Oracle® Developer Studio 12.5: パフォーマンスライブラリユーザーズガイド

Part No: E71963
2016 年 6 月

ORACLE

Part No: E71963

ドキュメントのアクセシビリティについて

オラクルのアクセシビリティについての詳細情報は、Oracle Accessibility ProgramのWeb サイト(http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc)を参照してください。

Oracle Supportへのアクセス

サポートをご契約のお客様には、My Oracle Supportを通して電子支援サービスを提供しています。詳細情報は(http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info)か、聴覚に障害のあるお客様は (http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs)を参照してください。

目次

このドキュメントの使用法 9

1 Oracle Developer Studio パフォーマンスライブラリの概要 11
Oracle Developer Studio パフォーマンスライブラリに含まれているライブラリ 11
Netlib について 12
関連ドキュメント 13
Oracle Developer Studio パフォーマンスライブラリの機能 14
数学ルーチン 14
以前の LAPACK バージョンとの互換性 15
Oracle Developer Studio パフォーマンスライブラリの使用開始 15
▼ SPARC プラットフォームでのトラップ 6 の有効化 16

2 Oracle Developer Studio パフォーマンスライブラリの使用 19
アプリケーションパフォーマンスの向上 19
Oracle Developer Studio パフォーマンスライブラリのルーチンによるルーチンの置き換え 19
ほかのライブラリのパフォーマンス向上 19
ツールを使用したコード再構築 20
Fortran インタフェース 20
Fortran 95 で使用するための Fortran `SUNPERF` モジュール 21
Fortran の例 22
C インタフェース 24
C の例 25

3 アプリケーションの最適化 27
32 ビット環境と 64 ビット環境の比較 27
Oracle Developer Studio パフォーマンスライブラリの使用 27
Fortran プログラムのリンク 28
C および C++ プログラムのリンク 28
コンパイルについて 28

64 ビットに対応したオペレーティング環境用のコードのコンパイル 29
64 ビット整数引数 29

4 並列処理 33
実行時の問題 33
並列度 34
同期メカニズム 35
並列処理の例 36

5 行列の操作 39
行列の格納スキーム 39
バンド格納 39
パック格納 40
行列のタイプ 41
一般行列 41
三角行列 41
対称行列 42
三重対角行列 43

6 スパース計算 45
スパース行列 45
対称スパース行列 46
構造的対称スパース行列 47
非対称スパース行列 47
スパース BLAS 48
Netlib スパース BLAS 48
NIST Fortran スパース BLAS 49
SPSOLVE インタフェース 50
SPSOLVE ルーチン 50
SPSOLVE ルーチンの呼び出し順序 51
SPSOLVE の例 52
SuperLU インタフェース 61
C からの SuperLU の呼び出し 62
Fortran からの SuperLU の呼び出し 66
SuperLU の例 67
スパース BLAS およびソルバーの参考資料 70

7 Oracle Developer Studio パフォーマンスライブラリの信号処理ルーチンの使用 73

順方向および逆方向 FFT ルーチン 74
1 次元 FFT ルーチン 76
2 次元 FFT ルーチン 83
3 次元 FFT ルーチン 88
コメント 93
コサインおよびサイン変換 94
高速コサインおよびサイン変換ルーチン 95
高速サイン変換 96
高速コサイン変換 97
離散高速コサインおよびサイン変換とそれらの逆変換 97
高速コサイン変換の例 101
高速サイン変換の例 103
畳み込みおよび相関 105
畳み込み演算 105
相関演算 106
Oracle Developer Studio パフォーマンスライブラリの畳み込みルーチンと相関ルーチン 107
畳み込みルーチンと相関ルーチンの引数 107
畳み込みルーチンと相関ルーチンの作業配列 WORK 109
サンプルプログラム: 畳み込み 111
参考資料 114

A Oracle Developer Studio パフォーマンスライブラリのルーチン 117
LAPACK ルーチン 117
BLAS1 ルーチン 140
BLAS2 ルーチン 141
BLAS3 ルーチン 142
スパース BLAS ルーチン 143
スパースソルバールーチン 144
信号処理ライブラリルーチン 145
FFT ルーチン 145
高速コサイン/サイン変換 148
畳み込みおよび相関ルーチン 148
その他の信号処理ルーチン 148
ソートルーチン 149

索引 151

このドキュメントの使用法

- **概要** – Oracle Developer Studio Fortran 95、C++、および C コンパイラでサポートされる、Oracle Developer Studio パフォーマンスライブラリサブルーチンの固有の拡張および機能を使用する方法について説明します。
- **対象読者** - アプリケーション開発者、システム開発者、アーキテクト、サポートエンジニア。
- **必要な知識** – Fortran または C 言語の実用的な知識、数値解析の基本知識、および Netlib (http://www.netlib.org) から入手できる LAPACK および BLAS 基本ライブラリに関するある程度の理解が必要です。

製品ドキュメントライブラリ

この製品および関連製品のドキュメントとリソースは http://www.oracle.com/pls/topic/lookup?ctx=E71939 で入手可能です。

フィードバック

このドキュメントに関するフィードバックを http://www.oracle.com/goto/docfeedback からお聞かせください。

♦ ♦ ♦ 第 1 章

1 Oracle Developer Studio パフォーマンスライブラリの概要

Oracle Developer Studio パフォーマンスライブラリは、線形代数や大量の数値計算を伴う問題を解くために最適化された高速な数学サブルーチンのセットです。Oracle Developer Studio パフォーマンスライブラリは、http://www.netlib.org の Netlib から入手できるパブリックドメインアプリケーションのコレクションに基づいています。これらのパブリックドメインアプリケーションが拡張され、Oracle Developer Studio パフォーマンスライブラリとしてバンドルされています。

このドキュメントは、Netlib で提供されている基本アプリケーションに対する Oracle 固有の拡張について説明します。基本ルーチンに関する参考資料は、Netlib および SIAM (Society for Industrial and Applied Mathematics) から入手できます。さらに、Oracle Developer Studio 12.5 パフォーマンスライブラリの公開関数およびサブルーチンについては、Oracle Developer Studio のマニュアルページのセクション 3P で詳しく説明されています。3P マニュアルページに関する情報を表示するには、`man -s 3p intro` と入力します。Oracle Developer Studio のマニュアルページのパスを `MANPATH` 変数に追加する方法の詳細については、『Oracle Developer Studio 12.5: インストールガイド』の「開発ツールとマニュアルページ用の環境変数の設定」を参照してください。

Oracle Developer Studio パフォーマンスライブラリに含まれているライブラリ

パフォーマンスライブラリには、次の標準ライブラリの拡張バージョンが含まれています。

- LAPACK バージョン 3.5.0 – 線形代数問題解決用です。
- BLAS1 (Basic Linear Algebra Subprograms、基本線形代数サブプログラム) – ベクトルとベクトルの演算実行用です。
- BLAS2 – 行列とベクトルの演算実行用です。
- BLAS3 – 行列と行列の演算実行用です。
- Netlib Sparse-BLAS - スパースベクトル演算実行用です。
- NIST Sparse-BLAS 0.5 - 基本的なスパース行列演算実行用です。
- SuperLU 3.0 - スパース連立 1 次方程式の解決用です。

- Sparse Solver - ダイレクトスパースソルバールーチン
- FFTPACK - 高速フーリエ変換の実行
- VFFTPACK - ベクトル化された高速フーリエ変換の実行
- XBLAS - 高精度基本線形代数サブルーチン
- その他のルーチン - 転置、畳み込み、相関、およびソート

注記 - LINPACK は Oracle Developer Studio パフォーマンスライブラリから削除されました。LAPACK バージョン 3.5.0 は、LINPACK および以前のバージョンの LAPACK すべてに代わるものです。LINPACK ルーチンがまだ必要な場合、LINPACK ライブラリおよびドキュメントは `http://www.netlib.org` から入手できます。

Oracle Developer Studio パフォーマンスライブラリは静的ライブラリ形式と動的ライブラリ形式の両方で使用できます。`sparcvis`、`sparcvis2`、および `sparcfmaf` には最適化された SPARC バージョンがあり、Oracle Solaris 11 および Oracle Linux オペレーティングシステムには高度なアーキテクチャーがあります。Oracle Solaris 11 システムおよび Oracle Linux システムの x86/x64 アーキテクチャー用に最適化されたバージョンもあります。すべてのバージョンがマルチプロセッサプラットフォーム上での並列プログラミングをサポートしています。詳細については、『Oracle Developer Studio 12.5: リリースノート』を参照してください。

Oracle Developer Studio パフォーマンスライブラリの LAPACK ルーチンは Fortran 95 コンパイラでコンパイルされており、Netlib の LAPACK バージョン 3.5.0 ライブラリとの互換性を保っています。パフォーマンスライブラリバージョンのこれらのルーチンは、Fortran 呼び出し可能ルーチンと同じ操作を実行し、標準の Netlib バージョンと同じインタフェースを持っています。

LAPACK にはドライバルーチン、計算ルーチン、および補助ルーチンが含まれています。補助ルーチンは予告なく変更されたり LAPACK から削除されたりする可能性があるため、パフォーマンスライブラリではサポートされていません。補助ルーチンはサポートされていないため、このユーザーガイドにも 3P セクションのマニュアルページにも記載されていません。

多くの補助ルーチンでは、ルーチン名の 2 番目および 3 番目の文字として LA が使用されていますが、そうでないものもあります。『LAPACK User's Guide, Third Edition』(`http://www.netlib.org/lapack/lug/`) の付録 B に補助ルーチンのリストがあります。

Netlib について

Netlib は数学関連のソフトウェア、論文、およびデータベースのオンラインリポジトリで、AT&T ベル研究所、テネシー大学、オークリッジ国立研究所、および世界中の専門家によって管理されています。

Netlib では、Oracle Developer Studio パフォーマンスライブラリで使用されているライブラリのほかにも多数のライブラリが提供されています。このようなライブラリの一部は、パフォーマンスライブラリで使用されているライブラリに似ていても異なっている場合があり、パフォーマンスライブラリとの互換性がないことがあります。

ほかのライブラリのルーチンを使用すると、Oracle Developer Studio パフォーマンスライブラリのルーチンとだけでなく、Netlib LAPACK 基本ルーチンとも互換性の問題が発生する可能性があります。ほかのライブラリのルーチンを使用する場合は、それらのライブラリに付属のドキュメントを参照してください。

たとえば、Netlib では CLAPACK ライブラリが提供されていますが、CLAPACK インタフェースは Oracle Developer Studio パフォーマンスライブラリに含まれている C インタフェースとは異なっています。Netlib には LAPACK 90 ライブラリパッケージもあります。LAPACK 90 ライブラリのインタフェースは、Oracle Developer Studio パフォーマンスライブラリの Fortran 95 インタフェースおよび Netlib LAPACK バージョン 3.5.0 のインタフェースとは異なっています。LAPACK 90 を使用する場合は、そのライブラリに付属のドキュメントを参照してください。

Oracle Developer Studio パフォーマンスライブラリでサポートされている基本ライブラリについては、Netlib にある詳細情報がこのユーザーズガイドの補足となる場合があります。『LAPACK User's Guide, Third Edition』(http://www.netlib.org/lapack/lug/) では、LAPACK のアルゴリズムおよびルーチンの使用方法が説明されていますが、基本ルーチンに対して Oracle Developer Studio パフォーマンスライブラリで行われた拡張は説明されていません。

関連ドキュメント

『*LAPACK User's Guide*』は、LAPACK バージョン 3.5.0 基本ルーチンに関する公式の解説書です。『*LAPACK Users' Guide*』のオンラインバージョンは http://www.netlib.org/lapack/lug/、印刷されたバージョンは SIAM (Society for Industrial and Applied Mathematics) http://www.siam.org から入手できます。

Oracle Developer Studio パフォーマンスライブラリのルーチンには、『*LAPACK Users' Guide*』で説明されていないパフォーマンス向上、拡張、および機能が含まれています。ただし、Oracle Developer Studio パフォーマンスライブラリでは LAPACK 基本ルーチンとの互換性が維持されているため、*LAPACK ガイド*は LAPACK ルーチンおよび Fortran インタフェースの参考資料として使用できます。

注記 - LINPACK は Oracle Developer Studio パフォーマンスライブラリから削除されました。LINPACK ライブラリおよびドキュメントは、http://www.netlib.org から引き続き入手できます。

Oracle Developer Studio パフォーマンスライブラリの基礎となっているパフォーマンスライブラリルーチンの説明については、次の場所を参照してください。

LAPACK バージョン 3.5.0	http://www.netlib.org/lapack/
BLAS、レベル 1 - 3	http://www.netlib.org/blas/
FFTPACK バージョン 4	http://www.netlib.org/fftpack/

VFFTPACK バージョン 2.1	http://www.netlib.org/vfftpack/
スパース BLAS	http://www.netlib.org/sparse-blas/index.html
NIST (米国標準技術局) Fortran スパース BLAS	http://math.nist.gov/spblas/
SuperLU バージョン 3.0	http://crd.lbl.gov/~xiaoye/SuperLU/

Oracle Developer Studio パフォーマンスライブラリの機能

Oracle Developer Studio パフォーマンスライブラリでは多くのルーチンのシリアル速度が向上し、多くのルーチンが並列化されているため、パフォーマンスライブラリのルーチンはシリアルプロセッサとマルチプロセッサ (MP) の両方のプラットフォームでアプリケーションのパフォーマンスを高めることができます。Oracle Developer Studio パフォーマンスライブラリのルーチンでは、Netlib 基本ライブラリには存在しない SPARC、AMD、および Intel に固有の最適化も得られます。

Oracle Developer Studio パフォーマンスライブラリでは、Netlib 基本ライブラリに次の最適化と拡張が加えられます。

- Fortran 95 および C 言語のインタフェースをサポートする拡張機能
- 型への非依存性やコンパイル時チェックなどの、Fortran 95 言語の機能
- パフォーマンスライブラリ内のさまざまなライブラリにわたって一貫した API
- LAPACK 1.0、2.0、3.0、3.1.1、3.3.1、3.4.2、および 3.5.0 ライブラリとの互換性
- パフォーマンスの向上、および場合によっては精度の向上
- 特定の SPARC および x86/x64 命令セットアーキテクチャー向けの最適化
- 64 ビット対応の Oracle Solaris および Linux オペレーティング環境のサポート
- SPARC および x86/x64 プラットフォーム向けの並列処理コンパイラオプションのサポート
- マルチプロセッサハードウェアオプションのサポート

数学ルーチン

Oracle Developer Studio パフォーマンスライブラリのルーチンは、次のタイプの線形代数および数値問題を解くために使用されます。

- *ベクトルと行列の基本演算* – ベクトルおよび行列の積。平面回転。1、2、および無限大ノルム。ランク 1、2、k、および 2k 更新
- *連立 1 次方程式* – フルランクの連立 1 次方程式の求解、誤差限界の計算、シルベスター方程式の求解、計算解の改良、係数行列の均衡化

- *最小二乗問題* – フルランク、一般化線形回帰、ランク不足、線形等式制約付き
- *固有値問題* – 固有値、一般化固有値、固有ベクトル、一般化固有ベクトル、シュールベクトル、一般化シュールベクトル
- *行列の因子分解または分解* – SVD、一般化 SVD、QL と LQ、QR と RQ、コレスキー、LU、シュール、LDL^T、UDU^T、および CS 分解
- *サポート演算* – 条件数、インプレースまたはアウトオブプレースでの転置、逆数、決定因子、慣性、高精度反復改良
- *スパース行列* – 直接法を使用し、フィルインを低減する順序付けアルゴリズムおよびユーザー指定の順序付けを選択して、対称、構造的対称、または非対称な係数行列を解く
- 1 次元および 2 次元の畳み込みおよび相関
- 高速フーリエ変換、フーリエ解析とフーリエ合成、コサイン変換と 1/4 波コサイン変換、コサイン変換と 1/4 波サイン変換
- 複素ベクトル FFT、および 2 次元および 3 次元の FFT
- ソート演算
- CBLAS インタフェース

以前の LAPACK バージョンとの互換性

LAPACK に基づく Oracle Developer Studio パフォーマンスライブラリのルーチンは、LAPACK 3.5.0 で拡張された機能および改良されたアルゴリズムをサポートしていますが、LAPACK 1.x、LAPACK 2.x、LAPACK 3.x、および LAPACK 4.x ライブラリとも完全に互換性を保っています。以前の LAPACK バージョンとの互換性を維持しているため:

- サブルーチン名または引数リストの変更によるリンクエラーが減少します。
- 生成される結果は、以前のバージョンの LAPACK で生成される結果と必ず一致します。
- 引数リストの相違によるプログラム終了が最小限に抑えられます。

Oracle Developer Studio パフォーマンスライブラリの使用開始

このセクションでは、Oracle Developer Studio パフォーマンスライブラリのルーチンを使用するアプリケーションをコンパイルする場合に使用する、もっとも基本的なコンパイラオプションについて説明します。

Oracle Developer Studio パフォーマンスライブラリを使用するには、次のいずれかのコマンドを入力します。

x86/x64 および SPARC プラットフォームの場合:

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

SPARC プラットフォームの場合:

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

x86/64 プラットフォームの場合、-xmemalign=8s は無視されるため、省略できます。

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

Oracle Developer Studio パフォーマンスライブラリと静的にリンクするには、-staticlib=sunperf をコマンド行に追加します。

Oracle Developer Studio パフォーマンスライブラリのルーチンは -dalign でコンパイルされているため、プログラムのいずれかのルーチンが Oracle Developer Studio パフォーマンスライブラリを呼び出す場合は、すべての Fortran ファイルのコンパイルにこのオプションを使用するようにしてください。SPARC プラットフォームの場合、Oracle Developer Studio パフォーマンスライブラリのルーチンを呼び出す C および C++ ユーザーコードは、オプション -xmemalign=8s を指定してコンパイルするようにしてください。-xmemalign=8s を使用できない場合は、回避策としてトラップ 6 を有効にすると、パフォーマンスは低下しますが境界不整列なデータを許容できます。詳細については、16 ページの「SPARC プラットフォームでのトラップ 6 の有効化」を参照してください。

x86/x64 プラットフォームではデータの境界整列の制限はありませんが、境界不整列なデータではメモリー転送を正しく処理するための追加手順が必要になることがあり、それによってパフォーマンスが低下する可能性があります。

-library=sunperf オプションは、Fortran の実行時ライブラリやマイクロタスキングライブラリといった追加のコンパイラライブラリおよびシステムライブラリをインクルードし、生成された実行可能ファイルまたは共有ライブラリの実行時検索パスを設定します。

要約すると、次のオプションを使用してください。

- すべての Fortran ファイルのコンパイル時には -dalign。

  SPARC プラットフォームでは -xmemalign=8s。または、トラップ 6 を有効にします
- コンパイルとリンクには同じコマンド行オプションを使用します
- -library=sunperf または -library=sunperf -staticlib=sunperf

アプリケーションのパフォーマンスを最適化する追加のオプションについては、28 ページの「コンパイルについて」および第4章「並列処理」を参照してください。

▼ SPARC プラットフォームでのトラップ 6 の有効化

SPARC プラットフォームでは、データの境界不整列によって障害が発生する可能性があるため、アプリケーションを -dalign または -xmemalign=8s でコンパイルできない場合は、境界不整列なデータのハンドラを提供するためにトラップ 6 を有効にします。SPARC プラットフォームでトラップ 6 を有効にするには、次の手順を実行します。

1. **このアセンブリコードを `trap6_handler.s` というファイルに配置します。**

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

2. **`trap6_handler.s` をアセンブルします。**

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

`fbe` は、アセンブリ言語ソースファイルからオブジェクトファイルを生成するコマンドです。

Oracle Developer Studio パフォーマンスライブラリから呼び出される最初の並列化可能サブルーチンが、`trap6_handler_` という名前のルーチンを呼び出します。`trap6_handler_` が指定されていない場合、Oracle Developer Studio パフォーマンスライブラリは何もしないデフォルトハンドラを呼び出します。境界不整列なデータのハンドラが指定されていないと、致命的なトラップが発生します。

3. **コマンド行に `trap6_handler.o` を含めます。**

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

♦ ♦ ♦ 第 2 章

2 Oracle Developer Studio パフォーマンスライブラリの使用

この章では、Fortran 95 または C で記述されたアプリケーションの実行速度を向上させるために Oracle Developer Studio パフォーマンスライブラリを使用する方法について説明します。多くのアプリケーションでは、Oracle Developer Studio パフォーマンスライブラリを使用することにより、ソースコードの変更や再コンパイルを行わずにパフォーマンスを向上させることができます。ただし、Oracle Developer Studio パフォーマンスライブラリで最高のパフォーマンスを得るには、アプリケーションに何らかの変更が必要になることがあります。

アプリケーションパフォーマンスの向上

以降のセクションでは、ソースコードの変更や再コンパイルを行わずに Oracle Developer Studio パフォーマンスライブラリのルーチンを使用する方法について説明します。

Oracle Developer Studio パフォーマンスライブラリのルーチンによるルーチンの置き換え

多くのアプリケーションには、LAPACK や BLAS などの Netlib 基本ライブラリが 1 つ以上使用されています。Oracle Developer Studio パフォーマンスライブラリではこれらのライブラリと同じインタフェースおよび機能が維持されているため、Netlib の基本ルーチンを Oracle Developer Studio パフォーマンスライブラリのルーチンに置き換えることができます。Oracle Developer Studio パフォーマンスライブラリのルーチンは、対応する Netlib ルーチンやほかのベンダーによって提供される同様のルーチンより高速化できるため、アプリケーションのパフォーマンスが向上します。

ほかのライブラリのパフォーマンス向上

多くの商用数学ライブラリは、BLAS および LAPACK の汎用ルーチンを核として構築されています。アプリケーションが別のライブラリの独自インタフェースに依存しているためその

ライブラリ全体の置き換えはできない場合、そのライブラリで使用されている BLAS および LAPACK ルーチンを Oracle Developer Studio パフォーマンスライブラリの BLAS および LAPACK ルーチンに置き換えることができます。コアルーチンの置き換えにはコード変更は必要ないため、独自のライブラリ機能は引き続き使用でき、ライブラリ内のその他のルーチンは変更せずに残すことができます。

ツールを使用したコード再構築

Oracle Developer Studio パフォーマンスライブラリのルーチンを直接使用しない一部のライブラリは、既存のコードを Oracle Developer Studio パフォーマンスライブラリのコードに置き換える自動コード再構築ツールを使用して変更できます。たとえば、ソースからソースへの変換ツールは、既存の BLAS コードの構造を、Oracle Developer Studio パフォーマンスライブラリの BLAS ルーチンの呼び出しに置き換えることができます。これらの変換ツールは、ユーザーによって記述された多くの行列乗算も認識し、それらを Oracle Developer Studio パフォーマンスライブラリの行列乗算サブルーチンの呼び出しに置き換えることができます。

Fortran インタフェース

Oracle Developer Studio パフォーマンスライブラリには、標準の LAPACK および BLAS ライブラリおよび既存のコードとの互換性を維持するために `f95` インタフェースおよびレガシー `f77` インタフェースが含まれています。Oracle Developer Studio パフォーマンスライブラリの `f95` およびレガシー `f77` インタフェースでは次の規則が使用されます。

- 引数はすべて、参照によって渡されます。
- 1 つの呼び出し内では引数の型に一貫性が必要です。たとえば、同じ呼び出しに `REAL*8` パラメータと `REAL*4` パラメータを混ぜないでください。
- 配列は列方向に格納されます。
- 標準的な Fortran の慣行に従って、インデックスは 1 オリジンです。

Oracle Developer Studio パフォーマンスライブラリのルーチンを呼び出すときは、次の情報に留意してください。

- Fortran 95 の `INTERFACE` 文でサブルーチンのプロトタイプ宣言をしないでください。代わりに、`USE SUNPERF` 文を使用します。
- Oracle Developer Studio パフォーマンスライブラリのルーチンを呼び出すルーチンをコンパイルする場合は、`-ext_names=plain` を使用しないでください。

Fortran 95 で使用するための Fortran SUNPERF モジュール

Oracle Developer Studio パフォーマンスライブラリでは、Fortran 95 プログラムでの使いやすさを向上させる Fortran モジュールが提供されています。このモジュールを使用するには、Fortran 95 コードに次の行を含めます。

```
USE SUNPERF
```

USE 文は、PROGRAM または SUBROUTINE 文を除き、コード内でほかのすべての文より前に置く必要があります。

SUNPERF モジュールは、呼び出しシーケンスを簡素化するインタフェースを持ち、次の機能を提供します。

- *型への非依存性* – Oracle Developer Studio パフォーマンスライブラリでは、データ引数の型を自動的に認識するインタフェースがサポートされているため、型に依存する接頭辞 (S、D、C、または Z) が不要になります。FORTRAN 77 のルーチンでは、名前の一部として型を指定する必要があります。たとえば、DGEMM は倍精度の行列乗算、SGEMM は単精度の行列乗算です。GEMM を Fortran 95 インタフェースで呼び出すと、Fortran が渡された引数から型を推測します。GEMM に単精度引数を渡すと、SGEMM を指定した場合と同じ結果になり、倍精度引数を渡すと、DGEMM を指定した場合と同じ結果になります。たとえば、CALL DSCAL(20,5.26D0,X,1) を CALL SCAL(20, 5.26D0, X, 1) に変更できます。
- *コンパイル時チェック* – FORTRAN 77 では、特定のルーチンにどのような引数を渡すべきかをコンパイラが判断することは通常できません。Fortran 95 では、USE SUNPERF 文を使用することにより、コンパイラは Oracle Developer Studio パフォーマンスライブラリの各ルーチンに渡すべきそれぞれの引数の数、型、サイズ、および形状を判断できます。コンパイラは、コンパイル時に呼び出しと期待される値とを照合し、エラーを表示できます。
- *64 ビット整数のサポート* – Oracle Developer Studio パフォーマンスライブラリで提供されている 64 ビットインタフェースを使用する場合は、整数引数を 64 ビットに上位変換し、ルーチン名に _64 を付加する必要があります。SUNPERF モジュールを使用すると、64 ビット整数が自動的に認識されるため、次のコード例に示すように、ルーチン名に _64 を付加する必要がなくなります。

```
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
```

Oracle Developer Studio パフォーマンスライブラリの 64 ビットインタフェースの詳細な使用方法については、29 ページの「64 ビットに対応したオペレーティング環境用のコードのコンパイル」を参照してください。

`sunperf.mod` ファイルは `-dalign` でコンパイルされているため、`USE SUNPERF` 文を含んでいるコードはすべて `-dalign` でコンパイルする必要があります。コードを `-dalign` でコンパイルしないと、次のエラーが発生します。

```
 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 の例

単一プロセッサアプリケーションのパフォーマンスを向上させるには、Oracle Developer Studio パフォーマンスライブラリルーチンの呼び出しに置き換えることができるアプリケーション内のコード構文を特定します。マルチプロセッサアプリケーションのパフォーマンスは、並列化の機会を特定することによって向上させることができます。

アプリケーションのパフォーマンスを向上させるために、Oracle Developer Studio パフォーマンスライブラリのルーチンを使用するようにコードを変更する場合は、Oracle Developer Studio パフォーマンスライブラリのルーチンとまったく同じ機能を持つコードブロックを特定します。次のコード例は行列とベクトルの積 y ← Ax + y で、これは `DGEMV` サブルーチンで置き換えることができます。

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

ほかにも、コードブロックが Oracle Developer Studio パフォーマンスライブラリの複数の呼び出しに相当する場合や、コードの一部を Oracle Developer Studio パフォーマンスライブラリのルーチンで置き換えられる場合があります。次のコード例について考えてみましょう。

```
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
```

このコード例は、Oracle Developer Studio パフォーマンスライブラリのルーチン `DGER` を使用して次のように書き直すことができます。

```
      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)
```

同じコード例を、Fortran 95 固有の文を使用して次のように書き直すこともできます。

```
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)
```

V2 内の負の数値を 0 に置き換えるコードは、それと同等のものが Oracle Developer Studio パフォーマンスライブラリにはないため、外側のループの外へ出されます。そのコードを独自のループに移動すると、ループの残りは一般行列 x のランク 1 更新であり、これは BLAS の DGER ルーチンで置き換えることができます。

パフォーマンスの向上量は、Oracle Developer Studio パフォーマンスライブラリのルーチンが使用するデータによっても異なります。たとえば、V2 に負または 0 の値が多数含まれていると、時間の大部分がランク 1 更新以外に費やされることがあります。この場合、コードを DGER の呼び出しに置き換えても、パフォーマンスは向上しない可能性があります。

使用する Oracle Developer Studio パフォーマンスライブラリルーチンは、ほかのループインデックスの評価によっても異なります。たとえば、K への参照がループインデックスである場合、上記のコード例のループはより大きいコード構造の一部になっている可能性があり、DGEMV または DGER のループを何らかの形式の行列乗算に変換できることがあります。その場合、DGER 呼び出しのループを使用するよりも、行列乗算ルーチンを 1 回呼び出す方がパフォーマンスを向上させることができます。

Oracle Developer Studio パフォーマンスライブラリのルーチンはすべて MT セーフ (マルチスレッドに対して安全) なので、Oracle Developer Studio パフォーマンスライブラリのルーチンの呼び出しが含まれているループを自動並列化コンパイラで並列化すると、マルチプロセッサプラットフォームでパフォーマンスを向上させることができます。

次のコード例は、Oracle Developer Studio パフォーマンスライブラリのルーチンに自動並列化コンパイラの並列化ディレクティブを組み合わせた例を示しています。

```
      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 パフォーマンスライブラリには、帯行列にベクトルを乗算する DGBMV という名前のルーチンがあります。このルーチンを適切な構造のループに配置することによって、Oracle Developer Studio パフォーマンスライブラリのルーチンを使用して帯行列に行列を乗算できます。ループ内にサブルーチン呼び出しがあると並列化が禁止されるため、コンパイラはこのループをデフォルトでは並列化しません。ただし、Oracle Developer Studio パ

