Parallel Hierarchical Grid

崔 涛

tcui@lsec.cc.ac.cn

科学与工程计算国家重点实验室
中科院计算数学与科学工程计算研究所

Outline I

Agenda

Parallel Hierarchical Grid

PHG（Parallel Hierarchical Grid）是我们自主设计的一个基于网格二分细化适合大规模分布式存储并行计算机的三维并行自适应有限元软件平台。

- 支持自适应有限元算法研究
  - h-自适应、p-自适应、hp-自适应
- 面向协调四面体网格、单元二分网格局部加密和放粗
- 提供并行自适应有限元程序开发平台
  - 屏蔽并行实现细节
  - 动态负载平衡
  - 高可扩展性
- 软件著作权证书登记号：2006SRBJ1700
- 主页: http://lsec.cc.ac.cn/phg

有限元基础 I

- 变分原理: 把一个力学问题（或其他学科的问题）用变分法化为求泛函极值（或驻值）的问题，就称为该物理问题（或其他学科的问物理题）的变分原理。

$$J(u) = min_{v \in V} J(v),$$

其中

$$J(v) = \frac{1}{2} A(v, v) - F(v)$$

变分原理在物理学中尤其是在力学中有广泛应用，如著名的虚功原理、最小位能原理等。

有限元基础 II

- 有限维子空间：$S_h = span\{\phi_1, \phi_2, \cdots, \phi_N\}$
- 变分近似法：
  - Ritz方法: 求$u_h = \sum_{i=1}^{N} c_i\phi_i \in S_N$, 满足:

$$J(u_h) = min_{v \in S_N} J(v)$$

线性方程组：

$$\sum_{j=1}^{N} A(\phi_i, \phi_j)c_j = F(\phi_i), \quad (i = 1, 2, \cdots, N).$$

  - Galerkin方法：求$u_h = \sum_{i=1}^{N} c_i\phi_i \in S_N$, 满足:

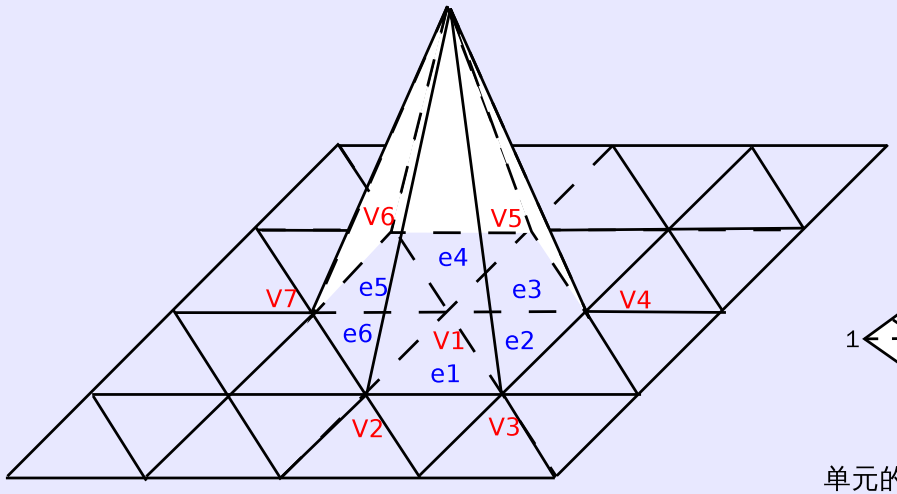
$$A(u_N, v) = F(v) \forall v \in S_N.$$

线性方程组：

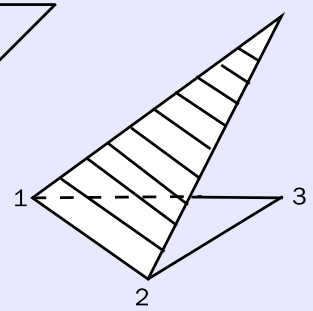
$$\sum_{j=1}^{N} A(\phi_j, \phi_i)c_j = F(\phi_i), \quad (i = 1, 2, \cdots, N).$$

有限元基础 III

- 有限元空间构造
  - 网格剖分（线段、三角形、四面体等）:
    - 网格单元之间没有重叠的内部。
    - 每一个单元的顶点(边)或者是边界点（边），或者是相邻单元的公共顶点（边）。
    - 如果方程的系数有间断，则应该用折线(面)逼近间断线（面）
  - 基函数、形状函数:



有限元空间的基函数

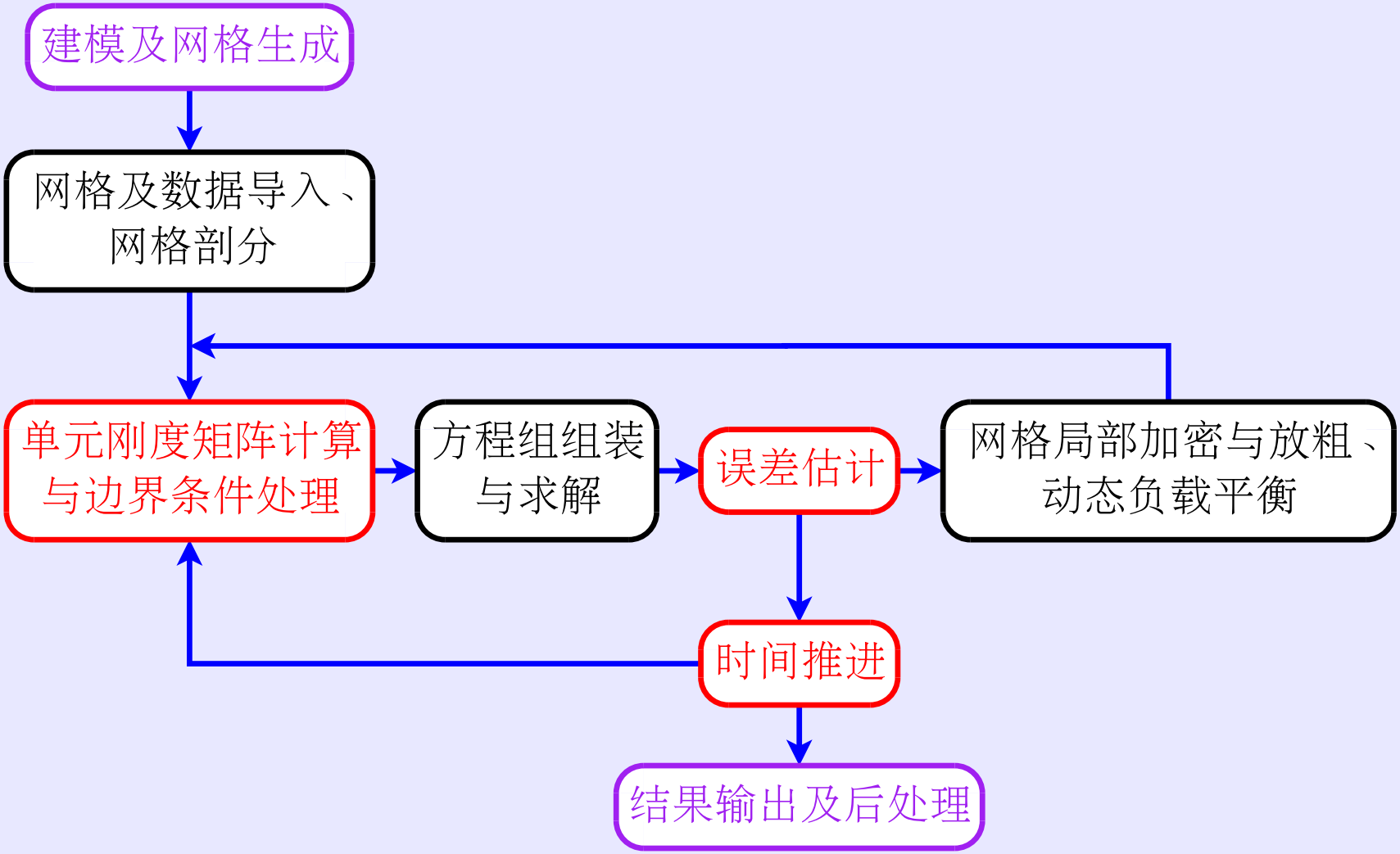


单元的形状函数

有限元基础 IV

- 边界条件处理。
  - 根据条件的形式:
    - 第一类（狄利克雷）边界条件
    - 第二类（诺伊曼）边界条件
    - 第三类边界条件
  - 根据几何位置: 点、线、面、体

