OpenMP parallelism for multi-dimensional grid-adaptive magnetohydrodynamic simulations.

R. Keppens[1] and G. Tóth[2]

[1] FOM-Instituut voor Plasma-Fysica Rijnhuizen, P.O. Box 1207, 3430 BE Nieuwegein, The Netherlands
[2] Department of Atomic Physics, Eötvös University, Pázmány Péter sétány 1, 1117 Budapest, Hungary

Correspondence to: keppens@rijnh.nl

Oral presentation at:

2nd International Conference on Computational Science, Amsterdam, The Netherlands, 21st-24th April 2002.

Appeared in:

Lecture Notes in Computer Science **2329**, Proceedings ICCS 2002, P.M.A. Sloot et al. (eds.), 940-949, Springer-Verlag.

OpenMP parallelism for multi-dimensional grid-adaptive magnetohydrodynamic simulations.

R. Keppens[1] and G. Tóth[2]

[1] FOM-Instituut voor Plasma-Fysica Rijnhuizen, P.O. Box 1207, 3430 BE Nieuwegein, The Netherlands
[2] Department of Atomic Physics, Eötvös University, Pázmány Péter sétány 1, 1117 Budapest, Hungary

Abstract. First results on the parallelism achieved by both automated and manually controlled OpenMP[3] programming are reported for 2D and 3D magnetohydrodynamic computations. The simulations exploit adaptive mesh refinement, capable of capturing flow features like shocks and other sharp discontinuities accurately and efficiently. Implementation details are discussed, alongside with their scaling properties on realistic plasma flow simulations.

1 Introduction

This paper focuses on the parallelization of a general-purpose software package – AMRVAC – for grid-adaptive numerical simulations. The AMRVAC package combines the versatility of the Versatile Advection Code (VAC[4]) with the advantages of Adaptive Mesh Refinement (AMR). AMRVAC [5, 4, 8] is particularly suited for time-evolving physical systems governed by (near-) conservation laws, e.g. the ideal magnetohydrodynamic (MHD) equations expressing conservation of mass, momentum, energy, and magnetic flux. The generality of the software resides in (i) a choice of the actual equations to solve for; (ii) a total independence of the implementation on the dimensionality of the problem (1D, 1.5D, 2D, 2.5D and 3D configurations are possible thanks to the LASY syntax [11]); and (iii) the availability of several high-resolution, shock-capturing spatial discretizations. Its grid-adaptivity follows an AMR technique, for which data structures and algorithmic issues handling the automated production of subgrids have been introduced by Berger [1]. Only fairly recently, this solution adaptive regridding strategy has been applied in multi-dimensional MHD simulations [10, 3, 9].

We recall that the AMR process involves the generation and destruction of hierarchically nested grids of subsequently finer resolution. In Keppens et al. [5], we concentrated on algorithmic improvements of the regridding and evaluated obtainable efficiencies by dynamic meshing for a variety of 1D, 2D and 3D problems. This 'efficiency' was characterized by the reduction in serial execution

[3] See `http://www.openmp.org`.
[4] See `http://www.phys.uu.nl/~toth`

times when comparing high resolution static grid simulations with corresponding AMR runs. In the latter simulations, fine meshes are triggered only when and where needed, significantly reducing computing costs. Furthermore, the memory requirements drop accordingly (by an order of magnitude for a 2D hydrodynamic shock problem as shown in [8]), allowing for much more realistic simulations of plasma behavior in regimes where both large-scale and small-scale structures are dynamically important.