フォーマンスライブラリのルーチンは MT セーフなので、並列化ディレクティブを使用して、このループを並列化するようにコンパイラに指示できます。

コンパイラディレクティブは、通常は並列化可能でないサブルーチン呼び出しが含まれているループを並列化する場合にも使用できます。たとえば、一部の連立 1 次方程式ソルバーは、ベンダーによっては MT セーフでないコードでルーチンが実装されている場合があるため、そのようなソルバーの呼び出しが含まれているループは、通常は並列化できません。連立 1 次方程式ソルバーのエキスパートドライバ (名前が `SVX` または `SVXX` で終わるルーチン) の呼び出しが含まれているループは、通常はほかの実装の LAPACK では並列化できません。Oracle Developer Studio パフォーマンスライブラリの LAPACK の実装では、そのような呼び出しを含むループの並列化が可能なので、マルチプロセッサプラットフォームのユーザーは、このようなループの並列化によってパフォーマンスを向上させることができます。

C インタフェース

Oracle Developer Studio パフォーマンスライブラリのルーチンは、FORTRAN 77、Fortran 95、または C プログラム内から呼び出すことができます。ただし、C プログラムでは引き続き FORTRAN 77 呼び出しシーケンスを使用する必要があります。

Oracle Developer Studio パフォーマンスライブラリには、LAPACK、BLAS、FFTPACK、VFFTPACK、SPARSE BLAS、および SPSOLVE の各ルーチンのネイティブ C インタフェースが含まれています。Oracle Developer Studio パフォーマンスライブラリの C インタフェースには次の機能があります。

- 関数名は C 名です
- 関数のインタフェースは C の規則に従います
- C インタフェースには、C 関数にとって冗長または不要な引数は含まれません

次の例では、`DGBCON` ルーチンについて標準の LAPACK Fortran インタフェースと Oracle Developer Studio パフォーマンスライブラリの C インタフェースを比較しています。

```
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)
```

引数の名前は同じで、同じ名前の引数の基底型は同じです。`NORM` や `N` など、入力値としてのみ使用されるスカラー引数は、C バージョンでは値によって渡されます。値を返すために使用される配列とスカラーは、参照によって渡されます。

Oracle Developer Studio パフォーマンスライブラリの C インタフェースは、Netlib で提供されている、標準ライブラリの `f2c` 変換である CLAPACK を改良します。たとえば、Fortran コンパイラとの互換性を維持するためのアンダースコアがすべての CLAPACK ルーチンの末尾には付加され、多くの場合、オブジェクト (`.o`) ファイル内のルーチン名の末尾にアンダースコア

が付加されます。Oracle Developer Studio パフォーマンスライブラリの C インタフェースには末尾のアンダースコアは必要ありません。

Oracle Developer Studio パフォーマンスライブラリ C インタフェースでは次の規則が使用されます。

- 入力専用のスカラーは、参照によってではなく値によって渡されます。複素数および倍精度複素数の引数は、C ではスカラー型として実装されていないため、スカラーとは見なされません。
- 複素数のスカラーは、構造体または長さ 2 の配列として渡すことができます。
- 引数の型は、C での型変換後も一致する必要があります。たとえば、C コンパイラは引数を自動的に倍精度に上位変換する場合があるため、単精度の実数値を渡すときには注意してください。
- 配列は列方向に格納されます。Fortran のプログラマにとって、これは配列を格納する自然な順序です。C のプログラマにとって、これは通常作業している順序を転置したものです。ドキュメントおよびマニュアルページで、行は列のことを指し、列は行のことを指します。
- 配列のインデックスは、C のように 0 オリジンではなく、Fortran の規則に従って 1 オリジンです。

  たとえば、`IDAMAX` の Fortran インタフェースは、C プログラムからは `idamax_` としてアクセスし、1 を返してベクトル内の最初の要素を示します。`idamax` の C インタフェースは、C プログラムからは `idamax` としてアクセスし、同様に 1 を返してベクトル内の最初の要素を示します。関数の戻り値、置換ベクトル、およびベクトルまたは配列のインデックスが使用される任意の場所で、この規則が守られます。

注記 - 一部の Oracle Developer Studio パフォーマンスライブラリルーチンは内部的に `malloc` を使用するため、Oracle Developer Studio パフォーマンスライブラリおよび `sbrk` を呼び出すユーザーコードは、正しく動作しない可能性があります。

SPARC バージョンの Oracle Developer Studio パフォーマンスライブラリは、グローバル整数レジスタ `%g2`、`%g3`、および `%g4` を 32 ビットモードで使用し、`%g2` から `%g5` までを 64 ビットモードでスクラッチレジスタとして使用します。ユーザーコードでは、これらのレジスタを一時的な格納に使用して Oracle Developer Studio パフォーマンスライブラリのルーチンを呼び出すことは避けてください。Oracle Developer Studio パフォーマンスライブラリのルーチンがこれらのレジスタを使用すると、データが上書きされます。

C の例

ユーザー作成のコードシーケンスを Oracle Developer Studio パフォーマンスライブラリルーチンの呼び出しに変換すると、アプリケーションのパフォーマンスが向上します。LAPACK から取った次のコード例で 1 つの例を示します。

```
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];
}
```

このコード例とまったく同じ機能を持つ Oracle Developer Studio パフォーマンスライブラリルーチンはありません。ただし、次のコード例のように、Oracle Developer Studio パフォーマンスライブラリルーチン `isamax` の呼び出しに置き換えることによってコードを高速化できます。

```
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];
```

Oracle Developer Studio パフォーマンスライブラリのネイティブ C `isamax` ルーチンを呼び出している上記のコード例と、CLAPACK の `isamax` ルーチンを呼び出している次のコード例で、相違を比較してください。

```
/* 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 アプリケーションの最適化

この章では、コンパイラオプションとリンクオプションを使用して、次に対してアプリケーションを最適化する方法について説明します。

- 特定の命令セットアーキテクチャー
- 32 ビットおよび 64 ビットに対応したオペレーティング環境

32 ビット環境と 64 ビット環境の比較

次の表は、32 ビットと 64 ビットのオペレーティング環境を比較したものです。これらの項目については、以降のセクションで詳細に説明されています。

表 1 32 ビットと 64 ビットのオペレーティング環境の比較

	32 ビット (ILP 32)	64 ビット (LP64)
`-xarch` (SPARC プラットフォーム)	`sparcvis`、`sparcvis2`、`sparcfmaf`	`sparcvis`、`sparcvis2`、`sparcfmaf`
`-xarch` (x86 プラットフォーム)	`generic`、`sse2`	`sse2`
アドレス指定	`-m32`	`-m64`
Fortran の整数	`INTEGER`、`INTEGER*4`	`INTEGER*8`
C の整数	`int`	`long`
浮動小数点	`S/D/C/Z`	`S/D/C/Z`
API	ルーチンの名前	ルーチンの名前と接尾辞 `_64`

Oracle Developer Studio パフォーマンスライブラリの使用

Oracle Developer Studio パフォーマンスライブラリは、このリリースで提供されている `f95` コンパイラを使用してコンパイルされました。Oracle Developer Studio パフォーマンスライブラリのルーチンは、`-dalign`、`-xparallel` を使用してコンパイルされました。

Fortran プログラムのリンク

プログラムをリンクするときは、`-dalign -library=sunperf` と、コンパイル時に使用されたものと同じコマンド行オプションを使用してください。

Oracle Developer Studio パフォーマンスライブラリは、ほかのライブラリをリンクするために使用される `-l` スイッチではなく、次のように `-library` スイッチでアプリケーションにリンクされます。

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

C および C++ プログラムのリンク

プログラムをリンクするときは、`-library=sunperf` と、コンパイル時に使用されたものと同じコマンド行オプションを使用してください。SPARC システムでコンパイルする場合は、次のように `-xmemalign=8s` オプションを含めます。`-xmemalign=8s` オプションは x86 および x64 プラットフォームでは無視されます。

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

`-dalign` または `-xmemalign=8s` をコンパイルに使用できない場合は、16 ページの「SPARC プラットフォームでのトラップ 6 の有効化」の説明に従って、トラップ 6 ハンドラを指定します。

コンパイルについて

最適なパフォーマンスを得るには、もっとも適切な `-xarch` オプションでコンパイルしてください。リンク時には、コンパイル時に使用されたものと同じ `-xarch` オプションを使用して、特定の命令セットアーキテクチャー用に最適化された Oracle Developer Studio パフォーマンスライブラリのバージョンを選択します。

注記 - 命令セットに固有の最適化オプションを使用すると、選択した命令セットアーキテクチャー上のアプリケーションのパフォーマンスは向上しますが、コードの移植性が制限されます。

さまざまな `-xarch` オプションの詳細については、『Oracle Developer Studio 12.5: Fortran ユーザーズガイド』または『Oracle Developer Studio 12.5: C ユーザーズガイド』を参照してください。

SPARC および x86 命令セットアーキテクチャーの `-xarch` の値は、`fbe`、`cc`、`CC`、および `f95` のマニュアルページにも示されています。

64 ビットに対応したオペレーティング環境用のコードのコンパイル

64 ビット対応のオペレーティング環境用のコードをコンパイルするには、-m64 を使用し、すべての整数引数を 64 ビット引数に変換します。64 ビットルーチンには 64 ビット整数を使用する必要があります。

Oracle Developer Studio パフォーマンスライブラリでは、32 ビットおよび 64 ビットのインタフェースが提供されています。64 ビットインタフェースを使用するには:

- **Oracle Developer Studio パフォーマンスライブラリのルーチン名を変更します。**C および Fortran 95 のコードの場合は、Oracle Developer Studio パフォーマンスライブラリのルーチンの名前に _64 を付加します (たとえば、rfftf_64 や CFFTB_64)。USE SUNPERF 文を含む Fortran 95 コードの場合、DGEMM などの固有インタフェースには接尾辞 _64 は厳密には必須ではありません。ただし、GEMM などの汎用インタフェースには接尾辞 _64 は必須です。
- **整数を 64 ビットに上位変換します。**倍精度変数、および倍精度複素数型変数の実数部と虚数部は、すでに 64 ビットです。整数のみが 64 ビットに上位変換されます。

64 ビット整数引数

これらの追加の 64 ビット整数インタフェースは、-m64 でリンクする場合にのみ使用できます。32 ビットオペレーティング環境用にコンパイルされたコード (-m32) で 64 ビット整数インタフェースを呼び出すことはできません。。

64 ビット整数インタフェースを直接呼び出すには、標準のライブラリ名に接尾辞 _64 を付加します。たとえば、daxpy() の代わりに daxpy_64() を使用します。

ただし、64 ビット整数インタフェースを間接的に呼び出す場合は、Oracle Developer Studio パフォーマンスライブラリのルーチンの名前に _64 を付加しないでください。パフォーマンスライブラリのルーチンを呼び出すと 32 ビットラッパーが呼び出され、このラッパーが 32 ビット整数を 64 ビット整数に上位変換し、64 ビットルーチンを呼び出したあと、64 ビット整数を 32 ビット整数に下位変換します。

最適なパフォーマンスを得るには、ルーチン名に _64 を付加してルーチンを直接呼び出してください。

C プログラムの場合は、int 引数の代わりに long を使用します。次のコード例では、64 ビット整数インタフェースを直接呼び出しています。

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

次のコード例では、64 ビット整数インタフェースを間接的に呼び出しています。

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

Fortran プログラムの場合は、すべての整数引数に 64 ビット整数を使用します。次の方法を使用して整数引数を 64 ビットに変換できます。

- バイトサイズが明示的に宣言されていないすべての整数とリテラル整数定数を 32 ビットから 64 ビットに上位変換するには、`-xtypemap=integer:64` を指定してコンパイルします。
- 特定の整数の宣言を上位変換するには、`INTEGER` または `INTEGER*4` を `INTEGER*8` に変更します。
- 整数リテラル定数を上位変換するには、定数に `_8` を付加します。

次のコード例について考えてみましょう。

```
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` 引数は、32 ビット環境では使用できません。32 ビットライブラリ、`v8plusa`、`v8plusb` のルーチンを 64 ビット引数で呼び出すことはできません。ただし、64 ビットルーチンを 32 ビット引数で呼び出すことはできます。

`-xtypemap` でコンパイルされていない Fortran 95 コードに定数を渡す場合は、リテラル定数に `_8` を付加して上位変換を発生させます。たとえば、Fortran 95 を使用している場合は、`CALL DSCAL(20,5.26D0,X,1)` を `CALL DSCAL(20_8,5.26D0,X,1_8)` に変更します。この例では、ルーチン名に `_64` が付加されていないため、コードに `USE SUNPERF` が含まれていると見なします。

次のコード例では、Fortran 95 から `CAXPY` を 32 ビット引数で呼び出しています。

```
      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)
```

次のコード例では、(`USE SUNPERF` 文を含まない) Fortran 95 から `CAXPY` を 64 ビット引数で呼び出しています。

```
      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)
```

64 ビット引数を使用するときは、USE SUNPERF 文が使用されていない場合は、ルーチン名に _64 を付加する必要があります。

次の Fortran 95 のコード例では、CAXPY を 64 ビット引数で呼び出しています。

```
      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)
```

C のルーチンでは、long のサイズは -m32 でコンパイルすると 32 ビット、-m64 でコンパイルすると 64 ビットになります。次のコード例では、dgbcon ルーチンを 32 ビット引数で呼び出しています。

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

次のコード例では、dgbcon ルーチンを 64 ビット引数で呼び出しています。

```
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

4 並列処理

この章では、共有メモリー並列化を使用したマルチプロセッサ環境で Oracle Developer Studio パフォーマンスライブラリを使用する方法について説明します。

実行時の問題

実行時に、コンパイラの並列化を使用して実行している場合、Oracle Developer Studio パフォーマンスライブラリはコンパイラと同じスレッドのプールを使用します。スレッドごとのスタックサイズはすべてのプラットフォームで少なくとも 8M バイトに設定する必要があります。 これは、`STACKSIZE` または `OMP_STACKSIZE` 環境変数 (K バイト単位) で行います。両方は使用できません。同時に 2 つの環境変数を使用して、2 つの値が異なる場合、プログラムはエラーメッセージを表示して停止します。

スレッドごとのスタックサイズを 8M バイトに設定するには:

```
my_host% setenv STACKSIZE 8192
```

POSIX または Oracle Solaris スレッドで実行するプログラムには `STACKSIZE` 環境変数を設定する必要はありません。この場合、パフォーマンスライブラリルーチンを呼び出すユーザー作成スレッドには、少なくとも 8M バイトのスタックサイズが必要です。パフォーマンスライブラリルーチンに十分なスタックサイズを提供できないと、スタックオーバーフローの問題が発生する可能性があります。スタックオーバーフローの問題の現象には、診断が難しくなる可能性がある実行時の障害が含まれます。ユーザー作成スレッドのスタックサイズの設定の詳細については、POSIX スレッド用の `pthread_create`(3C)、`pthread_attr_init`(3C)、および `pthread_attr_setstacksize`(3C) のマニュアルページまたは Oracle Solaris スレッド用の `thr_create`(3C) を参照してください。

ヒント - コアダンプの診断の問題が発生している場合は、スタックサイズを最小以上に増やしてみてください。

並列度

Oracle Developer Studio パフォーマンスライブラリ内の選択されたルーチンは、*OpenMP Fortran Application Program Interface* のコンパイラディレクティブ、ライブラリルーチン、および環境変数を使用して並列化されます。これらのルーチンで並列で使用するスレッドの数は、実行時に設定する環境変数 OMP_NUM_THREADS によって制御します。環境変数 PARALLEL を設定することもできますが、両方を設定する場合は、それらが同じ値を持つ必要があり、そうしないと実行時に致命的なエラーが発生します。両方の環境変数は、ユーザーコード内で Oracle Developer Studio パフォーマンスライブラリルーチン USE_THREADS または、OpenMP ルーチン OMP_SET_NUM_THREADS を呼び出すことによってオーバーライドできます。

ユーザーコードは、次を実行して並列化できます。

- 環境変数 OMP_NUM_THREADS を 1 より大きい値に設定します
- OpenMP API のディレクティブなどのコンパイラ並列ディレクティブを使用します
  該当するコンパイラフラグ -xopenmp=parallel または -xautopar を使用します

Oracle Developer Studio パフォーマンスライブラリルーチンは、次の条件が満たされている場合に並列で実行します。

- OMP_NUM_THREADS が 1 より大きい値に設定されています
- ルーチンが並列領域から呼び出されていません

Oracle Developer Studio パフォーマンスライブラリはその並列化で OpenMP ディレクティブを採用しており、入れ子並列化をサポートしていません。ユーザーコードが上記のように並列化されており、Oracle Developer Studio パフォーマンスライブラリルーチンを呼び出している場合、ルーチンが並列領域から呼び出されていることが検出された場合、ルーチンは順次実行されます。それ以外の場合、ルーチンは並列で実行されます。

POSIX または Oracle Solaris スレッドは、ユーザーコードの並列の選択された領域で実行するように作成することもできます。パフォーマンスライブラリルーチンがこの並列モデルで呼び出されると、ルーチンは並列領域から呼び出されていることを検出できません。そのため、環境変数 OMP_NUM_THREADS が 1 に設定されているか、または設定されていないか、またはユーザーコードの適切な場所で USE_THREADS (3P) の呼び出しが行われる必要があります。そうしないと、未定義の結果で入れ子並列化が行われます。

たとえば、次のコードセグメントを含むプログラムが -xopenmp=parallel とリンクされ、OMP_NUM_THREADS が 4 に設定されている場合、ループが並列で実行され、DGEMM の 4 つのインスタンスが同時に実行されます。ただし、サポートされる並列化のレベルは 1 つだけであるため、各 DGEMM インスタンスは順次実行されます。

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

次のコード例では、プログラムが -xautopar とリンクされていない場合、ループは並列化されませんが、DGEMM の各インスタンスは、4 つのスレッドで実行されます。

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

次のコードセグメントを含むプログラムが -xopenmp=parallel とリンクされており、OMP_NUM_THREADS が 1 より大きい値に設定されている場合、示されている領域は、1 つのスレッドで実行されます。ただし、各 DGEMM 呼び出しは、OMP_NUM_THREADS スレッドによって実行されます。

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

次のコード例では、実行に使用可能な OpenMP スレッド数に関係なく、最大で 2 方向の並列化になります。並列化のレベルは 1 つだけ存在し、それらは 2 つのセクションです。DGEMM 呼び出し内のそれ以上の並列化は抑制されます。

```
!$OMP PARALLEL SECTIONS
!$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
```

同期メカニズム

POSIX/Oracle Solaris スレッドモデルの 1 つの特性は、実行中のアプリケーションのバインドされているスレッドがアイドル状態のときに CPU を放棄することであり、そのために、共有 (登録超過) 環境で、優れたスループットとリソース使用状況を実現します。デフォルトで、コンパイラ並列化コードのバインドされているスレッドは、アイドル状態のときにスピン待機するため、システムに CPU リソースを競争するほかのアプリケーションがある場合に、ほぼ最適なスループットを発揮できます。この場合、環境変数 SUNW_MP_THR_IDLE を使用して、並列ジョブのそのシェアの完了後のスレッドの動作を制御できます。

```
my_host% setenv SUNW_MP_THR_IDLE value
```

ここで、*value* は、spin または sleep *n* s または sleep *n* ms のいずれかを指定でき、spin がデフォルトです。

sleep は、*n* 単位のスピン待機後にスレッドをスリープさせます。待機単位は秒 (s、デフォルトの単位) またはミリ秒 (ms) です。引数なしの sleep は並列タスクの完了後すぐにスレッドをスリープさせます。SUNW_MP_THR_IDLE に不正な値が含まれているか、または設定されていない場合、spin がデフォルトとして使用されます。

次の設定では、スレッドがそれぞれスピン待機 (デフォルトの動作) するか、スリープする前に 2 秒間スピンするか、またはスリープする前に 100 ミリ秒間スピンします。Oracle Developer Studio パフォーマンスライブラリルーチンを使用しても、コードのスピン待機動作は変更されません。

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

並列処理の例

このセクションでは、コンパイルおよびリンクオプションとともに OMP_NUM_THREADS 環境変数を使用して、順次または並列で実行するコードを作成する方法を説明します。

シリアルアプリケーションを作成するには:

- 1 つ以上の Oracle Developer Studio パフォーマンスライブラリルーチンを呼び出します
- -library=sunperf とリンクし、コマンド行の末尾にフラグを配置します。-xopenmp=parallel または -xautopar とコンパイルまたはリンクしないでください
- OMP_NUM_THREADS 環境変数を設定解除するか、1 に設定します

次の例は、共有 Oracle Developer Studio パフォーマンスライブラリ libsunperf.so とコンパイルおよびリンクする方法を示しています。

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

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

複数のプロセッサで実行する並列アプリケーションを作成するには:

- 1 つ以上の Oracle Developer Studio パフォーマンスライブラリルーチンを呼び出します
- コンパイルコマンドとリンクコマンドで同じ並列化オプション (-xopenmp=parallel または -xautopar) を使用します。
- -library=sunperf とリンクし、コマンド行の末尾にフラグを配置します。
- 実行可能ファイルを実行する前に、OMP_NUM_THREADS を使用可能なプロセッサの数に設定します

たとえば、24 個のプロセッサを使用するには、次のコマンドを入力します。

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

```
my_host% ./a.out
```

前の例では、Oracle Developer Studio パフォーマンスライブラリルーチンを並列で実行できますが、ユーザーコード my_app.f のどの部分も並列で実行されません。コンパイラが my_app.f の並列化を試みるには、コンパイル行で -xopenmp=parallel、または -xautopar のいずれかが必要です。

```
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 行列の操作

ほとんどの行列は、格納領域と計算時間の両方を節約する方法で格納できます。Oracle Developer Studio パフォーマンスライブラリでは、次の格納スキームが使用されます。

- バンド格納
- パック格納

Oracle Developer Studio パフォーマンスライブラリで処理される行列は、次の 4 形式のいずれかです。

- 一般
- 三角
- 対称
- 三重対角

格納スキームと行列タイプについては、以降のセクションで説明されています。

行列の格納スキーム

通常の形式で格納された配列を操作する Oracle Developer Studio パフォーマンスライブラリの一部のルーチンには、これらの特殊な格納形式を活用する対応するルーチンがあります。たとえば、DGBMV はバンド格納の一般行列とベクトルの積を生成し、DTPMV はパック格納の三角行列とベクトルの積を生成します。

バンド格納

バンド化された行列は、行列の *j* 番目の列が Fortran 配列の *j* 番目の列に対応するように格納されます。

次のコードは、一般配列内にバンド化された一般行列をバンド格納モードにコピーします。

```
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
```

バンド化された行列を格納するこの方法には、LAPACK および BLAS で使用される格納方法との互換性があります。

パック格納

パック化されたベクトルは、三角、対称、またはエルミート行列の代替表現です。配列は、要素を 1 列ずつ順にベクトルに格納することによって、ベクトルにパック化されます。単位対角行列の場合のように、対角要素の値が既知であっても、対角要素のための領域は常に予約されます。

上側の三角形が配列 `A` に一般格納された上三角行列または対称行列は、次のように転送して配列 `AP` にパック格納できます。このコードは、LAPACK ルーチン `DTPTRI` のコメントブロックから取得したものです。

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

同様に、下側の三角形が配列 `A` に一般格納された下三角行列または対称行列は、次のように転送して配列 `AP` にパック格納できます。

```
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
```

Rectangular Full Packed 形式

Rectangular Full Packed (RFP) 行列は、三角および対称行列を格納するためのデータ形式です。これは、格納領域全体を使用する標準パック形式の配列に、レベル 3 BLAS

を使用して高パフォーマンスを組み合わせます。詳細については、『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)』および Rectangular Full Packed 形式を使用するルーチンのマニュアルページの「詳細」セクションを参照してください。

行列のタイプ

一般行列はもっとも一般的なタイプで、Oracle Developer Studio パフォーマンスライブラリのほとんどの演算は一般行列で行われます。多くの場合、その他のタイプの行列を操作するルーチンがあります。たとえば、DGEMM は 2 つの一般行列の積を計算し、DTRMM は三角行列と一般行列の積を計算します。

一般行列

一般行列は、行列の要素と配列の要素が 1 対 1 で対応するように格納されます。行列 A の要素 A*ij* は、対応する配列 A の要素 A(*I*,*J*) に格納されます。一般行列の場合、各要素が明示的に格納されるため、特別な格納スキームはありません。対照的に、一般帯行列の場合は、ゼロでない上対角、対角、および下対角の要素のみが格納されます。次の例は、一般帯行列が 2 次元配列に格納される仕組みを示しています。配列の x で示された場所はアクセスされません。

一般帯行列	パック格納の一般帯行列
$\begin{bmatrix} a_{11} & a_{12} & a_{13} & 0 & 0 \\ a_{21} & a_{22} & a_{23} & a_{24} & 0 \\ 0 & a_{32} & a_{33} & a_{34} & a_{35} \\ 0 & 0 & a_{43} & a_{44} & a_{45} \\ 0 & 0 & 0 & a_{54} & a_{55} \end{bmatrix}$	$\begin{bmatrix} x & x & a_{13} & a_{24} & a_{35} \\ x & a_{12} & a_{23} & a_{34} & a_{45} \\ a_{11} & a_{22} & a_{33} & a_{44} & a_{55} \\ a_{21} & a_{32} & a_{43} & a_{54} & x \end{bmatrix}$

三角行列

三角行列には 2 つの格納スキームがあります。非パック格納スキームでは、行列が 2 次元配列に格納され、行列のすべての要素と配列の要素が 1 対 1 で対応します。ただし、行列内の

0 エントリは、配列内で設定もアクセスもされません (x で示された場所)。パック格納スキームでは、行列のゼロでない要素が、列ごとに 1 次元配列にパック化されます。

三角行列はパック格納を使用して格納できます。

三角帯行列	非パック格納の三角行列	パック格納の三角行列
$\begin{bmatrix} a_{11} & 0 & 0 \\ a_{21} & a_{22} & 0 \\ a_{31} & a_{32} & a_{33} \end{bmatrix}$	$\begin{bmatrix} a_{11} & x & x \\ a_{21} & a_{22} & x \\ a_{31} & a_{32} & a_{33} \end{bmatrix}$	$\begin{bmatrix} a_{11} \\ a_{21} \\ a_{31} \\ a_{22} \\ a_{32} \\ a_{33} \end{bmatrix}$

三角帯行列は、次のように 2 次元配列を使用して、パック格納で格納できます。x で示された要素はアクセスされません。

三角帯行列	パック格納の三角帯行列
$\begin{bmatrix} a_{11} & 0 & 0 \\ a_{21} & a_{22} & 0 \\ 0 & a_{32} & a_{33} \end{bmatrix}$	$\begin{bmatrix} a_{11} & a_{22} & a_{33} \\ a_{21} & a_{32} & x \end{bmatrix}$

対称行列

実対称行列または複素エルミート行列は、上側または下側の三角形の要素だけが 2 次元配列の対応する要素に明示的に格納される点で、三角行列に似ています。配列の残りの要素 (下記の x で示されたもの) は、設定もアクセスもされません。アクティブな上側または下側の三角形も、列ごとに 1 次元配列にパック化できます。

対称行列	非パック格納の対称行列	パック格納の対称行列
$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}$	$\begin{bmatrix} a_{11} & x & x \\ a_{21} & a_{22} & x \\ a_{31} & a_{32} & a_{33} \end{bmatrix}$	$\begin{bmatrix} a_{11} \\ a_{21} \\ a_{31} \\ a_{22} \\ a_{32} \\ a_{33} \end{bmatrix}$

三重対角行列

三重対角行列は、主対角、最初の上対角、および最初の下対角だけにゼロでない要素を持っています。これは 3 つの 1 次元配列を使用して格納されます。

三重対角行列	三重対角行列の格納
$\begin{bmatrix} a_{11} & a_{12} & 0 \\ a_{21} & a_{22} & a_{23} \\ 0 & a_{32} & a_{33} \\ 0 & 0 & a_{43} \end{bmatrix}$	$\begin{bmatrix} a_{21} \\ a_{32} \\ a_{43} \end{bmatrix} \begin{bmatrix} a_{11} \\ a_{22} \\ a_{33} \\ a_{44} \end{bmatrix} \begin{bmatrix} a_{12} \\ a_{23} \\ a_{34} \end{bmatrix}$

♦ ♦ ♦ 第 6 章

6 スパース計算

Oracle Developer Studio パフォーマンスライブラリには、スパース連立 1 次方程式の因子分解および求解に使用できる 2 つのソフトウェアパッケージ SPSOLVE および SuperLU があります。

SPSOLVE は、対称、構造的対称、および非対称な係数行列を、ユーザー指定の順序付けを含むいくつかの順序付け方法の 1 つを使用して解く、ルーチンのコレクションです。以前のリリースでは、SPSOLVE はスパースソルバーパッケージと呼ばれていました。これは主に Fortran で記述されており、FORTRAN 77 のみのインタフェースが含まれています。現在、Fortran 95 および C のインタフェースは提供されていません。Fortran 95 から SPSOLVE ルーチンを使用するには、FORTRAN 77 インタフェースを使用します。C から SPSOLVE を呼び出すには、ルーチン名にアンダースコアを付加し (`dgssin_()`、`dgssor_()` など)、引数を参照によって渡し、1 オリジンの配列インデックスを使用します。1 オリジンおよび 0 オリジンの配列インデックスの例については、47 ページの「非対称スパース行列」を参照してください。

Oracle Developer Studio パフォーマンスライブラリの SuperLU パッケージは、一般非対称スパース連立方程式を解くパブリックドメインアプリケーションの逐次バージョン (バージョン 3.0) です。逐次方式でありながら、SuperLU は並列化されたレベル 2 およびレベル 3 BLAS ルーチンのいくつかを使用します。SuperLU のアルゴリズム、ルーチン、およびデータ構造の詳細なドキュメントについては、70 ページの「スパース BLAS およびソルバーの参考資料」の項目 5、6、7 を参照してください。SuperLU は C で記述されているため、SuperLU ルーチンが Fortran ベースの SPSOLVE から呼び出されるか C ドライバプログラムから呼び出されるかにかかわらず、配列インデックスは 0 オリジンでなければなりません。詳細および例については、61 ページの「SuperLU インタフェース」を参照してください。

