Fortran ユーザーズガイド

Forte Developer 6 update 2
(Sun WorkShop 6 update 2)

Sun Microsystems, Inc.
901 San Antonio Road
Palo Alto, CA 94303
U.S.A. 650-960-1300

Part No. 816-0891-01
2001 年 8 月 Revision A

原典： *Fortran User's Guide*
Part No: 806-7988-10
Revision A

製品名の変更について

Sun は新しい開発製品戦略の一環として、Sun の開発ツール群の製品名を Sun WorkShop™ から Forte™ Developer に変更いたしました。製品自体の内容に変更はなく、従来通りの高品質をお届けいたします。

これまでの Sun の主力製品である基本プログラミングツールに、Forte Fusion™ や Forte™ for Java™ といった Forte 開発ツールの得意とする、マルチプラットフォームおよびビジネスアプリケーション実装の機能を盛り込むことで、より広範囲できめ細かな製品ラインが完成されました。

WorkShop 5.0 で使用されていた名称と、Forte Developer 6 で使用される新しい名称の対応については、以下の表をご覧ください。

旧名称	新名称
Sun Visual WorkShop™ C++	Forte™ C++ Enterprise Edition 6
Sun Visual WorkShop™ C++ Personal Edition	Forte™ C++ Personal Edition 6
Sun Performance WorkShop™ Fortran	Forte™ for High Performance Computing 6
Sun Performance WorkShop™ Fortran Personal Edition	Forte™ Fortran Desktop Edition 6
Sun WorkShop Professional™ C	Forte™ C 6
Sun WorkShop™ University Edition	Forte™ Developer University Edition 6

製品名の変更に加えて、次の 2 つの製品について大きな変更があります。

- Forte for High Performance Computing には Sun Performance WorkShop Fortran に含まれていたすべてのツール、および C++ コンパイラが含まれます。したがって、High Performance Computing のユーザーは開発用に 1 つの製品だけを購入すれば済むことになります。
- Forte Fortran Desktop Edition は以前の Sun Performance WorkShop Personal Edition と同じです。ただし、この製品に含まれる Fortran コンパイラでは、自動並列化されたコード、および明示的な指令に基づいた並列コードは生成できません。この機能は Forte for High Performance Computing に含まれる Fortran コンパイラでは使用できます。

Sun の開発製品を引き続きご利用いただきましてありがとうございます。今後もみなさまのご要望にお応えする製品をお届けできるよう努力してまいります。

目次

表目次

はじめに

このマニュアルでは、2 つの Sun WorkShop™ 6 Fortran コンパイラ `f77` (FORTRAN 77) および `f95` (Fortran 95) のコンパイル時の環境およびコマンド行オプションについて説明します。実行時エラーメッセージと新しい機能については、付録に記載してあります。

このマニュアルは、Fortran に関する実用的な知識を持ち、Sun Fortran コンパイラの効率的な使用法を学ぼうとしている、科学者、技術者、プログラマを対象に書かれています。また、Solaris™ オペレーティング環境や UNIX® の一般的な知識を持つ読者を対象としています。

入出力、アプリケーション開発、ライブラリの作成とその使用、プログラム解析、移植、最適化、並列化などの Fortran のプログラミングについては、関連マニュアル『Fortran プログラミングガイド』 を参照してください。

その他の関連マニュアルとして、『Fortran ライブラリ・リファレンス』、『FORTRAN 77 言語リファレンス』があります。xix ページの「関連マニュアル」を参照してください。

内容の紹介

このマニュアルは次の章と付録から構成されています。

第 1 章では、このコンパイラの機能について簡単に説明します。

第 2 章では、コンパイラの使用環境について説明します。

第 3 章では、すべてのコマンド行オプションについて詳しく説明します。

付録 A では、Fortran の実行時ライブラリおよびオペレーティング環境によって表示されるエラーメッセージについて説明します。

付録 B では、今回のリリースで追加された機能と変更された機能について説明します。

付録 C では、Sun `f95` コンパイラと標準の Fortran 95 との相違点、および `f77` プログラムとの非互換性について説明します。

付録 D では、 コンパイラの `-xtarget` オプションで使用できるプラットフォームシステム名とその展開について説明します。

付録 E では、`f77` や `f95` の Fortran コンパイラで認識されている指令について要約します。

書体と記号について

次の表と記述は、このマニュアルで使用している書体と記号について説明しています。

書体または記号	意味	例
`AaBbCc123`	コマンド名、ファイル名、ディレクトリ名、画面上のコンピュータ出力、コーディング例。	`.login` ファイルを編集します。 `ls -a` を使用してすべてのファイルを表示します。 `machine_name% You have mail.`
AaBbCc123	ユーザーが入力する文字を、画面上のコンピュータ出力と区別して表わします。	`machine_name% su` `Password:`
AaBbCc123 または ゴシック	コマンド行の可変部分。実際の名前または実際の値と置き換えてください。	`rm` *filename* と入力します。 `rm` ファイル名 と入力します。
『 』	参照する書名を示します。	『SPARCstorage Array ユーザーマニュアル』

書体または記号	意味	例
「 」	参照する章、節、または、強調する語を示します。	第 6 章「データの管理」を参照してください。 この操作ができるのは、「スーパーユーザー」だけです。
\	枠で囲まれたコード例で、テキストがページ行幅を超える場合、バックスラッシュは、継続を示します。	`machinename% grep '^#define \` `  XV_VERSION_STRING'`
➤	階層メニューのサブメニューを選択することを示します。	作成:「返信」➤「送信者へ」

- 小さい三角(△)は意味のある空白を示します。

```
ΔΔ36.001
```

- FORTRAN 77 の例はタブ書式で、Fortran 95 の例は自由書式で示します。FORTRAN 77 と Fortran 95 に共通の例は、特に明示がない限りタブ書式で示します。
- FORTRAN 77 規格では、「FORTRAN」とすべて大文字で表記する旧表記規則を使用しています。サンのマニュアルでは、FORTRAN と Fortran の両方を使用しています。現在の表記規則では、「Fortran 95」と小文字を使用しています。
- オンラインマニュアル (man) ページへの参照は、トピック名とセクション番号とともに表示されます。たとえば、GETENV への参照は、`getenv(3F)` と表示されます。`getenv(3F)` とは、このページにアクセスするためのコマンドが `man -s 3F getenv` であるという意味です。
- システム管理者は Sun Workshop Fortran コンパイラとそのサポート機器を、<*install_point*>/SUNWspro/ にインストールできます。<*install_point*>/SUNWspro/ は、通常、標準のインストールを示す `/opt` を指定します。本書では `/opt` をインストールポイントとして使用します。

シェルプロンプトについて

シェル	プロンプト
UNIX の C シェル	machine_name%
UNIX の Bourne シェルと Korn シェル	machine_name$
スーパーユーザー (シェルの種類を問わない)	#

サポートしているプラットフォーム

この Sun WorkShop™ リリースでは、Solaris™ SPARC™ プラットフォーム版と Solaris™ Intel プラットフォーム版をオペレーティング環境とするバージョン 2.6、7、および 8 をサポートしています。

Sun WorkShop の開発ツールとマニュアルページへのアクセス

Sun WorkShop の製品コンポーネントとマニュアルページは、標準の /usr/bin/ と /usr/share/man の各ディレクトリにインストールされていません。SunWorkShop のコンパイラとツールにアクセスするには、PATH 環境変数に SunWorkshop コンポーネントディレクトリを必要とします。SunWorkshop マニュアルページにアクセスするには、PATH 環境変数に SunWorkshop マニュアルページが必要です。

PATH 変数についての詳細は、csh(1)、sh(1) および ksh(1) のマニュアルページを参照してください。MANPATH 変数についての詳細は、 man(1) のマニュアルページを参照してください。このリリースにアクセスするために PATH および MANPATH 変数を設定する方法の詳細は、『Sun WorkShop 6 update 2 インストールガイド』を参照するか、システム管理者にお問い合わせください。

注 – この節に記載されている情報は Sun WorkShop 6 update 2 製品が /opt ディレクトリにインストールされていることを想定しています。 Sun WorkShop 製品が /opt 以外のディレクトリにインストールされている場合は、システム管理者に実際のパスをお尋ねください。

Sun WorkShop コンパイラとツールへのアクセス方法

PATH 環境変数を変更して Sun WorkShop コンパイラとツールにアクセスできるようにする必要があるかどうか判断するには以下を実行します。

PATH 環境変数を設定する必要があるかどうか判断するには

1. 次のように入力して、PATH 変数の現在値を表示します。

```
% echo $PATH
```

2. 出力内容から /opt/SUNWspro/bin を含むパスの文字列を検索します。

   パスがある場合は、PATH 変数は Sun WorkShop 開発ツールにアクセスできるように設定されています。パスがない場合は、次の指示に従って、PATH 環境変数を設定してください。

PATH 環境変数を設定して Sun WorkShop のコンパイラとツールにアクセスする

1. C シェルを使用している場合は、ホームの .cshrc ファイルを編集します。Bourne シェルまたは Korn シェルを使用している場合は、ホームの .profile ファイルを編集します。

2. 次のパスを PATH 環境変数に追加します。

   /opt/SUNWspro/bin

Sun WorkShop マニュアルページへのアクセス方法

Sun WorkShop マニュアルページにアクセスするために MANPATH 変数を変更する必要があるかどうかを判断するには以下を実行します。

MANPATH 環境変数を設定する必要があるかどうか判断するには

1. 次のように入力して、workshop マニュアルページを表示します。

```
% man workshop
```

2. 出力された場合、内容を確認します。

   workshop(1) マニュアルページが見つからないか、表示されたマニュアルページがインストールされたソフトウェアの現バージョンのものと異なる場合は、次の指示に従って MANPATH 環境変数を設定してください。

MANPATH 変数を設定して Sun WorkShop マニュアルページにアクセスする

1. C シェルを使用している場合は、ホームの .cshrc ファイルを編集します。Bourne シェルまたは Korn シェルを使用している場合は、ホームの .profile ファイルを編集します。
2. 次のパスを PATH 環境変数に追加します。

   /opt/SUNWspro/man

Sun WorkShop マニュアルへのアクセス

Sun WorkShop の製品マニュアルには、以下からアクセスできます。

- 製品マニュアルは、ご使用のローカルシステムまたはネットワークの製品にインストールされているマニュアルの索引から入手できます。

  Netscape™ Communicator 4.0 または互換性がある Netscape バージョンのブラウザで次のファイルにポイントします。

  /opt/SUNWspro/docs/ja/index.html

  製品ソフトウェアが /opt ディレクトリにインストールされていない場合は、システム上でこのディレクトリに相当するパスをシステム管理者に問い合わせてください。

■ マニュアルは、docs.sun.com の Web サイトで入手できます。

インターネットの docs.sun.com Web サイト (`http://docs.sun.com`) から、サンのマニュアルを読んだり、印刷することができます。マニュアルが見つからない場合はローカルシステムまたはネットワークの製品とともにインストールされているマニュアルの索引を参照してください。

関連マニュアル

次の表では、docs.sun.com の Web サイトで利用できる関連マニュアルについて説明します。

マニュアルコレクション	マニュアルタイトル	内容の説明
Forte for High Performance Computing Collection	Fortran プログラミングガイド	入出力、ライブラリ、プログラム分析、デバッグおよびパフォーマンスに関連する内容を記述しています。
	Fortran ライブラリ・リファレンス	Fortran コンパイラによって提供されるライブラリルーチンの詳細について説明しています。
	FORTRAN 77 言語リファレンス	Sun Fortran 77 言語の包括的な参照情報を記載しています。
数値計算ガイド	数値計算ガイド	浮動小数点演算における数値の精度に関する問題について説明しています。

マニュアルコレクション	マニュアルタイトル	内容の説明
Solaris 8 Reference Manual Collection	マニュアルページの節を参照。	Solaris のオペレーティング環境に関する情報を提供しています。
Solaris 8 Software Developer Collection	リンカーとライブラリ	Solaris のリンクエディタと実行時リンカーの操作について説明しています。
Solaris 8 Software Developer Collection	マルチスレッドのプログラミング	POSIX と Solaris スレッド API、同期オブジェクトのプログラミング、マルチスレッド化したプログラムのコンパイル、およびマルチスレッド化したプログラムのツール検索について説明します。

Sun のマニュアルの注文

製品マニュアルは `docs.sun.com` Web サイトまたは Fatbrain.com インターネットブックストアを通じて米国 Sun Microsystems,Inc. に直接注文できます。Fatbrain.com の Sun Documentation Center へは次の URL でアクセスできます。

`http://www.fatbrain.com/documentation/sun`

ご意見の送付先

米国 Sun Microsystems, Inc. では、マニュアルの向上に力を注いでおり、ユーザーのご意見やご提案をお待ちしております。ご意見などがありましたら、次のアドレスまで電子メールをお送りください。

`docfeedback@sun.com`

第1章

ご使用になる前に

このマニュアル (および関連マニュアルの『Fortran プログラミングガイド』) で説明する Sun WorkShop Fortran コンパイラ `f77` および `f95` は、SPARC および UltraSPARC™ プラットフォームの Solaris オペレーティング環境で使用できます。コンパイラは Fortran 言語規格に準拠しており、マルチプロセッサの並列化、最適化されたコードの精巧なコンパイル、C 言語と Fortran 言語のサポートなどの拡張機能を提供しています。

Fortran コンパイラは、Forte™ for High Performance Computing (Sun Performance WorkShop™) ソフトウェアの構成要素です。Fortran 90 のコンパイラである `f90` は、Fortran 95 のコンパイラ、`f95` に名前が変更されました。今回、`f90` コマンドは `f95` の別名になりました。どちらを使用しても、Fortran 95 コンパイラを起動できます。

規格への準拠

- `f77` は、ANSI X3.9-1978 Fortran 規格およびこれに対応する国際標準化機構 (ISO)1539-1980 規格に準拠しています。そのほか、 FIPS 69-1、BS 6832、MIL-STD-1753 にも準拠しています。
- `f95` は、ANSI X3.198-1992、ISO/IEC 1539:1991 および ISO/IEC 1539:1997 規格に準拠するように設計されました。
- 両コンパイラの浮動小数点演算は、IEEE 754-1985 規格および国際規格の IEC 60559:1989 に準拠しています。

- SPARC プラットフォームでは、UltraSPARC™ の実装を含む SPARC V8 および SPARC V9 の機能を利用した最適化を `f77` と `f90` の両方のコンパイラがサポートしています。これらの機能は『SPARC アーキテクチャマニュアル バージョン 8』(トッパン刊) および バージョン 9 (ISBN 0-13-099227-5) で定義されています。
- このマニュアルでは、「規格」 とは前述の規格のバージョンに準拠していることを意味します。また、これらの規格のバージョン以降の機能は、「規格外」 や 「拡張機能」 と呼ばれます。

これらの規格は規格団体によって改訂される場合があります。このコンパイラが準拠している規格のバージョンが改訂されたり、他のバージョンと交換されることがあります。その結果、Sun Fortran コンパイラのリリースが、将来的に、これまでのリリースと互換性を持たなくなる可能性があります。

Sun Fortran コンパイラの特徴

Sun Fortran コンパイラには以下の拡張機能があります。

- 引数、共通ブロック、パラメータなどの整合性をルーチン間で調べる、大域的なプログラム検査機能
- 自動的および明示的にループを並列化する機能など、マルチプロセッサシステムに対するサポートが最適化に統合されている。

注 - Fortran コンパイラの並列化機能には、Forte for HPC のライセンスが必要です。

- `f77`： VAX/VMS Fortran 5.0 拡張機能
  - `NAMELIST`
  - `DO WHILE`
  - 構造体、記録、共用体、マップ
  - 変数形式の式
  - 再帰
  - ポインタ
  - 倍精度複素数
  - 4 倍精度実数
  - 4 倍精度複素数

- TASK COMMON を含む、`f95` 用に拡張された CRAY 形式の並列化指令
- OpenMP 並列化指令は、`f95` で受け付けられます。
- 大域的、ピープホール、および潜在的な並列化の最適化によって、パフォーマンスの高いアプリケーションが生成されます。最適化したアプリケーションの実行速度が、最適化していないコードと比較した場合、明らかに迅速であることをベンチマークが示します。
- Solaris システムでは呼び出し方式が共通しているので、C または C++ 言語で作成したルーチンを Fortran プログラムと結合できます。
- UltraSPARC プラットフォームで 64 ビットの Solaris の環境をサポートします。
- 値による呼び出し `%VAL` が `f77` と `f95` に実装されています。
- FORTRAN 77 と Fortran 95 プログラム、およびオブジェクトバイナリに互換性があります。
- `f95` における間隔演算式

ソフトウェアがリリースされるたびにコンパイラに追加された新機能あるいは拡張機能に関する詳細は、「付録 B」を参照してください。

その他の Fortran ユーティリティ

Fortran でソフトウェアプログラムを開発するには、次のようなユーティリティを利用することができます。

- **`Sun WorkShop Performance Analyzer`** – シングルスレッドアプリケーションおよびマルチスレッドアプリケーション用の詳細なパフォーマンス解析ツール。`analyzer`(1) を参照してください。
- **`asa`** – この Solaris ユーティリティは、1 カラム目に Fortran の改行制御文字のあるファイルを印刷する際の Fortran の出力フィルタです。UNIX システムは改行制御を使用していないため、Fortran の改行制御規則で書式化されたファイルを UNIX のラインプリンタの規則にしたがった書式に変換するときに、`asa` を使用します。詳細は `asa`(1) を参照してください。
- **`fpp`** – Fortran ソースコードプロセッサ。`fpp`(1) を参照してください。

- **fsplit** – このユーティリティは、複数のルーチンで構成される 1 つの Fortran ファイルを、1 ルーチンが 1 ファイルに対応するように複数のファイルに分割します。FORTRAN 77 または Fortran 95 のソースファイルには fsplit を使用します。詳細は、fsplit(1) を参照してください。

デバッグユーティリティ

次のデバッグ用ユーティリティを利用することができます。

- **error** – (f77 のみ) コンパイルエラーメッセージと Fortran のソースファイルを統合するユーティリティです。このユーティリティは、Solaris のインストールをエンドユーザーシステムサポートではなく、開発者システムサポートでインストールした場合に限り、使用できます。また、SUNWbtool パッケージをインストールした場合にも使用できます。
- **-Xlist** – 引数や COMMON ブロックなどのルーチン間での整合性を検査するコンパイラオプションです。
- **Sun WorkShop** – dbx を使用したビジュアルなデバッギング環境を提供し、データビジュアライザおよびパフォーマンスデータコレクタが含まれます。

Sun Performance Library

Sun Performance Library™ は、線形代数やフーリエ変換の演算に使用できる最適化サブルーチンと関数のライブラリです。このライブラリは、LAPACK、BLAS1、BLAS2、BLAS3、FFTPACK、VFFTPACK、および LINPACK といった標準ライブラリを基盤として構築されており、通常 NetLib (www.netlib.org) から利用できます。

Sun Performance Libraryに含まれている各副プログラムは、標準ライブラリのバージョンと同じ演算を行い、同じインタフェースを使用しますが、通常、実行速度も速く、精度もより正確で、マルチプロセッシング環境でも使用することができます。

詳細は、Performance_library の README ファイルおよび『Sun Performance Library User's Guide for Fortran and C』を参照してください (パフォーマンスライブラリルーチンのマニュアルページは、3P のセクションにあります)。

区間演算

Fortran 95 のコンパイラは、新しい二つのコンパイラフラグである `-xia` および `-xinterval` を提供します。これによって、コンパイラは、新しい言語拡張を認識し、適切なコードを生成して、区間演算計算を実行します。

詳細は、『Fortran 95 区間演算プログラミングリファレンス』を参照してください。

マニュアルページ

オンラインマニュアル (man) ページでは、コマンド、関数、サブルーチンに関する説明を迅速に参照することができます。Sun WorkShop マニュアルページにアクセスするための環境変数 MANPATH の正しい設定値については、「はじめに」を参照してください。

マニュアルページは、次のコマンドによって表示することができます。

```
demo$ man topic
```

Fortran 関連のマニュアルでは、マニュアルページへの参照が必要な箇所では、トピック名とマニュアルセクション番号を示しています。たとえば、`f95`(1) を参照する場合は、コマンド行で `man f95` と入力します。`ieee_flags`(3M) などで示されるその他のセクションへは、`man` コマンドで `-s` オプションを指定してアクセスします。

```
demo$ man -s 3M ieee_flags
```

Fortran のライブラリルーチンについては、マニュアルページの 3F セクションに記載されています。

以下に Fortran を使用する場合の、関連マニュアルページを示します。

マニュアルページ	内容
f77(1) と f95(1)	Fortran コンパイラのコマンド行オプション
analyzer(1)	Sun WorkShop Performance Analyzer
asa(1)	改行制御文字がある Fortran ファイルの書式を印刷用に変換する
dbx(1)	対話形式のコマンド行デバッガ
fpp(1)	Fortran ソースコードプリプロセッサ (前処理系)
cpp(1)	C ソースコードプリプロセッサ
fsplit(1)	FORTRAN 77 ルーチンを単一ファイルに分割するプリプロセッサ
ieee_flags(3M)	浮動小数点の例外ビットを検証、設定、解除する
ieee_handler(3M)	浮動小数点例外を処理する
matherr(3M)	数学ライブラリエラー処理ルーチン
ild(1)	オブジェクトファイルに対して使用するインクリメンタルリンカー
ld(1)	オブジェクトファイルに対して使用するリンカー

README ファイル

READMEs ディレクトリには、新機能、ソフトウェアの互換性、バグについての説明、マニュアルが印刷された後に判明した情報を記載したファイルが置かれています。READMEs ディレクトリの場所は、Solaris のバージョンやソフトウェアがインストールされた場所によって異なります。パスは、*install_directory*/SUNWspro/READMEs/ja です。通常インストールの場合、*install_directory* は、/opt になります。

標準インストールの README*s* は /opt/SUNWspro/READMEs/ja にあります。

表 1-1　重要なREADME

README ファイル	内容
fortran_77	FORTRAN 77 コンパイラ f77 の本リリースの新機能と変更点、報告されている制限事項、マニュアルの訂正と補足
fortran_95	Fortran 95 コンパイラ f95 の本リリースの新機能と変更点、報告されている制限事項、マニュアルの訂正と補足
fpp_readme	fpp 機能の概要
interval_arithmetic	f95 における区間演算機能の概要
math_libraries	最適化された専門の数学ライブラリ
omp_directives.pdf	f95 に受け入れる OpenMP 指令の要約（PDFファイル）。
profiling_tools	パフォーマンスプロファイルツール prof、gprof、tcov の使用
runtime_libraries	一般ユーザーライセンスで再配布されるライブラリと実行可能ファイル
64 bit_Compilers	64 ビットの Solaris オペレーティング環境に合わせたコンパイル
performance_library	Sun Performance Library

すべてのコンパイラの README へは、-xhelp=readme コマンド行オプションで簡単にアクセスできます。たとえば、

```
f95 xhelp=readme
```

と指定すると、fortran_95 README ファイルが直接表示されます。

コマンド行ヘルプ

以下のようにコンパイラの -help オプションによって、f77 と f95 のコマンド行オプションの要約を表示することができます。

%f77 -help または
%95 -help

[ ] 中の項目は省略可能。< > 中の項目は変数パラメータ。
縦棒| はリテラル値の選択を意味します。例：
-someoption[=<yes|no>] とある場合、 -someoption は -someoption=yes を意味します。

-a:	tcov の基本ブロックごとのプロファイル処理用データ (旧形式) を収集するコードを生成
-ansi:	ANSI 規格以外の拡張機能を報告
-arg=local:	ENTRY 文の前後で実際の引数を保持
-autopar:	自動選択によるループの並列化 (WorkShop のライセンスが必要)
-Bdynamic:	動的なリンクも許容
-Bstatic:	静的なリンクのみ許容
-C:	実行時の添え字の範囲検査を行う
-c:	コンパイルのみ。.o ファイルを生成し、リンクは行わない
-cg89:	汎用の SPARC V7 アーキテクチャー用のコードを生成
-cg92:	SPARC V8 アーキテクチャー用のコードを生成
-copyargs:	定数引数に対する代入を許可

.... <続く>

第2章

Sun Fortran コンパイラの使用

この章では、Fortran 77 と Fortran 95 コンパイラの使用方法について説明します。

コンパイラの主な使用目的は、Fortran などの手続き型言語で記述されたプログラムを、コンピュータで実行できるデータファイルに変換することです。コンパイル処理の一部として、コンパイラから自動的にリンカーを起動して、実行可能ファイルを生成することもできます。

Sun Fortran 77 と Fortran 95 は次の目的で使用することもできます。

- マルチプロセッサ用の並列化実行ファイルを生成します (`-parallel` オプション) 。
- ソースファイルとサブルーチン間におけるプログラムの整合性を分析し、レポートを作成します (`-Xlist` オプション)。
- ソースファイルを以下のファイルに変換します。
  - 再配置可能なバイナリ (`.o`) ファイル。後で実行可能ファイルまたは静的ライブラリ (`.a`) ファイルにリンクされます。
  - 動的共有ライブラリ (`.so`) ファイル (`-G` オプション)。
- 実行可能ファイルにリンクします。
- 実行時デバッガを有効にして実行可能ファイルを生成します (`-g` オプション)。
- 文単位または手続き単位の実行時のプロファイルを有効にして、実行可能ファイルを生成します (`-pg` オプション)。
- 並列化ループの実行時プロファイルを有効にして、実行可能ファイルを生成します (`-Zlp` オプション)。
- ソースコードを調べて ANSI 標準への準拠を確認します (`-ansi` オプション)。

クイックスタート

ここでは Sun Fortran コンパイラを使用して、Fortran プログラムをコンパイルし、実行する方法について、簡単に説明します。コマンド行オプションの参考情報は、次の章で紹介します。

注 - この章に示すコマンド行の例では、主に `f77` を使用します。特に説明がない限り、`f95` にも同じように適用されます。ただし、出力の内容は多少異なります。

Fortran アプリケーションの基本的な実行手順は次のとおりです。まず、エディタを使用して、`.f`、`.for`、`.f90`、`f95`、`.F`、`.F90` または `F95` という拡張子の付いた Fortran のソースファイルを作成します。次に、コンパイラを起動して実行可能ファイルを生成し、最後にそのファイル名を入力してそのプログラムを実行します。

例 : 画面上にメッセージを出力するプログラムの作成例を示します。

```
demo% cat greetings.f
    PROGRAM GREETINGS
    PRINT *, 'Real programmers write Fortran!'
    END
demo% f77 greetings.f
greetings.f:
  MAIN greetings:
demo% a.out
 Real programmers write Fortran!
demo%
```

この例では、`f77` がソースファイル `greetings.f` をコンパイルし、デフォルトでは `a.out` という実行可能ファイルを生成します。プログラムを起動するには、コマンドプロンプトで実行可能ファイルの名前 `a.out` を入力します。

一般的には、UNIX のコンパイラは実行可能ファイルとして a.out というデフォルトではファイルを生成します。同じ名前のファイルがすでに存在する場合は、次回にコンパイルを実行すると、出力が上書きされ、上書きされると不都合な場合もあります。-o コンパイラオプションを使用すると、実行可能出力ファイルの名前を明示的に指定することができます。

```
demo% f77 -o greetings -fast greetings.f
greetings.f:
MAIN greetings:
demo%
```

上の例では、-o オプションによって、実行可能コードが greetings というファイルに書き込まれるようになります (慣例では、実行可能ファイルの名前には、メインソースファイルから拡張子を除いた名前を指定します)。

また別の方法として、コンパイル処理が終わるごとに、mv コマンドを使用してデフォルトの a.out ファイルの名前を変更することもできます。どちらの方法でも、実行可能ファイルの名前を入力してプログラムを実行します。

```
demo% greetings
Real programmers write Fortran!
demo%
```

以下に、f95 を使用した同じ例を示します。

```
demo% cat greetings.f95
program greetings
print*, 'Real programmers write Fortran 95!'
end
demo% f95 -o greetings greetings.f95
demo% greetings
 Real programmers write Fortran 95!
demo%
```

以下の項では、f77 および f95 で使用するコマンドの表記法、コンパイラのソース行指令、注意事項などについて解説します。次の章では、コマンド行の構文とすべてのオプションについて詳しく説明します。

コンパイラの起動

単純なコンパイラコマンドのシェルプロンプトでの起動方法は次のとおりです。

f77 <オプション> <ファイル名> ...	Fortran 77 コンパイラ起動
f95 <オプション> <ファイル名> ...	Fortran 95 コンパイラ起動

<ファイル> には、1 つ以上の Fortran のソースファイル名を指定します。拡張子として .f、.F、.f90、f95、.F90、F95 または .for が付いている <オプション> には 1 つまたは複数のコンパイラオプションフラグを指定します (.f90 または .f95 の拡張子が付いているファイルは、f95 コンパイラだけが認識する「自由書式」の Fortran 95 ソースファイルです)。

次の例では、f95 は 2 つのソースファイルをコンパイルして、実行時デバッガを有効な状態にして growth という名前の実行可能ファイルを生成します。

```
demo% f95 -g -o growth growth.f fft.f95
```

注 – Sun WorkShop 6 の Fortran 95 のコンパイラは、f95 または f90 のどちらのコマンドを使用しても起動できます。今回のリリースで f90 コマンドは f95 の別名になりました。

コンパイルとリンク処理の流れ

上記の例では、コンパイラは growth.o と fft.o のロードオブジェクトファイルを自動的に生成し、次にシステムリンカーを起動して growth という実行可能プログラムファイルを生成します。

コンパイルの終了後、オブジェクトファイル growth.o と fft.o が残ります。このため、ファイルの再リンクや再コンパイルが簡単に行うことができます。

コンパイルに失敗すると、それぞれのエラーごとにメッセージが表示されます。エラーがあるソースファイルについては .o ファイルや実行可能プログラムファイルは作成されません。

ファイル名の拡張子

コマンド行で入力するファイル名の拡張子によって、コンパイラがそのファイルをどのように処理するかが決まります。以下の表に示されていない拡張子の付いたファイル名、および拡張子のないファイル名は、リンカーに渡されます。

表 2-1　Fortran コンパイラが認識するファイル名の拡張子

拡張子	言語	処理
`.f`	Fortran 77 または Fortran 95 固定形式	ソースファイルをコンパイルし、オブジェクトファイを現在のディレクトリに出力する。オブジェクトファイルのデフォルト名は、ソースファイル名に拡張子`.o` を付けたもの。
`.f95` `.f90`	Fortran 95 自由形式	`.f` と同じ(`f95` のみ)。
`.for`	Fortran 77	`.f` と同じ。
`.F`	Fortran 77 または Fortran 95 固定形式	コンパイルの前に、Fortran 77 のソースファイルを Fortran または C のプリプロセッサで処理する。
`.F95` `.F90`	Fortran 95 自由形式	Fortran コンパイラでコンパイルする前に、Fortran 95 の自由形式のソースファイルを Fortran または C のプリプロセッサで処理する (`f95` のみ)。
`.S`	アセンブラ	アセンブルする前にアセンブラのソースファイルを C プリプロセッサで処理する。
`.s`	アセンブラ	アセンブラでソースファイルをアセンブルする。
`.il`	インライン展開	インライン展開コードのテンプレートファイルを処理する。コンパイラは、テンプレートを使用して、インライン呼び出しを指定したルーチンに展開する。
`.o`	オブジェクトファイル	オブジェクトファイルをリンカーに渡す。
`.a`, `.so`, `.so.`*n*	ライブラリ	ライブラリの名前をリンカーに渡す。`.a` ファイルは静的ライブラリ、`.so` と `.so.`*n* ファイルは動的ライブラリ。

Fortran 95 の自由形式については、付録 C「Fortran 95 の機能と相違点」を参照してください。

ソースファイル

Fortran コンパイラでは、コマンド行に複数のソースファイルを指定することができます。コンパイルユニットとも呼ばれる 1 つのソースファイル中に、複数の手続き (主プログラム、サブルーチン、関数、ブロックデータ、モジュールなど) を記述することができます。アプリケーションは、一つのファイルに一つのソースコード手続きを記述して構成することも、同時に処理される手続きを一つのファイルにまとめて構成することもできます。これらの構成方法の長所と欠点については、『Fortran プログラミングガイド』を参照してください。

ソースファイルプリプロセッサ

`f77` および `f95` は、`fpp` と `cpp` というソースファイルプリプロセッサをサポートしています。いずれのプリプロセッサもコンパイラから起動され、ソースコード「マクロ」とシンボリック定義を展開してから、コンパイルを開始します。デフォルトでは `fpp` が使用されます。`-xpp=cpp` オプションを指定すると、`fpp` から `cpp` にデフォルトを変更できます (-D*name* オプションの説明も参照してください)。

`fpp` は Fortran 言語専用のソースプリプロセッサです。詳細は、`fpp`(1) のマニュアルページと `fpp` の README を参照してください。`fpp` はデフォルトのプリプロセッサで、`f77` の場合は拡張子 `.F` のファイルに対して起動され、`f95` の場合は `.F`、`.F90` または `F95` 拡張子のファイルに対して起動されます。

`fpp` のソースコードは、次に示す web サイト Netlib から使用できます。

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

標準的な Unix C 言語のプリプロセッサについては、cpp(1) を参照してください。Fortran のソースファイルでは、`cpp` よりも `fpp` を使用することをお勧めします。

別々に実行するコンパイルとリンク

コンパイルとリンクをそれぞれ個別に実行することができます。`-c` オプションを指定すると、ソースファイルをコンパイルして `.o` オブジェクトファイルだけが生成され、実行可能ファイルは生成されません。`-c` オプションを指定しない場合、コンパイラはリンカーを起動します。このようにコンパイルとリンクを別々に実行すると、次の例に示すように、1 つのファイルを修正するための目的で全体を再コンパイルする必要がなくなります。

1 つのファイルをコンパイルし、別の手順で他のオブジェクトファイルとリンクする。

```
demo% f95  -c file1.f                    (新規オブジェクトファイルを作成)
demo% f95  -o prgrm file1.o file2.o file3.o (新規実行可能ファイルを作成)
```

リンクを実行する時には (2 行目)、プログラム全体を構成するのに必要なオブジェクトファイルをすべて指定してください。オブジェクトファイルが不足していると、未定義の外部参照エラー (ルーチンの不足) によって、リンクが失敗します。

コンパイルとリンクの整合性

コンパイルとリンクを別々に行う場合、コンパイルとリンクの各オプションを選択するときにそれらの整合性を確認しておく必要があります。以下のオプションを指定してプログラムのコンパイルを行った場合は、同じオプションを指定してリンクを行ってください。。

`-a, -autopar, -B`*x*`, -fast, -G, -L`*path*`, -l`*name*`, -mt, -xmemalign, -nolib, -norunpath, -p, -pg, -xlibmopt, -xlic_lib=`*name*`, -xprofile=`*p*

例 : -a を付けた sbr.f と -a なしの smain.f をコンパイルし、リンクを別々に行います (-a で tcov 旧式のプロファイルを呼び出す)。

```
demo$ f95 -c -a sbr.f
demo$ f95 -c smain.f
demo$ f95 -a sbr.o smain.o   (リンク : -a をリンカーに渡す)
```

オプションによっては、すべてのソースファイルをリンクも含めてそのオプションを指定してコンパイルする必要があります。以下にオプションを示します。

`-autopar, -aligncommon, -d`*x*`, -dalign, -dbl, -explicitpar, -f, -misalign, -native, -parallel, -r8, -xarch=`*a*`, -xcache=`*c*`, -xchip=`*c*`, -xF, -xtarget=`*t*`, -xtypemap, -ztext`

Fortran 95 と FORTRAN 77 のコンパイルが混在している場合のリンク

一般に、プログラム全体を構成するオブジェクトファイルのいずれかが、f95 でコンパイルされている場合、最後のリンクも f95 で実行する必要があります。f95 でコンパイルされた .o オブジェクトファイルがない場合のみ、f77 を使用して実行可能ファイルを生成してください。付録 C「Fortran 95 の機能と相違点」も参照してください。

認識されないコマンド行引数

コンパイラが認識できない引数がコマンド行で指定された場合、リンカーオプション、オブジェクトプログラムのファイル名、またはライブラリ名として解釈されます。

基本的には次のように区別されます。

- 認識されないオプション (- が付いている) には、警告メッセージが出力されます。
- 認識されない 非オプション (- が付いていない) には警告メッセージが出力されません。リンカーにも認識されない場合には、リンカーエラーメッセージが出力されます。

例 :

```
demo% f95 -bit move.f    <- -bit オプションは f95 では認識されません。
f95: 警告: ld が起動される場合は、オプション -bit は ld に渡されます。それ以外は無視されます
demo% f95 fast move.f      <- 入力ミス (-fast と入力しようとした)
ld: 重大なエラー: ファイル fast: ファイルをオープンできません: ファイルもディレクトリもありません。
ld: 重大なエラー: ファイル処理エラー。a.out へ書き込まれる出力がありません。
```

最初の例では、-bit は f95 では認識されず、このオプションはリンカー (ld) に渡されます 。ただし、ld では 1 文字のオプションを続けて並べることもできるため、-bit が -b -i -t と解釈されます。b、i、t はいずれも ld の有効なオプションであるからです。これは、ユーザーが意図している場合と、意図していない場合とがあります。

2 つ目の例では、f77/f95 の共通のオプションとして -fast を指定しようとしていますが、先頭のハイフンが抜けています。この場合も、コンパイラは引数をリンカーに渡し、リンカーはこれをファイル名と解釈します。

以上の例から、コンパイラコマンドを指定する場合には、十分な注意が必要であることがわかります。

モジュール (Fortran 95)

f95 は、ソースファイル中にある各 MODULE 宣言に対して、それぞれモジュールインタフェースファイルを自動的に作成し、USE 文で引用されるモジュールを検索します。見つかったモジュール (MODULE *module_name*) ごとに、コンパイラは、対応するファイルmodule_name.*mod* を現在のディレクトリ内に生成します。たとえば、ファイル mysrc.f95 中にある MODULE list 単位のモジュール情報ファイル list.mod は f95 によって生成されます。

コンパイラは、現在のディレクトリを検索して、USE 文の中で引用されている各モジュールファイルを検索します。モジュールファイルは、USE 文で MODULE を引用しているソースファイルをコンパイルする前にコンパイルします。-M のコマンド行オプションを使用すると、検索パスにディレクトリを追加できます。ただし、個々の .mod ファイルをコマンド行で直接指定することはできません。

f95 コンパイラも、 MODULE 文を含むすべてのソースファイルに対し、オブジェクトファイル *filename*.o を作成します。これらのオブジェクトファイルは、実行可能ファイル作成時のリンクに含める必要があります。187 ページ を参照してください。

指令

Fortran の注釈の書式であるソースコード指令を使用して、特殊な最適化または並列化の選択に関する情報をコンパイラに渡すことができます。コンパイラ指令は、プラグマとも呼ばれます。コンパイラは、一連の一般指令および並列化指令を認識します。Fortran 95 も OpenMP 共有メモリマルチプロセッシング指令を処理します。

f95 に固有の指令については付録 C で説明します。

f77 および f95 が認識する全指令の一覧は、付録 E に記載されています。

注 - 指令は Fortran 規格には含まれていません。

一般的な指令

一般的な Sun の Fortran 指令は次のような書式で使用します。

```
C$PRAGMA keyword ( a [ , a ] ... ) [ , keyword ( a [ , a ] ... ) ] ,...
C$PRAGMA SUN keyword ( a [ , a ] ... ) [ , keyword ( a [ , a ] ... ) ] ,...
C$PRAGMA SPARC keyword ( a [ , a ] ... ) [ , keyword ( a [ , a ] ... ) ] ,...
```

変数 keyword は特定の指令を表します。追加の引数やサブオプションも指定できます。指令によっては、上記に示す `SUN` や `SPARC` といった追加のキーワードを指定しなければなりません。

一般的な指令の構文は、次のとおりです。

- 1 カラム目は、注釈指示子の文字 `c`、`C`、`!`、`*` などです。
- `f95` の自由形式の ! は、認識される唯一の注釈指示子 (`!$PRAGMA`) です。本章に示す例では、固定形式を想定しています。
- 次の 7 文字は空白文字を入れず `$PRAGMA` とします。大文字でも小文字でもかまいません。
- `f77` の場合、! という注釈指示子文字を使用する指令は、行のどのカラムにも記述できます。`f95` の場合、このことは自由形式のソースプログラムでしか行えません。

制限事項は、次のとおりです。

- 最初の 8 文字の後では、空白は無視され、大文字と小文字は区別されません。
- 指令は注釈なので行をまたがって継続することはできません。ただし、1 行で完結している `C$PRAGMA` 行を複数使うことができます。
- 注釈が上記の構文条件を満たしていると、コンパイラが認識できる指令が 1 つまたは複数含まれていることになります。上記の構文条件を満たしていない場合は、警告メッセージが出力されます。

- C プリプロセッサの cpp は注釈行または指令行の中でマクロシンボル定義を展開します。Fortran プリプロセッサの fpp は注釈行の中でマクロの展開は行いませんが、正当な f77 と f95 の指令は認識し、指令キーワード外の置換を制限します。ただし、キーワード SUN が必要な指令には注意してください。cpp は小文字の sun を事前定義した値で置き換えます。また、cpp マクロ SUN を定義すると、SUN 指令キーワードが干渉されます。一般的な規則では、次のようにソースが cpp または fpp で処理される場合、プラグマは大文字と小文字を混在させて指定します。

```
C$PRAGMA Sun UNROLL=3
```

Fortran のコンパイラは、次の一般的な指令を認識します。

表 2-2　一般的な Fortran 指令

C 指令	C$PRAGMA C(list) 外部関数名のリストを C 言語ルーチンとして宣言します。
UNROLL 指令	C$PRAGMA SUN UNROLL=n 次のループが n 回まで展開できることをコンパイラに指示します。
WEAK 指示	C$PRAGMA WEAK(name[=name2]) name を弱いシンボルまたは name 2 の別名として宣言します。
OPT 指令	C$PRAGMA SUN OPT=n 副プログラムの最適化レベルに n を設定します。
PIPELOOP 指令	C$PRAGMA SUN PIPELOOP=n 反復間 n の次のループの依存関係を宣言する .
NOMEMDEP 指令	C$PRAGMA SUN NOMEMDEP 次のループにメモリの依存関係が存在しないことを宣言します。
PREFETCH 指令	C$PRAGMA SPARC_PREFETCH_READ_ONCE(name) C$PRAGMA SPARC_PREFETCH_READ_MANY(name) C$PRAGMA SPARC_PREFETCH_WRITE_ONCE(name) C$PRAGMA SPARC_PREFETCH_WRITE_ONCE(name) 名前の参照のために、先読み命令を生成するようにコンパイラに要求します。 (Requires -xprefetch option.)

