Fortran ユーザーズガイド

Sun™ Studio 10

Sun Microsystems, Inc.
www.sun.com

Part No. 819-1604-10
2005 年 1 月, Revision A

原典: *Fortran User's Guide : Sun Studio 10*
Part No: 819-0492-10
Revision A

Please
Recycle

目次

表目次

はじめに

Fortran ユーザーズガイドは、Sun™ Studio Fortran 95 コンパイラ `f95` 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。

このマニュアルは、Fortran 言語について実用的な知識を持ち、Fortran コンパイラの効果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® について一般的な知識を持つ読者を対象としています。

入出力、プログラム開発、ライブラリ、プログラムの分析とデバッグ、数値精度、移植、パフォーマンス、最適化、並列化、相互運用性についての重要な情報については、関連マニュアルの『Fortran プログラミングガイド』を参照してください。

書体と記号について

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、コード例。	.login ファイルを編集します。 ls -a を実行します。 % 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します。	マシン名% **su** Password:
AaBbCc123 または**ゴシック**	コマンド行の可変部分。実際の名前や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。	『Solaris ユーザーマニュアル』
「 」	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。 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こえる場合に、継続を示します。	% **grep '^#define ** **XV_VERSION_STRING'**

* 使用しているブラウザにより、これら設定と異なって表示される場合があります。

コードの記号	意味	記法	コード例
[ ]	角括弧には、オプションの引数が含まれます。	O[*n*]	-O4,-O
{ }	中括弧には、必須オプションの選択肢が含まれます。	d{y|n}	-dy
|	「パイプ」または「バー」と呼ばれる記号は、その中から 1 つだけを選択可能な複数の引数を区切ります。	B{dynamic|static}	-Bstatic
:	コロンは、コンマ同様に複数の引数を区切るために使用されることがあります。	R*dir*[:*dir*]	-R/local/libs:/U/a
…	省略記号は、連続するものの一部が省略されていることを示します。	-xinline=*f1*[,…*fn*]	-xinline=alpha,dos

- Δ は意味のある空白を示します。

```
ΔΔ36.001
```

- 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。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	マシン名%
UNIX の Bourne シェルと Korn シェル	$
スーパーユーザー (シェルの種類を問わない)	#

サポートされるプラットフォーム

この Sun Studio リリースは、SPARC® および x86 ファミリ (UltraSPARC®、SPARC64、AMD64、Pentium、Xeon EM64T) プロセッサアーキテクチャをサポートしています。サポートされるシステムの、Solaris オペレーティングシステムのバージョンごとの情報については、http://www.sun.com/bigadmin/hcl にあるハードウェアの互換性に関するリストで参照することができます。ここには、すべてのプラットフォームごとの実装の違いについて説明されています。

このドキュメントでは、“x86” という用語は、AMD64 または Intel Xeon/Pentium 製品ファミリと互換性があるプロセッサを使用して製造された 64 ビットおよび 32 ビットのシステムを指します。サポートされるシステムについては、ハードウェアの互換性に関するリストを参照してください。

Sun Studio ソフトウェアおよびマニュアルページへのアクセス

コンパイラおよびツールは、標準の /usr/bin/ および /usr/share/man の各ディレクトリにはインストールされません。コンパイラあるいはツールにアクセスするには、PATH 環境変数を正しく設定しておく必要があります (xx ページの「コンパイラとツールへのアクセス方法」を参照)。また、マニュアルページにアクセスするには、MANPATH 環境変数を正しく設定しておく必要があります (xxi ページの「マニュアルページへのアクセス方法」を参照)。

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、『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 Studio のコンパイラとツールが /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。製品ソフトウェアが /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。

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、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

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 変数はコンパイラとツールにアクセスできるように設定されています このパスがない場合は、次の手順に従って、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。Forte Developer ソフトウェア、Sun ONE Studio ソフトウェア、または Sun Studio の他のリリースをインストールしている場合は、インストール先のパスの前に、次のパスを追加します。

   /opt/SUNWspro/bin

マニュアルページへのアクセス方法

マニュアルページにアクセスするために MANPATH 環境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

1. 次のように入力して、dbx のマニュアルページを表示します。

```
% man dbx
```

2. 出力された場合、内容を確認します。

   dbx(1) の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、この節の指示に従って、MANPATH 環境変数を設定してください。

▼ MANPATH 環境変数を設定して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 MANPATH 環境変数に追加します。

   /opt/SUNWspro/man

統合開発環境へのアクセス方法

Sun Studio 統合開発環境 (IDE) には、C や C++、Fortran アプリケーションを作成、編集、構築、デバッグ、パフォーマンス解析するためのモージュールが用意されています。

IDE を起動するコマンドは、`sunstudio` です。このコマンドの詳細は、`sunstudio`(1) のマニュアルページを参照してください。

IDE が正しく動作するかどうかは、IDE がコアプラットフォームを検出できるかどうかに依存します。このため、`sunstudio` コマンドは、次の 2 つの場所でコアプラットフォームを探します。

- コマンドは、最初にデフォルトのインストールディレクトリ `/opt/netbeans/3.5V` を調べます。
- このデフォルトのディレクトリでコアプラットフォームが見つからなかった場合は、IDE が含まれているディレクトリとコアプラットフォームが含まれているディレクトリが同じであるか、同じ場所にマウントされているとみなします。たとえば IDE が含まれているディレクトリへのパスが `/foo/SUNWspro` の場合は、`/foo/netbeans/3.5V` ディレクトリにコアプラットフォームがないか調べます。

`sunstudio` が探す場所のどちらにもコアプラットフォームをインストールしていないか、マウントしていない場合、クライアントシステムの各ユーザーは、コアプラットフォームがインストールされているか、マウントされている場所 (*/installation_directory*`/netbeans/3.5V`) を、`SPRO_NETBEANS_HOME` 環境変数に設定する必要があります。

Forte Developer ソフトウェア、Sun ONE Studio ソフトウェア、または他のバージョンの Sun Studio ソフトウェアがインストールされている場合、IDE の各ユーザーは、`$PATH` のそのパスの前に、*/installation_directory*`/SUNWspro/bin` を追加する必要もあります。

`$PATH` には、*/installation_directory*`/netbeans/3.5V/bin` のパスは追加しないでください。

コンパイラとツールのマニュアルへのアクセス方法

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。
  `file:/opt/SUNWspro/docs/ja/index.html`

  製品ソフトウェアが `/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。
- マニュアルは、`docs.sun.com`sm の Web サイトで入手できます。以下に示すマニュアルは、インストールされている製品のマニュアルの索引から入手できます (`docs.sun.com` Web サイトでは入手できません)。
  - 『Standard C++ Library Class Reference』
  - 『標準 C++ ライブラリ・ユーザーズガイド』
  - 『Tools.h++ クラスライブラリ・リファレンスマニュアル』
  - 『Tools.h++ ユーザーズガイド』
- リリースノートは、`docs.sun.com` で入手できます。
- IDE の全コンポーネントのオンラインヘルプは、IDE 内の「ヘルプ」メニューだけでなく、多くのウィンドウおよびダイアログにある「ヘルプ」ボタンを使ってアクセスできます。

インターネットの Web サイト (`http://docs.sun.com`) から、Sun のマニュアルを参照したり、印刷したり、購入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。

注 – Sun では、本マニュアルに掲載した第三者の Web サイトのご利用に関しましては責任はなく、保証するものでもありません。また、これらのサイトあるいはリソースに関する、あるいはこれらのサイト、リソースから利用可能であるコンテンツ、広告、製品、あるいは資料に関して一切の責任を負いません。Sun は、これらのサイトあるいはリソースに関する、あるいはこれらのサイトから利用可能であるコンテンツ、製品、サービスのご利用あるいは信頼によって、あるいはそれに関連して発生するいかなる損害、損失、申し立てに対する一切の責任を負いません。

アクセシブルな製品マニュアル

マニュアルは、技術的な補足をすることで、ご不自由なユーザーの方々にとって読みやすい形式のマニュアルを提供しております。アクセシブルなマニュアルは以下の表に示す場所から参照することができます。製品ソフトウェアが /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。

マニュアルの種類	アクセシブルな形式と格納場所
マニュアル (サードパーティ製マニュアルは除く)	形式 : HTML 場所 : `http://docs.sun.com`
サードパーティ製マニュアル • 『Standard C++ Library Class Reference』 • 『標準 C++ ライブラリ・ユーザーズガイド』 • 『Tools.h++ クラスライブラリ・リファレンスマニュアル』 • 『Tools.h++ ユーザーズガイド』	形式 : HTML 場所 : `file:/opt/SUNWspro/docs/ja/index.html` のマニュアル索引
Readme およびマニュアルページ	形式 : HTML 場所 : `file:/opt/SUNWspro/docs/ja/index.html` のマニュアル索引
オンラインヘルプ	形式 : HTML 場所 : IDE 内の「ヘルプ」メニュー
リリースノート	形式 : HTML 場所 : `http://docs.sun.com`

コンパイラとツールに関する関連マニュアル

以下の表は、file:/opt/SUNWspro/docs/ja/index.html および http://docs.sun.com から参照できるマニュアルの一覧です。製品ソフトウェアが /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。

マニュアルタイトル	内容の説明
Fortran プログラミングガイド	入出力、ライブラリ、パフォーマンス、デバッグ、並列処理などに関する、Solaris 環境における効果的な Fortran コードの書き方について説明しています。
Fortran ライブラリ・リファレンス	Fortran ライブラリと組み込みルーチンについて詳しく説明しています。
Fortran ユーザーズガイド	f95 コンパイラのコンパイル時環境とコマンド行オプションについて説明しています。古い f77 プログラムを f95 に移行させる際の指針も含まれています。
C ユーザーズガイド	cc コンパイラのコンパイル時環境とコマンド行オプションについて説明しています。
C++ ユーザーズガイド	CC コンパイラのコンパイル時環境とコマンド行オプションについて説明しています。
OpenMP API ユーザーズガイド	OpenMP 多重処理 API の概要と実装に関する詳細について説明しています。
数値計算ガイド	浮動小数点演算における数値の正確性に関する問題について説明しています。

関連する Solaris マニュアル

次の表では、docs.sun.com の Web サイトで参照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
Solaris Reference Manual Collection	マニュアルページのセクションのタイトルを参照。	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 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 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。

開発者向けのリソース

http://developers.sun.com/prodtech/cc にアクセスし、以下のようなリソースを利用できます。リソースは頻繁に更新されます。

- プログラミング技術と最適な演習に関する技術文書
- プログラミングに関する簡単なヒントを集めた知識ベース
- コンパイラとツールのコンポーネントのマニュアル、ソフトウェアとともにインストールされるマニュアルの訂正
- サポートレベルに関する情報
- ユーザーフォーラム
- ダウンロード可能なサンプルコード
- 新しい技術の紹介

http://developers.sun.com でも開発者向けのリソースが提供されています。

技術サポートへの問い合わせ

製品についての技術的なご質問がございましたら、以下のサイトからお問い合わせください (このマニュアルで回答されていないものに限ります)。

http://jp.sun.com/service/contacting

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『Fortran プログラミングガイド』で説明する Sun™ Studio Fortran 95 コンパイラ (`f95`) は、SPARC®、UltraSPARC®、および x86 プラットフォーム上の Solaris オペレーティングシステムで使用可能です。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、洗練された最適なコードの出力、C 言語と Fortran 言語の混在のサポートなどの拡張機能を提供しています。

また、`f95` コンパイラには、従来の FORTRAN 77 ソースコードのほとんどが使用可能な FORTRAN 77 互換性モードもあります。単体の FORTRAN 77 コンパイラの提供はありません。FORTRAN 77 の互換性および移行問題については、第 5 章を参照してください。

1.1 規格への準拠

- `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。
- 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。
- `f95` は、Solaris プラットフォームでの SPARC® および x86 ファミリ (UltraSPARC®、SPARC64、AMD64、Pentium、Xeon EM64T) プロセッサアーキテクチャの機能を利用した最適化をサポートしています。
- 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外の機能は、「非標準」あるいは「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。

1.2 Fortran 95 コンパイラの特徴

Sun Studio Fortran 95 コンパイラには、次の特徴と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
- マルチプロセッサシステムのための、最適化された明示的自動ループ並列化機能
- VAX/VMS Fortran 拡張機能
  - 構造体、記録、共用体、マップ
  - 再帰
- OpenMP 並列化指令
- `TASKCOMMON` を含む Cray 形式の並列化指令
- 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。
- `%VAL` を使用した値による呼び出し
- FORTRAN 77 と Fortran 95 プログラム、およびオブジェクトバイナリ間に互換性があります。
- 区間演算プログラミング
- ストリーム入出力を含む Fortran 2003 のいくつかの機能

