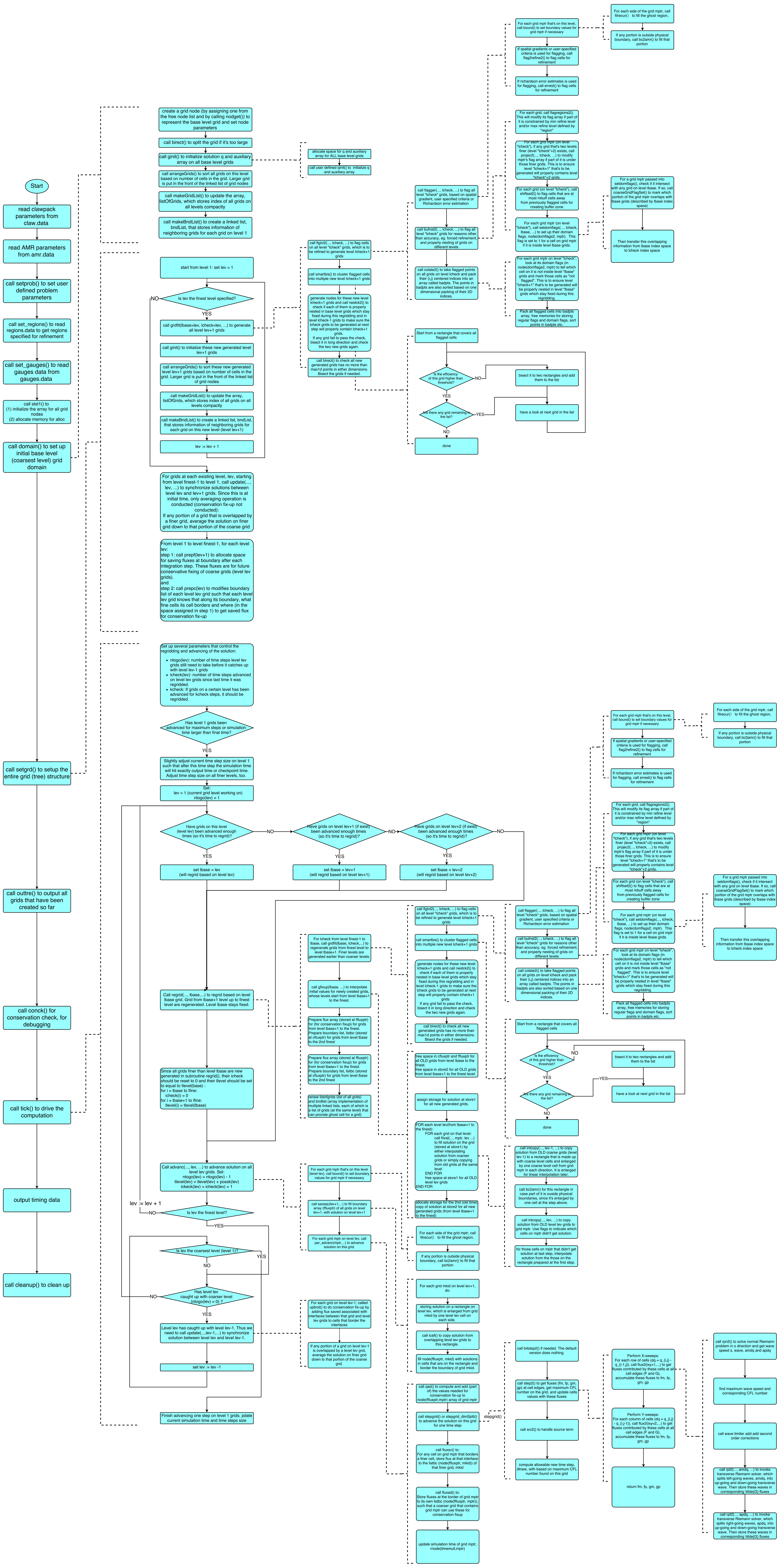
Start
read clawpack parameters from claw.data
read AMR parameters from amr.data
call setprob() to set user defined problem parameters
call set_regions() to read regions.data to get regions specified for refinement
call set_gauges() to read gauges data from gauges.data
call stst1() to (1) initialize the array for all grid nodes (2) allocate memory for alloc
call domain() to set up initial base level (coarsest level) grid domain
call setgrd() to setup the entire grid (tree) structure
call outtre() to output all grids that have been created so far
call conck() for conservation check, for debugging
call tick() to drive the computation
output timing data
call cleanup() to clean up
create a grid node (by assigning one from the free node list and by calling nodget()) to represent the base level grid and set node parameters
call birect() to split the grid if it's too large
call ginit() to initialize solution q and auxiliary array on all base level grids
call arrangeGrids() to sort all grids on this level based on number of cells in the grid. Larger grid is put in the front of the linked list of grid nodes
call makeGridList() to update the array, listOfGrids, which stores index of all grids on all levels compactly
call makeBndList() to create a linked list, bndList, that stores information of neighboring grids for each grid on level 1
allocate space for q and auxiliary array for ALL base level grids
call user defined qinit() to initialize q and auxiliary array
start from level 1: set lev = 1
Is lev the finest level specified?
