


# OpenFOAM矩阵组装的系统介绍(全)



Hasenoch

Take initiative.

李龙翔、汪洋等 28 人赞同了该文章

已关注

本人最近从源代码入手梳理了一下OpenFOAM中矩阵组装的过程（主要是fvMatrix），希望写一篇详细介绍来帮助大家了解OpenFOAM的底层构造。因为整个系统比较大，内容比较深，一些基础前置知识就不一一再讲了。当然我也在正文开始之前罗列下一些我觉得比较相关的基础知识。

## 1. LduAddressing (OpenFOAM矩阵存储和读取方式):

The Finite Volume Method in Computational Fluid Dynamics 中的200-202页

## 2.RTS (runTimeSelection机制):

OpenFOAM guide/runTimeSelection mechanism

openfoamwiki.net/index.php/OpenFOAM\_guide/runTi...

简单说一句，RTS大概就是通过表驱动的方式将属于同一基类的不同子类的构造函数的指针存储在一个哈希表中，通过相应的key在运行过程中确定并构造。

## 3. OpenFOAM离散格式

OpenFOAM中离散格式主要分为两类一个是fvM一个fvC，前者为隐形离散返回fvMatrix<Type>，后者为显式离散返回相应的场（比如volScalarField）。这个资料太多了，就给我一个我之前整理的离散格式之间的关系（只有空间离散）：

一个图整理空间离散格式，限制器和修正算法

16 赞同 · 0 评论 文章

## 4. OpenFOAM边界条件

这个不细说了，相信大家或多或少都用过或者自己写过。

正文开始，我们以下面对压力的求解具体介绍OpenFOAM中矩阵是如何被构造的：

```
fvScalarMatrix pEqn
(
    fvM::laplacian(rAU, p) == fvc::div(phiHbyA)
); //构建矩阵

pEqn.setReference(pRefCell, pRefValue);

pEqn.solve(mesh.solver(p.select(piso.finalInnerIter()))); //求解矩阵
```

下面的介绍将依靠本人自己画的一张关系图（fvM::laplacian(rAU, p), pEqn.solve()和BoundaryCondition的关系）展开：



首先矩阵有哪些内容构建起来呢？比较容易想到的肯定有fvM和fvC的离散项，Su，Sp，fvOptions等源项。但有一个比较大的问题，边界场是怎么影响矩阵构建的呢？我们都知道边界场可以通过下面函数来获取：

```
template<class Type, template<class> class PatchField, class GeoMesh>
inline const typename Foam::GeometricField<Type, PatchField, GeoMesh>::
Boundary&
Foam::GeometricField<Type, PatchField, GeoMesh>::boundaryField() const
{
    return boundaryField_;
}
```

但是它的类型既不是GeometricField<Type, PatchField, GeoMesh>也不是fvMatrix<Type>，那么他是怎么影响矩阵构建的呢？这里先给出答案：边界场是通过相应的四个Field<Type>：

1. valueInternalCoeffs()
2. valueBoundaryCoeffs()
3. gradientInternalCoeffs()
4. gradientBoundaryCoeffs()

参与到fvM和fvC中internalCoeffs()和boundaryCoeffs()的计算，最终在solve函数中作用于fvMatrix的diagonal和source项。

上述四项根据命名法我们可以大致理解其含义，value意味着作用于对流项（convection term），gradient作用于扩散项；Internal最后参与InternalCoeffs的构建（也就是最后作用于fvMatrix<Type>的对角项），Boundary最后参与boundaryCoeffs的构建（也就是最后作用于fvMatrix<Type>的source项。

如图zeroGradient边界场通过evaluate()函数得到（这是内部场更准确的说是临近边界的单元对边界场（patchInternalField()）的影响）。而边界上的gradientInternalCoeffs()和gradientBoundaryCoeffs()最终作用于fvM::laplacian(rAU, p)（事实上上面的gradientInternalCoeffs()和gradientBoundaryCoeffs()都等于零，具体可以查看源代码\$FOAM\_SRC/finiteVolume/fields/fvPatchFields/basic/zeroGradient/zeroGradientFvPatchField.C），如此一来就实现了边界和内部场的相互影响。