ソフトウェアのリリースの際にコンパイラに追加された新機能あるいは拡張機能についての詳細は、付録 B を参照してください。

1.3 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、次のユーティリティを利用できます。

- **Sun Studio パフォーマンスアナライザ** - 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 桁目に Fortran のキャリッジ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 システムはキャリッジ制御を使用していないため、Fortran のキャリッジ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従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。
- **fdumpmod** - ファイルまたはアーカイブに含まれるモジュールの名前を表示するユーティリティ。詳細は、`fdumpmod`(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。

1.4 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。

- **-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。
- **dbx** - 多機能で安定した実行時および静的デバッガを提供します。パフォーマンスデータコレクタも含まれます。

1.5 Sun Performance Library

Sun Performance Library™ は、線形代数やフーリエ変換の数値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` README ファイル、『Sun Performance Library User's Guide』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)。

1.6 区間演算

Fortran 95 のコンパイラは、新しい 2 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。(区間演算機能は、SPARC および UltraSPARC プラットフォームでのみサポートされます。)

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。

1.7 マニュアルページ

オンラインマニュアル (man) ページで、コマンド、関数、サブルーチンに関する説明を簡単に参照することができます。Sun Studio 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個所では、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。

`f95`(1)	Fortran 95 のコマンド行オプション
`analyzer`(1)	パフォーマンスアナライザ
`asa`(1)	Fortran キャリッジ制御印刷出力ポストプロセッサ
`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
fdumpmod(1)	MODULE (.mod) ファイルの内容を表示する
fsplit(1)	Fortran ソースルーチンを単一ファイルに分割するプリプロセッサ
ieee_flages(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

1.8 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、/opt/SUNWspro/READMEs/ja です。

表 1-1 重要な README

README ファイル	内容
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
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
performance_library	Sun Performance Library

各コンパイラの README ファイルは、-xhelp=readme コマンド行オプションで簡単に表示できます。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。

1.9 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、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。

```
%f95 -help=flags
[ ] 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。
-someoption[={yes|no}] の場合、-someoption は -someoption=yes と同等です。
_____________________________________________________________________
-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
-aligncommon[=<a>]   共通ブロックエレメントを指定された
                     境界に整列させる。<a>={1|2|4|8|16}
-ansi                ANSI 規格以外の拡張機能を報告
-autopar             自動選択によるループの並列化
-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
...etc
```

第2章

Fortran 95 の使用

この章では、Fortran 95 コンパイラ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。

Fortran 95 コンパイラは、次の処理にも使用します。

- 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 (`-openmp` 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)。
- 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)。

2.1 クイックスタート

ここでは Fortran 95 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。

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 : このプログラムは画面上にメッセージを表示します。

```
demo% cat greetings.f
      PROGRAM GREETINGS
      PRINT *, 'Real programmers write Fortran!'
      END
demo% f95 greetings.f
demo% a.out
 Real programmers write Fortran!
demo%
```

この例では、f95 がソースファイル greetings.f をコンパイルし、デフォルトで 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 というデフォルトのファイルを生成します。コンパイルごとに同じファイルに書き込みを行うと、不具合が生じる場合があります。さらには、そのようなファイルがすでに存在している場合、コンパイラの次の実行で上書きされてしまいます。代わりに、-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% f95 -o greetings greetings.f
demo% greetings
 Real programmers write Fortran!
demo%
```

上の例で、-o オプションを使用することにより、コンパイラは実行可能なコードをファイル greetings に書き込みます (規則により、実行可能ファイルはメインソースファイルと同じ名前を与えられますが、拡張子は付きません)。

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、シェルプロンプトで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。

以下の項では、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。

2.2 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。

f95 [*options*] *files...*

ここで、*files...* には、拡張子として .f、.F、.f90、.f95、.F90、.F95、または .for が付いている 1 つ以上の Fortran のソースファイル名を指定します。*options* 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 – f95 または **f90** コマンドのいずれを使用しても、Fortran 95 コンパイラを起動できます。

新規：コンパイラは、拡張子が .f03 または .F03 のソースファイルも受け付けます。これらは、.f95 および .F95 と同等とみなされ、ソースファイルに Fortran 2003 拡張機能が含まれていることを示す手段として利用できます。

コンパイラが受け付ける各種ソースファイルの拡張子については、2-4 ページの 2.2.2 節「ファイル名の拡張子」を参照してください。

2.2.1 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を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。

2.2.2 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 95 コンパイラが認識可能なファイル名の拡張子

拡張子	言語	コンパイラの動作
`.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
`.f03`	Fortran 2003 自由形式	`.f` と同じ
`.for`	FORTRAN 77 または Fortran 95	`.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 がコンパイルする前に、Fortran 95 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
`.F03`	Fortran 2003 自由形式	`.F95` と同じ
`.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
`.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
`.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 (テンプレートファイルは特殊なアセンブラファイル。`inline`(1) マニュアルページを参照)。
`.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 自由形式については、第 4 章を参照してください。

2.2.3 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、1 つのファイルに 1 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 1 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。

2.2.4 ソースファイルプリプロセッサ

`f95` は、`fpp` と `cpp` の 2 つの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、デフォルトでは、`.F`、`.F90`、`.F95`、または `.F03` という拡張子の付いたファイル上で起動します。

`fpp` のソースコードは、次の Netlib Web サイトにあります。

```
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。

2.2.5 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します。

```
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。

2.2.6 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。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なってください。すべてのソースファイルを、リンクも含めて指定してコンパイルする必要のあるオプションがあります。

第 3 章では、このようなオプションについて説明します。

例 : `sbr.f` を `-fast` でコンパイルし、C ルーチンをコンパイルします。次に、別途、リンクを行います。

```
demo% f95 -c -fast sbr.f
demo% cc -c -fast simm.c
demo% f95 -fast sbr.o simm.o          リンク: -fast をリンカーに渡す
```

2.2.7 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。

たとえば、次のとおりです。

```
demo% f95 -bit move.f                <- -bit は f95 オプションとして認識
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:
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、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。

2.2.8 Fortran 95 モジュール

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情報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。

モジュール情報ファイルを記述および検索するためのデフォルトのパスの設定方法については、-M *path* および -moddir *dirlist* オプションフラグを参照してください。

すべてのコンパイルユニットで暗黙的に MODULE 宣言を行う方法については、-use コンパイラオプションを参照してください。

fdumpmod(1) コマンドを使用すると、.mod モジュール情報ファイルの内容を表示できます。

詳細は、4-23 ページの 4.9 節「モジュールファイル」を参照してください。

2.3 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、4-21 ページの 4.8 節「指令」で説明します。`f95` が認識可能なすべての指令については、付録 D を参照してください。

注 – 指令は Fortran 規格には含まれていません。

2.3.1 一般的な指令

一般的な Fortran 95 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。
- ！という注釈指示子文字を使用する指令は、行のどの桁にも記述できます。

制限事項は、次のとおりです。

- Fortran テキストの場合と同様、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 f95 の指令は認識し、指令キーワード外の制限付き置換は実行します。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 指令の要約

C **指令**	C$PRAGMA C(*list*) 外部関数の名前リストを C 言語のルーチンとして宣言します。
IGNORE_TKR **指令**	C$PRAGMA IGNORE_TKR {*name* {, *name*} ...} コンパイラは、特定の呼び出しを解釈するとき、一般的な手続きのインタフェースで表示される仮引数名の型、種類、ランクを無視します。
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* の別名として宣言します。
OPT **指令**	C$PRAGMA SUN OPT=*n* サブプログラムの最適化レベルを *n* に設定します。
PIPELOOP **指令**	C$PRAGMA SUN PIPELOOP=*n* 次のループでは *n* 離れた反復間に依存関係があることを宣言します。

表 2-2　一般的な Fortran 指令の要約 (続き)

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$PRAGMA SPARC_PREFETCH_WRITE_MANY(*name*) 名前の参照のために、先読み命令を生成するようにコンパイラに要求します。-xprefetch オプションを指定する必要があります。
ASSUME 指令	C$PRAGMA [BEGIN] ASSUME (*expression* [, *probability*]) C$PRAGMA END ASSUME プログラム内の特定の個所において、コンパイラが真であると想定できる条件について表明を行います。

2.3.1.1 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 と 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)
```

2.3.1.2 IGNORE_TKR 指令

この指令では、コンパイラは、特定の呼び出しを解釈するとき、総称手続きのインタフェースで表示される仮引数名の型、種別、次元数を無視します。

たとえば、次の手続き引用仕様では、SRC はどのようなデータ型でもよく、LEN は KIND=4 または KIND=8 のいずれかであることが指定されます。

インタフェースブロックは、汎用的な手順名に対し 2 つの特定の手順を定義します。この例は、Fortran95 自由形式で示されます。

```
INTERFACE BLCKX

SUBROUTINE BLCK_32(LEN,SRC)
  REAL SRC(1)
!$PRAGMA IGNORE_TKR SRC
  INTEGER (KIND=4) LEN
END SUBROUTINE

SUBROUTINE BLCK_64(LEN,SRC)
  REAL SRC(1)
!$PRAGMA IGNORE_TKR SRC
  INTEGER (KIND=8) LEN
END SUBROUTINE

END INTERFACE
```

サブルーチンの呼び出し

```
INTEGER L
REAL S(100)
CALL BLCKX(L,S)
```

BLCKX の呼び出しによって、一般的なコンパイルでは BLCK_32 が呼び出され、-xtypemap=integer:64 を使用してコンパイルした場合は BLCK_64 が呼び出されます。S の実際の型は、どのルーチンを呼び出すかを定義しません。これによって、引数の型、種別、次元数に基づいてライブラリルーチンを呼び出すラッパーの一般的なインタフェースの記述を単純化できます。

形状引き継ぎの配列、Fortran ポインタ、割り当て可能な配列の仮引数は、指令では指定できません。名前が指定されていない場合は、形状引き継ぎの配列、Fortra ポインタ、割り当て可能な配列の仮引数を除いて、手続きのすべての仮引数に指令が適用されます。

2.3.1.3 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 は正の整数です。次の選択が可能です。

- *n*=1 の場合、オプティマイザは、どのループも展開しない可能性があります。
- *n*>1 の場合、オプティマイザは、ループを *n* 回展開する可能性があります。

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
```

2.3.1.4 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 (*name1*) によって、*name1* が優先順位の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、また、*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。

2.3.1.5 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。たとえば、次のように指定したとします。

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

-O4 を指定する f95 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 -O2 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 -O4 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。

2.3.1.6 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。

2.3.1.7 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、ループの反復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 (ループの伝達性) 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     K の値が存在しないかもしれないことはわかっています
C     相互反復の依存 (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。

2.3.1.8 PREFETCH 指令

-xprefetch オプションフラグを使用すると (3-91 ページの「-xprefetch[=a[,a]]」参照)、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。

2.3.1.9 ASSUME 指令

ASSUME 指令は、プログラムの特定地点の条件についてコンパイラにヒントを与えます。これらの表明は、コンパイラへの最適化指示のガイドラインとして役立ちます。また、プログラマは、それらの指令を使用して、実行時にプログラムの妥当性をチェックできます。ASSUME のフォーマットは 2 種類あります。

「単一表明」ASSUME の構文は以下のようになります。

```
C$PRAGMA ASSUME (expression [, probability])
```

また、「範囲表明」ASSUME は以下のようになります。

```
C$PRAGMA BEGIN ASSUME [expression [, probability)
    ステートメントのブロック
C$PRAGMA END ASSUME
```

単一表明形式を使用すると、プログラムのその地点でコンパイラが想定できる条件を示すことができます。範囲表明形式を使用すると、ステートメントの範囲内を通して成立する条件を示すことができます。範囲表明の BEGIN と END のペアは正しくネストされる必要があります。

必要な**式**は、上にリストされている以外でユーザー定義の演算子や関数呼び出しを含まないプログラムの特定地点で評価可能なブール式です。

オプションの *probability* 値は、0.0 から 1.0 までの実数、つまり整数の 0 または 1 であり、式が真となる可能性を示します。0.0 (または 0) の可能性は絶対に真にならないことを意味し、1.0 (または 1) は常に真になることを意味します。数値の指定がない場合、式は高い可能性で真とみなされますが、絶対ではありません。0 または 1 以外の可能性を持つ表明は「非確定表明」です。同様に、0 または 1 の可能性を持つ表明は「確定表明」です。

