A Distributed Memory Adaptive Mesh Refinement Package for inviscid Flow Simulations

Georg Bader and Ralf Deiterding
Institute of Mathematics, Technical University Cottbus, Germany
e-mail: {bader,deiterding}@math.tu-cottbus.de

1 Introduction

Solutions of inviscid fluid flow problems typically show a wide range of spatial and temporal scales. In order to achieve high resolution for the relevant physical phenomena numerical methods on nonuniform grids should be employed. Resolving a shock wave appropriately may require a fairly fine grid, while outside of such a region a coarse grid might be sufficient. The use of nonuniform grids proposes high resolution at moderate computational costs. Many different techniques using nonuniform grids are available.

Moving grid or front tracking methods try to follow the formation and temporal evolution of shocks and other discontinuities. Alternatively, shock capturing methods attempt to capture shocks and other discontinuities on fixed, locally adapted, grids.

There are two alternatives to achieve shock capturing. The first is to refine the grid in a cell-wise fashion. Recursive application of this idea leads to dynamic quadtree or octree data structures in 2 and 3 space dimensions. The storage size of the mere tree is considerable and data access is rather unstructured. This limits the achievable performance signiticantly.

2 Adaptive Mesh Refinement

An alternative approach is to refine regular patches of the computational domain rather than individual cells. The recursive application of this idea allows a high local resolution in parts of the domain. This approach has been introduced by Berger and Oliger [1][2]. Refinement on rectangular patches has the advantage that the data structures remain relatively simple and consist of a nested set of grids. On each grid a standard finite-volume method is used to sweep in a fully regular pattern over the data. Thus, high computational performance can be achieved.

Refinement of individual cells requires more work per cell for advancing the solution one time step, but has the advantage that fewer cells need to be refined since the refinement can be focussed where it is most needed. If a patchwise refinement is used, the cells flagged for refinement need to be clustered into rectangular patches of appropriate size.

In case of hyperbolic conservation laws, the CFL-condition requires a refinement in space and time by the same factor. For instance, refining a 2D problem by a factor of 2 in both space dimensions implies 4 times the grids cells and a halfing of the time step. Consequently, the Berger-Oliger-algorithm follows a recursive time integration scheme.

At an interface between coarse and fine grids we must insure that the discretisations used to update the solution on each grid are consistent with one another. In particular, for conservation laws we must ensure global conservativity of the computed solution. For this a "conservative fixup" has to be applied at the interfaces between coarse and fine grids. A further ingredient for a successful adaptive method is the criterion that marks cells for refinement. For hyperbolic equations approximative gradients are mostly used.

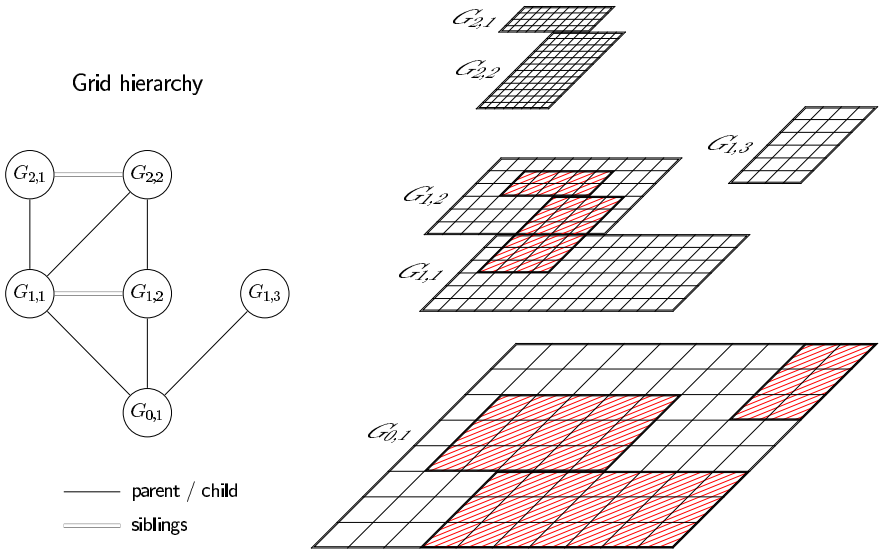

Figure 1: The AMR-algorithm employs a blockstructured refinement strategy.