スパース行列

スパース行列は、通常、必要な格納領域を最小限に抑える形式で表されます。疎であることの利点を活かし、ゼロを格納しないことによって、かなりの格納領域を節約できます。SPSOLVE および SuperLU で使用される格納形式は圧縮スパース列 (CSC) 形式で、これは Harwell-Boeing 形式とも呼ばれます。

CSC 形式では、スパース行列を 2 つの整数配列と 1 つの浮動小数点配列で表します。整数配列 (colptr と rowind) はスパース行列の非ゼロの位置を指定し、浮動小数点配列 (values) は非ゼロ値に使用されます。

列ポインタ (colptr) 配列は n+1 個の要素で構成されています。colptr(i) は i 番目の列の先頭を指し、colptr(i+1)-1 は i 番目の列の末尾を指します。行インデックス (rowind) 配列には非ゼロ値の行インデックスが格納されます。values 配列には、対応する非ゼロの数値が格納されます。

neqns 個の方程式と nnz 個の非ゼロ要素から成るスパース行列には次の行列データ形式が存在します。

- 対称
- 構造的対称
- 非対称

現在、SuperLU は非対称行列のみをサポートしています。多くの場合、もっとも効率的なデータ表現は具体的な問題によって異なります。以降のセクションでは、スパース行列データ形式の例を示します。

対称スパース行列

対称スパース行列は、すべての *i* および *j* について a(*i, j*) = a(*j, i*) である行列です。この対称性のため、ソルバールーチンに渡す必要があるのは下側の三角形の値のみです。上側の三角形は下側の三角形から判断できます。

対称行列の例を次に示します。この例の出典は、A. George および J. W-H. Liu 著、『Computer Solution of Large Sparse Positive Definite Systems』です。

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

A を CSC 形式で表すには、次のようにします。

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

構造的対称スパース行列

構造的対称スパース行列では、その非ゼロ値が、すべての *i* および *j* について、a(*i*, *j*) ≠ 0 であれば a(*j*, *i*) ≠ 0 であるという性質を持ちます。構造的対称行列の連立方程式を解く場合は、行列全体をソルバールーチンに渡す必要があります。

構造的対称行列の例を次に示します。

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

A を CSC 形式で表すには、次のようにします。

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

非対称スパース行列

非対称スパース行列では、すべての *i* および *j* について a(*i*, *j*) = a(*j*, *i*) となるわけではありません。行列の構造には明白なパターンがありません。非対称行列の連立方程式を解く場合は、行列全体をソルバールーチンに渡す必要があります。非対称行列の例を次に示します。

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

A を CSC 形式で表すには、次のようにします。

- 1 オリジンのインデックス:
  - colptr: 1, 6, 7, 8, 9, 11

    - rowind: 1, 2, 3, 4, 5, 2, 3, 4, 2, 5
    - values: 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0, 10.0
- 0 オリジンのインデックス:
    - colptr: 0, 5, 6, 7, 8, 10
    - rowind: 0, 1, 2, 3, 4, 1, 2, 3, 1, 4
    - values: 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0, 10.0

スパース BLAS

Oracle Developer Studio パフォーマンスライブラリのスパース BLAS パッケージは次の 2 つのパッケージに基づいています。

- Dodson、Grimes、および Lewis による Netlib スパース BLAS パッケージは、スパースベクトルの演算を行う基本線形代数サブルーチン (BLAS) のスパース拡張で構成されています。
- NIST (米国標準技術局) Fortran スパース BLAS ライブラリは、さまざまな格納形式のスパース行列について行列の積および三角行列の連立方程式の求解を実行するルーチンで構成されています。

スパース BLAS の詳細については、次のソースを参照してください。

- スパース BLAS ルーチンについては、各ルーチンのセクション 3P のマニュアルページを参照してください。
- Netlib スパース BLAS パッケージの詳細については、`http://www.netlib.org/sparse-blas/index.html` を参照してください。
- NIST Fortran スパース BLAS ルーチンの詳細については、`http://math.nist.gov/spblas/` を参照してください。

以降のセクションで説明されているように、Netlib スパース BLAS および NIST Fortran スパース BLAS ライブラリのルーチンはそれぞれ独自の命名規則を使用しています。

Netlib スパース BLAS

Netlib スパース BLAS の各ルーチンは、接頭辞-基底-接尾辞という形式の名前を持っています。

- 接頭辞はデータ型を表します。
- 基底は演算を表します。
- 接尾辞は、既存の密 BLAS ルーチンを直接拡張したルーチンかどうかを表します。

次の表に、Netlib スパース BLAS のベクトルルーチンの命名規則を示します。

表 2 Netlib スパース BLAS の命名規則

演算	名前の基底	接頭辞および接尾辞
ドット積	-DOT-	S-I D-I C-UI Z-UI C-CI Z-CI
ベクトルのスカラー倍をベクトルに加算	-AXPY-	S-I D-I C-I Z-I
ギブンス回転を適用	-ROT-	S-I D-I
x を y に収集	-GTHR-	S- D- C- Z- S-Z D-Z C-Z Z-Z
x を y に分散	-SCTR-	S- D- C- Z-

接頭辞は次のデータ型のいずれかです。

- S: SINGLE
- D: DOUBLE
- C: COMPLEX
- Z: COMPLEX*16 または DOUBLE COMPLEX

接尾辞 I、CI、および UI は、密 BLAS ルーチンを直接拡張したスパース BLAS ルーチンであることを表します。

NIST Fortran スパース BLAS

NIST Fortran スパース BLAS の各ルーチンは、*XYYYZZ* という形式の 6 文字の名前を持っています。ここでは:

- *X* はデータ型を表します。
- *YYY* はスパース格納形式を表します。
- *ZZ* は演算を表します。

次の表に、*X*、*YYY*、および *ZZ* の取り得る値を示します。

表 3 NIST Fortran スパース BLAS ルーチンの命名規則

ルーチン名の変数	指定可能な値と意味
X – データ型を 1 文字で指定します	S: 単精度 D: 倍精度 C: 複素数 Z: 倍精度複素数
YYY – スパース格納形式を 3 文字で指定します	単一エントリ形式: CSC: 圧縮スパース列 COO: 座標 CSR: 圧縮スパース行

ルーチン名の変数	指定可能な値と意味
	DIA: 対角 ELL: ellpack JAD: 鋸歯状対角 SKY: スカイライン ブロックエントリ形式: BCO: ブロック座標 BSC: ブロック圧縮スパース列 BSR: ブロック圧縮スパース行 BDI: ブロック対角 BEL: ブロック ellpack VBR: ブロック圧縮スパース行
ZZ – 演算を 2 文字で指定します	MM: 行列と行列の積 SM: 三角行列の連立方程式の求解 (COO 以外のすべての形式をサポート) RP: 右置換 (JAD 形式のみ)

SPSOLVE インタフェース

SPSOLVE は、初期化、フィルインを低減する順序付け、記号分解、数値分解、および三角求解という一連の手順で、スパース連立方程式の解を計算します。ユーザーコードでは、個々のルーチンを呼び出すか、ワンコールインタフェースを利用して、これらの手順を実行できます。

SPSOLVE ルーチン

下の表に、ユーザーがアクセスできる SPSOLVE のルーチンとその目的を示します。

表 4 SPSOLVE スパースソルバールーチン

ルーチン名	説明
`DGSSFS()`	スパースソルバーのワンコールインタフェース
`DGSSIN()`	スパースソルバーの初期化
`DGSSOR()`	フィルインを低減する順序付けおよび記号分解
`DGSSUO()`	ユーザー指定の順序付け置換を設定し、記号分解を実行します (`DGSSOR` の代わりに呼び出される)
`DGSSFA()`	行列の値の入力および数値分解
`DGSSSL()`	三角求解
`DGSSRP()`	ソルバーで使用される置換を返します
`DGSSCO()`	係数行列の条件数の推定を返します
`DGSSDA()`	スパースソルバーを割り当て解除します

ルーチン名	説明
DGSSPS()	ソルバー統計を出力します

構造は同じで数値が異なっている行列は、SPSOLVE のルーチンを次の順序で呼び出して解くことができます。

```
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
```

ワンコールインタフェースは通常インタフェースほど柔軟ではありませんが、単一行列を因子分解し、いくつかの右辺を解くという、もっとも一般的な事例に対応できます。次の例に示すように、追加の右辺を解くには dgsssl() を追加で呼び出します。

```
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 ルーチンの呼び出し順序

SPSOLVE を使用するには、そのルーチンを次の順序で呼び出す必要があります。

1. ワンコールインタフェース: 単一行列を解く場合
   a. DGSSFS() - 初期化、順序付け、因子分解、求解
   b. DGSSSL() - 追加の求解 (オプション): 必要に応じて DGSSSL() を繰り返します
   c. DGSSDA() - 作業領域の割り当て解除
2. 通常インタフェース: 同じ構造を持つ複数の行列を解く場合
   a. DGSSIN() - 初期化
   b. DGSSOR() または DGSSUO() - 順序付けおよび記号分解
   c. DGSSFA() - 因子分解
   d. DGSSSL() - 求解: 必要に応じて DGSSFA() または DGSSSL() を繰り返します
   e. DGSSDA() - 作業領域の割り当て解除

SPSOLVE の例

次の例では、ワンコールインタフェースによる対称行列の連立方程式の求解と、通常インタフェースによる対称行列の連立方程式の求解を示します。

例1「対称行列の連立方程式を解く - ワンコールインタフェース」では、対称行列の連立方程式を解くためにワンコールインタフェースを使用し、例2「対称行列の連立方程式を解く – 通常インタフェース」では、対称行列の連立方程式を解くために個々のルーチンを呼び出しています。

例5「C から SPSOLVE ルーチンを呼び出す」は、Fortran SPSOLVE インタフェースを C プログラムから呼び出す方法を示しています。C プログラムから Fortran ルーチンを呼び出す方法の詳細については、*Oracle Solaris Studio 12.4 の Fortran プログラミングガイド*を参照してください。

例 1 対称行列の連立方程式を解く - ワンコールインタフェース

```
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 対称行列の連立方程式を解く – 通常インタフェース

```
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 非対称な値を持つ構造的対称行列の連立方程式を解く – 通常インタフェース

```
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　　非対称行列の連立方程式を解く – 通常インタフェース

```
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　　C から SPSOLVE ルーチンを呼び出す

```
#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 インタフェース

SuperLU には 2 つのドライバルーチン (単純およびエキスパート) があります。これらを呼び出して、SPSOLVE のワンコールインタフェースと同様の方法で、一般非対称スパース連立方程式を完全に解くことができます。これらおよびほかの SuperLU のユーザー呼び出し可能ルーチンは、単精度、倍精度、複素数、および倍精度複素数のデータ型で提供されています。すべての外部ルーチンの単精度名を次の表に示します。これらのルーチンにはマニュアルページ (セクション 3P) が用意されています。また、アプリケーションで使用されているスパース行列データ構造については、SuperMatrix(3P) のマニュアルページを参照してください。

表 5 SuperLU 計算ルーチン

ルーチン	説明
sgstrf	因子分解を計算します
sgssvx	因子分解と求解 (エキスパートドライバ)
sgssv	因子分解と求解 (単純ドライバ)
sgstrs	三角求解を計算します
sgsrfs	計算解を改良し、誤差限界を提供します
slangs	1 ノルム、フロベニウスノルム、または無限大ノルムを計算します
sgsequ	行および列のスケーリングを計算します
sgscon	条件数の逆数を推定します
slaqgs	一般スパース行列を均衡化します

表 6 SuperLU ユーティリティールーチン

ルーチン	説明
LUSolveTime	求解段階で費やされた時間を返します
LUFactTime	因子分解段階で費やされた時間を返します
LUFactFlops	因子分解段階での浮動小数点演算の回数を返します

ルーチン	説明
LUSolveFlops	求解段階での浮動小数点演算の回数を返します
sQuerySpace	メモリー統計の情報を返します
sp_ienv	指定されたマシンに依存するパラメータを返します
sPrintPerf	計算ルーチンによって収集された統計を出力します
set_default_options	ソルバーの動作を制御するパラメータをデフォルトオプションに設定します
StatInit	パフォーマンス統計を格納する構造体を割り当て、初期化します
StatFree	パフォーマンス統計を格納する構造体を解放します
Destroy_Dense_Matrix	密形式の SuperMatrix を割り当て解除します
Destroy_SuperNode_Matrix	スーパーノード形式の SuperMatrix を割り当て解除します
Destroy_CompCol_Matrix	圧縮スパース列形式の SuperMatrix を割り当て解除します
Destroy_CompCol_Permuted	置換された圧縮スパース列形式の SuperMatrix を割り当て解除します
Destroy_SuperMatrix_Store	SuperMatrix の行列を格納する実際の格納領域を割り当て解除します
sCopy_CompCol_Matrix	圧縮スパース列形式の SuperMatrix をコピーします
sCreate_CompCol_Matrix	圧縮スパース列形式の SuperMatrix を割り当てます
sCreate_Dense_Matrix	密形式の SuperMatrix を割り当てます
sCreate_CompRow_Matrix	圧縮スパース行形式の SuperMatrix を割り当てます
sCreate_SuperNode_Matrix	スーパーノード形式の SuperMatrix を割り当てます
sp_preorder	元のスパース行列の列を置換します
sp_sgemm	SuperMatrix に密行列を乗算します

C からの SuperLU の呼び出し

SuperLU のルーチンは C で記述されています。したがって、*列および行に関連するインデックスは 0 オリジンでなければなりません*。次の例では、倍精度の単純ドライバ `dgssv()` を呼び出して、因子 L および U を計算し、解行列を求めます。

例 6 SuperLU 単純ドライバ

```
#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);
}
```

上記の例を実行すると、次のようになります。

```
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 エキスパートドライバ

```
#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);
}
```

上記の例を実行すると、次のようになります。

```
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
```

Fortran からの SuperLU の呼び出し

Fortran から SuperLU を呼び出すもっとも簡単な方法は、SPSOLVE インタフェースを使用することです。SuperLU は、非対称係数行列を解くためにルーチン `DGSSIN()` の入力引数 `MTXTYP` を介して選択できます。これは SPSOLVE の初期化ルーチンです。同じ引数がワンコールインタフェースルーチン `DGSSFS()` にも存在します。

`MTXTYP` の有効なオプションを次の表に示します。SuperLU を呼び出すには、行列タイプとして 's0' または 'S0' を選択します。SPSOLVE は Fortran ベースなので、入力行列に関連する列および行のインデックスはすべて 1 オリジンであるべきです。ただし、SuperLU を `DGSSIN()` または `DGSSFS()` から呼び出す (`MTXTYP` = 's0' または 'S0' を設定) 場合、これらのインデックスは 0 オリジンでなければなりません。

表 7 `DGSSIN()` および `DGSSFS()` の行列タイプオプション

オプション	行列のタイプ	ソルバー
'sp' または 'SP'	対称構造、正定値	SPSOLVE
'ss' または 'SS'	対称構造、対称値	SPSOLVE
'su' または 'SU'	対称構造、非対称値	SPSOLVE
'uu' または 'UU'	非対称構造、非対称値	SPSOLVE
's0' または 'S0'	非対称構造、非対称値	SuperLU

`DGSSIN()` のあとにルーチン `DGSSOR()` を呼び出して、フィルインを低減する順序付けおよび記号分解を実行する必要があります。文字引数 (`ORDMTHD`) は、順序付け方法を選択するた

めに使用されます。この引数はワンコールインタフェースルーチン DGSSFS() にも存在します。SPSOLVE および SuperLU の有効な順序付け方法を次の表に示します。DGSSOR() の代わりに DGSSUO() を呼び出すことにより、特定の順序付けをソルバーに提供することもできます。入力の置換配列は 0 オリジンでなければなりません。

表 8 DGSSOR() および DGSSFS() の行列順序付けオプション

オプション	順序付け方法	ソルバー
'nat' または 'NAT'	自然順序 (順序付けなし)	SPSOLVE、SuperLU
'mmd' または 'MMD'	A'*A の最小次数 (デフォルト)	SPSOLVE、SuperLU
'gnd' または 'GND'	一般入れ子分割	SPSOLVE
'spm' または 'SPM'	A'+A の最小次数順序付け	SuperLU
'sam' または 'SAM'	近似最小次数列	SuperLU

上記のとおり、一般入れ子分割法は SuperLU では使用できません。その一方で、A'+A の最小次数順序付けおよび近似最小次数列順序付けは SPSOLVE では使用できません。

SuperLU の例

次のコード例では、SPSOLVE の通常インタフェースおよびワンコールインタフェースを介して SuperLU を選択し、一般非対称行列の連立方程式の因子分解および求解を行う方法を示します。

例 8 SPSOLVE の通常インタフェースを介して SuperLU を呼び出す

```
      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
```

上記の例を実行すると、次のようになります。

```
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 SPSOLVE のワンコールインタフェースを介して SuperLU を呼び出す

```
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
```

上記の例を実行すると、次のようになります。

```
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
```

スパース BLAS およびソルバーの参考資料

次の書籍および論文には、スパース BLAS およびスパースソルバールーチンの詳細が記載されています。

1. D.S. Dodson、R.G. Grimes、および J.G. Lewis 著、『Sparse Extensions to the Fortran Basic Linear Algebra Subprograms』、ACM Transactions on Mathematical Software、1991 年 6 月、Vol 17、No. 2。
2. A. George および J. W-H. Liu 著、『Computer Solution of Large Sparse Positive Definite Systems』、Prentice-Hall Inc.、Englewood Cliffs, New Jersey、1981 年。
3. E. Ng および B. W. Peyton 著、『Block Sparse Cholesky Algorithms on Advanced Uniprocessor Computers』、SIAM M. Sci Comput.、14:1034-1056、1993 年。
4. Ian S. Duff、Roger G. Grimes、および John G. Lewis 著、『User's Guide for the Harwell-Boeing Sparse Matrix Collection (Release I)』、技術レポート TR/PA/92/86、CERFACS、Lyon、フランス、1992 年 10 月。
5. J. W. Demmel、J. R. Gilbert、および X. S. Li 著、『SuperLU User's Guide』、技術レポート LBNL-44289。
6. X. S. Li 著、『An Overview of SuperLU: Algorithms, Implementation, and User Interface』、ACM Transactions on Mathematical Software、2004 年。
7. 7. J. W. Demmel、S. C. Eisenstat、J. R. Gilbert、X. S. Li、J. W. H. Liu 著、『A supernodal approach to sparse partial pivoting』、SIAM J. Matrix Analysis and Applications、Vol 20、No. 3、1999 年、720-755 ページ。

♦ ♦ ♦ 第 7 章

7 Oracle Developer Studio パフォーマンスライブラリの信号処理ルーチンの使用

離散フーリエ変換 (DFT) は常に科学とエンジニアリングの多くの分野で重要な分析ツールです。ただし、DFT が広く使われるようになったのは、高速フーリエ変換 (FFT) が開発されてからです。これは、FFT では $O(N\log_2 N)$ 演算のみが必要ですが、DFT では $O(N^2)$ の計算が必要であるためです。,

Oracle Developer Studio パフォーマンスライブラリには FFT、畳み込みや相関などの関連 FFT 演算、および三角法による変換を計算する一連のルーチンが含まれています。

この章は、次の 3 つのセクションに分かれています。

- 順方向および逆方向 FFT ルーチン
- サインおよびコサイン変換
- 畳み込みおよび相関

各セクションには、ルーチンの使用方法を示す例が含まれています。

ヒント - Fortran 95 および C インタフェースと各ルーチンで使用される引数の型については、section 3P マニュアルページで個々のルーチンを参照してください。マニュアルページのルーチン名は小文字にする必要があります。

たとえば、SFFTC ルーチンのマニュアルページを表示するには、小文字でルーチン名を指定して、次のコマンドを使用します。

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

FFT ルーチンの概要の場合:

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

順方向および逆方向 FFT ルーチン

次の表に、FFT ルーチンの名前とそれらの呼び出しシーケンスを一覧表示します。倍精度ルーチン名は角括弧で囲んでいます。引数のデータ型とサイズの詳細については、個々のマニュアルページを参照してください。

- 表9「FFT 1 次元ルーチンとそれらの引数」
- 表10「FFT 2 次元ルーチンとそれらの引数」
- 表11「FFT 3 次元ルーチンとそれらの引数」

表 9 FFT 1 次元ルーチンとそれらの引数

ルーチン名	引数
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 2 次元ルーチンとそれらの引数

ルーチン名	引数
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 3 次元ルーチンとそれらの引数

ルーチン名	引数
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)

ルーチン名	引数
CFFTC3 [ZFFTZ3]	(OPT, N1, N2, N3, SCALE, X, LDX1, LDX2, Y, LDY1, LDY2, TRIGS, IFAC, WORK, LWORK, ERR)

Oracle Developer Studio パフォーマンスライブラリ FFT ルーチンは次の引数を使用します。

- OPT: ルーチンを初期化のために呼び出すか、または変換を計算するために呼び出すかを示すフラグ。
- N1、N2、N3: 1、2、3 次元変換の問題の次元。
- X: 入力配列。ここで、X の型は、ルーチンが複素数-複素数変換または複素数-実数変換の場合は COMPLEX です。実数-複素数変換の場合の X の型は REAL です。
- Y: 出力配列。ここで、Y の型は、ルーチンが複素数-複素数変換または実数-複素数変換の場合は COMPLEX です。複素数-実数変換の場合の Y の型は REAL です。
- LDX1、LDX2 および LDY1、LDY2: LDX1 および LDX2 は入力配列のリーディングディメンジョンで、LDY1 と LDY2 は出力配列のリーディングディメンジョンです。FFT ルーチンでは、出力で入力を上書きする (インプレース変換) ことも、出力を入力配列と別の配列に格納する (アウトオブプレース変換) こともできます。複素数-複素数変換では、入力データは出力データと同じサイズです。ただし、実数-複素数および複素数-実数変換では、入力データと出力データのメモリー要件が異なります。インプレース変換を計算する場合は、入力配列に、変換結果を収容するための十分な大きさがあることを注意して確認する必要があります。
- TRIGS: 三角法の重みを格納する配列。
- IFAC: 問題の次元の係数を格納する配列。問題のサイズは次のとおりです。
  - 1 次元 FFT: 次元 N1 の問題のサイズ
  - 2 次元 FFT: 次元 N1 および N2 の問題のサイズ
  - 3 次元 FFT: 次元 N1、N2、N3 の問題のサイズ

  N1、N2、および N3 は任意のサイズにできますが、実数-複素数または複素数-実数の変換は、次の場合にもっとも効率的に計算できます。

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

  、

  また、複素数-複素数変換は次の場合にもっとも効率的に計算されます:

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

  、

  ここで、p、q、r、s、t、u、および v は整数で、$p, q, r, s, t, u, v \geq 0$ です。
- WORK: ワークスペース。このサイズはルーチンと、ルーチンが並列化されている場合に、変換を計算するために使用されているスレッドの数によって異なります。
- LWORK: ワークスペースのサイズ。LWORK がゼロの場合、ルーチンは必要なサイズでワークスペースを割り当てます。

- SCALE: 出力をスケーリングするスカラー。文献では、逆変換が 1 次元変換の場合に 1/N1、2 次元変換の場合に 1/(N1 × N2)、および 3 次元変換の場合に 1/(N1 × N2 × N3) のスケーリング係数で定義されていることがあります。そのような場合、逆変換は正規化されていると言います。正規化 FFT のあとにその逆方向 FFT が行われると、結果は元の入力データになります。Oracle Developer Studio パフォーマンスライブラリ FFT ルーチンは正規化されません。ただし、SCALE に格納されている適切なスケーリング係数で逆方向 FFT ルーチンを呼び出すことによって、正規化を簡単に実行できます。
- ERR: ルーチンでエラーが検出された場合はゼロ以外の値を返し、それ以外の場合はゼロを返すフラグ。

1 次元 FFT ルーチン

1 次元 FFT ルーチンは、1 次元のみの実数または複素数データの FFT を計算します。データは 1 つ以上の複素数または実数のシーケンスです。1 つのシーケンスの場合、データはベクトルに格納されます。複数のシーケンスが変換される場合は、2 次元配列で列方向にシーケンスが格納され、1 次元 FFT が列方向に沿ってシーケンスごとに計算されます。1 次元順方向 FFT ルーチンは次を計算します。 $X(k)=\sum_{n=0}^{Nl-1} x(n)e^{\frac{-2\pi ink}{Nl}}, k=0,\ldots,Nl-1,$ ここで $i=\sqrt{-1}$ 。

または極形式で表すと:

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

。

逆方向 FFT ルーチンは、次を計算します $x(n)=\sum_{k=0}^{Nl-1} X(k)e^{\frac{2\pi ink}{Nl}}, n=0,\ldots,Nl-1,$

極形式では:

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

順変換では、入力がサイズ $N1$ の 1 つ以上の複素数シーケンスである場合、結果は、それぞれ N 1 の無関係なデータポイントから構成される 1 つ以上の複素数シーケンスになります。ただし、入力が、それぞれ $N1$ の実数データポイントを含む 1 つ以上の実数シーケンスである場合、結果は、 共役対称な 1 つ以上の複素数シーケンスになります。つまり:

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

$X(0)$ の虚部は常に 0 です。$N1$ が偶数の場合、 $X(\frac{N1}{2})$ の虚部もゼロです。両方のゼロは明示的に格納されます。各シーケンスの後半は前半から得られるため、 $\frac{N1}{2}+1$ 複素数データポイントのみが計算され、出力配列に格納されます。この章のここやほかの場所で、整数除算は切り捨てています。

逆変換では、$N1$ ポイントの複素数-複素数変換が計算される場合、各入力シーケンスに $N1$ の無関係なデータポイントが期待され、出力配列で $N1$ データポイントが返されます。ただし、$N1$ ポイントの複素数-実数変換が計算される場合、入力に、各共役対称入力シーケンスの最初の $\frac{N1}{2}+1$ の複素数データポイントのみが期待され、ルーチンは各出力シーケンスで $N1$ の実数データポイントを返します。

$N1$ の値ごとに、順変換または逆変換ルーチンを呼び出して、実際の FFT を計算する前に、$N1$ の係数と、それらの係数に関連付けられる三角法の重みを計算する必要があります。係数と三角法の重みは、$N1$ が変更されないかぎり、その後の変換で再利用できます。

次の表の表12「単精度 1 次元 FFT ルーチン」に、単精度 1 次元 FFT ルーチンとその目的を示します。入力および出力として 2 次元配列を使用するルーチンについては、表12「単精度 1 次元 FFT ルーチン」に、リーディングディメンジョン要件も示しています。同じ情報は、データ型が倍精度と倍精度複素数であることを除いて、対応する倍精度ルーチンに適用します。マッピングについては、表12「単精度 1 次元 FFT ルーチン」を参照してください。ルーチンとその引数の詳細な説明については、個々のマニュアルページを参照してください。

表 12 単精度 1 次元 FFT ルーチン

名前	目的	入力のサイズおよび型	出力のサイズおよび型	リーディングディメンジョン要件
SFFTC	OPT = 0 初期化			
	OPT = -1 単一ベクトルの実数-複素数の順方向 1 次元 FFT	$N1$、 実数	$\frac{N1}{2}+1$、 複素数	
SFFTC	OPT = 0 初期化			
	OPT = 1 単一ベクトルの複素数-実数の逆方向 1 次元 FFT	$\frac{N1}{2}+1$、 複素数	$N1$ 実数	
CFFTC	OPT = 0 初期化			
	OPT = -1 単一ベクトルの複素数-複素数の順方向 1 次元 FFT	$N1$、 複素数	$N1$、 複素数	
	OPT = 1 単一ベクトルの複素数-複素数の逆方向 1 次元 FFT	$N1$、 複素数	$N1$、 複素数	
SFFTCM	OPT = 0 初期化			

名前	目的	入力のサイズおよび型	出力のサイズおよび型	リーディングディメンジョン要件	
	OPT = -1 M ベクトルの実数-複素数の順方向 1 次元 FFT	$N1 \times M$、実数	$\left(\frac{NI}{2}+1\right)\times M$、複素数	LDX1 = 2 × LDY1	LDX1 ≥ $N1$
CFFTSM	OPT = 0 初期化				
	OPT = 1 M ベクトルの複素数-実数の逆方向 1 次元 FFT	$\left(\frac{NI}{2}+1\right)\times M$、複素数	$N1 \times M$、実数	LDX1 ≥ $\frac{NI}{2}+1$ LDY1=2 × LDX1	LDX1 ≥ $\frac{NI}{2}+1$ LDY1 ≥ $N1$
CFFTCM	OPT = 0 初期化				
	OPT = -1 M ベクトルの複素数-複素数の順方向 1 次元 FFT	$N1 \times M$、複素数	$N1 \times M$、複素数	LDX1 ≥ $N1$ LDY1 ≥ $N1$	LDX1 ≥ $N1$ LDY1 ≥ $N1$
	OPT = 1 M ベクトルの複素数-複素数の逆方向 1 次元 FFT	$N1 \times M$、複素数	$N1 \times M$、複素数	LDX1 ≥ $N1$ LDY1 ≥ $N1$	LDX1 ≥ $N1$ LDY1 ≥ $N1$

注記 - 表12「単精度 1 次元 FFT ルーチン」に関して、次のことに注意してください。

- LDX1 は入力配列のリーディングディメンジョンです。
- LDY1 は出力配列のリーディングディメンジョンです。
- N1 は FFT 問題の最初の次元です。
- N2 は FFT 問題の 2 番目の次元です。
- OPT = 0 でルーチンを呼び出して、ルーチンを初期化する場合に、実行される唯一のエラーチェックは、$N1 < 0$ かどうかを判断することです

