Introduction to the Capabilities and Use of the SUNDIALS Suite of Nonlinear and Differential/Algebraic Equation Solvers

ECP Annual Meeting, Houston, TX

ECP
EXASCALE COMPUTING PROJECT

CASC
Center for Applied Scientific Computing

SMU

Feb. 4, 2020

Carol S. Woodward, Daniel R. Reynolds, David J. Gardner, Cody J. Balos

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

Tutorial Outline

- **Overview of SUNDIALS (Carol Woodward)**
- How to use the time integrators (Daniel Reynolds)
- Which nonlinear and linear solvers are available and how to use them (David Gardner)
- Using SUNDIALS on (Pre) Exascale Machines (Cody Balos)
- Brief: How to download and install SUNDIALS (Cody Balos)

sundials SUite of Nonlinear and DIfferential-ALgebraic Solvers

- SUNDIALS is a software library consisting of ODE and DAE integrators and nonlinear solvers
  - 6 packages: CVODE(S), IDA(S), ARKode, and KINSOL
- Written in C with interfaces to Fortran (77 and 2003)
- Designed to be incorporated into existing codes
- Nonlinear and linear solvers and all data use is fully encapsulated from the integrators and can be user-supplied
- All parallelism is encapsulated in vector & solver modules and user-supplied functions
- Through the ECP, developing a rich infrastructure of support on exascale systems and applications
- Freely available; released under the BSD 3-Clause license (>27,000 downloads in 2019)
- Active user community supported by sundials-users email list
- Detailed user manuals are included with each package

https://computing.llnl.gov/casc/sundials

CVODE(S) and IDA(S) employ variable order and step BDF methods for integration

- CVODE solves ODEs *($\dot{y} = f(t, y)$)*
- IDA solves DAEs $F(t, y, \dot{y}) = 0$
  - Targets: implicit ODEs, index-1 DAEs, and Hessenberg index-2 DAEs
  - Optional routine solves for consistent values of y_0 and $\dot{y}_0$ for some cases
- Variable order and variable step size Linear Multistep Methods

$$\sum_{j=0}^{K_1} \alpha_{n,j} y_{n-j} + \Delta t_n \sum_{j=0}^{K_2} \beta_{n,j} \dot{y}_{n-j} = 0$$

- Both packages include stiff BDF methods up to 5th order (K_1 = 1,…,5 and K_2 = 0)
- CVODE includes nonstiff Adams-Moulton methods up to 12th order (K_1 = 1, K_2 = 1,…,12)
- Both packages include rootfinding for detecting sign change in solution-dependent functions
- CVODES and IDAS include both forward and adjoint (user supplies the adjoint operator) sensitivity analysis

ARKode is the newest package in SUNDIALS

- ARKode solves ODEs $M\dot{y} = f_I(t,y) + f_E(t,y), \quad y(t_0) = y_0$
  - M may be the identity or any nonsingular mass matrix (e.g., FEM)
- Multistage embedded methods (as opposed to multistep):
  - High order without solution history (enables spatial adaptivity)
  - Sharp estimates of solution error even for stiff problems
  - Implicit and additive multistage methods require multiple implicit solves per step
- Supplied with three steppers now (but easy to add others)
  - ERKStep: explicit Runge-Kutta methods for $\dot{y} = f(t,y), \quad y(t_0) = y_0$
  - ARKStep: explicit, implicit, or IMEX methods for $M\dot{y} = f_I(t,y) + f_E(t,y), \quad y(t_0) = y_0$
    - Split system into stiff, f_I, and nonstiff, f_E, components
  - MRIStep: two-rate multirate methods for $\dot{y} = f_f(t,y) + f_s(t,y), \quad y(t_0) = y_0$
    - Split the system into fast and slow components
    - More methods to come very soon

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

- Time step selection
  - Based on the method, estimate the time step error
  - Accept step if $||E(\Delta t)||_{WRMS} < 1$; Reject it otherwise

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

  - Choose next step, $\Delta t'$, so that $||E(\Delta t')||_{WRMS} < 1$
- CVODE and IDA also adapt order
  - Choose next order resulting in largest step meeting error condition

- 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
  - Ex: solution starting at a nonzero value but decaying to noise level, ATOL should be set to noise level

KINSOL solves systems of nonlinear algebraic equations, F(u) = 0

- Newton solvers: update iterate via $u^{k+1} = u^k + s^k, k = 0, ..., 1$
  - Compute the update by solving: $u^{k+1} = u^k + s^k \quad J(u^k)s^k = -F(u^k) \quad J(u) = \dfrac{\partial F(u)}{\partial u}$
  - An inexact Newton method approximately solves this equation
- Dynamic linear tolerance selection for use with iterative linear solvers

$$\|F(u^k) + J(u^k)s^k\| \leq \eta^k \|F(u^k)\|$$

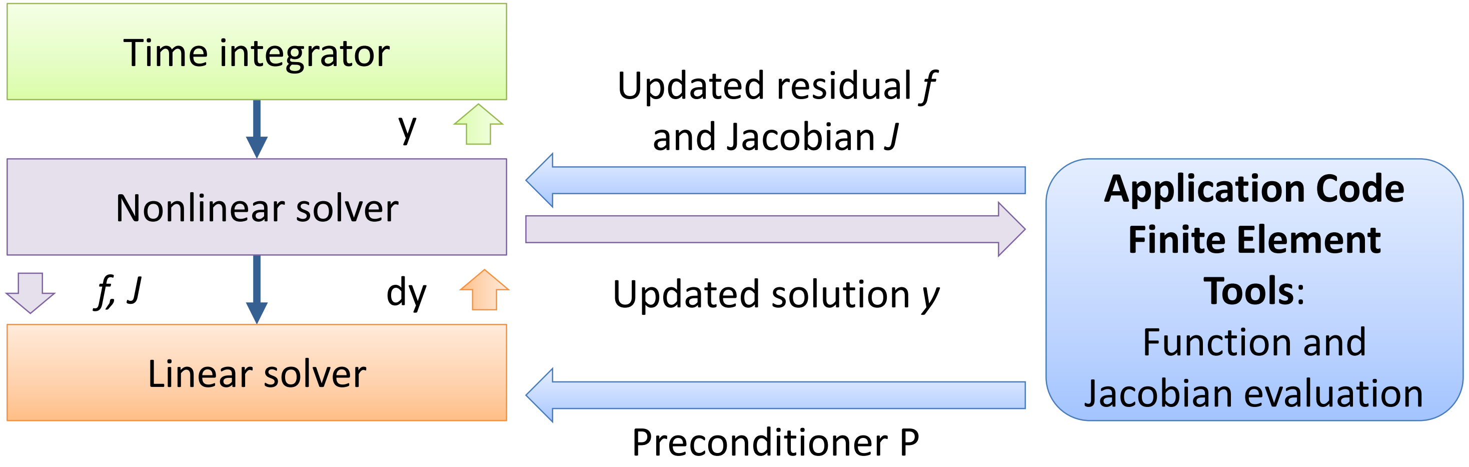
- Can separately scale equations and unknowns
- Backtracking and line search options for robustness
- Fixed point and Picard iterations with optional Anderson acceleration are also available

$$u^{k+1} = G(u^k), k = 0, 1, ...$$

$$u^{k+1} = u^k - L^{-1}F(u^k)$$

SUNDIALS uses modular design and control inversion to interface with application codes, external solvers, and encapsulate parallelism

- SUNDIALS integrators are built on shared vector, matrix, and solver APIs
- These APIs encapsulate the solution data and parallelism
- Several optional vector, matrix, and solver modules implementing the APIs are provided with SUNDIALS e.g.,
  - MPI vectors and solvers
  - GPU vectors and solvers
- It is straightforward to implement a problem-specific module tailored to the application


Time integrator
y
Nonlinear solver
f, *J*
dy
Linear solver
Updated residual *f* and Jacobian *J*
Updated solution *y*
Preconditioner P
Application Code Finite Element Tools: Function and Jacobian evaluation

Control passes from the integrator to the solvers and application code as the integration progresses

Time integrator and nonlinear solver are agnostic of vector data layout and specific solvers used

What’s new in SUNDIALS?

- High-order multirate methods that can integrate different portions of the problem with different time steps - current release includes 2nd and 3rd order two-rate methods that allow for explicit for the slow and explicit, implicit, and IMEX for the fast integrator
- New vector modules: Many-vector capability and MPI+X vectors
- Interface to PETSc nonlinear solvers (SNES API)
- Interface to SuperLU_DIST sparse direct linear solver
- Fortran 2003 interfaces (modernized from original F77 interface)
- Greater support for use in CUDA environments
  - Enhancements to the CUDA vector
  - Interface to the NVIDIA CuSparse batched QR sparse linear solver

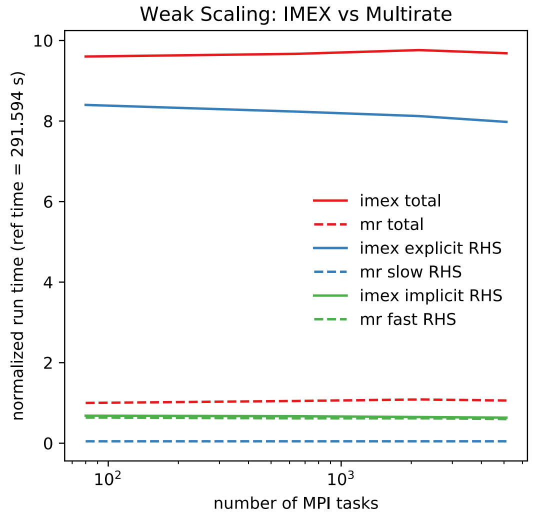
Results with SUNDIALS' multirate integrators are very encouraging

- Demonstrated scalability on primordial gas chemically reacting flow test with compressible hydrodynamics and stiff chemical reactions (10 chem. species):

$$w_t = -\nabla \cdot F(w) + R(w) + G(w,t)$$

  - w: density, momenta in each direction, total energy, and chemical densities (10)
  - F: advective fluxes; R reaction terms; and G: external forces

- Tested with:
  - 3rd order 2-rate method with slow explicit advection and fast implicit rxns (Factor ~1000 in step size)
  - 3rd order IMEX method using fast step



Normalized run times show ~10x speed ups for multirate methods using third order implicit for fast chemical reactions and third order explicit for slow fluid flow over similar third order IMEX method running at the single rate of the fast time scale.

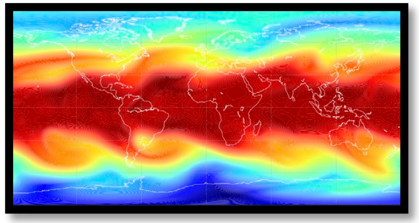
Multirate allows the explicit RHS evaluation (which requires MPI exchanges) to run at a far reduced time step than what is required for the single rate IMEX method to maintain stability.

What are we working on now?

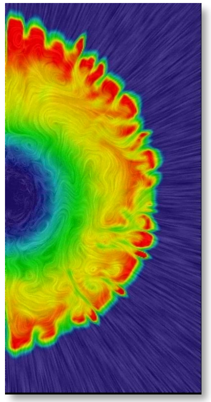
- Enhanced support for solving several ODE systems simultaneously on GPUs (useful for combustion applications)
  - New Get routines to allow for better diagnosis of “hard” systems for load balancing
  - Fusing more vector kernels for efficiency
  - Support for loading sparse matrices from a GPU
  - HIP vector and solvers as they become available
- Addition of capability to integrate a system with CVODE while projecting the solution onto a constraint manifold
- Capability to integrate with time-dependent mass matrices in ARKode
- Increased interoperability with other solver libraries SuperLU_DIST TRILINOS PETSc MAGMA
- More multirate methods, including implicit / explicit schemes

SUNDIALS: Used Worldwide in Applications from Research & Industry

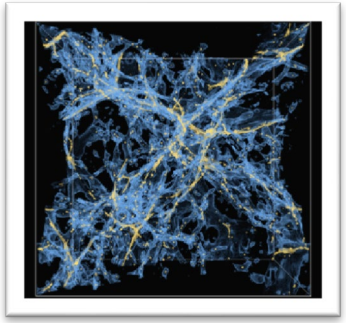
- Computational Cosmology (Nyx)
- Combustion (PELE)
- Atmospheric dynamics (DOE E3SM)
- Fluid Dynamics (NEK5000) (ANL)
- Dislocation dynamics (LLNL)
- 3D parallel fusion (SMU, U. York, LLNL)
- Power grid modeling (RTE France, ISU, LLNL)
- Sensitivity analysis of chemically reacting flows (Sandia)
- Large-scale subsurface flows (CO Mines, LLNL)
- Micromagnetic simulations (U. Southampton)
- Chemical kinetics (Cantera)
- Released in third party packages:
  - Red Hat Extra Packages for Enterprise Linux (EPEL)
  - SciPy – python wrap of SUNDIALS
  - Cray Third Party Software Library (TPSL)



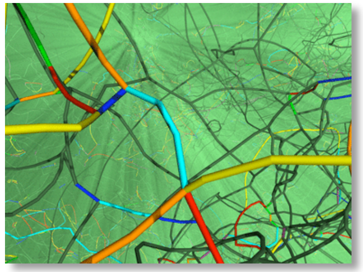
Atmospheric Dynamics



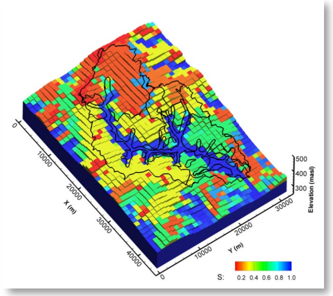
Core collapse supernova



Cosmology



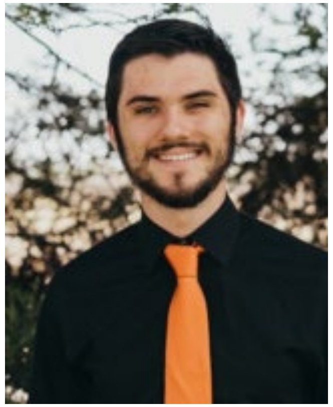
Dislocation dynamics



Subsurface flow

SUNDIALS Team

Current Team:



Cody Balos

David Gardner

Alan Hindmarsh

Dan Reynolds

Carol Woodward

Alumni:

Radu Serban

Scott D. Cohen, Peter N. Brown, George Byrne, Allan G. Taylor, Steven L. Lee, Keith E. Grant, Aaron Collier, Lawrence E. Banks, Steve G. Smith, Cosmin Petra, Slaven Peles, John Loffeld, Dan Shumaker, Ulrike M. Yang, James Almgren-Bell, Shelby L. Lockhart, Hilari C. Tiedeman, Ting Yan, Jean M. Sexton, and Chris White

Tutorial Outline

- Overview of SUNDIALS (Carol Woodward)
- **How to use the time integrators (Daniel Reynolds)**
- Which nonlinear and linear solvers are available and how to use them (David Gardner)
- Using SUNDIALS on (Pre) Exascale Machines (Cody Balos)
- Brief: How to download and install SUNDIALS (Cody Balos)

Time Integrators

This tutorial section covers basic usage of the SUNDIALS time integration packages (CVODE, IDA, ARKODE):

- Problem specification
- Integrator creation/initialization
- Advancing the solutions
- Retrieving optional outputs
- Advanced features

“Solving” Initial-Value Problems with SUNDIALS

- SUNDIALS’ integrators consider initial-value problems of a variety of types:
  - Standard IVP [CVODE]: $\dot{y}(t) = f(t, y(t)), \quad y(t_0) = y_0$
  - Linearly-implicit, split [ARKode]: $M\,\dot{y}(t) = f_1(t, y(t)) + f_2(t, y(t)), \quad y(t_0) = y_0$
  - Multirate [ARKode/MRIStep]: $\dot{y} = f^F(t, y) + f^S(t, y), \quad y(t_0) = y_0$
  - Differential-algebraic form [IDA]: $F(t, y(t), \dot{y}(t)) = 0, \quad y(t_0) = y_0, \quad \dot{y}(t_0) = \dot{y}_0$
- By “solve” we adapt time steps to meet user-specified tolerances:

$$\left[\frac{1}{N}\sum_{k=1}^{N}\left(\frac{\mathrm{error}_k}{\mathrm{rtol}\,|y_k| + \mathrm{atol}_k}\right)^2\right]^{1/2} < 1$$

  - $\mathrm{error} \in \mathbb{R}^N$ is the estimated temporal error in the time step
  - $y \in \mathbb{R}^N$ is the previous time-step solution
  - $\mathrm{rtol} \in \mathbb{R}$ encodes the desired relative solution accuracy (number of significant digits)
  - $\mathrm{atol} \in \mathbb{R}^N$ is the ‘noise’ level for any solution component (protects against $y_k = 0$)

The “Skeleton” for Using SUNDIALS Integrators

1. Initialize parallel or multi-threaded environment
2. Create vector of initial values, $y_0 \in \mathbb{R}^N$; if using IDA, also create $\dot{y}_0 \in \mathbb{R}^N$
3. Create and initialize integrator object (attaches $t_0, y_0, (\dot{y}_0)$, RHS/residual function(s))
4. Create matrix, linear solver, nonlinear solver objects (if applicable); attach to integrator
   - Defaults exist for some of these, but may be replaced with problem-specific versions
   - Parallel scalability hinges on appropriate choices (discussed in next section of tutorial)
5. Specify optional inputs to integrator and solver objects (tolerances, etc.)
6. Advance solution in time, either over specified time intervals $[a, b]$, or for single timesteps
7. Retrieve optional outputs
8. Free solution/solver memory; finalize MPI (if applicable)

C/C++ vs Fortran

SUNDIALS still supports legacy Fortran77 interfaces for most packages, but in the last year we have released fully-featured Fortran 2003 interfaces for nearly the entire suite:

- Leverage the `iso_c_binding` module and the `bind(C)` attribute from the F2003 standard.
- SUNDIALS' F2003 interfaces closely follow the C/C++ API
- Generic SUNDIALS structures, e.g. , `N_Vector`, are interfaced as Fortran derived types, and function signatures are matched but with an `F` prepending the name, e.g. `FN_VConst` instead of `N_VConst`.
- Constants are named exactly as they are in the C/C++ API.
- Accordingly, using SUNDIALS via the Fortran 2003 interfaces looks just like using it in C/C++.

The remainder of this tutorial will therefore focus on C/C++; please reserve questions regarding the F77 or F2003 interfaces for one-on-one discussions.

Supplying the Initial Condition Vector(s)

- As discussed earlier, all SUNDIALS integrators operate on data through the NVector API.
- Each provided vector module has a unique set of “constructors”, e.g.

```
N_Vector N_VNew_Serial(sunindextype length);

N_Vector N_VNew_Parallel(MPI_Comm comm, sunindextype loc_len, sunindextype glob_len);

N_Vector N_VMake_Cuda(sunindextype length, realtype *h_vdata, realtype *d_vdata);

N_Vector N_VMake_MPIManyVector(MPI_Comm comm, sunindextype n_subvec, N_Vector *varr);

N_Vector N_VMake_MPIPlusX(MPI_Comm comm, N_Vector x);
```

- Once an application creates a vector for their data, they fill it with the initial conditions for the problem and supply it to the integrator, who “clones” it to create its workspace.
- For PETSc, *hypre,* and Trilinos, the corresponding SUNDIALS NVector wrapper constructors take the native vector structure as their only input.

