Parallel in Time with SUNDIALS and XBraid

David J. Gardner[1], Robert D. Falgout [1], Daniel R. Reynolds[2], and Carol S. Woodward[1]

[1] *Center for Applied Scientific Computing, LLNL*
[2] *Department of Mathematics, SMU*

PinT 2020 – 9th Parallel-in-Time Workshop

Lawrence Livermore National Laboratory

CASC
Center for Applied Scientific Computing

LLNL-PRES-811622

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under contract DE-AC52-07NA27344. Lawrence Livermore National Security, LLC

SUNDIALS: SUite of Nonlinear and Differential / ALgebraic equation Solvers

- Software library of ODE and DAE time integrators and nonlinear solvers
  - Consists of six packages: CVODE(S), **ARKode**, IDA(S), and KINSOL
  - Written in C with interfaces to Fortran
  - *Designed to be easily incorporated into existing codes*
- Modular implementation
  - Data use is fully encapsulated by vector and matrix APIs
  - Nonlinear and linear solvers are fully encapsulated from the integrators
  - All parallelism is encapsulated in vectors, solvers, and user-supplied functions
  - *Vector, matrix, and solver modules can all be user-supplied*
- Availability and support
  - Freely available; BSD 3-Clause license; >27,000 downloads in 2019
  - Detailed user manuals and an active user community email list

https://computing.llnl.gov/sundials

ARKode – Adaptive Runge-Kutta methods

- The ARKode package is designed to work as an infrastructure for developing adaptive one-step time integration methods.
  - **ARKStep:** Provides ERK, DIRK, and IMEX ARK methods for problems of the form

$$M\frac{dy}{dt} = f^E(t,y) + f^I(t,y), \qquad y(t_0) = y_0$$

  - **ERKStep**: A streamlined module for ERK methods for problems of the form

$$\frac{dy}{dt} = f(t,y), \qquad y(t_0) = y_0$$

  - **MRIStep:** Multirate Infinitesimal Step (MIS) like methods for problems of the form

$$\frac{dy}{dt} = f^F(t,y) + f^S(t,y), \qquad y(t_0) = y_0$$

ARKStep – Additive Runge-Kutta methods

- IMEX ARK methods for problems with two split components:

$$M\frac{dy}{dt} = f^E(t,y) + f^I(t,y), \qquad y(t_0) = y_0$$

  - M is any nonsingular linear operator (mass matrix, typically $M = I$),
  - $f^E(t,y)$ contains the explicit (non-stiff) terms,
  - $f^I(t,y)$ contains the implicit (stiff) terms.

- Combine two s-stage RK methods; let $t^*_{n,j} = t_n + c_j h_n$, $h_n = t_{n+1} - t_n$:

$$Mz_i = My_n + h_n \sum_{j=1}^{i-1} A^E_{i,j} f^E\left(t^E_{n,j}, z_j\right) + h_n \sum_{j=1}^{i} A^I_{i,j} f^I\left(t^I_{n,j}, z_j\right), \qquad i = 1, \ldots, s$$

$$My_{n+1} = My_n + h_n \sum_{j=1}^{s} b^E_j f^E\left(t^E_{n,j}, z_j\right) + b^I_j f^I\left(t_{n,j}, z_j\right) \qquad \text{(solution)}$$

$$M\tilde{y}_{n+1} = My_n + h_n \sum_{j=1}^{s} \tilde{b}^E_j f^E\left(t^E_{n,j}, z_j\right) + \tilde{b}^I_j f^I\left(t_{n,j}, z_j\right) \qquad \text{(embedding)}$$

Time steps are chosen to minimize local truncation error and maximize efficiency

- All SUNDIALS integrators utilize local truncation error estimates to adapt the step size to meet user defined tolerances

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

  - With Runge-Kutta methods the error estimate is based on the difference between the computed and embedded solutions
  - Accept the step if $\|E(h_n)\|_{wrms} < 1$, otherwise reject the step
  - Choose h_{n+1} such that $\|E(h_{n+1})\| < 1$

- Relative tolerance ($rtol$) controls local error relative to the size of the solution
  - $rtol = 10^{-4}$ means that errors are controlled to 0.01%

- Absolute tolerances ($atol$) control error when a solution component may be small
  - e.g., $atol_i$ should be the noise level for a solution component