C 指令

C() 指令は、その引数が外部関数であることを指定します。これは、EXTERNAL 宣言と同義です。ただし、通常の外部名とは違って、Fortran コンパイラでは、これらの引数名に下線が付けられません。詳細は、『Fortran プログラミングガイド』の「C-Fortran インタフェース」の章を参照してください。

特殊な関数の C() 指令は、各副プログラム中にある、その関数への最初の引用よりも前に現われなければなりません。

例：C で ABC と XYZ をコンパイルします。

```
      EXTERNAL ABC, XYZ
C$PRAGMA C(ABC, XYZ)
```

UNROLL 指令

UNROLL 指令では、C$PRAGMA の後に SUN と指定する必要があります。

C$PRAGMA SUN UNROLL=*n* 指令は、次のループを *n* 回展開するようにオプティマイザに指示します。(コンパイラは、解析の結果、ループの展開が適切であると判断した場合のみ展開します)。

n には正の整数を指定します。以下の選択肢があります。

- *n*=1 の場合、オプティマイザは、どのループも展開できません。
- *n*>1 の場合、オプティマイザは、ループを *n* 回展開できます。

実際に展開されたループがあると、実行可能ファイルのサイズが大きくなります。パフォーマンスと最適化についての詳細は、『Fortran プログラミングガイド』を参照してください。

例 : ループを 2 回展開するときは、次のように指定します。

```
C$PRAGMA SUN UNROLL=2
```

WEAK 指令

WEAK 指令は、以前に定義されているよりも低い優先順位で同じシンボルを定義します。この指令は主に、ライブラリを作成する場合にソースファイル中で使用されます。この場合、優先順位が低いシンボルが解決されなくても、リンカーからはエラーメッセージは出力されません。

```
C$PRAGMA WEAK (name1 [=name2])
```

WEAK (name 1) によって、name1 が優先順位が低いシンボルとして定義されます。この場合、リンカーは name1 の定義が見つけられなくてもエラーメッセージを出力しません。

WEAK (name1=name2) によって、name1 が弱いシンボルとして、および name2 の別名として定義されます。

プログラムから呼び出された name1 が定義されていない場合、リンカーはライブラリの定義を使用します。ただし、プログラムで name1 の定義が行われている場合は、そのプログラムの定義が使用され、ライブラリ中にある name1 の優先順位が低い大域的な定義は使用されません。プログラムから name2 が直接呼び出されると、ライブラリの定義が使用されます。name2 の定義が重複すると、エラーが発生します。詳細は、Solaris の『リンカーとライブラリ』を参照してください。

OPT 指令

OPT 指令では、C$PRAGMA の後に SUN と指定する必要があります。

OPT 指令は副プログラムの最適化レベルを設定し、コンパイルコマンド行に指定されているレベルは上書きされます。指令は副プログラムの直前に指定する必要があり、その副プログラムだけに適用されます。次に例を示します。

```
C$PRAGMA SUN OPT=2
        SUBROUTINE smart(a,b,c,d,e)
        ...etc
```

-04 を指定する f77 コマンドでコンパイルする場合、指令はこのレベルを上書きして -02 でサブルーチンをコンパイルします。このルーチンの後に別の指令がない限り、次の副プログラムは -04 でコンパイルされます。

ルーチンを -xmaxopt[=*n*]オプションでコンパイルして、指令が認識されるようにする必要があります。このコンパイラオプションは PRAGMA OPT 指令の最適化の最大値を指定します。PRAGMA OPT に指定した最適化レベルが -xmaxopt レベルよりも大きいと、 -xmaxopt レベルが使用されます。

NOMEMDEP 指令

NOMEMDEP 指令では、C$PRAGMA の後に SUN と指定する必要があります。

この指令は DO ループの直前に指定する必要があります。オプティマイザに対し、ループにメモリの依存関係が存在しないことを宣言し、ループの並列化を禁止します。-parallel または -explicitpar オプションが必要です。

PIPELOOP=*n* 指令

PIPELOOP=*n* 指令では、C$PRAGMA の後に SUN と指定する必要があります。

指令は DO ループの直前に指定する必要があります。*n* には正の整定数かゼロを指定し、ループ間の依存関係をオプティマイザに指示します。ゼロの値は反復間の依存関係（ループの実行）がないことを示し、オプティマイザで自由にパイプラインできます。正の値の *n* はループの I 番目の反復が (I-*n*) 番目の反復に依存していることを意味し、一度に *n* 反復だけパイプラインできます。

```
C    We know that the value of K is such that there can be no
C    cross-iteration dependencies (E.g. K>N)
C$PRAGMA SUN PIPELOOP=0
      DO I=1,N
       A(I)=A(I+K) + D(I)
       B(I)=B(I) + A(I)
      END DO
```

最適化についての詳細は、『Fortran プログラミングガイド』を参照してください。

PERFETCH 指令

-xprefetch オプションフラグを使用すると、コンパイラに指示した一連の PREFETCH 指令 は、指定のデータ要素について先読み命令を生成することができます。先読み命令は、UltraSPARC プラットフォームでだけ使用できます。

```
C$PRAGMA SPARC_PREFETCH_READ_ONCE(name)
C$PRAGMA SPARC_PREFETCH_READ_MANY(name)
C$PRAGMA SPARC_PREFETCH_WRITE_ONCE(name)
C$PRAGMA SPARC_PREFETCH_WRITE_MANY(name)
```

先読み命令の詳細は、『C ユーザーズガイド』または『SPARC Architecture Manual, Version 9』も参照してください。

並列化指令

並列化指令は、コンパイラに対して、指令の後に続く DO ループまたはコードの範囲を並列化するように明示的に指示します。一般的な指令とは、構文が異なります。並列化指令は、-parallel または -explicitpar コンパイルオプションが指定されている場合のみ、認識されます。Fortran の並列化指令については『Fortran プログラミングガイド』を参照してください。

注 - Fortran コンパイラの並列化機能には、Forte for High Performance Computing (HPC) のライセンスが必要です。

Fortran のコンパイラは、Sun、Cray、OpenMP という三つの形式の並列化指令をサポートします。

デフォルトは、Sun 形式の並列化指令です (-mp=sun というコンパイラオプションを指定すると、明示的に選択されます)。Sun 指令には、$PAR という指令センチネルが付きます。

Cray 形式の並列化指令を使用するには、-mp=cray というコンパイラオプションを指定します。これには、MIC$ というセンチネルが付きます。Sun と Cray では、同じ指令でも解釈の仕方が異なります。詳細は、『Fortran プログラミングガイド』の並列化の章を参照してください。

Fortran 95 は、次の章に示すような OpenMP 並列化指令も受け付けます。

SUN/Cray の並列化指令の構文は以下のとおりです。

- 最初の文字は、1 カラム目になければなりません。
- 最初の文字は、c、C、*、! のいずれかです。
- 次の 4 文字は、空白を入れず $PAR (サン形式) または MIC$ (Cary 形式) とします。大文字でも小文字でもかまいません。
- その後に、指令のキーワードと修飾子を空白で区切って続けます。明示的な並列化指令のキーワードは次のとおりです。

```
TASKCOMMON、DOALL、DOSERIAL、DOSERIAL*
```

並列化指令では、キーワードの後にオプション修飾子を指定します。

例 : 共有変数でループを指定 :

```
C$PAR DOALL SHARED(yvalue)     Sun 形式
CMIC$ DOALL SHARED(yvalue)     Cray 形式
```

並列化およびこれらの指令の概要については付録 E を、詳細は『Fortran プログラミングガイド』をそれぞれ参照してください。

OpenMP 指令

Sun WorkShop 6 の Fortran 95 コンパイラは、OpenMP アーキテクチャレビューボードで指定したように、OpenMP Fortran 共有メモリマルチプロセッシング API を認識します。詳細は、OpenMPの web サイト http://www.openmp.org を参照してください。

OpenMP 指令を使用可能にするには、コマンド行オプション -openmp を指定してコンパイルしなければなりません (-openmp は、OpenMPに必要なコンパイラオプションを呼び出すマクロフラグです。31 ページを参照してください)。

OpenMP 指令の概要は、付録 E に記載されています。

OpenMP 指令は、Sun または Cray 形式のどちらか一方の並列化指令と併用できます。ただし、それは、これらの指令が互いに入れ子になっていない場合に限ります。Sun または Cray の指令と OpenMP を使用可能にするには、-mp=openmp、sun または -mp=openmp、cray と指定します。コンマの後には空白文字を入れません。31 ページを参照してください。

f95: ライブラリインタフェースと system.inc

Fortran 95 コンパイラは、ほとんどの非組み込みライブラリルーチンのインタフェースを定義するインクルードファイル system.inc を提供します。特にデフォルトのデータ型が -xtypemap で変更される場合は、呼び出す関数とその引数が正しく入力されていることを確実にするため、このインクルードファイルを宣言します。

たとえば、次のプログラムでは、関数 getpid() が明示的に入力されていないため、算術的な例外が発生します。

```
        integer(4) mypid
        mypid = getpid()
        print *, mypid
```

getpid() ルーチンは整数値を返しますが、関数の明示的な入力が宣言されていない場合、コンパイラは実数値が返されたものとみなします。この数値が整数に変換されると、浮動小数点エラーが生じる可能性が高まります。

このような場合、getpid() と自分が呼び出す関数を明示的に入力します。

```
        integer(4) mypid, getpid
        mypid = getpid()
        print *, mypid
```

このような問題は、-Xlist (大域的なプログラム検査) オプションで診断できます。Fortran 95 インクルードファイル 'system.inc' は、これらのルーチンの明示的なインタフェース定義を提供します。

```
        include 'system.inc'
        integer(4) mypid
        mypid = getpid()
        print *, mypid
```

Fortran ライブラリのルーチンを呼び出すプログラムに system.inc を含めると、インタフェースが自動的に定義され、コンパイラによる型の不一致の診断がサポートされます (詳細については、『Fortran ライブラリ・リファレンス』を参照してください)。

コンパイラの利用方法

Sun Fortran コンパイラを効果的に利用するための方法をいくつか紹介します。

ハードウェアプラットフォームの特定

コンパイラフラグの中には、特定のハードウェアプラットフォームのオプションセットに合わせてコードを生成できるものもあります。fpversion ユーティリティを実行すると、ネイティブプロセッサのハードウェアプラットフォームの仕様を表示することができます。

```
demo% fpversion
A SPARC-based CPU is available.
 CPU's clock rate appears to be approximately 467.1 MHz.
 Kernel says CPU's clock rate is 480.0 MHz.
 Kernel says main memory's clock rate is 120.0 MHz.

 Sun-4 floating-point controller version 0 found.
 An UltraSPARC chip is available.
 FPU's frequency appears to be approximately 492.7 MHz.

 Use "-xtarget=ultra2i -xcache=16/32/1:2048/64/1"
code-generation option.

 Hostid = hardware_host_id.
```

表示される値は、fpversion が呼び出されたときのシステムの負荷によって異なります。

詳細については fpversion(1) および『数値計算ガイド』を参照してください。

環境変数の使用

FFLAGS または OPTIONS 環境変数を設定して、オプションを指定することができます。

コマンド行で FFLAGS または OPTIONS のいずれかを明示的に指定します。make ファイルの暗黙のコンパイル規則を使用している場合は、make プログラムによって FFLAGS が自動的に使用されます。

例 : FLAGS を設定します (C シェル)。

```
demo% setenv FFLAGS '-fast -Xlist'
```

例 : FFLAGS を明示的に使用します。

```
demo% f95 $FFLAGS any.f
```

make を使用するときに、FFLAGS 変数が上記のように設定されており、makefile の暗黙のコンパイル規則が適用される場合 (すなわち明示的なコンパイラ・コマンド行がない場合) に make を実行すると、次のコンパイルを実行した場合と同じ意味になります。

```
f77 -silent -fast -Xlist files...
```

make はサンのすべてのコンパイラで使用できる協力なプログラム開発ツールです。make(1) マニュアルページおよび『Fortran プログラミングガイド』の第 3 章「プログラム開発」を参照してください。

注 - make が仮定するデフォルトの暗黙的規則では、.f95 および .mod (Fortran 95 のモジュールファイル) という拡張子付きのファイルを認識できません。詳細は、『Fortran プログラミングガイド』および Fortran 95 の README ファイルを参照してください。

メモリーサイズ

コンパイル処理は大量のメモリーを使用することがあります。必要なメモリーのサイズは、選択した最適化レベル、およびコンパイルするファイルのサイズや複雑さに依存します。SPARC プラットフォームでは、最適化でメモリーが不足した場合は、その時点の手続きを低いレベルで最適化し直し、後続のルーチンはコマンド行の -O*n* オプションで指定されていた本来のレベルで最適化を再開します。

ワークステーションには最低 24M バイトのメモリーが実装されている必要があります。推奨値は 32M バイトです。メモリーの使用量は、手続きのサイズ、最適化レベル、仮想メモリーの制限、ディスクのスワップファイルのサイズ、その他さまざまな要素によって異なります。

多数のルーチンを含む単一のソースファイルをコンパイルすると、メモリーやスワップ領域が不足することがあります。

コンパイラのメモリーが不足する場合は、最適化レベルを下げてください。または fsplit(1) を使用して、複数のルーチンが含まれているソースファイルを、1 ルーチンが 1 ファイルに対応するようにいくつかのファイルに分割してください。

スワップ領域の制限

SunOS™ のオペレーティングシステムコマンド、swap -s により、利用可能なスワップ領域が表示されます。詳細は、swap(1M) を参照してください。

例 : swap コマンドを使用します。

```
demo% swap -s
total: 40236k bytes allocated + 7280k reserved = 47516k used,
1058708k available
```

実際の実メモリーの容量は、次のコマンドで確認できます。

```
demo% /usr/sbin/dmesg | grep mem
mem = 655360K (0x28000000)
avail mem = 602476544
```

スワップ領域の拡大

スワップ領域のサイズを拡大するには、mkfile(1M) と swap(1M) を使用します。この操作は、スーパーユーザーだけが実行できます。mkfile によって特定のサイズのファイルを作成し、swap -a によってそのファイルをシステムのスワップ領域に追加します。

```
demo# mkfile -v 90m /home/swapfile
/home/swapfile 94317840 bytes
demo# /usr/sbin/swap -a  /home/swapfile
```

仮想メモリーの制限

最適化レベル `-O3` 以上のレベルで大規模なルーチン (1 つの手続きが数千行ものコードで構成されるルーチン) をコンパイルすると、メモリーがさらに必要になる場合があり、コンパイル時間のパフォーマンスが低下することもあります。この問題には、1 つプロセスで利用できる仮想メモリーの量を制限することによって対処することができます。

`sh` では、`ulimit` コマンドを使用します (`sh`(1) を参照してください)。

例 : 仮想メモリーを 16M バイト に制限します。

```
demo$ ulimit -d 16000
```

`csh` では、`limit` コマンドを使用します (`csh`(1) を参照してください)。

例 : 仮想メモリーを 16M バイト に制限します。

```
demo% limit datasize 16M
```

いずれの場合も、オプティマイザは 16M バイト のデータ領域で最適化を再実行します。

この制限はマシンで利用可能なスワップ領域の総量を超えることはできないので、実際は、大規模なコンパイルの進行中であってもマシンを普通に使用できる程度の小さい値を指定してください。コンパイル処理でスワップ領域の半分以上が使用されることのないように注意してください。

コンパイル処理でスワップ領域の半分以上が使用されることのないように注意してください。

例 : 32M バイト のスワップ領域のあるマシンでは、次のコマンドを使用します。

`sh` の場合:

```
demo$ ulimit -d 1600
```

csh の場合:

```
demo% limit datasize 16M
```

最適な設定は、最適化のレベルや、利用可能な実メモリーと仮想メモリーの量によって異なります。

64 ビットの Solaris 環境では、アプリケーションデータセグメントのサイズに対する弱い制限値は 2GB です。データ領域を追加割り当てする必要がある場合は、シェルの limit または ulimit コマンドを使用して制限値を削除します。

csh の場合 :

```
demo% limit datasize unlimited
```

sh、ksh の場合 :

```
demo% unlimit -d unlimited
```

詳細は、『Solaris 64 ビット開発ガイド』を参照してください。

第3章

コンパイラオプション

この章では `f77` および `f95` の各コンパイラのコマンド行オプションについて説明します。

オプションによっては、両方のコンパイラ (`f77` または `f95`) で使用できないものもあります。リファレンスセクションで使用できるオプションを確認ください。

コマンド構文

コンパイラのコマンドの構文は次のとおりです。

```
f77 [options] list_of_files additional_options
f95 [options] list_of_files additional_options
```

角括弧 (`[]`) の中の項目は省略可能なパラメータを示します。角括弧自体はコマンドの一部ではありません。[*options*] には、先頭にハイフン (-) を付けたオプションキーワードを指定します。オプションによっては、リスト中の次の項目を引数として取るものがあります。list_of_files には、ソース、オブジェクトまたはライブラリのファイル名を空白で区切って複数指定することができます。また、オプションによっては、ソースファイルリストよりも後に続けて指定しなければならないものがあります (たとえば、`-B`、`-l` および `-L`)。これらのオプションには、そのオプション用のファイルリストを指定してもかまいません。

オプションの構文

オプションの一般的な書式を以下に示します。

表 3-1　オプションの構文

構文の形式	例
-flag	`-g`
-flagvalue	`-Dnostep`
-flag=value	`-xunroll=4`
-flag value	`-o outfile`

次の表記規則に従って、オプションを説明しています。

表 3-2　オプションの表記規則

表記	意味	例 : テキスト / インスタンス
`[]`	角括弧は、省略可能な引数を表す。	`-O[n]` `-O4,-O`
`{}`	中括弧は、必須の引数を表す。	`-d{y|n}` `-dy`
`|`	縦棒記号は、いずれか一方を選択する引数を表す。	`-B{dynamic|static}` `-Bstatic`
`:`	コロンは、コンマと同様に、引数を区切る場合に使用することもある。	`-R`*dir*`[:`*dir*`]` `-R/local/libs:/U/a`
`...`	省略符号は、連続した項目の一部が省略されていることを示す。	`-xinline=`*f1*`[,...`*fn*`]` `-xinline=alpha,dos`

括弧、縦棒、省略符号は、オプションを記述するために使用している記号で、オプション自体の一部ではありません。

オプションの一般的な規則を以下に示します。

- `-l`*x* は `lib`*x*`.a` ライブラリにリンクするためのオプションです。 `-l`*x* は必ずファイル名リストの後に指定して、ライブラリの検索順序が保たれるようにしてください。

- 通常、コンパイラオプションは左から右の順序で処理されます。このため、マクロのオプション (別のオプションを含むオプションも) を意図的に上書きすることができます。
  - この規則はリンカーのオプションには適用されません。
  - ただし、オプションが同じコマンド行で繰り返される場合は、`-I`、`-L`、`-R` などは以前に指定した値を上書きせずに、順番に処理します。

ソースファイル、オブジェクトファイル、およびライブラリは、コマンド行に現れる順にコンパイルとリンクが実行されます。

オプションのまとめ

この節では、各コンパイラオプションを機能別に分類し、概略を説明しています。詳細は、表 3-3 の詳細欄に示すページを参照してください。

次の表に、`f77` および `f95` のコンパイラオプションを機能別にまとめます。この表には、廃止されたり使用されなくなったりしたオプションフラグは含まれていません。フラグによっては、複数の使用目的があるため、複数の箇所に記載されているものがあります。

表 3-3　機能別コンパイラオプション

機能	オプションフラグ
コンパイルモード	
コンパイルのみ。実行可能ファイルを生成しない。	`-c`
ドライバが作成するコマンドを表示するが、コンパイルは行わない。	`-dryrun`
書き込むオブジェクト、ライブラリ、実行可能ファイルの名前を指定する。	`-o` *filename*
コンパイルし、アセンブリコードだけを生成する。	`-S`
実行可能プログラムからシンボルテーブルを除外する。	`-s`
エラーメッセージ以外のコンパイラメッセージを出力しない。	`-silent`
一時ファイルのディレクトリへのパスを定義する。	`-temp=`*directory*
各コンパイルフェーズの経過時間を示す。	`-time`

表 3-3　機能別コンパイラオプション

機能	オプションフラグ
コンパイラおよびそのフェーズのバージョン番号を示す。	`-V`
冗長メッセージ	`-v`
コンパイルされるコード	
外部名の末尾に下線を追加/抑制する。	`-ext_names=`*x*
インライン化するユーザー関数を指定する。	`-inline=`*list*
コンパイル位置独立コードを指定する。	`-KPIC/-kpic`
特定の数学ライブラリルーチンをインライン化する。	`-libmil`
STOP で整数のステータス値をシェルに返す。	`-stop_status`*[=yn]*
コードアドレス空間を指定する。	`-xcode=`*x*
UltraSPARC の先読み命令を有効にする。	`-xprefetch`*[=x]*
オプションのレジスタを指定する。	`-xregs=`*x*
デフォルトのデータマッピングを指定する。	`-xtypemap=`*x*
データの境界整列	
COMMON ブロック内のデータの境界整列を指定する。	`-aligncommon[=`*n*`]`
強制的に COMMON ブロックデータの境界整列を行い、マルチワードのロード/ストアを可能にする。	`-dalign`
全データを 8 バイト境界に強制的に整列させる。	`-dbl_align_all`
COMMON ブロックデータを8 バイト境界に整列させる。	`-f`
メモリの境界整列と振る舞いを指定する。	`-xmemalign[=`*ab*`]`
デバック	
実行時に添字の範囲検査を有効にする。	`-C`
デバック用にコンパイルする。	`-g`
Sun WorkShop ソースブラウザを使用したブラウズのためにコンパイルする。	`-sb, -sbfast`
未宣言変数の検査を行う。	`-u`
Sun WorkShop Performance アナライザ用にコンパイル。	`-xF`
ソースのリスト処理を指定。	`-Xlist`*x*
オブジェクトファイルを使用せずにデバック機能を有効にする。	`-xs`

表 3-3　機能別コンパイラオプション

機能	オプションフラグ
診断	
非標準の拡張機能を報告する。	`-ansi`
特定のエラーメッセージの出力を抑制する。	`-erroff=`*list*
エラーメッセージとともにエラータグ名を表示する。	`-errtags`
コンパイラオプションの要約を表示する。	`-flags, -help`
コンパイラおよびその構成要素のバージョン番号を示す。	`-V`
冗長メッセージを表示する。	`-v`
並列化のメッセージを冗長に表示する。	`-vpara`
警告メッセージを表示/抑制する。	`-w`*n*
実行時の `taskcommon` の整合性検査を有効にする。	`-xcommonchk`
コンパイラの README ファイルを表示する。	`-xhelp=readme`
ライセンス	
ライセンスサーバー情報を表示する。	`-xlicinfo`
リンクおよびライブラリ	
動的/静的ライブラリを許す/必要とする。	`-B`*x*
動的/静的なライブラリのみのリンクを許可する。	`-dy, -dn`
動的 (共有オブジェクト) ライブラリを作成する。	`-G`
名前を動的ライブラリの名前を指定する。	`-h`*name*
ディレクトリをライブラリ検索パスに追加する。	`-L`*dir*
`lib`*name*`.a` または `lib`*name*`.so` というライブラリをリンクする。	`-l`*name*
実行時ライブラリの検索パスを実行可能プログラムに組み込む。	`-R`*dir*
インクリメントリンカ (`ild`) を使用不可にする。	`-xildoff`
最適化数学ライブラリをリンクする。	`-xlibmopt`
Sun のパフォーマンスライブラリをリンクする。	`-xlic_lib=sunperf`
リンクエディタのオプションを指定する。	`-z`*x*
再配置のない閉じたライブラリを生成する。	`-ztext`
数値および浮動小数点	
非標準の浮動小数点の設定を使用する。	`-fnonstd`

表 3-3　機能別コンパイラオプション

機能	オプションフラグ
SPARC 非標準浮動小数点を選択する。	`-fns`
入力中に実行時浮動小数点オーバフロー検査を有効にする。	`-fpover`
IEEE 浮動小数点丸めモードを選択する。	`-fpround=`*r*
浮動小数点最適化レベルを選択する。	`-fsimple=`*n*
浮動小数点トラップモードを選択する。	`-ftrap=`*t*
単精度定数を倍精度に変換する。	`-r8const`
区間演算を有効にし、適切な浮動小数点環境を設定する (-xinterval を含む)。	`-xia`*[=e]*
区間演算機能を有効にする。	`-xinterval`*[=e]*
最適化とパフォーマンス	
ループを解析して、データ依存関係を調べる。	`-depend`
オプションの集まりを使用して最適化する。	`-fast`
最適化レベルを指定する。	`-O`*n*
効率的なキャッシュ使用のためにデータレイアウトをパディングする。	`-pad`*[=p]*
局所変数をメモリースタックに割り当てる。	`-stackvar`
ループ展開を有効にする。	`-unroll[=`*m*`]`
ソースファイル間での最適化を有効にする。	`-xcrossfile[=`*n*`]`
内部手続きの最適化パスを呼び出す	`-xipo[=`*n*`]`
`#pragma OPT` に最高レベルの最適化を設定する。	`-xmaxopt[=`*n*`]`
メモリベースのトラップが発生しないであろうと表明する。	`-xsafe=mem`
コードサイズが増加する場合は、最適化を行わない。	`-xspace`
ベクトルライブラリ関数の呼び出しを自動的に作成する。	`-xvector[=`*yn*`]`
並列化 (注：Fortran の並列化機能には、Forte for High Performance Computing のライセンスが必要)	
DO ループの自動並列化を有効にする。	`-autopar`
指令で明示的に指定したループの並列化を有効にする。	`-explicitpar`

表 3-3　機能別コンパイラオプション

機能	オプションフラグ
ループの並列化情報を表示する。	`-loopinfo`
並列化の指令の形式 (Sun、Cray、OpenMT) を指定する。	`-mp=`*v*
マルチスレッド用にプログラミングされたコードをコンパイル	`-mt`
OpenMT API 指令を受け付け、適切な環境 (マクロ) を設定する。	`-openmp`
`-autopar -explicitpar -depend`の組み合わせでループを並列化する。	`-parallel`
自動並列化でループ内の縮約操作を認識する。	`-reduction`
並列化メッセージを冗長表示する。	`-vpara`
ソースコード	
プリプロセッサのシンボルを定義する。	`-D`*name*`[=`*val*`]`
プリプロセッサのシンボルの定義を取り消す。	`-Uname`
拡張 (132 文字) ソース行を受け入れる。	`-e`
.F、.F90 および .F95 のファイルにプリプロセッサを適用するが、コンパイルは行わない。	`-F`
固定書式として入力を受け付ける (`f95`)。	`-fixed`
すべてのソースファイルを fpp プリプロセッサの先行処理を行う。	`-fpp`
自由書式として入力を受け付ける (`f95`)。	`-free`
ファイル検索パスにディレクトリを追加する。	`-I`*di*
モジュール検索パスにディレクトリを追加する。	`-M`*dir*
大文字と小文字を区別する。	`-U`
使用するプリプロセッサ (`cpp` または `fpp`) を選択する。	`-xpp[={fpp|cpp}]`
再帰的な副プログラム呼び出しを許可する。	`-xrecursive`
ターゲットプラットフォーム	
ホストシステム用に最適化する。	`-native`
オプティマイザにターゲットのプラットフォームを指定する。	`-xarch=`*a*

表 3-3　機能別コンパイラオプション

機能	オプションフラグ
オプティマイザにターゲットのキャッシュプロパティを指定する。	`-xcache=`*a*
オプティマイザにターゲットのプロセッサを指定する。	`-xchip=`*a*
オプティマイザにターゲットのプラットフォームを指定する。	`-xtarget=`

頻繁に利用するオプション

Sun Fortran コンパイラには、オプションのコマンド行パラメータによって選択できる機能が数多くあります。以下の表に、頻繁に利用するオプションをまとめてあります。

表 3-4　頻繁に利用するオプション

目的		オプション
デバッグ	大域的にプログラムを検査し、ルーチン間での引数、共通ブロックなどの整合性を調べる。	`-Xlist`
デバッグ	`dbx` および Sun WorkShop デバッグ機能を使用するための追加のシンボルテーブル情報を生成する。	`-g`
パフォーマンス	実行速度の速い実行可能ファイルを作成する。	`-O[n]`
パフォーマンス	事前に定義されている一連のオプションを使用して、ネイティブプラットフォームのコンパイルと実行時間を改善する。	`-fast`
	動的 (`-Bdynamic`) または静的 (`-Bstatic`) ライブラリとのリンク	`-Bx`
コンパイルのみ	リンクを行わず、各ソースファイルごとに `.o` ファイルを作成する。	`-c`
出力ファイル	実行可能な出力ファイルの名前を `a.out` ではなく *nm* に指定する。	`-o` *nm*
ソースコード	固定書式の Fortan 77 のコードを `f95` を使用してコンパイルする。	`-fixed`

下位互換オプションと旧オプション

コンパイラの旧リリース、および Fortran の従来の機能との下位互換のオプションを示します。

表 3-5　下方互換オプション

目的	オプション
デフォルトのデータサイズを倍長にする。-xtypemap を使用する。	-r8 または -dbl
定数の引数への代入を可能にする。	-copyargs
呼び出し引数リストにおいてホレリス定数を文字または型なしとして扱う。	xhasc[{=yes|no}]
外部名に下線を使用しない。	-ext_names=e
非標準の算術演算を使用可能にする。	-fnostd
ホストシステムに合わせて最適化を行う。	-native
並びによる旧式の出力を行う。	-olddo
少なくとも 1 回は DO ループを実行する。	-onetrip
SPARC V7 アーキテクチャ用のコンパイル	-cg89
SPARC V8 アーキテクチャ用のコンパイル	-cg92

これらのオプションフラグは推奨していませんので使用しないでください。

現在使用されていないオプション

f77 および f95 の各コンパイラでサポートされていないオプションを示します。コンパイラコマンドで使用してもエラーの原因はなりません。これらのオプションは無視されるので、何の効果もありません。

表 3-6　現在使用されていないオプション

元の意味	オプション
ループツールによる解析用にコンパイルします。	-Zlp
スレッドアナライザ用にコンパイルします。	-ztha
例外トラップを無効にします (f95)。	-fnonstop

オプションリファレンス

この節では、すべての f77 および f95 コンパイラコマンド行オプションフラグについて示します。このフラグには、さまざまなリスク、制約、警告、相互作用、例、および詳細情報が含まれています。各説明では、オプションのプラットフォームでの利点が示されます。

次の表に、オプションの利用の可能性について示します。

凡例	オプションの利用可能性
f77	該当するプラットフォームの f77 でのみ使用できます。
f95	該当するプラットフォームの f90 でのみ使用できます。
f77/f95	該当するプラットフォームの f77 と f90 の両方で使用できます。

あるオプションが、特定のプラットフォーム上のコンパイラでは使用できない場合でも、メッセージは表示されません。そのようなオプションは無視されるので、影響は何もありません。

このオプションリファレンスでは、それぞれのオプションフラグについて説明します。

-a

tcov を使用する、旧式の基本ブロックごとのプロファイリングを行います。

- **f77/f95**

  tcov を使用する旧式の基本ブロックごとのプロファイルを行います。新しい形式のプロファイルについては、123 ページの「-xprofile=p」の tcov の説明を参照してください。詳細は、tcov(1) のマニュアルページおよび『プログラムのパフォーマンス解析』を参照してください。

  文の基本ブロックがそれぞれ実行される回数を数えるコードを挿入します。これによって、正常終了時に、各 .f ファイルに対して 1 個の .d ファイルを作成する、実行時記録機能が起動されます。.d ファイルには、対応するソースファイルの実行データが蓄積されます。後に、ソースファイルに対して tcov(1) ユーティリティを実行する

と、プログラムの統計情報を生成することができます。tcov によって出力される情報は、ソースファイルごとに .tcov ファイルに書き込まれます。-pg と gprof は -a と tcov を補います。

TCOVDIR 環境変数がコンパイル時に設定されている場合、この変数によって .d ファイルと .tcov ファイルを置くディレクトリが指定されます。この変数が設定されていない場合、.d ファイルは .f ファイルと同じディレクトリに置かれます。

-xprofile=tcov と -a オプションは、1 つの実行可能ファイル内では互換性があります。つまり、-xprofile=tcov を付けてコンパイルしたファイルと、-a を付けてコンパイルしたファイルを 1 つのプログラムにリンクすることができます。ただし、同一のファイルに両方のオプションを付けてコンパイルすることはできません。

コンパイルとリンクを分けて行う場合、-a を付けてコンパイルしたときは、リンクでも必ず -a を付けてください。なお、以前のバージョンでは -a を指定すると -O*n* は無効になりましたが、このバージョンでは -a と -O*n* を同時に指定することができます。

詳細は、『Fortran プログラミングガイド』の第 8 章「パフォーマンスプロファイリング」を参照してください。

-aligncommon [=*n*]

共通ブロック内のデータの境界整列を指定します。

- **f77/f95**

n は、1、2、4、8、または 16 を指定できます。これは、共通ブロック内のデータ要素について最大境界整列サイズ (バイト単位) を指定します。

たとえば、-aligncommon=4 と指定すると、4 バイト以上の自然整列サイズを保つ共通ブロック内の全データ要素が、4 バイト境界に整列します。

このオプションは、指定のサイズより小さい自然整列サイズを保つデータに影響しません。

-aligncommon を指定しないと、共通ブロック内のデータは、多くても 4 バイト境界に整列されます。

どのプラットフォームでも、値を指定せずに -aligncommon だけを指定すると、デフォルトの 1 が仮定され、共通ブロック内の全データは、1 バイト境界に整列されます (要素間のパディングは行われません)。

-aligncommon=16 は、64 ビットが有効ではないプラットフォーム (v9、v9a、または v9b 以外のプラットフォーム) において -aligncommon=8 に戻ります。

-ansi

規格以外の拡張機能を使用すると、警告メッセージを出力します。

- **f77/f95**

ソースコード中で、標準外の Fortran 77 または Fortran 95 の拡張機能を使用すると、警告メッセージが出力されます。

-arg=local

ENTRY 文の実引数を保存します。

- **f77**

このオプションを使用して代替入口のある副プログラムをコンパイルすると、f77 では コピー、ストアを使用して、仮引数と実引数の関連性を保存します。たとえば、次のプログラムを正しく実行するには -arg=local でコンパイルする必要があります。

```
      A = SETUP(ALPHA,BETA,GAMMA)
      ZORK = FXGAMMA(GCONST)
      ...
      FUNCTION SETUP(A1,A2,A3)
      ...
      ENTRY FXGAMMA(F)
      FXGAMMA = F*GAMMA
      ...
      RETURN
      END
```

このオプションを指定しないと、FXGAMMA を介してこのルーチンに入った時点で、SETUP の実引数が正しく引用される保証はありません。-arg=local に依存するコードは標準ではありません。

-autopar

ループの自動並列化を使用可能にします。Fortran 並列化機能には、Forte for HPC のライセンスが必要です。

- **f77/f95**

マルチプロセッサで並列処理の対象に適するループを探し、そのループを並列化します。内部反復データに依存するループを解析し、ループを再構築します。最適化レベルが -O3 以上に設定されていない場合は、自動的に -O3 に設定されます。

パフォーマンスを改善するには、-autopar などの並列化オプションを使用するときに -stackvar オプションも指定してください。

プログラム中に libthread スレッドライブラリへの明示的な呼び出しがある場合は、-autopar は使用しないでください。74ページの-mt の注意事項を参照してください。

-autopar オプションは、シングルプロセッサのシステムには適していません。シングルプロセッサのシステムでこのオプションを付けてコンパイルを行うと、通常は実行速度が低下します。

マルチスレッド環境で並列化プログラムを実行するには、実行前に PARALLEL (または OMP_NUM_THREADS) 環境変数を設定する必要があります。これは、プログラムが作成できる最大スレッド数を実行時システムに指示しています。デフォルトは 1 です。一般的に、ターゲットプラットフォーム上の PARALLEL 変数または OMP_NUM_THREADS 変数には、利用可能なプロセッサ数を設定します。

-autopar を使用してコンパイルとリンクを一度に行う場合、マルチスレッド処理ライブラリとスレッド対応の Fortran 実行時ライブラリが自動的にリンクされます。-autopar を使用してコンパイルとリンクを別々に行う場合は、適切なライブラリにリンクするために、-autopar を使用してリンクを行う必要があります。

-reduction オプションは、-autopar オプションと組み合わせて使用することもできます。その他の並列化オプションとして、-parallel と -explicitpar があります。

並列化についての詳細は、『Fortran プログラミングガイド』を参照してください。

-B{static|dynamic}

動的または静的のどちらかのライブラリリンクを指定します。

- **f77/f95**

  -B と dynamic または static の間に空白文字を入れないでください。-B を省略すると、デフォルトとして -Bdynamic が使用されます。

  - -Bdynamic : 動的リンクを優先する (共有ライブラリのリンク)。

- `-Bstatic` : 静的リンクをする必要がある (共有ライブラリなし)。

以下の点にも注意してください。

- `static` を指定した場合に動的ライブラリしか見つからないと、"library was not found"(ライブラリがありません) という警告メッセージが出力され、ライブラリのリンクは行われません。
- `dynamic` を指定した場合に静的ライブラリしか見つからないと、その静的ライブラリとリンクされます。警告メッセージは表示されません。

次のように、`-Bstatic` と `-Bdynamic` をコマンド行で切り替えて、何回でもライブラリを静的および動的にリンクすることができます。

```
f77 prog.f -Bdynamic -lwells -Bstatic -lsurface
```

これらはローダーおよびリンカーのオプションです。コンパイルコマンド上に `-Bx` オプションを指定してコンパイルとリンクを分けて行う場合は、リンク時にも `-Bx` オプションを指定する必要があります。

`-Bdynamic` と `-dn` の両方をコマンド行に指定することはできません。`-dn` を指定すると動的ライブラリのリンクが行われなくなるからです。

64 ビットの Solaris 環境では、ほとんどのシステムライブラリが共有動的ライブラリとして単独使用できます。これには、`libm.so` と `libc.so` (`libm.a` と `libc.a` は提供されていない) も含まれます。つまり、64 ビットの Solaris 環境で `-Bstatic` と `-dn` を指定するとリンクエラーが発生する場合があります。このような場合、アプリケーションを動的ライブラリとリンクさせる必要があります。

静的ライブラリと動的ライブラリについての詳細は、『Fortran プログラミングガイド』を参照してください。

-C

配列の引用で範囲を超えていないかを検査します。

- **f77/f95**

配列の添字が宣言されている範囲を超えると、セグメンテーションフォルトなどの予期しない結果になる場合があります。`-C` オプションはコンパイル時と実行時に、配列の添字に違反がないかどうかを検査します。

`-C` を指定すると、実行可能ファイルのサイズが大きくなる場合があります。

-C オプションを使用すると、配列の添字違反はエラーとして扱われます。ソースコードのコンパイル中に配列添字の範囲違反が検出されると、コンパイルエラーとして扱われます。

配列添字の違反が実行時だけに検出される場合、コンパイラは実行可能プログラムの中に範囲を検査するコードを生成します。この結果、実行時間が長くなることがあります。したがって、プログラムの開発やデバッグを行なっている間にこのオプションを使用して配列添字の検査を有効にしておき、最後に添字検査なしで最終バージョンの実行可能ファイルを再コンパイルすると効果的です。

-c

コンパイルだけを行い、.o オブジェクトファイルを生成します。リンクは行いません。

- **f77/f95**

リンクを行わずに、ソースファイルごとに .o ファイルを作成します。1 つのソースファイルだけをコンパイルする場合は、-o オプションを使用して、出力先の .o ファイルの名前を指定することができます。

-cg89

一般的な SPARC アーキテクチャ用にコンパイルを行います。(廃止)

- **f77/f95**

このオプションは -xtarget=ss2 と同義で、
-xarch=v7 -xchip=old -xcache=64/32/1 をマクロ化したものです。

-cg92

SPARC V8 アーキテクチャ用にコンパイルを行います。(廃止)

- **f77/f95**

このオプションは -xtarget=ss1000 と同義で、
-xarch=v8 -xchip=super -xcache=16/32/4:1024/32/1 をマクロ化したものです。

-copyargs

定数の引数へ代入を行えるようにします。

- **f77/f95**

  定数である仮引数を副プログラムが変更できるようにします。このオプションは、すでに作成済みのコードのコンパイル時と実行時にエラーが発生しないようにすることだけを目的としています。

  - `-copyargs` を指定しない場合、定数の引数をサブルーチンに渡し、そのサブルーチン内でその定数を変更しようとすると、実行が異常終了します。
  - `-copyargs` を指定した場合、定数の引数をサブルーチンに渡し、そのサブルーチン内でその定数を変更しようとしても、実行が必ずしも異常終了するとは限りません。

  `-copyargs` を指定しないと異常終了してしまうコードは、Fortran 規格に準拠していません。また、このようなコードは予測できない動作をすることがあります。

-D*name* [=*def*]

プリプロセッサのシンボル名を定義します。

- **f77/f95**

  このオプションは `.F`、`.F90` および `.F95` ソースファイルだけに適用します。

  `-D`*name=def* : シンボル *name* を値 *def* を持つものと定義します。

  `-D`*name* : シンボル *name* を 1 と定義します。

  コマンド行では、ソースファイル中に

  `#define` *name*[=*def*]

  と記述されている場合と同じように、このオプションは `name` を定義します。

  `=def` の指定がないと、シンボル 名 `name` は値 1 として定義されます。マクロシンボル *name* はプリプロセッサ `fpp` (または `cpp`。`-xpp` オプションを参照) に渡されて展開されます。

  事前定義されたマクロシンボルの前には二本の下線を付けます。Fortran 構文には事前定義されたマクロの実際の値は使用できません。事前定義されたマクロは、`fpp` か `cpp` のプリプロセッサ指令内だけに使用してください。

  - 製品バージョンは、`__SUNPRO_F77`、`__SUNPRO_F90`、`__SUNPRO_F95` に 16 進数で事前定義されています。たとえば、Sun WorkShop 6 のリリースの場合、`__SUNPRO_F77` は、`0x600` となります。

- 次のマクロは、該当するシステム上でそれぞれ事前定義されています。

  ```
  __sparc, __unix, __sun, __SVR4,
  __SunOS_5_6, __SunOS_5_7, __SunOS_5_8
  ```

  たとえば、SPARC システム上では、`__sparc` 値が定義されています。これらの値は、次のようなプリプロセッサ条件で使用することができます。

  ```
  #ifdef __sparc
  ```

