附录 A　并行程序开发工具与高性能程序库

本附录介绍几个高性能计算的基础开源软件。限于篇幅，这里仅限于简单介绍每个软件的功能、特点和基本用法，使读者对它们有一个基本的了解，在实际应用中能够有效地利用它们来完成特定的工作。本附录的内容主要依据相关软件的使用手册编写，细节内容请在具体使用时参阅它们所附带的资料。

A.1　BLAS

BLAS (Basic Linear Algebra Subroutines) 是一组高质量的基本向量、矩阵运算子程序。最早的 BLAS 是一组 Fortran 函数和子程序，后来又发展了其他语言接口，包括 C、Java 等。BLAS 的官方网址在

`http://www.netlib.org/blas/`

国内镜像为

`http://netlib.amss.ac.cn/blas/`。

由于 BLAS 涉及最基本的向量、矩阵运算，因此在程序中合理地调用 BLAS 子程序，并且在不同平台上选用经过特殊优化的 BLAS 库可以大大提高程序的性能。BLAS 的主要贡献是将高性能代数计算程序的开发同针对特定机器的性能优化独立开来：代数算法程序的开发者只需要运用适当的分块技术将计算过程变成矩阵、向量的基本运算并调用相应的 BLAS 子程序而不必考虑与计算机体系结构相关的性能优化问题 (后者往往是非常繁杂的)，而针对不同平台的优化 BLAS 库的开发则由计算机厂商和专业开发人员来完成。这一模式大提高了高性能代数程序的开发效率。线性代数软件包如 LAPACK、ScaLAPACK 等都是基于这一思想设计的。

对于 BLAS 库，现在有多种不同的优化实现，适用于 Intel/Linux 平台的主要有以下几种：

BLAS 参考实现

这是一组标准 Fortran 子程序，可以从 BLAS 的主页下载：`http://www.netlib.org/blas/index.html`；

ATLAS 库 (Automatically Tuned Linear Algebra Software)

它可以在不同平台上自动生成优化的 BLAS 库，其主页为 `http://math-atlas.sourceforge.net/`；

Goto 库

Kazushige Goto 开发的一套高性能 BLAS 库，其主页为 `http://www.cs.utexas.edu/users/flame/goto/`；

MKL 库 (Math Kernel Library)

Intel 为自己的 CPU 专门优化的基本数学运算库，其中包含 BLAS 库，其主页为 `http://www.intel.com/cd/software/products/asmo-na/eng/perflib/mkl/index.htm`。

前三种库可以免费下载，而 Intel MKL 库是商业软件，对于商业应用需要购买，而非商业应用可以免费使用。

BLAS 从结构上分成三个层次：Level 1 BLAS、Level 2 BLAS 和 Level 3 BLAS。其中 Level 1 BLAS 涉及向量和向量、向量和标量间的运算，Level 2 BLAS 涉及向量和矩阵间的运算，Level 3 BLAS 则涉及矩阵和矩阵间的运算。此外，还有一个辅助子程序 `XERBLA` 用于错误信息的打印。通常在优化 BLAS 库中，层次越高的子程序性能改善越大，例如，许多平台上优化 BLAS 库中的矩阵乘子程序 `DGEMM` 相对于它的标准 Fortran 版本 `dgemm.f` 的性能提升可接近 10 倍甚至

更多。因此，使用 BLAS 库的一个基本原则是：尽可能地使用 Level 3 BLAS 中的子程序，其次是 Level 2 BLAS 中的子程序。

BLAS 支持四种浮点数格式：单精度实数 (REAL)、双精度实数 (DOUBLE PRECISION)、单精度复数 (COMPLEX) 和双精度复数 (DOUBLE COMPLEX 或 COMPLEX*16)。BLAS 的子程序名中首字母表示浮点数类型：“S” 为单精度实数、“D” 为双精度实数、“C” 为单精度复数、“Z” 为双精度复数。下面的介绍中将主要以双精度实数子程序为例介绍，其他类型的子程序只需要将子程序名中的首字母 “D” 相应地换成 “S”、“C” 或 “Z” 即可。

A.1.1 Level 1 BLAS

Level 1 BLAS 包含一组标量与向量、向量与向量运算的子程序 [24]。

- 向量内积与模

$$x^{\mathrm{T}}y, \quad x^{\mathrm{H}}y, \quad \|x\|_2, \quad \|x\|_1, \quad \dots$$

相关的子程序有 DDOT，DDOTU，DDOTC，DNRM2，DASUM 等。

- 向量、标量运算

$$x := \alpha x, \quad y := x, \quad x \text{ 与 } y \text{ 交换}, \quad y := \alpha x + y$$

相关的子程序有 DSCAL，DCOPY，DSWAP，DAXPY。

- 平面旋转变换

相关的子程序有 DROT，DROTG，DROTM 和 DROTMG。

关于这些子程序的详细说明请参看有关文档或它们的 Fortran 源程序。

A.1.2 Level 2 BLAS

Level 2 BLAS 包含下面几类矩阵、向量运算子程序 [25]：

矩阵乘向量

有下面几种形式

$$y := \alpha Ax + \beta y, \quad y := \alpha A^{\mathrm{T}}x + \beta y, \quad y := \alpha \bar{A}^{\mathrm{T}}x + \beta y$$

其中 α 和 β 代表标量，x 和 y 代表向量，A 代表矩阵。A 可以是普通矩阵、对称 (Hermitian) 矩阵、带状矩阵或 (上或下) 三角矩阵。

秩 1、秩 2 修正

有下面几种形式

$$A := \alpha xy^{\mathrm{T}} + A, \quad A := \alpha x\bar{y}^{\mathrm{T}} + A,$$
$$H := \alpha x\bar{x}^{\mathrm{T}} + H, \quad H := \alpha x\bar{y}^{\mathrm{T}} + \bar{\alpha}y\bar{x}^{\mathrm{T}} + H$$

其中 H 代表 Hermitian 矩阵。

三角方程组求解

有下面几种形式

$$x := T^{-1}x, \quad x := T^{-T}x, \quad x := \bar{T}^{-T}x$$

其中 T 代表非奇异三角矩阵。

Level 2 BLAS 子程序名称中最后一个或两个字母表示运算类型：“MV” 表示矩阵乘向量 (Matrix 乘 Vector)，“R” 表示秩 1 修正，“R2” 表示秩 2 修正，“SV” 表示解线性方程组；中间两个字母表示矩阵类型：“GE” 表示普通矩阵，“GB” 表示普通带状矩阵，“HE” 表示 Hermitian 矩阵，“SY” 表示对称矩阵，“HP” 表示压缩存储的

Hermitian 矩阵，“SP” 表示压缩存储的对称矩阵，“HB” 表示带状 Hermitian 矩阵，“SB” 表示带状对称矩阵，“TR” 表示三角矩阵，“TP” 表示压缩存储的三角矩阵，“TB” 表示带状三角矩阵。例如，子程序 DGEMV 计算普通矩阵乘以向量，而子程序 DTRSV 求解普通三角线性方程组。

对于对称或 Hermitian 矩阵而言，BLAS 只使用它们的上三角或下三角部分，具体由参数 UPLO 指定。BLAS 允许两种存储格式，第一种格式按普通形式存储在一个二维数组中，第二种格式称为“压缩存储”格式，它将矩阵的行或列压缩存储在一个一维数组中：如果存储的是上三角部分 (UPLO = 'U')，则顺序存储矩阵的列，如果存储的是下三角部分 (UPLO = 'L')，则顺序存储矩阵的行。此外，由于 Hermitian 矩阵的对角线元素总是实数，BLAS 不使用它们的虚部。

对于三角矩阵，参数 UPLO 指定是上三角还是下三角矩阵。在压缩存储格式中三角矩阵总是按列依次存储。

其他存储格式 (如带状矩阵) 以及各子程序的详细说明请自行参看有关文档或它们的 Fortran 源程序。

A.1.3 Level 3 BLAS

Level 3 BLAS 由下面几类矩阵运算子程序构成 [26]：

矩阵乘积

包括下面几种形式：

$$C := \alpha AB + \beta C, \quad C := \alpha A^{\mathrm{T}} B + \beta C,$$
$$C := \alpha AB^{\mathrm{T}} + \beta C, \quad C := \alpha A^{\mathrm{T}} B^{\mathrm{T}} + \beta C$$

其中 α 和 β 为标量，A，B，C 为矩阵。

对称矩阵秩 k、秩 $2k$ 修正

有下面几类运算：

$$C := \alpha AA^{\mathrm{T}} + \beta C, \quad C := \alpha A^{\mathrm{T}}A + \beta C,$$
$$C := \alpha AB^{\mathrm{T}} + \alpha BA^{\mathrm{T}} + \beta C,$$
$$C := \alpha A^{\mathrm{T}}B + \alpha B^{\mathrm{T}}A + \beta C$$

其中 C 为对称矩阵。

矩阵与三角矩阵的乘积

有下面几类运算：

$$B := \alpha TB, \quad B := \alpha T^{\mathrm{T}}B, \quad B := \alpha BT, \quad B := \alpha BT^{\mathrm{T}}$$

其中 T 为三角矩阵。

求解含多个右端项的三角线性方程组

有下面几类运算：

$$B := \alpha T^{-1}B, \quad B := \alpha T^{-T}B, \quad B := \alpha BT^{-1}, \quad B := \alpha BT^{-T}$$

此外，上述包含矩阵转置的运算中对复矩阵还提供了相应的共轭转置运算，例如 $C := \alpha AA^{\mathrm{H}} + \beta C$。

Level 3 BLAS 子程序的命名规则与 Level 2 BLAS 类似：子程序名的最后几个字母用于表明矩阵运算的类型，“MM” 表示矩阵乘积 (Matrix 乘 Matrix)，“RK” 表示秩 k 修正，“R2K” 表示秩 $2k$ 修正，“SM” 表示解 (三角) 方程；中间两个字母表示矩阵的类型：“GE” 表示普通矩阵，“SY” 表示对称矩阵，“HE” 表示 Hermitian 矩阵，“TR” 表示三角矩阵。例如，DGEMM 表示普通矩阵乘积，而 DSYMM 表示对称矩阵乘积。

关于这些子程序的更详细的信息请自行参看有关文档或者它们的 Fortran 源程序。

A.2 LAPACK

LAPACK (Linear Algebra PACKage) 是由 Argonne 国家实验室、Courant 研究院和 NAG (Numerical Algorithms Group) 公司联合开发完成的线性代数函数库。LAPACK V1.0 发布于 1992 年 2 月，自 1994 年 9 月 V2.0 版发布以来受到广泛关注。1999 年 6 月发布了 V3.0 版，之后同年 10 月和 2000 年 5 月又分别发布了更新的 V3.0 版。LAPACK 的网址在 `http://www.netlib.org/lapack/`，国内镜像为 `http://netlib.amss.ac.cn/lapack/`。

LAPACK 包含了求解科学与工程计算中最常见的数值线性代数计算问题，如线性方程组、线性最小二乘问题、特征值问题和奇异值问题等。LAPACK 还可以实现矩阵分解和条件数估计等相关计算。

LAPACK 项目的最初目标是在共享存储向量并行计算机上高效地使用 EISPACK 和 LINPACK。由于 LINPACK 和 EISPACK 忽视了微处理器的多层存储结构的特点，以向量操作的形式调用 Level 1 BLAS 中的子程序完成基本运算，使得 Cache 利用率很低，处理器大部分时间花在从内存中存取数据而不是进行浮点运算，因而效率低下。LAPACK 利用分块技术解决了这个问题。其思想是对矩阵进行分块，通过分块将许多操作转换为矩阵运算，主要是矩阵乘法，这些运算调用 Level 3 BLAS 中的高效子程序来完成。此外，把原本较大的工作分为若干较小的部分也有助于提高 Cache 命中率，进一步改善程序的执行效率。移植 LAPACK 时，只要适当调整分块参数，便能使它的许多子程序的实际处理性能接近处理机的峰值性能。

A.2.1 LAPACK 软件包组成

1. 程序分类

在 LAPACK 软件包中，其子程序可以分为三类。它们是：

(1) 驱动程序 (driver routines)：用于解决一个完整问题，例如线性

方程组求解，QR 分解，或求一个实对称矩阵的特征值等。

(2) 计算程序 (computional routines)：也叫作简单 LAPACK 子程序，用以完成一个特定的计算任务，例如一个 $m \times m$ 矩阵的 LU 分解，或把一个普通实矩阵化简为上 Hessenberg 型。

(3) 辅助程序 (auxiliary routines)：是被驱动程序和计算程序调用的子程序。这些程序主要完成对子块的操作和一些常用的底层计算。例如生成初等 Householder 矩阵和计算矩阵范数等。

图 A.1 给出了 LAPACK 软件包的组成结构，其中 `SRC` 是存放源程序代码的目录。LAPACK 软件包中 `TESTING` 子目录下的 `LIN` 子目录存放测试线性系统求解程序正确性的源代码，`EIG` 子目录存放测试特征值问题求解程序正确性的源代码，而 `MAGTEN` 子目录存放生成测试矩阵的源代码。`TIMING` 子目录下的 `LIN` 子目录存放测试线性系统求解程序性能的源代码，而 `EIG` 子目录存放测试特征值问题求解程序性能的源代码。`BLAS` 子目录下的 `SRC` 子目录存放 BLAS 程序的源代码，`TESTING` 子目录存放测试 BLAS 程序正确性的源代码。`INSTALL` 子目录存放安装 LAPACK 软件包所需的 `Makefile`，`make.inc.*` 等文件。

2. 数据类型和精度

除了少数例外，LAPACK 对实数和复数数据类型提供相同的功能。例如对应于求解系数矩阵为实对称矩阵的线性方程组，LAPACK 亦提供程序求解系数矩阵为 Hermitian 矩阵和复型对称矩阵的线性方程组。然而 LAPACK 不提供相当于求解实对称三对角矩阵特征值的复型数据程序，因为 Hermitian 矩阵总是可以规约成实对称三对角矩阵。只要有可能，实型和复型对应的程序的源代码将尽量保持对应。从精度上来说，LAPACK 对所有的程序都提供单精度和双精度两个版本。双精度复型的程序需要机器的 Fortran 77 编译器对

