The Center for Astrophysical Thermonuclear Flashes

Transitioning from FLASH2 to FLASH3

Anshu Dubey

June 23, 2009

An Advanced Simulation & Computing (ASC)
Academic Strategic Alliances Program (ASAP) Center
at The University of Chicago

Most Noticeable Changes

Decentralization of the database

- Introduction of defined constants
  - permanent global constants in “constants.h”
  - compile time global constants in “Flash.h”
  - permanent and compile time local constants in the local unit header files
    - many local constants used to provide indexing into datastructures the way dBaseKeys used to provide in FLASH 2
  - Many maps are generated at compile time to cross reference quantities in different data structures
- Ownership of data by individual units
  - Arbitration on data shared by two or more units
  - Control on modifiability of the data
  - Definition of scope for groups of data
    - Unit scope data module, one per implementation of the unit
    - Subunit scope data module, one per implementation of the subunit
    - All other data modules follow the general FLASH inheritance
      - The directory in which the module exists, and all of its subdirectories have access to the data modules

Example Data on Grid

FLASH2

Get index into database

```
idens = dbaseKeyNumber('dens')
call dbasePutData(idens, ....)
```

FLASH3

use index directly

```
call Grid_putBlkData(DENS_VAR, ....)
```

Example Flash.h File

- The Flash.h file is written by the setup script for your specific application
- Declares variable and flux indices in multidimensional datastructures
- Defines number of dimensions, maximum blocks on a proc and much more ...

```
#define DENS_VAR 1
#define EINT_VAR 2
#define ENER_VAR 3
#define GAMC_VAR 4
#define GAME_VAR 5
#define PRES_VAR 6
#define TEMP_VAR 7
#define VELX_VAR 8
#define VELY_VAR 9
#define VELZ_VAR 10
#define MFRAC_SPEC 11
```

variable indices

```
#define NPROP_VARS 10
#define NSPECIES 1
#define NMASS_SCALARS 0
#define NUNK_VARS (NPROP_VARS + NSPECIES + NMASS_SCALARS)

#define E_FLUX 1
#define EINT_FLUX 2
#define P_FLUX 3
#define RHO_FLUX 4
#define U_FLUX 5
#define UT_FLUX 6
#define UTT_FLUX 7
```

flux indices

```
#define NPROP_FLUX 7
#define NSPECIES_FLUX 1
#define NMASS_SCALARS_FLUX 0
#define NFLUXES (NPROP_FLUX + NSPECIES_FLUX + NMASS_SCALARS_FLUX)

#define NDIM 2
#define MAXBLOCKS 1000
#define TEMP_SCRATCH_GRID_VAR 1
#define NSCRATCH_GRID_VARS 1
```

EOS Local constants

Constants indexing into EosData and Mask

basic EOS_PRES Pressure [p]
basic EOS_DENS Density [rho]
basic EOS_TEMP Temperature [temp]
…
derived EOS_DEZ Derivative of internal energy wrt atomic number/charge
derived EOS_DED Derivative of internal energy wrt density
derived EOS_DST Derivative of entropy wrt temperature
…

#define EOS_BEGIN 1
#define EOS_VARS 9
#define EOS_DERIVS 13
…

#define EOS_DPT 10
#define EOS_DPD 11
#define EOS_DET 12
…

Constants map from eosData to Unk

EOSMAP_PRES Pressure (normally PRES_VAR)
EOSMAP_DENS Density (normally DENS_VAR)
EOSMAP_TEMP Temperature (normally TEMP_VAR)
EOSMAP_GAMC Adiabatic index, Chandrasekhars Gamma1 (normally GAMC_VAR)
…

#define EOSMAP_BEGIN 1
#define EOSMAP_NUM_ROLES 13
…

#define EOSMAP_PRES 1
#define EOSMAP_DENS 2
#define EOSMAP_EINT 3
#define EOSMAP_TEMP 4

….