たとえば、DO ループが常に 10,000 より長いことがわかっている場合は、コンパイラに示しておくと、より良いコードを生成できます。通常、以下のループは、ASSUME プラグマがある場合の方がすばやく実行されます。

```
C$PRAGMA BEGIN ASSUME(__tripcount().GE.10000,1) !! a big loop
        do i = j, n
           a(i) = a(j) + 1
        end do
C$PRAGMA END ASSUME
```

特に ASSUME 指令の式クローズで使用するために 2 つの組み込み関数が用意されています (それらの名前の前には 2 つの下線が配置されます)。

__branchexp()	ブール制御式を持つ分岐ステートメントの直前に配置された単一表明で使用します。分岐ステートメントを制御するブール式と同じ結果を生成します。
__tripcount()	指令の直後または指令に閉じ込められたループのトリップカウントを生成します。単一表明で使用する場合、指令直後のステートメントは DO の最初の行となる必要があります。範囲表明で使用する場合、もっとも外側に閉じ込められたループに適用します。

この特殊な組み込み関数のリストは、将来的なリリースで拡大する可能性があります。

-xassume_control コンパイラオプションとともに使用します (3-66 ページの「-xassume_control[=keywords]」を参照してください)。たとえば、-xassume_control=check を使用してコンパイルした場合、トリップカウントが 10,000 を下回ると警告が発せられます。

-xassume_control=retrospective を使用してコンパイルを実行すると、プログラムの終了地点ですべての表明の真と偽を示す要約レポートが生成されます。-xassume_control の詳細については、f95 のマニュアルページを参照してください。

もう 1 つの例

```
C$PRAGMA ASSUME(__tripcount.GT.0,1)
       do i=n0, nx
```

-xassume_control=check を使用して上記の例をコンパイルすると、トリップカウントが 0 かマイナスになるため、そのループを使用しないよう実行時の警告が発せられます。

2.3.2 並列化指令 (SPARC)

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、-openmp、-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 並列化についての詳細は、『OpenMP API ユーザーズガイド』および『Fortran プログラミングガイド』を参照してください。

Fortran コンパイラは、OpenMP 共有メモリー並列化モデル、従来の Sun および Cray 指令をサポートしています。

現時点では、x86 プラットフォームでは、コンパイラの並列化機能は使用できません。

2.3.2.1 OpenMP 並列化指令

Fortran 95 コンパイラでは、並列プログラミングモデルとして OpenMP Fortran 共有メモリー多重処理 API を使用することをお勧めします。API は、OpenMP Architecture Review Board (http://www.openmp.org) によって指定されます。

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 (3-44 ページの「-openmp[={parallel|noopt|none}]」を参照してください)。

f95 で使用可能な OpenMP 指令についての詳細は、『OpenMP API ユーザーズガイド』を参照してください。

2.3.2.2 従来の Sun および Cray 並列指令

Sun 形式の並列指令は、-parallel および -explicitpar のデフォルトです。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。また、従来の Sun/Cray 並列化指令を OpenMP 指令に変換する方法については、『OpenMP API ユーザーズガイド』も参照してください。

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 桁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 $PAR (Sun 形式) または MIC$ (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
```

2.4 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)。

2.5 コンパイラの利用方法

Fortran 95 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。すべてのコンパイラオプションのリファレンスは、次の章に示します。

2.5.1 ハードウェアプラットフォームの特定 (SPARC)

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 SPARC 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.
 Kernel says CPU's clock rate is 750.0 MHz.
 Kernel says main memory's clock rate is 150.0 MHz.

 Sun-4 floating-point controller version 0 found.
 An UltraSPARC chip is available.

 Use "-xtarget=ultra3" 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。x86 では、fpversion は使用できません。

2.5.2 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 : FFLAGS 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。

```
f95 -fast -Xlist files...
```

make はサンのすべてのコンパイラで使用できる強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。

2.5.3 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。

コンパイラを実行するプロセッサには最低 64M バイトのメモリーが実装されている必要があります。256M バイトが推奨メモリーです。また、十分なスワップ領域が割り当てられる必要もあります。最低 200M バイトで、300M バイトが推奨値です。

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。

2.5.3.1 スワップ領域の制限

swap -s コマンドは利用可能なスワップ領域を表示します。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
```

2.5.3.2 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
```

2.5.3.3 仮想メモリーの制御

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 つの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 シェルでは、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 シェルでは、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。

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 シェルの場合 :

```
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 2G バイトです。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を解除します。

csh の場合 :

```
demo% limit datasize unlimited
```

sh、ksh の場合 :

```
demo$ ulimit -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、f95 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。

- コンパイラオプションフラグに使用する構文の説明 : 3-1 ページの 3.1 節「コマンド構文」
- 機能別のオプションのまとめ : 3-3 ページの 3.3 節「オプションのまとめ」
- 各コンパイラオプションフラグに関する詳細リファレンス : 3-12 ページの 3.4 節「オプションリファレンス」

3.1 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。

```
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。

3.2 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します。	`-O [n]` `-O4、-O`
{ }	中括弧は、必須の引数を表します。	`-d{y|n}` `-dy`
|	縦棒記号は、いずれか *1* つを選択する引数を表します。	`-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ります。	`-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します。	`-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。
- -xhasc[={yes|no}] などの複数の選択肢リストの最初の選択肢は、コマンド行に値なしでオプションのフラグが指定された場合に適用される値です。たとえば -xhasc は -xhasc=yes と指定するのと同じことです。
- 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。

3.3 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、以下の詳細欄に示すページを参照してください。

SPARC および x86 プラットフォーム両方ですべてのオプションを使用できるわけではないことに注意してください。使用可能かどうかについては、詳細オプションリファレンスの節で確認してください。

次の表に、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、複数の個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ません。	-c
ドライバが作成するコマンドを表示するが、コンパイルは行われません。	-dryrun
FORTRAN 77 拡張子および互換性をサポートします。	-f77
コンパイルされる .mod モジュールファイルを記述するためのパスを指定します。	-moddir=*path*
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します。	-o *filename*

表 3-3　機能別コンパイラオプション (続き)

機能	オプションフラグ
コンパイルし、アセンブリコードだけを生成します。	`-S`
実行可能プログラムからシンボルテーブルを除外します。	`-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ません。	`-silent`
一時ファイルのディレクトリへのパスを定義します。	`-temp=`*path*
各コンパイルフェーズの経過時間を示します。	`-time`
コンパイラおよびそのフェーズのバージョン番号を示します。	`-V`
冗長メッセージを表示します。	`-v`
標準外の別名を付ける状況を指定します。	`-xalias=`*list*
マルチプロセッサによるコンパイル	`-xjobs=`*n*
コンパイルされるコード	
外部名の末尾に下線を追加/抑制します。	`-ext_names=`*x*
インライン化するユーザー関数を指定します。	`-inline=`*list*
コンパイル位置独立コードを指定します。	`-KPIC/-kpic`
特定の数学ライブラリルーチンをインライン化します。	`-libmil`
STOP で整数のステータス値をシェルに返します。	`-stop_status[=`*yn*`]`
コードアドレス空間を指定します。	`-xcode=`*x*
UltraSPARC の先読み命令を有効にします。	`-xprefetch[=`*x*`]`
オプションのレジスタを指定します。	`-xregs=`*x*
デフォルトのデータマッピングを指定します。	`-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します。	`-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/ストアを可能にします。	`-dalign`
全データを 8 バイト境界に強制的に整列させます。	`-dbl_align_all`
COMMON ブロックデータを 8 バイト境界に整列させます。	`-f`
メモリーの境界整列と動作を指定します。	`-xmemalign[=`*ab*`]`

表 3-3　機能別コンパイラオプション (続き)

機能	オプションフラグ
デバック	
実行時に添字の範囲検査を有効にします。	`-C`
dbx を使用するデバッグのためにコンパイルします。	`-g`
ソースブラウザを使用するブラウズのためにコンパイルします。	`-sb, -sbfast`
未宣言変数の検査を行います。	`-u`
C$PRAGMA ASSUME 表明をチェックします。	`-xassume_control=check`
実行時のスタックオーバーフローを確認します。	`-xcheck=stkovf`
実行時の taskcommon の整合性検査を有効にします。	`-xcommonchk`
パフォーマンスアナライザのためにコンパイルします。	`-xF`
相互参照リストを作成します。	`-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します。	`-xs`
診断	
非標準の拡張機能を報告します。	`-ansi`
特定のエラーメッセージの出力を抑制します。	`-erroff=`*list*
エラーメッセージとともにエラータグ名を表示します。	`-errtags`
コンパイラオプションの要約を表示します。	`-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します。	`-V`
冗長メッセージを表示します。	`-v`
並列化メッセージを冗長表示します。	`-vpara`
警告メッセージを表示/抑制します。	`-w`*n*
コンパイラの README ファイルを表示します。	`-xhelp=readme`
ライセンス	
ライセンスサーバー情報を表示します。	`-xlicinfo`
リンクおよびライブラリ	
動的/静的ライブラリを許可します/要求します。	`-B`*x*
動的/静的なライブラリのみのリンクを許可します。	`-dy, -dn`

表 3-3　機能別コンパイラオプション (続き)

機能	オプションフラグ
動的 (共有オブジェクト) ライブラリを作成します。	`-G`
動的ライブラリの名前を指定します。	`-h`*name*
ディレクトリをライブラリ検索パスに追加します。	`-L`*path*
`lib`*name*`.a` または `lib`*name*`.so` というライブラリをリンクします。	`-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みます。	`-R`*path*
インクリメンタルリンカー `ild` を使用不可にします。	`-xildoff`
最適化数学ライブラリをリンクします。	`-xlibmopt`
Sun のパフォーマンスライブラリをリンクします。	`-xlic_lib=sunperf`
リンクエディタのオプションを指定します。	`-z`*x*
再配置のない閉じたライブラリを生成します。	`-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します。	`-fnonstd`
SPARC 非標準浮動小数点を選択します。	`-fns`
入力中に実行時浮動小数点オーバーフロー検査を有効にします。	`-fpover`
IEEE 浮動小数点丸めモードを選択します。	`-fpround=`*r*
浮動小数点最適化レベルを選択します。	`-fsimple=`*n*
浮動小数点トラップモードを選択します。	`-ftrap=`*t*
書式付き入出力のための丸め方法を指定します。	`-iorounding=`*mode*
単精度定数を倍精度に変換します。	`-r8const`
区間演算を有効にし、適切な浮動小数点環境を設定します (`-xinterval` を含む)。	`-xia`[`=`*e*]
区間演算機能を有効にします。	`-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ます。	`-depend`
オプションを一括で指定して最適化します。	`-fast`
最適化レベルを指定します。	`-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します。	`-pad`[`=`*p*]

表 3-3　機能別コンパイラオプション (続き)

機能	オプションフラグ
局所変数をメモリースタックに割り当てます。	`-stackvar`
ループ展開を有効にします。	`-unroll`[=*m*]
ソースファイル間での最適化を有効にします。	`-xcrossfile`[=*n*]
内部手続きの最適化パスを呼び出します。	`-xipo`[=*n*]
`#pragma OPT` に最高レベルの最適化を設定します。	`-xmaxopt`[=*n*]
コンパイラが生成する先読み命令を有効にするか、または調整します。	`-xprefetch=`*list*
先読み命令の自動生成をコントロールします。	`-xprefetch_level=`*n*
パフォーマンスプロファイルデータの生成または使用を有効にします。	`-xprofile=`*p*
メモリーベースのトラップが発生しないであろうと表明します。	`-xsafe=mem`
コードサイズが増加する場合は、最適化を行いません。	`-xspace`
ベクトルライブラリ関数の呼び出しを自動的に作成します。	`-xvector`[=*yn*]
並列化	
DO ループの自動並列化を有効にします。	`-autopar`
指令で明示的に指定したループの並列化を有効にします。	`-explicitpar`
ループの並列化情報を表示します。	`-loopinfo`
Cray 形式の並列化指令を指定します。	`-mp=CRAY`
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します。	`-mt`
OpenMT API 指令を受け付け、適切な環境を設定します。	`-openmp`[=*keyword*]
`-autopar -explicitpar -depend` の組み合わせでループを並列化します。	`-parallel`
自動並列化でループ内の縮約操作を認識します。	`-reduction`
並列化メッセージを冗長表示します。	`-vpara`
ソースコード	
プリプロセッサのシンボルを定義します。	`-D`*name*[=*val*]
プリプロセッサのシンボルの定義を取り消します。	`-U`*name*
拡張 (132 文字) ソース行を受け入れます。	`-e`

