上海大学

硕士学位论文

矩阵三角分解递归算法的研究与实现

姓名：陈建平

申请学位级别：硕士

专业：电气工程

指导教师：康惠骏

20050301

摘　要

以求解线性方程组为代表的数值计算在现代科学研究和工程技术中得到广泛应用，数值分析问题的求解方法、数值计算的算法以及数值计算的计算机软件长期以来一直是人们的重要研究课题。线性方程组的求解计算最终归结为系数矩阵的三角分解，包括一般稠密型线性方程组的 LU 分解和对称正定型线性方程组的 Cholesky 分解。对于大型线性方程组，由于数据量很大，若按标准的 LU 分解或 Cholesky 分解公式直接计算，不能很好地利用当今高性能计算机的多级存储结构(即高缓 cashe)，计算时调用的是 level-1 和 level-2 BLAS，运算效率不高。为此，需要采用分块算法，通过将矩阵分块，使运算在小的分块子矩阵间进行，从而能较好地利用高缓，同时调用 level-3 BLAS，提高运算效率。递归能自动地产生矩阵分块，分块子矩阵的阶数逐级减小并趋于方阵，具有良好的数据局部性（locality），计算中也能更多地调用 level-3 BLAS，适合于当今分层多级存储的计算机结构。

本文对包括LU分解和Cholesky分解在内的矩阵三角分解的递归算法进行了研究，给出了 LU 分解和 Cholesky 分解递归算法的详细推导过程，用支持递归过程的 FORTRAN90 语言对得到的递归算法进行了实现，将产生的算法和程序在计算机上运行和测试，并与现行的 LAPACK 标准算法和程序进行了比较。研究和测试结果表明，递归算法在大矩阵情况下，优于 LAPACK 中的分块算法，运算速度提高 10～20%。本文的研究对于数值计算算法的发展具有一定理论意义，得到的算法和程序也具有实用价值，可用于解决大型科学和工程计算问题，提高计算机的运算速度和效率。

关键词： LU 分解，Cholesky 分解，矩阵分块，递归算法，FORTRAN90

ABSTRACT

Numerical computation methods including solving linear systems of equations are widely used in scientific research and engineering technology. The methods, algorithms and computer softwares for numerical analysis have been important research topics for a long time. The solving and computation of linear systems of equations are finally reduced to the triangular factorization of the coefficient matrix, including the LU factorization for the general dense linear equations and the Cholesky factorization for the symmetric and positive definite equations. For large systems of linear equations, if the LU or Cholesky factorization is computed directly according to the formula, the memory hierarchies (the cache) of today's high-performance computers cannot be well utilized and level-1 and level-2 BLAS are used. This makes the computation less efficient. Blocking algorithms are used to increase the computation efficiency. By matrix blocking, computation can be done among smaller submatrices to make a good use of cashe and level-3 BLAS. Recursion can bring about automatic matrix blocking. The sizes of resulted submatrices decrease stage by stage. The matrix operands are more squarish. The level-3 BLAS can be fully used. This makes the recursion method fit the memory hierarchies of today's high-performance computers very well.

Studies on the recursive algorithms for LU and Cholesky factorization are made in this thesis. The recursive algorithms of LU and Cholesky factorization are derived in detail. They are then implemented in FORTRAN90 language, which supports recursion as a language feature. The developed algorithms and programs are run and tested on a computer. Comparison with the standard LAPACK algorithms is made. The results show that in the case of a big matrix, the recursive algorithms are 10~20% faster than the LAPACK algorithms. The work of this thesis has a theoretical significance for the development of numerical methods. The resulted algorithms and programs have practical values. They can be used to solve large scientific and engineering problems and increase the computation efficiency.

Keywords: LU factorization, Cholesky factorization, Matrix blocking, Recursive algorithm, FORTRAN90

原 创 性 声 明

本人声明：所呈交的论文是本人在导师指导下进行的研究工作。除了文中特别加以标注和致谢的地方外，论文中不包含其他人已发表或撰写过的研究成果。参与同一工作的其他同志对本研究所做的任何贡献均已在论文中作了明确的说明并表示了谢意。

签 名：陈建军 日 期：2005.4.30

本论文使用授权说明

本人完全了解上海大学有关保留、使用学位论文的规定，即：学校有权保留论文及送交论文复印件，允许论文被查阅和借阅；学校可以公布论文的全部或部分内容。

（保密的论文在解密后应遵守此规定）

签 名：陈建军 导师签名：康建荣 日期：2005.5.26

第一章 绪 论

1.1 课题来源

本课题来源于江苏省教育厅留学回国人员科研启动经费项目，项目批文号为苏教外（2000）392号。

1.2 研究目的与意义

以求解线性方程组的LU分解、Cholesky分解等为代表的数值计算在现代科学研究和工程技术中得到广泛应用，其应用领域包括地质勘探、汽车制造、桥梁设计、天气预报、航空航天、自动控制和信息处理等许多方面。数值分析问题的求解方法、数值计算的算法以及数值计算的计算机软件长期以来一直是人们的重要研究课题。随着计算机技术的发展，实现线性代数数值计算的计算机算法和软件也在不断发展。通用的基本线性代数子程序库BLAS（Basic Linear Algebra Subprograms）从1970年代的Level-1 BLAS（执行向量—向量运算），发展到1980年代的Level-2 BLAS（执行矩阵—向量运算），再到1990年代的Level-3 BLAS（执行矩阵—矩阵运算，能达到或基本达到当今各种高性能处理器的峰值性能）。以调用BLAS为核心运算的线性代数软件包也相应地从早期的LINPACK（LINear algebra PACKage）发展到现在的LAPACK（Linear Algebra PACKage）。

目前，超标量、超流水线、具有多级存储结构的高性能计算机已占据了数值计算领域的主导地位。这些高性能计算机的中央处理器CPU的运算速度很快，而存储器的速度相对较慢，两者之间存在较大的速度差距。为了解决这一矛盾，当今的计算机在高速的CPU和低速的存储器之间设置了一级或多级高速缓冲存储器（cashe，简称高缓），形成了多级存储结构。高缓的存取速度很快，但容量较小，其目的是将计算机当前正在运算的数据或经常需要访问的数据置于高缓中，加快CPU对这些数据的访问，从而提高计算机整体运算速度。对于当今的这种具有多级存储结构的计算机，在进行算法设计时，尤其是像稠密线性代数这样的具有大量数据运算的算法，若能有效地利用高缓，更多地调用Level-3 BLAS，就可以充

分发挥计算机的性能，提高运算效率。为此，现行的线性代数算法（如 LAPACK）通常采用分块（block）算法。通过将矩阵分块，使各部分子矩阵的数据运算能够更多地在高缓中进行，同时能调用 level-3 BLAS，提高运算速度。

作为一种新的线性代数的计算方法，美国 IBM 公司 Watson 研究中心的 Fred G. Gustavson 于 1997 年提出了递归（recursive）分解的方法，其基本思想如图 1.1 所示。采用递归过程，大的矩阵分解成小的分块子矩阵的运算，小的分块矩阵的运算再分解为更小的分块子矩阵的运算，直到子矩阵的分块足够小。

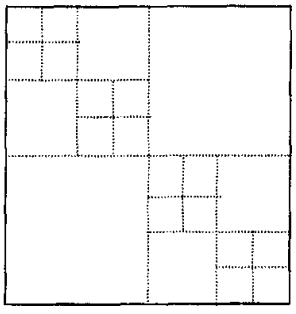

图 1.1　矩阵递归分块的基本思想

与一般固定分块的算法不同，递归算法缘于其本身的机制，能自动产生矩阵分块，分块子矩阵的阶数逐级减小并趋于方阵，具有良好的数据局部性（locality），这就使得递归算法非常适合当今分层多级存储的计算机结构。同时，递归算法结构简单，易于实现。

伴随着计算机技术的进步，计算机语言也在不断发展。FORTRAN 语言以适用于科学计算而见长，但 FORTRAN77 标准并不支持递归过程。新的 FORTRAN 语言标准 FORTRAN90 功能有了很大扩充，其中重要一点就是支持递归过程，递归由编译器自动处理，这给递归算法的实现带来很大方便。

本文将对数值计算中最常用的矩阵三角分解（包括 LU 分解和 Cholesky 分解）的递归算法进行研究，推导 LU 分解、Cholesky 分解的递归算法，用 FORTRAN90 语言对导出的递归算法进行实现，将产生的算法和程序在计算机上运行和测试，并与现有的算法进行比较，以期得到适合于现代高性能计算机的更有效的数值计算算法和程序。本文的研究对于数值计算算法的发展具有一定的理论意义，研究产生的算法和程序也将具有实用价值，可用于解决大型科学和工程计算问题，提

高计算机的运算速度和效率。

1.3 国内外概况

数值计算算法长期以来一直是应用数学和计算机科学领域的重要研究课题，国内有不少学者从事这方面的研究，近年来与本文相关或相近的研究有：文献[1]研究了基于网络机群的 Cholesky 并行分解算法，通过采用行卷帘存储方案和提前发送策略，减少了负载的不平衡，增加了计算通信的重叠，减少了通信时间，使得算法具有较高的并行加速比和效率。[2]提出了一种大型稀疏变带宽矩阵的存储方法及 Cholesky 分解法。通过用二个一维数组，其中一个数组输入时存储对称稀疏矩阵变等宽内的元素，输出时存储 Cholesky 下三角矩阵带宽内的元素；另一个数组存储对称变带宽矩阵对角线元素在前一维数组中的位置，使得运算所占的计算机内存空间大大减少。[3]针对求解大型有限元方程组的多重子结构法，提出了一种递归算法，能够减小对内存容量的要求和减少零元素的运算。[4]介绍了解决包括 LU 分解、Cholesky 分解在内的线性代数问题的并行算法，这些算法均是以快速的 BLAS 程序库为基础进行构建的，具有较高的运算效率。[5]提出了一种计算边界元法中具有稠密系数矩阵的方程组的分块算法，减小了内存的使用，提高了计算机的运算效率。[6]研究了基本线性代数子程序库 BLAS 的加速方法和实现技术。[7]介绍了 LAPACK 中的分块算法及其效果。

在国际上，不少大学和研究机构的学者一直从事数值计算算法的研究，以算法研究为基础，产生了实用的大型数值计算函数库和软件，如实现矩阵相乘等基本线性代数运算的通用线性代数子程序库 BLAS、适用于求解各种数值计算问题的线性代数软件包 LAPACK 等等。伴随着计算机技术的发展，这些算法和软件也在不断发展。针对当代具有多级存储结构的高性能计算机，1997 年美国 IBM 公司 Watson 研究中心的 Fred G. Gustavson 提出了采用递归的方法进行稠密型线性代数问题的计算[16]，通过递归分解，可以自动地产生矩阵分块，使算法适合于分层多级存储的计算机结构。此后其他学者也开始了这方面的研究，其中有美国田纳西大学计算机科学系的 Jack Dongarra、瑞典 Umea 大学计算科学系的 Andre Henriksson 和丹麦研究与教育计算中心的 Jerzy Wasniewski 等人。文献[17]研究了递归分块的数据格式，并将其应用于基本线性代数子程序库 BLAS 的运算中去。作为现代数

值计算软件如 LAPACK 的核心运算程序，这些递归分块数据格式的 BLAS 能够很好地利用计算机的分层多级存储结构，从而提高以调用 BLAS 为基本运算的数值计算算法的运算效率。文献[18]对压缩型（packed）存储格式的 Cholesky 分解的递归算法进行了研究，提出了一种 Packed Cholesky 分解递归算法，该算法建立在 level-3 BLAS 基础之上，比传统的标准 Packed Cholesky 分解算法的性能有很大提高。[19]研究了 QR 分解递归算法，针对算法中由于产生和执行更新操作所带来的大量额外开销，提出了一种混合型的递归算法。

如上所述，将递归引入数值分析方法的计算是一个较新的研究课题，国外有学者已开始了这方面的研究，并已取得一定成果，大量的研究工作仍在进行之中，有人预言递归算法有可能成为新一代数值计算软件的基础。

1.4 论文工作及内容安排

1.4.1 论文的主要研究工作

本文的主要研究内容为：

（1）从理论上推导 LU 分解递归算法和 Cholesky 分解递归算法，给出算法的描述。

（2）用 FORTRAN90 语言编程实现 LU 分解递归算法和 Cholesky 分解递归算法。

（3）将算法和程序在计算机上运行和测试，并与现有的算法和程序进行比较。

1.4.2 论文的内容安排

第一章介绍矩阵三角分解递归算法的研究背景、目的和意义、国内外研究状况、课题的主要研究工作和论文的内容安排。

第二章介绍包括 LU 分解和 Cholesky 分解在内的矩阵三角分解的概念和一般方法。

第三章在介绍分块算法概念的基础上，分别推导 LU 分解和 Cholesky 分解递归算法，给出算法的描述。

第四章首先简单介绍 FORTRAN90 语言、基本线性代数子程序库 BLAS 和线性代数软件包 LAPACK，然后用 FORTRAN90 对 LU 分解和 Cholesky 分解递归算法进行实

值计算软件如 LAPACK 的核心运算程序，这些递归分块数据格式的 BLAS 能够很好地利用计算机的分层多级存储结构，从而提高以调用 BLAS 为基本运算的数值计算算法的运算效率。文献[18]对压缩型（packed）存储格式的 Cholesky 分解的递归算法进行了研究，提出了一种 Packed Cholesky 分解递归算法，该算法建立在 level-3 BLAS 基础之上，比传统的标准 Packed Cholesky 分解算法的性能有很大提高。[19]研究了 QR 分解递归算法，针对算法中由于产生和执行更新操作所带来的大量额外开销，提出了一种混合型的递归算法。

