Oracle® Developer Studio 12.5：性能库用户指南

文件号码 E71964
2016 年 6 月

ORACLE

文件号码 E71964

文档可访问性

有关 Oracle 对可访问性的承诺，请访问 Oracle Accessibility Program 网站 http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc。

获得 Oracle 支持

购买了支持服务的 Oracle 客户可通过 My Oracle Support 获得电子支持。有关信息，请访问 http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info；如果您听力受损，请访问 http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs。

目录

使用本文档

- **概述** – 介绍了如何使用 Oracle Developer Studio Fortran 95、C++ 和 C 编译器支持的 Oracle Developer Studio 性能库子例程提供的独特扩展和功能。
- **目标读者** – 应用程序开发者、系统开发者、架构师、支持工程师。
- **必备知识** – 应具备 Fortran 或 C 语言的应用知识、数字分析的基础知识，并了解 Netlib 提供的基本 LAPACK 和 BLAS 库 (http://www.netlib.org)。

产品文档库

有关该产品及相关产品的文档和资源，可从以下网址获得：http://www.oracle.com/pls/topic/lookup?ctx=E71940。

反馈

可以在 http://www.oracle.com/goto/docfeedback 上提供有关本文档的反馈。

♦ ♦ ♦ 第 1 章

Oracle Developer Studio 性能库简介

Oracle Developer Studio 性能库是一组优化的高速数学子例程，用于对线性代数和其他数字密集型问题求解。Oracle Developer Studio 性能库以来自 http://www.netlib.org 上的 Netlib 的公共域应用程序集合为基础。这些公共域应用程序经过了增强并被捆绑成 Oracle Developer Studio 性能库。

本文档介绍了对来自 Netlib 的基础应用程序进行的特定于 Oracle 的增强。可从 Netlib 和 工业和应用数学学会 (Society for Industrial and Applied Mathematics, SIAM) 获得介绍这些基础例程的参考材料。另外，Oracle Developer Studio 手册页的第 3P 部分中详细介绍了 Oracle Developer Studio 12.5 性能库的公共函数和子例程。要了解有关 3P 手册页的信息，请键入 `man -s 3p intro`。有关将 Oracle Developer Studio 手册页路径添加到 `MANPATH` 变量的更多信息，请参见《Oracle Developer Studio 12.5：安装指南》 中的 “开发者工具和手册页访问设置”。

Oracle Developer Studio 性能库中包括的库

性能库包含下列标准库的增强版本：

- LAPACK 版本 3.5.0 – 用于对线性代数问题求解。
- BLAS1（Basic Linear Algebra Subprogram，基础线性代数子程序） – 用于执行“向量-向量”运算。
- BLAS2 – 用于执行“矩阵-向量”运算。
- BLAS3 – 用于执行“矩阵-矩阵”运算。
- Netlib Sparse-BLAS – 用于执行稀疏向量运算。
- NIST Sparse-BLAS 0.5 – 执行基本的稀疏矩阵运算。
- SuperLU 3.0 – 对稀疏线性方程组求解
- 稀疏求解器 – 直接稀疏求解器例程
- FFTPACK – 执行快速傅里叶变换
- VFFTPACK – 执行向量化快速傅里叶变换
- XBLAS – 超精确的基础线性代数子程序
- 其他例程 – 转置、卷积、相关和排序

注 - 已从 Oracle Developer Studio 性能库中删除了 LINPACK。LAPACK 版本 3.5.0 取代了 LINPACK 及所有早期版本的 LAPACK。如果仍需要 LINPACK 例程，您可以从 http://www.netlib.org 获得 LINPACK 库和文档。

Oracle Developer Studio 性能库有静态库和动态库两种形式。在 Oracle Solaris 11 和 Oracle Linux 操作系统中，有用于 sparcvis、sparcvis2 和 sparcfmaf 及高级体系结构的优化 SPARC 版本。在 Oracle Solaris 11 系统及 Oracle Linux 系统中，还有用于 x86/x64 体系结构的优化版本。所有版本都支持在多处理器平台上进行并行编程。有关详细信息，请参见《Oracle Developer Studio 12.5：发行说明》。

Oracle Developer Studio 性能库 LAPACK 例程是使用 Fortran 95 编译器编译的，它与 Netlib LAPACK 版本 3.5.0 库兼容。这些例程的性能库版本能够执行与 Fortran 可调用例程相同的操作，并具有与标准 Netlib 版本相同的接口。

LAPACK 包含驱动程序例程、计算例程和辅助例程。性能库不支持辅助例程，因为可能会在不进行通知的情况下在 LAPACK 中更改或删除辅助例程。由于不支持辅助例程，因此，本用户指南和第 3P 部分的手册页中没有对其进行介绍。

许多辅助例程在例程名称中包含 LA 作为第二个和第三个字符，而某些不是。《LAPACK User's Guide, Third Edition》 (http://www.netlib.org/lapack/lug/)（《LAPACK 用户指南，第三版》）的附录 B 提供了辅助例程的列表。

关于 Netlib

Netlib 是由贝尔实验室 (AT&T Bell Laboratories)、田纳西大学 (The University of Tennessee)、橡树岭国家实验室 (Oak Ridge National Laboratory) 及全球专家共同维护的数学软件、论文和数据库的在线系统信息库。

除了在 Oracle Developer Studio 性能库中使用的库以外，Netlib 还提供了许多其他库。虽然一些库看起来类似于在性能库中使用的库，但它们可能与性能库不同并且不兼容。

使用来自其他库的例程可能会产生兼容性问题，不仅是与 Oracle Developer Studio 性能库例程不兼容，而且还会与基础 Netlib LAPACK 例程不兼容。在使用来自其他库的例程时，请参阅这些库附带的文档。

例如，Netlib 提供了 CLAPACK 库，但 CLAPACK 接口与 Oracle Developer Studio 性能库附带的 C 接口不同。Netlib 上还提供了一个 LAPACK 90 库软件包。LAPACK 90 库包含的接口与 Oracle Developer Studio 性能库 Fortran 95 接口和 Netlib LAPACK 版本 3.5.0 接口不同。如果要使用 LAPACK 90，请参阅该库附带的文档。

对于 Oracle Developer Studio 性能库支持的基本库，Netlib 提供了详细信息，可作为本用户指南的补充。《LAPACK User's Guide, Third Edition》 (http://www.netlib.org/lapack/lug/)（《LAPACK 用户指南，第三版》）介绍了 LAPACK 算法以及例程的使用方法，但没有介绍 Oracle Developer Studio 性能库对这些基础例程进行的扩展。

相关文档

《*LAPACK User's Guide*》（《LAPACK 用户指南》）是 LAPACK 版本 3.5.0 基础例程的官方参考资料。《*LAPACK Users' Guide*》（《LAPACK 用户指南》）的在线版本的网址是：http://www.netlib.org/lapack/lug/，打印版本可从工业和应用数学学会 (Society for Industrial and Applied Mathematics, SIAM) 获得，网址是：http://www.siam.org。

Oracle Developer Studio 性能库例程包含《*LAPACK Users' Guide*》（《LAPACK 用户指南》）中未介绍的性能增强、扩展和功能。但是，因为 Oracle Developer Studio 性能库保留了与 LAPACK 基础例程的兼容性，所以可将《*LAPACK Guide*》（《LAPACK 指南》）用作 LAPACK 例程和 Fortran 接口的参考资料。

注 - 已从 Oracle Developer Studio 性能库中删除了 LINPACK。仍可从 http://www.netlib.org 获得 LINPACK 库和文档。

有关构成 Oracle Developer Studio 性能库的基础的性能库例程的信息，请参见以下位置。

LAPACK 版本 3.5.0	http://www.netlib.org/lapack/
BLAS，1 至 3 级	http://www.netlib.org/blas/
FFTPACK 版本 4	http://www.netlib.org/fftpack/
VFFTPACK 版本 2.1	http://www.netlib.org/vfftpack/
稀疏 BLAS	http://www.netlib.org/sparse-blas/index.html
NIST（National Institute of Standards and Technology，美国国家标准与技术研究院）Fortran Sparse BLAS	http://math.nist.gov/spblas/
SuperLU 版本 3.0	http://crd.lbl.gov/~xiaoye/SuperLU/

Oracle Developer Studio 性能库功能

Oracle Developer Studio 性能库例程在串行平台和多处理器 (MP) 平台上都能够提高应用程序的性能，这是因为很多性能库例程的串行速度都已得到提高，而且很多例程都已进行了并行化处理。Oracle Developer Studio 性能库例程还具有在基本 Netlib 库中未提供的 SPARC、AMD 和 Intel 特定优化。

Oracle Developer Studio 性能库对基本 Netlib 库提供了以下优化和扩展：

- 支持 Fortran 95 和 C 语言接口的扩展
- Fortran 95 语言功能，包括类型独立和编译时检查
- 在性能库中的不同库中一致的 API
- 与 LAPACK 1.0、2.0、3.0、3.1.1、3.3.1、3.4.2 和 3.5.0 库的兼容性
- 增强的性能，以及某些情况下更高的准确性
- 针对特定 SPARC 和 x86/x64 指令集体系结构的优化
- 支持启用了 64 位的 Oracle Solaris 和 Linux 操作环境
- 对 SPARC 和 x86/x64 平台支持并行处理编译器选项
- 支持多处理器硬件选项

数学例程

Oracle Developer Studio 性能库例程用于对以下类型的线性代数和数字问题求解：

- 基础向量和矩阵运算 – 向量和矩阵积；平面旋转；1 范数、2 范数和无穷范数；1 阶、2 阶、k 阶和 2k 阶更新
- 线性方程组 – 对满秩方程组求解、计算误差界、对 Sylvester 方程求解、细化计算得到的解、平衡系数矩阵
- 最小二乘 – 满秩、广义线性回归、秩亏、线性等式约束
- 特征值问题 – 特征值、广义特征值、特征向量、广义特征向量、舒尔向量、广义舒尔向量
- 矩阵因式分解或矩阵分解 – SVD、广义 SVD、QL 和 LQ、QR 和 RQ、Cholesky、LU、舒尔、LDL^T、UDU^T 和 CS 分解
- 支持运算 – 条件数、场内或场外转置、逆向、行列式、惯量、超精确的迭代细化
- 稀疏矩阵 – 使用直接方法和选择的填充简化排序算法以及用户指定的排序对对称、结构对称和不对称系数矩阵求解
- 一维和二维中的卷积和相关
- 快速傅里叶变换、傅里叶分析和傅里叶合成、余弦和四分之一波长余弦变换、余弦和四分之一波长正弦变换
- 二维和三维中的复数向量 FFT 和 FFT
- 排序运算
- CBLAS 接口

与早期 LAPACK 版本的兼容性

基于 LAPACK 的 Oracle Developer Studio 性能库例程支持 LAPACK 3.5.0 中扩展的功能和改进的算法，但与 LAPACK 1.x、LAPACK 2.x、LAPACK 3.x 和 LAPACK 4.x 库完全兼容。与早期 LAPACK 版本保持兼容可以：

- 减少由于子例程名称或参数列表更改而导致的链接错误。
- 确保结果与使用早期 LAPACK 版本生成的结果一致。
- 将由于参数列表的差异而导致的程序终止情况减至最少。

Oracle Developer Studio 性能库入门

本节介绍了用于编译使用 Oracle Developer Studio 性能库例程的应用程序的最基本的编译器选项。

要使用 Oracle Developer Studio 性能库，请键入以下命令之一。

在 x86/x64 和 SPARC 平台上：

```
my_system% f95 -dalign my_file.f -library=sunperf
```

在 SPARC 平台上：

```
my_system% cc -xmemalign=8s my_file.c -library=sunperf
my_system% CC -xmemalign=8s my_file.cpp -library=sunperf
```

在 x86/64 平台上，`-xmemalign=8s` 将被忽略，因此可将其省略：

```
my_system% cc my_file.c -library=sunperf
my_system% CC my_file.cpp -library=sunperf
```

要与 Oracle Developer Studio 性能库进行静态链接，请在命令行中添加 `-staticlib=sunperf`。

因为 Oracle Developer Studio 性能库例程是使用 `-dalign` 编译的，所以，如果程序中的任何例程进行 Oracle Developer Studio 性能库调用，则在编译所有 Fortran 文件时都应当使用此选项。在 SPARC 平台上，应使用选项 `-xmemalign=8s` 编译调用 Oracle Developer Studio 性能库例程的 C 和 C++ 用户代码。如果无法使用 `-xmemalign=8s`，则启用陷阱 6 是一个允许未对齐数据的低性能解决方法。有关更多详细信息，请参见在 SPARC 平台上启用陷阱 6 [16]。

虽然 x86/x64 平台上没有数据对齐限制，但是未对齐的数据可能需要额外指令来正确处理内存传输，这进而导致了性能降低。

`-library=sunperf` 选项包括其他编译器和系统库，如 Fortran 运行时和微任务库，并为所产生的可执行文件或共享库设置运行时搜索路径。

简而言之，可使用以下选项：

- 在编译时对所有 Fortran 文件使用 `-dalign`。

  在 SPARC 平台上使用 `-xmemalign=8s`，或启用陷阱 6
- 使用相同的命令行选项进行编译和链接

- `-library=sunperf` 或 `-library=sunperf -staticlib=sunperf`

有关可优化应用程序性能的其他选项，请参见“关于编译” [26]和第 4 章 并行处理。

▼ 在 SPARC 平台上启用陷阱 6

在 SPARC 平台上，如果数据未对齐导致失败，并且无法使用 `-dalign` 或 `-xmemalign=8s` 编译应用程序，则可以启用陷阱 6 为未对齐的数据提供处理程序。要在 SPARC 平台上启用陷阱 6，请执行以下操作：

1. **将此汇编代码放在名为 `trap6_handler.s` 的文件中。**

```
 .global trap6_handler_
 .text
 .align 4
trap6_handler_:
 retl
 ta    6
```

2. **对 `trap6_handler.s` 进行汇编。**

```
my_system% fbe trap6_handler.s
```

`fbe` 是将从汇编语言源文件创建对象文件的命令。

从 Oracle Developer Studio 性能库调用的第一个可并行子例程将调用名为 `trap6_handler_` 的例程。如果未指定 `trap6_handler_`，Oracle Developer Studio 性能库将调用不执行任何操作的默认处理程序。不提供处理程序来处理任何未对齐数据将会导致致命陷阱。

3. **在命令行中包括 `trap6_handler.o`。**

```
my_system% f95 any.f trap6_handler.o -library=sunperf
```

♦ ♦ ♦ 第 2 章

2 使用 Oracle Developer Studio 性能库

本章介绍了使用 Oracle Developer Studio 性能库提高用 Fortran 95 或 C 编写的应用程序的执行速度。使用 Oracle Developer Studio 性能库可以提高许多应用程序的性能，无需进行源代码更改或重新编译。然而，可能需要对应用程序进行一些修改才能借助 Oracle Developer Studio 性能库获得最佳性能。

提高应用程序性能

以下各节介绍了在不修改源代码或进行重新编译的情况下使用 Oracle Developer Studio 性能库例程的方法。

使用 Oracle Developer Studio 性能库例程来替换例程

许多应用程序都使用一个或多个基本 Netlib 库，例如 LAPACK 或 BLAS。因为 Oracle Developer Studio 性能库保留了这些库的相同接口和功能，所以可以使用 Oracle Developer Studio 性能库例程替换基本 Netlib 例程。由于 Oracle Developer Studio 性能库例程比相应的 Netlib 例程或其他供应商提供的类似例程快，因此，应用程序的性能得以提高。

提高其他库的性能

许多商业数学库是围绕 BLAS 和 LAPACK 通用例程的核心构建的。如果某个应用程序依赖于另一个库中的专用接口，而这些接口会阻止完全替换库，则可以使用 Oracle Developer Studio 性能库 BLAS 和 LAPACK 例程替换在该库中使用的 BLAS 和 LAPACK 例程。由于替换核心例程不需要进行任何代码更改，因此专用库功能仍然可用，库中的其他例程仍将保持不变。

使用工具重建代码

自动代码重建工具可将现有代码替换为 Oracle Developer Studio 性能库代码，从而可以修改未直接使用 Oracle Developer Studio 性能库例程的一些库。例如，某个源代码到源代码转换工具可以将现有的 BLAS 代码结构替换为对 Oracle Developer Studio 性能库 BLAS 例程的调用。这些转换工具还可以识别用户编写的许多矩阵乘法，并将其替换为对 Oracle Developer Studio 性能库中的矩阵乘法子例程的调用。

Fortran 接口

Oracle Developer Studio 性能库包含 `f95` 接口和旧的 `f77` 接口，以便保持与标准的 LAPACK 和 BLAS 库和现有代码的兼容性。Oracle Developer Studio 性能库 `f95` 和旧的 `f77` 接口使用以下约定：

- 所有参数都是通过引用传递的。
- 在同一调用中，参数类型必须一致。例如，不要在同一个调用中混合使用 `REAL*8` 和 `REAL*4` 参数。
- 数组是逐列存储的。
- 索引从 1 开始，与标准 Fortran 惯例一致。

在调用 Oracle Developer Studio 性能库例程时，应注意以下信息：

- 不要使用 Fortran 95 `INTERFACE` 语句构建子例程。应使用 `USE SUNPERF` 语句。
- 不要使用 `-ext_names=plain` 编译从 Oracle Developer Studio 性能库调用例程的例程。

Fortran `SUNPERF` 模块与 Fortran 95 结合使用

Oracle Developer Studio 性能库提供了一个 Fortran 模块，以便于使用 Fortran 95 程序的功能。要使用此模块，可在 Fortran 95 代码中包括以下行。

```
USE SUNPERF
```

`USE` 语句必须在代码中的所有其他语句的前面，但 `PROGRAM` 或 `SUBROUTINE` 语句除外。

`SUNPERF` 模块包含的接口简化了调用序列，并提供以下功能：

- 类型独立 – Oracle Developer Studio 性能库支持将自动识别数据参数类型的接口，因此不需要使用取决于类型的前缀（`S`、`D` `C` 或 `Z`）。在 FORTRAN 77 例程中，必须将类型指定为例程名称的一部分。例如，`DGEMM` 是一个双精度矩阵乘法，`SGEMM` 是一

个单精度矩阵乘法。在使用 Fortran 95 接口调用 GEMM 时，Fortran 将从传递的参数中推导出类型。将单精度参数传递到 GEMM 所获得的结果相当于指定了 SGEMM，传递双精度参数所获得的结果相当于指定了 DGEMM。例如，CALL DSCAL(20,5.26D0,X,1) 可更改为 CALL SCAL(20, 5.26D0, X, 1)。

- 编译时检查 – 在 FORTRAN 77 中，编译器通常无法确定应将什么参数传递到某个特定例程。在 Fortran 95 中，USE SUNPERF 语句使得编译器可以确定传递到每个 Oracle Developer Studio 性能库例程的每个参数的数量、类型、大小和形状。它可在编译过程中针对预期的值对调用进行检查，并显示错误。
- *64* 位整数支持 – 在使用 Oracle Developer Studio 性能库附带的 64 位接口时，必须将整数参数提升至 64 位，并且必须通过在例程名称中附加 _64 来修改例程名称。使用 SUNPERF 模块时，64 位整数将被自动识别，这就不需要在例程名称中附加 _64 了，如以下代码示例所示：

```
SUBROUTINE SUB(N,ALPHA,X,Y)
USE SUNPERF
INTEGER(8) N
REAL(8) ALPHA, X(N), Y(N)

! EQUIVALENT TO DAXPY_64(N,ALPHA,X,1_8,Y,1_8)
CALL DAXPY(N,ALPHA,X,1_8,Y,1_8)

END
```

有关使用 Oracle Developer Studio 性能库 64 位接口的详细说明，请参见“针对启用了 64 位的操作环境编译代码” [26]。

由于 sunperf.mod 文件是用 -dalign 编译的，因此必须使用 -dalign 编译包含 USE SUNPERF 语句的任何代码。如果未使用 -dalign 编译代码，则会发生以下错误。

```
use sunperf
            ^
   "test_code.f", Line = 2, Column = 11: ERROR: Procedure "SUNPERF" and this compilation must
both be compiled with -dalign, or without -dalign.
```

Fortran 示例

要提高单处理器应用程序的性能，请找出应用程序中可由 Oracle Developer Studio 性能库例程调用替换的代码结构。通过找出可以并行化的代码并将其并行化，可以提高多处理器应用程序的性能。

要通过修改代码以使用 Oracle Developer Studio 性能库例程来提高应用程序性能，请找出其功能与某个 Oracle Developer Studio 性能库例程的功能完全相同的代码块。以下代码示例是“矩阵-向量”积 y ← Ax + y，可使用 DGEMV 子例程来替换它。

```
      DO I = 1, N
```

```
        DO J = 1, N
            Y(I) = Y(I) + A(I,J) * X(J)
        END DO
    END DO
```

另外一些情况下，某个代码块可能相当于多个 Oracle Developer Studio 性能库调用，或包含可由 Oracle Developer Studio 性能库例程调用替换的部分代码。请考虑以下代码示例。

```
    DO I = 1, N
        IF (V2(I,K) .LT. 0.0) THEN
            V2(I,K) = 0.0
        ELSE
            DO J = 1, M
                X(J,I) = X(J,I) + Vl(J,K) * V2(I,K)
            END DO
        END IF
    END DO
```

代码示例可以重新编写，以使用 Oracle Developer Studio 性能库例程 DGER，如下所示。

```
    DO I = 1, N
        IF (V2(I,K) .LT. 0.0) THEN
           V2(I,K) = 0.0
        END IF
    END DO
    CALL DGER (M, N, 1.0D0, X, LDX, Vl(l,K), 1, V2(1,K), 1)
```

也可以使用 Fortran 95 特定语句重新编写同一代码示例，如下所示。

```
WHERE (V(1:N,K) .LT. 0.0) THEN
       V(1:N,K) = 0.0
END WHERE
CALL DGER (M, N, 1.0D0, X, LDX, Vl(l,K), 1, V2(1,K), 1)
```

由于要使用零替换 V2 中的负数的代码在 Oracle Developer Studio 性能库中没有自然的类似代码，因此该代码不再放在外层循环中。该代码移到自身的循环中后，循环的其余部分是一般矩阵 X 的 1 阶更新，可使用 BLAS 中的 DGER 例程进行替换。

性能的提高程度还取决于 Oracle Developer Studio 性能库例程使用的数据。例如，如果 V2 包含许多负值或零值，大部分时间可能不会花在 1 阶更新上。在这种情况下，将代码替换为对 DGER 的调用可能不会提高性能。

对其他循环索引求值会影响所使用的 Oracle Developer Studio 性能库例程。例如，如果对 K 的引用是一个循环索引，则上面所示的代码示例中的循环可能是一个较大代码结构的一部分，其中在 DGEMV 或 DGER 之间的循环可以转换为某种形式的矩阵乘法。如果是这样，对矩阵乘法例程的单次调用比通过调用 DGER 来使用循环更能提高性能。

由于所有 Oracle Developer Studio 性能库例程都是 MT 安全的（multithread safe，多线程安全），因此，使用自动并行化编译器将包含 Oracle Developer Studio 性能库例程调用的循环并行化可提高在多处理器平台上的性能。

以下代码示例显示了将 Oracle Developer Studio 性能库例程与自动并行化编译器并行化指令结合使用的示例。

```
      C$PAR DOALL
      DO I = 1, N
             CALL DGBMV ('No transpose', N, N, ALPHA, A, LDA,
     $     B(l,I), 1, BETA, C(l,I), 1)
      END DO
```

Oracle Developer Studio 性能库包含一个名为 DGBMV 的例程来将带状矩阵与向量相乘。通过将此例程放到正确构建的循环中，可使用 Oracle Developer Studio 性能库例程来将带状矩阵与矩阵相乘。默认情况下，编译器不会将此循环并行化，因为循环中存在的子例程调用会阻止并行化。不过，Oracle Developer Studio 性能库例程是 MT 安全的，因此，可以使用并行化指令来指示编译器将此循环并行化。

还可以使用编译器指令通过子例程调用来将通常不可并行的循环并行化。例如，通常无法将其中调用了某些线性方程组求解器的循环并行化，因为某些供应商在实施这些例程时使用的是非 MT 安全的代码。其中调用了线性方程组求解器的专业驱动程序（名称以 SVX 或 SVXX 结尾的例程）的循环通常不能通过其他 LAPACK 实施进行并行化。因为 Oracle Developer Studio 性能库中的 LAPACK 实施支持将包含此类调用的循环并行化，所以，多处理器平台的用户可以通过将这些循环并行化来提高性能。

C 接口

可从 FORTRAN 77、Fortran 95 或 C 程序内调用 Oracle Developer Studio 性能库例程。但是，C 程序必须仍然使用 FORTRAN 77 调用序列。

Oracle Developer Studio 性能库针对 LAPACK、BLAS、FFTPACK、VFFTPACK、SPARSE BLAS 和 SPSOLVE 中包含的每个例程都提供了原生 C 接口。Oracle Developer Studio 性能库 C 接口有以下特性：

- 函数名称有 C 名称
- 函数接口遵循 C 约定
- C 接口不包含 C 函数的冗余参数或不需要的参数

以下示例将 DGBCON 例程的标准 LAPACK Fortran 接口与 Oracle Developer Studio 性能库 C 接口进行了比较。

```
CALL DGBCON (NORM, N, NSUB, NSUPER, DA, LDA, IPIVOT, DANORM,
 DRCOND, DWORK, IWORK2, INFO)
void dgbcon(char norm, int n, int nsub, int nsuper, double *da,
 int lda, int *ipivot, double danorm, double drcond,
 int *info)
```

请注意，参数名称是相同的，并且同名参数具有相同的基本类型。在 C 版本中，仅用作输入值的标量参数（例如 NORM 和 N）是通过值传递的。将用来返回值的数组和标量是通过引用传递的。

Oracle Developer Studio 性能库 C 接口针对 CLAPACK 进行了改进，可从 Netlib 获取该接口，它是标准库的 `f2c` 转换。例如，所有 CLAPACK 例程名称后面都有一个结尾下划线，这是为了保持与 Fortran 编译器的兼容性，该编译器通常在对象 (`.o`) 文件中的例程名称后加一个下划线后缀。Oracle Developer Studio 性能库 C 接口不需要结尾下划线。

Oracle Developer Studio 性能库 C 接口使用以下约定：

- 纯输入标量是通过值而非通过引用传递的。复数和双精度复数参数不被视为标量，因为 C 不将它们实施为标量。
- 复数标量可作为长度 2 的结构或数组传递。
- 即使在 C 执行了类型转换后，参数类型也必须匹配。例如，在传递单精度实数值时要注意，因为 C 编译器可能会自动将该参数提升为双精度。
- 数组是逐列存储的。对于 Fortran 程序员而言，这是数组的自然存储顺序。对于 C 程序员而言，这是他们通常使用的顺序的转置。在文档和手册页中，对行的引用是指列，对列的引用是指行。
- 数组索引从 1 开始，这与 Fortran 约定一致，与 C 中的从 0 开始不同。

  例如，`IDAMAX` 的 Fortran 接口对于 C 程序而言是 `idamax_`，它返回 1 来表示向量中的第一个元素。`idamax` 的 C 接口对于 C 程序而言是 `idamax`，它返回 1 来表示向量的第一个元素。在函数返回值、排列向量以及使用向量或数组索引的其他任何地方都遵循此约定。

注 - 某些 Oracle Developer Studio 性能库例程在内部使用 `malloc`，因此对 Oracle Developer Studio 性能库和 `sbrk` 进行调用的用户代码可能无法正常工作。

SPARC 版的 Oracle Developer Studio 性能库使用 32 位模式的全局整数寄存器 `%g2`、`%g3` 和 `%g4` 以及 64 位模式的 `%g2` 至 `%g5` 作为临时寄存器。用户代码不应当使用这些寄存器进行临时存储，然后调用 Oracle Developer Studio 性能库例程。当 Oracle Developer Studio 性能库例程使用这些寄存器时，数据会被覆盖。

C 示例

将用户编写的代码序列转换为对 Oracle Developer Studio 性能库例程的调用可以提高应用程序性能。在 LAPACK 的基础上改写的以下代码示例显示了一个例子。

```
int    i;
float a[n], b[n], largest;

largest = a[0];
for (i = 0; i < n; i++)
{
if (a[i] > largest)
    largest = a[i];
```

```
    if (b[i] > largest
    largest = b[i];
}
```

没有 Oracle Developer Studio 性能库例程可以完全代替此代码示例的功能。不过，可通过多次调用 Oracle Developer Studio 性能库例程 `isamax` 来替换该代码，从而加快执行速度，如以下代码示例所示。

```
int    i, large_index;
float a[n], b[n], largest;

large_index = isamax (n, a, l) - 1;
largest = a[large_index];
large_index = isamax (n, b, l) - 1;
if (b[large_index] > largest)
     largest = b[large_index];
```

比较调用 Oracle Developer Studio 性能库中的原生 C 例程 `isamax`（如上一个代码示例所示）与调用 CLAPACK 中的 `isamax` 例程（如以下代码示例所示）之间的差异。

```
/* 1. Declare scratch variable to allow 1 to be passed by reference */
int one = l;
/* 2. Append underscore to conform to FORTRAN naming system     */
/* 3. Pass all arguments, even scalar input-only, by reference  */
/* 4. Subtract one to convert from FORTRAN indexing conventions */
large_index = isamax_ (&n, a, &one) - l;
largest = a[large_index]; large_index = isamax_ (&n, b, &one) - l;
if (b[large_index] > largest)
     largest = b[large_index];
```

♦ ♦ ♦ 第 3 章

3 优化应用程序

本章介绍了如何使用编译器和链接选项针对以下体系结构或操作环境优化应用程序：

- 特定指令集体系结构
- 启用了 32 位和 64 位的操作环境

32 位和 64 位环境的比较

下表显示了 32 位和 64 位操作环境的比较。以下各部分中详细介绍了这些项目。

表 1 32 位和 64 位操作环境的比较

	32 位 (ILP 32)	64 位 (LP64)
`-xarch`（在 SPARC 平台上）	`sparcvis`、`sparcvis2`、`sparcfmaf`	`sparcvis`、`sparcvis2`、`sparcfmaf`
`-xarch`（在 x86 平台上）	`generic`、`sse2`	`sse2`
寻址	`-m32`	`-m64`
Fortran 整数	`INTEGER`、`INTEGER*4`	`INTEGER*8`
C 整数	`int`	`long`
浮点数	`S/D/C/Z`	`S/D/C/Z`
API	例程名称	例程名称，带 `_64` 后缀

使用 Oracle Developer Studio 性能库

Oracle Developer Studio 性能库是使用此发行版附带的 `f95` 编译器编译的。Oracle Developer Studio 性能库例程是使用 `-dalign`、`-xparallel` 编译的。

链接 Fortran 程序

链接程序时，可使用 `-dalign -library=sunperf` 以及在编译时所使用的相同命令行选项。

Oracle Developer Studio 性能库通过 `-library` 开关而非其他库中用来进行链接的 `-l` 开关链接到应用程序，如下所示。

```
my_system% f95 -dalign my_file.f -library=sunperf
```

链接 C 和 C++ 程序

链接程序时，可使用 `-library=sunperf` 以及在编译时所使用的相同命令行选项。如果在 SPARC 系统上编译，则包括选项 `-xmemalign=8s`，如下所示。在 x86 和 x64 平台上将忽略 `-xmemalign=8s` 选项。

```
my_system% cc -xmemalign=8s my_file.c -library=sunperf
my_system% CC -xmemalign=8s my_file.cpp -library=sunperf
```

如果无法使用 `-dalign` 或 `-xmemalign=8s` 进行编译，请按在 SPARC 平台上启用陷阱 6 [16]中的说明提供陷阱 6 处理程序。

关于编译

使用最合适的 `-xarch` 选项进行编译可获得最佳性能。在链接时，使用编译时所用的同一 `-xarch` 选项，以选择针对特定指令集体系结构进行了优化的 Oracle Developer Studio 性能库版本。

注 - 使用特定于指令集的优化选项可提高所选指令集体系结构上的应用程序性能，但限制了代码可移植性。

有关不同 `-xarch` 选项的详细说明，请参阅《Oracle Developer Studio 12.5：Fortran 用户指南》或《Oracle Developer Studio 12.5：C 用户指南》。

`fbe`、`cc`、`CC` 和 `f95` 的手册页中也列出了用于 SPARC 和 x86 指令集体系结构的 `-xarch` 的值。

针对启用了 64 位的操作环境编译代码

要针对启用了 64 位的操作环境编译代码，可使用 `-m64`，并将所有整数参数转换为 64 位参数。64 位例程需要使用 64 位整数。

Oracle Developer Studio 性能库提供了 32 位和 64 位接口。要使用 64 位接口，请执行以下操作：

- **修改 Oracle Developer Studio 性能库例程名称。**对于 C 和 Fortran 95 代码，将 `_64` 附加到 Oracle Developer Studio 性能库例程的名称（例如，`rfftf_64` 或

CFFTB_64）。对于具有 USE SUNPERF 语句的 Fortran 95 代码，特定接口并不严格需要 _64 后缀，如 DGEMM。对于通用接口，仍需要 _64 后缀，如 GEMM。

- 将整数提升至 64 位。双精度变量和双精度复数变量的实部和虚部已经是 64 位了。只将整数提升至 64 位。

64 位整数参数

只有使用 -m64 进行链接时，才能使用这些附加的 64 位整数接口。针对 32 位操作环境编译的代码 (-m32) 无法调用 64 位整数接口。

要直接调用 64 位整数接口，请将后缀 _64 附加到标准库名称。例如，使用 daxpy_64() 而不是 daxpy()。

但是，如果间接调用 64 位整数接口，则不要将 _64 附加到 Oracle Developer Studio 性能库例程的名称。调用性能库例程将访问一个 32 位包装程序，该包装程序将 32 位整数提升至 64 位整数，然后调用 64 位例程，之后再将 64 位整数降至 32 位整数。

为获得最佳性能，可通过将 _64 附加到例程名称来直接调用该例程。

对于 C 程序，请使用 long 而不是 int 参数。以下代码示例显示了直接调用 64 位整数接口。

```
#include <sunperf.h>
long n, incx, incy;
double alpha, *x, *y;
daxpy_64(n, alpha, x, incx, y, incy);
```

以下代码示例显示了间接调用 64 位整数接口。

```
#include <sunperf.h>
int  n, incx, incy;
double alpha, *x, *y;
daxpy   (n, alpha, x, incx, y, incy);
```

对于 Fortran 程序，请为所有整数参数使用 64 位整数。可使用以下方法将整数参数转换为 64 位：

- 要将已声明但没有明确的字节大小和整型文字常数的所有整数从 32 位提升至 64 位，请使用 -xtypemap=integer:64 进行编译。
- 要提升特定的整数声明，请将 INTEGER 或 INTEGER*4 更改为 INTEGER*8。
- 要提升整型文字常数，可将 _8 附加到该常数。

请考虑以下代码示例。

```
INTEGER*8 N
REAL*8 ALPHA, X(N), Y(N)

! _64 SUFFIX: N AND 1_8 ARE 64-BIT INTEGERS
```

```
CALL DAXPY_64(N,ALPHA,X,1_8,Y,1_8)
```

INTEGER*8 参数不能在 32 位环境中使用。不能使用 64 位参数调用 32 位库中的例程 v8plusa、v8plusb。但是，可使用 32 位参数调用 64 位例程。

在传递 Fortran 95 代码中未使用 -xtypemap 编译的常数时，可将 _8 附加到文字常数以影响提升。例如，在使用 Fortran 95 时，将 CALL DSCAL(20,5.26D0,X,1) 更改为 CALL DSCAL(20_8,5.26D0,X,1_8)。此示例假定 USE SUNPERF 包括在代码中，因为未将 _64 附加到例程名称。

以下代码示例显示了使用 32 位参数从 Fortran 95 调用 CAXPY。

```
      PROGRAM TEST
      COMPLEX ALPHA
      INTEGER,PARAMETER :: INCX=1, INCY=1, N=10
      COMPLEX X(N), Y(N)

      CALL CAXPY(N, ALPHA, X, INCX, Y, INCY)
```