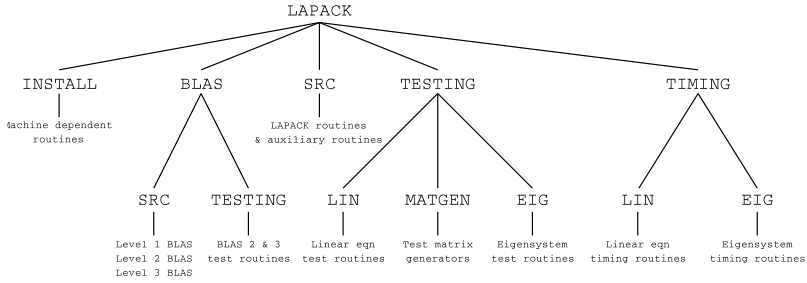

图 A.1 LAPACK 软件包目录结构

COMPLEX*16 数据类型的支持，而这种类型在大多数提供双精度计算的机器上都能支持。

3. 命名规则

LAPACK 的所有驱动和计算程序名称都具有 XYYZZZ 的形式，其中的第一个字母 X 表示程序的数据类型，如下表所示：

X 的取值	所表示的数据类型
S	单精度实型
D	双精度实型
C	单精度复型
Z	双精度复型

当提及某个程序而不管其数据类型时，通常将第一个字母用 “x” 代替。因此，xGETRF 将代表 SGETRF，DGETRF，CGETRF 和 ZGETRF 中的任一个或全部。

接下来的两个字母，YY，表示程序所操作的矩阵类型。这两个字母编码的大多数对实型和复型都适用，但少量的编码只适用于其中一种数据类型。因此当泛指在不同类型的矩阵上完成相同代数操作

的一类程序时，用 “xyy” 代替第一，第二和第三个字母。所以，xyyTRF 指所有进行三角分解的程序。

子程序命名形式的最后三个字母 ZZZ 表示该程序所完成的计算任务。例如，DGEBRD 表示该程序把一个双精度的普通实矩阵化成一个双对角阵。

辅助程序名除第 2 个和第 3 个字母通常是 “LA” 外，与驱动程序和计算程序的命名规则相似，例如 SLASCL，CLARFG。但是有两类例外：

(1) 辅助程序中非分块计算的程序与相应的分块算法的程序名相似，不同点只是前者最后一个字符是 “2”。比如 DGETF2 是与分块三角分解 DGETRF 对应的不分块三角分解程序；

(2) 一些被认为是 BLAS 功能扩充的程序与 BLAS 的命名规则相似，比如 CROT，CSYR 等。

A.2.2 LAPACK 程序文档

每个 LAPACK 程序的文档包含如下内容：

(1) SUBROUTINE 或 FUNCTION 声明以及紧随其后的用以说明参数类型和大小的变量说明部分；

(2) 程序功能的说明；

(3) 按参数序列顺序的参数描述部分；

(4) (可选) 代码段落的说明；

(5) (可选) 内部参数说明。

A.2.3 LAPACK 参数设计

1. 参数顺序

LAPACK 程序中的参数按如下顺序出现：

(1) 选项参数，

(2) 问题规模参数，

(3) 输入数组或标量参数，有些可能被结果覆盖，

(4) 输出数组或标量参数，

(5) 工作数组及相应的规模参数，

(6) 返回信息 (INFO)。

值得注意的是并非每种参数都在一个子程序中出现。

2. 参数说明

下面的例子展示了 LAPACK 中参数的说明格式。

TRANS	(输入) 字符型；“'N'”表示对原矩阵进行操作，“'T'”表示对原矩阵的转置进行操作。
M	(输入) 整型；矩阵 A 的行数；M 或 -M 等于 A 的行数。
A	(输入/输出) 双精度实型；维数为 (LDA, N)；输入为 $A_{m\times n}$；如果 M>=0，输出时 A 被程序 DGEQRF 返回的 QR 分解覆盖，如果 M<0，输出时 A 被程序 DGELQF 返回的 LQ 分解覆盖。
LDA	(输入) 整型；矩阵 A 第一维的大小；LDA>=M。

INFO (输出) 整型；0 表示成功，<0 表示第 -INFO 个参数有违法值，>0 则表示 U(INFO，INFO) 的值为零，即三角分解可以完成但上三角矩阵 U 是奇异的，因而不能用以求解线性方程组。

每个参数的描述按如下顺序排列：

(1) 参数的分类：(输入)，(输出)，(输入/输出)，(输入或输出)，(工作数组)，(工作数组/输出)；

(2) 参数数据类型；

(3) 若参数是数组，其大小；

(4) 该参数所需的或将要提供的数据的描述，或二者都有。对于后者，在描述中以 “On entry” 和 “On exit” 开头；

(5) 若该参数是输入的标量，对其值的限制 (如上例中的 “LDA>=M”)。

3. 选项参数

一些选项的参数是 CHARACTER*1 型的，这样的选项在 LAPACK 中有 SIDE，TRANS，UPLO，DIAG。

SIDE 在调用时的用法如下：

输入值	含义
'L'	在矩阵的左边乘以一个对称或三角阵
'R'	在矩阵的右边乘以一个对称或三角阵

TRANS 在调用时的用法如下：

输入值	含义
'N'	对原矩阵进行操作
'T'	对原矩阵的转置进行操作
'C'	对原矩阵的共轭转置进行操作

UPLO 参数用于 Hermitian、对称和三角阵的情形，用于指示对矩阵的上或下三角进行操作，如下所示：

输入值	含义
'U'	上三角
'L'	下三角

DIAG 在对三角阵进行操作的程序中用于指明该三角阵是否对角线元素为 1，如下所示：

输入值	含义
'U'	单位三角阵
'N'	非单位三角阵

当 DIAG 被指定为 'U' 值时，对应的对角元素将不被引用。

以上参数也可以用小写字母，但任何其他值都是非法的。为了增强程序的可读性，程序员也可以使用更长的字符串，但只有第一个字符有效。例如：

```
CALL DPOTRS('Upper', ...)
```

4. 工作数组

很多 LAPACK 程序需要一个或多个工作数组作为参数。这种数组的名字一般是 WORK，有时可以是 IWORK 和 RWORK，以区别不同的数据类型。紧随其后的是声明工作数组大小的 LWORK、LIWORK 或 LRWORK 参数。工作数组的第一个元素总是返回为了完成计算任务所需的最小空间。若用户提供的工作数组不够大，则程序会赋值给 INFO 并把正确的数组大小存在 WORK(1) 中，最后调用 XERBLA 程序报错。因此建议用户最好每次都检查程序返回的 INFO 值。

若用户对于该工作组需要多大的空间有疑虑，不妨将 LWORK 设为 -1，然后进行调用，再把 WORK(1) 中返回的值作为 LWORK 的正确值。把 LWORK 设为 -1 不会引发任何错误信息，而是被作为一个查询请求来处理。

5. 错误处理

所有程序都返回错误指示信息 INFO，告诉用户计算成功或失败。因而推荐用户每次调用都检查返回的 INFO 值。INFO 值的定义如下所示：

INFO = 0 成功完成计算任务；

INFO < 0 一个或多个参数有错，无法进行计算；

INFO > 0 在计算过程中失败。

如果使用标准的 XERBLA 程序，当程序出错时，LAPACK 将打印一条错误信息，并且当 INFO<0 时终止程序运行，所以 LAPACK 的所有函数通常不会返回 INFO<0。但是，对于非标准的 XERBLA，这种情况则有可能发生。

A.2.4 LAPACK 使用示例

1. 解普通线性方程组

本节演示 LAPACK 计算子程序 DGETRF 和 DGETRS 的调用方法。方程的系数矩阵 A 和右端项矩阵 B 分别按下面的公式初始化：

$$A_{ij} = \sum_{k=1}^{\min(i,j)} (i+j), \qquad B_{ij} = \sum_{k=1}^{\min(i,j)} (1+j)/(i+k)$$

初始化完成后，调用 DGETRF 子程序对该矩阵进行 LU 分解。之后，调用 DGETRS 求解该方程组。

代码 A.1: 解普通线性方程组。

文件名: code/lapack/lapack1.f

```
1       PROGRAM TEST
2  * .. Scalar Arguments ..
3       INTEGER INFO, LDA, LDB, N, NRHS
4       PARAMETER ( N = 500, NRHS = 20, LDA = N, LDB = N )
```

```
* .. Array Arguments ..
      INTEGER IPIV( N )
      DOUBLE PRECISION A( LDA, N ), B( LDB, NRHS )
* .. External Subroutines ..
      EXTERNAL DGETRF, DGETRS
* .. Intrinsic Functions ..
      INTRINSIC MAX

* .. Executable Statements ..
*     Get the value of matrix
*     Matrix values are L = min(i,j), A_ij = sum_{1<=k<=L}(i+j)
      CALL INITMTRA(N, N, A, LDA)
* Compute the LU factorization of A
      CALL DGETRF( N, N, A, LDA, IPIV, INFO )
      IF( INFO.EQ.0 ) THEN
*        Generate the right hand side of linear equations
*        Matrix values are L = min(i,j), B_ij = sum_{1<=k<=L}(1+j)/(i+k)
         CALL INITMTRB(N, NRHS, B, LDB)
*        Solve the system A*X = B, overwriting B with X
         CALL DGETRS( 'No transpose', N, NRHS, A, LDA, IPIV, B, LDB,
     &                INFO )
      END IF
      STOP
      END

************************************************************************
* 初始化矩阵的子程序
************************************************************************

      SUBROUTINE INITMTRA( M, N, A, LDA )
* ..Scalar Arguments..
      INTEGER M, N, LDA
* ..Array Arguments..
      DOUBLE PRECISION A(LDA,*)
* ..Intrinsic Functions..
      INTRINSIC MIN
```

```
41 * ..Local Arguments..
42       INTEGER I, J, K
43
44       DO 30 J = 1, N
45       DO 20 I = 1, M
46          A(I,J) = 0.0
47          DO 10 K = 1, MIN(I,J)
48             A(I,J) = A(I,J) + I + J
49    10    CONTINUE
50    20 CONTINUE
51    30 CONTINUE
52       RETURN
53       END
54
55 ************************************************************************
56
57       SUBROUTINE INITMTRB( M, N, B, LDB )
58 * ..Scalar Arguments..
59       INTEGER M, N, LDB
60 * ..Array Arguments..
61       DOUBLE PRECISION B(LDB,*)
62 * ..Intrinsic Functions..
63       INTRINSIC MIN
64 * ..Local Arguments..
65       INTEGER I, J, K
66
67       DO 30 J = 1, N
68       DO 20 I = 1, M
69          B(I,J) = 0.0
70          DO 10 K = 1, MIN(I,J)
71             B(I,J) = B(I,J) + (1 + J) / (I + K)
72    10    CONTINUE
73    20 CONTINUE
74    30 CONTINUE
75       RETURN
```

```
76      END
```

2. QR **类型矩阵分解**

本节演示 LAPACK 计算子程序 `DGEQRF` 的调用方法。矩阵 `A` 按下面的公式初始化：

$$A_{ij} = \begin{cases} 4 & i = j = 1 \\ 5 & i = j \neq 1 \\ 3 & i = j + 1 \\ 0 & \text{其他} \end{cases}$$

完成初始化后调用 `DGEQRF` 子程序对该矩阵进行 QR 分解。

代码 A.2: QR 类型矩阵分解。

文件名: `code/lapack/lapack2.f`

```
1       PROGRAM TESTQRF
2   * .. Scalar Arguments ..
3       INTEGER INFO, LDA, LWORK, M, N, MN
4       PARAMETER (M = 500, N = 500, LDA = N)
5       PARAMETER (LWORK = N*256, MN = M)
6   * .. Array Arguments ..
7       DOUBLE PRECISION A(LDA, N), TAU(MN), WORK(LWORK)
8   * .. External Subroutines ..
9       EXTERNAL DGEQRF
10
11  * .. Executable Statements ..
12  * Get the value of matrix A
13      CALL INITMTRA(M, N, A, LDA)
14  * Compute QR factorization of A
15      CALL DGEQRF(M, N, A, LDA, TAU, WORK, LWORK, INFO)
16      STOP
17      END
```

```
18
19 ************************************************************************
20 * 初始化矩阵的子程序
21 ************************************************************************
22
23       SUBROUTINE INITMTRA(M, N, A, LDA)
24 * ..Scalar Arguments..
25       INTEGER M, N, LDA
26       DOUBLE PRECISION ZERO, THR, FOUR, FIVE
27       PARAMETER( ZERO = 0.0D0, THR = 3.0D0, FOUR = 4.0D0, FIVE = 5.0D0 )
28 * ..Array Arguments..
29       DOUBLE PRECISION A(LDA, *)
30 * ..Local Arguments..
31       INTEGER I, J
32
33       DO 20 J=1, N
34       DO 10 I=1, M
35          IF( I .EQ. J .AND. I .EQ. 1 )THEN
36             A(I, J) = FOUR
37          ELSE IF( I .EQ. J .AND. I .NE. 1 ) THEN
38             A(I, J) = FIVE
39          ELSE IF( I .EQ. J+1 ) THEN
40             A(I, J) = THR
41          ELSE
42             A(I, J) = ZERO
43          END IF
44    10 CONTINUE
45    20 CONTINUE
46       RETURN
47       END
```

3. 上 Hessenberg 矩阵化简

本节演示 LAPACK 计算子程序 DGEHRD。矩阵 A 的初始化与 QR 类型矩阵分解相同。完成初始化后调用 DGEHRD 子程序将该矩阵化

简为上 Hessenberg 矩阵。

代码 A.3: 上 Hessenberg 矩阵化简。

文件名: code/lapack/lapack3.f

```
      PROGRAM TESTBRD
* .. Scalar Arguments ..
      INTEGER ILO, IHI, INFO, LDA, LWORK, N
      PARAMETER ( N = 500, LDA = N, ILO = 1, IHI = N, LWORK = N*256 )
* .. Array Arguments ..
      DOUBLE PRECISION A( LDA, N ), TAU (N-1), WORK(LWORK)
* .. External Subroutines ..
      EXTERNAL DGEHRD

* .. Executable Statements ..
* Get the value of matrix A
      CALL INITMTRA(M, N, A, LDA)
* Reduce to upper Hessenberg form
      CALL DGEHRD(N, ILO, IHI, A, LDA, TAU, WORK, LWORK, INFO)
      STOP
      END

*************************************************************************
* 初始化矩阵的子程序 (同 QR 分解，略)
*************************************************************************
```

