1 RKL1 Super-Time-Stepping

Super-time-stepping can offer a significant speedup compared to explicit integration for diffusion processes. With super-time-stepping, instead of taking the parabolic time-step,

$$\Delta t_{\text{parabolic}} \propto \frac{\Delta x^2}{D}, \tag{1}$$

we can take the hyperbolic step ($\propto \Delta x$) in a clever way.

1.1 Algorithm

For non-ideal diffusive physics, the RKL1 algorithm allows us to take a super-time-step equal to the hyperbolic time-step,

$$\Delta t_{\text{sts}} = \Delta t_{\text{hyperbolic}}. \tag{2}$$

The super-time-step is taken in s stages, where

$$s = \text{floor}\left[\frac{1}{2}\left(\sqrt{1 + 8\frac{\Delta t_{\text{sts}}}{\Delta t_{\text{parabolic}}}} - 1\right)\right] + 1. \tag{3}$$

For the jth stage in the total s stages, the solution vector, Y, is given as

$$Y_j = \mu_j Y_{j-1} + \nu_j Y_{j-2} + \tilde{\mu_j}\Delta t_{\text{sts}}\mathbf{M}(Y_{j-1}). \tag{4}$$

Above, Y_j can correspond to the conserved quantities u, and if magnetic fields are enabled, the face-centered b. $\mathbf{M}$ can be the hydrodynamic and electromagnetic fluxes, calculated using Y_{j-1}. The coefficients are defined as: $\mu_j = (2j-1)/j$, $\nu_j = (1-j)/j$, and $\tilde{\mu_j} = 2\mu_j/(s^2+s)$.

1.2 Additions

This `sts` branch implements the RKL1 alogirithm for diffusive processes in an operator-split update, before the integration carried out by the `TimeIntegratorTaskList`. When super-time-stepping is enabled, the explicit integration of diffusive processes is disabled.

1.2.1 `-sts` argument

Added `-sts` argument to `configure.py`. If enabled (`STS_ENABLED` in src/defs.hpp), this turns off the explicit integration of diffusion processes and instead turns on operator-split integration using the RKL1 algorithm.

1.2.2 `SuperTimeStepTaskList`

A stand-alone tasklist, `SuperTimeStepTaskList` (`src/task_list/sts_task_list.cpp`), is applied to execute the RKL1 algorithm. `TimeIntegratorTaskList` was not used because:

1. the number of stages in `TimeIntegratorTaskList` is set by the integrator chosen, however, the number of stages in RKL1 varies, depending on the ratio $\Delta t_{\rm hyperbolic}/\Delta t_{\rm parabolic}$.

2. the update procedure is different; for RKL1, we only care about fluxes from non-ideal diffusion. RKL1 does not require as many tasks, and ordering of tasks is different.

3. With a new `SuperTimeStepTaskList`, there are only minor changes to `TimeIntegratorTaskList`.

With this new task list, `nstages` is set by equation (3) in `main.cpp` and `DoTaskListOneStage(pmesh, stage)` executes the jth stage of a super-time-step, given by equation (4).

This task list is very similar to `TimeIntegratorTaskList`, so similar that we use the same `HydroTaskNames` namespace.

1.2.3 `Mesh` Public Variables

It is handy to always have on hand the hyperbolic and parabolic time-steps. Therefore, this `sts` branch adds

```
pmesh->dt_hyperbolic
pmesh->dt_parabolic
```

in the `Mesh` class. If `STS_ENABLED`,

`pmesh->dt=pmesh->dt_hyperbolic`,

unless this timestep would take us past `tlim`. `pmesh->dt_parabolic` is used to compute `nstages` in `main.cpp`. Along with the other time-stepping parameters in the `Mesh` class, this `sts` branch adds:

```
pmesh->muj
pmesh->nuj
pmesh->muj_tilde
```

which are used throughout the RKL1 update, see equation (4).