3 AMR on distributed memory computers

Numerical simulations of realistic fluid flow problems in 2 and especially in 3 space dimensions are limited by the computing power and to a somewhat lesser extent by the storage. Hence, it is natural to try to exploit the enormous computing power of parallel, distributed memory computers.

Today, implementations of AMR-algorithms for single processor computers or parallel computers with shared memory have reached a stable state, see for example [3][4][5]. But, implementations on parallel computers with a distributed memory architecture are still experimental, c. f. [6][8]. Nevertheless, the need for such programms increases, because the evolution in high performance computing points to distributed memory. The most powerful supercomputers (Cray T3E) and also the cheapest parallel computers (workstation-, PC-Cluster) use distributed memory.

Various approaches to parallelize the AMR-algorithm have been proposed [6][7]. In our implementation we follow an idea introduced by M. Parashar [8].

On top of basic classes that implement elementary functionality for a single patch, "distributed" grid functions are defined. The distribution of the data inside these grid functions is done automatically whenever new patches are created. All grid functions are distributed equally with respect to a grid hierarchy. The grid hierarchy does not store any data, but contains the information to derive all necessary relationships between different patches (fig. 1). Consequently, the grid hierarchy and its division to the computing nodes is globally known. The distribution of the grid hierarchy is defined with respect to the coarsest level. Thus, all higher level data resides on the same processor as the coarsest level data. This ensures that operations between different levels (restriction, prolongation) can be carried out strictly local.

Before an appropriate distribution of the grid hierarchy can be found, the workload for every cell of the coarsest level being overlaid by several higher level cells has to be estimated. This estimation is simple, because the AMR-algorithm employs a regular refinement strategy.

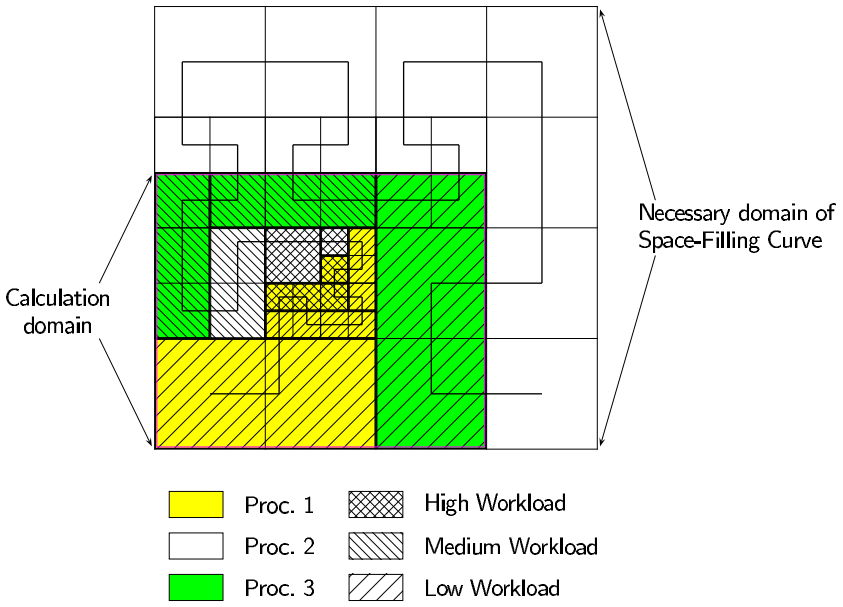

Figure 2: A generalization of Hilbert's space-filling curve is used to distribute grid blocks. The domain of the space-filling exceeds the calculation domain, if the number of cells in x- and y-direction are not of the same power of 2.

