Geosci. Model Dev. Discuss., 8, C1561–C1566, 2015
www.geosci-model-dev-discuss.net/8/C1561/2015/

Geoscientific Model Development Discussions

Interactive comment on "A new multiscale air quality transport model (Fluidity, 4.1.9) using fully unstructured anisotropic adaptive mesh technology" *by* J. Zheng et al.

Anonymous Referee #2

Received and published: 1 August 2015

This paper briefly describes algorithms implemented in Fluidity for unstructured anisotropic refinement applied to solution of scalar advection equations, using either a continuous or discontinuous Galerkin method. Several numerical experiments are presented illustrating the performance of the algorithms on standard test problems for advection in a two dimensional rectangular domain.

I have a number of serious concerns with this paper and do not believe it is suitable for publication in its present form.

1. It is not clear to me what is novel in this paper, or even whether the goal of the paper is aligned with the aims of this journal. The advection algorithms and also the adaptive

refinement algorithms are all implemented in Fluidity, but from the paper it is not at all clear whether the authors of the paper were involved in some new implementation in this version of the code, or are simply testing the code on some particular test problems. The title of the paper, explicitly mentioning Fluidity 4.1.9, makes it sound like the code is specifically designed for the problem discussed in the paper and the paper serves to describe the full code. However, in Section 4 it is stated that Fluidity solves 2D and 3D Navier-Stokes equations and multiphase flow problems over topography, while this paper only concerns scalar advection in two dimensions. So the paper does not seem to describe or test very much of Fluidity. Moreover there is no real discussion of a "new air quality model" anywhere in the paper. Standard 2D advection test problems are used. Advection equations may be used in air quality models but there does not seem to be anything specific to this application, and advection equations arise in many other situations, so it seems misleading to include this term in the title.

2. Are these specific advection algorithms and/or the adaptive mesh refinement algorithms significantly different in 4.1.9 than they were in 4.1.8? Or are the authors just noting the particular version that they happened to use for these tests of algorithms that have long been a part of Fluidity? If the latter, what is the novel algorithm or software development? A large number of papers have already been written on advection algorithms of the sort used here, which are often tested on similar problems. The anisotropic refinement algorithm is not described in any detail so it is also not clear if there is anything new here. This all needs to be better clarified.

3. The application of the algorithms to the test problems is not well described, e.g. the description on page 4345 of the error metric tensor is inadequate. In (13) it is stated that H is the Hessian matrix, but of what? The full discretization in terms of all degrees of freedom? How are the elements of this tensor used to determine where to refine?

4. It would be very useful if the authors would make the code available to accompany this paper, so that readers could potentially better understand the details of the tests performed. This would also be very useful to any reader who is interested in imple-

menting something similar in Fluidity.

5. It is not well explained why it is necessary to use an implicit method for the hyperbolic advection equation, for which explicit methods are more easily implemented and generally preferred for efficiency reasons. It is stated that very large CFL numbers (e.g. 80) are used, and presumably this is because of the highly anisotropic cells with very large aspect ratios. I assume these are stretched in the advection direction, as suggested by Figure 14. Presumably these very high CFL numbers result from comparing e.g. the velocity in the x-direction in this figure to the width of the cells in the y-direction. If the CFL number were truly this large in terms of the number of grid cells the flow advects through in one time step (e.g. if the flow were in the y-direction in Figure 14) then I believe the implicit method would be extremely dissipative and fairly useless, even if it did remain stable. However, this is not discussed in enough detail to figure out what is going on.

6. On page 4347, line 25, "advection subcycling" is mentioned but is not explained. Does this mean smaller time steps are used in smaller cells? If so, how are these time steps chosen? Since there is a continuous distribution of cell sizes this is not clear, nor is it clear what is done when adjacent cells are using different size time steps and hence updated a different number of times.

7.The anisotropic refinement illustrated in Figure 14 may work well for this flow field in which the streamlines are constant in time and hence the flow is always in a fixed direction at each point in the domain, but it is not at all obvious that the approach used here would work for advection in a real fluid flow (such as the sort Fluidity presumably computes when solving the Navier-Stokes equations, or the sort alluded to in the title of the manuscript). In most flows the direction of flow at each point will be changing dynamically. Even if the adaptive grid is constantly deformed in every time step, the flow would generally not be exactly aligned with the highly anisotropic cells and I suspect this would severely impact the accuracy. All three of the test problems presented in this paper have the feature that the flow directions are time-invariant (even problem

2, where the flow speed varies, has constant direction at each point). I believe the algorithm should be tested on more challenging problems.

8. The test problems also have large regions of the domain where the solution is constant and hence very few grid cells are needed. This is perhaps reasonable since the point of adaptive refinement is to handle problems where the features needing refinement are relatively isolated. But comparisons of accuracy versus number of cells is then somewhat arbitrary for these problems, since making the domain larger relative to the region where the solution is non-constant would greatly increase the number of grid cells needed for a given resolution on a uniform grid but have no impact on the number of cells needed for the adaptive algorithm. Hence one can make this ratio arbitrarily large by making the domain large, and test problem 3 in particular has a domain that is far larger than reasonable for the given problem.

9. There is no discussion in the paper of what order of accuracy the advection algorithm is expected to have for smooth solutions, nor even a mention of what order polynomials are used in the continuous or discontinuous Galerkin methods. This is strange, since the presumed advantage of using such methods over simpler and perhaps more efficient finite difference or finite volume methods is that they can achieve higher order. A potential user of Fluidity would surely want to know what orders are supported, along with some evidence that it delivers.

10. None of the test problems have smooth initial data for which this accuracy could be tested. I think some test should be performed of the order of accuracy on smooth data in addition to showing the performance on the sort of data used in the test problems shown.

11. The error plots in Figures 2 and 5 are logarithmic in x and linear in y, which is not a useful way to display the error. A log-log plot would make it easier to determine the order of accuracy.

12. Moreover, Figures 2 and 5 also seem to show that the error asymptotes to non-

zero values as the grid is refined for most of the methods displayed, which means the methods are not even converging, let alone exhibiting any reasonable order of accuracy. This seems to be a serious problem.

13. What are the units of CPU time in Figures 2 and 5? Seconds? If so, then apparently the uniform grid DG method in Figure 5 requires 11 hours of CPU time for one revolution of two-dimensional advection on a 400 by 400 grid! Even the adaptive DG code seems to take around 2 hours with h = 1/800, which seems quite excessive for this problem. Of course it would also be useful to state what computer these timings were done on, and how many cores were used since it is stated in the paper that Fluidity uses MPI for parallelization to thousands of cores, although it is not stated whether this is used in these examples.

14. If I am interpreting the timings right, I suspect the slowness of the code is due to the use of an implicit method for the advection problem. This leads me again to question the wisdom of such methods for this problem, since there are good explicit block-structured AMR algorithms implemented in software such as AMROC, Boxlib, Chombo, Clawpack, SAMRAI, etc. that I believe works quite efficiently on the sort of test problems presented here. More justification is needed for the value of the methods implemented in Fluidity than is presented in this paper.

Relatively minor points:

15. In (15), epsilon is a relative error tolerance that is presumably some positive value chosen by the user, so why is epsilon_min needed to ensure the denominator is nonzero?

16. On page 4345 line 15, I am not sure what is meant by imposing different limits on the cell sizes in different directions.

17. Page 4347, line 20 and I am not sure what is meant by the "Sweby limiter". Sweby's paper discussed many limiter such as minmod, superbee, etc., but I am not sure what

limiter is referred to here.

Interactive comment on Geosci. Model Dev. Discuss., 8, 4337, 2015.