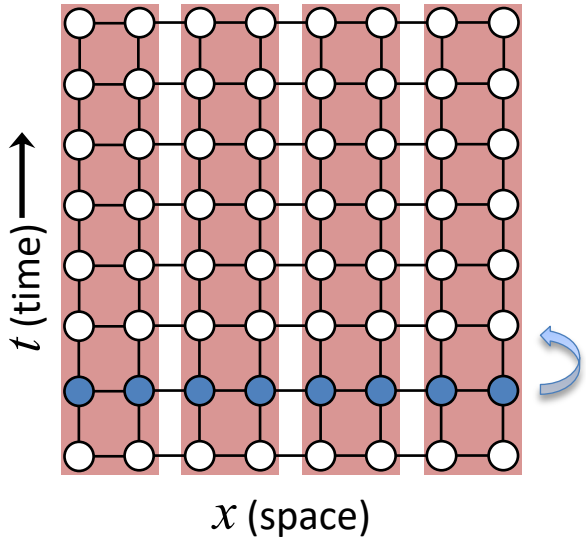
ARKode Flexibility Features

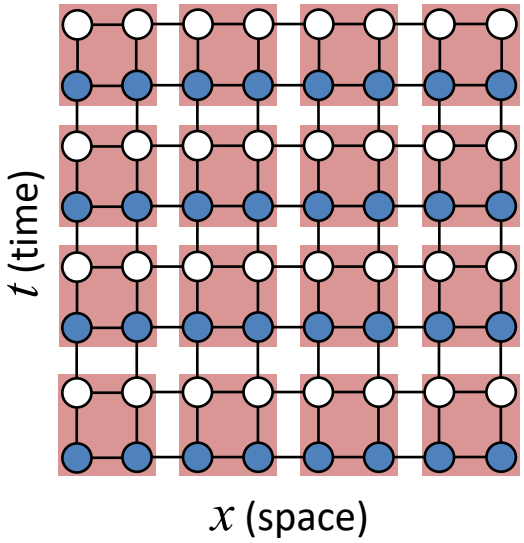
- Built on SUNDIALS vector, matrix, and solver APIs:
  - Several native modules including wrappers to other math libraries
  - Supports user defined modules

- ARKode includes several additional enhancements including:
  - Variety of built-in Butcher tables (ERK, DIRK, & IMEX) or user-supplied
  - Variety of built-in step size adaptivity controllers, or user-supplied
  - Variety of built-in implicit predictor algorithms, or user-supplied
  - Ability to specify that problem is linearly implicit
  - Ability to resize data structures based on changing IVP size
  - All internal solver parameters are user-modifiable

Significantly more parallel resources can be exploited with multigrid in time

- On massively parallel systems serial time integration becomes a bottleneck limiting the efficiency of simulations
- Parallel-in-time methods introduce an additional dimension of parallelism by distributing the workload in time across multiple processors

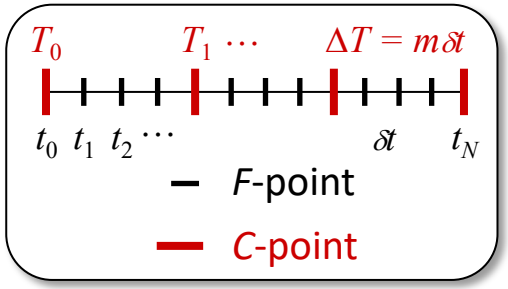


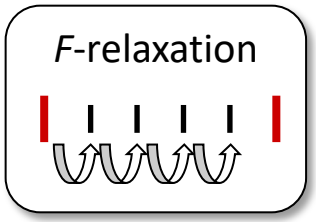


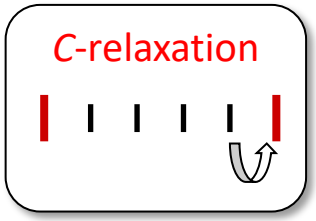
- Parallelize in space only
- Store only one time step
- Parallelize in space and time
- Store several time steps

Open source, flexible, and non-intrusive MGRIT implementation

- Create concurrency in the time dimension
- Non-intrusive with unchanged time discretization:
  - Converges to same solution as sequential stepping
  - Only stores *C*-points to minimize storage
  - Overlaps communication and computation
  - Extends to nonlinear problems with FAS formulation
- Speedups can be significant:
  - Useful only beyond some scale, there is a crossover point
  - Sometimes need significantly more parallelism just to break even
  - The more time steps, the more speedup potential
- **Allows for reuse of existing software**







