Available online at www.sciencedirect.com
SCIENCE DIRECT®
Journal of Computational Physics 199 (2004) 423–436
JOURNAL OF COMPUTATIONAL PHYSICS
www.elsevier.com/locate/jcp

A three-dimensional adaptive method based on the iterative grid redistribution

Desheng Wang [a], Xiao-Ping Wang [b,*]

[a] Department of Mathematics, The University of Xiangtan, Hunan, PR China
[b] Department of Mathematics, The Hong Kong University of Science and Technology, Clear Water Bay, Kowloon, Hong Kong

Received 14 October 2003; received in revised form 16 February 2004; accepted 19 February 2004
Available online 19 March 2004

Abstract

In this paper, we develop a three-dimensional adaptive method based on iterative grid redistribution technique introduced in [J. Comput. Phys. 159 (2000) 246]. The key step for the successful implementation is a fast algorithm for grid generation, which is composed of solving the linear grid equation systems and an efficient method for inverting a map by computing iso-surface intersections. To carry out the three-dimensional calculations, the whole procedure is also parallelized on a PC cluster. The improved and extended three-dimensional adaptive method is applied to solve PDEs with singular solutions that have three-dimensional structures. Numerical experiments have demonstrated the method's effectiveness.

1. Introduction

There are many physical phenomena that develop singular or nearly singular behavior in a localized regions. The numerical solutions of these problems require extremely fine mesh to resolve accurately the solution in small regions. The use of well-refined uniform meshes becomes computationally prohibitive when dealing with systems in multi-dimensions. Various adaptive methods are developed for this type of problems. Mesh adaptivity is usually in the form of local mesh refinements or through mesh mapping. For the numerical solutions of many problems from areas such as fluid dynamics, combustion and heat transfer and others, the required density of grid points is determined in general by the solution's gradients. Large gradient require a high grid-point density in order to increase the accuracy and at the same time decreases the cost of numerical calculations in comparison with the uniform grid, as is called adaptive computation. Various solution-adaptive methods which eliminates the need to have a priori qualitative estimate of the solution have been developed towards the above end, such as local adaptive mesh refinement [2], moving

* Corresponding author.
E-mail address: mawang@ust.hk (X.-P. Wang).

0021-9991/$ - see front matter © 2004 Elsevier Inc. All rights reserved.
doi:10.1016/j.jcp.2004.02.014

finite elements mesh refinement [15,16], adaptive node movement (see, e.g. [1,4,6,9,12] and references therein), or methods based on attraction and repulsion pseudo-forces between nodes [18]. In order to handle singular problems (in particular blow-up solutions) more effectively, Ren and Wang [19] introduced the iterative grid redistribution (IGR) method to give an effective control of grid density near the region of large solution variation. The method has been successfully applied to many problems with singular behavior [7,19].

In this paper, we make several modifications and improvements on the adaptive method introduced in [19], so that it is much more efficient and capable of handling three-dimensional problems that are of practical applications. In the adaptive method with IGR for time dependent PDEs, there are three main components: (1) a grid generation rule, (2) an iterative procedure and (3) an adaptive time integration of the underline PDE. In [19], grid generation is done by variational approach which results in a system of elliptic system. The solution of this system is then obtained by heat flow. Although affordable in two dimensions, it is too expensive for three-dimensional applications. The computational cost for time integration of the underline PDE in three-dimensional is further increased by the fact the number of terms generated from the chain rule (when calculating derivatives in the computational variables) increases exponentially with the order of the derivatives. Therefore, in order to solve a three-dimensional singular problem within a reasonable CPU time, it is necessary to make both the methods for grid generation and the time integration of the PDE more efficient.

For efficient grid generation, we introduce a fast algorithm which is based on an intuition that the curvilinear coordinates or equivalently the grid points are just the intersection points of the iso-surfaces (contour lines in two-dimensional) of the map $\xi(\mathbf{x})$ from the physical domain to the computational domain, which can be obtained easily from a linear decoupled elliptic system. Based on this intuition, we develop a fast algorithm consisting of solving the linear system for $\xi(\mathbf{x})$, constructing the iso-surfaces and finding the intersections. This direct grid generation method not only is very efficient but also eliminates the convergence and stability issues for the existing methods for solving the grid system [8,10,17, 20]. Furthermore, it can be parallelized effectively.

To speed up the three-dimensional computations, we parallelize the whole procedure. Since our time integration of PDE is explicit, parallelization can be done effectively with domain decompositions. With a finite difference scheme in space, message passing is only needed near the boundary of each sub-domain. This enables us to achieve a very good parallel efficiency. Our numerical tests show that the speed-up of the parallelization is almost linear to the processor numbers.

With these improvements, we are able to build up a three-dimensional parallel (in MPI) solver. The solver is then applied to several problems with three-dimensional singular structures. In the following, we first recall the iterative grid redistribution method in Section 2. The fast algorithm for grid generations and the parallelization procedure are described in Sections 3 and 4. The numerical examples of nonlinear Schrödinger equation and Keller–Seigel equation are given in Section 5.

2. Adaptive method based on iterative grid redistribution

In this section, we recall the IGR method introduced in [19]. The method consists of the following three parts:

(1) A grid generation rule that determines the mesh mapping $\mathbf{x} = T(\xi)$.
(2) An iterative procedure that controls the grid distribution near the singular points.
(3) A procedure for solving PDEs.

The step (2) is the key for the method to be successful for the problem with singular behavior. It is a procedure that improves the grid distribution near singular region if the mapping T in step (1) cannot achieve enough resolution in the singular region. To see this, let us assume that the mesh mapping T in (1)

has the tendency to move the grids toward a point $\mathbf{x}_0$ in the domain as in Fig. 1 where $\mathbf{x}_0$ is the center of the domain. Then the grid points will continue to move toward $\mathbf{x}_0$ as we iterate the mapping T. This gives certain control of the density of the grid points near the point $\mathbf{x}_0$ and therefore improves over step (1). We now explain the three steps in detail.