NO
YES
call grdfit(lbase=lev, lcheck=lev, ...) to generate all level lev+1 grids
call ginit() to initialize these new generated level lev+1 grids
call arrangeGrids() to sort these new generated level lev+1 grids based on number of cells in the grid. Larger grid is put in the front of the linked list of grid nodes
call makeGridList() to update the array, listOfGrids, which stores index of all grids on all levels compactly
call makeBndList() to create a linked list, bndList, that stores information of neighboring grids for each grid on this new level (level lev+1)
lev := lev + 1
For grids at each existing level, lev, starting from level finest-1 to level 1, call update(..., lev, ...) to synchronize solutions between level lev and lev+1 grids. Since this is at initial time, only averaging operation is conducted (conservation fix-up not conducted): If any portion of a grid that is overlapped by a finer grid, average the solution on finer grid down to that portion of the coarse grid
From level 1 to level finest-1, for each level lev: step 1: call prepf(lev+1) to allocate space for saving fluxes at boundary after each integration step. These fluxes are for future conservative fixing of coarse grids (level lev grids). and step 2: call prepc(lev) to modifies boundary list of each level lev grid such that each level lev grid knows that along its boundary, what fine cells its cell borders and where (in the space assigned in step 1) to get saved flux for conservation fix-up
call flglvl2(..., lcheck, ...) to flag cells on all level "lcheck" grids, which is to be refined to generate level lcheck+1 grids
call smartbis() to cluster flagged cells into multiple new level lcheck+1 grids
generate nodes for these new level lcheck+1 grids and call nestck2() to check if each of them is properly nested in base level grids which stay fixed during this regridding and in level lcheck-1 grids to make sure the lcheck grids to be generated at next step will properly contain lcheck+1 grids. If any grid fail to pass the check, bisect it in long direction and check the two new grids again.
call birect() to check all new generated grids has no more than max1d points in either dimensions. Bisect the grids if needed.
call flagger(..., lcheck, ...) to flag all level "lcheck" grids, based on spatial gradient, user specified criteria or Richardson error estimation
call bufnst2(..., lcheck, ...) to flag all level "lcheck" grids for reasons other than accuracy, eg. forced refinement, and properly nesting of grids on different levels
call colate2() to take flagged points on all grids on level lcheck and pack their (i,j) centered indices into an array called badpts. The points in badpts are also sorted based on one dimensional packing of their 2D indices.
Start from a rectangle that covers all flagged cells
Is the efficiency of this grid higher than threshold?
NO
YES
bisect it to two rectangles and add them to the list
Are there any grid remaining in the list?
YES
NO
have a look at next grid in the list
done
For each grid mptr that's on this level, call bound() to set boundary values for grid mptr if necessary
If spatial gradients or user-specified criteria is used for flagging, call flag2refine2() to flag cells for refinement
If richardson error estimates is used for flagging, call errest() to flag cells for refinement
For each side of the grid mptr, call filrecur() to fill the ghost region,
If any portion is outside physical boundary, call bc2amr() to fill that portion
For each grid, call flagregions2(). This will modify its flag array if part of it is constrained by min refine level and/or max refine level defined by "region"
For each grid mptr (on level "lcheck"), if any grid that's two levels finer (level "lcheck"+2) exists, call projec2(..., lcheck, ...) to modify mptr's flag array if part of it is under those finer grids. This is to ensure level "lcheck+1" that's to be generated will properly contains level "lcheck"+2 grids.