4. 三对角矩阵化简

本节演示 LAPACK 计算子程序 DSYTRD。矩阵 A 的初始化与 QR 类型矩阵分解相同。完成初始化后调用 DSYTRD 子程序将该矩阵化简为三对角矩阵。

代码 A.4: 三对角矩阵化简。

文件名: code/lapack/lapack4.f

```
      PROGRAM TESTTRD
```

```
2  * .. Scalar Arguments ..
3        CHARACTER*1 UPLO
4        INTEGER INFO, LDA, LWORK, N
5        PARAMETER ( UPLO = 'U', N = 500, LDA = N, LWORK = N*256 )
6  * .. Array Arguments ..
7        DOUBLE PRECISION A(LDA, N), D(N), E(N-1), TAU(N-1), WORK(LWORK)
8  * .. External Subroutines ..
9        EXTERNAL DSYTRD
10
11 * .. Executable Statements ..
12 * Get the value of matrix A
13       CALL INITMTRA(M, N, A, LDA)
14 * Call DSYTRD to reduce symmetric matrix to tridiagonal form
15       CALL DSYTRD(UPOL, N, A, LDA, D, E, TAU, WORK, LWORK, INFO)
16       STOP
17       END
18
19 ************************************************************************
20 * 初始化矩阵的子程序 (同 QR 分解，略)
21 ************************************************************************
```

5. 对称特征值问题

本节演示 LAPACK 计算子程序 DSTEQR 和 DSTERF。矩阵 A 按下面公式初始化：

$$\begin{cases} A_{ii} = 1/(i+1) & i = 1, \ldots, N \\ A_{ij} = A_{ji} = 1/(i+j) & i = j+1 \text{ 或 } i = j-1 \\ A_{ij} = 0 & \text{其他} \end{cases}$$

程序中主对角线元素存储在数组 D 中，次对角线元素存储在数组 E 中。完成初始化后首先调用 DSTEQR 子程序求该矩阵的特征值，然后调用 DSTERF 求解特征向量。

代码 A.5: 对称特征值问题。

文件名: code/lapack/lapack5.f

```
      PROGRAM TESTEIG
* .. Scalar Arguments ..
      CHARACTER*1 JOBZ
      INTEGER INFO, LDZ, N, NN
      PARAMETER ( JOBZ = 'V', N = 500, LDZ = N+1, NN = 2*N-2 )
* .. Array Arguments ..
      DOUBLE PRECISION D( N ), E( N ), WORK( NN ), Z( LDZ, N )
* .. Parameters ..
      DOUBLE PRECISION ZERO, ONE
      PARAMETER ( ZERO = 0.0D0, ONE = 1.0D0 )
* .. Local Scalars ..
      LOGICAL WANTZ
      INTEGER IMAX, ISCALE
      DOUBLE PRECISION BIGNUM, EPS, RMAX, RMIN, SAFMIN,
     &                 SIGMA, SMLNUM, TNRM
* .. External Functions ..
      LOGICAL LSAME
      DOUBLE PRECISION DLAMCH, DLANST
      EXTERNAL LSAME, DLAMCH, DLANST
* .. External Subroutines ..
      EXTERNAL DSCAL, DSTEQR, DSTERF
* .. Intrinsic Functions ..
      INTRINSIC SQRT

* .. Executable Statements ..
      WANTZ = LSAME( JOBZ, 'V' )
* Quick return if possible
      IF( N.EQ.0 ) RETURN
      IF( N.EQ.1 ) THEN
         IF( WANTZ ) Z( 1, 1 ) = ONE
         RETURN
      END IF
* Get machine constants
```

```
34        SAFMIN = DLAMCH( 'Safe minimum' )
35        EPS = DLAMCH( 'Precision' )
36        SMLNUM = SAFMIN / EPS
37        BIGNUM = ONE / SMLNUM
38        RMIN = SQRT( SMLNUM )
39        RMAX = SQRT( BIGNUM )
40  * Get the value of matrix A
41        CALL INITMTRA( N, D, E )
42  * Scale matrix to allowable range, if necessary
43        ISCALE = 0
44        TNRM = DLANST( 'M', N, D, E )
45        IF( TNRM.GT.ZERO .AND. TNRM.LT.RMIN ) THEN
46           ISCALE = 1
47           SIGMA = RMIN / TNRM
48        ELSE IF( TNRM.GT.RMAX ) THEN
49           ISCALE = 1
50           SIGMA = RMAX / TNRM
51        END IF
52        IF( ISCALE.EQ.1 ) THEN
53           CALL DSCAL( N, SIGMA, D, 1 )
54           CALL DSCAL( N-1, SIGMA, E( 1 ), 1 )
55        END IF
56  * For eigenvalues only, call DSTERF. For eigenvalues and
57  * eigenvectors, call DSTEQR
58        IF( .NOT.WANTZ ) THEN
59           CALL DSTERF( N, D, E, INFO )
60        ELSE
61           CALL DSTEQR( 'I', N, D, E, Z, LDZ, WORK, INFO )
62        END IF
63  * If matrix was scaled, then rescale eigenvalues appropriately
64        IF( ISCALE.EQ.1 ) THEN
65           IF( INFO.EQ.0 ) THEN
66              IMAX = N
67           ELSE
68              IMAX = INFO - 1
69           END IF
```

```
         CALL DSCAL( IMAX, ONE / SIGMA, D, 1 )
      END IF
      STOP
      END

**********************************************************************
* 初始化矩阵的子程序
**********************************************************************

      SUBROUTINE INITMTRA( N, D, E )
* ..Scalar Arguments..
      INTEGER N
* ..Array Arguments..
      DOUBLE PRECISION D( N ), E( N )
* ..Local Arguments..
      INTEGER I, ONE
      PARAMETER ( ONE = 1 )
* .. Intrinsic Functions ..
      INTRINSIC DBLE

      DO 10 I = ONE, N
         D(I) = DBLE(ONE)/(DBLE(ONE)+I)
   10 CONTINUE
      DO 20 I = ONE, N-1
         E(I) = DBLE(ONE)/(I+J)
   20 CONTINUE
      RETURN
      END
```

A.3 ScaLAPACK

ScaLAPACK (Scalable LAPACK) 是美国能源部 DOE2000 支持开发的 20 多个 ACTS 工具箱之一，由 Oak Ridge 国家实验室、加州大学 Berkeley 分校和 Illinois 大学等联合开发。它是 LAPACK 在分布

式存储环境中的扩展，主要运行在基于分布式存储和消息传递机制的 MIMD 计算机以及支持 PVM 或 MPI 的机群上。LAPACK 是适用于向量超级计算机、共享式存储并行计算机和各种单机上的线性代数运算程序包，ScaLAPACK 实现了其功能的一个子集。ScaLAPACK 的名称来源于 Scalable Linear Algebra PACKage 或 Scalable LAPACK 的缩写。ScaLAPACK 被设计为具有高效率、可移植性、可扩展性、可靠性、灵活性和易用性。

ScaLAPACK 可以求解线性方程组、线性最小二乘问题、特征值和奇异值问题。同时它也可以处理许多相关问题，如矩阵分解和估计条件数。它只适用于稠密矩阵和带状矩阵，对于普通稀疏矩阵不适用。与 LAPACK 类似，ScaLAPACK 也是基于块划分算法以减少进程间的通信。ScaLAPACK 的基本组成部分是 PBLAS (并行 BLAS) 和 BLACS。PBLAS 是 1、2、3 级 BLAS 的分布式存储版本；BLACS 则负责实现并行线性代数计算中常用的通信。在 ScaLAPACK 程序中，多数通信发生在 PBLAS 中，所以 ScaLAPACK 的顶层软件源代码看起来和 LAPACK 相似。

ScaLAPACK 使用基于 SPMD 模型的 Fortran 77 编程，采用显式消息传递进行通信。PBLAS 和 BLACS 使用 C 语言编程，但是具有 Fortran 77 接口。ScaLAPACK 程序提供四个版本，分别对应于单精度和双精度、实数和复数计算。

ScaLAPACK 和 LAPACK 已经成为进行线性代数运算的事实标准而被研究领域和工业界广泛使用，目前已经被集成到许多商业软件包中，包括 NAG Parallel Library，IBM Parallel ESSL，Cray LIBSCI，VNI IMSL Numerical Library，以及 Fujitsu，HP/Convex，Hitachi 和 NEC 计算机的软件库。截止到本书成稿时，ScaLAPACK 的最新版本为 1.7 版，于 2001 年 8 月发布。ScaLAPACK 的网站为

`http://www.netlib.org/scalapack`

国内镜像为

http://netlib.amss.ac.cn/scalapack

A.3.1 ScaLAPACK 体系结构

1. 软件组成

ScaLAPACK 软件的层次结构如图 A.2 所示。虚线下部标注为本地，本地组件在一个进程中被调用，其参数只存储在一个进程中。虚线上部标注为全局，全局组件是同步的并行程序，其参数，包括矩阵和向量，分布在多个进程中。

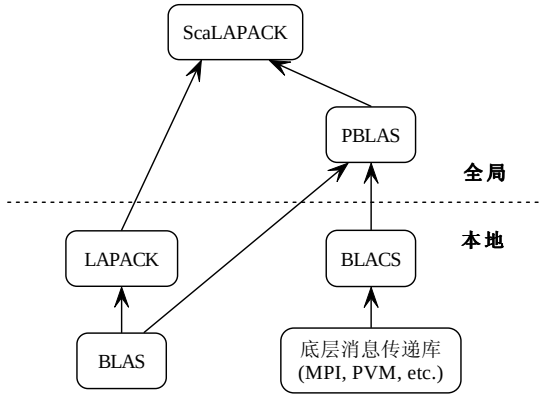

图 A.2 ScaLAPACK 软件的层次结构

LAPACK 线性代数程序库。关于 LAPACK 的介绍可参看 A.2。

BLAS 基本线性代数子程序库。BLAS 的一个重要目标是为基本线性代数计算提供可移植层。关于 BLAS 的介绍可参看 A.1。

PBLAS 并行 BLAS，执行消息传递并且其接口与 BLAS 尽可能相似。它简化了 ScaLAPACK 的设计，使得 ScaLAPACK 的代码与相应的 LAPACK 代码相当接近，有时甚至几乎一样。

BLACS 基本线性代数通信子程序库，是为线性代数设计的消息传递库。计算模型由一个一维或二维进程网格构成，每个进程存储矩阵和向量的一些片段。BLACS 包括点对点通信程序和聚合通信程序，也有构造、改变和查询进程网格的程序。同时，BLACS 具有上下文 (context) 的概念，对应于 MPI 中的通信器 (communicator)，它提供了分隔消息传递域的能力。BLACS 的重要目标是为通信提供专用于线性代数的可移植层。

2. 程序等级

与 LAPACK 类似，ScaLAPACK 中的程序也分为三个大类：

(1) 驱动程序 (driver routines)：用于求解标准类型的问题，例如，求解线性方程组和计算实对称矩阵的特征值。每个驱动程序调用一系列计算程序，并在可能时对程序进行全局和本地输入错误检查。

(2) 计算程序 (computational routines)：用于执行特定的计算任务，例如 LU 分解或将实对称矩阵化简为三对角形式。对程序进行全局和本地输入错误检查。作为整体，计算程序比驱动程序可以承担的任务范围更广。

(3) 辅助程序 (auxiliary routines)：又分为两类：一类是负责完成矩阵、向量分块后子块计算的程序，特别是实现了不分块版本算法的程序；另一类是执行一些常用底层计算的程序，例如缩放矩阵、计算矩阵范数、或生成初等 Householder 矩阵，将来它们会被考虑加入到 PBLAS 中。一般来说，在辅助程序中不进行输入错误检查，其例外是对于等同于第 2 级计算程序的辅助程序进行本地输入错误检查。

3. 命名规则

对于与 LAPACK 相对应的 ScaLAPACK 程序的名称，只是简单地在 LAPACK 名称前面加一个 “P”。因此 Fortran 77 的要求被放松 (破坏) 了，即允许子程序名称长度多于 6 个字符，在特定的 TOOLS 程序名称中还允许出现下划线 “_”。

与 LAPACK 相似，所有驱动程序和计算程序都有 PXYYZZZ 形式的名称，其中有些驱动程序的第七个字母为空。第二个字母 X 表示数据类型；后两个字母 YY 指明矩阵 (或最重要矩阵) 的数据类型；最后三个字母 ZZZ 指明执行的计算。辅助程序的名字遵从相似的规则，只是第三和第四个字母 YY 通常是 LA。

A.3.2 ScaLAPACK 程序介绍

1. 驱动程序：线性方程组求解

求解如下形式的线性方程组：

$$Ax = b \tag{A.1}$$

其中 A 是系数矩阵，b 是右端向量，x 是解向量。如果有多个右端向量，则可以写成

$$AX = B \tag{A.2}$$

其中 B 的列是独立的右端向量，X 的列是相应的解向量。

ScaLAPACK 提供了两类驱动程序用以求解线性方程组：简单驱动（名字以 SV 结尾）和专家驱动 (名字以 SVX 结尾)。简单驱动通过分解 A 求解方程组 $AX = B$，并用 X 覆盖 B。专家驱动还有一些额外的功能，例如：求解 $A^{\mathrm{T}}X = B$ 或 $A^{\mathrm{H}}X = B$；估计 A 的条件数，检查近奇异性，检查主元增长；改进解，计算向前和向后误差范围；在 A 中元素的数量级差别很大时，采用平衡方法来降低其条件数。

为了利用矩阵 A 的特性和存储方案，ScaLAPACK 提供了不同的驱动程序。它们几乎覆盖了线性方程组求解计算的全部功能，除了很少用到的矩阵求逆。

目前，ScaLAPACK 对于涉及带状和三对角矩阵的线性方程组只提供了简单驱动。应当注意的是在这些驱动中使用的带状和三对角分解得到的结果与 LAPACK 中同样分解得到的结果不同。为了并行的目的，ScaLAPACK 在矩阵中进行了额外的置换。

2. 驱动程序：线性最小二乘问题

求解线性最小二乘问题 (LLS)：

$$\min_x \|b - Ax\|_2 \tag{A.3}$$

其中 A 是一个 $m \times n$ 维矩阵，b 是一个给定的 m 维向量，x 是 n 维解向量。

驱动程序 `PxGELS` 在求解时假设 A 是满秩的，即 $\mathrm{rank}(A) = \min(m, n)$，它使用 A 的 QR 或 LU 分解进行计算，并且也可以求解关于 A^{T} 或 A^{H} 的问题。当 $m < n$ 时解是不唯一的，此时程序计算范数最小的解。

所有驱动程序都允许在同一次调用中处理多个右端向量 b 和相应的解 x，将这些向量分别存储为矩阵 B 和 X 的列。方程 (A.3) 对于每个右边向量独立求解，这与找到一个矩阵 X 使 $\|B - AX\|_2$ 最小是不同的。

