Seven-League Hydro Code

Philipp V. F. Edelmann, Heidelberger Institut für Theoretische Studien, Schloss-Wolfsbrunnenweg 35, 69118 Heidelberg, Germany

Friedrich K. Röpke, Heidelberger Institut für Theoretische Studien, Schloss-Wolfsbrunnenweg 35, 69118 Heidelberg, Germany

Description of the Code

The Seven-League Hydro (SLH) code is an astrophysical hydrodynamics code with a focus on applications in stellar astrophysics. Its distinguishing features are special low Mach number discretizations of the hydrodynamical fluxes and implicit time stepping. Both enable us to cover the long time scales involved in many problems of stellar evolution.

As the code uses implicit time discretization, we need to solve a nonlinear system of equations at every time step. We use the Newton-Raphson method for this, which involves the solution of a linear system for each Newton iteration. This system is extremely large. For a grid size of 1024^3 cells the number of unknowns is about 5 billion. We use iterative linear solvers, mostly BiCGSTAB, GMRES, and multigrid, for this system. This makes up the main workload in most cases. As the Mach numbers in our simulations are typically $\lesssim 10^{-3}$, it is still more efficient than explicit methods due to the much larger time steps that are possible with an implicit method.

Conventional compressible solvers are known to yield wrong results if used for flows at low Mach numbers. This problem is often fixed by making simplifications to underlying equation (e.g. Boussinesq or anelastic approximation). This causes problems for flows that also include regions that are not strictly in the low Mach regime. Therefore, SLH takes the approach to modify the numerical solver to behave correctly also in the low Mach number limit. The method of choice in SLH is a flux preconditioned Roe solver [1]. This makes SLH essentially an all Mach number code.

SLH is a relatively new code (development started in 2009) written in Fortran 95. We use MPI parallelization with an optional hybrid mode using OpenMP. It does not rely on external libraries except for an implementation of LAPACK. In particular, the implementations of the linear solvers are tailored to the sparse matrix structure that occurs in the solution of the Euler equations.

Output is written in a custom binary format using MPI-IO. Typically one file per time step is used, written using collective MPI-IO calls. There is another mode which writes one file per MPI process.

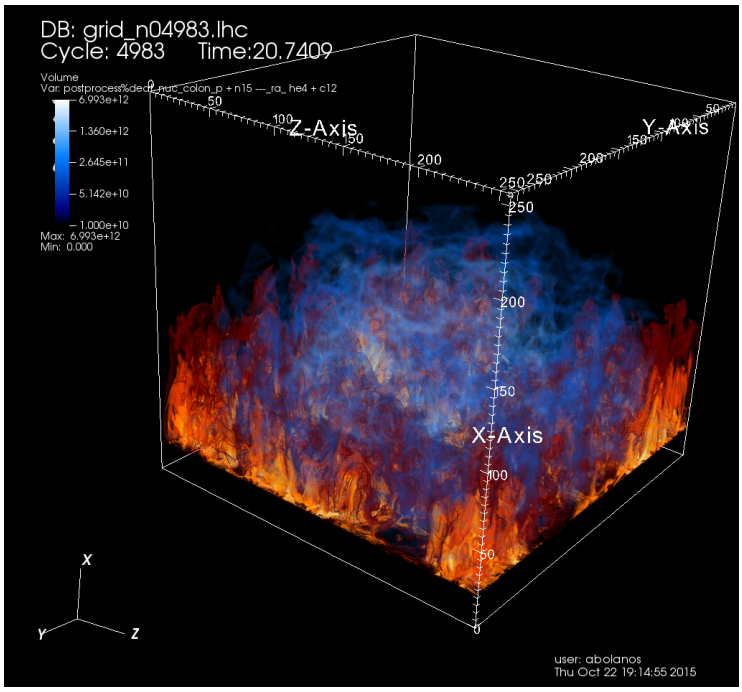

Figure 1: Nuclear reactions in the accreted envelope of a white dwarf star, possible progenitor to a classical nova. The orange color shows the contribution of the reaction $^{12}\mathrm{C}(\mathrm{p},\gamma)^{13}\mathrm{N}$ to energy release, the blue color shows $^{15}\mathrm{N}(\mathrm{p},\alpha)^{12}\mathrm{C}$. Image credit: Alejandro Bolaños (Würzburg University)

Results

During the workshop we were able to run our test setup on all racks of JUQUEEN. Previous tests were only run on up to 8 racks during normal operation. Also we were able to test our I/O routines on up to 16 racks.