以下代码示例显示了使用 64 位参数从 Fortran 95 调用 CAXPY（没有 USE SUNPERF 语句）。

```
      PROGRAM TEST
      COMPLEX   ALPHA
      INTEGER*8, PARAMETER :: INCX=1, INCY=1, N=10
      COMPLEX   X(N), Y(N)

      CALL CAXPY_64(N, ALPHA, X, INCX, Y, INCY)
```

在使用 64 位参数时，如果未使用 USE SUNPERF 语句，则必须将 _64 附加到例程名称。

以下 Fortran 95 代码示例显示了使用 64 位参数调用 CAXPY。

```
      PROGRAM TEST
      USE SUNPERF
      .
      .
      .
      COMPLEX   ALPHA
      INTEGER*8, PARAMETER :: INCX=1, INCY=1, N=10
      COMPLEX   X(N), Y(N)

      CALL CAXPY(N, ALPHA, X, INCX, Y, INCY)
```

在 C 例程中，在使用 -m32 编译时，long 的大小是 32 位；在使用 -m64 编译时，其大小是 64 位。以下代码示例显示了使用 32 位参数调用 dgbcon。

```
void dgbcon(char norm, int n, int nsub, int nsuper, double *da,
            int lda, int *ipivot, double danorm, double drcond,
            int *info)
```

以下代码示例显示了使用 64 位参数调用 dgbcon。

```
void dgbcon_64 (char norm, long n, long nsub, long nsuper,
                  double *da, long lda, long *ipivot, double danorm,
                double *drcond, long *info)
```

♦ ♦ ♦ 第 4 章

并行处理

本章介绍了在多处理器环境中将 Oracle Developer Studio 性能库与共享内存并行操作一起使用。

运行时问题

在运行时，如果与编译器并行操作一起运行，Oracle Developer Studio 性能库将与编译器使用相同的线程池。在所有平台上，必须将每个线程堆栈大小设置为至少 8 兆字节。可使用 `STACKSIZE` 或 `OMP_STACKSIZE` 环境变量进行设置（以千字节为单位）。不能同时使用这两个变量。如果同时使用这两个环境变量并且两个值不同，程序将会停止并显示错误消息。

要将每个线程堆栈大小设置为 8 兆字节，请执行以下操作：

```
my_host% setenv STACKSIZE 8192
```

对于运行 POSIX 或 Oracle Solaris 线程的程序而言，不需要设置 `STACKSIZE` 环境变量。在这种情况下，调用性能库例程的由用户创建的线程必须具有至少为 8 兆字节的堆栈大小。没有为性能库例程提供适当的堆栈大小会导致堆栈溢出问题。堆栈溢出问题的症状包括可能会出现很难诊断的运行时故障。有关为用户创建的线程设置堆栈大小的更多信息，请参见 POSIX 线程的 `pthread_create`(3C)、`pthread_attr_init`(3C) 和 `pthread_attr_setstacksize`(3C) 手册页，或 Oracle Solaris 线程的 `thr_create`(3C) 手册页。

提示 - 如果您在诊断核心转储时遇到问题，可尝试将堆栈大小增至最低要求以上。

并行度

已使用 *OpenMP Fortran* 应用程序接口中的编译器指令、库例程和环境变量将 Oracle Developer Studio 性能库中的一些选定例程并行化。这些例程并行使用的线程数量由环境变量 `OMP_NUM_THREADS` 进行控制，您可以在运行时设置这些变量。也可以设置环境变量 `PARALLEL`，但如果两者都设置，它们的值必须相同，否则在执行时会发生致命错误。

在用户代码中调用 Oracle Developer Studio 性能库例程 USE_THREADS 或 OpenMP 例程 OMP_SET_NUM_THREADS 可以覆盖这两个环境变量。

通过执行以下操作，可以将用户代码并行化：

- 将环境变量 OMP_NUM_THREADS 设置为大于 1 的值
- 使用编译器并行指令，如 OpenMP API 中的指令

  使用适当的编译器标志：-xopenmp=parallel 或 -xautopar

如果满足以下条件，Oracle Developer Studio 性能库例程将并行执行：

- OMP_NUM_THREADS 已设置为大于 1 的值
- 例程不是从并行区域调用的

Oracle Developer Studio 性能库在其并行化处理中使用 OpenMP 指令，并且不支持嵌套并行操作。如果按照上面的说明将用户代码并行化，并调用 Oracle Developer Studio 性能库例程，如果该例程检测到它是从并行区域调用的，则该例程将以串行方式执行。否则，例程将以并行方式执行。

还可以将 POSIX 或 Oracle Solaris 线程创建为在用户代码的选定区域中并行执行。如果在这种并行模式下调用性能库例程，该例程将无法检测到它是从并行区域调用的。因此，必须将环境变量 OMP_NUM_THREADS 设置为 1 或取消设置它，或者在用户代码的适当位置调用 USE_THREADS(3P)。否则，会使嵌套并行操作产生无法预料的结果。

例如，如果使用 -xopenmp=parallel 链接包含以下代码段的程序，并将 OMP_NUM_THREADS 设置为 4，那么循环将以并行方式执行，并有四个并行运行的 DGEMM 实例。但是，每个 DGEMM 实例将以串行方式运行，因为仅支持一层并行化处理。

```
!$OMP PARALLEL
    DO I = 1, N
        CALL DGEMM(...)
    END DO
!$OMP END PARALLEL
```

在下面的代码示例中，如果程序未使用 -xautopar 进行链接，则循环将不会并行化，但每个 DGEMM 实例将由四个线程执行。

```
    DO I = 1, N
        CALL DGEMM(...)
    END DO
```

如果使用 -xopenmp=parallel 链接包含以下代码段的程序，并将 OMP_NUM_THREADS 设置为大于 1 的值，则所示区域将由单个线程执行。但是，每个 DGEMM 调用将由 OMP_NUM_THREADS 个线程执行。

```
!$OMP SINGLE
    DO I = 1, N
        CALL DGEMM(...)
    END DO
```

```
!$OMP END SINGLE
```

在下面的代码示例中，最多是双向并行操作，不管可执行的 OpenMP 线程的数量是多少。只存在一层并行操作，这是两个部分。DGEMM 调用中的其他并行操作将被抑制。

```
!$OMP PARALLEL SECTIONS
!$OMP SECTION
    DO I = 1, N / 2
        CALL DGEMM(...)
    END DO
!$OMP SECTION
    DO I = N / 2 + 1, N
        CALL DGEMM(...)
    END DO
!$OMP END PARALLEL SECTIONS
```

同步机制

POSIX/Oracle Solaris 线程模型的一个特性是，正在运行的应用程序的绑定线程在空闲时会让出 CPU，这样就能在共享（超额订阅）环境中获得较高的吞吐量和资源利用率。默认情况下，已由编译器并行化的代码中的绑定线程在空闲时会旋转等待，如果系统中有其他应用程序在争用 CPU 资源，这就会导致吞吐量不佳。在这种情况下，可使用环境变量 SUNW_MP_THR_IDLE 来控制线程在完成其并行作业份额后的行为：

```
my_host% setenv SUNW_MP_THR_IDLE value
```

在这里，*value* 可以是 spin 或 sleep *n* s 或 sleep *n* ms，spin 是默认值。

在旋转等待 *n* 个单位后，sleep 将线程置于休眠状态。等待单位可以是秒（s，默认单位），也可以是毫秒 (ms)。不带参数的 sleep 将在并行任务完成后立即将线程置于休眠状态。如果 SUNW_MP_THR_IDLE 包含非法值或未设置，则将 spin 用作默认值。

以下设置将分别使线程旋转等待（默认行为）、休眠之前旋转 2 秒，或休眠之前旋转 100 毫秒。使用 Oracle Developer Studio 性能库例程不会改变代码的旋转等待行为。

```
% setenv SUNW_MP_THR_IDLE spin
% setenv SUNW_MP_THR_IDLE 2s
% setenv SUNW_MP_THR_IDLE 100ms
```

并行处理示例

本节演示了如何将 OMP_NUM_THREADS 环境变量与编译和链接选项结合使用来创建以串行和并行方式执行的代码。

要创建串行应用程序，请执行以下操作：

- 调用一个或多个 Oracle Developer Studio 性能库例程
- 使用 `-library=sunperf`，进行链接，将标志放在命令行末尾。不要使用 `-xopenmp=parallel` 或 `-xautopar` 进行编译或链接
- 取消设置 `OMP_NUM_THREADS` 环境变量或将其设置为 1

以下示例说明了如何使用共享的 Oracle Developer Studio 性能库 `libsunperf.so` 进行编译和链接。

```
my_host% cc -xmemalign=8s -xarch=native any.c -library=sunperf

my_host% f95 -dalign -xarch=native any.f95 -library=sunperf
```

要创建在多处理器上执行的并行应用程序，请执行以下操作：

- 调用一个或多个 Oracle Developer Studio 性能库例程
- 在编译和链接命令中使用相同的并行化选项（`-xopenmp=parallel` 或 `-xautopar`）
- 使用 `-library=sunperf`，进行链接，将标志放在命令行末尾。
- 在运行可执行文件之前，将 `OMP_NUM_THREADS` 设置为可用处理器的数量

例如，要使用 24 个处理器，请键入以下命令：

```
my_host% f95 -dalign -xarch=native my_app.f -library=sunperf
my_host% setenv OMP_NUM_THREADS 24
my_host% ./a.out
```

以上示例使 Oracle Developer Studio 性能库例程以并行方式运行，但用户代码 `my_app.f` 的任何部分都不会以并行方式运行。要使编译器尝试将 `my_app.f` 并行化，需要在编译行中使用 `-xopenmp=parallel` 或 `-xautopar`：

```
my_host% f95 -dalign -xopenmp=parallel my_app.f -library=sunperf
my_host% setenv OMP_NUM_THREADS 24
my_host% ./a.out
```

♦ ♦ ♦ 第 5 章

5 使用矩阵

大部分矩阵可以采用能够节省存储空间和计算时间的方式进行存储。Oracle Developer Studio 性能库使用以下存储方案：

- 带状存储
- 填充存储

Oracle Developer Studio 性能库可处理以下四种形式之一的矩阵：

- 一般
- 三角
- 对称
- 三对角

以下各节对这些存储方案和矩阵类型进行了说明。

矩阵存储方案

对已存储的数组进行处理的某些 Oracle Developer Studio 性能库例程通常具有可利用这些特殊存储形式的相应例程。例如，`DGBMV` 将构成带状存储中的一般矩阵和向量的积，`DTPMV` 将构成填充存储中的三角矩阵和向量的积。

带状存储

如果存储带状矩阵，该矩阵的第 *j* 列将与 Fortran 数组的第 *j* 列相对应。

以下代码将一般数组中的带状一般矩阵复制到带状存储模式中。

```
C     Copy the matrix A from the array AG to the array AB. The
C     matrix is stored in general storage mode in AG and it will
C     be stored in banded storage mode in AB. The code to copy
C     from general to banded storage mode is taken from the
C     comment block in the original DGBFA by Cleve Moler.
```

```
C
      NSUB = 1
      NSUPER = 2
      NDIAG = NSUB + 1 + NSUPER
      DO ICOL = 1, N
        I1 = MAX0 (1, ICOL - NSUPER)
        I2 = MIN0 (N, ICOL + NSUB)
        DO IROW = I1, I2
          IROWB = IROW - ICOL + NDIAG
          AB(IROWB,ICOL) = AG(IROW,ICOL)
        END DO
      END DO
```

这种用来存储带状矩阵的方法与 LAPACK 和 BLAS 使用的存储方法兼容。

填充存储

填充向量是三角、对称或厄尔米特矩阵的另一种表示形式。通过将元素按列的顺序存储到向量中，即可将数组填充到向量中。用于对角元素的空间总是处于保留状态，即使对角元素的值已知，例如在单位对角矩阵中。

上三角存储在数组 `A` 的一般存储中的上三角矩阵或对称矩阵可被传送至数组 `AP` 的填充存储中，如下所示。此代码来自 LAPACK 例程 `DTPTRI` 的注释块。

```
JC = 1
DO J = 1, N
   DO I = 1, J
      AP(JC+I-1) = A(I,J)
   END DO
   JC = JC + J
 END DO
```

同样，下三角存储在数组 `A` 的一般存储中的下三角矩阵或对称矩阵可被传送至数组 `AP` 的填充存储中，如下所示：

```
JC = 1
DO J = 1, N
   DO I = J, N
      AP(JC+I-1) = A(I,J)
   END DO
   JC = JC + N - J + 1
END DO
```

矩形全填充格式

矩形全填充 (Rectangular Full Packed, RFP) 矩阵是一种用于存储三角矩阵和对称矩阵的数据格式。它使用 3 级 BLAS 将标准填充格式数组结合起来，充分利用了存储和高性能。有关信息，请参见《Rectangular Full Packed Format for LAPACK Algorithms

Timings on Several Computers》 (http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.456.7563&rep=rep1&type=pdf)（《多台计算机上的 LAPACK 算法计时的矩形全填充格式》）和手册页的“更多详细信息”部分，以了解使用矩形全填充格式的例程。

矩阵类型

一般矩阵是最常用的类型，Oracle Developer Studio 性能库中的大部分运算都是在一般矩阵上进行的。在许多情况下，有一些例程使用其他类型的矩阵。例如，DGEMM 可计算两个一般矩阵的积，DTRMM 可计算一个三角矩阵和一个一般矩阵的积。

一般矩阵

一般矩阵的存储使矩阵元素与数组元素之间存在一一对应的关系。矩阵 A 的元素 A*ij* 存储在相应数组 A 的元素 A(*I*,*J*) 中。一般矩阵没有特殊的存储方案，因为其每个元素都是以显式方式存储的。与此相反，对于一般带状矩阵，只会存储其非零上对角、对角和下对角元素。以下示例说明了一般带状矩阵如何存储在二维数组中。不会访问带 x 标记的数组位置。

一般带状矩阵	填充存储中的一般带状矩阵
$\begin{bmatrix} a_{11} & a_{12} & a_{13} & 0 & 0 \\ a_{21} & a_{22} & a_{23} & a_{24} & 0 \\ 0 & a_{32} & a_{33} & a_{34} & a_{35} \\ 0 & 0 & a_{43} & a_{44} & a_{45} \\ 0 & 0 & 0 & a_{54} & a_{55} \end{bmatrix}$	$\begin{bmatrix} x & x & a_{13} & a_{24} & a_{35} \\ x & a_{12} & a_{23} & a_{34} & a_{45} \\ a_{11} & a_{22} & a_{33} & a_{44} & a_{55} \\ a_{21} & a_{32} & a_{43} & a_{54} & x \end{bmatrix}$

三角矩阵

三角矩阵有两种存储方案。在未填充方案中，矩阵存储在二维数组中，矩阵的所有元素和数组的元素之间存在一一对应关系，但不会在数组中设置或访问矩阵中的零条目（由 x 表示）。在填充存储方案中，矩阵的非零元素将按列填充在一维数组中。

可使用填充存储来存储三角矩阵。

三角带状矩阵	未填充存储中的三角矩阵	填充存储中的三角矩阵
$\begin{bmatrix} a_{11} & 0 & 0 \\ a_{21} & a_{22} & 0 \\ a_{31} & a_{32} & a_{33} \end{bmatrix}$	$\begin{bmatrix} a_{11} & x & x \\ a_{21} & a_{22} & x \\ a_{31} & a_{32} & a_{33} \end{bmatrix}$	$\begin{bmatrix} a_{11} \\ a_{21} \\ a_{31} \\ a_{22} \\ a_{32} \\ a_{33} \end{bmatrix}$

可使用二维数组将三角带状矩阵存储在填充存储中，如下所示。不会访问带 x 标记的元素。

三角带状矩阵	填充存储中的三角带状矩阵
$\begin{bmatrix} a_{11} & 0 & 0 \\ a_{21} & a_{22} & 0 \\ 0 & a_{32} & a_{33} \end{bmatrix}$	$\begin{bmatrix} a_{11} & a_{22} & a_{33} \\ a_{21} & a_{32} & x \end{bmatrix}$

对称矩阵

实数对称矩阵或复数厄尔米特矩阵与三角矩阵的相同之处在于，只有在其上三角或下三角中的元素才会显式存储在二维数组的相应元素中。既不会设置也不会访问数组其余的元素（由下方的 x 表示）。还可以将有效的上或下三角按列填充到一维数组中。

对称矩阵	未填充存储中的对称矩阵	填充存储中的对称矩阵
$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}$	$\begin{bmatrix} a_{11} & x & x \\ a_{21} & a_{22} & x \\ a_{31} & a_{32} & a_{33} \end{bmatrix}$	$\begin{bmatrix} a_{11} \\ a_{21} \\ a_{31} \\ a_{22} \\ a_{32} \\ a_{33} \end{bmatrix}$

三对角矩阵

三对角矩阵仅在主对角、第一个超对角和第一个次对角上有非零元素。使用三个一维数组对其进行存储。

三对角矩阵	三对角矩阵的存储
$\begin{bmatrix} a_{11} & a_{12} & 0 \\ a_{21} & a_{22} & a_{23} \\ 0 & a_{32} & a_{33} \\ 0 & 0 & a_{43} \end{bmatrix}$	$\begin{bmatrix} a_{21} \\ a_{32} \\ a_{43} \end{bmatrix} \begin{bmatrix} a_{11} \\ a_{22} \\ a_{33} \\ a_{44} \end{bmatrix} \begin{bmatrix} a_{12} \\ a_{23} \\ a_{34} \end{bmatrix}$

♦ ♦ ♦ 第 6 章

稀疏计算

Oracle Developer Studio 性能库有两个软件包 SPSOLVE 和 SuperLU，可用于计算稀疏线性方程组的因子并求解。

SPSOLVE 例程集合中的例程使用多种排序方法（包括用户指定的排序）之一对对称、结构对称和不对称系数矩阵进行求解。在以前的发行版中，SPSOLVE 被称为稀疏求解器包。它主要是用 Fortran 编写的，并且仅包含用于 FORTRAN 77 的接口。当前不提供 Fortran 95 和 C 接口。要从 Fortran 95 使用 SPSOLVE 例程，请使用 FORTRAN 77 接口。要从 C 调用 SPSOLVE，请将下划线附加到例程名称（`dgssin_`（）、`dgssor_`（） 等），通过引用传递参数，并使用从 1 开始的数组索引。有关从 1 开始和从 0 开始的数组索引，请参见“非对称稀疏矩阵” [43]。

Oracle Developer Studio 性能库中的 SuperLU 软件包是公共域应用程序的接续版本（版本 3.0），用于对一般非对称稀疏方程组进行求解。虽然是接续的，但 SuperLU 使用了多个并行化的 2 级和 3 级 BLAS 例程。有关 SuperLU 算法、例程和数据结构的详细说明，请参见“稀疏 BLAS 和求解器参考资料” [66]中的第 5、6、7 项。SuperLU 是用 C 编写的，它要求数组索引从 0 开始，不管 SuperLU 例程是从基于 Fortran 的 SPSOLVE 调用的，还是从 C 驱动程序调用的。有关更多详细信息和示例，请参见“SuperLU 接口” [56]。

稀疏矩阵

稀疏矩阵通常以所需存储最少的格式来表示。通过利用稀疏性并且不存储零，可以节约大量存储空间。SPSOLVE 和 SuperLU 使用的存储格式是压缩稀疏列 (Compressed Sparse Column, CSC) 格式，也称为 Harwell-Boeing 格式。

CSC 格式表示一个稀疏矩阵，它包含两个整数数组和一个浮点数组。整数数组（colptr 和 rowind）指定稀疏矩阵的非零值的位置，浮点数组（值）用于非零值。

列指针 (colptr) 数组由 n+1 个元素组成，其中 colptr(i) 指向第 i 列的开头，colptr(i+1)-1 指向第 i 列的末尾。行索引 (rowind) 数组包含非零值的行索引。值数组包含相应的非零数值。

由 neqns 方程和 nnz 非零值构成的稀疏矩阵有以下矩阵数据格式：

- 对称

- 结构对称
- 非对称

当前，SuperLU 仅支持非对称矩阵。最有效的数据表示形式通常取决于具体问题。以下各节显示了稀疏矩阵数据格式的示例。

对称稀疏矩阵

对称稀疏矩阵是这样一种矩阵：对于所有 *i* 和 *j*，a(*i*, *j*) = a(*j*, *i*)。由于此对称性，只有下三角形值需要传递到求解器例程。可根据下三角形确定上三角形。

对称矩阵的示例如下所示。此示例是从 A.George 和 J.W-H.Liu.合著的《Computer Solution of Large Sparse Positive Definite Systems》获得的。

$$A = \begin{bmatrix} 4.0 & 1.0 & 2.0 & 0.5 & 2.0 \\ 1.0 & 0.5 & 0.0 & 0.0 & 0.0 \\ 2.0 & 0.0 & 3.0 & 0.0 & 0.0 \\ 0.5 & 0.0 & 0.0 & 0.625 & 0.0 \\ 2.0 & 0.0 & 0.0 & 0.0 & 16.0 \end{bmatrix}$$

用 CSC 格式表示 *A*：

- colptr：1、6、7、8、9、10
- rowind：1、2、3、4、5、2、3、4、5
- 值：4.0、1.0、2.0、0.5、2.0、0.5、3.0、0.625、16.0

结构对称稀疏矩阵

结构对称稀疏矩阵的非零值的属性为：如果 a(*i*, *j*) ≠ 0，那么对于所有 *i* 和 *j*，a(*j*, *i*) ≠ 0。对结构对称方程组求解时，必须将整个矩阵传递到求解器例程。

结构对称矩阵的示例如下所示。

$$A = \begin{bmatrix} 1.0 & 3.0 & 0.0 & 0.0 \\ 2.0 & 4.0 & 0.0 & 7.0 \\ 0.0 & 0.0 & 6.0 & 0.0 \\ 0.0 & 5.0 & 0.0 & 8.0 \end{bmatrix}$$

用 CSC 格式表示 *A*：

- colptr：1、3、6、7、9
- rowind：1、2、1、2、4、3、2、4
- 值：1.0、2.0、3.0、4.0、5.0、6.0、7.0、8.0

非对称稀疏矩阵

非对称稀疏矩阵是，对于所有 *i* 和 *j*，没有 (*i*, *j*) = a(*j*, *i*)。矩阵的结构没有明显的模式。对非对称方程组求解时，必须将整个矩阵传递到求解器例程。非对称矩阵的示例如下所示。

$$A = \begin{bmatrix} 1.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 2.0 & 6.0 & 0.0 & 0.0 & 9.0 \\ 3.0 & 0.0 & 7.0 & 0.0 & 0.0 \\ 4.0 & 0.0 & 0.0 & 8.0 & 0.0 \\ 5.0 & 0.0 & 0.0 & 0.0 & 10.0 \end{bmatrix}$$

用 CSC 格式表示 *A*：

- 从 1 开始的索引：
  - colptr：1、6、7、8、9、11
  - rowind：1、2、3、4、5、2、3、4、2、5
  - 值：1.0、2.0、3.0、4.0、5.0、6.0、7.0、8.0、9.0、10.0
- 从 0 开始的索引：
  - colptr：0、5、6、7、8、10
  - rowind：0、1、2、3、4、1、2、3、1、4
  - 值：1.0、2.0、3.0、4.0、5.0、6.0、7.0、8.0、9.0、10.0

稀疏 BLAS

Oracle Developer Studio 性能库稀疏 BLAS 软件包基于以下两个软件包：

- Netlib Sparse BLAS 软件包，由 Dodson, Grimes 和 Lewis 开发，其中包括了基础线性代数子程序的稀疏扩展，用于对稀疏向量进行运算。
- NIST（美国国家标准与技术研究院，NIST）Fortran Sparse BLAS 库，其中包括的例程用于执行矩阵乘积，并对各种存储格式的稀疏矩阵的三角方程组进行求解。

有关其他稀疏 BLAS 信息，请参阅以下资源。

- 有关稀疏 BLAS 例程的信息，请参阅各例程的第 3P 部分手册页。
- 有关 Netlib Sparse BLAS 软件包的更多信息，请参阅 http://www.netlib.org/sparse-blas/index.html。
- 有关 NIST Fortran Sparse BLAS 例程的更多信息，请参阅 http://math.nist.gov/spblas/。

Netlib Sparse BLAS 和 NIST Fortran Sparse BLAS 库例程都使用自己的命名约定，如以下部分中的介绍。

Netlib Sparse BLAS

每个 Netlib Sparse BLAS 例程的名称都为“前缀-根-后缀”格式：

- 前缀表示数据类型。
- 根表示运算。
- 后缀表示例程是否是现有密集 BLAS 例程的直接扩展。

下表列出了 Netlib Sparse BLAS 向量例程的命名约定。

表 2 Netlib Sparse BLAS 命名约定

运算	名称的根	前缀和后缀
点积	-DOT-	S-I D-I C-UI Z-UI C-CI Z-CI
标量乘以一个向量，然后加上另一个向量	-AXPY-	S-I D-I C-I Z-I
应用吉文斯旋转	-ROT-	S-I D-I
将 x 集合到 y	-GTHR-	S- D- C- Z- S-Z D-Z C-Z Z-Z
将 x 分散到 y	-SCTR-	S- D- C- Z-

前缀可以是以下数据类型之一：

- S：SINGLE

- D：DOUBLE
- C：COMPLEX
- Z：COMPLEX*16 或 DOUBLE COMPLEX

I、CI 和 UI 后缀表示稀疏 BLAS 例程是密集 BLAS 例程的直接扩展。

NIST Fortran Sparse BLAS

每个 NIST Fortran Sparse BLAS 例程的名称都包含 6 个字符，格式为 *XYYYZZ*，其中：

- *X* 表示数据类型。
- *YYY* 表示稀疏存储格式。
- *ZZ* 表示运算。

下表显示了 *X*、*YYY* 和 *ZZ* 的可能值。

表 3　NIST Fortran Sparse BLAS 例程命名约定

例程名称中的变量	可接受的值和含义
X – 使用一个字符指定数据类型	S：单精度 D：双精度 C：复数 Z：双精度复数
YYY – 使用三个字符指定稀疏存储格式	单一条目格式： CSC：压缩稀疏列 COO：坐标 CSR：压缩稀疏行 DIA：对角线 ELL：ellpack JAD：锯齿对角线 SKY：天际线 块条目格式： BCO：块坐标 BSC：块压缩稀疏列 BSR：块压缩稀疏行 BDI：块对角线 BEL：块 ellpack VBR：块压缩稀疏行
ZZ – 使用两个字符指定运算	MM：矩阵-矩阵积 SM：三角方程组的解（除 COO 以外的所有格式都支持） RP：右排列（仅用于 JAD 格式）

SPSOLVE 接口

SPSOLVE 通过一系列步骤来计算稀疏方程组的解：初始化、排序以简化填充、符号因式分解、数值分解和三角求解。用户代码可以调用个体例程或使用一次调用的接口来执行这些步骤。

SPSOLVE 例程

下表列出了 SPSOLVE 中用户可访问的例程及其用途。

表 4 SPSOLVE 稀疏求解器例程

例程名称	说明
DGSSFS（）	稀疏求解器的一次调用接口
DGSSIN（）	稀疏求解器初始化
DGSSOR（）	填充简化顺序和符号因式分解
DGSSUO（）	设置用户指定的顺序排列并执行符号因式分解（替换 DGSSOR 进行调用）
DGSSFA（）	矩阵值输入和数值因式分解。
DGSSSL（）	三角求解
DGSSRP（）	返回求解器使用的排列
DGSSCO（）	返回系数矩阵的条件数估算值
DGSSDA（）	取消分配稀疏求解器
DGSSPS（）	打印求解器统计数据

可以按如下所示的顺序调用各个 SPSOLVE 例程，对结构相同但数值不同的矩阵进行求解：

```
call dgssin() ! initialization, input coefficient matrix structure
call dgssor() ! fill-reducing ordering, symbolic factorization
              ! (or call dgssuo() to specify a user ordering,
              ! and perform symbolic factorization)

do m = 1, number_of_structurally_identical_matrices
   call dgssfa() ! input coefficient matrix values, numeric                  ! factorization
   do r = 1, number_of_right_hand_sides
      call dgsssl() ! triangular solve
   enddo
enddo
```

一次调用接口不像标准接口那样灵活，但它涵盖了对单一矩阵进行因式分解以及对一些右侧数字求解的最常见情况。可以通过另外调用 dgsssl（） 来对其他右侧进行求解，如下例所示。

```
call dgssfs() ! initialization, input coefficient matrix structure
```

```
              ! fill-reducing ordering, symbolic factorization
              ! input coefficient matrix values, numeric factorization
              ! triangular solve
do r = 1, number_of_right_hand_sides
    call dgsssl() ! triangular solve
enddo
```

SPSOLVE 例程调用顺序

要使用 SPSOLVE，必须按如下所示的顺序调用其例程：

1. 一次调用接口：用于对单一矩阵求解
   a. `DGSSFS（）` - 初始化、排序、分解因子、求解
   b. `DGSSSL（）` - 其他求解（可选）：根据需要重复 `DGSSSL()`
   c. `DGSSDA（）` - 取消分配工作存储
2. 标准接口：用于对结构相同的多个矩阵进行求解
   a. `DGSSIN（）` - 初始化
   b. `DGSSOR（）` 或 `DGSSUO()` - 排序并以符号方式分解因子
   c. `DGSSFA（）` - 分解因子
   d. `DGSSSL（）` - 求解：根据需要重复 `DGSSFA（）` 或 `DGSSSL（）`
   e. `DGSSDA（）` - 取消分配工作存储

SPSOLVE 示例

以下示例显示了使用一次调用接口对对称方程组求解，并使用标准接口对对称方程组求解。

在例 1 “对对称方程组求解 – 一次调用接口”中，使用了一次调用接口对对称方程组求解，在例 2 “对对称方程组求解 – 标准接口”中，调用了个体例程来对对称方程组求解。

例 5 “从 C 调用 SPSOLVE 例程”显示了如何从 C 程序调用 Fortran SPSOLVE 接口。有关如何从 C 程序调用 Fortran 例程的更多信息，请参见《*Oracle Solaris Studio 12.4: Fortran Programming Guide*》。

例 1　　对对称方程组求解 – 一次调用接口

```
my_system% cat example_1call.f
      program example_1call
c
c  This program is an example driver that calls the sparse solver.
c    It factors and solves a symmetric system, by calling the
c    one-call interface.
c
```

```
      implicit none

      integer           neqns, ier, msglvl, outunt, ldrhs, nrhs
      character         mtxtyp*2, pivot*1, ordmthd*3
      double precision  handle(150)
      integer           colstr(6), rowind(9)
      double precision  values(9), rhs(5), xexpct(5)
      integer           i
c
c  Sparse matrix structure and value arrays.  From George and Liu,
c  page 3.
c    Ax = b, (solve for x) where:
c
c      4.0   1.0   2.0   0.5   2.0       2.0       7.0
c      1.0   0.5   0.0   0.0   0.0       2.0       3.0
c  A = 2.0   0.0   3.0   0.0   0.0   x = 1.0   b = 7.0
c      0.5   0.0   0.0   0.625 0.0      -8.0      -4.0
c      2.0   0.0   0.0   0.0  16.0      -0.5      -4.0
c
      data colstr / 1, 6, 7, 8, 9, 10 /
      data rowind / 1, 2, 3, 4, 5, 2, 3, 4, 5 /
      data values / 4.0d0, 1.0d0, 2.0d0, 0.5d0, 2.0d0, 0.5d0, 3.0d0,
     &              0.625d0, 16.0d0 /
      data rhs    / 7.0d0, 3.0d0, 7.0d0, -4.0d0, -4.0d0 /
      data xexpct / 2.0d0, 2.0d0, 1.0d0, -8.0d0, -0.5d0 /
c
c  set calling parameters
c
      mtxtyp= 'ss'
      pivot = 'n'
      neqns  = 5
      nrhs   = 1

      ldrhs  = 5
      outunt = 6
      msglvl = 0
      ordmthd = 'mmd'
c
c  call single call interface
c
      call dgssfs ( mtxtyp, pivot,  neqns , colstr, rowind,
     &              values, nrhs  , rhs,    ldrhs , ordmthd,
     &              outunt, msglvl, handle, ier             )
      if ( ier .ne. 0 ) goto 110
c
c  deallocate sparse solver storage
c
      call dgssda ( handle, ier )
      if ( ier .ne. 0 ) goto 110
c
c  print values of sol
c
      write(6,200) 'i', 'rhs(i)', 'expected rhs(i)', 'error'
      do i = 1, neqns
```

```
        write(6,300) i, rhs(i), xexpct(i), (rhs(i)-xexpct(i))
      enddo
      stop
  110 continue
c
c call to sparse solver returns an error
c
      write ( 6 , 400 )
     &      ' example: FAILED sparse solver error number = ', ier
      stop

  200 format(a5,3a20)

  300 format(i5,3d20.12) ! i/sol/xexpct values

  400 format(a60,i20) ! fail message, sparse solver error number

      end

my_system% f95 -dalign example_1call.f -library=sunperf
my_sytem% a.out
    i              rhs(i)     expected rhs(i)               error
    1  0.200000000000D+01  0.200000000000D+01 -0.528466159722D-13
    2  0.200000000000D+01  0.200000000000D+01  0.105249142734D-12
    3  0.100000000000D+01  0.100000000000D+01  0.350830475782D-13
    4 -0.800000000000D+01 -0.800000000000D+01  0.426325641456D-13
    5 -0.500000000000D+00 -0.500000000000D+00  0.660582699652D-14
```

例 2　　对对称方程组求解 – 标准接口

```
my_system% cat example_ss.f
      program example_ss
c
c  This program is an example driver that calls the sparse solver.
c  It factors and solves a symmetric system.

      implicit none

      integer           neqns, ier, msglvl, outunt, ldrhs, nrhs
      character         mtxtyp*2, pivot*1, ordmthd*3
      double precision  handle(150)
      integer           colstr(6), rowind(9)
      double precision  values(9), rhs(5), xexpct(5)
      integer           i
c
c  Sparse matrix structure and value arrays.  From George and Liu,
c  page 3.
c    Ax = b, (solve for x) where:
c
c      4.0   1.0   2.0   0.5   2.0       2.0       7.0
c      1.0   0.5   0.0   0.0   0.0       2.0       3.0
c  A = 2.0   0.0   3.0   0.0   0.0   x = 1.0   b = 7.0
c      0.5   0.0   0.0   0.625 0.0      -8.0      -4.0
```

```
c      2.0   0.0   0.0   0.0  16.0      -0.5      -4.0
c
      data colstr / 1, 6, 7, 8, 9, 10 /
      data rowind / 1, 2, 3, 4, 5, 2, 3, 4, 5 /
      data values / 4.0d0, 1.0d0, 2.0d0, 0.5d0, 2.0d0, 0.5d0,
     &             3.0d0, 0.625d0, 16.0d0 /
      data rhs    / 7.0d0, 3.0d0, 7.0d0, -4.0d0, -4.0d0 /
      data xexpct / 2.0d0, 2.0d0, 1.0d0, -8.0d0, -0.5d0 /

c
c  initialize solver
c
      mtxtyp= 'ss'
      pivot = 'n'
      neqns  = 5
      outunt = 6
      msglvl = 0
c
c  call regular interface
c
      call dgssin ( mtxtyp, pivot,  neqns , colstr, rowind,
     &              outunt, msglvl, handle, ier             )
      if ( ier .ne. 0 ) goto 110
c
c  ordering and symbolic factorization
c
      ordmthd = 'mmd'
      call dgssor ( ordmthd, handle, ier )
      if ( ier .ne. 0 ) goto 110
c
c  numeric factorization
c
      call dgssfa ( neqns, colstr, rowind, values, handle, ier )
      if ( ier .ne. 0 ) goto 110
c
c  solution
c
      nrhs   = 1
      ldrhs  = 5
      call dgsssl ( nrhs, rhs, ldrhs, handle, ier )
      if ( ier .ne. 0 ) goto 110
c
c  deallocate sparse solver storage
c
      call dgssda ( handle, ier )
      if ( ier .ne. 0 ) goto 110
c
c  print values of sol
c
      write(6,200) 'i', 'rhs(i)', 'expected rhs(i)', 'error'
      do i = 1, neqns
        write(6,300) i, rhs(i), xexpct(i), (rhs(i)-xexpct(i))
      enddo
      stop
```

```
  110 continue
c
c call to sparse solver returns an error
c
      write ( 6 , 400 )
     &      ' example: FAILED sparse solver error number = ', ier
      stop

  200 format(a5,3a20)

  300 format(i5,3d20.12) ! i/sol/xexpct values

  400 format(a60,i20) ! fail message, sparse solver error number

      end
my_system% f95 -dalign example_ss.f -library=sunperf
my_sytem% a.out
    i              rhs(i)     expected rhs(i)               error
    1  0.200000000000D+01  0.200000000000D+01 -0.528466159722D-13
    2  0.200000000000D+01  0.200000000000D+01  0.105249142734D-12
    3  0.100000000000D+01  0.100000000000D+01  0.350830475782D-13
    4 -0.800000000000D+01 -0.800000000000D+01  0.426325641456D-13
    5 -0.500000000000D+00 -0.500000000000D+00  0.660582699652D-14
```