2.1. Grid distribution based on the variational principle

We are concerned with generating a mesh that adapted to the solution $u(\mathbf{x})$ (at a given time) to a time dependent PDEs. In two and three spatial dimensions, mesh generation is commonly done using the variational approach, specifically by minimizing a functional of the coordinate mapping between the physical domain and the computational domain. The functional is chosen so that the minimum is suitably influenced by the desired properties of the solution of the PDE itself.

Let $\mathbf{x}$ and ξ denote the physical and computational coordinates, respectively, on a domain $\Omega \in \mathbf{R}^d$. A one-to-one coordinate transformation on Ω is denoted by

$$\mathbf{x} = \mathbf{x}(\xi), \quad \xi \in \Omega. \tag{2.1}$$

Here the grid on the computational domain ξ is uniform and the corresponding distribution gives the grid distribution in the physical domain. The functionals used in existing variational approaches for mesh generation and adaptation can usually be expressed in the form

$$E(\xi) = \int_{\Omega} \sum_{i,j,\alpha,\beta} g^{i,j} \frac{\partial \xi^{\alpha}}{\partial x^i} \frac{\partial \xi^{\beta}}{\partial x^j} \, \mathrm{d}\mathbf{x}, \tag{2.2}$$

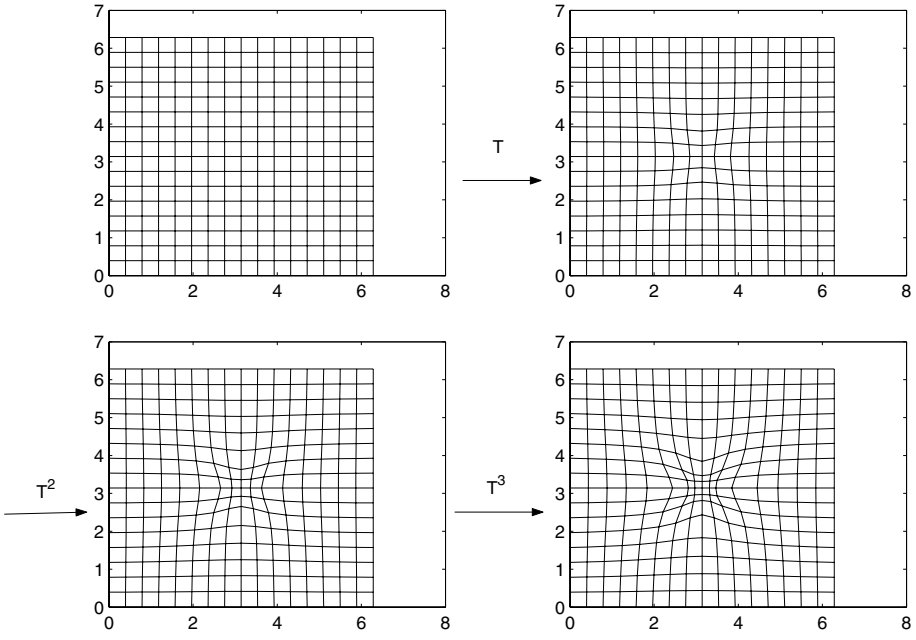

Fig. 1. The effect of map iterations.

where $G = (g_{i,j})$, $G^{-1} = (g^{i,j})$ are symmetric positive definite matrices that are monitor functions in a matrix form. Here $g^{i,j}$ depends on the underline solution $u(\mathbf{x})$ of the PDEs. The coordinate transformation and the mesh are determined from the Euler–Lagrange equation

$$\nabla \cdot (G^{-1}\nabla \xi) = 0. \tag{2.3}$$

In most of our cases, a diagonal matrix G is good enough for our purposes. One possible choice is $g_{i,j} = (\sqrt{1 + |\nabla u|^2})\delta_{i,j}$. In this case, (2.3) is a decoupled system in three components. We note that more terms can be added to the functional (2.2) to control other properties of the mesh, such as orthogonality of the mesh and the alignment of the mesh lines with a prescribed vector field [4].

2.2. An iterative grid redistribution procedure

As illustrated in [19], grid generation using the above variational approach may not produce satisfactory mesh resolution when the solution $u(\mathbf{x})$ is near singular. An iteration procedure is introduced in [19] to improve the grid distribution in such cases.

Let us first define the *mapping*

$$\mathbf{T} : (\mathbf{x}, u(\mathbf{x})) \to (\xi, v(\xi)) = (\xi, u(\mathbf{x}(\xi))).$$

Here $\mathbf{x} = \mathbf{x}(\xi)$ is determined from (2.3) with a monitor matrix involving $u(\mathbf{x})$. Therefore, the mapping from $\mathbf{x}$ to ξ is determined by u first, and u is then mapped to v by the change of variables.

If the monitor matrix G is properly chosen, the resulting mesh should concentrate more grid points in the regions with large variations. If the resolution in the singular region is not good enough, we iterate the mapping $\mathbf{T}$ as in Fig. 1. From the definition of $\mathbf{T}$, we note that $\mathbf{x} \to \xi$ is u dependent; therefore, the grid mapping is different in each iteration since u itself is iterated as we iterate T.

- Let $u^k(\mathbf{x}^k)$ be the function after k iterations.
- Determine the mapping $\mathbf{x}^k(\mathbf{x}^{k+1})$ from $u^k(\mathbf{x}^k)$ according to (2.3) where monitor matrix G^k is defined using $u^k(x^k)$.
- Define $u^{k+1}(\mathbf{x}^{k+1}) := u^k(\mathbf{x}^k(\mathbf{x}^{k+1}))$.