表 3-3　機能別コンパイラオプション (続き)

機能	オプションフラグ
.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、コンパイルは行いません。	-F
固定書式の Fortran 95 ソースを受け付けます。	-fixed
すべてのソースファイルを fpp プリプロセッサで先行処理します。	-fpp
自由形式の Fortran 95 ソースを受け付けます。	-free
ファイル検索パスにディレクトリを追加します。	-I*path*
モジュール検索パスにディレクトリを追加します。	-M*path*
大文字と小文字を区別します。	-U
ホレリスを実際の引数の文字として扱います。	-xhasc={yes|no}
使用するプリプロセッサ (cpp または fpp) を選択します。	-xpp[={fpp|cpp}]
再帰的な副プログラム呼び出しを許可します。	-xrecursive
ターゲットプラットフォーム	
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します。	-xarch=*a*
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します。	-xcache=*a*
オプティマイザにターゲットのプロセッサを指定します。	-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します。	-xtarget=*a*

3.3.1 頻繁に利用するオプション

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

作用	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ます。	`-Xlist`
デバッグ - dbx および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します。	`-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します。	`-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します。	`-fast`
動的 (`-Bdynamic`) または静的 (`-Bstatic`) ライブラリとリンクします。	`-B`*x*
コンパイルのみ - リンクを行わず、ソースファイルごとに `.o` ファイルを作成します。	`-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* に指定します。	`-o nm`
ソースコード - 固定形式 Fortran ソースコードをコンパイルします。	`-fixed`

3.3.2 マクロフラグ

マクロフラグによっては、別のフラグの組み合わせに展開されるマクロもあります。これらのマクロフラグは、ある機能を選択するために、通常一緒に表示されるオプションを簡単に指定できるように提供されるものです。

表 3-5 マクロオプションフラグ

オプションフラグ	展開
`-dalign`	`-xmemalign=8s -aligncommon=16`
`-f`	`-aligncommon=16`
`-fast`	`-xO5 -libmil -fsimple=2 -dalign -xlibmopt -depend -fns -ftrap=common -pad=local -xvector=yes -xprefetch=yes` *(SPARC)* `-xprefetch_level=2` *(SPARC)* `-nofstore` *(x86)*
`-fnonstd`	`-fns -ftrap=common`

表 3-5　マクロオプションフラグ (続き)

オプションフラグ	展開
-parallel	-autopar -explicitpar -depend
-xia=widestneed	-xinterval=widestneed -ftrap=%none -fns=no -fsimple=0
-xia=strict	-xinterval=strict -ftrap=%none -fns=no -fsimple=0
-xtarget	-xarch=*a* -xcache=*b* -xchip=*c*

コマンド行でマクロフラグの後に別のオプションを設定すると、このマクロの展開内容は上書きされます。たとえば、-fast を最適化レベル -O3 ともに使用する場合は、-O3 はコマンド行の -fast よりも後ろに記述する必要があります。

3.3.3　下位互換のための旧オプション

コンパイラの初期リリース、および Fortran の一部旧機能との下位互換のためのオプションを示します。

表 3-6　下位互換性オプション

作用	オプション
定数の引数への代入を可能にします。	-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います。	-xhasc[={yes|no}]
FORTRAN 77 拡張子および規則をサポートします。	-f77
非標準の算術演算 - 非標準の算術演算を使用可能にします。	-fnonstd
ホストシステムに合わせて最適化を行います。	-native
DO ループ - 少なくとも 1 回は DO ループを実行します。	-onetrip
従来の別名を付ける状況を許可します。	-xalias=*keywords*

移植性のある Fortran 95 プログラムを作成する際には、これらのオプションフラグは使用しないでください。

3.3.4 旧オプションフラグ

次のオプションは廃止されています。使用しないでください。将来のコンパイラでは、これらのオプションは削除される予定です。

表 3-7 旧 f95 オプション

オプションフラグ	同等なオプションフラグ
-a	-xprofile=tcov
-cg89	-xtarget=ss2
-cg92	-xtarget=ss1000
-native	-xtarget=native
-noqueue	ライセンスのキューイング。現在は必要ありません。
-p	プロファイリング。-pg またはパフォーマンスアナライザを使用してください。
-pic	-xcode=pic13
-PIC	-xcode=pic32
-sb	現在は必要ありません。
-sbfast	現在は必要ありません。
-silent	現在は必要ありません。

3.4 オプションリファレンス

この節では、すべての f95 コンパイラコマンド行オプションフラグについて説明します。これには、さまざまなリスク、制約、警告、相互作用、例、およびその他の詳細情報も含まれます。

各オプションは、特に付記していないかぎり、SPARC および x86 プラットフォームの両方で有効です。SPARC プラットフォームでのみ有効なオプションフラグは **(SPARC)** と付記しています。x86 プラットフォームでのみ有効なオプションフラグは **(x86)** と付記しています。

(廃止) と付記されているオプションフラグは廃止されており、使わないでください。多くの場合、代わりに使用すべき他のオプションまたはフラグに置き換えられています。

-a

(廃止) 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。

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。新しいプロファイリング方法については、-xprofile=tcov を参照してください。詳細は、tcov(1) マニュアルページを参照してください。また、マニュアル『プログラムのパフォーマンス解析』も参照してください。

-aligncommon[={1|2|4|8|16}]

共通ブロックおよび標準の数値連続型のデータの境界整列を指定します。

指定された値は、共通ブロックおよび標準数値連続型内のデータ要素の整列の最大値 (単位はバイト) を示します。

注 – 標準の数値連続型とは、SEQUENCE 文 1 つとデフォルトの要素データ型 (KIND= または **size* のどちらも付かない INTEGER、REAL、DOUBLEPRECISION、COMPLEX) からなる構造型です。REAL*8 などの他のすべての型は非標準の型になります。

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および数値連続型のデータは、多くても 4 バイト境界に整列されます。

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、共通ブロックおよび数値連続型の要素は、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

標準外の拡張機能を識別します。

ソースコード中で、標準外の 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 文に対する実際の引数を保持します。

このオプションを使って代替エントリポイントを持つサブプログラムをコンパイルする場合、f95 は、copy または restore を使用して、ダミー引数と実際の引数の関連付けを保持します。

このオプションは、従来の Fortran77 プログラムとの互換性のために用意されています。このオプションに依存するコードは、標準外です。

-autopar

ループの自動並列化を使用可能にします。

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。

-autopar オプションなどの並列化オプションを使用している場合は、-stackvar オプションも指定します。

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。詳細は、3-40 ページの「-mt」の注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。

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、PARALLEL 変数または OMP_NUM_THREADS 変数には、ターゲットプラットフォーム上の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}

動的または静的のいずれかライブラリリンクを指定します。

-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。

コマンド行で、-Bstatic と -Bdynamic を切り替えることができます。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。

```
f95 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 -B*x* 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 -B*x* 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

実行時に、配列の添字の範囲および適合性を検査します。

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 は、実行時に、配列の構文が適合しているかも検査します。

-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。

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

(廃止、SPARC) 一般的な SPARC アーキテクチャ用にコンパイルを行います。

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

(廃止、SPARC) SPARC V8 アーキテクチャ用にコンパイルを行います。

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。

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 *name* を定義します。

このオプションは .F、.F90、.F95、および .F03 ソースファイルだけに適用します。

-D*name*=*def*: *name* が値 *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。

このオプションはコマンド行で *name* を、

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

とソースファイルに記述されている場合のように定義します。=*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 2 つの下線を付けます。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 (初めに付く 2 つの下線に注意)。

- 製品バージョンは、__SUNPRO_F90、および __SUNPRO_F95 に 16 進数で事前定義されています。たとえば、__SUNPRO_F95 は、Sun Studio 10 リリースでは 0x810 です。
- 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、__i386 、
  __SunOS_5_6、__SunOS_5_7、__SunOS_5_8、__SunOS_5_9、
  __SunOS_5_10

  たとえば、SPARC システム上では、__sparc 値が定義されています。

- 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。
- 64 ビット x86 システムでは、__amd64 および __x86_64 マクロが定義されています。

コンパイラが作成した定義を表示するには、冗長メッセージオプション (-v) を付けてコンパイルします。

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
```

f95 は、デフォルトで 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。ただし、cpp とは異なり、fpp は Fortran 構文を理解できるので、Fortran プリプロセッサとしてはこちらを使用することをお勧めします。-xpp=cpp フラグを使すると、コンパイラは fpp ではなく cpp を使用します。

-dalign

COMMON ブロックおよび標準の数値連続型の整列を行い、高速なマルチワードのロード/ストアを生成します。

このフラグを使用すると、COMMON ブロック、数値連続型、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、移植性のないプログラムになります。

-dalign は、以下と等価なマクロです。

SPARC プラットフォームの場合、-xmemalign=8s -aligncommon=16

32 ビット x86 プラットフォームの場合、-aligncommon=8

64 ビット x86 プラットフォームの場合、-aligncommon=16

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。

-dalign は、-aligncommon を呼び出すので、標準の数値連続型も影響を受けます。3-12 ページの「-aligncommon[={1|2|4|8|16}]」を参照してください。

-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。

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。

-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[={yes|no}]

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。

依存解析は、-depend または -depend=yes によって使用可能になります。この解析は、-depend=no によって使用不可になります。これがコンパイラのデフォルト値です。

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、-fast、-autopar および -parallel オプションでも行われます。最適化レベル -O3 以上を指定した場合も、-depend が自動的に追加されます。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。3-19 ページの「-d{y|n}」を参照してください。

-dryrun

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。

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}

実行可能ファイル全体に対して、動的ライブラリ を使用可能または使用不可にします。

- -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。

-B*x* 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 はローダーおよび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。このため、64 ビット Solaris プラットフォームと 32 ビット Solaris x86 プラットフォーム、Solaris 10 リリース以降の 32 ビット Solaris プラットフォームのすべてで、-dn および -Bstatic がリンクエラーを引き起こすことがあります。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。

ソース行は、132 文字まで拡張できます。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に継続行を使用するときは、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[={**%all**|**%none**|*taglist*}]

タグ名によって一覧表示された警告メッセージを抑制します。

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。%all を指定した場合は、すべての警告メッセージが抑制されます。これは、-w オプションを指定するのと同じです。%none の場合、警告メッセージは抑制されません。

例 :

```
f95 -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[={**yes**|**no**}]

メッセージタグが各警告メッセージ付きで表示されます。

-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% f95 -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 を意味します。

-errwarn[={**%all**|**%none**|*taglist*}]

警告メッセージをエラーとして処理します。

taglist は、エラーとして処理する警告メッセージのコンマ区切りのタグ名リストです。%all を指定した場合は、すべての警告メッセージがエラーとして処理されます。%none の場合、警告メッセージはエラーとして処理されません。

-errtags も参照してください。

-explicitpar

Sun または Cray の指令で明示的に示されたループを並列化します。

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、またはこの両方の明示的な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。

Open/MP 明示的並列化指令を有効にするには、-explicitpar を使用しないでください。この代わりに -openmp を使用してください。3-44 ページの「-openmp[={parallel|noopt|none}]」を参照してください。

注 – -explicitpar 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、PARALLEL 変数または OMP_NUM_THREADS 変数には、ターゲットプラットフォーム上の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。-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 (3-39 ページの「-mp={%none|sun|cray}」) を使用して選択します。-explicitpar のデフォルトは Sun 指令です。Cray 指令を有効にするには、-explicitpar -mp=cray を使用してください。

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。

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=underscores: 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。このフラグを使用すると、Fortran 95 のルーチンから別のプログラム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。

.F、.F95、および .F03 ファイルに fpp プリ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、.f03) 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 :

```
f95 -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 ブロックの倍精度および 4 倍精度のデータを境界整列します。

-f は従来のオプションフラグで、-aligncommon=16 と同等です。-aligncommon を使用してください。

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 を使用すると、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。

注 – -f 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、-f が必要な場合、移植性のないプログラムになります。

-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、3-17 ページの「-dalign」を参照してください。これは、-dalign が -f と同様に -fast オプションの一部であるからです。

-f77[**=***list*]

