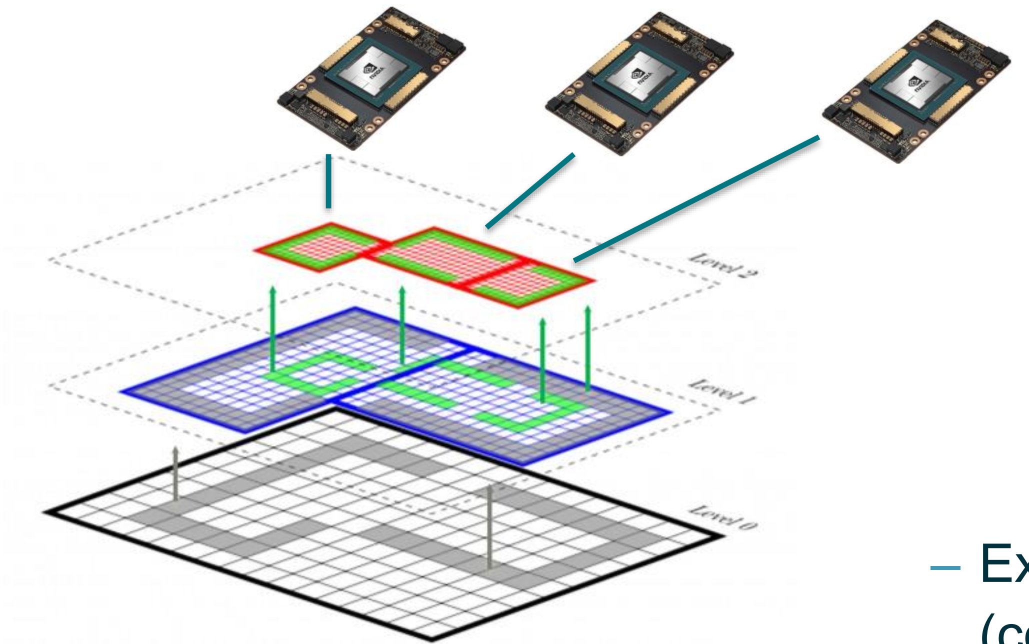
Using SUNDIALS to Efficiently Drive Time-Integration in AMReX-Based PDE Solvers

Andy Nonaka – Lawrence Berkeley National Laboratory

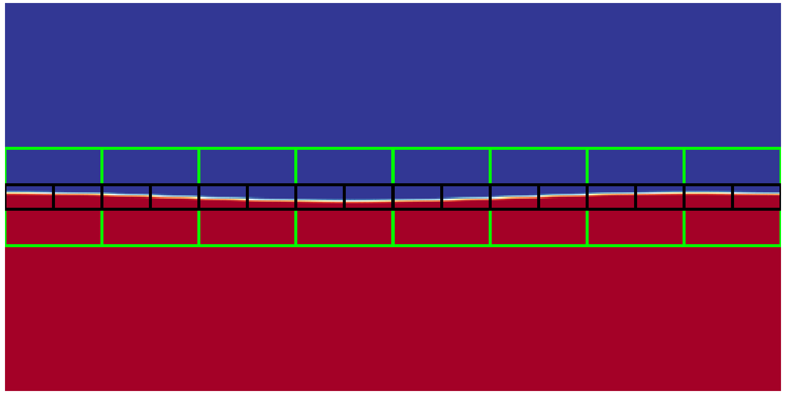
SUNDIALS User Experiences BoF Session | February 12, 2026

AMReX + SUNDIALS

- AMReX is an open-source (github) software framework supporting structured-mesh, spatially-adaptive PDE calculations.
  - Portable, proven scalable performance on NVIDIA, AMD, Intel-based systems.