Supplying the IVP to the Integrator – RHS/Residual Functions

Once the problem data is encapsulated in a vector, all that remains for basic SUNDIALS usage is specification of the IVP itself:

- CVODE and ARKODE specify the IVP through right-hand side function(s):

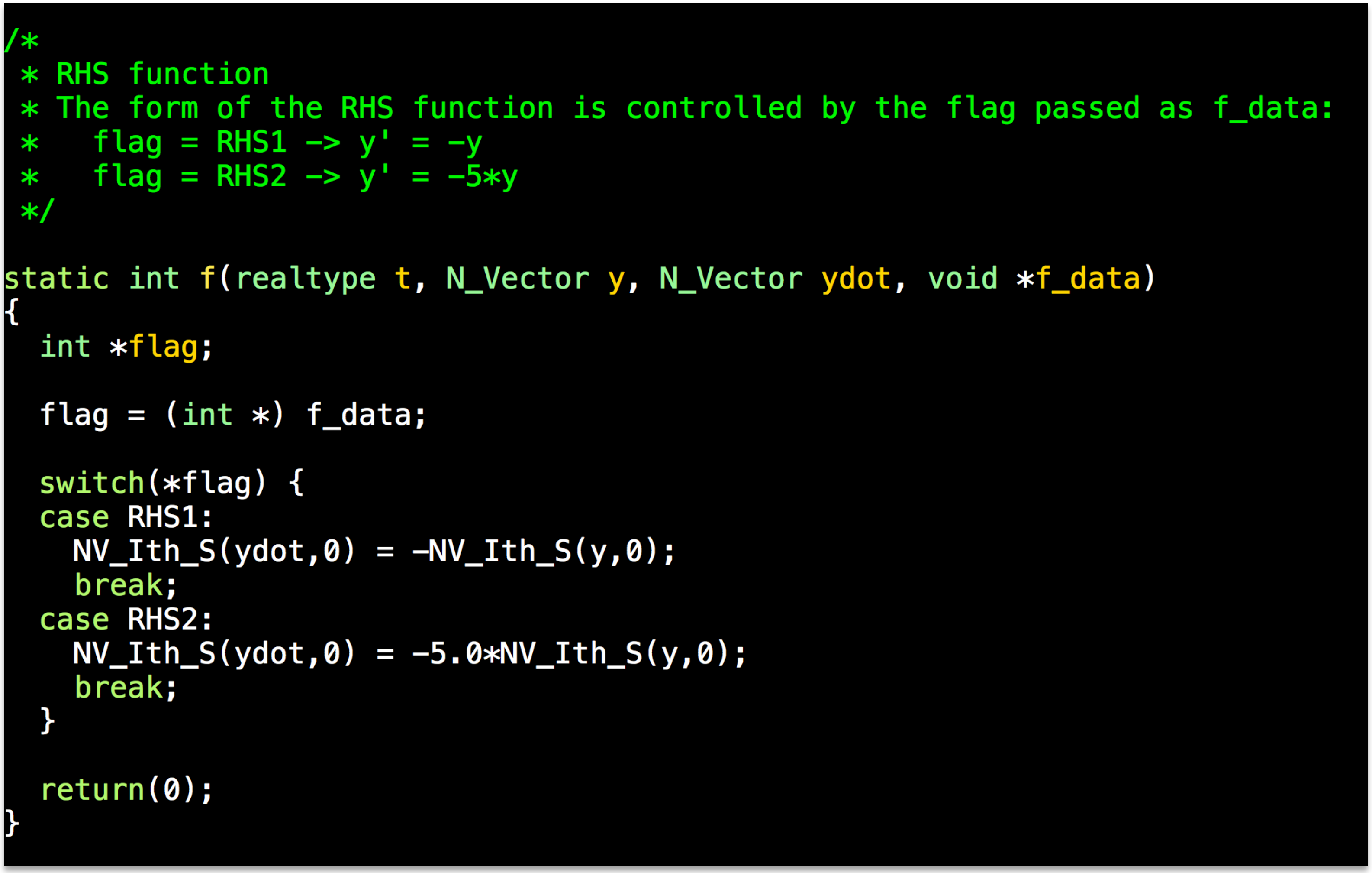
```
int (*RhsFn)(realtype t, N_Vector y, N_Vector ydot, void *user_data)
```

- IDA specifies the IVP through a residual function:

```
int (*ResFn)(realtype t, N_Vector y, N_Vector ydot, N_Vector r,
  void *user_data)
```

- The *user_data pointer enables problem-specific data to be passed through the SUNDIALS integrator and back to the RHS/residual routine (i.e., no global memory).

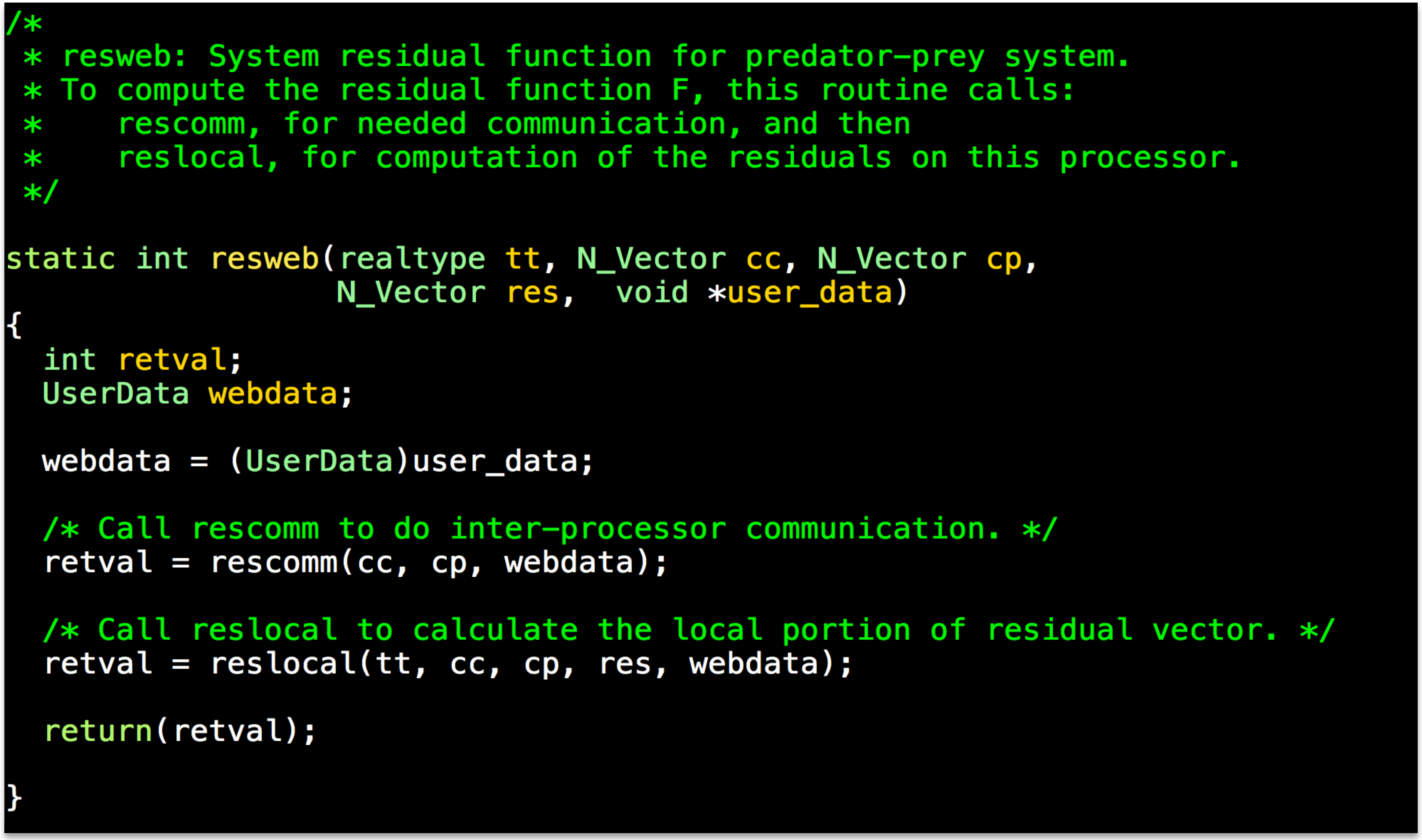
CVODE/ARKODE RHS Functions



```
/*
 * RHS function
 * The form of the RHS function is controlled by the flag passed as f_data:
 *   flag = RHS1 -> y' = -y
 *   flag = RHS2 -> y' = -5*y
 */

static int f(realtype t, N_Vector y, N_Vector ydot, void *f_data)
{
  int *flag;

  flag = (int *) f_data;

  switch(*flag) {
  case RHS1:
    NV_Ith_S(ydot,0) = -NV_Ith_S(y,0);
    break;
  case RHS2:
    NV_Ith_S(ydot,0) = -5.0*NV_Ith_S(y,0);
    break;
  }

  return(0);
}
```

Example:
cvDisc_dns.c

IDA Residual Function



```
/*
 * resweb: System residual function for predator-prey system.
 * To compute the residual function F, this routine calls:
 *    rescomm, for needed communication, and then
 *    reslocal, for computation of the residuals on this processor.
 */

static int resweb(realtype tt, N_Vector cc, N_Vector cp,
                  N_Vector res,  void *user_data)
{
  int retval;
  UserData webdata;

  webdata = (UserData)user_data;

  /* Call rescomm to do inter-processor communication. */
  retval = rescomm(cc, cp, webdata);

  /* Call reslocal to calculate the local portion of residual vector. */
  retval = reslocal(tt, cc, cp, res, webdata);

  return(retval);

}
```

Example:
idaFoodWeb_kry_p.c

Supplying the IVP to ARKODE – Mass Matrix Functions

When solving an IVP with non-identity mass matrix, users must supply either a routine to construct a mass matrix $M \in \mathbb{R}^{N\times N}$:

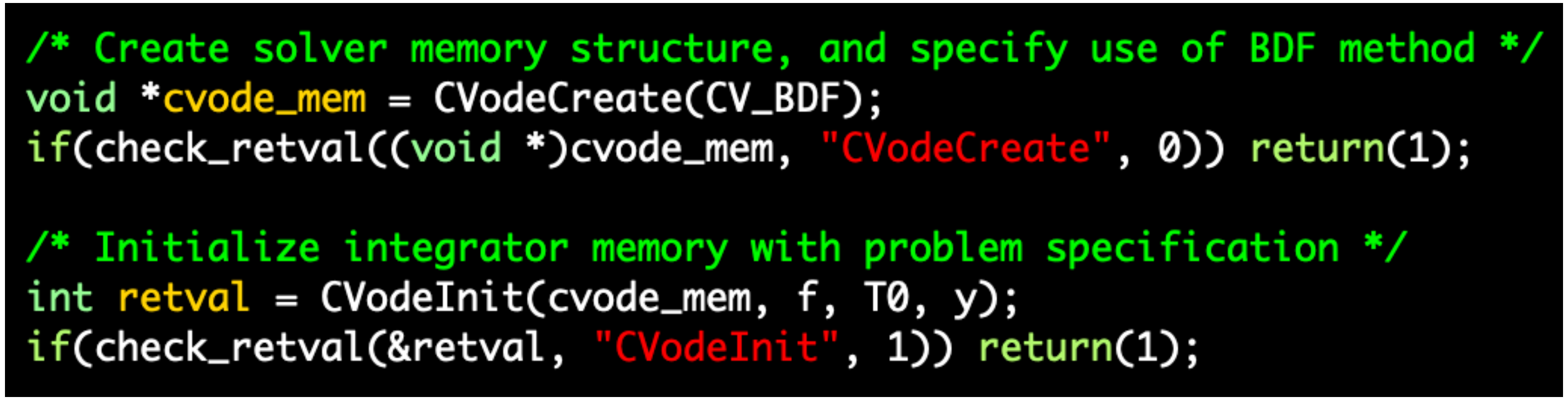
```
int (*ARKLsMassFn)(realtype t, SUNMatrix M, void *user_data,
            N_Vector tmp1, N_Vector tmp2, N_Vector tmp3);
```

or to perform the mass-matrix-vector product, $Mv : \mathbb{R}^N \rightarrow \mathbb{R}^N$:

```
int (*ARKLsMassTimesSetupFn)(realtype t, void *mtimes_data);

int (*ARKLsMassTimesVecFn)(N_Vector v, N_Vector Mv, realtype t,
                 void *mtimes_data);
```

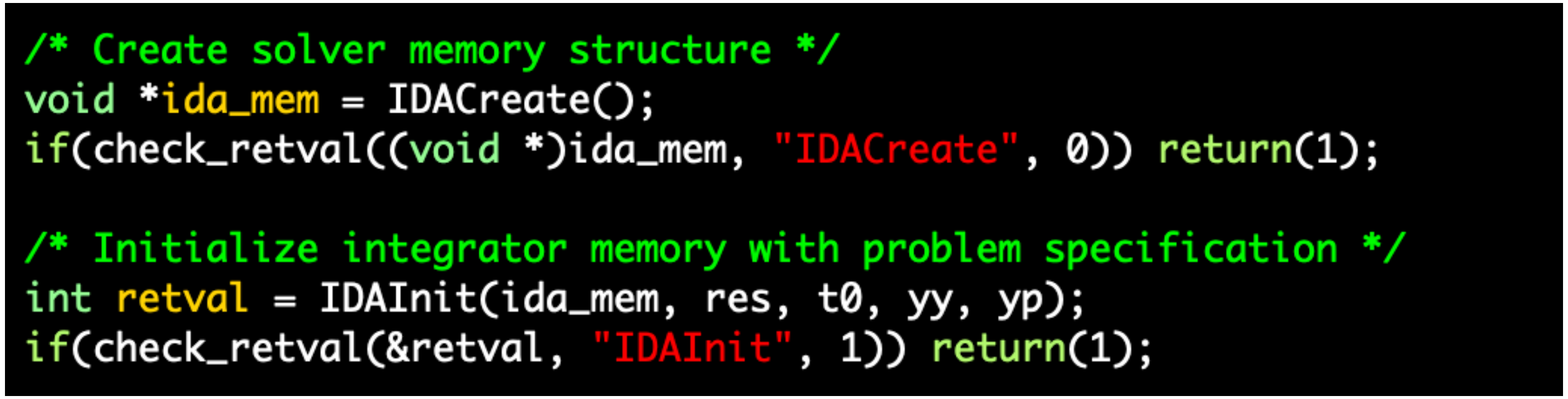
Initializing the Integrators – CVODE and IDA

The IVP inputs are supplied when constructing the integrator.



```
/* Create solver memory structure, and specify use of BDF method */
void *cvode_mem = CVodeCreate(CV_BDF);
if(check_retval((void *)cvode_mem, "CVodeCreate", 0)) return(1);

/* Initialize integrator memory with problem specification */
int retval = CVodeInit(cvode_mem, f, T0, y);
if(check_retval(&retval, "CVodeInit", 1)) return(1);
```

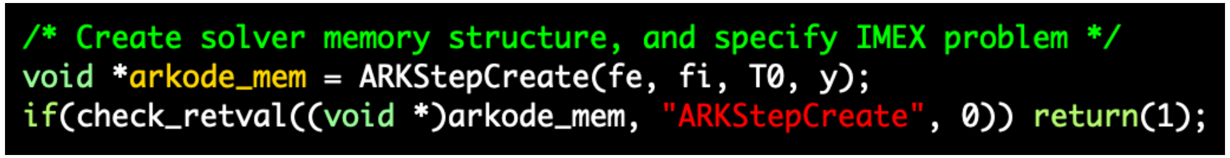
CVODE



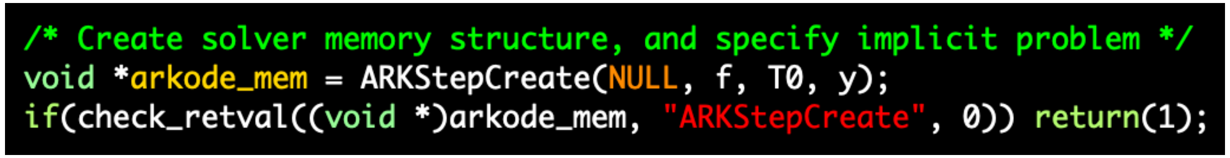
```
/* Create solver memory structure */
void *ida_mem = IDACreate();
if(check_retval((void *)ida_mem, "IDACreate", 0)) return(1);

/* Initialize integrator memory with problem specification */
int retval = IDAInit(ida_mem, res, t0, yy, yp);
if(check_retval(&retval, "IDAInit", 1)) return(1);
```

IDA

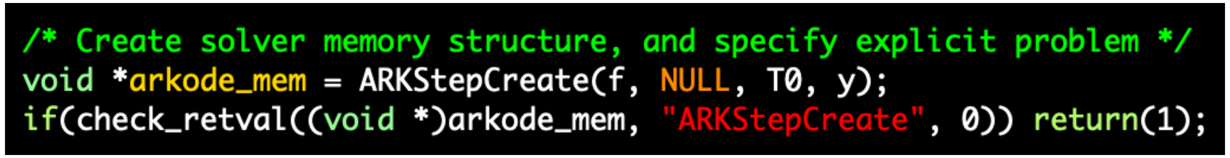
Initializing the Integrators – ARKODE



```
/* Create solver memory structure, and specify IMEX problem */
void *arkode_mem = ARKStepCreate(fe, fi, T0, y);
if(check_retval((void *)arkode_mem, "ARKStepCreate", 0)) return(1);
```

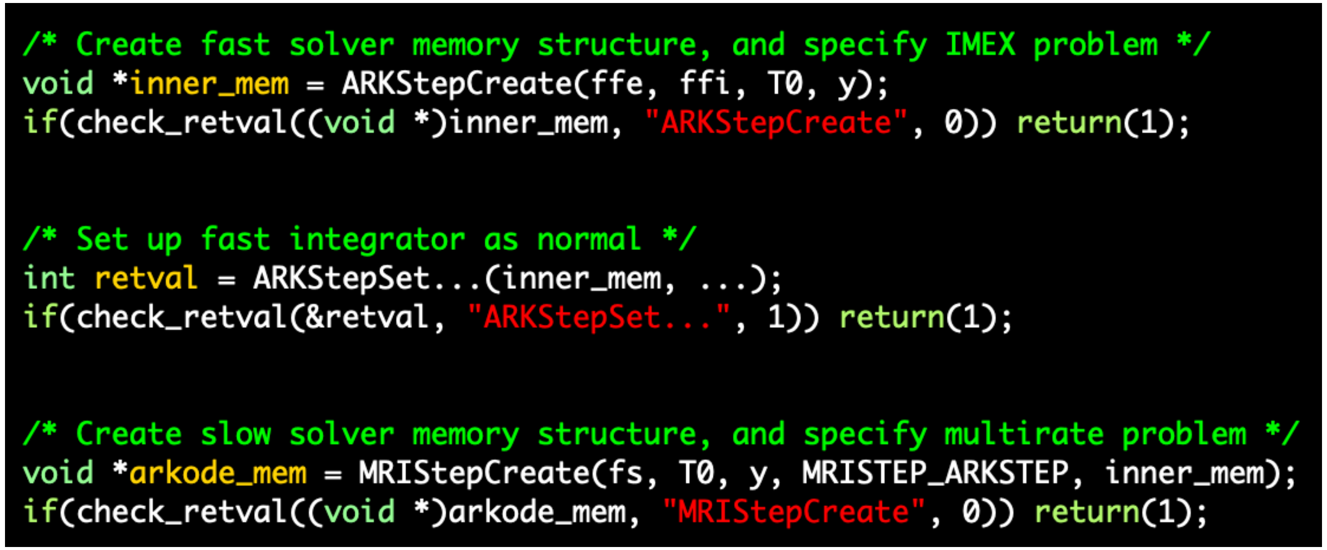


```
/* Create solver memory structure, and specify implicit problem */
void *arkode_mem = ARKStepCreate(NULL, f, T0, y);
if(check_retval((void *)arkode_mem, "ARKStepCreate", 0)) return(1);
```



```
/* Create solver memory structure, and specify explicit problem */
void *arkode_mem = ARKStepCreate(f, NULL, T0, y);
if(check_retval((void *)arkode_mem, "ARKStepCreate", 0)) return(1);
```

IMEX (top), implicit (middle), explicit (bottom)



```
/* Create fast solver memory structure, and specify IMEX problem */
void *inner_mem = ARKStepCreate(ffe, ffi, T0, y);
if(check_retval((void *)inner_mem, "ARKStepCreate", 0)) return(1);

/* Set up fast integrator as normal */
int retval = ARKStepSet...(inner_mem, ...);
if(check_retval(&retval, "ARKStepSet...", 1)) return(1);

/* Create slow solver memory structure, and specify multirate problem */
void *arkode_mem = MRIStepCreate(fs, T0, y, MRISTEP_ARKSTEP, inner_mem);
if(check_retval((void *)arkode_mem, "MRIStepCreate", 0)) return(1);
```

Multirate with IMEX at fast time scale

Optional Inputs (all Integrators)

A variety of optional inputs enable enhanced control over the integration process. Here we discuss the most often-utilized options (see documentation for the full set).