3. 驱动程序：标准特征值和奇异值问题

对称特征值问题 (SEP) 是要找到特征值 λ 和相应的特征向量 $z \neq 0$，满足 $Az = \lambda z$，其中 A 是实对称矩阵或复 Hermitian 矩阵，对于这种情况 λ 是实数。当所有特征值和特征向量被计算出之后，可以写出 A 的经典谱分解 $A = Z\Lambda Z^{\mathrm{T}}$，其中 Λ 是以特征值为对角元素的对角矩阵，Z 为正交矩阵 (或酉矩阵)，它的列为特征向量。

对于对称或 Hermitian 特征值问题，ScaLAPACK 提供了简单驱动和专家驱动两类程序。简单驱动只能计算所有的特征值和特征向量，而专家驱动则可以选择要计算的特征值的子集和相应的特征向量。

一个 $m \times n$ 维矩阵 A 的奇异值分解 (SVD) 由如下公式给出：

$$A = U\Sigma V^{\mathrm{T}}, \text{(在复数情况下是} A = U\Sigma V^{\mathrm{H}}) \tag{A.4}$$

其中 U 和 V 是正交 (酉) 阵，Σ 是 $m \times n$ 的对角矩阵，具有实对角元素 σ_i，满足 $\sigma_1 \geqslant \sigma_2 \geqslant \ldots \geqslant \sigma_{\min(m,n)} \geqslant 0$。$\sigma_i$ 是 A 的奇异值，U、V 的前 $\min(m,n)$ 列分别是 A 的左、右奇异向量。奇异值和奇异向量满足

$$Av_i = \sigma_i u_i \text{ 以及 } A^{\mathrm{T}}u_i = \sigma_i v_i (\text{或者 } A^{\mathrm{H}}u_i = \sigma_i v_i) \tag{A.5}$$

其中 u_i 和 v_i 分别是 U 和 V 的第 i 列。

4. 驱动程序：广义对称正定特征值问题 (GSEP)

ScaLAPACK 提供了一个专家驱动对于以下类型的问题计算所有特征值和特征向量：

$$Az = \lambda Bz \tag{A.6}$$

$$ABz = \lambda z \tag{A.7}$$

$$BAz = \lambda z \tag{A.8}$$

其中 A 和 B 是对称矩阵或 Hermitian 矩阵并且 B 是正定的。对于上述所有问题特征值 λ 均为实数。

5. 计算程序：线性方程组

求解公式 (A.1) 和 (A.2) 中的联立线性方程组，并假设 A 为 n 阶方阵，但是有些单独的程序允许 A 不是方阵。ScaLAPACK 在可

能的情况下为每种类型矩阵的每个不同存储方案都提供了不同的程序。

6. 计算程序：正交分解和最小二乘问题

ScaLAPACK 提供了许多程序用于分解普通 $m \times n$ 维矩阵 A 为正交矩阵 (对于复数是酉矩阵) 和三角矩阵 (可能是梯形阵) 的乘积，包括 QR 分解、LQ 分解、RQ 分解、QL 分解、RZ 分解等。正交矩阵或酉阵的一个重要特性是它们不改变向量的二范数：$\|x\|_2 = \|Qx\|_2$，因此有助于保持计算的数值稳定性，因为它们不放大舍入误差。正交分解用于线性最小二乘问题的求解，它们也可用于在求解特征值或奇异值问题时将矩阵分解。

7. 计算程序：广义正交分解

ScaLAPACK 提供了两个程序分别求解 $n \times m$ 维矩阵 A 和 $n \times p$ 维矩阵 B 的广义 QR 分解 (GQR) 以及 $m \times n$ 维矩阵 A 和 $p \times n$ 维矩阵 B 的广义 RQ 分解 (GRQ)，它们分别由一对分解给出

$$A = QR \quad 和 \quad B = QTZ \tag{A.9}$$

$$A = RQ \quad 和 \quad B = ZTQ \tag{A.10}$$

其中 Q 和 Z 分别为 $n \times n$ 维和 $p \times p$ 维的正交矩阵（或酉矩阵，如果 A 和 B 为复矩阵）。

8. 计算程序：对称特征值问题

A 是 $n \times n$ 维实对称或复 Hermitian 矩阵。如果 $Az = \lambda z$，λ 为特征值，非零列向量 z 则是相应的特征向量。不论 A 是实对称矩阵还是复 Hermitian 矩阵，λ 总是实数。对称特征问题程序的基本任务是计算给定矩阵 A 的 λ 的值和 (可选的) 相应特征向量 z。

9. 计算程序：非对称特征值问题

A 是 $n \times n$ 维方阵。如果 $Av = \lambda v$，则标量 λ 是特征值，非零列向量 v 是相应的右特征向量，而满足 $u^{\mathrm{H}}A = \lambda u^{\mathrm{H}}$ 的非零列向量 u 是左特征向量。该程序的基本任务是计算给定矩阵 A 的所有 n 个特征值，如果需要，也可以计算相应的右特征向量 v 和左特征向量 u。另一个基本任务是计算矩阵 A 的 Schur 分解。

10. 计算程序：奇异值分解

A 为 $m \times n$ 普通实矩阵。A 的奇异值分解 (SVD) 为，$A = U\Sigma V^{\mathrm{T}}$，其中 U 和 V 为正定阵，$\Sigma = \mathrm{diag}\{\sigma_1, \ldots, \sigma_r\}$，$r = \min(m, n)$，$\sigma_1 \geqslant \ldots \geqslant \sigma_r \geqslant 0$。如果 A 为复数，其奇异值分解为 $A = U\Sigma V^{\mathrm{H}}$，其中 U 和 V 为酉阵，Σ 和前面一样具有实对角元素。σ_i 称为奇异值，V 的前 r 列为右奇异向量，V 的前 r 列为左奇异向量。

11. 计算程序：广义对称正定特征值问题

计算广义特征值问题 $Az = \lambda Bz$，$ABz = \lambda z$，以及 $BAz = \lambda z$，其中 A 和 B 是实对称矩阵或复 Hermitian 矩阵，B 是正定的。这些问题中的每一个都可以用 B 的 Cholesky 分解化简为 $Cy = \lambda y$ 的标准对称特征值问题。

12. 数据分布

ScaLAPACK 要求所有全局数据 (向量或矩阵) 在调用 ScaLAPACK 程序前被分布到相关进程上。应用程序的数据在并行计算机的分级存储器结构中的布局对于决定并行代码的性能和可扩展性是非常关键的。

ScaLAPACK 采用块划分算法，并且尽可能使用面向块的第 3 级 BLAS 矩阵—矩阵运算来实现。这种方法可使浮点操作和内存访问的比例最大化，并尽可能重用存储在分级存储器结构的最高级存

储器中的数据，从而降低了数据在进程间传送的频率，及每次通信的启动开销 (延迟)。

用于求解稠密线性系统和特征值问题的 ScaLAPACK 程序假设所有全局数据都已经使用一维或二维块轮转数据分布方式分布到相关进程上。这种分布正是 ScaLAPACK 使用块划分算法的自然表达。用于求解窄带状线性系统和三对角系统的 ScaLAPACK 程序假设所有全局数据都已经使用一维块数据分布方式分布到进程上。

A.3.3 ScaLAPACK 安装

完整的 ScaLAPACK 软件包可以通过其在 Netlib 上的主页免费获得，也可以通过 WWW 或匿名 FTP 得到。ScaLAPACK 的参考源代码可以通过以下网址：

`http://www.netlib.org/scalapack/scalapack.tgz`

或国内镜像：

`http://netlibamss.ac.cn/scalapack/scalapack.tgz`

获得。用户可以通过编译参考源代码获得自己 ScaLAPACK 程序库。编译的方法将在稍后介绍。

同时，在 Netlib 也有一些已经编译好的 ScaLAPACK 库，分别适用于不同的计算机平台，包括 Cray T3E、Intel Paragon、IBM SP-2、SGI Origin 2000、DEC ALPHA、HP 9000、以及 Intel/Linux 等等。这些预编译好的库可以通过以下网址获得：

`http://www.netlib.org/scalapack/archives/`

`http://netlib.amss.ac.cn/scalapack/archives/`

安装 ScaLAPACK，要求系统中已经安装了 BLAS 和 BLACS 库。其中 BLAS 的安装可以参考 A.1，而 BLACS 则可以从它的主页

`http://www.netlib.org/blacs/index.html`

或国内镜像

`http://netlib.amss.ac.cn/blacs/index.html`

下载针对不同平台的源代码或者已经编译好的库。

获取了参考源代码之后，安装 ScaLAPACK 主要有以下四个步骤：

(1) 解压缩源代码打包文件 `scalapack.tgz`；

(2) 编辑文件 `SLmake.inc`，指定 MPI、BLAS、BLACS 各个库的位置；

(3) 编辑顶层的 `Makefile` 文件，然后键入 `make` 命令进行编译；

(4) 运行测试文件集。

1. 解压缩源代码

在源代码的打包文件中包括了 ScaLAPACK 的源文件、PBLAS 的源文件以及它们的测试文件集。可以用以下命令解压缩打包文件：

```
gunzip -c scalapack.tgz | tar xvf -
```

解压缩之后所有的文件都放入 `SCALAPACK` 目录中，产生的目录结构如图 A.3 所示：

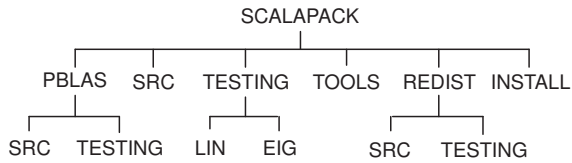

图 A.3 ScaLAPACK 软件的目录

ScaLAPACK 的源代码在 `SRC` 目录中，`PBLAS` 目录下是 PBLAS 的源代码和测试文件集，`TESTING` 目录下是 ScaLAPACK 的测试文件集。

2. 编辑 SLmake.inc 文件

在 SCALAPACK/INSTALL 目录中包含特定平台的 SLmake.inc 文件示例，包括 Intel i860、IBM SP、Cray T3E、SGI Origin、以及使用 MPI 和 PVM 的各种工作站和微机机群。首先将其中相应平台的示例文件拷贝到 SCALAPACK/SLmake.inc 文件中，例如在 Linux 平台上编译 ScaLAPACK，使用如下的命令：

```
$ cp INSTALL/SLmake.LINUX SLmake.inc
```

然后编辑 SLmake.inc，指定以下一些内容：

(1) 指定 ScaLAPACK 顶层目录所在的完整路径，称为 HOME。

(2) 确定需要安装库的平台。

(3) 指定所使用的编译器，链接器，编译、链接选项，库文件打包选项：CC，F77，NOOPT，CCFLAGS，F77FLAGS，LOADER，LOADFLAGS，ARCH，ARCHFLAGS 和 RANLIB。

(4) 指定调试和预处理选项：BLACSDBGLVL 和 CDEFS。BLACSDBGLVL 可以取 0 和 1，它表示 BLACS 的调试级别。CDEFS 可以是 -DAdd_，-DNoChange 或 -DUPCASE。

(5) 指定所需要的库文件的位置，包括：BLACS、MPI 或 PVM 以及 BLAS。

3. 编辑顶层 Makefile 并编译

在顶层目录中已经包括了编译建立库文件和所有测试执行文件的 Makefile 文件，这个文件一般不需要编辑。在编辑好 SLmake.inc 文件后，想要编译生成 ScaLAPACK 库文件，只需要简单地执行以下两个命令：

```
make
```

如果 SLmake.inc 文件中的各项都定义正确，经过几分钟的运行之后，库文件 libscalapack.a 就出现在顶层目录中了。

如果希望建立测试执行文件，可以使用

```
make exe
```

完成后，所有的测试执行文件存放在 TESTING 目录下。

如果只希望建立部分的库或测试执行文件，可以编辑 Makefile 文件，改变 lib 或 exe 的定义。要指定希望使用的数据类型，需要改变 PRECISIONS 的定义。在默认情况下，它的定义如下

```
PRECISIONS = single double complex complex16
```

它表示对于所有数据类型都进行编译，其中，single 指定单精度实型，double 指定双精度实型，complex 指定单精度复型，complex16 指定双精度复型。

当所有测试执行文件都运行完成之后，可以使用

```
make clean
```

移除全部目标文件和测试执行文件。也可以使用 make cleanlib 和 make cleanexe 分别移除库的目标文件或测试程序的目标文件和执行文件。

4. 运行测试文件集

在 ScaLAPACK 的源程序包中带有三类测试程序：PBLAS 测试、REDIST 测试、以及 ScaLAPACK 测试。它们可以用来检查生成的 ScaLAPACK 库的正确性及其性能。每个测试都有一个输入文件，用以指定矩阵规模、分块大小、进程网格尺寸等参数。PBLAS 测试包括了对于全部 3 级 PBLAS 程序的检查和计时；REDIST 测试是对于矩阵在任意进程网格间使用任意分块大小进行二维块轮转重分布的

检查；一共有 18 个 ScaLAPACK 测试程序分别进行 LU、Cholesky、带状 LU、带状 Cholesky、普通三对角、带状三对角、QR（RQ、LQ、QL、QP 和 TZ）、线性最小二乘、上 Hessenberg 化简、三对角化简、双对角化简、矩阵求逆、对称特征值问题、广义对称特征值问题、非对称特征值问题和奇异值问题的计算。

A.3.4 ScaLAPACK 编程指南

1. 调用 ScaLAPACK 程序

在用户自己的程序中调用 ScaLAPACK 程序需要四个基本步骤：

(1) 初始化进程网格

一台抽象的并行计算机的 P 个进程可以表示为一维数组，通常将这个一维数组映射到二维矩形网格可以更方便地表示算法，这个网格称为进程网格。在程序的开始用户需要初始化进程网格，得到默认的系统上下文。用户也可以查询进程网格以识别每个进程的坐标。

调用 ScaLAPACK TOOLS 程序 `SL_INIT` 初始化进程网格。这个程序使用进程行优先排序初始化一个 $P_r \times P_c$（在源代码中以 `NPROW` × `NPCOL` 表示）进程网格，得到默认的系统上下文。用户可以通过调用 `BLACS_GRIDINFO` 查询进程网格以识别每个进程的坐标 (`MYROW`, `MYCOL`)。

完成这项任务的典型代码如下：

```
CALL BLACS_GET( -1, 0, ICTXT )
CALL BLACS_GRIDINIT( ICTXT, 'Row-major', NPROW, NPCOL )
CALL BLACS_GRIDINFO( ICTXT, NPROW, NPCOL, MYROW, MYCOL )
```

