Adaptive Mesh Refinement for Singular Current Sheets in Incompressible Magnetohydrodynamic Flows

Holger Friedel, Rainer Grauer, and Christiane Marliani

Institut für Theoretische Physik I
Heinrich-Heine-Universität Düsseldorf
D-40225 Düsseldorf, Germany

The formation of current sheets in ideal incompressible magnetohydrodynamic flows in two dimensions is studied numerically using the technique of adaptive mesh refinement. The growth of current density is in agreement with simple scaling assumptions. As expected, adaptive mesh refinement shows to be very efficient for studying singular structures compared to non-adaptive treatments.

1. INTRODUCTION

The formation of singularities in hydro- and magnetohydrodynamic flows is still a controversial issue in the mathematics and physics community. Since mathematically only very little is known [1], one has to rely on numerical simulations. Even in very elaborate numerical experiments (see Bell and Marcus [2], Kerr [3]) non-adaptive treatment is limited very soon by the computer memory available, resulting in a resolution of less than 512 grid points in each spatial direction. Since the singular structures like tubes and sheets are not space filling, adaptive mesh codes seem to be the right choice for studying these problems, as has been done by Pumir and Siggia [4,5]. Unfortunately, the methods used in [4,5] could only refine the region around a singular point which lead in [4] to a substantial loss of energy. Of course, it is desired to refine all regions where the numerical resolution is insufficient. Modern adaptive mesh refinement algorithms, as introduced by Berger and Colella [6] and Bell *et al.* [7], do not possess the above limitations and are good candidates for studying singularity formation even in incompressible systems.

In this paper, we investigate the formation of singular current sheets described by the ideal incompressible magnetohydrodynamic equations (MHD equations) in two dimensions for the time evolution of the velocity field $\mathbf{u}$ and magnetic field $\mathbf{B}$. Using Elsässer variables $\mathbf{z}^{\pm} = \mathbf{u} \pm \mathbf{B}$, the MHD equations take the symmetric form

$$\partial_t \mathbf{z}^{\pm} + \mathbf{z}^{\mp} \cdot \nabla \mathbf{z}^{\pm} + \nabla p = 0 \ , \quad \mathrm{div}\ \mathbf{z}^{\pm} = 0 \ . \qquad (1)$$

The equations (1) are integrated in a periodic quadratic box of length 2π using adaptive mesh refinement with rectangular grids self-adjusting to the flow. In each rectangular grid, a projection method is used where the timestepping is performed in a second order upwind manner [8–10]. For the projection step, we need the vorticities $\omega^{\pm} = (\nabla \times \mathbf{z}^{\pm}) \cdot \mathbf{e}_z$ and potentials $\psi^{\pm}$ which are related by $\Delta \psi^{\pm} = \omega^{\pm}$.

The outline of the paper is as follows. In the next section, the adaptive mesh refinement algorithm is introduced. Then, we discuss the numerical results and compare the growth of current density with the prediction of Sulem *et al.* [11]. Finally, we conclude that adaptive mesh refinement is an ideal tool for studying singular structures and should be pursued further to study three dimensional problems as the finite time blow up in the incompressible Euler equations.

2. ADAPTIVE MESH REFINEMENT

2.1. General Strategy

The main idea of adaptive mesh refinement is simple. One starts with a grid of given resolution and integrates the partial differential equation as usual. As soon as some criterion is fulfilled, this initial grid is refined. This is done by marking all critical grid points where the discretization error exceeds a prescribed value. Then new grids with finer resolution and timestep are generated which cover all these critical points. These grids belonging to the next level are then filled with interpolated data from the first level. Then, one integrates both levels until the resolution again becomes insufficient. Now the critical points are collected over all grids of the actual level being refined. Filling the new grids with data is achieved by first taking data from the previous level and, if existing, data from former grids of the same resolution. This process is repeated recursively. In addition, to communicate the boundary conditions, each grid needs information about its parent grids and its neighbors. As one can see already, adaptive mesh refinement requires the management of lists of levels, critical points, grids, parent grids and neighbors. Therefore, we programmed the handling of those structures in C++ whereas the numerically expensive integrations are done in fortran. In order to encourage the reader to use adaptive mesh refinement we describe the above outline in more detail in the next paragraphs.

To deal with all the different lists we defined templated list classes and iterators which can be used for all classes representing levels, grids, critical points, parents and neighbors.

The integrator used for all grids is based on a projection method combined with second order upwinding. This scheme motivated by Bell *et al.* [8] was previously applied to incompressible magnetohydrodynamic flows in two dimensions [10]. It is clear, that the equations under consideration can be easily exchanged by other ones using an explicit algorithm since the structures needed for adaptive mesh refinement and the integrator are independent of each other.