- Tolerance specification – rtol with scalar or vector-valued atol, or user-specified routine to compute the error weight vector

$$w_k \approx \frac{1}{\text{rtol}\,|y_k| + \text{atol}_k} > 0, \quad k = 1, \ldots, N$$

- SetNonlinearSolver, SetLinearSolver – attaches desired nonlinear solver, linear solver and (optionally) matrix modules to the integrator.
- SetUserData – specifies the (void *user_data) pointer that is supplied to user routines.
- SetMaxNumSteps, SetMaxStep, SetMinStep, SetInitStep – provides guidance to time step adaptivity algorithms.
- SetStopTime – specifies the value of t_{stop} to use when advancing solution (this is retained until this stop time is reached or modified through a subsequent call).

Package-Specific Options (CVODE and IDA)

- SetMaxOrd – specifies the maximum order of accuracy for the method (the order is adapted internally, along with the step size).

- CalcIC (IDA-specific) – in certain cases will help find a consistent .$\dot{y}_0$

  – A variety of additional routines may be used for additional control over this algorithm.

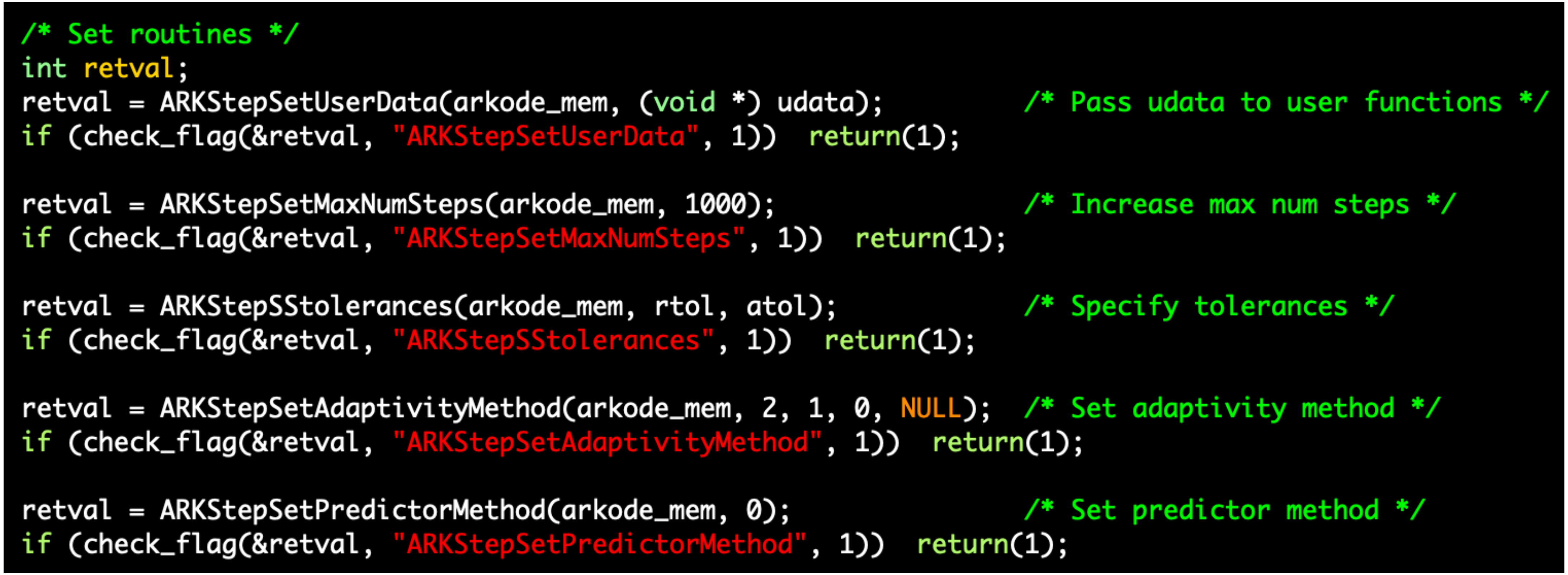
- SetId (IDA-specific) – specifies which variables are differential vs algebraic (useful when calling CalcIC above).

Package-Specific Options (ARKODE)

- SetFixedStep – disables time step adaptivity (and temporal error estimation/control).
- SetLinear – $f_1(t,y(t))$ depends *linearly* on y (disables nonlinear iteration).
- SetOrder – specifies the order of accuracy for the method.
- SetTables – allows user-specified ERK, DIRK or ARK Butcher tables.
- SetAdaptivityFn – allows user-provided routine for time step selection.
- MRIStep allows fast and slow time scales to be controlled independently, e.g., both using fixed step sizes, fast using temporal adaptivity, …

Supplying Options to the Integrators

After constructing the integrator, additional options may be supplied through various “Set” routines (example from ark_heat1D_adapt.c):



```
/* Set routines */
int retval;
retval = ARKStepSetUserData(arkode_mem, (void *) udata);         /* Pass udata to user functions */
if (check_flag(&retval, "ARKStepSetUserData", 1))  return(1);

retval = ARKStepSetMaxNumSteps(arkode_mem, 1000);                /* Increase max num steps */
if (check_flag(&retval, "ARKStepSetMaxNumSteps", 1))  return(1);

retval = ARKStepSStolerances(arkode_mem, rtol, atol);            /* Specify tolerances */
if (check_flag(&retval, "ARKStepSStolerances", 1))  return(1);

retval = ARKStepSetAdaptivityMethod(arkode_mem, 2, 1, 0, NULL);  /* Set adaptivity method */
if (check_flag(&retval, "ARKStepSetAdaptivityMethod", 1))  return(1);

retval = ARKStepSetPredictorMethod(arkode_mem, 0);               /* Set predictor method */
if (check_flag(&retval, "ARKStepSetPredictorMethod", 1))  return(1);
```

Supplying Custom Butcher tables to ARKODE

Users may construct custom Butcher tables and supply these to the integrator:

Constructor:

```
ARKodeButcherTable ARKodeButcherTable_Create(int s, int q, int p,
            realtype *c, realtype *A, realtype *b, realtype *b2);
```

Specification:

```
int ARKStepSetTables(void *arkode_mem, int q, int p,
                ARKodeButcherTable Bi,
                ARKodeButcherTable Be);
```

(either `Bi` or `Be` may be `NULL` to specify use of an ERK or DIRK method, respectively)

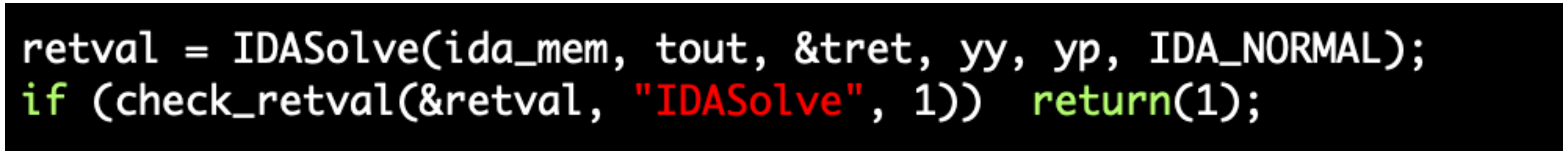
Usage Modes for SUNDIALS Integrators

While t_0 is supplied at initialization, the *direction* of integration is specified on the first call to advance the solution toward the output time t_{out}. This may occur in one of four “usage modes”:

- “Normal” – take internal steps until t_{out} is overtaken in the direction of integration, e.g. for forward integration $t_{n-1} < t_{\text{out}} \leq t_n$; the solution $y(t_{\text{out}})$ is then computed by interpolation.
- “One-step” – take a single internal step $y_{n-1} \to y_n$ and then return control back to the calling program. If this step will overtake t_{out} then $y(t_{\text{out}})$ is interpolated; otherwise y_n is returned.
- “Normal + TStop” – take internal steps until the next step will overtake t_{stop}; limit the next internal step so that $t_n = t_{\text{stop}}$. No interpolation is performed.
- “One-step + TStop” – performs a combination of both “One-step” and “TStop” modes above.

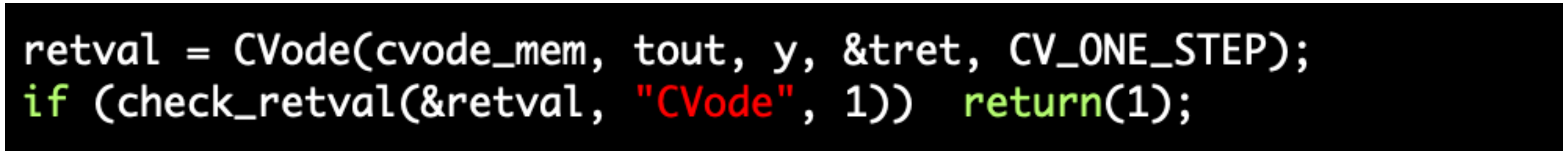
Advancing the Solution

Once all options have been set, the integrator is called to advance the solution toward t_{out}.



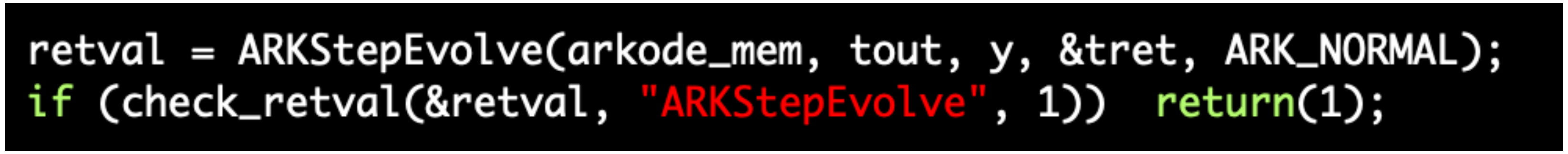
```
retval = IDASolve(ida_mem, tout, &tret, yy, yp, IDA_NORMAL);
if (check_retval(&retval, "IDASolve", 1))  return(1);
```

IDA

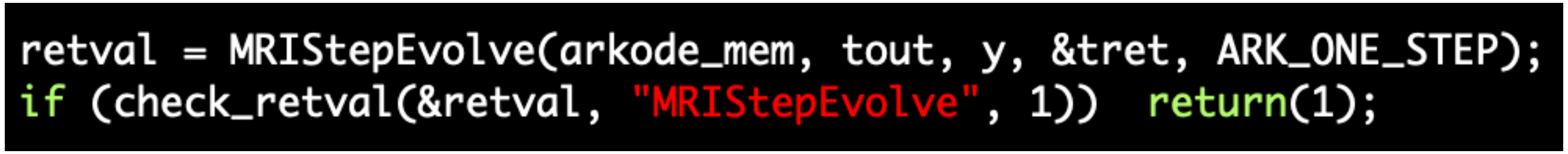


```
retval = CVode(cvode_mem, tout, y, &tret, CV_ONE_STEP);
if (check_retval(&retval, "CVode", 1))  return(1);
```

CVODE



```
retval = ARKStepEvolve(arkode_mem, tout, y, &tret, ARK_NORMAL);
if (check_retval(&retval, "ARKStepEvolve", 1))  return(1);
```



```
retval = MRIStepEvolve(arkode_mem, tout, y, &tret, ARK_ONE_STEP);
if (check_retval(&retval, "MRIStepEvolve", 1))  return(1);
```

ARKODE

Optional Outputs – General Time Integration

Either between calls to advance the solution, or at the end of a simulation, users may retrieve a variety of optional outputs from SUNDIALS integrators.

- GetDky (Dense solution output) – using the same infrastructure that performs interpolation in "normal" use mode, users may request values $\frac{d^k}{dt^k}y(t)$ for $t_{n-1} \le t \le t_n$, where $0 \le k \le k_{\max}$.
- Time integration statistics:
  - GetNumSteps – the total number of internal time steps since initialization
  - GetCurrentStep – the current internal time step size
  - GetCurrentTime – the current internal time (since this may have passed)t_{out}
  - GetCurrentOrder (IDA/CVODE) – the current method order of accuracy
  - GetEstLocalErrors – returns the current temporal error vector, $\mathrm{error} \in \mathbb{R}^N$

Optional Outputs – Algebraic Solver Statistics

- GetNumNonlinSolvIters – number of nonlinear solver iterations since initialization.
- GetNumNonlinSolvConvFails – number of nonlinear solver convergence failures.
- GetNumLinSolvSetups – number of calls to setup the linear solver or preconditioner.
- GetNumLinIters – number of linear solver iterations since initialization.
- GetNumLinConvFails – number of linear solver convergence failures.
- GetNumJacEvals, GetNumJtimesEvals, GetNumPrecEvals, GetNumPrecSolves – the number of calls to user-supplied Jacobian/preconditioner routines.

Optional Outputs – Miscellaneous

- GetErrWeights – returns the current error weight vector, $w \in \mathbb{R}^N$

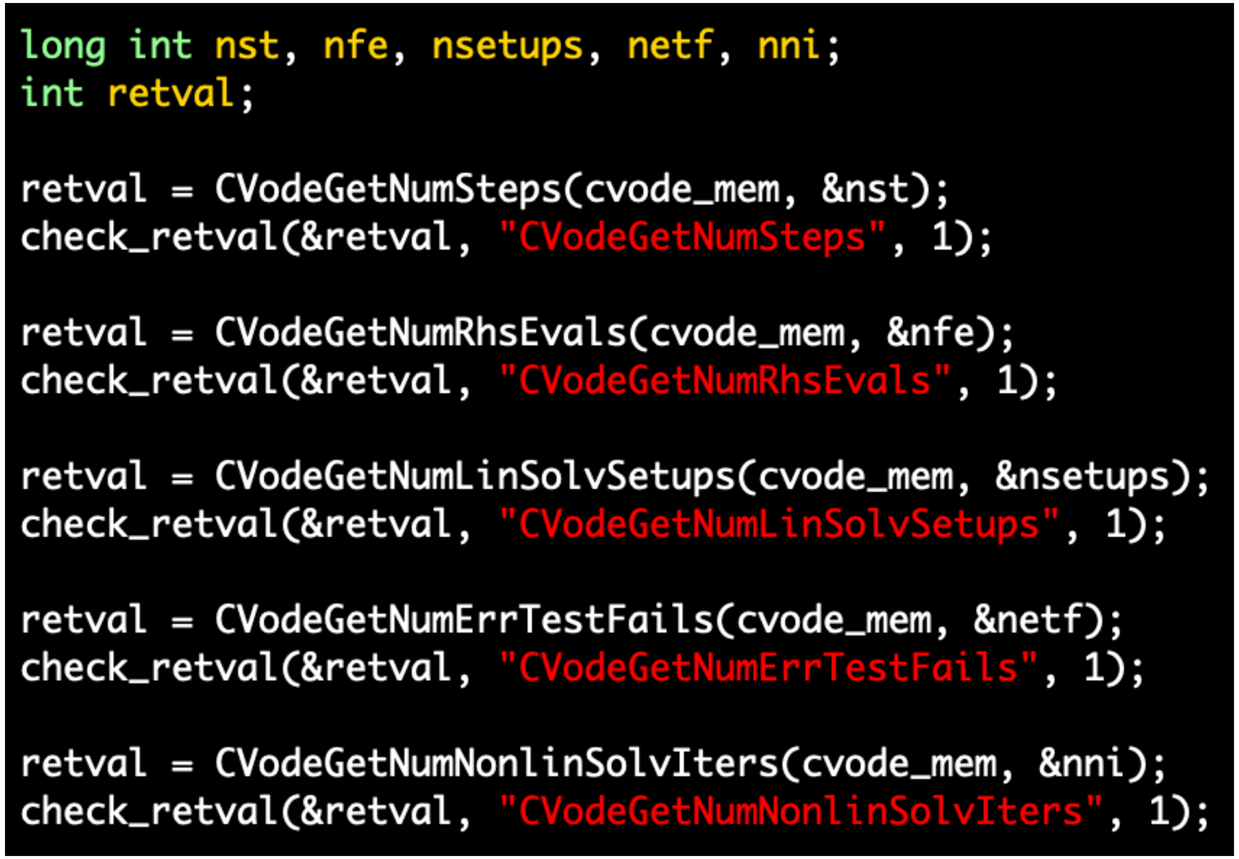
- GetWorkspace – returns the memory requirements for the integrator.

- GetLinWorkspace – returns the memory requirements for the linear solver.