The algorithm, which is used for the distribution has to meet several requirements. It must balance the estimated workload, while the data that has to be syncronized during the numerical solution procedure should be as small as possible. The data that has to be redistributed, if the grid hierarchy changes slightly should be few. The distribution algorithm must be fast, because it is carried out on-the-fly. Distribution strategies based on space-filling curves give a good compromise between these different requirements. A space-filling curve defines a continous mapping from $[0,1]$ onto $[0,1]^n$, $n \geq 2$, c. f. [9]. As such curves can be constructed recursively, they are locality preserving. By applying the mapping of a space-filling curve to the discrete index space of the coarsest level, the coarsest level cells become ordered. This sequence can easily be split with respect to equal size yielding a load-balanced distribution of the grid hierarchy. The computational time necessary for distribution can be decreased, if neighboring cells with the same workload are concatenated. In this case, an appropriately generalized space-filling must be employed (See fig. 2)[8].

Our implementation uses C++ for the complex data structures, while operations on a single patch are written in Fortran. This approach combines the clarity of an object-oriented design with the execution speed necessary for highly resolved two- and three-dimensional simulations. This combination is common practice in modern AMR-applications, see also [4][5][6][8].

4 Computational results

As an example we present the simulation of a Kelvin-Hemholtz instability. This kind of fluid instability develops under perturbations of planar contact discontinuities within gas flows. It starts with the well known rollup of the contact line and shows very rich nonstationary

Grid integration	67.0%	
Boundary update	10.0%	(6.5%)
Recomposition	9.5%	(4.5%)
Interpolation	4.6%	
Conservative Fixup	4.0%	
Clustering	2.9%	
Output	0.3%	
Not explicitely measured	1.7%	

Table 1: Breakdown of computational time after 1009 time steps of the Kelvin-Helmholtz instability at $t = 0.025s$. The portions of parallel communication in relation to the whole computational time are displayed in brackets.

structures at later times. See figure 4 for some illustrative snapshots of the Kelvin-Hemholtz instability. It is important to note that this particular example is highly sensitive to changes of the grid size.

The numerical results shown in figure 4 are obtained with a coarse grid of 120×120 cells. The coarse grid is refined over two levels, while the refinement factor is 4 in both space dimensions. This would correspond to a uniform grid of 1820×1820 cells. Carrying out a 2D calculation with 3 212 400 grid cells would be a tremendous waste of resources. Our parallel AMR-code needs a maximum of 1 100 000 cells for this computation. This is obviously a considerable gain. On the other hand, the adaptation process as well as the parallelization produces some overhead. Instead of presenting absolute computing times, which depend on many hardware and implementation details, we present in table 1 the fractions needed in different funtional units of this code. The table shows that the total computing time splits approximately in fractions 2/3 for actual solution time and 1/3 overhead including adaptation process and parallelization.

References

[1] M. Berger, J. Satzman, M. Welcome, *Three-dimensional adaptive mesh for hyperbolic conservation laws* SIAM J. Sci. Comput. 15, No.6, 127-138, 1994.

[2] M. Berger, P. Colella, *Local adaptive mesh refinement for shock hydrodynamics*, J. Comput. Phys. 82, 64-84, 1998.

[3] M. Berger, R. J. LeVeque, *Adaptive mesh refinement using wave-propagation algorithms for hyperbolic systems*, SIAM J. Numer. Anal. 35, No.6, 2298-2316, 1998.

[4] W.Y. Crutchfield, M.L Welcome, *Object-oriented implementation of adaptive mesh refinement algorithms.*, Scientific Programming, 2:145-156, 1993.

[5] H. Friedel, R. Grauer, C. Marliani, *Adaptive mesh refinement for singular current sheets in incompressible magnetohydrodynamics flows*, J. Comput. Phys. 134, No.1, 190-198, 1997.