次の例は、一連のシーケンスの 1 次元実数-複素数および複素数-実数 FFT の計算方法を示しています。

例 10 1 次元実数-複素数 FFT および複素数-実数 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, .3,0.0,0.0,0.0,0.0,7.,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
```

例 7-1 注記:

X の順方向 FFT は実際に次のようになります。

$$Z = \begin{matrix} (0.6, 0.0) & (24.0, 0.0) \\ (-0.2, 0.1) & (-1.5, 0.9) \\ (-0.2, 0.1) & (-1.5, 0.9) \end{matrix}$$

対称性のために、Z(2) は Z(1) の複素共益であるため、最初の 2 つのみの $\frac{NI}{2}+1=2$ 複素数値が格納されます。インプレース順変換の場合、実数配列 X を入力および出力として、SFFTCM が呼び出されます。SFFTCM では、出力配列の型を COMPLEX と想定するため、出力配列としての X のリーディングディメンジョンは X が複素数であるかのようにする必要があります。実数配列 X のリーディングディメンジョンは LDX = 2 × LDC であるため、複素数出力配列としての X のリーディングディメンジョンは LDC である必要があります。同様に、インプレース逆変換では、複素数配列 Z を入力および出力として、CFFTSM が呼び出されます。CFFTSM では、出力配列の型を REAL と想定するため、出力配列としての Z のリーディングディメンジョンは Z が実数であるかのようにする必要があります。複素数配列 Z のリーディングディメンジョンは LDZ であるため、実数出力配列としての Z のリーディングディメンジョンは LDZ × 2 である必要があります。

次の例、例11「1 次元複素数-複素数 FFT」では、一連のシーケンスの 1 次元複素数-複素数 FFT の計算方法を示しています。

例 11 1 次元複素数-複素数 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)
```

2 次元 FFT ルーチン

1 次元 FFT ルーチンでは、入力が 2 次元配列の場合、FFT が 1 次元のみに沿って、つまり、配列の列に沿って計算されます。2 次元 FFT ルーチンは、入力として 2 次元配列を受け取り、列と行の両方の次元に沿って FFT を計算します。具体的には、順方向 2 次元 FFT ルーチンは次のように計算します。

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

逆方向 2 次元 FFT ルーチンは次のように計算します。

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

順方向および逆方向の両方の 2 次元変換で、入力問題が $N1 \times N2$ である複素数-複素数変換は、同様に $N1 \times N2$ である複素数配列を生成します。

実数-複素数 2 次元変換 (順方向 FFT) の計算時に、実数入力配列の次元が $N1 \times N2$ である場合、結果は次の次元の複素数配列になります: $(\frac{N1}{2}+1)\times N2$ 。

逆に、次元 N1 × N2 の複素数-実数変換 (逆方向 FFT) の計算時、$(\frac{N1}{2}+1)\times N2$ 複素数配列が入力として必要です。実数-複素数および複素数-実数 1 次元 FFT と同様に、共役対称性のため、最初のみの $\frac{N1}{2}+1$ 複素数データポイントが最初の次元に沿って、入力または出力配列に格納される必要があります。複素数部分配列 $X(\frac{N1}{2}+1:N1-1,:)$ は次から取得でき、$X(0:\frac{N1}{2},:)$ 次のようになります。

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

2 次元変換を計算するには、FFT ルーチンを 2 回呼び出す必要があります。1 回の呼び出しでルーチンを初期化し、2 回目の呼び出しで実際に変換を計算します。初期化には、$N1$ および $N2$ の係数と、それらの計数に関連付けられる三角法の重みの計算が含まれます。その後の順変換または逆変換では、$N1$ と $N2$ が変更されないかぎり、初期化は必要ありません。

重要: 2 次元 FFT ルーチンからの戻り時に、$Y(0 : N - 1, :)$ に変換結果が格納され、$Y(N : LDY-1, :)$ の元の内容が上書きされます。ここで、$N = N1$ (複素数-複素数および複素数-実数変換の場合)、および $N = \frac{N1}{2}+1$ (実数-複素数変換の場合) です。

次の表の表13「単精度 2 次元 FFT ルーチン」に、単精度 2 次元 FFT ルーチンとその目的を示します。同じ情報は、データ型が倍精度と倍精度複素数であることを除いて、対応する倍精度ルーチンに適用します。マッピングについては、表13「単精度 2 次元 FFT ルーチン」を参照してください。ルーチンとその引数の詳細な説明については、個々のマニュアルページを参照してください。

表 13 単精度 2 次元 FFT ルーチン

名前	目的	入力のサイズ、型	出力のサイズ、型	リーディングディメンジョン要件	
SFFTC2	OPT = 0 初期化				
	OPT = -1 実数-複素数順方向 2 次元 FFT	N1 × N2、実数	$(\frac{N1}{2}+1)\times N2$、複素数	LDX1 = 2 × LDY1 LDY1 ≥ $\frac{N1}{2}+1$	LDX1 ≥ $N1$ LDY1 ≥ $\frac{N1}{2}+1$
CFFTS2	OPT = 0 初期化				
	OPT = 1 複素数-実数逆方向 2 次元 FFT	$(\frac{N1}{2}+1)\times N2$、複素数	$N1 \times N2$、実数	LDX1 ≥ $\frac{N1}{2}+1$ LDY1=2 × LDX1	LDX1 ≥ $\frac{N1}{2}+1$ LDY1≥ 2 × LDX1 LDY1 は偶数
CFFTC2	OPT = 0 初期化				
	OPT = -1 複素数-複素数順方向 2 次元 FFT	$N1 \times N2$、複素数	$N1 \times N2$、複素数	LDX1 ≥ $N1$ LDY1 = LDX1	LDX1 ≥ $N1$ LDY1 ≥ $N1$
	OPT = 1 複素数-複素数逆方向 2 次元 FFT	$N1 \times N2$、複素数	$N1 \times N2$、複素数	LDX1 ≥ $N1$ LDY1 = LDX1	LDX1 ≥ $N1$ LDY1 = LDX1

注記 - 表13「単精度 2 次元 FFT ルーチン」に関して、次のことに注意してください。

- LDX1 は入力配列のリーディングディメンジョンです。
- LDY1 は出力配列のリーディングディメンジョンです。
- N1 は FFT 問題の最初の次元です。
- N2 は FFT 問題の 2 番目の次元です。
- OPT = 0 でルーチンを呼び出して、ルーチンを初期化した場合に、実行される唯一のエラーチェックは、$N1$、$N2 < 0$ かどうかを判断することです。

例 12 2 次元配列の 2 次元実数-複素数 FFT および複素数-実数 FFT

次の例に、2 次元配列の 2 次元実数-複素数 FFT および複素数-実数 FFT を計算する方法を示します。

```
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
```

3 次元 FFT ルーチン

Oracle Developer Studio パフォーマンスライブラリには、3 次元 FFT を計算するルーチンが含まれています。この場合、FFT は 3 次元配列の 3 つすべての次元に沿って計算されます。順方向 FFT は次を計算します:

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

、

また、逆方向 FFT は次を計算します:

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

複素数-複素数変換で、入力の問題が $N1 \times N2 \times N3$ である場合、3 次元変換は、同様に $N1 \times N2 \times N3$ である複素数配列を生成します。実数-複素数 3 次元変換の計算時に、実数入力配列が次元 $N1 \times N2 \times N3$ である場合、結果は次の次元の複素数配列になります: $(\frac{N1}{2}+1) \times N2 \times N3$ 。逆に、次元 $N1 \times N2 \times N3$ の複素数-実数 FFT の計算時、$(\frac{N1}{2}+1) \times N2 \times N3$ 複素数配列が入力として必要です。実数-複素数および複素数-実数の 1 次元 FFT と同様に、共役対称性のため、最初のみの $\frac{N1}{2}+1$ 複素数データポイントが最初の次元に沿って格納される必要があります。複素数部分配列 $X(\frac{N1}{2}+1:N1-1,\ :,\ :)$ は次から取得でき、$X(0:\frac{N1}{2}+1,\ :,\ :)$ 次のようになります。

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

3 次元変換を計算するには、FFT ルーチンを 2 回呼び出す必要があります。1 回は初期化し、もう 1 回は実際に変換を計算します。初期化には、$N1$、$N2$、および $N3$ の係数と、それらの係数に関連付けられる三角法の重みの計算が含まれます。その後の順変換または逆変換では、$N1$、$N2$、および $N3$ が変更されないかぎり、初期化は必要ありません。

重要: 3 次元 FFT ルーチンからの戻り時に、$Y(0 : N - 1, :, :)$ に変換結果が格納され、$Y(N$:LDY1-1, :, :) の元の内容が上書きされます。ここで、複素数-複素数および複素数-実数変換では $N = N1$ で、実数-複素数変換では $N = \frac{N1}{2}+1$ (実数-複素数変換の場合) です。

表14「単精度 3 次元 FFT ルーチン」に、単精度 3 次元 FFT ルーチンとその目的を示します。同じ情報は、データ型が倍精度と倍精度複素数であることを除いて、対応する倍精度ルーチンに適用します。マッピングについては、表14「単精度 3 次元 FFT ルーチン」を参照してください。ルーチンとその引数の詳細な説明については、個々のマニュアルページを参照してください。

表 14 単精度 3 次元 FFT ルーチン

名前	目的	入力のサイズ、型	出力のサイズ、型	リーディングディメンジョン要件	
SFFTC3	OPT = 0 初期化				
	OPT = -1 実数-複素数順方向 3 次元 FFT	$N1 \times N2 \times N3$、実数	$(\frac{N1}{2}+1) \times N2 \times N3$ 複素数	LDX1=2 × LDY1 LDX2 ≥ $N2$ LDY1 ≥ $\frac{N1}{2}+1$ LDY2 = LDX2	LDX1 ≥ $N1$ LDX2 ≥ $N2$ LDY1 ≥ $\frac{N1}{2}+1$ LDY2 ≥ $N2$
CFFTS3	OPT = 0 初期化				
	OPT = 1 複素数-実数逆方向 3 次元 FFT	$(\frac{N1}{2}+1) \times N2 \times N3$ 複素数	$N1 \times N2 \times N3$、実数	LDX1 ≥ $\frac{N1}{2}+1$ LDX2 ≥ $N2$ LDY1=2 × LDX1 LDY2=LDX2	LDX1 ≥ $\frac{N1}{2}+1$ LDX2 ≥ $N2$ LDY1 ≥ 2 × LDX1 LDY1 は偶数 LDY2 ≥ $N2$
CFFTC3	OPT = 0 初期化				
	OPT = -1 複素数-複素数順方向 3 次元 FFT	$N1 \times N2 \times N3$、複素数	$N1 \times N2 \times N3$、複素数	LDX1 ≥ $N1$ LDX2 ≥ $N2$	LDX1 ≥ $N1$ LDX2 ≥ $N2$

名前	目的	入力のサイズ、型	出力のサイズ、型	リーディングディメンジョン要件	
				LDY1=LDX1	LDY1 ≥ $N1$
				LDY2=LDX2	LDY2 ≥ $N2$
	OPT = 1 複素数-複素数逆方向 3 次元 FFT	$N1 \times N2 \times N3$、複素数	$N1 \times N2 \times N3$、複素数	LDX1 ≥ $N1$	LDX1 ≥ $N1$
				LDX2 ≥ $N2$	LDX2 ≥ $N2$
				LDY1=LDX1	LDY1 ≥ $N1$
				LDY2=LDX2	LDY2 ≥ $N2$

注記 - 表14「単精度 3 次元 FFT ルーチン」に関して、次のことに注意してください。

- LDX1 は入力配列の最初のリーディングディメンジョンです。
- LDX2 は入力配列の 2 番目のリーディングディメンジョンです。
- LDY1 は出力配列の最初のリーディングディメンジョンです。
- LDY2 は出力配列の 2 番目のリーディングディメンジョンです。
- N1 は FFT 問題の最初の次元です。
- N2 は FFT 問題の 2 番目の次元です。
- N3 は FFT 問題の 3 番目の次元です。
- OPT = 0 でルーチンを呼び出して、ルーチンを初期化した場合に、実行される唯一のエラーチェックは、$N1$、$N2$、$N3 < 0$ かどうかを判断することです。

例 7-4 に、3 次元配列の 3 次元実数-複素数 FFT および複素数-実数 FFT を計算する方法を示します。

例 13 3 次元配列の 3 次元実数-複素数 FFT および複素数-実数 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
```

コメント

インプレース実数-複素数または複素数-実数変換を実行する場合は、入力配列のサイズが、結果を保持するために十分に大きいことを注意して確認する必要があります。たとえば、入力の型が最初のリーディングディメンジョン N の複素数配列に格納されている複素数である場合、同じ配列を使用して、実数の結果を格納するには、実数出力配列としてのその最初のリーディングディメンジョンは $2 \times N$ になります。逆に、入力の型が最初のリーディングディメンジョン $2 \times N$ の実数配列に格納されている実数である場合、同じ配列を使用して、複素数の結果を格納するには、複素数出力配列としてのその最初のリーディングディメンジョンは N になります。インプレースおよびアウトオブプレース変換のリーディングディメンジョン要件については、表12「単精度 1 次元 FFT ルーチン」、表13「単精度 2 次元 FFT ルーチン」、および表14「単精度 3 次元 FFT ルーチン」を参照してください。

1 次元および多次元 FFT で、実数-複素数変換または複素数-実数変換による実数データと複素数データ間の変換は、$N1$ 実数データポイントが次に対応しているため混乱することがあります: $\frac{N1}{2}+1$ 複素数データポイント。$N1$ 実数データポイントは N 1 複素数データポイントにマッピングしますが、複素数データには共役対称性があるため、$\frac{N1}{2}+1$ データポイントは複素数-実数変換の入力として、および実数-複素数変換の出力としてのみ格納される必要があります。多次元 FFT では、対称性は最初だけでなく、すべての次元に沿って存在します。ただし、2 次元および 3 次元 FFT ルーチンは、2 番目と 3 番目の次元の複素数データをそっくりそのまま格納します。

FFT ルーチンは、任意のサイズの $N1$、$N2$、および $N3$ を受け入れますが、$N1$、$N2$、および $N3$ の値が相対的に小さい素数に分解できる場合に、FFT をもっとも効率的に計算できます。実数-複素数または複素数-実数変換は次の場合にもっとも効率的に計算できます:

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

,

また、複素数-複素数変換は次の場合にもっとも効率的に計算されます:

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

ここで、*p*、*q*、*r*、*s*、*t*、*u*、および *v* は整数で、*p*、*q*、*r*、*s*、*t*、*u*、*v* ≥ 0 です。

例14「RFFTOPT の例」に示すように、関数 *x*FFTOPT を使用して、最適なシーケンスの長さを判断できます。入力シーケンスの長さを考慮して、関数は元の長さにもっとも近いサイズの最適な長さを返します。

例 14 RFFTOPT の例

```
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
```

コサインおよびサイン変換

特殊な対称性を持つ DFT への入力はさまざまなアプリケーションで発生します。対称性を利用する変換は通常、実数-複素数および複素数-実数 FFT 変換などのように、ストレージと計算数を節約します。パフォーマンスライブラリ コサインおよびサイン変換は、偶関数および奇関数内で検出された対称性プロパティーを利用する特殊なケースの FFT ルーチンです。

注記 - Oracle Developer Studio パフォーマンスライブラリのサインおよびコサイン変換ルーチンは FFTPACK (http://www.netlib.org/fftpack/) に含まれるルーチンに基づいています。V 接頭辞が付いたルーチンは、VFFTPACK (http://www.netlib.org/vfftpack/) に含まれているルーチンに基づいたベクトル化したルーチンです。

高速コサインおよびサイン変換ルーチン

次の表に、Oracle Developer Studio パフォーマンスライブラリの高速コサインおよびサイン変換を示します。倍精度ルーチンの名前は角括弧で囲んでいます。名前が「V」で始まるルーチンは、1 つ以上のシーケンスの変換を同時に計算できます。名前が「I」で終わるものは、初期化ルーチンです。

- 表15「偶数シーケンスの高速コサイン変換ルーチンとそれらの引数」
- 表16「1/4 波偶数シーケンスの高速コサイン変換ルーチンとそれらの引数」
- 表17「奇数シーケンスの高速サイン変換ルーチンとそれらの引数」
- 表18「1/4 波奇数シーケンスの高速サイン変換ルーチンとそれらの引数」

表 15 偶数シーケンスの高速コサイン変換ルーチンとそれらの引数

ルーチン名	引数
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 1/4 波偶数シーケンスの高速コサイン変換ルーチンとそれらの引数

ルーチン名	引数
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 奇数シーケンスの高速サイン変換ルーチンとそれらの引数

ルーチン名	引数
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 1/4 波奇数シーケンスの高速サイン変換ルーチンとそれらの引数

ルーチン名	引数
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)
VSINQI [VDSINQI]	(LEN, TABLE) 注記:

注記 - 前の表に関して次の情報に注意してください。

- M: 変換するシーケンスの数。
- LEN、LEN-1、LEN+1: 入力シーケンスの長さ。
- X: 変換するシーケンスを格納する実数の配列。出力で、実数変換結果は X に格納されます。
- TABLE: 変換ルーチンに必要な変換サイズに固有の定数の配列。定数は初期化ルーチンによって計算されます。
- WORK: 変換ルーチンに必要なワークスペース。単一のシーケンスを処理するルーチンで、WORK には初期化ルーチンによって計算された定数も格納されます。

高速サイン変換

よく見られる対称性のもう 1 つのタイプは、$n = -N+1, \cdots, 0, \cdots, N$ の場合に $x(n) = -x(-n)$ である奇数対称性です。高速コサイン変換の場合と同様に、高速サイン変換 (FST) は、奇数対称性を利用してメモリーと計算を節約します。実数奇数シーケンス x の場合、対称性は、$x(0) = -x(0) = 0$ であることを示します。したがって、x の長さが $2N$ の場合、FST を計算するために、x の $N = 1$ の値のみ必要です。ルーチン SINT は単一の実数奇数シーケンスの FST を計算し、VSINT は 1 つ以上のシーケンスの FST を計算します。[V]SINT を呼び出す前に、入力長 N-1 に関連付けられた三角法の定数と係数を計算するために、[V]SINTI を呼び出す必要があります。FST はそれ自身の逆変換です。VSINT を 2 回呼び出すと、元の N -1 データポイントになります。SINT を 2 回呼び出すと、$2N$ をかけた元の N -1 データポイントになります。

$-N+1, \cdots, 0, \cdots, N$ の場合に $x(n) = -x(-n - 1)$ のような対称性を持つ奇数シーケンスは、1/4 波奇数対称性があると言います。SINQF および SINQB はそれぞれ単一の実数 1/4 波奇数シーケンスの FST およびその逆変換を計算し、VSINQF および VSINQB は 1 つまたは複数のシーケンスを処理します。SINQB は正規化されないため、SINQB の入力として SINQF の結果を使用すると、$4N$ の係数でスケーリングされた元のシーケンスが生成されます。ただし、VSINQB は正規化されるため、VSINQF の呼び出しのあとの VSINQB の呼び出しによって、元のシーケンスが生成されます。1/4 波奇数対称性がある長さ $2N$ の実数シーケンスの FST には、N 入

力データポイントが必要で、N ポイントの結果シーケンスを生成します。[V]SINQI を呼び出して変換ルーチンを呼び出す前に初期化が必要です。

高速コサイン変換

実数偶数シーケンスを処理する FFT の特殊な形式は、高速コサイン変換 (FCT) です。実数シーケンス x は、$x(n) = x(-n)$ の場合に偶数の対称性があると言われます。ここで $n = -N + 1, \cdots, 0, \cdots, N$ です。長さ $2N$ のシーケンスの FCT には $N + 1$ 入力データポイントが必要で、サイズ $N + 1$ のシーケンスを生成します。ルーチン COST は単一の実数偶数シーケンスの FCT を計算し、VCOST は 1 つ以上のシーケンスの FCT を計算します。[V]COST を呼び出す前に、[V]COSTI を呼び出して、入力長 N +1 に関連付けられる三角法の定数と係数を計算する必要があります。FCT はそれ自身の逆変換です。VCOST を 2 回呼び出すと、元の N +1 データポイントになります。COST を 2 回呼び出すと、$2N$ をかけた元の N +1 データポイントになります。

$n = -N + 1, \cdots, 0, \cdots, N$ の場合に、$x(n) = x(-n - 1)$ のような対称性を持つ偶数シーケンス x は、1/4 波偶数対称性があると言います。COSQF および COSQB は単一の実数 1/4 波偶数シーケンスの FCT とその逆変換をそれぞれ計算します。VCOSQF および VCOSQB は 1 つ以上のシーケンスを処理します。[V]COSQB の結果は正規化されず、$\frac{1}{4N}$ でスケーリングされた場合、元のシーケンスが取得されます。1/4 波実数対称性がある長さ $2N$ の実数シーケンスの FCT には、N の入力データポイントが必要で、N ポイントの結果シーケンスを生成します。変換ルーチンを呼び出す前に、[V]COSQI を呼び出して、初期化が必要です。

離散高速コサインおよびサイン変換とそれらの逆変換

Oracle Developer Studio パフォーマンスライブラリルーチンは、次のセクションの方程式を使用して、高速コサインおよびサイン変換と逆変換を計算します。

[D]COST: シーケンスの順方向および逆方向高速コサイン変換 (FCT)

シーケンスの順方向および逆方向 FCT は次のように計算されます。

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

[D]COST 注記:

- N ポイントシーケンスの FCT を計算するために、$N + 1$ の値が必要になります。
- [D]COST は逆変換も計算します。[D]COST が 2 回呼び出されると、結果は $\frac{1}{2N}$ でスケーリングされた元のシーケンスになります。

V[D]COST: 複数シーケンスの順方向および逆方向高速コサイン変換 (VFCT)

複数シーケンスの順方向および逆方向 FCT は次のように計算されます。

i = 0、*M* - 1 の場合

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

V[D]COST 注記

- *M N* ポイントシーケンスの VFCT を計算するために、*M* × (*N*+1) の値が必要になります。
- 入力および出力シーケンスは、行方向に格納されます。
- V[D]COST は正規化され、それ自身の逆変換です。V[D]COST が 2 回呼び出されると、結果は元のデータになります。

[D]COSQF: 1/4 波偶数シーケンスの順方向 FCT

1/4 波偶数シーケンスの順方向 FCT は次のように計算されます。

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

N ポイント 1/4 波偶数シーケンスの順方向 FCT を計算するために、*N* の値が必要です。

[D]COSQB: 1/4 波偶数シーケンスの逆方向 FCT

1/4 波偶数シーケンスの逆方向 FCT は次のように計算されます

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

順変換ルーチンと逆変換ルーチンを呼び出すと、$\frac{1}{4N}$ でスケーリングされた元の入力になります。

V[D]COSQF: 1 つ以上の 1/4 波偶数シーケンスの順方向 FCT

1 つまたは複数の 1/4 波偶数シーケンスの順方向 FCT は次のように計算されます

i = 0、*M* - 1 の場合

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

V[D]COSQF 注記:

- 入力および出力シーケンスは、行方向に格納されます。
- 変換は正規化されるため、V[D]COSQF の呼び出し後すぐに、逆変換ルーチン V[D]COSQB が呼び出された場合、元のデータが取得されます。

V[D]COSQB: 1 つ以上の 1/4 波偶数シーケンスの逆方向 FCT

1 つ以上の 1/4 波偶数シーケンスの逆方向 FCT は次のように計算されます

i = 0、*M* - 1 の場合

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

V[D]COSQB 注記:

- 入力および出力シーケンスは、行方向に格納されます。
- 変換は正規化されるため、V[D]COSQB が V[D]COSQF の呼び出し後すぐに呼び出された場合、元のデータが取得されます。

[D]SINT: シーケンスの順方向および逆方向高速サイン変換 (FST)

シーケンスの順方向および逆方向 FST は次のように計算されます

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

.

[D]SINT 注記:

- *N* ポイントシーケンスの FST を計算するために、*N*-1 の値が必要です。
- [D]SINT は逆変換も計算します。[D]SINT が 2 回呼び出された場合、結果は $\frac{1}{2N}$ でスケーリングされた元のシーケンスになります。

V[D]SINT: 複数シーケンスの順方向および逆方向高速サイン変換 (VFST)

複数シーケンスの順方向および逆方向高速サイン変換は次のように計算されます

i = 0、*M* - 1 の場合

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

V[D]SINT 注記:

- *M N* ポイントシーケンスの VFST を計算するために、*M* × (*N* - 1) の値が必要です。
- 入力および出力シーケンスは、行方向に格納されます。
- V[D]SINT は正規化され、それ自身の逆変換です。V[D]SINT を 2 回呼び出すと、元のデータが生成されます。

[D]SINQF: 1/4 波奇数シーケンスの順方向 FST

1/4 波奇数シーケンスの順方向 FST は次のように計算されます

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

N ポイント 1/4 波奇数シーケンスの順方向 FST を計算するために、*N* の値が必要になります。

[D]SINQB: 1/4 波奇数シーケンスの逆方向 FST

1/4 波奇数シーケンスの逆方向 FST は次のように計算されます

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

順変換ルーチンと逆変換ルーチンを呼び出すと、$\frac{1}{4N}$ でスケーリングされた元の入力になります。

V[D]SINQF: 1 つ以上の 1/4 波奇数シーケンスの順方向 FST

1 つ以上の 1/4 波奇数シーケンスの順方向 FST は次のように計算されます

$i = 0, M - 1$ の場合

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

V[D]SINQF 注記:

- 入力および出力シーケンスは、行方向に格納されます。
- 変換は正規化されるため、V[D]SINQF の呼び出し後すぐに、逆変換ルーチン V[D]SINQB が呼び出された場合、元のデータが取得されます。

V[D]SINQB: 1 つ以上の 1/4 波奇数シーケンスの逆方向 FST

1 つ以上の 1/4 波奇数シーケンスの逆方向 FST は次のように計算されます

$i = 0, M - 1$ の場合

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

V[D]SINQB 注記:

- 入力および出力シーケンスは、行方向に格納されます。
- 変換は正規化されるため、V[D]SINQB が V[D]SINQF の呼び出し後すぐに呼び出された場合、元のデータが取得されます。

高速コサイン変換の例

例15「単一の実数偶数シーケンスの FCT と逆方向 FCT の計算」は、COST を呼び出して、実数偶数シーケンスの FCT と逆変換を計算します。実数シーケンスの長さが $2N$ の場合、$N + 1$ の入力データポイントのみを格納する必要があり、結果のデータポイントの数も $N + 1$ になります。結果は入力配列に格納されます。

例 15 単一の実数偶数シーケンスの FCT と逆方向 FCT の計算

```
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「2 つの実数 1/4 波偶数シーケンスの FCT および逆方向 FCT の計算」では、VCOSQF および VCOSQB を呼び出して、2 つの実数 1/4 波偶数シーケンスの FCT と逆方向 FCT をそれぞれ計算しています。実数シーケンスの長さが 2*N* の場合、*N* の入力データポイントのみを格納する必要があり、結果のデータポイントの数も *N* になります。結果は入力配列に格納されます。

例 16 2 つの実数 1/4 波偶数シーケンスの FCT および逆方向 FCT の計算

```
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 )
```

高速サイン変換の例

次の例の例17「2 つの実数 1/4 波偶数シーケンスの FCT および逆方向 FCT の計算」では、SINT を呼び出して、実数奇数シーケンスの FST と逆変換を計算しています。実数シーケンスの長さが 2*N* の場合、*N* - 1 の入力データポイントのみを格納する必要があり、結果のデータポイントの数も *N*- 1 になります。結果は入力配列に格納されます。

例 17 2 つの実数 1/4 波偶数シーケンスの FCT および逆方向 FCT の計算

```
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
```

次の例の例18「2 つの実数 1/4 波奇数シーケンスの FST および逆方向 FST の計算」では、VSINQF および VSINQB を呼び出して、2 つの実数 1/4 波奇数シーケンスの FST と逆方向 FST をそれぞれ計算しています。実数シーケンスの長さが 2*N* の場合、*N* の入力データポイントのみを格納する必要があり、結果のデータポイントの数も *N* になります。結果は入力配列に格納されます。

例 18 2 つの実数 1/4 波奇数シーケンスの FST および逆方向 FST の計算

```
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 )
```

畳み込みおよび相関

信号処理分野で特に頻繁に見られる FFT の 2 つの用途は、離散畳み込み演算と離散相関演算です。

畳み込み演算

2 つの関数 $x(t)$ と $y(t)$ がある場合、$x(t)$ および $y(t)$ の畳み込みのフーリエ変換は、x * y で示され、個々のフーリエ変換の積になります。DFT (x * y)=$X(\cdot)Y$ (* は畳み込み演算を示し、(·) は点別乗算を示します)。

一般に、$x(t)$ は通常、等間隔での $x(t)$ の一周期の有限期間でサンプリングされた一連の N データポイント x_j, $j = 0, \cdots, N-1$ によって離散的に表される連続した定期的な信号です。$y(t)$ は通常、ゼロから始まり、最大値に達したあとにゼロに戻る応答です。等間隔での $y(t)$ の離散化は、一連の N データポイント、y_k, $k = 0, \cdots, N-1$ を生成します。y_k の実際のサンプリングの数が N 未満である場合、データはゼロでパディングできます。離散畳み込みは次のように定義できます

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

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

は $k = 0, \cdots, N-1$ の値と同じですが、循環した順序です。

Oracle Developer Studio パフォーマンスライブラリルーチンにより、$k = 0, \cdots, N-1$ で上記の定義を使用するか、または FFT を使用して、畳み込みを計算できます。FFT を使用して、2 つのシーケンスの畳み込みを計算する場合、次の手順を実行します。

- X = x の順方向 FFT を計算します
- Y = y の順方向 FFT を計算します
- $Z = X (\cdot) Y <=> DFT (x * y)$ を計算します
- $z = Z$; $z = (x * y)$ の逆方向 FFT を計算します