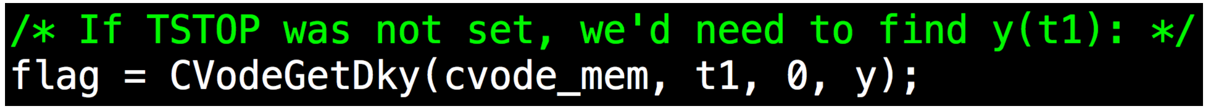
- GetNumRhsEvals, GetNumResEvals – returns the number of calls to the IVP RHS/residual function(s) by the integrator (nonlinear solve and time integration).

- GetNumLinRhsEvals, GetNumLinResEvals – returns the number of calls to the IVP RHS/residual function(s) by the linear solver (Jacobian or Jacobian-vector product approximation).

Retrieving Output from the Integrators



```
long int nst, nfe, nsetups, netf, nni;
int retval;

retval = CVodeGetNumSteps(cvode_mem, &nst);
check_retval(&retval, "CVodeGetNumSteps", 1);

retval = CVodeGetNumRhsEvals(cvode_mem, &nfe);
check_retval(&retval, "CVodeGetNumRhsEvals", 1);

retval = CVodeGetNumLinSolvSetups(cvode_mem, &nsetups);
check_retval(&retval, "CVodeGetNumLinSolvSetups", 1);

retval = CVodeGetNumErrTestFails(cvode_mem, &netf);
check_retval(&retval, "CVodeGetNumErrTestFails", 1);

retval = CVodeGetNumNonlinSolvIters(cvode_mem, &nni);
check_retval(&retval, "CVodeGetNumNonlinSolvIters", 1);
```



```
/* If TSTOP was not set, we'd need to find y(t1): */
flag = CVodeGetDky(cvode_mem, t1, 0, y);
```

Left: scalar-valued solver statistics from cvAdvDiffReac_kry.c

Right: dense solution output from cvDisc_dns.c

Advanced Features

This tutorial is only the beginning; SUNDIALS also supports a number of ‘advanced’ features to examine auxiliary conditions, change the IVP, and improve solver efficiency.

- *Root-finding* – while integrating the IVP, SUNDIALS integrators can find roots of a set of auxiliary user-defined functions $g_i(t, y(t)),\ i = 1, \ldots, N_r$; sign changes are monitored between time steps, and a modified secant iteration (along with `GetDky`) zeros in on the roots.

- *Reinitialization* – allows reuse of existing integrator memory for a “new” problem (e.g., when integrating across a discontinuity, or integrating many independent problems of the same size). All solution history and solver statistics are erased, but no memory is (de)allocated.

- *Constraint-handling* – positivity/negativity/non-positivity/non-negativity constraints may be set on individual solution components (handled through time step size adjustments).

Advanced Features – Continued

- *Resizing* (ARKODE) – allows resizing the problem and all internal vector memory, without destruction of temporal adaptivity heuristic information or solver statistics. This is primarily useful when integrating problems with spatial adaptivity.

- *Sensitivity Analysis* (CVODE/IDA) – allows computation of forward and adjoint solution sensitivities with respect to problem parameters.

- *Problem-specific algebraic solvers* – users are encouraged to supply custom nonlinear solvers, linear solvers, or preconditioners that leverage problem structure and domain-specific knowledge (see next portion of Tutorial for additional information).

Tutorial Outline

- Overview of SUNDIALS (Carol Woodward)
- How to use the time integrators (Daniel Reynolds)
- **Which nonlinear and linear solvers are available and how to use them (David Gardner)**
- Using SUNDIALS on (Pre) Exascale Machines (Cody Balos)
- Brief: How to download and install SUNDIALS (Cody Balos)

Nonlinear Systems in SUNDIALS

- SUNDIALS implicit integrators solve one or more nonlinear systems each time step:

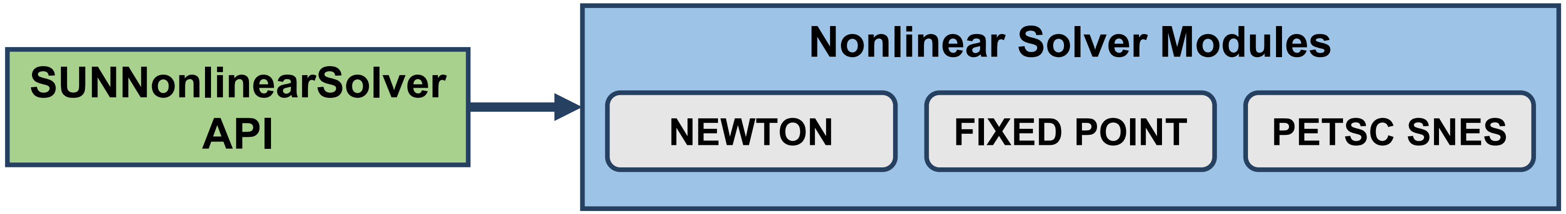
$$\left.\begin{array}{ll} \text{CVODE:} & y_n - h_n\beta_{n,0} f(t_n, y_n) - a_n = 0 \\ \text{ARKODE:} & Mz_i - h_n A^I_{i,i} f^I(t^I_{n,i}, z_i) - a_i = 0 \\ \text{IDA:} & F\left(t_n, y_n, h_n^{-1} \sum_{i=0}^{q} \alpha_{n,i} y_{n-i}\right) = 0 \end{array}\right\} F(y) = 0$$

root-finding problem

$$\left.\begin{array}{ll} \text{CVODE:} & h_n\beta_{n,0} f(t_n, y_n) + a_n = y_n \\ \text{ARKODE:} & h_n A^I_{i,i} f^I(t^I_{n,i}, z_i) + a_i = z_i \end{array}\right\} \mathrm{G}(y) = y$$

fixed-point problem

- SUNDIALS provides several nonlinear solver modules:



SUNNonlinearSolver API

Nonlinear Solver Modules

NEWTON

FIXED POINT

PETSC SNES

- User-defined or problem-specific nonlinear solver modules can be supplied by wrapping the solver according to the **SUNNonlinearSolver API**.

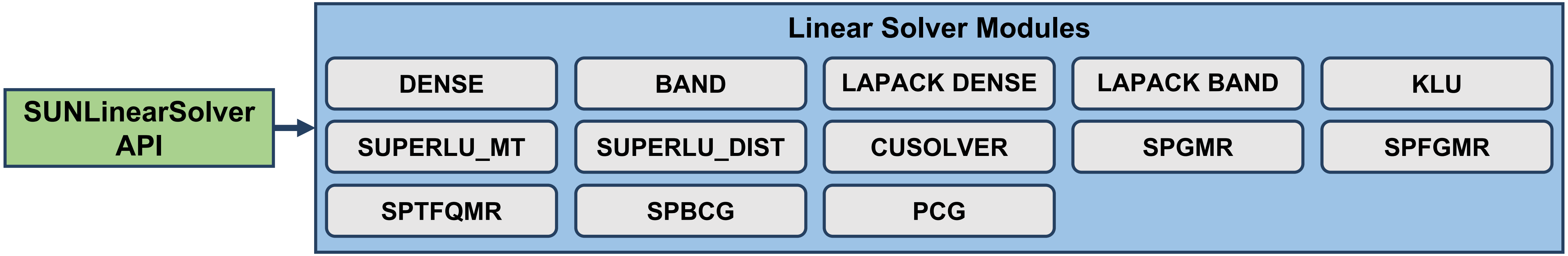
Linear Systems in SUNDIALS

- By default SUNDIALS integrators solve $F(y) = 0$ using a Newton iteration:

$$\left.\begin{aligned} y^{(m+1)} &= y^{(m)} + \delta^{m+1} \\ J\left(y^{(m)}\right)\delta^{(m+1)} &= -F\left(y^{(m)}\right) \qquad J \equiv \partial F / \partial y \end{aligned}\right\} Ax = b$$

Requires solving a general linear system each iteration

- SUNDIALS provides several nonlinear solver modules:



SUNLinearSolver API

Linear Solver Modules

DENSE

BAND

LAPACK DENSE

LAPACK BAND

KLU

SUPERLU_MT

SUPERLU_DIST

CUSOLVER

SPGMR

SPFGMR

SPTFQMR

SPBCG

PCG

- User-defined or problem-specific linear solver modules can be supplied by wrapping the solver according to the **SUNLinearSolver API**.

Linear Solver Types

- When using the default nonlinear solver (Newton), users only need to create and attach the desired linear solver object.

- SUNDIALS defines three linear solver types:
  - **Direct:** a matrix object is *required* and the solver computes the “exact” solution to the linear system defined by the matrix (e.g., Dense, LAPACK Band, KLU, SuperLU_DIST).
  - **Iterative (matrix-free):** a matrix object is *not required* and the solver computes an inexact solution to the linear system defined by the Jacobian-vector product routine (e.g., GMRES).
  - **Matrix-Iterative (matrix-based):** a matrix object is *required* and the solver computes an inexact solution to the linear system defined by the matrix (e.g., *hypre*).

- SUNDIALS provides several direct and iterative linear solver modules.

- Users may supply problem-specific direct, iterative, or matrix-iterative modules.

The “Skeleton” for Using SUNDIALS Integrators

1. Initialize parallel or multi-threaded environment
2. Create vector of initial values, $y_0 \in \mathbb{R}^{\mathrm{N}}$; if using IDA, also create $\dot{y}_0 \in \mathbb{R}^N$
3. Create and initialize integrator object (attaches $t_0, y_0, (\dot{y}_0)$, RHS/residual function(s))
4. **Create matrix, linear solver, nonlinear solver objects (if applicable); attach to integrator**
5. **Specify optional inputs to integrator and solver objects (tolerances, etc.)**
6. Advance solution in time, either over specified time intervals $[a, b]$, or for single timesteps
7. Retrieve optional outputs
8. Free solution/solver memory; finalize MPI (if applicable)

Creating & Attaching a Linear Solver

Direct linear solver example (Dense):
cvode/serial/cvRoberts_dns.c

a) Create a SUNMatrix object

```
SUNMatrix A =
  SUNDenseMatrix(NEQ, NEQ)
```

b) Create the SUNLinearSolver object

```
SUNLinearSolver LS =
  SUNLinSol_Dense(y, A)
```

c) Attach the linear solver

```
flag =
  CVodeSetLinearSolver(cvode_mem,
                       LS, A)
```

Iterative linear solver example (GMRES):
ida/parallel/idaFoodWeb_kry_p.c

a) Create the SUNLinearSolver object

```
SUNLinearSolver LS =
  SUNLinSol_SPGMR(cc, PREC_LEFT,
                  maxl)
```

b) Attach the linear solver

```
flag =
  IDASetLinearSolver(ida_mem,
                     LS, NULL)
```

Supplying a Jacobian or Jacobian-vector Product Function

- For direct or matrix-iterative linear solvers:
  - SetJacFn – specifies a user-supplied function to evaluate the Jacobian $J \equiv \partial f / \partial y$.
  - SetLinSysFn – specifies a user-supplied function to evaluate the linear system $A \equiv I - \gamma J$
    - For dense and banded matrices the Jacobian may be computed internally with finite differences (default).
    - For a sparse or user-defined matrix, the Jacobian or linear system function must be supplied.

- For iterative linear solvers:
  - SetJacTimes – specifies user-supplied Jacobian-vector product setup and times functions.
    - By default Jacobian-vector products are computed internally using a finite difference

Supplying a Preconditioner (ida/parallel/idaFoodWeb_kry_p.c)

- The IDASetPreconditioner function sets the preconditioner setup and solve functions:
  - The **setup** function preprocesses and/or evaluates Jacobian-related data needed by the preconditioner:

```
IDALsPrecSetupFn(realtype tt, N_Vector yy, N_Vector yp,
                 N_Vector rr, realtype c_j, void* user_data);
```

  - The **solve** function solves the preconditioner system $Pz = r$:

```
IDALsPrecSolvFn(realtype tt, N_Vector yy, N_Vector yp, N_Vector rr,
                N_Vector rvec, N_Vector zvec, realtype c_j,
                realtype delta, void* user_data)
```

Integrator and Linear Solver Options

- Solver specific options include:
  - SetGSType– sets the Gram-Schmidt orthogonalization type (CLASSICAL or MODIFIED).
  - SetMaxRestarts – sets the max number of GMRES restarts.
  - SetMaxl – updates the number of linear solver iterations.
- Integrator options include:
  - SetMaxStepsBetweenJac – (CVODE and ARKODE) – specifies the number of steps to wait before recommending to update the Jacobian or preconditioner.
  - SetMaxStepsBetweenLSet – (ARKODE) – specifies the number of steps between calls to the linear solver setup routine to potentially update the Jacobian or preconditioner.
  - SetEpsLin – specifies the scaling factor used to set the linear solver tolerance.

Nonlinear solver options

- SetMaxNonlinIters – sets the maximum number of nonlinear iterations.

- SetNonlinConvCoef – specifies the scaling factor used to set the nonlinear solver tolerance.

- Additional ARKODE options:
  - SetLinear – specifies if the implicit system is linearly implicit.
  - SetNonlinCRDown – sets the nonlinear convergence rate constant.
  - SetNonlinRDiv – sets the nonlinear divergence ratio.

Creating & Attaching a Non-default Nonlinear Solver

SUNDIALS fixed point solver:
cvode/parallel/cvAdvDiff_non_p.c

a) Create the SUNNonlinearSolver object

```
SUNNonlinearSolver NLS =
  SUNNonlinSol_FixedPoint(y, m)
```

b) Attach the nonlinear solver

```
flag =
CVodeSetNonlinearSolver(cv_mem,
                        NLS)
```

PETSc SNES:
arkode/C_petsc/ark_petsc_ex25.c

a) Create the SNES object

```
ierr = SNESCreate(my_comm, &snes)
```

b) Create the SUNNonlinearSolver object

```
SUNNonlinearSolver NLS =
  SUNNonlinSol_PetscSNES(y, snes)
```

c) Attach the linear solver

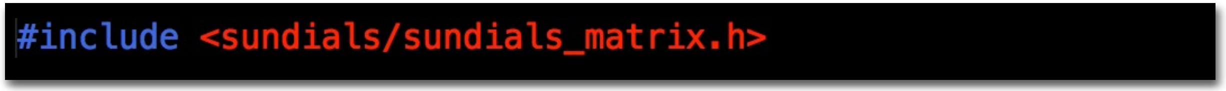
```
ierr =
  ARKStepSetLinearSolver(ark_mem,
                         NLS)
```

Supplying a Custom Linear Solver

- Example interfacing a *hypre* linear solver (and preconditioner) with ARKode:
  - examples/arkode/CXX_parhyp/ark_heat_2D_hypre.cpp
  - Creates a SUNMatrix wrapper for a *hypre* structured grid matrix
  - Creates a SUNLinearSolver wrapper for the *hypre* PCG solver with PFMG preconditioner
  - Matrix-iterative linear solver type

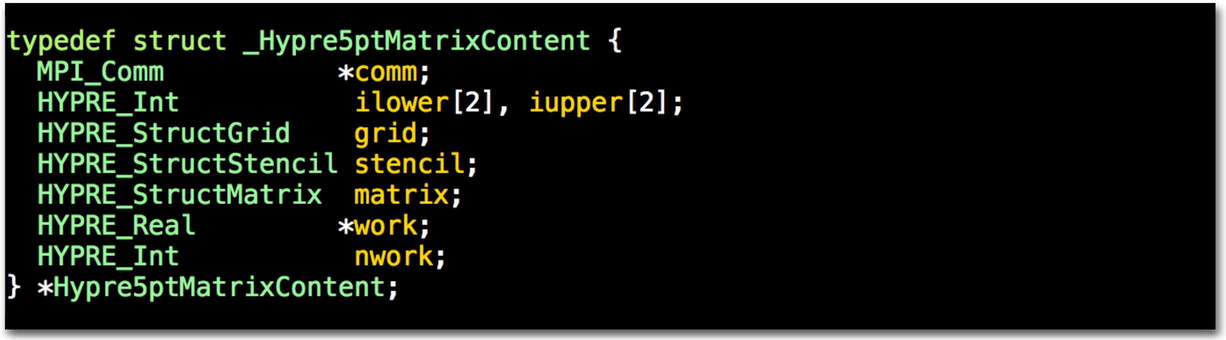
Creating a SUNMatrix Wrapper

- Header defining a **SUNMatrix**



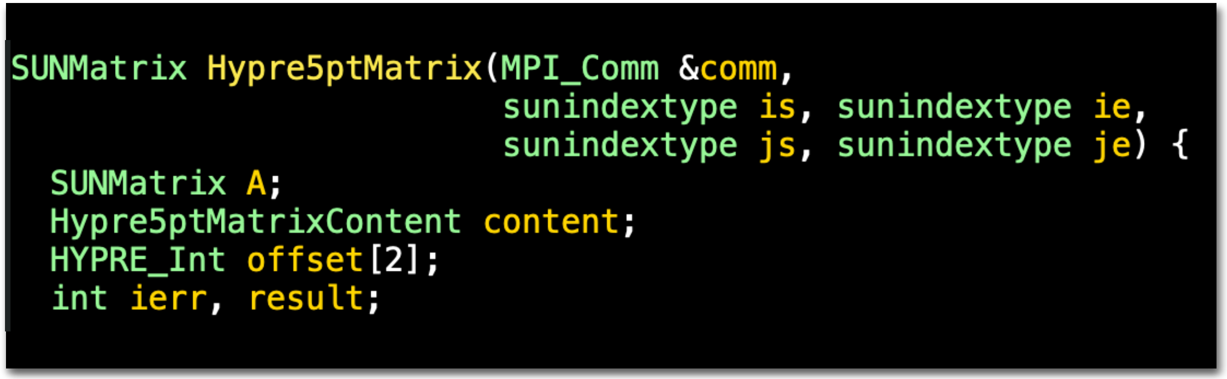
```
#include <sundials/sundials_matrix.h>
```

