Pseudo-code for CG routines and algorithms

William D. Henshaw,

Department of Mathematical Sciences,
Rensselaer Polytechnic Institute,
Troy, NY, USA, 12180.

June 26, 2021

Contents

1 Time-stepping pseudo code

1.1 DomainSolver::advanceAdamsPredictorCorrector

Here is an overview of the `DomainSolver::advanceAdamsPredictorCorrector` function (cg/common/src/advancePC.bC) This is an explicit Adams predictor-corrector time stepper.

```
DomainSolver::advanceAdamsPredictorCorrector( ..., numberOfSubSteps, ..)
{
  initialize
  for( int mst=1; mst≤numberOfSubSteps; mst++ )  take time steps
    ifadapt grids
      adaptGrids( ...  )
    ifmove grids
      moveGrids( ...  ); (Sec. 2.1)
      exposedPoints.interpolate(...)

     predictor step:
    getUt( ...  )
    interpolateAndApplyBoundaryConditions( ...  );
    solveForTimeIndependentVariables( ...  )
    correctMovingGrids( ...  )

    for( int correction=0; correction<numberOfCorrections; correction++ )
       corrector step:
      getUt( ...  )
      solveForTimeIndependentVariables( ...  )
      interpolateAndApplyBoundaryConditions( ...  );
      correctMovingGrids( ...  )
}
```

Figure 1: Pseudo-code outline of the advanceAdamsPredictorCorrector function.

2 Moving grid pseudo code

2.1 DomainSolver::moveGrids

Pseudo-code for `DomainSolver::moveGrids` (cg/common/src/move.C)

```
DomainSolver::moveGrids( t1,t2,t3,dt0, cgf1,cgf2,cgf3 )
{
  setInterfacesAtPastTimes( t1,t2,t3,dt0,cgf1,cgf2,cgf3 ); initialize interfaces
  parameters.dbase.get<MovingGrids>("movingGrids").moveGrids(t1,t2,t3,dt0,cgf1,cgf2,cgf3 );
  gridGenerator->updateOverlap( cg, mapInfo );  regenerate the grid with Ogen
}
```

2.2 MovingGrids::moveGrids

Pseudo-code for `MovingGrids::moveGrids` (cg/common/moving/src/MovingGrids.C)

```
MovingGrids::moveGrids( t1,t2,t3,dt0, cgf1,cgf2,cgf3 )
{
  First move the bodies (but not the grids):
  detectCollisions(cgf1);
  rigidBodyMotion( t1,t2,t3,dt0,cgf1,cgf2,cgf3 );
  moveDeformingBodies( t1,t2,t3,dt0,cgf1,cgf2,cgf3 );
  userDefinedMotion( t1,t2,t3,dt0,cgf1,cgf2,cgf3 );
  Apply any matrix motions:  rotate, shift, scale

  getGridVelocity( cgf2,t2 );
  Now move the grids:
  for( grid=0; grid<numberOfBaseGrids; grid++ )
    MatrixTransform & transform = *cgf3.transform[grid];
    if( moveOption(grid)==matrixMotion )
      apply specified rotation and/or shift
      transform.rotate(...)
    else if( moveOption(grid)==rigidBody )
      rotate and shift the rigid body
      transform.rotate(...)
  for( int b=0; b<numberOfDeformingBodies; b++ )
    deformingBodyList[b]->regenerateComponentGrids( newT, cgf3.cg );

  getGridVelocity( cgf3,t3 );
}
```

2.3 MovingGrids::moveDeformingBodies

Pseudo-code for `MovingGrids::moveDeformingBodies` (cg/common/moving/src/MovingGrids.C)

```
MovingGrids::moveDeformingBodies( t1,t2,t3,dt0, cgf1,cgf2,cgf3 )
{
  for( int b=0; b<numberOfDeformingBodies; b++ )
    deformingBodyList[b]->integrate( t1,t2,t3,cgf1,cgf2,cgf3, stress);
}
```

3 DeformingBodyMotion pseudo code

The `DeformingBodyMotion` class handles deforming bodies.