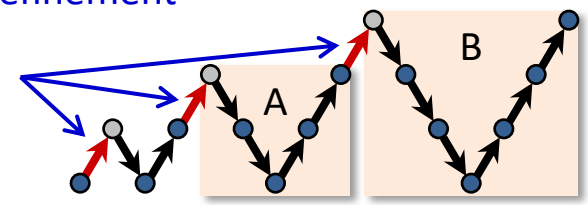
Relaxation alternates between *F* / *C*-points

- F-relax= integration over C intervals
- C-relax = one integration step

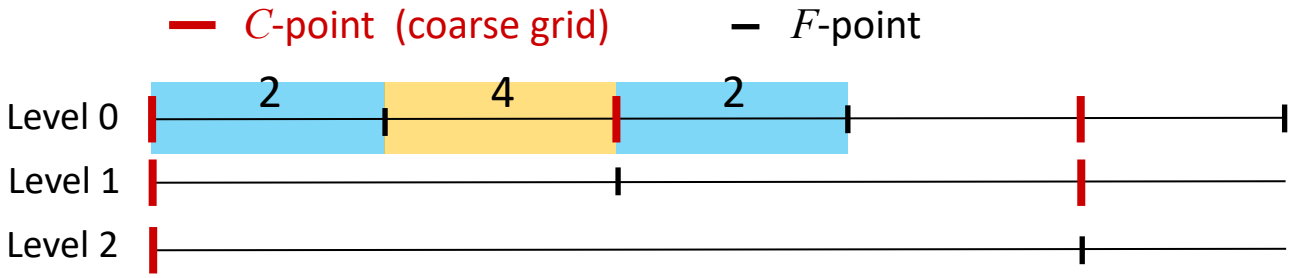
XBraid supports temporal adaptivity via an FMG-cycle

- User provides a refinement factor on the fine grid

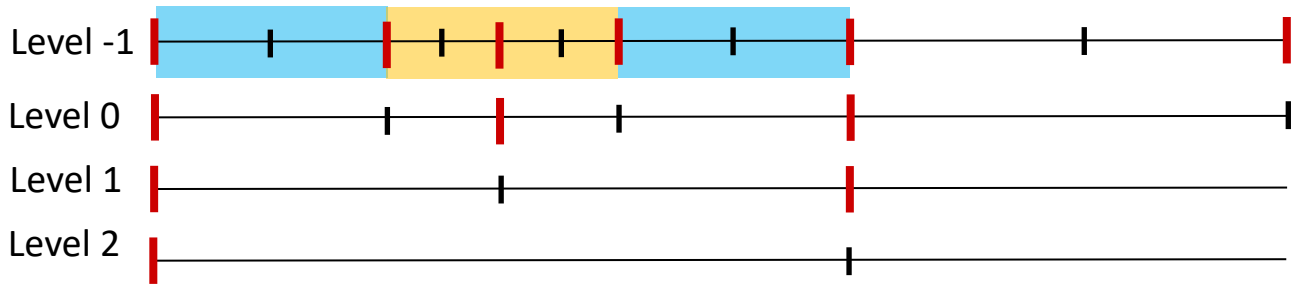
Temporal refinement done here



- Example time grid hierarchy (A)



- User requests refinement factors on the finest grid which generates a new grid and hierarchy (B)



Notice new C -pts

For more see Falgout et al. 2019

SUNDIALS + XBraid, what do we need?

- Two data structures:
  - An *App* structure holding any data needed by the interface
  - A *Vector* structure wrapping a SUNDIALS NVector
- Several wrapper functions:
  - Vector: *Clone, Free, Sum, SpatialNorm, and Buffer Size, Pack, and Unpack*
  - Other: *Init, Access, and Step*
- How much of the interfacing should the user supply and how much should be done by SUNDIALS? Goals:
  - Make things as simple as possible for a user that wants to try MGRIT
  - Provide as much flexibility as possible for the advanced user