- Matrix specific content structure



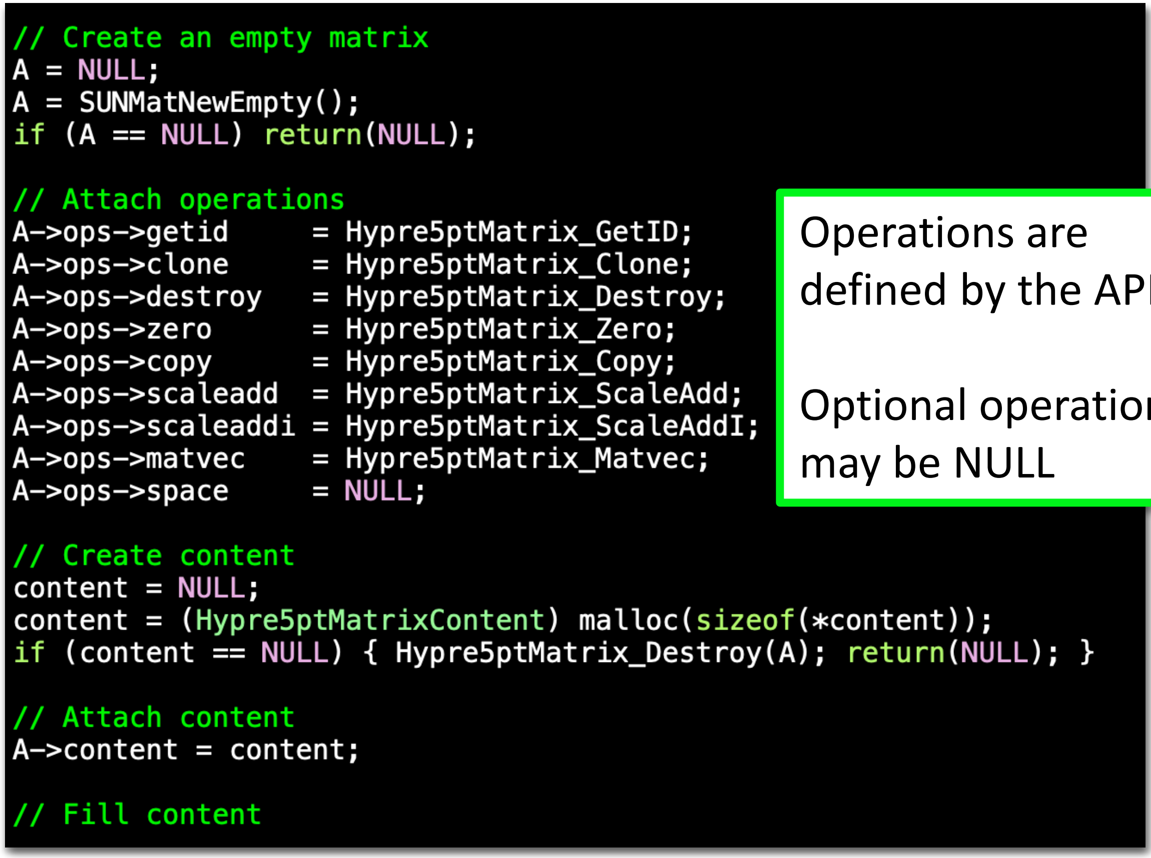
```
typedef struct _Hypre5ptMatrixContent {
  MPI_Comm            *comm;
  HYPRE_Int            ilower[2], iupper[2];
  HYPRE_StructGrid     grid;
  HYPRE_StructStencil  stencil;
  HYPRE_StructMatrix   matrix;
  HYPRE_Real          *work;
  HYPRE_Int            nwork;
} *Hypre5ptMatrixContent;
```

- Constructor to create a new matrix



```
SUNMatrix Hypre5ptMatrix(MPI_Comm &comm,
                         sunindextype is, sunindextype ie,
                         sunindextype js, sunindextype je) {
  SUNMatrix A;
  Hypre5ptMatrixContent content;
  HYPRE_Int offset[2];
  int ierr, result;
```

- Constructor continued



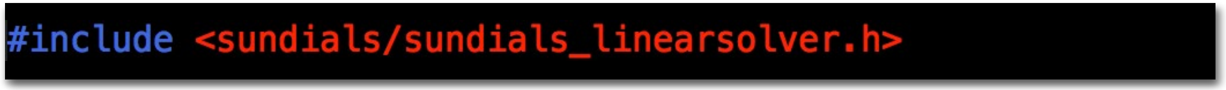
```
// Create an empty matrix
A = NULL;
A = SUNMatNewEmpty();
if (A == NULL) return(NULL);

// Attach operations
A->ops->getid     = Hypre5ptMatrix_GetID;
A->ops->clone     = Hypre5ptMatrix_Clone;
A->ops->destroy   = Hypre5ptMatrix_Destroy;
A->ops->zero      = Hypre5ptMatrix_Zero;
A->ops->copy      = Hypre5ptMatrix_Copy;
A->ops->scaleadd  = Hypre5ptMatrix_ScaleAdd;
A->ops->scaleaddi = Hypre5ptMatrix_ScaleAddI;
A->ops->matvec    = Hypre5ptMatrix_Matvec;
A->ops->space     = NULL;

// Create content
content = NULL;
content = (Hypre5ptMatrixContent) malloc(sizeof(*content));
if (content == NULL) { Hypre5ptMatrix_Destroy(A); return(NULL); }

// Attach content
A->content = content;

// Fill content
```

Operations are defined by the API

Optional operations may be NULL

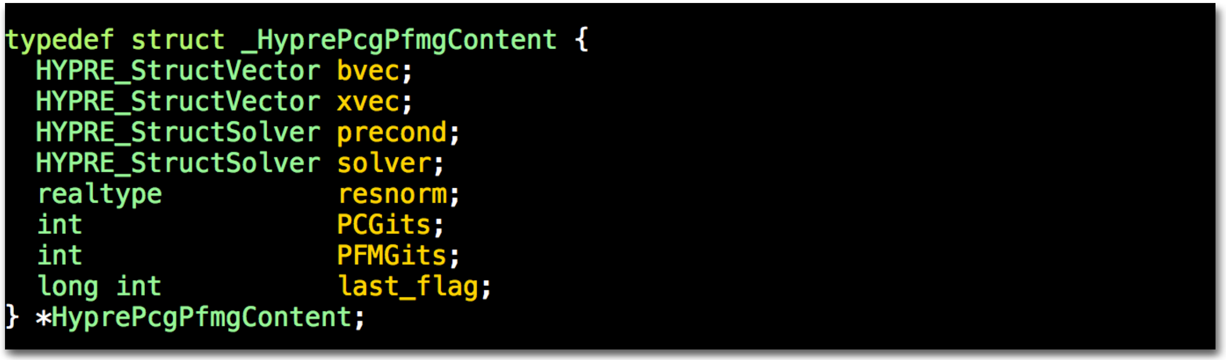
Creating a SUNLinearSolver Wrapper

- Header defining a **SUNLinearSolver**



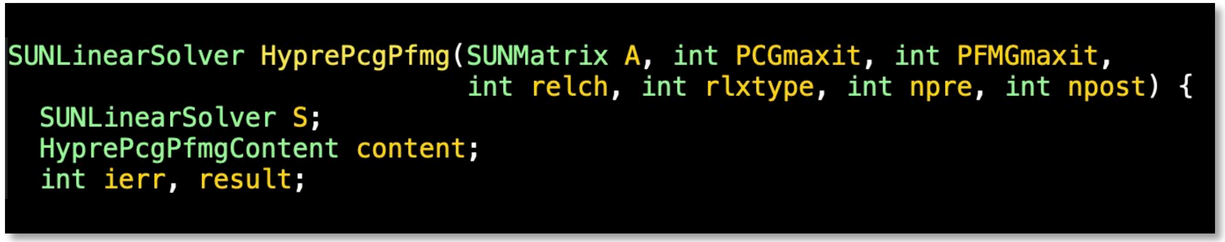
```
#include <sundials/sundials_linearsolver.h>
```

- Linear solver specific content structure



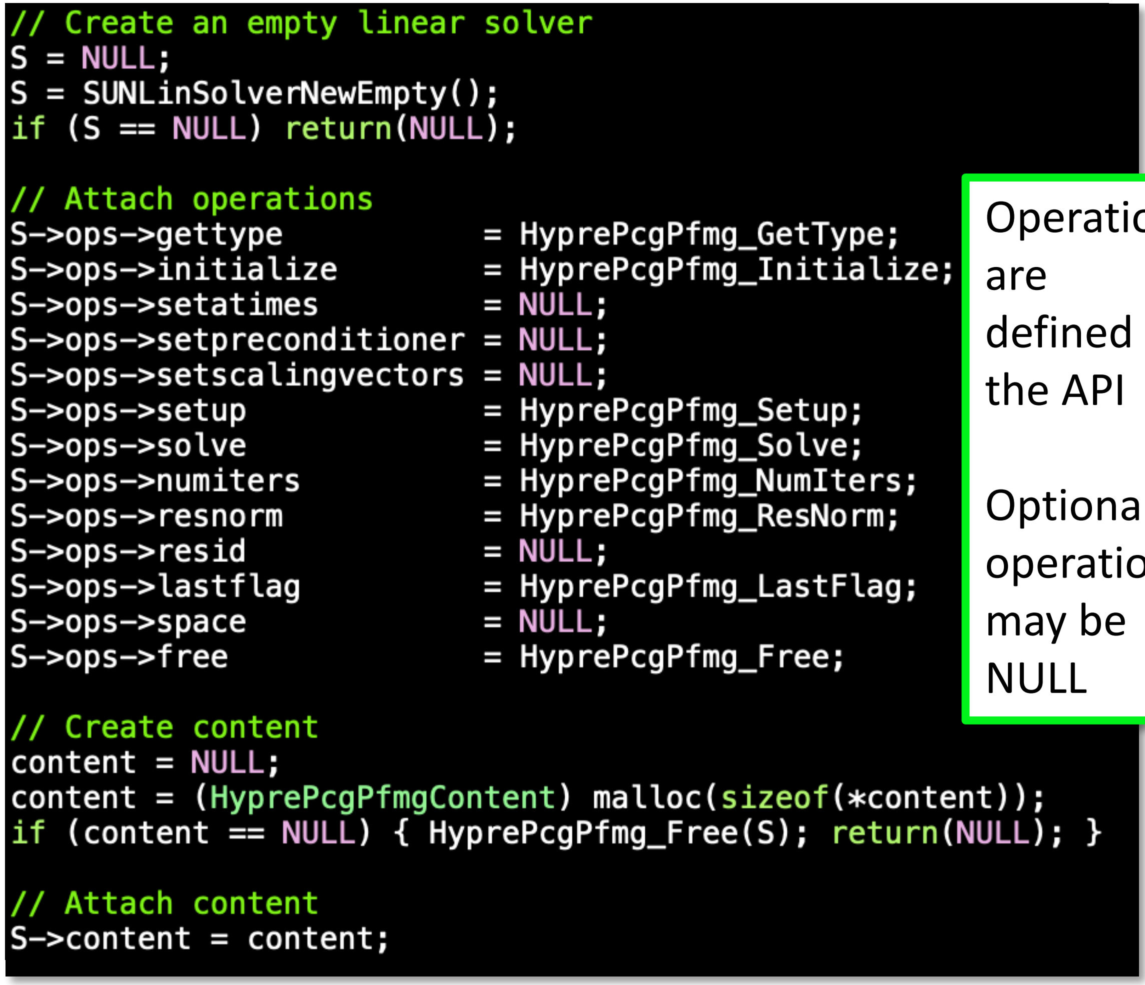
```
typedef struct _HyprePcgPfmgContent {
  HYPRE_StructVector bvec;
  HYPRE_StructVector xvec;
  HYPRE_StructSolver precond;
  HYPRE_StructSolver solver;
  realtype           resnorm;
  int                PCGits;
  int                PFMGits;
  long int           last_flag;
} *HyprePcgPfmgContent;
```

- Constructor to create a new linear solver



```
SUNLinearSolver HyprePcgPfmg(SUNMatrix A, int PCGmaxit, int PFMGmaxit,
                             int relch, int rlxtype, int npre, int npost) {
  SUNLinearSolver S;
  HyprePcgPfmgContent content;
  int ierr, result;
```

- Constructor continued



```
// Create an empty linear solver
S = NULL;
S = SUNLinSolverNewEmpty();
if (S == NULL) return(NULL);

// Attach operations
S->ops->gettype             = HyprePcgPfmg_GetType;
S->ops->initialize          = HyprePcgPfmg_Initialize;
S->ops->setatimes           = NULL;
S->ops->setpreconditioner = NULL;
S->ops->setscalingvectors = NULL;
S->ops->setup               = HyprePcgPfmg_Setup;
S->ops->solve               = HyprePcgPfmg_Solve;
S->ops->numiters            = HyprePcgPfmg_NumIters;
S->ops->resnorm             = HyprePcgPfmg_ResNorm;
S->ops->resid               = NULL;
S->ops->lastflag            = HyprePcgPfmg_LastFlag;
S->ops->space               = NULL;
S->ops->free                = HyprePcgPfmg_Free;

// Create content
content = NULL;
content = (HyprePcgPfmgContent) malloc(sizeof(*content));
if (content == NULL) { HyprePcgPfmg_Free(S); return(NULL); }

// Attach content
S->content = content;
```

Operations are defined by the API

Optional operations may be NULL

Creating & Attaching the User-supplied Linear Solver

a) Create the SUNMatrix object

```
SUNMatrix A = MyNewMatrix(…)
```

b) Create the SUNLinearSolver object

```
SUNLinearSolver LS = MyNewLinearSolver(…)
```

c) Attach the linear solver e.g.,

```
flag = ARKStepSetLinearSolver(mem, LS, A)
```

d) Set the function to compute the Jacobian (or linear system)

```
flag = ARKStepSetJacFn(mem, JacFn)
```

Creating a SUNNonlinearSolver Wrapper

Note SUNDIALS integrators pose the nonlinear systems to solve in predictor-corrector form.

- Header defining a **SUNNonLinearSolver**

```
#include "sundials/sundials_nonlinearsolver.h"
```

- Nonlinear solver specific content structure

```
typedef struct _SUNNonlinearSolverContent_Newton {

  /* functions provided by the integrator */
  SUNNonlinSolSysFn      Sys;
  SUNNonlinSolLSetupFn   LSetup;
  SUNNonlinSolLSolveFn   LSolve;
  SUNNonlinSolConvTestFn CTest;

  /* nonlinear solver variables */
  N_Vector    delta;
  booleantype jcur;
  int         curiter;
  int         maxiters;
  long int    niters;
  long int    nconvfails;
  void*       ctest_data;
} *SUNNonlinearSolverContent_Newton;
```

- Constructor to create a new nonlinear solver

```
SUNNonlinearSolver SUNNonlinSol_Newton(N_Vector y)
{
  SUNNonlinearSolver NLS;
  SUNNonlinearSolverContent_Newton content;

  /* Create an empty nonlinear linear solver object */
  NLS = NULL;
  NLS = SUNNonlinSolNewEmpty();
  if (NLS == NULL) return(NULL);

  /* Attach operations */
  NLS->ops->gettype          = SUNNonlinSolGetType_Newton;
  NLS->ops->initialize       = SUNNonlinSolInitialize_Newton;
  NLS->ops->solve            = SUNNonlinSolSolve_Newton;
  NLS->ops->free             = SUNNonlinSolFree_Newton;
  NLS->ops->setsysfn         = SUNNonlinSolSetSysFn_Newton;
  NLS->ops->setlsetupfn      = SUNNonlinSolSetLSetupFn_Newton;
  NLS->ops->setlsolvefn      = SUNNonlinSolSetLSolveFn_Newton;
  NLS->ops->setctestfn       = SUNNonlinSolSetConvTestFn_Newton;
  NLS->ops->setmaxiters      = SUNNonlinSolSetMaxIters_Newton;
  NLS->ops->getnumiters      = SUNNonlinSolGetNumIters_Newton;
  NLS->ops->getcuriter       = SUNNonlinSolGetCurIter_Newton;
  NLS->ops->getnumconvfails = SUNNonlinSolGetNumConvFails_Newton;

  /* Create content */
  content = NULL;
  content = (SUNNonlinearSolverContent_Newton) malloc(sizeof *content);
  if (content == NULL) { SUNNonlinSolFree(NLS); return(NULL); }

  /* Attach content */
  NLS->content = content;

  /* Fill content */
```

Operations are defined by the API

Optional operations may be NULL

Tutorial Outline

- Overview of SUNDIALS (Carol Woodward)
- How to use the time integrators (Daniel Reynolds)
- Which nonlinear and linear solvers are available and how to use them (David Gardner)
- **Using SUNDIALS on (Pre) Exascale Machines (Cody Balos)**
- Brief: How to download and install SUNDIALS (Cody Balos)

Using SUNDIALS on (Pre) Exascale Machines – Outline

- An introduction to SUNDIALS features for Pre-Exascale and Exascale machines
- ManyVector and MPI+X NVector modules
- A dive into the CUDA NVector module
- Enabling fused vector operations
- SUNDIALS GPU capable linear solvers
- Steps to using SUNDIALS with GPUs
- A High-Level look at a GPU-enabled SUNDIALS example
- SUNDIALS GPU performance considerations

Using SUNDIALS on (Pre) Exascale Machines

- The Exascale landscape:
  - Heterogeneous computational architectures
  - GPUs provide most of the FLOPS

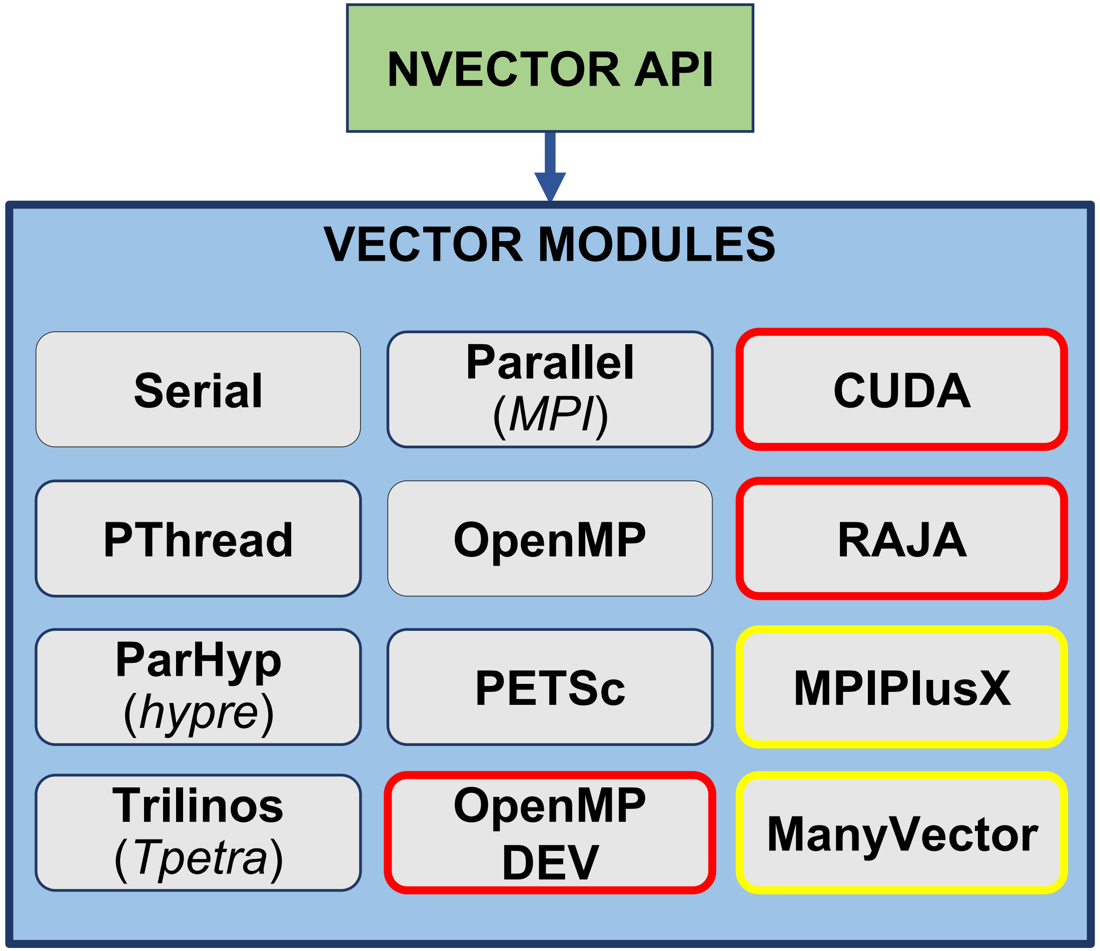
- Two main strategies for using SUNDIALS:
  1. SUNDIALS controls the main time-integration loop for the application, and a large ODE system is solved in a distributed manner (e.g. FEM applications)
  2. SUNDIALS is used as a local integrator for many small independent subsystems (e.g. per grid cell in an Adaptive Mesh Refinement application)