其中，`BLACS_GET` 获得默认上下文 `ICTXT`；`BLACS_GRIDINIT` 定

义进程网格，'Row-major' 说明网格是按照行优先的顺序排列的；BLACS_GRIDINFO 获得本进程在进程网格中的位置信息。

(2) 将矩阵分布到进程网格上

在调用 ScaLAPACK 程序之前所有全局矩阵必须分布在进程网格上。执行数据分布是用户的责任。每个要分布在进程网格上的全局矩阵都必须分配一个数组描述符，数组描述符通过调用 ScaLAPACK TOOLS 的程序 DESCINIT 可以很简单地初始化，它必须在调用 ScaLAPACK 程序之前设置。

矩阵的数组描述符用以下代码分配：

```
 CALL DESCINIT( DESCA, M, N, MB, NB, RSRC, CSRC, ICTXT,
$               MXLLDA, INFO )
 CALL DESCINIT( DESCB, N, NRHS, NB, NBRHS, RSRC, CSRC,
$               ICTXT, MXLLDB, INFO )
 CALL DESCINIT( DESCX, N, NRHS, NB, NB, 0, 0, ICTXT,
$               MAX(1, NP), INFO )
```

数组描述符各项的详细描述可以在 [30] 中找到。

之后可以使用如下的代码调用从数据文件中读入矩阵的数据并将其分布到进程网格上：

```
 CALL PDLAREAD( 'SCAEXMAT.dat', MEM(IPA), DESCA, 0, 0,
$               MEM(IPW) )
 CALL PDLAREAD( 'SCAEXRHS.dat', MEM(IPB), DESCB, 0, 0,
$               MEM(IPW) )
```

(3) 调用 ScaLAPACK 程序

当数据正确分布到进程网格上之后，用户就可以调用相应的驱动、计算、辅助程序进行需要的计算。各个程序的调用参数的详细解释可以在 ScaLAPACK 源代码的注释中找到。

下面的例子调用求解线性方程组 $AX = B$ 的简单驱动程序 PDGESV：

```
      CALL PDGESV( N, NRHS, MEM(IPA), 1, 1, DESCA, MEM(IPPIV),
     $             MEM(IPB), 1, 1, DESCB, INFO )
```

(4) 释放进程网络

在进程网格上执行完计算后，通过调用 BLACS_GRIDEXIT 释放进程网格。当所有计算完成后，程序通过调用 BLACS_EXIT 退出。

完成这个步骤的典型代码是：

```
      CALL BLACS_GRIDEXIT( ICTXT )
      CALL BLACS_EXIT( 0 )
```

具体的程序调用请参阅《ScaLAPACK 用户手册》[30]。

在编译链接程序时，需要指定 libscalapack.a 文件的位置，确保程序可以被正确链接。

2. 使用 ScaLAPACK 获得高性能的原则

ScaLAPACK 被设计为在分布式存储计算机上运行。一般来说，分布式存储计算机包括高效的消息传递系统、进程到处理机的一对一映射、成组调度程序和精心链接的通信网络。为了在分布式存储计算机上获得高性能，在编写和运行 ScaLAPACK 程序时，需要遵循以下一些原则：

(1) 使用正确数量的进程

根据经验，使用的进程数量要与所计算的问题规模相适应，[30] 中的建议为进程数量 NP 可以由经验公式

$$\mathtt{NP} = \mathtt{M} \times \mathtt{N}/1000000 \tag{A.11}$$

大致估计，即使得每个进程中本地子矩阵的规模大约为 1000×1000。需要指出的是，这个建议是在 1996–1997 年间基于 ScaLAPACK 1.4 版本的测试而做出的，现在随着处理器速度的提高和存储器的加大，本地子矩阵的规模也需要根据使用 ScaLAPACK 测试程序得到的结果做出相应调整。需要注意的是本地子矩阵不要过小而占用过多的进程使得通信时间加长，也不要将本地子矩阵划分得过大而超过物理存储器的容量。

(2) 使用高效的数据分布

进行数据分布时，使用分块尺寸 `MB` = `NB` = 64，当然这个参数是与使用的 BLAS 库相关的，不同的 BLAS 库在不同的环境中有不同的最佳的分块，在 [34] 中得到的最优分块是 Intel MKL 库的分块大小为 64 的倍数，ATLAS 生成的 BLAS 库的分块大小为 40 的倍数。

进程网格通常要尽量接近正方形。进程网格尺寸由两个参数决定，P 表示水平方向进程数量，Q 表示垂直方向进程数量。一般情况下需要使 $P = Q$ 或者 P 略小于 Q。

(3) 使用对于特定平台优化的 BLAS 和 BLACS 库

BLAS 和 BLACS 是 ScaLAPACK 的两个基本组件，它们的性能决定了 ScaLAPACK 的效率，因此要选择在特定平台上效率最高的 BLAS 和 BLACS。

除了以上这些大的原则之外，还有一些其他的问题需要考虑，包括：保证每结点的带宽，网络的延迟不能过大，使用同构的工作站或微机机群，正在使用的处理机上不要有其他任务，每个处理机上运行的进程数不要多于 CPU 数。其中有些问题是普通用户无法控制的。

A.4 FFTW

FFTW (The Fastest Fourier Transform in the West) 是一个免费的快速富氏变换 (FFT) 软件包，开发者是麻省理工学院的 Matteo Frigo 和 Steven G. Johnson，可从站点 `http://www.fftw.org`下载。该软件包用 C 语言开发，其核心技术 (编码生成器) 采用面向对象设计技术和面向对象语言 Caml 编写。FFTW 能自动适应系统硬件，因而可移植性很强，用户无须对系统干预太多。它能计算一维和多维离散傅立叶变换（Discrete Fourier Transform），其数据类型可以是实型、复型或半复型。该软件通过方案 (plan) 和执行器 (executor) 与用户进行接口，内部结构及其复杂性对用户透明，速度快。内部编译器、代码生成器利用 AST（Abstract Syntax Tree）在运行时生成适合所运行的机器的代码并自我优化。FFTW 为指定的变换生成一个方案，通过执行方案完成变换。它的运算性能远远领先于许多其它 FFT 软件，受到越来越多的科学研究和工程计算工作者的青睐。这里介绍的是 FFTW 3.0.1 版的基本用法，与 FFTW 2 相比，FFTW 3 在调用接口上发生了比较大的变化。除非特别提及，本书中 “FFTW” 一律指 FFTW 3[1]。所用到的材料来源于 `http://www.fftw.org/fftw3_doc/`，目的是给读者提供一个快速入门参考，详细说明请自行参考原始文档。

FFTW 实现了多种类型的变换，包括复型变换、实型变换、sin 变换、cos 变换和 Hartley 变换。在调用接口方面，FFTW 提供了基本接口和高级接口，后者提供了对 FFTW 更精细的控制。对于 Fortran 程序，FFTW 亦提供了相应的 Fortran 接口。这里仅以复型和实型变换函数为例给出 FFTW 的基本 C 接口函数。

[1]FFTW 2 支持共享存储多线程并行和分布式存储 MPI 并行，而 FFTW 3 目前只支持共享存储多线程并行，因此使用 MPI 并行的程序可能仍然需调要用 FFTW 2 的 MPI 函数。

FFTW 默认使用 double 型数据，并将复数定义为如下类型：

```
typedef double fftw_complex[2];
#define c_re(c) ((c)[0])
#define c_im(c) ((c)[1])
```

如果使用符合 C99 标准的 C 编译器，并且在 FFTW 的头文件之前包含了 complex.h 文件，则 fftw_complex 将被定义为 C 的默认复数类型 (如 complex<double>，这种情况下可以直接在代码中使用复数表达式)。

FFTW 的基本使用包括三个步骤：为特定的变换创建方案、(反复) 执行方案完成变换和释放方案。典型调用过程如下 (以三维复变换为例)：

```
#include <fftw3.h>
...
{
    fftw_complex *in, *out;     /* 变换数组 */
    fftw_plan p;                /* 方案 */
    ...
    /* 申请内存及创建方案 */
    in = fftw_malloc(sizeof(fftw_complex) * nx * ny * nz);
    out = fftw_malloc(sizeof(fftw_complex) * nx * ny * nz);
    p = fftw_plan_dft_3d(nx, ny, nz, in, out, FFTW_FORWARD, FFTW_ESTIMATE);
    ...
    /* 实施变换 (可反复执行) */
    fftw_execute(p);
    ...
    /* 释放方案、数组 */
    fftw_destroy_plan(p);
    fftw_free(in);
    fftw_free(out);
}
```

注意这里调用 fftw_malloc 函数为数组申请存储空间，它与 malloc

的区别是前者会根据 FFTW 的要求在需要时调整数组的对界。

A.4.1 复型变换

计算如下公式：

(1) 向前变换：

$$y[i_1, i_2, \ldots, i_d] = \sum_{j_1=0}^{n_1-1} \sum_{j_2=0}^{n_2-1} \cdots \sum_{j_d=0}^{n_d-1} x[j_1, j_2, \ldots, j_d] \omega_1^{-i_1 j_1} \omega_2^{-i_2 j_2} \cdots \omega_d^{-i_d j_d} \tag{A.12}$$

(2) 向后变换：

$$y[i_1, i_2, \ldots, i_d] = \sum_{j_1=0}^{n_1-1} \sum_{j_2=0}^{n_2-1} \cdots \sum_{j_d=0}^{n_d-1} x[j_1, j_2, \ldots, j_d] \omega_1^{i_1 j_1} \omega_2^{i_2 j_2} \cdots \omega_d^{i_d j_d} \tag{A.13}$$

其中 x 是 d 维复型数组，其元素为 $x[j_1, j_2, \ldots, j_d]$，$\omega_s = e^{2\pi\sqrt{-1}/n_s}$，$0 \leqslant j_s < n_s$，$s \in \{1, 2, \ldots, d\}$。$y$ 与 x 的结构一样。

先做一次向前变换，然后再做一次向后变换相当于将数组乘以 $\prod_{s=1}^{d} n_s$。

主要函数如下：

```
fftw_plan fftw_plan_dft_1d(int n,
                           fftw_complex *in, fftw_complex *out,
                           int sign, unsigned flags);
fftw_plan fftw_plan_dft_2d(int nx, int ny,
                           fftw_complex *in, fftw_complex *out,
                           int sign, unsigned flags);
fftw_plan fftw_plan_dft_3d(int nx, int ny, int nz,
                           fftw_complex *in, fftw_complex *out,
```

```
                          int sign, unsigned flags);
fftw_plan fftw_plan_dft   (int rank, const int *n,
                          fftw_complex *in, fftw_complex *out,
                          int sign, unsigned flags);
```

前三个函数分别用于一、二和三维变换，参数 nx (或 n)、ny 和 nz 分别给出 x、y 和 z 方向的变换大小。最后一个函数，fftw_plan_dft，用于任意维变换，参数 rank 给出维数，相当于式 (A.12) 和式 (A.13) 中的 d，而数组 n[] 则给出每维上的变换大小：n_i = n[i]，$i = 1, 2, \ldots, d$。

参数 in 和 out 分别给出输入和输出数组，它们相当于式 (A.12) 和式 (A.13) 中的 x 和 y。除非使用了 FFTW_ESTIMATE 标志，否则这些数组的内容在创建方案时会被破坏。

参数 sign 可以是 FFTW_FORWARD (+1) 或 FFTW_BACKWARD (-1)，用于指定向前还是向后变换。

参数 flags 包含一组按位表示的标志用于指定变换算法细节，常用的有下面一些标志，不同的标志可用 "|" 运算符组合起来使用：

(1) FFTW_ESTIMATE 不通过试算，直接选用一个相对合理的方案，变换性能不一定是最优的。使用该标志创建方案时不会破坏输入/输出数组 in/out 的内容。

(2) FFTW_MEASURE 通过进行几次试算产生一个优化的方案，它是默认的方案创建方式，创建方案的过程会花费一些时间，通常数秒钟。

(3) FFTW_PATIENT 与 FFTW_MEASURE 类似，但测试更多的可能性以求产生一个更好的方案，代价是创建方案需要的时间更长。

(4) FFTW_EXHAUSTIVE 比 FFTW_PATIENT 做更多的测试以求进一步优化变换方案，包括对许多通常并不一定会有效的算法进行测试。

(5) FFTW_DESTROY_INPUT 在执行变换时允许破坏输入数组 in 的内容 (有助于改善变换性能)。

(6) FFTW_PRESERVE_INPUT 在执行变换时不允许破坏输入数组 in 的内容。

函数 fftw_plan_dft_c2r 的默认行为是 FFTW_DESTROY_INPUT，其余函数的默认行为则是 FFTW_PRESERVE_INPUT。对于高维变换，函数 fftw_plan_dft_c2r 中不允许使用标志 FFTW_PRESERVE_INPUT，如果使用了这个标志，则方案创建将会失败 (函数返回空指针 NULL)。

A.4.2 实型变换

FFTW 的实型变换指输入数组为实型数组的情形，变换公式与复型变换一样。对于一维变换而言，假设 $x_0, \ldots, x_{n-1}$ 是一个实型数组，$y_0, \ldots, y_{n-1}$ 是它的 (正向) 离散傅里叶变换结果，则数组 y 满足条件：$y_i = \mathrm{conj}(y_{n-i})$，$0 \leqslant i < n$，这样的数组称为 Hermitian 数组。为了节省内存空间，实型变换函数中 Hermitian 数组采用长度为 $\lfloor n/2 \rfloor + 1$ 的复型数组存储。

高维 Hermitian 数组满足 $y_{i_1,\ldots,i_d} = \mathrm{conj}(y_{n_1-i_1,n_2-i_2,\ldots,n_d-i_d})$，$0 \leqslant i_k < n_k$，$k = 1, 2, \ldots, d$，这里假设数组在所有维上都是周期的，即 $y_{\ldots,n_k,\ldots} = y_{\ldots,0,\ldots}$。FFTW 中高维 Hermitian 数组存储在一个长度为 $n_1 \times \ldots \times n_{d-1} \times (\lfloor n_d/2 \rfloor + 1)$ 的复型数组中，数组最后一维的大小大约是变换大小的一半。

与复型变换不同的是，FFTW 的正向和反向实型变换创建方案时采用不同的函数。这些函数除了比相应的复型变换函数少一个参数 sign 外，其余参数形式和顺序是一样的，但是它们的输入和输出数组一个是实型数组、另一个是 Hermitian 数组。这些变换函数包括：

正向变换

```
fftw_plan fftw_plan_dft_r2c_1d(int n, double *in, fftw_complex *out,
                               unsigned flags);
fftw_plan fftw_plan_dft_r2c_2d(int nx, int ny, double *in, fftw_complex *out,
                               unsigned flags);
fftw_plan fftw_plan_dft_r2c_3d(int nx, int ny, int nz,
                               double *in, fftw_complex *out, unsigned flags);
fftw_plan fftw_plan_dft_r2c(int rank, const int *n,
                            double *in, fftw_complex *out, unsigned flags);
```