Numerical libraries for adaptive, multilevel communication and GPU kernel launches



Kelvin-Helmholtz instability demonstrating adaptively refine grids

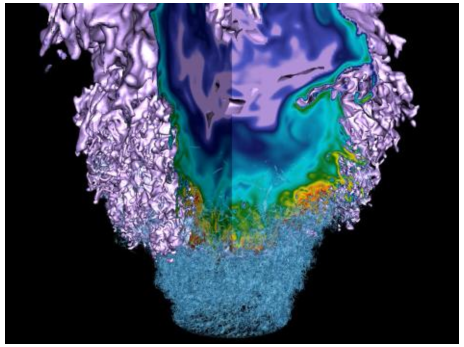
  - Extensive active applications: Fluids (complex, industrial, nanoscale, biological), plasmas, microelectronics, quantum design, epidemiology, astro/cosmology, combustion…

AMReX + SUNDIALS

- AMReX + SUNDIALS-ARKODE time integration allows for extreme flexibility with minimal user overhead
  - ERKStep, ARKStep, and MRIStep supports systems: $\dot{y} = f^E(t, y) + f^I(t, y) + f^F(t, y)$

$f^E(t, y)$	$f^I(t, y)$	$f^F(t, y)$
Slow-nonstiff (large Δt)	Slow-stiff (large Δt)	Fast (small Δt, subcycled within slow partitions)

  - Fluids example: Combustion (advection / diffusion / reaction)
    [see Loffeld et al., 38, *Int. J. High Perform. Comput. Appl.*, 2024 for a detailed study with MRI]
    - Advection: Explicit hydrodynamics (CFL limited by fluid+sound speed)
    - Diffusion: Could be implicit discretization (unconditionally stable)
      - In publication we grouped Diffusion with Advection; compressible hydro (not incompressible) they have similar time scales
    - Reaction: Explicit treatment (limited by very stiff reactions kinetics)



Example: Basic Tutorial Code

github.com/AMReX-Codes/amrex-tutorials

- /ExampleCodes/SUNDIALS/Reaction-Diffusion solves:

$$\frac{\partial \phi}{\partial t} = D\nabla^2\phi - R\phi$$

- Basic anatomy of code:

```
MultiFab phi(ba, dm, Ncomp, Nghost);
```

Container for multidimensional array of data

List of non-overlapping boxes

2D list mapping boxes to MPI ranks

Number of data components

Number of layers of ghost cells

```
TimeIntegrator<MultiFab> integrator(phi);
```

Now we define a TimeIntegrator data type and let the code know we will define various RHSs

```
integrator.set_rhs(rhs_function);
integrator.set_imex_rhs(rhs_function, rhs_implicit_function);
integrator.set_fast_rhs(rhs_fast_function);
```

$$\dot{y} = f^E(t,y) + f^I(t,y) + f^F(t,y)$$

```
integrator.set_rhs(rhs_function);
integrator.set_imex_rhs(rhs_function, rhs_implicit_function);
integrator.set_fast_rhs(rhs_fast_function);
```

$$\frac{\partial \phi}{\partial t} = D\nabla^2\phi - R\phi$$

Sample code defining “rhs_function”

```
auto rhs_function = [&](MultiFab& rhs_in, MultiFab& phi_in) {

    // loop over individual boxes in the MultiFab
    for ( MFIter mfi(phi_data); mfi.isValid(); ++mfi )
    {
        const Box& bx = mfi.validbox(); // extract the box we are working on

        // pointers to MultiFab data
        const Array4<const Real>& phi = phi_in.array(mfi);
        const Array4<Real>& rhs = rhs_in.array(mfi);

        // define the rhs
        // loop over all points in the box (GPU lambda functions)
        amrex::ParallelFor(bx, [=] AMREX_GPU_DEVICE (int i, int j, int k)
        {
            rhs(i,j,k) = D * ( (phi(i+1,j,k) - 2.*phi(i,j,k) + phi(i-1,j,k)) / (dx[0]*dx[0])
                              +(phi(i,j+1,k) - 2.*phi(i,j,k) + phi(i,j-1,k)) / (dx[1]*dx[1])
                              +(phi(i,j,k+1) - 2.*phi(i,j,k) + phi(i,j,k-1)) / (dx[2]*dx[2]) )
                         - R * phi(i,j,k);
        });
    }

};
```

```
integrator.set_rhs(rhs_function);
integrator.set_imex_rhs(rhs_function, rhs_implicit_function);
integrator.set_fast_rhs(rhs_fast_function);
```