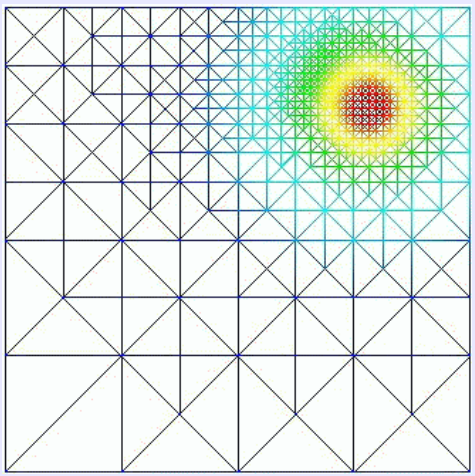
有限元基础 V

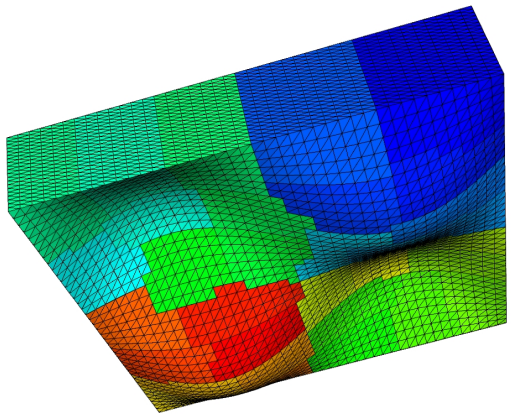


Figure： 有限元计算流程

PHG并行策略

- 空间区域分解、分布式网格





- 分布式自由度、矩阵、向量实现

PHG主要模块

网格管理

导入、导出、加密、放粗、剖分

参数管理

命令行选项、参数文件、运行中参数设定

有限元计算

基函数、有限元函数运算、数值积分

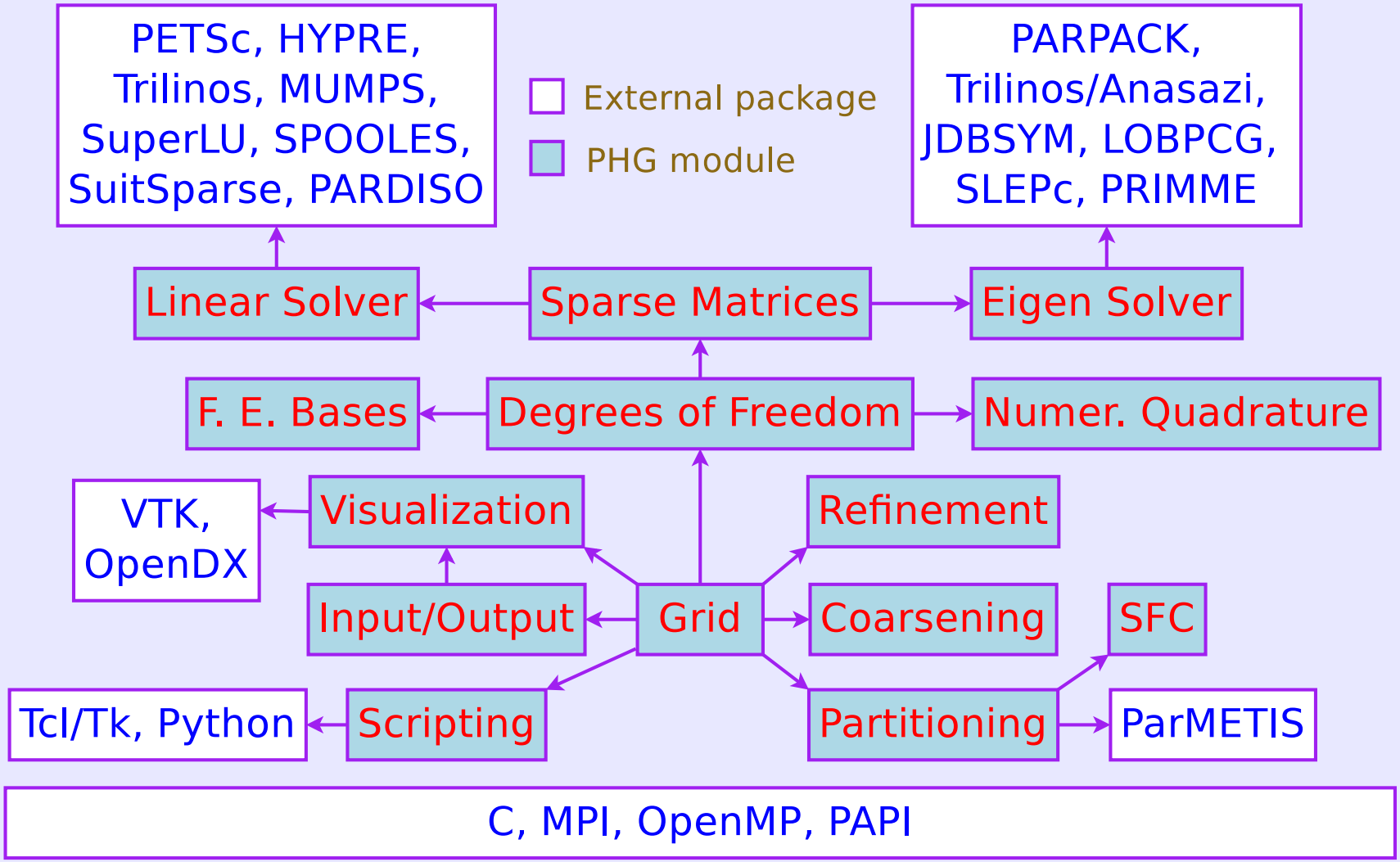
线性代数

分布式向量和稀疏矩阵管理，线性方程组求解，特征值与特征向量计算

可视化与后处理

导出VTK、OpenDX、Ensight等格式

PHG架构图


PETSc, HYPRE,
Trilinos, MUMPS,
SuperLU, SPOOLES,
SuitSparse, PARDISO
External package
PHG module
PARPACK,
Trilinos/Anasazi,
JDBSYM, LOBPCG,
SLEPc, PRIMME
Linear Solver
Sparse Matrices
Eigen Solver
F. E. Bases
Degrees of Freedom
Numer. Quadrature
VTK,
OpenDX
Visualization
Refinement
Input/Output
Grid
Coarsening
SFC
Tcl/Tk, Python
Scripting
Partitioning
ParMETIS
C, MPI, OpenMP, PAPI

Agenda

PHG的编译与安装

在PHG 源码的顶层目录中执行：

```
cp obj/MPICH2-x86_64/double/*.o src/.
rm -f src/{refine,coarsen}.c src/libphg.*
./configure
gmake
```

将编译生成PHG 的库libphg.a。注意，编译PHG 时必须使用GNU make，否则可能出错。PHG 的配置参数在运行configure 时通过选项或环境变量指定，运行

```
./configure --help
```

可以得到有关configure 的帮助信息。

PHG的编译与安装

configure 使用的环境变量主要有下面一些:

CC	指定C 编译器(默认为mpicc)
CFLAGS	指定C 编译选项
CPP	指定C 预处理器
CPPFLAGS	指定C/C++ 预处理选项(如“-I/opt/include”)
LDFLAGS	指定链接选项
LIBS	指定链接时使用的库
CXX	指定C++ 编译器(默认为mpiCC 或mpicxx)
CXXFLAGS	指定C++ 编译选项
FC	指定Fortran (90) 编译器
FCFLAGS	指定Fortran (90) 编译选项
F77	指定Fortran 77 编译器
FFLAGS	指定Fortran 77 编译选项

PHG的编译与安装

configure选项例子：

```
export local=/soft/x86_64/apps/OpenSoft/MVAPICH2/local
env CC=mpicc CXX=mpicxx FC=mpif90 F77=mpif77 LIBS=-lstdc++ \
    CPPFLAGS="-I${local}/include" \
    LDFLAGS="-L${local}/lib -Wl,-rpath,${local}/lib \
         -L/home_soft/soft/x86_64/lib/mkl/10.1.1.019/lib/em64t" \
        ./configure --enable-shared --enable-realloc-hack \
        --with-build-shared="gcc -shared" \
        --prefix=${local%/local}/phg-0.7.0 \
        --with-blas="-lmkl -lguide -lpthread"
        --with-lapack="-lmkl_lapack" \
        --with-metis-lib="-lparmetis -lmetis" \
        --with-superlu-incdir=${local}/include/superlu_dist \
        --with-superlu-lib=-lsuperlu_dist \
        --with-hypre-dir=${local}/hypre
```

C/C++编译器和MPI

PHG 的configure 脚本检测C 和C++ 编译器时优先检查mpicc、mpiCC、mpicxx 等命令。因此，如果您的MPI 系统提供了这样的编译命令，则在运行configure时不必指定任何关于MPI 的参数。如果您的MPI 系统使用了其它名称的编译器前端，可以用环境变量CC 和CXX 来指定它们。其它情况下，configure 会试图自行找出有关MPI 头文件、库等信息，如果失败，则需要用--with-mpi-libdir 和--with-mpi-incdir 选项分别指定MPI 库和头文件的路径，以及用--with-mpi-lib 选项指定MPI 的库文件，如：

```
CC=gcc CXX=g++ ./configure \
       --with-mpi-libdir=/opt/mpi/lib \
       --with-mpi-incdir=/opt/mpi/include \
       --with-mpi-lib="-lpmpich -lmpich \
       -lpmpich -lmpich -lpthread -lrt"
```

第三方软件接口

METIS/ParMETIS

如果开启了METIS/ParMETIS支持，并且configure 检测到了相应的头文件和库，则在运行程序时可以用选项“-partitioner metis” 来指定调用METIS/ParMETIS 进行网格划分或重划分。

Tcl/Tk、VTK

PHG 提供与Tcl/Tk 脚本语言的接口。为了使用Tcl/Tk 脚本功能，系统中必须安装有Tcl/Tk 及相应的开发环境。如果configure 无法自动检测到Tcl/Tk，则需要通过适当的选项(--with-tcl-* 和--with-tk-*) 来指定它们。

PHG的编译

解法器接口

除了PHG 提供的内部解法器PCG、GMRES 等之外，用户还可以调用外部解法器来求解线性方程组。目前PHG 支持的外部解法器有 PETSc、 Trilinos、HYPRE、 MUMPS、 SuperLU_Dist、 SPC 和 LASPack 等。在运行`configure`时可以指定启用哪些解法器接口(必要时需要提供头文件、库的位置等)。用户程序中具体使用哪种解法器由程序决定，或是在运行程序时由命令行选项“`-solver`”指定。LASPack 是一个串行解法器，适合在没有MPI 的系统上使用。

PHG的编译

BLAS 和LAPACK 库

一些外部软件包，如PETSc, HYPRE 和SuperLU_Dist 等需要用到BLAS 或LAPACK 库，其中PETSc 和HYPRE 需要LAPACK 和BLAS, 而SuperLU_Dist 则仅需要BLAS。如果希望在PHG 中同时使用这些包，则它们必须引用相同的BLAS 和LAPACK。用户可以通过configure的选项来指定BLAS 或/和LAPACK 库。

```
./configure  --disable-petsc --with-blas=-lgoto  \
             --with-lapack=-llapack
./configure  --disable-petsc \
             --with-lapack="-L/opt/intel/mkl/lib/32 \
             -lmkl_lapack -lmkl_def -lguide"
```

(注：可以用--with-lapack 同时指定BLAS 和LAPACK 库，但不应该用--with-blas 来指定LAPACK 库)。

PHG的安装

完成编译后，在源码目录中执行

```
gmake install
gmake install-doc
```

会将PHG 的库和头文件(及其它一些相关文件) 安装到--prefix选项指定的目录，其中后一条命令编译、安装PHG 的手册manual.pdf (需要CCT中文TEX)。

PHG的编译与安装

PHG 安装后的主要文件和目录结构如下：

/prefix/
- bin/phg
- bin/phg_tcl
- lib/libphg.a
- include/phg.h
- include/phg/{config.h,utils.h,...}
- share/phg/Makefile.inc
- share/phg/phg.tcl
- share/phg/phg-logo.gif
- share/doc/phg/{manual.pdf,README,...}
- share/doc/phg/examples/*

PHG的实用工具

PHG提供一条简单帮助命令phgdoc 用于查询PHG 函数的原型及宏。phgdoc 是一个简单的Shell 脚本，它根据命令行参数在PHG 的头文件中搜索相应的函数或宏名称并显示出来。例如：

```
% phgdoc phgImport
BOOLEAN phgImport(GRID *g, const char *filename, BOOLEAN distr);
% phgdoc phgDofCurl
#define phgDofCurl(src, dest, newtype, name) \
        phgDofCurl_(src, dest, newtype, name, __FILE__, __LINE__)
```

用户程序的编译和链接

假设用户源程序文件名为mycode.c，只需在Makefile 中加入下面一行：

```
include 目录/Makefile.inc
```

然后执行“gmake mycode”命令，便可编译生成可执行文件mycode。其中“目录”是文件Makefile.inc 所在的目录，它可以是PHG 的源码目录，也可以是“/prefix/share/phg”。