- `sparc`、`unix`、`sun` は、下線なしで事前定義されていますが、将来のリリースで削除される可能性があります。

- SPARC V9システムでは、`__sparcv9` マクロも定義されています。

コンパイラはデフォルトにより、 `fpp` (1) プリプロセッサを使用します。
C プリプロセッサ cpp(1) と同様に、`fpp` はソースコードマクロを展開して、コードを条件付きでコンパイルすることができます。また `cpp` とは異なり、`fpp` は Fortran の構文を認識して、Fortran のプリプロセッサとして優先して使用されます。`-xpp=cpp` フラグを指定すると、`fpp` ではなく `cpp` を強制的に使用することができます。

-dalign

COMMON ブロックデータの整列を行い、高速なマルチワードのロード/ストアを生成します。

- **f77/f95**

このフラグを使用すると、COMMON ブロック (および EQUIVALENCE クラス) のデータレイアウトが変更されるため、コンパイラは、そのデータに対する高速なマルチワードのロード/ストアを生成できるようになります。

データレイアウトは、`-f` フラグを指定した時と同じようになります。COMMON ブロックと EQUIVALENCE クラスの倍精度および 4 倍精度のデータが、メモリー内で「自然に」境界整列されます。これは、8 バイトの境界整列になります。なお、64 ビット環境で `-xarch=v9` または `-xarch=v9a` を指定してコンパイルを行うと、4 倍精度のデータは 16 バイトに境界整列されます。COMMON ブロック内のデータのデフォルト整列は、4 バイトの境界整列です。コンパイラも自然整列を前提とするため、高速なマルチワードのロード/ストアを生成してデータを参照できるようになります。

注 – -dalign を使用すると、データの境界整列が標準に合わなくなることがあります。これが原因で、EQUIVALENCE や COMMON の変数に問題が生じることがあります。さらに、-dalign が必要な場合に移植性のないプログラムになります。

-dalign は、-xmemalign=8s -aligncommon=16 と等価のマクロです。41 ページの -aligncommon および 118 ページの「-xmemalign[=<a><b>]」を参照してください。

-dbl （f77）と -dalign の両方を使用すると、デフォルトの INTEGER 変数が 8 バイトに境界整列され 64 ビットになります。ただし、この場合は -dbl よりも -xtypemap を推奨します。

```
-xtypemap=real:64,double:128,integer:64
```

ある 1 つの副プログラムに -dalign を付けてコンパイルした場合は、プログラムのすべての副プログラムに -dalign を付けてコンパイルしてください。このオプションは -fast オプションに含まれます。

-db

オプションの CIF ファイルを生成します。

- **f95**

拡張子 .T を付けて、オプションのコンパイラ情報ファイル (CIF) を生成します。このファイルは、Sun WorkShop ソースブラウザ で必要になることがあります。

-dbl

REAL、INTEGER、DOUBLE、COMPLEX のデフォルトサイズを倍長にします。

- **f77**

注 – このオプションと -r8 は将来のリリースで削除される可能性があるので、一般的な -xtypemapオプションを使用してください。

-dbl を使用すると、明示的にバイトサイズを指定しないで宣言されている、REAL、INTEGER、DOUBLE、および COMPLEX の変数に対するデフォルトのバイトサイズが拡張されます。

表 3-7　デフォルトのデータサイズ (バイト)と -dbl オプション

	-dbl オプションなし	-dbl オプションあり
データ型	デフォルト	SPARC
INTEGER	4	8
REAL	4	8
DOUBLE	8	16

このオプションは、変数、パラメータ、定数、および関数に適用されます。

また、LOGICAL は INTEGER、COMPLEX は 2 つの REAL、DOUBLE COMPLEX は 2 つの DOUBLE として扱われます。

-dbl と -r8 を比較すると、次のようになります。また、-dbl および -r8 は、より汎用的な -xtypemap= オプションで表すことができます。

-dbl: -xtypemap=real:64,double:128,integer:64 と同義

-r8: -xtypemap=real:64,double:128,integer:mixed と同義

これらのオプションによってデフォルトの DOUBLE PRECISION データが QUAD PRECISION (128 ビット) になりますが、パフォーマンスは低下する可能性があります。この場合、-dbl よりも -xtypemap=real:64, double:64, integer:64 を使用した方が適切です。

浮動小数点のすべてのデータ型に対し、-dbl は -r8 と同様に作用します。-r8 と -dbl を同時に指定すると、-dbl だけを指定したときと同じ結果となります。

- INTEGER と LOGICAL では、-dbl と -r8 の機能は異なります。
  - -dbl を指定すると、8 バイトを割り当て、8 バイトで演算します。
  - -r8 を指定すると、8 バイトを割り当てますが、4 バイトで演算します。

通常、1 つの副プログラムを -dbl 付きでコンパイルする場合は、そのプログラムのすべての副プログラムも -dbl 付きでコンパイルしてください。これは特に、各ファイル間での入出力形式が統一されていないプログラムでは重要になります。またこの

オプションを使用すると、関数名にデータのサイズを明示的に指定しない限り、ライブラリ関数の呼び出しも含め、関数のデフォルトのデータサイズが変更される点に注意する必要があります。

-dbl_align_all={yes|no}

8 バイトの境界上でデータを強制的に整列します。

- **f77/f95**

値には yes または no のいずれかを指定します。値が yes の場合、変数はすべて 8 バイトの境界上で整列されます。デフォルトは、-dbl_align_all=no です。

64 ビット環境で -xarch=v9 または -xarch=v9a を使用してコンパイルした場合、4 倍精度のデータは 16 バイトに境界整列されます。

このフラグによって、COMMON ブロック内のデータレイアウトやユーザー定義の構造体が変更されることはありません。

SPARC で、-dalign と併用してマルチワードのロード / ストアで追加した効率を有効にします。

使用した場合、すべてのルーチンをこのフラグでコンパイルする必要があります。

-depend

ループを解析してデータ依存を調べます。

- **f77/f95**

データ依存についてループを解析し、ループを再構築します。このオプションを指定すると、最適化レベルが指定されていない場合または O3 以下の場合は、自動的に最適化レベルが O3 に設定されます。-depend は、-fasti、-autopar および -parallel オプションでも行われます。(詳細は、『Fortran プログラミングガイド』を参照してください)。

-dn

動的ライブラリを使用不可能にします。51 ページの -d{y|n} を参照してください。

- **f77/f95**

-dryrun

f77 か f95 のコマンド行ドライバによって実行されるコマンド群を表示しますが、コンパイルは行いません。

- **f77/f95**

デバッグ時に便利です。このオプションにより、コンパイルを実行するために呼び出されるコマンドとサブオプションが表示されます。

-d{y|n}

実行可能形式全体を生成する時のライブラリのリンク形式 (動的または静的) 。

- **f77/f95**

  - -dy : 動的/共有ライブラリを使用できます。
  - -dn : 動的/共有ライブラリを使用できません。

このオプションを指定しない場合は、デフォルトとして -dy が使用されます。

-Bx とは異なり、このオプションは実行可能ファイル全体に適用され、コマンド行で 1 度だけ使用します。

-dy|-dn はローダーとリンカーのオプションです。これらのオプションを付けてコンパイルとリンクを別々に行う場合は、リンクでも同じオプションを指定する必要があります。

64 ビットの Solaris 環境で共有動的ライブラリとしてだけ使用できるシステムライブラリはほとんどありません。これには、libm.so と libc.so (libm.a と libc.a は提供されていない) も含まれます。つまり、64 ビットの Solaris 環境で -Bstatic と -dn を指定するとリンクエラーが発生する場合があります。このような場合、アプリケーションを動的ライブラリとリンクさせる必要があります。

-e

拡張された入力ソース行を受け付けます。

- **f77/f95**

132 文字までのソース行を受け付けます。コンパイラは 132 桁目まで各行の右側を空白で埋めます。-e オプションを指定してコンパイルする場合で継続行を使用する時は、文字定数が複数行にまたがらないようにしてください。複数行にまたがると、不必要な空白が定数中に挿入されてしまいます。

-erroff=*taglist*

taglist 名で指定した警告メッセージの表示を抑制します。

- **f77/f95**

各タグ名をコンマで区切った並び (taglist) で指定した警告メッセージの表示を抑制します。taglist に %none と指定した場合は、警告メッセージは抑制されません。taglist が %all と指定した場合は、すべての警告メッセージが抑制されます (-w オプションと同義です)。

例 :

```
f77 -erroff=WDECL_LOCAL_NOTUSED ink.f
```

-errtags オプションを使用して、警告メッセージに関連付けられているタグ名を表示します。(-errtags=no)

-errtags[={yes|no}]

各警告メッセージがメッセージタグ付きで表示されます。

- **f77/f95**

-errtags=yes を付けると、コンパイラの内部エラータグ名が警告メッセージとともに表示されます。デフォルトでは、タグは表示されません。(-errtags=no)

```
demo% f77 -errtags ink.f
ink.f:
 MAIN:
"ink.f", 11 行目: 警告: 局所変数 "i" が使用されていません。
(WDECL_LOCAL_NOTUSED)  <- 警告メッセージのタグ名
```

-errtags だけの場合は -errtags=yes のことです。

-explicitpar

Sun、Cray、および/または OpenMP の指令で明示的に示されたループまたは領域を並列化します。Fortran 並列化機能には、Forte for HPC のライセンスが必要です。

- **f77/f95**

コンパイラは、並列で実行すると、正確な結果が生成されないようなデータの依存が DO ループ中にある場合でも、並列コードを生成します。明示的な並列化を行う場合は、ループを正しく分析してデータ依存の問題がないことを確認してから、並列化の指令を使用してください。

並列化は、マルチプロセッサシステムのみに適用されます。

このオプションを使用すると、Sun、Cray、OpenMP といった明示的並列化指令が有効になります。並列化指令の直前にある DO ループには、スレッド化されたコードが生成されます。

注 – このオプションは、すでに libthread ライブラリへの呼び出しによって、独自にマルチスレッド処理を行なったプログラムをコンパイルする場合には使用できません。

並列化されたプログラムをマルチスレッド環境で実行するには、実行前に PARALLEL (または OMP_NUM_THREADS) 環境変数を設定しておく必要があります。これは、プログラムが作成できる最大スレッド数を実行時システムに指示しています。デフォルトは 1 です。一般的に、ターゲットプラットフォーム上の PARALLEL 変数または OMP_NUM_THREADS 変数には、利用可能なプロセッサ数を設定します。

-explicitpar を使用してコンパイルとリンクを 1 度に行う場合、マルチスレッド処理ライブラリとスレッド対応の Fortran 実行時ライブラリが自動的にリンクされます。-explicitpar を使用してコンパイルとリンクを分けて行う場合は、リンクにも -explicitpar を指定する必要があります。

explicitpar などの並列化オプションを使用する場合にパフォーマンスを改善するには、-stackvar オプションも指定してください。

有効な並列化指令の形式は -mp オプションを使用して選択します。選択できるものは、Sun、Cray または OpenMP です。

最適化レベルが -O3 以上に設定されていない場合は、自動的に -O3 に設定されます。

詳細は、『Fortran プログラミングガイド』の第 10 章「並列化」を参照してください。

-ext_names=*e*

外部名に下線を付けるかどうかを指定します。

- **f77/f95**

e には `plain` または `underscore` のどちらかを指定します。デフォルトは `underscore` です。

`-ext_names=plain`: 下線を付けません。

`-ext_names=underscore`: 下線を付けます。

外部名とは、サブルーチン、関数、ブロックデータ副プログラム、名前付き共通ブロックの名前のことです。このオプションは、ルーチンの入口の名前と、その呼び出しに使用する名前の両方に影響を与えます。このオプションによって、Fortran 77 のルーチンから別の言語のルーチンを呼び出したり、呼び出しを受けたりすることができます。

-F

ソースファイルプリプロセッサを起動します。ただしコンパイルは行いません。

- **f77/f95**

`.F` ファイル (`f95` の場合は `.F95` ファイル) に `fpp` プロセッサを適用し、同じファイル名で拡張子を `.f` (または `.f95`) に変えたファイルに結果を書き込みます。ただし、コンパイルは行いません。

例：

```
f77 -F source.F
```

を実行すると、ソースファイルが `source.f` に書き込まれます。

`fpp` は Fortran のデフォルトのプリプロセッサです。C のプリプロセッサ (`cpp`) は `-xpp=cpp` を指定すると、選択されます。

-f

COMMON ブロックのデータを整列します。

- **f77/f95**

  COMMON ブロックの倍精度および 4 倍精度のデータを境界整列します。

  このフラグを使用すると、COMMON ブロックと EQUIVALENCE クラスのデータレイアウトが変更されます。COMMON ブロックと EQUIVALENCE クラスの倍精度および 4 倍精度のデータが、メモリー内で「自然に」境界整列されます。これは、8 バイトの境界整列になります。なお、64 ビット環境で `-xarch=v9` または `-xarch=v9a` を指定してコンパイルを行うと、4 倍精度のデータは 16 バイトに境界整列されます。COMMON ブロック内のデータのデフォルト整列は、4 バイトの境界整列です。

  `-f` は `-aligncommon=16` と同等です

注 – `-f` フラグを使用すると、データの整列が標準外になることもあるため、EQUIVALENCE や COMMON 内の変数に問題が生じる可能性があります。また、`-f` フラグを必要とするプログラムは、移植性が失われる可能性もあります。

`-f` オプションと共に `-dbl (f77)` オプションを指定すると、すべての 64 ビットの整数データが 8 バイトに境界整列されます。

`-f` オプションを指定してプログラムのいずれかの部分をコンパイルする場合は、そのプログラムに含まれる副プログラムもすべて `-f` オプションを指定してコンパイルする必要があります。

このオプションを単独で使用すると、コンパイラで倍精度および 4 倍精度のデータに対して高速のマルチワードのフェッチ/ストア命令を生成することはできません。`-dalign` がこれを実行し、`-f` も呼び出します。`-f` よりも `-dalign` を使用することをお勧めします。詳細は、47 ページの「`-dalign`」を参照してください。これは、`-dalign` が `-f` も同様に `-fast` オプションの一部であるからです。

`-fast`

実行パフォーマンスを最適化するオプションを選択します。

- **f77/f95**

注 – このオプションは、リリースごと、またはコンパイラごとに変更されることのある他のオプションを選択する機能として定義されています。また、-fast で選択される一部のオプションは、すべてのプラットフォームで利用できません。-fast の拡張機能を確認するには、-v (verbose) フラグを使用してコンパイルしてください。

-fast は、特定のベンチマークアプリケーションのパフォーマンスを引き上げます。しかし、対象アプリケーションに適していないオプションが選択される場合もあります。-fast は、アプリケーションを最高のパフォーマンスでコンパイルするための第一歩として使用してください。ただし、追加調整が必要な場合もあります。-fast を使用したコンパイル時にプログラムが誤動作する場合、-fast を構成する個々のオプションを細かく調べ、自分のプログラムに適したオプションだけを呼び出してください。

また、-fast でコンパイルされたプログラムは、使用するデータセットにより、高いパフォーマンスと正確な結果を実現できないことがあります。浮動小数点演算の特定プロパティに依存するプログラムに対し、-fast を使用したコンパイルは避けてください。

-fast で選択されたオプションの一部は暗黙にリンクするため、コンパイルとリンクを別々に行う場合、-fast でコンパイルしたら、-fast でリンクするようにしてください。

コンパイルとリンクを別々に行う場合は、-fast でコンパイルをしたら、-fast でリンクするようにしてください。

-fast では、次のオプションを選択します。

- -dalign
- -depend
- -fns
- -fsimple=2
- -ftrap=%none (f77) または -ftrap=common (f95)
- -libmil
- -xtarget=native
- -O5
- -xlibmopt
- -pad=local
- -xvector=yes
- -xprefetch=yes

-fast が、選択するオプションの詳細は以下のとおりです。

- -xtarget=native オプション

コンパイルを行うのとは異なるマシンでプログラムを実行する場合は、-fast の後にコード生成オプションを付けます。たとえば、次のとおりです。

```
f77 -fast xtarget=ultra...
```

- -O5 最適化レベルオプション (旧コンパイラリリースでは、-fast に -O3 または -O4 を設定していました)。
- -depend オプションは、データの依存関係と再構成についてループを解析します。
- システムが提供するインライン展開テンプレート用の -libmil オプション
  例外処理を使用する C モジュールでは、-fast の後に -nolibmil（-fast -nolibmil のように）を付けます。
  -libmil を使うと errno の設定や、matherr(3m) の呼び出しによって、例外を検出することができなくなります。
- 積極的に浮動小数点を最適化しようとする -fsimple=2 オプション
  厳密に IEEE 754 標準に準拠する必要がある場合は -fsimple=2 は適していません。62 ページを参照してください (旧リリースでは、-fast は -fsimple=1を設定していました)。
- 共通ブロックの倍および 4 倍データ用に倍長ロードとストアを生成する -dalign オプション。このオプションを使用すると、標準外の形式で共通ブロックの Fortran データの境界整列が行われる可能性があります。
- -xlibmopt オプションは、最適化された数学ライブラリルーチンを選択します。
- -pad=local は、キャッシュの利用率を改善するために、適宜共通ブロック内の変数の間にパディングを挿入します (旧リリースの -fast では -pad=local を設定していませんでした)。
- -xvector=yes は、DO ループ内のある特定の数学ライブラリ呼び出しを、同等のベクトル化されたライブラリルーチンの単一呼び出しに変換します (旧リリースの -fast では、-xvector=yes を設定していませんでした)。
- -fns は、標準外の SPARC 浮動小数点演算の例外ハンドリングおよび段階的アンダーフローを選択します。59 ページを参照してください。
- Fortran 77 のすべてのトラッピングをオフにする -ftrap=%none 。共通の浮動小数点例外のトラッピング -ftrap=common は Fortran 95 で使用されます。

- -xprefetch=yes を指定すると、ハードウェア先読み命令を生成できます。

次に示すように、-fast オプションの後に別のオプションを付けて、このリストに追加したり削除したりできます。

```
f95 -fast -fsimple=1 -xnolibmopt ...
```

この例では、-fast で選択された -fsimple=2 の指定を変更し、-xlibmopt を無効にしています。

-fast は -dalign、-fns、-fsimple=2 を呼び出すため、-fast でコンパイルされたプログラムは、標準外の浮動小数点演算、標準外のデータ整列、および標準外の式評価の配列を招くことがあります。これらの選択オプションは、ほとんどのプログラムに適していない可能性があります。

-fixed

固定書式の Fortran 95 ソース入力ファイルを指定します。

- **f95**

コマンド行に指定するソースファイルはすべて、ファイル名の拡張子に関係なく f77 の固定書式として解釈されます。通常、f95 は .f のファイルだけを固定書式として解釈し、.f95 ファイルを自由書式として解釈します。

-flags

-help と同義です。

- **f77/f95**

-fnonstd

浮動小数点算術ハードウェアの非標準の初期化を行います。

- **f77/f95**

このオプションは、以下のオプションフラグの組み合わせと同義です。

- -fns -ftrap=common

-fnonstd を指定することは、Fortran 主プログラムの先頭で次の 2 つの呼び出しを行うのとほぼ同じです。

```
i=ieee_handler("set", "common", SIGFPE_ABORT)
 call nonstandard_arithmetic()
```

nonstandard_arithmetic() ルーチンは、旧式の abrupt_underflow() ルーチンの代わりです。

-fnonstd オプションを有効にするには、主プログラム全体にこのオプションを付けてコンパイルする必要があります。このオプションを使用すると、浮動小数点ハードウェアが初期化されて以下が実行されます。

- 浮動小数点例外で異常終了 (トラップ) します。
- 速度が改善する場合には、アンダーフローのフラッシュ時に、IEEE 規格の要求しているような非正規数ではなく、ゼロを生成します。

段階的アンダーフローおよび非正規数についての詳細は、-fns を参照してください。

-fnonstd オプションは、浮動小数点オーバーフロー、ゼロによる除算、無効な演算などの例外処理のためのハードウェアトラップを可能にします。これらのハードウェアトラップは SIGFPE シグナルに変換され、プログラムに SIGFPE ハンドラがなければメモリーダンプして終了します。

詳細は、ieee_handler(3m) と ieee_functions(3m) のマニュアルページ、『数値計算ガイド』、『Fortran プログラミングガイド』を参照してください。

-fns[={no|yes}]

SPARC の非標準の浮動小数点モードを選択します。

- **f77/f95**

デフォルトは SPARC の標準の浮動小数点モードです (-fns=no) (『Fortran プログラミングガイド』の第 6 章「浮動小数点演算」を参照してください)。

-fast などの -fns フラグが含まれるマクロフラグの後に =yes または =no オプションを使用して、-fns フラグを切り替えることができます。-fns は、-fns=yes と同じです。

このオプションフラグは、プログラムの実行開始時に、規格外の浮動小数点モードを有効にします。SPARC システムのなかには、規格外の浮動小数点モードを指定すると「段階的アンダーフロー」を無効にするシステムもあります。それが原因で、非正規数ではなくゼロにフラッシュされます。また、非正規オペランドがゼロに置き換えられます。このような SPARC システムでは、ハードウェアの段階的アンダーフローや非正規数がサポートされておらず、このオプションを使用するとプログラムのパフォーマンスを著しく改善することができます。

x が完全なアンダーフローの原因にならないとき、非正規数 *x* とは次の範囲にある数です。

表 3-8　非正規数 REAL と DOUBLE

データ型	範囲
REAL	0.0 < |x| < 1.17549435e-38
DOUBLE PRECISION	0.0 < |x| < 2.22507385072014e-308

非正規数に関する詳細は、『数値計算ガイド』を参照してください。また、このオプションおよび関連するオプションについては『Fortran プログラミングガイド』の第 6 章「浮動小数点演算」を参照してください。(演算によっては、非正規数を表すのに「指数が最小の非正規化数」という用語を使用している場合があります。)

デフォルトでは、浮動小数点は標準の設定に初期化されます。

- IEEE 754 浮動小数点演算は、例外時に異常終了しません。
- アンダーフローは段階的です。

-fns オプションを有効にするには、主プログラム全体にこのオプションを付けてコンパイルする必要があります。

-fpover[={yes|no}]

書式付きの入力で浮動小数点オーバーフローを検出します。

- **f77/f95**

-fpover=yes を指定すると、I/O ライブラリは書式付きの入力で実行時浮動小数点オーバーフローを検出し、エラー条件 (1031) を返します。デフォルトでは、このようなオーバーフローの検出は行いません (-fpover=no)。-fpover は -fpover=yes と同義です。

-fpp

fpp を使用して、入力の前処理を強制的に行います。

- **f95**

ファイルの拡張子に関係なく、f95 コマンド行上にリストされた全入力ソースファイルを fpp プリプロセッサに渡します (通常、fpp によって自動的に先行処理されるファイルは、拡張子が .F、.F90、または F95 のファイルだけです)。

-free

自由書式のソース入力ファイルを指定します。

- **f95**

コマンド行で指定したソースファイルはすべて、ファイル名の拡張子を問わず、f95 自由書式と解釈されます。通常、f95 は .f ファイルを固定書式、また .f95 ファイルを自由書式とみなします。

-fround=*r*

起動時に IEEE の丸めモードを有効にします。

- **f77/f95**

r には nearest、tozero、negative、positive のいずれかを指定します。

デフォルトは -fround=nearest です。

-fround オプションを有効にするには、主プログラム全体にこのオプションを付けてコンパイルする必要があります。

このオプションは、IEEE 754 の丸めモードを以下のように設定します。

- 定数式の評価時にコンパイラによって使用されます。
- 実行時のプログラム初期化中に設定されます。

r が tozero、negative、または positive の場合、プログラムの実行開始時に、オプションは丸め方向を *round-to-zero*、*round-to-negative-infinity*、または *round-to-positive-infinity* にそれぞれ設定します。-fround を指定しない場合は、デフォルトで -fround=nearest が使用され、丸め方向は *round-to-nearest* になります。このオプションの意味は、ieee_flags 関数の場合と同じです (『Fortran プログラミングガイド』の第 6 章「浮動小数点演算」を参照してください)。

-fsimple[=*n*]

浮動小数点の最適化の設定を選択します。

- **f77/f95**

  オプティマイザが浮動小数点演算に関する前提を単純化できるようにします (『Fortran プログラミングガイド』の第 6 章「浮動小数点演算」を参照してください)。

  一貫した結果を得るには、プログラム中のすべての副プログラムを同じ `-fsimple` オプションを付けてコンパイルする必要があります。

  n には、0、1、2 のいずれかを指定します。デフォルトは以下のとおりです。

  - `-fsimple`フラグを指定していない場合は、コンパイラは `-fsimple=0` とみなします。
  - `-fsimple` だけを指定した場合は、コンパイラは `-fsimple=1` とみなします。

  別の浮動小数点単純化レベルは次のとおりです。

  `-fsimple=0`

  前提を単純化しません。IEEE 754 に厳密に準拠してください。

  `-fsimple=1`

  若干の単純化を認めます。生成されるコードは IEEE 754 に厳密には準拠していませんが、大半のプログラムの数値結果は変わりありません。

  `-fsimple=1` を指定すると、オプティマイザは以下のことを前提とします。

  - IEEE 754 のデフォルトの丸め/トラップモードは、プロセス初期化後も変化しない。
  - 浮動小数点例外を除いて、外に現れない結果 (中間結果) を生成する演算は削除してもよい。
  - 演算対象として無限または非数を伴う演算において、非数を結果に反映させる必要はない。たとえば、`x*0` は `0` で置き換えてよい。
  - 演算がゼロの符号に応じて変化することはない。

-fsimple=1 を指定した場合、丸めや例外をまったく考慮しないで最適化を行うことはできません。特に、浮動小数点演算を、実行時に一定に保たれる丸めモードにおいて異なる結果を生成する浮動小数点演算と置き換えることはできません。

-fsimple=2

積極的な浮動小数点の最適化を許可します。このため、一部のプログラムは、数式の評価方法の変更が原因で、異なる数値結果を出すことがあります。特に、Fortran の標準規則は、部分式の明示的な括弧を重視して式の評価の配列を制御するため、-fsimple=2 によって違反が生じることがあります。その結果、Fortran の規則に依存するプログラムにおいて、数値の丸めに差異が生じる可能性があります。

たとえば、-fsimple=2 を使用すると、コンパイラは C-(A-B) を (C-A)+B として評価するため、最終的なコードがより良好に最適化されている場合、明示的な括弧について標準規則の違反が生じます。また、コンパイラは、*x/y* の反復演算を *x*z* で置き換えることがあります。この場合、*z=1/y* が 1 回だけ計算されて一時的に保存されるため、コストのかかる割り算が除去されます。

浮動小数点演算の特定プロパティに依存するプログラムは、-fsimple=2 でコンパイルすべきではありません。

ただし、-fsimple=2 を指定していても、-fsimple=2 を指定しなければ発生しない浮動小数点例外をプログラムに発生させるような最適化はできません。

-fast では、fsimple=2 を設定します。

-ftrap=*t*

起動時に有効になる浮動小数点のトラップモードを設定します。

- **f77/f95**

t には、以下のうち 1 つまたは複数の項目をコンマで区切って指定します。

%all, %none, %common, [no%]invalid, [no%]overflow, [no%]underflow,[no%]division, [no%]inexact

-ftrap=common は、-ftrap=invalid,overflow,underflow,division. のマクロです。

% は省略できません。

f77 のデフォルトは -ftrap=%none、f95 のデフォルトは -ftrap=common です。

このオプションは、プログラムの初期化時に確定される IEEE 754 のトラップモードを設定します。処理は左から右に行われます。共通の例外とは、演算不可能、ゼロによる除算、およびオーバーフローと定義されています。たとえば、`-ftrap=overflow` のように指定します。

例 : `-ftrap=%all,no%inexact` は、`inexact` を除くすべての例外に対して、トラップを設定するという意味です。

以下の点を除いて、`-ftrap=`*t* の意味は `ieee_flags()` と同じです。

- `%all` は、全トラップモードをオンにし、予期している例外にも予期していない例外にもトラップを発生させます。この代わりに `common` を使用してください。
- `%none` (`f77` のデフォルト) はすべてのトラップモードをオフにします。
- 先頭に付いている `no%` はそのトラップモードをオフにします。

このオプションを有効にするには、主プログラム全体にこのオプションを付けてコンパイルする必要があります。

詳細は、『Fortran プログラミングガイド』の第 6 章「浮動小数点演算」を参照してください。

`-G`

実行可能ファイルの代わりに、動的共有ライブラリを構築します。

- **f77/f95**

このオプションは、動的共有ライブラリを構築するようリンカーに指示します。`-G` を指定しないとリンカーは実行可能ファイルを構築します。出力ファイル名を指定するには、`-G` オプションと共に `-o` オプションを使用します。詳細は、『Fortran プログラミングガイド』の第 4 章「ライブラリ」を参照してください。

`-g`

デバッグとパフォーマンス分析のためにコンパイルします。

- **f77/f95**

`dbx(1)` または Sun WorkShop デバッグユーティリティによるデバッグ、および Sun WorkShop パフォーマンスアナライザによるパフォーマンス分析のために、シンボルテーブル情報を生成します。

-g の指定がなくてもある程度のデバッグはできますが、dbx とデバッグのすべての機能を使用するには、-g を付けてコンパイルする必要があります。

-g とともに指定したオプションの機能が制限される場合があります。

詳細に関しては、『dbx コマンドによるデバッグ』を参照してください。

コマンド行上に .o オブジェクトファイルを指定すると、-g オプションによって、-xildon がデフォルトのインクリメンタルリンカーオプションになります 。つまり、-g を指定すると、コマンド行で -G オプションまたはソースファイル名を指定しないかぎり、ld ではなく ild が自動的に起動されます。

Forte Developer 6 (Sun WorkShop 6) パフォーマンスアナライザの機能をフルに利用するには、-g でコンパイルします。一部のパフォーマンス分析機能は -g を必要としませんが、注釈付きのソースコード、一部の関数レベルの情報、およびコンパイラの注釈メッセージを確認するには、-g でコンパイルする必要があります (analyzer(1) のマニュアルページおよび『プログラムのパフォーマンス解析』を参照してください)。

-g で生成される注釈メッセージは、プログラムのコンパイル時にコンパイラの実行した最適化と変換について説明します。これらのメッセージは、ソースコードに挟みこまれているため、er_src(1) コマンドで表示できます。

注釈メッセージは、コンパイラが実際に最適化を実行した場合に限り表示されます。-x04、-fast などを使用して高度な最適化レベルを要求すると、注釈メッセージの表示される可能性が高くなります。

-h*name*

生成する動的共有ライブラリの名前を指定します。

- **f77/f95**

このオプションはリンカーに渡されます。詳細は、Solaris の『リンカーとライブラリ』および『Fortran プログラミングガイド』の第 4 章「ライブラリ」を参照してください。

-h*name* オプションにより、作成される共有動的ライブラリに、ライブラリの内部名として *name* という名前が記録されます。-h と *name* の間には空白文字を入れないでください (ライブラリ名が elp の場合を除く。この場合、空白が必要となる)。通常、*name* には -o の後に指定する名前と同じものを指定してください。-G を指定せずにこのオプションを使用しても意味がありません。

-h*name* オプションを省略すると、ライブラリファイルに内部名は記録されません。

ライブラリに内部名がある場合、このライブラリを引用する実行可能プログラムを実行するときは、実行時リンカーはあらゆるパスを検索して、同じ内部名を持つライブラリを探します。内部名を指定しておくと、実行時リンクの際に行うライブラリの検索が、より柔軟になります。このオプションは、共有ライブラリのバージョンを指定する場合にも使用できます。

共有ライブラリの内部名がない場合、リンカーは代わりに共有ライブラリファイルの特定のパスを使用します。

-help

コンパイルオプションの一覧を表示します。

- **f77/f95**

オプションの要約リストを表示します。111 ページの「-xhelp=h」も参照してください。

-I*dir*

インクルードファイルの検索パスに dir を追加します。

- **f77/f95**

インクルードファイルの検索パスの先頭に、ディレクトリ `dir` を挿入します。`-I` と *dir* の間には、空白文字を入れないでください。無効なディレクトリを指定した場合には、警告メッセージが表示されずに無視されます。

インクルードファイルの検索パスとは、インクルードファイルを探すために使用するディレクトリのリストです。インクルードファイルとは、プリプロセッサ指令 `#include`、または Fortran の `INCLUDE` 文に指定するファイルです。

例 : `/usr/app/include` でインクルードファイルを検索するには

```
demo% f95 -I/usr/app/include growth.F
```

コマンド行で複数回 `-I`dir オプションを指定することができます。各オプションを指定するごとに、検索パスリストの先頭に最初に検索するパスとして追加されます。

`INCLUDE` 文または `#include` 指令の相対パス名は次の順序で検索されます。

1. ソースファイルがあるディレクトリ

2. -I*dir* オプションで指定したディレクトリ

3. デフォルトのリストにあるディレクトリ

-Idir のデフォルトのリストは、コンパイラのインストールディレクトリに依存します。標準インストールの場合、コンパイラのソフトウェアパッケージは、/opt ディレクトリに置かれます。コンパイラが使用するインクルードファイルのデフォルトの検索パスは次のようになります。

f77 の場合

<install_dir>/SUNWspro/<release>/include/f77 /usr/include

f95 の場合

<install_dir>/SUNWspro/<release>/include/f90 /usr/include

<*install_dir*> は、インストールされているパッケージへのパス (通常インストールでは、/opt) であり、<*release*> は、ソフトウェアのリリースごとに異なるパスです。

-i2

デフォルトの整数サイズを 2 バイトに設定します。

- **f77**

整定数、論理定数、およびサイズが明示的に宣言されていない変数のデフォルトサイズを 2 バイトに設定します (ただし INTEGER*n Y とした場合は、-i2 オプションを指定しても Y は n バイトに宣言されます)。このオプションによってパフォーマンスが低下する場合があります。したがって、-i2 オプションを使用するよりも具体的な変数 INTEGER*2 を使用することをお勧めします。

-i4

デフォルトの整数サイズを 4 バイトに設定します。

- **f77**

整定数、論理定数、およびサイズが明示的に宣言されていない変数のデフォルトサイズを 4 バイトに設定します (ただし INTEGER*n Y とした場合は、-i4 オプションを指定しても Y は n バイトに宣言されます)。

INTEGER と LOGICAL のデフォルトサイズは 4 バイトですが、このオプションを使用することにより、デフォルトを 8 バイトに設定。-dbl や -r8 オプションによる設定を変更することができます。

```
demo% f77 -dbl -i4 *.f
コマンド行の警告: -i4 により -dbl の整数部が上書きされます。
```

-inline=[%auto][[,][no%]f1,...[no%]fn]

指定のルーチンのインライン化を有効または無効にします。

- **f77/f95**

オプティマイザが、f1,...,fn リストで指定されたユーザー作成ルーチンをインライン化します。ルーチン名に no% という接頭辞をつけると、そのルーチンのインライン化が無効になります。

インライン化とは最適化の手法の 1 つで、CALL や関数呼び出しなどの副プログラムの引用を、実際の副プログラムコードに効果的に置き換えます。インライン機能を有効にすると、オプティマイザが効率的なコードを生成できる機会が増えます。

リストには、関数とサブルーチンをコンマで区切って指定します。関数のインライン化を禁止するには、関数名に no% という接頭辞を付けます。

例：ルーチン xbar、zbar、vpoint をインライン化します。

```
demo$ f95 -O3 -inline=xbar,zbar,vpoint *.f
```

このオプションを使用するための条件を示します。ただし、条件が満たされていなくても、警告メッセージは出力されません。

- 最適化レベルが -O3 以上に設定されている。
- ルーチンのソースがコンパイルされているファイル中にある。ただし、-xcrossfile が指定されている場合を除く。
- コンパイラは、実際にインライン化した結果が安全で効果があるかどうかを判断する。

-inline を -O4 とともに指定すると、コンパイラが通常実行する自動インライン化機能が使用できなくなります (%auto も指定した場合は例外)。なお、-O4 を指定すると、コンパイラは通常、ユーザー作成のサブルーチンや関数をすべてインライン化し

ようとします。-O4 に -inline を追加すると、オプティマイザはリスト中にあるルーチンに限ってインライン化を行うため、実際にはパフォーマンスが低下します。この場合、%auto サブオプションを使用して、-O4 および -O5 で自動インライン化を有効にします。

```
demo% f95 -O4 -inline=%auto,no%zpoint *.f
```

上記の例では、-O4 の自動インライン化を有効にしながら、コンパイラが試みる zpoint() ルーチンのインライン化を無効にしています。

-Kpic

-pic と同義です。

- **f77/f95**

-KPIC

-PIC と同義です。

- **f77/f95**

-L*dir*

ライブラリ検索ディレクトリのリストに dir を追加します。

- **f77/f95**

オブジェクトライブラリの検索ディレクトリのリストの先頭にディレクトリ *dir* を追加します。-L と *dir* の間の空白文字はあってもなくてもかまいません。このオプションはリンカーに渡されます。70 ページの「-lx」を参照してください。

ld(1) は、実行可能ファイルを生成しながら、dir でアーカイブライブラリ (.a ファイル) と共有ライブラリ (.so ファイル)を探します。ld はまず *dir* を検索してから、デフォルトのディレクトリを探します (ライブラリの検索順序に関する詳細は、『Fortran プログラミングガイド』の第 4 章「ライブラリ」を参照してください)。LD_LIBRARY_PATH と -L*dir* の相対的な序列については ld(1) を参照してください。

注 – L*dir* を使用して /usr/lib または /usr/ccs/lib を指定すると、バンドルされていない libm はリンクされなくなります。これらのディレクトリはデフォルトで検索されます。

例: -L*dir* を使用して、ライブラリを検索するディレクトリを指定します。

```
demo$ f77 -Ldir1  -Ldir2  any.f
```

-l*x*

リンカー検索ライブラリのリストに、ライブラリ libx.a を追加します。

- **f77/f95**

-l*x* をリンカーに渡して、ld が未解決の参照を検索するためのライブラリを追加指定します。オブジェクトライブラリ lib*x* をリンクします。共有ライブラリ lib*x*.so が使用できる場合 (-Bstatic または -dn が指定されていない場合)、ld はこれを使用します。そうでなければ、ld は静的ライブラリ lib*x*.a を使用します。共有ライブラリを使用する場合は、名前は a.out に組み込まれます。-l と *x* の間には空白文字を入れないでください。

例 : ライブラリ libV77 をリンクします。

```
demo$ f77 any.f -lV77
```

複数のライブラリとリンクするには、-l*x* を再度使用してください。

例 : ライブラリ liby と libz をリンクします。

```
demo$ f77 any.f -ly -lz
```

ライブラリの検索パス、および検索順序については、『Fortran プログラミングガイド』の第 4 章「ライブラリ」を参照してください。

-libmil

最適化として libm ライブラリルーチンをインライン化します。

- **f77/f95**

  libm ライブラリルーチンの一部をインライン化します。このオプションによって、現在使用している浮動小数点オプションおよびプラットフォームにおいて最も速い実行可能ファイルを生成するインラインテンプレートが選択されます。

  詳細は、libm_single(3F) および libm_double(3F) のマニュアルページを参照してください。

-loopinfo

並列化の結果を表示します。Fortran コンパイラの並列化機能には、Forte for HPC のライセンスが必要です。

- **f77/f95**

  -parallel、-autopar、または -explicitpar の各オプションによって並列化されたループとされていないループを表示します。(オプション -loopinfo は、いずれかの並列化オプションと一緒に指定しなければなりません。)

  -loopinfo により、標準エラーに次のメッセージリストが出力されます。

```
demo% f77 -o shalow -fast -parallel -loopinfo shalow.f
shalow.f:
 MAIN shalow:
    inital:
    calc1:
...etc
"shalow.f", 325 行目: 並列化されません、利得なし (インラインループ)
"shalow.f", 172 行目: 並列化されます、逐次版が生成されました
"shalow.f", 173 行目: 並列化されません、利得なし
"shalow.f", 181 行目: 並列化されます、融合
"shalow.f", 182 行目: 並列化されません、利得なし
"shalow.f", 193 行目: 並列化されません、利得なし
"shalow.f", 199 行目: 並列化されます、逐次版が生成されました
"shalow.f", 200 行目: 並列化されません、利得なし
"shalow.f", 226 行目: 並列化されます、逐次版が生成されました
"shalow.f", 227 行目: 並列化されません、利得なし
...etc
```

  f77 のコンパイルと error(1) ユーティリティを使用すると、このメッセージリストをソースファイルにマージして、各ループに並列化の有無を示すタグを付けたソースを生成することができます。

例: 標準エラーを error ユーティリティに渡します。

```
demo$ f95 -autopar -loopinfo any.f 2>&1 | error オプション
```

error により、入力ソースファイルが書き換えられる点に注意してください。error について詳細は、error(1) マニュアルページ、または『Fortran プログラミングガイド』のデバッグに関する説明を参照してください。

-M*dir*

Fortran 95 モジュールの検索に使用するディレクトリに dir を追加します。

- **f95**

  モジュールファイルの検索に使用するディレクトリのリストに dir を追加します。-M と dir の間に空白文字は入れません。

  -M で指定したディレクトリは、現在のディレクトリの後に検索されます。モジュールのあるソースファイルをコンパイルすると、検出された MODULE ごとに .mod モジュールファイルが生成されます。Fortran 95 モジュールについての詳細は、187 ページの「モジュールファイル」を参照してください。

-misalign

境界整列に失敗したデータを受け付けます。

- **f77**

  -misalign オプションは、通常であればエラーになるようなメモリー境界整列に失敗したデータを受け付けます。COMMON 文および EQUIVALENCE 文の使用法によっては、データの境界整列が失敗する場合もあります (コンパイラの診断時)。-misalign オプションを使用すれば、コンパイラは作為的な境界整列の失敗を受け入れ、COMMON ブロックに本来のデータの境界整列を保証するためのパディングを挿入しません。しかし、これによってパフォーマンスはかなり低下します。このオプションを使用する代わりに、データが正しく整列されるようにコードを書き直すことをお勧めします。

  -misalign を使用する場合は、プログラム内のすべてのルーチンをこのオプションでコンパイルしなければなりません。コンパイルとリンクを別々に行う場合に、-misalign オプションでコンパイルするときは、リンクでもこのオプションを指定する必要があります。

-misalign は、-xmemalign=1i -aligncommon=1 と等価なマクロです。

118 ページの「-xmemalign[=<a><b>]」 を参照してください。

-mp={%none|sun|cray|openmp}

並列化指令の形式を選択します。Fortran 並列化機能には、Forte for HPC のライセンスが必要です。

- **f77/f95**

-mp オプションを省略した場合のデフォルトは %none になります。

-mp=sun	Sun 形式の指令 (接頭辞が C$PAR または !$PAR) を受け付けます。
-mp=cray	Cray 形式の指令 (接頭辞が CMIC$ または !MIC$) を受け付けます。
-mp=openmp	OpenMP Fortran 指令を受け付けます (f95 だけで使用できます)。
-mp=%none	全並列化指令を無視します。