- For strategy 1 at Exascale, the MPI ManyVector and MPI+X NVector modules as well as several GPU-enabled “local” NVector modules are useful SUNDIALS features

- For strategy 2 at Exascale, the ManyVector NVector module, several GPU-enabled “local” NVectors, and a SUNLinearSolver module for batched solves of small systems are useful SUNDIALS features

SUNDIALS NVector API

- SUNDIALS' integrators do not directly modify solution data; this is modified through vector operations, e.g., vector adds, norms, etc., defined by the *NVector API*
- Several optional NVector implementations (modules) are released as a part of SUNDIALS
  - CUDA, RAJA, and OpenMPDEV (target offloading) modules provide GPU support
  - Parallel, ParHyp, PETSc, and Trilinos modules are MPI distributed
  - ManyVector and MPIPlusX modules provide support for hybrid computation
- It is straightforward to implement a problem-specific NVector tailored to an application



The SUNDIALS ManyVector NVector module

- A mechanism for users to partition their simulation data among disparate computational resources
  - E.g., CPUs and GPUs
- Does not touch any vector data directly, instead it is a software layer to treat a collection of other NVector objects as a single cohesive NVector
- Can be used to easily partition data within a node or across nodes
- Also can be used to combine distinct MPI intracommunicators together into a multi-physics simulation

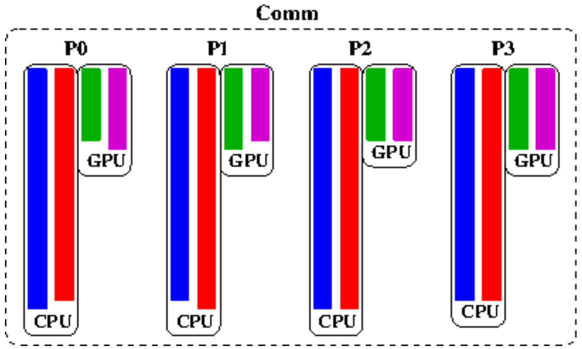


Figure 1, ManyVector use case for multi-rate or data partitioning, allowing for each vector to utilize distinct processing elements within the same node (e.g. red/blue on CPU and green/magenta on GPU) or for collective communications to be combined to minimize latency overhead (e.g., during Gram-Schmidt orthogonalization within linear or nonlinear solvers).

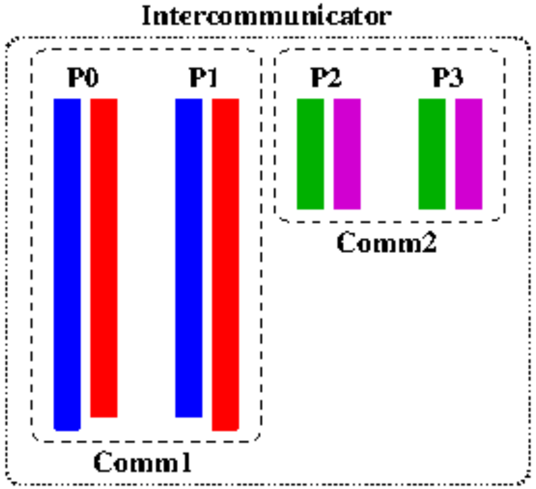
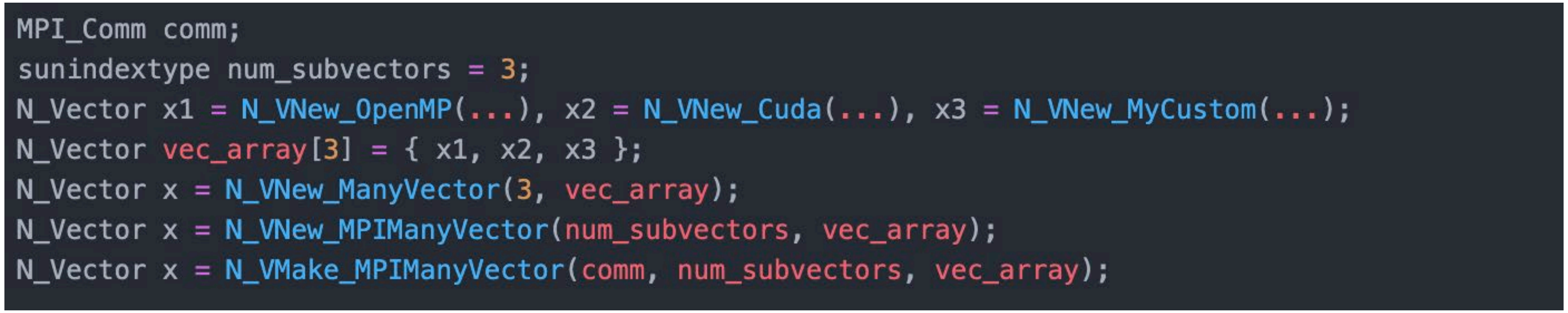


Figure 2, ManyVector use case for process-based multiphysics decompositions, wherein Comm1 connects processes 0 and 1 with physics operating on red/blue data, Comm2 connects processes 2 and 3 with physics operating on green/magenta data, and an MPI intercommunicator allows multiphysics coupling.

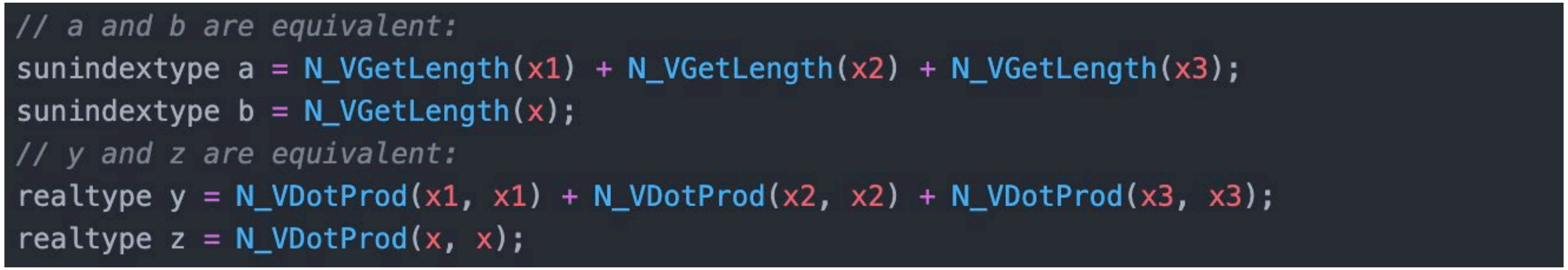
Using the ManyVector

- If MPI is needed, include nvector_mpimanyvector.h, otherwise include nvector_manyvector.h
- Constructors take an array of other NVector objects:



```
MPI_Comm comm;
sunindextype num_subvectors = 3;
N_Vector x1 = N_VNew_OpenMP(...), x2 = N_VNew_Cuda(...), x3 = N_VNew_MyCustom(...);
N_Vector vec_array[3] = { x1, x2, x3 };
N_Vector x = N_VNew_ManyVector(3, vec_array);
N_Vector x = N_VNew_MPIManyVector(num_subvectors, vec_array);
N_Vector x = N_VMake_MPIManyVector(comm, num_subvectors, vec_array);
```

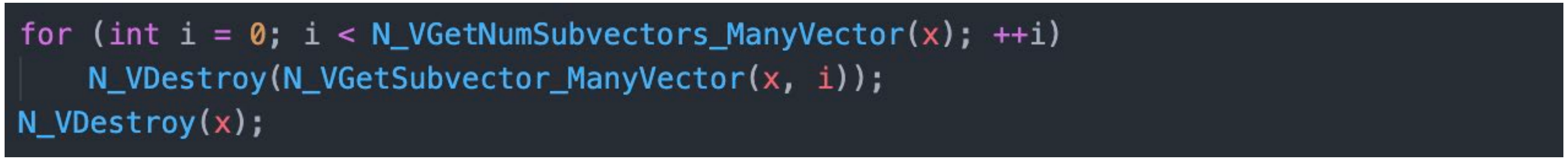
- After construction, the ManyVector behaves like a single cohesive vector with data ordered according to the ordering of the subvectors in the vector array:



```
// a and b are equivalent:
sunindextype a = N_VGetLength(x1) + N_VGetLength(x2) + N_VGetLength(x3);
sunindextype b = N_VGetLength(x);
// y and z are equivalent:
realtype y = N_VDotProd(x1, x1) + N_VDotProd(x2, x2) + N_VDotProd(x3, x3);
realtype z = N_VDotProd(x, x);
```

Using the ManyVector

- The N_VGetSubvector_[MPI]ManyVector function can be called to access the the subvectors after construction of the ManyVector
- N_VGetSubvectorArrayPointer_[MPI]ManyVector and N_VSetSubvectorArrayPointer_[MPI]ManyVector are available convenience functions for accessing the data of a subvector, but note that not all subvectors may have data that is directly accessible (e.g. the CUDA NVector when using device memory)
- *Note:* calling N_VDestroy on the ManyVector object does not destroy the subvectors!
  - Need to destroy each subvector, then free the ManyVector:

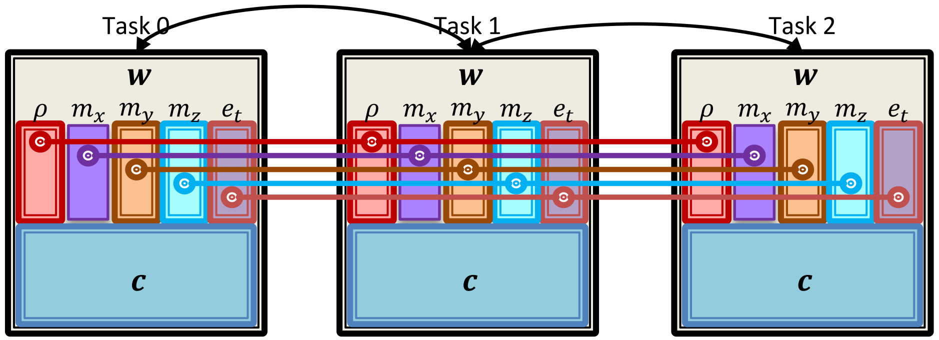


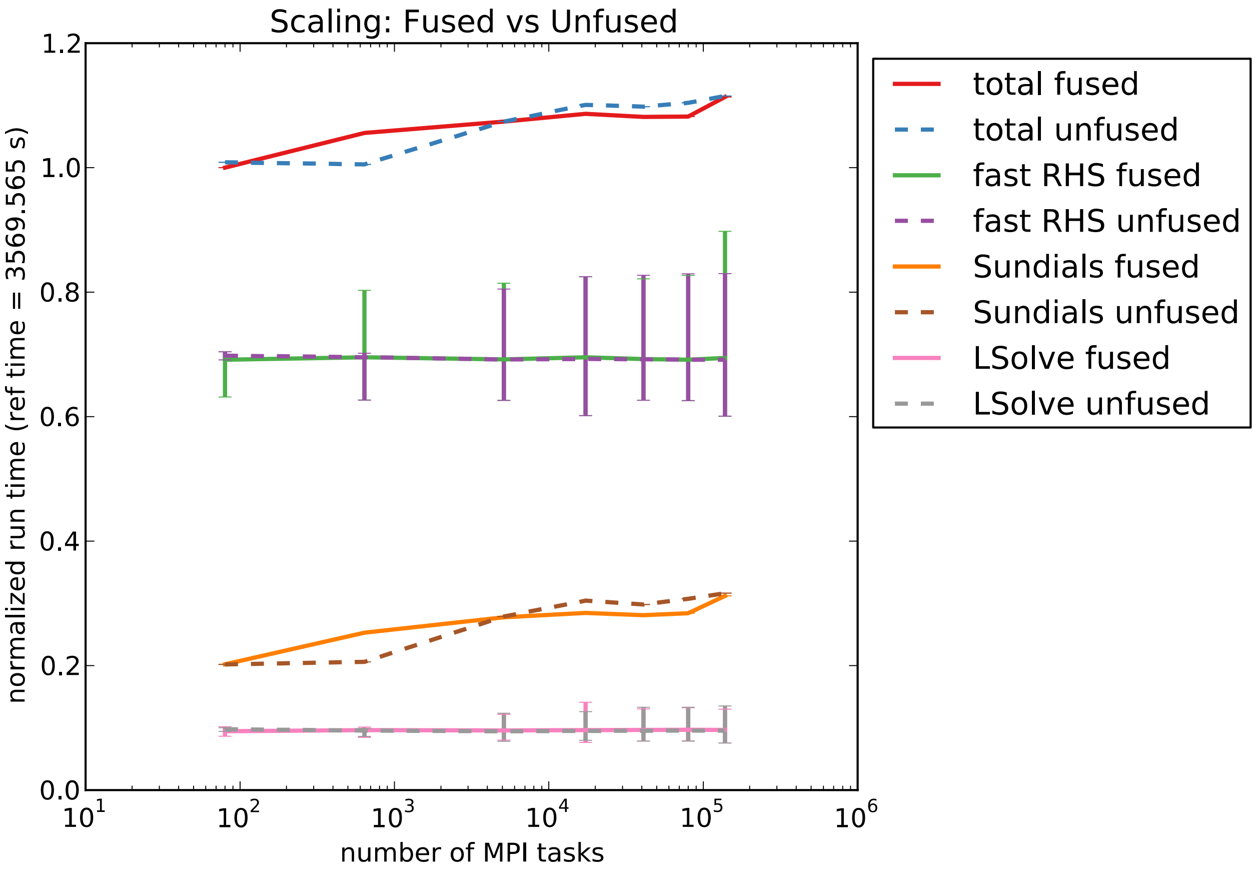
```
for (int i = 0; i < N_VGetNumSubvectors_ManyVector(x); ++i)
    N_VDestroy(N_VGetSubvector_ManyVector(x, i));
N_VDestroy(x);
```

ManyVector Performance Results

- Developed a *scalable multiphysics demonstration code* using the new many-vector module, fused vector operations, and a third order explicit-implicit multirate integrator

$$\frac{\partial \boldsymbol{w}}{\partial t} = -\nabla \cdot \boldsymbol{F}(\boldsymbol{w}) + \boldsymbol{R}(\boldsymbol{w}) + \boldsymbol{G}(\boldsymbol{x}, t)$$





- Observed 90% weak scaling efficiency using 40 MPI ranks on each of 2 to 3,456 nodes of OLCF Summit (80 to 138,240 CPU cores)

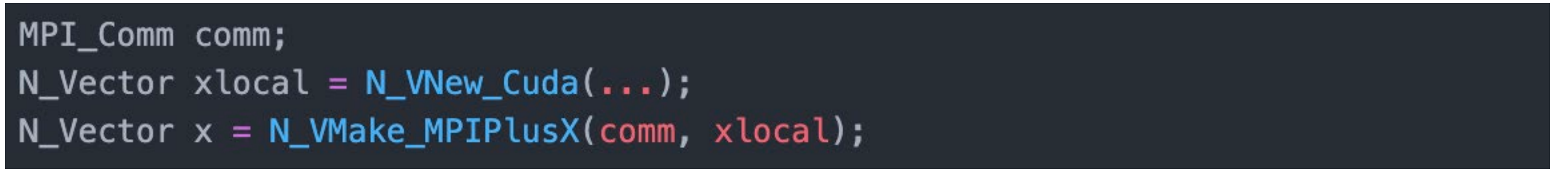
The SUNDIALS MPI+X NVector

- The MPI+X NVector adds MPI capabilities to any local (single-node) NVector “X”

- Really just a ManyVector with a single subvector and some convenience functions

- Defined in the header nvector_mpiplusx.h

- The constructor takes an MPI communicator and a local NVector



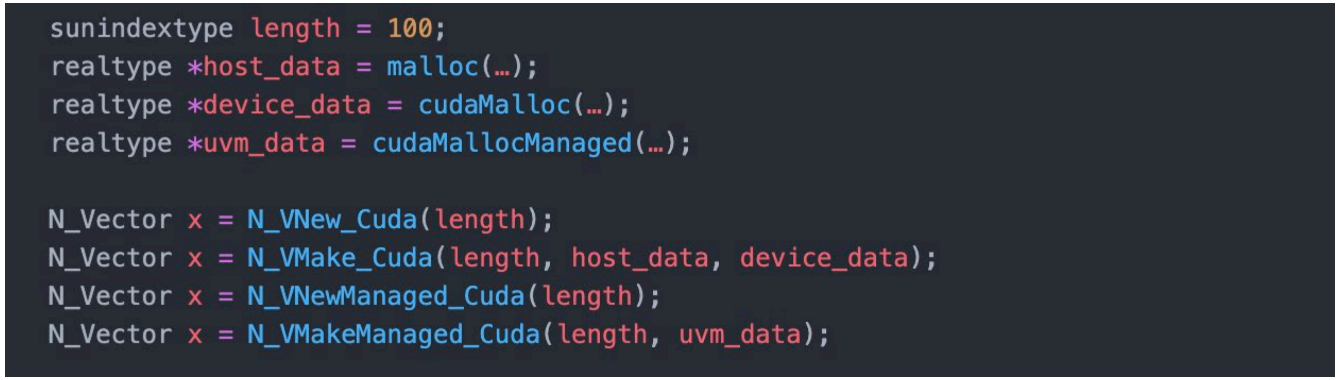
```
MPI_Comm comm;
N_Vector xlocal = N_VNew_Cuda(...);
N_Vector x = N_VMake_MPIPlusX(comm, xlocal);
```

- *Note*: you cannot call the local vector specific functions on the MPI+X vector
  - E.g. cannot do N_VCopyToDevice_Cuda(x), instead do N_VCopyToDevice_Cuda(xlocal)
  - The local vector can be accessed with the N_VGetLocalVector_MPIPlusX function
  - Other functions are provided for working with the local vector indirectly through the MPIPlusX vector

- *Note*: calling N_VDestroy on the MPI+X NVector object does not destroy the local vector, you must destroy the local vector separately