如上所述，将递归引入数值分析方法的计算是一个较新的研究课题，国外有学者已开始了这方面的研究，并已取得一定成果，大量的研究工作仍在进行之中，有人预言递归算法有可能成为新一代数值计算软件的基础。

1.4 论文工作及内容安排

1.4.1 论文的主要研究工作

本文的主要研究内容为：

（1）从理论上推导 LU 分解递归算法和 Cholesky 分解递归算法，给出算法的描述。

（2）用 FORTRAN90 语言编程实现 LU 分解递归算法和 Cholesky 分解递归算法。

（3）将算法和程序在计算机上运行和测试，并与现有的算法和程序进行比较。

1.4.2 论文的内容安排

第一章介绍矩阵三角分解递归算法的研究背景、目的和意义、国内外研究状况、课题的主要研究工作和论文的内容安排。

第二章介绍包括 LU 分解和 Cholesky 分解在内的矩阵三角分解的概念和一般方法。

第三章在介绍分块算法概念的基础上，分别推导 LU 分解和 Cholesky 分解递归算法，给出算法的描述。

第四章首先简单介绍 FORTRAN90 语言、基本线性代数子程序库 BLAS 和线性代数软件包 LAPACK，然后用 FORTRAN90 对 LU 分解和 Cholesky 分解递归算法进行实数软件包 LAPACK，然后用 FORTRAN90 对 LU 分解和 Cholesky 分解递归算法进行实

值计算软件如 LAPACK 的核心运算程序，这些递归分块数据格式的 BLAS 能够很好地利用计算机的分层多级存储结构，从而提高以调用 BLAS 为基本运算的数值计算算法的运算效率。文献[18]对压缩型（packed）存储格式的 Cholesky 分解的递归算法进行了研究，提出了一种 Packed Cholesky 分解递归算法，该算法建立在 level-3 BLAS 基础之上，比传统的标准 Packed Cholesky 分解算法的性能有很大提高。[19]研究了 QR 分解递归算法，针对算法中由于产生和执行更新操作所带来的大量额外开销，提出了一种混合型的递归算法。

如上所述，将递归引入数值分析方法的计算是一个较新的研究课题，国外有学者已开始了这方面的研究，并已取得一定成果，大量的研究工作仍在进行之中，有人预言递归算法有可能成为新一代数值计算软件的基础。

1.4 论文工作及内容安排

1.4.1 论文的主要研究工作

本文的主要研究内容为：

（1）从理论上推导 LU 分解递归算法和 Cholesky 分解递归算法，给出算法的描述。

（2）用 FORTRAN90 语言编程实现 LU 分解递归算法和 Cholesky 分解递归算法。

（3）将算法和程序在计算机上运行和测试，并与现有的算法和程序进行比较。

1.4.2 论文的内容安排

第一章介绍矩阵三角分解递归算法的研究背景、目的和意义、国内外研究状况、课题的主要研究工作和论文的内容安排。

第二章介绍包括 LU 分解和 Cholesky 分解在内的矩阵三角分解的概念和一般方法。

第三章在介绍分块算法概念的基础上，分别推导 LU 分解和 Cholesky 分解递归算法，给出算法的描述。

第四章首先简单介绍 FORTRAN90 语言、基本线性代数子程序库 BLAS 和线性代数软件包 LAPACK，然后用 FORTRAN90 对 LU 分解和 Cholesky 分解递归算法进行实

现，给出了主要程序语句，并对有关实现技术和方法进行了阐述。

第五章对算法和程序进行运行和测试，验证算法和程序的正确性，测量程序的运算时间，与 LAPACK 中的同类程序进行比较，给出了测试结果。

第六章总结本文所作的工作和取得的研究成果，对今后进一步的研究工作提出一些设想。

第二章 矩阵的三角分解

2.1 一般矩阵的 LU 分解

LU 分解是用于求解线性方程组的方法。在自然科学和工程技术中，有许多问题的求解最终都转化为线性方程组的求解问题，如大型电路系统方程的求解、曲线拟合中常用的最小二乘法、解非线性方程组、求解偏微分方程的差分法以及工程实践中大量存在的反演问题等等。线性方程组的求解在社会科学和经济贸易领域也有不少应用。在有些应用中，最终形成的线性方程组的系数有许多为零，这样的线性方程组称为稀疏型线性方程组。与此相对应，不具有较多零系数的一般线性方程组则称为稠密型线性方程组。求解大型线性方程组有两类方法，一类是直接法，这类方法在假设没有舍入误差的情况下，经过有限步的运算可以得到方程组的精确解。另一类方法是迭代法，解出的是方程组的近似解，常用于求解稀疏型线性方程组。本文主要研究最具一般性的大型稠密型线性方程组的直接法求解问题。在直接法当中，最有效最常用的方法是高斯消去法，由高斯消去法产生出矩阵的 LU 分解问题。

2.1.1 从高斯消去法到 LU 分解

设有方程组 Ax＝b，即

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n = b_1 \\ a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n = b_2 \\ \qquad\qquad \vdots \\ a_{n1}x_1 + a_{n2}x_2 + \ldots + a_{nn}x_n = b_n \end{cases} \tag{2.1}$$

其中，系数矩阵 $A=(a_{ij})_{n\times n}$ 非奇异，$x=(x_1, x_2, \ldots, x_n)^T$，$b=(b_1, b_2, \ldots, b_n)^T$。式（2.1）的增广矩阵为

$$(A|b)=\begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} & b_1 \\ a_{21} & a_{22} & \cdots & a_{2n} & b_2 \\ \vdots & \vdots & & \vdots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} & b_n \end{pmatrix} \tag{2.2}$$

高斯消去法是对此增广矩阵进行一系列的初等行变换（将某一行的倍数加到下面某一行），最终使 A 的对角线以下的元素化为零：

$$(A^{'}|b^{'})=\begin{pmatrix} a^{'}_{11} & a^{'}_{12} & \cdots & a^{'}_{1n} & b^{'}_{1} \\ & a^{'}_{22} & \cdots & a^{'}_{2n} & b^{'}_{2} \\ & & \ddots & \vdots & \vdots \\ & & & a^{'}_{nn} & b^{'}_{n} \end{pmatrix} \tag{2.3}$$

式中，A’ 为上三角阵，令 A’ =U，b’ =g，则高斯消去法将方程组 Ax=b 等价转化为上三角方程组 Ux=g，该上三角方程组很容易自下而上地回代求解。

由线性代数理论可知，对一个矩阵进行一次初等行变换，相当于给这个矩阵左乘一个相应的初等矩阵。因此，上述高斯消去法的消去过程可以用矩阵乘法表示为

$$L_k L_{k-1}\cdots L_2 L_1(A|b)=(U|g) \tag{2.4}$$

其中 $L_i(i=1,2,\cdots,k)$ 均为单位下三角阵（对角元为 1 的下三角阵）。由式（2.4）有

$$L_k L_{k-1}\cdots L_2 L_1 A=U \tag{2.5}$$

记 $L=(L_k L_{k-1}\cdots L_2 L_1)^{-1}$，它仍为单位下三角阵，则有

$$A=LU \tag{2.6}$$

式中，L 和 U 分别为下三角和上三角矩阵。这表明高斯消去法实际上隐含了式(2.6)所示的系数矩阵 A 的三角阵分解。有了 A=LU 的三角分解，则方程组 Ax=b 可以写成

$$LUx=b \tag{2.7}$$

它等价于两个三角方程组：

$$Ly = b \quad （上三角方程组） \tag{2.8}$$

$$Ux = y \quad （下三角方程组） \tag{2.9}$$

即

$$\begin{pmatrix} 1 & & & \\ l_{21} & 1 & & \\ \vdots & \vdots & \ddots & \\ l_{n1} & l_{n2} & \cdots & 1 \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix} \tag{2.10}$$

$$\begin{pmatrix} u_{11} & u_{12} & \cdots & u_{1n} \\ & u_{22} & \cdots & u_{2n} \\ & & \ddots & \vdots \\ & & & u_{nn} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix} \tag{2.11}$$

式（2.10）和（2.11）很容易通过回代的方式求解。这样，线性方程组的求解问题就归结为系数矩阵 A 的 LU 分解问题。

2.1.2 矩阵的 LU 分解

下一步的任务是给定系数矩阵 A，如何得到其 LU 分解，即得到下三角阵 L 和上三角阵 U 中的元素。常用的方法是 Doolittle 分解。设 A＝LU 即

$$\begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{pmatrix} = \begin{pmatrix} 1 & & & \\ l_{21} & 1 & & \\ \vdots & \vdots & \ddots & \\ l_{n1} & l_{n2} & \cdots & 1 \end{pmatrix} \begin{pmatrix} u_{11} & u_{12} & \cdots & u_{1n} \\ & u_{22} & \cdots & u_{2n} \\ & & \ddots & \vdots \\ & & & u_{nn} \end{pmatrix} \tag{2.12}$$

由矩阵乘法，第一步，

$$a_{1j} = u_{1j} \quad (j = 1.2, \cdots, n)$$

$a_{i1} = l_{i1} \cdot u_{11}$，故 $l_{i1} = a_{i1} / u_{11} \quad (i = 2,3, \cdots, n)$

这就求出了矩阵 U 的第 1 行和 L 的第 1 列的元素。由 U 的第 1 行和 L 的第 1 列元素，可求出 U 的第 2 行和 L 的第 2 列的元素。一般地，设 U 的前 k-1 行和 L 的前 k-1 列已经求出，则第 k 步，

$$a_{kj} = \sum_{m=1}^{k-1} l_{km} u_{mj} + u_{kj}$$

故

$$u_{kj} = a_{kj} - \sum_{m=1}^{k-1} l_{km} u_{mj} \qquad (j = k, k+1, \cdots, n)$$

又有

$$a_{ik} = \sum_{m=1}^{k-1} l_{im} u_{mk} + l_{ik} u_{kk}$$

故

$$l_{ik} = (a_{ik} - \sum_{m=1}^{k-1} l_{im} u_{mk}) / u_{kk} \qquad (i = k+1, k+2, \cdots, n)$$

这样就求出了 U 的第 k 行和 L 的第 k 列。综上所述，矩阵 A 的 LU 分解公式如下

$$\begin{cases} u_{1j} = a_{1j} \qquad (j = 1,2,\cdots,n) \\ l_{i1} = a_{i1} / u_{11} \qquad (i = 2,3,\cdots,n) \\ u_{kj} = a_{kj} - \sum_{m=i}^{k-1} l_{km} u_{mj} \qquad (j = k, k+1, \cdots, n) \\ l_{ik} = (a_{ik} - \sum_{m=1}^{k-1} l_{im} u_{mk}) / u_{kk} \quad (i = k+1, k+2, \cdots, n) \\ \qquad (k = 2,3,\cdots,n) \end{cases} \tag{2.13}$$

分解后的 L、U 的元素 l_{ik} 和 u_{kj} 可放回 A 的相应元素 a_{ik} 和 a_{kj} 所在的位置，不需占用新的存储单元，即最终有

$$A = \begin{pmatrix} u_{11} & u_{12} & \cdots & u_{1n} \\ l_{21} & u_{22} & \cdots & u_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ l_{n1} & l_{n2} & \cdots & u_{nn} \end{pmatrix} \tag{2.14}$$

2.2 对称矩阵的 Cholesky 分解

在实际工程应用中，许多问题的求解所归结出来的线性方程组的系数矩阵常常是对称正定的，如应用有限元求解结构力学问题时，最后就归结为求解线性方程组，其系数矩阵大多具有对称正定的性质。对于对称正定矩阵，利用其数据对称的特点，可以推导出更为有效的三角分解方法，这就是所谓的 Cholesky 分解法，它比一般的 LU 分解法节省大约一半的计算量和存储单元。下面给出 Cholesky 分解法的描述。

对线性方程组 Ax=b，若系数矩阵 A 对称正定，则可以证明 A 可以三角分解成 $A=LL^T$ 的形式（证明过程见参考文献[14]），L 为下三角阵，其对角元素全为正，即

$$\begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{pmatrix} = \begin{pmatrix} l_{11} & & & \\ l_{21} & l_{22} & & \\ \vdots & \vdots & \ddots & \\ l_{n1} & l_{n2} & \cdots & l_{nn} \end{pmatrix} \begin{pmatrix} l_{11} & l_{21} & \cdots & l_{n1} \\ & l_{22} & \cdots & u_{n2} \\ & & \ddots & \vdots \\ & & & l_{nn} \end{pmatrix} \tag{2.15}$$

式中，$a_{ij}=a_{ji}$（i，j=1，2，…，n），$l_{ii}>0$（i=1，2，…，n）。

L 阵中各元素的计算同样由矩阵乘法求得。第一步，有 $a_{11}=l_{11}^2$，$a_{i1}=l_{i1}\cdot l_{11}$，故可求得

$$\begin{aligned} l_{11} &= \sqrt{a_{11}} \\ l_{i1} &= a_{i1}/l_{11} \qquad (i=2,3,\cdots,n) \end{aligned} \tag{2.16}$$

一般地，设 L 的前 k-1 列元素已求出，则第 k 步，由矩阵乘法得

$$\sum_{m=1}^{k-1} l_{km}^2 + l_{kk}^2 = a_{kk}, \quad \sum_{m=1}^{k-1} l_{im} l_{km} + l_{ik} l_{kk} = a_{ik}$$

于是

$$\begin{aligned} l_{kk} &= \sqrt{a_{kk} - \sum_{m=1}^{k-1} l_{km}^2} \\ l_{ik} &= (a_{ik} - \sum_{m=1}^{k-1} l_{im} l_{km})/l_{kk} \qquad (i=k+1,k+2,\cdots,n) \end{aligned} \tag{2.17}$$