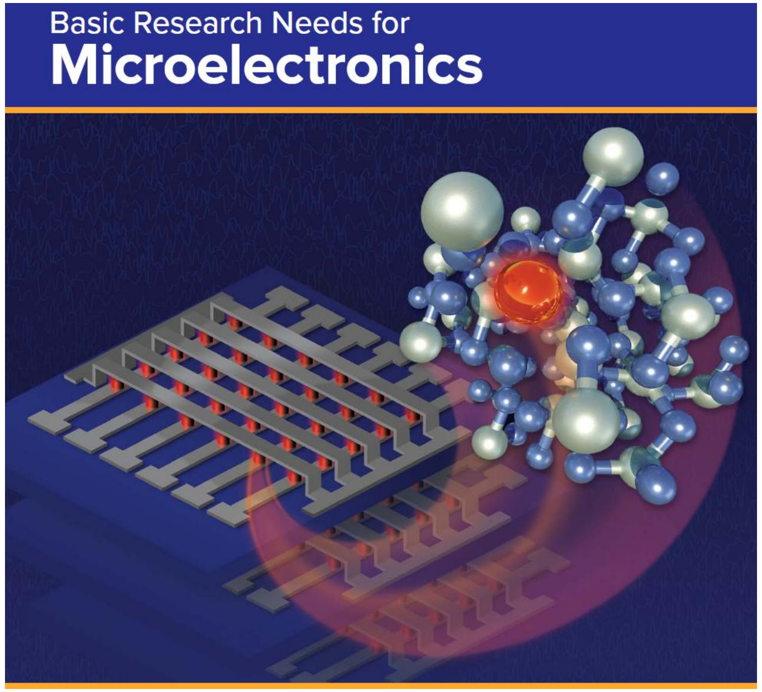
You can define “rhs_implicit_function and “rhs_fast_function” in exactly the same way. Partition the pieces as you desire!

Then, configure your inputs file for the desire integration type, and simply call:

```
for (int step = 1; step <= nsteps; ++step)
{
    time += dt;
    integrator.evolve(phi, time);
}
```

MagneX: Magnetic Materials for Low-Energy Microelectronics

- **Energy Efficiency**: Modeling magnetic materials with the Landau-Lifshitz-Gilbert (LLG) equation enables low-power spintronic devices, reducing energy consumption in microelectronics.
- **High-Speed Operation**: LLG equations help design high-speed magnetic memory and logic devices, enhancing performance compared to traditional electronics.
- **Miniaturization**: Accurate LLG modeling supports the development of nanoscale magnetic components, driving the miniaturization of powerful microelectronic systems.



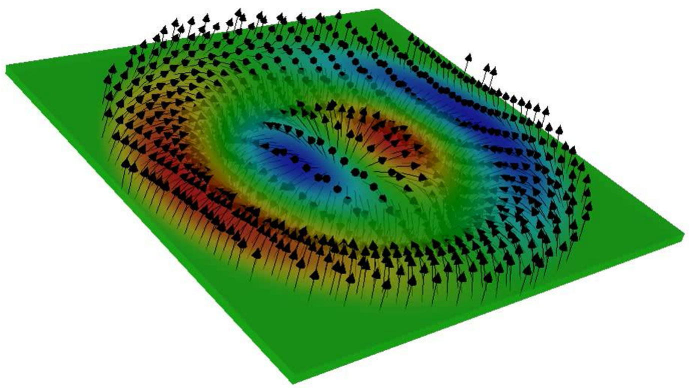
2018 Department of Energy workshop report

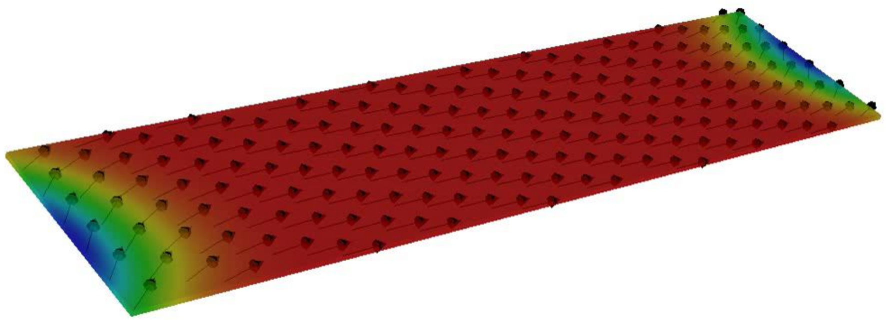
A. Nonaka, Y. Tang, W. Zhang, D. J. Gardner, Z. Yao, et al., **MagneX: A High-Performance, GPU-Enabled, Data-Driven Micromagnetics Solver for Spintronics**, submitted to arXiv, today!