For each grid (on level "lcheck"), call shiftset2() to flag cells that are at most mbuff cells away from previously flagged cells for creating buffer zone
For each grid mptr (on level "lcheck"), call setdomflags(..., lcheck, lbase, ...) to set up their domain flags, node(domflags2, mptr). This flag is set to 1 for a cell on grid mptr if it is inside level lbase grids.
For each grid mptr on level "lcheck", look at its domain flags (in node(domflags2, mptr) to tell which cell on it is not inside level "lbase" grids and mark those cells as "not flagged". This is to ensure level "lcheck+1" that's to be generated will be properly nested in level "lbase" grids which stay fixed during this regridding.
Pack all flagged cells into badpts array, free memories for storing regular flags and domain flags, sort points in badpts etc.
For a grid mptr passed into setdomflags(), check if it intersect with any grid on level lbase. If so, call coarseGridFlagSet() to mark which portion of the grid mptr overlaps with lbase grids (described by lbase index space)
Then transfer this overlapping information from lbase index space to lcheck index space
Set up several parameters that control the regridding and advancing of the solution:
• ntogo(lev): number of time steps level lev grids still need to take before it catches up with level lev-1 grids
• icheck(lev): number of time steps advanced on level lev grids since last time it was regridded.
• kcheck: If grids on a certain level has been advanced for kcheck steps, it should be regridded.
Has level 1 grids been advanced for maximum steps or simulation time larger than final time?
YES
Slightly adjust current time step size on level 1 such that after this time step the simulation time will hit exactly output time or checkpoint time. Adjust time step size on all finer levels, too.
Set: lev = 1 (current grid level working on) ntogo(lev) = 1
Have grids on this level (level lev) been advanced enough times (so it's time to regrid)?
NO
YES
set lbase = lev (will regrid based on level lev)
Have grids on level lev+1 (if exist) been advanced enough times (so it's time to regrid)?
NO
YES
set lbase = lev+1 (will regrid based on level lev+1)
Have grids on level lev+2 (if exist) been advanced enough times (so it's time to regrid)?
NO
YES
set lbase = lev+2 (will regrid based on level lev+2)
Call regrid(..., lbase,...) to regrid based on level lbase grid. Grid from lbase+1 level up to finest level are regenerated. Level lbase stays fixed.
Since all grids finer than level lbase are new generated in subroutine regrid(), their icheck should be reset to 0 and their tlevel should be set to equal to tlevel(base) : for i = lbase to lfine: icheck(i) = 0 for i = lbase+1 to lfine: tlevel(i) = tlevel(lbase)
Call advanc(..., lev, ...) to advance solution on all level lev grids. Set: ntogo(lev) = ntogo(lev) - 1 tlevel(lev) = tlevel(lev) + possk(lev) ickeck(lev) = icheck(lev) + 1
lev := lev + 1
NO
Is lev the finest level?
YES
Is lev the coarsest level (level 1)?
YES
NO
Has level lev caught up with coarser level (ntogo(lev) = 0) ?
NO
YES
Level lev has caught up with level lev-1. Thus we need to call update(...,lev-1,...) to synchronize solution between level lev and level lev-1.
set lev := lev -1
Finish advancing one step on level 1 grids. pdate current simulation time and time steps size
For lcheck from level finest-1 to lbase, call grdfit(lbase, lcheck,...) to regenerate grids from finest level to level lbase+1. Finer levels are generated earlier than coarser levels.
call gfixup(lbase, ...) to interpolate initial values for newly created grids, whose levels start from level lbase+1 to the finest.
Prepare flux array (stored at ffluxptr) for (for conservation fixup) for grids from level lbase+1 to the finest. Prepare boundary list, listbc (stored at cfluxptr) for grids from level lbase to the 2nd finest
Prepare flux array (stored at ffluxptr) for (for conservation fixup) for grids from level lbase+1 to the finest. Prepare boundary list, listbc (stored at cfluxptr) for grids from level lbase to the 2nd finest
renew listofgrids (list of all grids) and bndlist (array implementation of multiple linked lists, each of which is a list of grids (at the same level) that can provide ghost cell for a grid)
free space in cfluxptr and ffluxptr for all OLD grids from level lbase to the finest. free space in store2 for all OLD grids from level lbase+1 to the finest level.
assign storage for solution at store1 for all new generated grids.
