GPU Accelerated Linear Solvers for Implicit Time Integrators in the SUNDIALS Library

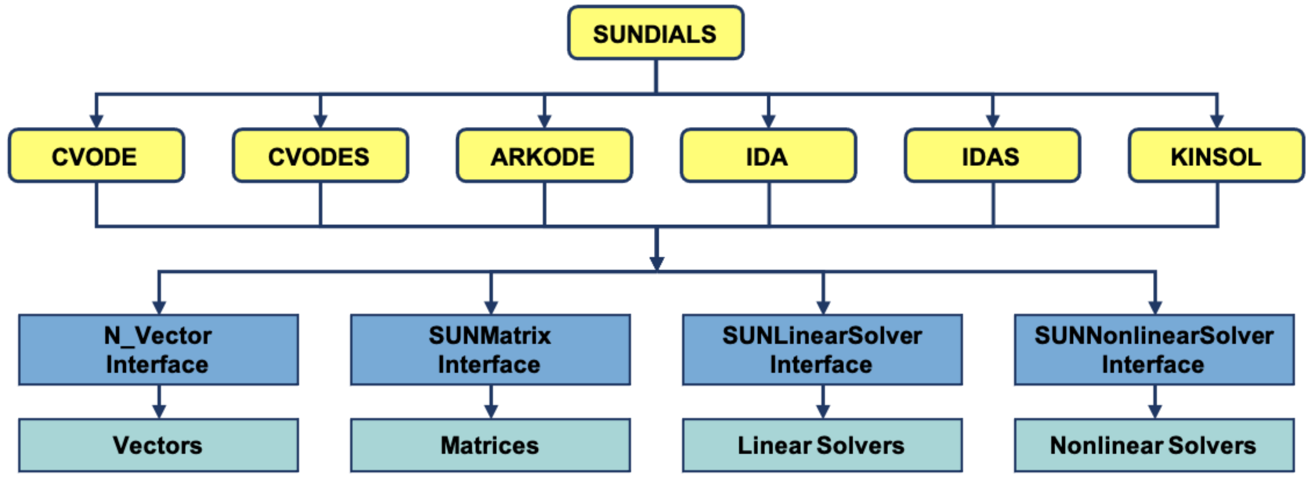
SIAM CSE25, MS9

3/3/25

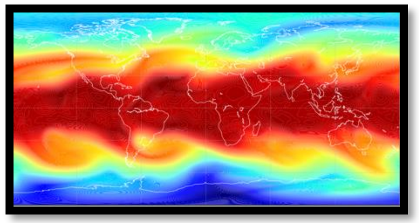
[1]**Cody Balos**, [1]David Gardner, [2]Daniel Reynolds, [1]Steven Roberts, [1]Carol Woodward
[1]Center for Applied Scientific Computing, LLNL
[2]Southern Methodist University

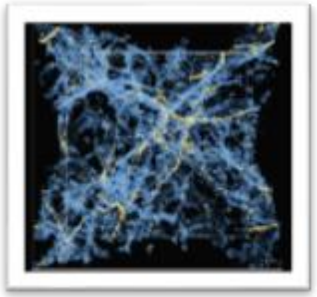
Prepared by LLNL under Contract DE-AC52-07NA27344.

SUNDIALS Overview



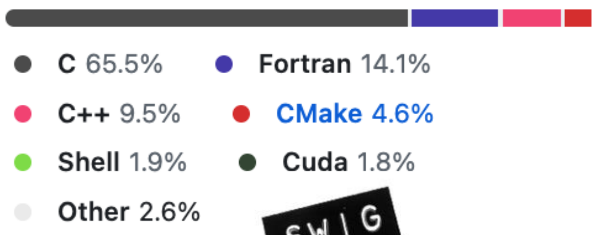
CVODE solves ODEs, $\dot{y} = f(t, y)$, with LMMs, **ARKODE** solves ODEs with one-step methods and supports IMEX + multirate, **IDA** solves DAEs, $F(t, y, \dot{y}) = 0$, **KINSOL** solves nonlinear algebraic systems, **S** variants add sensitivity analysis.


