# DGSWEM的代码结构

## 主程序：dgswe.F90

主程序主要调用以下子程序：

CALL message\_init() !初始化MPI，确定rank

CALL read\_input(myrank,dirname) ! 读取输入文件（dgswe.inp）

CALL sizes() ! 计算DoF编号、直/曲边单元的节点号

CALL read\_grid() ! 读取网格文件

CALL read\_forcing() ! 读取驱动力

CALL connect() ! 搜索边的连接关系

CALL area\_qpts() ! 获得面积积分的积分点

CALL edge\_qpts() ! 在边的积分点处，计算基函数值

CALL area\_basis() ! 在面积积分点处，计算基函数和导数值

CALL edge\_basis() ! 在边的积分点处，计算基函数和导数值

CALL element\_data() ! 计算单元的面积、边长、边的法向即地形的导数等

CALL initial() ! 计算初始条件

CALL interp\_forcing() ! 边界驱动力插值到边的积分点上去

CALL metis2(npart) ! 将计算域的单元/边分解为npart块

CALL edge\_partition2() ! 区域分解，准备单元/边的分块

! 读取ADCIRC前处理(ADCPREP)输出的信息文件

CALL read\_message\_files() ! 读取local-to-global单元和信息传递文件(???)

CALL message\_setup() ! 设置send/receive缓冲区和边的数据结构

CALL communication\_setup() ! 初始化MPI持续的信息调用

! 输出初始的数值解与测点的值

CALL output\_solution(.true.)

CALL output\_stations(.true.)

! 开始时间步迭代

DO it = 1,tstep

CALL rk() ! 调用Runge-Kutta时间推进：核心代码

! 按设置的输出频率，输出数值解与测点的值

CALL output\_solution

CALL output\_stations

END DO

CALL close\_output()

CALL finish(myrank)

## 子程序messenger2.F90

MODULE messenger2

SUBROUTINE directory\_name()

SUBROUTINE message\_init()

SUBROUTINE read\_message\_files()

SUBROUTINE message\_setup()

SUBROUTINE communication\_setup()

SUBROUTINE message\_recieve()

SUBROUTINE message\_send\_ldg()

SUBROUTINE end\_time(t\_start,nproc)

END MODULE messenger2

## 子程序read\_input()

在read\_input.F90, 包含：

MODULE read\_dginp

SUBROUTINE read\_input(myrank,dirname)

SUBROUTINE read\_fixed\_dginp(myrank,dirname)

SUBROUTINE read\_keyword\_dginp(myrank,dirname)

SUBROUTINE check\_errors(myrank,opt\_read)

SUBROUTINE dginp\_setup(myrank)

SUBROUTINE write\_local(pe)

SUBROUTINE write\_input(file\_unit)

SUBROUTINE write\_file\_SHAs(file\_unit,dirname)

END MODULE read\_dginp

## 子程序sizes()

在allocation.F90

MODULE allocation

SUBROUTINE sizes() !计算DoF编号、直/曲边单元的节点号

SUBROUTINE alloc\_grid\_arrays(stage) ! 计算网格相关的数组

SUBROUTINE alloc\_forcing\_arrays(stage) ! 驱动力数组

SUBROUTINE alloc\_connect\_arrays(stage) ! 连接关系数组

SUBROUTINE alloc\_qpt\_arrays(stage) ! 积分点数组

SUBROUTINE alloc\_basis\_arrays() ! 基函数数组

SUBROUTINE alloc\_trans\_arrays() ! 转换数组

SUBROUTINE alloc\_sol\_arrays() ！ 数值解数组

SUBROUTINE dealloc\_init\_arrays()

SUBROUTINE alloc\_ptr\_arrays() ! 观测点的数组

SUBROUTINE alloc\_blk\_arrays() ! 分区block的数组

END MODULE allocation

SUBROUTINE sizes()

ndof(), np, nnds, order

END

## 子程序read\_grid()

CALL read\_header(myrank,grid\_file,grid\_name,ne,nn)

CALL read\_coords(nn,xy,depth,h0)

CALL cpp\_transformation(coord\_sys,r\_earth,slam0,sphi0,nn,xy)

CALL read\_connectivity(ne,ect,el\_type)

CALL init\_element\_coordinates(ne,ctp,el\_type,nverts,xy,ect,elxy)

CALL read\_open\_boundaries(nope,neta,obseg,obnds)

CALL read\_flow\_boundaries(nbou,nvel,fbseg,fbnds)

CALL read\_bathy\_file(myrank,bathy\_file,hbp,ne,el\_type,nverts,depth,ect,elhb,hb\_file\_exists)

CALL read\_curve\_file(myrank,curve\_file,ctp,nbou,xy,bndxy,cb\_file\_exists)

IF (myrank == 0) THEN

CALL print\_grid\_info(grid\_file,grid\_name,ne,nn)

ENDIF

CALL grid\_size(ne,el\_type,ect,xy,el\_size)

cfl = 1d0

u = 0d0

CALL courant(p,ne,u,cfl,el\_type,nverts,nnds,elhb,el\_size)