1.2.4 `phydro->CalculateFluxes_STS()`

Inside the operator-split, we do not want to go through the entire calculate flux routine, instead, we only want fluxes from non-ideal effects. This function calls `phdif->AddHydroDiffusionFlux`, `phdif->AddHydroDiffusionEnergyFlux`, and `phdif->AddPoyntingFlux` as necessary.

1.2.5 `pfield->ComputeCornerE_STS()`

Similarly, inside the operator-split, we do not want to go through the entire calculation of corner electric fields, instead, we only want contributions from non-ideal effects. This function simply calls `pfdif->AddEMF`.

1.2.6 Use of `u, u1, u2` and `b, b1, b2`

Equation (4) shows that we need the Y_{j-1} and Y_{j-2} solutions, therefore we use `u, u1, u2` and `b, b1, b2` throughout our super-time-step. For a given stage j, `u1 and b1` hold Y_{j-1} and `u2 and b2` hold Y_{j-2}. Use of these registers in the `SuperTimeStepTaskList` does not interfere with `TimeIntegratorTaskList`.

1.3 Tests

Interestingly, even though RKL1 is expected to converge to only first-order, it still passes the existing diffusion test problems in `Athena++` (we have added `thermal_attenuation_sts.py` and `linear_wave3d_sts.py` tests to the diffusion test suite).

Because these tests are insufficiently discriminating against expected error convergence, we add four additional tests to the diffusion test suite: `viscous_diffusion.py, viscous_diffusion_sts.py, resistive_diffusion.py, and resistive_diffusion_sts.py`.

These test problems diffuse either a v_y (`viscous*.py`) or B_y (`resistive*.py`) Gaussian distribution in the x coordinate. These have analytic solutions. The test problems do a convergence analysis at a mere two resolutions. A more detailed convergence analysis is shown for both of these test problems on the next page:

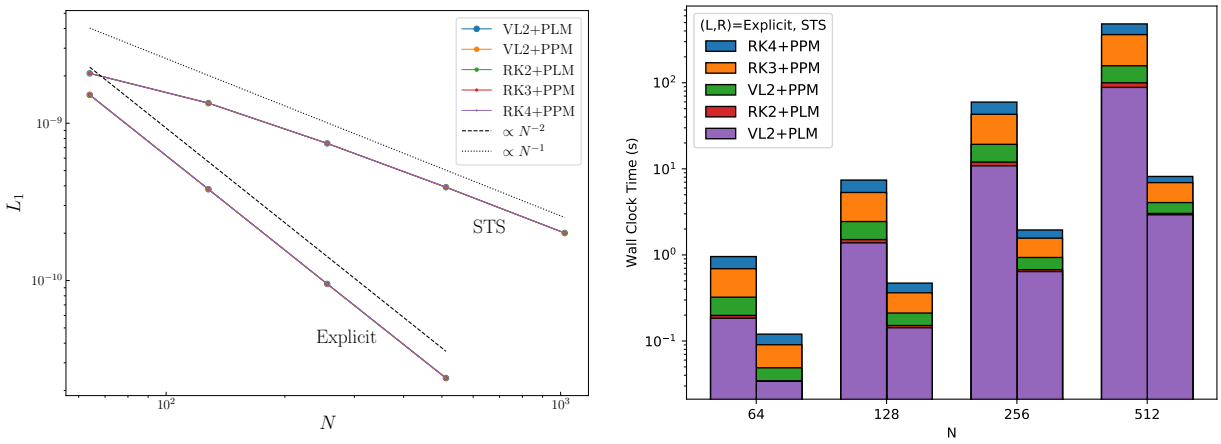

Figure 1: Convergence analysis and performance for Ohmic diffusion of a Gaussian magnetic field distribution.