The timestep on a given *level* is advanced as illustrated by the following piece of pseudo-code.

```
procedure integrate level
    do singlestep on level
    better boundary on level
    solve poisson equation on level

    if next level exists, then
        default boundary on next level
        do r times
            integrate next level
        update of level
    check criterion on level
```

Starting at time t_0, the procedure singlestep performs one timestep Δt_{level} on all grids of this level. In addition to the data within the grid itself the integration scheme needs boundary data which are by default obtained by interpolation in space and time from previous level data. In the subsequent procedure better boundary, the boundary data are, if possible, replaced by values from neighboring fine grids. After boundary data have been communicated on each grid, in order to perform the projection step Poisson equations with fixed boundary are solved for the potentials $\Delta\psi^{\pm} = \omega^{\pm}$. Now all data of the actual level are advanced to a time $t_0 + \Delta t_{level}$ and the recursion starts by integrating the next level if existing. The first step in this recursive process is achieved by supplying information about the default boundary data from parent grids. This is done by storing the increments calculated from the actual grids at time t_0 and parent grids at time $t_0 + \Delta t_{level}$. To achieve linear interpolation in time these increments are added to the boundary data at the end of singlestep. Storing only the increments in a special C++ boundary class avoids the memory overhead resulting from keeping data at present and previous times. On this next level, the timestep $\Delta t_{level+1}$ and the spatial discretization lengths are divided by a refinement factor r. Therefore, the procedure singlestep has to be called r-times on this new level in order to reach the time $t_0 + \Delta t_{level}$. Having completed this recursive integration loop, this level and all finer levels are advanced to time $t_0 + \Delta t_{level}$. Now the finer level data are used in procedure update to improve the values of the actual level. The procedure integrate is finished by checking if a certain criterion is fulfilled, that decides whether a refinement step is performed.

2.2. Regridding

The criterion for refinement is adapted to the problem of current sheet formation. The global maximum of vorticity and current density is calculated and compared to the values when the last refinement was done. Regridding is initiated, if the ratio of those maxima exceeds a prescribed value which is equal to the refinement factor r due to the scaling symmetry of the MHD equations (1). The result of regridding is a new list of levels starting below the actual level. This new list replaces the old one, which is then deleted.

For the problem of current sheet formation, it never happens that finer levels become obsolete. However, for other problems were the criterion for refinement can not be physically motivated, generation and rebuilding of the grid hierarchy is triggered after a certain number of timesteps on each level and based only on local errors. Then obsolete levels are discarded when resolution is sufficient.

The logical structure of the regridding procedure is shown in the subsequent pseudo-code.

```
procedure regridding level
    for all grids on level
        mark critical points and append them to a list
    cover the critical points with rectangles (saw up)

    nesting rectangles into their parents and
    assign parents and neighbors

    fill the new rectangles with default data

    calculate global maxima for comparison
    in the procedure check

    if old level of same resolution existed before
    regridding, then
        better data on new level from old level
        solve poisson equation on new level
        if finer level existed before regridding, then
            regridding of new level
        else
            assign global maxima from old level
    else
        solve poisson equation on new level
```

The procedure regridding starts with a loop over all grids of *level* to collect the critical points. Therefore, we calculate at each grid point the difference between the convection terms $\mathbf{z}^{\mp} \cdot \nabla \mathbf{z}^{\pm}$ evaluated using the spacing dx of the actual grid and double the spacing $2dx$. If this difference exceeds a prescribed threshold ϵ, we append this point and a surrounding rectangle of given size to the list of critical points. In the procedure saw up these critical points are covered with rectangles. The procedure nesting guarantees that they are properly nested into grids of the previous level allowing for more than one parent grid. At the same time, parent and neighbor grids are assigned to each new rectangle. Since these procedures are the most complex ones, they will be discussed in detail in the next subsections.

Now as each rectangle has information about his parents, the new rectangles are filled with spatially interpolated data in default data. To avoid discontinuities, interpolation is done on the fields containing the highest derivatives, namely $\nabla \times \mathbf{z}^{\pm}$. In order to supply boundary conditions for the solution of the Poisson equations, data for the potentials $\psi^{\pm}$ on the outermost boundary are assigned as well. Afterwards, global maxima needed in the procedure check are calculated.

If the recursive regridding was first invoked on the deepest level, the procedure is finished by solving the Poisson equations on the new level. Otherwise, data of the same resolution already existed and are used in better data to get more accurate values for the new grids. Data for the potentials are available after solving the Poisson equations. If the old level of same resolution was not the deepest level, the recursive regridding procedure is applied to the *new level*. In order to avoid unnecessary rebuilding of the level hierarchy, global maxima used as reference in check are assigned from the *old level* only in the other case.