## 子程序connect.F90

确定边的连接关系

## 子程序area\_qpts\_mod.F90

USE edge\_qpts\_mod, ONLY: gauss\_qpts ! 使用edge\_qpts\_mod.F90

！ 获得面积积分的积分点

SUBROUTINE area\_qpts(myrank,p,ctp,nel\_type,nqpta,mnqpta,wpta,qpta)

area\_qpts() 调用tri\_cubature和quad\_cubature（三角形和四边形单元的积分点）

SUBROUTINE tri\_cubature(p,nqpta,qpta) 直接对数组赋值。

SUBROUTINE quad\_cubature(p,nqpta,qpta)

quad\_cubature 调用 gauss\_qpts(p,npt,w,r)完成积分点常数的赋值。

## 子程序edge\_qpts\_mod.F90

SUBROUTINE edge\_qpts(myrank,p,ctp,nel\_type,nqpte,mnqpte,wpte,qpte)

edge\_qpts也调用

SUBROUTINE gauss\_qpts(p,nqpte,wpte,qpte)

## 子程序area\_basis和edge\_basis

在basis\_eval.F90, 包含：

SUBROUTINE area\_basis()

SUBROUTINE edge\_basis()

两个子程序都要使用：USE basis, ONLY: element\_basis

## 子程序element\_data.F90

计算单元的一系列函数

CALL curvilinear()

CALL shape\_functions\_area\_qpts()

CALL area\_transformation()

CALL normals()

CALL shape\_functions\_edge\_qpts()

CALL edge\_transformation()

CALL bathymetry\_interp\_area\_qpts()

CALL bathymetry\_interp\_edge\_qpts()

CALL bathymetry\_nodal2modal(hbp,mnnds,ne,el\_type,elhb,hbm)

还有do i=1,ned do i=1,nel

## 子程序initial.F90

! 设置初始条件

Hinit = 0d0

Qxinit(:,:) = 0d0

Qyinit(:,:) = 0d0

Zinit(:,:) = 0d0

## 子程序interp\_forcing.F90

! 首先，将度转换为弧度，将频率转换为周期

! 将输入的边界条件，插值到边界上边的积分点。

## 子程序metis2.F90和edge\_partition2.F90

metis2.F90基于CALL METIS\_PartGraphKway对非结构网格区域分解。

nparts == 1即为串行运行

DO el = 1,ne

part(el) = 1

ENDDO

SUBROUTINE edge\_partition2()

# 核心子程序rk()

龙格库塔的时间推进格式中，调用很多子程序。

SELECT CASE (rk\_type)

CASE(11) ! call swap(); call forward\_euler(); call nan\_check()

! 在forward\_euler()中实施第1阶段的RK积分，相当于1阶Euler积分

CASE(22) ! Euler + call rhs2()

! 在CASE(11)基础上，在执行第2阶段的RK积分，相当于2阶RK

CASE(33) ! 3-order RK

! 在CASE(22)基础上，再计算rhs2与第3阶段的RK积分，相当于3阶RK

CASE(45) ! RK(4,5)

! do stg=1, 5 实施5个阶段的RK积分，相当于4阶RK

END SELECT

## 子程序swap()和nan\_check()

在rk.f90

swap()保存旧时刻计算值

nan\_check()检查是否有NAN的计算值

## 子程序forward\_euler()

在rk.f90中

! 计算右手边项

CALL rhs2()

! First RK stage （RK的第1阶段）

DO blk = 1,npart+1

DO et = 1,nel\_type

IF (npartet(et,blk) > 0) THEN

## 子程序rhs2.F90

CALL interior\_edge\_eval() ! 内部"边"的计算

**if(esl>esl\_sol) then ! 执行LDG计算**

CALL area\_integration\_ldg ! LGD变量

CALL interior\_edge\_eval\_ldg\_Q

CALL interior\_edge\_nflux\_ldg

CALL interior\_edge\_nflux\_ldg

CALL boundary\_edge\_land\_ldg()

CALL boundary\_edge\_flow\_ldg()

CALL boundary\_edge\_elev\_ldg()

CALL edge\_integration\_ldg()

CALL linear\_solve\_ldg()

CALL interior\_edge\_eval\_ldg\_E()

**endif**

! 面积分

IF (esl > esl\_tol) THEN

CALL area\_integration(et,elblk(1,blk,et),elblk(2,blk,et),ndof(et),nqpta(et))

ELSE

CALL area\_integration\_no\_ldg(et,elblk(1,blk,et),elblk(2,blk,et),ndof(et),nqpta(et))

ENDIF

! 内部"边"计算和数值通量

CALL interior\_edge\_eval()

if(esl>esl\_sol) CALL interior\_edge\_eval\_ldg\_E()

CALL interior\_edge\_nflux()

! 边界上的边数值通量

CALL boundary\_edge\_land()

CALL boundary\_edge\_flow()

CALL boundary\_edge\_elev()

! 边的积分

CALL edge\_integration()

CALL linear\_solve()

## 子程序linear\_solve.F90

不是Ax=b线性求解器，是求解RHS的DG法。

! 选择对三角形和四边形单元求解

SUBROUTINE linear\_solve(et,sel,eel,ndof)

SUBROUTINE linear\_solve\_ldg(et,sel,eel,ndof)