3.1 DeformingBodyMotion::integrate

Pseudo-code for `DeformingBodyMotion::integrate` (cg/common/moving/src/DeformingBodyMotion.C) This function is called by `MovingGrids::movingGrids` to move the deforming body (but not the grid associated with the deforming body).

```
MovingGrids::integrate( t1,t2,t3,dt0, cgf1,cgf2,cgf3, stress )
{
  if( elasticShell )
    advanceElasticShell(t1,t2,t3,cgf1,cgf2,cgf3,stress,option);
  else if( ...  )

  for( int face=0; face<numberOfFaces; face++ )
    if( ...  )
    else if( userDefinedDeformingBodyMotionOption==interfaceDeform )
      The deformed surface is obtained from the boundaryData array:
      RealArray & bd = parameters.getBoundaryData(side,axis,grid,cg[grid]);
      x0 = bd;

}
```

3.2 DeformingBodyMotion::regenerateComponentGrids

Pseudo-code for `DeformingBodyMotion::regenerateComponentGrids` (cg/common/moving/src/DeformingBodyMoti
This function is called by `MovingGrids::movingGrids` (after calling `DeformingBodyMotion:;integrate`) to actually generate the grid associated with the deforming body.

```
DeformingBodyMotion::regenerateComponentGrids( const real newT, CompositeGrid & cg)
{
  for( int face=0; face<numberOfFaces; face++ )
    hyp.generate(); Call the hyperbolic grid generator.
    Save the grid in the GridEvolution list:
    gridEvolution[face]->addGrid(dpm.getDataPoints(),newT);
}
```

3.3 DeformingBodyMotion::correct

Pseudo-code for `DeformingBodyMotion::correct` (cg/common/moving/src/DeformingBodyMotion.C) This function is called by `MovingGrids::correctGrids`.

```
DeformingBodyMotion::correct( t1, t2, GridFunction & cgf1,GridFunction & cgf2 )
{
  This function currently does nothing.
}
```

4 Cgmp

4.1 Driver code 'main' for Cgmp

Pseudo-code for the main driver code for Cgmp (found in cg/mp/src/cgmpMain.C) This function reads in the overset grid, sets up the problem, solves the problem and finishes.

```
main( int argc, char *argv[] )
{
  Overture::start(argc,argv); // initialize Overture and A++/P++
  // Read command line arguments ...
  GenericGraphicsInterface & ps = *Overture::getGraphicsInterface("cgmp",false,argc,argv);

  CompositeGrid cg; // Object to hold the master overset grid for all domains.
  nameOfGridFile = readOrBuildTheGrid(ps, cg, loadBalance, ...); // Get the overset grid.

  Cgmp & mpSolver = *new Cgmp(cg,&ps,show,plotOption);

  mpSolver.setParametersInteractively(); // Setup the problem and PDE solvers.

  mpSolver.solve(); // Time-step the PDE to completion.

  mpSolver.printStatistics();
  Overture::finish();
}
```

4.2 Cgmp::setParametersInteractively

Pseudo-code for `Cgmp::setParametersInteractively` (cg/mp/src/setParametersInteractively.C) This function reads commands to setup the parameters for each DomainSolver (Cgad, Cgins, Cgsm, Cgcns,...). It initializes the list of interfaces and then request Cgmp run-time parameters.

```
Cgmp::setParametersInteractively( bool callSetup )
{
  while(true)
    gi.getAnswer("");
    if( answer.matches("setup" ) ) // Look for command:  setup 'domainName'
      setupDomainSolverParameters( domain,modelNames ); // Setup a domain.
    end
  end

  initializeInterfaces(gfIndex); // Create list of interfaces.

  DomainSolver::setParametersInteractively(callSetup); // Get Cgmp run-time parameters.
}
```