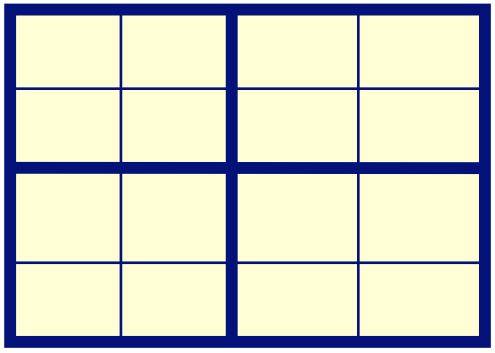
New Useful Features

Mesh Abstraction

- Simulations can choose mesh at configuration time
  - Two major versions of Paramesh, one of Uniform Grid
  - Relaxed the constraint of a fixed blocksize
  - Corresponding addition in IO support
- New physics solvers can be debugged with UG, and used with AMR in production
  - Useful diagnostic tools are easier to develop for UG
- Parallel initialization and handling of obstacles
  - Can start simulation with larger initial domain

Global Integer Index, CornerID

- Motivated by the need for accuracy in guard cells (next slide)
- Has proved to be useful in many other places
  - Tracking motion of non-Paramesh data associated with blocks, for example Lagrangian particles
  - Mapping from Paramesh to Uniform grid for Hybrid Poisson solve algorithm
- How it works :
  - For each dimension, every cell of the mesh is uniquely identified by a combination of its integer index and stride
  - Many cells can have the same integer index, or the same stride, but no two cells will have the same combination



If figure to the left is the whole domain:

- in the lower left half, large 2 cell in bold outline have corner id's of

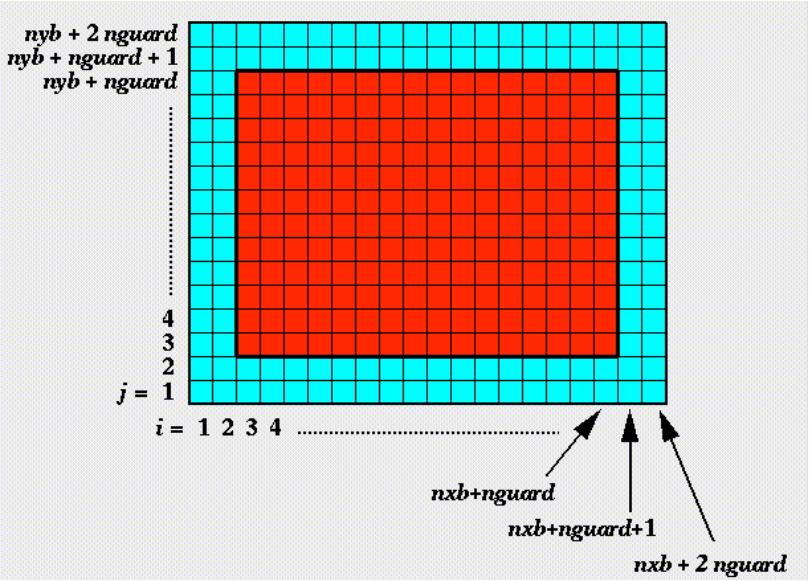
  {<0,2>,<0,2>} and {<2,2>,<0,2>}

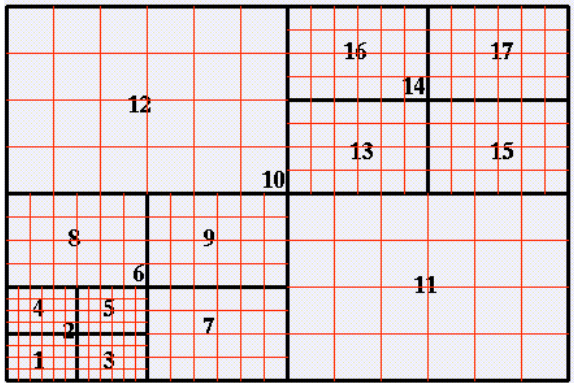
- smaller four cells within the large lower left hand corner cell have corner-ids'