Atmospheric Dynamics


Cosmology

Combustion

Languages:



Continuous Integration with:

Documentation:

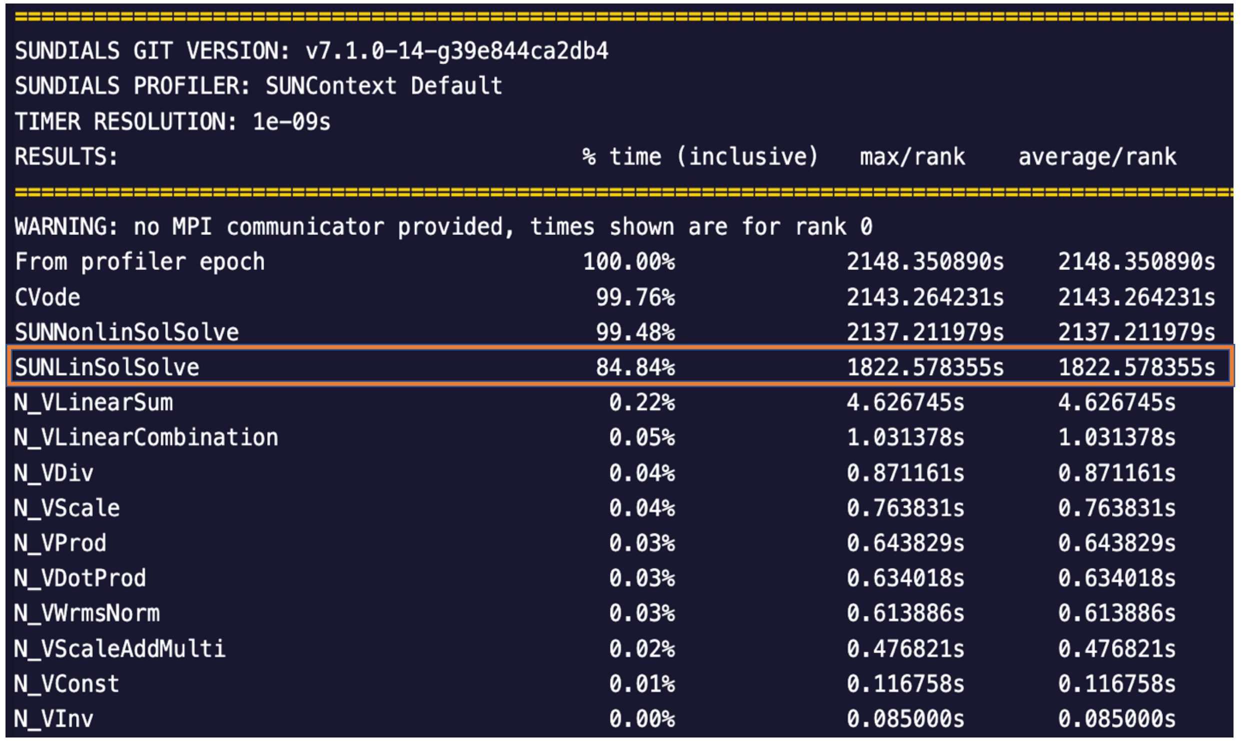
- Optional interfaces to 15 third-party libraries (for solvers and other stuff)!
- Supports NVIDIA, AMD, and Intel GPUs directly and and through RAJA/Kokkos
- Runs on Windows, MacOS, Linux, x86, ARM, Power etc., at a wide range of scales
- Visit computing.llnl.gov/projects/sundials for info