反向变换

```
fftw_plan fftw_plan_dft_c2r_1d(int n, fftw_complex *in, double *out,
                               unsigned flags);
fftw_plan fftw_plan_dft_c2r_2d(int nx, int ny, fftw_complex *in, double *out,
                               unsigned flags);
fftw_plan fftw_plan_dft_c2r_3d(int nx, int ny, int nz,
                               fftw_complex *in, double *out, unsigned flags);
fftw_plan fftw_plan_dft_c2r(int rank, const int *n,
                            fftw_complex *in, double *out, unsigned flags);
```

A.4.3 并行 FFTW

在共享内存型 (SMP) 并行计算机上，FFTW 支持多线程并行。用户通过下面几个函数来建立、控制多线程并行：

```
int fftw_init_threads(void);
void fftw_plan_with_nthreads(int nthreads);
void fftw_cleanup_threads(void);
```

其中函数 `fftw_init_threads` 用在所有多线程并行的变换函数之前，其作用是初始化多线程并行；函数 `fftw_plan_with_nthreads` 指定使用的线程数，调用它后，新创建的方案将使用所指定的线程数；函数 `fftw_cleanup_threads` 用在所有多线程变换完成之后，以释放 FFTW 的多线程函数占用的资源。

FFTW 2 中提供了适合于分布式存储并行系统的 MPI 版本，而

FFTW 3 中尚未实现这些功能。如果需要 MPI 并行，目前可以考虑使用 FFTW 2.1.5，也可以通过在部分空间方向上划分数据、并对数据在处理器间进行一次转置来调用 FFTW 3 的串行变换函数实现。

A.4.4 FFTW 计算实例

本节给出一个 1 维复型快速傅里叶变换的完整代码实例。

输入时间序列 $X \in C^n$，通过一维离散傅里叶变换得到输出频谱序列 $Y \in C^n$，即：

$$Y_j = \frac{1}{\sqrt{n}} \sum_{k=0}^{n-1} \omega^{-jk} X_k, \quad j = 0, 1, \ldots, n-1$$

其中 $\omega = e^{2\pi\sqrt{-1}/n}$。将上式写成矩阵形式为

$$Y = WX$$

其中 $W = (\omega^{jk})$ 为 $n \times n$ 阶循环方阵。直接采用矩阵乘向量运算，上式的浮点计算量为 $O(n^2)$。如果运用 FFT 算法计算上式，则浮点计算量降至 $O(n \log n)$。

代码 A.6: FFTW 程序实例。

文件名: `code/fftw/fftw-1d.c`

```
1  #include "fftw3.h"
2  #include <stdio.h>
3  #include <stdlib.h>
4  #include <math.h>
5
6  #define N       4
7  #define REAL    0
8  #define IMAG    1
9  #define PI      3.1415926535898
10
```

```
int main (void)
{
    fftw_complex *in, *out;
    fftw_plan p;
    double constants[N] = {10, 2.1, 4.7, 1.3};
    double f;
    int i, j;

    /* Allocate memory for the arrays */
    in = fftw_malloc(sizeof(fftw_complex) * N);
    out = fftw_malloc(sizeof(fftw_complex) * N);

    if ((in == NULL) || (out == NULL)) {
        printf ("Error: insufficient available memory\n");
    }
    else {
        /* Create the FFTW execution plan */
        p = fftw_plan_dft_1d(N, in, out, FFTW_FORWARD, FFTW_ESTIMATE);

        /* Initialize the input data */
        for (i = 0; i < N; i++) { /* All sampling points */
            in[i][REAL] = constants[0];
            in[i][IMAG] = 0;
            for (j = 1; j < N; j++) { /* All frequencies */
                in[i][REAL] += constants[j] * cos(j * i * 2 * PI / (double)N);
                in[i][IMAG] += constants[j] * sin(j * i * 2 * PI / (double)N);
            }
        }

        /* Execute plan */
        fftw_execute(p);

        /* Destroy plan */
        fftw_destroy_plan(p);

        /* Display results */
```

```
47          printf ("Constants[] = {");
48          for (i = 0; i < N; i++)
49              printf("%lf%s", constants[i], (i == N-1) ? "}\n" : ", ");
50
51          printf ("Input[][REAL] = {");
52          for (i = 0; i < N; i++)
53              printf("%lf%s", in[i][REAL], (i == N-1) ? "}\n" : ", ");
54
55          printf ("Output[][REAL] = {");
56          for (i = 0; i < N; i++)
57              printf("%lf%s", out[i][REAL], (i == N-1) ? "}\n" : ", ");
58
59          /* Scale output */
60          f = 1.0/sqrt((double)N);
61          for (i = 0; i < N; i++)
62              out[i][REAL] *= f;
63
64          /* Display final results */
65          printf ("Scaled[][REAL] = {");
66          for (i = 0; i < N; i++)
67              printf("%lf%s", out[i][REAL], (i == N-1) ? "}\n" : ", ");
68      }
69
70      /* Free allocated memory */
71      if (in != NULL) fftw_free(in);
72      if (out != NULL) fftw_free(out);
73
74      return 0;
75  }
```

A.5 PETSc

PETSc (Portable，Extensible Toolkit for Scientific Computation) 是美国能源部 DOE2000 支持开发的 20 多个 ACTS 工具箱之一，由

Argonne 国家实验室开发的可移植可扩展科学计算工具箱，主要用于在分布式存储环境高效求解偏微分方程组及相关问题。PETSc 所有消息传递通信均采用 MPI 标准实现。

PETSc 用 C 语言开发，遵循面向对象设计的基本特征，用户基于 PETSc 对象可以灵活开发应用程序。目前，PETSc 支持 Fortran 77/90、C 和 C++ 编写的串行和并行代码。

PETSc 是一系列软件和库的集合，三个基本组件 SLES、SNES 和 TS 本身基于 BLAS、LAPACK 和 MPI 等库实现，同时为 TAO、ADIC/ADIFOR、MATLAB、ESI 等工具提供数据接口或互操作功能，并具有极好的可扩展性能。PETSc 为用户提供了丰富的 Krylov 子空间迭代方法和预条件子，并提供错误检测、性能统计和图形打印等功能。

如今，越来越多的应用程序在 PETSc 环境上开发，并逐渐显示出 PETSc 在高效求解大规模数值模拟问题方面的优势和威力。

PETSc 的网站是：`http://www.mcs.anl.gov/petsc`。

A.5.1 PETSc 的系统结构

不同于其他微分/代数方程求解器，PETSc 为用户提供了一个通用的高层应用程序开发平台。基于 PETSc 提供的大量对象和解法库，用户可以灵活地开发自己的应用程序，还可随意添加和完善某些功能，如为线性方程求解提供预条件子、为非线性问题的牛顿迭代求解提供雅可比矩阵、为许多数值应用软件和数学库提供接口等。图 A.4 描述了 PETSc 在实现层次上的抽象。这里做简要说明。

应用程序

用户在 PETSc 环境下基于 PETSc 对象和算法库编写的串行或并行应用程序。尽管 PETSc 完全在 MPI 上实现，但 PETSc 程序具有固定的框架结构，包括初始化、空间释放和运行结束等环境语句。

PDE 求解器

用户基于 PETSc 的三个基本算法库 (TS、SNES 和 SLES) 构建的偏微方程求解器。但它却不是 PETSc 的基本组件。

TS (Time Stepping)

时间步进积分器，用于求解常微分方程 (ODE)，或依赖时间的空间离散化后的偏微分方程 (PDE)。对于非时间演化或稳态方程，PETSc 提供了伪时间步进积分器。TS 积分器最终依赖线性求解器 SLES 和非线性求解器 SNES 来实现。PETSc 为 PVODE 库提供了接口。另外，TS 的用法非常简单方便。

SNES (Nonlinear Solver)

非线性求解器，为大规模非线性问题提供高效的非精确或拟牛顿迭代解法。SNES 调用线性求解器 SLES，并采用线搜索和信赖域方法实现。SNES 依赖于雅可比矩阵求解，PETSc 既支持用户提供的有限差分程序，同时又为用户提供了 ADIC 等自动微分软件生成的微分程序接口。

SLES (Linear Solver)

线性求解器，求解大规模线性方程组，它是 PETSc 的核心部分。PETSc 几乎提供了所有求解线性方程组的高效求解器，既有串行求解也有并行求解，既有直接法求解也有迭代法求解。对于大规模线性方程组，PETSc 提供了大量基于 Krylov 子空间方法和各种预条件子的成熟而有效的迭代方法，以及其它通用程序和用户程序的接口。

KSP (Krylov Subspace)

Krylov 子空间方法，包括 Richardson 方法、共轭梯度法 (CG 和 BiCG)、广义最小残差法 (GMRES)、最小二乘 QR 分解 (LSQR) 等。

PC (Preconditioner)

预条件子，包括雅可比、分块雅可比、SOR / SSOR、不完全 Cholesky 分解、不完全 LU 分解、加性 Schwartz，多重网格等方法。

DRAW

应用程序的性能分析和结果显示。

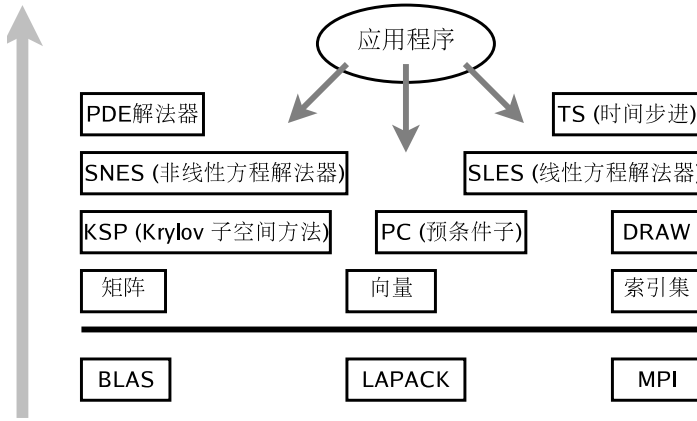

图 A.4 PETSc 实现的层次结构

A.5.2 PETSc 的基本特色

通过 PETSc 开发应用程序往往具有相当的难度。一方面，它需要用户本身具有较高的数值计算方法方面的专业知识和并行计算方法方面的编程技巧。另一方面，总的来说 PETSc 只是一个高级的应用程序开发环境，它为许多软件 (库) 和用户程序提供接口，用户只有充分熟悉和利用现有的软件资源和数学库的基础上才有可能开发出高效的应用程序。尽管如此，PETSc 仍然因为其具有其他软件不

可比拟的优点吸引着越来越多的用户用它来开发应用程序。下面一一介绍和分析 PETSc 的这些优点或特色。

计算能力 PETSc 为用户提供了丰富的算法和库资源。三个求解器（SLES、SNES 和 TS）构成了 PETSc 的核心组件。PETSc 不仅为中小规模线性方程组的求解提供了高效的直接方法，还为大规模 (稀疏) 线性方程组的迭代求解提供了多种 Krylov 子空间方法和多种预条件子。

兼容性 一方面，PETSc 具有很强的兼容能力，可在不同体系结构和不同操作系统环境高效运行。另一方面，PETSc 本身基于高性能的线性代数库 (BLAS 和 LAPACK) 和 MPI 消息传递环境实现，同时又充分吸收和融入了其他优秀软件的优点，如无结构网格区域分解和雅可比矩阵求解等方面的功能。

可扩展性 PETSc 的可扩展性主要体现在三个方面：计算性能的并行可扩展性、功能的可扩展性和计算能力的可扩展性。无论是在计算时间还是在浮点性能方面，PETSc 提供的范例程序都有良好的线性加速性能。面向对象的良好设计风格使得 PETSc 具有良好的功能扩展能力。作为一个高级应用程序开发平台，PETSc 特别适合于用来开发大型应用程序。

抽象数据类型 PETSc 基于面向对象技术实现，具有所有面向对象软件的可移植性、可继承性和可扩展性等基本程序特征。PETSc 的向量、矩阵等基本数据对象完全采用抽象数据类型实现，尽量对用户屏蔽数据对象的区域分解和存储等细节。所有 PETSc 格点数据对象的划分、初始化和存取等基本操作都由 DA 对象 (Distributed Array，即分布式数组) 来管理和相应 PETSc 库函数来实现。用户基于 PETSc 对象可以灵活开发其应用程

序。PETSc 对象和组件为构造大规模应用程序奠定了一个良好的基础。

输出能力 PETSc 具有良好的性能分析和图形输出功能，同时还具有高可用性，并具有很强的错误诊断能力。

总之，PETSc 在无论是在计算能力、设计风格还是在可兼容性和可扩展性等方面都显示出极大的优越性。PETSc 不但为科学与工程计算领域的科学家和工程师提供了强大的 (大规模) 偏微方程求解工具，而且也为高效算法设计提供了一个丰富的试验平台和计算环境。它使得算法的扩展和应用程序的个性化实现都更为容易。另外，PETSc 的这种设计风格增强了代码的重用性和编程的灵活性。

A.5.3 PETSc 的基本功能

这里主要介绍 PETSc 的三个核心组件及性能分析和接口功能。三个核心组件包括线性方程组求解器 (SLES)、非线性方程组求解器 (SNES) 和时间步进积分器 (TS)。

1. 线性方程组求解器

线性方程组求解器 (SLES) 构成了 PETSc 最核心的部分。它不仅是 PDE 方程求解器的基本内核，而且也是实现 PETSc 的其他两个核心组件 SNES 和 TS 的必不可少的部分。SLES 求解线性方程组

$$Ax = b \tag{A.14}$$

其中解算子 A 是 $n \times n$ 维非奇异矩阵，b 是 n 维右端向量，x 为 n 维解向量。SLES 在线性方程组求解环境的创建、Krylov 子空间方法和预条件子 (PC) 的选择、收敛性判据、LU 直接求解等方面为用户提供了强大的功能和灵活的操作方式。

2. 非线性方程组求解器

非线性方程组求解器 (SNES) 基于牛顿迭代法 (线搜索和信赖域方法)，在线性方程组求解器 (SLES) 的基础上实现。雅可比矩阵的求解是 SNES 求解器的重要组成部分。SNES 求解以下形式的非线性方程组

$$F(x) = 0 \tag{A.15}$$

其中算子 F 是 $\mathbb{R}^n \to \mathbb{R}^n$ 的函数。

3. 时间步进积分器

时间步进积分器 (TS) 用于求解 ODE 方程，或依赖时间的空间离散化后的 PDE 方程。TS 主要求解如下时间依赖问题

$$u_t = F(u, t) \tag{A.16}$$

其中 u 为有限维解向量，上式通常为运用有限差分或有限元方法对 PDE 进行离散后得到的常微分方程组。对于非时间演化或稳态方程，PETSc 提供了伪时间步进积分器。TS 积分器最终依赖线性方程组求解器 (SLES) 和非线性方程组求解器 (SNES) 来实现。PETSc 还提供了与 PVODE 求解器的接口。