FORTRAN 77 の互換性モードを選択します。

このオプションフラグによって、f77 コンパイラが使用可能な言語拡張機能を含むソースプログラムを含め、従来の FORTRAN 77 ソースプログラムの f95 (Fortran 95 コンパイラ) への移植が可能になります。

list は、以下のキーワードから選択された、コンマで区切られたリストです。

keyword	意味
%all	FORTRAN 77 のすべての互換性機能を有効にします。
%none	FORTRAN 77 のすべての互換性機能を無効にします。
backslash	文字列のバックスラッシュをエスケープシーケンスとして受け入れます。
input	f77 が受け付ける入力書式を許可します。
intrinsics	組み込み関数の認識を FORTRAN 77 組み込み関数にのみ制限します。
logical	以下に示す論理変数の FORTRAN 77 での使用法を受け入れます。 - 整数値を論理変数に割り当てる - 論理条件式の演算式を .TRUE. を表わす .NE.0 とともに使用する - 関連演算子 .EQ. および .NE. を論理演算子とともに使用する
misc	その他の f77 FORTRAN 77 拡張機能を許可します。

keyword	意味
output	並び出力および NAMELIST 出力を含む、f77 形式の出力を生成します。
subscript	配列添字として整数式以外の表現を許可する。
tab	無制限のソース行の長さを含む、f77 形式の TAB フォーマットを有効にします。72 文字未満のソース行に対して、空白文字のパディングは行われません。

すべてのキーワードは、no% を前に付けて無効にすることができます。

```
-f77=%all,no%backslash
```

-f77 が指定されない場合は、デフォルトとして -f77=%none が使用されます。リストなしの -f77 は、-f77=%all と同じ意味を持ちます。

例外トラップと -f77:

-f77 を指定すると、Fortran 95 のトラップモードが変更されず、-ftrap=common になります。Fortran 95 と FORTRAN 77 は、演算例外トラップの動作が異なります。FORTRAN 77 コンパイラは、演算例外が発生した後でも実行を継続することができます。-f77 によるコンパイルでも、プログラムはプログラム終了時に ieee_retrospective を呼び出して、演算例外が発生した場合はそれらの例外をすべて報告します。コマンド行の -f77 オプションフラグの後に -ftrap=none を指定すると、元の FORTRAN 77 の動作を真似ることができます。

f77 の互換性および FORTRAN 77 から Fortran 95 への移行についての詳細は、第 5 章を参照してください。

間違った結果を生じさせる可能性がある標準外のプログラミングの問題を処理する方法については、-xalias フラグも参照してください。

-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。

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は利用できません。-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、暗黙的にリンクするため、コンパイルとリンクを別々に行う場合は、-fast でコンパイルしたら、-fast を使用してリンクもしてください。

-fast では、次のオプションを選択します。

- -dalign
- -depend *(SPARC)*
- -fns
- -fsimple=2
- -ftrap=common
- -libmil
- -xtarget=native
- -O5
- -xlibmopt
- -pad=local *(SPARC)*
- -xvector=yes *(SPARC)*
- -xprefetch=yes
- -xprefetch_level=2
- -nofstore *(x86)*

-fast によって、選択されるオプションの詳細は、以下のとおりです。

- -xtarget=native ハードウェアターゲットオプション 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。

```
f95 -fast -xtarget=ultra ...
```

- -O5 最適化レベルオプション
- -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。*(SPARC)*
- 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。3-30 ページの「-fsimple[={1|2|0}]」を参照してください。

- 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。*(SPARC)*
- -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。
- -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。3-28 ページの「-fns[={yes|no}]」を参照してください。
- 共通の浮動小数点例外のトラッピング -ftrap=common は、Fortran 95 で有効です。
- -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。
- -xprefetch_level=2 は、先読み命令の挿入のデフォルトレベルを設定します。
- -nofstore は、式の精度を強制的に結果の精度にする設定を取り消します。*(x86)*

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。

-fast フラグで選択するオプションは、コンパイラのリリースによって変更されることがあります。

-fixed

固定書式の Fortran 95 ソース入力ファイルを指定します。

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。

-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。

このオプションは、以下のオプションフラグを組み合わせたマクロです。

```
-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。

-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。

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[={yes|no}]

非標準の浮動小数点モードを選択します。

デフォルトは標準の浮動小数点モード (-fns=no) です (『Fortran プログラミングガイド』 の「浮動小数点演算」の章を参照)。

-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。値を指定しない場合 -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、非標準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、非標準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「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」を表わすのに「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。

x86 プラットフォームの場合、このオプションは Pentium III および Pentium 4 (sse または sse2 命令セット) でのみ有効です。

-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。

-fpover=yes を指定すると、入出力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)。値を指定しない場合 -fpover は -fpover=yes と同じです。

-fpp

fpp を使用して、入力の前処理を強制的に行います。

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)。詳細は、3-91 ページの「-xpp={fpp|cpp}」を参照してください。

-fprecision={single|double|extended}

(x86) 非標準の浮動小数点丸め精度モードを初期設定します。

x86 で、浮動小数点精度モードを single か double、extended のいずれかに設定します。

single か double の場合、丸め精度モードは、プログラムの実行が始まるときに、それぞれ単精度、倍精度に設定されます。extended か、-fprecision が指定されなかった場合のデフォルトでは、丸め精度モードは拡張精度に初期設定されます。

このオプションは、x86 システム上でのみ、かつ、メインプログラムをコンパイルするときに使用する場合にのみ有効です。

-free

自由書式のソース入力ファイルを指定します。

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 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。

-fround={nearest|tozero|negative|positive}

起動時に IEEE の丸めモードを有効にします。

デフォルトは -fround=nearest です。

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。

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。

値が 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「浮動小数点演算」の章を参照)。

-fsimple[={1|2|0}]

浮動小数点の最適化の設定を選択します。

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「浮動小数点演算」の章を参照)。

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。

デフォルトは次のとおりです。

- -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。
- 値なしで -fsimple が指定された場合は、-fsimple=1 が使用されます。

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 は、-fsimple=2 を選択します。

-fstore

(x86) 浮動小数点式の精度を強制的に設定します。

代入文の場合、このオプションはあらゆる浮動小数点式を強制的に代入先の変数の精度にします。これはデフォルト値です。ただし、-fast オプションには、このオプションを無効にする -nofstore が含まれています。再びこのオプションを有効にするには、-fast の後に -fstore を続けてください。

-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。

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。

f95 のデフォルトは -ftrap=common です。これは、C および C++ コンパイラのデフォルト (-ftrap=none) と異なります。

起動時に IEEE 745 のトラップモードを有効にします。ただし、SIGFPE ハンドラは組み込まれません。トラップの設定と SIGFPE ハンドラのインストールを同時に行うには、ieee_handler(3M) か fex_set_handling(3M) を使用します。複数の値を指定した場合、それらの値は左から右に処理されます。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。

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 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「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。

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。-G を指定すると、動的ライブラリを構築します。出力ファイル名を指定するには、-G オプションとともに -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「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。

dbx(1) デバッグユーティリティによるデバッグ、および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 のマニュアルを参照してください。

注 – 以前のリリースのコンパイラでは、-g オプションは、コマンド行で .o オブジェクトファイルのみ指定されたとき、-xildon をデフォルトのインクリメンタルリンカーオプションにしていました (3-79 ページの「-xild{off|on}」を参照)。今後のリリースでは、この動作にするには、-xildon フラグを指定してコンパイルする必要があります。

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。(詳細については、analyzer(1) マニュアルページおよびマニュアル『Sun Studio 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、ソースコードに挿入されているため、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。-xO4、-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。

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「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* の間には空白文字があってもなくてもかまいません (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。

詳細は、3-79 ページの「-xhelp={readme|flags}」を参照してください。

-I*path*

INCLUDE ファイルの検索パスに *path* を追加します。

INCLUDE ファイルの検索パスの先頭に、ディレクトリパス *path* を挿入します。-I と *path* 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、INCLUDE 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 INCLUDE 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*path* 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 オプションで指定したディレクトリ
3. コンパイラのデフォルトの内部リストにあるディレクトリ
4. /usr/include/

プリプロセッサを呼び出すには、.F か .F90、.F95、.F03 拡張子付きのソースファイルをコンパイルする必要があります。

-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。

関数およびサブルーチン名のコンマ区切りのリストに指定されたユーザー作成のルーチンをインライン化するようオプティマイザに要求します。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。

%auto を指定すると、最適化レベル -xO4 または -xO5 での自動インライン化が有効になります。-inline で明示的なインライン化が指定されている場合、通常、これらの最適化レベルでの自動インライン化は無効になります。

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は次のとおりです。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、-xipo または -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。

-iorounding[={compatible|processor-defined}]

書式付き入出力の浮動小数点のモードを設定します。

すべての書式付き入出力操作の ROUND= 指示子を広域的に設定します。

-iorounding=compatible と指定する場合は、データ変換によって得られる値は、2 つのもっとも近い表示値のうち、より近い方の表示値になります。値が表示値のちょうど中間である場合は、0 から離れている方の表示値になります。

-iorounding=processor-defined を指定する場合は、丸めモードは、プロセッサのデフォルトのモードです。-iorounding が指定されない場合は、これがデフォルトになります。

-Kpic

(廃止) -xcode=pic13 と同義です。

-KPIC

(廃止) -xcode=pic32 と同義です。

-L*path*

ライブラリ検索ディレクトリパスのリストに *path* を追加します。

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 *path* を追加します。-L と *path* 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。詳細は、3-36 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、*path* 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 *path* 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「ライブラリ」を参照してください。) LD_LIBRARY_PATH と -L*path* の相対的な順序については、ld(1) を参照してください。

注 – -L*path* 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*path* 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% f95 -L./dir1 -L./dir2 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 lib*x*.a を追加します。

-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 libVZY をリンクします。

```
demo% f95 any.f -lVZY
```

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% f95 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「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。

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もっと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

ループの並列化結果を表示します。

-parallel、-autopar、-explicitpar オプションのいずれかで並列化されたループとそうでないループを示します。-loopinfo は次の並列化オプションを組み合わせる必要があります。

-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% f95 -o shalow -fast -parallel -loopinfo shalow.f
...
"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
```

–M*path*

MODULE ディレクトリかアーカイブ、ファイルのいずれかを指定します。

現在のコンパイルで参照されている Fortran 95 モジュールの検索で、指定されたパスが調べられます。現在のディレクトリの他に、このパスが調べられます。

path には、ディレクトリか .a アーカイブファイル (プリコンパイル済みモジュールファイルの場合)、.mod プリコンパイル済みモジュールファイルを指定できます。コンパイラは、ファイルの内容を検査してファイルの種類を判定します。

.a アーカイブファイルは、-M オプションフラグで、明示的に指定される必要があります。デフォルトでは、コンパイラはアーカイブファイルを検索しません。

USE 文にある MODULE 名と同じ名前の .mod ファイルのみが検索されます。たとえば USE ME 文があると、コンパイラは me.mod モジュールファイルのみ検索します。

検索時には、モジュールファイルの書き込み先のディレクトリが優先されます。これは、-moddir コンパイラオプションか MODDIR 環境変数で制御します。どちらも指定されていない場合は、現在のディレクトリがデフォルトの書き込み先ディレクトリになります。両方とも指定されている場合、-moddir フラグで指定されているパスが書き込み先ディレクトリになります。

このことは、-M フラグのみが指定されている場合は、-M フラグに指定されているすべてのオブジェクトの前に現在のディレクトリでモジュール検索が行われることを意味します。以前のリリースの動作をエミュレートするには、次を使用します。

```
-moddir=empty-dir -Mdir -M
```

ここで *empty-dir* は空のディレクトリへのパスです。

-M とパスの間に空白文字を入れないでください。

例 : -M/home/siri/PK15/Modules

Fortran 95 モジュールについての詳細は、4-23 ページの 4.9 節「モジュールファイル」を参照してください。

-moddir=*path*

コンパイルされた .mod MODULE ファイルの書き込み先を指定します。

コンパイラは、コンパイルした .mod MODULE 情報ファイルを *path* で指定されたディレクトリに書き込みます。ディレクトリパスは、MODDIR 環境変数で指定することもできます。両方が指定されている場合は、このオプションフラグが優先されます。

デフォルトでは、コンパイラは .mod ファイルの書き込み先として現在のディレクトリを使用います。

Fortran 95 モジュールについての詳細は、4-23 ページの 4.9 節「モジュールファイル」を参照してください。

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

Sun または Cray の並列化指令を選択します。

-explicitpar オプションを省略した場合のデフォルトは -mp=%none になります。

-explicitpar を指定した場合のデフォルトは -mp=sun 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=%none	全並列化指令を無視します。