For example, after two iterations, we have

$$(\mathbf{x}, u(\mathbf{x})) \to^T (\xi_1, v_1(\xi_1)) \to^T (\xi_2, v_2(\xi_2)).$$

In the first iteration, we determine a grid mapping $\mathbf{x}(\xi_1)$ and $v_1(\xi_1) = u(\mathbf{x}(\xi_1))$. In the second iteration, we have $\xi_1(\xi_2)$ and $v_2(\xi_2) = v_1(\xi_1(\xi_2))$.

The result of the iteration is to flatten out the monitor function gradually. In fact, if $u^k(\mathbf{x})$ and $\mathbf{x}^k(\xi)$ converge, then we must have $\mathbf{x}^k \to \mathbf{x}^*(\xi) = \xi$ and $u^k \to u^*(\mathbf{x})$ [19].

2.3. Adaptive procedure for solving PDEs

We now incorporate the iterative remeshing into a static adaptive method for solving PDEs (see [19] for details). Consider a PDE of the form

$$\psi_t = F(\mathbf{x}, \psi, D\psi, D^2\psi), \quad \mathbf{x} \in \Omega_p \tag{2.4}$$

supplemented with initial and boundary conditions.

Our adaptive procedure is as follows:

(0) Given an initial condition $\psi(\mathbf{x}, t = 0)$, the initial grid transform $\mathbf{x}(\xi)$ is determined from the iterative remeshing, which in turn gives an initial condition in the computational domain $\psi(\mathbf{x}(\xi), t = 0)$.

(1) Solve the PDE in the computational variable ξ with the grid transformation $\mathbf{x}(\xi)$ being fixed, until some t^* when the solution $\psi(\mathbf{x}(\xi), t^*)$ cannot meet a certain criterion set in terms of computational variables.
(2) Generate a new mesh by the iterative remeshing, based on $\psi(\mathbf{x}(\xi), t^*)$. The remeshing iteration stops if the criterion in (1) is satisfied. Interpolation is used to generate the solution at the new grid points.
(3) Go to (1) to continue the integration.

Eq. (2.4) will be transformed into computational variables (ψ, ξ) and discretized in the computational domain. It was pointed out in [7] that the usual central difference scheme has lower accuracy on non-uniform grid than that on the uniform grid. A modified discretization scheme was introduced in [7] which has higher accuracy.

3. Fast algorithm for grid generations

Grid generation is to obtain the curvilinear coordinate system $\mathbf{x} = \mathbf{x}(\xi)$ (for ξ on a uniform grid) from the above elliptic system (2.3)). However, (2.3) is a linear, decoupled system for the inverse map $\xi(\mathbf{x})$. Most of the existing algorithms begin with transforming (2.3) into a set of coupled quasi-linear elliptic equations in $\mathbf{x}(\xi)$ then solve the derived quasi-linear elliptic equations by either iteration methods (e.g. SOR, FLAGG, Multigrid [5,10,17]) or by heat flow [9]. In 2D, the iteration cost is usually affordable. But in three-dimensional, especially in the case of highly complex geometric boundaries and a large number of computational points, the CPU time necessary to generate an acceptable grid using these solvers can be excessively high. When coupled with the underline PDE solver, the percentage of CPU time spent generating the grid could equal, or even exceed that for solving the underline PDEs.

An alternative approach is to solve the linear system (2.3) for $\xi(\mathbf{x})$ (for $\mathbf{x}$ on a uniform grid), which can be done with standard linear solver, then find the inverse map to obtain $\mathbf{x}(\xi)$. The trade off is that the inversion of the map $\xi(\mathbf{x})$ can be expensive. In [8], an iterative method is used to invert the map $\mathbf{x}(\xi)$. However, convergence of the map is problem dependent. In this paper, we propose an efficient, direct method (without iteration) for inverting the map after $\xi(\mathbf{x})$ is obtained from (2.3).

To simplify the discussion, let us consider the two-dimensional case and assume that we solve (2.3) by a conjugate gradient method on a quadrilateral grid denoted by $\mathbf{Q}$. We then obtain a triangle finite element mesh $\mathbf{T}$ from $\mathbf{Q}$ through bisecting each quadrilateral element. Let us assume that we have an uniform grid in the computational domain ξ with grid points (ξ_i, η_i). We are looking for the corresponding grid points (x_i, y_i) in the physical domain. Let K be a triangle in the finite element mesh (physical domain) with three vertices A, B, C (Fig. 2) whose corresponding (ξ, η) values are $(\xi_A, \eta_A), (\xi_B, \eta_B), (\xi_C, \eta_C)$. Denote the extreme values of them by $\xi_{\min}, \xi_{\max}, \eta_{\min}, \eta_{\max}$. First, we find the values $\{\xi_{i1}, \xi_{i2}, \ldots, \xi_{ik}\}$ which fall between $\xi_{\min}$ and $\xi_{\max}$. For each value(say ξ_{i1}), we apply linear interpolation to find the linear segment which is an approximate for the contour $\xi = \xi_{i1}$. The contours are shown in (1) of Fig. 2 by the thick segments S_1, S_2. The η contours are shown in (2) of Fig. 2 by the thin segments T_1, T_2. The intersection points $(g_1, g_2, g_3, g_4$ in (3) of Fig. 2) of the ξ and η contour lines are the grid points in the physical domain that fall into triangle K.

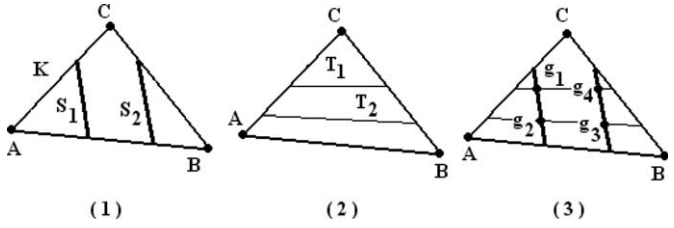

Fig. 2. Contours construction and intersection computation for grid points.