[6] S. R. Kohn, S. B. Baden, *A parallel software infrastructure for structured adaptive mesh methods*, Proc. of the Conf. on Supercomputing '95, Dec. 1995.

[7] M. Lemke, D. Quinlan, *An object-oriented approach for parallel self adaptive mesh refinement on block structured grids*, Hackbusch, Wolfgang (ed.) et al., Adaptive methods - algorithms, theory and applications. Proceedings of the 9th GAMM-Seminar Kiel, Germany, January 22-24, 1993. Braunschweig: Vieweg. Notes Numer. Fluid Mech. 46, 199-220, 1994.

[8] M. Parashar, J. C. Brown, *System engineering for high performance computing software: The HDDA/DAGH infrastucture for implementation of parallel adaptive mesh refinement*, Structured adaptive mesh refinement grid methods, IMA volumes in mathematics and its applications, Springer, Aug 1997.

[9] H. Sagan, *Space-filling curves*, Springer, New York, 1994.

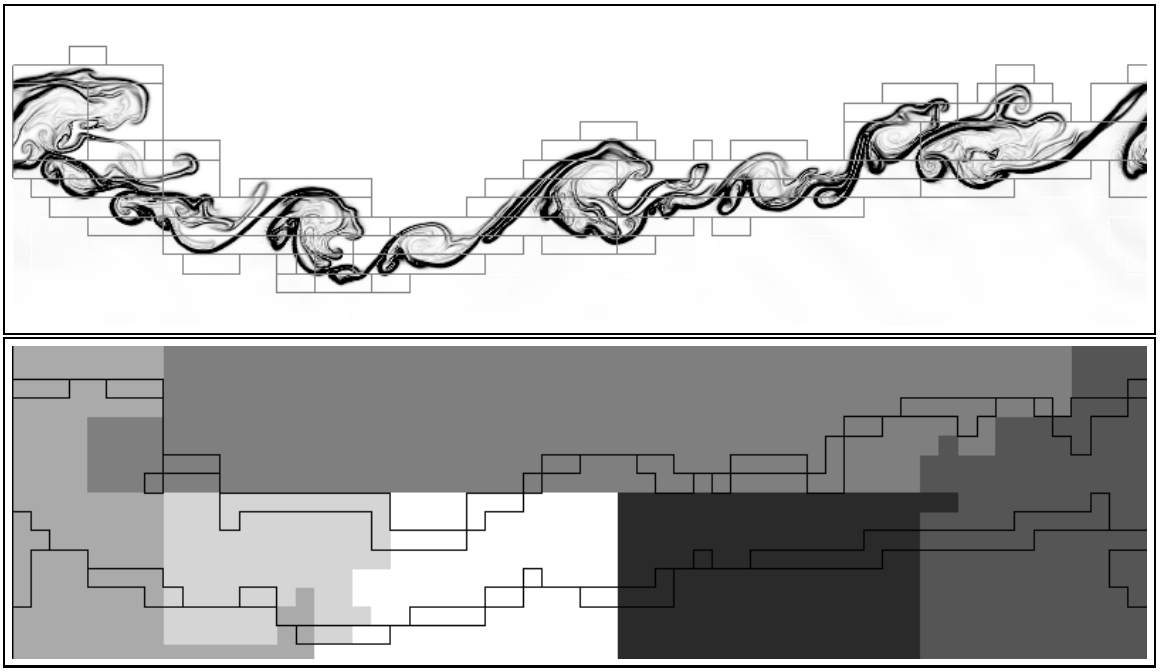

Figure 3: Patches on the 3rd level (top) and distribution on 6 computing nodes (bottom) at $t = 0.00874s$.