接下来是fvM和fvC的离散项，在这里我们以fvM::laplacian(rAU,p)为例，它首先会调用：

```
return fv::laplacianScheme<Type, GType>::New
(
    vf.mesh(),
    vf.mesh().laplacianScheme(name)
).ref().fvMlaplacian(gamma, vf);
```

其中New就是所谓RTS中的选择器，在这里我们不做展开，最后效果为调用了\$FOAM\_SRC/finiteVolume/finiteVolume/laplacianSchemes/gaussLaplacianScheme/gaussLaplacianScheme.H文件中的构造函数：

```
gaussLaplacianScheme(const fvMesh& mesh, Istream& is)
:
    laplacianScheme<Type, GType>(mesh, is)
{
}
```

之后调用\$FOAM\_SRC/finiteVolume/finiteVolume/laplacianSchemes/gaussLaplacianScheme/gaussLaplacianScheme.C中的类成员函数：

```
Foam::tmp<Foam::fvMatrix<Foam::Type>>
Foam::fv::gaussLaplacianScheme<Foam::Type, Foam::scalar>::fvMlaplacian
```

这里需要注意一个细节，事实上在gaussLaplacianScheme.C中也有同名函数：

```
template<class Type, class GType>
tmp<fvMatrix<Type>>
gaussLaplacianScheme<Type, GType>::fvMlaplacian
```

那么为什么OpenFOAM只调用前者呢，这里涉及到模板的偏特性（Partial Template Specialization）。简单说就是C++会优先选取更实例化的函数，OpenFOAM利用这个特性给上述函数瘦身（减少一些不需要的代码的运行（因为这是scalar不用像vector那样考虑多个维度））。

接下来上面函数会对fvMatrix<Type>赋值，具体说就是对主对角（fvM::negSumDiag()），对upper()进行赋值（也就是上三角，因为这里是对称矩阵所以无需对下三角lower()赋值，在调用lower()时fvMatrix<Type>会自动new一个和upper一样的）。而边界的影响会暂时存在internalCoeffs和boundaryCoeffs中。

最后的最后就是矩阵的求解：pEqn.solve()，这里有一些周折（比如会去fvMesh转一圈，当然这里也不展开了，有兴趣的可以自己去翻源码

\$FOAM\_SRC/finiteVolume/fvMatrices/fvMatrix/fvMatrix.C）最后solve函数会调用solveSegregated函数（当非coupled情况下）：

```
template<class Type>
Foam::SolverPerformance<Type> Foam::fvMatrix<Type>::solveSegregated
(
    const dictionary& solverControls
)
```

当然这里也存在上面所说的模板的偏特性（例如fvScalarMatrix），就不再重复了。

如图上述函数主要包括以下几部分：

1. addBoundarySource(source)
2. addBoundaryDiag(diag(), cmpt)
3. solver call
4. psi.correctBoundaryConditions();

其中第一个函数主要是通过之前存下来的boundaryCoeffs来对矩阵的source进行修改：

```
template<class Type>
void Foam::fvMatrix<Type>::addBoundarySource
(
    Field<Type>& source,
    const bool couples
) const
{
    forAll(psi_.boundaryField(), patchi)
    {
        const fvPatchField<Type>& ptf = psi_.boundaryField()[patchi];
        const Field<Type>& pbc = boundaryCoeffs_[patchi];

        if (!ptf.coupled())
        {
            addToInternalField(lduAddr().patchAddr(patchi), pbc, source);
        }
        else if (couples)
        {
            const tmp<Field<Type>> tpnf = ptf.patchNeighbourField();
            const Field<Type>& pnf = tpnf();

            const labelUList& addr = lduAddr().patchAddr(patchi);

            forAll(addr, facei)
            {
                source[addr[facei]] += cmptMultiply(pbc[facei], pnf[facei]);
            }
        }
    }
}
```