2.3. Grid Generation

The grid generation is performed in the procedure saw up acting on a list of rectangles. On first entry, this list consists of one rectangle which covers all critical points of that level. Each rectangle is now processed in the following way. First, it is decided in which direction the first cut will take place. Therefore, we calculate vectors in the x- and y-direction which contain the number of critical points in each column or row, respectively. According to Bell *et al.* [7], we call them horizontal and vertical *signatures* Σ. The first cut is done in the direction with larger fluctuations in signature. This is achieved in the procedure cut dim, which first seeks for the best cut in this direction. In the procedure cut zeroes of the signature and its turning points (zeroes of $\Delta_i = \Sigma_{i-1} - 2\Sigma_i + \Sigma_{i+1}$) are taken into account as possible cuts. If no such cuts are found, the mid point is chosen. A cut results in two lists of critical points. Each list is covered by a rectangle of minimal size. To every rectangle costs are assigned which are calculated as a sum of integration and memory costs ($\propto$ the area), boundary communication costs ($\propto$ the perimeter) and fixed costs (measuring the overhead for managing one additional grid). The two rectangles having the minimal costs are returned. Afterwards a loop over these two rectangles is performed. They are both given to the procedure cut to find the best cut in the other direction. The costs of the two new rectangles in comparison to the original one's are used to decide whether the second cut is accepted or not. This gives a list of two, three or four rectangles. Their costs are summed up, and if they are less than the costs of the rectangle which entered the procedure cut dim, they are returned to saw up. Otherwise, an empty list is given back. In the latter case, if the efficiency measured by the ratio of critical points and grid points in the rectangle is insufficient, we enforce a cut in the middle of the longer side of the rectangle. Now the new rectangles are appended to a temporary list, which is, if not empty, passed to the recursive procedure saw up again. This recursion is stopped when further cuts do not allow a reduction of costs anymore. The above treatment is summarized in the two following pieces of pseudo-code.

```
procedure saw up rectangles
   for all rectangles
      calculate Σ and variance in x- and y-direction
      if variance in x > variance in y, then
         apply cut dim on rectangle in x-direction
      else
         apply cut dim on rectangle in y-direction
      if no cut found and efficiency insufficient, then
         half rectangle in longer direction
      append resulting rectangles to temporary list
      saw up of temporary list of rectangles
      if temporary list is not empty, then
         replace actual rectangle by temporary list
```

```
procedure cut dim of rectangle in direction dim
   determine best cut in direction dim
   and return two rectangles

   loop over the two rectangles
      cut in other direction

      if costs are smaller than those of actual
      rectangle, then
         replace actual rectangle by list

   compare costs of new list (of 2-4 rectangles) with
   those of original rectangle and return cheapest
```

An example, where saw up produces three new rectangles is shown in Figure 1.

Let us briefly comment on the most important differences between our grid generation algorithm and the one by Bell *et al.* [7]. First, we do not sort the marked points into clusters before starting the grid generation itself. Clustering is achieved by the procedure cut dim that

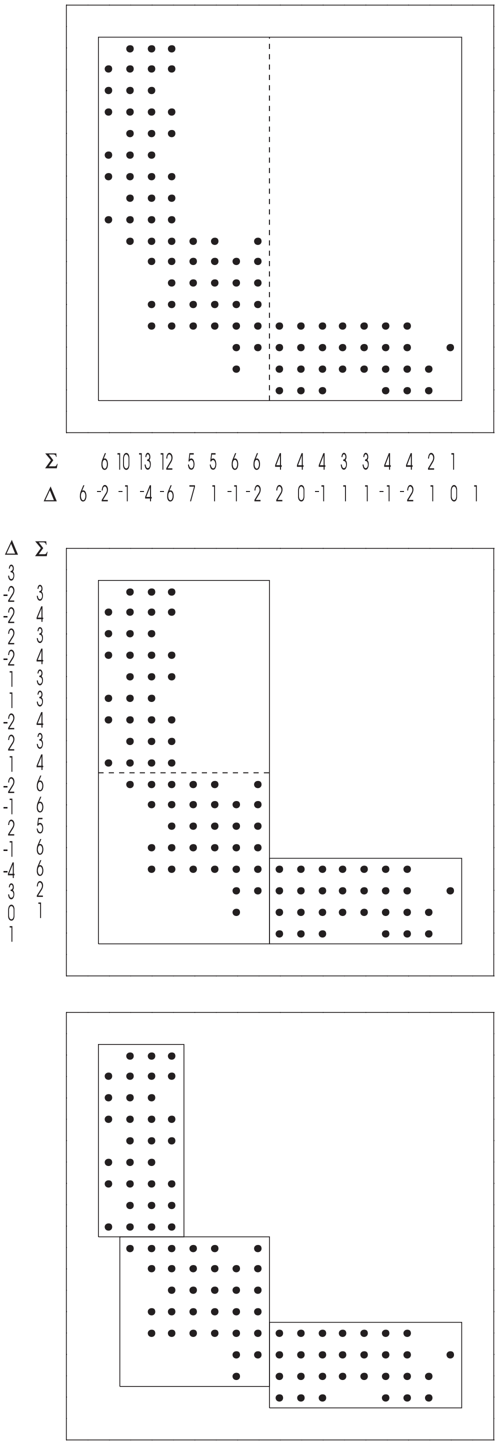

FIG. 1. The effect of procedure saw up.