1.1 Grid-adaptive magnetofluid simulations

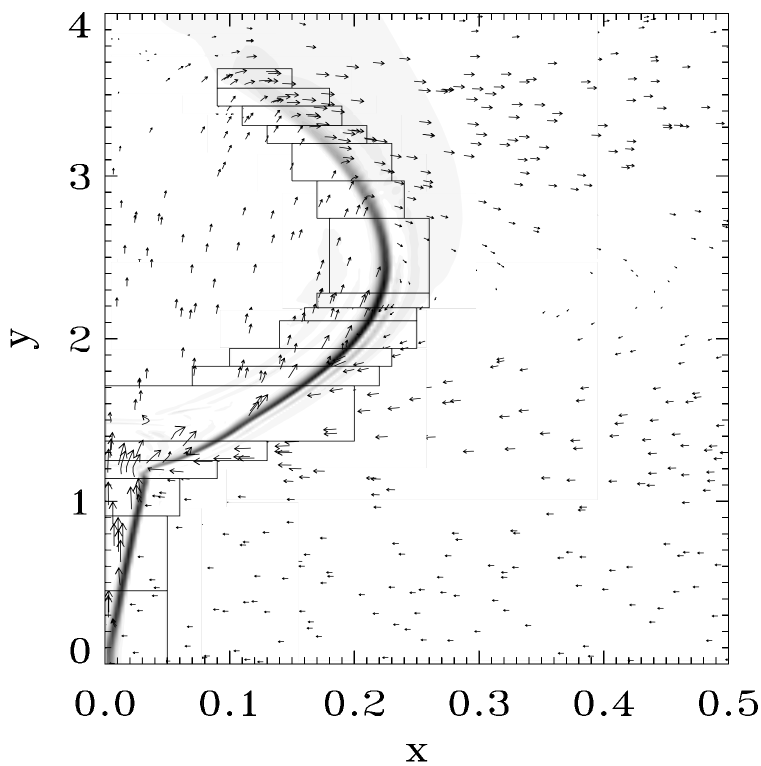

Fig. 1. A snapshot of the density (Schlieren contour plot) and the flow field at time $t = 20$ in a 2D resistive MHD reconnection simulation. The grid structure on the finest level $l = 4$ is indicated.

Figs. 1–2 illustrate two MHD applications which greatly benefit from dynamically evolving meshes. Fig. 1 shows a snapshot of the density structure and the plasma flow field in a 2D magnetic reconnection problem. The setup is taken from [12] and considers a sideways (x-direction) mass inflow from $x = \pm 1$ at an Alfvén Mach number $M_A = 0.04$ and a plasma beta $\beta(x = 1) = 1.5$. The latter

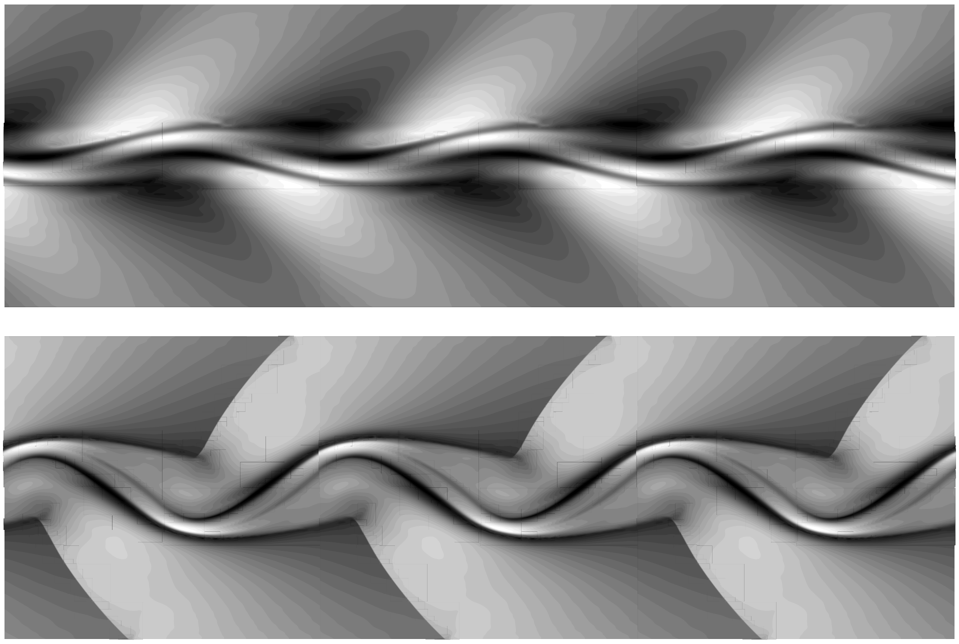

Fig. 2. The evolution of the density in a sinuously unstable, super-Alfvénic wake flow. Three periodic segments are shown at times $t = 110$ and $t = 150$.