用户程序的编译、链接可参考/prefix/share/doc/phg/examples/Makefile。其中“/prefix”代表PHG 的安装路径。

Agenda

ALBERTA格式

ALBERT 1.0 的初始网格文件格式，macro triangulations。

```
DIM:                    求解问题的维数
DIM_OF_WORLD:           空间维数
number of vertices:     顶点数(nv)
number of elements:     单元数(ne)

vertex coordinates:
顶点0坐标
顶点1坐标
... ...
顶点nv-1坐标

element vertices:
单元0顶点编号
单元1顶点编号
... ...
单元ne-1顶点编号
```

element bundaries:
单元0边界类型
单元1边界类型
... ...
单元ne-1边界类型

element type:
单元0类型
单元1类型
... ...
单元ne-1类型

element neighbours:
单元0的邻居单元
单元1的邻居单元
... ...
单元ne-1的邻居单元

- “element boundaries”中每行包含四个整数，给出相应单元四个面的边界类型，
  - 1 表示Dirichlet 边界，
  - <0 表示Neumann 边界，
  - 2–11 分别表示BDRY_USER0–BDRY_USER9，
  - 0 表示区域内部。
- 如果输入文件中不包含“element type”项，则PHG 会根据特定的算法自动为每个单元指定一种类型，并且相应修改改单元中四个顶点的顺序，以确保初始网格满足二分细化算法所要求的相容性条件。
- 如果输入文件中不包含“element boundaries”项，则PHG 将所有边界面的类型置为UNDEFINED。

Medit格式

- PHG 可以导入Medit mesh format 格式的网格，但只读入`Vertices`、`Tetrahedra`、`Hexahedra`、`Triangles` 和`Quadrilaterals` 数据，忽略其它数据
- 边界类型
  - 1 表示Dirichlet 边界，
  - 2 表示Neumann 边界，
  - 其他表示UNDEFINED。
- 如果使用Netgen生成网格的话，可以利用脚本`utils/netgen2medit` 将Netgen 的`.geo` 或neutral 格式文件转换为Medit 格式。
- PHG 提供了一个脚本`utils/tetgen2medit` 将tetgen生成的网格转换为Medit 格式。

边界类型

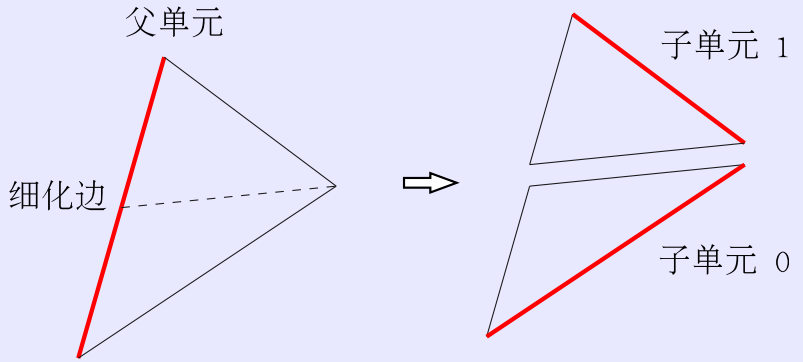
PHG提供的边界类型有：DIRICHLET、NEUMANN、BDRY_USER0– BDRY_USER9和、UNDEFINED。在调用phgImport 之前，用户可以调用phgImportSetBdryMapFunc 为其指定一个边界类型转换函数，如下例所示：

```
static int
bc_map(int bctype){
    switch (bctype) {
        case 1:         return DIRICHLET;
        case 2:         return NEUMANN;
        case 3:         return BDRY_USER1;
        case 4:         return BDRY_USER2;
        default:        return -1;    /* invalid bctype */
    }
}
... ...
phgImportSetBdryMapFunc(bc_map);
phgImport(... ...);
... ...
```