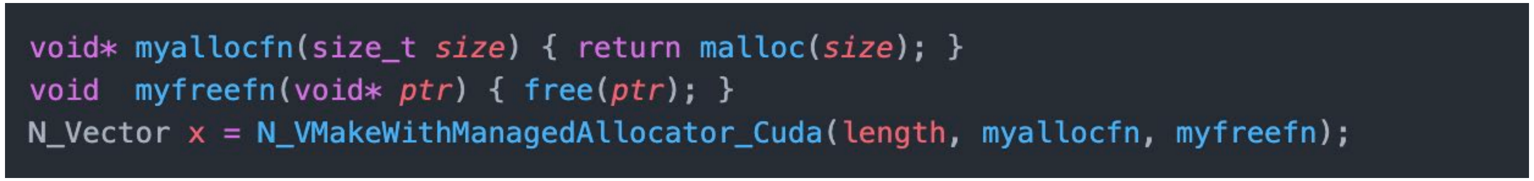
The CUDA NVector Module

- Can be constructed to utilize separate host and device memory or managed (unified virtual memory):



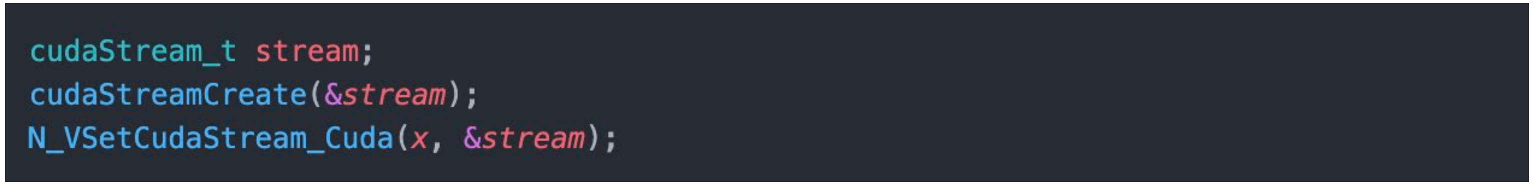
```
sunindextype length = 100;
realtype *host_data = malloc(…);
realtype *device_data = cudaMalloc(…);
realtype *uvm_data = cudaMallocManaged(…);

N_Vector x = N_VNew_Cuda(length);
N_Vector x = N_VMake_Cuda(length, host_data, device_data);
N_Vector x = N_VNewManaged_Cuda(length);
N_Vector x = N_VMakeManaged_Cuda(length, uvm_data);
```

- If using managed memory, a user can provide their own custom memory allocator routine. This is useful if you have a memory pool that you want to utilize.



```
void* myallocfn(size_t size) { return malloc(size); }
void  myfreefn(void* ptr) { free(ptr); }
N_Vector x = N_VMakeWithManagedAllocator_Cuda(length, myallocfn, myfreefn);
```

- To enable concurrent CUDA kernel execution, users can set the CUDA stream on which the CUDA kernels launched by the vector will execute:



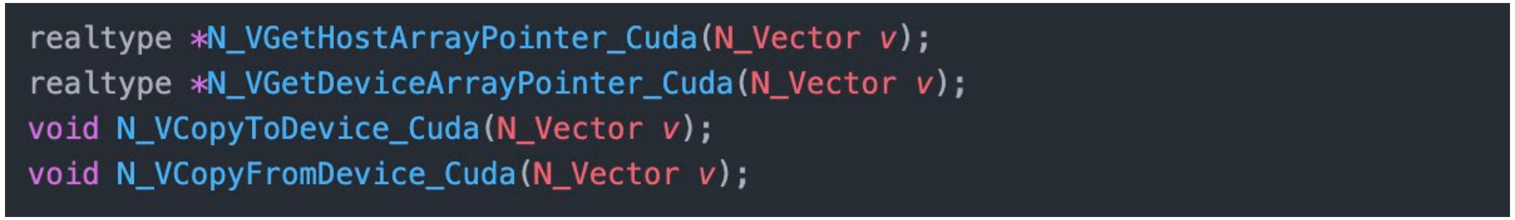
```
cudaStream_t stream;
cudaStreamCreate(&stream);
N_VSetCudaStream_Cuda(x, &stream);
```

The CUDA NVector Module

- UVM can be a good way to get started with the CUDA NVector since it is like using the serial NVector
  - Users don't have to worry about data coherency
  - Users can access underlying data with the generic `N_VGetArrayPointer` function

- Using separate host and device memory offers significantly better performance than UVM and is **recommended for production usage**

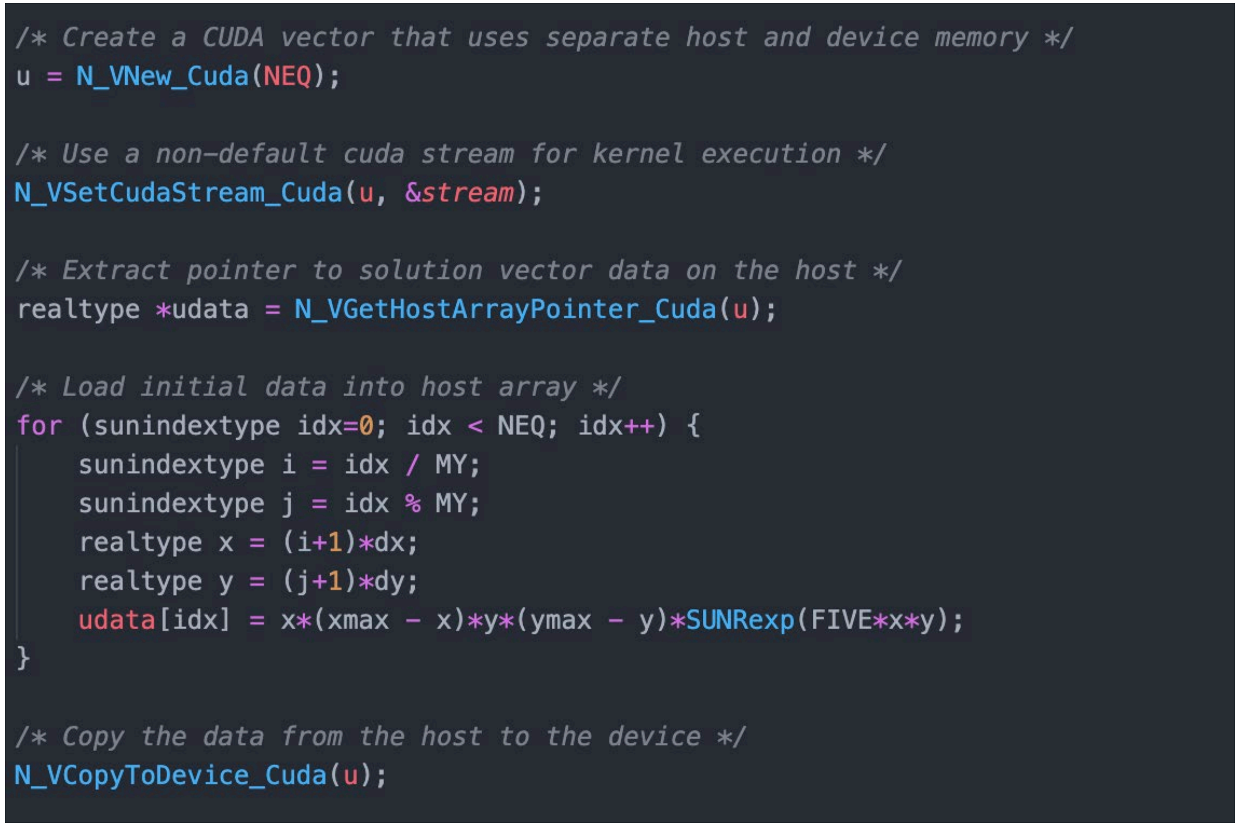
- When using separate host and device memory, these four functions will be useful:



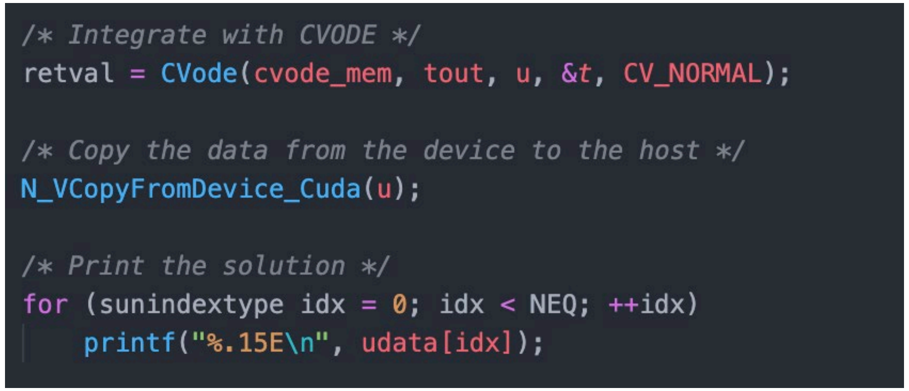
```
realtype *N_VGetHostArrayPointer_Cuda(N_Vector v);
realtype *N_VGetDeviceArrayPointer_Cuda(N_Vector v);
void N_VCopyToDevice_Cuda(N_Vector v);
void N_VCopyFromDevice_Cuda(N_Vector v);
```

- When using separate host and device memory, users must manually manage data coherency!
  - After receiving control from SUNDIALS, you must call `N_VCopyFromDevice_Cuda` if you want to access the data on the host
  - If you modify data on the host, you must then copy it to the device with `N_VCopyToDevice_Cuda` before passing control back to SUNDIALS
  - SUNDIALS only operates on the device data and *never moves it*

CUDA NVector Code Example



```
/* Create a CUDA vector that uses separate host and device memory */
u = N_VNew_Cuda(NEQ);

/* Use a non-default cuda stream for kernel execution */
N_VSetCudaStream_Cuda(u, &stream);

/* Extract pointer to solution vector data on the host */
realtype *udata = N_VGetHostArrayPointer_Cuda(u);

/* Load initial data into host array */
for (sunindextype idx=0; idx < NEQ; idx++) {
    sunindextype i = idx / MY;
    sunindextype j = idx % MY;
    realtype x = (i+1)*dx;
    realtype y = (j+1)*dy;
    udata[idx] = x*(xmax - x)*y*(ymax - y)*SUNRexp(FIVE*x*y);
}

/* Copy the data from the host to the device */
N_VCopyToDevice_Cuda(u);
```



```
/* Integrate with CVODE */
retval = CVode(cvode_mem, tout, u, &t, CV_NORMAL);

/* Copy the data from the device to the host */
N_VCopyFromDevice_Cuda(u);

/* Print the solution */
for (sunindextype idx = 0; idx < NEQ; ++idx)
    printf("%.15E\n", udata[idx]);
```

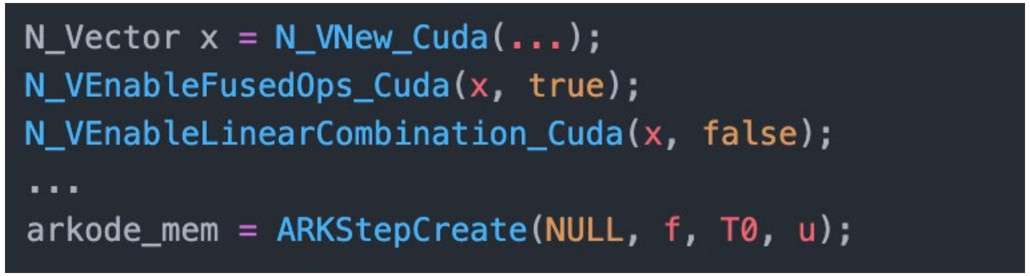
Enabling Fused Vector Operations

- Fused vector operations increase computation per vector operation

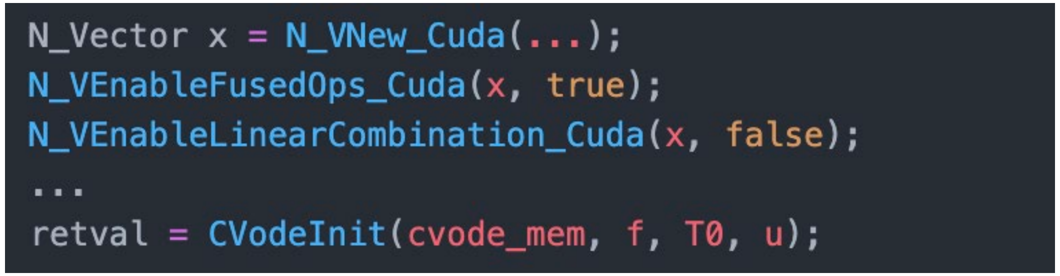
- Are particularly interesting when using GPUs because the CUDA kernel launch overhead associated with an operation is high

- The NVector API defines 9 fused vector operations
  - Can be enabled/disabled for vectors at runtime
  - Can be enabled/disable individually or together

- *Note:* Fused operations should be enabled/disabled prior to attaching the vector to an integrator:



```
N_Vector x = N_VNew_Cuda(...);
N_VEnableFusedOps_Cuda(x, true);
N_VEnableLinearCombination_Cuda(x, false);
...
arkode_mem = ARKStepCreate(NULL, f, T0, u);
```



```
N_Vector x = N_VNew_Cuda(...);
N_VEnableFusedOps_Cuda(x, true);
N_VEnableLinearCombination_Cuda(x, false);
...
retval = CVodeInit(cvode_mem, f, T0, u);
```

SUNDIALS GPU capable linear solvers

- All SUNDIALS iterative linear solvers are MPI and GPU ready
  - These solvers only modify data via vector operations, so you just need to use an appropriate NVector
  - Users can provide preconditioners to reduce the number of iterations
  - Perform well under strategy 1, but do not perform well under strategy 2 due to kernel launch overhead
- SUNDIALS also offers an interface to the cuSOLVER sparse batched QR method
  - Designed for block-diagonal linear systems where the matrix is of the form:

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

  - All blocks A_j must share the same sparsity pattern
  - This type of system arises when grouping small independent subsystems together
- Alternatively, you can provide a custom linear solver that conforms to the SUNLinearSolver API
  - This allows users to take advantage of new solvers quickly

Steps to using SUNDIALS with GPUs

1. Identify your usage strategy (see the second slide in this section of the tutorial)

2. Switch to a GPU-enabled NVector
   - Pair with the MPI ManyVector or MPI+X NVector module if distributing across nodes
   - Can use your own GPU-enabled data structures under a custom NVector

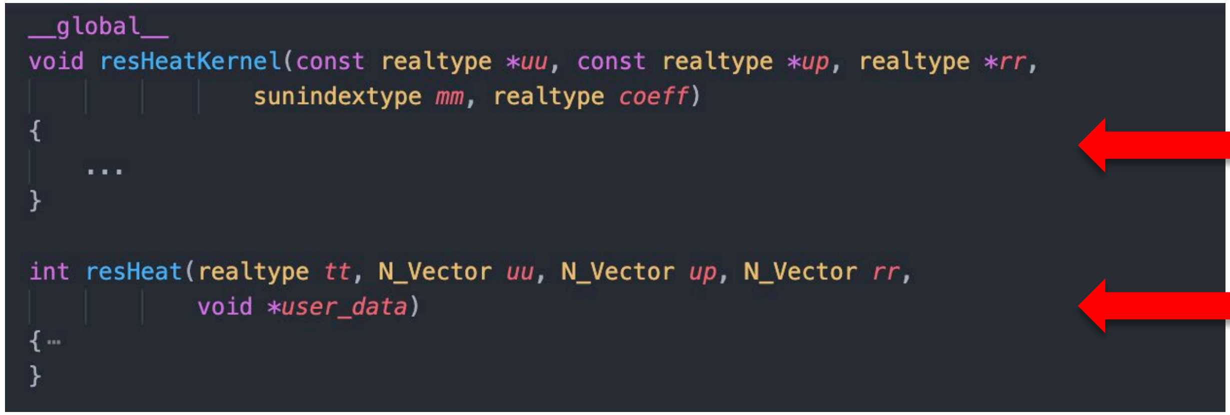
3. Switch to a GPU-capable linear solver (if necessary)

4. Port right-hand side function to the GPU
   - This is critical to avoiding excessive movement of data from the host to the device and vice versa (even when using UVM)

5. Port Jacobian function to the GPU (if necessary)
   - Also critical to avoiding excessive data movement
   - Caveat: SUNDIALS does not provide a GPU-enabled sparse matrix yet, but will very soon

A High-Level Look at a GPU-enabled SUNDIALS example

This example solves a 2D heat equation with IDA and the SPGMR linear solver (so no Jacobian function is needed).

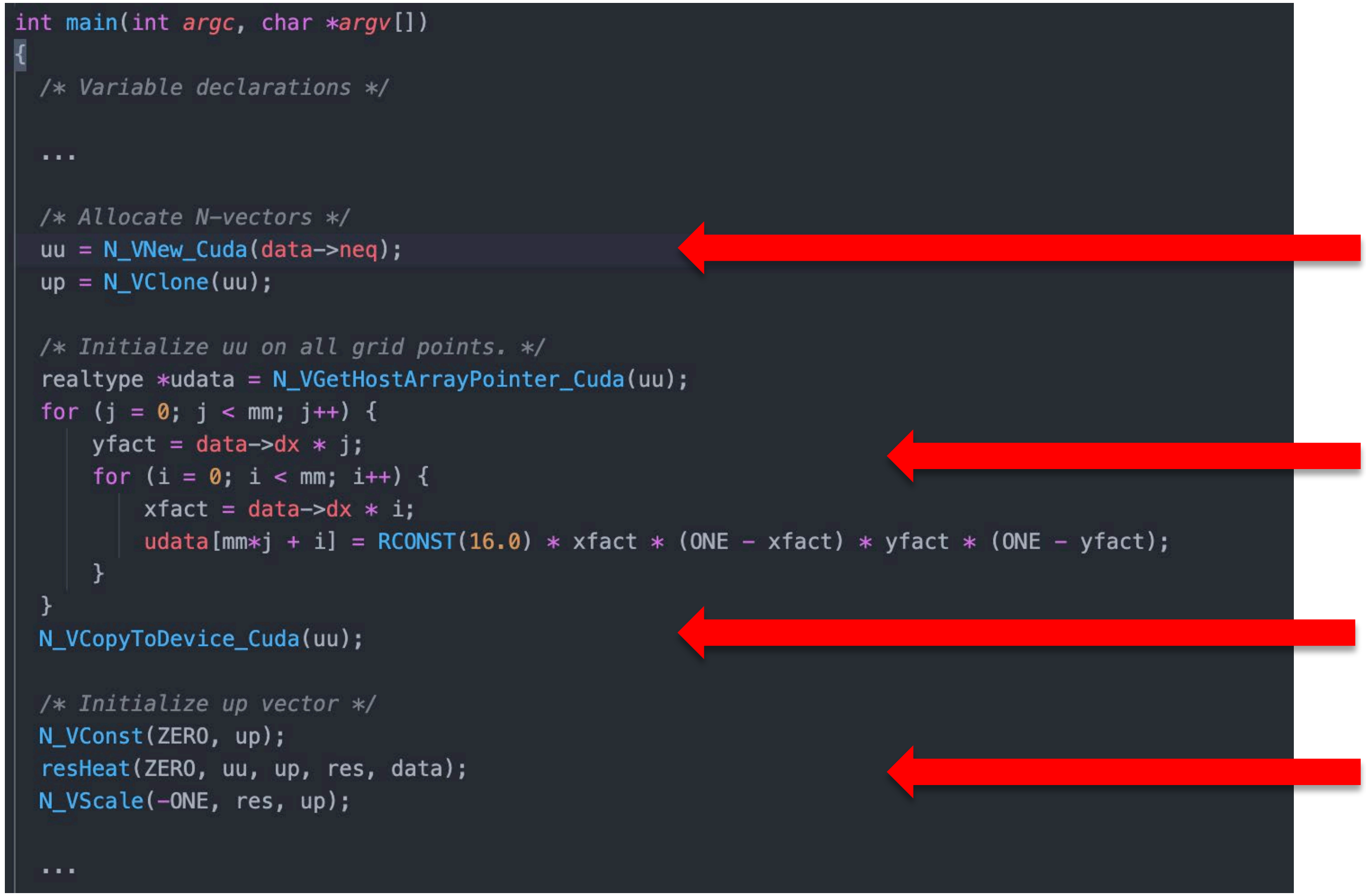


```
__global__
void resHeatKernel(const realtype *uu, const realtype *up, realtype *rr,
                sunindextype mm, realtype coeff)
{
    ...
}

int resHeat(realtype tt, N_Vector uu, N_Vector up, N_Vector rr,
            void *user_data)
{…
}
```

CUDA kernel that does actual residual computation on the GPU

Host function that unpacks data and launches (executes) the residual CUDA kernel. This is what will be provided to IDAInit().

