Bearclaw 2d advection

1 Theory

The conservation law

$$q_t + (uq)_x + (vq)_y = 0,$$

describes advection of the passive scalar $q(t, x, y)$ by the velocity field (u, v).

2 Implementation

Define the Bearclaw `problem` module.

2.1 Global definitions

A module is constructed with global definitions.

```
MODULE problem
  USE NodeInfoDef
  IMPLICIT NONE
  SAVE
  PRIVATE
  PUBLIC setprob,afterrun,qinit,b4step,setaux,src,physflux,problemBC, &
         afterstep,afterfixup,problemIO,problemBadCFL,problemErrFlag
  REAL (KIND=qPrec), DIMENSION(2) :: ubar
  CONTAINS
```

Definition of problem module
Uses Bearclaw Info structure
All variables have to be declared
Save variables between calls
Internal variables cannot be seen
Public entry points

(u, v) is global

2.2 Problem definition: `setprob`

```
    SUBROUTINE setprob
      CHARACTER*60 :: comment
```

Input parameters defining the problem from the `setprob.data` file.

```
      OPEN(unit=7,file='setprob.data',status='old',form='formatted')
      READ(7,*) ubar(1),comment  ! advection x-velocity
      READ(7,*) ubar(2),comment  ! advection y-velocity
```

Open the `setprob.data` file and read (u, v) advection velocity.

```
      CLOSE(7)
    END SUBROUTINE setprob
```

Close the `setprob.data` file.

- setprob.data - (update)

```
1.0e0 Advection velocity along x direction 1.0e0 Advection velocity along y direction
```

2.3 Problem definition: `afterrun`

```
    SUBROUTINE afterrun
    END SUBROUTINE afterrun
```

Actions after run completion

2.4 Problem definition: `problemBC`

```
    SUBROUTINE problemBC(Info)
      TYPE (NodeInfo) :: Info
    END SUBROUTINE problemBC
```

Define problem-specific BCs

2.5 Problem definition: `qinit`

```
SUBROUTINE qinit(Info)
  TYPE (NodeInfo) :: Info
  INTEGER i,j
  REAL (KIND=xPrec) :: x,y
  y=Info%Xlower(2)+Info%dX(2)/2.0
  DO j=1,Info%mX(2)
    x=Info%Xlower(1)+Info%dX(1)/2.0
    DO i=1,Info%mX(1)
      IF ( x>8 .AND. x<24 .AND. y>8 .AND. y<24 ) THEN
        Info%q(i,j,1,1,1)=1.0
      ELSE
        Info%q(i,j,1,1,1)=0.0
      END IF
      IF ( x>10 .AND. x<22 .AND. y>10 .AND. y<22 ) THEN
        Info%q(i,j,1,1,1)=0.0
      END IF
      x=x+Info%dX(1)
    END DO
    y=y+Info%dX(2)
  END DO
END SUBROUTINE qinit
```

Define initial condition
Work on Info grid structure
Local variables
$q(0,x,y)=1$ if $(x,y)\in S-S_1$
else $q(0,x,y)$=0
$S=[8,24]\times[8,24]$
$S_1=[10,22]\times[10,22]$

2.6 Problem definition: `b4step`

```
SUBROUTINE b4step(Info)
  TYPE (NodeInfo) :: Info
END SUBROUTINE b4step
```

Actions before each time step

2.7 Problem definition: `afterstep`

```
SUBROUTINE afterstep(Info)
  TYPE (NodeInfo) :: Info
END SUBROUTINE afterstep
```

Actions after each time step

2.8 Problem definition: `afterfixup`

```
SUBROUTINE afterfixup(Info)
  TYPE (NodeInfo) :: Info
END SUBROUTINE afterfixup
```

Actions after coarse grid update from fine grid values

2.9 Problem definition: `problemIO`

```
SUBROUTINE problemIO(nframe,tnow,IOrequest,Info,qmax,qmin)
  INTEGER :: nframe,IOrequest; REAL (KIND=qPrec) :: tnow
  TYPE (NodeInfo), OPTIONAL :: Info
  REAL (KIND=qPrec), OPTIONAL, DIMENSION(:) :: qmax,qmin
  INTEGER, PARAMETER :: UserBeforeGridIO=-1, UserAfterGridIO=-2
  INTEGER, PARAMETER :: UserIOMinMax=0,UserBeforeOutput=1, &
                        UserAfterOutput=2
END SUBROUTINE problemIO
```