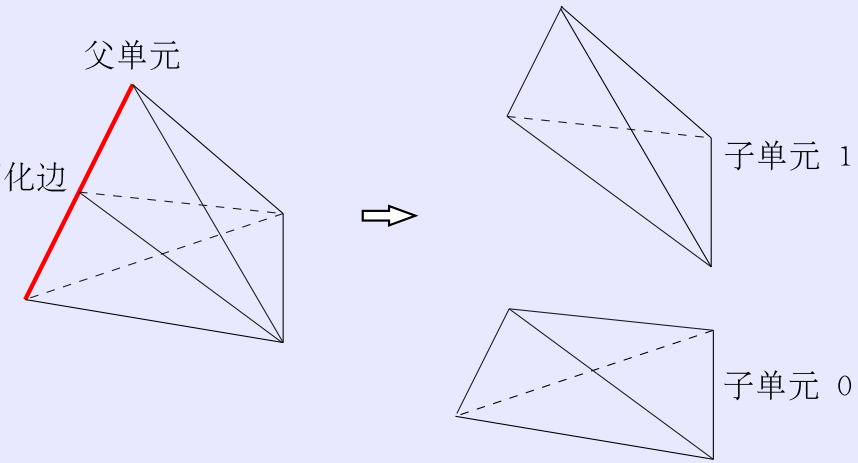
Agenda

三角形、四面体单元的二分细化

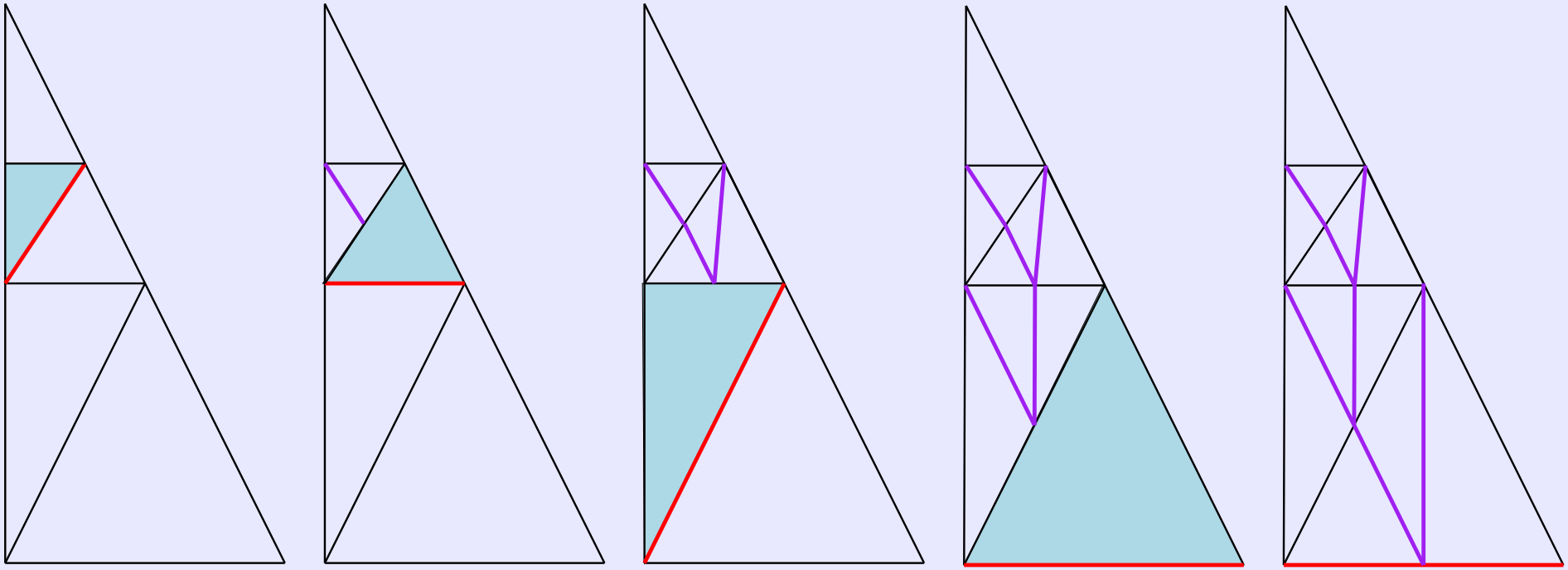
三角形单元：



四面体单元：



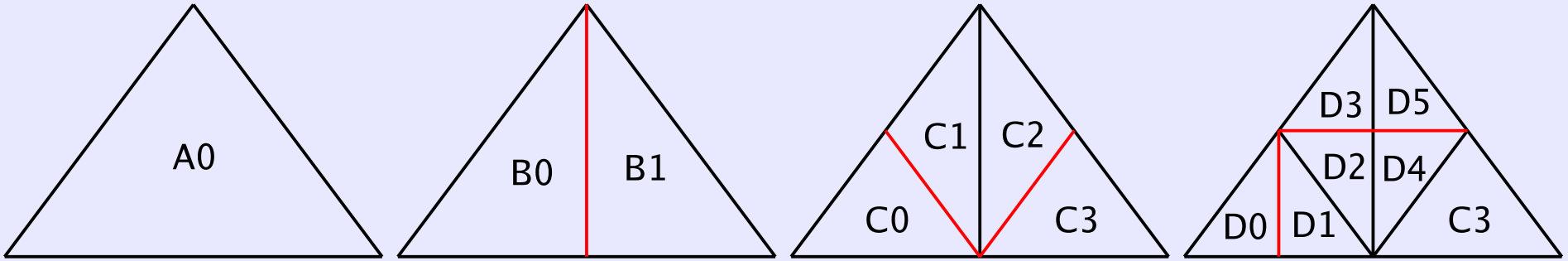
基于二分细化的自适应网格局部加密

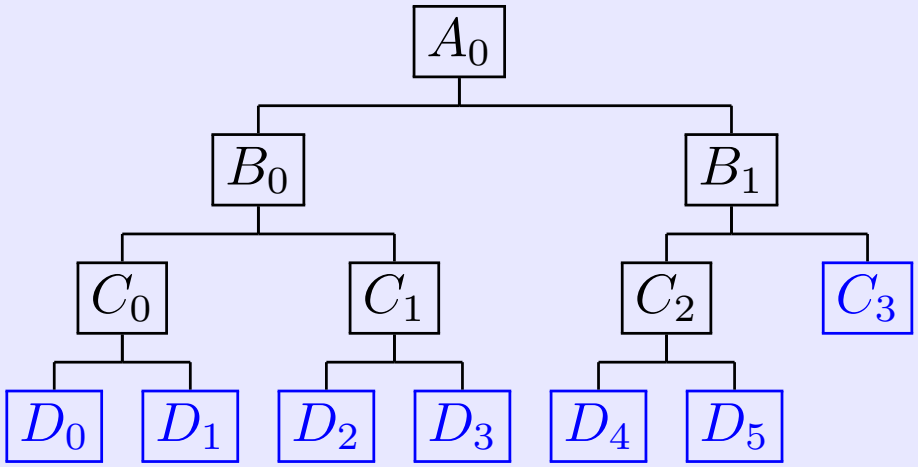


算法实现的主要困难：算法的非局部性。

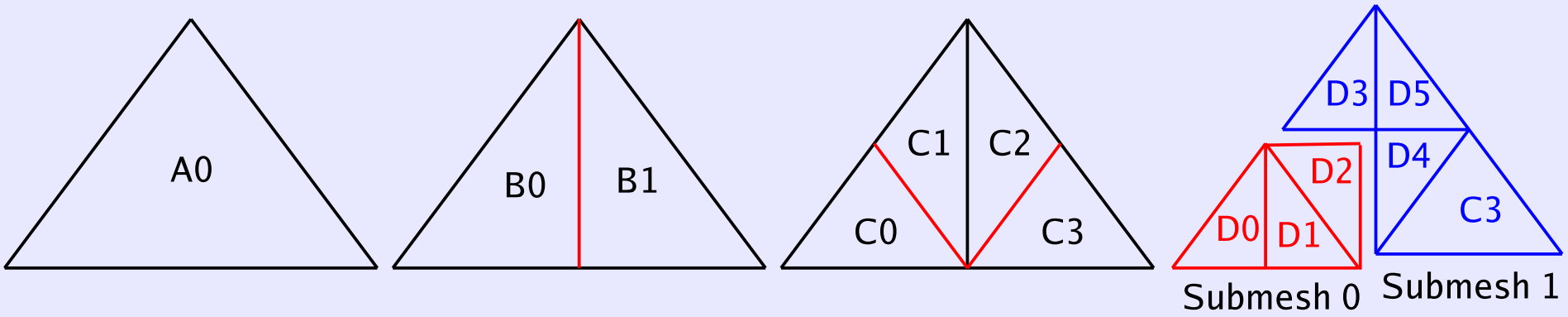
极端情形下，单个单元的细化可能导致大量甚至全部单元的细化。

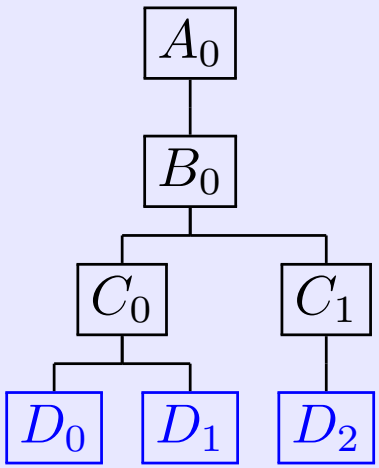
自适应层次网格结构

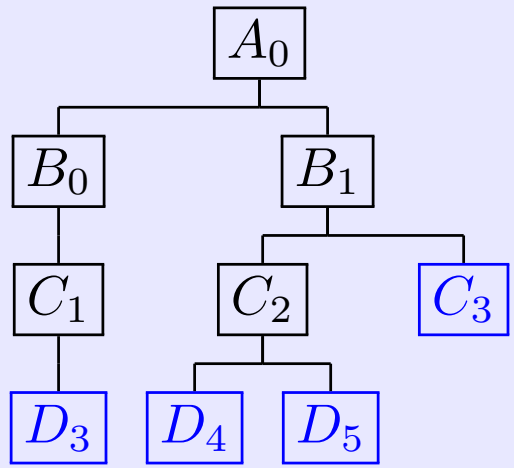

A0
B0
B1
C0
C1
C2
C3
D0
D1
D2
D3
D4
D5


A_0
B_0
B_1
C_0
C_1
C_2
C_3
D_0
D_1
D_2
D_3
D_4
D_5

分布式自适应层次网格结构

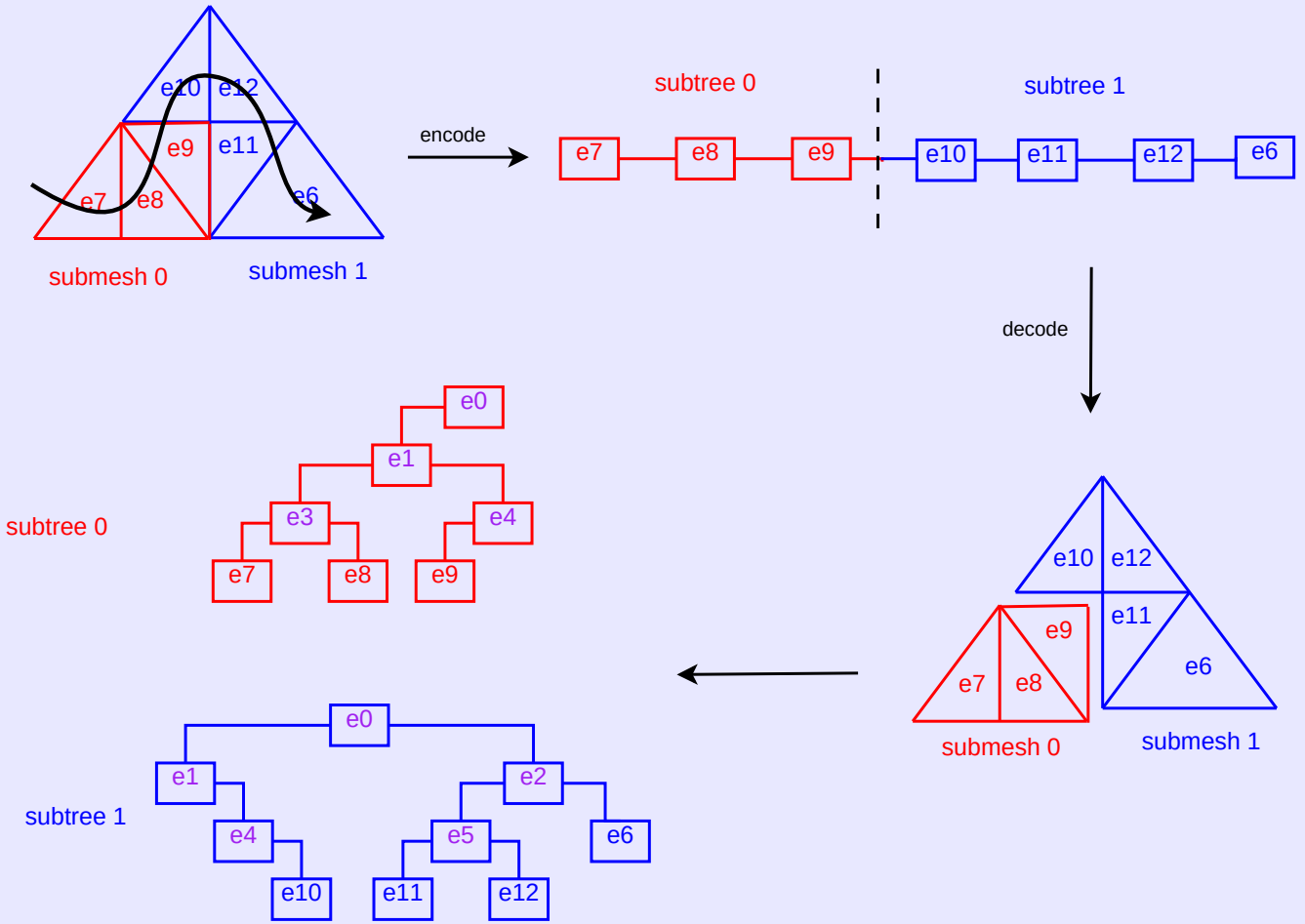






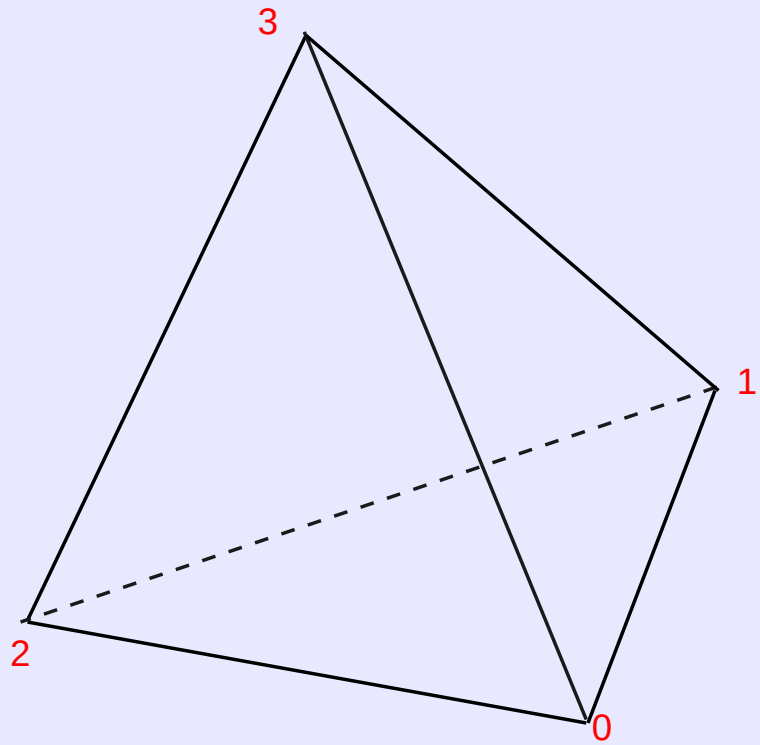
网格划分

- SFC, Hamilton path



- Parmetis, Zoltan.

单元内部顶点、边、面的编号



顶点、边和面的单元内编号

- 四个顶点分别编号为0、1、2、3，
- 六条边分别编号为
  - 0(包含顶点0–1)、
  - 1 (包含顶点0–2)、
  - 2 (包含顶点0–3)、
  - 3 (包含顶点1–2)、
  - 4 (包含顶点1–3)、
  - 5 (包含顶点2–3)。
- 四个面分别编号为0, 1, 2, 3，其中面i指不包含顶点i的面。

单元内编号、本地编号和全局编号

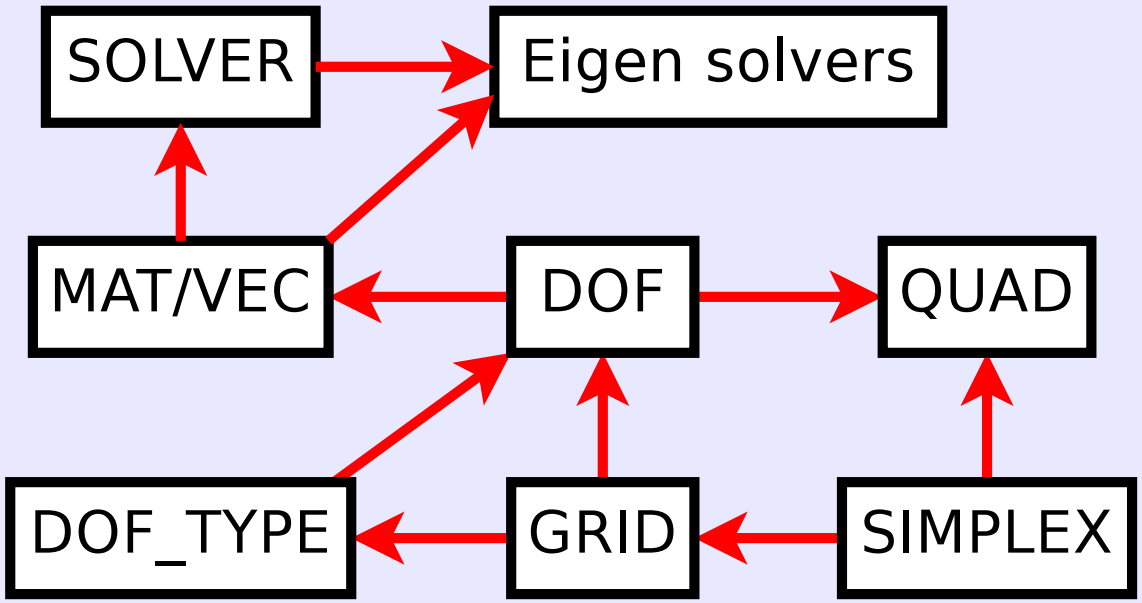
PHG中的几何对象，包括顶点、边、面、单元、自由度等，有三种不同的编号方式，分别称为