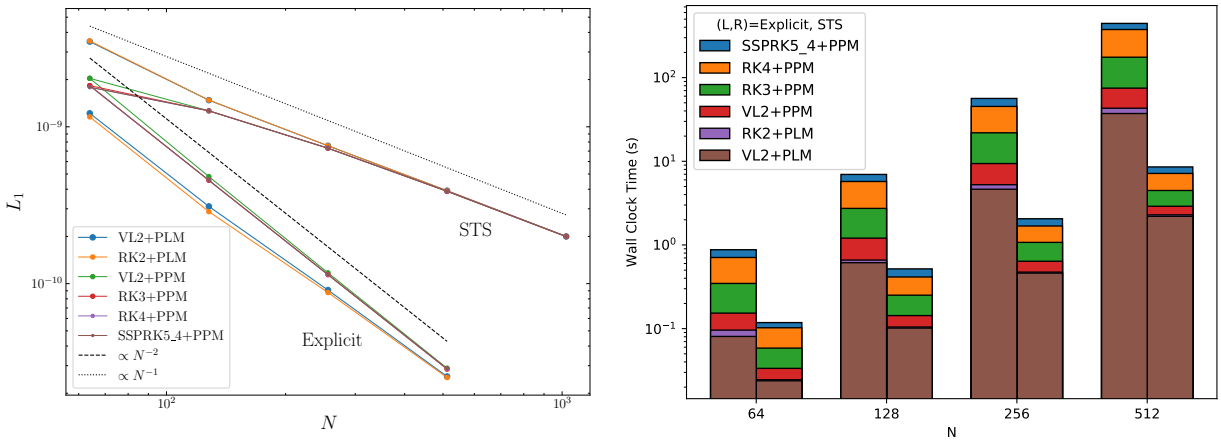

Figure 2: Convergence analysis and performance for viscous diffusion of a Gaussian velocity field distribution.

1.4 Results

The RKL1 algorithm converges at ∼1st order for all integrators and reconstruction methods for both test problems. The treatment of diffusive fluxes in the explicit integrator makes the scheme second-order no matter the temporal integrator or reconstruction method.

Now for performance. Because we are now taking the hyperbolic time-step during the main integration loop when super-time-stepping, siginificantly fewer steps are taken in the `TimeIntegratorTaskList`, i.e., doubling resolution reduces the timestep by a factor of 1/2 in STS, while explicit integration takes the hit of a factor of 1/4. This significantly decreases the wall clock time required for the simulations. Note the y-axes are logarithmic on the bar charts above. Also note: OpenMP and OpenMPI have been tested (for OpenMP, `@kfelker`'s `hotfix/omp_hydro4` branch was used); parallelization over any domain decomposition gives

identical results to a serial (single meshblock) case. For what it's worth, a strong-scaling test on the `linear_wave3d_sts.py` diffusion test problem (resolution of 256×128×128) shows near ideality up to 16 processes (the number of cores on my LANL machine) using purely OpenMPI. This needs to be followed up with more proceses, on more relevant machines (i.e., `stampede2`).

1.5 Incompatibilities/To-Do

1. Time-dependent boundary conditions (e.g., shearing boundary conditions): time-dependent BC's require knowing the "time". The "time" is not well-defined inside an operator-split update; further, "time" is even less well-defined inside a super-time-step stage.

2. Source terms: source terms should be applied given the diffusive fluxes in a super-time-step stage. However, again, "dt" is not really well-defined. The solution to this problem is to wrap the source terms inside the operator $\mathbf{M}$ that acts on Y_{j-1}, see equation (4). Note: this incompatiblity may show up in unexpected places; for example, self-gravity treats updates to energy as source terms, not fluxes. Therefore, if using self-gravity with an adiabatic EOS, `STS_ENABLED` will throw an error, whereas self-gravity with an isothermal EOS works as expected.

3. Non-Cartesian coordinates: Because source term functionality is not yet implemented, coordinate source terms are also not working.

4. Mesh-refinement, working on this...

1.6 Other Issues

1. In some configurations, the only update from diffusive fluxes is to a specific conserved quantity (e.g., only a total energy update to hydro fluxes due to Poynting flux in resistive MHD), however, the present implementation updates all conserved quantities, even though several fluxes are zeroes