  {<0,1>,<0,1>},{<1,1>,<0,1>},{<0,1>,<1,1>},{<1,1>,<1,1>}

Accuracy and Consistency of Data

- Physical variables in conservative and non conservative forms
  - conversions back and forth for interpolation
  - localizing to fine-coarse boundaries improves performance and reduces drifts
- Quantization level differences in coordinates of a grid point
  - ghost cell of one block is interior for another block
  - Paramesh provides corner bounds in real numbers
  - Need a unique integer ID for each cell
  - Derive coordinates from integer ID





Setup script and Config files

- Variables are grouped together based on their types such as per_volume, per_mass, generic etc
  - automatically determine if there is need to convert form before prolonging
- LIBRARY keyword
  - does not distinguish between external and internal libraries
  - see Library-HOWTO for details
- KERNEL keyword
  - causes all subdirectories to be recursively included
  - makes it possible to include third party software in its original directory organization
- Cross referencing the data structures
  - Examples : DENS_VAR in unk to DENS_PART_PROP in particles
  - Allows for generic implementation of functions such as Particles_updateAttributes
- Setup variables, which allow config files to pick compatible units more easily
  - example : USESETUPVARS parallelIO

```
IF parallelIO
  DEFAULT parallel
ENDIF
```

- PPDEFINE variables allow the source code to know which implementation of a particular unit got included

Unit Architecture

- Formalized from FLASH 2
  - Directory structure for inheritance, multiple alternative implementations, null implementations
  - Runtime environment management through arbitration in shared data
- Resolution of unsolved design issues
  - Introduction of sub-unit concept
  - Definition of the scope of data within a unit
  - Elimination of lateral communication between units
  - Unit test framework
- Use of guidelines and scripts instead of F90 modules to implement OO design
  - Follow the principle of one file one routine
  - use of F90 interfaces
  - Coding standards

API Interface

- Each unit has a well defined API
  - A set of public routines which other units can call
    - Only set of routines visible to other units
    - There is also unit level local API which has routines visible to all the subunits within the unit
  - Use FORTRAN interfaces to help force correct use of API

```
use Grid_interface, ONLY:  Grid_getBlkBoundBox
```

- Examples for Grid unit:
  - `Grid_get/putBlkData`
  - `Grid_getBlkBoundBox`
  - `Grid_updateRefinement`
  - `Grid_moveParticles`
  - `Grid_getNumProcs`
- Each routine should be fully documented with argument and algorithm descriptions, and where possible, examples

Robodocs API Grid_getBlkBoundBox

NAME

```
Grid_getBlkBoundBox
```

SYNOPSIS

```
Grid_getBlkBoundBox(integer(IN)  :: blockId,
                    real(OUT) :: boundBox(2, MDIM))
```

DESCRIPTION

```
Gets the physical domain bounding box of the block identified
by blockId.  For each dimension the left (lower or forward)
physical coordinate of the block edge and the right (upper or back)
physical coordinate of the block edge is returned.  See arguments
below for more detail.
```

ARGUMENTS

```
blockId -local block number

boundBox - returned array holding the boundBox coordinates in
           each dimension

          for readability, in constants.h we define IAXIS = 1, JAXIS = 2, KAXIS = 3

          boundBox(1,IAXIS) = left edge coordinate of block in x direction
          boundBox(2,IAXIS) = right edge coordinate of block in x direction
          boundBox(1,JAXIS) = top edge coordinate of block in y direction
          boundBox(2,JAXIS) = bottom edge coordinate of block in y direction
          boundBox(1,KAXIS) = front edge coordinate of block in z direction
          boundBox(2,KAXIS) = back edge coordinate of block in z direction
```

EXAMPLE

```
In 2 dimensions, if physical coordinates are ...

   _________________(0.5 1.0)
  |                 |
  |                 |
  |                 |
  |                 |
  |                 |
  |                 |
  |                 |
  |_________________ |
(-0.5, 0.0)
```