同じコンパイラ単位内では、OpenMP 指令は Sun または Cray のいずれかの指令と併用することができます。しかし、Sun 指令と Cray 指令は、同じコンパイル単位内で同時に使用することはできません。たとえば、

-mp=sun, openmp および -mp=cray, openmp は、指定できますが、-mp=sun, creay は指定できません。

並列化を有効にするには、-explicitpar (または -parallel) を指定する必要があります。正確さを期すには、-stackvar も指定します。

```
-explicitpar -stackvar -mp=openmp
```

OpenMP 用にコンパイルする場合、-openmp フラグを使用します。このフラグは、-mp=openmp と OpenMP に必要なその他のフラグを含んでいます。80 ページを参照してください。

これらの f77/f95 指令については、このマニュアルの付録 E にまとめられています。詳細は、『Fortran プログラミングガイド』の並列化に関する説明を参照してください。

-mt

マルチスレッド環境で使用しても安全なライブラリが必要になります。

- **f77/f95**

  スレッド環境 で使用しても安全なライブラリにリンクする必要があります。ユーザーが独自に低レベルのスレッド管理を行う場合 (たとえば、libthread ライブラリを呼び出す場合) は、-mt を使用してコンパイルすると、衝突を防ぐことができます。

  libthread ライブラリを呼び出す C のマルチスレッド C コードとFortran を併用する場合は、-mt を使用します。Solarisの『マルチスレッドのプログラミング』も参照してください。

  -autopar、-explicitpar または -parallel のオプションを使用すると、自動的に -mt の機能が有効になります。

  次の点に注意してください。

  - -mt を使用するときは、入出力を伴う関数の副プログラム自体を入出力文の一部として引用することはできません。このような入出力は再帰入出力と呼ばれ、デッドロックの原因になることがあります。
  - 一般的な注意として、ユーザー独自でマルチスレッド化したコードを -autopar、-explicitpar、または -parallel オプションを付けてコンパイルしないでください。コンパイラが生成するスレッドライブラリへの呼び出しとプログラム自体の呼び出しが衝突して、予測できない結果になることがあります。
  - シングルプロセッサシステムの場合に、-mt オプションを使用すると、パフォーマンスが低下することがあります。

-native

使用中のマシンに最適なパフォーマンスにします。(廃止)

- **f77/f95**

  このオプションは、-xtarget=native と同じです。-fast オプションでは、-xtarget=native と設定します。

-noautopar

自動並列化を行いません。

- **f77/f95**

  コマンド行で先に指定された -autopar で起動されている自動並列化を無効にします。

-nodepend

コマンド行で指定した -depend を取り消します。

- **f77/f95**

  コマンド行で先に指定された -depend を取り消します。

-noexplicitpar

明示的な並列化を行いません。

- **f77/f95**

  コマンド行で先に指定された -explicitpar で起動されている明示的な並列化を無効にします。

-nolib

システムライブラリとリンクしません。

- **f77/f95**

  システムライブラリや言語ライブラリと自動的にリンクを行いません。つまりデフォルトの -l*x* オプションを ld に渡さないということです。通常は、ユーザーがコマンド行で指定しなくても、システムライブラリは実行可能ファイルに自動的にリンクされます。

  -nolib オプションを使用すると、必要なライブラリの中の 1 つを静的にリンクするといった作業が容易になります。ただし、システムライブラリや言語ライブラリは、最終的な実行に必要なので、ユーザーが手作業でそれらのライブラリをリンクしてください。

  たとえば、libF77 と動的にリンクされているプログラムが、libF77 をもたないリモートシステム上でエラーになる例を考えてみます。-nolib オプションを使用することによって、ライブラリをプログラムに静的にリンクすることができます。

  f77 では、libF77 を静的にリンクし、libc を動的にリンクします。

```
demo$ f77 -nolib any.f -Bstatic -lF77 -Bdynamic -lm -lc
```

f95 では、libm を静的にリンクし、libc を動的にリンクします。

```
demo% f95 -nolib any.f95 -Bstatic -lm -Bdynamic -lc
```

-l*x* オプションの指定の順番には意味があります。例に示す順序で指定してください。

-nolibmil

コマンド行の -libmil を取り消します。

- **f77/f95**

このオプションは、次の例のように、-fast オプションの後に使用して、libm 数学ルーチンのインライン化を無効にします。

```
demo$ f77 -fast -nolibmil ...
```

-noqueue

ライセンスのキューイングを使用しません。

- **f77/f95**

このオプションを使用している場合には、コンパイラを実行するためのソフトウェアライセンスがないと、ユーザー要求をキューイングせず、コンパイルを行わずに返します。make ファイルでの判定用に、ゼロでない環境ステータスが返されます。

-noreduction

コマンド行の -reduction を無効にします。

- **f77/f95**

このオプションにより、-reduction オプションが無効になります。

-norunpath

実行可能ファイル中に、実行時共有ライブラリのパスを設定しません。

- **f77/f95**

コンパイラは通常、実行時リンカーが共有ライブラリを検索する位置を示すパスを実行可能ファイル中に設定します。このパスはインストールの形式によって異なります。-norunpath オプションは、実行可能ファイルにパスが組み込まれないようにします。

ライブラリを標準でない場所にインストールし、別のサイトで実行可能ファイルを実行したときに、ローダーがそのパスを検索しないようにする場合に、このオプションを使用します。-R*path* と比較してみてください。

詳細は、『Fortran プログラミングガイド』の第 4 章「ライブラリ」を参照してください。

-O [*n*]

実行時間を最適化します。

- **f77/f95**

n には 1、2、3、4、5 のいずれかを指定します。-O と *n* の間には空白文字を入れないでください。

-O[*n*] の指定がない場合、基本的な最適化のレベルは、局所の共通部分式の除去、および不要コードの分析だけに限られます。プログラムのパフォーマンスは、最適化なしの場合よりも、特定の最適化レベルを指定してコンパイルした方が、大幅に改善されることがあります。通常のプログラムには、-O オプション (レベル -O3) または -fast オプション (レベル -O5) を使用することをお勧めします。

-O*n* の各レベルには、それよりも低いレベルでの最適化が含まれています。一般に、プログラムのコンパイル時の最適化レベルが高いと、実行時のパフォーマンスも向上します。ただし、最適化レベルを高くすると、コンパイル時間が長くなり、実行可能ファイルのサイズが大きくなります。

-g オプションは -O*n* を抑制しませんが、-O*n* は -g のいくつかの機能を制限します。dbx に関するマニュアルを参照してください。

-O3 と -O4 のオプションでは、dbx から変数を表示できないという点で、デバッグ機能が制限されますが、dbx where コマンドを使用してシンボルを追跡することができます。

オプティマイザがメモリーを使い切ると、レベルを下げて最適化をやり直します。以降のルーチンでは元のレベルに戻ってコンパイルを行います。

最適化の詳細については、『Fortran プログラミングガイド』の第 8 章「パフォーマンスプロファイリング」と第 9 章「パフォーマンスと最適化」を参照してください。

-O

-O3 と同義です。

-O1

文レベルの最小限の最適化を行います。

高いレベルの最適化では、コンパイル時間が長すぎる場合、またはスワップ領域が不足する場合に、-O1 を使用します。

-O2

基本ブロックレベルの最適化を行います。

通常、生成されるコードのサイズが最も小さくなります (128 ページの「-xspace」を参照してください)。

-O3 を使用すると、コンパイル時間が長すぎる場合、スワップ領域が不足する場合、または生成される実行可能ファイルのサイズが大きすぎる場合には -O2 を使用します。これ以外の場合は、-O3 を使用してください。

-O3

関数レベルで、ループを展開し大域的に最適化を行います。

通常、-O2、-O1 を使用した場合よりも生成される実行可能ファイルのサイズが大きくなります。

-O4

同一ファイル内にあるルーチンを自動的にインライン化します。

インライン化が行われるため、生成される実行可能ファイルのサイズが通常大きくなります。

-g オプションを指定すると、-O4 による自動的なインライン化は行われません。-xcrossfile を使用すると、-O4 によるインライン化の範囲が拡張されます。

-O5

最高レベルの最適化を行います。

プログラム中で、全体の計算時間のうちの最大部分を消費する部分に限って適用してください。-O5 の最適化アルゴリズムは、ソースプログラム中でこのレベルを適用する部分が大きすぎると、コンパイルに時間がかかり、パフォーマンスが低下します。

プロファイルのフィードバックと併せて使用すると、最適化がパフォーマンスの向上につながる可能性が高まります。123 ページの「-xprofile=p」を参照してください。

-o *name*

書き込み先の実行可能ファイルの名前を指定します。

- **f77/f95**

-o と nm の間には空白文字を 1 つ入れてください。このオプションを省略すると、デフォルトとして実行可能ファイルが a.out に書き込まれます。また -c とともに使用すると、-o はターゲットの .o オブジェクトファイルの名前を指定します。また -G とともに使用すると、ターゲットの .so ライブラリファイルの名前を指定します。

-oldldo

以前のリスト形式の出力を選択します。

- **f77/f95**

並びによる出力で各記録の始まりにある空白を省きます。現在のデフォルトは f77 リリース 1.4 以前のバージョンとは異なり、Fortran 規格に合わせるため、空白を付けるようになっています。また、OPEN の FORM='PRINT' オプションにも注意してください。-oldldo をプログラムの一部だけに適用し、残りの部分にはこのオプションを指定せずにコンパイルすることもできます。

-onetrip

DO ループを 1 回だけ実行します。

- **f77/f95**

DO ループが、少なくとも 1 回は実行されるようにコンパイルします。標準 Fortran の DO ループは、一部の古典的な Fortran の実装とは異なり、上限が下限より小さい場合には、1 回も実行されません。

-openmp

Fortran 95 の OpenMP 指令で明示的な並列化を有効にします。Fortran 並列化機能には、Forte for HPC のライセンスが必要です。

- **f77/f95**

このマクロは、次に示すオプションを組み合わせたマクロです。

```
-mp=openmp -explicitpar -stackvar -D_OPENMP=200011
```

OpenMP 指令については、付録 E にまとめられています。

並列化されたプログラムをマルチスレッド環境で実行するには、実行前に PARALLEL (または OMP_NUM_THREADS) 環境変数を設定しておく必要があります。これは、プログラムが作成できる最大スレッド数を実行時システムに指示しています。デフォルトは 1 です。一般的に、ターゲットプラットフォーム上の PARALLEL 変数または OMP_NUM_THREADS 変数には、利用可能なプロセッサ数を設定します。

OpenMP では、プリプロセッサ記号 _OPENMP の定義に 10 進数 YYYYMM を含める必要があります。ここで、YYYY と MM は、この実装がサポートする OpenMP Fortran API のバージョンの年と月を示します。

-p

prof プロファイラを使用するプロファイル用にコンパイルします。

- **f77/f95**

プロファイル用のオブジェクトファイルを作成します。prof(1) を参照してください。コンパイルとリンクを分けて行う場合、-p オプションを付けてコンパイルしたときはリンクでも必ず -p オプションを付けてください。-p と prof は主に旧式のシステムとの互換性を保つため使用します。gprof と -pg の方を使用することをお勧めします。詳細は、『Fortran プログラミングガイド』のパフォーマンスプロファイルに関する説明を参照してください。

-pad[=*p*]

キャッシュを効率よく利用するためにパディングを挿入します。

- **f77/f95**

配列や文字変数が、静的な局所変数でまだ初期化されていない場合、または共通のブロックにある場合、間にパディングを挿入します。またキャッシュを効率的に利用できるように、データにパディングを追加します。いずれの場合も、配列または文字変数を等価にすることはできません。

p は、次のいずれかまたは両方です。

local	隣接する局所変数の間にパディングを追加挿入します。
common	共通ブロック変数の間にパディングを追加挿入します。

-pad のデフォルトは以下のとおりです。

- -pad[=*p*] オプションが指定されていない場合は、コンパイラはパディングを挿入しません。
- -pad だけが指定され *p* の指定がない場合は、コンパイラは局所変数と共通ブロック変数の両方にパディングを挿入します。

以下に示すコマンド行は同義です。

- `f77 -pad` *any*`.f`
- `f77 -pad=local,common` *any*`.f`
- `f77 -pad=common,local` *any*`.f`

-pad[=*p*] オプションは、以下の条件を満たす項目に適用されます。

- 配列、または文字変数になっている項目
- 静的で局所的または共通ブロックにある項目

局所変数、または静的変数については、90 ページの「-stackvar」の説明を参照してください。

-pad=common に関する制限事項

- 配列と文字列のどちらに対しても EQUIVALENCE 文を適用できません。
- ある共通ブロックを引用するファイルのコンパイルで -pad=common を指定するときは、その共通ブロックを引用するすべてのファイルのコンパイルで -pad=common を指定する必要があります。このオプションは、共通ブロック内の

変数の配置を変更します。あるプログラム単位をこのオプション付きでコンパイルし、別のプログラム単位をこのオプションなしでコンパイルすると、共通ブロック内の同じ位置への引用が、別の位置を引用してしまう可能性が生じます。

- -pad=common を指定する場合、別のプログラム単位にある共通ブロックの変数宣言を名前を除いて同じにする必要があります。共通ブロックの変数の間に挿入されるパディングの量は、このような変数の宣言内容に応じて異なります。別のプログラム単位にある変数のサイズやランクが異なる場合は、同じファイル内でも変数の位置が異なることがあります。
- -pad=common を指定する場合、共通ブロック変数を伴う EQUIVALENCE を宣言すると、エラーになります。
- -pad=common が指定されている場合、共通ブロック変数を伴う EQUIVALENCE を宣言すると、警告メッセージが表示されてエラーになります。ブロックはパディングされません。
- -pad=common が指定されている場合、共通ブロック内の配列のオーバーインデックスを避けてください。パディングされた共通ブロックで隣接データの位置を変更すると、予想外のかたちでオーバーインデックスが失敗します。

-parallel

-autopar、-explicitpar、-depend でループを並列化します。

Fortran 並列化機能には、Forte for HPC のライセンスが必要です。

- **f77/f95**

並列化するループの選択は、コンパイラによって自動的に、またユーザーの明示的な指令によって行われます。最適化レベルが -O3 よりも低い場合は、自動的に -O3 に設定されます。

パフォーマンスを改善するには、並列化オプション (-autopar も含む) を使用する場合には、-stackvar オプションも指定してください。

31 ページの -mp を使用して、Sun、Cray または f95 の OpenMP のいずれかの形式の並列化指令を選択します。

ユーザー独自のスレッド管理を行なっている場合は、-parallel は使用しないでください。31 ページの -mp の説明を参照してください。

-parallel のような並列化オプションは、マルチプロセッサシステムで実行するための実行可能プログラムを生成することを前提としています。シングルプロセッサシステムで並列化を行うと、通常はパフォーマンスが低下します。

並列化されたプログラムをマルチスレッド環境で実行するには、実行前に PARALLEL (または OMP_NUM_THREADS) 環境変数を設定しておく必要があります。これは、プログラムが作成できる最大スレッド数を実行時システムに指示しています。デフォルトは 1 です。一般的に、ターゲットプラットフォーム上の PARALLEL 変数または OMP_NUM_THREADS 変数には、利用可能なプロセッサ数を設定します。

-parallel を使用して、同じステップ内でコンパイルおよびリンクを行うと、リンクには自動的にマルチスレッドライブラリおよびスレッド対応の Fortran 実行時ライブラリが含まれます。-parallel を使用して、コンパイルおよびリンクを別々のステップで行う場合は、リンクする際にも -parallel を指定しなければなりません。

詳細は、『Fortran プログラミングガイド』の第 10 章「並列化」を参照してください。

-pg

gprof プロファイラを使用するプロファイル用にコンパイルします。

- **f77/f95**

  -p オプションを使用した場合と同様の形式でプロファイル用にコードをコンパイルします。ただし詳細な統計情報を記録する実行時記録メカニズムも起動され、プログラムが正常に終了すると、gmon.out ファイルが生成されます。gprof を実行すると、実行プロファイルが生成されます。詳細は、gprof(1) のマニュアルページおよび『Fortran プログラミングガイド』を参照してください。

  ライブラリオプションは、.f と .o ファイルの後に指定してください (-pg ライブラリは静的です)。

  コンパイルとリンクを分けて行う場合、-pg を付けてコンパイルしたときはリンクでも必ず -pg を付けてください。

-pic

共有ライブラリ用の位置独立コードをコンパイルします。

- **f77/f95**

このオプションは動的共有ライブラリをコンパイルするときに使用します。大域的なデータへの引用はそれぞれ、大域的なオフセットテーブル内でポインタの間接引用として生成されます。関数呼び出しは、プロシージャリンケージテーブルを通して pc 相対アドレッシングモードで生成されます。

- 大域オフセットテーブルのサイズは、SPARC では 8KB に制限されます。
- -pic と -PIC を同時に指定しないでください。

-pic は -xcode=pic13 と同義です。

-pic または -PIC には、次のようなパフォーマンス上の影響があります。

- -pic または -PIC のいずれかでコンパイルされたルーチンは、エントリで命令をいくつか実行することによって、共有ライブラリの大域変数や静的変数へのアクセスに使用する大域的なオフセットテーブルを指すようにレジスタを設定します。
- 大域変数または静的変数にアクセスするごとに、大域オフセットテーブルを介して余分な間接メモリー引用を行います。-PIC でコンパイルを実行すると、大域的または静的なメモリー引用を行うごとに、命令が 2 つ追加されます。

以上のパフォーマンス上の犠牲を比較した場合、-pic と -PIC を使用すると、ライブラリコードが共有されるため、必要なシステムメモリーが大幅に少なくなります。-pic または -PIC でコンパイルした共有ライブラリ中のコードの各ページは、そのライブラリを使用する各プロセスどうしで共有することができます。共有ライブラリ中にあるコードのページに 1 つでも -pic でコンパイルされていないメモリー参照 (直接メモリー参照) があると、このページは共有できなくなるため、ライブラリを使用したプログラムを実行するたびに、そのページのコピーが作成されます。

.o ファイルが -pic または -PIC でコンパイルされているかどうかを調べるには、nm コマンドを使用する方法が最も簡単です。

```
% nm file.o | grep _GLOBAL_OFFSET_TABLE_
U _GLOBAL_OFFSET_TABLE_
```

位置独立コードを含む .o ファイルには、_GLOBAL_OFFSET_TABLE_ への未解決の外部参照があります。未解決の参照は U の文字で示されます。

-pic または -PIC のどちらかを使用するかは、nm を使用して、そのライブラリ内で使用または定義されている特定の大域変数または静的変数の数を調べて判断します。_GLOBAL_OFFSET_TABLE_ のサイズが 8,192 バイト未満であれば、-pic を使用できます。これ以外の場合は、-PIC を使用します。

64 ビットの Solaris 環境で -xarch=v9 （または v9a か v9b ）を使用して共有動的ライブラリを構築する場合、-pic または -PIC オプション、あるいはこれらと同義の -xcode オプションを指定する必要があります。

-PIC

共有ライブラリ用の位置独立コードをコンパイルします。ただし 32 ビットアドレスを使用します。

- **f77/f95**

-pic に似ていますが、大域的なオフセットテーブルが 32 ビットアドレスの範囲にわたる点が異なります。このオプションは -pic では大域的なデータオブジェクトが多すぎる場合に使用します。-pic と -PIC オプションを同時に指定しないでください。

-PIC は -xcode=pic32 と同義です。

64 ビットの Solaris 環境で -xarch=v9 （または v9a か v9b ）を使用して共有動的ライブラリを構築する場合、-pic または -PIC オプション、あるいはこれらと同義の -xcode オプションを指定する必要があります。

-Qoption *pr ls*

オプションをコンパイル段階 *pr* に渡します。

- **f77/f95**

サブオプションリスト ls をコンパイルフェーズ pr に渡します。Qoption と pr および ls の間には必ず空白文字を入れます。Q は大文字でも小文字でもかまいません。ls には、コンパイル段階に適したサブオプションをコンマで区切って指定します。リスト中には空白文字を入れないでください。また、サブオプションの先頭にマイナス記号を付けることができます。

このオプションは主に、サポートスタッフによる内部デバッグ用として用意されています。LD_OPTIONS 環境変数を使用してリンカーにオプションを渡します。『Fortran プログラミングガイド』のリンクとライブラリに関する章を参照してください。

-qp

-p と同義です。

- **f77/f95**

-R *ls*

動的ライブラリの検索パスを実行可能ファイルに設定します。

- **f77/f95**

このオプションを指定すると、ld(1) リンカーは動的ライブラリ検索パスのリストを実行可能ファイルに格納します。

ls には、ライブラリ検索パスのディレクトリをコロンで区切って指定します。-R と *ls* の間には空白文字があってもなくてもかまいません。

このオプションを複数指定した場合は、それぞれのディレクトリリストがコロンで区切られて連結されます。

このリストは実行時に実行時リンカー ld.so が使用します。実行時に、このリストにあるパスで動的なライブラリを検索し、未解決の参照を解決しようとします。

このオプションは、動的ライブラリへのパスを指定するオプションを意識せずに出荷用の実行可能ファイルを実行できるようにしたいときに使用します。

-R *paths* を使用して実行可能ファイルを構築すると、ディレクトリパスはデフォルトのパスに追加されます。デフォルトのパスは、常に最後に検索されます。

標準のインストール: /opt/SUNWspro/lib

詳細は、『Fortran プログラミングガイド』の第 4 章「ライブラリ」および Solaris の『リンカーとライブラリ』を参照してください。

-r8

REAL、INTEGER、DOUBLE、および COMPLEX のデフォルトのバイトサイズを 2 倍に設定します。

- **f77/**

注 – このオプションと -dbl は、現在では使用されなくなっており、将来のリリースで削除される可能性があります。より一般的なオプションである -xtypemap を使用してください。

-r8 を使用すると、明示的なバイトサイズを指定せずに宣言されている、REAL、INTEGER、DOUBLE、および COMPLEX の変数のデフォルトのバイトサイズが拡張されます。

表 3-9　デフォルトのデータサイズ (バイト) と -r8

-r8 オプションなし		-r8 オプションあり
データ型	デフォルト	SPARC
INTEGER	4	8
REAL	4	8
DOUBLE	8	16

このオプションは、変数、パラメータ、定数、および関数に適用されます。

また、LOGICAL は INTEGER、COMPLEX は 2 つの REAL、DOUBLE COMPLEX は 2 つの DOUBLE として扱われます。

また、-dbl と -r8 は、より汎用的な -xtypemap= オプションで表すことができます。

-dbl: -xtypemap=real:64,double:128,integer:64 と同義

-r8: -xtypemap=real:64,double:128,integer:mixed と同義

これらのオプションによってデフォルトの DOUBLE PRECISION データが QUAD PRECISION (128 ビット) になりますが、パフォーマンスは低下する可能性があります。この場合、-dbl よりも -xtypemap=real:64,double:64,integer:64 を使用した方が適切です。

通常、1 つの副プログラムを -r8 付きでコンパイルする場合は、そのプログラムのすべての副プログラムも -r8 付きでコンパイルしてください。これは特に、各ファイル間で入出力形式が統一されていないプログラムでは重要になります。またこのオプションを使用すると、関数名にデータのサイズを明示的に指定しない限り、ライブラリ関数の呼び出しも含め、関数のデフォルトのデータサイズが変更される点に注意する必要があります。

このオプションが実行時のパフォーマンスに与える影響はかなり大きいものです。-r8 を付けてコンパイルすると、float = 15.0d0*float のような式が、定数 15 の宣言のために 4 倍精度で評価されることに注意してください。

-r8 と -i2 を両方指定した場合の結果は予測できません。

-r8const

単精度の定数を REAL * 8 の定数に変換します。

- **f77/f95**

単精度の REAL 定数はすべて REAL * 8 に変換されます。倍精度 (Real *8) 定数は変更されません。このオプションは、定数にだけ適用されます。定数と変数の両方を変換する場合は、129 ページの「-xtypemap=spec」 を使用してください。

-reduction

ループ中にある縮約演算を識別します。

- **f77/f95**

自動並列化中にループを解析し縮約演算を認識します。ループの縮約には、潜在的に丸めのエラーがあります。

縮約演算によって、配列内の要素が単一のスカラー値に変換されます。縮約演算の典型的な例として、あるベクトルの各要素をまとめる処理があります。このような演算は並列化の対象ではありませんが、-reduction を指定すると、コンパイラは縮約演算を認識し、特別な例として並列化します。コンパイラが認識する縮約演算については、『Fortran プログラミングガイド』の第 10 章「並列化」を参照してください。

このオプションは、自動並列化オプションの -autopar または -parallel を使用する場合のみ使用できます。それ以外の場合は無視されます。明示的に並列化されたループは縮約演算の解析の対象にはなりません。

例：-reduction オプションを付けた自動並列化処理

```
demo$ f77 -parallel -reduction any.f
```

-S

コンパイルし、アセンブリのソースコードだけを生成します。

- **f77/f95**

指定したプログラムをコンパイルし、アセンブリ言語の出力結果を、接尾辞.s の付いた名前のファイルに出力します。.o ファイルは作成しません。

-s

実行可能ファイルからシンボルテーブルを取り除きます。

- **f77/f95**

実行可能ファイルを縮小し、リバースエンジニアがしにくくします。また、このオプションを使用すると、dbx その他のツールによるデバッグができなくなり、-g オプショは無視されます。

-sb

Sun WorkShop ソースコードブラウザ用のテーブル情報を生成します。

- **f77/f90**

詳細は『Sun WorkShop の概要』を参照してください。

注 – -sb は、コンパイラが fpp または cpp プリプロセッサを経由して自動的に渡すソースファイル (すなわち、拡張子が .F、.F90 または .F95 のファイル) 上で使用することはできません。

-sbfast

ソースコードブラウザ用のテーブル情報のみを生成します。

- **f77/f95**

Sun WorkShop ソースブラウザ用のテーブル情報を生成した後、処理を終了します。アセンブルやリンクは行わず、オブジェクトファイルも作成しません。

注 – fpp または cpp プリプロセッサを介して自動的に渡されたソースファイル (つまり拡張子が .F、.F90、または .F95 のファイル) で -sbfast を使用することはできません。つまり -F オプションとともに使用することはできません

-silent

コンパイラメッセージの出力を抑制します。

- **f77/f95**

コンパイラから送られる重要でないメッセージの表示をすべて抑制します。エラーメッセージと警告メッセージは表示されます。デフォルトでは、コンパイル中に検出されたファイルとエントリの名前が表示されます。

-stackvar

局所変数すべてを強制的にメモリースタックに割り当てます。

- **f77/f95**

特に別の指定がない限り、ルーチン中にある局所変数と配列をメモリースタックに割り当てます。これにより、その変数や配列は静的ではなく自動変数になります。また、サブプログラムを呼び出してループをl並列化するときに、オプティマイザの自由度が高くなります。

並列化オプションを使用する場合は、-stackvar を使用するようにしてください。

次の条件に該当しない場合は、変数と配列は局所的となります。

- SUBROUTINE または FUNCTION 文中の引数 (すでにスタック上にある)
- COMMON、SAVE、STATIC 文中の大域的な項目
- 次のような型宣言文、または DATA 文で初期化された項目

REAL X/8.0/ または DATA X/8.0/

f77 のみ: 次の例のように、-stackvar を使用しているときに、Data 文の中の局所変数に対する実行可能な引用の後、この局所変数を初期化しようとすると、f77 への拡張となりエラーになります。

```
demo% cat stak.f
        real x
        x =  1.
        t = 0.
        print*, t
        data x/3.0/
        print *,x+t
        end
demo% f77 -o stak -stackvar stak.f
stak.f:
 MAIN:
"stak.f", 5 行目: エラー: 自動変数を初期化しようとしています: x
```

-stackvar を使用してサイズが大きい配列をスタック上に割り当てると、スタックからオーバーフローし、セグメンテーションフォルトが発生する場合があります。このような場合はスタックサイズを大きくする必要があります。

デフォルトのスタックサイズはメインスタックが 8M バイト、各スレッドスタックが 1 BM (SPARC V9 プラットフォームの場合は 2M バイト) です。引数なしで limit コマンドを実行すると、現在のメインスタックのサイズが表示されます。-stackvar を使用した時にセグメンテーションフォルトが発生する場合は、メインスタックとスレッドスタックのサイズを大きくしてみてください。

例 : 現在のメインスタックのサイズを表示します。

```
demo% limit
cputime         制限なし
filesize        制限なし
datasize        523256 kbytes
stacksize       8192 kbytes      <---
coredumpsize    制限なし
descriptors     64
memorysize      制限なし
demo%
```

例 : メインスタックのサイズを 64M バイト に設定します。

```
demo% limit stacksize 65536
```

例 : 各スレッドスタックのサイズを 8M バイト に設定します。

```
demo% setenv STACKSIZE 8192
```

並列化と -stackvar を併用する方法については、『Fortran プログラミングガイド』の第 10 章「並列化」を参照してください。limit コマンドの詳細については、csh(1) を参照してください。

-stop_status=*yn*

STOP 文により整数のステータス値を返します。

- **f77/f95**

yn には yes または no を指定します。デフォルトは yes です。

-stop_status=yes を付けると、STOP 文に整数の定数を入れることができます。その値は、プログラムの終了時に環境に渡されます。

```
STOP 123
```

STOP 文には、0 から 255 の範囲にある値を指定してください。これよりも大きい値は切り捨てられ、実行時メッセージが出力されます。ただし、

STOP *'stop string'*

は受け付けられます。この場合は環境にステータス値 0 が返されます。ただし、コンパイラの警告メッセージは出力されます。

このステータス環境変数は、C シェル (csh) では $status、また Bourne (sh) シェルと Korn (ksh) シェルでは $? です。

-temp=dir

一時ファイルのディレクトリを設定します。

- **f77/f95**

コンパイラによって使用される一時ファイル用のディレクトリを dir に設定します。このオプションでは空白文字を入れないでください。このオプションを指定しない場合、一時ファイルは /tmp ディレクトリに置かれます。

-time

各コンパイル段階の経過時間を表示します。

- **f77/f95**

各コンパイル段階で費された時間とリソースが表示されます。

-U

ソースファイル中の大文字と小文字を区別します。

- **f77/f95**

大文字と小文字を区別します。デフォルトでは、文字列定数中を除き、大文字をすべて小文字として解釈します。このオプションを指定すると、Delta、DELTA、および delta はすべて別の記号として解釈されます。

Fortran を別の言語に移植したり、混用したりする場合は、-U オプションを指定する必要があります。詳細は、『Fortran プログラミングガイド』を参照してください。

-Uname

プリプロセッサのマクロ名の定義を取り消します。

- **f77/f95**

このオプションは、fpp または cpp プリプロセッサを呼び出す .F および .F95 ソースファイルにのみ適用されます。このオプションは、同じコマンド行の -Dname で作成されたプリプロセッサのマクロ名の初期定義を削除します。この場合、オプションの順序に関係なく、コマンド行ドライバによって暗黙に配置された -Dname も対象となります。ソースファイルのマクロ定義には影響しません。コマンド行に複数の -Uname フラグを配置できます。-U とマクロ名のあいだにスペースを入れることはできません。

-u

未宣言の変数に対してメッセージを出力します。

- **f77/f95**

すべての変数に対するデフォルトの型を、Fortran の暗黙の型宣言を使用せずに「未宣言」にします。宣言していない変数に対して警告メッセージが出力されます。ただし、このオプションは、IMPLICIT 文や明示的に型を指定する文より優先されることはありません。

-unroll=*n*

DO ループの展開が可能な箇所で、使用可能にします。

- **f77/f95**

n は正の整数です。次の選択が可能です。

- n が 1 の場合、ループの展開をすべて禁止します。
- n が 2 以上の場合、オプティマイザはループを n 回展開します。

一般に、ループを展開するとパフォーマンスが改善されますが、実行可能ファイルのサイズが大きくなります。ループの展開と各種のコンパイラの最適化については、『Fortran プログラミングガイド』の第 9 章「パフォーマンスと最適化」を参照してください。また、20 ページの「UNROLL 指令」も参照してください。

-V

各コンパイラパスの名前とバージョンを表示します。

- **f77/f95**

コンパイラの実行時に、各パスの名前とバージョンを表示します。

上記の情報は、問題が発生した場合にご購入先に問い合わせる時に役立ちます。

-v

各コンパイラパスの詳細情報を表示します。

- **f77/f95**

-v と同様に、コンパイラの実行時にそれぞれのパス名を表示し、ドライバが使用したオプション、マクロフラグ展開および環境変数を詳細に表示します。

-vax=*v*

有効にする VMS Fortran の拡張機能を指定します。

- **f77/**

v には、1 つまたは複数のサブオプションをコンマで区切って指定します。各サブオプションのキーワードの前に no% を置くと (例 : no%logical_name)、無効にする機能を指定することができます。

主なオプションは、-vax=align と -vax=misalign です。

-vax=align は、境界整列されていないデータを許可せずに、すべてのサブオプションを選択します。これは、f77 リリース 3.0.1 以前の -xl オプションの動作です。

-vax=misalign は、サブオプションをすべて選択し、境界整列されていないデータを許可します。これは、f77 リリース 3.0.1 、4.0、4.2、5.0 および Sun WorkShop 6 の -xl オプションの動作です。

以下の表に選択できるサブオプションを示します。

表 3-10 -vax のサブオプション

-vax=	効果
blank_zero	数値欄の空白をゼロとして扱う。
bslash	文字定数中でバックスラッシュ (\) を使用できる。
debug	VMS Fortran の 'D' デバッグ文を使用できる。
logical_name	VMS Fortran 形式の論理ファイル名を使用できる。
oct_const	8 進数の定数を表す二重引用符を使用できる。
param	PARAMETER 文の非標準書式を使用できる。
rsize	バイト数ではなく文字数でアンフォーマットされた記録サイズを表す。
struct_align	VMS Fortran 内の構造体として境界整列する。

%all と %none を使用して、これらのサブオプションを全部選択する、またはまったく選択しない、のいずれかを指定できます。サブオプションは、リスト v が左から右に読み込まれるに従って蓄積されます。たとえば、1 つの機能を除いてすべてを使用可能にするには次のように指定します。

```
-vax=%all, no%rsize
```

「-misalign」および「-xl」を参照してください。

-vpara

並列化に関する詳細な警告メッセージを表示します。

- **f77/f95**

コンパイラが、並列化指令で明示的に指定されたループを分析するごとに、検出されるデータの依存関係に関する警告メッセージを出力します。ただし、ループの並列化は続けられます。

例 : 並列化に関する詳細な警告メッセージを出力する -vpara オプション

```
demo% f77 -explicitpar -vpara any.f
any.f:
 MAIN any:
"any.f", 11 行目 : 警告 : ループには参照を無効にする並列化が含まれているかもしれません
```

-w

警告メッセージを抑制します。

- **f77/f95**

ほとんどの警告メッセージを出力しないようにします。ただし、前に指定したオプションのすべて、あるいは一部が無効になるようなオプションを指定している場合には、警告メッセージが表示されます。

例 : -w オプションを使用しても警告メッセージが出力される場合

```
demo% f77  -w -03 -fast -O4  any.f
f95: 警告: -O4 は、すでに設定されている最適化レベル -03 に優先します
demo%
```

f95では、0 ~ 4 のレベルを指定できます。-w0 では抑制する警告メッセージが最も少なく、-w4 では抑制する警告メッセージが最も多くなります。-w は -w0 と同義です。

-Xlist[x]

リストを生成し、大域的なプログラム検査 (GPC) を実行します。

- **f77/f95**

このオプションを使用すると、潜在的なプログラムのバグを発見できます。このオプションは、予備のコンパイラパスを呼び出し、大域プログラムをとおして、副プログラムの引数、共通ブロック、およびパラメータの一貫性をチェックします。また、このオプションは行番号付きのソースコードリストを生成し、そのなかには相互参照表も含まれます。-Xlist オプションが発行するエラーメッセージは助言レベルの警告であり、プログラムのコンパイルやリンクを阻止するものではありません。

注 – ソースコードのすべての構文エラーを訂正してから、`-Xlist` でコンパイルを実行してください。構文エラーのあるソースコードでコンパイルを実行すると、予想外の結果が報告されることがあります。

例：ルーチン間の一貫性をチェックします。

```
demo% f95 -Xlist  fil.f
```

上の例により、出力ファイル `fil.lst` に次の項目が書き込まれます。

- 行番号付きのソースリスト (デフォルト)
- ルーチン間の矛盾についてのエラーメッセージ (リストに組み込まれています)
- 識別子の相互参照表 (デフォルト)

デフォルトにより、ファイル `name.lst` にリスト内容が書き込まれます。ここで、`name` はコマンド行に最初に配置されているソースファイルの名前です。

多数のサブオプションにより、さまざまな動作を柔軟に選択できます。これらのサブオプションは、`-Xlist` オプションの接尾辞によって指定されます。次の表を参照してください。

表 3-11

オプション	機能
`-Xlist`	エラー、リスト、および相互参照表を示します。
`-Xlistc`	コールグラフとエラーを示します。
`-XlistE`	エラーを示します。
`-Xlisterr[`*nnn*`]`	エラー *nnn* のメッセージを抑制します。
`-Xlistf`	エラー、リスト、および相互参照表を示します。オブジェクトファイルはありません。
`-Xlistfln`*dir*	ディレクトリ *dir* に .fln ファイルを配置します。ディレクトリ `dir` は既に存在している必要があります (`f77` のみ)。
`-Xlisth`	エラー検出時にコンパイルを終了します。

表 3-11

オプション	機能
`-XlistI`	ソースファイルとともに `#include` および `INCLUDE` ファイルを分析します。
`-XlistL`	リストとエラーのみを示します。
`-Xlistl`*n*	ページの長さを n 行に設定します。
`-Xlisto` *name*	レポートファイルの名前を *name*`.lst` に変更します。
`-Xlists`	相互参照表から参照されない名前を抑制します。
`-Xlistv`*n*	チェッキングレベルを n (1、2、3、または 4) に設定します。デフォルトは 2 です。
`-Xlistw[`*nnn*`]`	出力行の幅を *nnn* カラムに設定します。デフォルトは 79 です。
`-Xlistwar[`*nnn*`]`	警告 *nnn* のメッセージを抑制します。
`-XlistX`	相互参照表とエラーを示します。

オプション `-Xlistfln`*dir* は `f95` では利用できません。

詳細については、『Fortran プログラミングガイド』の「プログラムの解析とデバッグ」の章を参照してください。

`-xa`

`-a` と同義です。

- **f77/f95**

`-xarch=`*isa*

命令セットアーキテクチャ (ISA) を指定します。

表 3-12 -xarch ISA キーワード

プラットフォーム	有効な -xarch キーワード
SPARC	`generic, generic64, native, native64, v7, v8a, v8, v8plus, v8plusa, v8plusb, v9, v9a, v9b`

表 3-11 に、-xarch キーワード *isa* で受け付けられるアーキテクチャを示します。-xarch は単独でも使用できますが、-xtarget オプションが展開される一部であり、これを使用すると、特定の -xtarget オプションで設定した -xarch 値を無効にすることができます。たとえば、次のように指定すると、

```
% f95 -xtarget=ultra2 -xarch=v8plusb ...
```

-xtarget=ultra2 で設定した -xarch=v8 が無効になります。

このオプションは、指定の命令セットだけを許すことによって、コンパイラが指定の命令セットアーキテクチャの命令に対応するコードしか生成できないようにします。なお、ターゲット固有の命令が使用されるとは限りません。

このオプションを最適化で使用する場合は、適切なアーキテクチャを選択すると、そのアーキテクチャ上での実行パフォーマンスを向上させることができます。不適切な選択を行うと、意図したターゲットプラットフォームでは実行できないバイナリプログラムが生成されます。

表 3-13 プラットフォーム上でのもっとも一般的な -xarch オプションのまとめ

-xarch	パフォーマンス
generic	• 全プラットフォーム上で良好なパフォーマンスが得られます。
v8plusa	• 32 ビットモードの UltraSPARC-II プロセッサ上で最適なパフォーマンスが得られます。 • 他のプラットフォーム上で実行できません。
v8plusb	• 32 ビットモードの UltraSPARC-III プロセッサ上で最適なパフォーマンスが得られます。 • 他のプラットフォーム上で実行できません。
v9a	• 64 ビットモードの UltraSPARC-II プロセッサ上で最適なパフォーマンスが得られます。 • 他のプラットフォーム上で実行できません。
v9b	• 64 ビットモードの UltraSPARC-III プロセッサ上で最適なパフォーマンスが得られます。 • 他のプラットフォーム上で実行できません。

次の点にも注意してください。

- SPARC 命令セットアーキテクチャの V7、V8 および V8a は、すべてバイナリ互換があります。
- `v8plus` および `v8plusa` を指定してコンパイルしたオブジェクトバイナリファイル (`.o`) は、リンクし、まとめて実行することができますが、実行は SPARC V8plusa 互換プラットフォーム上だけに限ります。
- `v8plus`、`v8plusa` および `v8plusb` を指定してコンパイルしたオブジェクトバイナリファイル (`.o`) は、リンクし、まとめて実行することができますが、実行は SPARC `V8plusb` 互換プラットフォーム上だけに限ります。
- `-xarch` 値に `v9`、`v9a` および `v9b` を指定できるのは、UltraSPARC 64 ビットの Solaris環境だけです。
- `v9` および `v9a` を指定してコンパイルしたオブジェクトバイナリファイル (`.o`) は、リンクし、まとめて実行することができますが、実行は SPARC `V9a` 互換プラットフォーム上だけに限ります。
- `v9`、`v9a` および `v9b` を指定してコンパイルしたオブジェクトバイナリファイル (`.o`) は、リンクし、まとめて実行することができますが、実行は SPARC V9b 互換プラットフォーム上だけに限ります。

オプションの選択によっては、生成された実行可能プログラムのパフォーマンスが、初期のアーキテクチャよりかなり劣ることがあります。また、4 倍精度 (`REAL*16` および `long double`) 浮動小数点命令は、これらの命令セットアーキテクチャの多くで使用できますが、コンパイラは、それらの命令を生成したコードで使用しません。

表 3-13 に、SPARC プラットフォーム上で使用する各 `-xarch` キーワードについて詳細に説明します。

表 3-14 SPARC プラットフォーム上で使用できる各 `-xarch` 値

`-xarch`	意味
`generic`	**たいてのシステムで良好な32 ビットパフォーマンスを得るためのコンパイル** これは、デフォルトです。このオプションは、ほとんどのプロセッサ上で良好なパフォーマンスを得るために最善の命令セットを使用します、このとき、どのプロセッサ上でも大きなパフォーマンスの低下は見られません。「最善」の命令セットの定義は、新リリースごとに適宜調整できます。現時点では v7 です。

表 3-14 SPARC プラットフォーム上で使用できる各 -xarch 値 (続き)