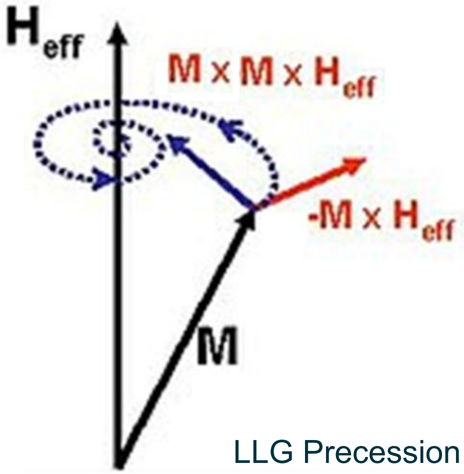
MagneX Equations

- Landau-Lifshitz-Gilbert equation

$$\frac{\partial \mathbf{M}}{\partial t} = \mu_0 \gamma_{\mathrm{L}} (\mathbf{M} \times \mathbf{H}_{\mathrm{eff}}) + \frac{\alpha \mu_0 \gamma_{\mathrm{L}}}{M_s} \mathbf{M} \times (\mathbf{M} \times \mathbf{H}_{\mathrm{eff}})$$







LLG Precession and damping dynamics

MagneX Equations

$$\frac{\partial \mathbf{M}}{\partial t} = \mu_0 \gamma_{\mathrm{L}} (\mathbf{M} \times \mathbf{H}_{\mathrm{eff}}) + \frac{\alpha \mu_0 \gamma_{\mathrm{L}}}{M_s} \mathbf{M} \times (\mathbf{M} \times \mathbf{H}_{\mathrm{eff}})$$

$$\mathbf{H}_{\mathrm{eff}} = \mathbf{H}_{\mathrm{bias}} + \mathbf{H}_{\mathrm{demag}} + \mathbf{H}_{\mathrm{ani}} + \mathbf{H}_{\mathrm{exch}} + \mathbf{H}_{\mathrm{DMI}}$$

- Bias/ Zeeman: user-specified “source term”; functional dependence (cheap)
- Demagnetization: FFT-based linear convolution with demagnetization tensor (expensive) **dominates computational runtime, but NOT stiff**

$$\mathbf{H}_{\mathrm{demag}}(\mathbf{r}) = \int_{\Omega} \mathbf{N}(\mathbf{r} - \mathbf{r}')\mathbf{M}(\mathbf{r}')d\mathbf{r}'$$

- Crystal Anisotropy: Stencil-based (cheap)

$$\mathbf{H}_{\mathrm{ani}} = \frac{2K_u}{\mu_0 M_s^2} (\mathbf{M} \cdot \mathbf{e}_K)\mathbf{e}_K$$

- Exchange Coupling: Stencil-based (cheap) (note: **very stiff** – limits time step)

$$\mathbf{H}_{\mathrm{exch}} = \frac{2}{\mu_0 M_s^2} \nabla \cdot A \nabla \mathbf{M}$$

- DMI Interaction: Stencil-based (cheap)

$$\mathbf{H}_{\mathrm{DMI}} = -\frac{2D}{\mu_0 M_s^2} [(\nabla \cdot \mathbf{M})\mathbf{e}_z - \nabla M_z]$$

MagneX: Increased Efficiency with MRI !

	RK4	ImEx	MRI
Maximum allowable Δt [s]	2.5×10^{-14}	5.0×10^{-15}	1.25×10^{-13}
Time-to-Solution [s]	0.133	0.920	0.069
Exchange evaluations per time step	4	12 (average)	37
Demagnetization evaluations per time step	4	3	3
Time steps	5	25	1
Total exchange evals	20	275	37
Total demagnetization evals	20	75	3

RUNTIME AND FUNCTION EVALUATION STATISTICS OVER A 1.25×10^{-13} [S] SIMULATION INTERVAL.

RK4: Single-rate

ImEx: Implicit demagnetization; second-order

MRI: For both the fast and slow timescales we use the three-stage, third-order explicit method by Knoth and Wolke, where the slow step size is the full time step, Δt, and the fast step size is $0.1\Delta t$

Thank you for attending!

Questions?