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=cray
```

OpenMP の並列化用にコンパイルするには、-openmp フラグを使用します。3-44 ページの「-openmp[={parallel|noopt|none}]」を参照してください。

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。

Sun および Cray 並列化指令の概要は、付録 D を参照してください。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

スレッド環境で使用しても安全なライブラリへのリンクを要求します。

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参照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、-mt オプションを付けるとパフォーマンスが低下することがあります。

-native

(廃止) ホストシステムに合わせて最適化を行います。

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

-noautopar

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

(SPARC) コマンド行で先に指定された -depend を取り消します。

-noexplicitpar

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。

-nofstore

(x86) コマンド行の -fstore を取り消します。

コンパイラのデフォルトは -fstore です。-fast には、-nofstore が含まれています。

-nolib

システムライブラリとリンクしません。

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。最終的な実行には、システムおよび言語ライブラリが必要です。手動でライブラリとのリンクを行ってください。このオプションを使用すると、すべてを管理できます。

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。

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% f95 -fast -nolibmil …
```

-noreduction

コマンド行の -reduction を無効にします。

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。

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「ライブラリ」を参照してください。

-O[*n*]

最適化レベルを指定します

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*] の指定がない場合は、基本的な最適化のレベルは、局所的な共通部分式の除去、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。

最適化についての詳細は、『Fortran プログラミングガイド』の「パフォーマンスプロファイリング」と「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もっとも小さくなります (-xspace を参照)。

-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。-depend を自動的に追加します。

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、-O4 では、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。詳細は、-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。

-o と *name* 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。

-onetrip

DO ループを 1 回だけ実行します。

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[**=**{**parallel**|**noopt**|**none**}]

Fortran 95 の OpenMP バージョン 2.0 の指令で明示的な並列化を有効にします。

フラグには、次のキーワードサブオプションを使用できます。

parallel
- OpenMP プラグマの認識を有効にし、そのプラグマにもとづいてプログラムが並列化されます。
- -xopenmp=parallel の最小限の最適化レベルは -xO3 です。コンパイラは、必要に応じて最適化のレベルを低いレベルから -xO3 に上げ、警告を表示します。
- プリプロセッサのトークン _OPENMP を 200011 と定義します。
- -stackvar を自動的に呼び出します。

noopt
- OpenMP プラグマの認識を有効にし、そのプラグマにもとづいてプログラムが並列化されます。
- コンパイラは、最適化レベルが -xO3 より低くてもレベルを上げません。最適化レベルを、-xO2 -openmp=noopt のように明示的に -xO3 よりも低く設定すると、エラーが表示されます。最適化レベルを -openmp=noopt で指定しない場合、OpenMP プラグマは認識され、そのプラグマに基づいてプログラムが並列化されますが、最適化は実行されません。
- プリプロセッサのトークン _OPENMP を 200011 と定義します。
- -stackvar を自動的に呼び出します。

none
OpenMP プラグマの認識を無効にし、最適化レベルを変更しません (デフォルト値)

サブオプションキーワードを指定しない -openmp は、-openmp=parallel と同義です。このデフォルトは将来のリリースで変更される可能性があります。

dbx で OpenMP プログラムをデバッグするには、-g -openmp=noopt を指定してコンパイルすれば、並列化部分にブレークポイントを設定して変数の内容を表示することができます。

OpenMP 指令の概要については、『OpenMP API ユーザーズガイド』を参照してください。

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、PARALLEL 変数または OMP_NUM_THREADS 変数には、ターゲットプラットフォーム上の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。

コンパイルとリンクを分けて行う場合は、リンク時にも -openmp を指定する必要があります。これは、OpenMP 指令を含むライブラリをコンパイルする場合、特に重要です。

-PIC

(廃止、SPARC) 32 ビットアドレスを使用して、共有ライブラリ用に位置独立コードをコンパイルします。

-PIC は -xcode=pic32 と同義です。位置独立コードについての詳細は、3-71 ページの「-xcode=keyword」を参照してください。

-p

(廃止) 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。

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。

配列や文字変数が、静的な局所変数で初期化されていない場合、または共通ブロックにある場合、間にパディングを挿入します。パディングは、キャッシュを効率的に利用できる位置にデータが配置されるように挿入されます。いずれの場合も、配列または文字変数を等値化することはできません。

p を指定する場合は、%none か local または common のいずれかまたは両方を指定する必要があります。

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。
%none	パディングを追加挿入しません (コンパイラのデフォルトです)。

local と common の両方を指定する場合、順序はどちらが先でもかまいません。

-pad のデフォルトは以下のとおりです。

- デフォルトではコンパイラはパディングを挿入しません。
- 値なしの -pad は -pad=local,common と指定するのと同じです。

-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、3-51 ページの「-stackvar」の説明を参照してください。

プログラムは次の制限事項に従っている必要があります。

- 配列と文字列のどちらに対しても EQUIVALENCE 文を適用 (等値化) 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 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。

-pad が使用されたときに、共通ブロックのコンパイルの一貫性が維持されるようにする必要があります。異なるプログラムユニットの共通ブロックを -pad=common を付けてコンパイルしたとき、その一貫性が維持されない場合は、エラーになります。-Xlist を付けたコンパイルでは、同じ名前の共通ブロックの長さがプログラムユニットの間で異なる場合に、そのことが報告されます。

-parallel

-autopar, -explicitpar, -depend で並列化します。

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。詳細は、3-21 ページの「-explicitpar」を参照してください。

-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、-autopar オプションも指定してください。

デフォルトでは、Sun 形式の並列化指令が有効です。Cray 形式の並列化指令を選択する場合は、-mp=cray を使用します (注 : OpenMP 並列化の場合は、-parallel ではなく、-openmp を使用します)。

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。3-40 ページの「-mt」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、PARALLEL 変数または OMP_NUM_THREADS 変数には、ターゲットプラットフォーム上の利用可能なプロセッサ数を設定します。

-parallel 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。-parallel 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 -parallel 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』の「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。

-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、ソースファイルと .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

(廃止、SPARC) 共有ライブラリ用に位置独立コードをコンパイルします。

-pic は -xcode=pic13 と同義です。位置独立コードについての詳細は、3-71 ページの「-xcode=keyword」を参照してください。

-Qoption *pr ls*

サブオプションリスト *ls* をコンパイル段階 *pr* に渡します。

Qoption、*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に使われます。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。

-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。

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 /opt/SUNWspro/lib は、常に最後に検索されます。

詳細は、『Fortran プログラミングガイド』の「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。

-r8const

単精度の定数を REAL*8 の定数に変換します。

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、3-102 ページの「-xtypemap=spec」を使用してください。

このオプションフラグを使用する際には注意が必要です。REAL*4 引数を期待するサブルーチンまたは関数が REAL*4 定数で呼び出される場合に、REAL*8 の指令を受け取ることになるため、インタフェースの問題が生じる可能性があります。また、入出力リストに REAL*4 定数がある書式なし write によって書き込まれた、書式なしデータファイルの読み取りプログラムで問題を生じる可能性もあります。

-reduction

ループ中にある縮約演算を識別します。

自動並列化中にループを解析し、縮約演算を調べます。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「並列化」を参照してください。

このオプションは、並列化オプション -autopar または -parallel とともに使用する場合にのみ使用できます。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% f95 -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。

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。

実行可能ファイルを縮小しますが、リバースエンジニアを困難にします。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

(廃止) ソースコードブラウザ用のテーブル情報を生成します。

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、.F95、または .F03 のファイル) 上で使用することはできません。また、-F オプションとともに使用することもできません。

-sbfast

(廃止) 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。

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。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 – -sbfast 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、.F95、または .F03 のファイル) 上で使用することはできません。また、-F オプションとともに使用することもできません。

-silent

(廃止) コンパイラメッセージの出力を抑制します。

通常、f95 コンパイラは、コンパイル中に、エラー診断以外のメッセージを発行しません。このオプションフラグは、従来の f77 コンパイラとの互換性を保つために準備されています。-f77 互換性フラグとともに使用しない場合は、このオプションフラグは必要ありません。

-stackvar

可能な場合はいつでも局所変数をメモリースタックに割り当てます。

このオプションは、再帰的で再入力可能なコードの記述を簡単にし、ループを並列化する際の最適化により自由度を与えることができます。

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。

局所変数は、仮引数ではない変数、COMMON 変数、外部スコープから継承された変数、または USE 文によってアクセス可能になったモジュール変数です。

-stackvar を有効にすると、局所変数は、属性 SAVE または STATIC を持たない限り、スタックに割り当てられます。明示的に初期化された変数は、SAVE 属性を使用して暗黙的に宣言されます。明示的に初期化されず、いくつかのコンポーネントは初期化されている構造変数は、デフォルトでは、SAVE を使用して暗黙的に宣言されません。また、SAVE または STATIC 属性を持つ変数と同等な変数は、暗黙的に SAVE または STATIC です。

静的に割り当てられた変数は、プログラムによって明示的に値を指定されない限り、暗黙的に 0 に初期化されます。スタックに割り当てられた変数は、構造変数のコンポーネントがデフォルトで初期化できる場合を除き、暗黙的に初期化されません。

-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。

プログラムを実行する初期スレッドには、メインスタックがあり、マルチスレッド化されたプログラムの各ヘルパースレッドには、それぞれスレッドスタックがあります。

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 4 M バイト (SPARC V9 プラットフォームの場合は 8M 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        523256K バイト
stacksize       8192K バイト       <---
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「並列化」を参照してください。limit コマンドについての詳細は、csh(1) を参照してください。

-xcheck=stkovf フラグを指定してコンパイルすると、スタックオーバーフロー状態に対する実行時の検査が有効になります。3-68 ページの「-xcheck=keyword」を参照してください。

-stop_status[={yes|no}]

STOP 文により整数のステータス値を返します。

デフォルトは -stop_status=no 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。

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。

各コンパイル段階で費や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。

大文字を小文字と同等には取り扱いません。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ることがあります。詳細は、『Fortran プログラミングガイド』の Fortran 95 への移植に関する章を参照してください。

-U*name*

プリプロセッサのマクロ *name* の定義を取り消します。

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 -D*name* で作成されたプリプロセッサのマクロ *name* 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 -D*name* 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 -U*name* フラグを配置できます。-U とマクロ *name* の間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。

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 *type* 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個所で、使用可能にします。

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「パフォーマンスと最適化」を参照してください。詳細は、2-11 ページの 2.3.1.3 節「UNROLL 指令」を参照してください。

-use=*list*

暗黙的な USE モジュールを指定します。

list は、モジュール名またはモジュールファイル名のコンマ区切りのリストです。

-use=*module_name* を使用してコンパイルすると、USE *module_name* 文をコンパイルされる各サブプログラムまたはモジュールに追加することになります。
-use=*module_file_name* を使用してコンパイルすると、指定されたファイルに含まれる各モジュールの USE *module_name* を追加することになります。

Fortran 95 モジュールについての詳細は、4-23 ページの 4.9 節「モジュールファイル」を参照してください。

-V

各コンパイラパスの名前とバージョンを表示します。

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ときに役立ちます。

-v

各コンパイラパスの詳細情報を表示します。

-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=*keywords*

VAX VMS Fortran 拡張機能を有効にすることを指定します。

keywords 指定子は、次のサブオプションのいずれか、またはこれらのサブオプションをいくつか組み合わせて、コンマで区切ったリストとして指定します。

`blank_zero`	書式付き入力の空白を内部ファイルでゼロと解釈します。
`debug`	'D' 文字で始まる行を、VMS Fortran と同じように、注釈行ではなく通常の Fortran 文として解釈します。
`rsize`	書式なしレコードサイズを、バイト単位ではなくワード単位で解釈します。
`struct_align`	VAX 構造体の成分を、メモリー内に VMS Fortran と同じようにレイアウトします。パディングは挿入しません。注： このサブオプションを指定すると、データの不正な整列が発生する場合があります。
`%all`	上記すべての VAX VMS 機能を有効にします。
`%none`	上記すべての VAX VMS 機能を無効にします。

サブオプションは個々に選択することもオフにすることもできます。個々にオフにするには、サブオプションの前に `no%` を付けます。

例：

```
-vax=debug,rsize,no%blank_zero
```

-vpara

並列化に関する詳細な警告メッセージを表示します。

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：詳細な並列化に関する警告

```
demo% f95 -explicitpar -vpara any.f
any.f:
 MAIN any:
"any.f", line 11: 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[*n*]

警告メッセージを表示または抑制します。

ほとんどの警告メッセージを表示または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。

n は、0、1、2、3、または 4 です。

-w0 は、エラーメッセージのみを表示します。これは、-w と同等です。-w1 はエラーと警告を表示します。これは、-w を省略したときのデフォルトです。
-w2 は、エラー、警告、および注意を表示します。
-w3 は、エラー、警告、注意、および注を表示します。
-w4 は、エラー、警告、注意、注、およびコメントを表示します。

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% f95 -w -parallel any.f
f95: 警告 並列化コードをサポートするために、最適化レベルが 0 から 3 に変更されました
demo%
```