The problem setup for the test runs was the Taylor–Green vortex [2, 3], a decaying vortex initial condition, which is often used to study turbulence properties of fluid dynamics schemes. As a part of project HWB07 we previously ran this setup for a range of resolutions (up to 1024^3) and discretizations and analyzed the turbulent energy spectrum and the numerical Reynolds number, especially its behavior at low Mach numbers.

There were no unexpected problems with SLH running in different configurations on JUQUEEN. The fact that SLH uses a static domain decomposition proved to be a small nuisance when scaling to the full machine as the grid size needs to accommodate the number of processes. This could be fixed by allowing a nonuniform distribution of the grid to the MPI ranks but as long as SLH does not include adaptive mesh refinement and load balancing this would likely deteriorate the scaling properties. An alternate approach would be to use less than 16 processes per node. The efficiency of the OpenMP part of the parallelization should be improved in this case. On JUQUEEN SLH currently only uses it for running 4 threads on the same core as this was tested to be the most efficient

configuration.

We performed two main series of scaling tests, one on a 1920^3 grid ranging from 4 to 24 racks and another one on a 2688^3 grid ranging from 14 to 28 racks. The results are shown in Fig. 2 and 3. As it is almost purely a local problem with mostly constant compute time, the computation of the fluxes and source terms (FS) always scales almost ideally. The overall scaling behavior is mostly dominated by the linear solver (LS) component, which takes roughly 90% of the total computation time. The fact that scaling is more than ideal for the first step in the runs with a 1920^3 grid is probably due to the fact that the configuration with the fewest number of nodes used about 0.9 GiB/core, almost all available memory. The deterioration of scaling efficiency at the last data point is possibly due to the non ideal domain decomposition of $192 \times 64 \times 32$ instead of $96 \times 64 \times 64$. The runs with a 2688^3 grid show a promising scaling behavior, reaching 88% of the ideal speed-up.

All the above tests were run without I/O but we performed a separate set of I/O benchmarks by writing the typical output of a 1920^3 grid (about 264 GiB). We tested writing to one large file using MPI-IO and writing to one file per process using standard functions from C stdio. No file system hints were set during the tests. The files were created prior to the measurement, which had a positive impact on performance. The results are shown in Table 3. Writing one file per process generally delivers better performance but it would have to be accompanied by a post-processing step that aggregates the output into one file for practical reasons. Proper use of file system hints could possibly make the *one file* scenario significantly faster.

Table 1: Strong scaling tests on 1920^3 grid. The total memory requirement is 57.5 TiB.

bg_size	rpn	MPI ranks	tpp	threads	GiB/core	time (s)
4096	16	65536	4	262144	0.90	2523.07
8192	16	131072	4	524288	0.45	993.014
12288	16	196608	4	786432	0.30	739.081
16384	16	262144	4	1048576	0.22	631.751
20480	16	327680	4	1310720	0.18	505.144
24576	16	393216	4	1572864	0.15	555.133

Table 2: Strong scaling tests on 2688^3 grid. The total memory requirement is 145.8 TiB.

bg_size	rpn	MPI ranks	tpp	threads	GiB/core	time (s)
14336	16	229376	4	917504	0.60	1472.58
21504	16	344064	4	1376256	0.40	1179.25
28672	16	458752	4	1835008	0.30	835.565

Apart from showing that SLH can scale reasonably to the full machine, participation in the scaling workshop enabled us to learn about new analysis

Table 3: Timings and I/O bandwidth reached when writing 264 GiB of output. Output was either written using MPI-IO to a single file (*one file*) or using C stdio using one file per process (*many files*). The files were create prior to measurement. The 16 rack test was not run in *many files* mode to avoid stress on the file system.

number of racks	4	8	16
walltime in s			
one file	45.3	29.8	54.1
many files	15.9	10.7	
bandwidth in GiB/s			
one file	5.8	8.9	4.9
many files	15.6	24.7	