-xarch	意味
generic64	たいていの 64 ビットシステムで良好なパフォーマンスを得るためのコンパイル このオプションは、どのようなプロセッサでもパフォーマンスを大きく下げることなく、たいていの 64 ビットプロセッサで良好なパフォーマンスを得るための最高の命令セットを使用します。新しいリリースが発表されるたびに「最高」の命令セットの定義が変わりますが、現時点では v9 と解釈されています。
native	当該システム上で良好なパフォーマンスを得るためのコンパイル これは、-fast オプションのデフォルトです。コンパイラは、コンパイルが実行されている現在のシステムプロセッサに適した設定を選択します。
native64	このシステムの 64 ビットモードで良好なパフォーマンスを得るためのコンパイル native と同様に、コンパイラは、コンパイルが実行されている現在のシステムプロセッサに適した設定を選択します。
v7	SPARC-V7 ISA 用のコンパイル コンパイラは、V7 ISA 上で良好なパフォーマンスが得られるようにコードを生成することができます。これは、V8 ISA で良好なパフォーマンスを得るために最善の命令セットを使用するのに相当します。なお、V7 ISA には mul および div という整数命令、および fsmuld 命令は含まれません。 例：SPARCstation1、SPARCstation2
v8a	SPARC-V8 ISA のV8a バージョン用のコンパイル 定義によれば、V8a は、V8 ISA を意味しますが、fsmuld 命令は含まれていません。このオプションを使用すると、コンパイラは、V8a ISA 上で良好なパフォーマンスが得られるようにコードを生成することができます。 例：microSPARC Iチップアーキテクチャに基づく任意のシステム
v8	SPARC-V8 ISA 用のコンパイル コンパイラは、V8 上で良好なパフォーマンスが得られるようにコードを生成することができます。 例：SPARCstation10

表 3-14 SPARC プラットフォーム上で使用できる各 -xarch 値 (続き)

-xarch	意味
v8plus	SPARC-V9 ISA のV8plus バージョン用のコンパイル 定義によれば、V8plus は、V9 ISA を意味しますが、V8plus ISA アーキテクチャで定義されている 32 ビットサブセットだけに限定され、Visual Instrction Set (VIS) や、その他の実装固有の ISA 拡張機能は含まれません。 • このオプションを使用すると、コンパイラは、V8plus ISA 上で良好なパフォーマンスが得られるようにコードを生成することができます。 • 作成されるオブジェクトコードは、SPARC -V8 + ELF32 形式であり、Solaris UltraSPARC 環境でだけ動作します。すなわち、V7 または V8 プロセッサ上では動作しません。 例：UltraSPARC チップアーキテクチャに基づく任意のシステム
v8pplusa	SPARC-V9 ISA のV8plusa バージョン用のコンパイル 定義によれば、V8plusa は、V8plus アーキテクチャを意味しますが、Visual Instruction Set (VIS) バージョン 1.0 および UltraSPARC 拡張機能も含まれます。 • このオプションを使用すると、コンパイラは、UltraSPARC アーキテクチャ上で良好なパフォーマンスが得られるようにコードを生成できますが、V8plus の仕様で定義されている32 ビットのサブセットに限定されます。 • 作成されるオブジェクトコードは、SPARC -V8 + ELF32 形式であり、Solaris UltraSPARC 環境だけ動作します。すなわち、V7 または V8 プロセッサ上では動作しません。 例：UltraSPARC チップアーキテクチャに基づく任意のシステム
v8plusb	UltraSPARC-III 拡張機能付きの SPARC-V8plus ISA の V8plusb バージョン用のコンパイル コンパイラは、Visual Instruction Set (VIS) バージョン 2.0 および UltraSPARC-III 拡張機能付きの UltraSPARC アーキテクチャで良好なパフォーマンスが得られるようにコードを生成することができます。 • 作成されるオブジェクトコードは、SPARC -V8 + ELF32 形式であり、Solaris UltraSPARC-III 環境でしか動作しません。 • このオプションを使用してコンパイルすると、UltraSPARC-III アーキテクチャで良好なパフォーマンスが得られるように、最善の命令セットが使用されます。

表 3-14 SPARC プラットフォーム上で使用できる各 -xarch 値 (続き)

-xarch	意味
v9	SPARC-V9 ISA 用のコンパイル コンパイラは、V9 SPARC アーキテクチャ上で良好なパフォーマンスが得られるようにコードを生成することができます。 • 作成される .o オブジェクトファイルは、ELF64 形式であり、同じ形式の他のSPARC -V9 オブジェクトファイルとだけリンクできます。 • 作成される実行可能プログラムは、64 ビットカーネルの64 ビット目がオンにされた Solaris オペレーティング環境を実行しているUltraSPARCプロセッサ上でだけ実行できます。 • -xarch=v9 が使用できるのは、64 ビット目がオンの Solaris環境でコンパイルする場合だけです。
v9a	UltraSPARC 拡張機能付きの SPARC-V9 ISA 用のコンパイル SPARC -V9 ISA に Visual Instruction Set (VIS) および UltraSPARC プロセッサに固有の拡張機能を追加し、V9 SPARC アーキテクチャ上で良好なパフォーマンスが得られるように、コンパイラがコードを生成できるようにします。 • 作成される .o オブジェクトファイルは、ELF64 形式であり、同じ形式の他のSPARC -V9 オブジェクトファイルとだけリンクできます。 • 作成される実行可能プログラムは、64 ビットカーネルの64 ビット目がオンにされた Solaris オペレーティング環境を実行しているUltraSPARCプロセッサ上でだけ実行できます。 • -xarch=v9a が使用できるのは、64 ビット目がオンの Solaris環境でコンパイルする場合だけです。
v9b	UltraSPARC-III 拡張機能付きの SPARC-V9 ISA 用のコンパイル SPARC -V9 ISA の V9a バージョンに UltraSPARC-III の拡張機能および VIS バージョン 2.0 を追加します。このオプションを指定してコンパイルすると、Solaris UltraSPARC-III 環境で良好なパフォーマンスが得られるように、最善の命令セットが使用されます。 • 作成される .o オブジェクトファイルは、SPARC-V9 ELF64 形式であり、同じ形式の他のSPARC -V9 オブジェクトファイルとだけリンクできます。 • 作成される実行可能プログラムは、64 ビットカーネルの64 ビット目がオンにされた Solaris オペレーティング環境を実行しているUltraSPARC-III プロセッサ上でだけ実行できます。 • -xarch=v9b が使用できるのは、64 ビット目がオンの Solaris オペレーティング環境でコンパイルする場合だけです。

-xautopar

-autopar と同義のオプションです。

- **f77/f95**

-xcache=*c*

オプティマイザ用のキャッシュ特性を定義します。

- **f77/f95**

  c には以下のいずれかを指定します。

  - generic
  - *s1 / l1 / a1*
  - *s1 / l1 / a1:s2 / l2 / a2*
  - *s1 / l1 / a1:s2 / l2 / a2:s3 / l3 / a3*

  si、*li*、*ai* の定義は以下のとおりです。

  *si*レベル *i* のデータキャッシュのサイズ (キロバイト単位)

  *li*レベル *i* のデータキャッシュのラインサイズ (バイト単位)

  *ai*レベル *i* のデータキャッシュの結合性

  このオプションは、オプティマイザが使用できるキャッシュ特性を指定します。特定のキャッシュ特性が必ず使用されるわけではありません。

  このオプションは、-xtarget オプションを展開した機能の一部です。-xtarget オプションで暗黙に指定された -xcache 値の指定を変更する場合に、このオプションを単独で使用します。

例：-xcache=16/32/4:1024/32/1 により次の内容が指定されます。

表 3-15 -xcache の値

値	意味
generic	どの SPARC プロセッサでもパフォーマンスが著しく低下することがないように、キャッシュ特性を定義する。これはデフォルト値です。
s1 / *l1* / *a1*	レベル 1 のキャッシュ特性を定義する。
s1 / *l1* / *a1*:*s2* / *l2* / *a2*	レベル 1 と 2 のキャッシュ特性を定義する。
s1 / *l1* / *a1*:*s2* / *l2* / *a2*:*s3* / *l3* / *a3*	レベル 1、2、3 のキャッシュ特性を定義する。

レベル 1 のキャッシュ： 16K バイト、32 バイト行サイズ、4 面結合

レベル 2 のキャッシュ： 1024K バイト、32 バイト行サイズ、ダイレクトマップ結合

-xcg89

-cg89 と同義です。

- **f77/f95**

-xcg92

-cg92 と同義です。

- **f77/f95**

-xchip=*c*

オプティマイザ用のターゲットプロセッサを指定します。

- **f77/f95**

このオプションは、処理対象となるプロセッサを指定することによって、タイミング特性を指定します。

このオプションは、-xtarget オプションを展開した機能の一部です。-xtarget オプションで暗黙に指定された -xchip 値の指定を変更する場合に、このオプションを単独で使用します。

-xchip=*c* は以下のものに影響を与えます。

- 命令の順序 (スケジューリング)
- 分岐をコンパイルする方法
- 同義の代替命令の選択

表 3-16 に、-xchip の有効な値をまとめてあります。

表 3-16 -xchip の値

値	最適化の対象
generic	ほとんどの SPARC プロセッサ
native	この 32 ビットホストプラットフォーム
old	SuperSPARC 以前のプロセッサ
super	SuperSPARC プロセッサ
super2	SuperSPARC II プロセッサ
micro	MicroSPARC プロセッサ
micro2	MicroSPARC II プロセッサ
hyper	HyperSPARC プロセッサ
hyper2	HyperSPARC II プロセッサ
powerup	Weitek' PowerUP プロセッサ
ultra	UltraSPARC プロセッサ
ultra2	UltraSPARC II プロセッサ
ultra2e	UltraSPARC IIe プロセッサ
ultra2i	UltraSPARC IIi プロセッサ
ultra3	UltraSPARC III プロセッサ

-xcode=*code*

SPARC プラットフォームのコードアドレス空間を指定します。

- **f77/f95**

code の値は以下のとおりです。

`abs32`	32 ビットの絶対アドレスを生成します。コード + データ + bss サイズを 2**32 バイトに制限します。次の 32 ビットのプラットフォームでのデフォルトです。 `-xarch=generic, v7, v8, v8a, v8plus, v8plusa`
`abs44`	44 ビットの絶対アドレスを生成します。コード + データ + bss サイズを 2**44 バイトに制限します。次の 64 ビットのプラットフォームだけで使用できます。 `-xarch=v9, v9a`
`abs64`	64 ビットの絶対アドレスを生成します。次の 64 ビットのプラットフォームだけで使用できます。 `-xarch=v9, v9a`
`pic13`	位置独立コード (スモールモデル) を生成します。-pic と同義です。32 ビットのプラットフォームでは 2**11 の、64 ビットのプラットフォームでは 2**10 の固有の外部シンボルを引用できます。
`pic32`	位置独立コード (ラージモデル) を生成します。-PIC と同義です。32 ビットのプラットフォームでは 2**30 の、64 ビットのプラットフォームでは 2**29 の固有の外部シンボルを引用できます。

デフォルト (`-xcode=`*code* を明示的に指定しない) は次のとおりです。

`-xcode=abs32` SPARC V8 および V7 プラットフォーム
`-xcode=abs64` SPARC および UltraSPARC V9 (`-arch=v9` または `v9a`)

64 ビットの Solaris 7 環境で `-xarch=v9` または `v9a` を使用して共有動的ライブラリを構築する場合、`-xcode=pic13` または `-xcode=pic32` (あるいは `-pic` か `-PIC`) を指定する必要があります。

-xcommonchk[={no|yes}]

共通ブロック不一致の実行時検査を行います。

- **f77/f95**

このオプションは、`TASK COMMON` や並列化を使用しているプログラムで共通ブロックに不一致がないかデバッグ検査を行います。(『Fortran プログラミングガイド』の「並列化」 の章で `TASK COMMON` に関する説明を参照してください。)

デフォルトは -xcommonchk=no です。共通ブロック不一致の実行時検査を行うとパフォーマンスが低下するので、デフォルトではこのオプションは無効になっています。-xcommon=yes はプログラム開発とデバッグのときだけ使用し、生産品質プログラムには使用しないでください。

-xcommonchk=yes でコンパイルすると実行時検査が行われます。1 つのソースプログラム単位で正規の共通ブロックとして宣言されている共通ブロックが TASK COMMON 指令の中で指定されていると、プログラムは停止し、不一致を示すエラーメッセージが出力されます。

例：tc.f における TASKCOMMON 指令の欠如

```
demo% cat tc.f
                common /x/y(1000)
                do 1 i=1,1000
 1              y(i) = 1.
                call z(57.)
                end
demo% cat tz.f
                subroutine z(c)
                common /x/h(1000)
C$PAR TASKCOMMON X
C$PAR DOALL
                do 1 i=1,1000
1                       h(i) = c* h(i)
                return
                end
demo% f95 -c -O4 -parallel -xcommonchk tc.f
demo% f95 -c -O4 -parallel -xcommonchk tz.f
demo% f95 -o tc -O4 -parallel -xcommonchk tc.o tz.o
demo% tc
エラー(libmtsk): threadprivate/taskcommon の宣言の不一致
   x_: tc.f の 1 行目では threadprivate/taskcommon として宣言しません
demo%
```

-xcrossfile[=*n*]

最適化とソースファイル間のインライン化を有効にします。

- **f77/f95**

  指定する場合、*n* には 0、または 1 を指定できます。

  通常、コンパイラが分析する範囲はコマンド行で指定した個別のファイルに限られます。たとえば、-O4 の自動インライン化は、単一のソースファイル中で定義され、引用される副プログラムに対してのみ行われます。

  -xcrossfile を付けると、コンパイラは、コマンド行で指定されたすべてのファイルを分析します。

  -xcrossfile オプションは、-O4 または -O5 を使用している場合のみ有効です。

  ファイル間のインライン化を行うと、ソースファイルどうしの相互依存関係が生まれます。-xcrossfile を指定してコンパイルしたファイルセット中のいずれかのファイルを変更した場合は、新しいコードが正しくインライン化されるように、全ファイルを再コンパイルする必要があります。
  インライン化については 68 ページの「–inline=[%auto][[,][no%]f1,...[no%]fn]」の説明を参照してください。

  デフォルトでは、コマンド行に -xcrossfile を指定しないので、-xcrossfile=0 となり、ファイル間の最適化は行われません。ファイル間の最適化を有効にするには、-xcrossfile (-crossfile=1 と同義) と指定します。

-xdepend

-depend と同義です。

- **f77/f95**

-xexplicitpar

-explicitpar と同義です。

- **f77/f95**

-xF

Sun WorkShop パフォーマンスアナライザにより、関数レベルの並べ替えを行います。

- **SPARC: f77/f95**

  コンパイラ、パフォーマンスアナライザ、リンカーを使用して、コアイメージで関数 (副プログラム) の並べ替えができます。`-xF` オプションでコンパイルすると、アナライザが実行されます。これにより、マップファイルを作成して、関数がどのように使用されるかに応じて、メモリー中の関数の順序を並べ替えることができます。その後、実行可能ファイルを構築するリンクにおいて、 `-M`*mapfile* リンカーオプションを使って、そのマップを使用するように指定することができます。これによって、実行可能ファイルの関数が別々のセクションに配置されます。

  メモリー中の副プログラムの並べ替えは、アプリケーションのテキストページフォルト時間がアプリケーションの実行時間に占める割合が大きい場合にのみ役に立ちます。その他の場合は並べは、アプリケーションの全体的なパフォーマンスは改善されません。パフォーマンスアナライザは、WorkShop に組み込まれています。アナライザについて詳細は、『Sun WorkShop の概要』および『プログラムのパフォーマンス解析』を参照してください。

`-xhasc [={yes|no}]`

ホレリス定数を実際の引数リストの文字列として扱います。

- **f77/f95**

  `-xhasc=yes` を指定すると、コンパイラは、サブルーチンまたは関数でホレリス定数が実際の引数として指定された場合にそれらの定数を文字列として扱います。これはデフォルトであり、Fortran 77 の標準に準拠しています (コンパイラが生成する実際の呼び出しリストには、各文字列の非表示の長さが示されます)。

  `-xhasc=no` を指定すると、ホレリス定数は副プログラムの型なしの値として扱われ、それらの値のアドレスだけが実際の引数リストに配置されます (副プログラムに渡される実際のコールリストに文字列の長さは示されません)。

  ホレリス定数で副プログラムが呼び出され、呼び出された副プログラムが引数を `INTEGER` (または `CHARACTER` 以外) と予測する場合、ルーチンを `-xhasc=no` でコンパイルします。

例：

```
demo% cat hasc.f
                call z(4habcd, 'abcdefg')
                end
                subroutine z(i, s)
                integer i
                character *(*) s
                print *, "string length = ", len(s)
                return
                end
demo% f77 -o has0 hasc.f
demo% has0
 string length =   4     <-- should be 7
demo% f77 -o has1 -xhasc=no hasc.f
demo% has1
 string length =   7   <-- now correct length for s
```

z への 4habcd の受け渡しは、-xhasc=no でコンパイルすることにより、正しく行われます。

このフラグは、旧式の Fortran プログラムの移植を支援するために提供されています。

-xhelp=*h*

オプションに関するヘルプ情報または README ファイルを表示します。

- **f77/f95**

h には readme か flags のいずれかを指定します。

-xhelp=readme : 本リリースのコンパイラに関する README ファイルの内容を表示します。
-xhelp=flags : コンパイラフラグ (オプション) を表示します。-help と同義です。

-xia[=*v*]

区間演算処理を有効化し、適切な浮動小数点環境を設定します。

- **f95**

v には、widestneed または strict のいずれかを指定します。指定しない場合のデフォルトとして widestneed が仮定されます。

Fortran 95 で拡張された区間演算の詳細は、『Fortran 95 区間演算プログラミングリファレンス』に記載されています。112 ページの「-xinterval[=v]」 を参照してください。

-xia フラグは、次のように展開されるマクロです。

-xia または -xia=widestneed	-xinterval=widestneed -ftrap=%none -fns=no -fsimple=0
-xia=strict	-xinterval=strict -ftrap=%none -fns=no -fsimple=0

-xild{off|on}

インクリメンタルリンカーを使用可能/不可能にします。

- **f77/f95**

-xildoff は、インクリメンタルリンカー ild を使用不可能にします。標準リンカー ld が代わりに使用されます。-xildon は、ld の代わりに ild を使用可能にします。

-g オプションを使用しない場合、-xildoff がデフォルトになります。また、-G を使用するかコマンド行で任意のソースファイルを指定する場合も、-xildoff がデフォルトになります。

-G を使用せず -g を使用し、コマンド行にソースファイルを指定しない場合 (オブジェクトファイルおよび/またはライブラリのみ)、-xildon がデフォルトになります。

『C ユーザーズガイド』で ild の節を参照してください。

-xinline=*list*

-inline と同義です。

- **f77/f95**

-xinterval[=*v*]

区間演算処理を有効化します。

- **f95**

v には、no、widestneed または strict のいずれかを指定します。指定しない場合のデフォルトは widestneed です。

no	区間演算処理を有効にしません。
widestneed	モードが混在した式に含まれる非間隔変数および定数を、式の中でもっとも広い間隔のデータ型に変換します。
strict	型や長さが混在した間隔式の使用を禁止します。間隔型および長さの変換はすべて明示的に行わなければなりません。

Fortran 95 で拡張された区間演算の詳細は、『Fortran 95 区間演算プログラミングリファレンス』に記載されています。111 ページの「-xia[=v]」 も参照してください。

-xipo[={0|1}

手続き間の最適化を実行します。

- **f77/f95**

内部手続き解析パスを呼び出すことにより、プログラム全体の最適化を実行します。-Xcrossfile と異なり、-xipo はリンク処理においてすべてのオブジェクトファイルに最適化を実行します。コンパイルコマンドのソースファイルだけに限定されません。

-xipo は、大きなマルチファイルアプリケーションをコンパイルおよびリンクする際に便利です。このフラグでコンパイルされたオブジェクトファイルは、それらのファイル内でコンパイルされた解析情報を保持します。これらの解析情報は、ソースおよびコンパイル前のプログラムファイルで内部手続き解析を可能にします。ただし、解析と最適化は、-xipo でコンパイルされたオブジェクトファイルに限られ、ライブラリのオブジェクトファイルまで拡張できません。

-xipo=0 は内部手続き解析を無効にし、-xipo=1 は有効にします。デフォルトは -xipo=0 であり、-xipo が値を伴わずに指定された場合、-xipo=1 が使用されます。

コンパイルとリンクを個別に実行する場合、-xipo をコンパイルとリンクの両方で指定しなければなりません。

単一のコンパイル/リンク処理での -xipo の使用例：

```
demo% f95 -xipo -xO4 -o prog  part1.f part2.f part3.f
```

最適化プログラムは、3 つのソースファイルすべてに対しファイル相互のインライン化を実行します。これは最終的なリンクステップで実行されるため、すべてのソースファイルのコンパイルを単一のコンパイル処理で実行する必要はありません。-xipo を随時指定することにより、個別のコンパイルが多数発生してもかまいません。

個別のコンパイル/リンク処理での -xipo の使用例：

```
demo% f95 -xipo -xO4 -c part1.f part2.f
demo% f95 -xipo -xO4 -c part3.f
demo% f95 -xipo -xO4 -o prog  part1.o part2.o part3.o
```

コンパイルステップで作成されるオブジェクトファイルは、それらのファイル内でコンパイルされる追加の分析情報を保持します。そのため、リンクステップにおいてファイル相互の最適化を実行できます。

ここでの制限事項は、-xipo でコンパイルを実行しても、ライブラリがファイル相互の内部手続き解析に含まれない点です。次の例を参照してください。

```
demo% f95 -xipo -xO4 one.f two.f three.f
demo% ar -r mylib.a one.o two.o three.o
...
demo% f95 -xipo -xO4 -o myprog main.f four.f mylib.a
```

ここで、one.f、two.f、および three.f の間、および main.f と four.f の間で内部手続きの最適化が実行されますが、main.f または four.f および mylib.a のルーチンのあいだでは内部手続きの最適化が実行されません (初回コンパイルで未定義のシンボルについて警告が発せられることがありますが、コンパイルとリンクの作業であるために内部手続きの最適化は実行されます)。

-xipo に関するその他の重要な情報：

- 少なくとも最適化レベル -x04 を必要とします。
- -xcrossfile と競合します。両方を使用した場合、コンパイルエラーが発生します。
- -xipo なしでコンパイルされたオブジェクトは、-xipo でコンパイルされたオブジェクトと自由にリンクできます。

- -xipo オプションは、ファイル間の最適化に必要な追加情報を格納するために、非常に大きなオブジェクトファイルを作成します。ただし、この追加情報は、最終的な実行可能ファイルの一部にはなりません。実行可能プログラムのサイズが拡大する原因は、最適化の追加実行にあります。
- このリリースにおいて、ファイル相互の副プログラムのインライン化は、-xipo で実行される唯一の内部手続きの最適化です。

-xl[d]

使用できる VMS Fortran 機能を追加します。

- **f77/**

-xl : コンパイラがより多くの VMS Fortran 機能を使用できるようになります。これは、-vax=misalign をマクロ化したもので、以下のような言語機能を提供します。94 ページの「-vax=v」の説明を参照してください。

特別なオプションを指定しなくても、ほとんどの VMS 機能は f77 で自動的に有効になりますが、VMS 機能によっては -xl オプションを指定する必要があります。

通常は、ソース文が VMS、f77、または f95 のいずれにも解釈できるときに、VMS として解釈させたい場合に -xl オプションを指定します。-xl オプションを指定すると、コンパイラは VMS として解釈します。

以下に示す VMS の言語機能は、このオプションを指定して有効にします。

- 書式なし記録のバイトではなくワードでのサイズ (-xl)
- VMS 形式の論理ファイル名 (-xl)
- 引用符文字 (") で始まる 8 進定数 (-xl)
- 文字定数内の本来の文字としてのバックスラッシュ (\) 文字 (-xl)
- PARAMETER 文の非標準書式 (-xl)
- VMS としての構造体の境界整列 (-xl)
- 注釈行あるいは Fortran 文としてのデバッグ文 (-xld)

VMS の構造体がどのように実装されているかを、プログラムが認識できる場合に、-xl を使用して VMS の境界整列を行なってください。

-xld を使用してデバッグ用の注釈 (D または d が 1 カラム目にある) をコンパイルさせます。-xld オプションを省略すると、注釈として扱われます。
(-xl と d の間には空白文字を入れないでください。)

C ルーチンと構造体を共有するプログラムでは、-xl は使用しません。

VMS ライブラリについての詳細は、『Fortran ライブラリ・リファレンス』を参照してください。F77 コンパイラが自動認識する『FORTRAN 77 言語リファレンス』の、VMS 言語拡張機能に関する記述も参照してください。

-xlang=pl

- **f95**

プログラミング言語 pl の実行時ライブラリを使用してリンク処理の準備を整えます。

f95 の場合、-xlang=f77 だけが許可されます。

f95 -xlang=f77 は、f77compat ライブラリを伴うリンクを暗示し、Fortran 95 オブジェクトファイルと Fortran 77 オブジェクトファイルのリンクを容易にします。このリンクにより、適切な実行環境が保証されます。

f95 および f77 のコンパイル済みオブジェクトを単一の実行可能ファイルにリンクする際に、f95 -xlang=f77 を使用します。

-xlibmil

-libmil と同義です。

- **f77/f95**

-xlibmopt

最適化された数学ルーチンを使用します。

- **f77/f95**

このオプションによって通常は高速なコードが生成されます。結果が若干異なる場合がありますが、このときは普通は最終ビットが違っています。このライブラリオプションをコマンド行で指定する順序には意味はありません。

-xlic_lib=*sunperf*

サンのパフォーマンスライブラリとリンクします。

- **f77/f95**

リンクするライブラリをコンマで区切って指定します。たとえば、次のとおりです。

```
f77 -o pgx -fast pgx.f -xlic_lib=sunperf
```

-l オプションと同様に、このオプションもコマンド行中でソースファイルおよびオブジェクトファイルの名前をすべて並べた後に指定します。

このオプションを使用して、サンのパフォーマンスライブラリとリンクさせなければなりません (『Sun Performance Library User's Guide』を参照してください)。

-xlicinfo

ライセンスサーバーの情報を表示します。

- **f77/f95**

このオプションはライセンスのあるシステムに関するライセンス情報を返します。特に、ライセンスサーバーの名前とライセンスをチェックアウトしている個々のユーザーの ID を返します。

通常、このオプションではコンパイルが行われず、ライセンスは獲得されません。このオプションは、他のオプションと一緒には使用されません。しかし、衝突するオプションが使用されると、コマンド行の最後のオプションが選択され、警告メッセージが出力されます。

-xloopinfo

-loopinfo と同義です。

- **f77/f95**

-xmaxopt[=*n*]

最適化プラグマを有効にして、最大最適化レベルを設定します。

- **f77/f95**

n には 1 ～ 5 の値を指定でき、それぞれ最適化レベル -O1 ～ -O5 に対応しています。指定しない場合、コンパイラは 5 を使用します。

このオプションは、C$PRAGMA SUN OPT=*n* 指令がソース入力に指定されている場合にその指令を有効にします。このオプションを指定しないと、コンパイラはこれらの指令行を注釈として解釈します。

このようなプラグマ指令が -xmaxopt フラグの最大レベルを超える最適化レベルで指定されている場合は、コンパイラは -xmaxopt で設定したレベルを使用します。

-xmemalign[=<*a*><*b*>]

メモリ境界整列の最大値の想定と、境界整列不正データへアクセスした時の振る舞いを指定します。

- **f77/f95**

境界整列がコンパイル時に決定できるメモリアクセスの場合、コンパイラは、そのデータ境界整列に適したロード/ストア命令のシーケンスを生成します。

境界整列がコンパイル時に決定できないメモリアクセスの場合、コンパイラは、境界整列を想定して、必要なロード/ストア命令のシーケンスを生成します。

-xmemalign フラグを使用すると、このような曖昧な状況の場合にコンパイラが想定するデータの最大メモリ境界整列を指定することができます。整列不正データへのメモリアクセスが行われた場合の実行時エラーの振る舞いも指定します。

指定する値は、二種類です。すなわち、数値の境界整列値 <a> と、英数字の振る舞いフラグ <*b*> です。

境界整列値 <*a*> に指定できる値は、次のとおりです。

1 最大で 1 バイトの境界整列を想定します。
2 最大で 2 バイトの境界整列を想定します。
4 最大で 4 バイトの境界整列を想定します。
8 最大で 8 バイトの境界整列を想定します。
16 最大で 16 バイトの境界整列を想定します。

不正境界整列データにアクセスした場合のエラーの振る舞いを表す <b> に指定できる値は、次のとおりです。

i アクセスを解釈し、実行を継続します。
S SIGBUS という信号を発生させます。
F 4 バイト以下の境界整列にだけ SIGBUS 信号を発生させます。

-xmemalign を指定しない場合のデフォルト値は、次のようになります。

- -xarch=generic、v7、v8、v8a、v8plus、v8plusa の場合は、4s
- C および C++ の -xarch=v9、v9a の場合は、8s
- Fortran の -xarch=v9、v9a の場合は、8f

値をまったく指定しない場合の -xmemalign のデフォルト値は、すべてのプラットフォームで 1i です。

-dalign (47 ページ参照) および -misalign (72 ページ) のオプションは、それぞれ次のマクロです。

-dalign は、-xmemalign=8s -aligncommon＝16 のマクロです。
-misalign は、-xmemalign=1i -aligncommon＝1 のマクロです。

-xnolib

-nolib と同義です。

- **f77/f95**

-xnolibmil

-nolibmil と同義です。

- **f77/f95**

-xnolibmopt

高速数学ライブラリを使用しません。

- **f77/f95**

-fast と併用すると、最適化済みの数学ライブラリとリンクされません。

```
f77 -fast -xnolibmopt ...
```

-xO*n*

-O*n*と同義です。

- **f77/f95**

-xopenmp

-openmp と同義です。

- **/f95**

-xpad

-pad と同義です。

- **f77**

-xparallel

-parallel と同義です。

- **f77/f95**

-xpg

-pg と同義です。

- **f77/f95**

-xpp={fpp|cpp}

ソースファイルプリプロセッサを選択します。

- **f77/f95**

デフォルトは -xpp=fpp です。

コンパイラは fpp(1) を使用して、.F または .f95 のソースファイルの前処理を行います。fpp(1) は Fortran 用のプリプロセッサです。旧バージョンでは、標準の C プリプロセッサ cpp が使用されていました。cpp を選択するには、-xpp=cpp を指定します。

-xprefetch[=a[,a]]

UltraSPARC II など、先読み命令をサポートするプラットフォーム上で先読み命令を有効にします。

- **f77/f95**

Fortran PREFETCH 指令の説明については、23 ページを参照してください。

UltraSPARC II など、先読みをサポートするアーキテクチャの先読み命令を有効にします (-xarch=v8plus、v8plusa、v9plusb、v9、v9a、または v9b)。

a は、次の値のいずれかです。

値	意味
auto	先読み命令の自動生成を有効にします。
no%auto	先読み命令の自動生成を無効にします。
explicit	明示的な先読みマクロを有効にします。
no%explicit	明示的な先読みマクロを無効にします。
latx:factor	指定の係数により、先読みからロード、および先読みからストアまでの応答時間を調整します。係数は正の浮動小数点数または整数です。
yes	-xprefetch=yes は -xprefetch=auto, explicit と同義です。
no	-xprefetch=no は -xprefetch=no%auto, no%explicit と同義です。

-xprefetch、-xprefetch=auto、および -xprefetch=yes を指定すると、コンパイラは生成したコードに自由に先読み命令を挿入します。その結果、先読みをサポートするアーキテクチャでパフォーマンスが向上します。

大型のマルチプロセッサで集約的なコードを実行する場合、-xprefetch=latx:factor を使用すると便利です。このオプションは、指定の係数により、先読みからロードまたはストアまでのデフォルトの応答時間を調整するようにコード生成プログラムに指示します。

先読みの応答時間とは、先読み命令を実行してから先読みされたデータがキャッシュで利用可能となるまでのハードウェアの遅延のことです。コンパイラは、先読み命令と先読みされたデータを使用するロードまたはストア命令の距離を決定する際に先読み応答時間の値を想定します。

注 – 先読みからロードまでのデフォルト応答時間は、先読みからストアまでのデフォルト応答時間と同じでない場合があります。

コンパイラは、幅広いマシンとアプリケーションで最適なパフォーマンスを得られるように先読み機構を調整します。しかし、コンパイラの調整作業が必ずしも最適であるとは限りません。メモリに負担のかかるアプリケーション、特に大型のマルチプロセッサでの実行を意図したアプリケーションの場合、先読みの応答時間の値を引き上

げることにより、パフォーマンスを向上できます。この値を引き上げるには、1 よりも大きな係数を使用します。1.5 から 2.0 までの値が、パフォーマンスの最大化に最も効果的であると考えられます。

データセットが全体的に外部キャッシュに常駐しているアプリケーションの場合、先読みの応答時間の値を引き下げることにより、パフォーマンスを向上できます。値を引き下げるには、1 よりも小さい係数を使用します。

-xprefetch=latx:factor オプションを使用するには、1.0 に近い係数から始めて、アプリケーションに対するパフォーマンステストを実行します。その後、テストの結果に応じて係数を増減し、パフォーマンステストを再実行します。係数の調整を継続し、最適なパフォーマンスに到達するまでパフォーマンステストを実行します。係数を小刻みに増減すると、しばらくはパフォーマンスに変わりがなく、唐突に異変が発生し、再び平常に戻ります。

デフォルト

-xprefetch が指定されていない場合、-xprefetch=no%auto, explicit とみなされます。

-xprefetch だけが指定される場合、-xprefetch=auto, explicit とみなされます。

デフォルト no%auto は、-xprefetch に引数を指定しないか、または auto か yes を引数にとる -xprefetch を使用して明示的に無効にするまで継続されます。たとえば、-xprefetch=explicit は -xprefetch=explicit, no%auto と同義です。

デフォルト explicit は、引数に no%explicit か no を指定して明示的に無効にするまで継続されます。たとえば、-xprefetch=auto は -xprefetch=auto, explicit と同義です。

-xprefetch または -xprefetch=yes などで自動先読みを有効にしても応答時間の係数が指定されていない場合、-xprefetch=latx:1.0 とみなされます。

相互作用

-xprefetch=explicit を指定すると、コンパイラは次の指令を認識します。

```
$PRAGMA SPARC_PREFETCH_READ_ONCE (name)
$PRAGMA SPARC_PREFETCH_READ_MANY (name)
$PRAGMA SPARC_PREFETCH_WRITE_ONCE (name)
$PRAGMA SPARC_PREFETCH_WRITE_MANY (name)
```

-xchip 設定は、デフォルトの応答時間、つまり latx:factor 設定の結果に影響します。

latx:factor サブオプションは、自動先読みが実行可能な場合に限り有効となります。つまり、yes または auto とともに使用されない限り、latx:factor は無視されます。

警告

明示的な先読みは、測定値によってサポートされた特殊な環境でのみ使用すべきです。

コンパイラは幅広いマシンやアプリケーションで最適なパフォーマンスを得られるように先読み機構を調整しているため、パフォーマンステストで明確な利点のある場合に限り、-xprefetch=latx:factor を使用すべきです。デフォルトの先読み応答時間はリリースにより変更される場合があります。そのため、新しいリリースに切り替えるたびに、応答時間の係数によるパフォーマンスの影響をテストすることをお勧めします。

-xprofile=*p*

実行時のプロファイル用データを収集、またはそのデータを使用して最適化を行います。

- **f77/f95**

p には collect[:*name*]、use[:*name*]、tcov のいずれかを指定します。最適化レベルは、-O2 以上を指定してください。

collect [:*name*]

後に -xprofile=use オプションを使用してプログラムをコンパイルする時に、オプティマイザが使用する実行頻度データを収集し保存します。コンパイラは、文の実行頻度を測定するコードを生成します。

name には、解析対象の実行可能ファイル名を指定します。この名前の指定は省略することができます。*name* が指定されていない場合は、`a.out` が実行可能ファイル名とみなされます。

`-xprofile=collect:` *name* でコンパイルしたプログラムは、デフォルトでは、実行時に `name.profile` サブディレクトリを作成し、実行時フィードバック情報が保存されます。サブディレクトリのファイル `feedback` に実行時のプロファイルデータが書き込まれます。プログラムを複数回実行すると、実行頻度データは `feedback` ファイルに蓄積されていくので、以前の実行頻度データは失われません。

環境変数 `SUN_PROFDATA` および `SUN_PROFDATA_DIR` を設定すると、`-xprofile=collect` でコンパイルされたプログラムが実行時のプロファイルデータを書き込むためのファイルおよびディレクトリを指定できます。これらの変数を設定すると、`-xprofile=collect` でコンパイルされたプログラムは、プロファイルデータを `$SUN_PROFDATA_DIR/$SUN_PROFDATA` に書き込みます。

これらの環境変数は、`tcov` で書き込まれたプロファイルデータファイルのパスと名前も指定します。`tcov`(1) マニュアルページを参照してください。

`use`[:*nm*]

実行頻度データを使用します。

`collect:`nm の場合と同じように nm には、実行可能ファイル名を指定します。名前の指定は省略することもできます。

`-xprofile=collect` オプションを付けてコンパイルした時に生成され、`feedback` ファイルに保存された実行頻度データを使用して、プログラムが最適化されます。

ソースファイル、その他のコンパイルオプションは、`feedback` ファイルが生成されるプログラムをコンパイルした時に使用したのと同じものを指定してください。`-xprofile=collect:`nm でコンパイルするときと、`-xprofile=use:`*nm* で最適化コンパイルをするときのプログラム名は同一のものを指定する必要があります。

`tcov`

新しい形式の `tcov` を使用して、基本的なカバレージ分析を行います。

コード計測方法は `-a` と似ていますが、各ソースファイルごとに `.d` ファイルは生成されません。最終的な実行可能ファイルに基づく名前のファイルが 1 つ生成されます。たとえば、実行可能ファイルが `stuff` である場合、`stuff.profile/tcovd` がデータファイルとなります。

tcov を実行する場合、-x オプションを付けて、新しい形式のデータを使用するよう指示する必要があります。-x オプションを指定しないと、tcov はデフォルトでは古い .d ファイルがあればそれをデータとして使用し、予測外の結果が生成されます。

-a とは異なり、TCOVDIR 環境変数はコンパイル時には影響しませんが、プログラムの実行時に使用され、プロファイルデータを置くサブディレクトリの位置が決まります。

詳細は、tcov(1) マニュアルページ、『Fortran プログラミングガイド』の第 8 章「パフォーマンスプロファイリング」および『プログラムのパフォーマンス解析』を参照してください。

注 – tcov で作成されたレポートは、-O4 または -inline によって副プログラムがインライン化されている場合、信頼性がない可能性があります。インライン化されているルーチンへの呼び出しの範囲は記録されません。

-xrecursive

RECURSIVE 属性をもたないルーチンが自分自身を再帰的に呼び出せるようにします。

- **f95**

RECURSIVE 属性で定義された副プログラムは、-xrecursive を使用してコンパイルされていない場合に限って、自分自身を再帰的に呼び出すことができます。

-xrecursive を使用してコンパイルすると、パフォーマンスが低下する可能性があります。-xrecursive は単独ではメモリスタック上に局所変数を割り当てないので、-xrecursive と -stackvar の併用も検討してください。

-xreduction

-reduction と同義です。

- **f77/f95**

-xregs=*r*

使用するレジスタを指定します。

- **f77/f95**

r には、以下の 1 つまたは複数の項目をコンマで区切って指定します。

```
[no%]appl, [no%]float
```

no には % を付けてください。

例 : -xregs=appl,no%float

デフォルトは -xregs=appl,float です。

- appl :アプリケーションレジスタを使用できるようにします。

  SPARC システムでは、ある特定のレジスタをアプリケーションレジスタと呼びます。これらのレジスタを使用すると必要なロードおよびストア命令が少なくてすむため、パフォーマンスが向上します。ただし、アセンブリコードで記述された古いライブラリプログラムとの間で衝突が起きることがあります。

  アプリケーションレジスタの組み合わせは、SPARC プラットフォームによって異なります。

  - -xarch=v8 または v8a の場合 – レジスタ %g2、%g3 および %g4
  - -xarch=v8 または v8a の場合 – レジスタ %g2、%g3 および %g4
  - -xarch=v8plus または v8plusa の場合 – レジスタ %g2、%g3 および %g4
  - -xarch=v9 または v9a の場合 – レジスタ %g2 および %g3

- no%appl :appl レジスタを使用しません。
- float :SPARC ABI に指定されているように浮動小数点レジスタを使用できるようにします。
  これらのレジスタは、プログラムに浮動小数点のコードがない場合でも使用することができます。
- no%float : 浮動小数点レジスタを使用しません。
  このオプションを使用すると、コンパイラは浮動小数点レジスタを浮動小数点演算の用途にのみ使用し、共通ブロックや構造体のコピーのためには使用しません。

デフォルトは -xregs=appl,float です。

-xs

オブジェクトファイル (.o) がなくても dbx によってデバッグを実行できるようにします。

- **f77/f95**

実行可能ファイルを別のディレクトリに移動した場合でも、オブジェクトファイル (.o) を無視してそのまま dbx を使用することができます。このオプションは、.o ファイルを残しておくことができない場合に使用します。

- コンパイラは、-s オプションをアセンブラに渡し、リンカーは dbx 用のシンボルテーブルをすべて実行可能ファイル内に配置します。
- このようにシンボルテーブルを処理する方法は旧式の方法です。この方法を非自動読み込みと呼ぶことがあります。
- リンカーによるリンクや dbx による初期化が遅くなります。

-xs を付けずに実行可能ファイルを移動する場合は、ソースファイルとオブジェクト (.o) ファイルの両方を移動するか、あるいは dbx の pathmap コマンドか use コマンドのいずれかでパスを設定する必要があります。

- これはシンボルテーブルを読み込む新しいデフォルトの方法です。この方法を自動読み込みと呼ぶことがあります。
- シンボルテーブルは、必要な場合にのみ dbx がシンボルテーブル情報を読み込むように、.o ファイル中に配置されます。したがって、リンク処理と、dbx による初期化が高速で実行されます。

-xsafe=mem

コンパイラは、メモリ保護の違反が発生していないことを想定できます。

- **f77/f95**

このオプションを使用する場合、コンパイラはメモリーに関するトラップが発生しないことを前提とします。SPARC V9 プラットフォーム上で投機的なロード命令を使用することができます。

このオプションは、次のアーキテクチャ (-xarch) を一つ指定して最適化レベル -05 で使用する場合に限り有効です：v8plus、v8plusa、v8plusb、v9、v9a、または v9b。