Implicit integrators require efficient linear solvers

- Implicit time integrators are well suited for stiff ODEs, $y'(t) = f(t, y(t))$
- Require us to solve an implicit system iteratively *within every time step*, $F(y^n) = y^n - \gamma f(t_n, y^n) - a_n = 0$
- With modified Newton's method, we must solve a linearized system *within every iteration within every time step*,

$$(I - \gamma J)\delta_{m+1} = -F(y^{n(m)})$$

$$J = \frac{\partial f}{\partial y}, \delta_{m+1} = y^{n(m+1)} - y^{n(m)}$$



```
===========================================================================================
SUNDIALS GIT VERSION: v7.1.0-14-g39e844ca2db4
SUNDIALS PROFILER: SUNContext Default
TIMER RESOLUTION: 1e-09s
RESULTS:                                        % time (inclusive)    max/rank     average/rank
===========================================================================================
WARNING: no MPI communicator provided, times shown are for rank 0
From profiler epoch                              100.00%         2148.350890s    2148.350890s
CVode                                             99.76%         2143.264231s    2143.264231s
SUNNonlinSolSolve                                 99.48%         2137.211979s    2137.211979s
SUNLinSolSolve                                    84.84%         1822.578355s    1822.578355s
N_VLinearSum                                       0.22%            4.626745s       4.626745s
N_VLinearCombination                               0.05%            1.031378s       1.031378s
N_VDiv                                             0.04%            0.871161s       0.871161s
N_VScale                                           0.04%            0.763831s       0.763831s
N_VProd                                            0.03%            0.643829s       0.643829s
N_VDotProd                                         0.03%            0.634018s       0.634018s
N_VWrmsNorm                                        0.03%            0.613886s       0.613886s
N_VScaleAddMulti                                   0.02%            0.476821s       0.476821s
N_VConst                                           0.01%            0.116758s       0.116758s
N_VInv                                             0.00%            0.085000s       0.085000s
```

Execution profile for In Medium Similarity Renormalization Group simulation using implicit method in SUNDIALS CVODE.

SUNDIALS integrators have time step adaptivity

- Based on the method, estimate the time step error

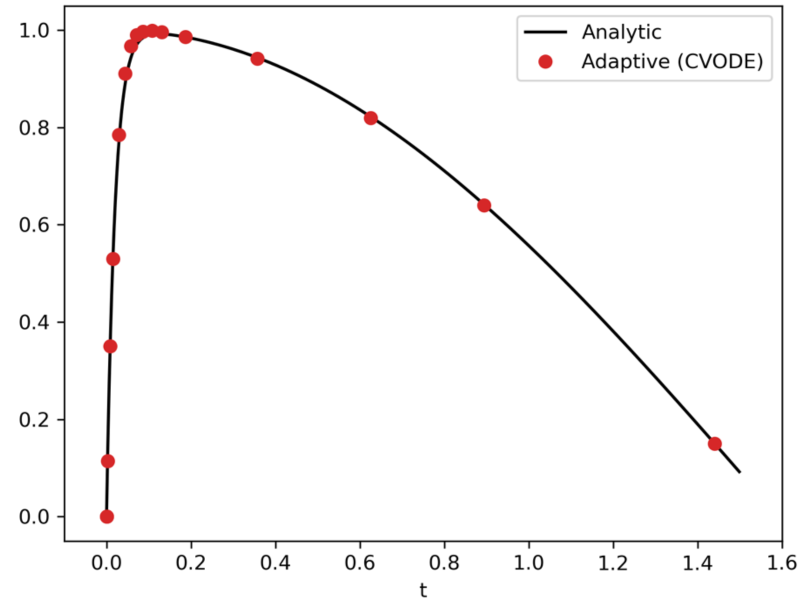
$$\Delta_n \equiv y^{n(m)} - y^{n(0)}$$

  - Accept step if $||\Delta_n||_{WRMS} \leq \epsilon$; Reject it otherwise

$$\|y\|_{\text{wrms}} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(w_i\ y_i)^2} \qquad w_i = \frac{1}{RTOL|y_i| + ATOL_i}$$

  - Choose next step, so that $||y^n - y^{n(0)}||_{WRMS}$ should be small

- ARKODE supplies advanced “error controllers” to adapt these step sizes while also meeting other objectives:
  - minimize failed steps
  - maximize step sizes
  - maintain smooth transitions in the step sizes