例 3　　对具有非对称值的结构对称方程组求解 – 标准接口

```
my_system% cat example_su.f
      program example_su
c
c  This program is an example driver that calls the sparse solver.
c    It factors and solves a structurally symmetric system
c    (w/unsymmetric values).
c
      implicit none

      integer           neqns, ier, msglvl, outunt, ldrhs, nrhs
      character         mtxtyp*2, pivot*1, ordmthd*3
      double precision  handle(150)
      integer           colstr(5), rowind(8)
      double precision  values(8), rhs(4), xexpct(4)
      integer           i

c
c  Sparse matrix structure and value arrays.  Coefficient matrix
c    has a symmetric structure and unsymmetric values.
c    Ax = b, (solve for x) where:
c
c      1.0   3.0   0.0   0.0       1.0        7.0
c      2.0   4.0   0.0   7.0       2.0       38.0
c  A = 0.0   0.0   6.0   0.0   x = 3.0   b = 18.0
c      0.0   5.0   0.0   8.0       4.0       42.0
```

```
c
      data colstr / 1, 3, 6, 7, 9 /
      data rowind / 1, 2, 1, 2, 4, 3, 2, 4 /
      data values / 1.0d0, 2.0d0, 3.0d0, 4.0d0, 5.0d0, 6.0d0, 7.0d0,
     &              8.0d0 /
      data rhs    / 7.0d0, 38.0d0, 18.0d0, 42.0d0 /
      data xexpct / 1.0d0, 2.0d0, 3.0d0, 4.0d0 /
c
c  initialize solver
c
      mtxtyp= 'su'
      pivot = 'n'
      neqns  = 4
      outunt = 6
      msglvl = 0
c
c  call regular interface
c
      call dgssin ( mtxtyp, pivot,  neqns , colstr, rowind,
     &              outunt, msglvl, handle, ier             )
      if ( ier .ne. 0 ) goto 110
c
c  ordering and symbolic factorization
c
      ordmthd = 'mmd'
      call dgssor ( ordmthd, handle, ier )
      if ( ier .ne. 0 ) goto 110
c
c  numeric factorization
c
      call dgssfa ( neqns, colstr, rowind, values, handle, ier )
      if ( ier .ne. 0 ) goto 110

c
c  solution
c
      nrhs   = 1
      ldrhs  = 4
      call dgsssl ( nrhs, rhs, ldrhs, handle, ier )
      if ( ier .ne. 0 ) goto 110
c
c  deallocate sparse solver storage
c
      call dgssda ( handle, ier )
      if ( ier .ne. 0 ) goto 110
c
c  print values of sol
c
      write(6,200) 'i', 'rhs(i)', 'expected rhs(i)', 'error'
      do i = 1, neqns
        write(6,300) i, rhs(i), xexpct(i), (rhs(i)-xexpct(i))
      enddo
      stop
  110 continue
```

```
c
c call to sparse solver returns an error
c
      write ( 6 , 400 )
     &      ' example: FAILED sparse solver error number = ', ier
      stop

  200 format(a5,3a20)

  300 format(i5,3d20.12)     ! i/sol/xexpct values

  400 format(a60,i20)   ! fail message, sparse solver error number

      end
my_system% f95 -dalign example_su.f -library=sunperf
my_system% a.out
    i              rhs(i)     expected rhs(i)               error
    1  0.100000000000D+01  0.100000000000D+01  0.000000000000D+00
    2  0.200000000000D+01  0.200000000000D+01  0.000000000000D+00
    3  0.300000000000D+01  0.300000000000D+01  0.000000000000D+00
    4  0.400000000000D+01  0.400000000000D+01  0.000000000000D+00
```

例 4　　对非对称方程组求解 – 标准接口

```
my_system% cat example_uu.f
      program example_uu
c
c  This program is an example driver that calls the sparse solver.
c    It factors and solves an unsymmetric system.
c
      implicit none

      integer           neqns, ier, msglvl, outunt, ldrhs, nrhs
      character         mtxtyp*2, pivot*1, ordmthd*3
      double precision  handle(150)
      integer           colstr(6), rowind(10)
      double precision  values(10), rhs(5), xexpct(5)
      integer           i
c
c  Sparse matrix structure and value arrays.  Unsummetric matrix A.
c    Ax = b, (solve for x) where:
c
c      1.0   0.0   0.0   0.0   0.0       1.0        1.0
c      2.0   6.0   0.0   0.0   9.0       2.0       59.0
c  A = 3.0   0.0   7.0   0.0   0.0   x = 3.0   b = 24.0
c      4.0   0.0   0.0   8.0   0.0       4.0       36.0
c      5.0   0.0   0.0   0.0  10.0       5.0       55.0
c
      data colstr / 1, 6, 7, 8, 9, 11 /
      data rowind / 1, 2, 3, 4, 5, 2, 3, 4, 2, 5 /
      data values / 1.0d0, 2.0d0, 3.0d0, 4.0d0, 5.0d0, 6.0d0, 7.0d0,
     &              8.0d0, 9.0d0, 10.0d0 /
```

```
      data rhs    / 1.0d0, 59.0d0, 24.0d0, 36.0d0, 55.0d0 /
      data xexpct / 1.0d0, 2.0d0, 3.0d0, 4.0d0, 5.0d0 /
c
c  initialize solver
c
      mtxtyp= 'uu'
      pivot = 'n'
      neqns  = 5
      outunt = 6
      msglvl = 3
      call dgssin ( mtxtyp, pivot,  neqns , colstr, rowind,
     &              outunt, msglvl, handle, ier             )
      if ( ier .ne. 0 ) goto 110

c
c  ordering and symbolic factorization
c
      ordmthd = 'mmd'
      call dgssor ( ordmthd, handle, ier )
      if ( ier .ne. 0 ) goto 110
c
c  numeric factorization
c
      call dgssfa ( neqns, colstr, rowind, values, handle, ier )
      if ( ier .ne. 0 ) goto 110
c
c  solution
c
      nrhs   = 1
      ldrhs  = 5
      call dgsssl ( nrhs, rhs, ldrhs, handle, ier )
      if ( ier .ne. 0 ) goto 110
c
c  deallocate sparse solver storage
c
      call dgssda ( handle, ier )
      if ( ier .ne. 0 ) goto 110
c
c  print values of sol
c
      write(6,200) 'i', 'rhs(i)', 'expected rhs(i)', 'error'
      do i = 1, neqns
        write(6,300) i, rhs(i), xexpct(i), (rhs(i)-xexpct(i))
      enddo
      stop
  110 continue
c
c call to sparse solver returns an error
c
      write ( 6 , 400 )
     &      ' example: FAILED sparse solver error number = ', ier
      stop

  200 format(a5,3a20)
```

```
  300 format(i5,3d20.12)     ! i/sol/xexpct values

  400 format(a60,i20)    ! fail message, sparse solver error number
      end

my_system% f95 -dalign example_uu.f -library=sunperf
my_system% a.out
  i              rhs(i)     expected rhs(i)              error
    1  0.100000000000D+01  0.100000000000D+01  0.000000000000D+00
    2  0.200000000000D+01  0.200000000000D+01  0.000000000000D+00
    3  0.300000000000D+01  0.300000000000D+01  0.000000000000D+00
    4  0.400000000000D+01  0.400000000000D+01  0.000000000000D+00
    5  0.500000000000D+01  0.500000000000D+01  0.000000000000D+00
```

例 5　　从 C 调用 SPSOLVE 例程

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <sys/time.h>
#include <sunperf.h>

int main() {
/*
 Sparse matrix structure and value arrays. Coefficient matrix
 is a general unsymmetric sparse matrix.

 Ax = b, (solve for x) where:

     1.0  0.0  7.0   9.0   0.0        1.0           17.0
     2.0  4.0  0.0   0.0   0.0        1.0            6.0
 A = 0.0  5.0  8.0   0.0   0.0    x = 1.0       b = 13.0
     0.0  0.0  0.0  10.0  11.0        1.0           21.0
     3.0  6.0  0.0   0.0  12.0        1.0           21.0
*/
/* Array indices must be one-based for calling SPSOLVE routines */
int colstr[]    = {1, 4, 7, 9, 11, 13};
int rowind[]    = {1, 2, 5, 2, 3, 5, 1, 3, 1, 4, 4, 5};
double values[] = {1.0, 2.0, 3.0, 4.0, 5.0, 6.0,
                   7.0, 8.0, 9.0, 10.0, 11.0, 12.0};
double rhs[]    = {17.0, 6.0, 13.0, 21.0, 21.0};
double xexpct[] = {1.0, 1.0, 1.0, 1.0, 1.0};

    int n = 5, nnz = 12, nrhs = 1, msglvl = 0, outunt = 6, ierr,
          i,j,k, int_ierr;
    double t[4], handle[150];
    char type[] = "uu", piv = 'n';

/* Last two parameters in argument list indicate lengths of
 * character arguments type and piv
 */
```

```
    dgssin_(type, &piv, &n, colstr, rowind, &outunt, &msglvl,
              handle, &ierr,2,1);
    if (ierr != 0) {
      int_ierr = ierr;
      printf("dgssin err = %d\n", int_ierr);
      return -1;
    }

    char ordmth[] = "mmd";
    dgssor_(ordmth, handle, &ierr, 3);
    if (ierr != 0) {
      int_ierr = ierr;
      printf("dgssor err = %d\n", int_ierr);
      return -1;
    }

    dgssfa_(&n, colstr, rowind, values, handle, &ierr);
    if (ierr != 0) {
      int_ierr = ierr;
      printf("dgssfa err = %d\n", int_ierr);
      return -1;
    }

    dgsssl_(&nrhs, rhs, &n, handle, &ierr);
    if (ierr != 0) {
      int_ierr = ierr;
      printf("dgsssl err = %d\n", int_ierr);
      return -1;
    }
    printf("i   computed solution      expected solution\n");
    for (i=0; i<n; i++)
      printf("%d         %lf            %lf\n", i,rhs[i], 1.0);
}

my_system%  cc -m32 -xmemalign=8s dr.c -library=sunperf
my_system%  ./a.out
i   computed solution      expected solution
0         1.000000            1.000000
1         1.000000            1.000000
2         1.000000            1.000000
3         1.000000            1.000000
4         1.000000            1.000000
```

SuperLU 接口

SuperLU 有两个驱动程序例程，即简单和专业，调用这些例程对一般非对称稀疏方程组求解的方式与 SPSOLVE 中的一次调用接口完全相同。可在单精度、双精度、复数和双精度复数数据类型中使用这些例程以及用户可调用的其他 SuperLU 例程。以下各表中列出了所有外部例程的单精度名称。这些例程都有可用的手册页（第 3P 部分）。有关在应用程序中使用的稀疏矩阵数据结构的说明，另请参见 SuperMatrix(3P) 手册页。

表 5 SuperLU 计算例程

例程	说明
sgstrf	计算因式分解
sgssvx	分解和求解（专业驱动程序）
sgssv	分解和求解（简单驱动程序）
sgstrs	计算三角求解
sgsrfs	改进计算得到的解；提供误差界
slangs	计算 1 范数、弗罗贝尼乌斯范数或无穷范数
sgsequ	计算行和列比例
sgscon	估算条件数的倒数
slaqgs	平衡一般稀疏矩阵

表 6 SuperLU 实用程序例程

例程	说明
LUSolveTime	返回在求解阶段所用的时间
LUFactTime	返回在因式分解阶段所用的时间
LUFactFlops	返回因式分解阶段中的浮点运算数
LUSolveFlops	返回求解阶段中的浮点运算数
sQuerySpace	返回有关内存统计数据的信息
sp_ienv	返回指定的计算机相关参数
sPrintPerf	打印由计算例程收集的统计信息
set_default_options	将控制求解器行为的参数设置为默认选项
StatInit	分配并初始化用于存储性能统计信息的结构
StatFree	释放用于存储性能统计信息的结构
Destroy_Dense_Matrix	取消分配密集格式的 SuperMatrix
Destroy_SuperNode_Matrix	取消分配超节点格式的 SuperMatrix
Destroy_CompCol_Matrix	取消分配压缩稀疏列格式的 SuperMatrix
Destroy_CompCol_Permuted	取消分配已排列的压缩稀疏列格式的 SuperMatrix
Destroy_SuperMatrix_Store	取消分配将矩阵存储在 SuperMatrix 中的实际存储
sCopy_CompCol_Matrix	复制压缩稀疏列格式的 SuperMatrix
sCreate_CompCol_Matrix	分配压缩稀疏列格式的 SuperMatrix
sCreate_Dense_Matrix	分配密集格式的 SuperMatrix
sCreate_CompRow_Matrix	分配压缩稀疏行格式的 SuperMatrix
sCreate_SuperNode_Matrix	分配超节点格式的 SuperMatrix
sp_preorder	排列原始稀疏矩阵的列
sp_sgemm	将 SuperMatrix 乘以稠密矩阵

从 C 调用 SuperLU

SuperLU 例程是用 C 编写的。因此，与列和行相关的索引必须从 *0* 开始。在以下示例中，将调用双精度简单驱动程序 `dgssv（）` 来计算因子 L 和 U，并对解矩阵求解。

例 6 SuperLU 简单驱动程序

```
#include <stdio.h>
#include <sunperf.h>

#define M 5
#define N 5

int main(int argc, char *argv[])
{
    SuperMatrix A, L, U, B1, B2;
    int      perm_r[M];  /* row permutations from partial pivoting */
    int      perm_c[N];  /* column permutation vector */
    int      info, i;
    superlu_options_t options;
    SuperLUStat_t stat;
    trans_t  trans = NOTRANS;

    printf("Example code calling SuperLU simple driver to factor a \n");
    printf("general unsymmetric matrix and solve two right-hand-side matrices\n");

   /* the matrix in Harwell-Boeing format. */
    int m = M;
    int n = M;
    int nnz = 12;
    double *dp;
   /* nonzeros of A, column-wise */
    double a[] = {1.0, 2.0, 3.0,  4.0, 5.0,  6.0,
                  7.0, 8.0, 9.0, 10.0, 11.0, 12.0};
   /* row index of nonzeros */
    int asub[] = {0, 1, 4, 1, 2, 4, 0, 2, 0, 3, 3, 4};
   /* column pointers */
    int xa[]   = {0, 3, 6, 8, 10, 12};

    /* Create Matrix A in the format expected by SuperLU */
    dCreate_CompCol_Matrix(&A, m, n, nnz, a, asub, xa, SLU_NC, SLU_D, SLU_GE);

    int nrhs = 1;
    double rhs1[] = {17.0, 6.0, 13.0, 21.0, 21.0};
    double rhs2[] = {17*.3, 6*.3, 13*.3, 21*.3, 21*.3};

    /* right-hand side matrix B1, B2 */
    dCreate_Dense_Matrix(&B1, m, nrhs, rhs1, m, SLU_DN, SLU_D, SLU_GE);
    dCreate_Dense_Matrix(&B2, m, nrhs, rhs2, m, SLU_DN, SLU_D, SLU_GE);

    /* set options that control behavior of solver to default parameters */
    set_default_options(&options);
```

```
    options.ColPerm = NATURAL;

    /* Initialize the statistics variables. */
    StatInit(&stat);
    /* factor input matrix and solve the first right-hand-side matrix */
    dgssv(&options, &A, perm_c, perm_r, &L, &U, &B1, &stat, &info);

    printf("\nsolution matrix B1:\n");
    dp = (double *) (((NCformat *)B1.Store)->nzval);
    printf("    i    rhs[i]     expected\n");
    for (i=0; i<M; i++)
      printf("%5d   %7.4lf     %7.4lf\n", i, dp[i], 1.0);
    printf("Factor time  = %8.2e sec\n", stat.utime[FACT]);
    printf("Solve time   = %8.2e sec\n\n\n", stat.utime[SOLVE]);

    /* solve the second right-hand-side matrix */
    dgstrs(trans, &L, &U, perm_c, perm_r, &B2, &stat, &info);

    printf("solution matrix B2:\n");
    dp = (double *) (((NCformat *)B2.Store)->nzval);
    printf("    i    rhs[i]     expected\n");
    for (i=0; i<M; i++)
      printf("%5d   %7.4lf     %7.4lf\n", i, dp[i], 0.3);
    printf("Solve time   = %8.2e sec\n", stat.utime[SOLVE]);

    StatFree(&stat);
    Destroy_CompCol_Matrix(&A);
    Destroy_SuperMatrix_Store(&B1);
    Destroy_SuperMatrix_Store(&B2);
    Destroy_SuperNode_Matrix(&L);
    Destroy_CompCol_Matrix(&U);
}
```

运行上例：

```
my_system% cc -xmemalign=8s simple.c -library=sunperf
my_system% a.out

Example code calling SuperLU simple driver to factor a
general unsymmetric matrix and solve two right-hand-side matrices

solution matrix B1:
    i    rhs[i]     expected
    0    1.0000      1.0000
    1    1.0000      1.0000
    2    1.0000      1.0000
    3    1.0000      1.0000
    4    1.0000      1.0000
Factor time  = 5.43e-02 sec
Solve time   = 6.76e-03 sec


solution matrix B2:
    i    rhs[i]     expected
```

```
    0    0.3000      0.3000
    1    0.3000      0.3000
    2    0.3000      0.3000
    3    0.3000      0.3000
    4    0.3000      0.3000
Solve time   = 6.76e-03 sec
```

例 7 SuperLU 专业驱动程序

```
#include <stdio.h>
#include <sunperf.h>

#define M    5
#define N    5
#define NRHS 1

int main(int argc, char *argv[])
{
    SuperMatrix A, L, U, B, X;
    int      perm_r[M];  /* row permutations from partial pivoting */
    int      perm_c[N];  /* column permutation vector */
    int      etree[N];   /* elimination tree */
    double   ferr[NRHS]; /* estimated forward error bound */
    double   berr[NRHS]; /* component-wise relative backward error */
    double   C[N], R[M]; /* column and row scale factors */
    double   rpg, rcond;
    char     equed[1];   /* Specifies the form of equilibration that was done */
    double   *work, *dp; /* user-supplied workspace */
    int      lwork = 0;  /* 0 for workspace to be allocated by system malloc */
    int      info, i;
    superlu_options_t options;
    SuperLUStat_t stat;
    mem_usage_t    mem_usage;

    printf("Example code calling SuperLU expert driver\n\n");

    /* the matrix in Harwell-Boeing format. */
    int m = M;
    int n = M;
    int nnz = 12;
   /* nonzeros of A, column-wise */
    double a[] = {1.0, 2.0, 3.0,  4.0, 5.0,  6.0,
                  7.0, 8.0, 9.0, 10.0, 11.0, 12.0};
    /* row index of nonzeros */
    int asub[] = {0, 1, 4, 1, 2, 4, 0, 2, 0, 3, 3, 4};
    /* column pointers */
    int xa[]   = {0, 3, 6, 8, 10, 12};
    int nrhs = NRHS;
    double rhs[] = {17.0, 6.0, 13.0, 21.0, 21.0};

    /* Create Matrix A in the format expected by SuperLU */
    dCreate_CompCol_Matrix(&A, m, n, nnz, a, asub, xa, SLU_NC, SLU_D, SLU_GE);
```

```
    /* right-hand-side matrix B */
    dCreate_Dense_Matrix(&B, m, nrhs, rhs, m, SLU_DN, SLU_D, SLU_GE);

    /*  solution matrix X */
    dCreate_Dense_Matrix(&X, m, nrhs, rhs, m, SLU_DN, SLU_D, SLU_GE);
    set_default_options(&options);
    options.ColPerm = NATURAL;

    /* Initialize the statistics variables. */
    StatInit(&stat);

    dgssvx(&options, &A, perm_c, perm_r, etree, equed, R, C, &L, &U, work, lwork,
           &B, &X, &rpg, &rcond, ferr, berr, &mem_usage, &stat, &info);
    dp = (double *) (((NCformat *)X.Store)->nzval);
    printf("    i    rhs[i]     expected\n");
    for (i=0; i<M; i++)
      printf("%5d   %7.4lf     %7.4lf\n",
             i, dp[i], 1.0);
    printf("Factor time  = %8.2e sec\n", stat.utime[FACT]);
    printf("Solve time   = %8.2e sec\n", stat.utime[SOLVE]);

    StatFree(&stat);
    Destroy_CompCol_Matrix(&A);
    Destroy_SuperMatrix_Store(&B);
    Destroy_SuperNode_Matrix(&L);
    Destroy_CompCol_Matrix(&U);
}
```

运行上例：

```
my_system% cc -xmemalign=8s expert.c -library=sunperf
my_system% a.out
Example code calling SuperLU expert driver

    i    rhs[i]     expected
    0    1.0000      1.0000
    1    1.0000      1.0000
    2    1.0000      1.0000
    3    1.0000      1.0000
    4    1.0000      1.0000
Factor time  = 1.25e-03 sec
Solve time   = 1.70e-04 sec
```

从 Fortran 调用 SuperLU

从 Fortran 调用 SuperLU 的最简单方式是通过 SPSOLVE 接口。可选择 SuperLU，通过 SPSOLVE 中的初始化例程 `DGSSIN（）` 的输入参数 `MTXTYP` 对非对称稀疏矩阵求解。一次调用接口例程 `DGSSFS（）` 中也有相同的参数。

下表列出了 `MTXTYP` 的有效选项。要调用 SuperLU，可选择 'S0' 或 'S0' 作为矩阵类型。由于 SPSOLVE 是基于 Fortran 的，因此与输入矩阵相关的所有列和行索引都必须从 1

开始。然而，如果通过 DGSSIN（） 或 DGSSFS（） 调用 SuperLU（通过设置 MTXTYP = 's0' 或 'S0'），则这些索引必须从 0 开始。

表 7 DGSSIN（） 和 DGSSFS（） 的矩阵类型选项

选项	矩阵类型	求解器
'sp' 或 'SP'	对称结构，正定值	SPSOLVE
'ss' 或 'SS'	对称结构，对称值	SPSOLVE
'su' 或 'SU'	对称结构，非对称值	SPSOLVE
'uu' 或 'UU'	非对称结构，非对称值	SPSOLVE
's0' 或 'S0'	非对称结构，非对称值	SuperLU

在调用例程 DGSSIN（） 之后必须调用例程 DGSSOR（），以执行填充简化排序和符号因式分解。可使用一个字符参数 (ORDMTHD) 选择所需的排序方法。一次调用接口例程 DGSSFS（） 中也存在此参数。下表列出了 SPSOLVE 和 SuperLU 的有效排序方法。也可以通过调用 DGSSUO（） 而不是 DGSSOR（） 来为求解器提供特定的排序方法。输入排列数组必须从 0 开始。

表 8 DGSSOR（） 和 DGSSFS（） 的矩阵排序选项

选项	排序方法	求解器
'nat' 或 'NAT'	自然排序（无排序）	SPSOLVE、SuperLU
'mmd' 或 'MMD'	A'*A 的最小度（默认值）	SPSOLVE、SuperLU
'gnd' 或 'GND'	一般嵌套分割	SPSOLVE
'spm' 或 'SPM'	基于 A'+A 的最小度排序	SuperLU
'sam' 或 'SAM'	近似最小度列	SuperLU

如上所示，一般嵌套分割方法在 SuperLU 中不可用。另一方面，基于 A'+A 的最小度排序和近似最小度列排序在 SPSOLVE 中不可用。

SuperLU 示例

以下代码示例显示了如何通过标准接口和 SPSOLVE 的一次调用接口选择 SuperLU 来对一般非对称方程组进行因式分解和求解。

例 8 通过 SPSOLVE 标准接口调用 SuperLU

```
      program SLU

c  This program is an example driver that calls the regular interface of SPSOLVE
c  to invoke SuperLU to factor and solve a general unsymmetric system.

      implicit none
```

```
      integer           neqns, ier, msglvl, outunt, ldrhs, nrhs, i
      character         mtxtyp*2, pivot*1, ordmthd*3
      double precision  handle(150)
      integer           colstr(6), rowind(12)
      double precision  values(12), rhs(5), xexpct(5)

c  Sparse matrix structure and value arrays.  Coefficient matrix
c    is a general unsymmetric sparse matrix.
c    Ax = b, (solve for x) where:

c       1.0   0.0   7.0   9.0   0.0      1.0       17.0
c       2.0   4.0   0.0   0.0   0.0      1.0        6.0
c  A =  0.0   5.0   8.0   0.0   0.0  x = 1.0   b = 13.0
c       0.0   0.0   0.0  10.0  11.0      1.0       21.0
c       3.0   6.0   0.0   0.0  12.0      1.0       21.0

c  Array indices must be zero-based for calling SuperLU
      data colstr / 0, 3, 6, 8, 10, 12 /
      data rowind / 0, 1, 4, 1, 2, 4, 0, 2, 0, 3, 3, 4 /
      data values / 1.0, 2.0, 3.0,  4.0, 5.0,  6.0,
     $              7.0, 8.0, 9.0, 10.0, 11.0, 12.0 /
      data rhs    / 17.0, 6.0, 13.0, 21.0, 21.0 /
      data xexpct / 1.0d0, 1.0d0, 1.0d0, 1.0d0, 1.0d0 /

c  initialize solver
      mtxtyp= 's0'
      pivot = 'n'
      neqns  = 5
      outunt = 6
      msglvl = 0

c  call regular interface
      call dgssin(mtxtyp, pivot,  neqns , colstr, rowind,outunt, msglvl,
     &            handle, ier)
      if ( ier .ne. 0 ) goto 110

c  ordering and symbolic factorization
      ordmthd = 'mmd'
      call dgssor(ordmthd, handle, ier)
      if ( ier .ne. 0 ) goto 110

c  numeric factorization
      call dgssfa ( neqns, colstr, rowind, values, handle, ier )
      if ( ier .ne. 0 ) goto 110

c  solution
      nrhs   = 1
      ldrhs  = 5
      call dgsssl ( nrhs, rhs, ldrhs, handle, ier )
      if ( ier .ne. 0 ) goto 110

c  deallocate sparse solver storage
      call dgssda ( handle, ier )
      if ( ier .ne. 0 ) goto 110
```

```
c  print values of sol
      write(6,200) 'i', 'rhs(i)', 'expected rhs(i)', 'error'
      do i = 1, neqns
        write(6,300) i, rhs(i), xexpct(i), (rhs(i)-xexpct(i))
      enddo
      stop

  110 continue
c call to sparse solver returns an error
      write ( 6 , 400 )
     &      ' example: FAILED sparse solver error number = ', ier
      stop

  200 format(4x,a1,3x,a6,3x,a15,4x,a6)
  300 format(i5,3x,f5.2,7x,f5.2,8x,e10.2)     ! i/sol/xexpct values
  400 format(a60,i20)   ! fail message, sparse solver error number
      end
```

运行上例：

```
my_system% f95 -dalign slu.f -library=sunperf
my_system% a.out

    i   rhs(i)   expected rhs(i)     error
    1    1.00        1.00          0.00E+00
    2    1.00        1.00         -0.33E-15
    3    1.00        1.00          0.22E-15
    4    1.00        1.00         -0.11E-15
    5    1.00        1.00          0.22E-15
```

例 9 通过一次调用 SPSOLVE 接口调用 SuperLU

```
program SLU_SINGLE
c  This program is an example driver that calls the regular interface of SPSOLVE
c  to invoke SuperLU to factor and solve a general unsymmetric system.

      implicit none
      integer           neqns, ier, msglvl, outunt, ldrhs, nrhs, i
      character         mtxtyp*2, pivot*1, ordmthd*3
      double precision  handle(150)
      integer           colstr(6), rowind(12)
      double precision  values(12), rhs(5), xexpct(5)

c  Sparse matrix structure and value arrays.  Coefficient matrix
c    is a general unsymmetric sparse matrix.
c    Ax = b, (solve for x) where:

c       1.0   0.0   7.0   9.0   0.0      1.0       17.0
c       2.0   4.0   0.0   0.0   0.0      1.0        6.0
c  A =  0.0   5.0   8.0   0.0   0.0  x = 1.0   b = 13.0
c       0.0   0.0   0.0  10.0  11.0      1.0       21.0
```

```
c       3.0   6.0   0.0   0.0  12.0      1.0       21.0

c  Array indices must be zero-based for calling SuperLU
      data colstr / 0, 3, 6, 8, 10, 12 /
      data rowind / 0, 1, 4, 1, 2, 4, 0, 2, 0, 3, 3, 4 /
      data values / 1.0, 2.0, 3.0,  4.0, 5.0,  6.0,
     $              7.0, 8.0, 9.0, 10.0, 11.0, 12.0 /
      data rhs    / 17.0, 6.0, 13.0, 21.0, 21.0 /
      data xexpct / 1.0d0, 1.0d0, 1.0d0, 1.0d0, 1.0d0 /

c  initialize solver
      mtxtyp= 's0'
      pivot = 'n'
      neqns  = 5
      outunt = 6
      msglvl = 0
      ordmthd = 'mmd'
      nrhs = 1
      ldrhs = 5

c  One-call routine of SPSOLVE
      call dgssfs (mtxtyp, pivot, neqns , colstr, rowind,
     &             values, nrhs , rhs, ldrhs , ordmthd,
     &             outunt, msglvl, handle, ier)
      if ( ier .ne. 0 ) goto 110

c  deallocate sparse solver storage
      call dgssda ( handle, ier )
      if ( ier .ne. 0 ) goto 110

c  print values of sol
      write(6,200) 'i', 'rhs(i)', 'expected rhs(i)', 'error'
      do i = 1, neqns
        write(6,300) i, rhs(i), xexpct(i), (rhs(i)-xexpct(i))
      enddo
      stop

  110 continue
c call to sparse solver returns an error
      write ( 6 , 400 )
     &      ' example: FAILED sparse solver error number = ', ier
      stop

  200 format(4x,a1,3x,a6,3x,a15,4x,a6)
  300 format(i5,3x,f5.2,7x,f5.2,8x,e10.2)     ! i/sol/xexpct values
  400 format(a60,i20)   ! fail message, sparse solver error number
      end
```

运行上例：

```
my_system% f95 -dalign slu_single.f -library=sunperf
my_system% a.out

    i   rhs(i)   expected rhs(i)     error
```

```
1    1.00        1.00          0.00E+00
2    1.00        1.00         -0.33E-15
3    1.00        1.00          0.22E-15
4    1.00        1.00         -0.11E-15
5    1.00        1.00          0.22E-15
```

稀疏 BLAS 和求解器参考资料

以下书籍和论文提供了有关稀疏 BLAS 和稀疏求解器例程的更多信息。

1. 由 D.S. Dodson、R.G. Grimes 和 J.G. Lewis 合著的《Sparse Extensions to the Fortran Basic Linear Algebra Subprograms》，发表于《ACM Transactions on Mathematical Software》，1991 年 6 月，第 17 卷 2 号。
2. 由 A.George 和 J.W-H.Liu 合著的《Computer Solution of Large Sparse Positive Definite Systems》，Prentice-Hall Inc.，新泽西州恩格尔伍德克利夫斯，1981 年。
3. 由 E.Ng 和 B. W. Peyton 合著的《Block Sparse Cholesky Algorithms on Advanced Uniprocessor Computers》，SIAM M.Sci Comput.，14:1034-1056，1993 年。
4. 由 Ian S.Duff、Roger G.Grimes 和 John G.Lewis 合著的《User's Guide for the Harwell-Boeing Sparse Matrix Collection (Release I)》，发表于 Technical Report TR/PA/92/86，CERFACS，法国里昂，1992 年 10 月。
5. 由 J. W. Demmel、J. R. Gilbert 和 X. S. Li 合著的《SuperLU User's Guide》，发表于 Technical Report LBNL-44289。
6. 由 X. S. Li 编著的《An Overview of SuperLU: Algorithms, Implementation, and User Interface, ACM Transactions on Mathematical Software》，2004 年。
7. 由 J. W. Demmel、S. C. Eisenstat、J. R. Gilbert、X. S. Li、J. W. H.Liu 合著的《A supernodal approach to sparse partial pivoting》，发表于《SIAM J.Matrix Analysis and Applications》，第 20 卷 3 号，1999 年，第 720-755 页。

♦ ♦ ♦ 第 7 章

使用 Oracle Developer Studio 性能库信号处理例程

在许多科学与工程领域中，离散傅里叶变换 (Discrete Fourier Transform, DFT) 始终是一个重要的分析工具。然而，直到开发了快速傅里叶变换 (Fast Fourier Transform, FFT) 之后，DFT 才被广泛使用。这是因为 DFT 需要进行 $O(N^2)$ 计算，而 FFT 只需要进行 $O(N\log_2 N)$ 运算。

Oracle Developer Studio 性能库包含一组例程，可计算 FFT、相关 FFT 运算（如卷积和相关）以及三角变换。

本章分为以下三节。

- 正向和逆向 FFT 例程
- 正弦和余弦变换
- 卷积和相关

每节都包括了相应的示例来说明如何使用例程。

提示 - 有关在每个例程中使用的 Fortran 95 和 C 接口以及参数类型的信息，请参见各个例程的第 3P 部分手册页。手册页的例程名称必须是小写的。

例如，要显示 `SFFTC` 例程的手册页，请使用以下命令，并用小写指定例程名称：