4. PETSc 的其他功能

PETSc 不仅能为应用程序提供完整的计算时间、存储代价和浮点性能方面的信息，还能为计算的每个阶段和每个程序对象提供详细的信息。这些信息对于程序调试与优化、算法分析与比较等都有非常大的帮助。

5. PETSc 与其他软件

PETSc 可扩展性的另一个方面还表现在它为非常广泛的一类数值软件和数学库提供了非常方便的程序接口，主要包括以下几种类型：

(1) 线性代数求解器，如 AMG，BlockSolve95，DSCPACK，hypre，ILUTP，LUSOL，SPAI，SPOOLES，SuperLU，SuperLU_Dist；

(2) 最优化软件，如 TAO，Veltisto；

(3) 离散化和网格生成和优化工具包，如 Overture，SAMRAI，SUMAA3d；

(4) 常微分方程求解器，如 PVODE；

(5) 其他，如 MATLAB，ParMETIS。

感兴趣的读者可自行参阅相关软件的手册。

A.5.4 PETSc 计算实例

本节讨论基于 PETSc 的三个核心求解器 SLES、SNES 和 TS 如何开发不同的应用程序，它们分别与一个或多个不同的 PETSc 范例程序相对应。PETSc 范例程序不仅给用户提供了大量的编程模版，同时也给用户选择合适的数值方法提供了很好的参考。这些典型的范例程序包括二维 Poisson 方程、二维 Bratu 方程和一维热传导方程的数值求解。

1. 二维泊松方程的求解

(1) 问题描述及其离散化

二维 Poisson 方程是典型的椭圆型方程，其初边值问题在物理与工程领域具有重要的应用。考虑长方形区域 $\Omega = (0,a) \times (0,b)$ 上的二维 Poisson 方程

$$\begin{cases} -\Delta u = f, & (x,y) \in \Omega \\ u = g, & (x,y) \in \partial\Omega \end{cases} \tag{A.17}$$

如果 $f=0$，方程 (A.17) 就退化为二维 Laplace 方程。采用均匀网格及五点有限差分格式将方程 (A.17) 离散化，并取 $h_x=a/n$，$h_y=b/m$ 为网格步长，$x_i=ih_x$，$y_j=jh_y$，$i=0,1,\ldots,n$，$j=0,1,\ldots,m$。则差分方程为

$$\begin{cases} \dfrac{2u_{i,j}-u_{i+1,j}-u_{i-1,j}}{h_x^2}+\dfrac{2u_{i,j}-u_{i,j+1}-u_{i,j-1}}{h_y^2}=f_{i,j} \\ u_{i,0}=g_{i,0},\quad u_{i,m}=g_{i,m},\quad u_{0,j}=g_{0,j},\quad u_{n,j}=g_{n,j} \\ \qquad i=1,2,....,n-1,\quad j=1,2,....,m-1 \end{cases} \tag{A.18}$$

令 $d=1/(2/h_x^2+2/h_y^2)$，$d_x=d/h_x^2$，$d_y=d/h_y^2$，则上式可改写为

$$\begin{cases} u_{i,j}-d_x(u_{i+1,j}+u_{i-1,j})-d_y(u_{i,j+1}+u_{i,j-1})=df_{i,j} \\ u_{i,0}=g_{i,0},\quad u_{i,m}=g_{i,m},\quad u_{0,j}=g_{0,j},\quad u_{n,j}=g_{n,j} \\ \qquad i=1,2,....,n-1,\quad j=1,2,....,m-1 \end{cases} \tag{A.19}$$

这是一个 $(n-1)\times(m-1)$ 维的大型稀疏线性方程组。为简单起见，这里取网格步长 $h_x=h_y$，并适当排列、整理以上方程组，则式 (A.19) 可写成如下分块形式：

$$\begin{bmatrix} A & I & & & \\ I & A & I & & \\ & \ddots & \ddots & \ddots & \\ & & I & A & I \\ & & & I & A \end{bmatrix} \begin{bmatrix} \mathbf{u}_1 \\ \mathbf{u}_2 \\ \vdots \\ \mathbf{u}_{m-2} \\ \mathbf{u}_{m-1} \end{bmatrix} = \begin{bmatrix} \mathbf{b}_1 \\ \mathbf{b}_2 \\ \vdots \\ \mathbf{b}_{m-2} \\ \mathbf{b}_{m-1} \end{bmatrix} \tag{A.20}$$

其中 I 为 $n-1$ 阶单位矩阵，A 为 $n-1$ 阶三对角对称矩阵，

$$A=\begin{bmatrix} -4 & 1 & & & \\ 1 & -4 & 1 & & \\ & \ddots & \ddots & \ddots & \\ & & 1 & -4 & 1 \\ & & & 1 & -4 \end{bmatrix},$$

$$\mathbf{u}_j=\begin{bmatrix} u_{1,j} \\ u_{2,j} \\ \vdots \\ u_{n-2,j} \\ u_{n-1,j} \end{bmatrix}, \quad \mathbf{b}_j=-4d\begin{bmatrix} f_{1,j} \\ f_{2,j} \\ \vdots \\ f_{n-2,j} \\ f_{n-1,j} \end{bmatrix}$$

图 A.5 显示了使用 PETSc 运行选项 `-mat_view_draw` 后打印的网格规模为 36×36 的稀疏矩阵结构，它与方程 (A.20) 中的系数矩阵相对应。

(2) SLES 求解器中的主要参数设置

PETSc 在运行参数列表中提供了许多可选功能来完成 SLES 求解器的使用，其中与求解本例有关的选项有 (参考 SLES 范例 `ex2`)：

`-m/-n`: x 和 y 方向网格点数目

`-ksp_type`: 选择 Krylov 子空间迭代方法的类型，包括 Richardson 方法、切比雪夫方法、共轭梯度方法 (CG)、广义最小残量法 (GMRES)、双共轭梯度方法 (BiCG)、双共轭梯度平方法 (BCGS) 等。

`-ksp_rtol/-ksp_atol`: 设置解向量的相对误差和绝对误差。

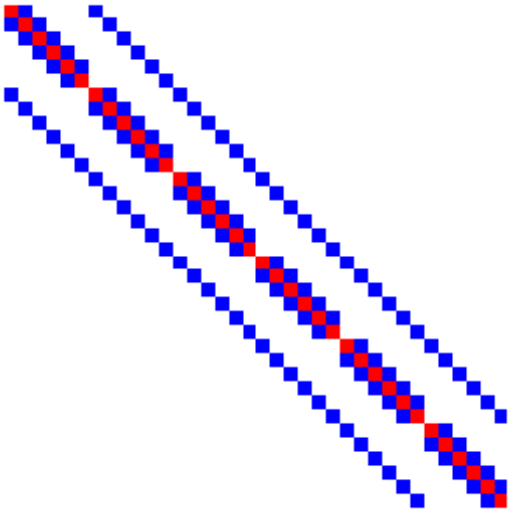

图 A.5 网格规模为 36×36 的稀疏矩阵结构：二维 Poisson 方程

`-pc_type`：选择预条件子的类型，包括雅可比矩阵、分块雅可比矩阵、加性 Schwartz 方法 (ASM)、多重网格方法等。

(3) SLES 求解器中的主要计算流程

- 运行参数的设置：包括网格点数目、解向量的相对误差或绝对误差、范数类型、迭代方法和预条件子等。
- 算子矩阵的填充：对于本例中求解的线性问题，需要填充的矩阵就是方程 (A.20) 中的系数矩阵 A。它是一个五对角的稀疏矩阵，通常可采用 PETSc 提供的稀疏行压缩格式 (AIJ)。
- 设置右端向量和解向量的初值。

- 启动 SLES 求解器。
- 结束 SLES 求解器并释放所有存储空间。

(4) 稀疏矩阵的赋值

稀疏矩阵的赋值关键是为函数 `MatCreatMPIAIJ` (并行稀疏行压缩格式) 或 `MatCreatMPIBAIJ` (并行稀疏矩阵的分块压缩格式) 提供四个参数：主对角分块每行非零元素的最大数目和每行非零元素的数目列表，非主对角分块每行非零元素的最大数目和每行非零元素的数目列表。根据方程 (A.20) 中表示的系数矩阵，每个进程划分中的主对角分块和非主对角分块每行非零元素的最大数目分别为 5 和 1。对于第一个进程划分，每行主对角非零元素的数目列表为：

$$
\begin{gathered}
3, 4, \ldots, 4 \\
5, 5, \ldots, 5 \\
\cdots \\
4, 4, \ldots, 3
\end{gathered}
$$

而每行非主对角非零元素的数目列表为

$$
\begin{gathered}
0, 0, \ldots, 0 \\
0, 0, \ldots, 0 \\
\cdots \\
1, 1, \ldots, 1
\end{gathered}
$$

其余进程划分可类似分析。当引用函数 `MatCreatMPIAIJ` 创建了一个稀疏矩阵结构后，用户就可以按通常的方式逐行对矩阵进行填充了。

(5) Krylov 子空间迭代方法和预条件子的选择

Krylov 子空间迭代方法是求解大型稀疏线性方程组的一类有效算法，其收敛速度依赖于矩阵特征值或奇异值的分布。如果选取一

个好的预条件子，各种 Krylov 子空间迭代方法在计算效率上通常没有多大差别，但实践中如何识别并构造一个合适的预条件子却甚为困难。

对本例 (即 SLES 范例 `ex2`) 的大量测试结果表明，在 PETSc 提供的所有 Krylov 子空间迭代方法中，共轭梯度方法 (CG) 是求解该问题最有效的迭代方法之一，它通常具有最小的计算时间和存储需求，而使用块雅可比矩阵预条件子则能成倍地加速其计算收敛性能。

2. 二维 Bratu 问题的求解

(1) 问题描述及其离散化

考虑区域 $\Omega = (0,1) \times (0,1)$ 上的二维 Bratu 方程

$$\begin{cases} -\nabla^2 u - \lambda e^u = 0, & (x,y) \in \Omega \\ u = 0, & (x,y) \in \partial\Omega \end{cases} \tag{A.21}$$

其离散化过程与 Poisson 方程的离散化完全类似，采用均匀网格及五点有限差分格式将方程 (A.21) 离散，并取 $h_x = 1/n$，$h_y = 1/m$ 为网格步长，$x_i = ih_x$，$y_j = jh_y$，$i = 0,1,\ldots,n$，$j = 0,1,\ldots,m$。则差分方程为

$$\begin{cases} \dfrac{2u_{i,j} - u_{i+1,j} - u_{i-1,j}}{h_x^2} + \dfrac{2u_{i,j} - u_{i,j+1} - u_{i,j-1}}{h_y^2} - \\ \qquad \lambda e^{u_{i,j}} = 0 \\ u_{i,0} = 0, \quad u_{i,m} = 0, \quad u_{0,j} = 0, \quad u_{n,j} = 0 \\ \qquad i = 1,2,....,n-1, \quad j = 1,2,....,m-1 \end{cases} \tag{A.22}$$

令 $\alpha = h_x/h_y$，$\beta = -\lambda h_x h_y$，

$$f_{i,j} = \left[(\alpha + \frac{1}{\alpha})u_{i,j} + \beta e^{u_{i,j}}\right] - \frac{1}{\alpha}(u_{i+1,j} + u_{i-1,j}) - \alpha(u_{i,j+1} + u_{i,j-1}) \tag{A.23}$$

将上式改写成向量形式

$$F(U) = 0 \tag{A.24}$$

其中 $F = [F_1, F_2, \ldots, F_{n-1}]$，$U = [u_{1,1}, u_{2,1}, \ldots, u_{n-1,m-1}]$，$F_j = [f_{1,j}, f_{2,j}, \ldots, f_{n-1,j}]$，$1 \leqslant j \leqslant m-1$。显然这里 F 为非线性函数，在应用牛顿迭代法来求解方程 (A.24) 的过程中需要求解其雅可比矩阵，用来形成每步迭代的线性方程组的解算子。根据式 (A.23)，雅可

比矩阵可简单表示为

$$\frac{\partial^{\mathrm{T}} F_j}{\partial U} = \begin{bmatrix} 1 & & & & & & & & & & & & \\ & -\frac{1}{\alpha} & & & & & & & & & & & \\ & \vdots & & -\frac{1}{\alpha} & & & & & & & & & \\ & \vdots & & \vdots & & \ddots & & & & & & & \\ & -\alpha & & \vdots & & & & -\frac{1}{\alpha} & & & & & \\ & \alpha+\frac{1}{\alpha}+\beta e^{u_{2,j}} & & -\alpha & & & & \vdots & & & & & \\ & -\alpha & & \alpha+\frac{1}{\alpha}+\beta e^{u_{3,j}} & & \ddots & & \vdots & & & & & \\ & \vdots & & -\alpha & & \ddots & & -\alpha & & & & & \\ & \vdots & & \vdots & & \ddots & & \alpha+\frac{1}{\alpha}+\beta e^{u_{n-1,j}} & & & & & \\ & \vdots & & \vdots & & & & -\alpha & & & & & \\ & -\frac{1}{\alpha} & & \vdots & & & & \vdots & & & & & \\ & & & -\frac{1}{\alpha} & & & & \vdots & & & & & \\ & & & & & \ddots & & \vdots & & & & & \\ & & & & & & & -\frac{1}{\alpha} & & & & & \\ & & & & & & & & & & & & 1 \end{bmatrix} \tag{A.25}$$

其中 $j = 1, \ldots, m-1$，分块雅可比矩阵维数为 $(m-1)\times(n-1)\times(n-1)$。图 A.6 显示了网格规模为 36×36 的雅可比稀疏矩阵结构。