evaluates two cuts at a time, one in each dimension, instead of independently performing one cut after another as they do. We choose the cuts to be performed by evaluation of costs and not according to a hierarchy in between the detected positions for a cut as done by Bell *et al.* [7]. Our algorithm needs slightly more computations, but has the advantage to lead to a more optimized covering of the marked points by rectangles. We decide whether the recursive cutting is ended or not on the basis of the costs and not on the fraction of marked points in the rectangle as they do. The definition of costs as discussed above allows to optimize the grid generation with respect to specific demands, e.g. given by the numerical integration scheme, physical problem and computer architecture.

2.4. Nesting

After generation of non-overlapping rectangles in the procedure saw up, it is not guaranteed that all rectangles are properly nested in the rectangles of the parent level. A typical example, where this is not the case, is shown in Figure 2.

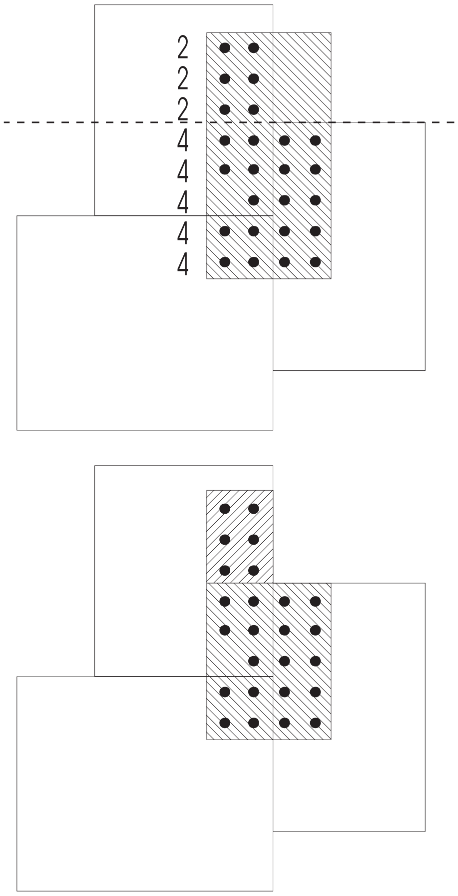

FIG. 2. The result of the nesting procedure.

To check, whether a rectangle is properly nested we calculate the sum of areas of intersections with all rectangles of the parent level. When this area equals the area of the actual rectangle it is guaranteed that this rectangle is properly nested. Otherwise, we proceed as follows. First, we determine the longest common edge of the just calculated intersections. Compared to Berger and Colella's [6] realization of nesting our algorithm avoids coinciding cuts of several levels and results in a small number of cuts only. We seek for cuts perpendicular to the longest common edge. Let us assume, as in Figure 2, that this edge lies in the y-direction. Then, we cut where the number of grid points covered by the intersections in each row changes. This list of rectangles is recursively tested for proper nesting. Obviously, this procedure is well suited to assign parents to each rectangle at the same time. After having obtained a list of properly nested rectangles, they get information about their neighbors.

2.5. Integral and local controls

In order to extract physical properties of the simulation, it is necessary to calculate integral quantities like kinetic and magnetic energy as well as maxima of current density and vorticity. The latter are easily obtained by looping over all grids and all levels. Integral quantities are calculated in the following way. First, on the coarsest level the energy E_{level} (swiss cheese energy) associated to the area not covered by grids of higher resolution is calculated. This is repeated down to the lowest level. Finally, the energy is obtained as a sum over all energies E_{level}.

2.6. Parallelization

On shared memory machines our adaptive mesh refinement code can be parallelized in an effective and straight forward way. The main time of the program is spent in the procedure `singlestep`. Since the number of grids is much higher than the number of processors parallelization is done by distributing the grids to the processors. That means that as soon as a singlestep on a grid is finished, the next grid is passed to the free processor. This results in a very effective utilization of all processors. All this can easily be done using standard Posix threads. The implementation on distributed memory machines using the shared memory access model is in work.

3. NUMERICAL RESULTS

In contrast to simulations of Frisch *et al.* [12] and Sulem *et al.* [11], we choose as initial condition a modified Orszag–Tang vortex, given by

$$\varphi^0(x,y) = \cos(x+1.4) + \cos(y+2.0) \ ,$$
$$\psi^0(x,y) = \frac{1}{3}\left[\cos(2x+2.3) + \cos(y+6.2)\right] \ .$$

This initial condition, which was already used in turbulence simulations [13,10], possesses less symmetry and is therefore more generic for the formation of small-scale structures. Computations are done with periodic boundary conditions on a square of length 2π. The initial spatial resolution was given by 256^2 grid points.