Adaptivity can give much more efficient (and accurate) results

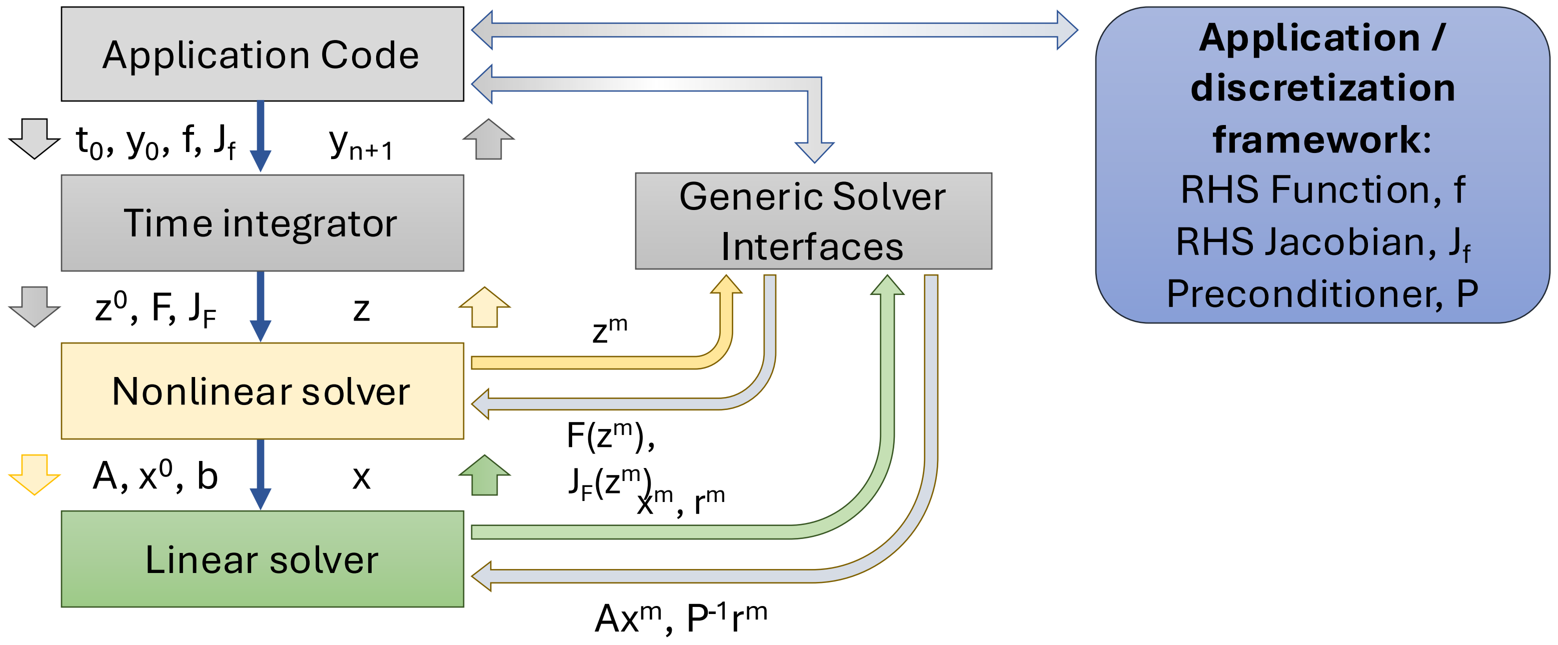
Integrator tolerances propagate down to solvers to keep the error that propagates up small

- Overall goal: keep the nonlinear iteration error from interfering with the local error control
- Stopping criteria for the nonlinear solver: $R||\delta_m||_{WRMS} \leq 0.1\epsilon$,
- The stopping criteria for iterative linear solvers:

$$||(I - \gamma J)\delta_m + F\left(y^{n(m)}\right)||_{WRMS} < 0.05\ (0.1\epsilon)$$

- When iterative linear solver libraries only allow 2-norm*, we must ”translate” our stopping criteria into a 2-norm tolerance
  - It would be great if we could provide solvers our norm