quantity measures the ratio of thermal to magnetic pressure $\beta = 2p/B^2$. We start from a static, isothermal equilibrium state where $B_x = 0$, $B_y = \tanh(10x)$ and the pressure profile is given by $p(x) = 0.5\, B_y^2(x=1)[\beta(x=1)+1] - B_y^2(x)/2$. Due to a spatially localized anomalous resistivity which allows for topological changes in the magnetic field structure around the origin, the initial magnetic field variation changes into an X-shaped feature centered on the origin. Symmetry arguments may reduce the simulated domain to a quadrant, the central region of which is shown in Fig. 1. We allowed for 4 grid levels, such that we locally achieve a resolution where $(\Delta x, \Delta y) = (0.0025, 0.005)$. Only the highest grid level structure, which nicely traces the locations of steep gradients, is indicated in the figure. As a second example, Fig. 2 shows the temporal evolution of the density in a supersonic (Mach number $M = 3$), super-Alfvénic $M_A = 5$ wake. The problem description is given by a planar velocity profile $\mathbf{v} = [1 - sech(y)]\hat{\mathrm{e}}_x$ pervaded by a 3D magnetic field configuration with components given by $B_x = A\tanh(y),\ B_z = A\,\mathrm{sech}(y)$. The stability properties of such current carrying, shear flow states were analyzed in [2]. Cross-stream (vertical in the figure) perturbations lead under the chosen parameter values to transverse striations of the wake, accompanied by rightwardly traveling compressive variations reaching far out in the cross-stream direction. These ultimately lead to magnetosonic shock fronts, clearly visible in the density evolution (bottom panel of the figure). The simulation shown in Fig. 2 exploited three grid levels, with a base resolution of 100×200 per periodic segment, locally reaching 800×1600. Again, the finest level grids (not shown) only cover regions with sharp flow features, and at all times maximally covered about 20% of the computational domain.

1.2 Parallel AMR

The data structures used in AMRVAC are discussed in [8]. It should be noted that the Berger [1] approach to create higher level (i.e., finer) grids fits these grids optimally to the flow details, so that possibly many different sized grids are present on a certain grid level (see, e.g. Fig. 1). Their different sizes pose a potential load-balancing problem when parallelizing a mesh refinement code over the grids. This difficulty can be overcome by reformulating the original AMR procedure to enforce the creation of equally sized blocks, as realized in parallel `MPI` implementations in [9, 7]. Without switching to this block-adaptive strategy, virtually shared memory parallelization for `ccNUMA`[5] architectures making use of `OpenMP` offers a viable alternative to parallel grid-adaptive simulations. In [4], first attempts with auto-parallelization on the SGI Origin 3800 were reported: while linear scaling was demonstrated for non-adaptive, Domain Decomposition (DD) usage of AMRVAC, experiments with fully grid-adaptive simulations showed no speedup. In the remaining of this paper, we detail that a suitable manual `OpenMP` parallelization effort is able to achieve fair to good scaling properties for multi-dimensional, multi-level AMR calculations.

2 `OpenMP` parallelization

2.1 Automated versus Manual `OpenMP` parallelism

At a minimal effort, we can rely on the present version (7.3.1.2m) of the MIPSpro Fortran 90 compiler as available on the SGI Origin 3800 at Amsterdam to automatically generate parallel code. This auto-parallelization almost necessarily parallelizes individual `do` loops, which should contain sufficient, parallelizable operations. For AMRVAC, most computations are associated with the discretized explicit time advancing of the set of unknowns (like density ρ, momenta $\rho\mathbf{v}$, energy e, and magnetic field $\mathbf{B}$ for MHD problems) for each individual grid at a certain AMR level. Since this coincides with the lowest level subroutines in the AMRVAC calling tree, we expect the automated parallelization to scale well, as long as there is sufficient work to be done on each grid seperately.

This is indeed confirmed by timing experiments of AMRVAC shown in Fig. 3 for a full 3D MHD jet simulation. The problem is taken from Keppens and Tóth [6], and considers a cylindrical jet flow with $\mathbf{v} = V_o \tanh(r - R_{jet})/0.1 R_{jet} \hat{\mathbf{e}}_x$, in a uniform magnetic field $\mathbf{B} = B_o \hat{\mathbf{e}}_x$, when the shear flow about radius $r = R_{jet}$

[5] `ccNUMA`: cache coherent Non-Uniform Memory Access.