Moving a FLASH2 module to FLASH3

- Create a ‘UnitName’_data FORTRAN module to store data needed by this unit.
  - Use ‘save’ attribute on all unit scope data
- Initialize data in UnitName_init.
  - This routine eliminates Flash2’s “first call” constructs
  - Do not call RuntimeParameters_get from ANYWHERE else
- Finalize data in UnitName_finalize.
- Make ‘UnitName_interface’ to describe the public routines API.
- Make ‘un_interface’ to describe the API local to the unit
- Most kernels are the same as in FLASH2 and may be arbitrarily messy
- Write good documentation in Robodoc format
- Document runtime parameters in the `Config` file
- When moving code, see the name changes chart and descriptions on the web.
- See “example of a FLASH3 Unit” on web

Example: Sod Shock Tube Setup

FLASH 2 version

```
subroutine init_block(block_no)

  use logfile, ONLY: stamp_logfile

  use dBase, ONLY: k2d, k3d, nxb, nyb, nzb, nguard, &
      ndim, ionmax, &
      dBaseKeyNumber, dBaseSpecies, &
      dBaseGetCoords, dBasePutData, &
      dBasePropertyInteger

  use runtime_parameters, ONLY: get_parm_from_context,
        GLOBAL_PARM_CONTEXT

  implicit none

! compute the maximum length of a vector in each coordinate direction
! (including guardcells)
  integer, parameter :: q = max(nxb+2*nguard, &
                          nyb+2*nguard*k2d, &
                          nzb+2*nguard*k3d)

  integer :: block_no

  integer :: i, j, k, n
  integer :: imax, jmax, kmax

  real, save :: xcos, ycos, zcos

  real :: lposn0, lposn

  real, dimension(q) :: x, y, z, xl, xr

  integer, save :: idens, itemp, ipres, iener, igame, igamc
  integer, save :: ivelx, ively, ivelz, inuc_begin

  real :: rho_zone, velx_zone, vely_zone, velz_zone, pres_zone, &
      ener_zone, ekin_zone

  integer, save :: iXvector, iYvector, iZvector
  integer, save :: iXcoord,  iYcoord,  iZcoord

  integer, save :: iPt
  integer, save :: izn, iznl, iznr

  real, save :: smallp,smallx

  real, save :: gamma

  real, save :: rho_left, rho_right, p_left, p_right,  &
      u_left, u_right, xangle, yangle, posn

  logical, save :: firstCall = .TRUE.
```

FLASH2 Sod contd…

```
if (firstCall) then

    MyPE = dBasePropertyInteger('MyProcessor')
    MasterPE = dBasePropertyInteger('MasterProcessor')

! grab pointers into the database, so we don't need expensive string
        compares
    idens = dBaseKeyNumber('dens')

  ivelx = dBaseKeyNumber('velx')
    ively = dBaseKeyNumber('vely')
    ivelz = dBaseKeyNumber('velz')

    iener = dBaseKeyNumber('ener')
    ipres = dBaseKeyNumber('pres')
    itemp = dBaseKeyNumber('temp')

    igame = dBaseKeyNumber('game')
    igamc = dBaseKeyNumber('gamc')

    inuc_begin = dBaseSpecies(1)

    iXcoord  = dBaseKeyNumber('xCoord')
    iYcoord  = dBaseKeyNumber('yCoord')
    iZcoord  = dBaseKeyNumber('zCoord')
```

```
call get_parm_from_context(GLOBAL_PARM_CONTEXT, 'smallp', smallp)
call get_parm_from_context(GLOBAL_PARM_CONTEXT, 'smallx', smallx)
 call get_parm_from_context(GLOBAL_PARM_CONTEXT, 'gamma', gamma)
call get_parm_from_context(GLOBAL_PARM_CONTEXT, 'rho_left', rho_left)
 call get_parm_from_context(GLOBAL_PARM_CONTEXT, 'rho_right', rho_right)
  call get_parm_from_context(GLOBAL_PARM_CONTEXT, 'p_left', p_left)
  call get_parm_from_context(GLOBAL_PARM_CONTEXT, 'p_right', p_right)

! convert the shock angle paramters
    xangle = xangle * 0.0174532925 ! Convert to radians.
    yangle = yangle * 0.0174532925
    xcos = cos(xangle)

    if (ndim == 1) then
      xcos = 1. ; ycos = 0. ; zcos = 0.
    elseif (ndim == 2) then
      ycos = sqrt(1. - xcos*xcos);  zcos = 0.
    elseif (ndim == 3) then
      ycos = cos(yangle)
      zcos = sqrt( max(0., 1. - xcos*xcos - ycos*ycos) )
    endif

  firstCall = .FALSE.

  endif
```