Clearly, the above procedure requires only O(1) computations which is optimal. Various tests have shown that the time for intersection computing is less than 10% of the time for solving the elliptic equations. Obviously this method is much more efficient than the indirect method that we mention earlier. Our numerical examples in the Section 5 show that the CPU time for the grid generation is only a very small part (usually is less than 1/10) of the CPU time for the whole computations. Another advantage of the direct method (over iterative methods) is that there is no convergence or stability issues involved.

Our direct method is also parallel friendly. Contour line constructions and intersection computations can be carried out in each sub-domain in a parallel manner with no communication cost.

4. Parallelization of the whole adaptation process

To speed up the computation, we parallelize the entire adaptive solution process. Since we use explicit scheme for time integration of the underline PDE on uniform mesh in the computational variables, a direct parallelization with domain decomposition is effective in speeding up the computations. The iterative grid generation part can also be parallelized effectively.

4.1. Domain decomposition and message passing scheme

For simplicity, we describe the domain decomposition scheme in two dimensions. Denote the number of processors allocated in our MPI program to be N_{PR}, and we divide the two-dimensional computational domain $[0,1]_2$ into $NP \times NQ$ sub-domains, where $N_{\mathrm{PR}} = NP \times NQ$. The following domain decomposition scheme is used which is also illustrated in Fig. 3.

(1) We divide an unit cube into equal or almost equal grid points in each direction and accordingly make each processor with almost equal computational load.
(2) There is one layer of grid points overlapped between two neighboring sub-domains (see Fig. 3) and the message passing is performed between these overlapped points. One layer overlapping is enough for our applications with second order PDEs.

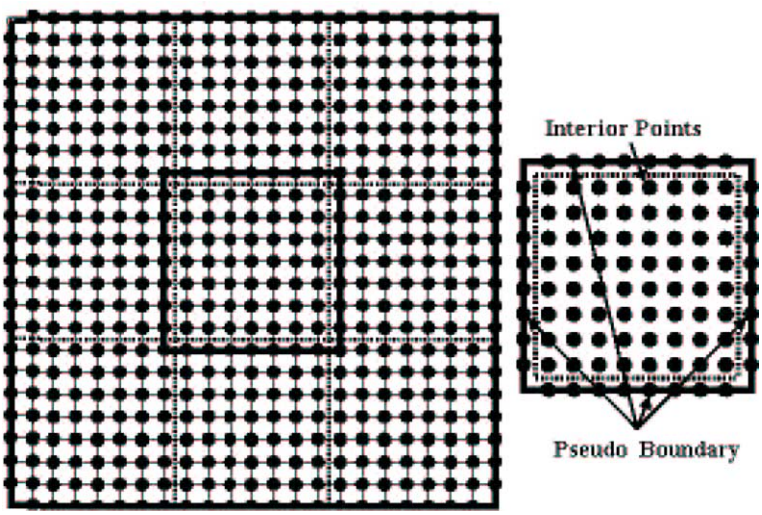

Fig. 3. Domain decomposition.

(3) Each sub-domain corresponds to a processor and the computation is only performed in parallel manner on the interior grid points of each sub-domain, the pseudo-boundary points are for message passing.

After the decomposition, the usual sequential computation is done on each sub-domain. When each iteration such as CG iteration or time integration was done, we need to update the values of each sub-domain's pseudo-boundary points through message passing. The message passing is done via one direction at a time with blocking passing format, which can guarantee that all the pseudo-boundary points' values can be updated successfully. Under special hardware environment, we can also use non-blocking message passing method to overlap the computation and the message passing (see [13] for details).

4.2. Parallelization of conjugate gradient method for the grid equations

Parallelization can be done for any existing conjugate gradient method (CGM), which involves only vector multiplications and inner product to determine parameters. The domain decomposition and message passing scheme can be chosen as described above. For vector multiplication, the natural parallelization is to perform each sub-domain's interior points multiplication. The computation of the parameters is done with some global reduce operations [13]. After each CG iteration, a message passing is performed to update each sub-domain's pseudo-boundary values.

4.3. Parallelization of iso-surfaces construction and the intersection computation

As there is no message passing required in the construction of the iso-surfaces and the determination of their intersections, the parallelization is straight forward and the speed up is optimal. The grid points are numbered lexicographically, a one-dimensional domain decomposition is enough for the parallelization. We divide the grid elements into N_{PR} (# of processors) almost equal domains. We then perform the iso-surfaces construction by linear interpolation and compute the intersections of these iso-surfaces in each domain as described in Section 3.

4.4. Parallelization of the time integration

For simplicity, we assume that, after finite difference approximation, the underline PDE has the form

$$U_t = F(U),$$

where $F(U)$ is a function of U. We apply a second-order Runge–Kutta scheme for time integration. Let Δt be the time-step size and initialize U^{old} by U^0, the two step Runge–Kutta scheme is as follows:

Step 1: $U^{\mathrm{mid}} = U^{\mathrm{old}} + \frac{\Delta t}{2} F(U^{\mathrm{old}})$.

Step 2: $U^{\mathrm{new}} = U^{\mathrm{old}} + \Delta t F(U^{\mathrm{mid}})$.

To parallelize the procedure, we use the domain decomposition and the message-passing as described before. The pseudo-code form of the parallelization is as follows:

0. Initialization U^{old} by U^0 for each sub-domain in parallel.
1. Perform *Step 1* in each sub-domain in parallel manner.
2. Perform message passing to update U^{mid} values at each sub-domain's pseudo-boundary points.
3. Perform *Step 2* in each sub-domain in parallel manner.
4. Perform message passing to update U^{new} values at each sub-domain's pseudo-boundary points. Let $U^{\mathrm{old}} = U^{\mathrm{new}}$ and goto 1.