警告

- アドレスの不正な整列やセグメンテーション違反などの障害発生時、障害のないロードがトラップを引き起こさないため、そうした障害の発生しないプログラムに対してのみこのオプションを使用すべきです。メモリーに関するトラップを引き起

こすプログラムが少ないため、ほとんどのプログラムでこのオプションを安全に使用できます。例外条件を扱うためにメモリーに関するトラップに明示的に依存するプログラムで、このオプションを使用しないでください。

-xsb

-sb と同義です。

- **f77/f95**

-xsbfast

-sbfast と同義です。

- **f77/f95**

-xspace

コードサイズが増大するような最適化は行いません。

- **f77/f95**

コードのサイズが増大するような最適化は行いません。

たとえば、コードのサイズが増大する場合は、ループの展開や並列化は行いません。

-xtarget=*t*

最適化の対象とするプラットフォームを指定します。

- **f77/f95**

命令セットと最適化の対象とするプラットフォームを指定します。

t には native、native64、generic、generic64、プラットフォーム名のいずれかを指定します。

-xtarget オプションは、実際のプラットフォームで発生する、-xarch、-xchip、-xcache をまとめて指定することができます。-xtarget の意味は = の後に指定した値を展開したものにあります。

対象となるハードウェア (コンピュータ) を正確にコンパイラに指定すると、パフォーマンスが向上するプログラムもあります。プログラムのパフォーマンスが重要な場合は、対象となるハードウェアを正確に指定してください。これは特に、新しい SPARC

プロセッサ上でプログラムを実行する場合に当てはまります。ただし、ほとんどのプログラムおよびより旧式の SPARC プロセッサでは、パフォーマンス向上はごくわずかなので、generic を指定することで十分です。

native : ホストプラットフォームに対してパフォーマンスを最適化します。

コンパイラは、ホストプラットフォームに対して最適化されたコードを生成します。コンパイラが動作しているマシンで利用できるアーキテクチャ、チップ、キャッシュ特性が選択されます。

native64: ネイティブの 64 ビット環境向けにコンパイルを実行します。

コンパイラが動作しているマシンの 64 ビット環境向けにアーキテクチャ、チップ、およびキャッシュのプロパティを設定します。

generic : 一般的なアーキテクチャ、チップ、キャッシュに対して最高のパフォーマンスが得られるようにします。

コンパイラは -xtarget=generic を次のように展開します。

```
-xarch=generic -xchip=generic -xcache=generic
```

これはデフォルトの値です。

generic64: 一般的な 64 ビット環境向けにコンパイルを実行します。

-xarch=v9 -xcache=generic -xchip=generic まで拡張されます。

プラットフォーム名 : 指定したプラットフォームに対して最高のパフォーマンスが得られるようにします。

コンパイラで使用できる現在の SPARC プラットフォームの名前 (-xtarget=ultra2i など) については、付録 D「-xtarget プラットフォームの展開」を参照してください。

-xtime

-time と同義です。

- **f77/f95**

-xtypemap=*spec*

デフォルトのデータサイズを指定します。

- **f77/f95**

  デフォルトのデータ型に対するバイトサイズを指定することができます。-dbl と -r8 よりもこのオプションを使用することをお勧めします。このオプションは、デフォルトサイズの変数および定数の両方に割り当てられます。

  指定する文字列 *spec* には、次の全部またはいずれかをコンマで区切ったリストとして指定します。

  ```
  real:size
  double:size
  integer:size
  ```

  指定できるデータのサイズ *size* は、real に対しては 64、double に対しては 128、integer に対しては 32、64、 mixed です。

  ```
  -xtypemap=real:64,double:64,integer:64
  ```

  このオプションは REAL XYZ (64 ビットの XYZ になる) のように明示的にバイトサイズを指定しないで宣言されたすべての変数に適用されます。また、単精度 REAL 定数はすべて、REAL*8 に変換されます。

  各プラットフォームで使用できる組み合わせは次のとおりです。

  - real:32
  - real:64
  - double:64
  - double:128
  - integer:32
  - integer:64
  - integer:mixed (f77 のみ)

  integer:mixed のマッピングは、8 バイトデータを指定しますが、4 バイト演算だけを行います。これは、f77 でだけ使用できます。integer:64 の使用をお勧めします。

  f77 フラグの -dbl および -r8 のオプションは、-xtypemap で次のように表すことができます。

  - -dbl *same as:* -xtypemap=real:64,double:128,integer:64
  - -r8 *same as:* -xtypemap=real:64,double:128,integer:mixed

  さらに 2 つの可能性があります。

  - -xtypemap=real:64,double:64,integer:mixed
  - -xtypemap=real:64,double:64,integer:64

これらは、デフォルトの REAL と DOUBLE を 8 バイトにします。また、-dbl や -r8 では DOUBLE PRECISION から QUAD PRECISION にできないので、これらを優先して使用します。

INTEGER と LOGICAL は等価として扱われます。また、COMPLEX は 2 つの REAL として扱われます。DOUBLE COMPLEX は DOUBLE と同じように扱われます。

-xunroll=*n*

-unroll=*n* と同義です。

- **f77/f95**

-xvector[={yes|no}]

SPARC ベクトルライブラリ関数を自動呼び出しします。

- **f77/f95**

-xvector=yes と指定すると、コンパイラは、必要に応じて、DO ループ内の数学ライブラリ呼び出しを特定して、同等のベクトル化されたライブラリルーチンの単一呼び出しに変換できます。その結果、ループカウントの大きいループのパフォーマンスが改善されます。

デフォルトは -xvector=no です。-xvector だけを指定すると、デフォルトで -xvector=yes になります。

このオプションは -depend もトリガーします。(コマンド行で -xvector に続けて -nodepend を指定すると、依存解析を取り消すことができます。)

-xvector が指定されている場合は、libmvec と libc ライブラリをロードステップに含めるように、コンパイラはリンカーに自動通知します。ただし、コンパイルとリンクを別々に行う場合は、必要なライブラリを正確に選択するためにリンクステップで -xvector を指定する必要があります。

-xvpara

-vpara と同義です。

- **f77**

-Zlp

ループツールを使用するループパフォーマンスのプロファイル用にコンパイルします。

- **f77/f95**

  このオプションとlooptool はサポートされていません。代わりに Sun WorkShop パフォーマンスアナライザを使用してください。

  詳細については、『プログラムのパフォーマンス解析』および analyzer(1) マニュアルページを参照してください。

-ztext

再配置を伴わない純粋なライブラリを生成します。

- **f77/f95**

  再配置が残っている場合はライブラリを作成しません。

  -ztext の主な目的は、生成されたライブラリが純粋なテキストであるかどうか、すべての命令が位置独立コードであるかどうかを確認することです。したがって、通常は -G および -pic と共に使用します。

  -ztext を指定すると、テキストセグメントに不完全な再配置がある場合、ld はライブラリを構築しません。データセグメントに不完全な再配置がある場合は、ld はライブラリを構築しますが、そのデータセグメントは書き込み可能となります。

  -ztext を指定しない場合、ld は再配置の状況とは無関係にライブラリを構築します。

  このオプションは主に、オブジェクトファイルが -pic を付けて作成されたかどうかが不明な場合に、ソースファイルとオブジェクトファイルの両方からライブラリを作成するときに使用します。

  例 : ソースファイルとオブジェクトファイルの両方からライブラリを作成します。

```
demo% f95 -G -pic -ztext -o MyLib -hMyLib a.f b.f x.o y.o
```

  また、コードが位置独立コードであるかどうかを確認するためにも、このオプションを使用します。-pic を付けずにコンパイルすると、純粋なテキストであるかどうかを確認できます。

例 : -pic を付けない場合は、純粋なテキストであるかどうかを確認します。

```
demo% f95 -G -ztext -o MyLib -hMyLib a.f b.f x.o y.o
```

-ztext を付けてコンパイルしても ld によってライブラリが構築されなかった場合は、-ztext を付けずにコンパイルし直すと ld によってライブラリが構築されます。-ztext を指定した場合に構築が失敗するということは、ライブラリ中に共有不可能な成分があることを示します。ただし、この場合でもその他の成分は共有できるはずですが、パフォーマンスが最高でない可能性もあります。

付録 A

実行時のエラーメッセージ

この付録では Fortran 入出力ライブラリ、シグナルハンドラ、およびオペレーティングシステムが生成するエラーメッセージについて説明します。

オペレーティングシステムのエラーメッセージ

オペレーティングシステムのエラーメッセージには、システムコールの失敗、C ライブラリのエラー、シェルの診断などがあります。システムコールのエラーメッセージは、`intro`(2) に記載されています。Fortran ライブラリを介して行われたシステムコールから、直接エラーメッセージが生成されることはありません。Fortran ライブラリの中にある次に示すシステムルーチンが、C のライブラリルーチンを呼び出し、それがエラーメッセージを生成します。

```
    integer system, status
    status = system("cp  afile bfile")
    print*, "status = ", status
    end
```

このようにすると、次のメッセージが表示されます。

```
cp: ファイルにアクセスできません。
 status =  512
```

シグナルハンドラのエラーメッセージ (f77)

プログラムの実行を開始する前に、Fortran 77 ライブラリは、プログラムを終了させるシグナルとしてシグナルハンドラ (sigdie) を設定します。sigdie は、シグナルの内容を示すメッセージを表示し、待機中の出力をフラッシュし、コアイメージを生成します。

現在のところ、エラーメッセージを生成する算術例外は、ゼロによる INTEGER*2 の除算です。その他の算術例外は無視されます。

次の例のように、SUB サブルーチンに渡されていないパラメータを SUB サブルーチンがアクセスしようとすると、シグナルハンドラのエラーとなります。

```
    CALL SUB()
    END
    SUBROUTINE SUB(I,J,K)
    I=J+K
    RETURN
    END
```

f77 でコンパイルして実行する場合、次のエラーメッセージが表示されます。

```
***  停止 sub 77
***  シグナル 11 SIGSEGV を受け取りました
セグメントエラー
```

Fortran 95 コンパイラは、エラーハンドラを設定しません。

入出力のエラーメッセージ (f77)

Fortran の入出力ライブラリは、この後に列挙するエラーメッセージを生成します。ERR が発生すると、IOSTAT 変数にエラー番号が返されます。

たとえば、次のプログラムでは、書式付き出力用にオープンされたファイルに書式なしの書き込みを行おうとしています。

```
WRITE( 6 ) 1
END
```

この場合、次のエラーメッセージが表示されます。

```
sue: [1003]  unformatted io not allowed
logical unit 6, name 'stdout'
lately:  writing sequential unformatted external IO
```

生成されるエラーメッセージは以下のとおりです。perror(3F) のマニュアルページにも、これらのエラーメッセージが記載されています。

エラー番号が 1000 より小さい場合は、システムエラーです。intro(2) を参照してください。

表 A-1 f77 実行時入出力メッセージ

エラー	メッセージ
1000	format のエラー (error in format) エラーメッセージを調べて、書式エラーの箇所を確認してください。このエラーは、10 レベルを超えて括弧を入れ子にした場合や、極端に長い書式文を使用した場合に発生する可能性があります。
1001	不正な装置番号 (illegal unit number) 論理ユニット番号 0 を閉じるのは不正です。また、負の装置番号は使用できません。装置番号の上限は 231 - 1 です。
1002	書式付き入出力は許可されません (formatted io not allowed) 論理ユニットが、書式なし入出力用に開かれています。
1003	書式なし入出力は許可されません (unformatted io not allowed) 論理ユニットが、書式付き入出力用に開かれています。
1004	直接入出力は許可されません (direct io not allowed) 論理ユニットが、順番探査用に開かれています。または、論理記録長が 0 に指定されています。
1005	順次入出力は許可されません (sequential io not allowed) 論理ユニットが、直接探査入出力用に開かれています。

表 A-1　f77 実行時入出力メッセージ (続き)

エラー	メッセージ
1006	ファイルの backspace ができません (can't backspace file) 論理ユニットに結合されたファイルをシークすることはできず、したがってバックスペースも行えません。そのファイルは tty デバイスまたはパイプの可能性があります。
1007	レコードの先頭を越えています (off beginning of record) 内部入力記録の先頭より前に左タブを設定しようとしています。
1008	ファイルの stat ができません (can't stat file) 当該のファイルに関して、システムがステータス情報を返すことができません。ディレクトリを読み込めない可能性があります。
1009	反復数の後に * がありません (no * after repeat count) 並び指定の入出力の繰り返し回数の後には、空白入れずに * が続いていなければなりません。
1010	レコードの終わりを越えています (off end of record) 書式付きの書き込み、または書式なしの読み込みか書き込みで、論理的な記録の終端を越えようとしました。
1011	<使用せず>
1012	不完全な並び入力 (incomprehensible list input) 並び入力は、宣言で指定したとおりでなければなりません。
1013	空き領域の不足 (out of free space) ライブラリが内部使用のために動的に作成するバッファ用のメモリーが不足しています (プログラムが大きすぎます)。
1014	接続されていない装置 (unit not connected) 論理ユニットが開かれていません。
1015	予期しない文字の読みとり (read unexpected character) ある種の書式変換では、非数値データを処理できません。
1016	不正な論理入力フィールド(illegal logical input field) logical データは T か F でなければなりません。
1017	'new' ファイルが存在します('new' file exists) 既存のファイルを status='new' で開こうとしています。
1018	'old' ファイルが見つかりません(can't find 'old' file) 存在しないファイルを status='old' で開こうとしています。
1019	認識できないシステムエラー(unknown system error) 発生するはずのない何らかの事態が発生しました。

表 A-1　f77 実行時入出力メッセージ (続き)

エラー	メッセージ
1020	シーク可能な条件が必要です (requires seek ability) シークを許可していないファイルに対してシークしようとしています。直接探査、書式なしの順番探査入出力、左タブの設定を伴う入出力処理を行うときは、シーク可能な条件が必要です。
1021	不正な引数 (illegal argument) open および関連する関数に対する引数の一部は、正当性を検査されます。デフォルトではない形式のものがしばしば検査対象となります。
1022	負数の反復数 (negative repeat count) 並びによる入力の繰り返し回数は、正の整数でなければなりません。
1023	装置に対する不正な操作 (illegal operation for unit) 論理ユニットに結合されているデバイスに対しては実行できない入出力処理を行おうとしています。テープの終端やファイルの終端を越えて読み込もうとした場合には、このエラーになります。
1024	<使用せず>
1025	オープン時の指定子が矛盾している (incompatible specifiers in open) 'new' オプションと access='append' を付けるなど、不正なオプションの組み合わせを指定してファイルを開こうとしています。
1026	namelist への不正な入力 (illegal input for namelis) 名前並びの読み込みで、不正なデータ項目があります。
1027	FILEOPT パラメータのエラー (error in FILEOPT parameter OPEN の使用で、FILEOPT の文字列の構文に誤りがあります。
1028	読み取り専用ファイルに対するWRITE (WRITE to readonly file) 論理ユニットが読み取り専用として開かれています。
1029	書き込み専用ファイルに対する READ (READ from writeonly file) 論理ユニットが書き込み専用として開かれています。
1030	数値入力変換中のオーバーフロー (overflow converting numeric input) 対応する入力変数に対し、整数入力データが大きすぎます。
1032	数値入力での指数オーバーフロー (exponent overflow on numeric input) 浮動小数点入力データが大きすぎて、対応する入力変数で表せません。

入出力のエラーメッセージ (f95)

Fortran 95 プログラムで生成されるエラーメッセージは、Fortran 77 プログラムのものと異なります。Fortran 95 でコンパイルおよび実行したプログラムの例を次に示します。

```
demo% cat wf.f
      WRITE( 6 ) 1
      END
demo% f95 -o wf wf.f
demo% wf

 ******  FORTRAN RUN-TIME SYSTEM  ******
 Error 1003:  unformatted I/O on formatted unit
 Location:  the WRITE statement at line 1 of "wf.f"
 Unit:  6
 File:  standard output
Abort
```

f95 メッセージにエラーの生じたソースコードのファイル名と行番号が示されていることから、アプリケーション開発者は、入出力文に ERR= 句を使用して実行時入出力エラーを検出することを検討すべきです。

次の表に f95 で出力される実行時入出力メッセージの一部を示します。

表 A-2　f95 の実行時入出力メッセージ

エラー	メッセージ
1000	書式エラー (error in format)
1001	不正な装置番号 (illegal unit number)
1002	書式なし装置に対する書式付き入出力 (fomatted I/O on unformatted unit)
1003	書式付き装置に対する書式なし入出力 (unformatted I/O on formatted unit)
1004	順番探査装置に対する直接探査入出力 (direct-access I/O on sequential-access unit)
1005	直接探査装置に対する順番探査入出力 (sequential-access I/O on direct-access unit)

表 A-2　f95 の実行時入出力メッセージ (続き)

エラー	メッセージ
1006	装置は BACKSPACE をサポートしません (device does not support BACKSPACE)
1007	レコードの先頭を越えています (off beginning of record)
1008	ファイルの stat ができません (can't stat file)
1009	反復数の後に * がありません (no * after repeat count)
1010	長過ぎる記録
1011	切り捨てエラー
1012	不完全な並び入力 (incomprehensible list input)
1013	空き領域の不足 (out of free space)
1014	接続されていない装置 (unit not connected)
1015	予期しない文字の読みとり (read unexpected character)
1016	不正な論理入力コード (illegal logical input field)
1017	'new' ファイルが存在します ('new' file exists)
1018	'old' ファイルが見つかりません (can't find 'old' file)
1019	認識できないシステムエラー (unknown system error)
1020	シーク可能な条件が必要です (requires seek ability)
1021	不正な引数 (illegal argument)
1022	負数の反復数 (negative repeat count)
1023	チャネルやデバイスに対する不正な操作 (illegal operation for channel or device)
1024	再入可能入出力
1025	オープン時の指定子が矛盾している (incompatible specifiers in open)
1026	namelist への不正な入力 (illegal input for namelist)
1027	FILEOPT パラメータのエラー (error in FILEOPT parameter)

表 A-2　f95 の実行時入出力メッセージ (続き)

エラー	メッセージ
1028	許可されない書き出し (writing not allowed)
1029	許可されない読み取り (reading not allowed)
1030	入力での整数オーバーフロー (integer overflow on input)
1031	入力での浮動小数点オーバーフロー (floating-point overflow on input)
1032	入力での浮動小数点アンダーフロー (floating-point underflow on input)
1051	閉じたデフォルト入力装置 (default input unit closed)
1052	閉じたデフォルト出力装置 (default output unit closed)
1053	接続されていない装置からの直接探査の READ (direct-access READ from unconnected unit)
1054	接続されていない装置への直接探査の WRITE (direct-access WRITE to unconnected unit)
1055	結合していない内部装置 (unassociated internal unit)
1056	内部装置の無効な引用 (null reference to internal unit)
1057	空の内部ファイル (empty internal file)
1058	書式なし装置に対する並び入出力 (list-directed I/O on unformatted unit)
1059	書式なし装置に対する変数群入出力 (namelist I/O on unformatted unit)
1060	内部ファイルの終端を越えて書き出ししようとしました (tried to write past end of internal file)
1061	結合していない ADVANCE 指定子 (unassociated ADVANCE specifier)
1062	ADVANCE 指定子が 'YES' または 'NO' ではありません (ADVANCE specifier is not 'YES' or 'NO')

表 A-2　f95 の実行時入出力メッセージ (続き)

エラー	メッセージ
1063	EOR 指定子が前進入力に対して指定されています (EOR specifier present for advancing input)
1064	SIZE 指定子が前進入力に対して指定されています (SIZE specifier present for advancing input)
1065	負数またはゼロの記録番号 (negative or zero record number)
1066	ファイルに存在しない記録 (record not in file)
1067	破壊された書式 (corrupted format)
1068	結合していない入力変数 (unassociated input variable)
1069	データ編集記述子より多い入出力項目 (more I/O-list items than data edit descriptors)
1070	添字三つ組にゼロの刻み幅 (zero stride in subscript triplet)
1071	DO 形ループにゼロの増分値 (zero step in implied DO-loop)
1072	負数の欄幅 (negative field width)
1073	ゼロ幅の欄 (zero-width field)
1074	文字列編集記述子が入力に用いられています (character string edit descriptor reached on input)
1075	ホレリス編集記述子が入力に用いられています (Hollerith edit descriptor reached on input)
1076	数字列に数字がありません (no digits found in digit string)
1077	指数に数字がありません (no digits found in exponent)
1078	範囲外の桁移動数 (scale factor out of range)
1079	数字が基数と等しいかまたは、基数を越えています (digit equals or exceeds radix)
1080	整数欄に予期しない文字 (unexpected character in integer field)

表 A-2　f95 の実行時入出力メッセージ (続き)

エラー	メッセージ
1081	実数欄に予期しない文字 (unexpected character in real field)
1082	論理欄に予期しない文字 (unexpected character in logical field)
1083	整数値に予期しない文字 (unexpected character in integer value)
1084	実数値に予期しない文字 (unexpected character in real value)
1085	複素数値に予期しない文字 (unexpected character in complex value)
1086	論理値に予期しない文字 (unexpected character in logical value)
1087	文字値に予期しない文字 (unexpected character in character value)
1088	変数群名の前に予期しない文字 (unexpected character before NAMELIST group name)
1089	変数群名がプログラム中の名前と一致しません (NAMELIST group name does not match the name in the program)
1090	変数群の項目に予期しない文字 (unexpected character in NAMELIST item)
1091	変数群の項目名に不揃いの括弧 (unmatched parenthesis in NAMELIST item name)
1092	変数群に存在しない変数 (variable not in NAMELIST group)
1093	変数群の実体名に多すぎる添字 (too many subscripts in NAMELIST object name)
1094	変数群の実体名に不十分な添字 (not enough subscripts in NAMELIST object name)
1095	変数群の実体名にゼロの刻み幅 (zero stride in NAMELIST object name)
1096	変数群の実体名に空の文字配列添字 (empty section subscript in NAMELIST object name)

表 A-2　f95 の実行時入出力メッセージ (続き)

エラー	メッセージ
1097	変数群の実体名に範囲外の添字 (subscript out of bounds in NAMELIST object name)
1098	変数群の実体名に空の文字列 (empty substring in NAMELIST object name)
1099	変数群の実体名に範囲外の部分列 (substring out of range in NAMELIST object name)
1100	変数群の実体名に予期しない成分 (unexpected component name in NAMELIST object name)
1111	結合していない ACCESS 指定子 (unassociated ACCESS specifier)
1112	結合していない ACTION 指定子 (unassociated ACTION specifier)
1113	結合していない BINARY 指定子 (unassociated BINARY specifier)
1114	結合していない BLANK 指定子 (unassociated BLANK specifier)
1115	結合していない DELIM 指定子 (unassociated DELIM specifier)
1116	結合していない DIRECT 指定子 (unassociated DIRECT specifier)
1117	結合していない FILE 指定子 (unassociated FILE specifier)
1118	結合していない FMT 指定子 (unassociated FMT specifier)
1119	結合していない FORM 指定子 (unassociated FORM specifier)
1120	結合していない FORMATTED 指定子 (unassociated FORMATTED specifier)
1121	結合していない NAME 指定子 (unassociated NAME specifier)
1122	結合していない PAD 指定子 (unassociated PAD specifier)

表 A-2　f95 の実行時入出力メッセージ (続き)

エラー	メッセージ
1123	結合していない POSITION 指定子 (unassociated POSITION specifier)
1124	結合していない READ 指定子 (unassociated READ specifier)
1125	結合していない READWRITE 指定子 (unassociated READWRITE specifier)
1126	結合していない SEQUENTIAL 指定子 (unassociated SEQUENTIAL specifier)
1127	結合していない STATUS 指定子 (unassociated STATUS specifier)
1128	結合していない UNFORMATTED 指定子 (unassociated UNFORMATTED specifier)
1129	結合していない WRITE 指定子 (unassociated WRITE specifier)
1130	長さゼロのファイル名 (zero length file name)
1131	ACCESS 指定子が 'SEQUENTIAL' または 'DIRECT'ではありません (ACCESS specifier is not 'SEQUENTIAL' or 'DIRECT')
1132	ACTION 指定子が 'READ', 'WRITE' または 'READWRITE' ではありません (ACTION specifier is not 'READ', 'WRITE' or 'READWRITE')
1133	BLANK 指定子が 'ZERO' または 'NULL' ではありません (BLANK specifier is not 'ZERO' or 'NULL')
1134	DELIM 指定子が 'APOSTROPHE', 'QUOTE', または 'NONE' ではありません (DELIM specifier is not 'APOSTROPHE', 'QUOTE', or 'NONE')
1135	予期しない FORM 指定子 (unexpected FORM specifie)
1136	PAD 指定子が 'YES' または 'NO' ではありません (PAD specifier is not 'YES' or 'NO')
1137	POSITION 指定子が 'APPEND', 'ASIS', または 'REWIND' ではありません (POSITION specifier is not 'APPEND', 'ASIS', or 'REWIND')

表 A-2　f95 の実行時入出力メッセージ (続き)

エラー	メッセージ
1138	RECL 指定子が zero または negative です (RECL specifier is zero or negative)
1139	直接探査ファイルに対して記録長が指定されていません (no record length specified for direct-access file)
1140	予期しない STATUS 指定子 (unexpected STATUS specifier)
1141	接続されている装置に対して 'OLD' でない status が指定されています (status is specified and not 'OLD' for connected unit)
1142	STATUS 指定子が 'KEEP' または 'DELETE' ではありません (STATUS specifier is not 'KEEP' or 'DELETE'
1143	一時ファイルに対して指定した status 'KEEP' (status 'KEEP' specified for a scratch file)
1144	不当な status の値 (impossible status value)
1145	一時ファイルに対してファイル名が指定されました (a file name has been specified for a scratch file)
1146	読み取り中または書き出し中の装置を開こうとしています (attempting to open a unit that is being read from or written to)
1147	読み取り中または書き出し中の装置を閉じようとしています (attempting to close a unit that is being read from or written to)
1148	ディレクトリを開こうとしています (attempting to open a directory)
1149	ファイルはシンボリックリンクで、status が 'OLD' です (status is 'OLD' and the file is a dangling symbolic link)
1150	ファイルはシンボリックリンクで、status が 'NEW' です (status is 'NEW' and the file is a symbolic link)
1151	使用できる一時ファイル名がありません (no free scratch file names)

表 A-2　f95 の実行時入出力メッセージ (続き)

エラー	メッセージ
1152	デフォルト装置に対する指定子 ACCESS='STREAM'(specifier ACCESS='STREAM' for default unit)
1153	デフォルト装置に対するストリーム探査(stream-access to default unit)
1161	装置は REWIND をサポートしていません (device does not support REWIND)
1162	BACKSPACE には読みとり権が必要です (read permission required for BACKSPACE)
1163	直接探査装置に対する BACKSPACE (BACKSPACE on direct-access unit)
1164	バイナリ装置に対する BACKSPACE (BACKSPACE on binary unit)
1165	backspace 中にファイルの終わりになりました (end-of-file seen while backspacing)
1166	ENDFILE には書き込み権が必要です (write permission required for ENDFILE)
1167	直接探査装置に対する ENDFILE (ENDFILE on direct-access unit)
1168	順番または直接探査装置に対するストリーム探査 (stream-access to sequential or direct-access unit)
1169	未接続の装置に対するストリーム探査(stream-access to unconnected unit)
1170	ストリーム探査装置に対する直接探査(direct-access to stream-access unit)
1171	不正な POS 指定子の値(incorrect value of POS specifier)
1172	結合していない ASYNCHRONOUS 指定子(unassociated ASYNCHRONOUS specifier)
1173	結合していない DECIMAL 指定子(unassociated DECIMAL specifier)

表 A-2　f95 の実行時入出力メッセージ (続き)

エラー	メッセージ
1174	結合していない IOMSG 指定子(unassociated IOMSG specifier)
1175	結合していない ROUND 指定子(unassociated ROUND specifier)
1176	結合していない STREAM 指定子(unassociated STREAM specifier)
1177	ASYNCHRONOUS 指定子が 'YES' または 'NO' ではありません。(ASYNCHRONOUS specifier is not 'YES' or 'NO')
1178	ROUND 指定子が 'UP'、'DOWN'、'ZERO'、'NEAREST'、'COMPATIBLE'、または 'PROCESSOR-DEFINED' ではありません。(ROUND specifier is not 'UP', 'DOWN', 'ZERO', 'NEAREST', 'COMPATIBLE' or 'PROCESSOR-DEFINED')
1179	DECIMAL 指定子が 'POINT' または 'COMMA' ではありません。(DECIMAL specifier is not 'POINT' or 'COMMA')
1180	RECL 指定子は、ストリーム探査装置の OPEN 文で許可されていません。(RECL specifier is not allowed in OPEN statement for stream-access unit)
1181	割り付けされている配列を割り付けしようとしています (attempting to allocate an allocated array)
1182	結合していないポインタの割り付け解放 (deallocating an unassociated pointer)
1183	結合していない割り付け済み配列の割り付け解放 (deallocating an unallocated allocatable array)
1184	ポインタにより割り付け配列の割り付け解放 (deallocating an allocatable array through a pointer)
1185	ALLOCATE 文により割り付けされていない実体の割り付け解放 (deallocating an object not allocated by an ALLOCATE statement)
1186	実体の一部の割り付け解放 (deallocating a part of an object)

表 A-2　f95 の実行時入出力メッセージ (続き)

エラー	メッセージ
1187	割り付けより大きな実体の割り付け解放 (deallocating a larger object than was allocated)
1191	配列組み込み関数にわたされた割り付けされていない配列 (unallocated array passed to array intrinsic function)
1192	不正な次元数 (illegal rank)
1193	小さなソースサイズ (small source size)
1194	ゼロの配列サイズ (zero array size)
1195	形状に負の要素 (negative elements in shape)
1196	不正な種別 (illegal kind)
1197	形状不適合の配列 (nonconformable array)
2001	無効な定数、構造体、または名前 (invalid constant, structure, or component name)
2002	生成されていないハンドル (handle not created)
2003	短過ぎる文字引数 (character argument too short)
2004	長過ぎる、または短過ぎる配列引数 (array argument too long or too short)
2005	ファイル、記録、またはディレクトリストリームの終わり (end of file, record, or directory stream)

付録 B

各リリースにおける機能変更

この付録では、f77 と f90 の今回のリリースで追加された機能と、旧リリースから変更された機能について説明します。

Fortran 95 の新機能と変更

この節では、f95 の今回のリリースとそれ以前のリリースに固有の新機能および動作上の変更点についてまとめてあります。

Sun Workshop 6 update 2 における f95 の新機能

以下に、Sun Workshop 6 update 2 でリリースされた Fortran 95 コンパイラにおける新機能と変更された機能を示します。

- ALLOCATBLE 属性の拡張：Fortran 95 規格団体の最近の決定により、ALLOCATABLE 属性で使用できるデータ要素が拡張されました。以前、この属性で使用できるデータ要素はローカルに格納された配列変数に制限されていましたが、現在では次の要素を使用できます。
  - 構造体の配列成分
  - ダミー配列
  - 配列関数の結果

  割り付け要素は、記憶領域に関連付けられているすべての場所 (COMMON ブロックと EQUIVALENCE 文) で使用が禁止されています。割り付け配列成分は SEQUENCE 型になることがありますが、そのような型のオブジェクトは COMMON および EQUIVALENCE で使用できません。付録 C の 151 ページを参照してください。

- Fortran 2000 の VALUE 属性：f95 は VALUE 型の宣言属性を認識します。この属性とともに副プログラムのダミー入力引数を指定すると、実際の引数は「値」によって渡されます。付録 C の 152 ページを参照してください。

- OpenMP 2.0 Fortran API のサポート：f95 は、Fortran 95 の OpenMP 2.0 API の仕様をサポートします。改良項目として、配列の WORKSHARE、REDUCTION、変数の THREADPRIVATE、SINGLE 指令の COPYPRIVATE が挙げられます。

  OpenMP 2.0 の仕様については http://www.openmp.org/specsを参照してください。また、付録 E も参照してください。

- OpenMP ライブラリインタフェース：コンパイラは、OpenMP Fortran ライブラリルーチンへのインタフェースを定義するために、インクルードファイル 'omp_lib.h' およびインタフェースモジュール omp_lib を提供します。付録 E の 211 ページを参照してください。

- 内部手続きの最適化 (-xipo)：この新しいコンパイラフラグは、内部手続き解析パスを呼び出すことにより、プログラム全体の最適化を実行します。-xcrossfile と異なり、-xipo は、リンク処理においてすべてのオブジェクトファイルで最適化を実行し、コンパイルコマンドで指定されたソースファイルに限定されません。-xipo は、大きなマルチファイルアプリケーションをコンパイルおよびリンクする際に便利です。第 3 章の 113 ページを参照してください。

- VAX Fortran の構造体：f77 からのプログラムの移行を支援するために、f95 は VAX Fortran の STRUCTURE 文と UNION 文を受け付けます。これらは、Fortran 95 の「構造型」の先駆けとなるものです。付録 C の 177 ページを参照してください。

- ストリーム入出力：Fortran 2000 に対し提案されているもう 1 つの機能が新しい「ストリーム入出力スキーマ」です。ストリーム入出力は、データファイルを連続したバイトのシーケンスとして扱い、1 から始まる正の整数でアドレスを定義します。ストリーム入出力を使用可能にするには、ACCESS='STREAM' でファイルを宣言します。POS=integer_expression 指定子をともなう READ 文または WRITE 文でファイルの位置を指定します。付録 C を参照してください。

- 大域的なプログラム検査：-Xlist オプションを使用すると、f95 の GPC の外観は f77 に近くなり、サブオプション -Xlistc、-Xlist、-Xlists、-Xlistvn、および -Xlistw[*n*] が含まれます。第 3 章の 96 ページを参照してください。

- Fortran ライブラリインタフェース：f95 は、Fortran ライブラリの正しいデータ型を宣言するために、インクルードファイル system.inc を認識します。すべてのルーチンに文 INCLUDE 'system.inc' を記述してください。戻り値が正しく入力されていることを確認するために、非組み込み Fortran ライブラリルーチンが参照されます。『Fortran ライブラリ・リファレンス』を参照してください。

Sun Workshop 6 update 1 における f95 の新機能

以下に、Sun Workshop 6 update 1 でリリースされた Fortran 95 コンパイラにおける新機能と変更された機能を示します。

- UltraSPARC III のサポート：-xtarget および -xchip オプションは ultra3 を受け付け、コンパイラは UltraSPARC III プロセッサに対応する最適化コードを生成します。第 3 章の 128 ページを参照してください。

- -fast に先読みを追加：-xprefetch フラグが -fast オプションセットに追加されました。-fast は、コンパイルするプラットフォームの実行速度を最大限に引き上げるために、多数の最適化フラグを自動的に設定します。-xprefetch を追加することにより、UltraSPARC II および III の先読み機構を利用し、データ処理ループを使用するコードのパフォーマンスを大幅に向上できます。第 3 章の 55 ページを参照してください。

- int2 組み込みのサポート：Fortran 95 および (Fortran 77) コンパイラは、データ型を 2 バイトの整数に変換するために int2 組み込みをサポートします。組み込みとしての int2 (M=int2(J,2)) は、Fortran 77 の多くのレガシーコードに存在するため、互換性を維持する目的から Fortran 95 コンパイラにも実装されています。ただし、int の方が、標準の Fortran 95 にとって好ましい組み込みです (M=int(J,2))。

- -xlang における言語の混在したリンク：新しい -xlang オプションを使用すると、f77 でコンパイルされたオブジェクトファイルおよびライブラリと f95 のオブジェクトファイルを簡単にリンクできます。-xlang を使用すると、適切な実行環境が保証されます。CC(1) マニュアルページおよび第 3 章の 116 ページを参照してください。

Sun WorkShop 6 における f95 の新機能

以下に、Sun Performance WorkShop 6 でリリースされた Fortran 95 コンパイラにおける新機能と変更された機能を示します。

- 準拠：f95 は、Fortran 95 規格に完全に準拠しています。
- 新コマンド：Fortran 95 のコンパイラは、f95 または f90 のいずれかのコマンドで起動できます。
- 最適化コードのデバック：-g を他のオプションと併用することに関する制約が緩和され、並列化コードおよび -O4 または -O5 を指定した最適化コードを、dbx および Sun WorkShop デバッカを使用してデバックできるようになりました。
- ソースファイル名の拡張子：コンパイラは、.f95 および .f90 というファイル拡張子に加え、.F95 および .F90 という拡張子も受け付けるようになりました。
- 区間演算：今回のリリースでは、区間演算計算を可能にするいくつかの拡張機能を実装しました。詳細は、『Fortran 95 区間演算プログラミングリファレンス』および区間演算の README ファイルを参照してください。
- 配列の最適化の向上：コンパイラは、-O4 および -O5のレベルで配列に関して積極的な最適化を行うようになりました。
- ハイパーリンク診断メッセージ：Sun WorkShop オンラインヘルプは、「構築」ウィンドウに表示される f95 エラー診断の解釈についてオンラインヘルプ情報を提供するようになり、エラーメッセージから書斎なオンラインヘルプへのハイパーテキストリンクを作成します。
- OpenMP：コンパイラは、OpenMP の明示的な並列化指令を受け入れます。OpenMP 仕様は、http://www.openmp.org/ で表示できます。
- Cray 形式の DOALL 並列化指令に AUTOSCOPE が追加されました。
- 新コマンド行オプションおよび変更されたコマンド行オプション
  - -aligncommon は、COMMONブロック内の要素を指定のバイト境界に整列させます。
  - -r8const は、単精度データ定数を REAL*8 に変換します。
  - -xinterval および -xia を使用すると、区間演算展開が有効になります。
  - -xmemalign は、データ要素のメモリ内での一般的な境界整列を指定します。
  - -om=openmp および -openmp を使用すると、OpenMP の明示的並列化指令の固有のコンパイルが可能になります。
  - UltraSPARC 先読み命令を有効にするために -xprefetch の機能拡張が行われ、サブオプションが追加されました。
  - -xrecursive を使用すると、RECURSIVE 属性が指定されていない副プログラムからの再帰的呼び出しが可能になります。
  - -xtypemap では、可能なデータ型指定のセットが拡張されました。
  - -fast が機能拡張され、-O5、-fsimple=2、-xvector=yes および -pad=common を設定できます。
- f95 の並列化機能を使用するには、Sun WorkShop HPC ライセンスが必要です。

f90 2.0 の新機能

Sun WorkShop 5.0 でリリースされた f90 2.0 のコンパイラには、f90 1.2 リリースに対する次の新機能と変更された機能が含まれています。

- 新しいオプション :
  - ほとんどの f77 オプションが f90 で認識されます。
  - -fpover が入出力処理で浮動小数点のオーバーフローを検出します。
  - -xcode=*code* で SPARC プラットフォームのメモリーアドレスモデルを指定します。
  - -xcommonchk で COMMON ブロック宣言に不一致がないか実行時検査を行います。
  - -xprefetch を使用すると、コンパイラは UltraSPARC II プラットフォームで先読み命令を生成できます。
  - -xvector を使用すると、コンパイラは DO ループ内の特定の数学ライブラリ呼び出しをベクトル化された数学ルーチンの単一呼び出しで置き換えることができます。
- 変更されたオプション :
  - -xcrossfile[=*n*] オプションのレベル番号が追加されました。
  - -fns[={yes|no}] オプションの yes/no が追加されました。
  - -Ztha 現在ではこのオプションは無視されます。
- 新しい機能 :
  - -xarch=v9 または v9a を使用して 64 ビットの SPARC プラットフォームで 64 ビットの Solaris 7 環境用にコンパイルします。
  - 入出力ライブラリで大型ファイル (2GB 以上) をサポートします。
  - 64 ビットの Solaris オペレーティング環境で大きな配列をサポートします。
  - サン形式の指令をデフォルトとして使用できます。
  - REDUCTION 指令に変数リストの配列を使用できます。
  - SPARC の場合 : TASK COMMON 指令が COMMON の変数を非公開として宣言します。
  - 新しい最適化プラグマでコンパイラの最適化レベルをルーチンごとに設定できます。
- 入出力の相違点 (f90 1.2 と比較した場合)
  - NAMELIST 出力書式 :

1. 2 の場合 :
1つの PRINT 文に含まれる変数はすべて改行なしで 1 行に書き込まれます。

2.0 の場合 :
各変数が別々の行に出力されます。

1.2 の場合 :
値を区切るのにコンマを使用します。

2.0 の場合 :
値を区切るのに空白文字を 1 つ使用します。

1.2 の場合 :
繰り返される値は r* 書式 (3*8.22) を使用して出力されます。

2.0 の場合 :
繰り返される値はすべて明示的に (8.22 8.22 8.22) 出力されます。

1.2 の場合 :
整数の浮動小数点の末尾がゼロなしで出力されます (1.)。

2.0 の場合 :
浮動小数点の整数に末尾のゼロを付けて出力されます (1.0)。

1.2 の場合 :
出力される値が、同じ型の変数として読み取られた値と異なることがあります。読み取り時に 0.1 であった値が 0.100000001 として出力されます。

2.0 の場合 :
書き込まれた値と読み取られた値が同じになるように、必要最小限の桁数で出力されます。0.1 は 0.1 として出力されます。

1.2 の場合 :
標準の要件に従うと、ゼロの値は指数形式で出力されます。ただし、1.2 では 0.E+0 と出力されます。

2.0 の場合 :
ゼロを 0.0E+0 として出力します。

1.2 の場合 :
複素数値のコンマと虚数部との間に空白を出力します (1., 0.E+0)。

2.0 の場合 :
コンマは使用しません (1.0,0.0E+0)。

- NAMELIST 入力書式 :

2.0 では、グループ名の前に $ や & を付けて入力できます。& は Fortran 90 の規格で認められている唯一の書式で、NAMELIST 出力で書き込まれます。

2.0 では、グループの最終データ項目が CHARACTER の場合を除いて、$ を入力の終わりを示すシンボルとして使用できます。これは入力として処理されます。

2.0 では、NAMELIST 入力をレコードの最初の桁から開始することができます。

- PRINT * コンマで区切った出力は廃止しました。
- OPEN FORM='BINARY' は、レコードマークなしで規格外の raw テキストの入出力を行うことができます。FORM='BINARY' でファイルを開いた場合、ファイルにレコード長が埋め込まれないことを除いて、FORM='UNFORMATTED' を使用した場合とほぼ同じ効果が得られます。このデータなしでは、レコードの開始場所と終了場所を特定することはできません。したがって、どの位置まで後退させるのか分からないので、FORM='BINARY' ファイルを BACKSPACE 文で移動することはできません。'BINARY' ファイルに対して読み取り (READ) を実行すると、入力並び上にある変数を満たすのに必要なだけのデータが読み取られます。詳細は、付録 C または『FORTRAN 77 言語リファレンス』を参照してください。
- 異なる装置に対する再帰的入出力が可能です (これは、f90 I/O ライブラリが「MT-Warm」 だからです)。
- 順次書式付き出力、並びによる出力、および NAMELIST 出力でのデフォルトのレコード長は RECL=2147483646 (2^{31}-2 上文字) です (以前のデフォルトは 267)。
- ENCODE と DECODE が、認識、実装されています。方法については、『FORTRAN 77 言語リファレンス』に記載されています。
- 一時ファイルの命名規則は f77 の場合と同じです。
- 以下に示すように、ADVANCE='NO' で非前進入出力を有効にできます。

```
write(*,'(a)',ADVANCE='NO') 'n= '
read(*,*) n
```