4.3 Cgmp::setupDomainSolverParameters

Pseudo-code for `Cgmp::setupDomainSolverParameters` (cg/mp/src/setParametersInteractively.C) This function builds the PDE solvers for the different domains and sets all the run-time parameters (e.g. coefficient of thermal conductivity, coefficient of viscosity etc.) for each domain solver.

```
Cgmp::setupDomainSolverParameters( int domain, vector<aString> & modelNames )
// modelNames (input):  list of available models (PDE solvers such as Cgins, Cgad, Cgcns, Cgsm)
{
  while(true)
```

```
    gi.getAnswer("");
    if( answer.matches("set solver") ) // Look for command:  set solver 'solverType'

      // Construct the PDE solver for a given domain:
      domainSolver[domain] = buildModel( solverType, cg.domain[domain],...);

    else if( answer.matches("solver parameters") )
      domainSolver[domain]->setParametersInteractively(false); // Set run-time parameters for
one domain.
    end
  end

}
```

4.4 Cgmp::buildModel

Pseudo-code for `Cgmp::buildModel` (cg/mp/src/cgmpMain.C) This function builds a particular PDE solver such as Cgad, Cgins, Cgcns, etc.

```
DomainSolver* Cgmp::buildModel( const aString & modelName, CompositeGrid & cg, ...  )
// modelNames (input):  list of available models (PDE solvers such as Cgins, Cgad, Cgcns, Cgsm)
{
  DomainSolver *solver=NULL;
  if( modelName == "Cgins" )
    solver = new Cgins(cg,ps,show,plotOption);
  else if( modelName == "Cgcns" )
    solver = new Cgcns(cg,ps,show,plotOption);
  else if( modelName == "Cgad" )
    solver = new Cgad(cg,ps,show,plotOption);
  else if( modelName == "Cgsm" )
    solver = new Cgsm(cg,ps,show,plotOption);
  ...
  end
  solver->parameters.dbase.get<DomainSolver*>("multiDomainSolver")=this;
  return solver;
}
```

4.5 Cgmp::initializeInterfaces

Pseudo-code for `Cgmp::initializeInterfaces` (cg/mp/src/assignInterfaceBoundaryConditions.C) This function locates interfaces (by matching grid faces on different domains) and builds a list of information about the interfaces. The values of `interfaceType(side,dir,axis)` can currently be `noInterface`, `heatFluxInterface`, or `tractionInterface`. These are set when assigning domain boundary conditions.

```
Cgmp::initializeInterfaces( std::vector<int> & gfIndex )
{
  ForDomain( d1 )
    interfaceType1 = domainSolver[d1]->parameters.dbase.get<IntegerArray>("interfaceType");
    for( grid1, dir1,side1 )
      if( interfaceType1(side1,dir1,grid1) != Parameters::noInterface ) //T his face is on an
interface
        if( This face matches an existing face of an interface )
          GridList & gridList = interfaceDescriptor.gridListSide1
OR  interfaceDescriptor.gridListSide2;
          // Create a new (matched) face on the interface:
          gridList.push_back(GridFaceDescriptor(d1,grid1,side1,dir1));
        else
          interfaceList.push_back(InterfaceDescriptor()); // Add a new interface to the list.
          InterfaceDescriptor & interface = interfaceList.back();
          // Create a new (unmatched) face on the interface:
          interface.gridListSide1.push_back(GridFaceDescriptor(d1,grid1,side1,dir1));
        end
      end
    end
  end
}
```

4.6 Cgmp::solve

Pseudo-code for `Cgmp::solve` (cg/mp/src/solve.C)

```
Cgmp::solve()
{
  cycleZero(); // Call domain solvers before time-steps start.
  buildRunTimeDialog();

  for( int step=0; step<maximumNumberOfSteps && !finish; )

    if( t≥ nextTimeToPrint )
      printTimeStepInfo(step,t,cpuTime);
      saveShow( gf[current] );
      finish=plot(t, optionIn, tFinal);
      if( finish ) break;
    end

    dtNew = getTimeStep( gf[current] ); // choose time step
    computeNumberOfStepsAndAdjustTheTimeStep(t,tFinal,nextTimeToPrint,numberOfSubSteps,dtNew);

    advance(tFinal); // advance to t=nextTimeToPrint

  end
}
```