The temporal evolution of the current density is shown in the contour plots of Figure 3. In addition to the contour levels, the rectangle hierarchy is plotted. The first plot shows the grid after the first refinement has taken

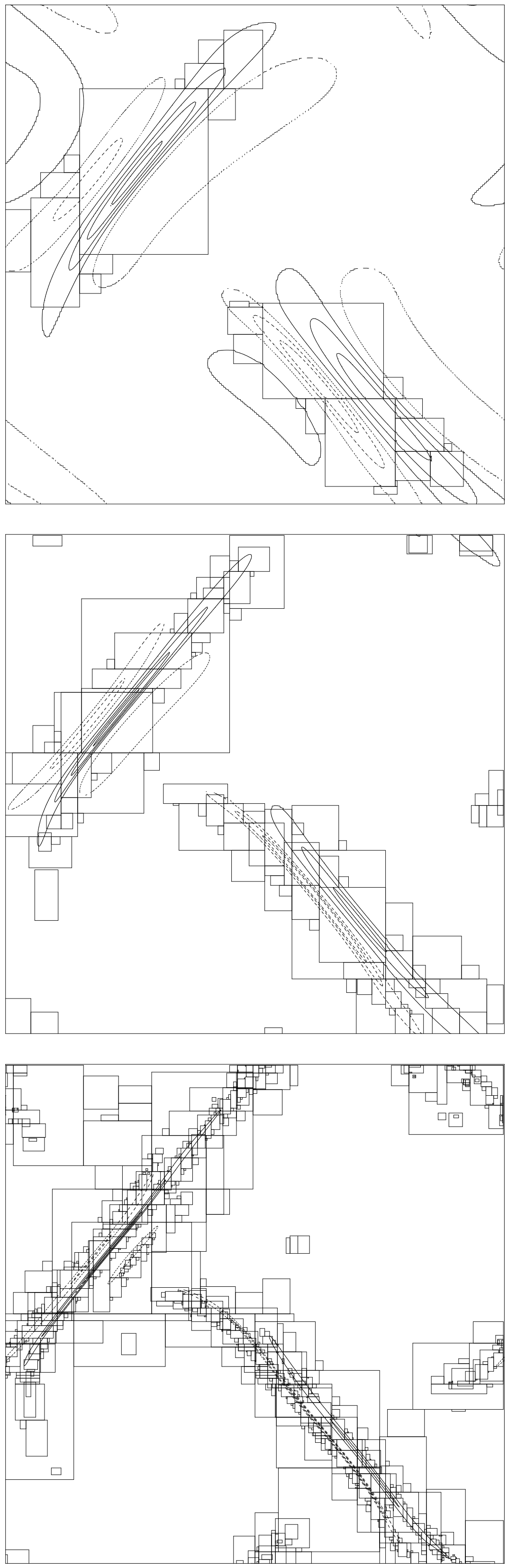

FIG. 3. Evolution of the current density at times 1.6, 2.2 and 2.7.