-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。

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、相互参照表など、このオプションは行番号付きのソースコードリストも生成します。
-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中断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 3-9　-Xlist サブオプション

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出力しません。
-Xlisth	エラー検出時にコンパイルを終了します。
-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。
-XlistMP	OpenMP 指令を検査します (**SPARC**)。
-Xlisto *name*	レポートファイルを *file*.lst ではなく、*name* に出力します。
-Xlists	相互参照表から参照されない名前を抑制します。
-Xlistv*n*	検査レベルを *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表示します。

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。

-x386

(x86) -xtarget=386 と同義です。

-x486

(x86) -xtarget=486 と同義です。

-xa

-a と同義です。

-xalias[=*keywords*]

コンパイラが仮定する別名付けの程度を指定します。

標準規格以外のプログラム手法によっては、コンパイラの最適化方法と干渉する状況になります。オーバーインデックスおよびポインタの使用、および大域変数または一意ではない変数を副プログラムの引数として渡すことは、不明確な状況を引き起こし、予定どおりにコードが実行されない場合があります。

-xalias フラグを使用すると、別名付けが Fortran の標準規則からどのくらい離れているかをコンパイラに知らせることができます。

フラグには、キーワードのリストがある場合も、ない場合もあります。キーワードのリストはコンマで区切られ、各キーワードはプログラムにおける別名付けの状況を表わしています。

キーワードに接頭辞 no% が付いている場合は、その別名付けが存在しないことを表わします。

別名付けのキーワードは、次のとおりです。

表 3-10　-xalias オプションキーワード

keyword	意味
dummy	副プログラムの仮 (形式的な) パラメータは、お互いに別名を付けることができ、大域変数にも別名を付けます。
no%dummy	(デフォルト)。Fortran 標準規格に従って仮パラメータを使用します。パラメータはお互いに別名を付けたり、大域変数に別名を付けることはしません。
craypointer	(デフォルト)。Cray ポインタは、そのアドレスを LOC() 関数が受け取るようなすべての大域変数または局所変数を指すことができます。また、2 つの Cray ポインタが同一のデータを指す可能性もあります。このような前提に基づいて、いくつかの最適化が抑制されるため安全です。
no%craypointer	Cray ポインタは malloc() によって得られるアドレスなど、一意のメモリーアドレスのみ指します。また、2 つの Cray ポインタが同一のデータを指すことはありません。この前提によって、コンパイラは Cray ポインタ参照を最適化することができます。
actual	コンパイラは、実際の副プログラムの引数を、大域変数のように扱います。引数を副プログラムに渡すことは、Cray ポインタを通して別名を付けることになります。
no%actual	(デフォルト) 引数を渡しても、別名は付けられません。

表 3-10　-xalias オプションキーワード (続き)

keyword	意味
overindex	• COMMON ブロックの要素のリファレンスは、COMMON ブロックまたは同等のグループの要素を参照します。 • COMMON ブロックまたは同等のグループの要素を実際の引数として副プログラムに渡すと、呼び出される副プログラムに、COMMON ブロックまたは同等のグループの要素へのアクセスを提供することになります。 • 連続構造型は、COMMON ブロックのように扱われ、このような変数の要素は、その変数の別の要素に別名を付けます。 • 各配列境界には違反しますが、前述したものを除いては、参照される配列の要素は配列内に残ります。配列構文、WHERE、および FORALL 文は、オーバーインデックスを考慮していません。これらの構造構文でオーバーインデックスが起こる場合は、DO ループとして書き直す必要があります。
no%overindex	(デフォルト) 配列境界は違反しません。配列の参照は、他の変数を参照しません。
ftnpointer	外部関数の呼び出しによって、Fortran ポインタは、どのような型、種類、またはランクのターゲット変数でもポイントする場合があります。
no%ftnpointer	(デフォルト) Fortran ポインタは標準規則に準拠します。

リストなしで -xalias を指定すると、Fortran の別名付け規則に違反しないほとんどのプログラムで最高のパフォーマンスを得ることができます。これは、次に対応します。

```
no%dummy,no%craypointer,no%actual,no%overindex,no%ftnpointer
```

有効にするには、-xalias は、最適化レベル -x03 以上でコンパイルするときに使用する必要があります。

-xalias フラグが指定されていない場合は、コンパイラのデフォルトでは、次の Cray ポインタを除き、Fortran 95 の標準に準拠していると仮定されます。

```
no%dummy,craypointer,no%actual,no%overindex,no%ftnpointer
```

別名付けの状況の例、および -xalias を使用した指定方法については、『Fortran プログラミングガイド』の移植に関する章を参照してください。

-xarch=*isa*

命令セットアーキテクチャ (ISA) を指定します。

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。

表 3-11 -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
x86	generic、native、386、pentium_pro、sse、sse2、amd64

-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。たとえば次のとおりです。

```
% 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 3-12 は、SPARC プラットフォーム上でのもっとも一般的な -xarch オプションをまとめています。

表 3-12 SPARC プラットフォーム上で一般的な -xarch オプション

-xarch=	パフォーマンス
generic	• サポートされているすべての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。

表 3-12　SPARC プラットフォーム上で一般的な -xarch オプション

-xarch=	パフォーマンス
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。 • その他のプラットフォーム上では実行できません。
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。 • その他のプラットフォーム上では実行できません。
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。 • その他のプラットフォーム上では実行できません。

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、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、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。

SPARC プラットフォームで -xarch が指定されなかった場合のデフォルトは v8plus、x86 プラットフォームでは 386 です。

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 3-13 SPARC プラットフォーム上の -xarch の値

-xarch=	意味 (SPARC)
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。新リリースが発表されるたびに「最善」の命令セットの定義が変わりますが、現時点では v8plus と解釈されています。
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「最善」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 : SPARCstation 10

表 3-13　SPARC プラットフォーム上の -xarch の値 (続き)

-xarch=	意味 (SPARC)
v8plus	**SPARC-V9 ISA の V8plus バージョン用のコンパイル** 定義上、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
v8plusa	**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 3-13　SPARC プラットフォーム上の -xarch の値 (続き)

-xarch=	意味 (SPARC)
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 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 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 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 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 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 Solaris 環境でコンパイルする場合だけです。

表 3-14 に、x86 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。x86 で -xarch が指定されなかった場合のデフォルトは generic です。

表 3-14　x86 プラットフォーム上の -xarch の値

-xarch=	意味 (x86)
generic	多く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をします。これはデフォルトで、-xarch=386 と同義です。
generic64	多くのシステムで良好な 64 ビットパフォーマンスを得るためのコンパイルをします。現在は、amd64 と解釈されます。
native	コンパイルを行うマシンの x86 アーキテクチャで良好なパフォーマンスを得られるようにコンパイルを実行します。たいていの x86 プロセッサで良好なパフォーマンスを得るのに最良の命令セットを使用します。「最良な命令セット」の内容は、新しいリリースごとに調整される可能性があります。
native64	コンパイルを行うマシンの 64 ビット x86 アーキテクチャで良好なパフォーマンスを得られるようにコンパイルを実行します。
386	命令セットを Intel 386/486 アーキテクチャに限定します。
pentium_pro	命令セットを Pentium Pro アーキテクチャに制限します。
sse	pentium_pro に SSE 命令セットを追加します (下記を参照)。
sse2	pentium_pro に SSE2 命令セットを追加します (下記を参照)。
amd64	AMD64 64 ビット x86 命令セット用のコンパイルをします。

x86 プラットフォームに関する注意事項

Solaris x86 SSE/SSE2 Pentium 4 互換プラットフォームで動作するよう -xarch={sse | sse2} を使ってコンパイルしたプログラムは、SSE/SSE2 対応のプラットフォームでのみ実行する必要があります。SSE/SSE2 に対応していないプラットフォームでそうしたプログラムを実行すると、セグメント例外が発生したり、明示的な警告メッセージなしに不正な結果が発生することがあります。SSE/SSE2 でコンパイルされたバイナリが SSE/SSE2 に対応していないプラットフォームで実行されることがないようにするための OS およびコンパイラに対するパッチが、後日提供される可能性があります。

Pentium 4 互換プラットフォームの場合、Solaris 9 9/04 以降のリリースは SSE/SSE2 に対応しています。これより前のバージョンの Solaris は SSE/SSE2 に対応していません。このことは、.il インラインアセンブリ言語関数を使用しているプログラムや、SSE/SSE2 命令を利用している __asm() アセンブラコードにも当てはまります。

コンパイルとリンクを別々に行う場合は、必ずコンパイラを使ってリンクし、-xarch={sse | sse2} で適切な起動ルーチンがリンクされるようにしてください。

x86 の浮動小数点レジスタのサイズが 80 ビットであるため、x86 上と SPARC 上での算術演算結果が異なることがあります。こうした違いが出ないようにするには、-fstore オプションを使用してください。あるいは、ハードウェアが SSE2 をサポートしている場合は、-xarch=sse2 を付けてコンパイルしてください。

-xassume_control[=*keywords*]

ASSUME プラグマを制御するパラメータを設定します。

このフラグを使用して、コンパイラがソースコード内の ASSUME プラグマを処理する方法を制御します。

プログラマは ASSUME プラグマを使用することによって、コンパイラがより良い最適化を得るために使用できる特殊な情報を表明することができます。これらの表明は、確率を指定することができます。確率が 0 または 1 の場合は確実 (certain) とされ、それ以外の場合は不確実 (non-certain) とみなされます。

また、可能性または確実性を指定して、次に DO ループのトリップカウント、または分岐が起こることを表明することもできます。

f95 コンパイラが認識する ASSUME プラグマの説明については、2-14 ページの 2.3.1.9 節「ASSUME 指令」を参照してください。

-xassume_control オプションの *keywords* には、1 つのサブオプションキーワードまたはコンマで区切られたキーワードのリストを指定します。認識されるキーワードサブオプションは、次のとおりです。

optimize	ASSUME プラグマで指定される表明は、プログラムの最適化に影響を与えます。
check	コンパイラは確実とマークされたすべての表明の正確さを検査するコードを生成し、表明が違反している場合は実行時メッセージを出力します。プログラムは fatal が指定されていなければ、処理を続行します。
fatal	check とともに使用すると、確実とマークされている表明が違反を起こすとプログラムは終了します。
retrospective[:*d*]	*d* パラメータはオプションの許容値で、1 未満の正の実数定数を指定する必要があります。デフォルトは「.1」です。retrospective を指定すると、すべての表明について真と偽をカウントするコードをコンパイルします。値が許容値 *d* を超えると、プログラム終了時に一覧が出力されます。
%none	すべての ASSUME プラグマが無視されます。

コンパイラのデフォルトは次のとおりです。

```
-xassume_control=optimize
```

これは、コンパイラが ASSUME プラグマを認識し、最適化に影響を与えますが、検査は行わないという意味です。

パラメータを指定しない場合、-xassume_control は次と同等です。