In the above parallel computation, the message passing is kept at minimum and our numerical statistics show that the speed-up can get up to 0.75 which shows a good parallel efficiency.

5. Numerical examples

5.1. Nonlinear Schrödinger equation: point singularity

We first solve the nonlinear Schrödinger equation (NLS)

$$\mathrm{i}\psi_t + \Delta\psi + |\psi|^2\psi = 0, \quad (x,y,z) \in \Omega_p,\ t > 0,$$

$$\psi(x,y,z,t)|_{\partial\Omega_p} = 0$$

with cubic nonlinearity in three dimensions. The singular solutions with one blow up point were successfully computed using the dynamic re-scaling method first developed in [14] for the radially symmetric case and generalized in two and three dimensions in [11]. Later, Ren and Wang [19] applied the iterative grid redistribution to solutions with multiple point singularities in two dimensions. As a first test example of our three-dimensional method, we shall compute the singular solution with the single blow up point in three dimensions.

The initial condition is set to be

$$\psi(x,y,z,0) = 8.5\mathrm{e}^{-4(x^2+y^2+z^2)}.$$

The physical domain Ω_p is chosen to be $[-2,2]^3$, while the computational domain is chosen to be $[0,1]^3$.

In grid generation, the monitor function is taken to be

$$w(\xi,\eta,\zeta) = \left(1 + \alpha\frac{|\psi(\xi,\eta,\zeta)|}{|\psi|_{\max}} + \beta\frac{|\nabla\psi(\xi,\eta,\zeta)|}{|\nabla\psi|_{\max}}\right).$$

The remeshing criterion is set so that

$$\frac{|\nabla\psi(\xi,\eta,\zeta)|}{|\psi|_{\max}} < \mathrm{TOL}.$$

Here we choose $\mathrm{TOL} = 6$ and let $\alpha = \beta = 1.0$. The time step size is chosen to satisfy CFL condition $\Delta t = 0.01h_{\min}^2$ where $h_{\min}$ is the minimal size of the current grid elements.

The above three-dimensional NLS equation is solved on Ω_c with $80 \times 80 \times 80$ grid points in parallel programs. About 29 remeshing steps are conducted before the computation reaches $t = 0.0261420$ when the maximum value of the solution is 1.5636×10^4. Fig. 4 shows the sectional view of the contours filling for the solutions at $t = 0.0261420$ and the local mesh distribution. where one can see that the minimum mesh size is about 2.796×10^{-5}. The singularity is well solved with more than 1000 grid points within the cube of size 10^{-4}. And the maximum of the solution as a function of t is shown in Fig. 5. From that, we see a three-dimensional blow up solution well captured by our adaptive method.

Our computations were performed on a PC-Cluster installed with MPI system. For comparisons, we solve the equation to the same physical time $t = 0.0261420$ in two cases: one using 8 processors, while the other using only 1 processor. The CPU cost for 8 processors calculation is 5700 s, while that for one processor calculation is 33,200 s. This shows that our parallel computation's speed up can get up to almost 0.75.

To illustrate the efficiency of our proposed grid generation, we show the following remeshing CPU time statistics. For each remeshing, including solving the grid equation with conjugate gradient method and the grid map inversion, it takes only less than 20 s. The total CPU time for all remeshing steps is less than 600 s, almost one-tenth of the total CPU time (5700 s). For the grid generation algorithm in [19], the same ratio is almost one-forth (1650/5590) for a two-dimensional problem.

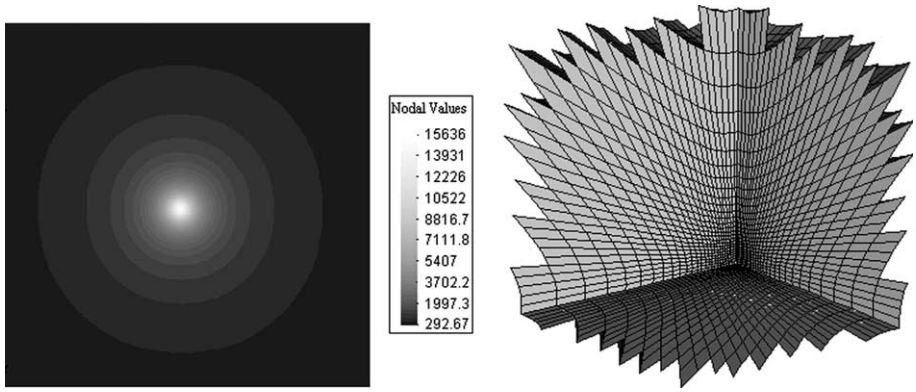

Fig. 4. Density plot of the solution of NLS in a cross-section (left) where the maximum value reaches 1.5636×10^4 and local mesh distribution near singularity (right) where the minimum mesh size is about 2.796×10^{-5}.

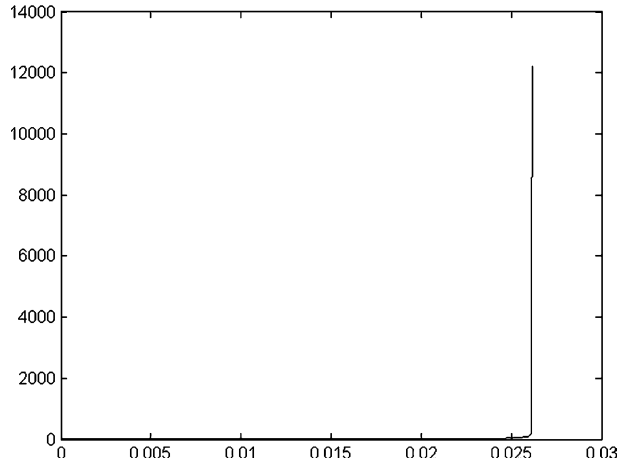

Fig. 5. Maximum values versus time.