- 内部ファイルの入出力の処理は、f90 1.2 よりも Fortran 90 標準に近くなりました。また、内部入出力を行うルーチンを入出力リストで呼び出すことができます。これは、1.2 (または f77) では許可されていませんでした。

- 操作上の相違点 :
  - モジュールの扱い方が異なります。1 つ以上の MODULE 単位を含むソースコードをコンパイルすると、情報ファイル (名前 .mod) がモジュールごとに生成されます。この情報ファイルの名前はモジュールの名前で、.mod 拡張子を付けて小文字で生成されます。.mod ファイルは、モジュールが USE 文で指定される前に使用できなければなりません。つまり、MODULE をUSE 文で参照するファイルをコンパイルする前に、すべての MODULE ファイルがコンパイルされている (およびモジュール情報ファイルも作成されている) 必要があります。
  - -ftrap=common はデフォルトのトラップモードです。
  - Sun Performance Library のルーチンが配列処理を行うように自動リンクされます。
- 新しい言語要素 :
  - Fortran 95 の言語要素がいくつか実装されています。

属性 PURE と ELEMENTAL
MAXVAL と MINVAL の拡張形式

- 新しいデータ型が認識されます。

```
COMPLEX*32   REAL*16
INTEGER*8   (および *1、*2)
LOGICAL*8   (および *1、*2)
```

- いくつかのデータ表現が f90 1.2 以降に変更されています。

INTEGER*2 は、現在では、4 バイトではなく 2 バイトです。
INTEGER*1 は、現在では、4 バイトではなく 1 バイトです。
LOGICAL*2 は、現在では、4 バイトではなく 2 バイトです。
LOGICAL*1 は、現在では、4 バイトではなく 1 バイトです。

これにより、1.2 コンパイラでコンパイルした f90 プログラムで書き込まれたこれらのデータ項目を含むバイナリデータファイルを読み取るプログラムに影響します。これに対処するには、2.0 でコンパイルするときに *1 や *2 ではなく INTEGER*4 または LOGICAL*4 になるように宣言を変更します。

- 値による呼び出し %VAL が f77 と同様に実装されています。ただし、f90 2.0 では、REAL*8 と REAL*16 を double および long double として C ルーチンに渡すことができます。

- f77 および C の f90 2.0 との互換性 :
  - f77 と f90 のオブジェクトバイナリを混在させるには、libF77 ではなく f77 互換ライブラリ libf77compat とリンクします。たとえば、メインプログラムが f77 プログラムの場合でも、f90 ..*files*.. -1f77compat としてリンクを行います。
  - f90 COMMON の構造は、現在では、f77 と互換性があります。
  - f90 スカラーポインタは、C ポインタと互換性があります。

Fortran 77 の新機能と変更点

ここでは、今回の Sun WorkShop 6 update 2 リリースと旧リリースで f77 に固有で新たに追加された機能と変更点について説明します。

Sun Workshop 6 update 2 における f77 の新機能

Sun Workshop 6 update 2 において、f77 に新機能は導入されませんでした。

Sun Workshop 6 update 1 における f77 の新機能

Sun Workshop 6 update 1 において f77 に導入された新機能と変更された機能を次に示します。

- UltraSPARC III のサポート：-xtarget および -xchip オプションは ultra3 を受け付け、コンパイラは UltraSPARC III プロセッサに対応する最適化コードを生成します。第 3 章の 128 ページを参照してください。
- -fast に先読みを追加：-xprefetch フラグが -fast オプションセットに追加されました。-fast は、コンパイルするプラットフォームの実行速度を最大限に引き上げるために、多数の最適化フラグを自動的に設定します。-xprefetch を追加することにより、UltraSPARC II および III の先読み機構を利用し、データ処理ループを使用するコードのパフォーマンスを大幅に向上できます。第 3 章の 55 ページを参照してください

Sun WorkShop 6 での f77 の新機能

次に、Sun WorkShop 6 の Fortran 77 には、次の新機能および変更された機能が含まれています。

- 入出力拡張機能：OPEN (FORM='BINARY') を指定したファイルのオープンは、レコードマークがついていないシーケンシャルバイナリ (未書式化) ファイルとして扱われます。詳細は、『FORTRAN 77 言語リファレンス』を参照してください。
- 最適化コードのデバック：-g を他のオプションと併用することに関する制約が緩和され、並列化コードおよび -O4 または -O5 を指定した最適化コードを、dbx および Sun WorkShop デバッカーを使用してデバックできるようになりました。
- 新コマンド行オプションと変更されたコマンド行オプション
  - -aligncommon は、COMMONブロック内の要素を指定のバイト境界に整列させます。
  - -r8const は、単精度データ定数を REAL*8 に変換します。
  - -xmemalign は、データ要素のメモリ内で一般的な境界整列を指定します。
  - UltraSPARC 先読み命令を有効にするために、-xprefetch の機能拡張が行われ、サブオプションが追加されました。
  - -xtypemap では、可能なデータ型指定のセットが拡張されました。
  - -fast が機能拡張され、-O5、-fsimple=2、-xvector=yes および -pad=common を設定できます。
- f77 の並列化機能を使用するには、Sun WorkShop HPC ライセンスが必要です。
- ハイパーリンク診断メッセージ：Sun WorkShop オンラインヘルプは、「構築」ウィンドウに表示される f77 エラー診断の解釈についてオンラインヘルプ情報を提供するようになり、エラーメッセージから詳細なオンラインヘルプへのハイパーテキストリンクを作成します。

f77 5.0 の新機能

f77 5.0 の新機能と変更点は次のとおりです。

- 新しいオプション :
  - -fpover が入出力処理で浮動小数点のオーバーフローを検出します。
  - -xcode=*code* で SPARC プラットフォームのメモリーアドレスモデルを指定します。
  - -xcommonchk で COMMON ブロック宣言に不一致がないか実行時検査を行います。

- `-xmaxopt` は OPT=n プラグマを有効にし、ソースコードで OPT プラグマが使用できる最大最適化レベルを制御します。
- `-xprefetch` を使用すると、コンパイラは UltraSPARC II プラットフォームで先取り命令を生成できます。
- `-xvector` を使用すると、コンパイラは DO ループ内の特定の数学ライブラリ呼び出しをベクトル化された数学ルーチンの単一呼び出しで置き換えることができます。

- 変更されたオプション :
  - `-xcrossfile[=`*n*`]` オプションのレベル番号が追加されました。
  - `-fns[={yes|no}]` オプションの yes/no が追加されました。
  - `-Ztha` 現在ではこのオプションは無視されます。

- 新しい機能 :
  - `-xarch=v9` または `v9a` を使用して 64 ビットの SPARC プラットフォームで 64 ビットの Solaris 7 環境用にコンパイルします。
  - 入出力ライブラリで大型ファイル (2GB 以上) をサポートします。
  - 64 ビットの Solaris 7 環境で大規模配列をサポートします。
  - 動的配列 (動的サイズのローカル配列) が実装されています (『FORTRAN 77 言語リファレンス』を参照)。
  - `REDUCTION` 指令に変数リストの配列を使用できます。
  - SPARC の場合 : `TASK COMMON` 指令が COMMON の変数を非公開として宣言します。
  - Fortran 90 スタイルの定数でバイトサイズ (64 ビット、8 バイト、定数に対して `12345678_8` など) を指定できます。
  - 新しい最適化プラグマでコンパイラの最適化レベルをルーチンごとに設定できます。
  - 2000 年以降も正しく処理される `date_AND_time( )` ライブラリルーチン

f77 4.2 の新機能

f77 4.0 リリース以降に追加、または変更された f77 4.2 の機能は、次のとおりです。

- 新しいオプション :
  - `-dbl_align_all`
  - `-errtags=yes|no` および `-errofs=`*taglist*
  - `-stop_status=no|yes`
  - `-xcrossfile`

- `-xlic_lib=libs`
- `-xpp=fpp|cpp`
- `-xtypemap=type:spec,.`

- 変更されたオプション :
  - `-fround`、`-fsimple`、`-ftrap`、`-xprofile=tcov`、`-xspace`、`-xunroll` の各オプションが Intel の各プラットフォームで使用できます。
  - `-xtarget`、`-xarch`、`-xchip` の各オプションは、SPARC Ultra、Intel の各プラットフォームでも使用できます。
  - `-vax=` オプションによって、各 VAX/VMS Fortran 機能を有効または無効にすることができます。
  - デフォルトのソースファイルプリプロセッサが、`cpp`(1) から `fpp`(1) に変更されています。

FORTRAN 77 の上位互換

FORTRAN 77 5.0 のソースは、旧リリースと互換性があります。ただし、オペレーティングシステムの変更やバグ修正のために、多少の変更が加えられています。

Fortran 3.0/3.0.1 と 4.0

Solaris 2 上で Fortran 3.0/3.0.1 を使用してコンパイル/リンクした実行可能ファイル (`a.out`)、ライブラリ (`.a`)、およびオブジェクトファイル (`.o`) は、Solaris 2 上で実行する Fortran 5.0 と互換性があります。

Solaris 1 アプリケーションの実行

Solaris 1 上で生成した実行可能ファイルを Solaris 2 上で実行するには、バイナリ互換パッケージ (BCP) をインストールする必要があります。

Solaris 1 でコンパイル/リンクした実行可能ファイルは、Solaris 2 でも実行できます。ただし、Solaris 2.x でコンパイル/リンクした場合に比べ、パフォーマンスは低下します。

Solaris 1 で Fortran 2.0.1 を使用してコンパイル/リンクしたライブラリ (`.a`) とオブジェクトファイル (`.o`) は、Fortran 5.0 とは互換性が保証されません。

技術用語について

本書では、Fortran 特有の用語の和訳については、日本標準規格 Fortran (JIS X 3001-1994)で使用されている用語に準じるよう努めました。ただし、以下に挙げるものについては、他言語との関連性を考慮し、また、WorkShop 製品全体との整合性を優先し、日本標準規格とは異なる訳語を用いています。

表 B-1　JIS X 3001-1994 の訳と本書の訳の違い

英語	JIS X 3001-1994 での和訳	本書での和訳
allocate	割付け	割り付け
ampersand	アンド記号	アンパサンド
assign	割当て	割り当て
assumed	引継ぎ	引き継ぎ
assumed-seize dummy array	擬寸法仮配列	大きさ引き継ぎ仮配列
call	呼出し	呼び出し
error condition	誤り条件	エラー条件
inqury	問合せ	問い合わせ
intrinsic	組込み	組み込み
ising model	イジング模型	イジングモデル
optionally signed integer constant	符号付き又は符号なしの整定数	符号付き、または符号なしの整定数
parameter	パラメタ	パラメータ
procedure	手続	手続き
reading	読込み	読み取り
scalar	スカラ	スカラー
scale factor	けた移動数	桁移動数
scope	有効範囲	スコープ
scoping unit	有効域	スコープ単位

なお、JIS X 3001-1994 で用いられる用語の調査に際しては、駒沢大学の西村和夫先生、ならびに、江戸川大学の高田正之先生にご協力いただきましたことを、感謝いたします。

付録 C

Fortran 95 の機能と相違点

この付録では、以下の Fortran の機能の主な相違点について説明します。

- 標準の Fortran 95 と Sun Fortran 95
- FORTRAN 77 と Fortran 95

機能

Sun WorkShop 6 Fortran 95 は、以下に示す機能を備えています。

継続行の制限

`f95` と `f77` では 99 行まで行を継続することができます (開始行が 1 行とその後の継続行が 99 行ということです)。標準の Fortran 95 では、固定形式の場合で 19 行まで、自由形式の場合で 39 行までです。

固定形式固定形式のソースの行

固定形式のソースの場合、1 行に 73 文字以上使用できます。ただし、73 カラム目以降はすべて無視されます。標準の Fortran 95 では、行の長さは 72 文字までです。

指令

f95 では、指令行が固定形式の CDIR$、!DIR$、CMIC$、C$PRAGMA、C$OMP、または固定形式か自由形式の !DIR$、!MIC$、!$PRAGMA、!$OMP で開始できるようになりました。指令の要約については、付録 E を参照してください。標準 Fortran 95 には、指令の機能はありません。

- f95 のタブ書式の行の長さは 72 カラムです。そのため、次の行に継続する文字列にタブが挿入された場合、予想外の結果を招くことがあります。

```
ソースファイル：
^Iprint *, "Tab on next line
^I1this  continuation line starts with a tab."
^Iend
```

```
コードの実行結果：
Tab on next line                                    this  continuation
 line starts with a tab.
```

使用するソースの書式

f95 で使用するソースの書式は、オプション、指令、拡張子によって異なります。

拡張子が .f または .F のファイルは、固定形式とみなされます。拡張子が .f90、.f95、.F90、または .F95 のファイルは、自由形式とみなされます。

表 C-1 F95 ソース書式のコマンド行のオプション

オプション	作用
-fixed	すべてのソースファイルが Fortran の固定形式で記述されていると解釈します。
-free	すべてのソースファイルが Fortran の自由形式で記述されていると解釈します。

-free および -fixed オプションは、ファイル名の拡張子よりも優先されます。！DIR $ FREE 指令または ！DIR $ FIXED 指令を使用すると、オプションおよびファイル名の拡張子よりも優先されます。

書式の混在

以下のように、異なるソースの書式を混在させてもかまいません。

- 1 つの `f95` のコマンドで、固定形式のソースファイルと自由形式のソースファイルを混在させることができます。
- 指令を使用することによって、1 つのファイル内で、！DIR＄ FREE 指令または！DIR＄ FIXED 指令を使用して自由形式と固定形式を混在させることができます。

大文字・小文字の区別

Sun Fortran 95 では、デフォルトでは大文字・小文字が区別されません。すなわち、`AbcDeF` という変数は、`abcdef` と同じ文字列として扱われます。`-U` オプションを付けてコンパイルすると、コンパイラは大文字と小文字を区別します。

既知の制限

Fortran 95 のプログラム単位には、最高 65,535 個の導出型、および最高 16,777,215 個の定数を定義できます。

ブール (Boolean) 型

`f95` では、ブール型の定数と式をサポートしています。ただし、ブール型の変数、配列、文はサポートしていません。

ブール型に関する規則

- マスク処理 -- ビット単位の論理式ではブール型の結果が生成されます。個々のビットは、対応する演算対象のビットで行われた 1 つまたは複数の論理演算の結果を表します。
- 2 進の算術演算子および関係演算子では、以下のように処理されます。
  - 一方の演算対象がブール型の場合は、そのまま演算が実行されます。
  - 両方の演算対象がブール型の場合は、両者を整数であるとみなして演算が実行されます。
- ユーザー定義の関数によってブール型の結果を生成することはできません。ただし、一部の (標準でない) 組み込み関数では可能です。

- ブール型と論理型には、以下のような相違点があります。
  - 変数、配列、関数は論理型にできますが、ブール型にすることはできません。
  - LOGICAL 文はありますが、BOOLEAN 文はありません。
  - 論理型の変数、定数、または式は、2 つの値 TRUE または FALSE だけしか表しません。ブール型の変数、定数、または式は、任意のバイナリ値を表すことができます。
  - 論理型は、算術式、関係式、ビット単位の論理式で使用できませんが、ブール型はこれらすべてにおいて有効です。

ブール型定数の別の書式

f95 では、ブール型定数 (8 進、16 進、ホレリス) を、以下に示すような書式 (2 進ではありません) で使用することができます。標準の Fortran では、変数をブール型と宣言することはできません。

8 進

書式は *dddddd*B です。*d* は任意の 8 進数です。

- B または b のどちらの文字を使用してもかまいません。
- 1 桁から 11 桁までの 8 進数 (0 から 7) を使用できます。
- 11 桁の 8 進数は 32 ビットの完全なワードを表します。左端の数字は常に 0、1、2、3 のいずれかです。
- 8 進の個々の数字は 3 ビットの値を表します。
- 右端の桁は、右 3 ビット (29、30、31 ビット) の内容を表します。
- 11 桁未満の場合は、値は右揃えされます。ワードの右端にある n ビットから 31 ビットまでが使用され、それ以外のビットはゼロになります。
- 空白は無視されます。

入出力の書式指定では B という文字は 2 進数であることを示しますが、それ以外の場合は 8 進数であることを表します。

16 進

d が任意の 16 進の数字である X′*ddd*′ または X″*ddd*″ の書式です。

- 1 桁から 8 桁までの 16 進数 (0 から 9、A から F) を使用できます。
- 文字は大文字でも小文字でもかまいません (X、x、A から F、a から f)。
- 数字は引用符 (アポストロフィ) または二重引用符で囲まなければなりません。
- 空白は無視されます。
- 16 進数の始めに + か - の記号を付けてもかまいません。
- 8 桁の 16 進数は 32 ビットの完全なワードを表しています。この 32 ビットワードの各ビットの内容は、同じ値を表す 2 進数に対応しています。
- 8 桁未満の場合は、値は右揃えされます。ワードの右端にある *n* ビットから 31 ビットまでが使用され、それ以外のビットはゼロになります。

ホレリス

ホレリスデータには、以下の書式を使用できます。

*n*H...'...'H"..."H

*n*L...'...'L"..."L

*n*R...'...'R"..."R

上記の ... は文字列を表し、*n* は文字数を表します。

- ホレリス定数はブール型です。
- ビット単位の論理式に文字定数がある場合は、その式はホレリスとみなされます。
- ホレリス定数には 4 文字まで使用することができます。

例 : 8 進と 16 進の定数の表現例を示します。

ブール型定数	内部の 8 進数
0B	00000000000
77740B	00000077740
X″ABE″	00000005276

ブール型定数	内部の 8 進数
X″-340″	37777776300
X′1 2 3′	00000000443
X′FFFFFFFFFFFFFFFF′	37777777777

例 : 代入文での 8 進と 16 進の使用例を示します。

```
i = 1357B

j = X″28FF″

k = X′-5A′
```

算術式の中で 8 進数または 16 進数の定数を使用すると、結果が未定義になることがあります。ただし、構文エラーにはなりません。

別の場所におけるブール型定数の使用

f95 では、DATA 文以外の場所で BOZ 定数を使用することができます。

B′*bbb*′O′*ooo*′Z′*zzz*′

B″*bbb*″O″*ooo*″Z″*zzz*″

このような BOZ 定数が実数変数に代入されている場合には、型は変換されません。

標準の Fortran では、BOZ 定数は DATA 文でしか使用できません。

数値データ型のサイズの略記法

f95 では、宣言文、関数文、IMPLICIT 文において、以下のような非標準の書式で型を宣言することができます。1 列目の形式は一般に使用されていますが、非標準の Fortran 95 です。2 列目の種別番号はベンダーにより変わります。

表 C-2　数値データ型のサイズの表記法

非標準	宣言子	短縮書式	意味
INTEGER*1	INTEGER(KIND=1)	INTEGER(1)	1 バイトの符号付き整数
INTEGER*2	INTEGER(KIND=2)	INTEGER(2)	2 バイトの符号付き整数
INTEGER*4	INTEGER(KIND=4)	INTEGER(4)	4 バイトの符号付き整数

表 C-2　数値データ型のサイズの表記法

非標準	宣言子	短縮書式	意味
LOGICAL*1	LOGICAL(KIND=1)	LOGICAL(1)	1 バイト論理型
LOGICAL*2	LOGICAL(KIND=2)	LOGICAL(2)	2 バイト論理型
LOGICAL*4	LOGICAL(KIND=4)	LOGICAL(4)	4 バイト論理型
REAL*4	REAL(KIND=4)	REAL(4)	IEEE の単精度浮動小数点数 (4 バイト)
REAL*8	REAL(KIND=8)	REAL(8)	IEEE の倍精度浮動小数点数 (8 バイト)
REAL*16	REAL(KIND=16)	REAL(16)	IEEE の 4 倍精度浮動小数点 (16 バイト)
COMPLEX*8	COMPLEX(KIND=4)	COMPLEX(4)	単精度複素数 (各部に 4 バイト)
COMPLEX*16	COMPLEX(KIND=8)	COMPLEX(8)	倍精度複素数 (各部に 8 バイト)
COMPLEX*32	COMPLEX(KIND=16)	COMPLEX(16)	4 倍精度複素数 (各部に 16 バイト)

Cray ポインタ

Cray ポインタとは、別の変数や配列などのアドレスを値にもつ変数のことです。この別のもののことを、指示先と呼びます。

f95 は Cray ポインタをサポートしています。標準の Fortran 95 ではサポートされていません。

構文

Cray ポインタの POINTER 文は以下の形式で記述します。

```
POINTER ( ポインタ名, 指示先名 [配列の仕様] ), ...
```

ポインタ名、指示先名、配列の仕様のそれぞれの意味は、以下のとおりです。

ポインタ名	対応する <指示先名> へのポインタです。ポインタ名には指示先のアドレスが含まれます。 制限事項: ポインタ名にはスカラー変数名を指定してください (派生型は指定できません)。 定数、構造体の名前、配列、関数を指定することはできません。
指示先名	対応する <ポインタ名> の指示先です。 制限事項 : 変数名、配列の宣言子、配列名を指定してください。
配列の仕様	配列の仕様を指定する場合は、明示的な実体があるもの (定数または非定数のサイズを持つもの)、または仮のサイズをもつものを指定してください。

例 : 2 つの指示先に対して Cray ポインタを宣言します。

```
    POINTER ( p, b ), ( q, c )
```

上記の例では、Cray ポインタ `p` とその指示先 `b`、Cray ポインタ `q` とその指示先 `c` を宣言しています。

例 : 配列に対して Cray ポインタを宣言します。

```
    POINTER ( ix, x(n, 0:m) )
```

この例では、Cray ポインタ `ix` とその指示先 `x` を宣言しています。同時に、`x` は `n` と `m+1` 次元の配列であることを宣言しています。

Cray ポインタの目的

ポインタを使用すると、記憶領域の特定の場所に変数を動的に対応づけ、ユーザーが管理する記憶領域にアクセスすることができます。

Cray ポインタでは、メモリーの絶対的な場所にアクセスすることができます。

Cray ポインタと Fortran 95 のポインタ

Cray ポインタは次のように宣言します。

`POINTER` （ **ポインタ名**, 指示先名 [配列の仕様] ）

Fortran 95 のポインタは次のように宣言します。

`POINTER` **オブジェクト名**

この 2 種類のポインタを混在させることはできません。

Cray ポインタの機能

- 指示先が引用されるたびに、`f95` はポインタの現在の値を指示先のアドレスとして使用します。
- Cray ポインタ型の文では、ポインタと指示先の両方を宣言します。
- Cray ポインタは Cray 型のポインタです。
- Cray ポインタの値は、32 ビットプロセッサ上で領域の 1 単位を占め、64 ビット SPARC V9 プロセッサ上で領域の 2 単位を占めます。
- Cray ポインタは `COMMON` の並びまたは仮引数で使用することができます。
- Cray ポインタの値が定義されるまでは、指示先にアドレスはありません。
- 指示先として配列が指定されている場合、その配列を指示先配列と呼びます。

この場合の配列の宣言子は以下の場所に指定することができます。

  - 独立した型宣言文
  - 独立した `DIMENSION` 文
  - ポインタ文自体

- 配列の宣言子が副プログラムにある場合、次元の設定は以下の場所で確認できます。
  - 共通ブロックにある変数、または
  - 仮引数である変数
- 各次元のサイズは、指示先が引用されるときではなく、副プログラムの処理が始まるときに認識されます。

Cray ポインタの制限事項

- 指示先名は、CHARACTER*(*) で型宣言された変数であってはなりません。
- 指示先名が配列の宣言子である場合は、明示的な実体があるもの、(定数または非定数のサイズを持つもの)、または仮のサイズをもつものでなければなりません。
- Cray ポインタを配列にすることはできません。
- Cray ポインタを以下のように扱うことはできません。
  - 別の Cray ポインタまたは Fortran のポインタの引用先にする
  - 構造体の成分にする
  - 他のデータ型で宣言する
- Cray ポインタを以下の場所で使用することはできません。
  - PARAMETER 文または PARAMETER 属性を含む型宣言文
  - DATA 文

Cray ポインタの指示先の制限事項

- Cray ポインタの指示先を、SAVE、DATA、EQUIVALENCE、COMMON、PARAMETER 文で使用することはできません。
- Cray ポインタの指示先を仮引数にすることはできません。
- Cray ポインタの指示先を関数値にすることはできません。
- Cray ポインタの指示先を構造体または構造体の成分にすることはできません。
- Cray ポインタの指示先を派生型にすることはできません。

Cray ポインタの使用法

Cray ポインタには以下のようにして値を割り当てることができます。

- 絶対アドレスに設定します。

例 : `q = 0`

- 整数変数の加減式によって割り当てます。

例 : `p = q + 100`

- Cray ポインタは整数ではありません。Cray ポインタを実数変数に割り当てることはできません。
- LOC 関数 (非標準) を使用して Cray ポインタを定義することができます。

例 : `p = LOC ( x )`

例 : Cray ポインタの使用例

```
SUBROUTINE  sub ( n )
COMMON pool(100000)
INTEGER blk(128), word64
REAL a(1000), b(n), c(100000-n-1000)
POINTER ( pblk, blk ), (ia, a ), ( ib, b ), &
    ( ic, c ), ( address, word64 )
DATA address / 64 /
pblk = 0
ia = LOC( pool )
ib = ia + 1000
ic = ib + n
...
```

上記の例を説明します。

- word64 は絶対アドレス 64 の内容を参照します。
- blk はメモリーの最初の 128 ワードを占める配列です。
- a は空白共通ブロックにある配列で、長さは 1,000 です。
- b は a の後に位置し、長さは n です。
- c は b の後に位置します。
- a、b、c は pool 領域に関連付けられています。
- word64 は blk(17) と同じです。Cray ポインタはバイトアドレスであり、blk の整数要素はそれぞれ 4 バイトの長さがあるためです。

その他の言語拡張機能

ALLOCATABLE 属性の拡張機能

Fortran 95 規格団体の最近の決定により、ALLOCATABLE 属性で使用できるデータ要素が拡張されました。以前、この属性はローカルに格納された配列変数に制限されていました。現在では、次の要素を使用できます。

- 構造体の配列成分
- ダミー配列
- 配列関数の結果

割り付け要素は、記憶領域に関連付けられているすべての場所 (COMMON ブロックと EQUIVALENCE 文) で使用が禁止されています。割り付け配列成分は SEQUENCE 型になることがありますが、そのような型のオブジェクトは COMMON および EQUIVALENCE で使用できません。

VALUE 属性 (Fortran 2000)

f95 コンパイラは、VALUE 型の宣言属性を認識します。この属性を Fortran 2000 の規格にすることが提案されています。

この属性とともに副プログラムのダミー入力引数を指定すると、実際の引数は「値」によって渡されます。次の例では、リテラル値を引数とする Fortran 95 副プログラムを呼び出す C 言語の主プログラムにおいて VALUE 属性を使用しています。

```
C コード:
#include <stdlib.h>
int main(int ac, char *av[])
{
    to_fortran(2);
}

Fortran コード:
       subroutine to_fortran(i)
       integer, value :: i
       print *, i
       end
```

ストリーム入出力 (Fortran 2000)

新しい「ストリーム」入出力スキーマは、Fortran 2000 の規格草稿の一部として提案されています。ストリーム入出力探査は、データファイルを連続したバイトのシーケンスとして扱い、1 から始まる正の整数でアドレスを定義できます。ストリーム入出力ファイルは、OPEN 文の ACCESS='STREAM' 指定子で宣言します。バイトアドレスにファイルを配置するには、READ 文または WRITE 文で POS=*scalar_integer_expression* 指定子を使用する必要があります。INQUIRE 文は、ACCESS='STREAM'、指定子 STREAM=*scalar_character_variable*、および POS=*scalar_integer_variable* を受け付けます。

STRUCTURE と UNION (VAX Fortran)

`f77` からのプログラムの移行をサポートするために、`f95` は、VAX Fortran の `STRUCTURE` および `UNION` 文を受け付けます。これらは、Fortran 95 の「構造型」の先駆けです。構文の詳細については、『FORTRAN 77 言語リファレンス』を参照してください。

`STRUCTURE` のフィールド宣言は、次のいずれかになります。

- 基礎構造体 - 別の `STRUCTURE` 宣言、または事前に定義されたレコード
- `UNION` 宣言
- `TYPE` 宣言。初期値を含むこともできます。
- `SEQUENCE` 属性を保持する構造型 (`f95` のみ)

`f77` と同様に、`POINTER` 文をフィールド宣言として使用することはできません。

また、`f95` には次のような拡張機能があります。

- 構造体のフィールド宣言の記号として、'.' または '%' を使用できます (`struct.field` または `struct%field`)。
- 構造体を書式化された入出力文に配置できます。
- 構造体を `PARAMETER` 文で初期化できます。書式は、構造型の初期化と同じです。
- 構造体を構造型の成分として配置できますが、構造型は `SEQUENCE` 属性として宣言する必要があります。

入出力拡張機能

Sun FORTRAN 77 の一部の入出力拡張機能が Fortran 95 のコンパイラに追加されました。

- NAMELIST 入出力形式

  入力時にグループ名の先頭に $ または & が付きます。& は、Fortran 95 の規格だけが受け付ける形式であり、NAMELIST 出力によっても書き込まれます。

  入力の終了を表す記号として $ を受け付けます。ただし、グループ内の最後のデータ項目が CHARACTER データである場合は別です。その場合、$ は、入力データとして扱われます。

  NAMELIST 入力は、レコードの最初のカラムから開始することができます。

- OPEN （...,FORM='BINARY'）は、レコードマークなしのバイナリデータとしてファイルを扱います。

  FORM='BINARY' が指定されたファイルのオープンは、FORM='UNFORMATTED' を指定した場合と同じ効果があります。ただし、レコード長がファイルに埋め込まれません。このデータがないと、どこでレコードが開始し、どこで終了するかを示す手段がありません。したがって、FORM='BINARY' が指定されたファイルに BACKSPACE 操作を行うことはできません。なぜなら、どこまで現在記録を前に位置付けするかを指示する方法がないからです。'BINARY' ファイルに対する READ は、入力リスト上の変数を満たすのに必要なだけのデータを読み込みます。

  - WRITE 文：データは、バイナリ形式のファイルに書き込まれます。出力リストで指定したバイト数が転送されます。
  - READ 文：データは、入力リスト上の変数に読み込まれます。リストで必要なバイト数が転送されます。ファイル上にレコードマークがないので、「レコードの終わり」というエラーが検出される可能性はありません。検出されるエラーは、「ファイルの終わり」またはシステムの異常だけです。
  - INQUIRE 文：FORM="BINARY" を指定してオープンされたファイルに対する INQUIRE では、次のものが返されます。

    ```
    FORM="BINARY"
    ACCESS="SEQUENTIAL"
    DIRECT="NO"
    ```

FORMATTED="NO"
UNFORMATTED="YES"
RECL= AND NEXTREC= は、未定義。

- BACKSPACE 文：使用できません。エラーが返されます。
- ENDFILE 文：通常どおり、現在の位置でファイルが打ち切られます。
- REWIND 文：通常どおり、ファイルをデータの開始に位置付けます。

- さまざまな装置に対し再帰的な入出力が可能です (これは、f95 の入出力ライブラリが「MT-Warm」だからです)。
- RECL=2147483646 (2^{31} -2) は、順番に書式化された、並びによる変数群 出力上のデフォルトのレコード長です。
- ENCODE および DECODE は、『FORTRAN 77 言語リファレンス』で説明するように認識され、実装されています。
- スクラッチファイルの命名方法は、f77 と同じです。
- 次に示すように、ADVANCE='NO' で非前進入出力が可能になります。

```
write(*,'(a)',ADVANCE='NO') 'n= '
read(*,*) n
```

指令

コンパイラ指令は、特別な動作をするようにコンパイラに指示します。プラグマとも呼ばれます。

コンパイラ指令は 1 行または複数行のテキストとしてソースプログラムに挿入されます。コンパイラ指令は一見注釈に似ていますが、注釈にはない特別な文字が付加されています。Fortran 95 以外のほとんどのコンパイラでは指令を注釈として扱うので、コードの一定の移植性は保たれます。

f95 (および f77) では、サン形式の指令がデフォルトとして設定されています。Cray 形式の指令に切り換えるには、コンパイラコマンド行フラグの -mp=cray を使用してください。

Fortran 指令については、付録 E にまとめられています。

f95 の特殊な指令行の書式

f95 は、第 2 章に示すような f95/f77 の一般的な指令に加え、独自の特殊な指令を認識します。構文は次のようになります。

```
!DIR$ d1, d2, ...
```

ソースが固定形式の場合

- CDIR$ または !DIR$ を 1 カラム目から 5 カラム目に記述します。
- 指令を 7 カラム目以降に記述します。
- 73 カラム目以降は無視されます。
- 新たに始まる指令行では、6 カラム目を空白とします。
- 指令の継続行では、6 カラム目に空白以外の文字を記述します。

ソースが自由形式の場合

- !DIR$ の後に空白を 1 つつけて、行の任意の位置に記述できます。!DIR$ 文字は、その行の空白でない最初の文字となります。
- 指令は空白の後に記述します。
- 新たに始まる指令行では、!DIR$ の直後に空白、タブ、または改行が続きます。
- 指令の継続行では、!DIR$ の直後に空白、タブ、改行以外の文字が続きます。

これらのことから、!DIR$ を 1 カラム目から 5 カラム目に記述しておけば、自由形式または固定形式のどちらのソースでも機能することがわかります。

FIXED 指令と FREE 指令

指令行の後に続くソース行の書式を指定します。

範囲

指令が適用される範囲は、ファイル内に指令が出現してから最後までの部分、または次に FREE あるいは FIXED が出現するまでの部分です。

使用法

- 1 つのソースファイル内でソースの書式を切り換えることができます。
- INCLUDE ファイルのソースの書式を切り換えることができます。INCLUDE ファイルの先頭に指令を挿入すると、INCLUDE ファイルが処理された後に、ソースの書式が INCLUDE ファイルの処理前の書式に戻ります。

制限事項

FREE 指令と FIXED 指令には以下の制限事項があります。

- どちらの指令もコンパイラの指令行に単独で指定します (継続行にしないでください)。
- どちらの指令もソースコードの任意の位置に指定できます。その他の指令は作用するプログラム中に指定する必要があります。

例 : FREE 指令を指定します。

```
!DIR$ FREE
    DO i = 1, n
    a(i) = b(i) * c(i)
    END DO
```

並列化の指令

並列化の指令は、コンパイラに次の DO ループの並列化処理を指示する特別な注釈です。これらに関する概要は、付録 E と『Fortran プログラミングガイド』に記載されています。f95 は、f77 の Sun および Cray 形式の並列化指令だけでなく、OpenMP Fortran API 指令も認識します。

注 - Fortran の並列化機能には、Sun WorkShop HPCのライセンスが必要です。

組み込み関数

f95 は標準の処理を拡張した組み込み関数をサポートしています。

表 C-3　非標準の組み込み関数

名前	定義	関数の型	引数	引数	備考
COT	余接関数	実数	実数	([X]=x)	P, E
DDIM	正差	倍精度実数	倍精度実数	([X]=x,[Y=]y)	P, E
LEADZ	先行の 0 ビットの数を調べる	整数	ブール型、整数、実数、ポインタ	([I=]i)	NP, I
POPCNT	設定されたビットの数を調べる	整数	ブール型、整数、実数、ポインタ	([I=]i)	NP, I
POPPAR	ビットの設定数のパリティを演算する	整数	ブール型、整数、実数、ポインタ	([X=]x)	NP, I

上記の表の備考欄の意味は以下のとおりです。

備考	意味
P	名前を引数として渡すことができる
NP	名前を引数として渡すことはできない
E	組み込み関数の外部コードは実行時に呼び出される
I	f95 が組み込み関数のインラインコードを生成する

Fortran 77 との互換性

標準に準拠した固定形式 (*filename*.f) の FORTRAN 77 ソースコードは、Fortran 95 と互換性があります。非標準の拡張機能 (VMS Fortran など) には互換性がないため、f95 でコンパイルできない場合があります。

f95 と f77 の非互換性問題

現行リリースの f95 で f77 をコンパイルおよびテストしたときに生じた非互換性の問題の一部を次に列挙します。これらの問題は、f95 の比較機能の欠如、または動作の相違点が原因となって発生します。次の項目は、Fortran 77 の非標準の拡張機能であり、f77 でサポートされていても f95 でサポートされていません。

- 入出力 (178 ページおよび 184 ページも参照してください)
  - 並び出力が異なる形式を使用しています。
  - 変数の書式が Fortran 95 で利用できません。
  - Fortran 95 では、ACCESS='APPEND' のファイルを開くことができません。
  - f95 は、直接探査ファイルで BACKSPACE または ENDFILE を許可しません。
  - Fortran 95 には、形式編集記述子でフィールド幅を明示的に指定する必要があります。たとえば、FORMAT(I) は許可されません。
  - f95 は、出力形式で f77 のエスケープシーケンス (\n \t \') を認識しません。
  - f95 は、OPEN 文の FILEOPT= を認識しません。
  - f95 は、直接探査入出力でレコード番号を指定する 'n 書式 (READ (2 '13) X,Y,Z) を認識しません。
- データ型、宣言、および用法
  - プログラムユニットにおいて、Fortran 95 の IMPLICIT 文がユニットの最初の宣言文に先行しなければなりません。
  - f95 では 7 つしか配列添字を使用できません。f77 では 20 個まで使用できます。
  - Fortran 95 では、LOGICAL 変数と INTEGER 変数を置き換えて使用できません。
  - Fortran 95 Cray ポインタは、一部の組み込み呼び出しで使用できません。
  - 型の宣言でスラッシュを使用する f77 形式の初期化は、Fortran 95 では受け付けられません。
  - Fortran 95 では、非ポインタ変数に対する Cray 文字ポインタを文字ポインタ以外のその他の Cray ポインタに割り当てることはできません。
  - Fortran 95 では、型サイズの異なる項目 (たとえば、REAL*8、INTEGER*4) を同一の Cray ポインタがポイントすることはできません。
  - Fortran 95 は、BYTE データ型を受け付けません。

- Fortran 95 では、非整数を配列添字として使用できません。
- f95 では、関連演算子 .EQ. および .NE. を論理演算対象とともに使用できません。

- プログラム、サブルーチン、関数、文
  - Fortran 95 では、PROGRAM 文に名前が必要です。
  - f95 の名前の最大長は 31 文字です。
  - Fortran 95 の関数は、サブルーチンと同様に、CALL 文で呼び出すことができません。
  - Fortran 95 では、関数の戻り値を定義する必要があります。
  - f77 では特定の組み込み関数にさまざまな型の引数を指定できますが、f95 では引数の型を統一する必要があります。
  - f95 は、デバッギングの注釈 ("D" で始まる注釈行) を認識できません。
  - Fortran 95 のタブ書式では、ソース行が 72 カラムを超えることはできません。
  - f95 のタブ書式では、文字列が継続行に及ぶ場合、72 カラム目で途切れることになります (166 ページを参照してください)。

- コマンド行オプション
  - f95 は、-vax コンパイラオプションを認識しません。

入出力の互換性

f95 が f77 の互換ライブラリにリンクされるので、f77 と f95 のバイナリ入出力には互換性があります。

バイナリ入出力に互換性があるので、以下のような状況でも実行することができます。

- 同じプログラムで、ある記録を f95 で書き込み、f77 で読み込むことができます。
- f95 のプログラムでファイルを書き込み、f77 のプログラムで読み込むことができます。

読み込まれた数字と書き込まれた数字が一致しない場合があります。

- 書式なしの場合：読み込まれた数字と書き込まれた数字は同じです。

- 浮動小数点の書式付きの場合：読み込まれた数字と書き込まれた数字が一致しないことがあります。これは、基数変換ルーチンが若干異なることや、大文字・小文字、空白、プラス・マイナス記号などの規則が異なるためです。

  例 : 1.0e12、1.0E12、1.0E+12

- 並びによる入出力の場合：読み込まれた数字と書き込まれた数字とが異なることがあります。これは、コンマ、空白、ゼロ、繰り返し因子などの配置規則が異なるためです。

  例 : '0.0' と '.0'

  例 : ' 7' と '7'

  例 : '3,4,5' と '3 4 5'

  例 : '3*0' と '0 0 0'

  上記の問題は以下のような式で発生します。

  ```
  integer::v(3)=(/0,0,0/); print *,v
  ```

  例 : '0.333333343' と '0.333333'

  上記の問題は以下のような式で発生します。

  ```
  PRINT *, 1.0/3.0
  ```

f77 でコンパイルしたルーチンとのリンク

- `f77` と `f95` のオブジェクトバイナリを混在させるには、`f95` および `f77` 互換性ライブラリ `libf77compat` (`libF77` ではない) とリンクします。`-xlang=f77` オプションを使用すると、その作業を容易に実行できます。主プログラムが `f77` プログラムであっても、`f95` でリンクを実行します。

例 : f95 のメインと f77 のサブルーチンです。

```
demo$ cat m.f95
CHARACTER*74 :: c = 'テストです。'
   CALL echo1( c )
END
demo$ cat s.f
    SUBROUTINE echo1( a )
    CHARACTER*74 a
    PRINT*, a
    RETURN
    END
demo$ f77 -c -silent s.f
demo$ f95 -xlang=f77 m.f95 s.o
demo$ a.out
 テストです。
demo$
```

- 通常、FORTRAN 77 ライブラリは f95 と互換性があります。

例 : FORTRAN 77 のライブラリからルーチンを呼び出す f95 のメインです。

```
demo$ cat tdtime.f95
        REAL e, dtime, t(2)
        e = dtime( t )
        DO i = 1, 100000
           as = as + cos(sqrt(float(i)))
        END DO
        e = dtime( t )
       PRINT *, '経過:', e, ', ユーザー:', t(1), ', システム:', t(2)
        END
demo$ f95 tdtime.f95
demo$ a.out
 経過:0.14, ユーザー:0.14, システム:0.0E+0
demo$
```

dtime(3F) を参照してください。

組み込み関数

Fortran 95 の標準機能として、以下に示すような組み込み関数が新たにサポートされました。これらの関数は FORTRAN 77 にはありません。

以下の名前をユーザープログラムで使用する場合は、EXTERNAL 文を追加し、組み込み関数ではなくユーザー関数を使用するように f95 に指示する必要があります。