式（2.16）和（2.17）即为对称正定矩阵 A 的 Cholesky 分解公式。事实上，式

（2.16）可以合并到式（2.17）中，最终有

$$\begin{cases} k=1,2,\cdots,n \\ l_{kk}=\sqrt{a_{kk}-\sum_{m=1}^{k-1}l_{km}^2} \\ l_{ik}=(a_{ik}-\sum_{m=1}^{k-1}l_{im}l_{km})/l_{kk} \qquad (i=k+1,k+2,\cdots,n) \end{cases} \tag{2.18}$$

一旦求得下三角阵 L 后，线性方程组 Ax=b 的解就很容易通过求解两个三角方程组 Ly=b 和 L^Tx=y 得到。

从式（2.18）可看到，L 阵中元素的计算只需用到矩阵 A 下三角部分元素，即

$$\begin{pmatrix} a_{11} & & & \\ a_{21} & a_{22} & & \\ \vdots & \vdots & \ddots & \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{pmatrix}=\begin{pmatrix} l_{11} & & & \\ l_{21} & l_{22} & & \\ \vdots & \vdots & \ddots & \\ l_{n1} & l_{n2} & \cdots & l_{nn} \end{pmatrix}\begin{pmatrix} l_{11} & l_{21} & \cdots & l_{n1} \\ & l_{22} & \cdots & l_{n2} \\ & & \ddots & \vdots \\ & & & l_{nn} \end{pmatrix} \tag{2.19}$$

与 LU 分解相比，Cholesky 分解只需计算一个三角阵 L 的元素，同时分解后得到的下三角阵 L 中的元素可放回 A 的下三角元素所在的位置。因此，Cholesky 分解可节省大约一半的计算量和存储单元。

同样，若给定 A 的上三角部分元素，则 A 可分解成 $A=U^TU$ 的形式，即

$$\begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ & a_{22} & \cdots & a_{2n} \\ & & \ddots & \vdots \\ & & & a_{nn} \end{pmatrix}=\begin{pmatrix} u_{11} & & & \\ u_{12} & u_{22} & & \\ \vdots & \vdots & \ddots & \\ u_{1n} & u_{2n} & \cdots & u_{nn} \end{pmatrix}\begin{pmatrix} u_{11} & u_{12} & \cdots & u_{1n} \\ & u_{22} & \cdots & u_{2n} \\ & & \ddots & \vdots \\ & & & u_{nn} \end{pmatrix} \tag{2.20}$$

式中 U 为上三角阵，它与上面 L 的关系为 $U=L^T$，其元素的计算公式相应为

$$\begin{cases} k=1,2,\cdots,n \\ u_{kk}=\sqrt{a_{kk}-\sum_{m=1}^{k-1}u_{mk}^{\ 2}} \\ u_{ki}=(a_{ki}-\sum_{m=1}^{k-1}u_{mk}u_{mi})\Big/u_{kk} \qquad i=k+1,k+2,...,n \end{cases} \tag{2.21}$$

2.3 小结

本章从求解线性方程组入手，引入矩阵三角分解的概念。从高斯消去法出发，一般线性方程组的求解经过推导，最终化归为将系数矩阵 A 分解为下三角矩阵 L 和上三角矩阵 U 的乘积 A=LU 即矩阵的 LU 分解问题，并给出了 L 和 U 中元素的计算公式。对于工程实际中经常存在的系数矩阵满足对称正定性质的线性方程组，其系数矩阵的三角分解可以简化成 $A=LL^T$ 或 $A=U^TU$ 的形式，即所谓的 Cholesky 分解，L 或 U 中的元素可通过给定公式计算。

本章的目的是介绍 LU 分解和 Cholesky 分解的概念和一般方法，为后面推导 LU 分解和 Cholesky 分解递归算法提供必要的基础和铺垫。

第三章　矩阵三角分解递归算法的推导

3.1 LU 分解分块算法

根据式（2.13）直接计算 LU 分解，是按列或按行计算，属于向量与向量相乘，实现时调用的是 level-1 和 level-2 BLAS，同时运算在整个大矩阵 A 的范围内进行，数据访问的跨度很大，不能很好地利用高缓，因而运算效率不高。为了提高计算效率，可采用分块算法。

将矩阵 A 分成 2×2 块，方程 A=LU 即式（2.12）可写成

$$\begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} = \begin{pmatrix} L_{11} & \\ L_{21} & L_{22} \end{pmatrix} \begin{pmatrix} U_{11} & U_{12} \\ & U_{22} \end{pmatrix} \tag{3.1}$$

式中，分块子矩阵 L_{11}、L_{22} 为下三角阵，U_{11}、U_{22} 为上三角阵，其余为矩形阵。计算式（3.1）可得

$$A_{11} = L_{11}U_{11} \tag{3.2}$$

$$A_{12} = L_{11}U_{12} \tag{3.3}$$

$$A_{21} = L_{21}U_{11} \tag{3.4}$$

$$A_{22} = L_{21}U_{12} + L_{22}U_{22} \tag{3.5}$$

式（3.2）的含义是对子矩阵 A_{11} 进行 LU 分解，在解出 L_{11}、U_{11} 后，由式（3.3）计算 U_{12}

$$U_{12} = L_{11}^{-1}A_{12} \tag{3.6}$$

由式（3.4）计算 L_{21}

$$L_{21} = A_{21}U_{11}^{-1} \tag{3.7}$$

令

$$A'_{22} = A_{22} - L_{21}U_{12} \tag{3.8}$$

则式（3.5）成为

$$A'_{22} = L_{22}U_{22} \tag{3.9}$$

式（3.8）的含义为通过 L_{21} 和 U_{12}（已解出）对 A_{22} 进行修正或更新。式（3.9）为对修正后的 A'_{22} 进行 LU 分解。

这样，矩阵 A 的 LU 分解运算就转化成分块子矩阵 A_{11} 和 A'_{22} 的 LU 分解以及 U_{12}、L_{21} 的求解和 A_{22} 的修正计算。只要 A_{11} 的分块足够小，A_{11} 的分解计算（运用式(2.13)）就有可能在高缓内进行。U_{12}、L_{21} 的求解（式（3.6）、（3.7））和 A_{22} 的修正计算（式（3.8））也都是在较小的子矩阵之间进行的，并且均为矩阵与矩阵之间的运算，可调用效率高的 level-3 BLAS 完成。接下来，对子矩阵 A'_{22} 的 LU 分解，可进行同样的分块处理和计算，直至最终产生的 A'_{22} 足够小，可直接用不分块的方法即式（2.13）计算。分块算法的过程可用图 3.1 直观表示。

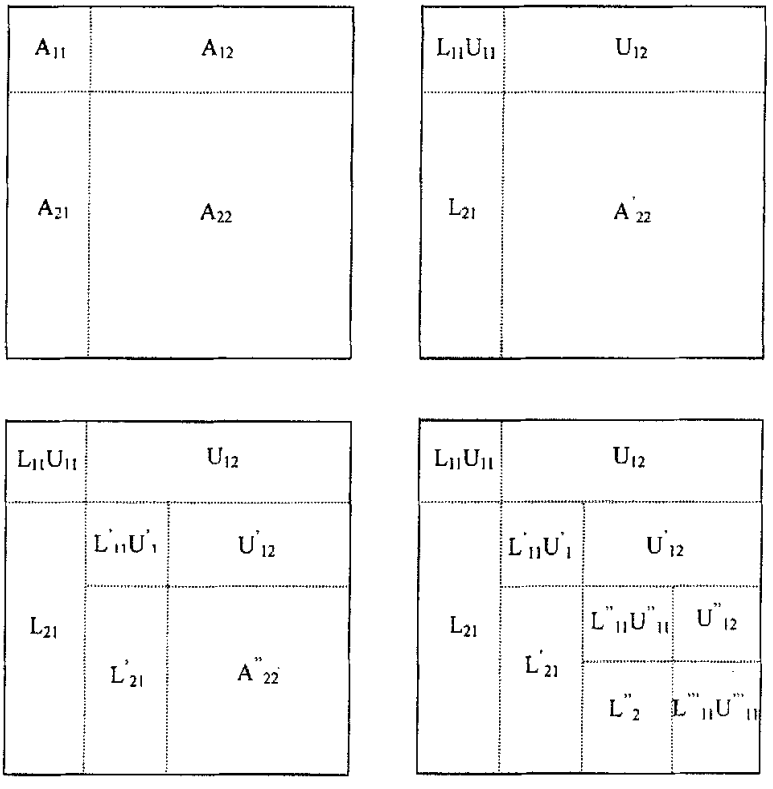

图 3.1　LU 分解分块算法的图示

上述 LU 分解分块算法不增加浮点运算量，也不需要增加额外的存储单元，由于较好地利用了高缓和调用 level-3 BLAS，其运算效率要比不分块的算法提高许

多，目前 LAPACK 中就采用这种算法。

3.2 LU 分解递归算法

上面介绍的分块算法的一个很重要的方面就是分块大小的确定，它通常需要针对特定目标机进行大量的试验和分析以获得最佳分块尺寸。同时，分块大小在运算过程中是固定不变的。递归算法也是一种分块算法，它与上述分块算法的不同之处在于，它是通过递归调用来自动完成矩阵分块，分块的大小逐步减小，并且更趋于方阵。下面推导 LU 分解的递归算法。

3.2.1 算法的推导

不失一般性，设矩阵 A 的阶数为 m×n，且 m≥n。令 d=n/2（d 取整数），将式（2.12）写成分块形式

$$\begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} = \begin{pmatrix} L_{11} & \\ L_{21} & L_{22} \end{pmatrix} \begin{pmatrix} U_{11} & U_{12} \\ & U_{22} \end{pmatrix} \tag{3.10}$$

式中，分块子矩阵 A_{11}、L_{11} 和 U_{11} 的阶数为 d×d，A_{12} 和 U_{12} 的阶数为 d×(n-d)，A_{21} 和 L_{21} 的阶数为(m-d)×d，A_{22} 和 L_{22} 的阶数为(m-d)×(n-d)，U_{22} 的阶数为(n-d)×(n-d)。子矩阵中，L_{11}、L_{22} 为下三角阵，U_{11}、U_{22} 为上三角阵，其余为矩形阵。计算式（3.10）可得

$$A_{11} = L_{11}U_{11} \tag{3.11}$$

$$A_{21} = L_{21}U_{11} \tag{3.12}$$

$$A_{12} = L_{11}U_{12} \tag{3.13}$$

$$A_{22} = L_{21}U_{12} + L_{22}U_{22} \tag{3.14}$$

式（3.11）和式（3.12）可合并写成

$$\begin{pmatrix} A_{11} \\ A_{21} \end{pmatrix} = \begin{pmatrix} L_{11} \\ L_{21} \end{pmatrix} (U_{11}) \tag{3.15}$$

其含义是对$\begin{pmatrix} A_{11} \\ A_{21} \end{pmatrix}$进行 LU 分解，解出 L_{11}、L_{21}和 U_{11}后，由式（3.13）计算 U_{12}

$$U_{12} = L_{11}^{-1} A_{12} \tag{3.16}$$

令

$$A'_{22} = A_{22} - L_{21} U_{12} \tag{3.17}$$

则式（3.14）成为

$$A'_{22} = L_{22} U_{22} \tag{3.18}$$

式（3.17）的含义为通过 L_{21}和 U_{12}（已解出）对 A_{22}进行修正或更新。式（3.18）为对修正后的 A'_{22}进行 LU 分解。

这样，矩阵 A 的 LU 分解运算就转化成分块子矩阵$\begin{pmatrix} A_{11} \\ A_{21} \end{pmatrix}$和 A'_{22}的 LU 分解以及 U_{12}的求解和 A_{22}的修正计算。以上完成了一级分解，下一步对子矩阵$\begin{pmatrix} A_{11} \\ A_{21} \end{pmatrix}$和 A'_{22}的 LU 分解分别再进行同样的处理，将它们各自分解成列数约为(n/4)的子矩阵的运算。这种递归分解过程可以一直重复下去，直到最终产生的子矩阵$\begin{pmatrix} A_{11} \\ A_{21} \end{pmatrix}$和 A'_{22}的列数足够小，或者更一般地，直到它们的列数为 1。此时$\begin{pmatrix} A_{11} \\ A_{21} \end{pmatrix}$和 A'_{22}为列向量，其 LU 分解由式（2.13）中的前两式即可求得。于是，LU 分解递归算法可以描述为：

LU 分解 A：　$A=LU$ （递归过程）

如果 n>1，则

d=n/2

LU 分解$\begin{pmatrix} A_{11} \\ A_{21} \end{pmatrix}$：　$\begin{pmatrix} A_{11} \\ A_{21} \end{pmatrix} = \begin{pmatrix} L_{11} \\ L_{21} \end{pmatrix} (U_{11})$　（递归调用）