```
% man -s 3P sfftc
```

要获得 FFT 例程的概述，请使用以下命令：

```
% man -s 3P fft
```

正向和逆向 FFT 例程

以下各表列出了 FFT 例程的名称及其调用序列。双精度例程名称位于方括号中。有关数据类型和参数大小的详细信息，请参见各手册页。

- 表 9 “FFT 线性例程及其参数”

- 表 10 “FFT 二维例程及其参数”
- 表 11 “FFT 三维例程及其参数”

表 9 FFT 线性例程及其参数

例程名称	参数
CFFTS [ZFFTD]	(OPT, N1, SCALE, X, Y, TRIGS, IFAC, WORK, LWORK, ERR)
SFFTC [DFFTZ]	(OPT, N1, SCALE, X, Y, TRIGS, IFAC, WORK, LWORK, ERR)
CFFTSM [ZFFTDM]	(OPT, N1, N2, SCALE, X, LDX1, Y, LDY1, TRIGS, IFAC, WORK, LWORK, ERR)
SFFTCM [DFFTZM]	(OPT, N1, N2, SCALE, X, LDX1, Y, LDY1, TRIGS, IFAC, WORK, LWORK, ERR)
CFFTC [ZFFTZ]	(OPT, N1, SCALE, X, Y, TRIGS, IFAC, WORK, LWORK, ERR)
CFFTCM [ZFFTZM]	(OPT, N1, N2, SCALE, X, LDX1, Y, LDY1, TRIGS, IFAC, WORK, LWORK, ERR)

表 10 FFT 二维例程及其参数

例程名称	参数
CFFTS2 [ZFFTD2]	(OPT, N1, N2, SCALE, X, LDX1, Y, LDY1, TRIGS, IFAC, WORK, LWORK, ERR)
SFFTC2 [DFFTZ2]	(OPT, N1, N2, SCALE, X, LDX1, Y, LDY1, TRIGS, IFAC, WORK, LWORK, ERR)
CFFTC2 [ZFFTZ2]	(OPT, N1, N2, SCALE, X, LDX1, Y, LDY1, TRIGS, IFAC, WORK, LWORK, ERR)

表 11 FFT 三维例程及其参数

例程名称	参数
CFFTS3 [ZFFTD3]	(OPT, N1, N2, N3, SCALE, X, LDX1, LDX2, Y, LDY1, LDY2, TRIGS, IFAC, WORK, LWORK, ERR)
SFFTC3 [DFFTZ3]	(OPT, N1, N2, N3, SCALE, X, LDX1, LDX2, Y, LDY1, LDY2, TRIGS, IFAC, WORK, LWORK, ERR)
CFFTC3 [ZFFTZ3]	(OPT, N1, N2, N3, SCALE, X, LDX1, LDX2, Y, LDY1, LDY2, TRIGS, IFAC, WORK, LWORK, ERR)

Oracle Developer Studio 性能库 FFT 例程使用以下参数。

- `OPT`：此标志指示是调用例程来初始化变换还是来计算变换。
- `N1`、`N2`、`N3`：一维、二维和三维变换的问题维。

- X：输入数组，如果例程是复数到复数变换或复数到实数变换，则 X 的类型是 COMPLEX。对于实数到复数变换，X 的类型是 REAL。
- Y：输出数组，如果例程是复数到复数变换或实数到复数变换，则 Y 的类型是 COMPLEX。对于复数到实数变换，Y 的类型是 REAL。
- LDX1、LDX2 和 LDY1、LDY2：LDX1 和 LDX2 是输入数组的主维，LDY1 和 LDY2 是输出数组的主维。FFT 例程允许输出覆盖输入，这是场内变换，或允许输出存储在除输入数组以外的另一个数组中，这是场外变换。在复数到复数变换中，输入数据的大小与输出数据相同。然而，实数到复数变换以及复数到实数变换对输入数据和输出数据有不同的内存要求。在计算场内变换时必须谨慎，应确保输入数组足够大，以容纳变换结果。
- TRIGS：包含三角权重的数组。
- IFAC：包含问题维的因子的数组。问题大小如下：
  - 线性 FFT：维 N1 的问题大小
  - 二维 FFT：维 N1 和 N2 的问题大小
  - 三维 FFT：维 N1、N2 和 N3 的问题大小

  虽然 N1、N2 和 N3 可以是任意大小，但在以下情况下，可以最有效地计算实数到复数变换或复数到实数变换：

  $$N1, N2, N3 = 2^p \times 3^q \times 4^r \times 5^s$$

  ，

  在以下情况下，可以最有效地计算复数到复数变换：

  $$N1, N2, N3 = 2^p \times 3^q \times 4^r \times 5^s \times 7^t \times 11^u \times 13^v$$

  ，

  其中 p、q、r、s、t、u 和 v 是整数，且 p、q、r、s、t、u、$v \geq 0$。
- WORK：工作区，其大小取决于例程以及用于计算变换的线程数量（如果该例程已并行化）。
- LWORK：工作区的大小。如果 LWORK 是零，例程将分配具有所需大小的工作区。
- SCALE：对输出进行缩放的一个标量。有时，对于一维变换，使用 1/N1 的比例因子定义逆变换，对二维变换使用 1/(N1 × N2)，对三维变换使用 1/(N1 × N2 × N3)。在这种情况下，我们说对逆变换进行了归一化处理。如果归一化的 FFT 之后是逆向 FFT，结果就是原始的输入数据。Oracle Developer Studio 性能库 FFT 例程没有进行归一化处理。不过，通过使用存储在 SCALE 中的合适比例因子调用逆向 FFT 例程，可以很容易地进行归一化处理。
- ERR：如果在例程中遇到错误，该标志将返回非零值，否则返回零。

线性 FFT 例程

线性 FFT 例程仅计算一维的实数或复数数据的 FFT。这些数据可以是一个或多个复数或实数序列。对于单一序列，数据将存储在向量中。如果要转换多个序列，则这些序列将

逐列存储在二维数组中，并沿列方向计算每个序列的一维 FFT。线性正向 FFT 例程计算：$X(k)=\sum_{n=0}^{N1-1} x(n)e^{\frac{-2\pi ink}{N1}}, k=0,\ldots,N1-1,$ 其中 $i=\sqrt{-1}$ 。

或以极坐标形式表示为：

$$X(k)=\sum_{n=0}^{N1-1} x(n)\left(\cos\left(\frac{2\pi nk}{N1}\right)-i\sin\left(\frac{2\pi nk}{N1}\right)\right), k=0,\ldots,N1-1$$

。

逆向 FFT 例程计算 $x(n)=\sum_{k=0}^{N1-1} X(k)e^{\frac{2\pi ink}{N1}}, n=0,\ldots,N1-1,$

以极坐标形式表示为：

$$x(n)=\sum_{n=0}^{N1-1} X(k)\left(\cos\left(\frac{2\pi nk}{N1}\right)+i\sin\left(\frac{2\pi nk}{N1}\right)\right), k=0,\ldots,N1-1$$

通过正变换，如果输入是大小为 *N*1 的一个或多个复数序列，结果将是一个或多个复数序列，每个序列由 *N*1 个不相关的数据点组成。然而，如果输入是一个或多个实数序列，每个序列包含 *N*1 个实数数据点，则结果将是共轭对称的一个或多个复数序列。即：

$$X(k)=X*(N1-k), k=\frac{N1}{2}+1,\ldots,N1-1$$

X(0) 的虚部始终为零。如果 *N*1 是偶数，$X(\frac{N1}{2})$ 也为零。将显式存储两个零。因为可从每个序列的前半部分得到后半部分，所以仅计算 $\frac{N1}{2}+1$ 复数数据点并将其存储在输出数组中。在本章此处和其他地方，将对整数除法进行向下舍入。

通过逆变换，如果要计算 *N*1 点复数到复数变换，则每个输入序列中应当有 *N*1 个不相关数据点，并在输出数组中返回 *N*1 个数据点。然而，如果要计算 *N*1 点复数到实数变换，则输入中只应当有每个共轭对称输入序列的前 $\frac{N1}{2}+1$ 个复数数据点，例程将在每个输出序列中返回 *N*1 个实数数据点。

对于 *N*1 的每个值，在计算实际 FFT 之前，必须调用正向或逆向例程来计算 *N*1 的因子以及与这些因子关联的三角权重。只要 *N*1 保持不变，就可以在后续变换中重用这些因子和三角权重。

下面的表 12 “单精度线性 FFT 例程”列出了单精度线性 FFT 例程及其用途。对于使用二维数组作为输入和输出的例程，表 12 “单精度线性 FFT 例程”还列出了主维要求。这

些信息同样适用于相应的双精度例程，只是其数据类型是双精度和双精度复数。有关映射，请参见表 12 “单精度线性 FFT 例程”。有关例程及其参数的完整说明，请参见各手册页。

表 12 单精度线性 FFT 例程

名称	用途	大小和输入类型	大小和输出类型	主维要求	
SFFTC	OPT = 0 初始化				
	OPT = -1 单一向量的实数到复数正向线性 FFT	*N*1, 实数	$\frac{N1}{2}+1$, 复数		
SFFTC	OPT = 0 初始化				
	OPT = 1 单一向量的复数到实数逆向线性 FFT	$\frac{N1}{2}+1$, 复数	*N*1 实数		
CFFTC	OPT = 0 初始化				
	OPT = -1 单一向量的复数到复数正向线性 FFT	*N*1, 复数	*N*1, 复数		
	OPT = 1 单一向量的复数到复数逆向线性 FFT	*N*1, 复数	*N*1, 复数		
SFFTCM	OPT = 0 初始化				
	OPT = -1 M 个向量的实数到复数正向线性 FFT	*N*1 × *M*, 实数	$\left(\frac{N1}{2}+1\right)\times M$, 复数	LDX1 = 2 × LDY1	LDX1 ≥ *N*1
CFFTSM	OPT = 0 初始化				
	OPT = 1 M 个向量的复数到实数逆向线性 FFT	$\left(\frac{N1}{2}+1\right)\times M$, 复数	*N*1 × *M*, 实数	LDX1 ≥ $\frac{N1}{2}+1$ LDY1=2 × LDX1	LDX1 ≥ $\frac{N1}{2}+1$ LDY1 ≥ *N*1
CFFTCM	OPT = 0 初始化				
	OPT = -1 M 个向量的复数到复数正向线性 FFT	*N*1 × *M*, 复数	*N*1 × *M*, 复数	LDX1 ≥ *N*1 LDY1 ≥ *N*1	LDX1 ≥ *N*1 LDY1 ≥ *N*1
	OPT = 1 M 个向量的复数到复数逆向线性 FFT	*N*1 × *M*, 复数	*N*1 × *M*, 复数	LDX1 ≥ *N*1 LDY1 ≥ *N*1	LDX1 ≥ *N*1 LDY1 ≥ *N*1

注 - 请注意以下有关表 12 “单精度线性 FFT 例程”的信息：

- LDX1 是输入数组的主维。
- LDY1 是输出数组的主维。
- N1 是 FFT 问题的第一个维。
- N2 是 FFT 问题的第二个维。
- 使用 OPT = 0 调用例程以初始化例程时，唯一进行的错误检查是确定是否 *N*1 < 0

以下示例说明了如何计算一组序列的线性实数到复数和复数到实数 FFT。

例 10 线性实数到复数 FFT 和复数到实数 FFT

```
my_system% cat testscm.f
       PROGRAM TESTSCM
       IMPLICIT NONE
       INTEGER :: LW, IERR, I, J, K, LDX, LDC
       INTEGER,PARAMETER :: N1 = 3, N2 = 2, LDZ = N1,
      $        LDC = N1, LDX = 2*LDC
       INTEGER, DIMENSION(:) :: IFAC(128)
       REAL :: SCALE
       REAL, PARAMETER :: ONE = 1.0
       REAL, DIMENSION(:) :: SW(N1), TRIGS(2*N1)
       REAL, DIMENSION(0:LDX-1,0:N2-1) :: X, V, Y
       COMPLEX, DIMENSION(0:LDZ-1, 0:N2-1) :: Z
* workspace size
       LW = N1
       SCALE = ONE/N1
       WRITE(*,*)
      $ 'Linear complex-to-real and real-to-complex FFT of a sequence'
       WRITE(*,*)
       X = RESHAPE(SOURCE = (/.1, .2, .3,0.0,0.0,0.0,7.,8.,9.,
      $    0.0, 0.0, 0.0/), SHAPE=(/6,2/))
       V = X
       WRITE(*,*) 'X = '
       DO I = 0,N1-1
         WRITE(*,'(2(F4.1,2x))') (X(I,J), J = 0, N2-1)
       END DO
       WRITE(*,*)
* intialize trig table and compute factors of N1
       CALL SFFTCM(0, N1, N2, ONE, X, LDX, Z, LDZ, TRIGS, IFAC,
      $ SW, LW, IERR)
       IF (IERR .NE. 0) THEN
          PRINT*,'ROUTINE RETURN WITH ERROR CODE = ', IERR
          STOP
       END IF

* Compute out-of-place forward linear FFT.
* Let FFT routine allocate memory.
       CALL SFFTCM(-1, N1, N2, ONE, X, LDX, Z, LDZ, TRIGS, IFAC,
```

```
      $            SW, 0, IERR)
       IF (IERR .NE. 0) THEN
         PRINT*,'ROUTINE RETURN WITH ERROR CODE = ', IERR
         STOP
       END IF
       WRITE(*,*) 'out-of-place forward FFT of X:'
       WRITE(*,*)'Z ='
       DO I = 0, N1/2
          WRITE(*,'(2(A1, F4.1,A1,F4.1,A1,2x))') ('(',REAL(Z(I,J)),
     $ ',',AIMAG(Z(I,J)),')', J = 0, N2-1)
       END DO
       WRITE(*,*)
* Compute in-place forward linear FFT.
* X must be large enough to store N1/2+1 complex values
       CALL SFFTCM(-1, N1, N2, ONE, X, LDX, X, LDC, TRIGS, IFAC,
      $            SW, LW, IERR)
       IF (IERR .NE. 0) THEN
          PRINT*,'ROUTINE RETURN WITH ERROR CODE = ', IERR
          STOP
       END IF
       WRITE(*,*) 'in-place forward FFT of X:'
       CALL PRINT_REAL_AS_COMPLEX(N1/2+1, N2, 1, X, LDC, N2)
       WRITE(*,*)
* Compute out-of-place inverse linear FFT.
       CALL CFFTSM(1, N1, N2, SCALE, Z, LDZ, X, LDX, TRIGS, IFAC,
      $            SW, LW, IERR)
       IF (IERR .NE. 0) THEN
          PRINT*,'ROUTINE RETURN WITH ERROR CODE = ', IERR
          STOP
       END IF
       WRITE(*,*) 'out-of-place inverse FFT of Z:'
       DO I = 0, N1-1
          WRITE(*,'(2(F4.1,2X))') (X(I,J), J = 0, N2-1)
       END DO
       WRITE(*,*)
* Compute in-place inverse linear FFT.
       CALL CFFTSM(1, N1, N2, SCALE, Z, LDZ, Z, LDZ*2, TRIGS,
      $            IFAC, SW, 0, IERR)
       IF (IERR .NE. 0) THEN
          PRINT*,'ROUTINE RETURN WITH ERROR CODE = ', IERR
          STOP
       END IF

       WRITE(*,*) 'in-place inverse FFT of Z:'
       CALL PRINT_COMPLEX_AS_REAL(N1, N2, 1, Z, LDZ*2, N2)
       WRITE(*,*)
       END PROGRAM TESTSCM
       SUBROUTINE PRINT_COMPLEX_AS_REAL(N1, N2, N3, A, LD1, LD2)
       INTEGER N1, N2, N3, I, J, K
       REAL A(LD1, LD2, *)
       DO K = 1, N3
          DO I = 1, N1
             WRITE(*,'(5(F4.1,2X))') (A(I,J,K), J = 1, N2)
          END DO
```

```
         WRITE(*,*)
      END DO
      END
      SUBROUTINE PRINT_REAL_AS_COMPLEX(N1, N2, N3, A, LD1, LD2)
      INTEGER N1, N2, N3, I, J, K
      COMPLEX A(LD1, LD2, *)
      DO K = 1, N3
         DO I = 1, N1
            WRITE(*,'(5(A1, F4.1,A1,F4.1,A1,2X))') ('(',REAL(A(I,J,K)),
     $            ',',AIMAG(A(I,J,K)),')', J = 1, N2)
         END DO
         WRITE(*,*)
      END DO
      END
my_system% f95 -dalign testscm.f -xlibrary=sunperf
my_system% a.out
Linear complex-to-real and real-to-complex FFT of a sequence
X =
0.1 7.0
0.2 8.0
0.3 9.0
out-of-place forward FFT of X:
Z =
( 0.6, 0.0) (24.0, 0.0)
(-0.2, 0.1) (-1.5, 0.9)
in-place forward FFT of X:
( 0.6, 0.0) (24.0, 0.0)
(-0.2, 0.1) (-1.5, 0.9)
out-of-place inverse FFT of Z:
0.1 7.0
0.2 8.0
0.3 9.0

in-place inverse FFT of Z:
0.1 7.0
0.2 8.0
0.3 9.0
```

示例 7-1 说明：

X 的正向 FFT 实际上是：

```
      ( 0.6, 0.0)  (24.0, 0.0)
Z =   (-0.2, 0.1)  (-1.5, 0.9)
      (-0.2, 0.1)  (-1.5, 0.9)
```

由于对称性，Z(2) 是 Z(1) 的共轭复数，因此仅存储前两个 $\frac{NI}{2}+1=2$ 复数值。对于场内正变换，将使用实数数组 X 作为输入和输出来调用 SFFTCM。因为 SFFTCM 要求输出数组的类型为 COMPLEX，所以作为输出数组的 X 的主维必须就像 X 是复数那样。由于实数数组 X 的主维是 LDX = 2 × LDC，因此作为复数输出数组的 X 的主维必须是 LDC。同样，

对于场内逆变换，将使用复数数组 Z 作为输入和输出来调用 CFFTSM。因为 CFFTSM 要求输出数组的类型为 REAL，所以作为输出数组的 Z 的主维必须就像 Z 是实数那样。由于复数数组 Z 的主维是 LDZ，因此作为实数输出数组的 Z 的主维必须是 LDZ × 2。

以下示例例 11 “线性复数到复数 FFT”说明了如何计算一组序列的线性复数到复数 FFT。

例 11 线性复数到复数 FFT

```
my_system% cat testccm.f
       PROGRAM TESTCCM
       IMPLICIT NONE
       INTEGER :: LDX1, LDY1, LW, IERR, I, J, K, LDZ1, NCPUS,
      $           USING_THREADS, IFAC(128)
       INTEGER, PARAMETER :: N1 = 3, N2 = 4, LDX1 = N1, LDZ1 = N1,
      $                      LDY1 = N1+2
       REAL, PARAMETER :: ONE = 1.0, SCALE = ONE/N1
       COMPLEX :: Z(0:LDZ1-1,0:N2-1), X(0:LDX1-1,0:N2-1),
      $           Y(0:LDY1-1,0:N2-1)

       REAL :: TRIGS(2*N1)
       REAL, DIMENSION(:), ALLOCATABLE :: SW
* get number of threads
       NCPUS = USING_THREADS()
* workspace size
       LW = 2 * N1 * NCPUS
       WRITE(*,*)'Linear complex-to-complex FFT of one or more sequences'
       WRITE(*,*)
       ALLOCATE(SW(LW))
       X = RESHAPE(SOURCE =(/(.1,.2),(.3,.4),(.5,.6),(.7,.8),(.9,1.0),
      $ (1.1,1.2),(1.3,1.4),(1.5,1.6),(1.7,1.8),(1.9,2.0),(2.1,2.2),
      $ (1.2,2.0)/), SHAPE=(/LDX1,N2/))
       Z = X
       WRITE(*,*) 'X = '
       DO I = 0, N1-1
          WRITE(*,'(5(A1, F5.1,A1,F5.1,A1,2X))') ('(',REAL(X(I,J)),
      $           ',',AIMAG(X(I,J)),')', J = 0, N2-1)
       END DO
       WRITE(*,*)* intialize trig table and compute factors of N1
       CALL CFFTCM(0, N1, N2, SCALE, X, LDX1, Y, LDY1, TRIGS, IFAC,
      $            SW, LW, IERR)
       IF (IERR .NE. 0) THEN
         PRINT*,'ROUTINE RETURN WITH ERROR CODE = ', IERR
         STOP
       END IF
* Compute out-of-place forward linear FFT.
* Let FFT routine allocate memory.
       CALL CFFTCM(-1, N1, N2, ONE, X, LDX1, Y, LDY1, TRIGS, IFAC,
      $            SW, 0, IERR)
       IF (IERR .NE. 0) THEN
          PRINT*,'ROUTINE RETURN WITH ERROR CODE = ', IERR
          STOP
```

```
       END IF
* Compute in-place forward linear FFT. LDZ1 must equal LDX1
       CALL CFFTCM(-1, N1, N2, ONE, Z, LDX1, Z, LDZ1, TRIGS,
      $            IFAC, SW, 0, IERR)
       WRITE(*,*) 'in-place forward FFT of X:'
       DO I = 0, N1-1
          WRITE(*,'(5(A1, F5.1,A1,F5.1,A1,2X))') ('(',REAL(Z(I,J)),
      $           ',',AIMAG(Z(I,J)),')', J = 0, N2-1)
       END DO

       WRITE(*,*)
       WRITE(*,*) 'out-of-place forward FFT of X:'
       WRITE(*,*) 'Y ='
       DO I = 0, N1-1
          WRITE(*,'(5(A1, F5.1,A1,F5.1,A1,2X))') ('(',REAL(Y(I,J)),
      $        ',',AIMAG(Y(I,J)),')', J = 0, N2-1)
       END DO
       WRITE(*,*)
* Compute in-place inverse linear FFT.
       CALL CFFTCM(1, N1, N2, SCALE, Y, LDY1, Y, LDY1, TRIGS, IFAC,
      $            SW, LW, IERR)
       IF (IERR .NE. 0) THEN
          PRINT*,'ROUTINE RETURN WITH ERROR CODE = ', IERR
          STOP
       END IF
       WRITE(*,*) 'in-place inverse FFT of Y:'
       WRITE(*,*) 'Y ='
       DO I = 0, N1-1
          WRITE(*,'(5(A1, F5.1,A1,F5.1,A1,2X))') ('(',REAL(Y(I,J)),
      $           ',',AIMAG(Y(I,J)),')', J = 0, N2-1)
       END DO
       DEALLOCATE(SW)
       END PROGRAM TESTCCM
my_system% f95 -dalign testccm.f -library=sunperf
my_system% a.out
Linear complex-to-complex FFT of one or more sequences
X =
( 0.1, 0.2) ( 0.7, 0.8) ( 1.3, 1.4) ( 1.9, 2.0)
( 0.3, 0.4) ( 0.9, 1.0) ( 1.5, 1.6) ( 2.1, 2.2)
( 0.5, 0.6) ( 1.1, 1.2) ( 1.7, 1.8) ( 1.2, 2.0)
in-place forward FFT of X:
(  0.9,  1.2) (  2.7,  3.0) (  4.5,  4.8) ( 5.2,  6.2)
( -0.5, -0.1) ( -0.5, -0.1) ( -0.5, -0.1) ( 0.4, -0.9)
( -0.1, -0.5) ( -0.1, -0.5) ( -0.1, -0.5) ( 0.1,  0.7)
out-of-place forward FFT of X:
Y =
(  0.9,  1.2) (  2.7,  3.0) (  4.5,  4.8) ( 5.2,  6.2)
( -0.5, -0.1) ( -0.5, -0.1) ( -0.5, -0.1) ( 0.4, -0.9)
( -0.1, -0.5) ( -0.1, -0.5) ( -0.1, -0.5) ( 0.1,  0.7)
in-place inverse FFT of Y:
Y =
( 0.1, 0.2) ( 0.7, 0.8) ( 1.3, 1.4) ( 1.9, 2.0)
( 0.3, 0.4) ( 0.9, 1.0) ( 1.5, 1.6) ( 2.1, 2.2)
( 0.5, 0.6) ( 1.1, 1.2) ( 1.7, 1.8) ( 1.2, 2.0)
```

二维 FFT 例程

对于线性 FFT 例程，如果输入是一个二维数组，则将仅沿着一个维计算 FFT，也就是沿着数组的列。二维 FFT 例程将二维数组作为输入，并同时沿着列和行维计算 FFT。具体来说，正向二维 FFT 例程计算如下：

$$X(k,n)=\sum_{l=0}^{N2-1}\sum_{j=0}^{N1-1}x(j,l)e^{\frac{-2\pi iln}{N2}}e^{\frac{-2\pi ijk}{N1}},\quad k=0,\ldots,N1-1,n=0,\ldots,N2-1$$

逆向二维 FFT 例程计算如下：

$$x(j,l)=\sum_{n=0}^{N2-1}\sum_{k=0}^{N1-1}X(k,n)e^{\frac{2\pi iln}{N2}}e^{\frac{2\pi ijk}{N1}},\quad j=0,\ldots,N1-1,l=0,\ldots,N2-1$$

对于正向和逆向二维变换，输入问题是 *N*1 × *N*2 的复数到复数变换产生的复数数组也是 *N*1 × *N*2。

在计算实数到复数二维变换（正向 FFT）时，如果实数输入数组的维是 *N*1 × *N*2，则结果将是一个复数数组，且维数是 $(\frac{N1}{2}+1)\times N2$。

相反，在计算维是 N1 × N2 的复数到实数变换（逆向 FFT）时，需要一个 $(\frac{N1}{2}+1)\times N2$ 复数数组作为输入。与实数到复数及复数到实数线性 FFT 一样，由于共轭对称性，只有前 $\frac{N1}{2}+1$ 个复数数据点需要沿着第一个维存储在输入或输出数组中。复数子数组 $X(\frac{N1}{2}+1:N1-1,:)$ 可通过 $X(0:\frac{N1}{2},:)$ 如下所示：

$$\begin{aligned}X(k,n) &= X*(N1-k,n),\\ k &= \frac{N1}{2}+1,\ldots,N1-1\\ n &= 0,\ldots,N2-1\end{aligned}$$

要计算二维变换，必须调用两次 FFT 例程。第一次调用对例程进行初始化，第二次调用实际对变换进行计算。初始化包括计算 *N*1 和 *N*2 的因子以及与这些因子关联的三角权重。在后续的正向或逆向变换中，只要 *N*1 和 *N*2 保持不变，就不必进行初始化了。

重要信息：从二维 FFT 例程返回时，*Y*(0 : *N* - 1, :) 将包含变换结果，并且 *Y*(*N* : LDY-1, :) 的原始内容将被覆盖。此处，在复数到复数和复数到实数变换中，*N* = *N*1；在实数到复数变换中，*N* = $\frac{N1}{2}+1$ 。

下面的表 13 “单精度二维 FFT 例程”列出了单精度二维 FFT 例程及其用途。这些信息同样适用于相应的双精度例程，只是其数据类型是双精度和双精度复数。有关映射，请参见表 13 “单精度二维 FFT 例程”。有关例程及其参数的完整说明，请参阅各手册页。

表 13　单精度二维 FFT 例程

名称	用途	大小，输入类型	大小，输出类型	主维要求	
SFFTC2	OPT = 0 初始化				
	OPT = -1 实数到复数正向二维 FFT	N1 × N2，实数	$(\frac{N1}{2}+1)\times N2$ 复数	LDX1 = 2 × LDY1 ，LDY1 ≥ $\frac{N1}{2}+1$	LDX1 ≥ *N*1 LDY1 ≥ $\frac{N1}{2}+1$
CFFTS2	OPT = 0 初始化				
	OPT = 1 复数到实数逆向二维 FFT	$(\frac{N1}{2}+1)\times N2$ 复数	*N*1 × *N*2，实数	LDX1 ≥ $\frac{N1}{2}+1$ ，LDY1=2 × LDX1	LDX1 ≥ $\frac{N1}{2}+1$ LDY1≥ 2 × LDX1 LDY1 是偶数
CFFTC2	OPT = 0 初始化				
	OPT = -1 复数到复数正向二维 FFT	*N*1 × *N*2，复数	*N*1 × *N*2，复数	LDX1 ≥ *N*1 LDY1 = LDX1	LDX1 ≥ *N*1 LDY1 ≥ *N*1
	OPT = 1 复数到复数逆向二维 FFT	*N*1 × *N*2，复数	*N*1 × *N*2，复数	LDX1 ≥ *N*1 LDY1 = LDX1	LDX1 ≥ *N*1 LDY1 = LDX1

注 - 请注意以下有关表 13 “单精度二维 FFT 例程”的信息

- LDX1 是输入数组的主维。
- LDY1 是输出数组的主维。
- N1 是 FFT 问题的第一个维。
- N2 是 FFT 问题的第二个维。
- 使用 OPT = 0 调用例程以初始化例程时，唯一进行的错误检查是确定是否 *N*1、*N*2 < 0。

例 12　　二维数组的二维实数到复数 FFT 和复数到实数 FFT

以下示例说明了如何计算二维数组的二维实数到复数 FFT 和复数到实数 FFT。

```
my_system% cat testsc2.f
       PROGRAM TESTSC2
       IMPLICIT NONE
       INTEGER, PARAMETER :: N1 = 3, N2 = 4, LDX1 = N1,
      $         LDY1 = N1/2+1, LDR1 = 2*(N1/2+1)
       INTEGER LW, IERR, I, J, K, IFAC(128*2)
       REAL, PARAMETER :: ONE = 1.0, SCALE = ONE/(N1*N2)
       REAL :: V(LDR1,N2), X(LDX1, N2), Z(LDR1,N2),
      $        SW(2*N2), TRIGS(2*(N1+N2))
       COMPLEX :: Y(LDY1,N2)
       WRITE(*,*) $'Two-dimensional complex-to-real and real-to-complex FFT'
       WRITE(*,*)
       X = RESHAPE(SOURCE = (/.1, .2, .3, .4, .5, .6, .7, .8,
      $            2.0,1.0, 1.1, 1.2/), SHAPE=(/LDX1,N2/))
       DO I = 1, N2
          V(1:N1,I) = X(1:N1,I)
       END DO
       WRITE(*,*) 'X ='
       DO I = 1, N1
          WRITE(*,'(5(F5.1,2X))') (X(I,J), J = 1, N2)
       END DO
       WRITE(*,*)
* Initialize trig table and get factors of N1, N2
       CALL SFFTC2(0,N1,N2,ONE,X,LDX1,Y,LDY1,TRIGS,
      $            IFAC,SW,0,IERR)
* Compute 2-dimensional out-of-place forward FFT.
* Let FFT routine allocate memory.
* cannot do an in-place transform in X because LDX1 < 2*(N1/2+1)
       CALL SFFTC2(-1,N1,N2,ONE,X,LDX1,Y,LDY1,TRIGS,
      $            IFAC,SW,0,IERR)
       WRITE(*,*) 'out-of-place forward FFT of X:'
       WRITE(*,*)'Y ='
       DO I = 1, N1/2+1
          WRITE(*,'(5(A1, F5.1,A1,F5.1,A1,2X))')('(',REAL(Y(I,J)),
      $           ',',AIMAG(Y(I,J)),')', J = 1, N2)
       END DO
       WRITE(*,*)

* Compute 2-dimensional in-place forward FFT.
* Use workspace already allocated.
* V which is real array containing input data is also
* used to store complex results; as a complex array, its first
* leading dimension is LDR1/2.
       CALL SFFTC2(-1,N1,N2,ONE,V,LDR1,V,LDR1/2,TRIGS,
      $            IFAC,SW,LW,IERR)
       WRITE(*,*) 'in-place forward FFT of X:'
       CALL PRINT_REAL_AS_COMPLEX(N1/2+1, N2, 1, V, LDR1/2, N2)
* Compute 2-dimensional out-of-place inverse FFT.
* Leading dimension of Z must be even
       CALL CFFTS2(1,N1,N2,SCALE,Y,LDY1,Z,LDR1,TRIGS,
```

```
      $           IFAC,SW,0,IERR)
       WRITE(*,*) 'out-of-place inverse FFT of Y:'
       DO I = 1, N1
          WRITE(*,'(5(F5.1,2X))') (Z(I,J), J = 1, N2)
       END DO
       WRITE(*,*)
* Compute 2-dimensional in-place inverse FFT.
* Y which is complex array containing input data is also
* used to store real results; as a real array, its first
* leading dimension is 2*LDY1.
       CALL CFFTS2(1,N1,N2,SCALE,Y,LDY1,Y,2*LDY1,
      $            TRIGS,IFAC,SW,0,IERR)
       WRITE(*,*) 'in-place inverse FFT of Y:'
       CALL PRINT_COMPLEX_AS_REAL(N1, N2, 1, Y, 2*LDY1, N2)
       END PROGRAM TESTSC2
       SUBROUTINE PRINT_COMPLEX_AS_REAL(N1, N2, N3, A, LD1, LD2)
       INTEGER N1, N2, N3, I, J, K
       REAL A(LD1, LD2, *)
       DO K = 1, N3
          DO I = 1, N1
             WRITE(*,'(5(F5.1,2X))') (A(I,J,K), J = 1, N2)
          END DO
          WRITE(*,*)
       END DO
       END

       SUBROUTINE PRINT_REAL_AS_COMPLEX(N1, N2, N3, A, LD1, LD2)
       INTEGER N1, N2, N3, I, J, K
       COMPLEX A(LD1, LD2, *)
       DO K = 1, N3
          DO I = 1, N1
             WRITE(*,'(5(A1, F5.1,A1,F5.1,A1,2X))') ('(',REAL(A(I,J,K)),
      $              ',',AIMAG(A(I,J,K)),')', J = 1, N2)
          END DO
          WRITE(*,*)
       END DO
       END
my_system% f95 -dalign testsc2.f -library=sunperf
my_system% a.out
Two-dimensional complex-to-real and real-to-complex FFT
x =
0.1 0.4 0.7 1.0
0.2 0.5 0.8 1.1
0.3 0.6 2.0 1.2
out-of-place forward FFT of X:
Y =
(  8.9, 0.0) ( -2.9,  1.8) ( -0.7, 0.0) ( -2.9, -1.8)
( -1.2, 1.3) (  0.5, -1.0) ( -0.5, 1.0) (  0.5, -1.0)
in-place forward FFT of X:
( 8.9, 0.0) ( -2.9, 1.8) ( -0.7, 0.0) ( -2.9, -1.8)
( -1.2, 1.3) ( 0.5, -1.0) ( -0.5, 1.0) ( 0.5, -1.0)
out-of-place inverse FFT of Y:
0.1 0.4 0.7 1.0
0.2 0.5 0.8 1.1
```

```
0.3 0.6 2.0 1.2
in-place inverse FFT of Y:
0.1 0.4 0.7 1.0
0.2 0.5 0.8 1.1
0.3 0.6 2.0 1.2
```

三维 FFT 例程

Oracle Developer Studio 性能库包括了用于计算三维 FFT 的例程。在这种情况下，将沿着三维数组的所有三个维计算 FFT。正向 FFT 计算

$$X(k,n,m)=\sum_{h=0}^{N3-1}\sum_{l=0}^{N2-1}\sum_{j=0}^{N1-1}x(j,l,h)\ e^{\frac{-2\pi ihm}{N3}}\ e^{\frac{-2\pi iln}{N2}}\ e^{\frac{-2\pi ijk}{N1}},$$
$$k = 0, \ldots, N1-1$$
$$n = 0, \ldots, N2-1$$
$$m = 0, \ldots, N3-1$$

逆向 FFT 计算

$$x(j,l,h)=\sum_{m=0}^{N3-1}\sum_{n=0}^{N2-1}\sum_{k=0}^{N1-1}X(k,n,m)\ e^{\frac{2\pi ihm}{N3}}\ e^{\frac{2\pi iln}{N2}}\ e^{\frac{2\pi ijk}{N1}},$$
$$j = 0, \ldots, N1-1$$
$$l = 0, \ldots, N2-1$$
$$h = 0, \ldots, N3-1$$

在复数到复数变换中，如果输入问题是 *N*1 × *N*2 × *N*3，则三维变换产生的复数数组也是 *N*1 × *N*2 × *N*3。在计算实数到复数三维变换时，如果实数输入数组的维数是 *N*1 × *N*2 × *N*3，则结果将是一个复数数组，且维数是 $(\frac{N1}{2}+1) \times N2 \times N3$ 相反，在计算维数是 *N*1 × *N*2 × *N*3 的复数到实数 FFT 时，$(\frac{N1}{2}+1) \times N2 \times N3$ 复数数组作为输入。与实数到复数及复数到实数线性 FFT 一样，由于共轭对称性，只有前 $\frac{N1}{2}+1$ 个复数数据点需要沿着第一个维存储。复数子数组 $X(\frac{N1}{2}+1:N1-1, :, :)$ 可通过 $X(0:\frac{N1}{2}+1, :, :)$ 如下所示：

$$X(k,n,m) = X*(N1-k,n,m),$$
$$k = \frac{N1}{2}+1, \ldots, N1-1$$
$$n = 0, \ldots, N2-1$$
$$m = 0, \ldots, N3-1$$

要计算三维变换，必须调用两次 FFT 例程：第一次是进行初始化，第二次是实际计算变换。初始化包括计算 *N*1、*N*2 和 *N*3 的因子以及与这些因子关联的三角权重。在后续的正向或逆向变换中，只要 *N*1、*N*2 和 *N*3 保持不变，就不必进行初始化了。

重要信息：从三维 FFT 例程返回时，*Y*(0 : *N* - 1, :, :) 将包含变换结果，并且 *Y*(*N*:LDY1-1, :, :) 的原始内容将被覆盖。此处，在复数到复数和复数到实数变换中，*N* = *N*1；在实数到复数变换中，*N* = $\frac{N1}{2}+1$ 。

表 14 “单精度三维 FFT 例程”列出了单精度三维 FFT 例程及其用途。这些信息同样适用于相应的双精度例程，只是其数据类型是双精度和双精度复数。有关映射，请参见表 14 “单精度三维 FFT 例程”。有关例程及其参数的完整说明，请参见各手册页。

表 14　单精度三维 FFT 例程

名称	用途	大小，输入类型	大小，输出类型	主维要求	
SFFTC3	OPT = 0 初始化				
	OPT = -1 实数到复数正向三维 FFT	*N*1 × *N*2 × *N*3，实数	$(\frac{N1}{2}+1) \times N2 \times N3$ 复数	LDX1=2 × LDY1 LDX2 ≥ *N*2 LDY1 ≥ $\frac{N1}{2}+1$ LDY2 = LDX2	LDX1 ≥ *N*1 LDX2 ≥ *N*2 LDY1 ≥ $\frac{N1}{2}+1$ LDY2 ≥ *N*2
CFFTS3	OPT = 0 初始化				
	OPT = 1 复数到实数逆向三维 FFT	$(\frac{N1}{2}+1) \times N2 \times N3$ 复数	*N*1 × *N*2 × *N*3，实数	LDX1 ≥ $\frac{N1}{2}+1$ LDX2 ≥ *N*2 LDY1=2 × LDX1 LDY2=LDX2	LDX1 ≥ $\frac{N1}{2}+1$ LDX2 ≥ *N*2 LDY1 ≥ 2 × LDX1 LDY1 是偶数 LDY2 ≥ *N*2
CFFTC3	OPT = 0 初始化				
	OPT = -1 复数到复数正向三维 FFT	*N*1 × *N*2 × *N*3，复数	*N*1 × *N*2 × *N*3，复数	LDX1 ≥ *N*1 LDX2 ≥ *N*2 LDY1=LDX1	LDX1 ≥ *N*1 LDX2 ≥ *N*2 LDY1 ≥ *N*1

名称	用途	大小，输入类型	大小，输出类型	主维要求	
				LDY2=LDX2	LDY2 ≥ *N*2
	OPT = 1 复数到复数逆向三维 FFT	*N*1 × *N*2 × *N*3，复数	*N*1 × *N*2 × *N*3，复数	LDX1 ≥ *N*1	LDX1 ≥ *N*1
				LDX2 ≥ *N*2	LDX2 ≥ *N*2
				LDY1=LDX1	LDY1 ≥ *N*1
				LDY2=LDX2	LDY2 ≥ *N*2

注 - 请注意以下有关表 14 “单精度三维 FFT 例程”的信息：

- LDX1 是输入数组的第一个主维。
- LDX2 是输入数组的第二个主维。
- LDY1 是输出数组的第一个主维。
- LDY2 是输出数组的第二个主维。
- N1 是 FFT 问题的第一个维。
- N2 是 FFT 问题的第二个维。
- N3 是 FFT 问题的第三个维。
- 使用 OPT = 0 调用例程以初始化例程时，唯一进行的错误检查是确定是否 *N*1、*N*2、*N*3 < 0。

示例 7-4 说明了如何计算三维数组的三维实数到复数 FFT 和复数到实数 FFT。

例 13 三维数组的三维实数到复数 FFT 和复数到实数 FFT