畳み込みの興味深い特性の 1 つは、2 つの多項式の積が実際の畳み込みであることです。m 項の多項式

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

と n 項の多項式

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

の積には、次によって取得可能な $m+n-1$ 係数があります。

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

ここで $k = 0, \cdots, m + n - 2$ です。

相関演算

畳み込みに密接に関係があるのが、相関演算です。これは、直接に重複しているか、一方が他方と相対的にシフトされている場合に、2 つのシーケンスの相関を計算します。畳み込みと同様に、FFT を使用して、次のように 2 つのシーケンスの相関を効率的に計算できます。

- 2 つの入力シーケンスの FFT を計算します。
- 一方のシーケンスの結果の変換と他方の変換のシーケンスの複素共役の点別の積を計算します。
- 積の逆方向 FFT を計算します。

パフォーマンスライブラリ内のルーチンにより、次の定義で相関を計算することもできます。

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

畳み込み演算と相関演算のサンプリング入力データを解釈する方法は、数多くあります。畳み込みルーチンと相関ルーチンの引数リストには、次のケースを処理するためのパラメータが含まれます。

- 信号関数や応答関数は、さまざまなサンプリング時間に開始できます。
- 信号の一部のみを出力に含めることもできます。
- 信号関数や応答関数は、明示的に格納されていない 1 つ以上のゼロから開始できます。

Oracle Developer Studio パフォーマンスライブラリの畳み込みルーチンと相関ルーチン

Oracle Developer Studio パフォーマンスライブラリには、表19「畳み込みおよび相関ルーチン」に示す畳み込みルーチンが含まれています。

表 19 畳み込みおよび相関ルーチン

ルーチン	引数	機能
SCNVCOR、DCNVCOR、CCNVCOR、ZCNVCOR	CNVCOR,FOUR,NX,X,IFX, INCX,NY, NPRE,M,Y,IFY, INC1Y,INC2Y,NZ,K, Z, IFZ,INC1Z,INC2Z,WORK, LWORK	1 つ以上のベクトルによるフィルタの畳み込みまたは相関
SCNVCOR2、DCNVCOR2、CCNVCOR2、ZCNVCOR2	CNVCOR,METHOD,TRANSX, SCRATCHX, TRANSY, SCRATCHY,MX,NX,X,LDX, MY,NY,MPRE,NPRE,Y,LDY, MZ,NZ,Z, LDZ,WORKIN, LWORK	2 つの行列の 2 次元畳み込みまたは相関
SWIENER、DWIENER	N_POINTS,ACOR,XCOR, FLTR,EROP, ISW,IERR	2 つの信号のウィーナーデコンボリューション

[S,D,C,Z]CNVCOR ルーチンは、1 つ以上の入力ベクトルでフィルタの畳み込みや相関を計算するために使用されます。[S,D,C,Z]CNVCOR2 ルーチンは、2 つの行列の 2 次元畳み込みまたは相関を計算するために使用します。

畳み込みルーチンと相関ルーチンの引数

1 次元の畳み込みルーチンと相関ルーチンでは、表20「1 次元畳み込みおよび相関ルーチン SCNVCOR、DCNVCOR、CCNVCOR、および ZCNVCOR の引数」に示す引数を使用します。

表 20 1 次元畳み込みおよび相関ルーチン SCNVCOR、DCNVCOR、CCNVCOR、および ZCNVCOR の引数

引数	定義
CNVCOR	'V' または 'v' は、畳み込みを計算することを指定します。'R' または 'r' は相関を計算することを指定します。

引数	定義
FOUR	'T' または 't' はフーリエ変換法を使用することを指定します。'D' または 'd' は、畳み込みや相関を畳み込みと相関の定義から計算する直接法を使用することを指定します。†
NX	フィルタベクトルの長さで、NX ≥ 0 です。
X	フィルタベクトル
IFX	X の最初の要素のインデックスで、NX ≥ IFX ≥ 1
INCX	X のベクトルの要素間の刻み幅で、INCX > 0 です。
NY	入力ベクトルの長さで、NY ≥ 0 です。
NPRE	Y ベクトルの前に付ける暗黙的なゼロの数で、NPRE ≥ 0 です。
M	入力ベクトルの数で、M ≥ 0 です。
Y	入力ベクトル。
IFY	Y の最初の要素のインデックスで、NY ≥ IFY ≥ 1
INC1Y	Y の入力ベクトルの要素間の刻み幅で、INC1Y > 0 です。
INC2Y	Y の入力ベクトル間の刻み幅で、INC2Y > 0 です。
NZ	出力ベクトルの長さで、NZ ≥ 0 です。
K	Z ベクトルの数で、K ≥ 0 です。K < M である場合、最初の K ベクトルのみが処理されます。K > M である場合、すべての入力ベクトルが処理され、終了時に、最後の M-K 出力ベクトルにゼロが設定されます。
Z	結果ベクトル
IFZ	Z の最初の要素のインデックスで、NZ ≥ IFZ ≥ 1
INC1Z	Z の出力ベクトルの要素間の刻み幅で、INCYZ > 0 です。
INC2Z	Z の出力ベクトル間の刻み幅で、INC2Z > 0 です。
WORK	作業配列
LWORK	作業配列の長さ

† 畳み込む 2 つのシーケンスの長さが似ている場合、FFT 法は直接法より高速です。ただし、小さいフィルタで大きな時系列信号を畳み込む場合など、一方のシーケンスが他方よりはるかに大きい場合、FFT ベースの方法よりも直接法のほうが実行が速くなります。

2 次元の畳み込みルーチンと相関ルーチンは、表21「2 次元畳み込みおよび相関ルーチン SCNVCOR2、DCNVCOR2、CCNVCOR2、および ZCNVCOR2 の引数」に示す引数を使用します。

表 21 2 次元畳み込みおよび相関ルーチン SCNVCOR2、DCNVCOR2、CCNVCOR2、および ZCNVCOR2 の引数

引数	定義
CNVCOR	'V' または 'v' は、畳み込みを計算することを指定します。'R' または 'r' は相関を計算することを指定します。
METHOD	'T' または 't' はフーリエ変換法を使用することを指定します。'D' または 'd' は、畳み込みや相関を畳み込みと相関の定義から計算する直接法を使用することを指定します。†
TRANSX	'N' または 'n' は、X が最初の行列であることを指定し、'T' または 't' は、X の転置行列がフィルタ行列であることを指定します

引数	定義
SCRATCHX	'N' または 'n' は、X を保持する必要があることを指定し、'S' または 's' は X をスクラッチスペースに使用できることを指定します。X がスクラッチスペースに使用された呼び出しから戻ったあと、X の内容は未定義になります。
TRANSY	'N' または 'n' は Y が入力行列であることを指定し、'T' または 't' は Y の転置行列が入力行列であることを指定します
SCRATCHY	'N' または 'n' は、Y を保持する必要があることを指定し、'S' または 's' は Y をスクラッチスペースに使用できることを指定します。Y がスクラッチスペースに使用された呼び出しから戻ったあと、X の内容は未定義になります。
MX	フィルタ行列 X の行数で、MX ≥ 0
NX	フィルタ行列 X の列数で、NX ≥ 0
X	フィルタ行列。X は SCRATCHX が 'N' または 'n' の場合に終了時に変更されず、SCRATCHX が 'S' または 's' の場合は終了時に未定義になります。
LDX	フィルタ行列 X を格納する配列のリーディングディメンジョン。
MY	入力行列 Y の行数で、MY ≥ 0 です。
NY	入力行列 Y の列数で、NY ≥ 0 です。
MPRE	入力行列 Y ベクトルの各行の前に付ける暗黙的なゼロの数で、MPRE ≥ 0 です。
NPRE	入力行列 Y の各列の前に付ける暗黙的なゼロの数で、NPRE ≥ 0 です。
Y	入力行列。Y は SCRATCHY が 'N' または 'n' の場合に終了時に変更されず、SCRATCHY が 'S' または 's' の場合は終了時に未定義になります。
LDY	入力行列 Y を格納する配列のリーディングディメンジョン。
MZ	出力ベクトルの数で、MZ ≥ 0 です。
NZ	出力ベクトルの長さで、NZ ≥ 0 です。
Z	結果ベクトル
LDZ	結果の行列 Z を格納する配列のリーディングディメンジョンで、LDZ ≥ MAX(1,MZ) です。
WORKIN	作業配列
LWORK	作業配列の長さ

† 畳み込む 2 つの行列のサイズが似ている場合、FFT 法は直接法より高速です。ただし、小さいフィルタで大きなデータセットを畳み込む場合など、一方のシーケンスが他方よりはるかに大きい場合、FFT ベースの方法よりも直接法のほうが実行が速くなります。

畳み込みルーチンと相関ルーチンの作業配列 WORK

1 次元および 2 次元畳み込みルーチンと相関ルーチンで使用される WORK 作業配列の最小次元は、表24「畳み込みルーチンと相関ルーチンで使用される WORK 作業配列の最小次元とデータ型」に示しています。1 次元畳み込みルーチンと相関ルーチンの最小次元は、引数 NPRE、NX、NY、および NZ の値によって異なります。

2 次元畳み込みルーチンと相関ルーチンの最小次元は、表22「2 次元ルーチンの最小作業配列サイズに影響する引数: SCNVCOR2、DCNVCOR2、CCNVCOR2、ZCNVCOR2」に示す引数の値によって異なります。

表 22 2 次元ルーチンの最小作業配列サイズに影響する引数: SCNVCOR2、DCNVCOR2、CCNVCOR2、ZCNVCOR2

引数	定義
MX	フィルタ行列の行数
MY	入力行列の行数
MZ	出力ベクトル数
NX	フィルタ行列の列数
NY	入力行列の列数
NZ	出力ベクトルの長さ
MPRE	入力行列の各行の前に付ける暗黙的なゼロの数
NPRE	入力行列の各列の前に付ける暗黙的なゼロの数
MPOST	MAX(0,MZ-MYC)
NPOST	MAX(0,NZ-NYC)
MYC	MPRE + MPOST + MYC_INIT、ここで MYC_INIT は、表23「MYC_INIT および NYC_INIT の依存関係」に示すように、フィルタ行列と入力行列によって異なります。
NYC	NPRE + NPOST + NYC_INIT、ここで NYC_INIT は、表23「MYC_INIT および NYC_INIT の依存関係」に示すように、フィルタ行列と入力行列によって異なります。

MYC_INIT および NYC_INIT は次によって異なります。ここで、X はフィルタ行列で、Y は入力行列です。

表 23 MYC_INIT および NYC_INIT の依存関係

ルーチン	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)

最小の作業配列サイズに割り当てられる値を、表24「畳み込みルーチンと相関ルーチンで使用される WORK 作業配列の最小次元とデータ型」に示します。

表 24 畳み込みルーチンと相関ルーチンで使用される WORK 作業配列の最小次元とデータ型

ルーチン	最小作業配列サイズ (WORK)	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 †	MY = NY の場合: MYC + 8、MY ≥ NY の場合: MYC + NYC + 16	COMPLEX、COMPLEX*16

1

サンプルプログラム: 畳み込み

次の例は、CCNVCOR を使用して 2 つの複素数ベクトルの FFT 畳み込みを実行します。

例 19 フーリエ変換法と複素数データを使用した 1 次元畳み込み

```
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
```

引数の別名化またはさまざまな INC 引数の不適切に選択した値のため、いずれかのベクトルが書き込み可能なベクトルに重複している場合、結果は未定義になり、実行のたびにさまざまに異なることがあります。

[1] LWORK によって示されるワークスペースのサイズが十分に大きくない場合、ルーチン内でメモリーが割り当てられます。

もっとも一般的な計算の形式で、もっとも高速に実行するケースは、フィルタベクトル X を Y の列に格納されている一連のベクトルに適用し、結果を Z の列に配置することです。この場合、INCX = 1、INC1Y = 1、INC2Y ≥ NY、INC1Z = 1、INC2Z ≥ NZ です。別の一般的な形式は、フィルタベクトル X を Y の行に格納されている一連のベクトルに適用し、結果を Z の行に格納することです。この場合、INCX = 1、INC1Y ≥ NY、INC2Y = 1、INC1Z ≥ NZ、INC2Z = 1 です。

畳み込みを使用して、多項式の積を計算できます。次の例の例20「フーリエ変換法と実数データを使用した 1 次元畳み込み」は SCNVCOR を使用して、$1 + 2x + 3x^2$ と $4 + 5x + 6x^2$ の積を計算しています。

例 20 フーリエ変換法と実数データを使用した 1 次元畳み込み

```
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.
```

上の例のように、出力ベクトルを入力ベクトルより大きくすると、入力の末尾に暗黙的にゼロが追加されます。すべてのベクトルで実際にゼロは必要なく、例ではまったく使用されていませんが、暗黙的なゼロで行われるパディングは、入力ベクトルの循環桁送りではなく、切り捨て桁送りの効果があります。

次の例の例21「ベクトルと巡回行列の積を計算するために使用される畳み込み」では、ベクトル [1、2、3] と初期列ベクトル [4、5、6] で定義された巡回行列間の積を計算しています。

例 21 ベクトルと巡回行列の積を計算するために使用される畳み込み

```
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.
```

次の例と前の例の違いは、出力ベクトルの長さが入力ベクトルの長さと同じであるため、入力ベクトルの末尾に暗黙的なゼロがありません。桁送りする暗黙的なゼロがないと、前の例の切り捨て桁送りの効果は発生せず、循環桁送りは巡回行列の積になります。

例 22 直接法を使用した 2 次元折り畳み

```
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
```

参考資料

DFT または FFT の詳細については、次のソースを参照してください。

Briggs, William L.、および Henson, Van Emden 著、『*The DFT: An Owner's Manual for the Discrete Fourier Transform*』、Philadelphia, PA: SIAM 発行、1995 年。

Brigham, E. Oran 著、『*The Fast Fourier Transform and Its Applications*』、Upper Saddle River, NJ: Prentice Hall 発行、1988 年。

Chu, Eleanor および George, Alan 著、『*Inside the FFT Black Box: Serial and Parallel Fast Fourier Transform Algorithms*』、Boca Raton, FL: CRC Press 発行、2000 年。

Press, William H.、Teukolsky, Saul A.、Vetterling, William T.、および Flannery, Brian P. 著、『*Numerical Recipes in C: The Art of Scientific Computing*』第 2 版、Cambridge, United Kingdom: Cambridge University Press 発行、1992 年。

Ramirez, Robert W. 著、『*The FFT: Fundamentals and Concepts*』、Englewood Cliffs, NJ: Prentice-Hall, Inc. 発行、1985 年。

Swartzrauber, Paul N. 著、Vectorizing the FFTs、In Rodrigue, Garry ed.『*Parallel Computations*』、New York: Academic Press, Inc. 発行、1982 年。

Strang, Gilbert 著、『*Linear Algebra and Its Applications*』第 3 版.、Orlando, FL: Harcourt Brace & Company 発行、1988 年。

Van Loan, Charles 著、『*Computational Frameworks for the Fast Fourier Transform*』、Philadelphia, PA: SIAM 発行、1992 年。

Walker, James S. 著、『*Fast Fourier Transforms*』、Boca Raton, FL: CRC Press 発行、1991 年。

♦ ♦ ♦ 付 録 A

Oracle Developer Studio パフォーマンスライブラリのルーチン

この付録では、Oracle Developer Studio パフォーマンスライブラリのルーチンをライブラリ、ルーチン名、および機能別に示します。

機能の説明と Fortran および C のインタフェースのリストについては、個々のルーチンのセクション 3P マニュアルページを参照してください。たとえば、SBDSQR ルーチンのマニュアルページを表示するには、**man -s 3P sbdsqr** と入力します。マニュアルページのルーチン名は小文字を使用します。

多くのルーチンには、異なるデータ型で動作する別々のルーチンが存在します。各ルーチンを個別に示す代わりに、ルーチン名に小文字の *x* を使用して、単精度、倍精度、複素数、および倍精度複素数データ型を表します。たとえば、ルーチン *x*BDSQR の場合、次のデータ型で動作する 4 つのルーチンがあります。

- SBDSQR – 単精度データ型
- DBDSQR – 倍精度データ型
- CBDSQR – 複素数データ型
- ZBDSQR – 倍精度複素数データ型

S、D、C、および Z のルーチン名が使用できない場合、接頭辞 x は使用せず、各ルーチン名を示します。64 ビット対応のオペレーティング環境にも、対応する 64 ビットルーチンがあります (リストには示されていません)。それらの名前は接尾辞 _64 で表されます。たとえば、64 ビットバージョンの *x*BDSQR は次のとおりです。

- SBDSQR_64
- DBDSQR_64
- CBDSQR_64
- ZBDSQR_64

LAPACK ルーチン

次の一連の表に、Oracle Developer Studio パフォーマンスライブラリの LAPACK ルーチンを示します。(P) はルーチンが並列化されていることを表します。

- 表25「二重対角行列ルーチン」
- 表26「共通ルーチンまたは計算ルーチン」
- 表27「コサイン-サイン (CS) 分解ルーチン」
- 表28「対角行列ルーチン」
- 表29「一般帯行列ルーチン」
- 表30「一般行列 (非対称または長方) ルーチン」
- 表31「一般行列 - 一般化問題 (一般行列のペア) ルーチン」
- 表32「一般三重対角行列ルーチン」
- 表33「エルミート帯行列ルーチン」
- 表34「エルミート行列ルーチン」
- 表35「パック格納のエルミート行列ルーチン」
- 表36「上ヘッセンベルグ行列ルーチン」
- 表37「上ヘッセンベルグ行列 - 一般化問題 (ヘッセンベルグと三角行列) ルーチン」
- 表38「パック格納の実直交行列ルーチン」
- 表39「実直交行列ルーチン」
- 表40「対称/エルミート正定値帯行列ルーチン」
- 表41「対称/エルミート正定値行列ルーチン」
- 表42「パック格納の対称/エルミート正定値行列ルーチン」
- 表43「対称/エルミート正定値三重対角行列ルーチン」
- 表44「実対称帯行列ルーチン」
- 表45「パック格納の対称行列ルーチン」
- 表46「実対称三重対角行列ルーチン」
- 表47「対称行列ルーチン」
- 表48「三角帯行列ルーチン」
- 表49「三角行列 - 一般化問題 (三角行列のペア) ルーチン」
- 表50「パック格納の三角行列ルーチン」
- 表51「Rectangular Full-Packed (RFP) 形式および標準パック形式の三角行列ルーチン」
- 表52「三角行列ルーチン」
- 表53「台形行列ルーチン」
- 表54「ユニタリ行列ルーチン」
- 表55「パック格納のユニタリ行列ルーチン」

表 25 二重対角行列ルーチン

ルーチン	機能
SBDSDC (P) または DBDSDC (P)	分割統治法を使用して、二重対角行列の特異値分解 (SVD) を計算します。

ルーチン	機能
*x*BDSQR	陰的シフトなし QR 法を使用して、実上/下二重対角行列の SVD を計算します。
SLARTGS または DLARTGS	二重対角行列の SVD 問題の陰的 QR 反復にバルジを導入するために設計された平面回転を生成します。SBBCSD または DBBCSD によって使用されます。

表 26 共通ルーチンまたは計算ルーチン

ルーチン	機能
CHLA_TRANSTYPE	BLAST 指定の整数定数を、転置演算を指定する文字列に変換します。
CLA_HERPVGRW (P) または ZLA_HERPVGRW (P)	複素エルミート行列のピボット拡大係数の逆数 norm(A)/norm(U) を計算します。
ILADIAG	行列が単位対角を持つかどうかを指定する文字列を、関連する BLAST 指定の整数定数に変換します。
ILAPREC	中間精度を指定する文字列を、関連する BLAST 指定の整数定数に変換します。
ILATRANS	転置演算を指定する文字列を、関連する BLAST 指定の整数定数に変換します。
ILAENV	問題に依存するパラメータをローカル環境に応じて選択するために、LAPACK ルーチンから呼び出されます。
ILAUPLO	上/下三角行列を指定する文字列を、関連する BLAST 指定の整数定数に変換します。
ILAVER	LAPACK のバージョンを返します。
*x*LA_GBRPVGRW	実/複素一般帯行列のピボット拡大係数の逆数 norm(A)/norm(U) を計算します。
*x*LA_GERPVGRW (P)	一般不定値行列のピボット拡大係数の逆数 norm(A)/norm(U) を計算します。
*x*LA_PORPVGRW (P)	実対称/エルミート正定値行列のピボット拡大係数の逆数 norm(A)/norm(U) を計算します。
*x*LA_SYRPVGRW (P)	実/複素対称不定値行列のピボット拡大係数の逆数 norm(A)/norm(U) を計算します。
SLAMRG (P) または DLAMRG (P)	個別にソートされた 2 セットのエントリを昇順にソートされた 1 つのセットにマージするための置換リストを作成します。
CLANHF (P) または ZLANHF (P)	RFP 形式のエルミート行列の 1 ノルム、フロベニウスノルム、無限大ノルム、または最大絶対値要素の値を返します。
SLANSF (P) または DLANSF (P)	RFP 形式の実対称行列の 1 ノルム、フロベニウスノルム、無限大ノルム、または最大絶対値要素の値を返します。
*x*LARSCL2 (P)	ベクトルに対して逆数の対角スケーリングを実行します。
*x*LASCL2 (P)	ベクトルに対して対角スケーリングを実行します。
SLASQ1 または DLASQ1	実正方二重対角行列の特異値を計算します。SBDSQR または DBDSQR によって使用されます。
SLASQ2 または DLASQ2	実対称正定値三重対角行列 (高い相対精度) のすべての固有値を計算します。SBDSQR と SSTEGR または DBDSQR と DSTEGR によって使用されます。
SLASQ3 または DLASQ3	減次をチェックし、シフトを計算し、DQDS 法を呼び出します。SBDSQR または DBDSQR によって使用されます。
SLASQ4 または DLASQ4	以前の変換の値を使用して、最小固有値への近似を計算します。SBDSQR または DBDSQR によって使用されます。
SLASQ5 または DLASQ5	反復形式の DQDS 変換を 1 つ計算します。SBDSQR と SSTEGR または DBDSQR と DSTEGR によって使用されます。

ルーチン	機能
SLASQ6 または DLASQ6	アンダーフローおよびオーバーフローに対する保護付きで、反復形式の DQD 変換 (シフトは 0 に等しい) を 1 つ計算します。SBDSQR と SSTEGR または DBDSQR と DSTEGR によって使用されます。
SLASRT または DLASRT	ベクトル内の数値を昇順または降順にソートします。
*x*LATRZ (P)	実/複素上台形行列を直交変換によって因子分解します。
CROT、ZROT	ギブンス平面回転を適用します。SROT/DROT はレベル 1 BLAS に含まれています。

表 27 コサイン-サイン (CS) 分解ルーチン

ルーチン	機能
*x*BBCSD (P)	二重対角ブロック形式のユニタリ/直交行列の CS 分解を計算します。
SORCSD (P) または DORCSD (P)	実分割直交行列の CS 分解を計算します。
SORCSD2BY1 または DORCSD2BY1 (P)	正規直交列を持ち、2 × 1 ブロック構造に分割されている M × Q 行列の CS 分解を計算します。
CUNCSD (P) または ZUNCSD (P)	M × M 分割ユニタリ行列の CS 分解を計算します。

表 28 対角行列ルーチン

ルーチン	機能
SDISNA (P) または DDISNA (P)	実対称/複素エルミート行列の固有ベクトルに関する条件数の逆数を計算します。

表 29 一般帯行列ルーチン

ルーチン	機能
CGBBRD または ZGBBRD	複素一般帯行列を直交変換によって上二重対角形式に縮約します。
SGBBRD (P) または DGBBRD (P)	実一般帯行列を直交変換によって上二重対角形式に縮約します。
*x*GBCON	LU 因子分解を使用して、一般帯行列の条件数の逆数を推定します。
*x*GBEQU (P)	一般帯行列を均衡化し、その条件数を減らすように、行および列のスケーリングを計算します。
*x*GBEQUB (P)	一般帯行列を均衡化し、その条件数を減らすように、行および列のスケーリングを計算します。スケーリング係数が基数のべき乗に限定される点で、CGEEQU とは異なります。
*x*GBRFS (P)	係数行列が帯行列である場合に、連立 1 次方程式の計算解を改良し、解の誤差限界と後退誤差推定を提供します。
*x*GBRFSX (P)	帯行列の連立 1 次方程式の計算解を改良し、誤差限界と後退誤差推定を提供します。このコードは、ノルムから見た誤差限界に加え、可能な場合は最大成分から見た誤差限界も提供します。
*x*GBSV	一般帯行列の連立 1 次方程式を解きます (単純ドライバ)。
*x*GBSVX (P)	一般帯行列の連立 1 次方程式を解きます (エキスパートドライバ)。

ルーチン	機能
*x*GBSVXX (P)	一般帯行列の連立 1 次方程式を解きます (エキスパートドライバ、高精度)。要求された場合は、ノルムから見た誤差限界と最大成分から見た誤差限界の両方を返します。
*x*GBTF2 (P)	部分ピボットおよび行の入れ替えを使用して、実/複素一般帯行列の LU 因子分解を計算します (非ブロック化アルゴリズム)。
*x*GBTRF (P)	部分ピボットおよび行の入れ替えを使用して、一般帯行列の LU 因子分解を計算します。
*x*GBTRS	*x*GBTRF で計算された因子分解を使用して、一般帯行列の連立 1 次方程式を解きます。
*x*LA_GBAMV	実/複素帯行列の誤差限界を計算する行列-ベクトル演算を実行します。
*x*LA_GBRFSX_EXTENDED	高精度反復改良を実行することによって実/複素一般帯行列の連立 1 次方程式の計算解を改良し、解の誤差限界と後退誤差推定を提供します。

表 30 一般行列 (非対称または長方) ルーチン

ルーチン	機能
SGEJSV (P) または DGEJSV (P)	実一般行列の特異値分解 (SVD) を計算します。
DSGESV	実一般行列の連立 1 次方程式の解を計算します (反復改良を使用した混合精度)。
SGESVJ (P) または DGESVJ (P)	実一般行列の特異値分解 (SVD) を計算します。前処理付きヤコビ SVD 法を実装します。SGEQP3、SGEQR、および SGELQF または DGEQP3、DGEQRF、および DGELQF をプリプロセッサとして使用するため、精度が向上する可能性があります。
ZCGESV	複素一般行列の連立 1 次方程式の解を計算します (反復改良を使用した混合精度)。
ZCPOSV	複素正定値行列の連立 1 次方程式の解を計算します (反復改良を使用した混合精度)。
*x*GEBAK	*x*GEBAL から出力された均衡化行列の計算された固有ベクトルに後退変換を行うことによって、一般行列の右/左固有ベクトルを生成します。
*x*GEBAL (P)	実/複素一般行列を均衡化します。
*x*GEBD2	一般行列を二重対角形式に縮約します (非ブロック化アルゴリズム)。
*x*GEBRD	一般行列をユニタリ/直交変換によって上/下二重対角形式に縮約します (ブロック化アルゴリズム)。
*x*GECON	*x*GETRF で計算された因子分解を使用して、一般行列の条件数の逆数を推定します。
*x*GEEQU (P)	一般長方行列を均衡化し、その条件数を減らすように、行および列のスケーリングを計算します。
*x*GEEQUB (P)	一般長方行列を均衡化し、その条件数を減らすように、行および列のスケーリングを計算します。スケーリング係数が基数のべき乗に限定される点で、*x*GETRF とは異なります。
*x*GEES	一般行列の固有値およびシュール因子分解を計算します (単純ドライバ)。
*x*GEESX	一般行列の固有値とシュール因子分解を計算します (エキスパートドライバ)。
*x*GEEV (P)	一般行列の固有値と左および右固有ベクトルを計算します (単純ドライバ)。
*x*GEEVX (P)	一般行列の固有値と左および右固有ベクトルを計算します (エキスパートドライバ)。
*x*GEGS	*x*GGES に置き換えられた非推奨のルーチン。
*x*GEGV (P)	*x*GGEV に置き換えられた非推奨のルーチン。

ルーチン	機能
*x*GEHD2	一般正方行列をユニタリ/直交相似変換によって上ヘッセンベルグ形式に縮約します (非ブロック化アルゴリズム)。
*x*GEHRD (P)	一般行列を直交相似変換によって上ヘッセンベルグ形式に縮約します。
*x*GELQ2	実/複素一般長方行列の LQ 因子分解を計算します (非ブロック化アルゴリズム)。
*x*GELQF	一般長方行列の LQ 因子分解を計算します。
*x*GELS (P)	A の QR または LQ 因子分解を使用して、優決定系の連立 1 次方程式の最小二乗解を計算します。
*x*GELSD	分割統治法と A の QR または LQ 因子分解を使用して、優決定系の連立 1 次方程式の最小二乗解を計算します。
*x*GELSS	一般長方行列の SVD を使用して、線形最小二乗問題の最小ノルム解を計算します (単純ドライバ)。
*x*GELSX (P)	*x*SELSY に置き換えられた非推奨のルーチン。
*x*GELSY (P)	完全直交因子分解を使用して、線形最小二乗問題の最小ノルム解を計算します。
*x*GEMQRT	一般行列を、直交行列で変換した結果で上書きします。この直交行列は、*x*GEQRT から返されるコンパクト WY 表現を使用して生成された基本鏡映行列どうしの積として定義されます。
*x*GEQL2	実/複素一般長方行列の QL 因子分解を計算します (非ブロック化アルゴリズム)。
*x*GEQLF	実/複素一般長方行列の QL 因子分解を計算します。
*x*GEQP3	レベル 3 BLAS を使用して、一般長方行列の QR 因子分解を計算します。
*x*GEQPF	*x*GEQP3 に置き換えられた非推奨のルーチン。
*x*GEQR2	実/複素一般長方行列の QR 因子分解を計算します (非ブロック化アルゴリズム)。
*x*GEQR2P	非負対角要素を持つ実/複素一般長方行列の QR 因子分解を計算します (非ブロック化アルゴリズム)。
*x*GEQRFP	実/複素一般長方行列の QR 因子分解を計算します。
*x*GEQRT	Q のコンパクト WY 表現を使用して、実/複素一般行列のブロック化 QR 因子分解を計算します。
*x*GEQRT2	Q のコンパクト WY 表現を使用して、実/複素一般行列の QR 因子分解を計算します。
*x*GEQRT3 (P)	Q のコンパクト WY 表現を使用して、実/複素一般行列の QR 因子分解を再帰的に計算します。
*x*GERFS (P)	連立 1 次方程式の解を改良します。
*x*GERFSX (P)	連立 1 次方程式の計算解を改良し、解の誤差限界と後退誤差推定を提供します (高精度)。
*x*GERQ2	非ブロック化アルゴリズムを使用して、実/複素一般長方行列の RQ 因子分解を計算します。
*x*GERQF	実/複素一般長方行列の RQ 因子分解を計算します。
*x*GESDD	分割統治法を使用して、実/複素一般長方行列の特異値分解 (SVD) を計算します (ドライバ)。
*x*GESV	一般行列の連立 1 次方程式を解きます (単純ドライバ)。
*x*GESVD	実/複素一般行列の特異値分解 (SVD) を計算します (ドライバ)。
SGESVJ または DGESVJ	実一般長方行列の特異値分解 (SVD) を計算します。
*x*GESVX (P)	一般行列の連立 1 次方程式を解きます (エキスパートドライバ)。

