非结构代数求解库
Unstructured **A**lgebra **P**ackages(**UNAP**)
简单使用说明

顾寒锋
hanfenggu@gmail.com

2019 年 4 月 24 日

1 UNAP 由来

线性代数求解是各类偏微分方程组求解过程中的重要一环，往往在整个求解过程中占据了绝大部分的时间，其性能的高低直接影响了整个程序的求解效率。同时该模块相对独立，可以作为单独的组件供其他上层应用调用。目前世界上比较广为采用的优秀的代表有美国 Argonne 国家实验室开发的 PETSc，美国 Lawrence Livermore 国家实验室开发的 Hypre 及美国 Sandia 国家实验室开发的 Trilinos。这些求解库均采用了开源形式，但由于开发时间较早，且支持的问题类型众多，代码量已经变得相当庞大，代码结构也是相当复杂，因为通用性牺牲了一定的高效性；采用的语言也较老（普遍为 c 语言），无法方便的进行二次开发和改造；同时这些软件在设计之初主要关注了 MPI 性能，对目前在高性能计算领域普遍采用的异构平台缺乏支持。

因此，考虑到上面几点，我们重新设计、开发了一款面向异构平台的、采用了 C++ 语言的轻量级的非结构代数求解库 UNAP。目前版本支持的求解器主要有：Preconditioned Conjugate Gradient(PCG)，Preconditioned Bi-Conjugate Gradient Stabilized(PBiCGStab) 和 Algebric MultiGrid(AMG)。矩阵格式支持结构对称的 LDU 和 CSR（还在开发中）。

2 UNAP 组成

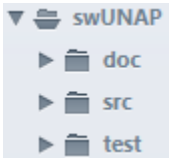

图 1: swUNAP 根目录组成

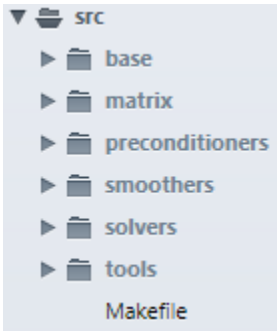

图 2: src 目录组成

UNAP 的根目录如图 1所示。doc 文件夹包含了说明文档，src 文件夹包含 UNAP 的主体代码模块，test 文件夹则包含了一些测试算例和数据。src 目录结构如图 2所示：base 包含了基本的宏、向量操作、MPI 函数和指向多重对象的 PtrList。matrix 文件夹包含了关于矩阵结构的定义和操作，如 LDU、CSR 类型矩阵，矩阵的特征值求解、矩阵在 AMG 的粗化等。preconditioner 文件夹包含了在 Krylov 子空间法中常用的几类预条件子：DIC（求解对称矩阵），DILU（求解非对称矩阵），diagonal（Jacobi 类型迭代）和 MG（多重网格预条件子，还未完善）。smoothers 文件夹主要包含了在 MG 中用到的两类光滑器：Gauss-Seidel 和 Chebyshev。solvers 文件夹下主要包含了前述的几种求解器（PCG，PBiCGStab 和 AMG）。tools 文件夹下包含了一些工具，包括矩阵格式的转换（coo、csr 和 ldu 之间的转换，完成了部分），ldu 类型矩阵的打印输出，从 Hypre 或 OpenFOAM 打印输出矩阵的读取，计时函数、从核代码和与 fortran 的接口等。