- 单元内编号: 在$0 - 3$之间
- 本地编号: 本地编号又称为子网格内编号或局部编号，它指对象在当前子网格中的编号，在$0 - (\texttt{nvert} - 1)$之间
- 全局编号: 全局编号则指对象在全局网格中的编号。在$0 - (\texttt{nvert_global} - 1)$ 之间

基本数据类型

- FLOAT、INT、UINT、SHORT、USHORT、CHAR、BYTE、BOOLEAN
- 常量：DIM、NVert、NFace、NEdge
- GTYPE: 描述几何对象的几何类型，可取值为
  - VERTEX、EDGE、FACE
  - DIAGONAL、OPPOSITE、MIXED、UNKNOWN
- BTYPE: 描述边界类型

PHG的主要数据结构



Agenda

网格数据结构: ELEMENT

```
typedef struct ELEMENT_ {
    struct ELEMENT_     *children[2];
    void                *neighbours[NFace];
    void                *parent;
    INT                 verts[NVert];    顶点本地编号
    INT                 edges[NEdge];    边本地编号
    INT                 faces[NFace];    面本地编号
    INT                 index;
    SHORT               mark;
    SHORT               region_mark;
    BTYPE               bound_type[NFace];
    ... ...
} ELEMENT;
```

网格数据结构: GRID

```
typedef struct GRID_ {
    FLOAT       lif;            /* 负载不平衡因子 */
    COORD       *verts;         /* 顶点坐标 */
    struct DOF_ *geom;      /* 存储网格几何信息 */
    struct DOF_ **dof;      /* 与该网格关联的自由度列表*/
    ... ...
    INT         nleaf;
    INT         nvert, nvert_global;
    INT         nedge, nedge_global;
    INT         nface, nface_global;
    INT         nelem, nelem_global;
    ELEMENT *roots;     /**< Root elements */
    INT         nroot;
    INT         ntree;
   ... ...
    int         rank;           /* 进程号 */
    int         nprocs;         /* 进程数 (子网格数) */
#if USE_MPI
    MPI_Comm    g->comm;        /* MPI 通信器 */
#endif
    ... ...
```

数据结构：GRID

- verts 数组用于保存子网格中所有顶点的迪卡尔坐标，按顶点的本地编号顺序存放。
- nleaf 给出当前子网格包含的单元数，即当前子树包含的叶子单元数。
- nvert_global、nedge_global、nface_global 和nelem_global 分别给出当前全局网格中的顶点数、边数、面数和单元数。
- nvert、nedge、nface 和nelem 分别等于顶点、边、面和单元的最大本地编号加1。
- types_vert、types_edge、types_face 和types_elem，它们是维数分别为nvert、nedge、nface 和nelem 的数组，分别给出当前子网格中的顶点、边、面和单元的边界属性。

网格操作

导入与销毁

```
GRID *phgNewGrid(int flags);
BOOLEAN phgImport(GRID *g, const char *filename, BOOLEAN distr);
void phgFreeGrid(GRID **gptr);
```

网格单元遍历[a]

[a]该宏定义不能嵌套执行

```
GRID *g;
ELEMENT *e;

ForAllElements(g,e){
... ...
```

网格操作

网格剖分与负载均衡

```
int phgBalanceGrid(GRID *g, FLOAT lif_threshold,
     INT submesh_threshold, struct DOF_ *weights, FLOAT power);
void phgRedistributeGrid(GRID *g);
void phgPartitionGrid(GRID *g);
```

网格加密与放粗

```
void phgRefineAllElements(GRID *g, int level);
void phgRefineMarkedElements(GRID *g);
void phgCoarsenMarkedElements(GRID *g);
```

Agenda

自由度管理模块

- 用于存储、管理任何分布在网格上的数据，如几何数据(Jacobian, 法向量, 单元、面的直径等)，有限元函数，误差估计子，等等
- 有限元基函数：L_2, H^1, $\boldsymbol{H}_{\text{curl}}$, $\boldsymbol{H}_{\text{div}}$, Lagrange型, Hierarchical型。
- 提供有限元函数的插值、投影、代数和微分运算，支持用于处理常量和解析函数的特殊自由度类型

自由度类型 I

DOF_TYPE 描述自由度对象的基本特征，其数据结构中的主要成员如下：

```
typedef struct DOF_TYPE_ {
    ...                         /* 积分点基函数值缓存区 */
    const char      *name;              /* 自由度类型的名称或描述 */
    FLOAT       *points;            /* 自由度位置（重心坐标）*/
    BYTE        *orders;            /* 各个基函数的多项式次数 */
    struct DOF_TYPE_    *grad_type;     /* 梯度值的自由度类型 */

    /* 函数指针 */
    DOF_INTERP_FUNC InterpC2F;          /* 粗网格到细网格插值函数 */
    DOF_INTERP_FUNC InterpF2C;          /* 细网格到粗网格插值函数 */
    DOF_INIT_FUNC   InitFunc;           /* 投影函数（Π_h）*/
    DOF_BASIS_FUNC  BasFuncs;           /* 基函数 */
    DOF_BASIS_GRAD  BasGrads;           /* 基函数梯度 */

    BOOLEAN     invariant;          /* 基函数是否与单元形状无关 */
```

自由度类型 II

```
    BOOLEAN     free_after_use;     /* 是否需要自动释放 */
    SHORT       id;             /* 自由度类型编号 */
    SHORT       nbas;               /* 一个单元中的自由度个数 */
    BYTE        order;              /* 基函数的最高多项式次数 */
    CHAR        continuity;         /* 有限元函数的连续性 */
    SHORT       dim;                /* 基函数维数 */

    SHORT       np_vert;            /* 每个顶点上的自由度个数 */
    SHORT       np_edge;            /* 每条边上的自由度个数 */
    SHORT       np_face;            /* 每个面上的自由度个数 */
    SHORT       np_elem;            /* 每个单元中的自由度个数 */
} DOF_TYPE;
```

数组points 顺序给出顶点、边、面和单元自由度的插值点信息，分别用0维、1维、2维和3维重心坐标表示。points 的总长度应该是

$$\texttt{np\_vert} + 2 \times \texttt{np\_edge} + 3 \times \texttt{np\_face} + 4 \times \texttt{np\_elem}$$

例如，对于四阶Lagrange元，每个顶点上有1个自由度，每条边上有三个自由度，每个面上

自由度类型 III

有三个自由度，每个单元体内有1个自由度，points 数组的内容如下：

```
static FLOAT points = {
    1.,                 /* 顶点自由度位置 */
    .75,.25, .5,.5, .25,.75,        /* 边自由度位置 */
    .5,.25,.25, .25,.5,.25, .25,.25,.5, /* 面自由度位置 */
    .25,.25,.25,.25         /* 体自由度位置 */
};
```

当自由度类型没有插值点时，应该将points 置为NULL。PHG 中假定，当points 不等于NULL 时，表明自由度的值等于其插值点处的函数值。

自由度对象数据结构

自由度对象数据结构中的主要成员如下：

```
typedef struct DOF_ {
    char     *name;        /* 名称或描述 */
    GRID     *g;       /* 网格对象 */
    DOF_TYPE     *type;        /* 自由度类型 */
    FLOAT    *data;        /* 存储自由度数据的缓冲区地址 */
    ... ...
    SHORT    dim;          /* 自由度对象的维数 */
} DOF;
```

自由度管理模块

自由度创建与销毁

```
DOF *u, *grad;
u = phgDofNew(g, DOF_DEFAULT, 1, "u", DofNoAction);
... ...
phgDofFree(&u);
```

直接访问自由度数据

```
DofVertexData(dof, 顶点本地编号)
DofEdgeData(dof, 边的本地编号)
DofFaceData(dof, 面的本地编号)
DofElementData(dof, 单元本地编号)
DofData(dof)
```

访问自由度数据–例子

```
    INT i, n;
    GRID *g;
    DOF *dof;
    FLOAT sum, *data;
    ... ...
    n = DofGetDataCount(dof);              /* 本地自由度数组大小 */
    data = DofData(dof);                   /* 本地自由度数据地址 */
    sum = 0.0;
    for (i = 0; i < n; i++, data++) {
        if (!DofIsOwner(x, i))  continue;
        sum += (*data) * (*data);
    }
#if USE_MPI
    if (g->nprocs > 1) {                   /* 子网格间全局求和 */
         ... ...
    }
#endif
```

自由度数据存放与编号

- data 指向存储自由度数据的缓冲区，它的长度等于自由度对象在当前子网格中所有自由度的个数(假设与自由度对象相关联的网格对象为g)每个进程只保存属于自己子网格部分的自由度数据。对于同时属于多个子网格的顶点、边或面自由度，它们重复存储在不同进程中并保持一致。自由度对象的数据分别按顶点、边、面和体自由度根据它们的本地编号顺序连续存放，当网格变化时PHG 会自动对它们进行调整。确切地，一个子网格中自由度数据的存放顺序可以用下面的数组示意：

```
{FLOAT[g->nvert][type->np_vert][dim],
 FLOAT[g->nedge][type->np_edge][dim],
 FLOAT[g->nface][type->np_face][dim],
 FLOAT[g->nelem][type->np_elem][dim]}
```

- 自由度采用局部编号，在不同子网格中有不同的编号，只有一个子网格是他的属主，通过MAP和VEC对象创建和管理自由度对象的全局编号。

自由度赋值 I

自由度类型中的投影函数(又叫初始化或赋值函数) InitFunc 根据函数值计算指定的自由度值，即将一个函数投影到指定的有限元空间，其接口类型为DOF_INIT_FUNC，具体形式如下：

```
void InitFunc(DOF *dof, ELEMENT *e, GTYPE type, int index,
      DOF_USER_FUNC userfunc,
      DOF_USER_FUNC_LAMBDA userfunc_lambda,
      const FLOAT *funcvalues, FLOAT *dofvalues,
      FLOAT **pdofvalues)
```

参数type 指定要计算的自由度类别，VERTEX 表示顶点自由度，EDGE 表示边自由度，FACE 表示面自由度，VOLUME 表示单元自由度。参数index 给出顶点、边或面在单元中的编号，当type 为VOLUME 时index 值被忽略。InitFunc 计算指定位置的自由度值，并将结果放在参数dofvalues 指向的缓冲区中返

自由度赋值 II

回给调用程序，该缓冲区由调用的程序提供，长度为dof->dim × np_xxxx，数据在缓冲区中的的存放顺序为FLOAT[np_xxxx][dof->dim] (xxxx 根据type 的不同值分别代表vert、edge、face 或elem)。注意，对Lagrange 型基函数，当type 为EDGE 或FACE 并且np_xxxx > 1 时，InitFunc 需要根据边或面的顶点的全局编号调整数据的存放顺序，对np_xxxx 组数据(每组包含dof->dim 个数) 进行适当置换，以保证相邻单元间数据的一致性。