ルーチン	機能
*x*GESVXX (P)	一般行列の連立 1 次方程式の解を計算します (高精度)。
*x*GETF2	部分ピボットおよび行の入れ替えを使用して、実/複素一般行列の LU 因子分解を計算します (非ブロック化アルゴリズム)。
*x*GETRF (P)	部分ピボットおよび行の入れ替えを使用して、一般長方行列の LU 因子分解を計算します。
*x*GETRI	*x*GETRF で計算された因子分解を使用して、一般行列の逆行列を計算します。
*x*GETRS	*x*GETRF で計算された因子分解を使用して、一般行列の連立 1 次方程式を解きます。
SGSVJ0 (P) または DGSVJ0 (P)	SGESVJ または DGESVJ のプリプロセッサ。特定のピボットのみを対象としてヤコビ回転を適用します。
SGSVJ1 (P) または DGSVJ1 (P)	SGESVJ または DGESVJ のプリプロセッサ。SGESVJ または DGESVJ と同じ方法でヤコビ回転を適用しますが、収束をチェックしません (停止基準)。
*x*LA_GEAMV (P)	実/複素一般行列の誤差限界を計算する行列-ベクトル演算を実行します。
CLA_GERCOND_C (P) または ZLA_GERCOND_C (P)	複素一般行列の op(A)*inv(diag(c)) の無限大ノルム条件数を計算します。C は REAL ベクトルです。
CLA_GERCOND_X (P) または ZLA_GERCOND_X (P)	複素一般行列の op(A)*inv(diag(x)) の無限大ノルム条件数を計算します。X は COMPLEX ベクトルです。
SLA_GERCOND(P) または DLA_GERCOND (P)	実一般行列の Skeel 条件数を推定します。
*x*LA_GERFSX_EXTENDED (P)	高精度反復改良を実行することによって実/複素一般行列の連立 1 次方程式の計算解を改良し、解の誤差限界と後退誤差推定を提供します。
*x*LA_GERFSX_GBRPVGRW	実/複素一般行列のピボット拡大係数の逆数 norm(A)/norm(U) を計算します。
*x*LALS0 (P)	分割統治 SVD 法を使用した最小二乗問題の求解に後方乗算係数を適用します。*x*LALSA によって使用されます。
CLALSA (P) または ZLALSA (P)	コンパクト形式の複素行列の SVD を計算します。SGELSD によって使用されます。
SLALSA または DLALSA	コンパクト形式の実行列の SVD を計算します。SGELSD または DGELSD によって使用されます。
*x*LALSD (P)	SVD を使用して最小二乗問題を解きます。*x*GELSD によって使用されます。

表 31 一般行列 - 一般化問題 (一般行列のペア) ルーチン

ルーチン	機能
*x*GGBAK	*x*GGBAL からの出力に基づき一般化固有値問題の右/左固有ベクトルを生成します。
*x*GGBAL (P)	一般化固有値問題のために一般行列のペアを均衡化します。
*x*GGES	2 つの非対称行列の一般化固有値、シュール形式、およびオプションで左および/または右シュールベクトルを計算します (単純ドライバ)。
*x*GGESX	一般化固有値、シュール形式、およびオプションで左および/または右シュールベクトルを計算します (エキスパートドライバ)。
*x*GGEV (P)	2 つの非対称行列の一般化固有値および左および/または右一般化固有ベクトルを計算します (単純ドライバ)。
*x*GGEVX (P)	2 つの非対称行列の一般化固有値および左および/または右一般化固有ベクトルを計算します (エキスパートドライバ)。
*x*GGGLM (P)	一般ガウス-マルコフ線形モデル (GLM) 問題を解きます。

ルーチン	機能
*x*GGHRD (P)	直交変換を使用して、2 つの行列を一般化上ヘッセンベルグ形式に縮約します。
*x*GGLSE	GRQ (一般化 RQ) 因子分解を使用して LSE (制約付き線形最小二乗問題) を解きます。
*x*GGQRF	2 つの行列の一般化 QR 因子分解を計算します。
*x*GGRQF	2 つの行列の一般化 RQ 因子分解を計算します。
*x*GGSVD	一般化特異値分解を計算します (ドライバ)。
*x*GGSVP (P)	一般化特異値分解を計算するための前処理段階として直交/ユニタリ行列を計算します。

表 32 一般三重対角行列ルーチン

ルーチン	機能
*x*GTCON	*x*GTTRF で計算された LU 因子分解を使用して、三重対角行列の条件数の逆数を推定します。
*x*GTRFS (P)	一般三重対角行列の連立 1 次方程式の解を改良します。
*x*GTSV (P)	一般三重対角行列の連立 1 次方程式を解きます (単純ドライバ)。
*x*GTSVX	一般三重対角行列の連立 1 次方程式を解きます (エキスパートドライバ)。
*x*GTTRF (P)	部分ピボットおよび行の入れ替えを使用して、一般三重対角行列の LU 因子分解を計算します。
*x*GTTRS	*x*GTTRF で計算された因子分解を使用して、一般三重対角行列の連立 1 次方程式を解きます。
*x*GTTS2 (P)	*x*GTTRF で計算された LU 因子分解を使用して、三重対角行列の連立 1 次方程式を解きます。

表 33 エルミート帯行列ルーチン

ルーチン	機能
CHBEV または ZHBEV	エルミート帯行列のすべての固有値および固有ベクトルを計算します。新しいバージョンの CHBEVD または ZHBEVD に置き換えることを推奨します。
CHBEVD または ZHBEVD	エルミート帯行列のすべての固有値および固有ベクトルを計算し、分割統治法を使用して固有ベクトルを計算します (ドライバ)。
CHBEVX (P) または ZHBEVX (P)	エルミート帯行列の選択された固有値および固有ベクトルを計算します。
CHBGST (P) または ZHBGST (P)	エルミート定値帯行列の一般化固有値問題を標準形式に縮約します。
CHBGV または ZHBGV	エルミート定値帯行列の一般化固有値問題のすべての固有値および固有ベクトルを計算します。新しいバージョンの CHBGVD または ZHBGVD に置き換えることを推奨します。
CHBGVD または ZHBGVD	エルミート定値帯行列の一般化固有値問題のすべての固有値および固有ベクトルを計算し、分割統治法を使用して固有ベクトルを計算します (ドライバ)。
CHBGVX (P) または ZHBGVX (P)	エルミート定値帯行列の一般化固有値問題の選択された固有値および固有ベクトルを計算します。

ルーチン	機能
CHBTRD (P) または ZHBTRD (P)	エルミート帯行列をユニタリ相似変換によって実対称三重対角形式に縮約します。

表 34 エルミート行列ルーチン

ルーチン	機能
CHECON または ZHECON	CHETRF または ZHETRF で計算された因子分解を使用して、エルミート行列の条件数の逆数を推定します。
CHECON_ROOK または ZHECON_ROOK	CHETRF_ROOK または ZHETRF_ROOK で計算された因子分解を使用して、エルミート行列の条件数の逆数を推定します。
CHEEQUB (P) または ZHEEQUB (P)	エルミート行列を均衡化し、2 ノルムに関してその条件数を減らすように、行および列のスケーリングを計算します。
CHEEV または ZHEEV	エルミート行列のすべての固有値および固有ベクトルを計算します (単純ドライバ)。新しいバージョンの CHEEVR または ZHEEVR に置き換えることを推奨します。
CHEEVD または ZHEEVD	エルミート行列のすべての固有値および固有ベクトルを計算し、分割統治法を使用して固有ベクトルを計算します (ドライバ)。新しいバージョンの CHEEVR または ZHEEVR に置き換えることを推奨します。
CHEEVR または ZHEEVR	複素エルミート行列の選択された固有値および固有ベクトルを計算します。
CHEEVX (P) または ZHEEVX (P)	エルミート行列の選択された固有値および固有ベクトルを計算します (エキスパートドライバ)。
CHEGST または ZHEGST	CPOTRF または ZPOTRF で計算された因子分解を使用して、エルミート定値行列の一般化固有値問題を標準形式に縮約します。
CHEGV または ZHEGV	複素エルミート定値行列の一般化固有値問題のすべての固有値および固有ベクトルを計算します。新しいバージョンの CHEGVD または ZHEGVD に置き換えることを推奨します。
CHEGVD または ZHEGVD	複素エルミート定値行列の一般化固有値問題のすべての固有値および固有ベクトルを計算し、分割統治法を使用して固有ベクトルを計算します (ドライバ)。
CHEGVX または ZHEGVX	複素エルミート定値行列の一般化固有値問題の選択された固有値および固有ベクトルを計算します。
CHERFS (P) または ZHERFS (P)	係数行列がエルミート不定値行列である場合に、連立 1 次方程式の計算解を改良します。
CHERFSX (P) または ZHERFSX (P)	係数行列がエルミート不定値行列である場合に、連立 1 次方程式の計算解を改良します (高精度)。
CHESV または ZHESV	複素エルミート不定値行列の連立 1 次方程式を解きます (単純ドライバ)。複素エルミート行列の因子分解を計算するために CHETRF が呼び出されます。
CHESV_ROOK または ZHESV_ROOK	複素エルミート不定値行列の連立 1 次方程式を解きます (単純ドライバ)。複素エルミート行列の因子分解を計算するために CHETRF_ROOK が呼び出されます。
CHESVX または ZHESVX	複素エルミート不定値行列の連立 1 次方程式を解きます (エキスパートドライバ)。
CHESVXX (P) または ZHESVXX (P)	対角ピボット因子分解を使用して、複素正方対称行列の連立 1 次方程式の解を計算します (高精度)。
CHETD2 または ZHETD2	複素エルミート行列をユニタリ相似変換によって実対称三重対角形式に縮約します (非ブロック化アルゴリズム)。
CHETF2 (P) または ZHETF2 (P)	対角ピボット法を使用して、複素エルミート行列の因子分解を計算します (非ブロック化アルゴリズム)。

ルーチン	機能
CHETF2_ROOK (P) または ZHETF2_ROOK (P)	制限付きバンチ-カウフマン (「rook」) 対角ピボット法を使用して、複素エルミート行列の因子分解を計算します (非ブロック化アルゴリズム)。
CHETRD または ZHETRD	エルミート行列をユニタリ相似変換によって実対称三重対角形式に縮約します。
CHETRF (P) または ZHERTF (P)	対角ピボット法を使用して、複素エルミート不定値行列の因子分解を計算します。
CHETRF_ROOK (P) または ZHERTF_ROOK (P)	バンチ-カウフマン (「rook」) 対角ピボット法を使用して、複素エルミート不定値行列の因子分解を計算します。
CHETRI (P) または ZHETRI (P)	CHETRF または ZHETRF で計算された因子分解を使用して、複素エルミート不定値行列の逆行列を計算します。
CHETRI_ROOK (P) または ZHETRI_ROOK (P)	CHETRF_ROOK または ZHETRF_ROOK で計算された因子分解を使用して、複素エルミート不定値行列の逆行列を計算します。
CHETRI2 または ZHETRI2	CHETRF または ZHETRS で計算された因子分解を使用して、複素エルミート不定値行列の逆行列を計算します。ワークスペース領域のリーディングディメンジョンの大きさを設定してから、逆行列を実際に計算する CHETRI2X または ZHETRI2X を呼び出します (高精度)。
CHETRI2X (P) または ZHETRI2X (P)	CHETRF または ZHETRS で計算された因子分解を使用して、複素エルミート不定値行列の逆行列を計算します (高精度)。
CHETRS (P) または ZHETRS (P)	CHETRF または ZHETRF で計算された因子分解を使用して、複素エルミート不定値行列を解きます。
CHETRS_ROOK (P) または ZHETRS_ROOK (P)	CHETRF_ROOK または ZHETRF_ROOK で計算された因子分解を使用して、複素エルミート不定値行列を解きます。
CHETRS2 (P) または ZHETRS2 (P)	CHETRF または ZHERTF で計算され CSYCONV または ZSYCONV で変換された因子分解を使用して、複素エルミート行列の連立 1 次方程式を解きます。
CHFRK (P) または ZHFRK (P)	RFP 形式の行列に対してエルミートのランク k 演算を実行します。
CLA_HEAMV または ZLA_HEAMV	複素エルミート不定値行列の誤差限界を計算する行列-ベクトル演算を実行します。
CLA_HERCOND_C (P) または ZLA_HERCOND_C (P)	複素エルミート不定値行列の op(A)*inv(diag(c)) の無限大ノルム条件数を計算します。C は REAL ベクトルです。
CLA_HERCOND_X (P) または ZLA_HERCOND_X (P)	複素エルミート不定値行列の op(A)*inv(diag(x)) の無限大ノルム条件数を計算します。X は COMPLEX ベクトルです。
CLA_HERFSX_EXTENDED (P) または ZLA_HERFSX_EXTENDED (P)	高精度反復改良を実行することによって複素エルミート不定値行列の連立 1 次方程式の計算解を改良し、解の誤差限界と後退誤差推定を提供します。
CLAHEF (P) または ZLAHEF (P)	対角ピボット法を使用して、複素エルミート不定値行列の部分的因子分解を計算します。CHETRF または CHETRF によって使用されます。
CLAHEF_ROOK (P) または ZLAHEF_ROOK (P)	バンチ-カウフマン (「rook」) 対角ピボット法を使用して、複素エルミート不定値行列の部分的因子分解を計算します。CHETRF_ROOK または CHETRF_ROOK によって使用されます。

表 35 パック格納のエルミート行列ルーチン

ルーチン	機能
CHPCON または ZHPCON	CHPTRF または ZHPTRF で計算された因子分解を使用して、パック格納のエルミート不定値行列の条件数の逆数を推定します。
CHPEV または ZHPEV	パック格納のエルミート行列のすべての固有値および固有ベクトルを計算します (単純ドライバ)。新しいバージョンの CHPEVD または ZHPEVD に置き換えることを推奨します。
CHPEVX (P) または ZHPEVX (P)	パック格納のエルミート行列の選択された固有値および固有ベクトルを計算します (エキスパートドライバ)。
CHPEVD または ZHPEVD	パック格納のエルミート行列のすべての固有値および固有ベクトルを計算し、分割統治法を使用して固有ベクトルを計算します (ドライバ)。
CHPGST または ZHPGST	係数行列がパック格納されている場合に、エルミート定値行列の一般化固有値問題を標準形式に縮約します。CPPTRF または ZPPTRF で計算された因子分解を使用します。
CHPGV または ZHPGV	係数行列がパック格納されている場合に、エルミート定値行列の一般化固有値問題のすべての固有値および固有ベクトルを計算します (単純ドライバ)。新しいバージョンの CHPGVD または ZHPGVD に置き換えることを推奨します。
CHPGVD または ZHPGVD	係数行列がパック格納されている場合に、エルミート定値行列の一般化固有値問題のすべての固有値および固有ベクトルを計算し、分割統治法を使用して固有ベクトルを計算します (ドライバ)。
CHPGVX または ZHPGVX	係数行列がパック格納されている場合に、複素エルミート定値行列の固有値問題の選択された固有値および固有ベクトルを計算します (エキスパートドライバ)。
CHPRFS (P) または ZHPRFS (P)	係数行列がパック格納のエルミート不定値行列である場合に、連立 1 次方程式の計算解を改良します。
CHPSV または ZHPSV	係数行列がパック形式で格納されたエルミート行列である場合に、複素連立 1 次方程式の解を計算します (単純ドライバ)。
CHPSVX または ZHPSVX	係数行列がパック形式で格納されたエルミート行列である場合に、対角ピボット因子分解を使用して、複素連立 1 次方程式の解を計算します (エキスパートドライバ)。
CHPTRD または ZHPTRD	パック形式で格納された複素エルミート行列を、ユニタリ相似変換によって実対称三重対角形式に縮約します。
CHPTRF または ZHPTRF	バンチ-カウフマン対角ピボット法を使用して、パック格納の複素エルミート行列の因子分解を計算します。
CHPTRI または ZHPTRI	CHPTRF または ZHPTRF で計算された因子分解を使用して、パック格納の複素エルミート不定値行列の逆行列を計算します。
CHPTRS (P) または ZHPTRS (P)	CHPTRF または ZHPTRF で計算された因子分解を使用して、パック形式で格納された複素エルミート不定値行列を解きます。

表 36 上ヘッセンベルグ行列ルーチン

ルーチン	機能
*x*HSEIN (P)	逆反復を使用して、上ヘッセンベルグ行列の指定された右および/または左固有ベクトルを計算します。
CHSEQR または ZHSEQR	マルチシフト QR 法を使用して、複素上ヘッセンベルグ行列の固有値およびシュール因子分解を計算します。
SHSEQR (P) または DHSEQR (P)	マルチシフト QR 法を使用して、実上ヘッセンベルグ行列の固有値およびシュール因子分解を計算します。

表 37 上ヘッセンベルグ行列 - 一般化問題 (ヘッセンベルグと三角行列) ルーチン

ルーチン	機能
*x*HGEQZ (P)	シングル/ダブルシフト QZ 法を使用して、複素行列ペア (H,T) の固有値を計算します。ここで、H は上ヘッセンベルグ行列、T は上三角行列です。このタイプの行列ペアは *x*GGHRD によって生成されます。

表 38 パック格納の実直交行列ルーチン

ルーチン	機能
SOPGTR (P) または DOPGTR (P)	SSPTRD または DSPTRD で求められた実三重対角行列から直交変換行列を生成します。
SOPMTR または DOPMTR	SSPTRD または DSPTRD によって三重対角形式に縮約された直交変換行列を、実一般行列に乗算します。

表 39 実直交行列ルーチン

ルーチン	機能
SORBDB または DORBDB	実分割直交行列のブロックを同時に二重対角化します。
SORBDB1 または DORBDB1	正規直交列を持つ Tall and Skinny 行列のブロックを同時に二重対角化します (バリアント 1)。
SORBDB2 または DORBDB2	正規直交列を持つ Tall and Skinny 行列のブロックを同時に二重対角化します (バリアント 2)。
SORBDB3 または DORBDB3	正規直交列を持つ Tall and Skinny 行列のブロックを同時に二重対角化します (バリアント 3)。
SORBDB4 または DORBDB4	正規直交列を持つ Tall and Skinny 行列のブロックを同時に二重対角化します (バリアント 4)。
SORBDB5 または DORBDB5	Q の正規直交列に対して列ベクトル X を直交化します。
SORBDB6 または DORBDB6	Q の正規直交列に対して列ベクトル X を直交化します。SORBDB4 または DORBDB5 によって使用されます。
SORG2L (P) または DORG2L (P)	SGEQLF または DGEQLF で求められた QL 因子分解から、実直交行列 Q の全部または一部を生成します (非ブロック化アルゴリズム)。
SORG2R (P) または DORG2R (P)	SGEQRF または DGEQRF で求められた QR 因子分解から、実直交行列 Q の全部または一部を生成します (非ブロック化アルゴリズム)。
SORGBR (P) または DORGBR	SGEBRD または DGEBRD で求められた二重対角形式への縮約から、実直交変換行列を生成します。
SORGHR (P) または DORGHR (P)	SGEHRD または DGEHRD で求められたヘッセンベルグ形式への縮約から、実直交変換行列を生成します。
SORGL2 (P) または DORGL2 (P)	SGELQF または DGELQF で求められた正規直交行を持つ実長方行列を生成します。
SORGLQ (P) または DORGLQ (P)	SGELQF または DGELQF で求められた LQ 因子分解から、実直交行列 Q を生成します。
SORGQL (P) または DORGQL (P)	SGEQLF または DGEQLF で求められた QL 因子分解から、実直交行列 Q を生成します。

ルーチン	機能
SORGQR (P) または DORGQR (P)	SGEQRF または DGEQRF で求められた QR 因子分解から、実直交行列 Q を生成します。
SORGR2 (P) または DORGR2 (P)	SGEQRF または DGEQRF で求められた RQ 因子分解から、実直交行列 Q の全部または一部を生成します (非ブロック化アルゴリズム)。
SORGRQ (P) または DORGRQ (P)	SGERQF または DGERQF で求められた RQ 因子分解から、実直交行列 Q を生成します。
SORGTR (P) または DORGTR (P)	SSYTRD または DSYTRD によって三重対角形式に縮約された実直交行列を生成します。
SORM2L または DORM2L	SGEQLF または DGEQLF で求められた QL 因子分解から得られた直交行列を、実一般行列に乗算します (非ブロック化アルゴリズム)。
SORM2R または DORM2R	SGEQRF または DGEQRF で求められた QR 因子分解から得られた直交行列を、実一般行列に乗算します (非ブロック化アルゴリズム)。
SORMBR または DORMBR	SGEBRD または DGEBRD によって二重対角形式に縮約された直交行列を、実一般行列に乗算します。
SORMHR または DORMHR	SGEHRD または DGEHRD によってヘッセンベルグ形式に縮約された直交行列を、実一般行列に乗算します。
SORML2 または DORML2	SGELQF で求められた LQ 因子分解から得られた直交行列を、実一般行列に乗算します (非ブロック化アルゴリズム)。
SORMLQ または DORMLQ	SGELQF または DGELQF で求められた LQ 因子分解から得られた直交行列を、実一般行列に乗算します。
SORMQL または DORMQL	SGEQLF または DGEQLF で求められた QL 因子分解から得られた直交行列を、実一般行列に乗算します。
SORMQR または DORMQR	SGEQRF または DGEQRF で求められた QR 因子分解から得られた直交行列を、実一般行列に乗算します。
SORMR2 または DORMR2	STZRZF または DTZRZF で求められた RQ 因子分解から得られた直交行列を、実一般行列に乗算します (非ブロック化アルゴリズム)。
SORMR3 または DORMR3	STZRZF または DTZRZF で求められた RZ 因子分解から得られた直交行列を、実一般行列に乗算します (非ブロック化アルゴリズム)。
SORMRQ または DORMRQ	SGERQF または DGERQF で求められた RQ 因子分解から得られた直交行列を、実一般行列に乗算します。
SORMRZ または DORMRZ	STZRZF または DTZRZF で求められた RZ 因子分解から得られた直交行列を、実一般行列に乗算します。
SORMTR または DORMTR	SSYTRD または DSYTRD によって三重対角形式に縮約された直交変換行列を、実一般行列に乗算します。

表 40 対称/エルミート正定値帯行列ルーチン

ルーチン	機能
*x*PBCON	*x*PBTRF で求められたコレスキー因子分解を使用して、対称/エルミート正定値帯行列の条件数の逆数を推定します。
*x*PBEQU (P)	対称/エルミート正定値帯行列の均衡化スケーリング係数を計算します。
*x*PBRFS (P)	対称/エルミート正定値帯行列の連立 1 次方程式の解を改良します。
*x*PBSTF	実対称正定値帯行列の split コレスキー因子分解を計算します。

ルーチン	機能
*x*PBSV	対称/エルミート正定値帯行列の連立 1 次方程式を解きます (単純ドライバ)。
*x*PBSVX (P)	対称/エルミート正定値帯行列の連立 1 次方程式を解きます (エキスパートドライバ)。
*x*PBTF2	実対称/複素エルミート正定値帯行列のコレスキー因子分解を計算します (非ブロック化アルゴリズム)。
*x*PBTRF	対称/エルミート正定値帯行列のコレスキー因子分解を計算します。
*x*PBTRS	*x*PBTRF で計算されたコレスキー因子分解を使用して、実対称/複素エルミート正定値帯行列の連立 1 次方程式を解きます。

表 41 対称/エルミート正定値行列ルーチン

ルーチン	機能
CLA_PORCOND_C (P) または ZLA_PORCOND_C (P)	複素エルミート正定値行列の op(A)*inv(diag(c)) の無限大ノルム条件数を計算します。C は REAL ベクトルです。
CLA_PORCOND_X (P) または ZLA_PORCOND_X (P)	複素エルミート正定値行列の op(A)*inv(diag(x)) の無限大ノルム条件数を計算します。X は COMPLEX ベクトルです。
SLA_PORCOND (P) または DLA_PORCOND(P)	実対称正定値行列の Skeel 条件数を推定します。
*x*LA_LIN_BERR (P)	成分から見た相対後退誤差を計算します。
*x*LA_PORFSX_EXTENDED (P)	高精度反復改良を実行することによって実対称/複素エルミート正定値行列の連立 1 次方程式の計算解を改良し、解の誤差限界と後退誤差推定を提供します。
*x*LA_WWADDW	ベクトル W を二重の単一ベクトル (X, Y) に加算します。これは、現存している IBM の 16 進および 2 進の浮動小数点演算すべてに使用できますが、10 進には使用できません。
*x*PFTRF	実対称/エルミート正定値帯行列のコレスキー因子分解を計算します。
*x*PFTRI	*x*PFTRF で計算されたコレスキー因子分解を使用して、実対称/エルミート正定値行列の逆行列を計算します。
*x*PFTRS	*x*PFTRF で計算されたコレスキー因子分解を使用して、対称/エルミート正定値行列の連立 1 次方程式を解きます。
*x*POCON	*x*POTRF で求められたコレスキー因子分解を使用して、対称/エルミート正定値行列の条件数の逆数を推定します。
*x*POEQU (P)	対称/エルミート正定値行列の均衡化スケーリング係数を計算します。
*x*POEQUB (P)	対称/エルミート正定値行列を均衡化し、2 ノルムに関してその条件数を減らすように、行および列のスケーリングを計算します。
*x*PORFS (P)	コレスキー因子分解された対称/エルミート正定値行列の連立 1 次方程式の解を改良します。
*x*PORFSX (P)	係数行列が実対称/エルミート正定値行列である場合に、連立 1 次方程式の計算解を改良し、解の誤差限界と後退誤差推定を提供します (高精度)。
*x*POSV	対称/エルミート正定値行列の連立 1 次方程式を解きます (単純ドライバ)。
*x*POSVX (P)	対称/エルミート正定値行列の連立 1 次方程式を解きます (エキスパートドライバ)。
*x*POSVXX (P)	実対称/エルミート正定値行列の連立 1 次方程式を解きます (エキスパートドライバ、高精度)。要求された場合は、ノルムから見た誤差限界と最大成分から見た誤差限界の両方を返します。
*x*POTRF	実対称/エルミート正定値行列のコレスキー因子分解を計算します。

ルーチン	機能
*x*POTRI	*x*POTRF で計算されたコレスキー因子分解を使用して、実対称/エルミート正定値行列の逆行列を計算します。
*x*POTRS	*x*POTRF で計算されたコレスキー因子分解を使用して、実対称/エルミート正定値行列の連立 1 次方程式を解きます。
ZCPOSV	複素正定値行列の連立 1 次方程式の解を計算します (反復改良を使用した混合精度)。

表 42 パック格納の対称/エルミート正定値行列ルーチン

ルーチン	機能
*x*PPCON	コレスキー因子分解されたパック格納の対称正定値行列の条件数の逆数を推定します。
*x*PPEQU (P)	パック格納の対称/エルミート正定値行列の均衡化スケーリング係数を計算します。
*x*PPRFS (P)	コレスキー因子分解されたパック格納の対称/エルミート正定値行列の連立 1 次方程式の解を改良します。
*x*PPSV	パック格納の対称/エルミート正定値行列の連立 1 次方程式を解きます (単純ドライバ)。
*x*PPSVX (P)	パック格納の対称/エルミート正定値行列の連立 1 次方程式を解きます (エキスパートドライバ)。
*x*PPTRF	パック形式で格納された実対称/エルミート正定値行列のコレスキー因子分解を計算します。
*x*PPTRI	*x*PPTRF で求められたコレスキー因子分解を使用して、パック格納の実対称/エルミート正定値行列の逆行列を計算します。
*x*PPTRS	係数行列がパック格納されている場合に、*x*PPTRF で求められたコレスキー因子分解を使用して、実対称/エルミート正定値行列の連立 1 次方程式を解きます。
*x*PSTF2 (P)	実対称/エルミート半正定値行列のコレスキー因子分解を完全ピボットで計算します。このバージョンのアルゴリズムはレベル 2 BLAS を呼び出します。
*x*PSTRF (P)	実対称/エルミート半正定値行列のコレスキー因子分解を完全ピボットで計算します。このバージョンのアルゴリズムはレベル 3 BLAS を呼び出します。