SUNDIALS keeps state data on the GPU



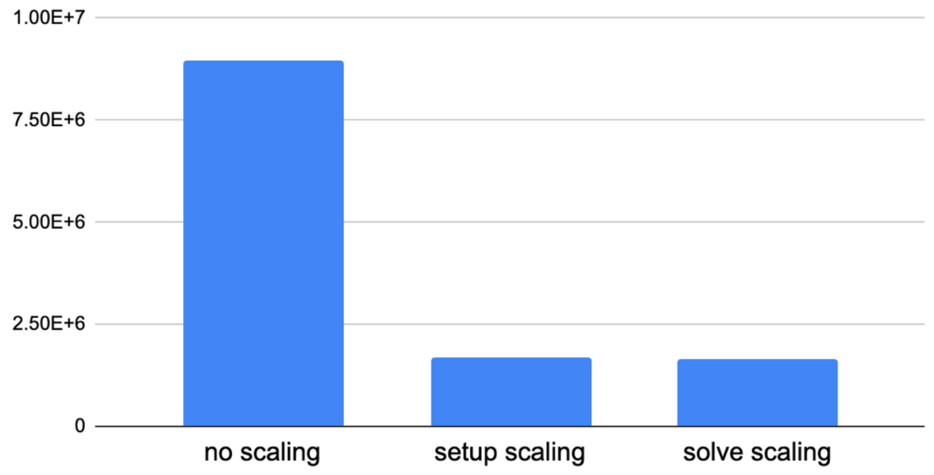
- Integrator logic runs on the CPU, but state data is on the GPU throughout – so solvers receive data on the GPU and should return it to us on the GPU
- We have a memory “helper” API to access application memory pools – solver libraries could benefit from exploiting this too

SUNDIALS iterative “matrix-free” linear solvers

- SUNDIALS has 5 built-in iterative matrix-free linear solvers: CG, BiCGSTAB, GMRES, FGMRES, PTFQMR
- All leverage scaling and preconditioning to balance error between solution components and accelerate convergence
$$(S_1 P_1^{-1} A P_2^{-1} S_2^{-1}) S_2 P_2 x = S_1 \; P_1^{-1} b$$
- S_1 and S_2 are diagonal matrices of scaling factors – within our implicit integrators they are equivalent to diag(ATOL)
- Since these iterative methods only require vector operations, they are GPU-enabled the moment you use a GPU N _Vector

We have an interface to Ginkgo for matrix-based iterative solvers

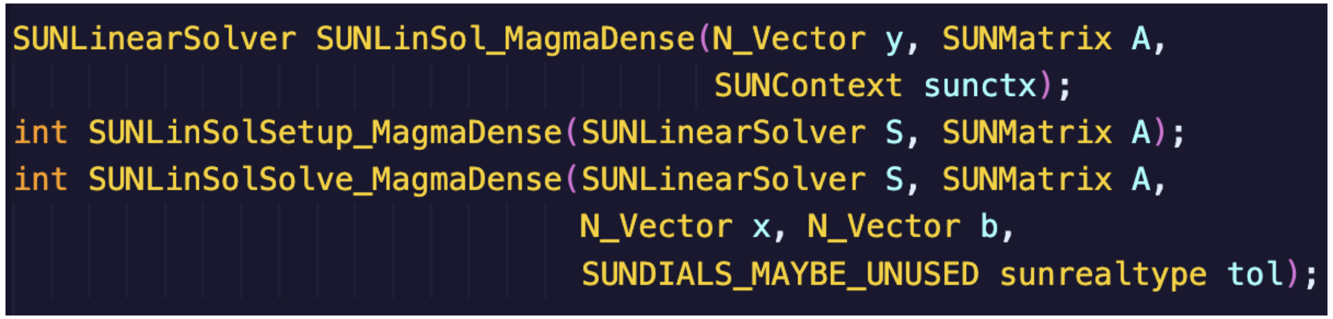
- Our first interface to an iterative solver library
  - Can access most (maybe all) of the iterative solvers in Ginkgo: CG, BiCGSTAB, GMRES, …
  - SUNMatrix implementation uses Ginkgo for matrix storage and operations (sparse or dense)
  - Can use any of Ginkgo's executors (CPU or GPU)
  - Presents as a C++ interface to our users (atypical)
- Issues we encountered resolved by Ginkgo team:
  - Needed matrix operation: $cA + I$
  - Lack of scaling capability hurt performance
  - Couldn't change stopping criteria efficiently