FLASH3 Version : The data file

```
module Simulation_data

  implicit none

  !! *** Runtime Parameters *** !!

  real, save :: sim_rhoLeft, sim_rhoRight, sim_pLeft, sim_pRight
  real, save :: sim_uLeft, sim_uRight, sim_xAngle, sim_yAngle, sim_posn
  real, save :: sim_gamma, sim_smallP, sim_smallX

  !! *** Variables pertaining to Simulation Setup 'Sod' *** !!
  real, save :: sim_xCos, sim_yCos, sim_zCos
  logical, save :: sim_gCell

end module Simulation_data
```

The Simulation_init File

```
subroutine Simulation_init(myPE)

  use Simulation_data
  use RuntimeParameters_interface, ONLY :
         RuntimeParameters_get

implicit none
#include "Flash.h"

  integer, intent(in) :: myPE

  call RuntimeParameters_get('smallp', sim_smallP)
  call RuntimeParameters_get('smallx', sim_smallX)
 call RuntimeParameters_get('gamma', sim_gamma)
   call RuntimeParameters_get('sim_rhoLeft', sim_rhoLeft)
 call RuntimeParameters_get('sim_rhoRight',
         sim_rhoRight)
 call RuntimeParameters_get('sim_pLeft', sim_pLeft)
call RuntimeParameters_get('sim_pRight', sim_pRight)
  call RuntimeParameters_get('sim_uLeft', sim_uLeft)
 call RuntimeParameters_get('sim_uRight', sim_uRight)
call RuntimeParameters_get('sim_xangle', sim_xAngle)
call RuntimeParameters_get('sim_yangle', sim_yAngle)
  call RuntimeParameters_get('sim_posn', sim_posn)

! convert the shock angle paramters
  sim_xAngle = sim_xAngle * 0.0174532925 ! Convert to radians.
  sim_yAngle = sim_yAngle * 0.0174532925

  sim_xCos = cos(sim_xAngle)

  if (NDIM == 1) then
     sim_xCos = 1.
     sim_yCos = 0.
     sim_zCos = 0.

  elseif (NDIM == 2) then
     sim_yCos = sqrt(1. - sim_xCos*sim_xCos)
     sim_zCos = 0.

  elseif (NDIM == 3) then
     sim_yCos = cos(sim_yAngle)
     sim_zCos = sqrt( max(0., 1. - sim_xCos*sim_xCos -
 sim_yCos*sim_yCos) )
  endif
end subroutine Simulation_init
```

Common Mistakes in FLASH3

- Retain the order of calls in Driver_evolveFlash and Driver_initFlash
- Use the FLASH APIs in general!
  - Don't use Paramesh datastructures directly
  - Don't call private functions of other units
  - Don't use data variables from other modules
- Use the Grid and Geometry APIs!
  - Don't use NXB directly; use Grid_getBlkIndexLimits instead
  - Don't calculate your own grid coordinates; use Grid_getCellsCoords or Grid_getBlkCenterCoords instead
- Use the FLASH define files!
  - Use Flash_mpi.h and FLASH_REAL datatype rather than mpif.h and MPI_DOUBLE_PRECISION or MPI_REAL
- Don't duplicate runtime parameters indiscriminately
  - OK to redefine in Simulation Unit
  - Pay attention to setup script warnings!
- Use Fortran90 features to help
  - Use interfaces
  - Use coding violations script (run every night)