表 43 対称/エルミート正定値三重対角行列ルーチン

ルーチン	機能
*x*PTCON	*x*PTTRF で計算されたコレスキー因子分解を使用して、実対称/エルミート正定値三重対角行列の条件数の逆数を推定します。
*x*PTEQR (P)	実対称/エルミート正定値行列のすべての固有ベクトル、およびオプションで固有値を計算します。
*x*PTRFS (P)	対称/エルミート正定値三重対角行列の連立 1 次方程式の解を改良します。
*x*PTSV	実対称/エルミート正定値三重対角行列の連立 1 次方程式を解きます (単純ドライバ)。
*x*PTSVX	実対称/エルミート正定値三重対角行列の連立 1 次方程式を解きます (エキスパートドライバ)。
*x*PTTRF	実対称/エルミート正定値三重対角行列の LDL^H または LDL^T 因子分解を計算します。

ルーチン	機能
*x*PTTRS	*x*PTTRF で求められた LDL^H または LDL^T 因子分解を使用して、実対称/エルミート正定値三重対角行列の連立 1 次方程式を解きます。
*x*PTTS2 (P)	*x*PTTRF で計算された LDL^H または LDL^T 因子分解を使用して、三重対角行列の連立 1 次方程式を解きます。*x*PTTRS によって使用されます。

表 44 実対称帯行列ルーチン

ルーチン	機能
SSBEV または DSBEV	実対称帯行列のすべての固有値、およびオプションで左および/または右固有ベクトルを計算します (単純ドライバ)。新しいバージョンの SSBEVD または DSBEVD に置き換えることを推奨します。
SSBEVD または DSBEVD	実対称帯行列のすべての固有値、およびオプションで固有ベクトルを計算します。固有ベクトルが要求された場合は、分割統治法を使用します (ドライバ)。
SSBEVX (P) または DSBEVX (P)	対称帯行列の選択された固有値、およびオプションで左および/または右固有ベクトルを計算します (エキスパートドライバ)。
SSBGST (P) または DSBGST (P)	対称定値帯行列の一般化固有値問題を標準形式に縮約します。
SSBGV または DSBGV	対称定値帯行列の一般化固有値問題のすべての固有値、およびオプションで固有ベクトルを計算します (単純ドライバ)。新しいバージョンの SSBGVD または DSBGVD に置き換えることを推奨します。
SSBGVD または DSBGVD	対称定値帯行列の一般化固有値問題のすべての固有値、およびオプションで固有ベクトルを計算し、分割統治法を使用して固有ベクトルを計算します (単純ドライバ)。
SSBGVX (P) または DSBGVX (P)	対称定値帯行列の一般化固有値問題の選択された固有値および固有ベクトルを計算します (エキスパートドライバ)。
SSBTRD (P) または DSBTRD (P)	対称帯行列を直交相似変換によって実対称三重対角形式に縮約します。

表 45 パック格納の対称行列ルーチン

ルーチン	機能
*x*SPCON	*x*SPTRF で計算された因子分解を使用して、パック格納の実/複素対称行列の条件数の逆数を推定します。
SSFRK (P) または DSFRK (P)	RFP 形式の実行列に対して対称のランク k 演算を実行します。
SSPEV または DSPEV	パック格納の対称行列のすべての固有値および固有ベクトルを計算します (単純ドライバ)。新しいバージョンの SSPEVD または DSPEVD に置き換えることを推奨します。
SSPEVD または DSPEVD	パック格納の対称行列のすべての固有値、およびオプションで左および/または右固有ベクトルを計算します。固有ベクトルが要求された場合は、分割統治法を使用します (単純ドライバ)。
SSPEVX (P) または DSPEVX (P)	パック格納の対称行列の選択された固有値および固有ベクトルを計算します (エキスパートドライバ)。
SSPGST または DSPGST	係数行列がパック格納されている場合に、実対称定値行列の一般化固有値問題を標準形式に縮約します。SPPTRF または DPPTRF で計算された因子分解を使用します。新しいバージョンの SSPGVD または DSPGVD に置き換えることを推奨します。

ルーチン	機能
SSPGV または DSPGV	係数行列がパック格納されている場合に、実対称定値行列の一般化固有値問題のすべての固有値および固有ベクトルを計算します (単純ドライバ)。新しいバージョンの SSPGVD または DSPGVD に置き換えることを推奨します。
SSPGVD または DSPGVD	係数行列がパック格納されている場合に、実対称定値行列の一般化固有値問題のすべての固有値および固有ベクトルを計算し、分割統治法を使用して固有ベクトルを計算します (ドライバ)。
SSPGVX または DSPGVX	係数行列がパック格納されている場合に、実対称定値行列の一般化固有値問題の選択された固有値および固有ベクトルを計算します (エキスパートドライバ)。
DSPOSV	実対称正定値行列の連立 1 次方程式の解を計算します。まず*単精度*で行列の因子分解を試み、必要な場合は次に*倍精度*で試みます。
*x*SPRFS (P)	係数行列がパック格納の対称不定値行列である場合に、実/複素連立 1 次方程式の計算解を改良します。
*x*SPSV	係数行列がパック格納の対称行列である場合に、実/複素連立 1 次方程式の解を計算します (単純ドライバ)。
*x*SPSVX	係数行列がパック格納の対称行列である場合に、対角ピボット因子分解を使用して、連立 1 次方程式の解を計算します (エキスパートドライバ)。
SSPTRD または DSPTRD	パック形式で格納された実対称行列を、直交相似変換によって実対称三重対角形式に縮約します。
*x*SPTRF	バンチ-カウフマン対角ピボット法を使用して、パック格納の対称行列の因子分解を計算します。
*x*SPTRI	*x*SPTRF で計算された因子分解を使用して、パック格納の対称不定値行列の逆行列を計算します。
*x*SPTRS (P)	*x*SPTRF で計算された因子分解を使用して、パック格納の実/複素対称行列の連立 1 次方程式を解きます。

表 46 実対称三重対角行列ルーチン

ルーチン	機能
*x*LAED0 (P)	分割統治法を使用して、縮約されていない実/複素対称三重対角行列のすべての固有値および対応する固有ベクトルを計算します。*x*STEDC によって使用されます。
SLAED1 (P) または DLAED1 (P)	ランク 1 対称行列による変更後の、実対角行列の更新された固有システムを計算します。元の行列が三重対角行列である場合に、SSTEDC または DSTEDC によって使用されます。
SLAED2 (P) または DLAED2 (P)	2 セットの固有値をソートされた 1 つのセットにマージし、問題のサイズのデフレートを試みます。SSTEDC または DSTEDC によって使用されます。
SLAED3 (P) または DLAED3	永年方程式の根を見つけ、固有ベクトルを更新します。元の行列が三重対角行列である場合に、SSTEDC または DSTEDC によって使用されます。
SLAED4 (P) または DLAED4 (P)	永年方程式の単根を見つけます。SSTEDC または DSTEDC によって使用されます。
SLAED5 または DLAED5	2 × 2 永年方程式を解きます。SSTEDC または DSTEDC によって使用されます。
SLAED6 または DLAED6	原点にもっとも近い正または負の根を計算します (永年方程式の求解における 1 つのニュートン過程)。
*x*LAED7 (P)	ランク 1 対称行列による変更後の、対角行列の更新された固有システムを計算します。元の行列が密行列である場合に、*x*STEDC によって使用されます。

ルーチン	機能
*x*LAED8 (P)	2 セットの固有値をソートされた 1 つのセットにマージし、永年方程式をデフレートします。元の行列が密行列である場合に、*x*STEDC によって使用されます。
SLAED9 (P) または DLAED9 (P)	永年方程式の根を見つけ、固有ベクトルを更新します。元の行列が密行列である場合に、SSTEDC または DSTEDC によって使用されます。
SLAEDA (P) または DLAEDA (P)	対角行列のランク 1 変更を決定するベクトルを計算します。元の行列が密行列である場合に、SSTEDC または DSTEDC によって使用されます。
SLAGTF または DLAGTF (P)	部分ピボットおよび行の入れ替えを使用して、行列 T - (lambda * I) の LU 因子分解を計算します。ここで、T は一般三重対角行列、lambda はスカラーです。SSTEIN または DSTEIN によって使用されます。
SSTEBZ または DSTEBZ	実対称三重対角行列の固有値を計算します。
CSTEDC (P) または ZSTEDC (P)	分割統治法を使用して、対称三重対角行列のすべての固有値、およびオプションで固有ベクトルを計算します。フル/バンド複素エルミート行列が CHETRD/ZHETRD、CHPTRD/ZHPTRD、または CHBTRD/ZHBTRD によって三重対角形式に縮約されている場合、この行列の固有ベクトルも見つけることができます。
SSTEDC または DSTEDC	分割統治法を使用して、複素対称三重対角行列のすべての固有値および固有ベクトルを計算します。フル/バンド実対称行列が SSYTRD、SSPTRD、または SSBTRD、あるいは DSYTRD、DSPTRD、または DSBTRD によって三重対角形式に縮約されている場合、この行列の固有ベクトルも見つけることができます。
*x*STEGR	Relatively Robust Representation を使用して、実対称三重対角行列の選択された固有値および固有ベクトルを計算します。*x*STEGR は、改良された *x*STEMR ルーチンの互換性ラッパーです。
*x*STEIN (P)	逆反復を使用して、実対称三重対角行列の選択された固有ベクトルを計算します。
*x*STEMR (P)	Relatively Robust Representation を使用して、実対称三重対角行列の選択された固有値、およびオプションで固有ベクトルを計算します。
*x*STEQR (P)	QL または QR 法の Pal-Walker-Kahan バリアントを使用して、実対称三重対角行列のすべての固有値および固有ベクトルを計算します。
SSTERF (P) または DSTERF (P)	QL または QR 法の root-free バリアントを使用して、実対称三重対角行列のすべての固有値および固有ベクトルを計算します。
SSTEV または DSTEV	実対称三重対角行列のすべての固有値および固有ベクトルを計算します (単純ドライバ)。新しいバージョンの SSTEVR または DSTEVR に置き換えることを推奨します。
SSTEVD または DSTEVD	実対称三重対角行列のすべての固有値および固有ベクトルを計算します (単純ドライバ)。新しいバージョンの SSTEVR または DSTEVR に置き換えることを推奨します。
SSTEVR または DSTEVR	Relatively Robust Representation を使用して、実対称三重対角行列の選択された固有値および固有ベクトルを計算します。
SSTEVX (P) または DSTEVX (P)	実対称三重対角行列の選択された固有値および固有ベクトルを計算します (エキスパートドライバ)。
*x*STSV	係数行列が対称三重対角行列である場合に、連立 1 次方程式の解を計算します (非ブロック化アルゴリズム)。
*x*STTRF (P)	バンチ-カウフマン対角ピボット法を使用して、実/複素対称三重対角行列の因子分解を計算します (非ブロック化アルゴリズム)。

表 47 対称行列ルーチン

ルーチン	機能
*x*LA_SYAMV	実/複素対称不定値行列の誤差限界を計算する行列-ベクトル演算を実行します。

ルーチン	機能
CLA_SYRCOND_C (P) または ZLA_SYRCOND_C (P)	実/複素対称不定値行列の op(A)*inv(diag(c)) の無限大ノルム条件数を計算します。C は REAL ベクトルです。
CLA_SYRCOND_X (P) または ZLA_SYRCOND_X (P)	実/複素対称不定値行列の op(A)*inv(diag(x)) の無限大ノルム条件数を計算します。X は COMPLEX ベクトルです。
SLA_SYRCOND (P) または DLA_SYRCOND(P)	実対称不定値行列の Skeel 条件数を推定します。
*x*LA_SYRFSX_EXTENDED(P)	高精度反復改良を実行することによって実/複素対称不定値行列の連立 1 次方程式の計算解を改良し、解の誤差限界と後退誤差推定を提供します。
*x*LASYF	対角ピボット法を使用して、実/複素対称行列の部分的因子分解を計算します。*x*SYTRF によって使用されます。
*x*LASYF_ROOK	制限付きバンチ-カウフマン (「rook」) 対角ピボット法を使用して、実/複素対称行列の部分的因子分解を計算します。*x*SYTRF_ROOK によって使用されます。
*x*SYCON	*x*SYTRF で計算された因子分解を使用して、実/複素対称行列の条件数の逆数を推定します。
*x*SYCON_ROOK	*x*SYTRF_ROOK で計算された因子分解を使用して、実/複素対称行列の条件数の逆数を推定します。
*x*SYCONV (P)	SSYTRF または DSYTRF で計算された行列を下および上三角行列に変換します。また、その逆の変換も行います。
*x*SYEQUB (P)	実/複素対称行列を均衡化し、2 ノルムに関してその条件数を減らすように、行および列のスケーリングを計算します。
SSYEV または DSYEV	対称行列のすべての固有値および固有ベクトルを計算します (単純ドライバ)。新しいバージョンの SSYEVR または DSYEVR に置き換えることを推奨します。
SSYEVD または DSYEVD	対称行列のすべての固有値および固有ベクトルを計算し、分割統治法を使用して固有ベクトルを計算します (エキスパートドライバ)。新しいバージョンの SSYEVR または DSYEVR に置き換えることを推奨します。
SSYEVR または DSYEVR	実対称三重対角行列の選択された固有値および固有ベクトルを計算します。
SSYEVX (P) または DSYEVX (P)	実対称行列の固有値および固有ベクトルを計算します (エキスパートドライバ)。
SSYGS2 または DSYGS2	SPOTRF または DPOTRF から得られた因子分解結果を使用して、実対称定値行列の一般化固有値問題を標準形式に縮約します (非ブロック化アルゴリズム)。
SSYGST または DSYGST	SPOTRF または DPOTRF で計算された因子分解を使用して、対称定値行列の一般化固有値問題を標準形式に縮約します。
SSYGV または DSYGV	対称定値行列の一般化固有値問題のすべての固有値および固有ベクトルを計算します。新しいバージョンの SSYGVD または DSYGVD に置き換えることを推奨します。
SSYGVD または DSYGVD	対称定値行列の一般化固有値問題のすべての固有値および固有ベクトルを計算し、分割統治法を使用して固有ベクトルを計算します (ドライバ)。
SSYGVX または DSYGVX	対称定値行列の一般化固有値問題の選択された固有値および固有ベクトルを計算します (エキスパートドライバ)。
*x*SYRFS (P)	係数行列が対称不定値行列である場合に、連立 1 次方程式の計算解を改良します。
*x*SYRFSX (P)	係数行列が対称不定値行列である場合に、連立 1 次方程式の計算解を改良し、解の誤差限界と後退誤差推定を提供します (高精度)。
*x*SYSV	実/複素対称不定値行列の連立 1 次方程式を解きます (単純ドライバ)。対角ピボット法を使用して複素対称行列の因子分解を計算するために *x*SYTRF が呼び出されます。

ルーチン	機能
*x*SYSV_ROOK	実/複素対称不定値行列の連立 1 次方程式を解きます (単純ドライバ)。制限付きバンチ-カウフマン (「rook」) 対角ピボット法を使用して複素対称行列の因子分解を計算するために *x*SYTRF_ROOK が呼び出されます。
*x*SYSVX	実/複素対称不定値行列の連立 1 次方程式を解きます (エキスパートドライバ)。
*x*SYSVXX (P)	実/複素対称不定値行列の連立 1 次方程式を解きます (エキスパートドライバ、高精度)。要求された場合は、ノルムから見た誤差限界と最大成分から見た誤差限界の両方を返します。
SSYTD2 または DSYTD2	実対称行列を直交相似変換によって実対称三重対角形式に縮約します (非ブロック化アルゴリズム)。
*x*SYTF2	対角ピボット法を使用して、実/複素対称不定値行列の因子分解を計算します (非ブロック化アルゴリズム)。
*x*SYTF2_ROOK	制限付きバンチ-カウフマン (「rook」) 対角ピボット法を使用して、実/複素対称不定値行列の因子分解を計算します (非ブロック化アルゴリズム)。
SSYTRD または DSYTRD	実対称行列を直交相似変換によって実対称三重対角形式に縮約します。
*x*SYTRF (P)	バンチ-カウフマン対角ピボット法を使用して、実/複素対称不定値行列の因子分解を計算します (ブロック化アルゴリズム)。
*x*SYTRI	*x*SYTRF で計算された因子分解を使用して、実/複素対称不定値行列の逆行列を計算します。
*x*SYTRI_ROOK	*x*SYTRF_ROOK で計算された因子分解を使用して、実/複素対称不定値行列の逆行列を計算します。
*x*SYTRI2	*x*SYTRF で計算された因子分解を使用して、実/複素対称不定値行列の逆行列を計算します。ワークスペースの*リーディングディメンジョン*を設定してから、逆行列を実際に計算する *x*SYTRF2X を呼び出します。
*x*SYTRI2X (P)	*x*SYTRF で計算された因子分解を使用して、実/複素対称不定値行列の逆行列を計算します。*x*SYTRI2 によって使用されます。
*x*SYTRS (P)	*x*SYTRF で計算された因子分解を使用して、実/複素対称行列の連立 1 次方程式を解きます。
*x*SYTRS_ROOK (P)	*x*SYTRF_ROOK で計算された因子分解を使用して、実/複素対称行列の連立 1 次方程式を解きます。
*x*SYTRS2 (P)	*x*SYTRF で計算され *x*SYCONV で変換された因子分解を使用して、実/複素対称行列の連立 1 次方程式を解きます。

表 48 三角帯行列ルーチン

ルーチン	機能
*x*TBCON	三角帯行列の条件数の逆数を推定します。
*x*TBRFS (P)	三角帯行列の連立 1 次方程式の解の誤差限界および誤差推定を求めます。
*x*TBTRS	三角帯行列の連立 1 次方程式を解きます。

表 49 三角行列 - 一般化問題 (三角行列のペア) ルーチン

ルーチン	機能
*x*TGEVC (P)	*x*GGHRD および *x*HGEQZ で計算された、実/複素三角行列のペアの右および/または左固有ベクトルの一部または全部を計算します。
*x*TGEXC	直交/ユニタリ等価変換を使用して、実/複素行列ペアの一般化シュール分解を並べ替えます。
*x*TGSEN (P)	実/複素行列ペアの一般化シュール分解を並べ替え、一般化固有値を計算します。
*x*TGSJA (P)	*x*GGSVP から得られた 2 つの実/複素三角/台形行列から、一般化特異値分解 (SVD) を計算します。
CTGSNA (P) または ZTGSNA (P)	一般化シュール正準形の 2 つの行列の指定された固有値および固有ベクトルに関する条件数の逆数を推定します。
STGSNA または DTGSNA	一般化実シュール正準形の 2 つの行列の指定された固有値および固有ベクトルに関する条件数の逆数を推定します。
*x*TGSYL	一般化シルベスター方程式を解きます。

表 50 パック格納の三角行列ルーチン

ルーチン	機能
*x*TPCON	パック格納の三角行列の条件数の逆数を推定します。
*x*TPMQRT	「三角-五角」ブロック鏡映行列から得られた実/複素直交行列を、2 つのブロックから成る一般行列に適用します。
*x*TPQRT	コンパクト WY 表現を使用して、実/複素「三角-五角」行列のブロック化 QR 因子分解を計算します。この行列は、1 つの三角ブロックと 1 つの五角ブロックで構成されています。
*x*TPQRT2	コンパクト WY 表現を使用して、実/複素「三角-五角」行列の QR 因子分解を計算します。この行列は、1 つの三角ブロックと 1 つの五角ブロックで構成されています。
*x*TPRFS (P)	係数行列がパック格納の三角行列である場合に、実/複素連立 1 次方程式の解の誤差限界と後退誤差推定を提供します。解は *x*TPTRS またはほかの方法で事前に取得するようにします。
*x*TPTRI	パック格納の実/複素三角行列の逆行列を計算します。
*x*TPTRS	係数行列がパック格納されている場合に、実/複素三角行列の連立 1 次方程式を解きます。
*x*TPTTF	実/複素三角行列を標準パック形式 (TP) から Rectangular Full Packed 形式 (TF) にコピーします。
*x*TPTTR	実/複素三角行列を標準パック形式 (TP) から標準フルパック形式 (TR) にコピーします。

表 51 Rectangular Full-Packed (RFP) 形式および標準パック形式の三角行列ルーチン

ルーチン	機能
*x*TFSM (P)	実/複素行列の行列方程式を解きます。1 つのオペランドは RFP 形式の三角行列です。
*x*TFTRI	RFP 形式で格納された実/複素三角行列の逆行列を計算します。

ルーチン	機能
*x*TFTTP	実/複素三角行列を Rectangular Full-Packed 形式 (TF) から標準パック形式 (TP) にコピーします。
*x*TFTTR	実/複素三角行列を Rectangular Full-Packed 形式 (TF) から標準フル形式 (TR) にコピーします。
*x*TPTTF	実/複素三角行列を標準パック形式 (TP) から Rectangular Full Packed 形式 (TF) にコピーします。
*x*TPTTR	実/複素三角行列を標準パック形式 (TP) から標準フルパック形式 (TR) にコピーします。
*x*TRTTF	実/複素三角行列を標準フル形式 (TR) から Rectangular Full-Packed 形式 (TF) にコピーします。
*x*TRTTP	実/複素三角行列を標準フル形式 (TR) から標準パック形式 (TP) にコピーします。

表 52 三角行列ルーチン

ルーチン	機能
*x*TRCON	実/複素三角行列の逆数または条件数を推定します。
*x*TREVC (P)	実/複素上三角行列の右および/または左固有ベクトルを計算します。
*x*TREXC	直交/ユニタリ相似変換を使用して、実/複素行列のシュール因子分解を並べ替えます。
*x*TRRFS (P)	実/複素三角行列の連立 1 次方程式の誤差限界と誤差推定を提供します。
CTRSEN (P) または ZTRSEN (P)	複素行列 A = Q*T*Q**H のシュール因子分解を並べ替えて、選択された固有値クラスタが上三角行列 T の対角の先頭位置に現れるようにし、Q の先頭列が対応する右不変部分空間の正規直交基底を形成するようにします。
STRSEN または DTRSEN	実行列 A = Q*T*Q**T の実シュール因子分解を並べ替えて、選択された固有値クラスタが上三角行列 T の対角の先頭位置に現れるようにし、Q の先頭列が対応する右不変部分空間の正規直交基底を形成するようにします。
*x*TRSNA (P)	上準三角行列の選択された固有値および固有ベクトルに関する条件数の逆数を推定します。
*x*TRSYL	シルベスター行列方程式を解きます。
*x*TRTRI	実/複素三角行列の逆行列を計算します (非ブロック化アルゴリズム)。
*x*TRTRS	三角行列の連立 1 次方程式を解きます。

表 53 台形行列ルーチン

ルーチン	機能
*x*LARZ	基本鏡映行列 (*x*TZRZF で求められる) を実/複素一般行列に適用します。
*x*LARZB (P)	ブロック鏡映行列またはその転置行列を実一般行列に適用します。あるいは、ブロック鏡映行列またはその共役転置行列を複素一般行列に適用します。
*x*LARZT	k 個の基本鏡映行列の積として定義される実/複素ブロック鏡映行列 H の三角係数 T を生成します。
*x*LATZM	*x*ORMZ に置き換えられた非推奨のルーチン。*x*TZRQF で生成されたハウスホルダー行列を実/複素行列に適用します。
*x*TZRQF (P)	*x*TZRZF に置き換えられた非推奨のルーチン。

ルーチン	機能
*x*TZRZF (P)	長方上台形行列を直交変換によって上三角形式に縮約します。

表 54 ユニタリ行列ルーチン

ルーチン	機能
CUNBDB または ZUNBDB	M × M 分割ユニタリ行列のブロックを同時に二重対角化します。
CUNBDB1 または ZUNBDB1	正規直交列を持つ Tall and Skinny 行列のブロックを同時に二重対角化します (バリアント 1)。
CUNBDB2 または ZUNBDB2	正規直交列を持つ Tall and Skinny 行列のブロックを同時に二重対角化します (バリアント 2)。
CUNBDB3 または ZUNBDB3	正規直交列を持つ Tall and Skinny 行列のブロックを同時に二重対角化します (バリアント 3)。
CUNBDB4 または ZUNBDB4	正規直交列を持つ Tall and Skinny 行列のブロックを同時に二重対角化します (バリアント 4)。
CUNBDB5 または ZUNBDB5	Q の正規直交列に対して列ベクトル X を直交化します。
CUNBDB5 または ZUNBDB5	Q の正規直交列に対して列ベクトル X を直交化します。CUNBDB5 または ZUNBDB5 によって使用されます。
CUNCSD2BY1 または ZUNCSD2BY1	正規直交列を持ち、2 × 1 ブロック構造に分割されている M × Q 行列の CS 分解を計算します。
CUNG2L (P) または ZUNG2L (P)	CGEQLF または ZGEQLF で求められた正規直交列を持つ M × N 複素行列 Q を生成します。この Q は、次数 M の基本鏡映行列 K 個の積の最後の N 列として定義されます。
CUNG2R (P) または ZUNG2R (P)	CGEQRF または ZGEQRF で求められた正規直交列を持つ M × N 複素行列 Q を生成します。この Q は、次数 M の基本鏡映行列 K 個の積の最後の N 列として定義されます。
CUNGBR (P) または ZUNGBR (P)	CGEBRD または ZGEBRD で求められた二重対角形式への縮約から、ユニタリ変換行列を生成します。
CUNGHR (P) または ZUNGHR (P)	CGEHRD または ZGEHRD で求められたヘッセンベルグ形式への縮約から、直交変換行列を生成します。
CUNGL2 (P) または ZUNGL2 (P)	CGELQF または ZGELQF で求められた LQ 因子分解から、ユニタリ行列 Q の全部または一部を生成します (非ブロック化アルゴリズム)。
CUNGLQ (P) または ZUNGLQ (P)	CGELQF または ZGELQF で求められた LQ 因子分解から、ユニタリ行列 Q を生成します。
CUNGQL (P) または ZUNGQL (P)	CGEQLF または ZGEQLF で求められた QL 因子分解から、ユニタリ行列 Q を生成します。
CUNGQR (P) または ZUNGQR (P)	CGEQRF または ZGEQRF で求められた QR 因子分解から、ユニタリ行列 Q を生成します。
CUNGR2 (P) または ZUNGR2 (P)	CGERQF または ZGERQF で求められた RQ 因子分解から、ユニタリ行列 Q の全部または一部を生成します (非ブロック化アルゴリズム)。
CUNGRQ (P) または ZUNGRQ (P)	CGERQF または ZGERQF で求められた RQ 因子分解から、ユニタリ行列 Q を生成します。
CUNGTR (P) または ZUNGTR (P)	CHETRD または ZHETRD によって三重対角形式に縮約されたユニタリ行列を生成します。

ルーチン	機能
CUNM2L または ZUNM2L	CGEQLF または ZGEQLF で求められた QL 因子分解から得られたユニタリ行列を、一般行列に乗算します (非ブロック化アルゴリズム)。
CUNM2R または ZUNM2R	CGEQRF または ZGERLF で求められた QR 因子分解から得られたユニタリ行列を、一般行列に乗算します (非ブロック化アルゴリズム)。
CUNMBR または ZUNMBR	CGEBRD または ZGEBRD によって二重対角形式に縮約されたユニタリ変換行列を、一般行列に乗算します。
CUNMHR または ZUNMHR	CGEHRD または ZGEHRD によってヘッセンベルグ形式に縮約されたユニタリ行列を、一般行列に乗算します。
CUNML2 または ZUNML2	CGELQF または ZGELQF で求められた LQ 因子分解から得られたユニタリ行列を、一般行列に乗算します (非ブロック化アルゴリズム)。
CUNMLQ または ZUNMLQ	CGELQF または ZGELQF で求められた LQ 因子分解から得られたユニタリ行列を、一般行列に乗算します。
CUNMQL または ZUNMQL	CGEQLF または ZGEQLF で求められた QL 因子分解から得られたユニタリ行列を、一般行列に乗算します。
CUNMQR または ZUNMQR	CGEQRF または ZGEQRF で求められた QR 因子分解から得られたユニタリ行列を、一般行列に乗算します。
CUNMR2 または ZUNMR2	CGERQF または ZGERQF で求められた RQ 因子分解から得られたユニタリ行列を、一般行列に乗算します (非ブロック化アルゴリズム)。
CUNMR3 または ZUNMR3	CTZRZF または ZTZRZF で求められた RZ 因子分解から得られたユニタリ行列を、一般行列に乗算します (非ブロック化アルゴリズム)。
CUNMRQ または ZUNMRQ	CGERQF または ZGERQF で求められた RQ 因子分解から得られたユニタリ行列を、一般行列に乗算します。
CUNMRZ または ZUNMRZ	CTZRZF または ZTZRZF で求められた RZ 因子分解から得られたユニタリ行列を、一般行列に乗算します。
CUNMTR または ZUNMTR	CHETRD または ZHETRD によって三重対角形式に縮約されたユニタリ変換行列を、一般行列に乗算します。

表 55 パック格納のユニタリ行列ルーチン

ルーチン	機能
CUPGTR (P) または ZUPGTR (P)	CHPTRD または ZHPTRD で求められた三重対角行列からユニタリ変換行列を生成します。
CUPMTR または ZUPMTR	CHPTRD または ZHPTRD によって三重対角形式に縮約されたユニタリ変換行列を、一般行列に乗算します。

BLAS1 ルーチン

表56「BLAS1 (Basic Linear Algebra Subprograms、レベル 1) ルーチン」に、Oracle Developer Studio パフォーマンスライブラリの BLAS1 ルーチンを示します。現在、Oracle Developer Studio パフォーマンスライブラリの BLAS1 ルーチンは並列化されていません。