3 主要代数求解器使用说明

在用户程序中使用 UNAP 的求解器主要步骤是：

1) 创建矩阵对象，需要的信息有：行数，对角线系数，上三角系数及对应的行、列号，上三角系数的个数，下三角系数（若与上三角系数不一样）

2) 创建求解器对象及对应的需要的组件对象（如 CG 法中的 preconditioner，MG 法中粗层网格算子），后面会分别说明

3) 选择控制参数

4) 调用 solver 的 solve 函数求解

然后编译可执行程序，并链接 libswunap.a(sw) 或 libunap.so(x86)。下面举例说明（可参见 test/ex11.cpp）

3.1 PCG 和 PBiCGStab

这两类方法都属于 Krylov 子空间迭代法，其中 PCG 用来求解对称方程，PBiCGStab 用来求解对称、不对称方程。

```
//- construct PBiCGStab solver by matrix A
lduDiagPrecond precond(lduA);

//- preconditioners
//- DIC
// lduDICPrecond precond(lduA);

//- DILU
// lduDILUPrecond precond(lduA);

//- Diagonal(Jacobi)
PBiCGStab PBiCGStabSolver(precond);

//- solver controls
PBiCGStabSolver.SET_minIter(1);    //- set minimum iteration numbers
PBiCGStabSolver.SET_maxIter(10);   //- set maximum iteration numbers
PBiCGStabSolver.SET_ifPrint(true); //- print information when calculating

//- solve phase
```

```
matrix::solverPerformance solverPerf = PBiCGStabSolver.solve(x, lduA, b);

if(!MYID)
    COUT << "After " << solverPerf.nIterations() << " iterations, the
        solution is converged!" << ENDL;
```

3.2 AMG

可用于求解对称、不对称方程，但对称方程会用到最快梯度下降法，收敛速度比不对称的要快。

```
//- using upper coefficients in matrix as the weights of coarsening in AMG
//- alternative using face areas
scalarField weights(nFaces);
forAll(i, nFaces)
{
    weights[i] = mag(lduA.upper()[i]);
}

//- MG setup phase
//- construct coarse grid using upper coefficients
lduAgglomeration aggl(lduA);
aggl.agglomerate(weights);
PtrList<matrix::smoother> sm(aggl.size());

//- using Gauss-Seidel smoother
//- be noted that GS is not compatible with MLB
// forAll(i, aggl.size())
// {
//     lduGaussSeidelSmoother* smLocPtr = new lduGaussSeidelSmoother;
//     sm.setLevel(i, *smLocPtr);
// }

//- using Chebyshev smoother
forAll(i, aggl.size())
{
    chebySmoother* smLocPtr = new chebySmoother;
    sm.setLevel(i, *smLocPtr);
}

//- construct AMG solver
MGSolver MG(lduA, aggl, sm);
```

```
33 //- this part will using MLB to reorder matrix, b and x
   #ifdef SW_SLAVE
35 lduA.constructMLBIterator();
   lduA.reorderVector(b);
37 lduA.reorderVector(x);
   aggl.agglomerationReorderTopo();
39 lduA.reorderLDUValues();
   #endif
41
   //- MG controls
43 MG.SET_tolerance(tol);  //- set absolute tolerance
   MG.SET_relTol(relTol);  //- set relative tolerance
45 MG.SET_nPreSweeps(1);   //- pre-smooth numbers in V-cycle
   MG.SET_maxIter(15);     //- maximum iteration numbers
47 MG.SET_ifPrint(true);   //- print information when calculating

49 #ifdef SWTIMER
   swTimer::startTimer("MG Solve");
51 #endif
   //- solve phase
53 matrix::solverPerformance solverPerf = MG.solve(x, lduA, b);

55 #ifdef SW_SLAVE
   lduA.restoreVector(x);
57 #endif

59 #ifdef SWTIMER
   swTimer::endTimer("MG Solve");
61 #endif

63 //- print iteration numbers
   if(!MYID)
65    COUT << "After " << solverPerf.nIterations() << " iterations, the
          solution is converged!" << ENDL;
```

4 编译说明

目前版本支持 x86 和 sw 下分别编译，通过在 make 时定义 PLA=x86 或 PLA=sw 来选择。sw 版本又分别支持主核和从核版本，通过在 makefile 中控制“-DSW_SLAVE”来进行有条件编译。

4.1 编译 lib

在根目录或 src 目录下执行 make all

4.2 编译测试算例

在 test 目录下执行./swSub exName nProcs 或./x86Sub exName nProcs。exName 是测试例子的名字，如要测试例子 ex11.cpp，则 exName=ex11，测试例子 ex12f.f90，则 exName=ex12f，注意 fortran 程序名字结尾带有 f。nProcs 是运行的进程数，每个算例可以运行的进程数请参看其对应的测试数据包含几个进程，如 ex11.cpp 中使用了 exData/openfoam/cavity/20w 目录下的数据，共有 1、2、4、16 个进程可供选择。

4.3 一键编译

在根目录下执行./swSub exName nProcs 或./x86Sub exName nProcs，会同时编译 lib 文件。其他同 4.2。

5 其它