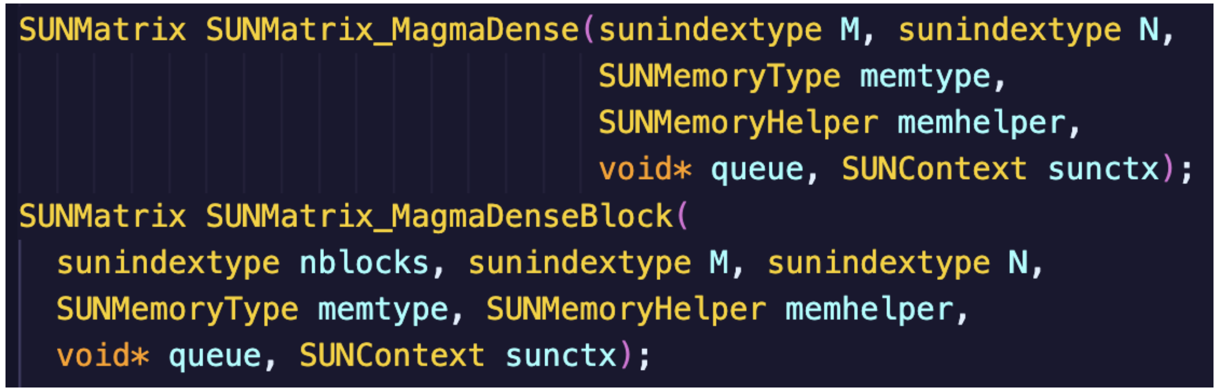
Comparison of Ginkgo batched GMRES in Pele reacting flow problem without scaling (left), with scaling only during matrix setup, and with scaling during every solve. Y axis is the number of linear iterations.

Interface to MAGMA

- We have a unified interface to MAGMA for GPU-enabled dense direct methods for batched and non-batched linear systems
- MAGMA based SUNMatrix and SUNLinearSolver implementations
- Since we lag our Jacobian, we can reuse factorizations across solves
- Call getrf only when Jac. is updated
- Call getrs every nonlinear iteration



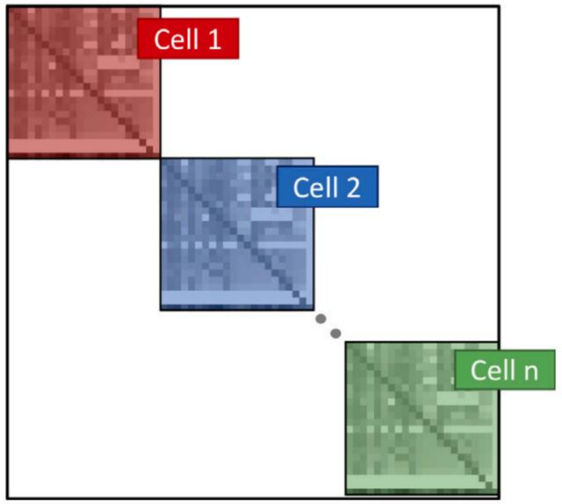
```
SUNLinearSolver SUNLinSol_MagmaDense(N_Vector y, SUNMatrix A,
                                     SUNContext sunctx);
int SUNLinSolSetup_MagmaDense(SUNLinearSolver S, SUNMatrix A);
int SUNLinSolSolve_MagmaDense(SUNLinearSolver S, SUNMatrix A,
                              N_Vector x, N_Vector b,
                              SUNDIALS_MAYBE_UNUSED sunrealtype tol);
```



```
SUNMatrix SUNMatrix_MagmaDense(sunindextype M, sunindextype N,
                               SUNMemoryType memtype,
                               SUNMemoryHelper memhelper,
                               void* queue, SUNContext sunctx);
SUNMatrix SUNMatrix_MagmaDenseBlock(
  sunindextype nblocks, sunindextype M, sunindextype N,
  SUNMemoryType memtype, SUNMemoryHelper memhelper,
  void* queue, SUNContext sunctx);
```