表 56 BLAS1 (Basic Linear Algebra Subprograms、レベル 1) ルーチン

ルーチン	機能
SASUM、DASUM、SCASUM、DZASUM	ベクトルの絶対値の和
*x*AXPY	スカラーとベクトルの積をベクトルに加算
*x*COPY	ベクトルをコピー
SDOT、DDOT、DSDOT、SDSDOT、CDOTU、ZDOTU、DQDOTA、DQDOTI	ドット積 (内積)。4 倍精度 DQDOTA、DQDOTI は SPARC でのみ使用可能
CDOTC、ZDOTC	ドット積 (第 1 ベクトルを共役化)
SNRM2、DNRM2、SCNRM2、DZNRM2	ベクトルのユークリッドノルム
*x*ROTG	ギブンス平面回転を設定
SROT、DROT、CSROT、ZDROT	ギブンス平面回転を適用
SROTMG、DROTMG	修正ギブンス平面回転を設定
SROTM、DROTM	修正ギブンス回転を適用
ISAMAX、IDAMAX、ICAMAX、IZAMAX	最大絶対値を持つ要素のインデックス
*x*SCAL、CSSCAL、ZDSCAL	ベクトルをスカラー倍
*x*SWAP	2 つのベクトルを交換
CVMUL、ZVMUL	複素ベクトルどうしの積のスカラー倍を計算

BLAS2 ルーチン

表57「BLAS2 (Basic Linear Algebra Subprograms、レベル 2) ルーチン」に、Oracle Developer Studio パフォーマンスライブラリの BLAS2 ルーチンを示します。(P) はルーチンが並列化されていることを表します。

表 57 BLAS2 (Basic Linear Algebra Subprograms、レベル 2) ルーチン

ルーチン	機能
*x*GBMV	バンド格納の行列とベクトルの積
*x*GEMV (P)	一般行列とベクトルの積
SGER (P)、DGER (P)、CGERC (P)、ZGERC (P)、CGERU (P)、ZGERU (P)	一般行列に対するランク 1 更新
CHBMV または ZHBMV	バンド格納のエルミート行列とベクトルの積
CHEMV (P) または ZHEMV (P)	エルミート行列とベクトルの積
CHER (P) または ZHER (P)	エルミート行列に対するランク 1 更新
CHER2 または ZHER2	エルミート行列に対するランク 2 更新

ルーチン	機能
CHPMV (P) または ZHPMV (P)	パック格納のエルミート行列とベクトルの積
CHPR または ZHPR	パック格納のエルミート行列に対するランク 1 更新
CHPR2 または ZHPR2	パック格納のエルミート行列に対するランク 2 更新
SSBMV または DSBMV	バンド格納の対称行列とベクトルの積
SSPMV (P) または DSPMV (P)	パック格納の対称行列とベクトルの積
SSPR または DSPR	パック格納の実対称行列に対するランク 1 更新
SSPR2 (P) または DSPR2 (P)	パック格納の実対称行列に対するランク 2 更新
*x*SYMV (P)	対称行列とベクトルの積
SSYR (P) または DSYR (P)	実対称行列に対するランク 1 更新
SSYR2 (P) または DSYR2 (P)	実対称行列に対するランク 2 更新
*x*TBMV	バンド格納の三角行列とベクトルの積
*x*TBSV	バンド格納の三角行列の連立 1 次方程式の解
*x*TPMV	パック格納の三角行列とベクトルの積
*x*TPSV	パック格納の三角行列の連立 1 次方程式の解
*x*TRMV (P)	三角行列とベクトルの積
*x*TRSV (P)	三角行列の連立 1 次方程式の解

BLAS3 ルーチン

表58「BLAS3 (Basic Linear Algebra Subprograms、レベル 3) ルーチン」に、Oracle Developer Studio パフォーマンスライブラリの BLAS3 ルーチンを示します。(P) はルーチンが並列化されていることを表します。

表 58 BLAS3 (Basic Linear Algebra Subprograms、レベル 3) ルーチン

ルーチン	機能
*x*GEMM (P)	2 つの一般行列の積
CHEMM (P) または ZHEMM (P)	エルミート行列と一般行列の積
CHERK (P) または ZHERK (P)	エルミート行列のランク k 更新
CHER2K (P) または ZHER2K (P)	エルミート行列のランク 2k 更新
*x*SYMM (P)	対称行列と一般行列の積
*x*SYRK (P)	対称行列のランク k 更新
*x*SYR2K (P)	対称行列のランク 2k 更新

ルーチン	機能
*x*TRMM (P)	三角行列と一般行列の積
*x*TRSM (P)	三角行列の連立方程式の解

スパース BLAS ルーチン

表59「スパース BLAS ルーチン」に、Oracle Developer Studio パフォーマンスライブラリのスパース BLAS ルーチンを示します。(P) はルーチンが並列化されていることを表します。

表 59 スパース BLAS ルーチン

ルーチン	機能
*x*AXPYI	スパースベクトル *X* のスカラー倍をフルベクトル *Y* に加算します。
*x*BCOMM (P)	ブロック座標形式の行列と行列の乗算。
*x*BDIMM (P)	ブロック対角形式の行列と行列の乗算。
*x*BDISM (P)	ブロック対角形式の三角求解。
*x*BELMM (P)	ブロック Ellpack 形式の行列と行列の乗算。
*x*BELSM (P)	ブロック Ellpack 形式の三角求解。
*x*BSCMM (P)	ブロック圧縮スパース列形式の行列と行列の乗算。
*x*BSCSM (P)	ブロック圧縮スパース列形式の三角求解。
*x*BSRMM (P)	ブロック圧縮スパース行形式の行列と行列の乗算。
*x*BSRSM (P)	ブロック圧縮スパース行形式の三角求解。
*x*COOMM (P)	座標形式の行列と行列の乗算。
*x*CSCMM (P)	圧縮スパース列形式の行列と行列の乗算。
*x*CSCSM (P)	圧縮スパース列形式の三角求解。
*x*CSRMM (P)	圧縮スパース行形式の行列と行列の乗算。
*x*CSRSM (P)	圧縮スパース行形式の三角求解。
*x*DIAMM (P)	対角形式の行列と行列の乗算。
*x*DIASM (P)	対角形式の三角求解。
SDOTI、DDOTI、CDOTUI、または ZDOTUI	スパースベクトルとフルベクトルのドット積を計算します。
CDOTCI または ZDOTCI	スパースベクトルとフルベクトルの共役ドット積を計算します。
*x*ELLMM (P)	Ellpack 形式の行列と行列の乗算。
*x*ELLSM (P)	Ellpack 形式の三角求解。
*x*GTHR	フルベクトルが与えられた場合に、スパースベクトルおよび対応するインデックスベクトルを作成します。
*x*GTHRZ	フルベクトルが与えられた場合に、スパースベクトルおよび対応するインデックスベクトルを作成し、フルベクトルをゼロにします。
*x*JADMM (P)	鋸歯状対角形式の行列と行列の乗算。

ルーチン	機能
SJADRP または DJADRP	鋸歯状対角行列の右置換。
*x*JADSM (P)	鋸歯状対角形式の三角求解。
SROTI または DROTI	スパースベクトルとフルベクトルにギブンス回転を適用します。
*x*SCTR	スパースベクトルおよび対応するインデックスベクトルが与えられた場合に、それらの要素をフルベクトルに入れます。
*x*SKYMM (P)	スカイライン形式の行列と行列の乗算。
*x*SKYSM (P)	スカイライン形式の三角求解。
*x*VBRMM (P)	可変ブロックスパース行形式の行列と行列の乗算。
*x*VBRSM (P)	可変ブロックスパース行形式の三角求解。

スパースソルバールーチン

次の表に、Oracle Developer Studio パフォーマンスライブラリの SPSOLVE および SuperLU スパースソルバーのルーチンを示します。(P) はルーチンが並列化されていることを表します。

表 60 SPSOLVE ルーチン

ルーチン	機能
xGSSFS (P)	SPSOLVE のワンコールインタフェース。
xGSSIN	SPSOLVE の初期化。
xGSSOR	埋め込みを低減する順序付けおよび記号分解。
xGSSFA (P)	行列の値の入力および数値分解。
xGSSSL	三角求解。
xGSSUO	ユーザー指定の順序付け置換を設定します。
xGSSRP	ソルバーで使用される置換を返します。
xGSSCO	係数行列の条件数の推定を返します。
xGSSDA	SPSOLVE のメモリーを割り当て解除します。
xGSSPS	ソルバー統計を出力します。

表 61 SuperLU ルーチン

ルーチン	機能
xgstrf	因子分解を計算します
xgssvx	因子分解と求解 (エキスパートドライバ)
xgssv	因子分解と求解 (単純ドライバ)
xgstrs	三角求解を計算します
xgsrfs	計算解を改良し、誤差限界を提供します
xlangs	1 ノルム、フロベニウスノルム、または無限大ノルムを計算します

ルーチン	機能
xgsequ	行および列のスケーリングを計算します
xgscon	条件数の逆数を推定します
xlaqgs	一般スパース行列を均衡化します
LUSolveTime	求解段階で費やされた時間を返します
LUFactTime	因子分解段階で費やされた時間を返します
LUFactFlops	因子分解段階での浮動小数点演算の回数を返します
LUSolveFlops	求解段階での浮動小数点演算の回数を返します
xQuerySpace	メモリー統計の情報を返します
sp_ienv	指定されたマシンに依存するパラメータを返します
xPrintPerf	計算ルーチンによって収集された統計を出力します
set_default_options	ソルバーの動作を制御するパラメータをデフォルトオプションに設定します
StatInit	パフォーマンス統計を格納する構造体を割り当て、初期化します
StatFree	パフォーマンス統計を格納する構造体を解放します
Destroy_Dense_Matrix	密形式の SuperMatrix を割り当て解除します
Destroy_SuperNode_Matrix	スーパーノード形式の SuperMatrix を割り当て解除します
Destroy_CompCol_Matrix	圧縮スパース列形式の SuperMatrix を割り当て解除します
Destroy_CompCol_Permuted	置換された圧縮スパース列形式の SuperMatrix を割り当て解除します
Destroy_SuperMatrix_Store	SuperMatrix の行列を格納する実際の格納領域を割り当て解除します
xCopy_CompCol_Matrix	圧縮スパース列形式の SuperMatrix をコピーします
xCreate_CompCol_Matrix	圧縮スパース列形式の SuperMatrix を割り当てます
xCreate_Dense_Matrix	密形式の SuperMatrix を割り当てます
xCreate_CompRow_Matrix	圧縮スパース行形式の SuperMatrix を割り当てます
xCreate_SuperNode_Matrix	スーパーノード形式の SuperMatrix を割り当てます
sp_preorder	元のスパース行列の列を置換します
sp_sgemm sp_dgemm sp_cgemm sp_zgemm	SuperMatrix に密行列を乗算します

信号処理ライブラリルーチン

Oracle Developer Studio パフォーマンスライブラリには、高速フーリエ変換、サイン変換とコサイン変換、および畳み込みと相関を計算するためのルーチンが含まれています。

FFT ルーチン

Oracle Developer Studio パフォーマンスライブラリでは、以前のリリースの Oracle Developer Studio パフォーマンスライブラリで提供されていた FFTPACK および VFFTPACK ルーチンの一部に代わる、一連の FFT インタフェースが提供されています。古

い FFT インタフェースは下位互換性のために残されていますが、新しいインタフェースを使用することをお勧めします。個々の FFT ルーチンの詳細については、セクション 3P のマニュアルページを参照してください。

表62「FFT ルーチン」は、Oracle Developer Studio パフォーマンスライブラリの FFT ルーチンと、対応する FFTPACK および VFFTPACK ルーチンのマッピングを示しています。(P) はルーチンが並列化されていることを表します。

表 62 FFT ルーチン

ルーチン	置き換え対象	機能
CFFTC (P)	CFFTI	三角法の重みと係数のテーブルを初期化します。あるいは、複素数列の 1 次元前方/逆 FFT を計算します。
	CFFTF (P)	
	CFFTB (P)	
CFFTC2 (P)	CFFT2I	三角法の重みと係数のテーブルを初期化します。あるいは、2 次元複素数配列の 2 次元前方/逆 FFT を計算します。
	CFFT2F (P)	
	CFFT2B (P)	
CFFTC3 (P)	CFFT3I	三角法の重みと係数のテーブルを初期化します。あるいは、3 次元複素数配列の 3 次元前方/逆 FFT を計算します。
	CFFT3F (P)	
	CFFT3B (P)	
CFFTCM (P)	VCFFTI	三角法の重みと係数のテーブルを初期化します。あるいは、2 次元複素数配列に格納された一連のデータ列の 1 次元前方/逆 FFT を計算します。
	VCFFTF (P)	
	VCFFTB (P)	
CFFTS	RFFTI、RFFTB	三角法の重みと係数のテーブルを初期化します。あるいは、複素数列の 1 次元逆 FFT を計算します。
	EZFFTI、EZFFTB	
CFFTS2	RFFT2I	三角法の重みと係数のテーブルを初期化します。あるいは、2 次元複素数配列の 2 次元逆 FFT を計算します。
	RFFT2B	
CFFTS3 (P)	RFFT3I	三角法の重みと係数のテーブルを初期化します。あるいは、3 次元複素数配列の 3 次元逆 FFT を計算します。
	RFFT3B	
CFFTSM	VRFFTI	三角法の重みと係数のテーブルを初期化します。あるいは、2 次元複素数配列に格納された一連のデータ列の 1 次元逆 FFT を計算します。
	VRFFTB (P)	
DFFTZ	DFFTI、DFFTF	三角法の重みと係数のテーブルを初期化します。あるいは、倍精度数列の 1 次元前方 FFT を計算します。
	DEZFFTI、DEZFFTF	
DFFTZ2	DFFT2I	三角法の重みと係数のテーブルを初期化します。あるいは、2 次元倍精度配列の 2 次元前方 FFT を計算します。
	DFFT2F	
DFFTZ3 (P)	DFFT3I	三角法の重みと係数のテーブルを初期化します。あるいは、3 次元倍精度配列の 3 次元前方 FFT を計算します。

ルーチン	置き換え対象	機能
	DFFT3F	
DFFTZM	VDFFTI	三角法の重みと係数のテーブルを初期化します。あるいは、2 次元倍精度配列に格納された一連のデータ列の 1 次元前方 FFT を計算します。
	VDFFTF (P)	
SFFTC	RFFTI、RFFTF	三角法の重みと係数のテーブルを初期化します。あるいは、実数シーケンスの 1 次元前方 FFT を計算します。
	EZFFTI、EZFFTF	
SFFTC2	RFFT2I	三角法の重みと係数のテーブルを初期化します。あるいは、2 次元実数配列の 2 次元前方 FFT を計算します。
	RFFT2F	
SFFTC3 (P)	RFFT3I	三角法の重みと係数のテーブルを初期化します。あるいは、3 次元実数配列の 3 次元前方 FFT を計算します。
	RFFT3F	
SFFTCM	VRFFTI	三角法の重みと係数のテーブルを初期化します。あるいは、2 次元実数配列に格納された一連のデータ列の 1 次元前方 FFT を計算します。
	VRFFTF (P)	
ZFFTD	DFFTI、DFFTB	三角法の重みと係数のテーブルを初期化します。あるいは、倍精度複素数列の 1 次元逆 FFT を計算します。
	DEZFFTI、DEZFFTB	
ZFFTD2	DFFT2I	三角法の重みと係数のテーブルを初期化します。あるいは、2 次元倍精度複素数配列の 2 次元逆 FFT を計算します。
	DFFT2B	
ZFFTD3 (P)	DFFT3I	三角法の重みと係数のテーブルを初期化します。あるいは、3 次元倍精度複素数配列の 3 次元逆 FFT を計算します。
	DFFT3B	
ZFFTDM	VDFFTI	三角法の重みと係数のテーブルを初期化します。あるいは、2 次元倍精度複素数配列に格納された一連のデータ列の 1 次元逆 FFT を計算します。
	VDFFTB (P)	
ZFFTZ (P)	ZFFTI	三角法の重みと係数のテーブルを初期化します。あるいは、倍精度複素数列の 1 次元前方/逆 FFT を計算します。
	ZFFTF (P)	
	ZFFTB (P)	
ZFFTZ2 (P)	ZFFT2I	三角法の重みと係数のテーブルを初期化します。あるいは、2 次元倍精度複素数配列の 2 次元前方/逆 FFT を計算します。
	ZFFT2F (P)	
	ZFFT2B (P)	
ZFFTZ3 (P)	ZFFT3I	三角法の重みと係数のテーブルを初期化します。あるいは、3 次元倍精度複素数配列の 3 次元前方/逆 FFT を計算します。
	ZFFT3F (P)	
	ZFFT3B (P)	
ZFFTZM (P)	VZFFTI	三角法の重みと係数のテーブルを初期化します。あるいは、2 次元倍精度複素数配列に格納された一連のデータ列の 1 次元前方/逆 FFT を計算します。
	VZFFTF (P)	
	VZFFTB (P)	

高速コサイン/サイン変換

Oracle Developer Studio パフォーマンスライブラリの高速コサイン/サイン変換ルーチンは、FFTPACK (http://www.netlib.org/fftpack/) 内のルーチンに基づいています。接頭辞 V の付いたルーチンは、VFFTPACK (http://www.netlib.org/vfftpack/) 内のルーチンに基づく、ベクトル化されたルーチンです。

表63「サインおよびコサイン変換ルーチン」に、Oracle Developer Studio パフォーマンスライブラリのサインおよびコサイン変換ルーチンを示します。

表 63 サインおよびコサイン変換ルーチン

ルーチン	機能
COSQB、DCOSQB、VCOSQB、VDCOSQB	コサイン 1/4 波合成。
COSQF、DCOSQF、VCOSQF、VDCOSQF	コサイン 1/4 波変換。
COSQI、DCOSQI、VCOSQI、VDCOSQI	コサイン 1/4 波変換および合成を初期化します。
COST、DCOST、VCOST、VDCOST	コサイン偶数波変換。
COSTI、DCOSTI、VCOSTI、VDCOSTI	コサイン偶数波変換を初期化します。
SINQB、DSINQB、VSINQB、VDSINQB	サイン 1/4 波合成。
SINQF、DSINQF、VSINQF、VDSINQF	サイン 1/4 波変換。
SINQI、DSINQI、VSINQI、VDSINQI	サイン 1/4 波変換および合成を初期化します。
SINT、DSINT、VSINT、VDSINT	サイン奇数波変換。
SINTI、DSINT、VSINTI、VDSINTI	サイン奇数波変換を初期化します。

畳み込みおよび相関ルーチン

表64「畳み込みおよび相関ルーチン」に、Oracle Developer Studio パフォーマンスライブラリの畳み込みおよび相関ルーチンを示します。

表 64 畳み込みおよび相関ルーチン

ルーチン	機能
xCNVCOR	畳み込みまたは相関を計算します
xCNVCOR2	2 次元の畳み込みまたは相関を計算します

その他の信号処理ルーチン

表65「畳み込みおよび相関ルーチン」に、Oracle Developer Studio パフォーマンスライブラリのその他の信号処理ルーチンを示します。

表 65 畳み込みおよび相関ルーチン

ルーチン	機能
RFFTOPT、DFFTOPT、CFFTOPT、ZFFTOPT	もっとも近い FFT の長さを計算します
SWIENER または DWEINER	2 つの信号のウィナーデコンボリューションを実行します
xTRANS (P)	配列を転置します

各ルーチンの使用方法については、セクション 3P のマニュアルページを参照してください。

ソートルーチン

表66「ソートルーチン」に、Oracle Developer Studio パフォーマンスライブラリのソートルーチンを示します。

表 66 ソートルーチン

ルーチン	機能
BLAS_DSORT (P)	クイックソートアルゴリズムを使用して実数 (倍精度) ベクトル X を昇順または降順にソートします。
BLAS_DSORTV (P)	クイックソートアルゴリズムを使用して実数 (倍精度) ベクトル X を昇順または降順にソートし、P を置換ベクトルで上書きします。
BLAS_DPERMUTE (P)	DSORTV で出力された置換ベクトル P に関して実数 (倍精度) 配列を置換します。
BLAS_ISORT (P)	クイックソートアルゴリズムを使用して整数ベクトル X を昇順または降順にソートします。
BLAS_ISORTV (P)	クイックソートアルゴリズムを使用して実数ベクトル X を昇順または降順にソートし、P を置換ベクトルで上書きします。
BLAS_IPERMUTE (P)	DSORTV で出力された置換ベクトル P に関して整数配列を置換します。
BLAS_SSORT (P)	クイックソートアルゴリズムを使用して実数ベクトル X を昇順または降順にソートします。
BLAS_SSORTV (P)	クイックソートアルゴリズムを使用して実数ベクトル X を昇順または降順にソートし、P を置換ベクトルで上書きします。
BLAS_SPERMUTE (P)	DSORTV で出力された置換ベクトル P に関して実数配列を置換します。

索引

数字・記号

_64、ルーチン名に付加, 21, 29
%g2、%g3、%g4、および %g5 グローバル整数レジスタ, 25
1/4 波奇数シーケンス
　高速サイン変換ルーチン, 96
1/4 波偶数シーケンス
　高速コサイン変換ルーチン, 95
2D FFT ルーチン
　逆方向 2D FFT, 83
　共役対称性, 84
　順方向 2D FFT, 83
　データストレージ形式, 84
　入力としての実数シーケンス, 84
　入力としての複素数シーケンス, 84
　ルーチン, 74, 85
3D FFT ルーチン
　逆方向 3D FFT, 88
　共役対称性, 88
　順方向 3D FFT, 88
　データストレージ形式, 88
　入力としての実数シーケンス, 88
　入力としての複素数シーケンス, 88
　ルーチン, 74, 89
64 ビットコード
　「64 ビット対応の Oracle Solaris オペレーティング環境」も参照, 29
　C, 31
　Fortran 95, 30
64 ビット整数インタフェースの呼び出し, 29
64 ビット整数インタフェース、呼び出し, 29
64 ビット整数引数, 21
　整数を 64 ビットに上位変換, 29, 30
64 ビット対応の Oracle Solaris オペレーティング環境
　コードのコンパイル, 29
　ルーチン名に _64 を付加, 29
64 ビット対応の Solaris オペレーティング環境
　整数の上位変換, 30

あ

アーキテクチャー, 12
一般行列, 121, 123
一般三重対角行列, 124
一般帯行列, 120
上ヘッセンベルグ行列, 127, 128
エルミート行列, 125
エルミート帯行列, 124

か

開発環境にルーチンを含める, 19
型への非依存性, 21
環境変数
　OMP_STACKSIZE, 33
　STACKSIZE, 33
奇数シーケンス
　高速サイン変換ルーチン, 95
機能, 14
共役対称, 76
共役対称性
　2D FFT ルーチン, 84
　3D FFT ルーチン, 88
　FFT ルーチン, 76
行列
　一般, 41, 121
　一般、一般化問題, 123
　一般三重対角, 124
　一般帯, 120
　一般のペア, 123
　一般、ペア, 123
　上ヘッセンベルグ, 127, 128

エルミート, 125
エルミート帯, 124
構造的対称スパース, 47
三角, 41, 137, 138
三角帯, 136
実対称三重対角, 133
実対称帯, 132
実直交, 128
スパース, 45
対角, 120, 120
台形, 138
対称, 42, 134
対称/エルミート正定値, 130
対称/エルミート正定値三重対角, 131
対称/エルミート正定値帯, 129
対称スパース, 46
二重対角, 118
パック格納のエルミート, 127
パック格納の三角, 137, 137
パック格納の対称, 132
パック格納の対称/エルミート正定値, 131
パック格納の実直交, 128
パック格納のユニタリ, 140
バンド化, 39
三重対角, 43
偶数シーケンス
高速コサイン変換ルーチン, 95
グローバル整数レジスタ, 25
構造的対称スパース行列, 47
高速コサイン変換ルーチン, 97
1/4 波偶数シーケンス, 95
逆変換 (1/4 波偶数シーケンス), 98
逆変換 (複数の 1/4 波偶数シーケンス), 99
偶数シーケンス, 95
順変換 (1/4 波偶数シーケンス), 98
順変換 (複数の 1/4 波偶数シーケンス), 99
順方向および逆方向, 97
複数シーケンス, 98
高速サイン変換ルーチン, 96
1/4 波奇数シーケンス, 96
奇数シーケンス, 95
逆変換 (1/4 波奇数シーケンス), 100
逆変換 (複数の 1/4 波奇数シーケンス), 101
順変換 (1/4 波奇数シーケンス), 100
順変換 (複数の 1/4 波奇数シーケンス), 101
順方向と逆方向, 99
順方向と逆方向 (複数シーケンス), 100
高速フーリエ変換
FFT を参照, 73
互換性、LAPACK, 15
コサイン変換, 94
コンパイル時チェック, 21

さ

サイン変換, 94
三角帯行列, 136
三角行列, 41, 137, 138
三重対角行列, 43
実対称三重対角行列, 133
実対称帯行列, 132
実直交行列, 128
自動コード再構築ツール, 20
スパース行列, 45
構造的対称, 47
対称, 46
スパース BLAS, 143
スレッド
同期, 35
スレッドの数, 34
整数引数を 64 ビットに上位変換, 29, 30
セクション 3P マニュアルページ, 117
相関, 106

た

対角行列, 120, 120
台形行列, 138
対称/エルミート正定値行列, 130
対称/エルミート正定値三重対角行列, 131
対称/エルミート正定値帯行列, 129
対称行列, 42, 134
対称スパース行列, 46
畳み込み, 105
畳み込みと相関
引数, 107
ルーチン, 107, 107
データ型
引数, 107
データストレージ形式
2D FFT ルーチン, 84

3D FFT ルーチン, 88
FFT ルーチン, 77
同期, 35

な

二重対角行列, 118

は

パック格納, 40
パック格納のエルミート行列, 127
パック格納の三角行列, 137, 137
パック格納の対称/エルミート正定値行列, 131
パック格納の対称行列, 132
パック格納の実直交行列, 128
パック格納のユニタリ行列, 140
バンド化された行列, 39
引数
FFT ルーチン, 75
畳み込みと相関, 107
引数データ型
サマリー, 107
並列処理
スレッドの数, 34

ま

マニュアルページ
section 3P, 73
セクション 3P, 117

や

呼び出し規則
C, 25
f77/f95, 20

ら

離散フーリエ変換
DFT を参照, 73
ルーチン
1 次元 FFT ルーチン, 74, 77
2D FFT ルーチン, 74, 85
3D FFT ルーチン, 74, 89
BLAS1, 141
BLAS2, 141
BLAS3, 142
C の呼び出し規則, 25, 25
f95 の呼び出し規則, 20
FFTPACK, 146, 148
LAPACK, 117
VFFTPACK, 146, 148
逆方向高速コサイン変換ルーチン, 97
逆方向高速コサイン変換ルーチン (1/4 波偶数シーケンス), 98
逆方向高速コサイン変換ルーチン (複数の 1/4 波偶数シーケンス), 99
逆方向高速サイン変換ルーチン, 99
逆方向高速サイン変換ルーチン (1/4 波奇数シーケンス), 100
逆方向高速サイン変換ルーチン (複数シーケンス), 100
逆方向高速サイン変換ルーチン (複数の 1/4 波奇数シーケンス), 101
高速コサイン変換ルーチン, 95, 95, 97
高速コサイン変換ルーチン (複数シーケンス), 98
高速サイン変換ルーチン, 95, 96, 96
順方向および逆方向 FFT, 74
順方向高速コサイン変換ルーチン, 97
順方向高速コサイン変換ルーチン (1/4 波偶数シーケンス), 98
順方向高速コサイン変換ルーチン (複数の 1/4 波偶数シーケンス), 99
順方向高速サイン変換ルーチン, 99
順方向高速サイン変換ルーチン (1/4 波奇数シーケンス), 100
順方向高速サイン変換ルーチン (複数シーケンス), 100
順方向高速サイン変換ルーチン (複数の 1/4 波奇数シーケンス), 101
スパース BLAS, 143
畳み込みと相関, 107, 107
ルーチンの置き換え, 19

B

BLAS1, 11, 141

BLAS2, 11, 141
BLAS3, 11, 142

C

C
  64 ビットコード, 31
  配列の格納, 25
  ルーチンの呼び出し規則, 25
  例, 25
C インタフェース
  Fortran インタフェースとの比較, 24
  利点, 24
  ルーチンの呼び出し規則, 25
CLAPACK, 13

D

`-dalign`, 28
DFT, 73
  FFT と DFT の効率性, 73, 73

F

`f95` インタフェース
  呼び出し規則, 20
FFT, 73
  FFT と DFT の効率性, 73, 73
FFT ルーチン
  1 次元 FFT ルーチン, 74, 77
  1 次元逆方向 FFT, 76
  1 次元逆方向 FFT (極形式), 76
  1 次元順方向 FFT, 76
  1 次元順方向 FFT (極形式), 76
  2D FFT ルーチン, 74
  3D FFT ルーチン, 74
  共役対称性, 76
  順方向と逆方向, 74
  データストレージ形式, 77
  入力としての実数シーケンス, 76
  入力としての複素数シーケンス, 76
  引数, 75
  もっとも効率的な計算のためのシーケンス長, 75
  もっとも効率的な計算のためのシーケンスの長さ, 93
FFTPACK, 12, 94, 146, 148
Fortran 95
  64 ビットコード, 30
  `USE SUNPERF`, 21
  型への非依存性, 21
  コンパイル時チェック, 21
Fortran インタフェース
  サマリー, 20

L

`-library=sunperf`, 15, 28
LAPACK 90, 13
LAPACK, 11, 117
LAPACK の互換性, 15
LINPACK, 12

M

`malloc`, 25
MT セーフルーチン, 23

N

Netlib, 12
Netlib Sparse-BLAS 0.5, 11
Netlib Sparse-BLAS, 11
Netlib スパース BLAS
  命名規則, 48
NIST Fortran スパース BLAS
  命名規則, 49

O

`OMP_STACKSIZE` 環境変数, 33

S

section 3P マニュアルページ, 73
Sparse Solver , 12
SPSOLVE スパースソルバールーチン, 50
`STACKSIZE` 環境変数, 33
SUNPERF モジュール, 21

SuperLU 3.0, 11
SuperLU スパースソルバールーチン, 61

U

USE SUNPERF
　Fortran 95 機能の有効化, 21

V

VFFTPACK, 12, 94, 146, 148

X

-xarch, 28
*x*FFTOPT, 94
XBLAS, 12