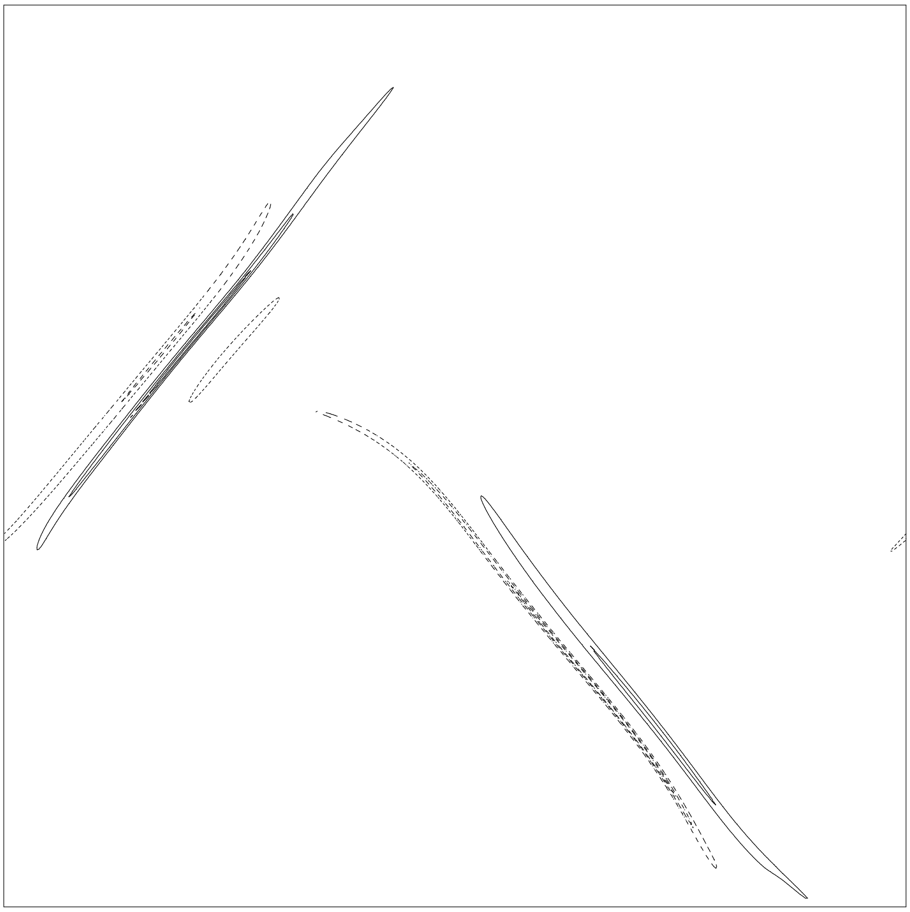

FIG. 4. Current density at time 2.7.

place. The contour plot at time $t = 2.2$ contains already 3 levels. At the final time $t = 2.7$ a total of 5 levels are present. Figure 4 is a contour plot at the same time as the last one of Figure 3. To avoid hiding the sharpness of the current sheets no rectangles are included. In the actual simulation, the refinement factor was equal to $r = 2$. On a workstation with 128 Mbyte of main memory, four refinements could be realized corresponding to a resolution of 4096^2 grid points with a non-adaptive scheme. The limiting factor is the amount of main memory available, whereas up to this resolution computational costs are very moderate.

In the first picture the current sheets start to form, afterwards they evolve into thiner and thiner sheets and the maxima of current density and vorticity are increasing continually. The current density is growing exponentially in time. In Figure 5 a semilogarithmic plot of the maximum current density in the upper sheet is depicted. Included is a fit to an exponential function given by $j_{\rm fit}(t) = 0.5 \exp(2.115\, t)$. This functional behavior is in agreement with the results of Sulem *et al.* [11]. A detailed analysis of the asymptotic scaling behavior and a comparison to the predictions in [11] will be presented elsewhere.

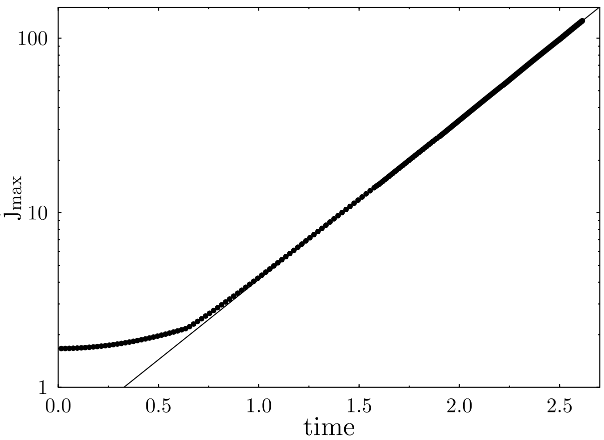

FIG. 5. Amplification of current density.

The second order upwind scheme produces substantial energy dissipation only if underresolved steep gradients have formed. Therefore, the energy conservation is a measure whether the singular current sheets are sufficiently resolved. In Figure 6 we give a plot of energy as a function of time. To be more precise, total energy is conserved to within less than 1 %.

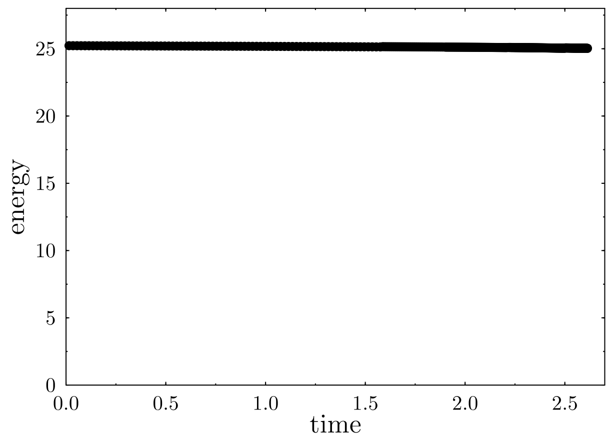

FIG. 6. Energy conservation.