解 U_{12}：　$U_{12} = L_{11}^{-1} A_{12}$

计算 A'_{22}：　$A'_{22} = A_{22} - L_{21} U_{12}$

LU 分解 A'_{22}：　$A'_{22} = L_{22}U_{22}$　（递归调用）

否则（n==1）

$$u_{11} = a_{11}$$
$$l_{i1} = a_{i1} / u_{11} \quad (i = 2,3,\cdots,m)$$

图 3.2 以 4×4 阶矩阵为例，用图形表示出递归分解的过程，图中用斜线分隔显示分解形成的上三角阵 U 和下三角阵 L。

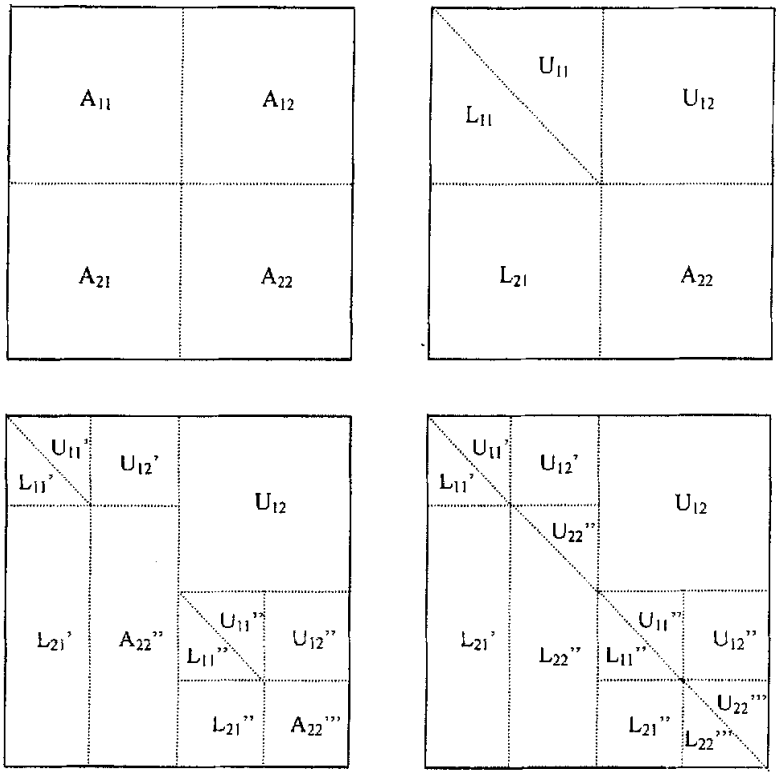

图 3.2　LU 分解递归算法的分解过程

从以上分解过程可以看出，矩阵分块在递归求解的过程中自动完成，分块的大小逐级减小，比前面的分块算法更接近方阵，使得算法具有更好的数据局部性，能够很好地利用当今计算机分层多级存储的结构。同时，上述递归算法结构简单，易于实现。整个算法的算术运算除了式（2.13）中前两式的简单运算外，主要由式（3.16）和式（3.17）构成，它们可以调用高效率的 level-3 BLAS。

3.2.2 增加选主元的步骤

为了避免式（2.13）中的除数为零或绝对值相对很小，LU 分解在实际应用中需要增加选主元的步骤。因此，在上述 LU 分解递归算法中，也需要加入选主元的过程。通常采用部分选主元，即选取列主元，进行行交换的方法。对矩阵 A 进行一系列行交换，相当于用一个排列矩阵 P 左乘矩阵 A，然后再对 PA 进行 LU 分解。PA 的 LU 分解递归转化成 $P_1\begin{pmatrix}A_{11}\\A_{21}\end{pmatrix}$ 和 $P_2A'_{22}$ 的 LU 分解。P_1 和 P_2 分别为对 $\begin{pmatrix}A_{11}\\A_{21}\end{pmatrix}$ 和 A'_{22} 进行选主元，作行交换形成的排列矩阵。在解得 $P_1\begin{pmatrix}A_{11}\\A_{21}\end{pmatrix}$ 后，需要用 P_1 对 $\begin{pmatrix}A_{12}\\A_{22}\end{pmatrix}$ 进行同样的换行，然后再对 $\begin{pmatrix}A_{12}\\A_{22}\end{pmatrix}$ 进行计算和分解。同样，在解得 $P_2A'_{22}$ 后，需要用 P_2 对 A_{21}（L_{21}）进行同样的换行以得到 A_{21}（L_{21}）中元素的正确位置。这样，最终形成的带有部分选主元的 LU 分解递归算法的描述为:

LU 分解 PA：$PA = LU$ （递归过程）

如果 n>1，则

d=n/2

LU 分解 $P_1\begin{pmatrix}A_{11}\\A_{21}\end{pmatrix}$：$P_1\begin{pmatrix}A_{11}\\A_{21}\end{pmatrix}=\begin{pmatrix}L_{11}\\L_{21}\end{pmatrix}(U_{11})$ （递归调用）

$\begin{pmatrix}A_{12}\\A_{22}\end{pmatrix}$ 换行：$\begin{pmatrix}A_{12}\\A_{22}\end{pmatrix}=P_1\begin{pmatrix}A_{12}\\A_{22}\end{pmatrix}$

解 U_{12}：$U_{12}=L_{11}^{-1}A_{12}$

计算 A'_{22}：$A'_{22}=A_{22}-L_{21}U_{12}$

LU 分解 $P_2A'_{22}$：$P_2A'_{22}=L_{22}U_{22}$ （递归调用）

A_{21} 换行：$A_{21}=P_2A_{21}$

否则（n=1）

找出（a_{11}，a_{21}，…，a_{m1}）中的主元，设为 a_{x1}

将 a_{11} 和 a_{x1} 换位（同时产生排列矩阵 P），然后计算

$$u_{11} = a_{11}$$
$$l_{i1} = a_{i1} / u_{11} \quad (i = 2,3,\cdots,m)$$

3.3 Cholesky 分解分块算法

与 LU 分解的情况相似，用式（2.18）或（2.21）直接计算 Cholesky 分解，是按列或按行计算，为向量与向量相乘，实现时调用的是 level-1 和 level-2 BLAS，同时运算在整个大矩阵 A 的范围内进行，不能很好地利用高缓，运算效率不高。为了提高计算效率，同样也有分块算法。

我们以 $A=U^TU$ 即式（2.20）的分解形式，介绍分块算法。将矩阵 A 分成 2×2 块，式（2.20）可写成

$$\begin{pmatrix} A_{11} & A_{12} \\ & A_{22} \end{pmatrix} = \begin{pmatrix} U_{11}^T & \\ U_{12}^T & U_{22}^T \end{pmatrix} \begin{pmatrix} U_{11} & U_{12} \\ & U_{22} \end{pmatrix} \tag{3.19}$$

式中，分块子矩阵 A_{11}、A_{22}、U_{11}、U_{22} 为上三角阵，U_{11}^T、U_{22}^T 为下三角阵，其余为矩形阵。计算式（3.19）可得

$$A_{11}=U_{11}^TU_{11} \tag{3.20}$$

$$A_{12}=U_{11}^TU_{12} \tag{3.21}$$

$$A_{22}=U_{12}^TU_{12}+U_{22}^TU_{22} \tag{3.22}$$

式（3.20）为对子矩阵 A_{11} 进行 Cholesky 分解，由式（2.21）可计算出 U_{11}。在解得 U_{11}（亦即 U_{11}^T）后，由式（3.21）计算 U_{12}

$$U_{12}=(U_{11}^T)^{-1}A_{12} \tag{3.23}$$

然后通过式（3.22）计算 U_{22}。将式（3.22）改写成

$$A_{22}-U_{12}^TU_{12}=U_{22}^TU_{22} \tag{3.24}$$

令

$$A_{22}'=A_{22}-U_{12}^TU_{12} \tag{3.25}$$

则式（3.24）成为

$$A_{22}' = U_{22}^{T} U_{22} \tag{3.26}$$

式（3.25）的含义为通过 U_{12}（U_{12}^{T}）对 A_{22} 进行修正或更新，修正后的子矩阵 A_{22}' 仍为上三角阵（对称正定）。式（3.26）为对 A_{22}' 进行 Cholesky 分解，由式（2.21）计算出 U_{22}。

这样，矩阵 A 的 Cholesky 分解运算就转化成分块子矩阵 A_{11} 和 A_{22}' 的 Cholesky 分解以及 U_{12} 的求解和 A_{22} 的修正计算。只要 A_{11} 的分块足够小，A_{11} 的分解计算（运用式（2.21））就有可能在高缓内进行。U_{12} 的求解（式（3.23））和 A_{22} 的更新计算（式（3.25））也都是在较小的子矩阵之间进行的，并且均为矩阵与矩阵之间的运算，可调用效率高的 level-3 BLAS 完成。接下来，对子矩阵 A_{22}' 的 Cholesky 分解，可进行同样的分块处理和计算，直至最终产生的 A_{22}' 足够小，可直接用不分块的方法即式（2.21）计算。分块算法的分解过程可用图 3.3 直观表示。

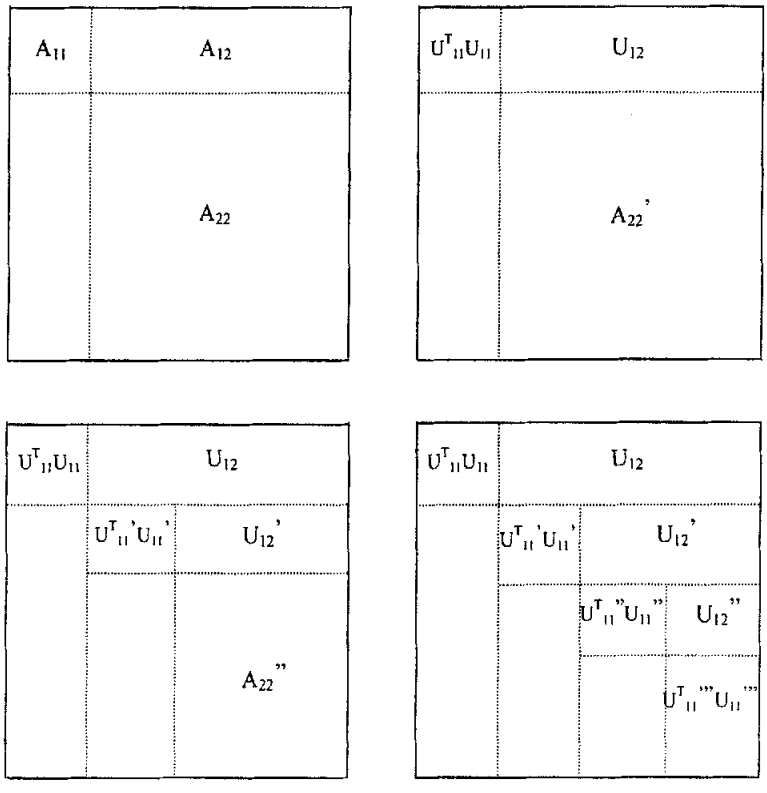

图 3.3　Cholesky 分解分块算法的图示

与 LU 分解分块算法一样，上述 Cholesky 分解分块算法不增加浮点运算量，也不需要增加额外的存储单元，能够较好地利用高缓和调用 level-3 BLAS，其运算效率要比不分块的算法提高许多。

3.4 Cholesky 分解递归算法

同样以上三角情形即式（2.20）为例讨论 Cholesky 分解递归算法。设矩阵 A 的阶数为 n×n，令 p=n/2（p 取整数），将式（2.20）写成分块形式

$$\begin{pmatrix} A_{11} & A_{12} \\ & A_{22} \end{pmatrix} = \begin{pmatrix} U_{11}^{T} & \\ U_{12}^{T} & U_{22}^{T} \end{pmatrix} \begin{pmatrix} U_{11} & U_{12} \\ & U_{22} \end{pmatrix} \tag{3.27}$$

式中，子矩阵 A_{11} 和 U_{11} 的阶数为 p×p，A_{12} 和 U_{12} 的阶数为 p×(n-p)，A_{22} 和 U_{22} 的阶数为(n-p)×(n-p)。其中，A_{11}、U_{11}、A_{22}、U_{22} 为上三角阵，A_{12}、U_{12} 为矩形阵。计算式（3.27）可得

$$A_{11}=U_{11}^{T}U_{11} \tag{3.28}$$

$$A_{12}=U_{11}^{T}U_{12} \tag{3.29}$$

$$A_{22}=U_{12}^{T}U_{12}+U_{22}^{T}U_{22} \tag{3.30}$$

式（3.28）为对子矩阵 A_{11} 进行 Cholesky 分解。在解出 U_{11}（亦即 U_{11}^{T}）后，由式（3.29）计算 U_{12}

$$U_{12}=(U_{11}^{T})^{-1}A_{12} \tag{3.31}$$

然后通过式（3.30）计算 U_{22}。将式（3.30）改写成

$$A_{22}-U_{12}^{T}U_{12}=U_{22}^{T}U_{22} \tag{3.32}$$

令

$$A_{22}'=A_{22}-U_{12}^{T}U_{12} \tag{3.33}$$

则式（3.32）成为

$$A_{22}'=U_{22}^{T}U_{22} \tag{3.34}$$

式（3.33）的含义为通过 U_{12} 对 A_{22} 进行修正或更新。修正后的子矩阵 A_{22}' 仍为上三角阵（对称正定）。式（3.34）为对 A_{22}' 进行 Cholesky 分解。

这样，原来n×n阶矩阵A的Cholesky分解运算就转化成大小为(n/2)×(n/2)的分块子矩阵 A_{11}、U_{11}、A_{12}、U_{12}和 A_{22}的运算。下一步对式（3.28）和式（3.34）中子矩阵 A_{11}和 A_{22}' 的Cholesky分解可以进行同样的处理，将它们分别分解成大小为(n/4)×(n/4)的子矩阵的运算。这种过程可以一直重复下去，直到最终产生的子矩阵 A_{11}等足够小，或者更简单地，直到 A_{11}的阶数为1。此时 A_{11}只有一个元素，其Cholesky分解为 $U_{11}=A_{11}^{1/2}$。于是，Cholesky分解的递归算法可以描述为：

Cholesky分解A：$A=U^TU$ （递归过程）

如果n>1，则

$p=n/2$

Cholesky分解 A_{11}：$A_{11}=U_{11}^TU_{11}$ （递归调用）

解 U_{12}：$U_{11}^TU_{12}=A_{12}$

计算 A_{22}'：$A_{22}'=A_{22}-U_{12}^TU_{12}$

Cholesky分解 A_{22}'：$A_{22}'=U_{22}^TU_{22}$ （递归调用）

否则（n=1）

计算U：$U=A^{1/2}$

图3.4以4×4阶矩阵为例，用图形表示出递归分解的过程，图中用斜线分隔显示分解形成的上三角阵U和下三角阵L。

与LU分解递归算法一样，矩阵分块在递归求解的过程中自动完成，分块的大小逐级减小，并且均为方阵，使得算法具有很好的数据局部性。算法的结构非常简单，实现起来很方便。整个算法的算术运算除了一个平方根运算外，主要由式（3.29）和式（3.33）构成，它们都可以调用高效率的level-3 BLAS完成。