Fortran 95 の組み込み関数には、以下のものがあります。

```
ADJUSTL,ADJUSTR,ALL,ALLOCATED,ANY,BIT_SIZE,COUNT,CSHIFT
,DIGITS,DOT_PRODUCT,EOSHIFT,EPSILON,EXPONENT,HUGE,KIND,
LBOUND,LEN_TRIM,MATMUL,MAXEXPONENT,MAXLOC,MAXVAL,MERGE,
MINEXPONENT,MINLOC,MINVAL,NEAREST,PACK,PRECISION,
PRESENT, PRODUCT,RADIX,RANGE,REPEAT,RESHAPE,RRSPACING,
SCALE,SCAN, SELECTED_INT_KIND,SELECTED_REAL_KIND,
SET_EXPONENT,SHAPE,SIZE,SPACING,SPREAD,SUM,TINY,
TRANSFER,TRANSPOSE,UBOUND, UNPACK,VERIFY
```

将来のバージョンとの互換性

ソースコードは、f95 の本リリースと将来のリリースで互換となる予定です。

f95 の本リリースでモジュール情報ファイルを作成する場合、そのファイルが将来のリリースと互換性があるかどうかは保証されません。

言語の混在

Solaris システムでは、C で書かれたルーチンを Fortran のプログラムと組み合わせることができます。C と Fortran では呼び出し規則が共通なためです。

モジュールファイル

Fortran 95 の MODULE を含むファイルをコンパイルすると、ソースで検出された MODULE ごとにモジュールインターフェースファイル (.mod ファイル) が生成されます。ファイル名は MODULE 名を基に付けられます。たとえば、MODULE.xyz からは xyz.mod (すべて小文字) というファイル名が作成されます。

コンパイルを実行すると、MODULE 文を含むソースファイルごとにモジュール実装オブジェクトファイル (.o) が生成されます。モジュール実装オブジェクトファイルとその他すべてのオブジェクトファイルをリンクすると、実行可能ファイルを作成できます。

コンパイラは、現在の作業ディレクトリで、モジュールインタフェースファイルと実装オブジェクトファイルを作成します。コンパイラは、USE *modulename* 文のコンパイル時、現在の作業ディレクトリでインタフェースファイルを探します。-M*path* オプションを使用すると、コンパイラに追加の検索パスを提供できます。モジュール実装オブジェクトファイルは、リンク処理のコマンド行に明示的に列挙しなければなりません。

通常、プログラマは、ファイルごとに単一の MODULE を定義し、MODULE 文とそれを含むソースファイルに同じ名前を割り当てます。

.mod ファイルは、アーカイブファイルに格納できません。つまり、単一のファイルに連結できません。

例：

```
demo% cat mod_one.f90
 MODULE one
    ...
 END MODULE
demo% cat mod_two.f90
 MODULE two
    ...
 END MODULE
demo% cat main.f90
 USE one
 USE two
   ...
 END
demo% f95 -o main mod_one.f90 mod_two.f90 main.f90
```

上の例では、すべてのファイルが一度にコンパイルされます。モジュールソースファイルは、主プログラムでの使用前に最初にコンパイルされます。コンパイルによって次のファイルが作成されます。

```
main
main.o
one.mod
```

```
mod_one.o
two.mod
mod_two.o
```

次の例では、各単位を個別にコンパイルし、それらをリンクします。

```
demo% f95 -c mod_one.f90 mod_two.f90
demo% f95 -c main.f90
demo% f95 -o main main.o mod_one.o mod_two.o
```

main.f90 のコンパイル時、コンパイラは、現在のディレクトリから one.mod および two.mod を検索します。これらのファイルは、USE 文のモジュールを参照するファイルをコンパイルする前にコンパイルしておく必要があります。その後の手順で、モジュール実装オブジェクトファイル mod_one.o および mod_two.o をその他すべてのオブジェクトファイルとリンクして、実行可能ファイルを作成します。

付録 D

-xtarget プラットフォームの展開

この付録では、-xtarget オプションのプラットフォームシステム名とその展開について説明します。

表 D-1 に示すように、-xtarget の値は -xarch、-xchip、-xcache オプションのそれぞれの値に展開されます。fpversion(1) を実行すると、対象システムの定義を確認できます。

たとえば、-xtarget=sun4/15 は、-xarch=v8a -xchip=micro -xcache=2/16/1 を意味します。

表 D-1 -xtarget の展開

-xtarget=	-xarch	-xchip	-xcache
generic	generic	generic	generic
generic64	v9	generic	generic
cs6400	v8	super	16/32/4:2048/64/1
entr150	v8plusa	ultra	16/32/1:512/64/1
entr2	v8plusa	ultra	16/32/1:512/64/1
entr2/1170	v8plusa	ultra	16/32/1:512/64/1
entr2/1200	v8plusa	ultra	16/32/1:512/64/1
entr2/2170	v8plusa	ultra	16/32/1:512/64/1
entr2/2200	v8plusa	ultra	16/32/1:512/64/1
entr3000	v8plusa	ultra	16/32/1:512/64/1
entr4000	v8plusa	ultra	16/32/1:512/64/1
entr5000	v8plusa	ultra	16/32/1:512/64/1
entr6000	v8plusa	ultra	16/32/1:512/64/1

表 D-1　-xtarget の展開 (続き)

-xtarget=	-xarch	-xchip	-xcache
sc2000	v8	super	16/32/4:2048/64/1
solb5	v7	old	128/32/1
solb6	v8	super	16/32/4:1024/32/1
ss1	v7	old	64/16/1
ss10	v8	super	16/32/4
ss10/20	v8	super	16/32/4
ss10/30	v8	super	16/32/4
ss10/40	v8	super	16/32/4
ss10/402	v8	super	16/32/4
ss10/41	v8	super	16/32/4:1024/32/1
ss10/412	v8	super	16/32/4:1024/32/1
ss10/50	v8	super	16/32/4
ss10/51	v8	super	16/32/4:1024/32/1
ss10/512	v8	super	16/32/4:1024/32/1
ss10/514	v8	super	16/32/4:1024/32/1
ss10/61	v8	super	16/32/4:1024/32/1
ss10/612	v8	super	16/32/4:1024/32/1
ss10/71	v8	super2	16/32/4:1024/32/1
ss10/712	v8	super2	16/32/4:1024/32/1
ss10/hs11	v8	hyper	256/64/1
ss10/hs12	v8	hyper	256/64/1
ss10/hs14	v8	hyper	256/64/1
ss10/hs21	v8	hyper	256/64/1
ss10/hs22	v8	hyper	256/64/1
ss1000	v8	super	16/32/4:1024/32/1
ss1plus	v7	old	64/16/1
ss2	v7	old	64/32/1
ss20	v8	super	16/32/4:1024/32/1
ss20/151	v8	hyper	512/64/1

表 D-1　-xtarget の展開 (続き)

-xtarget=	-xarch	-xchip	-xcache
ss20/152	v8	hyper	512/64/1
ss20/50	v8	super	16/32/4
ss20/502	v8	super	16/32/4
ss20/51	v8	super	16/32/4:1024/32/1
ss20/512	v8	super	16/32/4:1024/32/1
ss20/514	v8	super	16/32/4:1024/32/1
ss20/61	v8	super	16/32/4:1024/32/1
ss20/612	v8	super	16/32/4:1024/32/1
ss20/71	v8	super2	16/32/4:1024/32/1
ss20/712	v8	super2	16/32/4:1024/32/1
ss20/hs11	v8	hyper	256/64/1
ss20/hs12	v8	hyper	256/64/1
ss20/hs14	v8	hyper	256/64/1
ss20/hs21	v8	hyper	256/64/1
ss20/hs22	v8	hyper	256/64/1
ss2p	v7	powerup	64/32/1
ss4	v8a	micro2	8/16/1
ss4/110	v8a	micro2	8/16/1
ss4/85	v8a	micro2	8/16/1
ss5	v8a	micro2	8/16/1
ss5/110	v8a	micro2	8/16/1
ss5/85	v8a	micro2	8/16/1
ss600/120	v7	old	64/32/1
ss600/140	v7	old	64/32/1
ss600/41	v8	super	16/32/4:1024/32/1
ss600/412	v8	super	16/32/4:1024/32/1
ss600/51	v8	super	16/32/4:1024/32/1
ss600/512	v8	super	16/32/4:1024/32/1
ss600/514	v8	super	16/32/4:1024/32/1

表 D-1　-xtarget の展開 (続き)

-xtarget=	-xarch	-xchip	-xcache
ss600/61	v8	super	16/32/4:1024/32/1
ss600/612	v8	super	16/32/4:1024/32/1
sselc	v7	old	64/32/1
ssipc	v7	old	64/16/1
ssipx	v7	old	64/32/1
sslc	v8a	micro	2/16/1
sslt	v7	old	64/32/1
sslx	v8a	micro	2/16/1
sslx2	v8a	micro2	8/16/1
ssslc	v7	old	64/16/1
ssvyger	v8a	micro2	8/16/1
sun4/110	v7	old	2/16/1
sun4/15	v8a	micro	2/16/1
sun4/150	v7	old	2/16/1
sun4/20	v7	old	64/16/1
sun4/25	v7	old	64/32/1
sun4/260	v7	old	128/16/1
sun4/280	v7	old	128/16/1
sun4/30	v8a	micro	2/16/1
sun4/330	v7	old	128/16/1
sun4/370	v7	old	128/16/1
sun4/390	v7	old	128/16/1
sun4/40	v7	old	64/16/1
sun4/470	v7	old	128/32/1
sun4/490	v7	old	128/32/1
sun4/50	v7	old	64/32/1
sun4/60	v7	old	64/16/1
sun4/630	v7	old	64/32/1
sun4/65	v7	old	64/16/1

表 D-1　-xtarget の展開 (続き)

-xtarget=	-xarch	-xchip	-xcache
sun4/670	v7	old	64/32/1
sun4/690	v7	old	64/32/1
sun4/75	v7	old	64/32/1
ultra	v8plusa	ultra	16/32/1:512/64/1
ultra1/140	v8plusa	ultra	16/32/1:512/64/1
ultra1/170	v8plusa	ultra	16/32/1:512/64/1
ultra1/200	v8plusa	ultra	16/32/1:512/64/1
ultra2	v8plusa	ultra2	16/32/1:512/64/1
ultra2/1170	v8plusa	ultra	16/32/1:512/64/1
ultra2/1200	v8plusa	ultra	16/32/1:1024/64/1
ultra2/1300	v8plusa	ultra2	16/32/1:2048/64/1
ultra2/2170	v8plusa	ultra	16/32/1:512/64/1
ultra2/2200	v8plusa	ultra	16/32/1:1024/64/1
ultra2/2300	v8plusa	ultra2	16/32/1:2048/64/1
ultra2e	v8plusa	ultra2e	16/32/1:256/64/4
ultra2i	v8plusa	urtra2i	16/32/1:i512/64/1
ultra3	v8plusa	ultra3	64/32/4:8192/256/1

付録 E

Fortran 指令の要約

この付録では、`f77` や `f95` の Fortran コンパイラで認識されている指令について要約します。

- 一般的な Fortran 指令
- Sun の並列化指令
- Cray の並列化指令
- OpenMP の Fortran 95 指令、ライブラリルーチン、および環境

注 - Fortran 並列化機能を使用するには、Sun WorkShop HPC ライセンスが必要です。

一般的な Fortran 指令

f77 でも f95 でも適用される一般的な指令については、第 2 章で説明しています。

表 E-1　一般的な Fortran 指令の要約

Format	
C$PRAGMA キーワード （ *a* [, *a*] … ）*[, keyword (a [, a] …)] ,… C$PRAGMA SUN キーワード （ *a* [, *a*] … ）*[, keyword (a [, a] …)] ,… C$PRAGMA SPARC キーワード （ *a* [, *a*] … ）*[, keyword (a [, a] …)] ,… 1 桁目のコメント指示子には、c、C、!、または * を使用できます (上記の例では、C を使用しています。f95 自由書式では ! を使用しなければなりません)。	
C 指令	C$PRAGMA C (*list*) 外部関数の名前リストを C 言語のルーチンとして宣言します。
UNROLL 指令	C$PRAGMA SUN UNROLL=*n* コンパイラに、次のループは長さ *n* に展開できることを伝えます。
WEAK 指令	C$PRAGMA WEAK (*name*[=*name2*]) *name* を弱い記号 (weak symbol) または *name2* のエイリアスとして宣言します。
OPT 指令	C$PRAGMA SUN OPT=*n* サブプログラムの最適化レベルを *n* に設定します。
NOMEMDEP 指令	C$PRAGMA SUN NOMEMDEP 次のループにメモリの依存関係が存在しないことを宣言します。 (-parallel または -explicitpar が必要です。)
PIPELOOP 指令	C$PRAGMA SUN PIPELOOP=*n* 反復 *n* 間のループの依存性を別に宣言します。
PREFETCH 指令	C$PRAGMA SPARC_PREFETCH_READ_ONCE (*name*) C$PRAGMA SPARC_PREFETCH_READ_MANY (*name*) C$PRAGMA SPARC_PREFETCH_WRITE_ONCE (*name*) C$PRAGMA SPARC_PREFETCH_WRITE_MANY (*name*) コンパイラに *name* へ参照したときに先読み命令を生成するように要求します (-xprefetch オプションを指定する必要があります)。

特殊な Fortran 95 指令

次の指令は、`f95` でのみ使用できます。詳細については、付録 C を参照してください。

表 E-2　特殊な Fortran 95 指令

書式	`!DIR$` *directive* `!DIR$&` ...	: 最初の行 : 継続行 固定書式の場合、C は CDIR$ 指令 ... のように指令指示子としても受け取られます。その行は、1 桁目から始まる必要があります。 自由書式ソースの場合、その行の前でも空白を使用できます。
`FIXED/FREE` 指令	`!DIR$ FREE` `!DIR$ FIXED`	これらの指令では、それ以降の行のソースフォーマットを指定します。これらは、次に `FREE` 指令または `FIXED` 指令が出現するまで、残りのソースファイルに適用されます。

Sun の並列化指令

Sun 形式の並列化指令は、デフォルト (-mp=sun コンパイラオプション)であり、「Fortran プログラミングガイド」の並列化に関する章で詳しく説明しています。

表 E-3　Sun 形式の並列化指令の要約

書式	C$PAR *directive [optional_qualifiers]*　: 最初の行 C$PAR& *[more_qualifiers]*　: 継続行 固定書式。指令指示子には、C (上記の例)、c、*、または ! を使用できます。 修飾子が複数ある場合は、コンマで区切ります。72 桁目を越える文字は、-e コンパイラオプションを指定していない限り無視されます。
TASKCOMMON 指令	C$PAR TASKCOMMON *block_name* 共通ブロック *block_name* の変数をスレッド非公開として宣言します。これは、スレッドに対しては非公開ですが、スレッド内ではグローバルとなります。共通ブロック TASKCOMMON を宣言するには、そのブロックの全共通宣言の後にこの指令を指定する必要があります。
DOALL 指令	C$PAR DOALL *[* 修飾子 *]* それ以降の DO ループを並列化します。修飾子は、次のとおりです。 PRIVATE(*list*)　リストの名前を PRIVATE として宣言します。 SHARED(*list*)　リストの名前を SHARED として宣言します。 MAXCPUS(*n*)　多くても使用するスレッドは *n* 個です。 READONLY(*list*)　リストの変数は、ループ内で変更されません。 SAVELAST　すべての非公開変数の最終的な値を保存します。 STOREBACK(*list*)　リストの変数の最終的な値を保存します。 REDUCTION(*list*)　リストの変数は、縮約変数です。 SCHEDTYPE(*type*)　スケジューリング型を使用します (デフォルトは STATIC です)。 STATIC SELF(*nchunk*) FACTORING[(*m*)] GSS[(*m*)]
DOSERIAL 指令	C$PAR DOSERIAL 以降のループの並列化を無効にします。
DOSERIAL* 指令	C$PAR DOSERIAL* 以降のループのネストの並列化を無効にします。

Cray の並列化指令

Cray 形式の並列化指令については、『Fortran プログラミングガイド』の並列化に関する章で詳しく説明しています。
`-mp=cray` コンパイラオプションを指定する必要があります。

表 E-4　Cray の並列化指令の要約

書式	`CMIC$` *directive qualifiers* : 最初の行 `CMIC$&` *[more_qualifiers]* : 継続行 固定書式。指令指示子には、`C` (上記の例)、`c`、*、または ! を使用できます。`f95` 自由書式の場合、`!MIC$` より前に空白を指定できます。
DOALL 指令	`CMIC$ DOALL SHARED(`*list*`), PRIVATE(`*list*`)` *[, more_qualifiers]* 以降のループの並列化を無効にします。 スコーピング修飾子が必要です (*list* が空でない場合)—ループ内の変数はすべて PRIVATE 句または SHARED 句で宣言されていなければなりません。 `PRIVATE(`*list*`)` リストの名前を PRIVATE として宣言します。 `SHARED(`*list*`)` リストの名前を SHARED として宣言します。 `AUTOSCOPE` 変数の範囲を自動的に判別します。 次にオプションを示します。 `MAXCPUS(`*n*`)` 多くても使用するスレッドは *n* 個です。 `SAVELAST` すべての非公開変数の最終的な値を保存します。 スケジューリング修飾子は 1 つだけ指定できます。 `GUIDED` Sun 形式の `GSS(64)` と等価です。 `SINGLE` Sun 形式の `SELF(1)` と等価です。 `CHUNKSIZE(`*n*`)` Sun 形式の `SELF(n)` と等価です。 `NUMCHUNKS(`*m*`)` Sun 形式の `SELF(n/m)` と等価です。 デフォルトのスケジューリング型は、Sun 形式の `STATIC` です。これと等価の Cray 形式のスケジューリング型はありません。これらのスケジューリング型の解釈方法は、Sun 形式と Cray 形式で異なります。詳細については、『Fortran プログラミングガイド』を参照してください。

表 E-4　Cray の並列化指令の要約 (続き)

TASKCOMMON 指令	CMIC$ TASKCOMMON *block_name* 指定した共通ブロックの変数をスレッド非公開として宣言します。これは、スレッドに対しては非公開ですが、スレッド内ではグローバルとなります。共通ブロック TASKCOMMON を宣言するには、そのブロックの全共通宣言の直前または直後にこの指令を指定する必要があります。
DOSERIAL 指令	CMIC$ DOSERIAL 以降のループの並列化を無効にします。
DOSERIAL* 指令	CMIC$ DOSERIAL* 以降のループのネストの並列化を無効にします。

Fortran 95 の OpenMP 指令

Sun の Fortran 95 コンパイラでは、OpenMP バージョン 2.0 の Fortran API がサポートされています。コンパイラフラグの -openmpを使用すれば、これらの指令を使用できるようになります (88 ページを参照)。

ここでは、f95 でサポートされている OpenMP 指令、ライブラリルーチン、環境変数を示します。OpenMP を使う並列プログラミングの詳細については、http://www.openmp.org/ の OpenMP 2.0 Fortran 仕様を参照してください。

次の表に、f95 でサポートされている OpenMP 指令を要約します。角括弧 ([...]) で囲まれている項目は、オプションです。コンパイラは、指令と同じ行で感嘆符 (!) に後続する文字列を注釈として認識します。-openmp によるコンパイル時、CPP/FPP 変数 _OPENMP が定義され、#ifdef_OPENMP および #endif の条件付きコンパイルに使用されることがあります。

表 E-5 Fortran 95 における OpenMP 指令の要約

指令の形式 (固定)	C$OMP *directive optional_clauses...* !$OMP *directive optional_clauses...* *$OMP *directive optional_clauses...* 1桁目から始まらなければなりません。継続行については、6 桁メニュー空白以外の文字化ゼロ以外の文字が指定されていなければなりません。
指令の形式 (自由)	!$OMP *directive optional_clauses...* 任意の場所で指定でき、その前に空白があってもかまいません。継続行は、アンパサンド (&) により識別されます。
条件付きコンパイル	1 桁目と 2 桁目が !$、C$、*$ のいずれかで始まるソース行 (固定書式)、または前に空白が指定されたソース行 (自由書式) は、コンパイラオプション -openmp または -mp=openmp が指定されている場合にのみコンパイルされます。
PARALLEL 指令	!$OMP PARALLEL [*clause*[[,] *clause*]...] ブロックに対する転送のない Fortran 文のブロック !$OMP END PARALLEL 並列化の範囲を定義します。この範囲は、複数のスレッドにより並列で実行されるコードのブロックです。*clause* には PRIVATE(*list*), SHARED(*list*), DEFAULT(*option*), FIRSTPRIVATE(*list*), REDUCTION(*list*), IF(*logical_expression*), COPYIN(*list*) , NUM_THREADS(*integer_expression*) のいずれかを指定できます。

表 E-5　Fortran 95 における OpenMP 指令の要約 (続き)

DO 指令	!$OMP DO [*clause*[[,] *clause*]...] *do_loop* 文のブロック [!$OMP END DO [NOWAIT]] DO 指令は、直後の DO ループの反復を並列で実行するように指定します。この指令は、並列化範囲内に指定しなければなりません。*clause* には PRIVATE(*list*), FIRSTPRIVATE(*list*), LASTPRIVATE(*list*), REDUCTION(*list*), SCHEDULE(*type*), ORDERED のいずれかを指定できます。
SECTIONS 指令	!$OMP SECTIONS [*clause*[[,] *clause*]...] [!$OMP SECTION] 転送のない *Fortran*文のブロック [!$OMP SECTION *Fortran* 文のオプションブロック] ... !$OMP END SECTIONS [NOWAIT] チーム内のスレッド間で分割されるコードの非反復セクションを囲みます。各セクションは、チーム内のスレッドにより 1 度実行されます。*clause* には PRIVATE(*list*), FIRSTPRIVATE(*list*), LASTPRIVATE(*list*), REDUCTION(*list*) のいずれかを指定できます。 各セクションの前には SECTION 指令があり、これは最初のセクションで選択できます。
SINGLE 指令	!$OMP SINGLE [*clause*[[,] *clause*]...] 転送のない *Fortran* 文のブロック !$OMP END SINGLE [*end-modifier*] SINGLE 指令により囲まれた文はチーム内の 1 スレッドによってのみ実行されます。文の SINGLE ブロックを実行しないチームのスレッドは、NOWAIT が指定されない限り、END SINGLE 指令で待機しています。*clause*には、PRIVATE(*list*) または FIRSTPRIVATE(*list*) のいずれかを指定できます。*end-modifier* は、COPYPRIVATE(*list*) [[,]COPYPRIVATE(*list...*)] または NOWAIT のいずれかです。

表 E-5　Fortran 95 における OpenMP 指令の要約 (続き)

WORKSHARE 指令	!$OMP WORKSHARE *Fortran* 文のブロック !$OMP END WORKSHARE [NOWAIT] これらの指令で囲まれたコードブロックの実行作業を個々の作業単位に分割します。チームのスレッドが作業を共有するため、各単位は 1 回しか実行されません。
PARALLEL DO 指令	!$OMP PARALLEL DO [*clause*[[,] *clause*]...] *do_loop* 文のブロック [!$OMP END PARALLEL] 1つの DO ループに含まれている並列領域を指定する簡単な方法です。PARALLEL 指令のすぐ後に DO 指令を指定します。*clause* には、PARALLEL 指令や DO 指令で指定可能な句のいずれかを使用できます。
PARALLEL SECTIONS 指令	!$OMP PARALLEL SECTIONS [*clause*[[,] *clause*]...] [!$OMP SECTION] ブロックに対する転送のない *Fortran* 文のブロック [!$OMP SECTION *Fortran* 文のオプションブロック] ... !$OMP END PARALLEL SECTIONS 1 つの SECTIONS 指令に含まれている並列領域を指定する簡単な方法です。PARALLEL 指令のすぐ後に SECTIONS 指令を指定します。*clause* には、PARALLEL 指令や SECTIONS 指令で指定可能な句のいずれかを使用できます。
PARALLEL WORKSHARE 指令	!$OMP PARALLEL WORKSHARE[*clause*[[,] *clause*]...] *block of Fortran statements* !$OMP END PARALLEL WORKSHARE 単一の WORKSHARE 指令を含む並列領域を指定するためのショートカットを提供します。*clause* には、PARALLEL または WORKSHARE 指令によって受け付けられる句のいずれか 1 つを指定できます
同期指令	

表 E-5　Fortran 95 における OpenMP 指令の要約 (続き)

MASTER 指令	!$OMP MASTER ブロックに対する転送のない *Fortran* 文のブロック !$OMP END MASTER これらの指令で囲まれている文のブロックは、チームのマスタースレッドだけが実行します。その他のスレッドは、このブロックをスキップして、処理を継続します。マスターセクションの開始または終了では、暗黙的な境界はありません。
CRITICAL 指令	!$OMP CRITICAL [(*name*)] ブロックに対する転送のない *Fortran* 文のブロック !$OMP END CRITICAL [(*name*)] これらの指令で囲まれている文のブロックへのアクセスを、一度に 1 スレッドだけに制限します。オプション *name* の引数には識別のための領域名を指定します。指定のない CRITICAL 指令では、すべて同じ名前にマッピングされます。クリティカルなセクション名とは、プログラムの大域要素を指します。名前がほかの要素と競合する場合は、プログラムの動作が定まらなくなります。CRITICAL 指令で名前を指定した場合は、END CRITICAL 指令にもその名前を指定しなければなりません。
BARRIER 指令	!$OMP BARRIER チーム内のすべてのスレッドの同期をとります。各スレッドは、チーム内のその他すべてのスレッドがこの点に到達するまで待機します。

表 E-5　Fortran 95 における OpenMP 指令の要約 (続き)

ATOMIC 指令	!$OMP ATOMIC 特定のメモリ位置を自動的に更新するようにし、複数の同時書き込みスレッドが生じないようにします。 指令は、そのすぐ後の文にのみ適用されます。その文は、次の形式のいずれかでなければなりません。 *x = x operator expression* *x = expression operator x* *x = intrinsic(x, expression)* *x = intrinsic(expression, x)* ただし、 • *x* は、intrinsic 型のスカラーです。 • *expression* は、*x* を参照しないスカラー式です。 • *intrinsic* は、MAX、MIN、IAND、IOR、IEOR のいずれかです。 • *operator* は、 +、 -、 *、 /、 .AND.、 .OR.、 .EQV.、 .NEQV. のいずれかです。 この実装は、対象セクションのターゲット文を囲むことにより、すべての ATOMIC 指令と置き換わります。
FLUSH 指令	!$OMP FLUSH [(*list*)] スレッド可視変数は、この指令が出現したところでメモリに書き戻されます。FLUSH 指令は、実行スレッドとグローバルメモリの演算に一貫性を提供するだけです。省略可能な *list* では、フラッシュする必要のある変数をコンマで区切って列挙します。FLUSH 指令は、次の指令に対し暗示されます。 BARRIER、CRITICAL/ENDCRITICAL、ENDDO、END SECTIONS、ENDSINGLE、 ENDWORKSHARE、 ORDERED/ENDORDERED、PARALLEL/ENDPARALLEL、 PARALLEL/ENDPARALLELDO、 PARALLELSECTIONS/ENDPARALLELSECTIONS、 PARALLELWORKSHARE/ENDPARALLELWORKSHARE。 NOWAIT が指定されている場合、FLUSH は暗示されません。 FLUSH は、DO、MASTER/ENDMASTER、SECTIONS、 SINGLE、および WORKSHARE によって暗示されません。

表 E-5　Fortran 95 における OpenMP 指令の要約 (続き)

ORDERED 指令	!$OMP ORDERED ブロックに対する転送のない *Fortran* 文のブロック !$OMP END ORDERED この指令により囲まれた文のブロックは、反復がループの実行順に実行されるときの順序で実行されます。これは、DO 指令または PARALLEL DO 指令の動的拡張機能でのみ指定できます。ORDERED 句は、ブロックを囲んでいる一番近い DO 指令で指定しなければなりません。
データ環境指令	
THREADPRIVATE 指令	!$OMP THREADPRIVATE (*list*) 変数のリストおよび指定した共通ブロックをスレッドに対して非公開としますが、そのスレッド内では大域となります。共通ブロック名はスラッシュで囲む必要があります。共通ブロックを THREADPRIVATE にするには、そのブロックのすべての COMMON 宣言の後でこの指令を指定しなければなりません。
データスコープ (有効範囲) 句	
上記のいくつかの指令は、指令により閉じた変数のスコープ属性を制御するための句を受け取ります。データスコープ句が指令に指定されていない場合は、その指令に関係のある変数のデフォルトスコープは SHARED となります。*list* には、スコープ単位でアクセス可能な指定された変数や共通ブロックのリストをコンマで区切って指定します。共通ブロック名は、スラッシュで囲まなければなりません (/ABLOCK/ など)。	
PRIVATE 句	PRIVATE(*list*) チーム内の各スレッドに対して非公開となるように、変数のリストをコンマで区切って宣言します。
SHARED 句	SHARED(*list*) チーム内のすべてのスレッドは、*list* に指定された変数を共有し、同じ記憶領域にアクセスします。

表 E-5　Fortran 95 における OpenMP 指令の要約 (続き)

DEFAULT 句	DEFAULT(PRIVATE | SHARED | NONE) 並列領域内のすべての変数にスコーピング属性を指定します。THREADPRIVATE 変数は、この句の影響を受けません。この句を指定しない場合は、DEFAULT(SHARED) であるとみなされます。
FIRSTPRIVATE 句	FIRSTPRIVATE(*list*) *list* の変数は、PRIVATE です。また、変数の非公開コピーは、構成前から存在している元のオブジェクトから初期化されます。
LASTPRIVATE 句	LASTPRIVATE(*list*) *list* の変数は、PRIVATE です。また、DO 指令で LASTPRIVATE 句が指定されていると、順序が最後の反復を実行するスレッドにより変数のバージョンが構成前に更新されます。SECTIONS 指令では、意味的に最後の SECTION を実行するスレッドにより、それが持っているオブジェクトのバージョンが構成前に更新されます。
REDUCTION 句	REDUCTION([*operator* | *intrinsic*] : *list*) *operator* は、 + * - .AND. .OR. .EQV. .NEQV. のいずれかです。 *intrinsic* は、 MAX MIN IAND IOR IEOR のいずれかです。 *list* の変数は、intrinsic タイプの名前付き変数でなければなりません。 REDUCTION 句は、前述の ATOMIC 指令で示したフォームの縮約文でのみ縮約変数が使用される領域で使用することを目的としています。*list* の変数は、囲んだコンテキストの中で SHARED でなければなりません。各変数の非公開コピーは、それが PRIVATE の場合、スレッドごとに作成されます。縮約の終わりに、各非公開コピーの最終値を元の値と組み合わせて、共有変数が更新されます。

表 E-5　Fortran 95 における OpenMP 指令の要約 (続き)

COPYIN 句	COPYIN(*list*) COPYIN 句は、変数、共通ブロック、および THREADPRIVATE として宣言される共通ブロックの変数に対してのみ適用されます。並列領域で、COPYIN 句はチームのマスタースレッドのデータが並列領域の先頭で共有ブロックのスレッド非公開コピーにコピーされるように指定します。
COPYPRIVATE 句	COPYPRIVATE(*list*) チームの特定メンバーから他のメンバーに値をブロードキャスト通信するために、非公開変数 (共有オブジェクトへのポインタ) を使用します。リストに示されている変数は、COPYPRIVATE を指定する SINGLE 構文の PRIVATE または FIRSTPRIVATE 句で使用できません。
DO 指令と PARALLEL DO 指令のスケジューリング句	
SCHEDULE 句	SCHEDULE(*type* [,*chunk*]) DO ループの反復がチームのスレッド間でどのように分割されるかを指定します。*type* には、次のいずれかを指定します。SCHEDULE 句を指定しなければ、STATIC スケジューリングが使用されます。
STATIC スケジューリング	SCHEDULE(STATIC, *chunk*) 反復は、チャンクにより指定されたサイズに分割されます。分割された反復は、スレッド番号の順序で総当り形式でチームのスレッドに割り当てられます。*chunk* は、スカラー整数式でなければなりません。

表 E-5　Fortran 95 における OpenMP 指令の要約 (続き)

DYNAMIC スケジューリング	SCHEDULE(DYNAMIC, *chunk*) 反復は、チャンクにより指定されたサイズに分割されます。各スレッドは、反復空間を終了すると、次の反復セットを動的に取得します。
GUIDED スケジューリング	SCHEDULE(GUIDED, *chunk*) GUIDED を指定すると、チャンクサイズは実行された各反復セットによりべき乗的に縮約されます。*chunk* は、その都度実行する反復の最小数を指定します。デフォルトのチャンクのサイズは 1 です。反復セットの初回サイズは、ループを実行するスレッドの数でループの反復回数を割算した数値です。
RUNTIME スケジューリング	SCHEDULE(RUNTIME) スケジューリングが、実行時まで遅延されます。スケジュール type やチャンクサイズは、OMP_SCHEDULE 環境変数の設定値をもとに決定されます。（デフォルトは STATIC です。）

OpenMP ライブラリルーチン

OpenMP Fortran API ライブラリルーチンは、外部手続きです。次の要約で、*int_expr* はデフォルトでスカラー整数式であり、*logical_expr* はデフォルトのスカラー論理式です。

INTEGER (4) および LOGICAL (4) を返す OMP_ 関数は組み込みではないため、正しく宣言する必要があります。正しく宣言されない場合、コンパイラは REAL と判断します。下に要約した OpenMP Fortran 実行時ライブラリルーチンのインタフェース宣言は、Fortran インクルードファイル omp_lib.h および Fortran 95 MODULE omp_lib によって提供されます。これについては、Fortran OpenMP 2.0 仕様書で説明されています。これらのライブラリルーチンを参照するすべてのプログラム単位において、INCLUDE 'omp_lib.h' 文または #include "omp_lib.h" プリプロセッサ指令、または USE omp_lib 文を配置してください。

-Xlist を指定してコンパイルを実行すると、あらゆる型の不一致が報告されます。

表 E-6　Fortran 95 OpenMP ライブラリルーチンの要約

実行環境ルーチン
OMP_SET_NUM_THREADS サブルーチン SUBROUTINE OMP_SET_NUM_THREADS(*int_expr*) 次の並列領域で使用するためにスレッド数を設定します。
OMP_GET_NUM_THREADS 関数 INTEGER(4) FUNCTION OMP_GET_NUM_THREADS() 呼び出し元の並列領域を実行しているチーム内に現在あるスレッドの数を返します。
OMP_GET_MAX_THREADS 関数 INTEGER(4) FUNCTION OMP_GET_NUM_THREADS( ) OMP_GET_NUM_THREADS 関数への呼び出しで返すことが可能な最大値を返します。
OMP_GET_THREAD_NUM 関数 INTEGER(4) FUNCTION OMP_GET_THREAD_NUM() チーム内のスレッド番号を返します。これは、0 から OMP_GET_NUM_THREADS()-1 までの数値です。マスタースレッドはスレッド 0 です。
OMP_GET_NUM_PROCS 関数 INTEGER(4) FUNCTION OMP_GET_NUM_PROCS() プログラムで使用可能なプロセッサの数を返します。
OMP_IN_PARALLEL 関数 LOGICAL(4) FUNCTION OMP_IN_PARALLEL() 並列で実行している領域の動的拡張内から呼び出された場合、.TRUE. を返します。それ以外の場合は、.FALSE. を返します。
OMP_SET_DYNAMIC サブルーチン

表 E-6　Fortran 95 OpenMP ライブラリルーチンの要約

	SUBROUTINE OMP_SET_DYNAMIC(*logical_expr*) プログラムの並列実行で使用可能なスレッド数の動的調整を有効または無効にします。デフォルトにより、動的調整は有効になります。
OMP_GET_DYNAMIC 関数	
	LOGICAL(4) FUNCTION OMP_GET_DYNAMIC() 動的スレッド調整が有効な場合、.TRUE. を返します。それ以外の場合は、.FALSE. を返します。
OMP_SET_NESTED サブルーチン	
	SUBROUTINE OMP_SET_NESTED(*logical_expr*) ネストされた並列化機能を有効または無効にします。(デフォルトにより、ネストされた並列化機能は無効になります)。ネストされた並列化機能はサポートされていません。
OMP_GET_NESTED 関数	
	LOGICAL(4) FUNCTION OMP_GET_NESTED() ネストされた並列化機能が有効な場合、.TRUE. を返します。それ以外の場合は、.FALSE. を返します。ネストされた並列化機能はサポートされていません。この関数は常に .FALSE. を返します。
ロックルーチン	

2 種類のロックがサポートされています。単純ロックとネストロックです。ネストロックは、ロック解除される前に同一スレッドによって複数回にわたりロックできます。単純ロックは、すでにロック状態に入っている場合にロックできません。単純ロックの変数は単純ロックルーチンにのみ渡すことができ、ネストロックの変数はネストロックルーチンにのみ渡すことができます。
 	
ロック変数 *var* は、次のルーチンからしかアクセスできません。
そのためには、パラメータ OMP_LOCK_KIND および OMP_NEST_LOCK_KIND を使用します (omp_lib.h インクルードファイルおよび omp_lib モジュールで定義されています)。次に例を示します。
 	
`INTEGER(KIND=OMP_LOCK_KIND) :: var`
`INTEGER(KIND=OMP_NEST_LOCK_KIND) :: nvar`	

表 E-6　Fortran 95 OpenMP ライブラリルーチンの要約

OMP_INIT_LOCK サブルーチン SUBROUTINE OMP_INIT_LOCK(*var*) SUBROUTINE OMP_INIT_NEST_LOCK(*nvar*) 以降の呼び出しで使用するために、ロック変数 *var* に関連付けられているロックを初期化します。初期の状態でロックは解除されています。
OMP_DESTROY_LOCK サブルーチン SUBROUTINE OMP_DESTROY_LOCK(*var*) SUBROUTINE OMP_DESTROY_NEST_LOCK(*nvar*) 指定したロック変数 *var* が関連付けられているロックがあれば、それから *var* との関連付けを解除します。
OMP_SET_LOCK サブルーチン SUBROUTINE OMP_SET_LOCK(*var*) SUBROUTINE OMP_SET_NEST_LOCK(*nvar*) 指定したロックが使用可能な状態になるまで、実行スレッドを待機させます。ロックが使用可能になると、スレッドにロックの所有権が与えられます。
OMP_UNSET_LOCK サブルーチン SUBROUTINE OMP_UNSET_LOCK(*var*) SUBROUTINE OMP_UNSET_NEST_LOCK(*nvar*) 実行スレッドから、ロックの所有権を解放します。スレッドがそのロックを所有していない場合は、動作は不確定となります。
OMP_TEST_LOCK 関数 LOGICAL FUNCTION OMP_TEST_LOCK(*var*) INTEGER FUNCTION OMP_TEST_NEST_LOCK(*nvar*) ロック変数 *var* に関連付けられているロックを設定しようと試みます。単純ロックが無事に設定された場合に .TRUE. を返し、それ以外の場合に .FALSE. を返します。 OMP_TEST_NEST_LOCK は、*nvar* と関連付けられたロックが無事に設定された場合に新たなネスト回数を返し、それ以外の場合に 0 を返します。
タイミングルーチン 次の 2 つの関数は、倍精度の数値 (REAL(8)) を返し、時計のタイマーをサポートします。

表 E-6　Fortran 95 OpenMP ライブラリルーチンの要約

OMP_GET_WTIME 関数 `REAL(8) FUNCTION OMP_GET_WTIME()` 「過去の任意の時間」から経過した時計時間 (秒単位) に等しい倍精度の数値を返します。
OMP_GET_WTICK 関数 `REAL(8) FUNCTION OMP_GET_WTICK()` クロック刻みの間隔 (秒単位) に等しい倍精度の数値を返します。

OpenMP 環境変数

表 E-7 と表 E-8 は、OpenMP プログラムの実行を制御する OpenMP Fortran API 環境変数を要約したものです。

表 E-7　OpenMP Fortran 環境変数の要約

OMP_SCHEDULE スケジュール型が RUNTIME として指定された DO 指令や PARALLEL DO 指令のスケジュール型を設定します。指定しない場合、STATIC のデフォルト値が使用されます。値は″ type［,*chunk*］″ です。 例：`setenv OMP_SCHEDULE "GUIDED,4"`
OMP_NUM_THREADS NUM_THREADS 句つまりOMP_SET_NUM_THREADS() サブルーチンの呼び出しにより設定されていない場合に、実行時に使用するスレッド数を設定します。設定しない場合、デフォルト値 1 が使用されます。値は正の整数です (現在の最大数は 128)。 例：`setenv OMP_NUM_THREADS 16`
OMP_DYNAMIC 並列領域の実行で使用可能なスレッド数の動的調整機能を有効または無効にします。設定しない場合、デフォルト値 TRUE が使用されます。値は TRUE または FALSE です。 例：`setenv OMP_DYNAMIC FALSE`
OMP_NESTED ネストされた並列機能を有効または無効にします。(ネストされた並列化機能はサポートされていません)。値は TRUE または FALSE です。設定されていない場合のデフォルトは FALSE です。 例：`setenv OMP_NESTED TRUE`

表 E-8　OpenMP Fortran API の一部ではない環境変数

SUNW_MP_WARN 実行時ライブラリによって発行される警告メッセージを制御します。TRUE に設定した場合、実行時ライブラリは警告メッセージを stderr に発行します。FALSE に設定した場合、警告メッセージは発行されません。デフォルトは FALSE です。 例： setenv SUNW_MP_WARN TRUE
SUNW_MP_THR_IDLE プログラムの並列部分を実行する各スレッドの end-of-task ステータスを制御します。この値は、spin、sleep ns、または sleep nms に設定できます。デフォルトは SPIN であり、スレッドは並列タスクの完了後、新しい並列タスクが到着するまでスピン (busy-wait) します。SLEEP *time* は、並列タスクの完了後にスレッドがスピン状態を継続する時間を指定します。スレッドがスピンしているあいだに新しいタスクが到着した場合、スレッドは即座に新しいタスクを実行します。それ以外の場合、スレッドはスリープ状態に入り、新しいタスクの到着時に動作を再開します。*time* は秒 (ns または n) またはミリ秒 (nms) で指定できます。 引数なしで SLEEP を指定すると、スレッドは並列タスクの完了直後にスリープ状態に入ります。SLEEP、SLEEP (0)、SLEEP (0s)、および SLEEP (0ms) はすべて同等です。 例： setenv SUNW_MP_THR_IDLE (50ms)
STACKSIZE スレッドのスタックサイズを設定します。値の単位はキロバイトです。 例： setenv STACKSIZE 8192　スレッドのスタックサイズを 8M バイト に設定します。

索引

か

き

す

せ

そ

た

れ