5.2. Keller–Seigel equation: complex singularity

Our next example is the three-dimensional Keller–Segal (KS) model for bacterial pattern formation:

$$\rho_t = \epsilon\Delta\rho - \nabla\cdot(\rho\nabla C), \quad (x,y,z) \in \Omega_p,\ t > 0,$$

$$C_t = \Delta C + \rho.$$

Here ρ is the bacterial density and C is the attractant field. In some cases, the concentration of the attractant C draws the bacteria together and they achieve an infinite density with complicated geometric structures. It is observed in [3] that the high density regions initially collapse into cylindrical structures (line singularities) which subsequently destabilize and break up into spherical aggregates (point singularities). Here, we are interested in simulating the phenomena with the KS model. Two cases will be considered. The

first case has the initial attractant concentration on a straight line. Subsequent evolution shows a transition from line singularity to point singularities. The second case has the initial attractant concentration on a curved line which shows faster development of point singularities.

5.2.1. From line singularity to point singularity

The first initial condition is an uniform density distribution and an attractant field with a small perturbation in the z direction on the domain $[0,1]^3$

$$\rho(x,y,z,0) = 1.0,$$

$$C(x,y,z,0) = \mathrm{e}^{-10((x-0.5)^2+(y-0.5)^2)}(1+0.01|\sin 2\pi z|).$$

The ϵ is taken to be 0.01.

The above equation is solved on a uniform mesh in Ω_c with $40\times 40\times 200$ grid points. The monitor function and the parameters α and β are chosen as that in the NLS point singularity problem, and the TOL is chosen to be 8.0. The computation is continued until $t = 0.7534794$ when the maximum density reaches about 4.6×10^4.

The density ρ, at first, increases gradually, showing a line singularity along the central line $(0.5,0.5,z), z = 0,1$ until t reaches about 0.70. Then, this line singularity changes to multiple points singularity on two points which are initially perturbed to the maximal: $(0.5,0.5,0.25)$ and $(0.5,0.5,0.75)$. The density increases quickly and blows up at these two points in the end (Fig. 6). The color Fig. 7 shows the transition from line singularity to point singularity in contour filling forms corresponding to times: $t_1 = 0.0023$; $t_2 = 0.736591$; $t_3 = 0.7511$; and $t_4 = 0.75347497$. Fig. 8 shows the cutting view of the local grids corresponding to line singularity and points singularity, respectively. Fig. 9 shows the local sectional view of the grid around one singular point. Also, Fig. 6 shows the log plot of density values along the cental line $(0.5,0.5,z)$ $(0<z<1)$ versus time, which again illustrates the singularity transition.

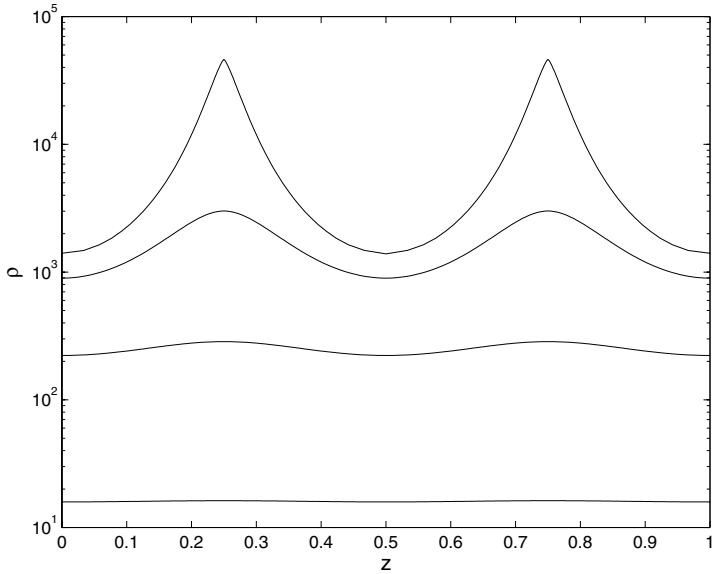

Fig. 6. Transition from line singularity to point singularities for KS: ρ (log scale) along the center line $(0.5,0.5,z)$.

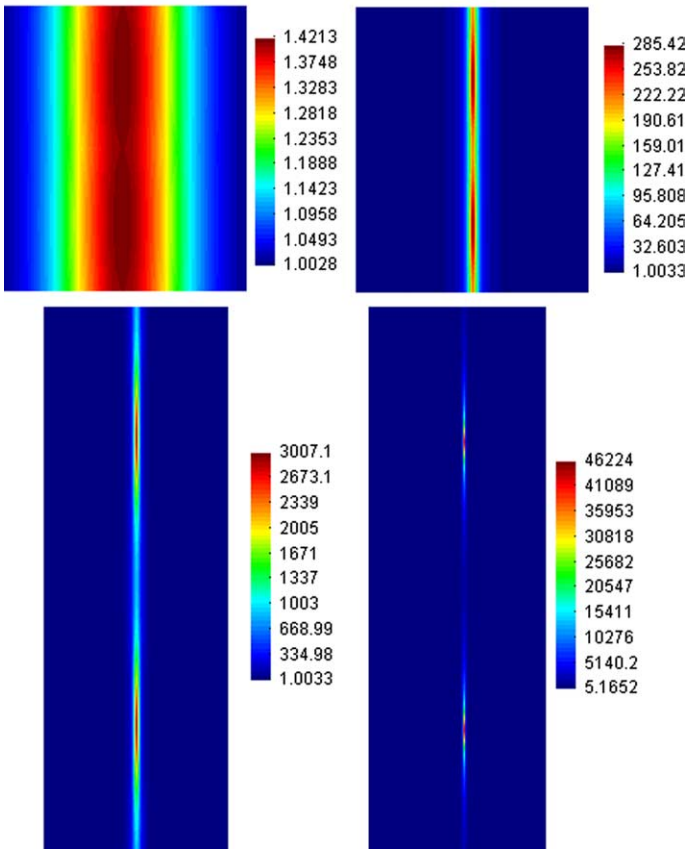

Fig. 7. Cross-sectional view of the color density plot for the same solution as in the previous figure.