FOR each level lev(from lbase+1 to the finest): FOR each grid on that level: call filval(..., mptr, lev ...) to fill solution on the grid (stored at store1) by either interpolating solution from coarser grids or simply copying from old grids at the same level END FOR free space at store1 for all OLD level lev grids END FOR
allocate storage for the 2nd (old time) copy of solution at store2 for all new generated grids (from level lbase+1 to the finest)
call intcopy(..., lev-1, ...) to copy solution from OLD coarse grids (level lev-1) to a rectangle that is made up with coarse level cells and enlarged by one coarse level cell from grid mptr in each direction. It is enlarged for linear interpolation later.
call bc2amr() for this rectangle in case part of it is ouside physical boundaries, since it's enlarged by one cell at the step above.
call intcopy(..., lev, ...) to copy solution from OLD level lev grids to grid mptr. Use flags to indicate which cells on mptr didn't get solution.
for those cells on mptr that didn't get solution at last step, interpolate solution from the those on the rectangle prepared at the first step.
For each grid mptr that's on this level (level lev), call bound() to set boundary values for grid mptr if necessary
call saveqc(lev+1,...) to fill boundary array (ffluxptr) of all grids on level lev+1, with solution on level lev+1
For each grid mptr on level lev, call par_advanc(mptr,...) to advance solution on this grid.
For each grid mkid on level lev+1, do:
storing solution on a rectangle on level lev, which is enlarged from grid mkid by one level lev cell on each side.
call icall() to copy solution from overlapping level lev grids to this rectangle.
fill node(ffluxptr, mkid) with solutions in cells that are on the rectangle and border the boundary of grid mkid.
call qad() to compute and add (part of) the values needed for conservation fix-up to node(ffluxptr,mptr) array of grid mptr
call stepgrid() or stepgrid_dimSplit() to advance the solution on this grid for one time step
stepgrid()
call fluxsv() to: For any cell on grid mptr that borders a finer cell, store flux at that interface to the listbc (node(ffluxptr, mkid)) of that finer grid, mkid
call fluxad() to: Store fluxes at the border of grid mptr to its own listbc (node(ffluxptr, mptr)), such that a coarser grid that contains grid mptr can use these for conservation fixup
update simulation time of grid mptr, rnode(timemult,mptr)
For each grid on level lev-1, called upbnd() to do conservation fix-up by adding flux saved associated with interfaces between that grid and level lev grids to cells that border the interfaces
If any portion of a grid on level lev-1 is overlapped by a level lev grid, average the solution on finer grid down to that portion of the coarse grid
call b4step2() if needed. The default version does nothing.
call step2() to get fluxes (fm, fp, gm, gp) at cell edges, get maximum CFL number on the grid, and update cells values with these fluxes
call src2() to handle source term
compute allowable new time step, dtnew, with based on maximum CFL number found on this grid
Perform X-sweeps: For each row of cells (dq = q_{i,j} - q_{i-1,j}), call flux2(ixy=1,...) to get fluxes contributed by these cells at all cell edges (F and G), accumulate these fluxes to fm, fp, gm, gp
Perform Y-sweeps: For each column of cells (dq = q_{i,j} - q_{i,j-1}), call flux2(ixy=2,...) to get fluxes contributed by these cells at all cell edges (F and G), accumulate these fluxes to fm, fp, gm, gp
return fm, fp, gm, gp
call rpn2() to solve normal Riemann problem in x direction and get wave speed s, wave, amdq and apdq
find maximum wave speed and corresponding CFL number
call wave limiter add add second order corrections
call rpt2(..., amdq, ...) to invoke transverse Riemann solver, which splits left-going waves, amdq, into up-going and down-going transverse wave. Then store these waves in corresponding \tilde{G} fluxes
call rpt2(..., apdq, ...) to invoke transverse Riemann solver, which splits right-going waves, apdq, into up-going and down-going transverse wave. Then store these waves in corresponding \tilde{G} fluxes