RNPL

Robert L. Marsa and Matthew W. Choptuik

Center for Relativity

The University of Texas at Austin

October 1995

This work is supported by:

Texas Advanced Research Project TARP-085 [R.A. Matzner]

Cray Research Grant [R.A. Matzner]

NSF PHY9318152 (ARPA supplemented) [Grand Challenge Alliance]

(Center for Relativity, University of Texas at Austin)

Getting Started

- Where to get RNPL

  `http://godel.ph.utexas.edu`

  Choose `GC Toolkit`

  Find "RNPL"

- Where to get Manuals

  The **User's Guide** and **Reference Manual** are available in HTML and Postscript via the `GC Toolkit` link on the web.

  Both manuals are included in the distribution (.tex)

- Where to report problems

  `marsa@hoffmann.ph.utexas.edu`

  `matt@infeld.ph.utexas.edu`

RNPL Overview

RNPL is a language for helping researchers to quickly construct finite-difference programs to solve time-dependant systems of partial differential equations. Depending on which language features are used, RNPL can aid the researcher in three main ways:

- Constructing Complete Programs
- Constructing Skeleton Programs
- Re-engineering Existing Programs

The RNPL compiler translates RNPL code into C or FORTRAN which can then be edited or compiled directly, giving the researcher complete control over the final program. Special provisions are made for allowing the use of existing update routines.

Benefits

Most FD programs have the following form:

```
Read (initial) state
for NUM_TIME_STEPS
  for NUM_UPDATES
    (do until convergence)
      U(grid_funcs) → grid_funcs
    (end do)
  end for
  (output)
end for
Write (final) state
```

RNPL can be used to

- Construct the entire program
- Construct everything but `U`

Automatic Features

- Process control, Checkpointing, Interactive I/O
- Parameter management, Memory management
- Parallelization, AMR (Future)
- Ease of Use

Example 1: RNPL constructs `U`

```
parameter float xmin := 0
parameter float xmax := 100
parameter float ymin := 0
parameter float ymax := 100
parameter float zmin := 0
parameter float zmax := 100
parameter float amp := 1.0
parameter float cx := 50.0
parameter float cy := 50
parameter float cz := 50
parameter float sx
parameter float sy
parameter float sz

rect coordinates t,x,y,z
uniform rect grid g1 [1:Nx][1:Ny][1:Nz] {xmin:xmax}{ymin:ymax}{zmin:zmax}
float phi on g1 at -1,0,1

operator D_LF(f,t) := (<1>f[0][0][0] - <-1>f[0][0][0])/(2*dt)
operator D_LF(f,t,t) := (<1>f[0][0][0] - 2*<0>f[0][0][0] + <-1>f[0][0][0])/(dt*dt)
operator D_LF(f,x,x) := (<0>f[1][0][0] - 2*<0>f[0][0][0] + <0>f[-1][0][0])/(dx*dx)
operator D_LF(f,y,y) := (<0>f[0][1][0] - 2*<0>f[0][0][0] + <0>f[0][-1][0])/(dy*dy)
operator D_LF(f,z,z) := (<0>f[0][0][1] - 2*<0>f[0][0][0] + <0>f[0][0][-1])/(dz*dz)

evaluate residual phi {
            [1:Nx][1:Ny][1:1] := D_LF(phi,t);
          [1:Nx][1:Ny][Nz:Nz] := D_LF(phi,t);
            [1:Nx][1:1][1:Nz] := D_LF(phi,t);
          [1:Nx][Ny:Ny][1:Nz] := D_LF(phi,t);
            [1:1][1:Ny][1:Nz] := D_LF(phi,t);
          [Nx:Nx][1:Ny][1:Nz] := D_LF(phi,t);
     [2:Nx-1][2:Ny-1][2:Nz-1] := D_LF(phi,t,t) = D_LF(phi,x,x) +
                                 D_LF(phi,y,y) + D_LF(phi,z,z) }
initialize phi {
  [1:Nx][1:Ny][1:Nz] := amp*exp(-(x-cx)^2/sx^2)*
                            exp(-(y-cy)^2/sy^2)*exp(-(z-cz)^2/sz^2) }

looper iterative
auto update phi
```

Example 2: User constructs `U`

```
parameter float xmin := 0
parameter float xmax := 100
parameter float ymin := 0
parameter float ymax := 100
parameter float zmin := 0
parameter float zmax := 100
parameter float amp := 1.0
parameter float cx := 50.0
parameter float cy := 50
parameter float cz := 50
parameter float sx
parameter float sy
parameter float sz

rect coordinates t,x,y,z
uniform rect grid g1 [1:Nx][1:Ny][1:Nz] {xmin:xmax}{ymin:ymax}{zmin:zmax}
float phi on g1 at -1,0,1

looper standard
evolve3d.inc evolve3d UPDATES phi HEADER phi[phnp1,phn,phnm1],x,y,z,dt

!----------------------------
!  This routine updates the following grid functions
!  phi
!----------------------------
      subroutine evolve3d(phnp1,phn,phnm1,g1_shp,g1_bds,x,y,z,dt)
        implicit none

        include 'globals.inc'

        integer g1_shp(3)
        integer g1_bds(6)
        real*8  phnp1(g1_bds(1):g1_bds(2),g1_bds(3):g1_bds(4),
     &    g1_bds(5):g1_bds(6))
        real*8  phn(g1_bds(1):g1_bds(2),g1_bds(3):g1_bds(4),
     &    g1_bds(5):g1_bds(6))
        real*8  phnm1(g1_bds(1):g1_bds(2),g1_bds(3):g1_bds(4),
     &    g1_bds(5):g1_bds(6))
        real*8  x(*)
        real*8  y(*)
        real*8  z(*)
        real*8  dt

!   contents of evolve3d.inc

        return
      end
```