4.7 Cgmp::multiDomainAdvance

Pseudo-code for `Cgmp::multiDomainAdvance` (cg/mp/src/multiDomainAdvance.C) This function may call `multiDomainAdvanceNew` or `multiStageAdvance`, depending on the options.

```
Cgmp::multiDomainAdvance( real & t, real & tFinal )
{
  if( multiDomainAlgorithm==MpParameters::stepAllThenMatchMultiDomainAlgorithm )
    // This new algorithm supports AMR:
    return multiDomainAdvanceNew(t,tFinal);
  else if( multiDomainAlgorithm==MpParameters::multiStageAlgorithm )
    // User-defined multi-stage algorithm:
    return multiStageAdvance(t,tFinal);
  end

  if( initialize )
    initializeInterfaceBoundaryConditions( t,dt,gfIndex );
    ForDomain( d ) assignInterfaceRightHandSide( d, t, dt, correct, gfIndex );
    ForDomain( d ) domainSolver[d]->initializeTimeStepping( t,dt );
  end

  // Take some time steps:
  for( int i=0; i<numberOfSubSteps; i++ )
    ForDomain( d )
      domainSolver[d]->startTimeStep( t,dt,...  );
      numberOfRequiredCorrectorSteps=...; gridHasChanged=...;
    end
    if( gridHasChanged )
      initializeInterfaces(gfIndex); initializeInterfaceBoundaryConditions(...);
    end

    for( int correct=0; correct<=numberOfCorrectorSteps; correct++ )
      ForDomain( d )
        assignInterfaceRightHandSide( d, t+dt, dt, correct, gfIndex ); // (Sec. 5.2)
        domainSolver[d]->takeTimeStep( t,dt,correct,advanceOptions[d] );
      end
      if( hasConverged = checkInterfaceForConvergence( ..  )  )  break;
    end

    ForDomain( d )
      domainSolver[d]->endTimeStep( td,dt,advanceOptions[d] );
    end
    t+=dt;
  end
}
```

4.8 Cgmp::multiDomainAdvanceNew

Pseudo-code for `Cgmp::multiDomainAdvanceNew` (cg/mp/src/multiDomainAdvanceNew.bC). This is the new version of the multi-domain advance routine that supports more general time stepping and the use of AMR.

```
Cgmp::multiDomainAdvanceNew( real &t, real & tFinal )
{
  if( initialize )
    initializeInterfaceBoundaryConditions( t,dt,gfIndex );
    ForDomain( d ) assignInterfaceRightHandSide( d, t, dt, correct, gfIndex );
    ForDomain( d ) domainSolver[d]->initializeTimeStepping( t,dt );
  // Take some time steps:
  for( int i=0; i<numberOfSubSteps; i++ )
    ForDomain( d )
      domainSolver[d]->startTimeStep( t,dt,...  );
      numberOfRequiredCorrectorSteps=...; gridHasChanged=...;
    if( gridHasChanged )
      initializeInterfaces(gfIndex); initializeInterfaceBoundaryConditions(...);
    // Get current interface residual and save current interface values :
    getInterfaceResiduals( t, dt, gfIndex, maxResidual, saveInterfaceTimeHistoryValues );

      // Stage I: advance the solution but do not apply BC's:
      ForDomain( d )
        assignInterfaceRightHandSide( d, t+dt, dt, correct, gfIndex );
        domainSolver[d]->takeTimeStep( t,dt,correct,step but no BC's );

      // Stage II: Project interface values part 1:
      interfaceProjection( t+dt, dt, correct, gfIndex,set interface values );

      // Stage III: evaluate the interface conditions and apply the boundary conditions:
      ForDomain( d )
        assignInterfaceRightHandSide( d, t+dt, dt, correct, gfIndex ); //(Sec. 5.2)
        domainSolver[d]->takeTimeStep( t,dt,correct,apply BC's );

      // Stage IV: Project interface values part 2:
      interfaceProjection( t+dt, dt, correct, gfIndex,set interface ghost values );

      if( hasConverged = checkInterfaceForConvergence( ..  )  )  break;

    ForDomain( d ) domainSolver[d]->endTimeStep( td,dt,advanceOptions[d] );
    t+=dt;
}
```