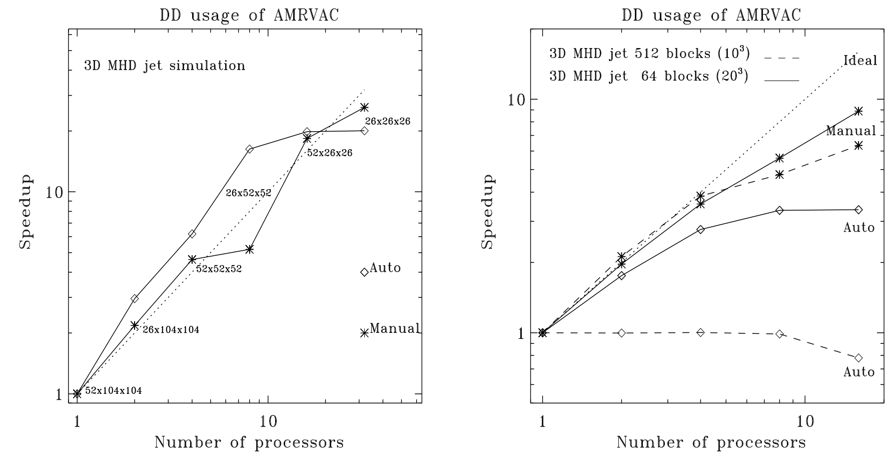

Fig. 3. Scaling behavior of both automated and manual parallelism on a 3D MHD simulation using AMRVAC as a Domain Decompositioner. *Left panel:* changing the number of blocks in accord with the number of processors. *Right panel:* running a fixed number of blocks on a varying number of processors. In both panels, the total problem size is kept constant.

is Kelvin-Helmholtz destabilized. Note that we fixed the number of refinement levels to one, so AMRVAC is only used as a Domain Decompositioner. Hence, the grid does not adjust dynamically, but is merely split into size-controlled equal blocks. The left panel of Fig. 3 shows that the automated parallelization can even achieve superlinear speedup behavior as we split a fixed size 3D problem of resolution $52 \times 104 \times 104$ in N blocks and run the N-block case on N processors. The scaling continues up to 16 processors, but no longer improves when going to 32 processors as the block size of the domains becomes 26^3. The superlinearity at lower processor numbers is due to better cache usage when the block size changes from the single processor run. Unfortunately, in our dimension-independent, general implementation the cache usage is difficult to control actively (as an indication, we probably reach at most 10% of the peak performance on 1 processor for this particular problem).

The right panel in Fig. 3 confirms that the individual block size is the critical factor for parallel efficiency in auto-parallelized code. The same 3D MHD problem was now run on a 80^3 grid, first split into 64 blocks of size 20^3 and subsequently run on 1, 2, 4, 8, and 16 processors. A second timing measurement is also shown, where the 80^3 problem is always split into 512 blocks of size 10^3. While the latter experiment shows no speedup at all, the first one reaches a speedup of 2.7 on 4 processors, with a marginal 'gain' up to 3.3 on 8 processors. Clearly, the low-level parallelism achieved by the compiler is easily lost when grids become too small.

In the following section 2.2, we outline how we made use of manually inserted `OpenMP` directives to bring the parallelism of the code to its natural level, namely to allow for parallel execution over the grids present at any individual level. Both panels of Fig. 3 also show the achieved speedups in this manually parallelized mode for all three DD experiments discussed above. Ideally, since the load balance is optimal in these cases, linear speedup can be obtained. In fact, we reach a speedup of 20 on 32 processors in the blocksize-adjusted experiment (left panel), while the speedup on 16 processors for the 512 and 64 block run is 6.34 and 8.95, respectively. With these scalings, we clearly outperform the automated parallel executions, although there is still room for improvement.