```
my_system% cat testsc3.f
       PROGRAM TESTSC3
       IMPLICIT NONE
       INTEGER LW, NCPUS, IERR, I, J, K, USING_THREADS, IFAC(128*3)
       INTEGER, PARAMETER :: N1 = 3, N2 = 4, N3 = 2, LDX1 = N1,
      $                      LDX2 = N2, LDY1 = N1/2+1, LDY2 = N2,
      $                      LDR1 = 2*(N1/2+1), LDR2 = N2
       REAL, PARAMETER :: ONE = 1.0, SCALE = ONE/(N1*N2*N3)

       REAL :: V(LDR1,LDR2,N3), X(LDX1,LDX2,N3), Z(LDR1,LDR2,N3),
      $        TRIGS(2*(N1+N2+N3))
       REAL, DIMENSION(:), ALLOCATABLE :: SW
       COMPLEX :: Y(LDY1,LDY2,N3)
       WRITE(*,*)
      $'Three-dimensional complex-to-real and real-to-complex FFT'
       WRITE(*,*)
* get number of threads
       NCPUS = USING_THREADS()
* compute workspace size required
       LW = (MAX(MAX(N1,2*N2),2*N3) + 16*N3) * NCPUS
```

```
       ALLOCATE(SW(LW))
       X = RESHAPE(SOURCE =
      $    (/ .1, .2, .3, .4, .5, .6, .7, .8, .9,1.0,1.1,1.2,
      $     4.1,1.2,2.3,3.4,6.5,1.6,2.7,4.8,7.9,1.0,3.1,2.2/),
      $     SHAPE=(/LDX1,LDX2,N3/))
       V = RESHAPE(SOURCE =
      $    (/.1,.2,.3,0.,.4,.5,.6,0.,.7,.8,.9,0.,1.0,1.1,1.2,0.,
      $     4.1,1.2,2.3,0.,3.4,6.5,1.6,0.,2.7,4.8,7.9,0.,
      $     1.0,3.1,2.2,0./), SHAPE=(/LDR1,LDR2,N3/))
       WRITE(*,*) 'X ='
       DO K = 1, N3
          DO I = 1, N1
             WRITE(*,'(5(F5.1,2X))') (X(I,J,K), J = 1, N2)
          END DO
          WRITE(*,*)
       END DO
* Initialize trig table and get factors of N1, N2 and N3
       CALL SFFTC3(0,N1,N2,N3,ONE,X,LDX1,LDX2,Y,LDY1,LDY2,TRIGS,
      $            IFAC,SW,0,IERR)
* Compute 3-dimensional out-of-place forward FFT.
* Let FFT routine allocate memory.
* cannot do an in-place transform because LDX1 < 2*(N1/2+1)
       CALL SFFTC3(-1,N1,N2,N3,ONE,X,LDX1,LDX2,Y,LDY1,LDY2,TRIGS,
      $            IFAC,SW,0,IERR)
       WRITE(*,*) 'out-of-place forward FFT of X:'
       WRITE(*,*)'Y ='
       DO K = 1, N3
          DO I = 1, N1/2+1
             WRITE(*,'(5(A1, F5.1,A1,F5.1,A1,2X))')('(',REAL(Y(I,J,K)),
      $              ',',AIMAG(Y(I,J,K)),')', J = 1, N2)
          END DO
          WRITE(*,*)
       END DO

* Compute 3-dimensional in-place forward FFT.
* Use workspace already allocated.
* V which is real array containing input data is also
* used to store complex results; as a complex array, its first
* leading dimension is LDR1/2.
       CALL SFFTC3(-1,N1,N2,N3,ONE,V,LDR1,LDR2,V,LDR1/2,LDR2,TRIGS,
      $            IFAC,SW,LW,IERR)
       WRITE(*,*) 'in-place forward FFT of X:'
       CALL PRINT_REAL_AS_COMPLEX(N1/2+1, N2, N3, V, LDR1/2, LDR2)
* Compute 3-dimensional out-of-place inverse FFT.
* First leading dimension of Z (LDR1) must be even
       CALL CFFTS3(1,N1,N2,N3,SCALE,Y,LDY1,LDY2,Z,LDR1,LDR2,TRIGS,
      $            IFAC,SW,0,IERR)
       WRITE(*,*) 'out-of-place inverse FFT of Y:'
       DO K = 1, N3
         DO I = 1, N1
            WRITE(*,'(5(F5.1,2X))') (Z(I,J,K), J = 1, N2)
         END DO
         WRITE(*,*)
       END DO
```

```
* Compute 3-dimensional in-place inverse FFT.
* Y which is complex array containing input data is also
* used to store real results; as a real array, its first
* leading dimension is 2*LDY1.
       CALL CFFTS3(1,N1,N2,N3,SCALE,Y,LDY1,LDY2,Y,2*LDY1,LDY2,
      $            TRIGS,IFAC,SW,LW,IERR)
       WRITE(*,*) 'in-place inverse FFT of Y:'
       CALL PRINT_COMPLEX_AS_REAL(N1, N2, N3, Y, 2*LDY1, LDY2)
       DEALLOCATE(SW)
       END PROGRAM TESTSC3
       SUBROUTINE PRINT_COMPLEX_AS_REAL(N1, N2, N3, A, LD1, LD2)
       INTEGER N1, N2, N3, I, J, K
       REAL A(LD1, LD2, *)
       DO K = 1, N3
          DO I = 1, N1
             WRITE(*,'(5(F5.1,2X))') (A(I,J,K), J = 1, N2)
          END DO
          WRITE(*,*)
       END DO
       END
       SUBROUTINE PRINT_REAL_AS_COMPLEX(N1, N2, N3, A, LD1, LD2)
       INTEGER N1, N2, N3, I, J, K
       COMPLEX A(LD1, LD2), *)
       DO K = 1, N3
          DO I = 1, N1
             WRITE(*,'(5(A1, F5.1,A1,F5.1,A1,2X))') ('(',REAL(A(I,J,K)),
      $              ',',AIMAG(A(I,J,K)),')', J = 1, N2)
          END DO
          WRITE(*,*)
       END DO
       END
my_system% f95 -dalign testsc3.f -xlibrary=sunperf
my_system% a.out
Three-dimensional complex-to-real and real-to-complex FFT
X =
0.1 0.4 0.7 1.0
0.2 0.5 0.8 1.1
0.3 0.6 0.9 1.2
4.1 3.4 2.7 1.0
1.2 6.5 4.8 3.1
2.3 1.6 7.9 2.2
out-of-place forward FFT of X:
Y =
( 48.6, 0.0) ( -9.6, -3.4) ( 3.4, 0.0) ( -9.6, 3.4)
( -4.2, -1.0) ( 2.5, -2.7) ( 1.0, 8.7) ( 9.5, -0.7)
(-33.0, 0.0) (  6.0, 7.0) ( -7.0,  0.0) (  6.0, -7.0)
(  3.0, 1.7) ( -2.5, 2.7) ( -1.0, -8.7) ( -9.5,  0.7)
in-place forward FFT of X:
( 48.6, 0.0) ( -9.6, -3.4) ( 3.4, 0.0) ( -9.6, 3.4)
( -4.2, -1.0) ( 2.5, -2.7) ( 1.0, 8.7) ( 9.5, -0.7)
(-33.0, 0.0) (  6.0, 7.0) ( -7.0, 0.0) (  6.0, -7.0)
(  3.0, 1.7) ( -2.5, 2.7) ( -1.0, -8.7) ( -9.5, 0.7)
out-of-place inverse FFT of Y:
0.1 0.4 0.7 1.0
```

```
0.2 0.5 0.8 1.1
0.3 0.6 0.9 1.2
4.1 3.4 2.7 1.0
1.2 6.5 4.8 3.1
2.3 1.6 7.9 2.2
in-place inverse FFT of Y:
0.1 0.4 0.7 1.0
0.2 0.5 0.8 1.1
0.3 0.6 0.9 1.2
4.1 3.4 2.7 1.0
1.2 6.5 4.8 3.1
2.3 1.6 7.9 2.2
```

注释

执行场内实数到复数或复数到实数变换时必须谨慎，应确保输入数组足够大以容纳结果。例如，如果输入的类型是存储在复数数组中的复数且第一个主维是 *N*，要使用相同的数组存储实数结果，则作为实数输出数组，其第一个主维将是 2 × *N*。相反，如果输入的类型是存储在实数数组中的实数且第一个主维是 2 × *N*，要使用相同的数组存储复数结果，则作为复数输出数组，其第一个主维将是 *N*。可在表 12 “单精度线性 FFT 例程”、表 13 “单精度二维 FFT 例程” 和表 14 “单精度三维 FFT 例程”中找到场内和场外变换的维要求。

在线性和多维 FFT 中，通过实数到复数或复数到实数变换在实数与复数数据之间进行的变换可能会让人困惑，因为 *N*1 个实数数据点对应于 $\frac{N1}{2}+1$ 个复数数据点。*N*1 个实数数据点映射到 *N*1 个复数数据点，但由于复数数据中有共轭对称性，因此只有 $\frac{N1}{2}+1$ 个数据点需要作为复数到实数变换中的输入进行存储，以及作为实数到复数变换中的输出进行存储。在多维 FFT 中，对称性存在于所有维中，而不仅仅在第一个维中。然而，二维和三维 FFT 例程将存储第二个和第三个维的全部复数数据。

虽然 FFT 例程接受任意大小的 *N*1、*N*2 和 *N*3，但如果将 *N*1、*N*2 和 *N*3 的值分解为相对较小的质数，则可以最有效地计算 FFT。如果满足以下条件，则可以最有效地计算实数到复数或复数到实数变换

$$N1, N2, N3 = 2^p \times 3^q \times 4^r \times 5^s$$

，

在以下情况下，可以最有效地计算复数到复数变换：

$$N1, N2, N3 = 2^p \times 3^q \times 4^r \times 5^s \times 7^t \times 11^u \times 13^v$$

，

其中 *p*、*q*、*r*、*s*、*t*、*u* 和 *v* 是整数，且 *p*、*q*、*r*、*s*、*t*、*u*、*v* ≥ 0。

可使用函数 *x*FFTOPT 确定最佳序列长度，如例 14 “ RFFTOPT 示例” 所示。给定输入序列长度后，该函数将返回大小最接近原始长度的最佳长度。

例 14 RFFTOPT 示例

```
my_system% cat fft_ex01.f
      PROGRAM TEST
      INTEGER         N, N1, N2, N3, RFFTOPT
C
      N = 1024
      N1 = 1019
      N2 = 71
      N3 = 49
C
      PRINT *, 'N Original  N Suggested'
      PRINT '(I5, I12)', (N, RFFTOPT(N))
      PRINT '(I5, I12', (N1, RFFTOPT(N1))
      PRINT '(I5, I12)', (N2, RFFTOPT(N2))
      PRINT '(I5, I12)', (N3, RFFTOPT(N3))
      END

my_system% f95 -dalign fft_ex01.f -library=sunperf
my_system% a.out
 N Original  N Suggested
 1024        1024
 1019        1024
   71          72
   49          49
```

余弦和正弦变换

对具有特殊对称性的 DFT 进行输入会发生在各种应用程序中。利用对称性进行的变换通常能节省存储空间并减少计算次数，就像实数到复数和复数到实数 FFT 变换那样。性能库余弦和正弦变换是 FFT 例程的特殊情况，它利用了偶函数和奇函数中的对称特性。

注 - Oracle Developer Studio 性能库正弦和余弦变换例程以 FFTPACK (http://www.netlib.org/fftpack/) 中包含的例程为基础。具有 V 前缀的例程是根据 VFFTPACK (http://www.netlib.org/vfftpack/) 中包含的例程进行了向量化处理的例程。

快速余弦和正弦变换例程

以下各表列出了 Oracle Developer Studio 性能库快速余弦和正弦变换。双精度例程名称位于方括号中。名称以 'V' 开头的例程可同时计算一个或多个序列的变换。名称以 'I' 结尾的例程是初始化例程。

表 15 偶序列例程的快速余弦变换及其参数

例程名称	参数
COST [DCOST]	(LEN+1, X, WORK)
COSTI [DCOSTI]	(LEN+1, WORK)
VCOST [VDCOST]	(M, LEN+1, X, WORK, LD, TABLE)
VCOSTI [VDCOSTI]	(LEN+1, TABLE)

表 16 四分之一波长偶序列例程的快速余弦变换及其参数

例程名称	参数
COSQF [DCOSQF]	(LEN, X, WORK)
COSQB [DCOSQB]	(LEN, X, WORK)
COSQI [DCOSQI]	(LEN, WORK)
VCOSQF [VDCOSQF]	(M, LEN, X, WORK, LD, TABLE)
VCOSQB [VDCOSQB]	(M, LEN, X, WORK, LD, TABLE)
VCOSQI [VDCOSQI]	(LEN, TABLE)

表 17 奇序列例程的快速正弦变换及其参数

例程名称	参数
SINT [DSINT]	(LEN-1, X, WORK)
SINTI [DSINTI]	(LEN-1, WORK)
VSINT [VDSINT]	(M, LEN-1, X, WORK, LD, TABLE)
VSINTI [VDSINTI]	(LEN-1, TABLE)

表 18 四分之一波长奇序列例程的快速正弦变换及其参数

例程名称	参数
SINQF [DSINQF]	(LEN, X, WORK)
SINQB [DSINQB]	(LEN, X, WORK)
SINQI [DSINQI]	(LEN, WORK)
VSINQF [VDSINQF]	(M, LEN, X, WORK, LD, TABLE)
VSINQB [VDSINQB]	(M, LEN, X, WORK, LD, TABLE)

例程名称	参数
VSINQI [VDSINQI]	(LEN, TABLE) 注：

注 - 请注意以下有关上述表格的信息：

- M：要变换的序列数。
- LEN、LEN-1、LEN+1：输入序列的长度。
- X：包含要变换的序列的实数数组。在输出时，实数变换结果将存储在 X 中。
- TABLE：变换例程所需的特定于变换大小的常数数组。常数是由初始化例程计算得出的。
- WORK：变换例程所需的工作区。在对单一序列进行运算的例程中，WORK 还包含由初始化例程计算得出的常数。

快速正弦变换

经常遇到的另一种类型的对称性是奇对称，其中对于 $n = -N+1, \ldots, 0, \ldots, N$，$x(n) = -x(-n)$。与快速余弦变换相同，快速正弦变换 (Fast Sine Transform, FST) 利用奇对称来节省存储空间并减少计算次数。对于实数奇序列 x，对称意味着 $x(0) = -x(0) = 0$。因此，如果 x 的长度为 $2N$，那么只需要 x 的 $N = 1$ 个值来计算 FST。例程 SINT 可计算单一实数奇序列的 FST，而 VSINT 可计算一个或多个序列的 FST。在调用 [V]SINT 之前，必须调用 [V]SINTI 来计算与输入长度 N-1 相关的三角常数和因子。FST 是其自身的逆变换。调用 VSINT 两次将会生成原始的 N -1 个数据点。调用 SINT 两次将会生成原始的 N -1 个数据点乘以 $2N$。

对称的奇序列（如 $x(n) = -x(-n - 1)$，其中 $-N+1, \ldots, 0, \ldots, N$）被认为具有四分之一波长奇对称性。SINQF 和 SINQB 分别计算单一实数四分之一波长奇序列的 FST 及其逆变换，而 VSINQF 和 VSINQB 可对一个或多个序列进行运算。SINQB 未经过归一化处理，因此使用 SINQF 的结果作为 SINQB 中的输入会生成按 $4N$ 的因子缩放的原始序列。不过，VSINQB 经过了归一化处理，因此调用 VSINQF 之后再调用 VSINQB 将会生成原始序列。长度为 $2N$ 且具有四分之一波长奇对称的实数序列的 FST 需要 N 个输入数据点，并生成一个 N 点结果序列。在调用变换例程之前，需要通过调用 [V]SINQI 来进行初始化。

快速余弦变换

对实数偶序列进行运算的一种特殊形式的 FFT 是快速余弦变换 (Fast Cosine Transform, FCT)。如果 $x(n) = x(-n)$，其中 $n = -N + 1, \ldots, 0, \ldots, N$，则认为实数序列 x 具有偶对称性。长度为 $2N$ 的序列的 FCT 需要 $N + 1$ 个输入数据点，并生成大小为 $N + 1$ 的序列。例程 COST 可计算单一实数偶序列的 FCT，而 VCOST 可计算一个或多个序列的 FCT。

在调用 [V]COST 之前，必须调用 [V]COSTI 来计算与输入长度 *N* + 1 相关的三角常数和因子。FCT 是其自身的逆变换。调用 VCOST 两次将会生成原始的 *N* +1 个数据点。调用 COST 两次将会生成原始的 *N* +1 个数据点乘以 2*N*。

对称的偶序列 *x*（如 *x*(*n*) = *x*(-*n* - 1)，其中 *n* = -*N* + 1, ... , 0, ..., *N*）被认为具有四分之一波长偶对称性。COSQF 和 COSQB 分别计算单一实数四分之一波长偶序列的 FCT 及其逆变换。VCOSQF 和 VCOSQB 可对一个或多个序列进行运算。[V]COSQB 的结果未经过归一化处理，如果按 $\frac{1}{4N}$ 缩放，则会获得原始序列。长度为 2*N* 且具有四分之一波长偶对称的实数序列的 FCT 需要 *N* 个输入数据点，并生成一个 *N* 点结果序列。在调用变换例程之前，需要通过调用 [V]COSQI 来进行初始化。

离散快速余弦和正弦变换及其逆变换

Oracle Developer Studio 性能库例程使用以下部分中的方程来计算快速余弦和正弦变换以及逆变换。

[D]COST：序列的正向和逆向快速余弦变换 (FCT)

序列的正向和逆向 FCT 计算如下：

$$X(k) = x(0)+2\sum_{n=1}^{N-1} x(n)\cos\left(\frac{\pi nk}{N}\right)+x(N)\cos(\pi k),\quad k = 0,\ldots,N$$

[D]COST 注：

- 需要 *N* + 1 个值来计算 *N* 点序列的 FCT。
- [D]COST 还可以用来计算逆变换。如果调用两次 [D]COST，结果将是缩放了 $\frac{1}{2N}$ 倍的原始序列。

V[D]COST：多个序列的正向和逆向快速余弦变换 (VFCT)

多个序列的正向和逆向 FCT 计算如下：

对于 *i* = 0，*M* - 1

$$X(i,k) = \frac{x(i,0)}{2N}+\frac{1}{N}\sum_{n=1}^{N-1} x(i,n)\cos\left(\frac{\pi nk}{N}\right)+x\frac{(i,N)}{2N}\cos(\pi k),\quad k = 0,\ldots,N$$

V[D]COST 注：

- 需要 *M* × (*N*+1) 个值来计算 *M N* 点序列的 VFCT。
- 输入和输出序列将逐行存储。
- V[D]COST 已经过归一化处理，并且是其自身的逆变换。如果调用两次 V[D]COST，结果将是原始数据。

[D]COSQF：四分之一波长偶序列的正向 FCT

四分之一波长偶序列的正向 FCT 计算如下：

$$X(k) = x(0)+2\sum_{n=1}^{N-1} x(n)\cos\left(\frac{\pi(2k+1)}{2N}\right), \quad k = 0,\ldots,N-1$$

需要 *N* 个值来计算 *N* 点四分之一波长偶序列的 FCT。

[D]COSQB：四分之一波长偶序列的逆向 FCT

四分之一波长偶序列的逆向 FCT 计算如下

$$x(n)=\sum_{k=0}^{N-1} X(k)\cos\left(\frac{\pi n(2k+1)}{2N}\right), \quad n=0, \ldots, N-1$$

.

调用正向和逆向例程将会生成缩放了 $\frac{1}{4N}$ 倍的原始输入。

V[D]COSQF：一个或多个四分之一波长偶序列的正向 FCT

一个或多个四分之一波长偶序列的正向 FCT 计算如下

对于 *i* = 0，*M* - 1

$$X(i,k) = \frac{1}{N}\left[x(i,0)+2\sum_{n=1}^{N-1} x(i,n)\cos\left(\frac{\pi n(2k+1)}{2N}\right)\right], \quad k = 0,\ldots,N-1$$

V[D]COSQF 注：

- 输入和输出序列将逐行存储。

- 变换已进行了归一化处理，因此，如果在调用 V[D]COSQF 之后立即调用逆向例程 V[D]COSQB，则会获得原始数据。

V[D]COSQB：一个或多个四分之一波长偶序列的逆向 FCT

一个或多个四分之一波长偶序列的逆向 FCT 计算如下

对于 *i* = 0，*M* - 1

$$x(i,n)=\sum_{k=0}^{N-1} X(i,k)\cos\left(\frac{\pi n(2k+1)}{2N}\right),\quad n=0,\ \ldots,N-1$$

V[D]COSQB 注：

- 输入和输出序列将逐行存储。
- 变换已进行了归一化处理，因此，如果在调用 V[D]COSQF 之后立即调用 V[D]COSQB，则会获得原始数据。

[D]SINT：序列的正向和逆向快速正弦变换 (FST)

序列的正向和逆向 FST 计算如下

$$X(k)=2\sum_{n=0}^{N-2} x(n)\sin\left(\frac{\pi(n+1)(k+1)}{N}\right),\quad k=0,\ \ldots,N-2$$

[D]SINT 注：

- 需要 *N*-1 个值来计算 *N* 点序列的 FST。
- [D]SINT 还可用来计算逆变换。如果调用两次 [D]SINT，结果将是缩放了 $\frac{1}{2N}$ 倍的原始序列。

V[D]SINT：多个序列的正向和逆向快速正弦变换 (VFST)

多个序列的正向和逆向快速正弦变换计算如下

对于 *i* = 0，*M* - 1

$$X(i,k)=\frac{2}{\sqrt{2N}}\sum_{n=0}^{N-2} x(i,n)\sin\left(\frac{\pi(n+1)(k+1)}{N}\right),\quad k=0,\ \ldots,N-2$$

V[D]SINT 注：

- 需要 *M* × (*N* - 1) 个值来计算 *M N* 点序列的 VFST。
- 输入和输出序列将逐行存储。
- V[D]SINT 已经过归一化处理，并且是其自身的逆变换。调用 V[D]SINT 两次将生成原始数据。

[D]SINQF：四分之一波长奇序列的正向 FST

四分之一波长奇序列的正向 FST 计算如下

$$X(k)=2\sum_{n=0}^{N-2} x(n)\sin\left(\frac{\pi(n+1)(2k+1)}{2N}\right)+x(N-1)\cos(\pi k),\quad k=0,\ \ldots,N-1$$

需要 *N* 个值来计算 *N* 点四分之一波长奇序列的 FST。

[D]SINQB：四分之一波长奇序列的逆向 FST

四分之一波长奇序列的逆向 FST 计算如下

$$x(n)=2\sum_{k=0}^{N-1} X(k)\sin\left(\frac{\pi(n+1)(2k+1)}{2N}\right),\quad n=0,\ \ldots,N-1$$

调用正向和逆向例程将会生成缩放了 $\frac{1}{4N}$ 倍的原始输入。

V[D]SINQF：一个或多个四分之一波长奇序列的正向 FST

一个或多个四分之一波长奇序列的正向 FST 计算如下

对于 *i* = 0，*M* - 1

$$X(i,k)=\frac{1}{\sqrt{4N}}\left[2\sum_{n=0}^{N-2} x(n,i)\sin\left(\frac{\pi(n+1)(2k+1)}{2N}\right)+x(N-1,i)\cos\pi k\right],\quad k\ =\ 0,\ \ldots,N-1$$

V[D]SINQF 注：

- 输入和输出序列将逐行存储。

- 变换已进行了归一化处理，因此，如果在调用 V[D]SINQF 之后立即调用逆向例程 V[D]SINQB，则会获得原始数据。

V[D]SINQB：一个或多个四分之一波长奇序列的逆向 FST

一个或多个四分之一波长奇序列的逆向 FST 计算如下

对于 *i* = 0，*M* - 1

$$x(n,i)=\frac{4}{\sqrt{4\mathrm{N}}}\sum_{k=0}^{N-1}X(k,i)\sin\left(\frac{\pi(n+1)(2\mathrm{k}+1)}{2\mathrm{N}}\right),\quad n\ =\ 0,\ \ldots,N-1$$

V[D]SINQB 注：

- 输入和输出序列将逐行存储。
- 变换已进行了归一化处理，因此，如果在调用 V[D]SINQF 之后立即调用 V[D]SINQB，则会获得原始数据。

快速余弦变换示例

例 15 “计算单一实数偶序列的 FCT 和逆向 FCT”调用 COST 来计算实数偶序列的 FCT 和逆变换。如果实数序列的长度为 2*N*，则只需要存储 *N* + 1 个输入数据点，所产生的数据点的数量也是 *N* + 1。结果存储在输入数组中。

例 15　　计算单一实数偶序列的 FCT 和逆向 FCT

```
my_system% cat cost.f
       program Drive cost
       implicit none
       integer,parameter :: len=4
       real x(0:len),work(3*(len+1)+15), z(0:len), scale
       integer i
       scale = 1.0/(2.0*len)
       call RANDOM_NUMBER(x(0:len))
       z(0:len) = x(0:len)
         write(*,'(a25,i1,a10,i1,a12)')'Input sequence of length ',
     $         len,' requires ', len+1,' data points'
       write(*,'(5(f8.3,2x),/)')(x(i),i=0,len)
       call costi(len+1, work)
       call cost(len+1, z, work)
       write(*,*)'Forward fast cosine transform'
       write(*,'(5(f8.3,2x),/)')(z(i),i=0,len)
       call cost(len+1, z, work)
       write(*,*)
     $    'Inverse fast cosine transform (results scaled by 1/2*N)'
```

```
       write(*,'(5(f8.3,2x),/)')(z(i)*scale,i=0,len)
       end
my_system% f95 -dalign cost.f -library=sunperf
my_system% a.out
Input sequence of length 4 requires 5 data points
0.557 0.603 0.210 0.352 0.867
Forward fast cosine transform
3.753 0.046 1.004 -0.666 -0.066
Inverse fast cosine transform (results scaled by 1/2*N)
0.557 0.603 0.210 0.352 0.867
```

例 16 “计算两个实数四分之一波长偶序列的 FCT 和逆向 FCT”调用 VCOSQF 和 VCOSQB 来分别计算两个实数四分之一波长偶序列的 FCT 和逆向 FCT。如果实数序列的长度为 2*N*，则只需要存储 *N* 个输入数据点，所产生的数据点的数量也是 *N*。结果存储在输入数组中。

例 16 计算两个实数四分之一波长偶序列的 FCT 和逆向 FCT

```
my_system% cat vcosq.f
    program vcosq
    implicit none
    integer,parameter :: len=4, m = 2, ld = m+1
    real x(ld,len),xt(ld,len),work(3*len+15), z(ld,len)
    integer i, j
    call RANDOM_NUMBER(x)
    z = x
    write(*,'(a27,i1)')' Input sequences of length ',len
    do j = 1,m
       write(*,'(a3,i1,a4,4(f5.3,2x),a1,/)')
 $     'seq',j,' = (',(x(j,i),i=1,len),')'
    end do
    call vcosqi(len, work)
    call vcosqf(m,len, z, xt, ld, work)
    write(*,*)
 $ 'Forward fast cosine transform for quarter-wave even sequences'
    do j = 1,m
       write(*,'(a3,i1,a4,4(f5.3,2x),a1,/)')
 $     'seq',j,' = (',(z(j,i),i=1,len),')'
    end do
    call vcosqb(m,len, z, xt, ld, work)
    write(*,*)
 $  'Inverse fast cosine transform for quarter-wave even sequences'

    write(*,*)'(results are normalized)'
    do j = 1,m
       write(*,'(a3,i1,a4,4(f5.3,2x),a1,/)')
 $     'seq',j,' = (',(z(j,i),i=1,len),')'
    end do
    end

my_system% f95 -dalign vcosq.f -library=sunperf
my_system% a.out
```

```
Input sequences of length 4
seq1 = (0.557 0.352 0.990 0.539 )
seq2 = (0.603 0.867 0.417 0.156 )
Forward fast cosine transform for quarter-wave even sequences
seq1 = (0.755 -.392 -.029 0.224 )
seq2 = (0.729 0.097 -.091 -.132 )
Inverse fast cosine transform for quarter-wave even sequences
(results are normalized)
seq1 = (0.557 0.352 0.990 0.539 )
seq2 = (0.603 0.867 0.417 0.156 )
```

快速正弦变换示例

在以下示例例 17 “计算两个实数四分之一波长偶序列的 FCT 和逆向 FCT”中，将调用 `SINT` 来计算实数奇序列的 FST 和逆变换。如果实数序列的长度为 2*N*，则只需要存储 *N* - 1 个输入数据点，所产生的数据点的数量也是 *N* - 1。结果存储在输入数组中。

例 17　　计算两个实数四分之一波长偶序列的 FCT 和逆向 FCT

```
my_system% cat sint.f
      program Drive sint
      implicit none
      integer,parameter :: len=4
      real x(0:len-2),work(3*(len-1)+15), z(0:len-2), scale
      integer i
      call RANDOM_NUMBER(x(0:len-2))
      z(0:len-2) = x(0:len-2)
      scale = 1.0/(2.0*len)
      write(*,'(a25,i1,a10,i1,a12)')'Input sequence of length ',
     $       len,' requires ', len-1,' data points'
      write(*,'(3(f8.3,2x),/)')(x(i),i=0,len-2)
      call sinti(len-1, work)
      call sint(len-1, z, work)
      write(*,*)'Forward fast sine transform'
      write(*,'(3(f8.3,2x),/)')(z(i),i=0,len-2)

      call sint(len-1, z, work)
      write(*,*)
     $ 'Inverse fast sine transform (results scaled by 1/2*N)'
      write(*,'(3(f8.3,2x),/)')(z(i)*scale,i=0,len-2)
      end
my_system% f95 -dalign sint.f -library=sunperf
my_system% a.out
Input sequence of length 4 requires 3 data points
0.557 0.603 0.210
Forward fast sine transform
2.291 0.694 -0.122
Inverse fast sine transform (results scaled by 1/2*N)
0.557 0.603 0.210
```

在以下示例例 18 “计算两个实数四分之一波长奇序列的 FST 和逆向 FST”中，将调用 VSINQF 和 VSINQB 来分别计算两个实数四分之一波长奇序列的 FST 和逆向 FST。如果实数序列的长度为 2*N*，则只需要存储 *N* 个输入数据点，所产生的数据点的数量也是 *N*。结果存储在输入数组中。

例 18　　计算两个实数四分之一波长奇序列的 FST 和逆向 FST