Batched systems arise in reacting flow simulations

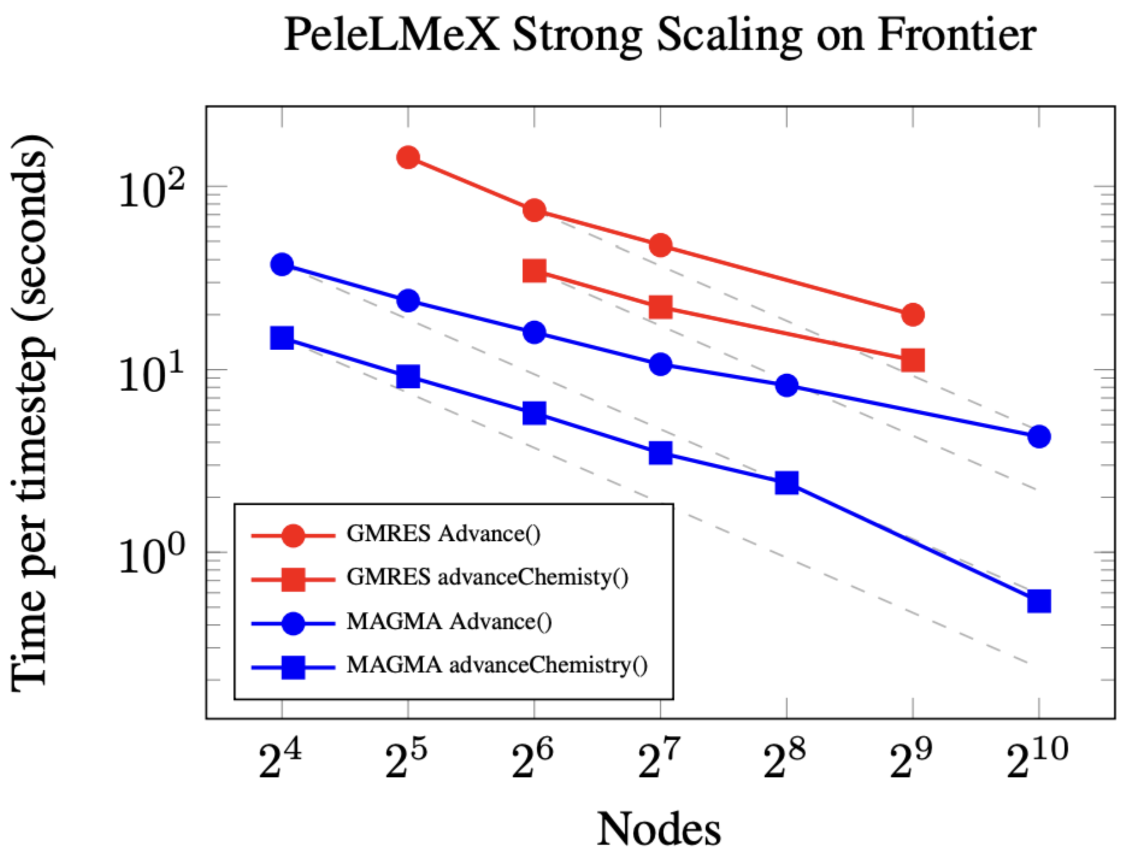
- Pele combustion codes simulate reacting flows, $\frac{\partial u}{\partial t} = F + R$, where R is reaction term
- Operator splitting approach yields independent ODEs for R in each mesh cell
- We batch the ODEs together for integration
  - Increases work for GPU
  - The Jacobian becomes block diagonal
  - Each block has the same sparsity pattern
- Batched linear solvers provide as much as 10x speedup over non-batched in Pele



$$\mathbf{A} = \begin{bmatrix} \mathbf{A_1} & 0 & \cdots & 0 \\ 0 & \mathbf{A_2} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \mathbf{A_n} \end{bmatrix}$$