Cholesky分解依赖于矩阵A的正定性，正定性可以保证不必选主元。

3.5 小结

本章在介绍LU分解和Cholesky分解分块算法的基础上，对LU分解和Cholesky分解递归算法进行了详细的推导，给出了算法的描述。按第一章给出的

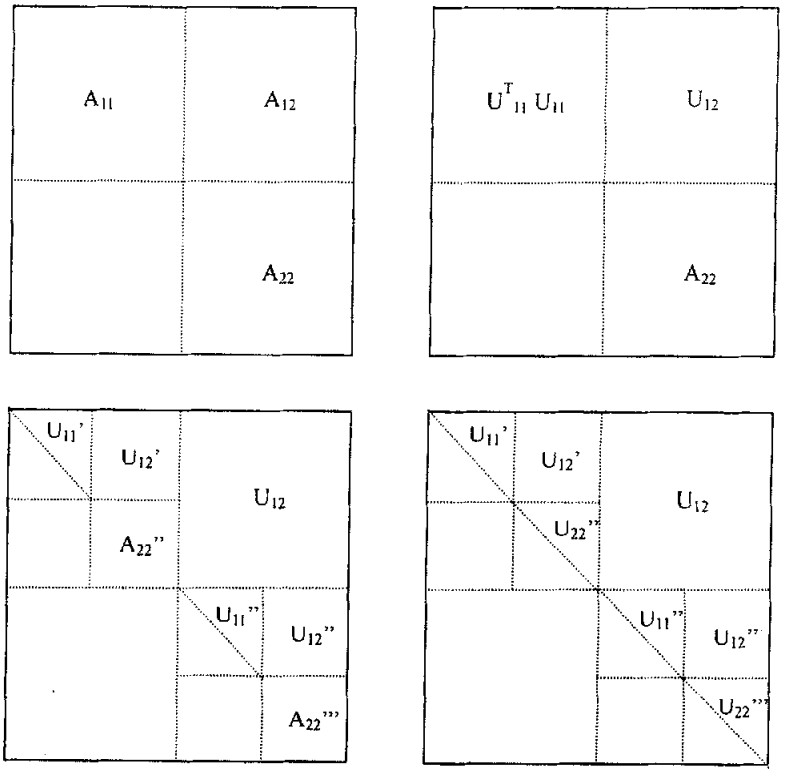

图 3.4 Cholesky 分解递归算法的分解过程

公式直接计算 LU 分解或 Cholesky 分解，为按列或按行计算，属于向量与向量相乘，不能很好地利用计算机的高缓和调用高效率的 level-3 BLAS,运算效率不高。通过将矩阵分块，使大矩阵的 LU 分解或 Cholesky 分解计算转化成若干个小的分块子矩阵的运算。由于分块子矩阵的尺寸减小，因而能更好地利用高缓；同时子矩阵间的运算可以调用高效的 level-3 BLAS，使得运算效率提高。递归算法利用递归本身的机制，能自动地产生矩阵分块，分块子矩阵的阶数逐级减小，趋于方阵，具有良好的数据局部性，适合当今计算机分层多级存储的结构。同时，递归算法结构简单，易于实现。

第四章 矩阵三角分解递归算法的实现

4.1 FORTRAN 语言的特点

FORTRAN 是世界上最早出现的高级编程语言。FORTRAN 是 FORmula TRANslate（公式翻译）的缩写，所以 FORTRAN 语言的主要用途是科学计算。FORTRAN 语言自上个世纪 50 年代诞生以来不断发展，从 FORTRAN Ⅰ、FORTRAN Ⅱ、FORTRAN Ⅳ到 1972 年形成 FORTRAN66 标准。在这以后，FORTRAN 语言受到了国际上的广泛接受，统治了几乎所有的数值计算领域。随着计算机软硬件技术的发展和结构化程序设计方法的提出，FORTRAN66 又经过修改和扩充，于 1978 年形成了 FORTRAN77 标准。FORTRAN77 公布之后在国际上得到广泛应用，多数计算机系统都配备了 FORTRAN77 编译程序，国内外关于 FORTRAN77 的教材、参考资料和应用程序相当丰富。但 FORTRAN77 还不是完全结构化的语言。国际上为了推动 FORTRAN 语言的结构化和现代化作了不懈的努力，经过长时间酝酿和研制，于 1991 年 5 月推出了 FORTRAN90 标准。之后不久又出现了 FORTRAN95，更新的标准目前又在准备之中。

FORTRAN90（95）对 FORTRAN77 做了较大的扩充和完善，新标准中增加了许多新的特性和功能。它们包括

（1） 引入了数组运算

（2） 提高了数值计算的功能

（3） 内在数据类型的参数化

（4） 用户定义的数据类型

（5） 用户定义的运算和赋值

（6） 引入模块数据及过程定义的功能

（7） 引入指针概念

（8） 引入更多的控制构造和递归过程

（9） 引入新的输入输出功能及动态可分配数组

与 C/C++等其它高级语言相比，FORTRAN 语言在科学计算方面具有得天独厚的优越性。首先，FORTRAN 语言从诞生到现在 40 多年的历史中，在科学技术研究的

各个领域积累了大量的 FORTRAN 语言程序，其中很多是经过长期实践证明是正确、可靠的，如果用其它语言重新编写，其正确性和可靠性都还要再由实践和时间来验证。FORTRAN 语言标准的历次修订都尽量保持向下兼容，这就使前人编写的 FORTRAN 程序可以不改动或只需很小改动就可以供现在使用。其次，FORTRAN 语言书写和语法要求严格，更适合严谨的科学计算。例如，C 语言中的数组不提供越界检查，这使数组的应用更加灵活。但是在科学计算方面数组的这种“散漫”用法是相当危险的，如果允许访问到错误的内存地址，其计算的结果是不可预测的，将会在科学和工程应用上造成不可估量的损失。第三，FORTRAN 语言可以直接对数组和复数进行运算。矩阵（包括向量）是科学计算最重要的的单元，科学计算中需要对矩阵进行大量的运算，在计算机中矩阵是以数组的形式存储的。第四，在并行计算方面，FORTRAN 语言仍是不可替代的。巨型计算机的实现，包括 IBM 的深蓝，我国的银河、曙光系列，以及大型多节点计算机的实现，无一不是依靠并行处理。近年来对 FORTRAN 语言的扩展中有很多是对数组的高水平操作，这些对并行优化是特别有利的。第五，FORTRAN 语言是一种编译语言，运行速度快。近年来 Matlab 语言非常流行，很大程度上得益于它对矩阵运算的简化。但它是类似于 Basic 语言的一种解释语言，其循环效率非常低。在 Matlab 中如果要大量使用循环，就不得不调用 C/C++或 FORTRAN 程序。科学计算中普遍存在大量的循环，这就使得 Matlab 语言的应用受到很大制约，而 FORTRAN 语言则显得得心应手。

正是由于 FORTRAN 语言的上述优点，使得数十年来 FORTRAN 语言一直占据科学计算的主导地位。建筑、气象、航空等许多领域涉及大量数据计算的应用软件均采用 FORTRAN 语言编制。包括基本矩阵运算、解线性方程组、特征值问题求解等运算在内的通用数值计算软件如 BLAS、LINPACK、LAPACK 等都无一例外是使用 FORTRAN 语言编写的。

4.2 基本线性代数子程序库 BLAS

BLAS 是 Basic Linear Algebra Subprograms（基本线性代数子程序库）的缩写。据统计，在大型科学工程计算中有 80%的部分与线性代数的计算有关。而各

种线性代数问题的计算中都涉及到大量的标量、向量和矩阵之间的基本运算，如向量和矩阵的相加、相乘、回代等等。BLAS 是针对这些广泛应用的基本线性代数运算而开发的高效子程序（函数）库。作为基本运算模块，其它数值计算软件或程序（如 LAPACK）就可以调用 BLAS 中的子程序来进行更复杂的线性代数运算，从而提高计算和编程效率。BLAS 经过 20 多年的发展和完善，是目前世界上最著名的标准基本线性代数子程序包，几乎被所有有关矩阵计算的软件所采用。

BLAS 库程序分为三类。第一类是作向量——向量运算，称为 Level-1 BLAS。它执行向量定标、向量拷贝、向量内积、寻找向量的最大分量等向量运算。第二类是作向量——矩阵运算，称为 Level-2 BLAS。它执行向量与矩阵相乘、矩阵秩 1 修正、求解三角系统等向量与矩阵的运算。第三类是作矩阵——矩阵运算，称为 Level-3 BLAS。它执行矩阵乘法、秩 k 修正、具有多右端项的矩阵三角系统求解等矩阵间的运算。

在这三类 BLAS 中，Level-1 BLAS 程序最简单，但运算效率低，不能很好地利用当今高性能计算机的性能，目前主要是为方便起见（可以直接调用）用于一些不重要的计算。Level-2 BLAS 在不少向量型处理器上能够达到接近峰值的性能，但在其它向量处理器或 RISC 处理器上，它们的性能受到不同级别存储器之间数据移动速率的限制。Level-3 BLAS 运算效率最高，它克服了上述 Level-2 BLAS 的限制，能够达到或基本达到当今各种高性能处理器的峰值性能。同时，Level-3 BLAS 还具有很好的并行计算性能。因此，设计和编写的数值计算软件或程序若能更多地调用 Level-3 BLAS，将会大大提高运算效率。

下面将本文涉及到的几个 BLAS 子程序的名称和功能简单作一介绍。

IDAMAX：Level-1 BLAS，找出一个向量中绝对值最大的元素的位置。

DSWAP：Level-1 BLAS，将两个向量中的元素进行交换。

DSCAL：Level-1 BLAS，用一个常数对一个向量进行定标（scale）。

DGER：Level-2 BLAS，执行矩阵秩 1 修正。运算形式为

$$A=\alpha x y^{T}+A$$

其中，α 是标量，x、y 为向量，A 为 $m\times n$ 阶矩阵。

DGEMV：Level-2 BLAS，执行矩阵与向量相乘。运算形式为

$$y=\alpha Ax+\beta y$$

其中，α、β是标量，x、y为向量，A为m×n阶矩阵，A的位置也可以是A^T。

DGEMM：Level-3 BLAS，执行一般矩阵的乘法运算。运算形式为

$$C=\alpha AB+\beta C,$$

其中，α、β为标量，A为m×k阶矩阵，B为k×n阶矩阵，C为为m×n阶矩阵，A和B的位置也可以是A^T和B^T。

DSYRK：Level-3 BLAS，执行对称矩阵的秩k修正。运算形式为

$$C=\alpha AA^T+\beta C \quad 或 \quad C=\alpha A^TA+\beta C$$

其中，α、β为标量，C为n×n阶对称矩阵，A为n×k或k×n阶矩阵。

DTRSM：Level-3 BLAS，执行三角矩阵方程的求解。运算形式为

$$AX=\alpha B \quad 或 \quad XA=\alpha B,$$

其中，α为标量，A为上三角或下三角矩阵，X和B为m×n阶矩阵；X为待求矩阵，解出后写入B。式中A的位置也可以是A^T。

4.3 线性代数软件包 LAPACK

LAPACK是Linear Algebra Package（线性代数软件包）的缩写。它是由美国国家自然科学基金(NSF)等资助开发的著名公开软件，1994年发布V2.0版，1999年发布了最新的V3.0版。LAPACK是针对现代高性能计算机与共享存储并行计算机设计和研制的线性代数软件包，由于其性能高、应用范围广，目前已被移植到多种平台上，并已成为线性代数计算事实上的标准。

LAPACK包含了求解科学与工程计算中最常见的数值线性代数问题，如求解线性方程组、线性最小二乘问题、特征值问题和奇异值问题。LAPACK还可以实现矩阵分解（如LU分解、Cholesky分解）和条件数估计等相关运算。在LAPACK中，采用了分块算法来提高数据的访问速度和数据的再利用；使用高效的Level-3 BLAS来执行大量的计算，从而获得更高的性能。

LAPACK是一个庞大的软件包，约有30万行FORTRAN源代码。LAPACK中的程

序可分为驱动程序（driver routine）、计算程序（computational routine）和辅助程序（auxiliary routine）三类。驱动类程序进行标准问题的求解，如解线性方程组和计算实对称矩阵特征值。计算类程序执行各种计算任务，如 LU 分解、三角方程组的回代求解。这类程序通常是被驱动类程序调用作为其中的计算模块。辅助类程序执行上述两类程序用到的各种各样的基本操作或子任务。它们也可以用于其它软件的开发。LAPACK 支持单精度、双精度和复数等多种数据类型，提供普通矩阵、带状矩阵、正定矩阵以及三对角矩阵等多种矩阵类型的计算程序。