Application-specific I/O

2.10 Problem definition: `setaux`

```
SUBROUTINE setaux(Info)
  TYPE (NodeInfo) :: Info
END SUBROUTINE setaux
```

Define initial auxilliary variables

2.11 Problem definition: `src`

```
SUBROUTINE src(Info)
  TYPE (NodeInfo) :: Info
END SUBROUTINE src
```

Define source term

2.12 Problem definition: `problemBadCFL`

```
SUBROUTINE problemBadCFL(Info)
  TYPE (NodeInfo) :: Info
END SUBROUTINE problemBadCFL
```

Actions if CFL > 1

2.13 Problem definition: `problemErrFlag`

```
SUBROUTINE problemErrFlag(Info,CoarseInfo)
  TYPE (NodeInfo) :: Info        ! Current grid
  TYPE (NodeInfo) :: CoarseInfo  ! Coarsened version of current grid
  Info%ErrorFlags=1
END SUBROUTINE problemErrFlag
```

Application-specific refinement criteria

2.14 Problem definition: `physflux`

2.14.1 Variable declarations

```
SUBROUTINE physflux(ixy,indx,irequest,Info,q,f,s,A)
  INTEGER ixy
  INTEGER indx(MaxDims)


  INTEGER irequest

  TYPE (NodeInfo) :: Info  ! Data associated with this grid
  REAL (KIND=qPrec), POINTER, DIMENSION (:,:) :: q,f,s
  REAL (KIND=qPrec), POINTER, DIMENSION (:,:,:) :: A
  !:physfluxInternalDeclarations:!
  INTEGER mbc,mx,nq,tdir,j
  INTEGER, POINTER, DIMENSION(:) :: iCell,iEdge,iLft,iRgt
  REAL (KIND=qPrec) :: stran,zero=0.
  ! Shorter local names for the wave propagation quantities
  REAL (KIND=qPrec), POINTER, DIMENSION (:,:) :: Apdq,Amdq,speed
  REAL (KIND=qPrec), POINTER, DIMENSION (:,:) :: Asdq,BmAsdq,BpAsdq
  REAL (KIND=qPrec), POINTER, DIMENSION (:,:,:) :: wave
```

Physical fluxes, solve Riemann prob. Direction along which to solve Indices of slice, indx(ixy) has no significance, other indices give position of this 1D slice in index space Request code, definitions in NodeInfoGlobal.f90

2.14.2 Variable definitions

```
  mx=Info%mXnow
  mbc=Info%mbc
  nq=Info%NrVars
  tdir=Info%tdir
  ! Index ranges:
  iEdge=>Info%I1D       ! Interior faces: 2-mbc <= iEdge <= mx+mbc
  iCell=>Info%I1Dcells  ! Cell centers:   1-mbc <= iCell <= mx+mbc
  iLft=>Info%I1Dleft    ! Left faces:     1-mbc <= iLft  <= mx+mbc-1
  iRgt=>Info%I1Dright   ! Right faces:    2-mbc <= iRgt  <= mx+mbc
  Apdq=>Info%Apdq; Amdq=>Info%Amdq
  speed=>Info%speed; wave=>Info%wave
  Asdq=>Info%Asdq; BmAsdq=>Info%BmAsdq; BpAsdq=>Info%BpAsdq
```

Get information from Info structure

2.14.3 Request responses

`SELECT CASE (irequest)`	Identify current request

Initialize

`CASE (Initialize)`	

Finalize

`CASE (Finalize)`	

RequestNormalWaves

`CASE (RequestNormalWaves)`
`Apdq=0.; Amdq=0.; speed=0.; wave=0.`
`wave(iEdge,1,1)=q(iRgt,1)-q(iLft,1)`
`speed(iEdge,1)=ubar(ixy)`
`IF (ubar(ixy)<0.) THEN`
`Amdq(iEdge,1)=speed(iEdge,1)*wave(iEdge,1,1)`
`ELSE`
`Apdq(iEdge,1)=speed(iEdge,1)*wave(iEdge,1,1)`
`END IF`
	Solve 1D Riemann problem along normal direction and compute left,right-going fluctuations

RequestTransverseWaves

`CASE (RequestTransverseWaves)`
`BmAsdq=0.; BpAsdq=0.`
`stran = ubar(tdir)`
`IF (stran<0.) THEN`
`BmAsdq(iEdge,1) = stran * Asdq(iEdge,1)`
`ELSE`
`BpAsdq(iEdge,1) = stran * Asdq(iEdge,1)`
`END IF`
`END SELECT`
	Decompose solution of normal Riemann problem into transverse fluctuations