```
my_system% cat vsinq.f
      program vsinq
      implicit none
      integer,parameter :: len=4, m = 2, ld = m+1
      real x(ld,len),xt(ld,len),work(3*len+15), z(ld,len)
      integer i, j
      call RANDOM_NUMBER(x)
      z = x
      write(*,'(a27,i1)')' Input sequences of length ',len
      do j = 1,m
         write(*,'(a3,i1,a4,4(f5.3,2x),a1,/)')
     $         'seq',j,' = (',(x(j,i),i=1,len),')'
      end do
      call vsinqi(len, work)
      call vsinqf(m,len, z, xt, ld, work)
      write(*,*)
    $ 'Forward fast sine transform for quarter-wave odd sequences'
      do j = 1,m
      write(*,'(a3,i1,a4,4(f5.3,2x),a1,/)')
    $       'seq',j,' = (',(z(j,i),i=1,len),')'
      end do

      call vsinqb(m,len, z, xt, ld, work)
      write(*,*)
    $ 'Inverse fast sine transform for quarter-wave odd sequences'
      write(*,*)'(results are normalized)'
      do j = 1,m
      write(*,'(a3,i1,a4,4(f5.3,2x),a1,/)')
    $       'seq',j,' = (',(z(j,i),i=1,len),')'
      end do
      end
my_system% f95 vsinq.f -library=sunperf
my_system% a.out
Input sequences of length 4
seq1 = (0.557 0.352 0.990 0.539 )
seq2 = (0.603 0.867 0.417 0.156 )
Forward fast sine transform for quarter-wave odd sequences
seq1 = (0.823 0.057 0.078 0.305 )
seq2 = (0.654 0.466 -.069 -.037 )
Inverse fast sine transform for quarter-wave odd sequences
(results are normalized)
seq1 = (0.557 0.352 0.990 0.539 )
seq2 = (0.603 0.867 0.417 0.156 )
```

卷积和相关

经常遇到的 FFT 的两个应用（尤其是在信号处理区域中）是离散卷积和离散相关运算。

卷积运算

假设有两个函数 *x*(*t*) 和 *y*(*t*)，*x*(*t*) 和 *y*(*t*) 的卷积的傅里叶变换（表示为 x * y）是各个傅里叶变换的积：DFT (x * y)=*X ( ) Y*，其中 * 表示卷积运算，(·) 表示逐点乘法。

通常，*x*(*t*) 是由一组 N 个数据点 x_j, *j* = 0, …, *N* -1 以离散方式表示的连续和周期性信号，这些数据点是在有限持续时间内取样的，通常是以等间隔在 *x*(*t*) 的一个周期内取样。*y*(*t*) 通常是以零开始的响应，峰值达到最大值，然后返回到零。在相等间隔对 *y*(*t*) 进行离散化处理将生成一组 *N* 个数据点，y_k, *k* = 0, …, *N* -1。如果 y_k 中的实际取样数量小于 *N*，则使用零填充数据。然后可将离散卷积定义为

$$(x*y)_j \equiv \sum_{k=\frac{-N}{2}+1}^{\frac{N}{2}} x_{j-k}\,y_k,\ j=0,\ldots,N-1$$

$$y_k, k = -\frac{N}{2}+1,\ldots,\frac{N}{2}$$

的值与 *k* = 0, …, *N* -1 的值相同，但使用环绕顺序。

Oracle Developer Studio 性能库例程允许使用上述定义通过 *k* = 0, …, *N* -1 或使用 FFT 来计算卷积。如果使用 FFT 计算两个序列的卷积，请执行以下步骤：

- 计算 *X* = x 的正向 FFT
- 计算 *Y* = y 的正向 FFT
- 计算 *Z* = *X ( ) Y* <=> *DFT (x * y)*
- 计算 *z* = *Z* 的逆向 FFT；*z* = (*x* * *y*)

卷积的一个有趣的特性是两个多项式的积实际上是一个卷积。*m* 项多项式

$$a(x) = a_0 + a_1x + ... + a_{m-1}x^{m-1}$$

和 *n* 项多项式

$$b(x) = b_0 + b_1x + ... + b_{n-1}x^{n-1}$$

的积具有 *m*+*n*-1 系数，可通过以下方程获得该系数：

$$c_k = \sum_{j=\max((k-(m-1)),0)}^{\min(k,n-1)} a_j b_{k-j}$$

其中 *k* = 0, …, *m* + *n* - 2。

相关运算

与卷积密切相关的是相关运算。它计算直接叠加的两个序列的相关，或一个序列相对于另一个序列发生移位时的相关。与卷积相同，可以使用 FFT 有效地计算两个序列的相关，如下所述：

- 计算两个输入序列的 FFT。
- 计算一个序列所产生的变换和另一个序列的变换的共轭复数的逐点积。
- 计算积的逆向 FFT。

性能库中的例程还允许按以下定义计算相关：

$$\mathrm{Corr}(x,y)_j \equiv \sum_{k=0}^{N-1} x_{j+k} y_k, \quad j = 0,\ldots,N-1$$

有许多方法可以解释卷积和相关运算的取样输入数据。卷积和相关例程的参数列表包含可处理以下情况的参数：

- 信号和/或响应函数可能从不同取样时间开始。
- 您可能只需要部分信号来生成输出。
- 信号和/或响应函数可能从未显式存储的一个或多个零开始。

Oracle Developer Studio 性能库卷积和相关例程

Oracle Developer Studio 性能库包含如表 19 “卷积和相关例程”所示的卷积例程。

表 19　卷积和相关例程

例程	参数	功能
SCNVCOR、DCNVCOR、CCNVCOR、ZCNVCOR	NX,X,IFX, INCX,NY, NPRE,M,Y,IFY, INC1Y,INC2Y,NZ,K, Z, IFZ,INC1Z,INC2Z,WORK, LWORK	包含一个或多个向量的过滤器的卷积或相关
SCNVCOR2、DCNVCOR2、CCNVCOR2、ZCNVCOR2	TRANSX, SCRATCHX, TRANSY, SCRATCHY,MX,NX,X,LDX, MY,NY,MPRE,NPRE,Y,LDY, MZ,NZ,Z, LDZ,WORKIN, LWORK	两个矩阵的二维卷积或相关

例程	参数	功能
SWIENER、DWIENER	N_POINTS,ACOR,XCOR, FLTR,EROP, ISW,IERR	两个信号的维纳解卷积

[S,D,C,Z]CNVCOR 例程用于计算包含一个或多个输入向量的过滤器的卷积或相关。[S,D,C,Z]CNVCOR2 例程用于计算两个矩阵的二维卷积或相关。

卷积和相关例程的参数

一维卷积和相关例程使用表 20 “一维卷积和相关例程 SCNVCOR、DCNVCOR、CCNVCOR 和 ZCNVCOR 的参数”中显示的参数。

表 20 一维卷积和相关例程 SCNVCOR、DCNVCOR、CCNVCOR 和 ZCNVCOR 的参数

参数	定义
CNVCOR	'V' 或 'v' 指定要计算卷积。'R' 或 'r' 指定要计算相关。
FOUR	'T' 或 't' 指定要使用傅里叶变换方法。'D' 或 'd' 指定要使用直接方法，即根据卷积和相关的定义计算卷积或相关。†
NX	过滤器向量的长度，其中 NX≥ 0。
X	过滤器向量
IFX	X 的第一个元素的索引，其中 NX ≥ IFX ≥ 1
INCX	X 中向量的元素之间的跨距，其中 INCX > 0。
NY	输入向量的长度，其中 NY≥ 0。
NPRE	放在 Y 向量前面的隐式零的数量，其中 NPRE ≥ 0。
M	输入向量的数量，其中 M≥ 0。
Y	输入向量。
IFY	Y 的第一个元素的索引，其中 NY ≥ IFY ≥ 1
INC1Y	Y 中输入向量的元素之间的跨距，其中 INC1Y > 0。
INC2Y	Y 中输入向量之间的跨距，其中 INC2Y > 0。
NZ	输出向量的长度，其中 NZ≥ 0。
K	Z 向量的数量，其中 K ≥ 0。如果 K < M，则仅处理前 K 个向量。如果 K > M，则处理所有输入向量，并在退出时将最后 M-K 个输出向量设置为零。
Z	结果向量
IFZ	Z 的第一个元素的索引，其中 NZ ≥ IFZ ≥ 1
INC1Z	Z 中输出向量的元素之间的跨距，其中 INCYZ > 0。
INC2Z	Z 中输出向量之间的跨距，其中 INC2Z > 0。
WORK	工作数组
LWORK	工作数组的长度。

†如果要进行卷积的两个序列的长度相同，FFT 方法要比直接方法快。但是，如果一个序列比另一个大得多，例如，在将大时序信号与小过滤器进行卷积时，直接方法的执行速度比基于 FFT 的方法快。

二维卷积和相关例程使用表 21 “二维卷积和相关例程 SCNVCOR2、DCNVCOR2、CCNVCOR2 和 ZCNVCOR2 的参数”中显示的参数。

表 21 二维卷积和相关例程 SCNVCOR2、DCNVCOR2、CCNVCOR2 和 ZCNVCOR2 的参数

参数	定义
CNVCOR	'V' 或 'v' 指定要计算卷积。'R' 或 'r' 指定要计算相关。
METHOD	'T' 或 't' 指定要使用傅里叶变换方法。'D' 或 'd' 指定要使用直接方法，即根据卷积和相关的定义计算卷积或相关。†
TRANSX	'N' 或 'n' 指定 X 是过滤器矩阵，'T' 或 't' 指定 X 的转置是过滤器矩阵
SCRATCHX	'N' 或 'n' 指定必须保留 X，'S' 或 's' 指定可将 X 用于暂存空间。在从将 X 用于暂存空间的调用返回后，X 的内容为未定义。
TRANSY	'N' 或 'n' 指定 Y 是输入矩阵，'T' 或 't' 指定 Y 的转置是输入矩阵
SCRATCHY	'N' 或 'n' 指定必须保留 Y，'S' 或 's' 指定可将 Y 用于暂存空间。在从调用（其中 X 用于暂存空间）返回后，Y 的内容为未定义。
MX	过滤器矩阵 X 的行数，其中 MX ≥ 0
NX	过滤器矩阵 X 的列数，其中 NX ≥ 0
X	过滤器矩阵。当 SCRATCHX 是 'N' 或 'n' 时，X 在退出时保持不变；当 SCRATCHX 是 'S' 或 's' 时，它在退出时将为未定义。
LDX	包含过滤器矩阵 X 的数组的主维。
MY	输入矩阵 Y 的行数，其中 MY ≥ 0。
NY	输入矩阵 Y 的列数，其中 NY ≥ 0
MPRE	放在输入矩阵 Y 向量每行前面的隐式零的数量，其中 MPRE ≥ 0。
NPRE	放在输入矩阵 Y 每列前面的隐式零的数量，其中 NPRE ≥ 0。
Y	输入矩阵。当 SCRATCHY 是 'N' 或 'n' 时，Y 在退出时保持不变；当 SCRATCHY 是 'S' 或 's' 时，它在退出时将为未定义。
LDY	包含输入矩阵 Y 的数组的主维。
MZ	输出向量的数量，其中 MZ ≥ 0。
NZ	输出向量的长度，其中 NZ ≥ 0。
Z	结果向量
LDZ	包含结果矩阵 Z 的数组的主维，其中 LDZ ≥ MAX(1,MZ)。
WORKIN	工作数组
LWORK	工作数组的长度。

†如果要进行卷积的两个矩阵的大小相同，FFT 方法要比直接方法快。但是，如果一个序列比另一个大得多，例如，在将大数据集与小过滤器进行卷积时，直接方法的执行速度比基于 FFT 的方法快。

卷积和相关例程的工作数组 WORK

表 24 “与卷积和相关例程一起使用的 WORK 工作数组的最小维数和数据类型”中显示了与一维和二维卷积和相关例程一起使用的 WORK 工作数组的最小维数。一维卷积和相关例程的最小维数取决于参数 NPRE、NX、NY 和 NZ 的值。

二维卷积和相关例程的最小维数取决于表 22 “影响二维例程的最小工作数组大小的参数：SCNVCOR2、DCNVCOR2、CCNVCOR2 和 ZCNVCOR2”中显示的参数的值。

表 22 影响二维例程的最小工作数组大小的参数：SCNVCOR2、DCNVCOR2、CCNVCOR2 和 ZCNVCOR2

参数	定义
MX	过滤器矩阵中的行数
MY	输入矩阵中的行数
MZ	输出向量数
NX	过滤器矩阵中的列数
NY	输入矩阵中的列数
NZ	输出向量长度
MPRE	放在输入矩阵每行前面的隐式零的数量
NPRE	放在输入矩阵每列前面的隐式零的数量
MPOST	MAX(0,MZ-MYC)
NPOST	MAX(0,NZ-NYC)
MYC	MPRE + MPOST + MYC_INIT，其中 MYC_INIT 取决于过滤器和输入矩阵，如表 23 “MYC_INIT 和 NYC_INIT 依赖项”中所示
NYC	NPRE + NPOST + NYC_INIT，其中 NYC_INIT 取决于过滤器和输入矩阵，如表 23 “MYC_INIT 和 NYC_INIT 依赖项”中所示

MYC_INIT 和 NYC_INIT 取决于以下矩阵，其中 X 是过滤器矩阵，Y 是输入矩阵。

表 23 MYC_INIT 和 NYC_INIT 依赖项

例程	Y		Transpose(Y)	
	X	Transpose(X)	X	Transpose(X)
MYC_INIT	MAX(MX,MY)	MAX(NX,MY)	MAX(MX,NY)	MAX(NX,NY)
NYC_INIT	MAX(NX,NY)	MAX(MX,NY)	MAX(NX,MY)	MAX(MX,MY)

表 24 “与卷积和相关例程一起使用的 WORK 工作数组的最小维数和数据类型”中显示了分配给最小工作数组大小的值。

表 24 与卷积和相关例程一起使用的 WORK 工作数组的最小维数和数据类型

例程	最小工作数组大小 (WORK)	类型
SCNVCOR、DCNVCOR	4*(MAX(NX,NPRE+NY) + MAX(0,NZ-NY))	REAL、REAL*8
CCNVCOR、ZCNVCOR	2*(MAX(NX,NPRE+NY) + MAX(0,NZ-NY))	COMPLEX、COMPLEX*16
SCNVCOR2†、DCNVCOR2†	MY + NY + 30	COMPLEX、COMPLEX*16
CCNVCOR2†、ZCNVCOR2†	如果 MY = NY：MYC + 8 如果 MY ≥ NY: MYC + NYC + 16	COMPLEX、COMPLEX*16

1

示例程序：卷积

以下示例使用 CCNVCOR 执行两个复数向量的 FFT 卷积。

例 19　　使用傅里叶变换方法和复数数据的一维卷积

```
my_system% cat con_ex20.f
      PROGRAM TEST
C
      INTEGER           LWORK
      INTEGER           N
      PARAMETER        (N = 3)
      PARAMETER        (LWORK = 4 * N + 15)
      COMPLEX           P1(N), P2(N), P3(2*N-1), WORK(LWORK)
      DATA P1 / 1, 2, 3 /,  P2 / 4, 5, 6 /
C
      EXTERNAL          CCNVCOR
C
      PRINT *, 'P1:'
      PRINT 1000, P1
      PRINT *, 'P2:'
      PRINT 1000, P2

      CALL CCNVCOR ('V', 'T', N, P1, 1, 1, N, 0, 1, P2, 1, 1, 1,
     $              2 * N - 1, 1, P3, 1, 1, 1, WORK, LWORK)
C
      PRINT *, 'P3:'
      PRINT 1000, P3
C
 1000 FORMAT (1X, 100(F4.1,' +',F4.1,'i  '))
C
      END
my_system% f95 -dalign con_ex20.f -xlibrary=sunperf
my_system% a.out
 P1:
  1.0 + 0.0i   2.0 + 0.0i   3.0 + 0.0i
 P2:
  4.0 + 0.0i   5.0 + 0.0i   6.0 + 0.0i
 P3:
  4.0 + 0.0i  13.0 + 0.0i  28.0 + 0.0i  27.0 + 0.0i  18.0 + 0.0i
```

如果任何向量与可写向量发生重叠（由于使用了参数别名，或者由于为各个 INC 参数选择了不恰当的值），则结果都将为未定义，并且每次运行时都不同。

最常用的计算形式以及执行最快的情况是将过滤器向量 X 应用于存储在 Y 的列中的一系列向量，并将结果放在 Z 的列中。在这种情况下，INCX = 1，INC1Y = 1，INC2Y

[1]如果由 LWORK 指定的工作区大小不够大，则在例程中分配内存。

≥ NY，INC1Z = 1，INC2Z ≥ NZ。另一种常用形式是将过滤器向量 X 应用于存储在 Y 的行中的一系列向量，并将结果存储在 Z 的行中，在这种情况下，INCX = 1，INC1Y ≥ NY，INC2Y = 1，INC1Z ≥ NZ，以及 INC2Z = 1。

可使用卷积来计算多项式的积。以下示例例 20 “使用傅里叶变换方法和实数数据的一维卷积”使用 SCNVCOR 计算 $1 + 2x + 3x^2$ 和 $4 + 5x + 6x^2$ 的积。

例 20　使用傅里叶变换方法和实数数据的一维卷积

```
my_system% cat con_ex21.f
      PROGRAM TEST
      INTEGER     LWORK, NX, NY, NZ
      PARAMETER  (NX = 3)
      PARAMETER  (NY = NX)
      PARAMETER  (NZ = 2*NY-1)
      PARAMETER  (LWORK = 4*NZ+32)
      REAL        X(NX), Y(NY), Z(NZ), WORK(LWORK)
C
      DATA X / 1, 2, 3 /,  Y / 4, 5, 6 /, WORK / LWORK*0 /
C
      PRINT 1000, 'X'
      PRINT 1010, X
      PRINT 1000, 'Y'
      PRINT 1010, Y
      CALL SCNVCOR ('V', 'T', NX, X, 1, 1,
     $NY, 0, 1, Y, 1, 1, 1,   NZ, 1, Z, 1, 1, 1, WORK, LWORK)
      PRINT 1020, 'Z'
      PRINT 1010, Z
 1000 FORMAT (1X, 'Input vector ', A1)
 1010 FORMAT (1X, 300F5.0)
 1020 FORMAT (1X, 'Output vector ', A1)
      END
my_system% f95 -dalign con_ex21.f -library=sunperf
my_system% a.out
 Input vector X
    1.   2.   3.
 Input vector Y
    4.   5.   6.
 Output vector Z
    4.  13.  28.  27.  18.
```

使输出向量比输入向量长，如上例所示，可将零隐式添加到输入末尾。实际上，零不是在任何向量中都是必需的，在示例中也未使用零，但使用隐式零进行填充会产生输入向量的舍入移位的效果，而不是循环移位效果。

以下示例例 21 “用于计算向量和循环矩阵的积的卷积”将计算向量 [1, 2, 3] 和初始列向量 [4, 5, 6] 定义的循环矩阵之间的积。

例 21　用于计算向量和循环矩阵的积的卷积

```
my_system% cat con_ex22.f
```

```
      PROGRAM TEST
C
      INTEGER     LWORK, NX, NY, NZ
      PARAMETER  (NX = 3)
      PARAMETER  (NY = NX)
      PARAMETER  (NZ = NY)
      PARAMETER  (LWORK = 4*NZ+32)
      REAL        X(NX), Y(NY), Z(NZ), WORK(LWORK)
C
      DATA X / 1, 2, 3 /,  Y / 4, 5, 6 /, WORK / LWORK*0 /
C
      PRINT 1000, 'X'
      PRINT 1010, X
      PRINT 1000, 'Y'
      PRINT 1010, Y
      CALL SCNVCOR ('V', 'T', NX, X, 1, 1,
     $NY, 0, 1, Y, 1, 1, 1,   NZ, 1, Z, 1, 1, 1,
     $WORK, LWORK)
      PRINT 1020, 'Z'
      PRINT 1010, Z
C
 1000 FORMAT (1X, 'Input vector ', A1)
 1010 FORMAT (1X, 300F5.0)
 1020 FORMAT (1X, 'Output vector ', A1)
      END
my_system% f95 -dalign con_ex22.f -library=sunperf
my_system% a.out
 Input vector X
    1.   2.   3.
 Input vector Y
    4.   5.   6.
 Output vector Z
   31.  31.  28.
```

下面的示例和前面的示例之间的区别在于，输出向量的长度与输入向量的长度相同，所以在输入向量的末尾没有隐式零。由于没有可以移位到其中的隐式零，因此不会产生前面示例中的舍入移位的效果，而循环移位将产生循环矩阵积。

例 22　　使用直接方法的二维卷积

```
my_system% cat con_ex23.f
      PROGRAM TEST
C
      INTEGER           M, N
      PARAMETER        (M = 2)
      PARAMETER        (N = 3)
C
      INTEGER           I, J
      COMPLEX           P1(M,N), P2(M,N), P3(M,N)
      DATA P1 / 1, -2, 3, -4, 5, -6 /,  P2 / -1, 2, -3, 4, -5, 6 /
      EXTERNAL          CCNVCOR2
C
      PRINT *, 'P1:'
```