下面对本文涉及到的几个 LAPACK 程序作一介绍。

（1） DGETF2

DGETF2 执行普通 m×n 阶矩阵 A 的 LU 分解计算，采用不分块的算法。带有部分选主元，即选取列主元进行行交换。其分解形式为 A=PLU，P 为排列矩阵，L 为下三角阵（若 m>n，其对角元素为 1），U 为上三角阵（若 m<n，其对角元素为 1）。程序的主要语句段如下：

```
      DO 10 J = 1, MIN( M, N )
*        Find pivot and test for singularity.
         JP = J - 1 + IDAMAX( M-J+1, A( J, J ), 1 )
         IPIV( J ) = JP
         IF( A( JP, J ).NE.ZERO ) THEN
*           Apply the interchange to columns 1:N.
            IF( JP.NE.J )
     $         CALL DSWAP( N, A( J, 1 ), LDA, A( JP, 1 ), LDA )
*           Compute elements J+1:M of J-th column.
            IF( J.LT.M )
     $         CALL DSCAL( M-J, ONE / A( J, J ), A( J+1, J ), 1 )
         ELSE IF( INFO.EQ.0 ) THEN
            INFO = J
         END IF
```

```
            IF( J.LT.MIN( M, N ) ) THEN
*              Update trailing submatrix.
               CALL DGER( M-J, N-J, -ONE, A( J+1, J ), 1, A( J, J+1 ), LDA,
     $                    A( J+1, J+1 ), LDA )
            END IF
   10    CONTINUE
```

程序中的主要运算为调用 DSWAP、DSCAL 和 DGER，其中核心运算为 DGER，它是 Level-2 BLAS。DSWAP、DSCAL 为 Level-1 BLAS。这些子程序的功能前面已作介绍。

（2） DGETRF

DGETRF 执行与上面相同的普通 m×n 阶矩阵 A 的 LU 分解计算，采用的是分块算法。程序的主要语句为：

```
*  Determine the block size for this environment.
      NB = ILAENV( 1, 'DGETRF', ' ', M, N, -1, -1 )
      IF( NB.LE.1 .OR. NB.GE.MIN( M, N ) ) THEN
*        Use unblocked code.
         CALL DGETF2( M, N, A, LDA, IPIV, INFO )
      ELSE
*  Use blocked code.
         DO 20 J = 1, MIN( M, N ), NB
            JB = MIN( MIN( M, N )-J+1, NB )
*  Factor diagonal and subdiagonal blocks and test for exact singularity.
            CALL DGETF2( M-J+1, JB, A( J, J ), LDA, IPIV( J ), IINFO )
*        Adjust INFO and the pivot indices.
            IF( INFO.EQ.0 .AND. IINFO.GT.0 )
     $         INFO = IINFO + J - 1
            DO 10 I = J, MIN( M, J+JB-1 )
```

```
                IPIV( I ) = J - 1 + IPIV( I )
   10         CONTINUE
*          Apply interchanges to columns 1:J-1.
           CALL DLASWP( J-1, A, LDA, J, J+JB-1, IPIV, 1 )
           IF( J+JB.LE.N ) THEN
*          Apply interchanges to columns J+JB:N.
              CALL DLASWP( N-J-JB+1, A( 1, J+JB ), LDA, J, J+JB-1,
     $                     IPIV, 1 )
*             Compute block row of U.
              CALL DTRSM( 'Left', 'Lower', 'No transpose', 'Unit', JB,
     $                    N-J-JB+1, ONE, A( J, J ), LDA, A( J, J+JB ),
     $                    LDA )
              IF( J+JB.LE.M ) THEN
*          Update trailing submatrix.
                 CALL DGEMM( 'No transpose', 'No transpose', M-J-JB+1,
     $                       N-J-JB+1, JB, -ONE, A( J+JB, J ), LDA,
     $                       A( J, J+JB ), LDA, ONE, A( J+JB, J+JB ),
     $                       LDA )
              END IF
           END IF
   20    CONTINUE
```

程序中分块子矩阵的 LU 分解计算调用的是上述 DGETF2 程序。求解 U_{12}和计算 A_{22}'调用的是 Level-3 BLAS DTRSM 和 DGEMM。子程序 ILAENV 和 DLASWP 是 LAPACK 中的辅助程序，ILAENV 用于确定当前运行环境下的最佳分块尺寸，DLASWP 用于执行行交换操作。与 DGETF2 相比，DGETRF 由于采取了矩阵分块，且有一半以上的运算是由 Level-3 BLAS 完成的，其运算速度一般要比 DGETF2 快。

（3） DPOTF2

DPOTF2 执行实对称正定矩阵 A 的 Cholesky 分解，采用不分块的算法。其分解形式为 $A=U^TU$（使用 A 的上三角部分元素）或 $A=LL^T$（使用 A 的下三角部分元素），其中 U 和 L 分别为上三角和下三角矩阵。下面给出计算 $A=U^TU$ 的主要程序语句：

```
*        Compute the Cholesky factorization A = U'*U.
         DO 10 J = 1, N
*           Compute U(J,J) and test for non-positive-definiteness.
            AJJ = A( J, J ) - DDOT( J-1, A( 1, J ), 1, A( 1, J ), 1 )
            IF( AJJ.LE.ZERO ) THEN
               A( J, J ) = AJJ
               GO TO 30
            END IF
            AJJ = SQRT( AJJ )
            A( J, J ) = AJJ
*           Compute elements J+1:N of row J.
            IF( J.LT.N ) THEN
               CALL DGEMV( 'Transpose', J-1, N-J, -ONE, A( 1, J+1 ),
     $                     LDA, A( 1, J ), 1, ONE, A( J, J+1 ), LDA )
               CALL DSCAL( N-J, ONE / AJJ, A( J, J+1 ), LDA )
            END IF
   10    CONTINUE
```

程序中的主要运算为调用 DGEMV 和 DSCAL，核心运算为 DGEMV，是 Level-2 BLAS，DSCAL 为 Level-1 BLAS。

（4） DPOTRF

DPOTRF 同样执行实对称正定矩阵 A 的 Cholesky 分解，但采用分块算法。其分解形式与上相同。下面给出计算 $A=U^TU$ 的主要程序语句：

```
*        Determine the block size for this environment.
```

```
      NB = ILAENV( 1, 'DPOTRF', UPLO, N, -1, -1, -1 )
      IF( NB.LE.1 .OR. NB.GE.N ) THEN
*        Use unblocked code.
         CALL DPOTF2( UPLO, N, A, LDA, INFO )
      ELSE
*        Use blocked code.
            DO 10 J = 1, N, NB
*     Update and factorize the current diagonal block and test for
*     non-positive-definiteness.
               JB = MIN( NB, N-J+1 )
               CALL DSYRK( 'Upper', 'Transpose', JB, J-1, -ONE,
     $                     A( 1, J ), LDA, ONE, A( J, J ), LDA )
               CALL DPOTF2( 'Upper', JB, A( J, J ), LDA, INFO )
               IF( INFO.NE.0 )  GO TO 30
               IF( J+JB.LE.N ) THEN
*                 Compute the current block row.
                  CALL DGEMM( 'Transpose', 'No transpose', JB, N-J-JB+1,
     $                        J-1, -ONE, A( 1, J ), LDA, A( 1, J+JB ),
     $                        LDA, ONE, A( J, J+JB ), LDA )
                  CALL DTRSM( 'Left', 'Upper', 'Transpose', 'Non-unit',
     $                        JB, N-J-JB+1, ONE, A( J, J ), LDA,
     $                        A( J, J+JB ), LDA )
               END IF
   10       CONTINUE
```

程序中分块子矩阵的 Cholesky 分解计算调用的是上面的 DPOTF2 程序，其余运算调用 Level-3 BLAS DSYRK、DGEMM 和 DTRSM。DPOTRF 大部分的运算是由 Level-3 BLAS 完成的，其运算效率要高于 DPOTF2。

以上介绍的 BLAS 和 LAPACK 程序的数据类型都是双精度，程序名的首字母为 D，本文后面都将针对双精度数据类型讨论。

4.4 LU 分解递归算法的 FORTRAN90 实现

4.4.1 FORTRAN90 的递归过程

递归过程是本文用到的 FORTRAN90 的主要功能。FORTRAN77 不具备递归功能，若用其实现递归算法，递归过程中递归的级数、分块子矩阵的位置、阶数等参数的堆栈管理必须通过程序语句实现，比较复杂。FORTRAN90 支持递归过程，递归自动地由编译器处理，非常便于递归算法的实现，产生的程序简单、高效。

所谓递归过程就是在过程内直接或间接地调用该过程自身。递归过程有两种：递归函数和递归子程序。本文采用递归子程序，其一般形式为：

RECURSIVE SUBROUTINE 子程序名（[哑元名表]）
[说明部分]
[执行部分]
[内部辅程序部分]
END SUBROUTINE 子程序名

递归子程序与一般子程序的定义类似，区别只是在 SUBROUTINE 前加上 RECURSIVE，表明该子程序是递归型的，这样在程序的内部就可以自己调用自己，即直接 CALL 该子程序名，使用非常方便。

4.4.2 算法的实现程序

采用上面介绍的 FORTRAN90 的递归过程，第 3.2 节推导的 LU 分解递归算法可以很方便地用 FORTRAN90 语言实现。整个递归分解过程定义为递归子程序，算法中求解 U_{12}（即式 $L_{11}U_{12}=A_{12}$）的计算，L_{11} 为三角阵，属三角矩阵方程的求解，可以调用 Level-3 BLAS 程序 DTRSM。A_{22}' 的计算（即式 $A_{22}'=A_{22}-L_{21}U_{12}$），可以调

用 Level-3 BLAS 程序 DGEMM。通过 P_1 和 P_2 对 $\begin{pmatrix} A_{12} \\ A_{22} \end{pmatrix}$ 和 A_{21} 的换行操作可调用 LAPACK 辅助程序 DLASWP。最终找主元以及各元素除以主元的操作可通过 BLAS 程序 IDAMAX 和 DSCAL 来完成。相应的程序语句为

```
RECURSIVE SUBROUTINE RLUF(M,N,A,PIV)
    IMPLICIT NONE
    INTEGER, INTENT(IN) :: M,N
    DOUBLE PRECISION, DIMENSION(1:M,1:N), INTENT(INOUT) :: A
    INTEGER, DIMENSION(1:M), INTENT(OUT) :: PIV
    INTEGER :: P,IPIV,IDAMAX,I
    DOUBLE PRECISION :: STMP
    INTEGER, DIMENSION(1:M) :: PIV2
    DOUBLE PRECISION, PARAMETER :: ONE=1.D0

    DO I=1,M
       PIV(I)=I
       PIV2(I)=0
    ENDDO
    IF (N .EQ. 1) THEN
       IPIV=IDAMAX(M,A(1:M,1:1),1)
       PIV(1)=PIV(IPIV)
       STMP=A(1,1)
       A(1,1)=A(IPIV,1)
       A(IPIV,1)=STMP
       IF (M .GT. 1) THEN
          CALL DSCAL(M-1,ONE/A(1,1),A(2,1),1)
       ENDIF
    ELSE
```

```
        P=N/2
        CALL RLUF(M,P,A(1:M,1:P),PIV(1:M))
        CALL DLASWP(N-P,A(1:M,P+1:N),M,1,M,PIV(1:M),1)
        CALL DTRSM('L','L','N','U',P,N-P,ONE,A(1:P,1:P),P,A(1:P,P+1:N),P)
        CALL DGEMM('N','N',M-P,N-P,P,-ONE,A(P+1:M,1:P),M-P,A(1:P,P+1:N),&
                    P,ONE,A(P+1:M,P+1:N),M-P)
        CALL RLUF(M-P,N-P,A(P+1:M,P+1:N),PIV2(P+1:M))
        CALL DLASWP(P,A(P+1:M,1:P),M-P,1,M-P,PIV2(P+1:M),1)
        DO I=1,M-P
          IF (PIV2(P+I) .NE. I) PIV(P+I)=PIV(P+PIV2(P+I))
        ENDDO
      ENDIF
    END SUBROUTINE RLUF
```

程序中第一句Recursive Subroutine RLUF说明子程序RLUF为递归型子程序，即可以自己调用自己。程序的输入参数，A为待分解矩阵，M和N为A的阶数，PIV为排列矩阵，在分解过程中产生。这些参数在每次递归调用时包含分块子矩阵的信息。程序中第25行和30行的CALL RLUF即为递归调用该子程序。程序内调用各BLAS和LAPACK程序的参数分别按相应各子程序的要求设置，反映了参与运算的相关子矩阵的大小、元素位置、矩阵形态等信息。程序的数据类型为双精度。

由上述程序可见，整个程序的核心运算为调用Level-3 BLAS DGEMM和DTRSM，可最大限度地发挥当今高性能计算机的运算效率。

4.5 Cholesky分解递归算法的FORTRAN90实现

4.5.1 算法的实现程序

根据3.4节的算法描述，Cholesky分解递归算法同样可以用FORTRAN90的递归型子程序实现。算法中求解U_{12}即式$U_{11}^{T}U_{12}=A_{12}$的计算，U_{11}^{T}为三角阵，

属三角矩阵方程的求解，可以调用 Level-3 BLAS 程序 DTRSM。A_{22} 的修正即式 $A_{22}' = A_{22} - U_{12}^{T}U_{12}$ 的计算，A_{22} 为对称阵，属于对称阵的秩 k 修正，可以调用 Level-3 BLAS 程序 DSYRK。最终形成的程序为

```
RECURSIVE SUBROUTINE RPOTRF(N,A,LDA)
  IMPLICIT NONE
  INTEGER, INTENT(IN) :: N, LDA
  DOUBLE PRECISION,INTENT(INOUT) :: A(LDA,*)
  INTEGER :: P
  DOUBLE PRECISION,PARAMETER :: ONE=1.D0
  IF (N .EQ. 1) THEN
    A(1,1)=SQRT(A(1,1))
  ELSE
    P=N/2
    CALL RPOTRF(P,A,LDA)
    CALL DTRSM('L','U','T','N',P,N-P,ONE,A(1,1),LDA,A(1,P+1),LDA)
    CALL DSYRK('U','T',N-P,P,-ONE,A(1,P+1),LDA,ONE,A(P+1,P+1),LDA)
    CALL RPOTRF(N-P,A(P+1,P+1),LDA)
  ENDIF
END SUBROUTINE RPOTRF
```