In order to further illustrate that the current sheets are well resolved, in Figure 7 we show one dimensional cuts in x-direction through the maximum of current density

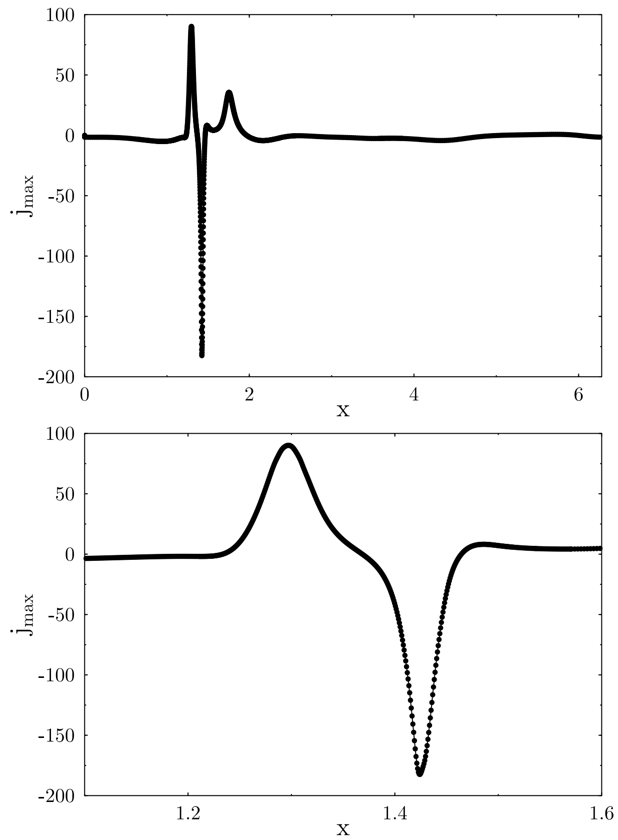

FIG. 7. Cuts of current density in x-direction through the maximum at time 2.72.

in the upper half of the integration range. In the upper plot the x-range equals the periodicity length. The lower one with a reduced plot range shows that the grid points of the finest levels very well resolve the current sheet.

In the previous section we mentioned that a refinement would take place when the discretization error for the nonlinearity exceeds a prescribed value ϵ. The choice of the parameter ϵ is crucial for the numerical accuracy. If ϵ is taken too large, certain regions may be underresolved which can lead to reconnection and violation of energy conservation. Decreasing systematically the value of ϵ has the effect that reconnection phenomena are suppressed. Below a certain threshold the numerical results proved to be independent of ϵ. However, it should be noted that small changes in ϵ do only slightly alter the efficiency and memory consumption. The simulations shown in Figure 3 were performed with $\epsilon = 0.025$.

Applying adaptive mesh refinement to the evolution of singular structures like current sheets in magnetohydrodynamics is motivated by the expected reduction of memory needed to resolve them. This is well justified by the numerical results. To give the reader an impression of how many grids are generated on the different levels and of the number of grid points contained in each level's grids, we display values for the level hierarchy at time $t = 2.7$ in the following tables. In Table I the results are shown for a simulation with refinement factor $r = 2$ and in II for another one with $r = 4$.

TABLE I
Statistics for simulation with $r = 2$.

level	number of grids	grid points in level
0	1	70225
1	51	168033
2	100	341349
3	178	734426
4	417	1557221

TABLE II
Statistics for simulation with $r = 4$.

level	number of grids	grid points in level
0	1	70225
1	49	506073
2	195	2331952

From level to level the total number of grid points grows much less than by a factor of r^2 necessary for a non-adaptive treatment. For r chosen equal to 2, one can see that even for the very small value of ϵ prescribed here it increases no more than by a factor of about 2. This promises that the compression rate will improve the more refinements are performed.

In Table III simulations with different refinement factors are compared with regard to the total number of grid points on all levels. The number of grid points on one data field with the same grid spacing as the finest level in the adaptive code is called the non-adaptive size. In the last row we give the ratio of the grid points, adaptively and non-adaptively. For the investigated hierarchy of 5 levels with refinement factor $r = 2$ this ratio is about 17 %. When the finest levels are equally resolved, the compression for both refinement factors is practically indistinguishable. For the comparison of adaptive versus non-adaptive treatment, the compression rate based on counting grid points does not fully reflect the total improvement in main memory consumption. In upwind schemes several auxiliary fields have to be stored. In non-adaptive simulations these full sized fields are present all the time whereas here they are needed only temporarily during the execution of singlestep on a small grid.

TABLE III
Comparison of different refinement factors.

	$r = 2$	$r = 4$
total number of grid points	2871254	2908250
non-adaptive size	16851025	16851025
ratio	0.170	0.172

Let us shortly discuss the portion of time spent in routines related to adaptive mesh refinement. Using an analyzing tool for performance analysis we could bound that portion to be less than 8% including checking, regridding and all boundary and update communications.

We want to finish this section with an impressive comparison of the results for the amplification of the current density for several non-adaptive grid sizes and for the adaptive code. Figure 8 is a parametric plot of the maximum of current density as a function of the fit $j_{\rm fit}(t) = 0.5 \exp(2.115\, t)$ already depicted in Figure 5. In addition to the results of the adaptive mesh refinement code we include data obtained with fixed grids of resolutions 128^2, 256^2, 512^2 and 1024^2. It should be noted that integrating a grid of fixed size all data has to fit into the computer's main memory, whereas for the adaptive simulations the size of the grid on which timestepping is actually performed is relevant. Until the simulations become underresolved, a linear behavior is also observed in the non-adaptive simulations. Then the upwind method introduces numerical viscosity leading to reconnection processes and substantial energy dissipation.