(2) SNES 求解器中的主要参数设置

PETSc 在运行参数列表中提供了丰富的可选功能来完成 SNES 求解器和 DA 对象的使用，其中与求解本例有关的选项有 (参考

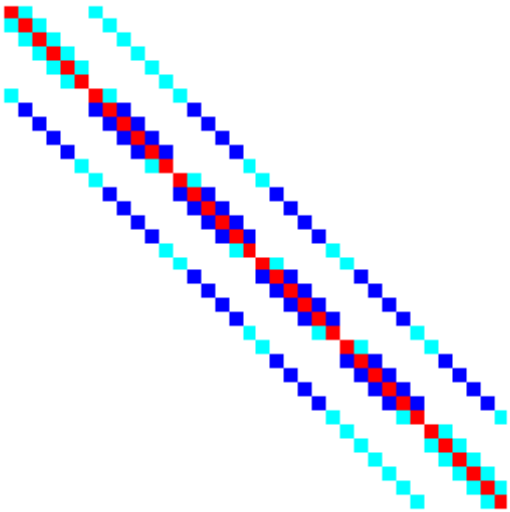

图 A.6 网格规模为 36×36 的稀疏雅可比矩阵结构：二维 Bratu 方程

SNES 范例 ex5)：

`-da_grid_x/-da_grid_y`：设置 x 和 y 方向网格点数目

`-fd_jacobian`：选择有限差分方法求解雅可比矩阵

`-adic_jacobian`：选择自动微分方法求解雅可比矩阵

`-adicmf_jacobian`：选择“无矩阵”格式和自动微分方法求解雅可比矩阵

`-snes_type`：选择牛顿迭代类型，包括一维线搜索和信赖域方法

`-snes_rtol/-snes_atol`：设置解向量的相对误差和绝对误差

`-snes_max_it`: 设置最大牛顿迭代步数

`-ksp_type`: 选择 Krylov 子空间迭代方法的类型

`-pc_type`: 选择预条件子的类型

`-ksp_rtol/-ksp_atol`: 设置每步求解线性方程组解向量的相对误差和绝对误差

`-ksp_max_it`: 设置每步求解线性方程组的最大迭代步数

(3) SNES 求解器中的主要计算流程

- 运行参数的设置：包括网格点数目、求解雅可比矩阵的方法、牛顿迭代类型、牛顿迭代的精度和最大迭代步数、Krylov 子空间迭代方法和预条件子、线性方程组求解的精度和最大迭代步数等。
- 创建 DA 向量对象：包括创建函数对象、解向量和雅可比矩阵对象。
- 非线性函数和雅可比矩阵: 见式 (A.24) 和式 (A.25)。
- 设置向量的边值：见式 (A.22)，均置零。
- 启动 SNES 求解器：主要计算流程见图 A.7。
- 打印输出结果和性能统计。
- 结束 SNES 求解器并释放所有存储空间。

(4) 雅可比矩阵的着色和求解

PETSc 提供了三种有效的方法来求解雅可比矩阵，包括有限差分方法、自动微分方法和“无矩阵”方法。这里仅做简单介绍。

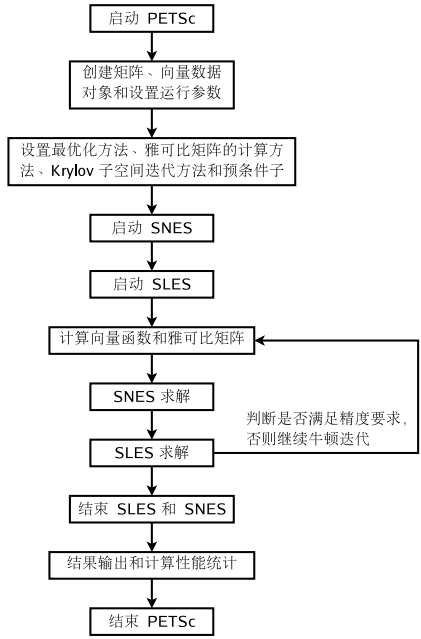

图 A.7 非线性求解器 (SNES) 的主要计算流程

有限差分方法 传统有限差分方法求解函数导数由于因数值方法带来的截断误差和因机器有效精度的影响，具有计算代价高和精度低的缺点。事实上，选择一个合适的变量增量值本身就是一个非常棘手的问题。

自动微分方法 自动微分方法在无截断误差意义上分析求解函数的导数，具有计算时间代价小、精度高和可靠性好等优点。切线性模式和伴随模式分别为与之相对应的最基本的代码实现形式，它们在精确求解函数梯度和雅可比矩阵方面分别具有不同的计算时间和空间存储代价。伴随模式在求解函数梯度方面具有理

想的计算时间代价，但其存储代价往往随问题的计算规模线性增长。

“无矩阵”方法 事实上，在使用牛顿迭代方法求解非线性方程组的过程中，只需求解雅可比矩阵与向量的乘积，因此许多矩阵运算本身可以借助向量操作来实现。然而，PETSc 目前尚未为“无矩阵”方法提供任何预条件子。

无论采用哪种方法来求解雅可比矩阵，都须首先求解其稀疏结构。当雅可比矩阵各元素取绝对值后的任意两行或两列相互“正交”时，通过适当设置初始输入向量后，雅可比矩阵的这两行或两列的所有元素都可以在一次微分函数或差分近似中同时求出。所有这样的初始输入向量经顺序排列后就形成初始输入矩阵。而如何根据雅可比矩阵的稀疏结构来求解初始输入矩阵，就是通常意义上的雅可比矩阵着色问题 (参考文献 [43, 44])。

(5) 迭代方法和预条件子的选择

此外，由于牛顿迭代的每一步都要求解一个线性方程组，用户还须为 SNES 中的迭代求解器 (KSP) 提供合适的迭代方法和预条件子。其中 PETSc 默认取广义最小残量法 (GMRES) 和零级优化的不完全 LU 分解预条件子。关于不同迭代方法和预条件子的选取，用户可以参考 429 页“二维泊松方程的求解”，这里不再赘述。

3. 一维热传导方程的求解

(1) 问题描述及其离散化

热传导方程是最简单的抛物型方程，其初边值问题在物理与工程领域具有广泛的应用。考虑区域 $\Omega = (0,1)$ 上的一维热传导方程

$$\begin{cases} u_t = u_{xx}, & x \in \Omega, 0 < t \leqslant T \\ u(0,x) = \sin 6\pi x + 3\sin 2\pi x, & x \in \Omega \\ u(t,0) = u(t,1) = 0, & 0 \leqslant t \leqslant T \end{cases} \tag{A.26}$$

这是一个二阶线性方程，解析解取为 $u(t,x) = e^{-36\pi^2 t}\sin 6\pi x + 3e^{-4\pi^2 t}\sin 2\pi x$。将方程 (A.26) 等距离散化，取网格步长 $h = 1/m$，$x_i = ih$，$i = 0,1,\ldots,m$，就得到如下形式的时间依赖问题

$$\begin{cases} (u_j)_t = \dfrac{u_{j+1} - 2u_j + u_{j-1}}{h^2} \\ u_j(0) = \sin 6\pi jh + 3\sin 2\pi jh \\ u_0(t) = u_m(t) = 0 \end{cases} \tag{A.27}$$

将上式改写成向量形式

$$U_t = AU \tag{A.28}$$

其中 $U = [u_1(t), u_2(t), \ldots, u_{m-1}(t)]$，系数矩阵

$$A = \frac{1}{h^2}\begin{bmatrix} -2 & 1 & & & \\ 1 & -2 & 1 & & \\ & \ddots & \ddots & \ddots & \\ & & 1 & -2 & 1 \\ & & & 1 & -2 \end{bmatrix}$$

显然系数矩阵 A 不依赖于时间，时间积分的每一步中只须重新设置右端向量。对于时间参数的离散化，TS 求解器在实现过程中可以分别采用显式向前 Euler 方法和隐式向后 Euler 方法。如果采用隐式向后 Euler 方法，离散化后可得

$$\begin{cases} \dfrac{u_j^{i+1}-u_j^i}{\Delta t^i} = \dfrac{u_{j+1}^{i+1}-2u_j^{i+1}+u_{j-1}^{i+1}}{h^2} & \\ u_j^0 = \sin(6\pi jh)+3\sin(2\pi jh), & j=0,1,....,m \\ u_0^i = u_m^i = 0, & i=0,1,....,n \end{cases} \tag{A.29}$$

取固定时间步长 $\tau = T/n$，并写成向量形式

$$[(1+2r)I - rC]\,U_h^{k+1} = U_h^k \tag{A.30}$$

其中 $r=\tau/h^2$，$U_h^k=[u_1^k,u_2^k,\ldots,u_{m-1}^k]$，而矩阵

$$C = \frac{1}{h^2}\begin{bmatrix} 0 & 1 & & & \\ 1 & 0 & 1 & & \\ & \ddots & \ddots & \ddots & \\ & & 1 & 0 & 1 \\ & & & 1 & 0 \end{bmatrix}$$

这是一个三对角线性方程组，可以利用 SLES 求解器的迭代方法求解。在选择时间步长和空间步长时，一定要注意为了保持计算过程的稳定性，不同离散格式对它们的限制是不同的。

(2) TS 求解器中的主要参数设置

PETSc 在运行参数列表中提供了许多可选功能来完成 TS 求解器的使用，其中与用向后 Euler 方法求解热传导方程有关的有：

`-m`: 网格点数目

`-ts_type`: 选择积分类型，有向前 Euler (`euler`)、向后 Euler (`beuler`) 和拟时间步进积分法 (`pseudo`)

`-ts_max_time`: 最终时间

`-ts_max_steps`: 最大时间积分步数

`-time_dependent_rhs`: 选择右端项为时间依赖项

`-TSSetInitialTimeStep`: 设置初始时间和时间步长

`-ksp_type`: 选择 Krylov 子空间迭代方法的类型

`-pc_type`: 选择预条件子的类型

(3) TS 求解器中的主要计算流程

- 运行参数的设置：包括积分类型、网格点数目、初始和最终时间、时间步长、最大时间积分步数、迭代方法和预条件子等。
- 设置求解矩阵：本例中求解线性问题，需要填充的线性算子就是式 (A.28) 中的系数矩阵 A。它是一个三对角的稀疏矩阵，用户可采用 PETSc 提供的稀疏行矩阵填充结构 (AIJ)。
- 向量初值的设置：设置求解向量初始时刻的值。
- 启动 TS 求解器：主要计算流程见图 A.8。
- 打印输出结果和性能统计。
- 结束 TS 求解器并释放所有存储空间。

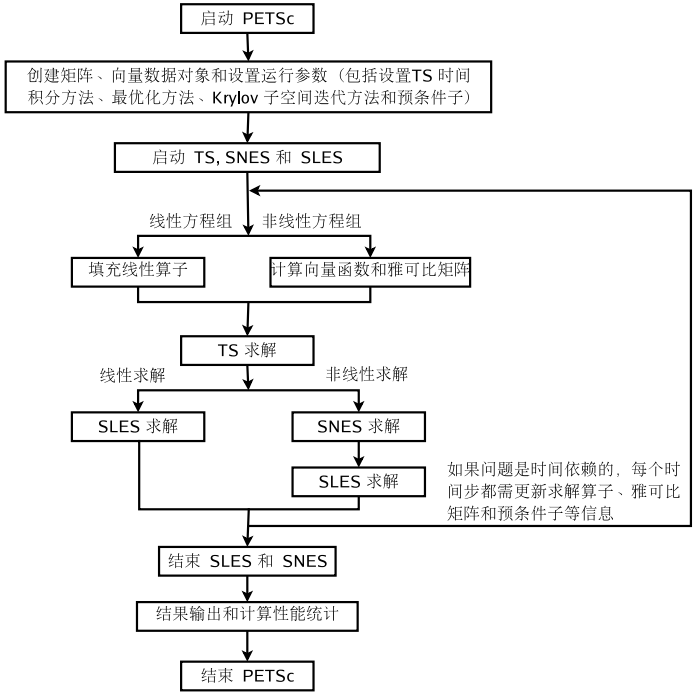

图 A.8 时间步进积分器 TS 的主要求解流程：隐式 Euler 方法

(4) 迭代方法和预条件子的选择

如果采用向后 Euler 方法，用户还须为 SLES 中的迭代求解器 (KSP) 提供合适的迭代方法和预条件子，其中 PETSc 默认选取广义最小残量法 (GMRES) 和零级优化的不完全 LU 分解预条件子。关于不同迭代方法和预条件子的选取，用户可以参考 429 页“二维泊松方程的求解”，这里不再赘述。

A.5.5 PETSc 小结

PETSc 为用户提供了一个高层的 PDE 应用程序开发平台，它由三个核心求解器，即线性求解器 SLES、非线性求解器 SNES 和时间步进积分器 TS 组成。PETSc 不仅提供丰富的 Krylov 子空间迭代方法和预条件子，还具有完善的性能统计、对象性能分析和图形可视化能力。用户基于 PETSc 的基本对象和库函数可以灵活地开发应用程序，同时也可以向 PETSc 添加新算法和预条件子等功能。此外，PETSc 还为许多软件和库提供了方便的接口。

PETSc 提供的分布式存储对象 (DA) 向用户屏蔽了物理存储的具体实现方式和细节，这种被尽可能追求的代数抽象让用户设计自己的数据结构和应用程序更为方便，客观上也降低了并行程序开发的难度。这种抽象的一个负面因素就是：远离具体存储实现的用户从理论上难以通过组织应用程序来获得程序的最优计算性能。另外，庞大的 PETSc 对象一方面占用系统的资源过大而影响了程序性能的提高，另一方面也增加了用户学习和应用其编程的难度。

PETSc 采用面向对象的程序技术开发，这使之具有现代软件可扩展性和可移植性好的程序风格，但同时也增加了那些仅具有 C 或 Fortran 程序开发经验的科技人员学习和应用的难度。不过，PETSc 提供的许多标准范例极大地方便了应用程序的开发。PETSc 还为许多专业软件提供了方便的接口，它们扩展了 PETSc 的应用范围和功能。

PETSc 除了学习难度较大之外，还要求几乎所有 PETSc 应用程序都必须在其提供的程序框架 (数据结构) 上开发，这有悖于人们习惯在用户程序框架下引用其他软件库和函数的传统思路。

大量测试结果表明：PETSc 应用程序具有良好的并行可扩展性能，但实际达到的浮点计算性能仍然比较低。例如，在深腾 6800 系统上，求解大规模稀疏线性方程组的 SLES 程序其单机浮点性能仅

为 80–120Mflops，大致为单机系统峰值性能的 4% ~ 8% 左右；使用一维线搜索和信赖域方法求解大规模非线性方程组的 SNES 程序其单机浮点性能分别为 30–70Mflops 和 25–40Mflops，占系统峰值性能的百分比更低；而求解大规模时间积分方程的 TS 程序其单机浮点性能仅达到 20–70Mflops。相对于 SLES 求解器而言，SNES 与 TS 求解器的单机浮点性能要更低一些。

不可否认，在高端应用程序开发平台软件的开发方面，PETSc 为新一代数值软件工具树立了一个优秀典范。面向对象的程序设计技术使得 PETSc 的所有对象和库，都具有标准化的程序接口和高度统一的程序设计风格和功能方面的巨大可扩展能力。