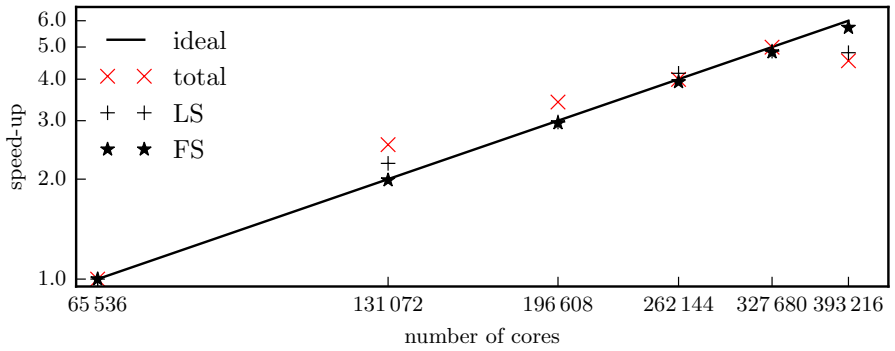

Figure 2: Strong scaling on a 1920^3 grid. The code was run with one MPI process per core and 4 OpenMP threads per process. The different markers show speed-up for the total runtime, the linear solver (LS), and the computation of fluxes and source terms (FS). The point of reference for each is the lowest number of cores. The raw data are given in Table 1.

tools and platform-specific settings, especially with respect to I/O. Additionally, we were able to get advice directly from one of the SIONlib developers, which helped us evaluate its usefulness for our I/O scenario and devise a strategy of integrating it with SLH with minimal programming effort.

Conclusions

The tests during the scaling workshop showed that SLH can scale to the full machine (Fig. 3) at 88% of the ideal speed-up compared to half the machine. Most potential for future improvement is definitely in the linear solver part of the code, which includes most of the collective communication. The I/O tests revealed the fact that SLH achieves better performance when writing one file per process even for more than 10^5 processes. This prompted the decision to include

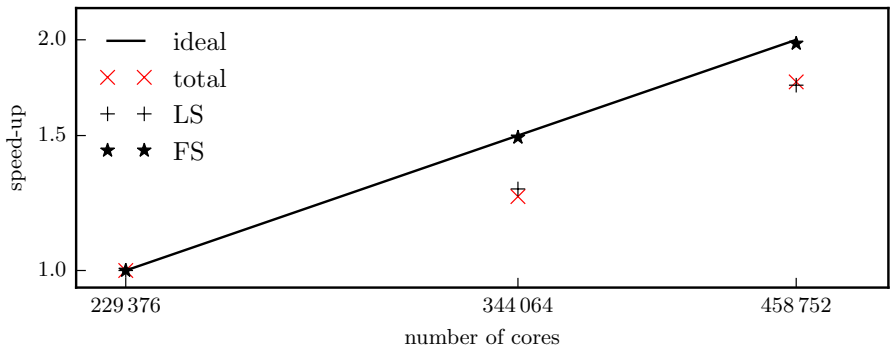

Figure 3: Strong scaling on a 2688^3 grid. The code was run with one MPI process per core and 4 OpenMP threads per process. The different markers show speed-up for the total runtime, the linear solver (LS), and the computation of fluxes and source terms (FS). The point of reference for each is the lowest number of cores. The raw data are given in Table 2.

SIONlib as an additional output method for SLH. We will also investigate the impact of file system hints for MPI-IO output in future tests.

Acknowledgments

This work was supported by the Klaus Tschira Foundation.

References

[1] Miczek, F. and Röpke, F. K. and Edelmann, P. V. F.; *New numerical solver for flows at various Mach numbers*; Astronomy and Astrophysics **576** (2015) A50; [DOI: 10.1051/0004-6361/201425059]

[2] Taylor, G. I. and Green, A. E.; *Mechanism of the Production of Small Eddies from Large Ones*; Royal Society of London Proceedings Series A **158** (1937) 499–521

[3] Drikakis, D. and Grinstein, F. F. and Youngs, D; *Simulation of transition and turbulence decay in the Taylor–Green vortex*; Journal of Turbulence **8** (2007) N20