Reacting flow simulation on Frontier with MAGMA

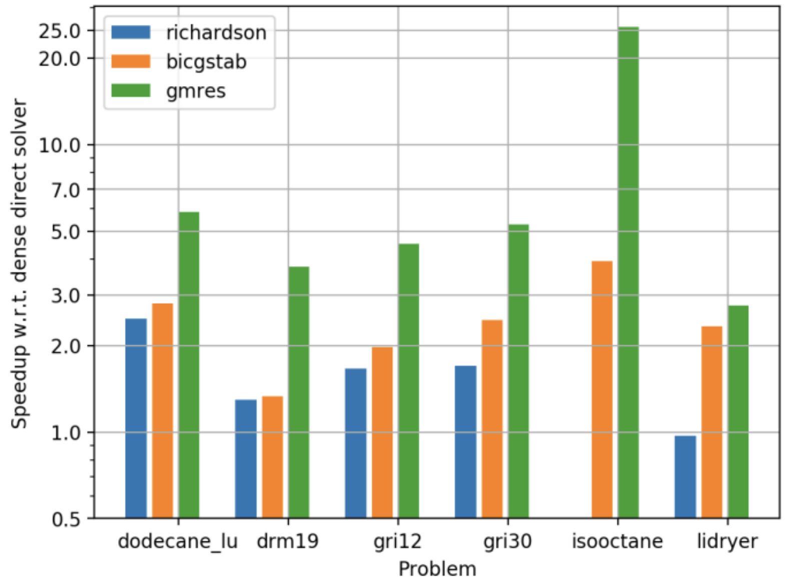
- Lucas Esclapez conducted PeleLMeX simulation strong scaling test on up to 1024 nodes of the Frontier supercomputer
- Compared unpreconditioned, non-batched SUNDIALS GMRES to batched LU in MAGMA
- *advanceChemistry()* routine is the ODE integration



https://doi.org/10.1177/10943420241280060

We worked with Ginkgo on batched iterative solvers

- With our consultation, Ginkgo team developed batched iterative solvers (Richardson, BiCGSTAB, GMRES)
- Data always on the GPU, with 3 total kernel launches (scale, solve, unscale)
- Monolithic solve kernel allows one matrix block per GPU threadblock and allows shared memory to be exploited
- https://doi.org/10.1109/ScalA54577.2021.00010 for details



Problem	Size	Non-zeros (A)	Non-zeros (L+U)
dodecane_lu	54	2,332 (80%)	2,754 (94%)
drm19	22	438 (90%)	442 (91%)
gri12	33	978 (90%)	1,018 (93%)
gri30	54	2,560 (88%)	2,860 (98%)
isooctane	144	6,135 (30%)	20,307 (98%)
lidryer	10	91 (91%)	91 (91%)

Solver performance compared to dense direct batched method from cuSOLVER for different matrices extracted from Pele. Credit Aggarwal et al.

Summary and Conclusions

- Implicit time integrators need efficient linear solvers
- The linear solves are are repeated within a nonlinear iteration within every time step – information can be reused across solves
- Error based time adaptivity affect the solver stopping criteria – would be great if linear solver libraries allowed different norms
- To achieve good GPU performance – solvers must receive the state data on the GPU and give the solution back to us there
- Batched linear solvers allows us to batch ODEs to saturate GPU

computing.llnl.gov/sundials

This material is based upon work supported by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research, Scientific Discovery through Advanced Computing (SciDAC) program through the FASTMath Institute.

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344. LLNL-PRES-2003100.

Questions?

CASC
Center for Applied Scientific Computing