程序的输入参数，A 为待分解矩阵，N 为 A 的阶数，它们在每次递归调用时则反映分块子矩阵的信息。参数 LDA 为矩阵 A 的主导尺寸（Leading Dimension），它在递归调用过程中保持不变。只要给出每个分块子矩阵第一个元素的位置，就可以根据该 LDA 参数访问各个分块子矩阵的元素。这样，在每一次调用 RPOTRF、DTRSM 和 DSYRK 对分块子矩阵进行计算时，就不需要对有关子矩阵的元素进行拷备，节省了内存的使用。程序的结构非常简单、清晰，整个运算几乎全部由 Level-3 BLAS（DTRSM 和 DSYRK）完成，保证了算法在现代高性能计算机上的高效运行。

4.5.2 矩阵元素的重排

上一节讨论的程序在递归计算 TRSM 和 SYRK 的过程中，使用的是固定的 LDA 参数来访问分块子矩阵的元素。尽管随着递归运算的进行，需要计算的子矩阵的尺寸越来越小，但存储器访问的跨距（memory stride）保持不变并且很大（等于矩阵的阶数）。如果我们将递归产生的分块子矩阵的元素重新排列，使各个子矩阵的元素依次集中放在一起，那么计算时随着递归的进行，存储器访问的跨距就会越来越小，从而能减小存储器访问时间，提高运算速度。

下面以 8 阶矩阵为例，说明矩阵元素重排的过程。为了方便起见，用顺序数字代表矩阵元素，虚线表示逐级递归分解产生的分块。输入矩阵为

1	9	17	25	33	41	49	57
2	10	18	26	34	42	50	58
3	11	19	27	35	43	51	59
4	12	20	28	36	44	52	60
5	13	21	29	37	45	53	61
6	14	22	30	38	46	54	62
7	15	23	31	39	47	55	63
8	16	24	32	40	48	56	64

第一级递归分解，矩阵元素重排后的顺序为

1	17	5	21	33	49	37	53
2	18	6	22	34	50	38	54
3	19	7	23	35	51	39	55
4	20	8	24	36	52	40	56
9	25	13	29	41	57	45	61
10	26	14	30	42	58	46	62
11	27	15	31	43	59	47	63
12	28	16	32	44	60	48	64

第二级递归分解，矩阵元素的重排顺序为

1	17	5	21	33	49	37	53
2	18	6	22	34	50	38	54
9	25	7	23	35	51	45	61
10	26	8	24	36	52	46	62
3	19	13	29	41	57	39	55
4	20	14	30	42	58	40	56
11	27	15	31	43	59	47	63
12	28	16	32	44	60	48	64

第三级即最后一级递归分解将 2×2 阶的子矩阵分解成 1×1 阶的分块，矩阵元素的重排顺序与前一级相同。

在重排后的数据格式下，递归产生的每个分块子矩阵的元素均以按列存放的方式连续放在一起。给定每个子矩阵第一个元素的地址，则子矩阵中的元素就可以使用大小等于该子矩阵尺寸的 LD 参数来访问。由于每一级递归分解子矩阵的尺寸减小一半，使得 LD 参数即存储器访问的跨距越来越小，从而减少存储器访问时间。同时，各个子矩阵的元素集中连续放在一起，在递归计算时，参与运算的有关子矩阵的元素更容易一起同时全部进入高缓，提高了高缓的使用效率。

一般情况下，输入矩阵的元素是按自然顺序排列的，需要将它们转换成上述分块矩阵元素顺序的排列格式，相应的实现矩阵元素重排的程序为

```
SUBROUTINE REORD(A, M, BUF, N)
    IMPLICIT NONE
    INTEGER, INTENT(IN) :: M, N
    DOUBLE PRECISION, INTENT(INOUT) :: A(M,*)
    DOUBLE PRECISION, INTENT(INOUT) :: BUF(N,*)
    INTEGER :: I, J, K, L, II, JJ

!---------- move A12 to BUF ----------
```

```
      DO J = 1, M-N
         DO I = 1, N
            BUF(I, J) = A(I, J+N)
         ENDDO
      ENDDO
!---------- reorder A22 ----------
      K = -1
      L = M + M*M
      DO J = M, N+1, -1
         DO I = M, N+1, -1
            K = K + 1
            JJ = M - K/M
            II = L - K - JJ*M
            A(II, JJ) = A(I, J)
         ENDDO
      ENDDO
!---------- reorder A12 ----------
      DO J = M-N, 1, -1
         DO I = N, 1, -1
            K = K + 1
            JJ = M - K/M
            II = L - K - JJ*M
            A(II, JJ) = BUF(I, J)
         ENDDO
      ENDDO
!---------- move A11 to BUF ----------
      DO J = 1, N
```

```
            DO I = 1, N
                BUF(I, J) = A(I, J)
            ENDDO
        ENDDO
!---------- reorder A21 ----------
        DO J = N, 1, -1
            DO I = M, N+1, -1
                K = K + 1
                JJ = M - K/M
                II = L - K - JJ*M
                A(II, JJ) = A(I, J)
            ENDDO
        ENDDO
!---------- reorder A11 ----------
        DO J = N, 1, -1
            DO I = N, 1, -1
                K = K + 1
                JJ = M - K/M
                II = L - K - JJ*M
                A(II, JJ) = BUF(I, J)
            ENDDO
        ENDDO
    END SUBROUTINE REORD
```

针对元素重排后的矩阵，Cholesky 分解递归算法的计算程序则相应为

```
    RECURSIVE SUBROUTINE RCHOL(A, M, BUF, N)
        IMPLICIT NONE
```

```
      INTEGER, INTENT(IN) :: M, N
      DOUBLE PRECISION, INTENT(INOUT) :: A(M,*)
      DOUBLE PRECISION, INTENT(INOUT) :: BUF(N,*)
      INTEGER :: P, L, I2, J2
      DOUBLE PRECISION, PARAMETER :: ONE=1.D0

      IF (M .EQ. 1) THEN
         A(1,1) = SQRT( A(1,1) )
      ELSE
         P = M/2
         L = 2*M*P - P*P
         J2 = 1 + L/M
         I2 = 1 + M + L - J2*M
         CALL REORD(A(1,1), M, BUF, P)
         CALL RCHOL(A(1,1), P, BUF, P/2)
         CALL DTRSM('L','U','T','N', P, M-P, ONE, A(1,1), P, A(1,P+1), P)
         CALL DSYRK('U','T', M-P, P, -ONE, A(1,P+1), P, ONE, A(I2,J2), M-P)
         CALL RCHOL(A(I2,J2), M-P, BUF, (M-P)/2)
         CALL RECOV(A(1,1), M, BUF, P)
      ENDIF
END SUBROUTINE RCHOL
```

程序中，子程序 RECOV 将分块排列的元素格式再恢复成原先的自然顺序。其程序语句与前面元素重排的程序 REORD 类似，不再列出。

4.6 小结

本章简要介绍了 FORTRAN90 语言的特点和性能，对与本文工作有关的基本线性代数子程序库BLAS和线性代数软件包LAPACK及其中的有关程序也作了必

要的介绍和描述。在此基础上，运用 FORTRAN90 语言对第二章推导得出的 LU 分解和 Cholesky 分解递归算法进行了实现，给出了主要程序语句，并对有关实现技术和方法进行了阐述。FORTRAN90 语言支持递归过程，递归自动地由编译器处理，非常便于递归算法的实现。BLAS 为通用的标准线性代数子程序库，包含各种线性代数的基本运算，为各类数值计算软件所调用，其程序分为 level-1、level-2 和 level-3 三类，其中以 level-3 最能充分利用当今高性能计算机的优越性能，运算效率最高。LAPACK 是目前常用的线性代数计算软件包，其中包括计算矩阵 LU 分解和 Cholesky 分解的程序，这些程序后面将用于进行算法性能的比较。

本章给出的 LU 分解和 Cholesky 分解递归算法实现程序采用 FORTRAN90 的递归功能，主要运算均通过调用高效率的 level-3 BLAS 完成，保证了算法在现代高性能计算机上的高效运行。

第五章 算法的测试与运行结果

5.1 算法的测试

算法和程序的测试包含三个方面，一是正确性的测试，二是运算时间的测量，三是与现有其它算法和程序的比较。

5.1.1 正确性的测试

正确性的测试是验证所研制的算法和程序的计算是否正确。这可以通过将所编程序的运算结果与 LAPACK 中的同类程序的计算结果进行比较来进行。对同一个输入矩阵，将本文开发的 LU 分解递归算法程序 RLUF 与 LAPACK 中的 LU 分解程序 DGETRF 或 DGETF2 的运算结果进行比较，若两者结果一致，则运算正确。同样地，将 Cholesky 分解递归算法程序 RPOTRF 与 LAPACK 中的 Cholesky 分解程序 DPOTRF 或 DPOTF2 的运算结果进行比较，结果相同则运算正确。为了保证验证结果具有普遍意义，所用输入矩阵的元素应随机产生，这一点可利用 LAPACK 中的辅助类程序 DLAGGE 和 DLAGSY。其中 DLAGGE 随机产生一个 m×n 阶的一般矩阵，DLAGSY 随机产生一个 n 阶的对称矩阵，具体调用形式后面测试程序中给出。

5.1.2 运算速度的测试

在验证了算法和程序的正确性以后，下一步就是程序的运行时间即程序运算速度的测试。我们可以用读取计算机系统时间的方法来进行测量，具体做法为：在被测试的程序（如 RLUF）运行开始时读取一下系统时间，在程序运行结束时再读取一下系统时间，后者减去前者即为所测程序的运行时间。FORTRAN 语言提供几个与系统时间有关的内部子函数，如 CPU_TIME、SYSTEM_CLOCK、GETTIM 等。本文采用 CPU_TIME，它返回的是以秒为单位的 CPU 时间，精确到 0.01 秒。其调用形式为 CALL CPU_TIME(time)，参数 time 为实数。

算法和程序测试的另一个任务就是与其它具有代表性的算法和程序进行比较，这方面 LAPACK 程序是一个很好的标准。在运行和测试本文所开发程序的同时，

我们在同一个测试程序中，针对同一个输入矩阵，采用同样的测量方法，调用和运行 LAPACK 程序，测量和记录其运行时间，与本文开发的程序进行比较。

5.1.3 测试程序

下面以 LU 分解递归算法（RLUF）为例，给出测试程序，其中参与比较的有 LAPACK 的分块算法程序 DGETRF 和不分块的算法程序 DGETF2。

```
PROGRAM MAIN
    IMPLICIT NONE
    DOUBLE PRECISION, ALLOCATABLE :: A(:,:), B(:,:), D(:), WORK(:)
    INTEGER, ALLOCATABLE :: PIV(:)
    INTEGER, DIMENSION(1:4) :: ISEED
    INTEGER :: INFO
    INTEGER :: I,N,M
    REAL :: TIME_BEGIN, TIME_END, TIME, ERROR

    WRITE(*,*) 'Type an integer (N)'     ! 键盘输入矩阵阶数
    READ(*,*) N
    WRITE(*,*) 'N=', N

    ALLOCATE( A(N,N) )
    ALLOCATE( B(N,N) )
    ALLOCATE( PIV(N) )
    ALLOCATE( D(N), WORK(N+N) )
    M=N
!------ 产生输入矩阵 A ------
    DO I=1,N
        D(I)=DBLE(I)
```

```
      ENDDO
      ISEED(1)=10
      ISEED(2)=987
      ISEED(3)=400
      ISEED(4)=1
     !------ 调用 DLAGGE 随机产生 A ------
      CALL DLAGGE(N,N,N-1,N-1,D,A,N,ISEED,WORK,INFO)
      B = A    ! 保存 A 中的元素于 B
  !------ 运行和测试 DGETRF ------
      PIV = 0
      CALL CPU_TIME (TIME_BEGIN)
      CALL DGETRF(M,N,A,M,PIV,INFO)  ! 输入 A，算完后 A 中为计算结果
      CALL CPU_TIME (TIME_END)
      TIME=TIME_END-TIME_BEGIN    ! 得到 DGETRF 的运算时间
      WRITE(*,*) 'Time of operation for DGETRF was ', TIME, 'seconds'
      WRITE(*,*)
  !------ 运行和测试 DGETF2 ------
      A = B    ! 使用原来的输入矩阵 A
      PIV = 0
      CALL CPU_TIME (TIME_BEGIN)
      CALL DGETF2(M,N,A,M,PIV,INFO)    ! A 中存放计算结果
      CALL CPU_TIME (TIME_END)
      TIME=TIME_END-TIME_BEGIN    ! 得到 DGETF2 的运算时间
      WRITE(*,*) 'Time of operation for DGETF2 was ', TIME, 'seconds'
      WRITE(*,*)
```

```
!------ 运行和测试 RLUF ------
    PIV = 0
    CALL CPU_TIME (TIME_BEGIN)
    CALL RLUF(M,N,B,PIV)  ！输入 B 即为原来的 A，B 最后为计算结果
    CALL CPU_TIME (TIME_END)
    TIME=TIME_END-TIME_BEGIN    ！得到 RLUF 的运算时间
    WRITE(*,*) 'Time of operation for RLUF was ', TIME, 'seconds'
    WRITE(*,*)
!------ 验证运算正确性 ------
    A = A - B   ！比较 DGETF2 和 RLUF 的运算结果
    ERROR＝MAXVAL(ABS(A))
    WRITE(*,*) 'MAXVAL(ABS(A-B)) = ', ERROR
    WRITE(*,*)
END PROGRAM MAIN
```