SUNBraid: Vector Wrapper

```c
struct _braid_Vector_struct
{
  N_Vector y;
};

/* Poiner to vector wrapper (same as braid_Vector) */
typedef struct _braid_Vector_struct *SUNBraid_Vector;
```

```c
/* Compute alpha x + beta y -> y */
int SUNBraid_Vector_Sum(braid_App app,
                        braid_Real alpha, braid_Vector x,
                        braid_Real beta, braid_Vector y)
{
  N_VLinearSum(alpha, x->y, beta, y->y, y->y);
  return(0);
}
```

```c
/* Pack message buffer */
int SUNBraid_Vector_BufPack(braid_App app,
                            braid_Vector u, void *buffer,
                            braid_BufferStatus bstatus)
{
  int flag; /* return flag */

  /* Fill buffer */
  flag = N_VBufPack(u->y, buffer);
  if (flag != 0) return(1);

  return(0);
}
```

- Only a few small wrapper scripts are needed to use an NVector with XBraid
- Added new NVector buffer size, pack, and unpack functions:
  - N_VBufSize(v, size)
  - N_VBufPack(v, buf)
  - N_VBufUnpack(v, buf)
- Custom NVector modules only need to add these new functions to utilize the interface wrappers

SUNBraid: App Structure

```
/* Pointer to operations structure */
typedef struct _SUNBraidOps *SUNBraidOps;

/* Structure containing function pointers to operations */
struct _SUNBraidOps
{
  int (*init)(braid_App app, void *sun_mem, MPI_Comm comm_w,
              MPI_Comm comm_t, realtype tf, sunindextype nt,
              braid_Core *core);
  int (*free)(braid_App *app);
  int (*getudata)(braid_App app, void **user_data);
  int (*getvec)(braid_App app, N_Vector *y);
};

/* Define XBraid App structure */
struct _braid_App_struct
{
  void        *content;
  SUNBraidOps ops;
};

/* Pointer to the interface object (same as braid_App) */
typedef struct _braid_App_struct *SUNBraid_App;
```

- Generic SUNBraid_App structure follows the same object oriented design used for other SUNDIALS modules
- Individuals integrators or the user define the app content and implement the object operations
- This flexibly in defining the content makes it is easy to support simple uses case while accommodating advanced users

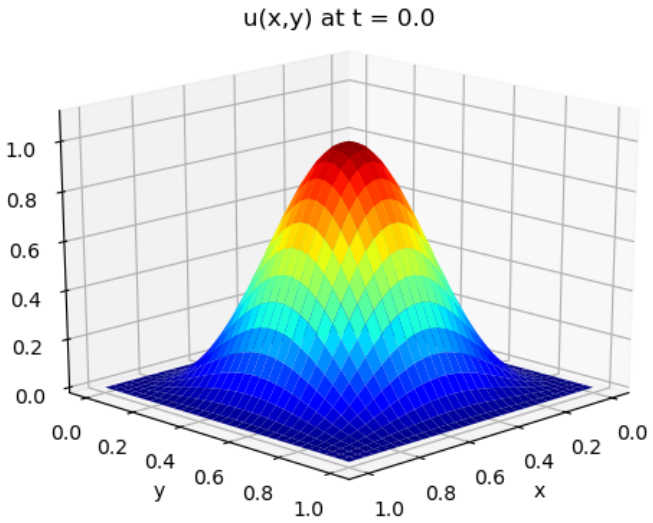
ARKBraid: App Structure

```
/* Define ARKode app content */
struct _ARKBraidAppContent
{
  /* ARKode memory structure */
  ARKodeMem ark_mem;

  /* Refinement options */
  booleantype refine;
  int max_refine;
  int rfactor_limit;

  /* Interface functions that a user may override */
  braid_PtFcnSpatialNorm spatialnorm;
  braid_PtFcnInit        init;
  braid_PtFcnAccess      access;
  braid_PtFcnStep        step;
};

typedef struct _ARKBraidAppContent *ARKBraidContent;
```

- The ARKode SUNBraid_App structure content holds
  - integrator memory,
  - optional parameters,
  - interface function pointers
- User setups the integrator as normal and attaches it with SUNBraidApp_Init(…)
- Set and Get functions allow users to modify or query most content

Example: 2D Heat Equation

- Problem setup:
  - Second order center differences in space
  - Second order SDIRK method in time
  - 128 x 128 mesh
  - MPI parallel in space
  - Inexact Newton with SUNDIALS PGC preconditioned using *hypre* PFMG
- Test Setups:
  - Fixed time stepping from [0, 1] with a step size of 1.0e-5
  - Adaptive time stepping from [0, 20] with various adaptivity controllers