2.15 Close problem module

`END SUBROUTINE physflux` `END MODULE problem`	

3 Simulations

3.1 Data files

- bear.data - (update)

```
!=====================================================================================
! BEARCLAW bear.data input file. Global parameters valid for all root-level grids.
!=====================================================================================
!:RunFlags:! | Variable | Description
!=========================================================================== F 0 Restart,
Frame Resume from checkpoint data dump F LevelEqSets Solve different equations on grid levels F
LevelMethods Apply different algorithms on grid levels F SaveAtFixedTimes F=maintain CFL, T=save data
at desired times F MaintainAuxArrays Treat aux similarily to q in MPI runs F InitialAMRonly Generate
initial AMR structure and stop F OutputStyleParams Outputstyle line contains additional formatting
!=========================================================================== !:RunParameters:!
!=========================================================================== 1 nRootGrids Number
```

```
of root-level grids 3 MaxLevels Maximum number of grid refinement levels 2 2 2 2 2 CoarsenRatio ... of
child grid to obtain parent spacing 4 MinimumGridPoints ... along one dimension 0 TimeStepMethod 0 fixed
dt, 1 variable dt 0.d0 t0 initial time (if not Restart) 10.00d0 tfinal final time 4 0.5 MaxCFLRetry,
rCFL Try reducing CFL by this ratio this many times 3 OutputStyle 1 AMRCLAW, 2 TECPLOT, 3 HDF, 9
GnuPlot, 11 VTK 10 OutputFrames Number of data checkpoints T T T T T T OutputLevel Level output flag
!================================================================================
```

- grid.data - (update)

```
!================================================================================
! BEARCLAW grid.data input file. Parameters specific to root-level grid.
!================================================================================ !:GridParameters:!
Variable Description !=============================================================================================
2 nDim Grid spatial dimensions 4 MaxLevel Max grid refinement levels for this grid 32 mx Cells in
x direction 32 my Cells in y direction 1 32 mGlobal(1) Global index extents of this grid x 1 32
mGlobal(2) Global index extents of this grid (y-direction) 0.0d0 xlower Left edge of computational
domain 32.0d0 xupper Right edge of computational domain 0.0d0 ylower Bottom edge of computational
domain 32.0d0 yupper Top edge of computational domain 2 mbc Number of ghost cells at each boundary
1 mthbc(1) Left boundary condition code 1 mthbc(2) Right boundary condition code 1 mthbc(3)
Bottom boundary condition code 1 mthbc(4) Top boundary condition code 0.95d0 dtv(1) Initial time
step (constant dt TimeStepMethod=0) 1.0d99 dtv(2) Max allowable time step 1.00d0 cflv(1) Max
allowable Courant number 1.00d0 cflv(2) Desired Courant number 1.0 cflv(3) Time step relaxation
parameter !=============================================================================================
!:MultiphysicsParameters:! - one value if LevelEqSets==F else (>=MaxLevel) values
!================================================================================ ! NrVars = Number
of primary field variables 1 ! nEquationSet = Equation set for these fields 1 ! maux = Number of
auxilliary fields 0 !=============================================================================================
!:GridRefinementParameters:! - (>=MaxLevel) values for each parameter
!================================================================================ ! qTolerance =
Field variable tolerances that trigger refinement 1.0e-3 1.0e-6 1.0e-9 1.0e-3 1.0e-3 1.0e-3 ! xTolerance
= Spatial tolerances that trigger refinement 5.0e-2 5.0e-2 5.0e-2 5.0e-2 5.0e-2 5.0e-2 ! iBuffer
= Size of buffer arround area flagged for refinement 2 2 2 2 2 2 ! DesiredFillRatios= New subgrids
should have this percentage of flagged cells 0.85 0.85 0.85 0.85 0.85 0.85 ! InterpOpt = Interpolation
method used to obtain child data from parent ! (0=minmod, 1=constant, 2=centered, 3=left, 4=right,
5=piecewise) 0 0 0 0 0 0 ! ErrorFlagOpt = Error flag method ! (0=child, 1=parent, 2=apriori 3=user)
0 0 0 0 0 0 !=============================================================================================
!:NumericalSchemeParameters:! one value if LevelMethods==F else (>=MaxLevel) values
!================================================================================ 0 method(1) =
(reserved) 1 method(2) = convergence order 1 method(3) = transverse convergence order 0 method(4) =
verbosity of wavebear output 0 method(5) = source term splitting 0 method(6) = 0 split q differences, 1
split flux differences 0 method(7) = radius of slab around current 1D array of cells
1 mwaves = number of waves in each Riemann solution 4 mthlim(mw) = limiter for each wave (mw=1,
mwaves) !=============================================================================
!:UserRootLevelParameters:!
!================================================================================ ! (none for this
application) !=============================================================================================
```