参数userfunc、user_func_lambda 和funcvalues 分别为两个函数指针和一个数组指针，它们给出计算函数值的函数指针或包含函数值的缓冲区地址，三个指针中必须有且仅有一个为非空指针。userfunc 指向一个关于x, y, z 的函数，其接口类型为DOF_USER_FUNC，具体如下：

```
void userfunc(FLOAT x, FLOAT y, FLOAT z, FLOAT *values)
```

自由度赋值 III

userfunc_lambda 则指向一个关于重心坐标的函数，其接口类型为DOF_USER_FUNC_LAMBDA，具体如下：

```
void userfunc_lambda(DOF *dof, ELEMENT *e, int bno,
     const FLOAT lambda[], FLOAT *values)
```

当funcvalues 为非空指针时，它指向存有预先计算好的函数值的缓冲区。此时要求自由度类型中的points 成员是非空指针，并且各个位置的自由度值完全由该位置的函数值确定。funcvalues 中包含指定顶点、边、面或单元处的所有np_xxxx 个位置的函数值，共计DofDim(dof) × np_xxxx 个数，这些位置根据points 中的数据确定。

自由度赋值 IV

一些基函数，如hierarchical basis 类的基函数，在计算边、面或体自由度时，需要用到低维位置处的自由度值。例如计算边自由度时需要用到该边的两个顶点处的自由度值，计算面自由度时需要用到该面的三个顶点和三条边处的自由度值。为了避免重复计算，PHG 中的函数在调用`InitFunc` 时总是在一个单元内严格按从低维到高维的顺序进行。因此，在计算一个位置的自由度时，在其上的低维位置处的自由度值已经计算完毕，可以直接引用。如果参数`pdofvalues` 为空指针，表明所需的低维位置处的自由度值可以从`dof->data` 中得到。如果参数`pdofvalues` 为非空指针，则表明`dof->data` 中的数据不可用，需要从`pdofvalues` 所指向的地址获得所需的低维位置处的自由度值。`pdofvalues` 中包含15个指针，分别指向4个顶点、6条边、4个面和体内的自由度数据，

自由度基函数 I

自由度类型成员BasFuncs 负责计算指定重心坐标位置的部分或全部基函数值，其接口类型为DOF_BASIS_FUNC，接口参数如下：

```
const FLOAT *BasFuncs(DOF *dof, ELEMENT *e, int no0, int no1,
                      const FLOAT *lambda)
```

其中no0 和no1 为局部基函数编号范围(依次按照单元中顶点¿、边、面和体自由度的顺序编号，从0 开始)，表示计算no0 到no1 - 1 之间的所有基函数，如果no1 <= 0 则表示no1 = dof->type->nbas。

lambda[Dim + 1] 为重心坐标。该函数返回一个缓冲区地址，其中包含所指定范围的或全部基函数的值，该缓冲区由BasFuncs 提供，通常是静态的，每次调用BasFuncs时缓冲区中的内容会被新值替换，或者缓冲区的地址会失效。BasFuncs 共返回dof->type->dim * (no1 - no0) 个值，按FLOAT[][dof->type->dim] 的顺序排列。

自由度基函数 II

自由度类型成员BasGrads 负责计算指定重心坐标位置处部分或全部基函数关于重心坐标的梯度值，其接口类型为DOF_BASIS_GRAD，接口参数为：

```
const FLOAT *BasGrads(DOF *dof, ELEMENT *e, int no0, int no1,
                      const FLOAT *lambda)
```

各参数的含义与BasFuncs 类似，函数返回值的个数正好是BasFuncs 函数的Dim + 1倍，存储顺序为FLOAT[][dof->type->dim][Dim + 1]。

自由度管理模块

自由度对象的运算

```
phgDofCopy, phgDofAXPY, phgDofAXPBY, phgDofMatVec, phgDofMM
phgDofNormL1, phgDofNormL1vec, phgDofNormL2, phgDofNormL2vec
phgDofGradient, phgDofDivergence, phgDofCurl
```

有限元函数求值

```
FLOAT *phgDofEval(DOF *dof, ELEMENT *e, const FLOAT lambda[],
                  FLOAT *values);
FLOAT *phgDofEvalCurl(DOF *dof, ELEMENT *e, const FLOAT lambda[],
    const FLOAT *gradbas, FLOAT *values);
FLOAT *phgDofEvalDivergence(DOF *dof, ELEMENT *e, const FLOAT lambda[],
    const FLOAT *gradbas, FLOAT *values);
FLOAT *phgDofEvalGradient(DOF *dof, ELEMENT *e, const FLOAT lambda[],
    const FLOAT *gradbas, FLOAT *values);
void phgInterGridDofEval(DOF *u, INT n, COORD *pts,
                         FLOAT *values, int flag);
```

访问邻居单元自由度数据

```
NEIGHBOUR_DATA *aa;
ELEMENT *e;
int i;
FLOAT t;

aa = phgDofInitNeighbourData(u, NULL);
ForAllElements(g, e){
   for (i = 0; i < NFace; i++) {
      if( 边界面 )
           t=*DofElementData(u, e->index);
      else {
           t=(*DofElementData(u, e->index) +
               *phgDofNeighbourData(aa, e, i, 0, NULL))*0.5;
      }
      ... ...
   }
}
phgDofReleaseNeighbourData(&aa);
```

特殊自由度类型 I

PHG提供了两个特殊自由度类型，包括常量型自由度类型DOF_CONSTANT和解析型自由度类型DOF_ANALYTIC。

- 常量型自由度类型可用于存储取值为常数的函数。例如，下面的代码定义了一个表示取值为常向量(1,2,3) 的自由度对象：

```
GRID *g;
DOF *u;
FLOAT values[] = {1., 2., 3.};
... ...
u = phgDofNew(g, DOF_CONSTANT, 3,
              "constant vector", DofNoAction);
phgDofSetDataByValues(u, values);
... ...
```

特殊自由度类型 II

PHG不允许对常量型自由度对象进行微分运算。

- 解析型自由度类型用于处理解析函数。一个解析型自由度对象既可以与一个基于迪卡尔坐标的函数相关联(userfunc成员), 也可以与一个基于重心坐标的函数相关联(userfunc_lambda成员)。其中userfunc 既可以通过函数phgDofNew 的参数设定，也可以调用函数phgDofSetFunction设定，而userfunc_lambda 则只能调用函数phgDofSetLambdaFunction 设定。

不允许对其进行微分运算，也不能做为函数phgDofAXPY 中的“y”参数。

几何自由度对象 I

文件geom.c 中实现了一个特殊自由度对象，为用户提供有限元计算中经常用到的一些几何量，包括单元面的面积、法向量、直径，单元的体积、直径¿ 和重心坐标Jacobian。geom.c 中提供了一组名为phgGeomXxxxxx 的函数供用户调用来获取这些几何量。

```
phgGeomGetDiameter,      phgGeomGetVolume,
phgGeomGetFaceArea,      phgGeomGetFaceDiameter,
phgGeomGetFaceNormal,    phgGeomGetFaceOutNormal,
phgGeomGetJacobian
```

定义新的自由度类型

定义一个新的自由度类型时，只需填写相应的数据类型，并实现相应的接口函数，可以参考lagrange.c 中Lagrange 元的定义，或geom.c 中几何量的定义。接口函数中，只需提供需要用到的函数，用不到的可以不提供，直接写成NULL 即可。

```
static FLOAT P1_points[] = {_F(1.0)};
DOF_TYPE DOF_P1_ = {DofReserved,
            "P1", P1_points, NULL, DOF_DG0, NULL, NULL,
            P1_interp, phgDofInterpF2CGeneric, phgDofInitFuncPoint,
            P1_bas, P1_grad, NULL, FE_H1,
            TRUE, FALSE, -1, /* invariant, free_after_use, id */
            NVert, 1, 0, 1,  /* nbas, order, cont, dim */
            1, 0, 0, 0};     /* np_vert, np_edge, np_face, np_elem */
static DOF_TYPE DOF_JUMP_ = { DofReserved,
    "Face jump", NULL, NULL, NULL, NULL, NULL,
    NULL, NULL, NULL,
    NULL, NULL, NULL, FE_None,
    TRUE, FALSE, -1,
    NFace, 0, -1, 1,
    0, 0, 1, 0};
```

Agenda

数值积分模块

- 提供一维(线段)、二维(三角形)、三维(四面体)上任意阶代数精度的高效数值积分公式
- 提供单元上多种形式单、双、三线性型计算，方便单元刚度矩阵、载荷向量的计算

例：下述函数计算u的第i个基函数φ_i与v的第j个基函数ψ_j的梯度乘积的积分$\int_e \nabla \varphi_i \boldsymbol{A} \nabla \psi_j \,\mathrm{d}x$:

```
phgQuadGradBasAGradBas(ELEMENT *e, DOF *u,
                       int i, DOF *A, DOF *v, int j, -1);
```

$\boldsymbol{A}$ 可以是普通有限元函数、常量或解析函数，其取值可以是标量、向量或(3×3) 矩阵，$\boldsymbol{A}$ 取NULL 表示$\boldsymbol{A} = 1$。

数值积分

计算单元基函数或其梯度、旋度等的积分，以及跳量。部分接口如下：

- phgQuadDofDotDof： $\iiint_e \Pi_h U \cdot \Pi_h V \,\mathrm{d}V$
- phgQuadBasDotBas： $\iiint_e W_n \cdot M_m \,\mathrm{d}V$
- phgQuadCurlBasDotCurlBas： $\iiint_e \boldsymbol{\nabla}\times W_n \cdot \boldsymbol{\nabla}\times M_m \,\mathrm{d}V$
- phgQuadDofTimesBas： $\iiint_e \Pi_h U \cdot M_n \,\mathrm{d}V$
- phgQuadDofDotBas： $\iiint_e \Pi_h U \cdot M_n \,\mathrm{d}V$
- phgQuadFuncDotBas： $\iiint_e f \cdot M_n \,\mathrm{d}V$
- phgQuadFaceJump： 计算某个DOF对象在网格面上的跳量

自定义数值积分接口 I

```
quad = phgQuadGetQuad3D(order);
g1 = phgQuadGetBasisValues(e, u, n, quad);
g2 = phgQuadGetBasisValues(e, v, m, quad);
d = 0.;
w = quad->weights;
for (i = 0; i < quad->npoints; i++) {
    d0 = 0.;
    for (j = 0; j < nvalues; j++) {
        d0 += *(g1++) * (*(g2++));
    }
    d += d0 * (*(w++));
}
return d * phgGeomGetVolume(u->g, e);
```

相关接口:

自定义数值积分接口 II

- phgQuadGetBasisValues 函数说明： 输入参数为单元e，自由度u，基函数的序号n，积分类型quad。该函数返回自由度u在单元e上的第n个基函数在积分子quad的求值点处的值。

```
FLOAT *phgQuadGetBasisValues(ELEMENT *e, DOF *u, int n,
                             QUAD *quad)
```

- phgQuadGetBasisGradient 函数说明： 参数含义同phgQuadGetBasisValues，计算的是基函数关于笛卡尔坐标的梯度。

```
FLOAT *phgQuadGetBasisGradient(ELEMENT *e, DOF *u, int n,
                               QUAD *quad)
```