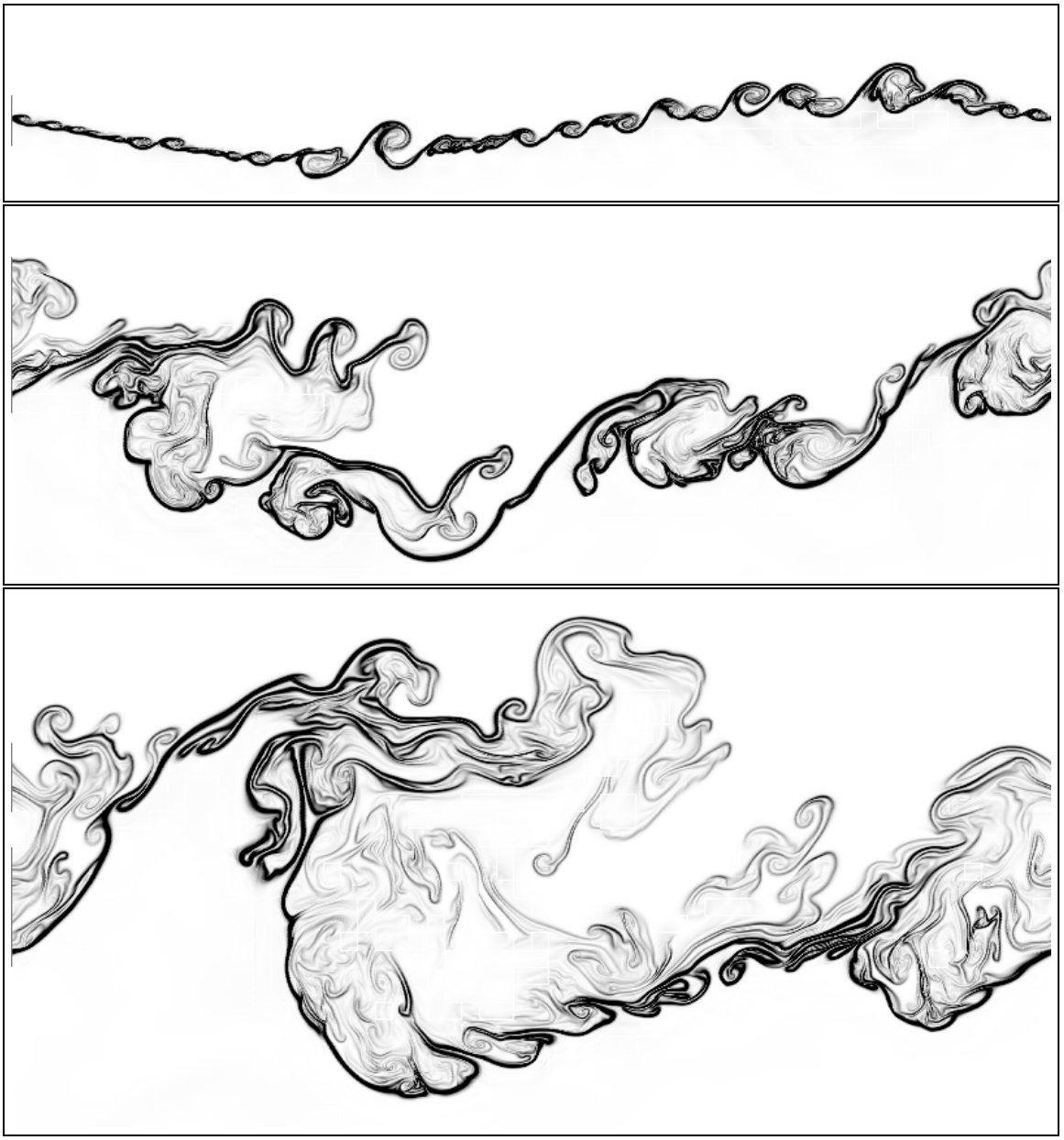

Figure 4: Schlieren plot of density at $t = 0.00375s$, $0.01375s$ and $0.025s$.