程序中 ALLOCATABLE 为可分配数组属性（声明），ALLOCATE 则为相应的数组分配语句。利用这一对语句，可根据输入矩阵的大小，动态地给数组分配内存，提高数组的使用效率，动态数组分配也是 FORTRAN90 的一个新的功能。测试矩阵的阶数在程序开始运行时由键盘输入，这样便于对不同阶数的矩阵进行测试。数组 D、WORK 和 ISEED 为用于随机产生输入矩阵 A 所需的参数，根据 LAPACK 子程序 DLAGGE 的要求设置。语句 MAXVAL(ABS(A))用于比较本文开发的程序 RLUF 的运算结果与 LAPACK 程序 DGETF2 的结果是否一致。ABS()和 MAXVAL()均为 FORTRAN90 的内部函数，ABS(A)为取 A(=A-B)中元素的绝对值，MAXVAL(ABS(A))为取 ABS(A)中的最大值。若其结果为零（或接近于零），则 RLUF 的运算结果正确。用这种方法可以方便准确地验证程序的运算结果，无论矩阵的尺寸多大。程序中的矩阵运算 A＝B、A＝A－B 等使用的是直接对矩阵操作的语句，简便易读，它们也是 FORTRAN90 新增的功能。这些运算在 FORTRAN77 中需要用二重嵌套循环来实现。

Cholesky 分解递归算法的测试程序类似，不再列出。

5.1.4 测试平台和编译环境

测试设备采用 PC 计算机，CPU 为奔腾 4(带有 512 KB 的 L2 cashe)，主频 2.0 GHz，内存 256 MB，操作系统为 Windows XP。程序的编译采用支持 FORTRAN90 标准的 Microsoft Visual FORTRAN5.0。用于比较的有关 LAPACK 程序以及程序中调用到的 LAPACK 和 BLAS 子程序均取自于 LAPACK 和 BLAS 程序库。

5.2 运行结果

我们在不同矩阵阶数下分别对 LU 分解递归算法程序和 Cholesky 分解递归算法程序的运算速度进行了测试，并与相应的 LAPACK 程序进行了比较。结果表明，随着矩阵阶数的增大，本文研究产生的递归算法程序的性能优于 LAPACK 程序，下面给出具体的测试数据。

5.2.1 LU 分解递归算法

表 5.1 给出了从 300 阶到 2000 阶不同矩阵阶数下 RLUF、DGETRF 和 DGETF2 的运算时间。其中，RLUF 为第 4.4.2 节中给出的 LU 分解递归算法程序，DGETRF 为 LAPACK 中的 LU 分解分块算法程序，DGETF2 为 LAPACK 中的 LU 分解不分块的算法程序。表格中第 5 行数据为 RLUF 比 DGETRF 速度提高的百分比。

表 5.1 RLUF 与 DGETRF 及 DGETF2 的运算速度比较（时间单位：S）

阶数 时间 算法	300	400	500	600	700	800	900	1000	1100
RLUF	0.60	1.65	3.24	5.71	9.23	13.57	19.28	26.53	34.66
DGETRF	0.49	0.77	1.71	6.37	10.54	16.04	22.79	31.36	41.80
DGETF2	1.10	2.75	5.11	9.34	14.78	21.80	31.14	42.79	55.97
速度提高%	-22.5	-114	-89.5	10.4	12.4	15.4	15.4	15.4	17.1

表 5.1（续）

阶数 时间 算法	1200	1300	1400	1500	1600	1700	1800	1900	2000
RLUF	44.60	56.41	70.58	85.79	104.5	124.5	148.0	171.8	207.3
DGETRF	54.60	68.88	86.28	106.1	129.5	154.1	182.9	214.7	254.2
DGETF2	72.39	91.34	113.9	139.8	170.6	202.5	239.9	285.2	330.9
速度提高%	18.3	18.1	18.2	19.1	19.3	19.2	19.1	20	18.5

由表 5.1，当矩阵阶数大于 600 后，递归算法 RLUF 比 LAPACK 分块算法 DPOTRF 速度提高 10%～20%。表 5.1 还表明，LAPACK 不分块的算法 DGETF2 的运算速度在大矩阵情况下，明显低于分块算法 DGETRF 和递归算法 RLUF，由此验证了矩阵分块（递归自动分块）算法的优越性。

5.2.2 Cholesky 分解递归算法

表 5.2 给出了从 300 阶到 2000 阶不同矩阵阶数下 RPOTRF、RPOTRF1 和 DPOTRF 的运算时间。其中，RPOTRF 为第 4.5.1 节中给出的 Cholesky 分解递归算法程序，RPOTRF1 表示第 4.5.2 节介绍的矩阵元素重排后的算法和程序，DPOTRF 为 LAPACK 中的 Cholesky 分解分块算法程序。表格中第 5 行数据为 RPOTRF 比 DPOTRF 速度提高的百分比。

表 5.2　RPOTRF 及 RPOTRF1 与 DPOTRF 的运算速度比较（时间单位：S）

阶数 时间 算法	300	400	500	600	700	800	900	1000	1100
RPOTRF	0.27	0.78	1.51	2.49	4.15	6.00	8.72	12.07	15.40
RPOTRF1	0.26	0.75	1.44	2.37	3.93	5.66	8.25	11.36	14.49
DPOTRF	0.21	0.56	1.14	2.07	3.72	6.65	9.80	13.83	17.82
速度提高%	-20.9	-28.1	-24.3	-16.7	-10.4	9.8	11.0	12.7	13.6

表 5.2（续）

阶数 时间 算法	1200	1300	1400	1500	1600	1700	1800	1900	2000
RPOTRF	19.25	24.33	32.28	39.16	48.30	58.98	69.79	79.12	91.63
RPOTRF1	17.99	22.68	30.34	36.46	44.82	56.47	64.41	72.87	84.21
DPOTRF	22.31	28.46	37.45	46.12	56.77	69.55	82.79	93.52	108.8
速度提高%	13.7	14.5	13.8	15.1	14.9	15.2	15.7	15.4	15.8

由表5.2，当矩阵阶数大于800后，递归算法RPOTRF比LAPACK分块算法DPOTRF速度提高10%～16%。同时，矩阵元素顺序重排后的递归算法运算速度进一步提高，RPOTRF1比RPOTRF又快了5～8%。由表5.2还可看出，由于运算量和数据量减少一半，Cholesky分解所需的运算时间不到LU分解的一半。

以上测试结果表明，递归算法由于能较好地产生矩阵分块和调用 level-3 BLAS，在大矩阵的情况下，能很好地发挥当今高性能计算机的运算效率。

5.3 小结

本章对第三、第四章研究和开发的LU分解和Cholesky分解递归算法和程序进行了运行和测试，验证算法和程序的正确性，测量程序的运算时间，与LAPACK中的同类程序进行比较，给出了测试结果，对测试手段和程序也给予了必要的描述。测试平台采用奔4PC计算机，编译环境为Microsoft Visual FORTRAN5.0。程序正确性的测试通过将运算结果与LAPACK中同类程序的结果进行比较来进行，运算时间的测试通过读取机器CPU的时间来进行，运算速度的比较通过对同一输入矩阵，运行和测试LAPACK中的同类程序来进行。运行和测试结果表明，本文研究产生的LU分解和Cholesky分解递归算法和程序比LAPACK中的同类算法和程序运算速度提高10～20%。

第六章 总结与展望

6.1 本文工作的总结

本文对用于求解大型稠密线性方程组的矩阵三角分解（包括 LU 分解和 Cholesky 分解）的递归算法进行了研究。完成的主要工作为

（1） 从理论上对递归算法进行了推导，给出了算法的描述。

（2） 用 FORTRAN90 语言有效地实现了递归算法，程序的核心运算调用高效的 Level-3 BLAS。

（3） 对研制的算法和程序在计算机上进行了运行和测试，并与通用的 LAPACK 标准算法和程序进行了比较。

（4） 研究产生的算法和程序比 LAPACK 标准算法和程序结构简单，运算速度提高 10%～20%。

本文的研究结果表明，递归是计算稠密线性方程组的有效方法，非常适合当今分层多级存储的高性能计算机。

6.2 进一步研究的设想

除了 LU 分解、Cholesky 分解以外，递归算法同样适用于 QR 分解等其它线性代数问题。事实上，BLAS 库中的基本单元（如 TRSM、SYRK 和 GEMM 等）也可以采用递归的方法实现。在递归算法基础上，对矩阵元素的排列方式进行调整，将每一级递归产生的各个分块子矩阵的元素以标准的按列存放（或按行存放）的顺序依次连续放在一起，可以进一步提高递归算法的效率。这些可以作为我们今后进一步研究的课题。

在测试设备上，本文目前限于条件使用的是 PC 计算机。今后有条件可在更专业的数值计算设备如大型计算机、共享存储并行计算机上对算法和程序进行试验和测试。还应针对不同的机型，研究和试验最佳的矩阵递归分块大小。

参 考 文 献

[1] 王顺绪，周树茎. 卷帘行存储下的一种并行 Cholesky 分解及其在 PAR95 上的实现. 南京航空航天大学学报，1999.8，vol. 31（4）

[2] 阮百尧，熊彬. 大型对称变带宽方程组的Cholesky分解法. 物探化探计算技术，2000.11, vol. 22（4）

[3] 崔俊之，梁俊. 求解多重子结构的递归算法及其软件实现. 数值计算与计算机应用，1995.3，vol. 16（1）

[4] Chi Xuebin. Parallel implementation of Linear Algebra Problems on Dawning-1000. J. of Comput. Sci. & Technol，Mar.，1998，vol.13（2）

[5] 刘小坤等. 边界元法中非对称满系数矩阵方程组的分块求解法. 数值计算与计算机应用，1995.3，vol. 16（1）

[6] 李玉成，朱鹏. BLAS 的加速方法和实现技术. 数值计算与计算机应用，1998.9，vol. 19（3）

[7] 李玉成. LAPACK 中的分块算法及其效果. 数值计算与计算机应用，2001.9，vol. 22（3）

[8] 罗晓广，李晓梅. 求解实对称带状矩阵特征值问题的一种分治算法. 数值计算与计算机应用，1998.9，vol. 19（3）

[9] 莫则尧，刘兴平，缪振民. 应用程序并行与优化关键技术研究. 数值计算与计算机应用，2002.3，vol. 23（1）

[10] 迟学斌. 应用嵌套排序的并行 Cholesky 分解算法. 数值计算与计算机应用，1995.12，vol. 16（4）

[11] 刘兴平，张景琳. 共享存储并行机的算法设计. 数值计算与计算机应用，1997.9，vol. 18（3）

[12] 张健飞，姜弘道. 对称正定矩阵的并行 LDL^T 分解算法实现. 计算机工程与设计，2003.10，vol. 24（10）

[13] 关治，陈景良. 数值计算方法. 北京：清华大学出版社，1990

[14] 封建湖等. 数值分析原理. 北京：科学出版社，2001

[15] 邓巍巍，王越男. Visual FORTRAN 编程指南. 人民邮电出版社，2000

[16] Fred G. Gustavson. Recursion leads to automatic variable blocking for dense linear algebra. IBM Journal of Research and Development,1997, vol. 41 (6)

[17] F. Gustavson, A. Henriksson, I. Jonsson, B. Kagstrom, and P. Ling, Recursive blocked data formats and BLAS's for dense linear algebra algorithms, In Proceedings of Applied Parallel Computing, PARA'98, 1998

[18] Bjarne S. Anderson, Jerzy Wasniewsky, and Fred G. Gustavson. A recursive formulation of Cholesky factorization of a matrix in packed storage. ACM Transactions on Mathematical Software, 2001.6, vol. 27 (2)

[19] E. Elmroth and F. Gustavson, Applying recursion to serial and parallel QR factorization leads to better performance, IBM Journal of Research and Development, 2000.7, vol. 44 (4)

[20] Bo Kagstrom, Per Ling, and Charles van Loan. GEMM-based level 3 BLAS: high-performance model implementations and performance evaluation benchmark. ACM Transactions on Mathematical Software, 1998.9, vol. 24 (3)

[21] F. Gustavson and I. Jonsson. Minimal-storage high-performance Cholesky factorization via blocking and recursion. IBM Journal of Research and Development, 2000.11, vol. 44 (6)

[22] B. Andersen, F. Gustavson, A. Karaivanov, J. Wasniewski, and P. Yalamov. LAWRA–linear algebra with recursive algorithms, in Proceedings of the 3rd International Conference on Parallel Processing and Applied Mathematics, Kazimierz Dolny, Poland, 1999. 9

[23] S. Toledo, Locality of Reference in LU Decomposition with partial pivoting, SIAM J. Matrix Anal. Appl., 1997.10, vol. 18 (4)

[24] Dongarra J, et al. A Set of Level 3 Basic Linear Algebra Subprograms. ACM Trans. on Math. Softw., 1990, vol. 16 (1)

[25] E . Anderson, et al. LAPACK Users' Guide, Second Edition. Philadelphia: SIAM, 1995

[26] J. Dongarra, et al. Numerical Linear Algebra for High Performance Computers, Society for Industrial & Applied Mathematics, 1998

[27] IBM. Engineering and Scientific Subroutine Library, Guide and Reference. 1994, SC23-0526-01

[28] Demmel J W. Applied Numerical Linear Algebra. Philadelphia: SIAM, 1997

[29] Metcalf S and Reid J. FORTRAN90/95 Explained. Oxford: Oxford University Press, 1996

[30] Richard L.Burden,J. Douglas Faires. Numerical Analysis (Seventh Edition). Thomson Learning Inc. 北京：高等教育出版社（影印版），2001

作者发表的相关学术论文

1、陈建平. LU 分解递归算法的研究. 计算机科学，2004.8，31(6)：141～142

2、陈建平. LU 分解递归算法的实现. 数值计算与计算机应用，2004.12，25(4)：315～320