```
      PRINT 1000, ((P1(I,J), J = 1, N), I = 1, M)
      PRINT *, 'P2:'
      PRINT 1000, ((P2(I,J), J = 1, N), I = 1, M)
C
      CALL CCNVCOR2 ('V', 'Direct', 'No Transpose X', 'No Overwrite X',
     $   'No Transpose Y', 'No Overwrite Y', M, N, P1, M,
     $   M, N, 0, 0, P2, M, M, N, P3, M, 0, 0)
C
      PRINT *, 'P3:'
      PRINT 1000, ((P3(I,J), J = 1, N), I = 1, M)
C
 1000 FORMAT (3(F5.1,' +',F5.1,'i  '))
C
      END
my_system% f95 -dalign con_ex23.f -library=sunperf
my_system% a.out
 P1:
  1.0 +  0.0i    3.0 +  0.0i    5.0 +  0.0i
 -2.0 +  0.0i   -4.0 +  0.0i   -6.0 +  0.0i
 P2:
 -1.0 +  0.0i   -3.0 +  0.0i   -5.0 +  0.0i
  2.0 +  0.0i    4.0 +  0.0i    6.0 +  0.0i
 P3:
-83.0 +  0.0i  -83.0 +  0.0i  -59.0 +  0.0i
 80.0 +  0.0i   80.0 +  0.0i   56.0 +  0.0i
```

参考资料

有关 DFT 或 FFT 的其他信息，请参见以下资源。

由 Briggs, William L. 和 Henson, Van Emden. 合著的《*The DFT: An Owner's Manual for the Discrete Fourier Transform*》。宾夕法尼亚州费城：SIAM，1995 年。

由 Brigham, E.Oran. 编著的《*The Fast Fourier Transform and Its Applications*》。新泽西州上鞍河镇：Prentice Hall，1988 年。

由 Chu, Eleanor 和 George, Alan 合著的《*Inside the FFT Black Box: Serial and Parallel Fast Fourier Transform Algorithms*》。佛罗里达州波卡拉顿：CRC Press，2000 年。

由 Press, William H.、Teukolsky, Saul A.、Vetterling, William T. 和 Flannery, Brian P. 合著的《*Numerical Recipes in C: The Art of Scientific Computing*》第二版。英国剑桥：Cambridge University Press，1992 年。

由 Ramirez, Robert W. 编著的《*The FFT: Fundamentals and Concepts*》。新泽西州恩格尔伍德克利夫斯：Prentice-Hall, Inc.，1985 年。

由 Swartzrauber, Paul N. 编著的《Vectorizing the FFTs》。发表于 Rodrigue, Garry 版的《*Parallel Computations*》。纽约：Academic Press, Inc.，1982 年。

由 Strang, Gilbert. 编著的《*Linear Algebra and Its Applications*》第三版。佛罗里达州奥兰多：Harcourt Brace & Company，1988 年。

由 Van Loan, Charles. 编著的《*Computational Frameworks for the Fast Fourier Transform*》。宾夕法尼亚州费城：SIAM，1992 年。

由 Walker, James S. 编著的《*Fast Fourier Transforms*》。佛罗里达州波卡拉顿：CRC Press，1991 年。

♦ ♦ ♦ 附 录 A

Oracle Developer Studio 性能库例程

本附录按库、例程名称和功能列出了 Oracle Developer Studio 性能库例程。

有关 Fortran 和 C 接口的功能描述和列表，请参阅各例程的第 3P 部分手册页。例如，要显示 SBDSQR 例程的手册页，可键入 **man -s 3P sbdsqr**。手册页例程名称使用小写字母。

对于许多例程，存在对不同数据类型进行运算的不同例程。系统没有单独列出每个例程，而是在例程名称中使用一个小写 *x* 来表示单精度、双精度、复数和双精度复数数据类型。例如，例程 *x*BDSQR 可作为四个例程使用，它们对以下数据类型进行运算：

- SBDSQR – 单精度数据类型
- DBDSQR – 双精度数据类型
- CBDSQR – 复数数据类型
- ZBDSQR – 双精度复数数据类型

如果某个例程名称不能用于 S、D、C 和 Z，则不会使用前缀 x，并将列出每个例程名称。在启用了 64 位的操作环境中还提供了对应的 64 位例程（但未列出）。其名称用 _64 后缀表示。例如，64 位版本的 *x*BDSQR 如下所示：

- SBDSQR_64
- DBDSQR_64
- CBDSQR_64
- ZBDSQR_64

LAPACK 例程

以下各表列出了 Oracle Developer Studio 性能库 LAPACK 例程。(P) 表示已进行了并行化处理的例程。

表 25　双对角矩阵例程

例程	功能
SBDSDC (P) 或 DBDSDC (P)	使用分治法计算双对角矩阵的奇异值分解 (Singular Value Decomposition, SVD)。
*X*BDSQR	使用隐式零位移 QR 算法计算实数上或下双对角矩阵的 SVD。
SLARTGS 或 DLARTGS	生成平面旋转，以在双对角 SVD 问题的隐式 QR 迭代中造成凸面。由 SBBCSD 或 DBBCSD 使用。

表 26　常见例程或计算例程

例程	功能
CHLA_TRANSTYPE	将 BLAST 指定的整数常数转换为用来指定转置运算的字符串。

例程	功能
CLA_HERPVGRW (P) 或 ZLA_HERPVGRW (P)	计算复数厄尔米特矩阵的倒数主元增长因子 norm(A)/norm(U)。
ILADIAG	将用来指定矩阵是否有单位对角的字符串转换为相关的 BLAST 指定的整数常数。
ILAPREC	将用来指定中间精度的字符串转换为相关的 BLAST 指定的整数常数。
ILATRANS	将用来指定转置运算的字符串转换为相关的 BLAST 指定的整数常数。
ILAENV	从 LAPACK 例程调用，以选择用于本地环境的与问题相关的参数。
ILAUPLO	将用来指定上或下三角矩阵的字符串转换为相关的 BLAST 指定的整数常数。
ILAVER	返回 LAPACK 版本。
*X*LA_GBRPVGRW	计算实数或复数一般带状矩阵的倒数主元增长因子 norm(A)/norm(U)。
*X*LA_GERPVGRW (P)	计算一般不定矩阵的倒数主元增长因子 norm(A)/norm(U)。
*X*LA_PORPVGRW (P)	计算实数对称矩阵或厄尔米特正定矩阵的倒数主元增长因子 norm(A)/norm(U)。
*X*LA_SYRPVGRW (P)	计算实数或复数对称不定矩阵的倒数主元增长因子 norm(A)/norm(U)。
SLAMRG (P) 或 DLAMRG (P)	创建一个排列表，将两个单独排序的集合的条目以升序顺序合并到单个集合中。
CLANHF (P) 或 ZLANHF (P)	返回 RFP 格式的厄尔米特矩阵的 1 范数、弗罗贝尼乌斯范数、无穷范数的值或绝对值最大的元素的值。
SLANSF (P) 或 DLANSF (P)	返回 RFP 格式的实数对称矩阵的 1 范数、弗罗贝尼乌斯范数、无穷范数的值或绝对值最大的元素的值。
*X*LARSCL2 (P)	对向量执行倒数对角缩放。
*X*LASCL2 (P)	对向量执行对角缩放。
SLASQ1 或 DLASQ1	计算实数方形双对角矩阵的奇异值。由 SBDSQR 或 DBDSQR 使用。
SLASQ2 或 DLASQ2	计算实数对称正定三对角矩阵的所有特征值（高相对精度）。由 SBDSQR 和 SSTEGR 或 DBDSQR 和 DSTEGR 使用。
SLASQ3 或 DLASQ3	检查是否有收缩，计算位移并调用 DQDS 算法。由 SBDSQR 或 DBDSQR 使用。
SLASQ4 或 DLASQ4	使用先前变换产生的值计算最小特征值的近似值。由 SBDSQR 或 DBDSQR 使用。
SLASQ5 或 DLASQ5	以乒乓形式计算一次 DQDS 变换。由 SBDSQR 和 SSTEGR 或 DBDSQR 和 DSTEGR 使用。
SLASQ6 或 DLASQ6	以乒乓形式计算一次 DQD 变换（位移等于零），防止发生下溢和上溢。由 SBDSQR 和 SSTEGR 或 DBDSQR 和 DSTEGR 使用。
SLASRT 或 DLASRT	以递增或递减顺序对向量中的数字排序。
*X*LATRZ (P)	通过正交变换计算实数或复数上梯形矩阵的系数。
CROT, ZROT	应用吉文斯平面旋转。注意，SROT/DROT 包括在 1 级 BLAS 中。

表 27 余弦正弦 (CS) 分解例程

例程	功能
*X*BBCSD (P)	计算双对角块形式的酉矩阵或正交矩阵的 CS 分解。
SORCSD (P) 或 DORCSD (P)	计算实数分区正交矩阵的 CS 分解。
SORCSD2BY1 或 DORCSD2BY1 (P)	计算其正交列已分区到 2x1 块结构中的 MxQ 矩阵 X 的 CS 分解。
CUNCSD (P) 或 ZUNCSD (P)	计算 MxM 已分区酉矩阵的 CS 分解。

表 28 对角矩阵例程

例程	功能
SDISNA (P) 或 DDISNA (P)	计算实数对称或复数厄尔米特矩阵的特征向量条件数的倒数。

表 29 一般带状矩阵例程

例程	功能
CGBBRD 或 ZGBBRD	通过正交变换将复数一般带状矩阵简化为上双对角形式。
SGBBRD (P) 或 DGBBRD (P)	通过正交变换将实数一般带状矩阵简化为上双对角形式。
*X*GBCON	使用 LU 因式分解估算一般带状矩阵条件数的倒数。
*X*GBEQU (P)	计算行和列比例以平衡一般带状矩阵并简化其条件数。
*X*GBEQUB (P)	计算行和列比例以平衡一般带状矩阵并简化其条件数。与 CGEEQU 不同的是，它将比例因子限制为基数的幂。
*X*GBRFS (P)	当系数矩阵为带状时，改进计算得到的线性方程组的解，并提供解的误差界和向后误差估算值。
*X*GBRFSX (P)	改进计算得到的带状线性方程组的解，并提供误差界和向后误差估计值。除了提供范数误差界以外，该代码还提供最大分量误差界（如果有）。
*X*GBSV	对一般带状线性方程组求解（简单驱动程序）。
*X*GBSVX (P)	对一般带状线性方程组求解（专业驱动程序）。
*X*GBSVXX (P)	对一般带状线性方程组求解（专业驱动程序，超精度）。如果需要，将返回范数误差界和最大分量误差界。
*X*GBTF2 (P)	使用部分主元消元法和行交换计算实数或复数一般带状矩阵的 LU 因式分解（未分块算法）。
*X*GBTRF (P)	使用部分主元消元法和行交换计算一般带状矩阵的 LU 因式分解。
*X*GBTRS	使用由 *X*GBTRF 计算得到的因式分解对一般带状线性方程组求解。
*X*LA_GBAMV	执行“矩阵-向量”运算以计算实数或复数带状矩阵的误差界。
*X*LA_GBRFSX_EXTENDED	通过执行超精确迭代细化来改进计算得到的实数或复数一般带状矩阵的线性方程组的解，并提供解的误差界和向后误差估计值。

表 30 一般矩阵（非对称或矩形）例程

例程	功能
SGEJSV (P) 或 DGEJSV (P)	计算实数一般矩阵的奇异值分解 (SVD)。
DSGESV	计算具有一般矩阵的实数线性方程组的解（将精度与迭代细化混合起来）。
SGESVJ (P) 或 DGESVJ (P)	计算实数一般矩阵的奇异值分解 (SVD)。实施预调节的雅可比 SVD 算法。使用 SGEQP3、SGEQR 和 SGELQF 或 DGEQP3、DGEQRF 和 DGELQF 作为预处理程序，这意味着准确性更高。
ZCGESV	计算具有一般矩阵的复数线性方程组的解（将精度与迭代细化混合起来）。
ZCPOSV	计算具有正定矩阵的复数线性方程组的解（将精度与迭代细化混合起来）。
*X*GEBAK	通过对 *X*GEBAL 输出的平衡矩阵的计算得到的特征向量进行向后变换来形成一般矩阵的右侧或左侧特征向量。
*X*GEBAL (P)	平衡实数或复数一般矩阵。

例程	功能
*X*GEBD2	将一般矩阵简化为双对角形式（未分块算法）。
*X*GEBRD	通过酉变换或正交变换将一般矩阵简化为上或下双对角形式（分块算法）。
*X*GECON	使用由 *X*GETRF 计算得到的因式分解估算一般矩阵条件数的倒数。
*X*GEEQU (P)	计算行和列比例以平衡一般矩形矩阵并简化其条件数。
*X*GEEQUB (P)	计算行和列比例以平衡一般矩形矩阵并简化其条件数。与 *X*GETRF 不同的是，它将比例因子限制为基数的幂。
*X*GEES	计算一般矩阵的特征值和舒尔因式分解（简单驱动程序）。
*X*GEESX	计算一般矩阵的特征值和舒尔因式分解（专业驱动程序）。
*X*GEEV (P)	计算一般矩阵的特征值和左侧及右侧特征向量（简单驱动程序）。
*X*GEEVX (P)	计算一般矩阵的特征值和左侧及右侧特征向量（专业驱动程序）。
*X*GEGS	已废弃的例程，已由 *X*GGES 替换。
*X*GEGV (P)	已废弃的例程，已由 *X*GGEV 替换。
*X*GEHD2	通过酉变换或正交相似变换将一般方形矩阵简化为上海森伯格形式（未分块算法）。
*X*GEHRD (P)	通过正交相似变换将一般矩阵简化为上海森伯格形式。
*X*GELQ2	计算实数或复数一般矩形矩阵的 LQ 因式分解（未分块算法）。
*X*GELQF	计算一般矩形矩阵的 LQ 因式分解。
*X*GELS (P)	使用 A 的 QR 或 LQ 因式分解计算超定线性方程组的最小二乘解。
*X*GELSD	使用分治法和 A 的 QR 或 LQ 因式分解计算超定线性方程组的最小二乘解。
*X*GELSS	使用一般矩形矩阵的 SVD 计算线性最小二乘问题的最小范数解（简单驱动程序）。
*X*GELSX (P)	已废弃的例程，已由 *X*SELSY 替换。
*X*GELSY (P)	使用完全正交因式分解计算线性最小二乘问题的最小范数解。
*X*GEMQRT	使用由正交矩阵进行的一般矩阵变换的结果覆盖一般矩阵，定义为使用 *X*GEQRT 返回的精简 WY 表示法生成的基本反射器的积。
*X*GEQL2	计算实数或复数一般矩形矩阵的 QL 因式分解（未分块算法）。
*X*GEQLF	计算实数或复数一般矩形矩阵的 QL 因式分解。
*X*GEQP3	使用 3 级 BLAS 计算一般矩形矩阵的 QR 因式分解。
*X*GEQPF	已废弃的例程，已由 *X*GEQP3 替换。
*X*GEQR2	计算实数或复数一般矩形矩阵的 QR 因式分解（未分块算法）。
*X*GEQR2P	使用非负对角元素计算实数或复数一般矩形矩阵的 QR 因式分解（未分块算法）。
*X*GEQRFP	计算实数或复数一般矩形矩阵的 QR 因式分解。
*X*GEQRT	使用 Q 的精简 WY 表示法计算实数或复数一般矩阵的分块 QR 因式分解。
*X*GEQRT2	使用 Q 的精简 WY 表示法计算实数或复数一般矩阵的 QR 因式分解。
*X*GEQRT3 (P)	使用 Q 的精简 WY 表示法递归计算实数或复数一般矩阵的 QR 因式分解。
*X*GERFS (P)	细化线性方程组的解。
*X*GERFSX (P)	改进计算得到的线性方程组的解，并提供解的误差界和向后误差估计值（超精度）。
*X*GERQ2	使用未分块算法计算实数或复数一般矩形矩阵的 RQ 因式分解。

例程	功能
*X*GERQF	计算实数或复数一般矩形矩阵的 RQ 因式分解。
*X*GESDD	使用分治法计算实数或复数一般矩形矩阵的奇异值分解 (SVD)（驱动程序）。
*X*GESV	对一般线性方程组求解（简单驱动程序）。
*X*GESVD	计算实数或复数一般矩阵的奇异值分解 (SVD)（驱动程序）。
SGESVJ 或 DGESVJ	计算实数一般矩形矩阵的奇异值分解 (SVD)。
*X*GESVX (P)	对一般线性方程组求解（专业驱动程序）。
*X*GESVXX (P)	计算一般矩阵的线性方程组的解（超精度）。
*X*GETF2	使用部分主元消元法和行交换计算实数或复数一般矩阵的 LU 因式分解（未分块算法）。
*X*GETRF (P)	使用部分主元消元法和行交换计算一般矩形矩阵的 LU 因式分解。
*X*GETRI	使用由 *X*GETRF 计算得到的因式分解计算一般矩阵的逆向值。
*X*GETRS	使用由 *X*GETRF 计算得到的因式分解对一般线性方程组求解。
SGSVJ0 (P) 或 DGSVJ0 (P)	SGESVJ 或 DGESVJ 的预处理程序。应用仅以特定主元为目标的雅可比旋转。
SGSVJ1 (P) 或 DGSVJ1 (P)	SGESVJ 或 DGESVJ 的预处理程序。以和 SGESVJ 或 DGESVJ 相同的方式应用雅可比旋转，但不检查收敛（停止准则）。
*X*LA_GEAMV (P)	执行“矩阵-向量”运算以计算实数或复数一般矩阵的误差界。
CLA_GERCOND_C (P) 或 ZLA_GERCOND_C (P)	计算复数一般矩阵的 op(A)*inv(diag(c)) 的无穷范数条件数。C 是一个 REAL 向量。
CLA_GERCOND_X (P) 或 ZLA_GERCOND_X (P)	计算复数一般矩阵的 op(A)*inv(diag(x)) 的无穷范数条件数。X 是一个 COMPLEX 向量。
SLA_GERCOND (P) 或 DLA_GERCOND (P)	估算实数一般矩阵的斯基尔条件数。
*X*LA_GERFSX_EXTENDED (P)	通过执行超精确迭代细化来改进计算得到的实数或复数一般矩阵的线性方程组的解，并提供解的误差界和向后误差估计值。
*X*LA_GERFSX_GBRPVGRW	计算实数或复数一般矩阵的倒数主元增长因子 norm(A)/norm(U)。
*X*LALS0 (P)	使用分治 SVD 法对最小二乘问题求解时应用后乘数。由 *X*LALSA 使用。
CLALSA (P) 或 ZLALSA (P)	以精简形式计算复数矩阵的 SVD。由 SGELSD 使用。
SLALSA 或 DLALSA	以精简形式计算实数矩阵的 SVD。由 SGELSD 或 DGELSD 使用。
*X*LALSD (P)	使用 SVD 对最小二乘问题求解。由 *X*GELSD 使用。

表 31 一般矩阵广义问题（一般矩阵对）例程

例程	功能
*X*GGBAK	根据 *X*GGBAL 的输出形成广义特征值问题的右侧或左侧特征向量。
*X*GGBAL (P)	平衡广义特征值问题的一般矩阵对。
*X*GGES	计算两个非对称矩阵的广义特征值和舒尔形式，并可选择性地计算左侧和/或右侧舒尔向量（简单驱动程序）。
*X*GGESX	计算广义特征值和舒尔形式，并可选择性地计算左侧和/或右侧舒尔向量（专业驱动程序）。
*X*GGEV (P)	计算两个非对称矩阵的广义特征值以及左侧和/或右侧广义特征向量（简单驱动程序）。

例程	功能
*X*GGEVX (P)	计算两个非对称矩阵的广义特征值以及左侧和/或右侧广义特征向量（专业驱动程序）。
*X*GGGLM (P)	对一般高斯-马尔可夫线性模型 (Gauss-Markov linear model, GLM) 问题求解。
*X*GGHRD (P)	使用正交变换将两个矩阵简化为广义上海森伯格形式。
*X*GGLSE	使用 GRQ（Generalized RQ，广义 RQ）因式分解对 LSE（Constrained Linear Least Squares，约束线性最小二乘）问题求解。
*X*GGQRF	计算两个矩阵的广义 QR 因式分解。
*X*GGRQF	计算两个矩阵的广义 RQ 因式分解。
*X*GGSVD	计算广义奇异值分解（驱动程序）。
*X*GGSVP (P)	计算正交矩阵或酉矩阵，以作为计算广义奇异值分解的预处理步骤。

表 32 一般三对角矩阵例程

例程	功能
*X*GTCON	使用由 *X*GTTRF 计算得到的 LU 因式分解估算三对角矩阵条件数的倒数。
*X*GTRFS (P)	细化一般三对角线性方程组的解。
*X*GTSV (P)	对一般三对角线性方程组求解（简单驱动程序）。
*X*GTSVX	对一般三对角线性方程组求解（专业驱动程序）。
*X*GTTRF (P)	使用部分主元消元法和行交换计算一般三对角矩阵的 LU 因式分解。
*X*GTTRS	使用由 *X*GTTRF 计算得到的因式分解对一般三对角线性方程组求解。
*X*GTTS2 (P)	使用由 *X*GTTRF 计算得到的 LU 因式分解对具有三对角矩阵的线性方程组求解。

表 33 厄尔米特带状矩阵例程

例程	功能
CHBEV 或 ZHBEV	计算厄尔米特带状矩阵的所有特征值和特征向量。建议替换为较新版本的 CHBEVD 或 ZHBEVD。
CHBEVD 或 ZHBEVD	计算厄尔米特带状矩阵的所有特征值和特征向量，并使用分治法计算特征向量（驱动程序）。
CHBEVX (P) 或 ZHBEVX (P)	计算厄尔米特带状矩阵的所选特征值和特征向量。
CHBGST (P) 或 ZHBGST (P)	将厄尔米特正定带状广义特征值问题简化为标准形式。
CHBGV 或 ZHBGV	计算广义厄尔米特正定带状特征值问题的所有特征值和特征向量。建议替换为较新版本的 CHBGVD 或 ZHBGVD。
CHBGVD 或 ZHBGVD	计算广义厄尔米特正定带状特征值问题的所有特征值和特征向量，并使用分治法计算特征向量（驱动程序）。
CHBGVX (P) 或 ZHBGVX (P)	计算广义厄尔米特正定带状特征值问题的所选特征值和特征向量。
CHBTRD (P) 或 ZHBTRD (P)	通过使用酉相似变换，将厄尔米特带状矩阵简化为实数对称三对角形式。

表 34 厄尔米特矩阵例程

例程	功能
CHECON 或 ZHECON	使用由 CHETRF 或 ZHETRF 计算得到的因式分解估算厄尔米特矩阵条件数的倒数。
CHECON_ROOK 或 ZHECON_ROOK	使用由 CHETRF_ROOK 或 ZHETRF_ROOK 计算得到的因式分解估算厄尔米特矩阵条件数的倒数。
CHEEQUB (P) 或 ZHEEQUB (P)	计算行和列比例以平衡厄尔米特矩阵，并按照 2 范数简化其条件数。
CHEEV 或 ZHEEV	计算厄尔米特矩阵的所有特征值和特征向量（简单驱动程序）。建议替换为较新版本的 CHEEVR 或 ZHEEVR。
CHEEVD 或 ZHEEVD	计算厄尔米特矩阵的所有特征值和特征向量，并使用分治法计算特征向量（驱动程序）。建议替换为较新版本的 CHEEVR 或 ZHEEVR。
CHEEVR 或 ZHEEVR	计算复数厄尔米特矩阵的所选特征值和特征向量。
CHEEVX (P) 或 ZHEEVX (P)	计算厄尔米特矩阵的所选特征值和特征向量（专业驱动程序）。
CHEGST 或 ZHEGST	使用由 CPOTRF 或 ZPOTRF 计算得到的因式分解将厄尔米特正定广义特征值问题简化为标准形式。
CHEGV 或 ZHEGV	计算复数广义厄尔米特正定特征值问题的所有特征值和特征向量。建议替换为较新版本的 CHEGVD 或 ZHEGVD。
CHEGVD 或 ZHEGVD	计算复数广义厄尔米特正定特征值问题的所有特征值和特征向量，并使用分治法计算特征向量（驱动程序）。
CHEGVX 或 ZHEGVX	计算复数广义厄尔米特正定特征值问题的所选特征值和特征向量。
CHERFS (P) 或 ZHERFS (P)	当系数矩阵是厄尔米特不定矩阵时，改进计算得到的线性方程组的解。
CHERFSX (P) 或 ZHERFSX (P)	当系数矩阵是厄尔米特不定矩阵时，改进计算得到的线性方程组的解（超精度）。
CHESV 或 ZHESV	对复数厄尔米特不定线性方程组求解（简单驱动程序）。将调用 CHETRF 来计算复数厄尔米特矩阵的因式分解。
CHESV_ROOK 或 ZHESV_ROOK	对复数厄尔米特不定线性方程组求解（简单驱动程序）。将调用 CHETRF_ROOK 来计算复数厄尔米特矩阵的因式分解。
CHESVX 或 ZHESVX	对复数厄尔米特不定线性方程组求解（专业驱动程序）。
CHESVXX (P) 或 ZHESVXX (P)	使用斜旋转分解法计算具有方形对称矩阵的复数线性方程组的解（超精度）。
CHETD2 或 ZHETD2	通过酉相似变换将复数厄尔米特矩阵简化为实数对称三对角形式（未分块算法）。
CHETF2 (P) 或 ZHETF2 (P)	使用斜旋转方法计算复数厄尔米特矩阵的因式分解（未分块算法）。
CHETF2_ROOK (P) 或 ZHETF2_ROOK (P)	使用有界 Bunch-Kaufman ("rook") 斜旋转方法计算复数厄尔米特矩阵的因式分解（未分块算法）。
CHETRD 或 ZHETRD	通过使用酉相似变换，将厄尔米特矩阵简化为实数对称三对角形式。
CHETRF (P) 或 ZHERTF (P)	使用斜旋转方法计算复数厄尔米特不定矩阵的因式分解。
CHETRF_ROOK (P) 或 ZHERTF_ROOK (P)	使用 Bunch-Kaufman ("rook") 斜旋转方法计算复数厄尔米特不定矩阵的因式分解。
CHETRI (P) 或 ZHETRI (P)	使用由 CHETRF 或 ZHETRF 计算得到的因式分解计算复数厄尔米特不定矩阵的逆向值。
CHETRI_ROOK (P) 或 ZHETRI_ROOK (P)	使用由 CHETRF_ROOK 或 ZHETRF_ROOK 计算得到的因式分解计算复数厄尔米特不定矩阵的逆向值。

例程	功能
CHETRI2 或 ZHETRI2	使用由 CHETRF 或 ZHETRS. 计算得到的因式分解计算复数厄尔米特不定矩阵的逆向值。在调用实际计算逆向值的 CHETRI2X 或 ZHETRI2X 之前设置工作区的主维（超精度）。
CHETRI2X (P) 或 ZHETRI2X (P)	使用由 CHETRF 或 ZHETRS 计算得到的因式分解计算复数厄尔米特不定矩阵的逆向值（超精度）。
CHETRS (P) 或 ZHETRS (P)	使用由 CHETRF 或 ZHETRF 计算得到的因式分解对复数厄尔米特不定矩阵求解。
CHETRS_ROOK (P) 或 ZHETRS_ROOK (P)	使用由 CHETRF_ROOK 或 ZHETRF_ROOK 计算得到的因式分解对复数厄尔米特不定矩阵求解。
CHETRS2 (P) 或 ZHETRS2 (P)	使用由 CHETRF 或 ZHERTF 计算得到的以及由 CSYCONV 或 ZSYCONV 转换得到的因式分解对具有复数厄尔米特矩阵的线性方程组求解。
CHFRK (P) 或 ZHFRK (P)	对 RFP 格式的矩阵执行厄尔米特 k 阶运算。
CLA_HEAMV 或 ZLA_HEAMV	执行“矩阵-向量”运算以计算复数厄尔米特不定矩阵的误差界。
CLA_HERCOND_C (P) 或 ZLA_HERCOND_C (P)	计算复数厄尔米特不定矩阵的 op(A)*inv(diag(c)) 的无穷范数条件数。C 是一个 REAL 向量。
CLA_HERCOND_X (P) 或 ZLA_HERCOND_X (P)	计算复数厄尔米特不定矩阵的 op(A)*inv(diag(x)) 的无穷范数条件数。X 是一个 COMPLEX 向量。
CLA_HERFSX_EXTENDED (P) 或 ZLA_HERFSX_EXTENDED (P)	通过执行超精确迭代细化来改进计算得到的复数厄尔米特不定矩阵的线性方程组的解，并提供解的误差界和向后误差估计值。
CLAHEF (P) 或 ZLAHEF (P)	使用斜旋转方法计算复数厄尔米特不定矩阵的部分因式分解。由 CHETRF 或 CHETRF 使用。
CLAHEF_ROOK (P) 或 ZLAHEF_ROOK (P)	使用 Bunch-Kaufman ("rook") 斜旋转方法计算复数厄尔米特不定矩阵的部分因式分解。由 CHETRF_ROOK 或 CHETRF_ROOK 使用。

表 35 填充存储例程中的厄尔米特矩阵

例程	功能
CHPCON 或 ZHPCON	使用由 CHPTRF 或 ZHPTRF 计算得到的因式分解估算填充存储中厄尔米特不定矩阵条件数的倒数。
CHPEV 或 ZHPEV	计算填充存储中厄尔米特矩阵的所有特征值和特征向量（简单驱动程序）。建议替换为较新版本的 CHPEVD 或 ZHPEVD。
CHPEVX (P) 或 ZHPEVX (P)	计算填充存储中厄尔米特矩阵的所选特征值和特征向量（专业驱动程序）。
CHPEVD 或 ZHPEVD	计算填充存储中厄尔米特矩阵的所有特征值和特征向量，并使用分治法计算特征向量（驱动程序）。
CHPGST 或 ZHPGST	将厄尔米特正定广义特征值问题简化为标准形式，其中系数矩阵在填充存储中，并使用由 CPPTRF 或 ZPPTRF 计算得到的因式分解。
CHPGV 或 ZHPGV	计算广义厄尔米特正定特征值问题的所有特征值和特征向量，其中系数矩阵在填充存储中（简单驱动程序）。建议替换为较新版本的 CHPGVD 或 ZHPGVD。
CHPGVD 或 ZHPGVD	计算广义厄尔米特正定特征值问题的所有特征值和特征向量，其中系数矩阵在填充存储中，并使用分治法计算特征向量（驱动程序）。
CHPGVX 或 ZHPGVX	计算复数厄尔米特正定特征值问题的所选特征值和特征向量，其中系数矩阵在填充存储中（专业驱动程序）。
CHPRFS (P) 或 ZHPRFS (P)	当系数矩阵是填充存储中的厄尔米特不定矩阵时，改进计算得到的线性方程组的解。

例程	功能
CHPSV 或 ZHPSV	计算复数线性方程组的解，其中系数矩阵是以填充格式存储的厄尔米特矩阵（简单驱动程序）。
CHPSVX 或 ZHPSVX	使用斜旋转因式分解计算复数线性方程组的解，其中系数矩阵是以填充格式存储的厄尔米特矩阵（专业驱动程序）。
CHPTRD 或 ZHPTRD	通过酉相似变换，将以填充形式存储的复数厄尔米特矩阵简化为实数对称三对角形式。
CHPTRF 或 ZHPTRF	使用 Bunch-Kaufman 斜旋转方法计算复数厄尔米特填充矩阵的因式分解。
CHPTRI 或 ZHPTRI	使用由 CHPTRF 或 ZHPTRF 计算得到的因式分解计算填充存储中复数厄尔米特不定矩阵的逆向值。
CHPTRS (P) 或 ZHPTRS (P)	使用由 CHPTRF 或 ZHPTRF 计算得到的因式分解对以填充格式存储的复数厄尔米特不定矩阵求解。

表 36 上海森伯格矩阵例程

例程	功能
*X*HSEIN (P)	使用逆迭代计算上海森伯格矩阵的指定的右侧和/或左侧特征向量。
CHSEQR or ZHSEQR	使用乘法移位 QR 算法计算复上海森伯格矩阵的特征值和舒尔因式分解。
SHSEQR (P) 或 DHSEQR (P)	使用乘法移位 QR 算法计算实上海森伯格矩阵的特征值和舒尔因式分解。

表 37 上海森伯格矩阵广义问题（海森伯格和三角矩阵）例程

例程	功能
*X*HGEQZ (P)	使用单/双移位 QZ 方法计算复数矩阵对 (H,T) 的特征值，H 是上海森伯格矩阵，T 是上三角。此类型的矩阵对是由 *X*GGHRD 生成的。

表 38 填充存储例程中的实数正交矩阵

例程	功能
SOPGTR (P) 或 DOPGTR (P)	从由 SSPTRD 或 DSPTRD 确定的实数三对角矩阵生成正交变换矩阵。
SOPMTR 或 DOPMTR	将实数一般矩阵乘以由 SSPTRD 或 DSPTRD 简化为三对角形式的正交变换矩阵。

表 39 实数正交矩阵例程

例程	功能
SORBDB 或 DORBDB	使实数分区的正交矩阵的块同时双对角化。
SORBDB1 或 DORBDB1	使具有正交列的高瘦矩阵（变体 1）的块同时双对角化。
SORBDB2 或 DORBDB2	使具有正交列的高瘦矩阵（变体 2）的块同时双对角化。
SORBDB3 或 DORBDB3	使具有正交列的高瘦矩阵（变体 3）的块同时双对角化。
SORBDB4 或 DORBDB4	使具有正交列的高瘦矩阵（变体 4）的块同时双对角化。
SORBDB5 或 DORBDB5	使列向量 X 相对于 Q 的正交列正交化。

例程	功能
SORBDB6 或 DORBDB6	使列向量 X 相对于 Q 的正交列正交化。由 SORBDB4 或 DORBDB5 使用。
SORG2L (P) 或 DORG2L (P)	从 QL 因式分解生成所有或部分实数正交矩阵 Q，如 SGEQLF 或 DGEQLF 确定的那样（未分块算法）。
SORG2R (P) 或 DORG2R (P)	从 QR 因式分解生成所有或部分实数正交矩阵 Q，如 SGEQRF 或 DGEQRF 确定的那样（未分块算法）。
SORGBR (P) 或 DORGBR	通过简化为双对角形式生成实数正交变换矩阵，如 SGEBRD 或 DGEBRD 确定的那样。
SORGHR (P) 或 DORGHR (P)	生成简化为海森伯格形式的实数正交变换矩阵，如 SGEHRD 或 DGEHRD 确定的那样。
SORGL2 (P) 或 DORGL2 (P)	生成具有正交行的实数矩形矩阵，如 SGELQF 或 DGELQF 返回的那样。
SORGLQ (P) 或 DORGLQ (P)	从 LQ 因式分解生成实数正交矩阵 Q，如 SGELQF 或 DGELQF 返回的那样。
SORGQL (P) 或 DORGQL (P)	从 QL 因式分解生成实数正交矩阵 Q，如 SGEQLF 或 DGEQLF 返回的那样。
SORGQR (P) 或 DORGQR (P)	从 QR 因式分解生成实数正交矩阵 Q，如 SGEQRF 或 DGEQRF 返回的那样。
SORGR2 (P) 或 DORGR2 (P)	从 RQ 因式分解生成所有或部分实数正交矩阵 Q，如 SGEQRF 或 DGEQRF 确定的那样（未分块算法）。
SORGRQ (P) 或 DORGRQ (P)	从 RQ 因式分解生成实数正交矩阵 Q，如 SGERQF 或 DGERQF 返回的那样。
SORGTR (P) 或 DORGTR (P)	生成由 SSYTRD 或 DSYTRD 简化为三对角形式的实数正交矩阵。
SORM2L 或 DORM2L	将实数一般矩阵乘以来自由 SGEQLF 或 DGEQLF 确定的 QL 因式分解的正交矩阵（未分块算法）。
SORM2R 或 DORM2R	将实数一般矩阵乘以来自由 SGEQRF 或 DGEQRF 确定的 QR 因式分解的正交矩阵（未分块算法）。
SORMBR 或 DORMBR	将实数一般矩阵乘以简化为双对角形式的正交矩阵，如 SGEBRD 或 DGEBRD 确定的那样。
SORMHR 或 DORMHR	将实数一般矩阵乘以由 SGEHRD 或 DGEHRD 简化为海森伯格形式的正交矩阵。
SORML2 或 DORML2	将实数一般矩阵乘以来自由 SGELQF 确定的 LQ 因式分解的正交矩阵（未分块算法）。
SORMLQ 或 DORMLQ	将实数一般矩阵乘以来自 LQ 因式分解的正交矩阵，如 SGELQF 或 DGELQF 返回的那样。
SORMQL 或 DORMQL	将实数一般矩阵乘以来自 QL 因式分解的正交矩阵，如 SGEQLF 或 DGEQLF 返回的那样。
SORMQR 或 DORMQR	将实数一般矩阵乘以来自 QR 因式分解的正交矩阵，如 SGEQRF 或 DGEQRF 返回的那样。
SORMR2 或 DORMR2	将实数一般矩阵乘以来自由 STZRZF 或 DTZRZF 确定的 RQ 因式分解的正交矩阵（未分块算法）。
SORMR3 或 DORMR3	将实数一般矩阵乘以来自由 STZRZF 或 DTZRZF 确定的 RZ 因式分解的正交矩阵（未分块算法）。
SORMRQ 或 DORMRQ	将实数一般矩阵乘以来自由 SGERQF 或 DGERQF 返回的 RQ 因式分解的正交矩阵。
SORMRZ 或 DORMRZ	将实数一般矩阵乘以来自 RZ 因式分解的正交矩阵，如 STZRZF 或 DTZRZF 返回的那样。
SORMTR 或 DORMTR	将实数一般矩阵乘以由 SSYTRD 或 DSYTRD 简化为三对角形式的正交变换矩阵。

表 40 对称或厄尔米特正定带状矩阵例程

例程	功能
*X*PBCON	使用由 *X*PBTRF 返回的丘拉斯基因式分解估算对称或厄尔米特正定带状矩阵的条件数的倒数。
*X*PBEQU (P)	计算对称或厄尔米特正定带状矩阵的平衡比例因子。
*X*PBRFS (P)	细化对称或厄尔米特正定带状线性方程组的解。
*X*PBSTF	计算实数对称正定带状矩阵的分裂丘拉斯基因式分解。
*X*PBSV	对对称或厄尔米特正定带状线性方程组求解（简单驱动程序）。
*X*PBSVX (P)	对对称或厄尔米特正定带状线性方程组求解（专业驱动程序）。
*X*PBTF2	计算实数对称或复数厄尔米特正定带状矩阵的丘拉斯基因式分解（未分块算法）。
*X*PBTRF	计算对称或厄尔米特正定带状矩阵的丘拉斯基因式分解。
*X*PBTRS	使用由 *X*PBTRF 计算得到的丘拉斯基因式分解对具有实数对称或复数厄尔米特正定带状矩阵的线性方程组求解。

表 41 对称或厄尔米特正定矩阵例程

例程	功能
CLA_PORCOND_C (P) 或 ZLA_PORCOND_C (P)	计算复数厄尔米特正定矩阵的 op(A)*inv(diag(c)) 的无穷范数条件数。C 是一个 REAL 向量。
CLA_PORCOND_X (P) 或 ZLA_PORCOND_X (P)	计算复数厄尔米特正定矩阵的 op(A)*inv(diag(x)) 的无穷范数条件数。X 是一个 COMPLEX 向量。
SLA_PORCOND (P) 或 DLA_PORCOND (P)	估算实数对称正定矩阵的斯基尔条件数。
*X*LA_LIN_BERR (P)	计算分量的相对向后误差。
*X*LA_PORFSX_EXTENDED (P)	通过执行超精确迭代细化来改进计算得到的实数对称或复数厄尔米特正定矩阵的线性方程组的解，并提供解的误差界和向后误差估计值。
*X*LA_WWADDW	将向量 W 添加到双倍单一向量 (X, Y) 中。这对所有现存的 IBM 的十六进制和二进制浮点数算术有效，但对十进制无效。
*X*PFTRF	计算实数对称或厄尔米特正定带状矩阵的丘拉斯基因式分解。
*X*PFTRI	使用由 *X*PFTRF 计算得到的丘拉斯基因式分解计算实数对称或厄尔米特正定矩阵的逆向值。
*X*PFTRS	使用由 *X*PFTRF 计算得到的丘拉斯基因式分解对具有对称或厄尔米特正定矩阵的线性方程组求解。
*X*POCON	使用由 *X*POTRF 返回的丘拉斯基因式分解估算对称或厄尔米特正定矩阵的条件数的倒数。
*X*POEQU (P)	计算对称或厄尔米特正定矩阵的平衡比例因子。
*X*POEQUB (P)	计算行和列比例以平衡对称或厄尔米特正定矩阵，并按照 2 范数简化其条件数。
*X*PORFS (P)	细化丘拉斯基分解的对称或厄尔米特正定矩阵中的线性方程组的解。
*X*PORFSX (P)	当系数矩阵是实数对称或厄尔米特正定矩阵时，改进计算得到的线性方程组的解，并提供解的误差界和向后误差估算值（超精度）。
*X*POSV	对对称或厄尔米特正定线性方程组求解（简单驱动程序）。

例程	功能
*X*POSVX (P)	对对称或厄尔米特正定线性方程组求解（专业驱动程序）。
*X*POSVXX (P)	对实数对称或厄尔米特正定线性方程组求解（专业驱动程序，超精度）。如果需要，将返回范数误差界和最大分量误差界。
*X*POTRF	计算实数对称或厄尔米特正定矩阵的丘拉斯基因式分解。
*X*POTRI	使用由 *X*POTRF 计算得到的丘拉斯基因式分解计算实数对称或厄尔米特正定矩阵的逆向值。
*X*POTRS	使用由 *X*POTRF 计算得到的丘拉斯基因式分解对实数对称或厄尔米特正定线性方程组求解。
ZCPOSV	计算具有正定矩阵的复数线性方程组的解（将精度与迭代细化混合起来）。

表 42 填充存储例程中的对称或厄尔米特正定矩阵

例程	功能
*X*PPCON	估算填充存储中丘拉斯基分解的对称正定矩阵的条件数的倒数。
*X*PPEQU (P)	计算填充存储中对称或厄尔米特正定矩阵的平衡比例因子。
*X*PPRFS (P)	细化填充存储中丘拉斯基分解的对称或厄尔米特正定矩阵中的线性方程组的解。
*X*PPSV	对填充存储中的对称或厄尔米特正定矩阵中的线性方程组求解（简单驱动程序）。
*X*PPSVX (P)	对填充存储中的对称或厄尔米特正定矩阵中的线性方程组求解（专业驱动程序）。
*X*PPTRF	计算以填充格式存储的实数对称或厄尔米特正定矩阵的丘拉斯基因式分解。
*X*PPTRI	使用由 *X*PPTRF 返回的丘拉斯基因式分解计算填充存储中实数对称或厄尔米特正定矩阵的逆向值。
*X*PPTRS	使用由 *X*PPTRF 返回的丘拉斯基因式分解对实数对称或厄尔米特正定线性方程组求解，其中系数矩阵在填充存储中。
*X*PSTF2 (P)	使用实数对称或厄尔米特正半定矩阵的全主元消元法计算丘拉斯基因式分解。此版本的算法将调用 2 级 BLAS。
*X*PSTRF (P)	使用实数对称或厄尔米特正半定矩阵的全主元消元法计算丘拉斯基因式分解。此版本的算法将调用 3 级 BLAS。

表 43 对称或厄尔米特正定三对角矩阵例程

例程	功能
*X*PTCON	使用由 *X*PTTRF 返回的丘拉斯基因式分解估算实数对称或厄尔米特正定三对角矩阵的条件数的倒数。
*X*PTEQR (P)	计算实数对称或厄尔米特正定矩阵的所有特征向量，并可选择性地计算特征值。
*X*PTRFS (P)	细化对称或厄尔米特正定三对角线性方程组的解。
*X*PTSV	对实数对称或厄尔米特正定三对角线性方程组求解（简单驱动程序）。
*X*PTSVX	对实数对称或厄尔米特正定三对角线性方程组求解（专业驱动程序）。
*X*PTTRF	计算实数对称或厄尔米特正定三对角矩阵的 LDL^H 或 LDL^T 因式分解。
*X*PTTRS	使用由 *X*PTTRF 返回的 LDL^H 或 LDL^T 因式分解对实数对称或厄尔米特正定三对角线性方程组求解。