A High-Level Look at a GPU-enabled SUNDIALS example



```
int main(int argc, char *argv[])
{
  /* Variable declarations */

  ...

  /* Allocate N-vectors */
  uu = N_VNew_Cuda(data->neq);
  up = N_VClone(uu);

  /* Initialize uu on all grid points. */
  realtype *udata = N_VGetHostArrayPointer_Cuda(uu);
  for (j = 0; j < mm; j++) {
      yfact = data->dx * j;
      for (i = 0; i < mm; i++) {
          xfact = data->dx * i;
          udata[mm*j + i] = RCONST(16.0) * xfact * (ONE - xfact) * yfact * (ONE - yfact);
      }
  }
  N_VCopyToDevice_Cuda(uu);

  /* Initialize up vector */
  N_VConst(ZERO, up);
  resHeat(ZERO, uu, up, res, data);
  N_VScale(-ONE, res, up);

  ...
```

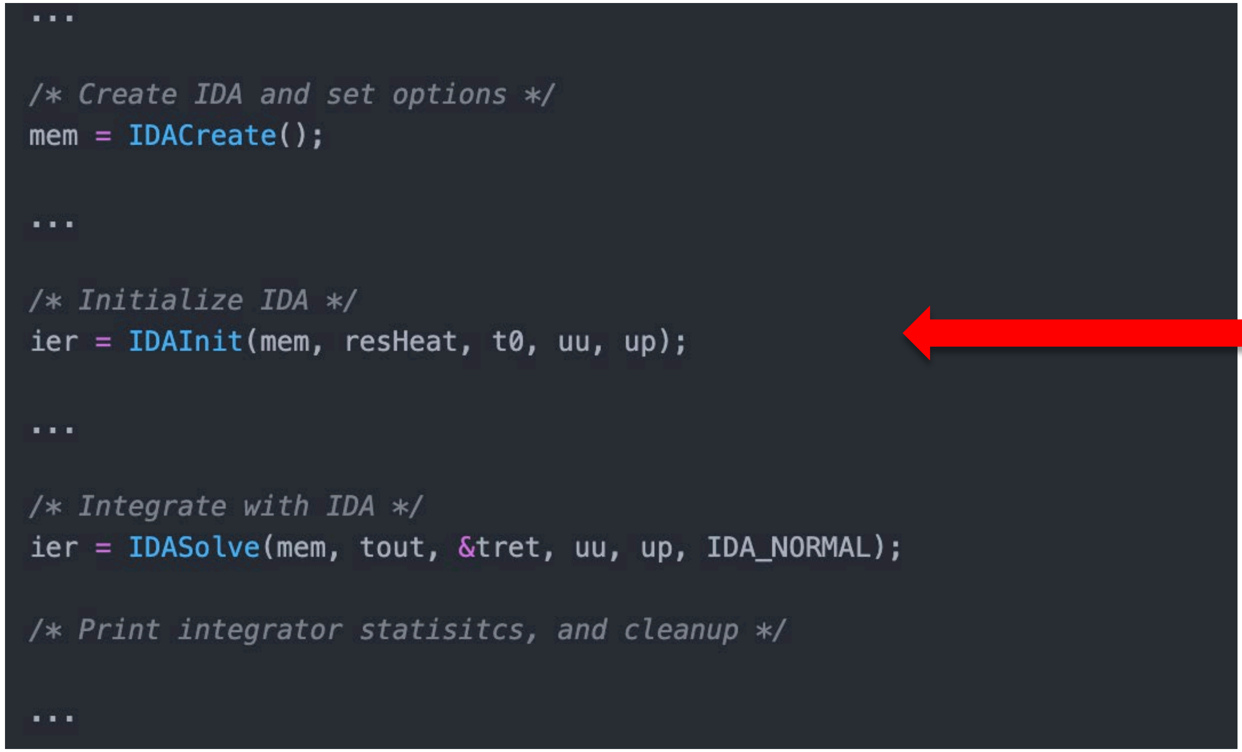
CUDA vector is created to use separate host and device memory

uu is initialized on the host

uu is copied to the device

up is initialized on the device

A High-Level Look at a GPU-enabled SUNDIALS example

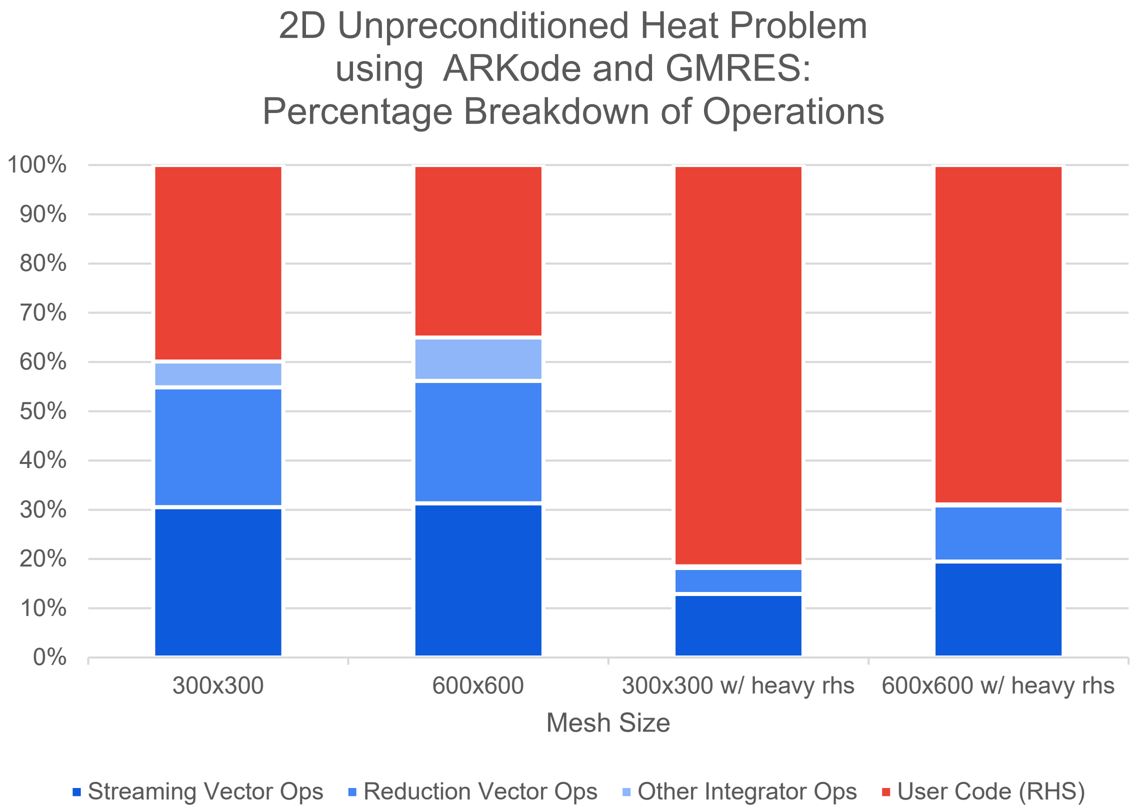


```
...

/* Create IDA and set options */
mem = IDACreate();

...

/* Initialize IDA */
ier = IDAInit(mem, resHeat, t0, uu, up);

...

/* Integrate with IDA */
ier = IDASolve(mem, tout, &tret, uu, up, IDA_NORMAL);

/* Print integrator statisitcs, and cleanup */

...
```

IDA is initialized, the host function, resHeat, *that calls the CUDA kernel* is provided to IDA.

SUNDIALS GPU Performance Considerations

- Under strategy 1, speedup is easier to obtain
  - Long vectors and "heavy" right-hand side functions can overcome overhead
- Under strategy 2, need to group independent subsystems into a larger system
  - Can introduce other problems, like heterogeneity in subsystem stiffness
- We are actively working on new features for increasing performance with GPUs
- We are interested in any results form profiling SUNDIALS+GPUs in your application
- Talk to us about any other performance concerns/questions



"Heavy" RHS includes a 2x cost sleep function in the RHS eval to mimic applications with more work in the RHS function

Tutorial Outline

- Overview of SUNDIALS (Carol Woodward)
- How to use the time integrators (Daniel Reynolds)
- Which nonlinear and linear solvers are available and how to use them (David Gardner)
- Using SUNDIALS on (Pre) Exascale Machines (Cody Balos)
- **Brief: How to download and install SUNDIALS (Cody Balos)**

Acquiring SUNDIALS

- Download the tarball from the SUNDIALS website and build with CMake
  - https://computing.llnl.gov/projects/sundials/sundials-software
  - Latest (v5.1.0) and archived versions, and individual packages (e.g., CVODE) available
  - Most configurable

- Download the tarball from the SUNDIALS GitHub page and build with CMake
  - https://github.com/LLNL/sundials/releases
  - Latest and archived versions available
  - Most configurable

- Install SUNDIALS using Spack
  - “spack install sundials”
  - Latest and recent versions available
  - Highly configurable via spack variants. E.g., “spack install sundials+cuda”.

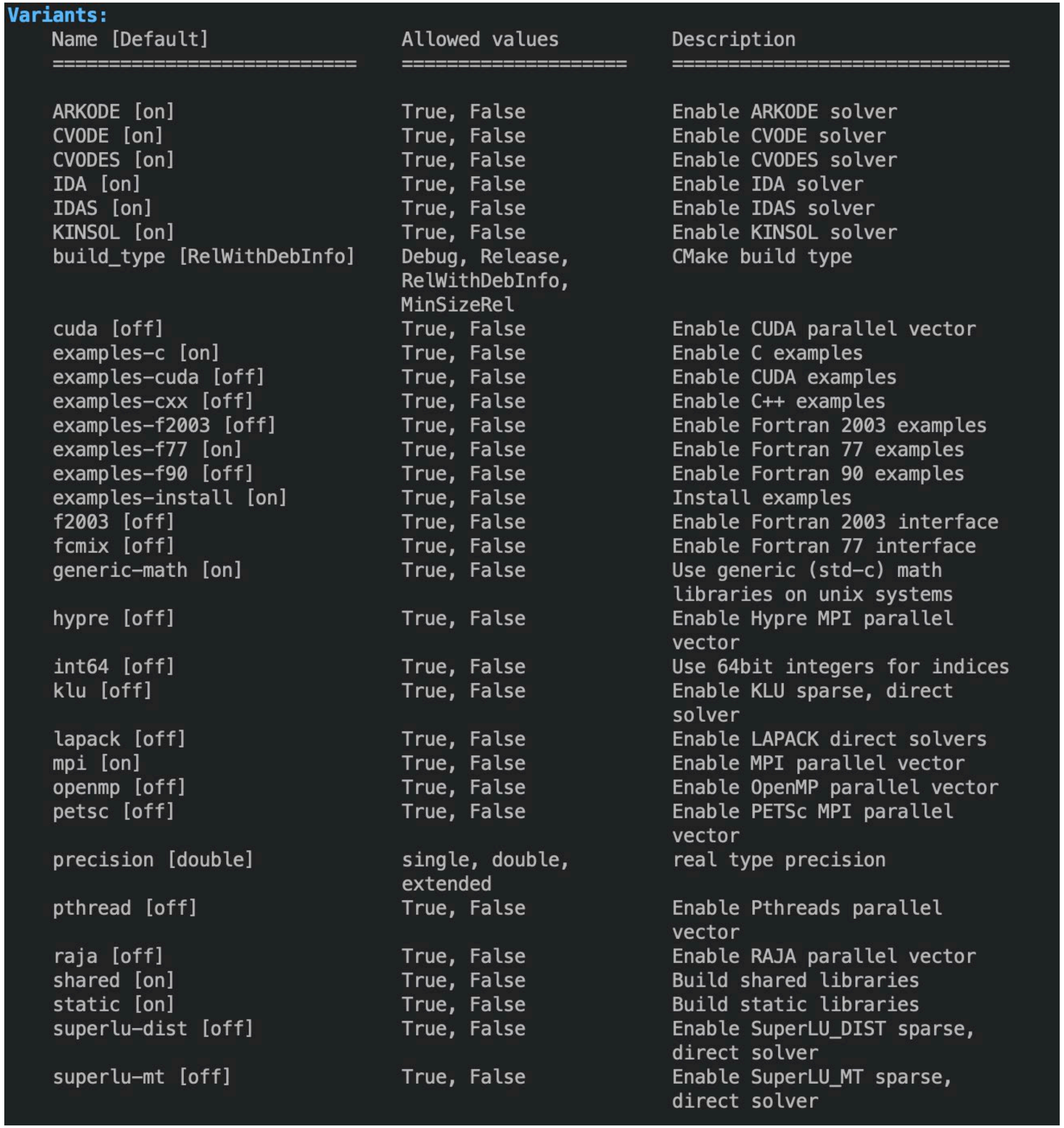
- Install SUNDIALS as part of the xSDK using Spack
  - “spack install xsdk”
  - Will install the xSDK with SUNDIALS v5.0.0

Installing SUNDIALS with Spack

Spack

- Spack (see https://spack.io/) is an easy way to install SUNDIALS
- The SUNDIALS team maintains a Spack package that allows a user to easily install SUNDIALS with one command: `spack install sundials`
- The default configuration installed with `spack install sundials` depends on the environment
- Use the command `spack spec sundials` to see what SUNDIALS options `spack install sundials` will turn on
- The SUNDIALS spack installation is configured through Spack “variants”
- Run `spack info sundials` to see the available “variants” of SUNDIALS available
- SUNDIALS with MPI and *hypre* enabled can be installed with the command:

```
% spack install sundials+mpi+hypre
```



```
Variants:
    Name [Default]                 Allowed values          Description
    ===========================    ====================    ==============================

    ARKODE [on]                    True, False             Enable ARKODE solver
    CVODE [on]                     True, False             Enable CVODE solver
    CVODES [on]                    True, False             Enable CVODES solver
    IDA [on]                       True, False             Enable IDA solver
    IDAS [on]                      True, False             Enable IDAS solver
    KINSOL [on]                    True, False             Enable KINSOL solver
    build_type [RelWithDebInfo]    Debug, Release,         CMake build type
                                   RelWithDebInfo,
                                   MinSizeRel
    cuda [off]                     True, False             Enable CUDA parallel vector
    examples-c [on]                True, False             Enable C examples
    examples-cuda [off]            True, False             Enable CUDA examples
    examples-cxx [off]             True, False             Enable C++ examples
    examples-f2003 [off]           True, False             Enable Fortran 2003 examples
    examples-f77 [on]              True, False             Enable Fortran 77 examples
    examples-f90 [off]             True, False             Enable Fortran 90 examples
    examples-install [on]          True, False             Install examples
    f2003 [off]                    True, False             Enable Fortran 2003 interface
    fcmix [off]                    True, False             Enable Fortran 77 interface
    generic-math [on]              True, False             Use generic (std-c) math
                                                           libraries on unix systems
    hypre [off]                    True, False             Enable Hypre MPI parallel
                                                           vector
    int64 [off]                    True, False             Use 64bit integers for indices
    klu [off]                      True, False             Enable KLU sparse, direct
                                                           solver
    lapack [off]                   True, False             Enable LAPACK direct solvers
    mpi [on]                       True, False             Enable MPI parallel vector
    openmp [off]                   True, False             Enable OpenMP parallel vector
    petsc [off]                    True, False             Enable PETSc MPI parallel
                                                           vector
    precision [double]             single, double,         real type precision
                                   extended
    pthread [off]                  True, False             Enable Pthreads parallel
                                                           vector
    raja [off]                     True, False             Enable RAJA parallel vector
    shared [on]                    True, False             Build shared libraries
    static [on]                    True, False             Build static libraries
    superlu-dist [off]             True, False             Enable SuperLU_DIST sparse,
                                                           direct solver
    superlu-mt [off]               True, False             Enable SuperLU_MT sparse,
                                                           direct solver
```

Installing SUNDIALS via the xSDK

- The Extreme-scale Scientific Software Development Kit (xSDK) provides a foundation for an extensible scientific software ecosystem
- As a member of the xSDK, SUNDIALS is installed with the xSDK Spack package

```
% spack install xsdk
```

- SUNDIALS v5.0.0 (v5.1.0 is the newest) is included in the latest xSDK release - v0.5.0
- See https://xsdk.info for more information about the xSDK and getting it installed

Help Building and Installing SUNDIALS

- An in-depth guide on building and installing SUNDIALS is contained in the root of all SUNDIALS tarballs as INSTALL_GUIDE.pdf
- The guide details how to configure SUNDIALS with CMake as well as every possible SUNDIALS CMake option
- The guide can also be found in Appendix A of the user guide for any SUNDIALS package
- Users can also check the sundials-users email list archive at: http://sundials.2283335.n4.nabble.com
- Users can post queries to the sundials-users email list. For more info see: https://computing.llnl.gov/projects/sundials/support

CASC

Center for Applied
Scientific Computing

Lawrence Livermore
National Laboratory

SMU®

Disclaimer

This document was prepared as an account of work sponsored by an agency of the United States government. Neither the United States government nor Lawrence Livermore National Security, LLC, nor any of their employees makes any warranty, expressed or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States government or Lawrence Livermore National Security, LLC. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States government or Lawrence Livermore National Security, LLC, and shall not be used for advertising or product endorsement purposes.