3.2 Results

```
Shell session inside TeXmacs pid = 22857
Shell] make clean > /dev/null; make outclean > /dev/null; make distclean > /dev/null
Shell] make > /dev/null
Shell] xbear > run.log
Shell] make anim.gif > /dev/null
Shell] animate anim.gif & > /dev/null
  [1] 26245
Shell] make frames
  convert anim.gif frame%05d.png
  [1]+  Done                    animate anim.gif
Shell]
```

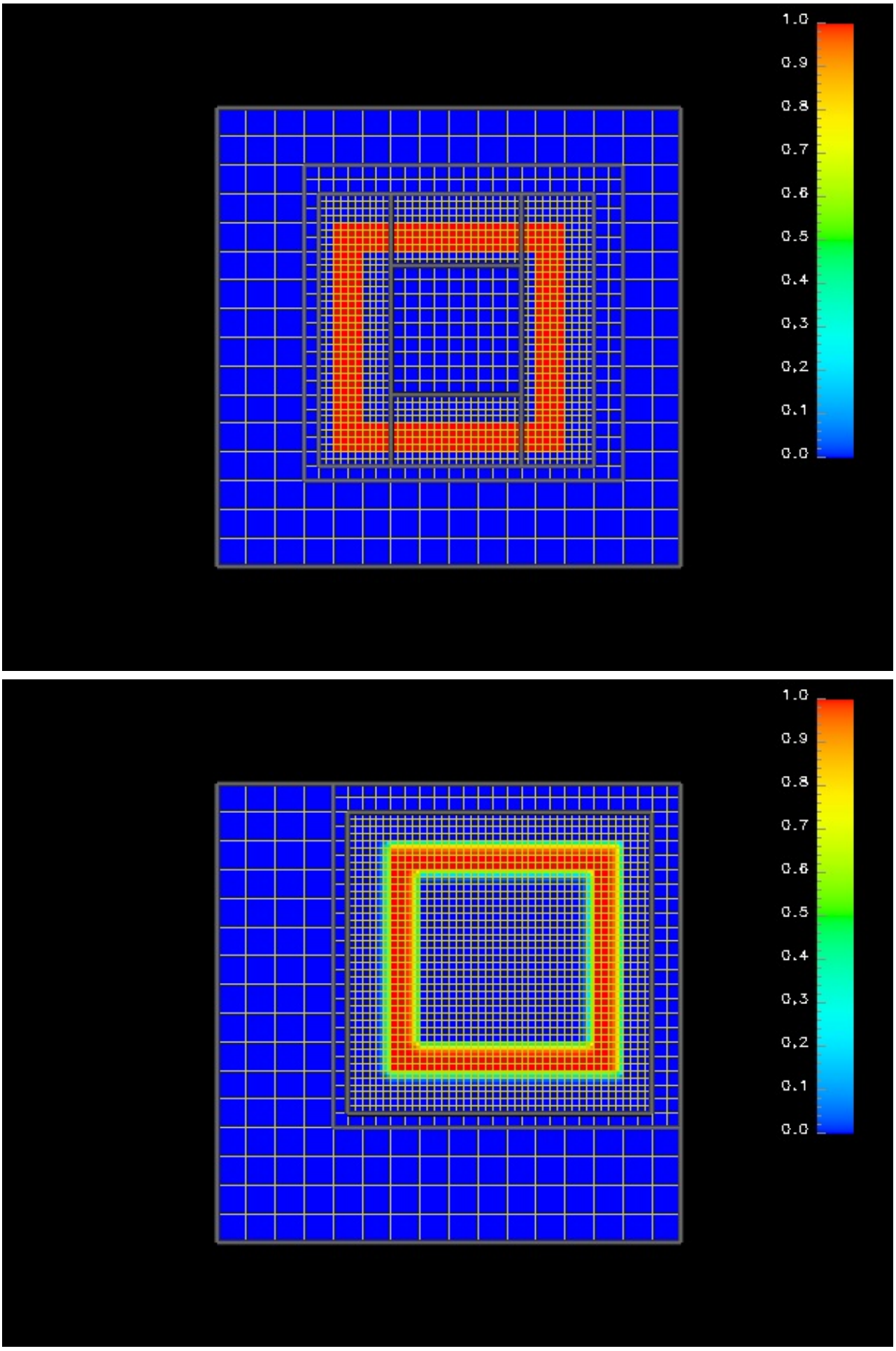

Figure 1. Initial and final frames