- phgQuadGetBasisCurl 函数说明： 参数含义同phgQuadGetBasisValues，计算的是基函数的旋度。

```
FLOAT *phgQuadGetBasisCurl(ELEMENT *e, DOF *u, int n,
                           QUAD *quad)
```

自定义数值积分接口 III

- phgQuadGetDofValues函数说明： 计算自由度对象的值。

```
FLOAT *phgQuadGetDofValues(ELEMENT *e, DOF *u, QUAD *quad)
```

Agenda

映射、矩阵、向量模块

PHG 的映射(MAP) 描述一个向量在进程间的分布，以及向量元素与自由度之间的对应关系。

- 简单映射

```
MAP *phgMapCreateSimpleMap(MPI_Comm comm, INT m, INT M);
```

- 自由度映射

```
MAP *phgMapCreate(DOF *u, ...);
```

调用该函数时用一组自由度变量做为其参数，参数表必须以空指针NULL结束。所生成的MAP 对应于参数表给出的自由度对象中的所有自由度构成的向量，向量的全局大小为所有自由度对象的自由度数之和，每个进程中的分块大小等于该进程所拥有的自由度个数(注意，同时属于多个进程的自由度只被其中一个进程所“拥有”，由自由度所在的点、边、面或单元的OWNER标志确定)。

映射中的编号

向量编号 元素在分布式向量中的序号为元素的向量编号，

局部向量编号 在0到m之间

全局向量编号 局部向量编号+本进程中第一个元素的全局编号

局部自由度编号(局部编号) 基于本地自由度编号。

- 对于简单映射，局部编号= 本定向量编号
- 对于自由度映射，局部编号= 所有自由度对象中自由度的局部编号顺序联接起来。

PHG 提供一组函数用来转换这些编号，包括：

phgMapE2L 由一个单元中的自由度编号得到映射中的局部编号

phgMapD2L 由一个自由度对象中的自由度编号到映射中的局部编号

phgMapL2V 由元素的局部编号得到它的局部向量编号

phgMapL2G 由元素的局部编号得到它的全局向量编号

映射中的nlocal和localsize

在MAP 数据结构中，nlocal 成员为向量的本地分块大小(对应于m)。为了便于矩阵向量操作，向量的局部编号中还允许包含一部分不属于本地的向量分量，localsize 成员为向量的局部编号的总数，localsize $\geq$ nlocal, 局部编号在[0:nlocal-1] 之间的分量为本地分量，而局部编号在[nlocal:localsize-1] 之间的分量则为非本地分量，它们的全局向量编号由数组O2Gmap[] 给出，该数组的大小为localsize-nlocal。因此，假设一个元素的局部向量编号为i，则它的全局向量编号为：

$$\begin{cases} i + n_0 & \text{如果 } i < \texttt{nlocal} \quad \text{(本地元素)} \\ \texttt{L2Gmap}[i - \texttt{nlocal}] & \text{如果 } i \geq \texttt{nlocal} \quad \text{(非本地元素)} \end{cases}$$

其中n_0表示本进程中的最小全局向量编号。

对于通过自由度对象创建的映射，localsize 成员等于位于本地子网格上的自由度的数目，而nlocal 成员则等于本进程所拥有的自由度的数目。

在简单映射中，localsize = nlocal，映射中不存储不属于本地的元素，局部编号和局部向量编号是一样的。

向量

- 创建向量:

```
VEC *phgMapCreateVec(MAP *map, int nvec);
```

其中nvec指定向量的维数。

- 消毁向量:

```
void phgVecDestroy(VEC **vec_ptr);
```

向量组装

一个向量创建后，既可以调用使用局部编号的函数

```
phgVecAddEntry
phgVecAddEntries
```

以及使用全局编号的函数phgVecAddGlobalEntry phgVecAddGlobalEntries 添加向量元素，完成对向量元素的赋值后，应该调用phgVecAssemble 函数对向量进行组装。

需要注意的是，使用函数phgVecAddXXXXXX 只能向一个尚未组装的向量添加元素，如果想向一个已经组装的向量添加元素，可以先调用phgVecDisassemble 对向量进行卸装。

向量本地分块的数据存储在VEC 结构的data 成员中，数据长度为map->nlocal × nvec。不属于本地分块，但位于本地子网格的自由度所对应的向量元素存储在VEC 结构的offp_data 成员中，数据长度为

$$(\texttt{map->localsize} - \texttt{map->nlocal}) \times \texttt{nvec}$$

向量组装时，不属于本地的元素会被发送叠加到相应的进程。

自由度与向量间的数据传递 I

向量和与之相关联的自由度对象之间传递数据

`phgMapLocalDataToDof` 和`phgMapDofToLocalData`

多维向量与自由度对象数组之间传递数据

`phgMapVecToDofArrays` 和`phgMapDofArraysToVec`

矩阵 I

- 创建矩阵：

```
MAT *phgMapCreateMat(MAP *rmap, MAP *cmap);
```

rmap给出矩阵的行映射，cmap给出矩阵的列映射，它们可以是同一个映射(行列具有相同大小和分布的方阵)，也可以是不同的映射。

- 消毁矩阵：

```
void phgMatDestroy(MAT **mat_ptr);
```

矩阵 II

- 添加矩阵元素

phgMatAddEntry	phgMatAddEntries
phgMatAddGlobalEntry	phgMatAddGlobalEntries
phgMatAddGLEntry	phgMatAddGLEntries
phgMatAddLGEntry	phgMatAddLGEntries

- 对矩阵元素赋值

phgMatSetEntry	phgMatSetEntries
phgMatSetGlobalEntry	phgMatSetGlobalEntries
phgMatSetGLEntry	phgMatSetGLEntries
phgMatSetLGEntry	phgMatSetLGEntries

- 组装: phgMatAssemble

特殊矩阵 I

- “无矩阵”矩阵: 只有矩阵的大小和分布信息、没有矩阵数据的矩阵，它调用一个用户提供的函数来完成矩阵与向量的乘积操作。

```
MAT *phgMapCreateMatrixFreeMat(MAP *rmap,
                               MAP *cmap, MV_FUNC mv_func,
                                    void *mv_data0, ...);
```

- 分块矩阵: `phgMatCreateBlockMatrix`

矩阵、向量运算 I

- 函数phgMatAXPBY 计算$Y := \alpha X + \beta Y$，X、Y 均为矩阵，它们必须拥有相同大小和分布的行、列映射。
- 函数phgMatVec 计算$y := \alpha\,\mathrm{op}(A)x + \beta y$，$x$、$y$ 为向量，A 为矩阵，函数中的参数op 可取为
  - MAT_OP_N ($\mathrm{op}(A) = A$)、
  - MAT_OP_T ($\mathrm{op}(A) = \mathrm{trans}(A)$)
  - MAT_OP_D ($\mathrm{op}(A) = \mathrm{diag}(A)$)。

  当op 为MAT_OP_N} 时，该函数要求 \verbx-¿map— 与A->cmap 相同，y->map 与A->rmap 相同。而当op 为MAT_OP_T 时，则要求x->map 与A->rmap，y->map 与A->cmap 相同。

Agenda

线性解法器模块

- 内置解法器：PCG, GMRES, AMS 等
- 外部解法器接口：
  - 直接法：MUMPS, SuperLU, SPOOLES 等
  - 迭代法：HYPRE, PETSc, Trilinos/AztecOO, LASPack 等
- Dirichlet边界条件处理
- 通过命令行选项可灵活选择解法器、预条件子组合

示例

```
mpirun -np 64 maxwell -ams_aux_solver trilinos
mpirun -np 64 maxwell -default_solver hypre \
      -hypre_solver pcg -hypre_pc ams
```

未知量及编号 I

PHG的解法器中的未知量就是创建解法器时指定的自由度对象中的自由度。解法器中对未知量有三种编号方式，

局部自由度编号 将所有自由度从0开始按自由度对象及自由度本地编号的顺序依次编排产生的。获取某个特定DOF对象在某个特定单元中的某个自由度的局部自由度编号的接口形式如下：

```
phgSolverMapE2L(SOLVER *solver, int dof_no,
                ELEMENT *e, int index)
```

局部向量编号 一个进程中，存储在本地的向量分量从0开始的顺序编号

全局向量编号 局部向量编号+本进程中的第一个向量分量的全局向量编号

未知量及编号 II

用户接口函数

1. 调用phgSolverCreate 创建解法器对象，其中需要指定做为未知量的一组DOF 对象，PHG 根据这些DOF 对象建立从DOF 到线性系统解向量间的映射关系。
2. 调用phgSolverAddMatrixEntry 或phgSolverAddMatrixEntries 向线性系统中添加矩阵元素，调用phgSolverAddRHSEntry 或 phgSolverAddRHSEntries 添加右端项。PHG的初始矩阵及右端项均为0，这些函数将新的元素累加到矩阵或右端项上。必要时，也可使用相应的基于全局向量编号的接口函数。
3. (可选) 调用phgSolverAssemble 完成线性系统的组装。
4. 调用phgSolverSolve 求解线性系统，其中需要提供一组DOF 对象，它们的类型、维数必须与创建解法器对象时提供的DOF 对象完全匹配，求解前包含初始近似解，求解完成后返回最终得到的解。

线性解法器用户接口函数 I

PHG提供的常用解法器接口函数有：

创建解法器对象

```
SOLVER *phgSolverCreate(OEM_SOLVER *oem_solver,
                        DOF *u, ...)
```

销毁解法器对象，释放占用的资源

```
int phgSolverDestroy(SOLVER *solver)
```

线性解法器用户接口函数 II

累加线性方程组系数矩阵

```
int phgSolverAddMatrixEntry(SOLVER *solver,
                            INT row,INT col,FLOAT value)
int phgSolverAddMatrixEntries(SOLVER *solver,
                              INT nrows, INT *rows,
                              INT ncols, INT *cols,
                              FLOAT *values)
```

线性解法器用户接口函数 III

累加线性方程组右端项

```
int phgSolverAddRHSEntry(SOLVER *solver, INT index,
                         FLOAT value)
int phgSolverAddRHSEntries(SOLVER *solver, INT n,
                         INT *indices, FLOAT *values)
```

求解线性方程组

```
int phgSolverSolve(SOLVER *solver,
                   BOOLEAN destroy, DOF *u, ...)
```

特征值、特征向量计算

PHG的特征值问题解法器接口：PARPACK, JDBSYM, LOBPCG, PRIMME, SLEPc, Trilinos 等。

相关接口

```
int phgEigenSolve(MAT *A, MAT *B, int n,
                  int which, FLOAT tau,
                  FLOAT *evals, VEC **evecs,
                  int *nit);

int phgMatEigenSolve(MAT *A, MAT *B, int n,
                  int which, FLOAT tau, int *nit,
                  FLOAT *evals, MAP *map, DOF **u,
                  ...);
```

Agenda

参数控制及命令行选项 I

运行程序时，可以通过命令行选项来设定PHG 的主要参数，并且可以在程序运行灵活地设置任意参数。PHG提供数百个命令行选项用于内部参数的控制，同时，提供方便的接口供用户定义自己的命令行选项。