4.9 Cgmp::multiStageAdvance

Pseudo-code for `Cgmp::multiStageAdvance` (cg/mp/src/multiStageAdvance.bC). This is yet a newer version of the multi-domain advance algorithm. This version was developed to deal with the FSI-AMP schemes involving incompressible fluids and elastic bodies. It uses `interfaceCommunicationMode==Parameters::requestInterfaceDataWhenNeeded` so that domain solvers request interface data when they need it. Each multi-domain time-step is separated into **stages**. Associated with each stage are a subset of domains and a set of operations. For example,

Stage 1 : Time-step Cgins domains but do not apply boundary conditions.
Stage 2 : Time-step and apply BC's to Cgsm domains.
Stage 3 : Apply BC's to Cgins domains.

```
Cgmp::multiStageAdvance( real & t, real & tFinal )
{
  if( initialize )
    initializeInterfaceBoundaryConditions( t,dt,gfIndex ); // Choose heatFlux conditions.
    ForDomain( d ) domainSolver[d]->initializeTimeStepping( t,dt );
  end

  // Take some time steps:
  for( int i=0; i<numberOfSubSteps; i++ )
    ForDomain( d )
      // Start the time-step and return required number of correction steps.
      domainSolver[d]->startTimeStep( t,dt,gfIndexCurrent[d],gfIndexNext[d],advanceOptions[d] );
    end

    // Each time-step consists of a predictor and zero or more correction steps.
    for( int correct=0; correct<=numberOfCorrectorSteps; correct++ )

      for( int stage=0; stage<numberOfStages; stage++ ) // Execute each stage
        StageInfo & stageInfo = stageInfoList[stage]; // Holds info on this stage.
        for( domains d involved in this stage )
          // stageInfo.action :  takeStep and/or applyBoundaryConditions
          advanceOptions[d].takeTimeStepOption=stageInfo.action;

          domainSolver[d]->takeTimeStep( t,dt,correct,advanceOptions[d] );

        end
      end
      // Check if the corrections have completed.
      if( relaxCorrectionSteps && correctionIterationsHaveConverged &&
          correct>=minimumNumberOfCorrections ) break;
    end

    ForDomain( d )
      domainSolver[d]->endTimeStep( td,dt,advanceOptions[d] );
    end
    t+=dt;
  end
}
```

4.10 Cgmp::checkInterfaceForConvergence

Pseudo-code for `Cgmp::checkInterfaceForConvergence` (in cg/mp/src/multiDomainAdvance.C) This function checks the residual in the interface equations for convergence of the sub time-step iterations.

```
bool Cgmp::checkInterfaceForConvergence( int correct, int numberOfCorrectorSteps, ...  )
{
  if( check residuals for convergence )

    // Evaluate the max residuals in the conditions at each interface
    // NOTE: the history of interface iterates are saved here:
    getInterfaceResiduals( tNew, dt, gfIndex, maxResidual, saveInterfaceIterateValues );

    // check if the interface iterations have converged:
    interfaceIterationsHaveConverged=true;
    for( int inter=0; inter<interfaceList.size(); inter++ )
      if( correct==0 )
        initialResidual[inter]=maxResidual[inter]; // Save initial residual
      else if( correct==1 )
        firstResidual[inter]=maxResidual[inter]; // Save first residual
      end          interfaceIterationsHaveConverged = interfaceIterationsHaveConverged &&
              maxResidual[inter] < interfaceList[inter].interfaceTolerance;
      if( !interfaceIterationsHaveConverged ) break;
    end

    if( interfaceIterationsHaveConverged && correct >= numberOfRequiredCorrectorSteps )
      Save statistics about interface iterations ...
      return true; Iterations have completed.
    else
      return false;
    end

  end

}
```

4.11 Cgmp::getInterfaceResiduals

Pseudo-code for `Cgmp::getInterfaceResiduals` (in cg/mp/src/assignInterfaceBoundaryConditions.C) This function evaluates the residual in the jump conditions at each interface.