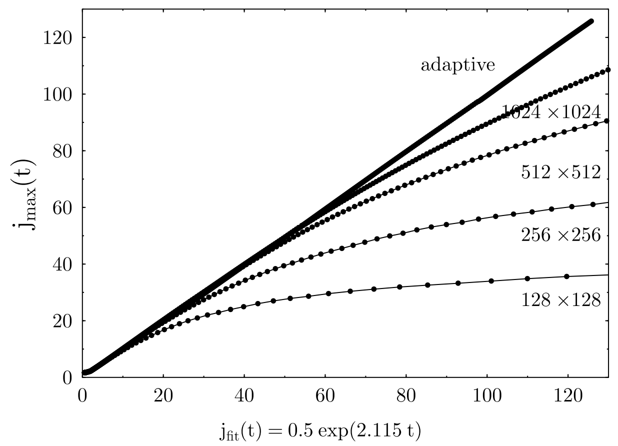

FIG. 8. Comparison of adaptive and non-adaptive simulations.

4. CONCLUSIONS

The complexity of adaptive mesh refinement compared to non-adaptive treatments should not be underestimated. On the other hand, the growing progress of object oriented programming languages helps enormously to reduce the difficulties in programming. To give some impression, the programs needed for regridding, nesting and the handling of data structures are only about 3000 lines of C++ code.

As we have demonstrated, adaptive mesh refinement is a powerful tool to study the evolution of singular structures as the formation of current sheets in ideal MHD. Other problems of this type like in the axisymmetric [14] and the full three dimensional Euler equations are natural candidates for this method. Work in this direction is in progress.

Whether adaptive mesh refinement is also a useful concept for simulating turbulent hydro- and magnetohydrodynamic flows will depend on how efficiently the small scale structures can be covered by hierarchically nested grids.

ACKNOWLEDGMENT

We like to thank K. H. Spatschek for his continuous support. This work was performed under the auspices of the Sonderforschungsbereich 191.

[1] J. T. Beale, T. Kato, and A. Majda, Remarks on the breakdown of smooth solutions for the 3D Euler equations, *Comm. Math. Phys.* **94**, 61 (1984).

[2] J. B. Bell and D. L. Marcus, Vorticity intensification and transition to turbulence in three-dimensional Euler equation, *Comm. Math. Phys.* **147**, 371 (1992).

[3] R. M. Kerr, Evidence for a singularity of the three-dimensional, incompressible Euler equations, *Phys. Fluids A* **5**, 1725 (1993).

[4] A. Pumir and E. Siggia, Collapsing solutions to the 3D Euler equations, *Phys. Fluids. A* **2**, 220 (1990).

[5] A. Pumir and E. Siggia, Development of singular solutions to the axisymmetric Euler equations, *Phys. Fluids. A* **4**, 1472 (1992).

[6] M. J. Berger and P. Colella, Local Adaptive Mesh Refinement for Shock Hydodynamics, *J. Comput. Phys.* **82**, 64 (1989).

[7] J. Bell, M. Berger, J. Saltzman, and M. Welcome, Three-dimensional adaptive mesh refinement for hyperbolic conservation laws, *SIAM J. Sci. Comput.* **15**, 127 (1994).

[8] J. B. Bell, P. Colella, and H. M. Glaz, A second-order projection method for the incompressible Navier–Stokes equation, *J. Comput. Phys.* **85**, 257 (1989).

[9] J. B. Bell and D. L. Marcus, A second-order projection method for variable-density flows, *J. Comput. Phys.* **101**, 334 (1992).

[10] R. Grauer and C. Marliani, Numerical and analytical estimates for the structure functions in two-dimensional magnetohydrodynamic flows, *Phys. Plasmas* **2**, 41 (1995).

[11] P. L. Sulem, U. Frisch, A. Pouquet, and M. Meneguzzi, On the exponential flattening of current sheets near neutral X-points in two-dimensional ideal MHD flow, *J. Plasma Phys.* **33**, 191 (1985).

[12] U. Frisch, A. Pouquet, P. L. Sulem, and M. Meneguzzi, The Dynamics of two-dimensional ideal MHD, *J. Méc. Théor. Appl.* , 191 (1983).

[13] D. Biskamp and H. Welter, Dynamics of decaying two-dimensional magnetohydrodynamic turbulence, *Phys. Fluids B* **1**, 1964 (1989).

[14] R. Grauer and T. C. Sideris, Finite time singularities in ideal fluids with swirl, *Physica D* **88**, 116 (1995).