2.2 Implementation details

The manual `OpenMP` parallelization of our adaptive mesh refinement code concentrates on the most time consuming parts of the code: denoting the set of unknowns at time t^n on grid i_g at grid level l as $U_j^n(i_g, l)$, where the subscript j indicates a cell index within the grid i_g, a partial step of an explicit conservative update can be denoted as:

$$U_j^{n+1}(i_g, l) = U_j^n(i_g, l) - \frac{\Delta t^n(l)}{\Delta x(l)} \left(f_{j+1/2}^n - f_{j-1/2}^n \right).$$

In this expression, the numerical fluxes at the cell interfaces $f_{j+1/2}^n$ and $f_{j-1/2}^n$ are calculated from the known time level U^n with a certain stencil in the cell index range j. Note that the temporal $\Delta t^n(l)$ and spatial $\Delta x(l)$ increments vary from level to level. By surrounding each grid i_g on level l with a border of ghost cells conform with the stencil of the flux evaluation, this operation can be done in parallel over the collection of grids i_g on level l. In pseudocode, this is achieved as follows

```
if(level==1)then
  !$omp parallel do &
  !$omp& shared(level,dt) private(igrid) schedule(static)
  do igrid= 1,ngrids
     call process_grid(igrid,level,dt)
  end do
else
  !$omp parallel do &
  !$omp& shared(level,dt) private(igrid) schedule(dynamic)
  do igrid= 1,ngrids
     call process_grid(igrid,level,dt)
  end do
endif
```

The number of grids on level 'level' is given by `ngrids`, and these are indexed by `igrid`. Since the number of grids on level $l = 1$ is fully controlled by setting the overall computational domain and imposing a maximal size for an individual grid at pre-processing, we enforce `static` scheduling there. This ensures a perfect load balance on level $l = 1$. At higher levels, the number of grids varies unpredictably and their size is only controlled in upper bound. Therefore, we resort to `dynamic` scheduling.

While the code segment above is responsible for most of the obtained parallelism, there are in total five instances in the code where work has to be done for all grids at a certain level. Apart from the partial advance step, they correspond to filling the ghost cells for all grids at a certain level, adding a source term, making an error estimate (needed to flag cells which need refinement), and updating all grids at a certain level by information available from overlapping finer level grids. We have inserted `OpenMP` directives to allow for parallel execution of all these five level-wide computations.

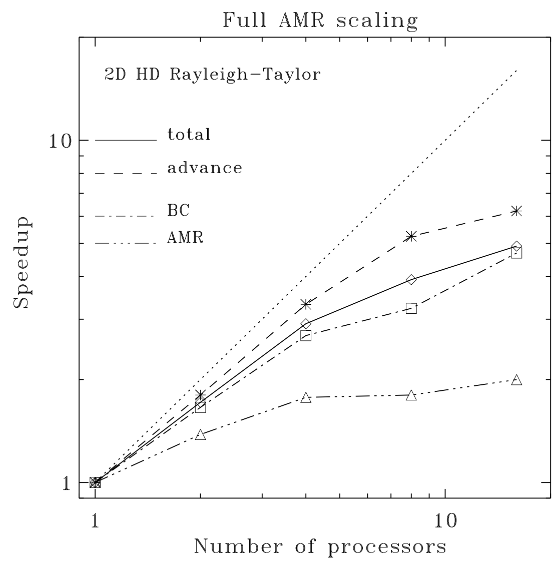

Fig. 4. Scaling of a realistic 2D hydrodynamic simulation exploiting AMR with 5 levels. The total scaling is from timings of actual simulations up to time $t = 0.5$. Individual contributions from time advancing, filling ghost cells ('BC') and error estimation plus updates ('AMR') are indicated.

2.3 Scaling results for full grid-adaptive computations

To evaluate full grid-adaptive parallelism, beyond the DD usage as discussed in section 2.1, we perform timings for a realistic application. We set up a 2D hydrodynamic simulation of a Rayleigh-Taylor unstable configuration where a heavy compressible fluid rests on top of a lighter one (density contrast of 10) in an external gravitational field. We simulate on a $[0,1] \times [0,1]$ domain, with gravity $\mathbf{g} = -\hat{e}_y$ pointing downwards. We let a dense fluid with $\rho_{\mathrm{dense}} = 1$ rest on top of a light fluid with $\rho_{\mathrm{light}} = 0.1$ above the interface $y_{\mathrm{int}} = 0.8 + 0.05 \sin 8\pi x$, so that 4 wavelengths are present in the domain at $t = 0$. The pressure field is set from a centered differenced hydrostatic balance $dp/dy = -\rho$, ensuring that the pressure about y_{int} equals unity. We allow for 5 refinement levels, with a base resolution of 48×48 and refinement ratios between consecutive levels fixed to 2. Regridding is done every sixth timestep per level, and as advocated in [5], we use a different spatial discretization on the finest level than on all underlying levels. To account for the effect of many refinements happening in a true AMR simulation, we perform timings of simulations running till time $t = 0.5$, up to which point 228 time steps have been taken on the coarsest level. To make this a tough test for the load balancing on levels $l > 1$, we enforce a maximal grid size of 24×24, ensuring the creation of many small-sized grids. Not unexpectedly, automated parallelization fails completely on this problem.