例程	功能
*x*PTTS2 (P)	使用由 *x*PTTRF 计算的 LDL^H 或 LDL^T 因式分解对三对角线性方程组求解。由 *x*PTTRS 使用。

表 44 实数对称带状矩阵例程

例程	功能
SSBEV 或 DSBEV	计算实数对称带状矩阵的所有特征值，并可选择性地计算左侧和/或右侧特征向量（简单驱动程序）。建议替换为较新版本的 SSBEVD 或 DSBEVD。
SSBEVD 或 DSBEVD	计算实数对称带状矩阵的所有特征值，并可选择性地计算特征向量。如果需要特征向量，它将使用分治算法（驱动程序）。
SSBEVX (P) 或 DSBEVX (P)	计算对称带状矩阵的所选特征值，并可选择性地计算左侧和/或右侧特征向量（专业驱动程序）。
SSBGST (P) 或 DSBGST (P)	将对称正定带状广义特征值问题简化为标准形式。
SSBGV 或 DSBGV	计算广义对称正定带状特征值问题的所有特征值，并可选择性地计算特征向量（简单驱动程序）。建议替换为较新版本的 SSBGVD 或 DSBGVD。
SSBGVD 或 DSBGVD	计算广义对称正定带状特征值问题的所有特征值，并可选择性地计算特征向量，以及使用分治法计算特征向量（简单驱动程序）。
SSBGVX (P) 或 DSBGVX (P)	计算广义对称正定带状特征值问题的所选特征值和特征向量（专业驱动程序）。
SSBTRD (P) 或 DSBTRD (P)	通过使用正交相似变换，将对称带状矩阵简化为实数对称三对角形式。

表 45 填充存储例程中的对称矩阵

例程	功能
*x*SPCON	使用由 *x*SPTRF 计算得到的因数分解估算实数或复数对称填充矩阵条件数的倒数。
SSFRK (P) 或 DSFRK (P)	对 RFP 格式的实数矩阵执行对称 k 阶运算。
SSPEV 或 DSPEV	计算填充存储中对称矩阵的所有特征值和特征向量（简单驱动程序）。建议替换为较新版本的 SSPEVD 或 DSPEVD。
SSPEVD 或 DSPEVD	计算填充存储中对称矩阵的所有特征值，并可选择性地计算左侧和/或右侧特征向量。如果需要特征向量，它将使用分治算法（简单驱动程序）。
SSPEVX (P) 或 DSPEVX (P)	计算填充存储中对称矩阵的所选特征值和特征向量（专业驱动程序）。
SSPGST 或 DSPGST	将实数对称正定广义特征值问题简化为标准形式，其中系数矩阵在填充存储中，并使用由 SPPTRF 或 DPPTRF 计算得到的因式分解。建议替换为较新版本的 SSPGVD 或 DSPGVD。
SSPGV 或 DSPGV	计算实数广义对称正定特征值问题的所有特征值和特征向量，其中系数矩阵在填充存储中（简单驱动程序）。建议替换为较新版本的 SSPGVD 或 DSPGVD。
SSPGVD 或 DSPGVD	计算实数广义对称正定特征值问题的所有特征值和特征向量，其中系数矩阵在填充存储中，并使用分治法计算特征向量（驱动程序）。
SSPGVX 或 DSPGVX	计算实数广义对称正定特征值问题的所选特征值和特征向量，其中系数矩阵在填充存储中（专业驱动程序）。
DSPOSV	计算具有实数对称正定矩阵的实数线性方程组的解：首先尝试分解单精度的矩阵，然后根据需要，分解双精度矩阵。

例程	功能
*x*SPRFS (P)	当系数矩阵是填充存储中的对称不定矩阵时，改进计算得到的实数或复数线性方程组的解。
*x*SPSV	计算实数或复数线性方程组的解，其中系数矩阵是填充存储中的对称矩阵（简单驱动程序）。
*x*SPSVX	使用斜旋转因式分解计算线性方程组的解，其中系数矩阵是填充存储中的对称矩阵（专业驱动程序）。
SSPTRD 或 DSPTRD	使用正交相似变换将以填充形式存储的实数对称矩阵简化为实数对称三对角形式。
*x*SPTRF	使用 Bunch-Kaufman 斜旋转方法计算对称填充矩阵的因式分解。
*x*SPTRI	使用由 *x*SPTRF 计算得到的因式分解计算填充存储中对称不定矩阵的逆向值。
*x*SPTRS (P)	使用由 *x*SPTRF 计算得到的因式分解对具有填充存储中的实数或复数对称矩阵的线性方程组求解。

表 46 实数对称三对角矩阵例程

例程	功能
*x*LAED0 (P)	使用分治法计算实数或复数未简化的对称三对角矩阵的所有特征值及相应的特征向量。由 *x*STEDC 使用。
SLAED1 (P) 或 DLAED1 (P)	计算由一阶对称矩阵修改后的实对角矩阵的更新的特征系统。当原始矩阵是三对角矩阵时，由 SSTEDC 或 DSTEDC 使用。
SLAED2 (P) 或 DLAED2 (P)	将两组特征值合并为一个有序集合，并尝试缩小问题的大小。由 SSTEDC 或 DSTEDC 使用。
SLAED3 (P) 或 DLAED3	求特征方程的根并更新特征向量。当原始矩阵是三对角矩阵时，由 SSTEDC 或 DSTEDC 使用。
SLAED4 (P) 或 DLAED4 (P)	求特征方程的单根。由 SSTEDC 或 DSTEDC 使用。
SLAED5 或 DLAED5	对 2x2 特征方程求解。由 SSTEDC 或 DSTEDC 使用。
SLAED6 或 DLAED6	计算离原点最近的正根或负根（对特征方程求解中的单步牛顿法）。
*x*LAED7 (P)	计算由一阶对称矩阵修正后的对角矩阵的更新的特征系统。当原始矩阵是稠密矩阵时，由 *x*STEDC 使用。
*x*LAED8 (P)	将两组特征值合并为一个有序集合，并缩小特征方程。当原始矩阵是稠密矩阵时，由 *x*STEDC 使用。
SLAED9 (P) 或 DLAED9 (P)	求特征方程的根并更新特征向量。当原始矩阵是稠密矩阵时，由 SSTEDC 或 DSTEDC 使用。
SLAEDA (P) 或 DLAEDA (P)	计算一个向量，该向量决定了对角矩阵的一阶修正。当原始矩阵是稠密矩阵时，由 SSTEDC 或 DSTEDC 使用。
SLAGTF 或 DLAGTF (P)	使用部分主元消元法和行交换计算矩阵 T - (lambda * I) 的 LU 因式分解，其中 T 是一般三对角矩阵，lambda 是标量。由 SSTEIN 或 DSTEIN 使用。
SSTEBZ 或 DSTEBZ	计算实数对称三对角矩阵的特征值。
CSTEDC (P) 或 ZSTEDC (P)	使用分治法计算对称三对角矩阵的所有特征值，并可选择性地计算特征向量。如果使用了 CHETRD/ZHETRD、CHPTRD/ZHPTRD 或 CHBTRD/ZHBTRD 将复数厄尔米特全矩阵或带状矩阵简化为三对角形式，那么还可以求该矩阵的特征向量。
SSTEDC 或 DSTEDC	使用分治法计算复数对称三对角矩阵的所有特征值和特征向量。如果使用了 SSYTRD、SSPTRD 或 SSBTRD；DSYTRD、DSPTRD 或 DSBTRD 将实数对称全矩阵或带状矩阵简化为三对角形式，那么还可以求该矩阵的特征向量。

例程	功能
*x*STEGR	使用相对强劲表示法计算实数对称三对角矩阵的所选特征值和特征向量，*x*STEGR 是改进的 *x*STEMR 例程的兼容性包装程序。
*x*STEIN (P)	使用逆迭代计算实数对称三对角矩阵的所选特征向量。
*x*STEMR (P)	使用相对强劲表示法计算实数对称三对角矩阵的所选特征值，并可选择性地计算特征向量。
*x*STEQR (P)	使用 QL 或 QR 算法的 Pal-Walker-Kahan 变体计算实数对称三对角矩阵的所有特征值和特征向量。
SSTERF (P) 或 DSTERF (P)	使用无根 QL 或 QR 算法变体计算实数对称三对角矩阵的所有特征值和特征向量。
SSTEV 或 DSTEV	计算实数对称三对角矩阵的所有特征值和特征向量（简单驱动程序）。建议替换为较新版本的 SSTEVR 或 DSTEVR。
SSTEVD 或 DSTEVD	计算实数对称三对角矩阵的所有特征值和特征向量（简单驱动程序）。建议替换为较新版本的 SSTEVR 或 DSTEVR。
SSTEVR 或 DSTEVR	使用相对强劲表示法计算实数对称三对角矩阵的所选特征值和特征向量。
SSTEVX (P) 或 DSTEVX (P)	计算实数对称三对角矩阵的所选特征值和特征向量（专业驱动程序）。
*x*STSV	计算线性方程组的解，其中系数矩阵是一个对称三对角矩阵（未分块算法）。
*x*STTRF (P)	使用 Bunch-Kaufman 斜旋转方法计算实数或复数对称三对角矩阵的因式分解（未分块算法）。

表 47 对称矩阵例程

例程	功能
*x*LA_SYAMV	执行“矩阵-向量”运算来计算实数或复数对称不定矩阵的误差界。
CLA_SYRCOND_C (P) 或 ZLA_SYRCOND_C (P)	计算实数或复数对称不定矩阵的 op(A)*inv(diag(c)) 的无穷范数条件数。C 是一个 REAL 向量。
CLA_SYRCOND_X (P) 或 ZLA_SYRCOND_X (P)	计算实数或复数对称不定矩阵的 op(A)*inv(diag(x)) 的无穷范数条件数。X 是一个 COMPLEX 向量。
SLA_SYRCOND (P) 或 DLA_SYRCOND (P)	估算实数对称不定矩阵的斯基尔条件数。
*x*LA_SYRFSX_EXTENDED (P)	通过执行超精确迭代细化来改进计算得到的实数或复数对称不定矩阵的线性方程组的解，并提供解的误差界和向后误差估计值。
*x*LASYF	使用斜旋转方法计算实数或复数对称矩阵的部分因式分解。由 *x*SYTRF 使用。
*x*LASYF_ROOK	使用有界 Bunch-Kaufman ("rook") 斜旋转方法计算实数或复数对称矩阵的部分因式分解。由 *x*SYTRF_ROOK 使用。
*x*SYCON	使用由 *x*SYTRF 计算得到的因数分解估算实数或复数对称矩阵条件数的倒数。
*x*SYCON_ROOK	使用由 *x*SYTRF_ROOK 计算得到的因数分解估算实数或复数对称矩阵条件数的倒数。
*x*SYCONV (P)	将由 SSYTRF 或 DSYTRF 计算得到的矩阵转换为下和上三角矩阵，反之亦然。
*x*SYEQUB (P)	计算行和列比例以平衡实数或复数对称矩阵，并按照 2 范数简化其条件数。
SSYEV 或 DSYEV	计算对称矩阵的所有特征值和特征向量（简单驱动程序）。建议替换为较新版本的 SSYEVR 或 DSYEVR。
SSYEVD 或 DSYEVD	计算对称矩阵的所有特征值和特征向量，并使用分治法计算特征向量（专业驱动程序）。建议替换为较新版本的 SSYEVR 或 DSYEVR。

例程	功能
SSYEVR 或 DSYEVR	计算实数对称三对角矩阵的所选特征值和特征向量。
SSYEVX (P) 或 DSYEVX (P)	计算实数对称矩阵的特征值和特征向量（专业驱动程序）。
SSYGS2 或 DSYGS2	使用从 SPOTRF 或 DPOTRF 获得的因式分解结果将实数对称正定广义特征值问题简化为标准形式（未分块算法）。
SSYGST 或 DSYGST	使用由 SPOTRF 或 DPOTRF 计算得到的因式分解将对称正定广义特征值问题简化为标准形式。
SSYGV 或 DSYGV	计算广义对称正定特征值问题的所有特征值和特征向量。建议替换为较新版本的 SSYGVD 或 DSYGVD。
SSYGVD 或 DSYGVD	计算广义对称正定特征值问题的所有特征值和特征向量，并使用分治法计算特征向量（驱动程序）。
SSYGVX 或 DSYGVX	计算广义对称正定特征值问题的所选特征值和特征向量（专业驱动程序）。
*x*SYRFS (P)	当系数矩阵是对称不定矩阵时，改进计算得到的线性方程组的解。
*x*SYRFSX (P)	当系数矩阵是对称不定矩阵时，改进计算得到的线性方程组的解，并提供解的误差界和向后误差估算值（超精度）。
*x*SYSV	对实数或复数对称不定线性方程组求解（简单驱动程序）。将调用 *x*SYTRF，使用斜旋转方法计算复数对称矩阵的因式分解。
*x*SYSV_ROOK	对实数或复数对称不定线性方程组求解（简单驱动程序）。将调用 *x*SYTRF_ROOK，使用有界 Bunch-Kauffman ("rook") 斜旋转方法计算复数对称矩阵的因式分解。
*x*SYSVX	对实数或复数对称不定线性方程组求解（专业驱动程序）。
*x*SYSVXX (P)	对实数或复数对称不定线性方程组求解（专业驱动程序，超精度）。如果需要，将返回范数误差界和最大分量误差界。
SSYTD2 或 DSYTD2	通过正交相似变换，将实数对称矩阵简化为实数对称三对角形式（未分块算法）。
*x*SYTF2	使用斜旋转方法计算实数或复数对称不定矩阵的因式分解（未分块算法）。
*x*SYTF2_ROOK	使用有界 Bunch-Kauffman ("rook") 斜旋转方法计算实数或复数对称不定矩阵的因式分解（未分块算法）。
SSYTRD 或 DSYTRD	通过使用正交相似变换，将实数对称矩阵简化为实数对称三对角形式。
*x*SYTRF (P)	使用 Bunch-Kaufman 斜旋转方法计算实数或复数对称不定矩阵的因式分解（分块算法）。
*x*SYTRI	使用由 *x*SYTRF 计算得到的因式分解计算实数或复数对称不定矩阵的逆向值。
*x*SYTRI_ROOK	使用由 *x*SYTRF_ROOK 计算得到的因式分解计算实数或复数对称不定矩阵的逆向值。
*x*SYTRI2	使用由 *x*SYTRF 计算得到的因式分解计算实数或复数对称不定矩阵的逆向值。在调用实际计算逆向值的 *x*SYTRF2X 之前设置工作区的主维。
*x*SYTRI2X (P)	使用由 *x*SYTRF 计算得到的因式分解计算实数或复数对称不定矩阵的逆向值。由 *x*SYTRI2 使用。
*x*SYTRS (P)	使用由 *x*SYTRF 计算得到的因式分解对具有实数或复数对称矩阵的线性方程组求解。
*x*SYTRS_ROOK (P)	使用由 *x*SYTRF_ROOK 计算得到的因式分解对具有实数或复数对称矩阵的线性方程组求解。
*x*SYTRS2 (P)	使用由 *x*SYTRF 计算得到的以及由 *x*SYCONV 转换得到的因式分解对具有实数或复数对称矩阵的线性方程组求解。

表 48 三角带状矩阵例程

例程	功能
*X*TBCON	估算三角带状矩阵条件数的倒数。
*X*TBRFS (P)	确定误差界和估算值，以对三角带状线性方程组求解。
*X*TBTRS	对三角带状线性方程组求解。

表 49 三角矩阵广义问题（三角矩阵对）例程

例程	功能
*X*TGEVC (P)	计算实数或复数三角矩阵对的一些或全部右侧和/或左侧特征向量，由 *X*GGHRD 和 *X*HGEQZ 计算得到。
*X*TGEXC	使用正交或酉等效变换对实数或复数矩阵对的广义舒尔分解重新排序。
*X*TGSEN (P)	对实数或复数矩阵对的广义舒尔分解重新排序，并计算广义特征值。
*X*TGSJA (P)	从来自 *X*GGSVP 的两个实数或复数三角或梯形矩阵计算广义奇异值 (SVD)。
CTGSNA (P) 或 ZTGSNA (P)	以广义舒尔规范形式估算两个矩阵的指定特征值和特征向量的条件数的倒数。
STGSNA 或 DTGSNA	以广义实数舒尔规范形式估算两个矩阵的指定特征值和特征向量的条件数的倒数。
*X*TGSYL	对广义西尔维斯特方程求解。

表 50 填充存储例程中的三角矩阵

例程	功能
*X*TPCON	估算填充存储中三角矩阵条件数的倒数。
*X*TPMQRT	将从“三角形五角形”块反射器获得的实数或复数正交矩阵应用于一般矩阵，它由两个块组成。
*X*TPQRT	使用精简 WY 表示法计算实数或复数“三角形五角形”矩阵的分块 QR 因式分解，该矩阵由一个三角形块和一个五角形块组成。
*X*TPQRT2	使用精简 WY 表示法计算实数或复数“三角形五角形”矩阵的 QR 因式分解，该矩阵由一个三角形块和一个五角形块组成。
*X*TPRFS (P)	为具有三角填充系数矩阵的实数或复数线性方程组提供解的误差界和向后误差估算值。该解最初应由 *X*TPTRS 或使用某些其他方法获得。
*X*TPTRI	计算填充存储中实数或复数三角矩阵的逆向值。
*X*TPTRS	对系数矩阵在填充存储中的实数或复数三角线性方程组求解。
*X*TPTTF	将实数或复数三角矩阵从标准填充格式 (TP) 复制到矩形全填充格式 (TF)。
*X*TPTTR	将实数或复数三角矩阵从标准填充格式 (TP) 复制到标准完整填充格式 (TR)。

表 51 矩形完整填充 (RFP) 格式中的三角矩阵和标准填充格式例程

例程	功能
*X*TFSM (P)	对具有实数或复数矩阵的矩阵方程求解。一个操作数是 RFP 格式的三角矩阵。

例程	功能
*X*TFTRI	计算以 RFP 格式存储的实数或复数三角矩阵的逆向值。
*X*TFTTP	将实数或复数三角矩阵从矩形完整填充格式 (TF) 复制到标准填充格式 (TP)。
*X*TFTTR	将实数或复数三角矩阵从矩形完整填充格式 (TF) 复制到标准完整格式 (TR)。
*X*TPTTF	将实数或复数三角矩阵从标准填充格式 (TP) 复制到矩形全填充格式 (TF)。
*X*TPTTR	将实数或复数三角矩阵从标准填充格式 (TP) 复制到标准完整填充格式 (TR)。
*X*TRTTF	将实数或复数三角矩阵从标准完整格式 (TR) 复制到矩形完整填充格式 (TF)。
*X*TRTTP	将实数或复数三角矩阵从标准完整格式 (TR) 复制到标准填充格式 (TP)。

表 52 三角矩阵例程

例程	功能
*X*TRCON	估算实数或复数三角矩阵条件数的倒数。
*X*TREVC (P)	计算实数或复数上三角矩阵的右侧和/或左侧特征向量。
*X*TREXC	使用正交或酉相似变换对实数或复数矩阵的舒尔因式分解重新排序。
*X*TRRFS (P)	为具有实数或复数三角矩阵的三角线性方程组提供误差界和估算值。
CTRSEN (P) 或 ZTRSEN (P)	对复数矩阵 A = Q*T*Q**H 的舒尔因式分解重新排序，使选定的特征值群集出现在上三角矩阵 T 的对角线的主要位置中，并使 Q 的主列构成相应的右不变子空间的标准正交基。
STRSEN 或 DTRSEN	对实数矩阵 A = Q*T*Q**T 的舒尔因式分解重新排序，使选定的特征值群集出现在上三角矩阵 T 的对角线的主要位置中，并使 Q 的主列构成相应的右不变子空间的标准正交基。
*X*TRSNA (P)	估算上准三角矩阵的所选特征值和特征向量的条件数的倒数。
*X*TRSYL	对西尔维斯特矩阵方程求解。
*X*TRTRI	计算实数或复数三角矩阵的逆向值（未分块算法）。
*X*TRTRS	对三角线性方程组求解。

表 53 梯形矩阵例程

例程	功能
*X*LARZ	将基本反射器（由 *X*TZRZF 返回）应用于实数或复数一般矩阵。
*X*LARZB (P)	将块反射器或其转置应用于实数一般矩阵，或将块反射器或其共轭转置应用于复数一般矩阵。
*X*LARZT	构成实数或复数块反射器 H 的三角因子 T，该反射器定义为 k 基本反射器的积。
*X*LATZM	已废弃的例程，已由 *X*ORMZ 替换。将由 *X*TZRQF 生成的豪斯霍尔德矩阵应用于实数或复数矩阵。
*X*TZRQF (P)	已废弃的例程，已由例程 *X*TZRZF 替换。
*X*TZRZF (P)	通过正交变换，将矩形上梯形矩阵简化为上三角形式。

表 54 酉矩阵例程

例程	功能
CUNBDB 或 ZUNBDB	使 MxM 已分区酉矩阵的块同时双对角化。
CUNBDB1 或 ZUNBDB1	使具有正交列的高瘦矩阵（变体 1）的块同时双对角化。
CUNBDB2 或 ZUNBDB2	使具有正交列的高瘦矩阵（变体 2）的块同时双对角化。
CUNBDB3 或 ZUNBDB3	使具有正交列的高瘦矩阵（变体 3）的块同时双对角化。
CUNBDB4 或 ZUNBDB4	使具有正交列的高瘦矩阵（变体 4）的块同时双对角化。
CUNBDB5 或 ZUNBDB5	使列向量 X 相对于 Q 的正交列正交化。
CUNBDB5 或 ZUNBDB5	使列向量 X 相对于 Q 的正交列正交化。由 CUNBDB5 或 ZUNBDB5 使用。
CUNCSD2BY1 或 ZUNCSD2BY1	计算其正交列已分区到 2x1 块结构中的 MxQ 矩阵 X 的 CS 分解。
CUNG2L (P) 或 ZUNG2L (P)	生成具有正交列的 MxN 复数矩阵 Q，它定义为 M 阶的 K 基本反射器的积的最后 N 列，如 CGEQLF 或 ZGEQLF 返回的那样。
CUNG2R (P) 或 ZUNG2R (P)	生成具有正交列的 MxN 复数矩阵 Q，它定义为 M 阶的 K 基本反射器的积的最后 N 列，如 CGEQRF 或 ZGEQRF 返回的那样。
CUNGBR (P) 或 ZUNGBR (P)	通过简化为双对角形式生成酉变换矩阵，如 CGEBRD 或 ZGEBRD 确定的那样。
CUNGHR (P) 或 ZUNGHR (P)	生成简化为海森伯格形式的正交变换矩阵，如 CGEHRD 或 ZGEHRD 确定的那样。
CUNGL2 (P) 或 ZUNGL2 (P)	从 LQ 因式分解生成所有或部分酉矩阵 Q，如 CGELQF 或 ZGELQF 确定的那样（未分块算法）。
CUNGLQ (P) 或 ZUNGLQ (P)	从 LQ 因式分解生成酉矩阵 Q，如 CGELQF 或 ZGELQF 返回的那样。
CUNGQL (P) 或 ZUNGQL (P)	从 QL 因式分解生成酉矩阵 Q，如 CGEQLF 或 ZGEQLF 返回的那样。
CUNGQR (P) 或 ZUNGQR (P)	从 QR 因式分解生成酉矩阵 Q，如 CGEQRF 或 ZGEQRF 返回的那样。
CUNGR2 (P) 或 ZUNGR2 (P)	从 RQ 因式分解生成所有或部分酉矩阵 Q，如 CGERQF 或 ZGERQF 确定的那样（未分块算法）。
CUNGRQ (P) 或 ZUNGRQ (P)	从 RQ 因式分解生成酉矩阵 Q，如 CGERQF 或 ZGERQF 返回的那样。
CUNGTR (P) 或 ZUNGTR (P)	生成由 CHETRD 或 ZHETRD 简化为三对角形式的酉矩阵。
CUNM2L 或 ZUNM2L	将一般矩阵乘以来自由 CGEQLF 或 ZGEQLF 确定的 QL 因式分解的酉矩阵（未分块算法）。
CUNM2R 或 ZUNM2R	将一般矩阵乘以来自由 CGEQRF 或 ZGERLF 确定的 QR 因式分解的酉矩阵（未分块算法）。
CUNMBR 或 ZUNMBR	将一般矩阵乘以简化为双对角形式的酉变换矩阵，如 CGEBRD 或 ZGEBRD 确定的那样。
CUNMHR 或 ZUNMHR	将一般矩阵乘以由 CGEHRD 或 ZGEHRD 简化为海森伯格形式的酉矩阵。
CUNML2 或 ZUNML2	将一般矩阵乘以来自由 CGELQF 或 ZGELQF 确定的 LQ 因式分解的酉矩阵（未分块算法）。
CUNMLQ 或 ZUNMLQ	将一般矩阵乘以来自 LQ 因式分解的酉矩阵，如 CGELQF 或 ZGELQF 返回的那样。
CUNMQL 或 ZUNMQL	将一般矩阵乘以来自 QL 因式分解的酉矩阵，如 CGEQLF 或 ZGEQLF 返回的那样。
CUNMQR 或 ZUNMQR	将一般矩阵乘以来自 QR 因式分解的酉矩阵，如 CGEQRF 或 ZGEQRF 返回的那样。
CUNMR2 或 ZUNMR2	将一般矩阵乘以来自由 CGERQF 或 ZGERQF 确定的 RQ 因式分解的酉矩阵（未分块算法）。
CUNMR3 或 ZUNMR3	将一般矩阵乘以来自由 CTZRZF 或 ZTZRZF 确定的 RZ 因式分解的酉矩阵（未分块算法）。

例程	功能
CUNMRQ 或 ZUNMRQ	将一般矩阵乘以来自 RQ 因式分解的酉矩阵，如 CGERQF 或 ZGERQF 返回的那样。
CUNMRZ 或 ZUNMRZ	将一般矩阵乘以来自 RZ 因式分解的酉矩阵，如 CTZRZF 或 ZTZRZF 返回的那样。
CUNMTR 或 ZUNMTR	将一般矩阵乘以由 CHETRD 或 ZHETRD 简化为三对角形式的酉变换矩阵。

表 55 填充存储例程中的酉矩阵

例程	功能
CUPGTR (P) 或 ZUPGTR (P)	从由 CHPTRD 或 ZHPTRD 确定的三对角矩阵生成酉变换矩阵。
CUPMTR 或 ZUPMTR	将一般矩阵乘以由 CHPTRD 或 ZHPTRD 简化为三对角形式的酉变换矩阵。

BLAS1 例程

表 56 “BLAS1（Basic Linear Algebra Subprogram, Level 1，基础线性代数子程序，1 级）例程”列出了 Oracle Developer Studio 性能库 BLAS1 例程。当前没有对 Oracle Developer Studio 性能库 BLAS1 例程进行并行化处理。

表 56 BLAS1（Basic Linear Algebra Subprogram, Level 1，基础线性代数子程序，1 级）例程

例程	功能
SASUM、DASUM、SCASUM、DZASUM	向量绝对值的和
*X*AXPY	标量和向量的积加上一个向量
*X*COPY	复制向量
SDOT、DDOT、DSDOT、SDSDOT、CDOTU、ZDOTU、DQDOTA、DQDOTI	点积（内积）四精度 DQDOTA、DQDOTI 仅在 SPARC 上可用
CDOTC、ZDOTC	结合第一个向量的点积
SNRM2、DNRM2、SCNRM2、DZNRM2	向量的欧几里得范数
*X*ROTG	设置吉文斯平面旋转
SROT、DROT、CSROT、ZDROT	应用吉文斯平面旋转
SROTMG、DROTMG	设置修正后的吉文斯平面旋转
SROTM、DROTM	应用修正后的吉文斯旋转
ISAMAX、IDAMAX、ICAMAX、IZAMAX	绝对值最大的元素的索引
*X*SCAL、CSSCAL、ZDSCAL	对向量进行缩放
*X*SWAP	将两个向量进行交换
CVMUL、ZVMUL	计算按比例缩放的复数向量的积

BLAS2 例程

表 57 “BLAS2（Basic Linear Algebra Subprogram, Level 2，基础线性代数子程序，2 级）例程”列出了 Oracle Developer Studio 性能库 BLAS2 例程。(P) 表示已进行了并行化处理的例程。

表 57 BLAS2（Basic Linear Algebra Subprogram, Level 2，基础线性代数子程序，2 级）例程

例程	功能
*X*GBMV	带状存储中的矩阵和向量的积
*X*GEMV (P)	一般矩阵和向量的积
SGER (P)、DGER (P)、CGERC (P)、ZGERC (P)、CGERU (P)、ZGERU (P)	1 阶更新至一般矩阵
CHBMV 或 ZHBMV	带状存储中的厄尔米特矩阵和向量的积
CHEMV (P) 或 ZHEMV (P)	厄尔米特矩阵和向量的积
CHER (P) 或 ZHER (P)	1 阶更新至厄尔米特矩阵
CHER2 或 ZHER2	2 阶更新至厄尔米特矩阵
CHPMV (P) 或 ZHPMV (P)	填充存储中的厄尔米特矩阵和向量的积
CHPR 或 ZHPR	1 阶更新至填充存储中的厄尔米特矩阵
CHPR2 或 ZHPR2	2 阶更新至填充存储中的厄尔米特矩阵
SSBMV 或 DSBMV	带状存储中的对称矩阵和向量的积
SSPMV (P) 或 DSPMV (P)	填充存储中的对称矩阵和向量的积
SSPR 或 DSPR	1 阶更新至填充存储中的实数对称矩阵
SSPR2 (P) 或 DSPR2 (P)	2 阶更新至填充存储中的实数对称矩阵
*X*SYMV (P)	对称矩阵和向量的积
SSYR (P) 或 DSYR (P)	1 阶更新至实数对称矩阵
SSYR2 (P) 或 DSYR2 (P)	2 阶更新至实数对称矩阵
*X*TBMV	带状存储中的三角矩阵和向量的积
*X*TBSV	带状存储中三角线性方程组的解
*X*TPMV	填充存储中的三角矩阵和向量的积
*X*TPSV	填充存储中三角线性方程组的解
*X*TRMV (P)	三角矩阵和向量的积
*X*TRSV (P)	三角线性方程组的解

BLAS3 例程

表 58 “BLAS3（Basic Linear Algebra Subprogram, Level 3，基础线性代数子程序，3 级）例程”列出了 Oracle Developer Studio 性能库 BLAS3 例程。(P) 表示已进行了并行化处理的例程。

表 58 BLAS3（Basic Linear Algebra Subprogram, Level 3，基础线性代数子程序，3 级）例程

例程	功能
*X*GEMM (P)	两个一般矩阵的积
CHEMM (P) 或 ZHEMM (P)	厄尔米特矩阵和一般矩阵的积
CHERK (P) 或 ZHERK (P)	厄尔米特矩阵的 k 阶更新
CHER2K (P) 或 ZHER2K (P)	厄尔米特矩阵的 2k 阶更新
*X*SYMM (P)	对称矩阵和一般矩阵的积
*X*SYRK (P)	对称矩阵的 k 阶更新
*X*SYR2K (P)	对称矩阵的 2k 阶更新
*X*TRMM (P)	三角矩阵和一般矩阵的积
*X*TRSM (P)	三角方程组的解

稀疏 BLAS 例程

表 59 “稀疏 BLAS 例程”列出了 Oracle Developer Studio 性能库稀疏 BLAS 例程。(P) 表示已进行了并行化处理的例程。

表 59 稀疏 BLAS 例程

例程	功能
*X*AXPYI	将稀疏向量 *X* 的纯量倍数加到全向量 *Y*。
*X*BCOMM (P)	块坐标矩阵与矩阵相乘。
*X*BDIMM (P)	块对角格式矩阵与矩阵相乘。
*X*BDISM (P)	块对角格式三角求解。
*X*BELMM (P)	块 Ellpack 格式矩阵与矩阵相乘。
*X*BELSM (P)	块 Ellpack 格式三角求解。
*X*BSCMM (P)	块压缩稀疏列格式矩阵与矩阵相乘。
*X*BSCSM (P)	块压缩稀疏列格式三角求解。
*X*BSRMM (P)	块压缩稀疏行格式矩阵与矩阵相乘。
*X*BSRSM (P)	块压缩稀疏行格式三角求解。
*X*COOMM (P)	坐标格式矩阵与矩阵相乘。

例程	功能
*X*CSCMM (P)	压缩稀疏列格式矩阵与矩阵相乘。
*X*CSCSM (P)	压缩稀疏列格式三角求解。
*X*CSRMM (P)	压缩稀疏行格式矩阵与矩阵相乘。
*X*CSRSM (P)	压缩稀疏行格式三角求解。
*X*DIAMM (P)	对角线格式矩阵与矩阵相乘。
*X*DIASM (P)	对角线格式三角求解。
SDOTI、DDOTI、CDOTUI 或 ZDOTUI	计算稀疏向量和全向量的点积。
CDOTCI 或 ZDOTCI	计算稀疏向量和全向量的共轭点积。
*X*ELLMM (P)	Ellpack 格式矩阵与矩阵相乘。
*X*ELLSM (P)	Ellpack 格式三角求解。
*X*GTHR	给定一个全向量，创建稀疏向量和相应的索引向量。
*X*GTHRZ	给定一个全向量，创建稀疏向量和相应的索引向量并使全向量归零。
*X*JADMM (P)	锯齿对角线矩阵与矩阵相乘。
SJADRP 或 DJADRP	锯齿对角线矩阵的右排列。
*X*JADSM (P)	锯齿对角线三角求解。
SROTI 或 DROTI	将吉文斯旋转应用于稀疏向量和全向量。
*X*SCTR	给定一个稀疏向量和相应的索引向量，将这些元素放到全向量中。
*X*SKYMM (P)	天际线格式矩阵与矩阵相乘。
*X*SKYSM (P)	天际线格式三角求解。
*X*VBRMM (P)	可变块稀疏行格式矩阵与矩阵相乘。
*X*VBRSM (P)	可变块稀疏行格式三角求解。

稀疏求解器例程

以下各表列出了 Oracle Developer Studio 性能库中 SPSOLVE 和 SuperLU 稀疏求解器中的例程。(P) 表示已进行了并行化处理的例程。

表 60 SPSOLVE 例程

例程	功能
xGSSFS (P)	调用一次 SPSOLVE 的接口。
xGSSIN	SPSOLVE 初始化。
xGSSOR	填充简化顺序和符号因式分解。
xGSSFA (P)	矩阵值输入和数值因式分解。
xGSSSL	三角求解。
xGSSUO	设置用户指定的顺序排列。
xGSSRP	返回求解器使用的排列。

例程	功能
xGSSCO	返回系数矩阵的条件数估算值。
xGSSDA	取消分配 SPSOLVE 内存。
xGSSPS	打印求解器统计数据。

表 61 SuperLU 例程

例程	功能
xgstrf	计算因式分解
xgssvx	分解和求解（专业驱动程序）
xgssv	分解和求解（简单驱动程序）
xgstrs	计算三角求解
xgsrfs	改进计算得到的解；提供误差界
xlangs	计算 1 范数、弗罗贝尼乌斯范数或无穷范数
xgsequ	计算行和列比例
xgscon	估算条件数的倒数
xlaqgs	平衡一般稀疏矩阵
LUSolveTime	返回在求解阶段所用的时间
LUFactTime	返回在因式分解阶段所用的时间
LUFactFlops	返回因式分解阶段中的浮点运算数
LUSolveFlops	返回求解阶段中的浮点运算数
xQuerySpace	返回有关内存统计数据的信息
sp_ienv	返回指定的计算机相关参数
xPrintPerf	打印由计算例程收集的统计信息
set_default_options	将控制求解器行为的参数设置为默认选项
StatInit	分配并初始化用于存储性能统计信息的结构
StatFree	释放用于存储性能统计信息的结构
Destroy_Dense_Matrix	取消分配密集格式的 SuperMatrix
Destroy_SuperNode_Matrix	取消分配超节点格式的 SuperMatrix
Destroy_CompCol_Matrix	取消分配压缩稀疏列格式的 SuperMatrix
Destroy_CompCol_Permuted	取消分配已排列的压缩稀疏列格式的 SuperMatrix
Destroy_SuperMatrix_Store	取消分配将矩阵存储在 SuperMatrix 中的实际存储
xCopy_CompCol_Matrix	复制压缩稀疏列格式的 SuperMatrix
xCreate_CompCol_Matrix	分配压缩稀疏列格式的 SuperMatrix
xCreate_Dense_Matrix	分配密集格式的 SuperMatrix
xCreate_CompRow_Matrix	分配压缩稀疏行格式的 SuperMatrix
xCreate_SuperNode_Matrix	分配超节点格式的 SuperMatrix
sp_preorder	排列原始稀疏矩阵的列
sp_sgemm sp_dgemm sp_cgemm sp_zgemm	将 SuperMatrix 乘以稠密矩阵

信号处理库例程

Oracle Developer Studio 性能库包含用于计算快速傅里叶变换、正弦和余弦变换以及卷积和相关的例程。

FFT 例程

Oracle Developer Studio 性能库提供了一组 FFT 接口，它们取代了早期 Oracle Developer Studio 性能库发行版中提供的 FFTPACK 和 VFFTPACK 例程的子集。库中包括了以前的 FFT 接口以进行向后兼容，建议用户使用新接口。有关各 FFT 例程的信息，请参见第 3P 部分的手册页。

表 62 “FFT 例程”介绍了 Oracle Developer Studio 性能库 FFT 例程和相应的 FFTPACK 及 VFFTPACK 例程之间的映射。(P) 表示已进行了并行化处理的例程。

表 62 FFT 例程

例程	替换	功能
CFFTC (P)	CFFTI	初始化三角权重和因子表，或计算复数序列的一维正向或逆向 FFT。
	CFFTF (P)	
	CFFTB (P)	
CFFTC2 (P)	CFFT2I	初始化三角权重和因子表，或计算二维复数数组的二维正向或逆向 FFT。
	CFFT2F (P)	
	CFFT2B (P)	
CFFTC3 (P)	CFFT3I	初始化三角权重和因子表，或计算三维复数数组的三维正向或逆向 FFT。
	CFFT3F (P)	
	CFFT3B (P)	
CFFTCM (P)	VCFFTI	初始化三角权重和因子表，或计算存储在二维复数数组中的一组数据序列的一维正向或逆向 FFT。
	VCFFTF (P)	
	VCFFTB (P)	
CFFTS	RFFTI、RFFTB	初始化三角权重和因子表，或计算复数序列的一维逆向 FFT。
	EZFFTI、EZFFTB	
CFFTS2	RFFT2I	初始化三角权重和因子表，或计算二维复数数组的二维逆向 FFT。
	RFFT2B	
CFFTS3 (P)	RFFT3I	初始化三角权重和因子表，或计算三维复数数组的三维逆向 FFT。
	RFFT3B	

例程	替换	功能
CFFTSM	VRFFTI	初始化三角权重和因子表，或计算存储在二维复数数组中的一组数据序列的一维逆向 FFT。
	VRFFTB (P)	
DFFTZ	DFFTI、DFFTF	初始化三角权重和因子表，或计算双精度序列的一维正向 FFT。
	DEZFFTI、DEZFFTF	
DFFTZ2	DFFT2I	初始化三角权重和因子表，或计算二维双精度数组的二维正向 FFT。
	DFFT2F	
DFFTZ3 (P)	DFFT3I	初始化三角权重和因子表，或计算三维双精度数组的三维正向 FFT。
	DFFT3F	
DFFTZM	VDFFTI	初始化三角权重和因子表，或计算存储在二维双精度数组中的一组数据序列的一维正向 FFT。
	VDFFTF (P)	
SFFTC	RFFTI、RFFTF	初始化三角权重和因子表，或计算实数序列的一维正向 FFT。
	EZFFTI、EZFFTF	
SFFTC2	RFFT2I	初始化三角权重和因子表，或计算二维实数数组的二维正向 FFT。
	RFFT2F	
SFFTC3 (P)	RFFT3I	初始化三角权重和因子表，或计算三维实数数组的三维正向 FFT。
	RFFT3F	
SFFTCM	VRFFTI	初始化三角权重和因子表，或计算存储在二维实数数组中的一组数据序列的一维正向 FFT。
	VRFFTF (P)	
ZFFTD	DFFTI、DFFTB	初始化三角权重和因子表，或计算双精度复数序列的一维逆向 FFT。
	DEZFFTI、DEZFFTB	
ZFFTD2	DFFT2I	初始化三角权重和因子表，或计算二维双精度复数数组的二维逆向 FFT。
	DFFT2B	
ZFFTD3 (P)	DFFT3I	初始化三角权重和因子表，或计算三维双精度复数数组的三维逆向 FFT。
	DFFT3B	
ZFFTDM	VDFFTI	初始化三角权重和因子表，或计算存储在二维双精度复数数组中的一组数据序列的一维逆向 FFT。
	VDFFTB (P)	
ZFFTZ (P)	ZFFTI	初始化三角权重和因子表，或计算双精度复数序列的一维正向或逆向 FFT。
	ZFFTF (P)	
	ZFFTB (P)	
ZFFTZ2 (P)	ZFFT2I	初始化三角权重和因子表，或计算二维双精度复数数组的二维正向或逆向 FFT。
	ZFFT2F (P)	
	ZFFT2B (P)	
ZFFTZ3 (P)	ZFFT3I	初始化三角权重和因子表，或计算三维双精度复数数组的三维正向或逆向 FFT。

例程	替换	功能
ZFFTZM (P)	ZFFT3F (P) ZFFT3B (P) VZFFTI VZFFTF (P) VZFFTB (P)	初始化三角权重和因子表，或计算存储在二维双精度复数数组中的一组数据序列的一维正向或逆向 FFT。

快速余弦和正弦变换

Oracle Developer Studio 性能库快速余弦和正弦变换例程以 FFTPACK (http://www.netlib.org/fftpack/) 中包含的例程为基础。具有 V 前缀的例程是根据 VFFTPACK (http://www.netlib.org/vfftpack/) 中包含的例程进行了向量化处理的例程。

表 63 “正弦和余弦变换例程”列出了 Oracle Developer Studio 性能库正弦和余弦变换例程。

表 63 正弦和余弦变换例程

例程	功能
COSQB、DCOSQB、VCOSQB、VDCOSQB	余弦四分之一波长合成。
COSQF、DCOSQF、VCOSQF、VDCOSQF	余弦四分之一波长变换。
COSQI、DCOSQI、VCOSQI、VDCOSQI	初始化余弦四分之一波长变换和合成。
COST、DCOST、VCOST、VDCOST	余弦偶数波长变换。
COSTI、DCOSTI、VCOSTI、VDCOSTI	初始化余弦偶数波长变换。
SINQB、DSINQB、VSINQB、VDSINQB	正弦四分之一波长合成。
SINQF、DSINQF、VSINQF、VDSINQF	正弦四分之一波长变换。
SINQI、DSINQI、VSINQI、VDSINQI	初始化正弦四分之一波长变换和合成。
SINT、DSINT、VSINT、VDSINT	正弦奇数波长变换。
SINTI、DSINT、VSINTI、VDSINTI	初始化正弦奇数波长变换。

卷积和相关例程

表 64 “卷积和相关例程”列出了 Oracle Developer Studio 性能库卷积和相关例程。

表 64 卷积和相关例程

例程	功能
XCNVCOR	计算卷积或相关

例程	功能
XCNVCOR2	计算二维卷积或相关

其他信号处理例程

表 65 “卷积和相关例程”列出了 Oracle Developer Studio 性能库的其他信号处理例程。

表 65 卷积和相关例程

例程	功能
RFFTOPT、DFFTOPT、CFFTOPT、ZFFTOPT	计算最近 FFT 的长度
SWIENER 或 DWEINER	执行两个信号的维纳解卷积
XTRANS (P)	转置数组

有关使用每个例程的信息，请参见第 3P 部分的手册页。

排序例程

表 66 “排序例程”列出了 Oracle Developer Studio 性能库排序例程。

表 66 排序例程

例程	功能
BLAS_DSORT (P)	使用快速排序算法以升序或降序对实数（双精度）向量 X 排序。
BLAS_DSORTV (P)	使用快速排序算法以升序或降序对实数（双精度）向量 X 排序，并使用排列向量覆盖 P。
BLAS_DPERMUTE (P)	按 DSORTV 输出的排列向量 P 排列实（双精度）数组。
BLAS_ISORT (P)	使用快速排序算法以升序或降序对整数向量 X 排序。
BLAS_ISORTV (P)	使用快速排序算法以升序或降序对实数向量 X 排序，并使用排列向量覆盖 P。
BLAS_IPERMUTE (P)	按 DSORTV. 输出的排列向量 P 来排列整数数组。
BLAS_SSORT (P)	使用快速排序算法以升序或降序对实数向量 X 排序。
BLAS_SSORTV (P)	使用快速排序算法以升序或降序对实数向量 X 排序，并使用排列向量覆盖 P。
BLAS_SPERMUTE (P)	按 DSORTV 输出的排列向量 P 来排列实数数组。

索引

E

F

G

H

J

K

L

M

N

Z