The computation were done with 8 parallel processors which takes about 12 CPU hours (which would need 70 h on a single computation node). About 16 remeshings were performed. The ratio of the remeshing time to the total time is about (1:10) which again shows the efficiency of the new grid generation scheme.

5.2.2. Three points blow-up

Next, we change the initial data slightly so that the initial attractant field is concentrated on a curved line instead of a straight line along z direction

$$\rho(x, y, z, 0) = 1.0,$$

$$C(x, y, z, 0) = \mathrm{e}^{-10((x-x0(z))^2+(y-0.5)^2)},$$

where $x0(z) = 0.5 + 0.1\cos(2\pi z)$, Then the density ρ quickly develops maximum at three turning points of the line $x0(z)$ and results in point singularities at $(0.6, 0.5, 0.0)$, $(0.4, 0.5, 0.5)$, $(0.6, 0.5, 1.0)$. Fig. 10 shows the contour filling of the density changing along the time. The four fillings are corresponding to times $t_1 = 0.0012$; $t_2 = 0.5974$; $t_3 = 0.6825$; $t_4 = 0.7393$. Fig. 11 shows the distribution of grids. The CPU time, speed-up of our parallel program and other computation statistics demonstrate the effectiveness of the proposed iterative grid redistribution and its modifications in fast grid generation and parallelization.

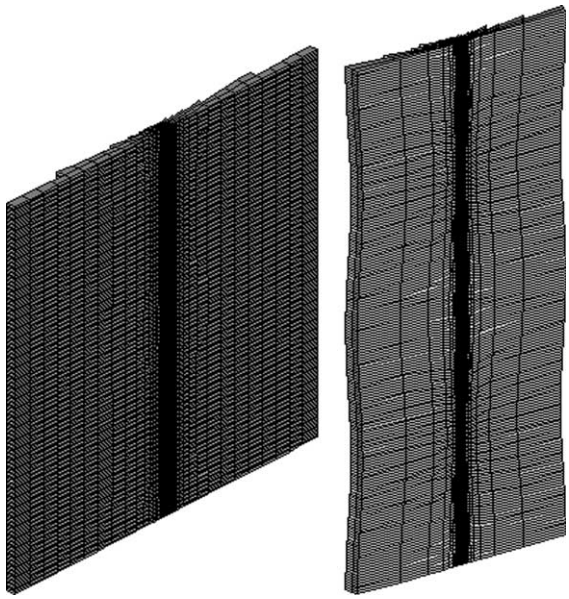

Fig. 8. Local grid distribution near the line singularity.

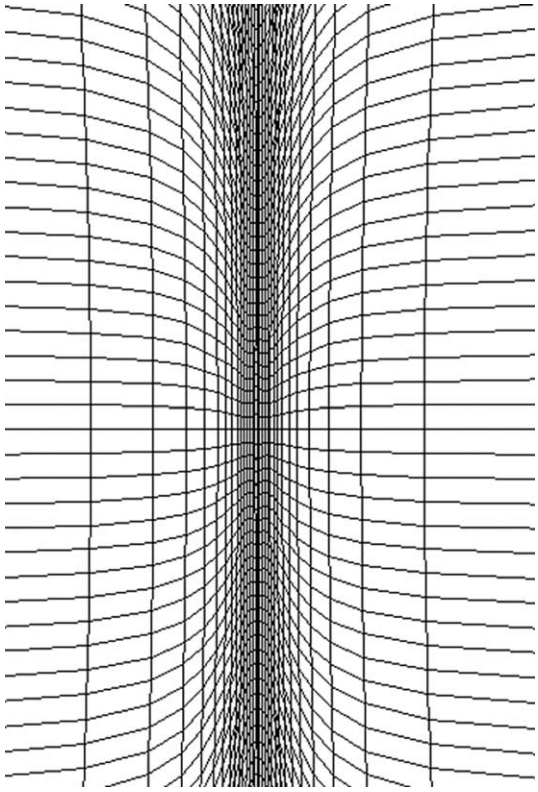

Fig. 9. Local view of the grid around a singular point.

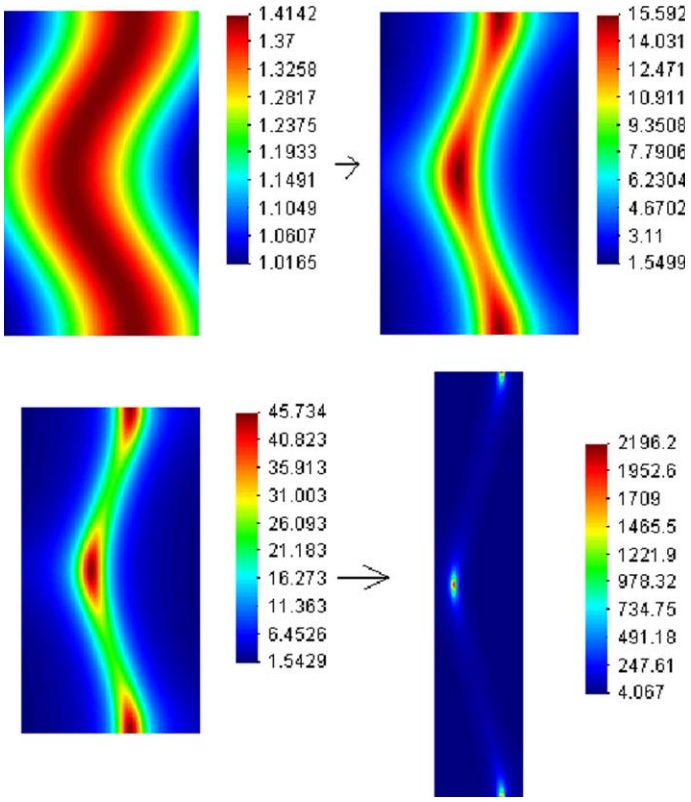

Fig. 10. Three points blow up for KS: cross-sectional view of color density plot of the solution.