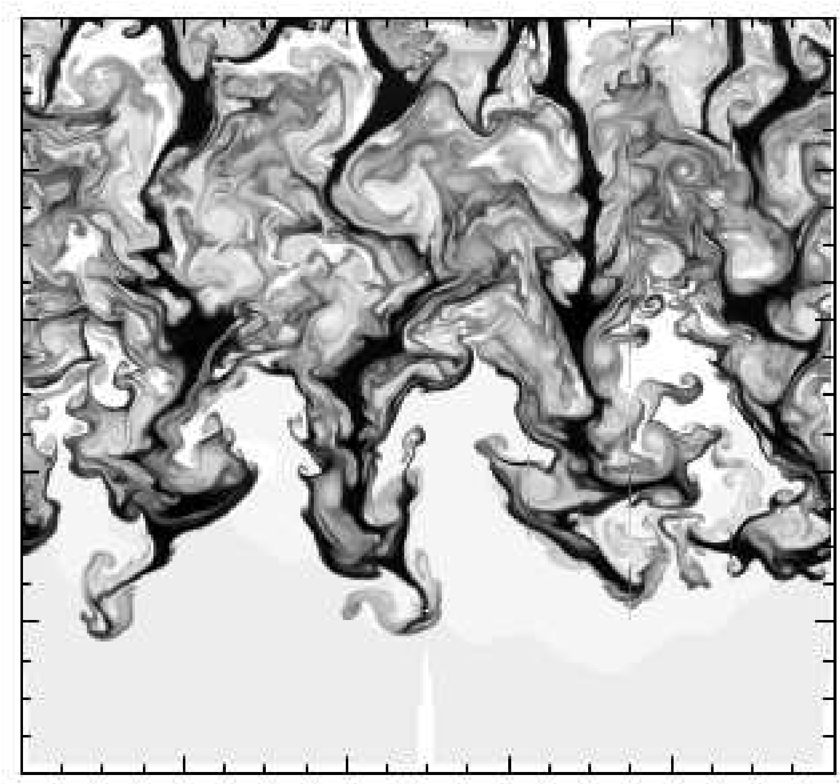

Fig. 5. A snapshot of the simulation used in the scaling experiment from Fig. 4. The density structure on the unit square is shown at time $t = 1.75$.

The result of the timings of this simulation up to 16 processors on the SGI Origin 3800 are shown in Fig. 4. We get a speedup of 2.9 on 4 processors, which only marginally improves up to 4.9 on 16 processors. If we analyze the obtained parallelism over the parallelized code segments discussed above, the most time consuming part – the partial advancing and the source term additions – scales best with a speedup of 6.2 on 16 processors. The ghost cell filling takes up a significant fraction (up to 32 % for the 4 processor run) of the computing time in this experiment due to the small maximal grid size enforced. As this is even more influenced by different grid sizes, this is responsible for the worsened scaling. The part indicated by AMR in Fig. 4 scales even less, but this combined time spent on error estimation and updating coarser from finer levels amounts to only 13 % of the total computing time on 4 processors. Overall, the obtained parallelism is in fact quite encouraging, given the severity of the test. Our timings indicate that for this particular 2D hydro simulation, it is beneficial to use up to 4 processors. We continued the calculation in that fashion and show a snapshot of the density field at time $t = 1.75$ in Fig. 5. Note the intricate small-scale structure developing in this mixing process.