可方便地将一组命令行选项放在一个文件中来作为参数文件的使用。

示例

```
mpirun -np 8 poisson -tol 1e-4 -fpetrap none
mpirun -np 8 eigen -eigen_solver arpack \
      -arpack_solver mumps
mpirun -np 8 maxwell -options_file phg-asm.options
```

参数控制及命令行选项 II

PHG 的程序可以通过命令行选项的形式来设定运行参数。PHG 的命令行选项根据其来源及定义方式可以分成三类：

1. PHG 内部定义的选项
2. PHG 调用的第三方软件(如PETSc)所提供的选项
3. 用户程序自行定义的选项

其中，前两类选项对任何PHG 程序都是通用的，而第三类选项则取决于用户程序。

参数控制及命令行选项 III

用户除可以直接在命令行上给出选项外,也可以将一组选项放在一个文件, 然后在命令行中用“-options_file 文件名” 来调入文件中的选项，还可以在phgInit之前调用phgOptionsPreset函数来定义一些预置的选项，用户还可以通过环境变量PHG_OPTIONS来设定选项。

用各种形式给出的命令行选项的处理顺序如下，同一选项多次出现时以最后一次的值为准：

1. 用phgOptionsPreset 函数指定的选项
2. 环境变量PHG_OPTIONS给出的选项
3. 文件“可执行文件名.options” 中给出的选项
4. 命令行中给出的选项

参数控制及命令行选项 IV

PHG 提供一组函数phgOptionsRegisterXXXX，供用户程序添加自己的命令行选项。这些函数的参数中需要提供一个全局或静态变量的地址，当运行程序的命令行中给出相应的选项时，PHG 会相应修改该变量，使之对应于参数的值。

注意，用户自定义选项必须在调用phgInit 之前进行。

```
phgOptionsRegisterFloat("a", "Coefficient", &a);
phgOptionsRegisterInt("mem_max", "Max Mem(MB)", &mem_max);
phgOptionsRegisterNoArg("dump_rhs", "Dump RHS", &dump_rhs);
phgOptionsRegisterFilename("mesh_file", "Mesh file",
                           (char **)&fn);
```

Agenda

网格导出及可视化

- 采用MPI–2并行I/O输出网格及任意自由度数据
- 可输出VTK 和OpenDX 格式以进行可视化处理
- 利用Tcl/Tk 提供脚本语言控制功能及与VTK的直接接口，可实现GUI及交互式可视化
- 软件：Paraview, VisIt, MayaVi, OpenDX

相关接口

```
phgExportALBERT(g, "mesh.dat");
phgExportMedit(g, "mesh.mesh");
phgExportVTK(g, "mesh.vtk", E, curl_E, NULL);
phgExportDX(g, "mesh.dx"", E, curl_E, NULL);
```

Agenda

程序实例

simplest.c	Poisson方程，2阶Lgrange元
poisson.c	Poisson方程，任意H1元
eigen.c	Laplace算子的特征值和特征向量
heat.c	热传导方程
non-smooth.c	二阶间断系数椭圆型问题
elastic.c	弹性力学方程
maxwell.c	时谐Maxwell方程，任意阶棱单元
maxwell-eigen.c	时谐Maxwell方程，特征值问题

程序实例 I

`examples/simplest.c` 是PHG 中最简单的自适应有限元计算程序实例。它求解下述Dirichlet边界条件Poisson 方程：

$$\begin{cases} -\Delta u = f & x \in \Omega \\ u = g & x \in \partial\Omega \end{cases} \tag{1}$$

- 主程序变量`g`为网格对象。DOF 对象`u_h` 和`f_h` 分别存放数值解和右端函数，类型为`DOF_DEFAULT` (默认情况下为`DOF_P2`, 即2 阶Lagrange 元，可在运行程序时通过命令行选项`-dof_type` 设定为其它类型)。`grad_u` 用于计算和保存数值解的梯度。`error` 用于保存误差指示子，类型为`DOF_P0` (分片常数)。

```
GRID *g;
DOF *u_h, *f_h, *grad_u, *error;
SOLVER *solver;

phgInit(&argc, &argv);                    /* 初始化 PHG */
```

程序实例 II

```
g = phgNewGrid(-1);                                  /* 创建网格对象 */
phgImport(g, "cube.dat", FALSE);                     /* 导入网格文件 */
u_h = phgDofNew(g, DOF_DEFAULT, 1, "u_h", DofInterpolation);
phgDofSetDataByValue(u_h, 0.0);
f_h = phgDofNew(g, DOF_DEFAULT, 1, "f_h",  func_f);
error = phgDofNew(g, DOF_P0, 1, "error indicator", DofNoAction);
while (TRUE) {                                       /* 自适应循环 */
  phgBalanceGrid(g, 1.2, -1);                        /* 调整子网格数目与分布 */
  solver = phgSolverCreate(SOLVER_DEFAULT, u_h, NULL); /* 创建解法器 */
  build_linear_system(solver, u_h, f_h);  /* 形成线性方程组 */
  phgSolverSolve(solver, TRUE, u_h, NULL); /* 求解线性方程组 */
  phgSolverDestroy(&solver);                         /* 注销线性解法器 */
  grad_u = phgDofGradient(u_h, NULL, NULL, NULL); /* 计算数值解梯度 */
  estimate_error(u_h, f_h, grad_u, error);/* 计算误差指示子 */
  phgDofFree(&grad_u);                               /* 注销数值解梯度(不再需要) */
  if (满足终止条件) break;                           /* 判断是否结束计算 */
  phgMarkElements(g,..., est, ...);                  /* 根据误差指示子标注细化单元 */
```

程序实例 III

```
    phgRefineMarkedElements(g);                    /* 网格局部细化 */
  }
  phgFreeGrid(&g);                                 /* 注销网格对象 */
  phgFinalize();                                   /* 退出 PHG */
```

创建自由度对象u_h 时使用参数DofInterpolation，它使得当网格细化或粗化时自动对u_h 进行插值，创建f_h 时指明调用函数func_f() 对其进行赋值，而创建error 时使用了参数DofNoAction，表示不对自由度对象进行任何自动的赋值或插值处理，只是为它开辟相应的存储空间。grad_u 由函数phgDofGradient 创建，每次使用完毕后立即释放。phgSolverCreate(SOLVER_DEFAULT, u_h, NULL) 创建一个以u_h 为未知量的解法器对象，使用的解法器为SOLVER_DEFAULT (默认为PCG，用户可以用命令行选项"-solver" 来指定使用其它解法器)。函数phgSolverCreate用一个可变参数表列出作为未知量的自由度对象，这些自由度对象在解法器中依次编号为0, 1, 等等。

程序实例 IV

- 形成线性方程组

函数build_linear_system()形成线性方程组，它对当前网格的所有单元进行遍历，计算每个单元的单元刚度矩阵和右端项，然后迭加到线性系统中去。对网格中单元的遍历通过宏ForAllElements 来进行。假设线性方程组的系数矩阵为$A(I,J)$，右端向量为$B(I)$，$I,J=0,\ldots,M-1$，M 为未知量个数，则计算过程如下：

```
ForAllElements(g, e) {                  /* 对单元进行遍历 */
    N = DofGetNBas(u_h, e);             /* 单元基函数的个数 */
    for (i = 0; i < N; i++) {
        I = phgSolverMapE2L(solver, 0, e, i);
        type = phgDofGetElementBoundaryType(u_h, e, i);
        if (type & DIRICHLET) {
            将 1.0 累加到 A(I,I);
            将 u 的边界值累加到 B(I);
            continue;
        }
        for (j = 0; j < N; j++) {
```

程序实例 V

```
                J = phgSolverMapE2L(solver, 0, e, j);
                计算 ∫_e ∇φ_i · ∇φ_j 并累加到 A(I,J);
            }
            计算 ∫_e f φ_i 并累加到 B(I);
        }
    }
```

其中，φ_i、φ_j 代表单元e 中的局部基函数。phgSolverMapE2L(solver, 0, e, i)计算solver的自由度对象0 (即u_h) 在单元e中的第i 个未知量在方程组中的局部编号，phgDofGetElementBoundaryType 返回未知量的边界类型。$\int_e \nabla\varphi_i \cdot \nabla\varphi_j$ 的计算调用函数phgQuadGradBasDotGradBas完成，$\int_e f\varphi_i$ 的计算调用函数phgQuadDofTimesBas 完成。程序中利用$\int_e \nabla\varphi_i \cdot \nabla\varphi_j$ 对i、j 的对称性减少计算量。

程序实例 VI

- 计算每个单元上的误差指示子，并存储在DOF 对象error 中。这里采用的误差指示子为：

$$\eta_e^2 = h_e^2 \|\Delta u_h + f_h\|_{0,e}^2 + \sum_{f \in F(e), f \subset \Omega} h_f \|l \nabla u_h \cdot n_f]\|_{0,f}^2$$

其中h_e 为单元e 的直径，$F(e)$ 为e 的面的集合，h_f 为面f 的直径，n_f 为面f 的单位法向量，[·] 表示跨过面的跳量。具体计算代码如下：

```
DOF *jump, *residual;
jump = phgQuadFaceJump(grad_u, DOF_PROJ_DOT, NULL, -1);
            /*  Δu_h  */
residual = phgDofDivergence(grad_u, NULL, NULL, NULL);
phgDofAXPY(1., f_h, &residual);      /*  Δu_h + f_h  */
ForAllElements(g, e) {
    int i;
    FLOAT eta, h;
    FLOAT diam = phgGeomGetDiameter(g, e);
    e->mark = 0;                /* clear refinement mark */
```

程序实例 VII

```
        eta = 0.0;
        /* for each face F compute [grad_u \cdot n] */
        for (i = 0; i < NFace; i++) {
            if (e->bound_type[i] & (DIRICHLET | NEUMANN))
                continue;       /* boundary face */
            h = phgGeomGetFaceDiameter(g, e, i);
            eta += *DofFaceData(jump, e->faces[i]) * h;
        }
        eta = eta*.5 +
             diam*diam*phgQuadDofDotDof(e, residual, residual, -1);
        *DofElementData(error, e->index) = Sqrt(eta);
    }
    phgDofFree(&jump);
```

实系数例子：src

复系数例子：src

常用自适应细化/粗化策略

最大误差策略（Maximum strategy）

给定$\gamma \in (0,1)$选取所有满足以下不等式的单元$t \in \mathcal{T}$标记为细化

$$\eta_t > \gamma \max_{t \in Y} \eta_t$$

误差等分布策略（Equidistribution strategy）

设N_t为$\mathcal{T}$中总的单元数，如果我们假设在所有单元上的误差平均分配，则选取满足以下不等式的单元，标记为细化。

$$\eta_t > \frac{tol}{N_t^{1/p}}$$

保证误差下降策略（Guaranteed error reduction strategy）

给定参数$\theta \in (0,1)$，标记集合$\mathcal{A} \subseteq \mathcal{T}$，使得$\sum_{t \in \mathcal{A}} \eta_t^p \geq (1-\theta)^p \eta^p$