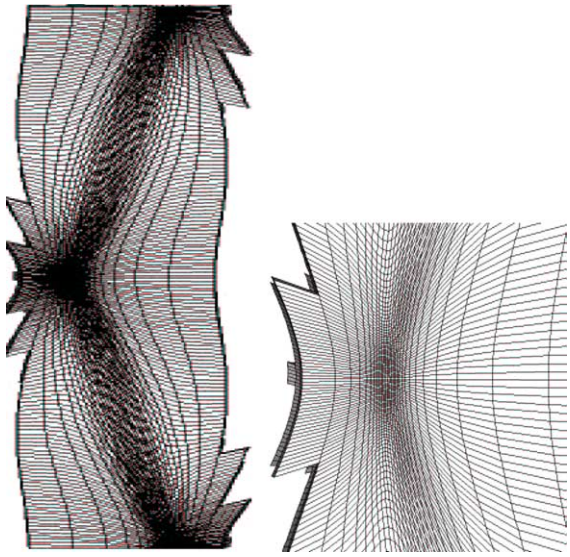

Fig. 11. Cutting view of the grids (left) and a local grid distribution near a singularity.

6. Conclusions

We have extended the iterative grid redistribution method for computing singularities from two dimensions to three dimensions. Two major improvements have been made: (1) a fast algorithm for efficient and robust grid generation; (2) parallelization of the whole adaptation process, which has almost optimal speed-up efficiency. With these modifications, the implemented three-dimensional iterative grid redistribution method is now a powerful tool for resolving not only point singularities, but also for complex singularities such as line singularity, transitions from line singularity to point singularities. We demonstrated the effectiveness of our method with several three-dimensional examples. We note that the analysis of global error for our method (or for moving mesh method in general) is a difficult problem which needs further study.

Acknowledgements

This work is supported in part through the Research Grant Council of Hong Kong by Grant HKUST 6176/99P and HKUST 6143/01P. Part of the work was done when D.S. Wang was a Postdoc at Institute of Computational Mathematics AMSS, Chinese Academy of Sciences.

References

[1] D.A. Anderson, Grid cell volume control with an adaptive generator, Appl. Math. Comput. 35 (1990) 209.
[2] M. Berger, P. Colella, Local adaptive mesh refinement for shock hydrodynamics, J. Comput. Phys. 82 (1989) 64–84.
[3] M.D. Betterton, M.P. Brenner, Collapsing bacterial cylinders, Phys. Rev. E 64 (2001) 061904.
[4] J. Brackbill, An adaptive grid with directional control, J. Comput. Phys. 108 (1993) 38.
[5] R. Camarero, M. Younis, Efficient generation of body-fitted coordinates for cascades using multigrid, AIAA J. 18 (5) (1980).
[6] H. Ceniceros, T.Y. Hou, An efficient dynamically adaptive mesh for potentially singular solutions, J. Comput. Phys. 172 (2001) 1–31.
[7] G. Fibich, W. Ren, X.P. Wang, Numerical simulations of self-focusing of ultrafast laser pulses, Phys. Rev. E 67 (2003) 056603.
[8] R. Hagmeijer, Grid adaption based on modified anistropic diffusion equations formulated in the parametric domain, J. Comput. Phys. 115 (1994) 169–183.
[9] W. Huang, R.D. Russell, Moving mesh strategy based upon gradient flow equation for two dimensional problems, SIAM J. Sci. Comput. 20 (1999) 998–1015.
[10] S.A. Jordan, M.L. Spaulding, A fast algorithm for grid generation, J. Comput. Phys. 104 (1993) 118–128.
[11] M. Landman, G.C. Papanicolaou, P.L. Sulem, C. Sulem, X.P. Wang, Stability of isotropic singularities for the nonlinear Schrödinger equation, Physica D 47 (1991) 393–415.
[12] R. Li, T. Tang, P.-W. Zhang, A moving mesh finite element algorithm for singular problems in two and three space dimensions, J. Comput. Phys. 177 (2002) 365–393.
[13] Message Passing Interface Forum, MPI: A Message-Passing Interface Standard (Version 1.1), Technical Report, 1995. Available from <Http://www.mpi-forum.org>.
[14] D.W. McLaughlin, G. Papanicolaou, C. Sulem, P.L. Sulem, Phys. Rev. A 34 (1986) 1200.
[15] K. Miller, R.N. Miller, Moving finite elements I, SIAM J. Numer. Anal. 18 (1981) 1019–1032.
[16] P.K. Moore, J.E. Flaherty, J. Comput. Phys. 98 (1992) 54.
[17] J.F. Thompson, Z.U.A. Warsi, C.W. Mastin, Numerical Grid Generation, North-Holland, New York, 1985.
[18] M.M. Rai, D.A. Anderson, Grid evolution in time asymptotic problems, J. Comput. Phys. 43 (1981) 327–344.
[19] W. Ren, X.-P. Wang, An iterative grid redistribution method for singular problems in multiple dimensions, J. Comput. Phys. 159 (2000) 246–273.
[20] A. Winslow, Numerical solution of the quasi-linear Poisson equation, J. Comput. Phys. 1 (1996) 149.