3 Conclusions and Outlook

On the multi-processor `ccNUMA` architecture of the SGI Origin 3800, we evaluated both automated and manually parallelized AMR simulations of 2D and 3D MHD problems. Automated parallelization can achieve linear scaling for non-adaptive, Domain Decomposition approaches as long as the block size per processor is of order 20^3 or higher for the 3D MHD simulations discussed. As yet, automated parallelization is inadequate for multi-level AMR runs. However, a rather straightforward manual parallelization strategy relies on dynamically scheduled `OpenMP` 'threads' to execute the time integration of multiple grids at the same AMR grid level in parallel. With this strategy, we demonstrated fair to good scaling properties for realistic multi-level applications. A more extensive set of timing measurements for a set of problems differing in complexity (pure hydrodynamic or full MHD simulations) and dimensionality (1D through 3D) will be a valuable extension to this work. The AMRVAC package will be used for multi-dimensional magnetized plasma studies where both global and local phenomena play an equally important role in the dynamics. With the ongoing developments in the `OpenMP` effort to become a portable standard for shared memory parallelism, it seems fair to state that `OpenMP` parallelism for dynamically regridded multi-dimensional computations offers a promising alternative to message passing implementations.

Acknowledgements. This work was performed as part of the research programme of the 'Stichting voor Fundamenteel Onderzoek der Materie' (FOM) and Euratom, with financial support from the 'Nederlandse Organisatie voor Wetenschappelijk Onderzoek' (NWO) and computing resources by 'Nationale Computer Faciliteiten'. G.T. has been partly supported by the Education Ministry of Hungary (grant No. FKFP-0242-2000) and the Hungarian Science Foundation (OTKA, grant No. D-25519). R.K. thanks Peter Michielse (SGI Netherlands) for the `OpenMP` workshop (oct. 2001) which led to code improvements.

References

1. Berger, M.J.: Data structures for adaptive grid generation, SIAM J. Sci. Stat. Comput. **7**(3), 904 (1986)
2. Dahlburg, R.B., Keppens, R., Einaudi, G.: The compressible evolution of the super-Alfvénic magnetized wake, Phys. of Plasmas **8**(5), 1697-1706 (2001)
3. Friedel, H., Grauer, R., Marliani, C.: Adaptive mesh Refinement for Singular Current Sheets in Incompressible Magnetohydrodynamic Flows, J. Comput. Phys. **134**, 190-198 (1997)
4. Keppens, R., Nool, M., Goedbloed, J.P.: Zooming in on 3D magnetized plasmas with grid-adaptive simulations, in *Parallel Computational Fluid Dynamics – Recent Developments and Advances*, edited by P. Wilders et al. (Elsevier Science B.V., in press 2002)
5. Keppens, R., Nool, M., Tóth, G., Goedbloed, J.P.: Adaptive mesh refinement for conservative systems: multi-dimensional efficiency evaluation, submitted to J. Comput. Phys. (2001)
6. Keppens, R., Tóth, G.: Nonlinear dynamics of Kelvin-Helmholtz unstable magnetized jets: Three-dimensional effects, Phys. of Plasmas **6**(5), 1461-1469 (1999)
7. MacNeice, P., Olson, K.M., Mobarry, C., de Fainchtein, R., Packer, C.: PARAMESH: A parallel adaptive mesh refinement community toolkit, Comp. Phys. Comm. **126**, 330 (2000)
8. Nool, M., Keppens, R.: AMRVAC: a multidimensional grid-adaptive magnetofluid dynamics code, submitted to Comp. Methods in Applied Math. (2001)
9. Powell, K.G., Roe, P.L., Linde, T.J., Gombosi,T. I., De Zeeuw, D.L.: A Solution-Adaptive Upwind Scheme for Ideal Magnetohydrodynamics, J. Comput. Phys. **154**, 284-309 (1999)
10. Steiner, O., Knölker, M., Schüssler, M.: Dynamic interaction of convection with magnetic flux sheets: first results of a new MHD code, in *Proc. NATO advanced research workshop ASI Series C-433,* Solar Surface Magnetism, edited by R.J. Rutten and C.J. Schrijver, p. 441-470 (Kluwer Dordrecht, 1994)
11. Tóth, G.: The LASY Preprocessor and its Application to General Multi-Dimensional Codes, J. Comput. Phys. **138**, 981 (1997)
12. Tóth, G., Keppens, R., Botchev, M. A.: Implicit and semi-implicit schemes in the Versatile Advection Code: numerical tests, Astron. Astrophys. **332**, 1159-1170 (1998)

This article was processed using the LaTeX macro package with LLNCS style