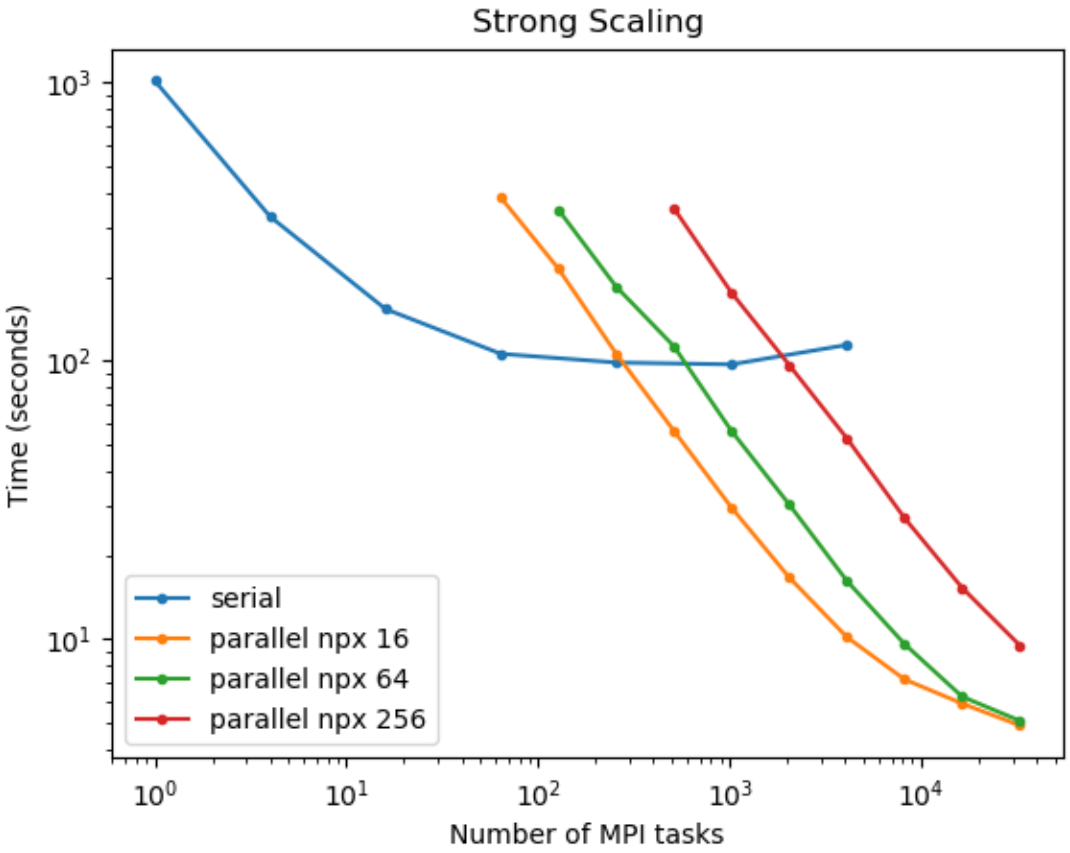
$$\frac{du}{dt} = \nabla^2 u + b(x, y, t)$$

