneev N.M., Magnitskii N.A., Sidorov S.V. New approach to the incompressible flow turbulence. // Proc. ISA RAS, Vol33, pp.49-65, 2008.
[10] Evstigneev N.M. Solution of 3D nonviscous compressible gas equations on unstructured meshes using the distributed computing approach. // J. Numerical methods and programming., Vol8, pp. 252-264, 2007.
[11] Cignoni P., Montani C., Scopigno R., Dewall: A fast divide & conquer Delaunay triangulation algorithm in Ed // Computer J. 2006. 19, No2, pp 178-181.
[12] T. F. Dupont, Y. Liu. Back and forth error compensation and correction methods for removing errors induced by uneven gradients of the level set function. // Journal of Computational Physics, vol. 190, no. 1, pp. 311–324, 2003.
[13] http://developer.download.nvidia.com/compute/cuda/2_0/docs/NVIDIA_CUDA_Programm ing_Guide_2.0.pdf
[14] A. Roshko. Experiments on the ow past a circular cylinder at very high Reynolds number.// Journal of Fluid Mechanics, 10:345-356, 1961.