- This function should be cleaned up.
- The interface residuals can probably be more efficiently computed within the domain solvers.

```
int Cgmp::getInterfaceResiduals( real t, real dt, vector<int> & gfIndex, vector<real> &
maxResidual,
    InterfaceValueEnum saveInterfaceValues =doNotSaveInterfaceValues )
{
  for( int inter=0; inter < interfaceList.size(); inter++ ) // Loop over interfaces.
    InterfaceDescriptor & interfaceDescriptor = interfaceList[inter];
    for( int interfaceSide=0; interfaceSide<=1; interfaceSide++ ) // two sides
      for( int face=0; face<gridList.size(); face++ ) // faces on this interface
        if( interfaceType==Parameters::heatFluxInterface )
          // Mixed BC is a0*T + a1*T.n
          info.a[0]=1.; info.a[1]=0.; // eval T
          domainSolver[domain]->interfaceRightHandSide( get ,interfaceDataOptions,info,...  );
          info.a[0]=0.; info.a[1]=ktc; // eval k*T.n
          domainSolver[domain]->interfaceRightHandSide( get,interfaceDataOptions,info, ...);
        else if( interfaceType==Parameters::tractionInterface )
          FINISH ME..
        end

        if( saveInterfaceValues==saveInterfaceTimeHistoryValues ||
            saveInterfaceValues==saveInterfaceIterateValues )
          Save time-history or sub-time-step iterations.
        end

      end// end for face
    end
  end
  // ---- Transfer data to the opposite side of the interface -----
  // ---- and evaluate the jump conditions -----
  for( int interfaceSide=0; interfaceSide<=1; interfaceSide++ )

    interfaceTransfer.transferData( domainSource, domainTarget, ...  )

    if( interfaceType==Parameters::heatFluxInterface )
      // Compute the maximum error in $[T]$ and $[\mathcal{K}T_n]$
    end
  end
}
```

5 Interfaces

5.1 Cgmp::initializeInterfaceBoundaryConditions

Pseudo-code for `Cgmp::initializeInterfaceBoundaryConditions` (cg/mp/src/assignInterfaceBoundaryConditions. This function determines how the boundary conditions on a heat-flux interface should be assigned. For example, for a partitioned Dirichlet-Neumann (DN) approach, which sides is Dirichlet/Neumann depends on the material parameters $\mathcal{K}$ and $\mathcal{D}$.

```
Cgmp::initializeInterfaceBoundaryConditions( real t, real dt, std::vector<int> & gfIndex )
{
  InterfaceList & interfaceList = parameters.dbase.get<InterfaceList>("interfaceList");
  if( interfaceList.size()==0 )
    initializeInterfaces(gfIndex);
  end

  for( int inter=0; inter < interfaceList.size(); inter++ )
    InterfaceDescriptor & interfaceDescriptor = interfaceList[inter];
    for( face ) // Loop over faces on this interface
      if( interfaceType1(side1,dir1,grid1)==Parameters::heatFluxInterface )
        $K_r \overset{\text{def}}{=} (\mathcal{K}_1/\mathcal{K}_2)\sqrt{\mathcal{D}_2/\mathcal{D}_1}$; // See discussion in [?]
        if( solveCoupledEquatons ) // Explicit time-stepping
          // Apply Neumann BC on both sides when solving the coupled interface equations.
        else if( useMixedInterfaceConditions )
          // Choose mixed BC coefficients ...
        else if( $K_r > 1$ ) // For partitioned DN approach
          // Domain 1 is Neumann, domain 2 is Dirichlet.
          gridDescriptor1.interfaceBC=neumannInterface;
          gridDescriptor1.a[0]=0.; gridDescriptor1.a[1]=$\mathcal{K}_1$;
          gridDescriptor2.interfaceBC=dirichletInterface;
          gridDescriptor2.a[0]=1.; gridDescriptor2.a[1]=0.;
        else
          // Domain 1 is Dirichlet, domain 2 is Neumann.
          gridDescriptor1.interfaceBC=dirichletInterface;
          gridDescriptor1.a[0]=1.; gridDescriptor1.a[1]=0.;
          gridDescriptor2.interfaceBC=neumannInterface;
          gridDescriptor2.a[0]=0.; gridDescriptor2.a[1]=$\mathcal{K}_2$;
        end
        // Save the info about the interface condition in the domain solver:  (e.g.  in bcData)
        domainSolver[d1]->setInterfaceBoundaryCondition( gridDescriptor1 );
        domainSolver[d2]->setInterfaceBoundaryCondition( gridDescriptor2 );
      end
    end

  end
}
```

5.2 Cgmp::assignInterfaceRightHandSide

The `Cgmp::assignInterfaceRightHandSide` function is used to get interface values from a source domain and set interface values on a target domain. It is used in the Cgmp::multiDomainAdvance routine 4.7.