The forcing term is chosen such that the analytic solution is

$$u(x, y, t) = \sin^2(\pi\, x) \sin^2(\pi\, y) \cos^2(\pi t)$$

Comparison of serial-in-time and parallel-in-time codes

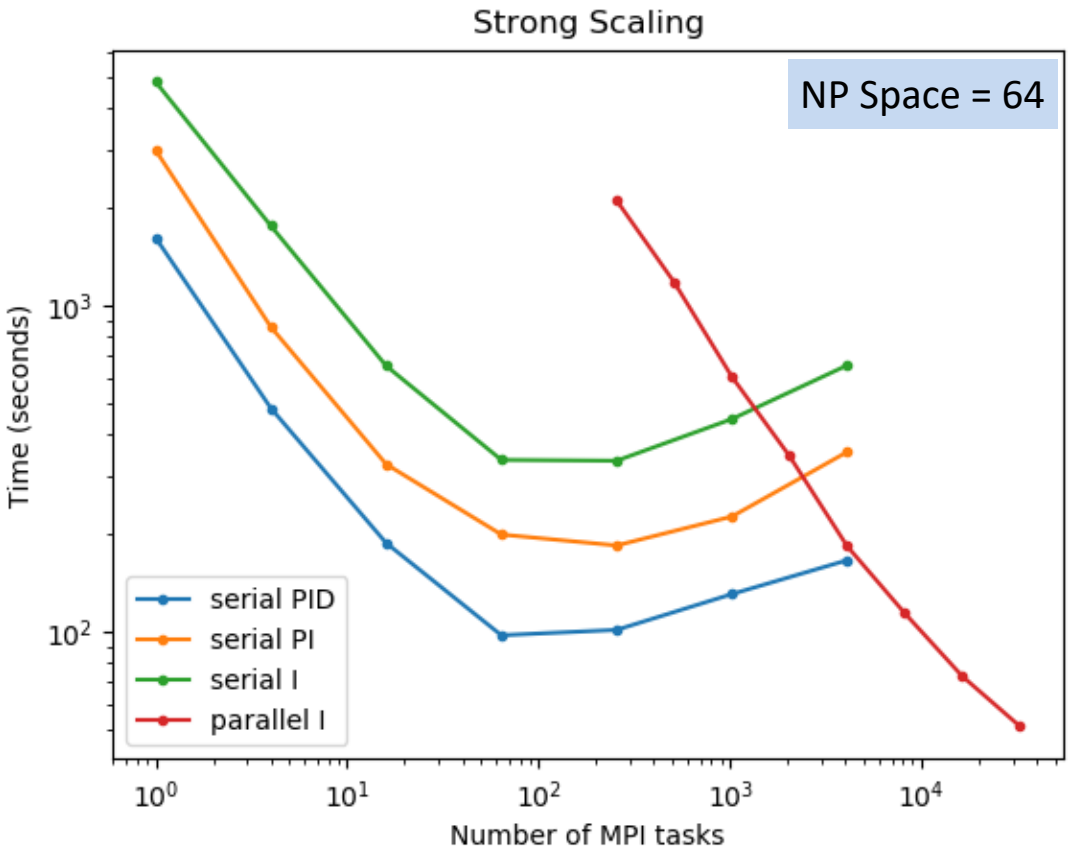
Example: 2D Heat Equation Fixed Step



NP Space	Speedup	Crossover
16	≈20x	≈256 (16)
64	≈20x	≈600 (10)
256	≈10x	≈1,024 (4)

- Setup:
  - Time interval [0, 1]
  - Step size 1.0e-5 s
  - Coarsening factors level 0 = 16, other levels = 2
  - ARKStep rtol = 10^{-4}, atol = 10^{-9}
  - Xbraid tol = $10^{-9}/\sqrt{dx * dy * dt}$
  - 2-norm in time and space

Example: 2D Heat Equation Adaptive Step



Controller	Speedup	Crossover
I	≈6.5x	≈256 (4)
PI	≈3.5x	≈4,096 (64)
PID	≈2x	≈10,000 (157)

Controller	Steps	Attempted
I	79,731	117,087
PI	51,285	62,109
PID	37,203	37,268
PinT (I)	63,263	--

- Setup: Time interval [0, 20], adaptive step sizes, coarsening factor 4 on all levels, ARKStep rtol = 10^{-4}, atol = 10^{-9}, Tol = $10^{-6}/\sqrt{dx * dy}$, max-norm in time, 2-norm in space

Takeaways and Current Work

- The non-intrusive design of the XBriad library makes it straight forward to interface with SUNDIALS' ARKode package
- MGRIT has shown potential for significant speedups by exposing parallelism in the time domain
- The SUNBraid interface requires minimal modification to existing code to get started and is flexible enough to accommodate advanced use cases

- Current work includes:
  - Exploring approaches for adapting solver tolerances and integrator parameters during the solve
  - Adding an option to supply a residual function to XBraid from SUNDIALS

- The XBraid interface will be release in SUNDIALS later this summer

Acknowledgements

This material is based upon work supported by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research.

This research was supported by the Exascale Computing Project (17-SC-20-SC), a collaborative effort of the U.S. Department of Energy Office of Science and the National Nuclear Security Administration.

computing.llnl.gov/sundials

computing.llnl.gov/xbraid

Thank You!

CASC

Center for Applied
Scientific Computing

Lawrence Livermore
National Laboratory