第二个函数addBoundaryDiag调用了addToInternalField函数来对矩阵的主对角线来进行修改：

```
template<class Type>
template<class Type2>
void Foam::fvMatrix<Type>::addToInternalField
(
    const labelUList& addr,
    const Field<Type2>& pf,
    Field<Type2>& intf
) const
{
    if (addr.size() != pf.size())
    {
        FatalErrorInFunction
        << "addressing (" << addr.size()
        << ") and field (" << pf.size() << ") are different sizes" << endl
        << abort(FatalError);
    }

    forAll(addr, facei)
    {
        intf[addr[facei]] += pf[facei]; //intf是diag
    }
}
```

第三个函数就是喜闻乐见的矩阵求解器的调用了（这里面又是一个大坑，之前研究了很久的gamg，自己写了新版的gamg求解器发现效率不高就提了issue给社区，大家有兴趣可以看看）

https://develop.openfoam.com/Development/openfoam/-/issues/2054

develop.openfoam.com/Development/openfoam/-/iss...

```
// Solver call
solverPerf = lduMatrix::solver::New
(
    psi.name() + pTraits<Type>::componentNames[cmpt],
    *this,
    bouCoeffsCmpt,
    intCoeffsCmpt,
    interfaces,
    solverControls
)->solve(psiCmpt, sourceCmpt, cmpt);
```

最后一个函数就是correctBoundaryConditions()，至于要额外说这个呢，因为想给写求解器的小伙伴们一个小tips：就是solve()函数自带边界修正，所以当手动修改内场值的时候请手动执行psi.correctBoundaryConditions()哦，不然你就会发现最后结果始终不对或者发散:D

以上就是OpenFOAM矩阵组装和求解的全部内容，其中有些内容参考了知乎上陈与论的文章，但也发现了其中的一些问题并修改过来，而且把其中的debug过程给略去了更加方便大家阅读（当然本人也是debug了一遍）。其实写这个也是为了在以后方便自己翻阅，同时多把知识分享给有需要的人是一件很快乐而且有点成就感的事情。再次谢谢能读到这里的小伙伴们呀~

参考文献：

- 1.cfd-online.com/Forums/o...
2. 陈与论：用GdbOF追踪laplacian函数在OpenFOAM中的定义过程
3. 陈与论：【翻译】OpenFOAM中的空间离散化与矩阵系数
4. OpenFOAMv2012源码，OpenFOAMv2012 API
5. The Finite Volume Method in Computational Fluid Dynamics

祝好!

编辑于 2021-04-23 07:25

openfoam

矩阵计算

计算流体力学 (CFD)

文章被以下专栏收录



OpenFOAM底层与算法

系统介绍openFOAM底层和算法

10 条评论

切换为时间排序

写下你的评论...

扶摇而上 2021-04-22

这都能看懂!! 我感觉读懂of比写一份还难

Hasenoch (作者) 回复 扶摇而上 2021-04-22

大佬很强了 能自己写一份。我也觉得架构是最难的

Micro 2021-08-29

就是solve()函数自带边界修正，所以当手动修改内场值的时候请手动执行psi.correctBoundaryConditions()哦，，，为啥修改内场值的时候需要手动更新边界呢

Hasenoch (作者) 回复 Micro 2021-08-29

比如边界是calculated，那内场值更新了边界也要更新。在极端点，内场网格变了，边界肯定要变化。

Micro 回复 Hasenoch (作者) 2021-08-29

欧欧好的，就是觉得有点反直觉，一般不应该是用边界值决定内场值吗，这里成了，给定内场，去计算边界值

Hasenoch (作者) 回复 Micro 2021-08-29

分情况，有时确实不需要计算的。比如fixedValue就不用更新。但这个总的操作是一直有的，具体指定到每个边界带来的操作不同。对应的，zeroGradient对内场也不会有什么修正（对应的gradientCoefficient都是0），但还都是存在总的函数调用。这两者的关系是，在求解矩阵的时候边界作用于内场，求解完成后根据更新的内场的值在更新边界。

Micro 回复 Hasenoch (作者) 2021-08-29

确实，我自己的问题