Here is `Cgmp::assignInterfaceRightHandSide` (cg/mp/src/assignInterfaceBoundaryConditions.C)

```
int Cgmp::assignInterfaceRightHandSide( int d, real t, real dt, int correct, std::vector<int> &
gfIndex )
// d :  target domain, assign the interface RHS for this domain.
{
  if( interfaceList.size()==0 )
    initializeInterfaces(gfIndex); // Create the list of interfaces.

  for( each interface on domain d )
    InterfaceDescriptor & interfaceDescriptor = interfaceList[inter];
    // Target and source grid functions:
    GridFunction & gfTarget = domainSolver[domainTarget]->gf[gfIndex[domainTarget]];
    GridFunction & gfSource = domainSolver[domainSource]->gf[gfIndex[domainSource]];

    // Get source data:
    for( int face=0; face<gridListSource.size(); face++ )
      domainSolver[domainSource]->interfaceRightHandSide( getInterfaceRightHandSide, ...  );

    // Transfer the source arrays to the target arrays:
    interfaceTransfer.transferData(...  );

    // Adjust the target data before assigning:
    for( int face=0; face<gridListTarget.size(); face++ )
      // Extrapolate the initial guess.
      // Under-relaxed iteration.

    // Assign the target data:
    for( int face=0; face<gridListTarget.size(); face++ )
      domainSolver[domainTarget]->interfaceRightHandSide( setInterfaceRightHandSide,...);

}
```

5.3 DomainSolver::interfaceRightHandSide

The `DomainSolver::interfaceRightHandSide` function is used to get or set interface values. Each DomainSolver (cgad, cgcns, cgins, cgsm,...) has a version of this routine. The generic version appears in cg/common/src/interfaceBoundaryConditions.C.

Here is `Cgcns::interfaceRightHandSide` (cg/cns/src/interface.bC)

```
Cgcns::interfaceRightHandSide( InterfaceOptionsEnum option, int interfaceDataOptions,
      GridFaceDescriptor & info, GridFaceDescriptor & gfd, int gfIndex, real t )
{
  RealArray & bd = parameters.getBoundaryData(side,axis,grid,mg); // Interface data on this
domain.
  RealArray & f = *info.u; // Interface data from another domain.

  if( interfaceType(side,axis,grid)==Parameters::heatFluxInterface )
    // Set RHS for a heatFlux interface
    if( option==setInterfaceRightHandSide )
      bd(I1,I2,I3,tc)=f(I1,I2,I3,tc); // Set T (using values from another domain).
    else if( option==getInterfaceRightHandSide )
      // Evaluate $a_0 T + a_1 T_n$ (to send to another domain).
      f(I1,I2,I3,tc) = a[0]*uLocal(I1,I2,I3,tc) + a[1]*( normal(I1,I2,I2,0)*ux + ...  );
    end

  else if( interfaceType(side,axis,grid)==Parameters::tractionInterface )
    // Set RHS for a traction interface.
    if( option==setInterfaceRightHandSide )
      bd(I1,I2,I3,V)=f(I1,I2,I3,V); // Set positions of the interface.
    else if( option==getInterfaceRightHandSide )
      parameters.getNormalForce( gf[gfIndex].u,traction,ipar,rpar );
      f(I1,I2,I3,V)=traction(I1,I2,I3,D);

      // Optionally save a time history of some quantities.
      InterfaceDataHistory & idh = gfd.interfaceDataHistory; // Holds interface history.
      if( interfaceDataOptions & Parameters::tractionRateInterfaceData )
        RealArray & f0 = idh.interfaceDataList[prev].f;
        f(I1,I2,I3,Vt)= (f(I1,I2,I3,V) - f0(I1,I2,I3,V))/dt; // Time derivative of the traction.
      end
    end
  end
}
```