```
-xassume_control=check,fatal
```

この場合、コンパイラは certain とマークされたすべての ASSUME プラグマを受け付け、検査しますが、最適化には影響を与えません。表明が無効の場合、プログラムは終了します。

-xautopar

-autopar と同義です。

-xcache=*c*

オプティマイザ用のキャッシュ特性を定義します。

c には以下のいずれかを指定します。

- generic
- native
- *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行サイズ (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。特別な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。

表 3-15　-xcache の値

値	意味
generic	どの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します。これはデフォルト値です。
native	ホストプラットフォームで良好なパフォーマンスを得るためのキャッシュ特性を定義します。
s1/*l1*/*a1*	レベル 1 のキャッシュ特性を定義します。
s1/*l1*/*a1*:*s2*/*l2*/*a2*	レベル 1 と 2 のキャッシュ特性を定義します。
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します。

例 : -xcache=16/32/4:1024/32/1 は次の内容を指定します。

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

(SPARC) -cg89 と同義です。

-xcg92

(SPARC) -cg92 と同義です。

-xcheck=*keyword*

実行時の特別な検査を生成します。

keyword には以下のいずれかを指定します。

keyword	機能
stkovf	副プログラムのエントリのスタックオーバーフローを実行時に検査します。スタックオーバーフローが検出されると、SIGSEGV セグメントフォルトが発生します (*SPARC* のみ)。
no%stkovf	スタックオーバーフローの実行時の検査を無効にします (*SPARC* のみ)
init_local	局所変数の特定の初期化を実行します。 コンパイラは局所変数を、代入される前にプログラムが使用すると算術例外を起こす可能性がある値に初期化します。ALLOCATE 文によって割り当てられたメモリーも同じように初期化されます。 モジュール変数、SAVE 変数、COMMON ブロックの変数は初期化されません。
no%init_local	局所変数の初期化を無効にします。これはデフォルト値です。
%all	実行時のすべての検査機能を有効にします。
%none	実行時のすべての検査機能を無効にします。

スタックオーバーフローは、特に、スタックに大きな配列が割り当てられるマルチスレッドアプリケーションで、近傍のスレッドスタックのデータを警告なしに破壊する可能性があります。スタックオーバーフローの可能性がある場合は、-xcheck=stkovf を使用してすべてのルーチンをコンパイルします。ただし、このフラグを使用してコンパイルしても、このフラグを使用せずにコンパイルしたルーチンでスタックオーバーフローが起こる可能性があるので、すべてのスタックオーバーフローの状況が検出されるわけではありません。

-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。

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 -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。

-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

次の表に、-xchip の有効なプロセッサ名の値をまとめてあります。

表 3-16　-xchip でよく使われる SPARC プロセッサ名

-xchip=	最適化の対象
generic	ほとんどの SPARC プロセッサ (デフォルト値)
native	このホストプラットフォーム
ultra	UltraSPARC プロセッサ
ultra2	UltraSPARC II プロセッサ
ultra2e	UltraSPARC IIe プロセッサ
urtra2i	UltraSPARC IIi プロセッサ
ultra3	UltraSPARC III プロセッサ
ultra3cu	UltraSPARC IIIcu プロセッサ
ultra4	UltraSPARC IV プロセッサ

次の表は古く、あまり利用されていない -xchip プロセッサ名です。参照のためだけに記載しています。

表 3-17　-xchip であまり使われることのない SPARC プロセッサ名

-xchip=	最適化の対象
old	SuperSPARC より前のプロセッサ
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

x86 プラットフォーム：-xchip 値は、386、486、pentium、pentium_pro、pentium3、pentium4、opteron、generic、および native のどれかです。

-xcode=*keyword*

(SPARC) 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。

keyword の値は以下のとおりです。

keyword	機能
`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。

`-xcode=`*keyword* を明示的に指定しなかった場合のデフォルトは次のとおりです。

`-xcode=abs32` SPARC V8 および V7 プラットフォーム。

`-xcode=abs44` UltraSPARC V9 (`-xarch=v9`)。

位置独立コード

実行時のパフォーマンスを向上されるために動的共有ライブラリを使用するときには、`-xcode=pic13` または `-xcode=pic32` を使用します。

動的実行可能ファイルのコードは、通常、メモリーの固定アドレスに結び付けられ、位置独立コードは、プロセスのどのようなアドレス空間でもロードすることができます。

位置独立コードを使用する場合は、大域オフセットテーブルを使用した直接的なリファレンスとして、再配置可能なリファレンスが作成されます。共有オブジェクトの頻繁にアクセスされる項目は、`-xcode=pic13` または `-xcode=pic32` を使用してコンパイルすると、位置独立コード以外のコードによって多数回行われる再配置が必要なくなるという利点があります。

大域オフセットテーブルのサイズは、8K バイトに制限されます。

xcode={pic13|pic32} には、次のようなパフォーマンス上の影響があります。

- -xcode=pic13 または -xcode=pic32 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参照を行います。pic32 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参照を行うごとに、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、-xcode=pic13 と -xcode=pic32 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。-xcode=pic13 または -xcode=pic32 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 -xcode=pic13 または -xcode=pic32 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もっとも簡単です。

```
nm file.o | grep _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。

-xcode=pic13 または -xcode=pic32 のどちらを使用すべきか決定するときは、elfdump -c (詳細は elfdump(1) のマニュアルページを参照) を使用することによって、セクションヘッダー (sh_name: .got) を探して、大域オフセットテーブル (GOT) のサイズを調べてください。sh_size 値が GOT のサイズです。GOT のサイズが 8,192 バイトに満たない場合は -xcode=pic13、そうでない場合は -xcode=pic32 を指定します。

一般に、-xcode の使用方法に決定に際して、次の指針に従ってください。

- 実行可能ファイルを構築する場合は、-xcode=pic13 と -xcode=pic32 のどちらも使わない。
- 実行可能ファイルへのリンク専用のアーカイブライブラリを構築する場合は、-xcode=pic13 と -xcode=pic32 のどちらも使わない。
- 共有ライブラリを構築する場合は、-xcode=pic13 からスタートし、GOT のサイズが 8,192 バイトを超えたら、-xcode=pic32 を使用する。
- 共有ライブラリへのリンク用のアーカイブライブラリを構築する場合は、-xcode=pic32 のみ使用する。

動的ライブラリを構築する場合は、-xcode=pic13 または pic32 (または -pic または -PIC) オプションを使用してコンパイルしてください。Solaris の『リンカーとライブラリ』を参照してください。

-xcommonchk[={yes | no}]

共通ブロック不一致の実行時検査を行います。

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。-xcommonchk=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。値を指定しない場合 -xcommonchk は -xcommonchk=yes と同じです。

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[={1|0}]

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。

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。3-34 ページの「-inline=[%auto][[,][no%]f1,...[no%]fn]」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。

また、.s のアセンブリソースファイルは、ファイル間解析には関係しません。また、-S を指定したコンパイルでは、-xcrossfile フラグは無視されます。

-xdebugformat={stabs|dwarf}

Sun Studio コンパイラのデバッガ情報の形式は、"stabs" 形式から "dwarf" 形式に移行しつつあります。このリリースでのデフォルトの設定は -xdebugformat=stabs です。

デバッグ情報を読み取るソフトウェアを保守している場合は、今回からそのようなツールを stab 形式から dwarf 形式へ移行するためのオプションが加わりました。

このオプションは、ツールを移植する場合に新しい形式を使用する方法として使用してください。デバッガ情報を読み取るソフトウェアを保守していないか、ツールでこれらの内のいずれかの形式のデバッガ情報が必要でなければ、このオプションを使用する必要はありません。

-xdebugformat=stabs は、stab 標準形式を使用してデバッグ情報を生成します。

-xdebugformat=stabs は、dwarf 標準形式を使用してデバッグ情報を生成します。

-xdebugformat を指定しない場合は、コンパイラでは -xdebugformat=stabs が指定されます。引数なしでこのオプションを指定するとエラーになります。

このオプションは、-g オプションによって記録されるデータの形式に影響します。この情報の一部形式はまた、このオプションを使って制御されます。したがって、-g を使用しなくても、-xdebugformat は有効です。

dbx とパフォーマンスアナライザソフトウェアは、stab 形式と dwarf 形式を両方とも認識するので、このオプションを使用しても、ツールの機能にはまったく影響を与えません。

これは過渡的なインタフェースなので、今後のリリースでは、マイナーリリースであっても互換性なく変更されることがあります。stab と dwarf のどちらであっても、特定のフィールドまたは値の詳細は、今後とも変更される可能性があります。

-xdepend

-depend と同義です。

-xexplicitpar

-explicitpar と同義です。

-xF

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。

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、-Mmapfile 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。アナライザについての詳細は、『プログラムのパフォーマンス解析』を参照しください。

-xfilebyteorder=*options*

ビッグエンディアンとリトルエンディアン式プラットフォーム間のファイルの共有のサポート

このフラグは、書式なし入出力ファイルのバイト順序とバイト列を定義します。*options* には、次のフラグを組み合わせたものを指定する必要があります。少なくとも 1 つ、指定が存在する必要があります。

little*max_align***:***spec*

big*max_align***:***spec*

native:*spec*

max_align は、ターゲットプラットフォームの最大バイト列を宣言します。指定できる値は 1、2、4、8、16 です。境界整列は、C 言語の構造体との互換性を維持するため、プラットフォーム依存のバイト列を使用する Fortran VAX 構造体と Fortran 95 派生型に適用されます。

最大バイト列が *max_align* プラットフォームでは、**little** は「リトルエンディアン」ファイルを表します。たとえば `little4` は 32 ビット x86 ファイルを表し、`little16` は 64 ビット x86 ファイルを表します。

big は、最大バイト列が *max_align* の「ビッグエンディアン」ファイルを指定します。たとえば `big8` は SPARC V8 (32 ビット) ファイルを表し、`big16` は SPARC V9 (64 ビット) ファイルを表します。

native は、コンパイルしているプロセッサプラットフォームが使用しているのと同じバイト順序、バイト列の「ネイティブ」ファイルを表します。以下は、「ネイティブ」と見なされます。

プラットフォーム	「ネイティブ」に相当するフラグ
32 ビット SPARC V8	`big8`
64 ビット SPARC V9	`big16`
32 ビット x86	`little4`
64 ビット x86 (`amd64`)	`little16`

spec には、コンマ区切りのリストで以下を指定します。

%all

unit

filename

%all は、SCRATCH として開かれるか、-xfilebyteorder フラグで明示的に指定する以外のすべてのファイルとその他論理ユニットを表します。%all は、1 回だけ指定できます。

unit は、プログラムによって開かれた特定の Fortran ユニット番号を表します。

filename は、プログラムによって開かれた特定の Fortran ファイル名を表します。

例:

```
-xfilebyteorder=little4:1,2,afile.in,big8:9,bfile.out,12
-xfilebyteorder=little8:%all,big16:20
```

備考:

このオプションは、STATUS="SCRATCH" を指定して開かれたファイルには適用されません。それらのファイルに対する入出力操作には、つねにネイティブプロセッサのバイト順序とバイト列が使用されます。

コンパイラコマンド行に -xfilebyteorder が指定されていない場合の最初のデフォルトは、-xfilebyteorder=native:%all です。

このオプションには、ファイル名およびユニット番号をそれぞれ 1 回だけ宣言できます。

コマンド行に -xfilebyteorder を含める場合は、little か big、または native の少なくとも 1 つの指定と組み合わせる必要があります。

このフラグで明示的に宣言されなかったファイルは、ネイティブファイルと見なされます。たとえば -xfilebyteorder=little4:zfile.out を付けて zfile.out をコンパイルした場合、このファイルは、4 バイトの最大データ整列規則を持つリトルエンディアンの 32 ビット x86 ファイルと宣言され、他のすべてのファイルはネイティブファイルになります。

ファイルに指定されたバイト順序はネイティブプロセッサと同じであるが、バイト列が異なる場合は、バイトスワップが行われないにしても、適切なパディングが使用されます。たとえば -xarch=amd64 を付けた、64 ビット x86 プラットフォーム向けのコンパイルで、-xfilebyteorder=little4:*filename* が指定された場合などがそうです。

ビッグエンディアンとリトルエンディアン式プラットフォーム間で共有されるデータレコード内で宣言する型は、同じサイズである必要があります。たとえば、-xtypemap=integer:64,real:64,double:128 を付けてコンパイルした SPARC 実行可能ファイルの生成するファイルを、-xtypemap=integer:64,real:64,double:64 を付けてコンパイルした x86 実行可能ファイルが読み取ることはできません。これは、両者のデフォルトの倍精度データ型のサイズが異なるためです。

REAL*16 データを含む書式なしのファイルは、REAL*16 をサポートしていない x86 プラットフォームでは使用できないことに注意してください。

ネイティブ以外のファイルとして指定されたファイルに対して、UNION/MAP データオブジェクト全体を使った入出力操作を行うと、実行時入出力エラーになります。ネイティブ以外のファイルに対しては、MAP の個別メンバーを使った入出力操作のみ行うことができます。UNION/MAP を含む VAX レコード全体を使った入出力操作は行えません。

-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。

-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 の標準に準拠しています (コンパイラが生成する実際のコールリストには、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% f95 -o has0 hasc.f
demo% has0
 string length =   4    <-- 7 でなくてはなりません
demo% f95 -o has1 -xhasc=no hasc.f
demo% has1
 string length =   7  <-- s の長さを修正します
```

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、従来の FORTRAN 77 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={readme|flags}

要約したヘルプ情報を表示します。

-xhelp=readme　このコンパイラのリリースのオンライン README ファイルを表示します。

-xhelp=flags　コンパイラのオプションフラグを一覧表示します。-help と同義です。

-xia[={widestneed|strict}]

(SPARC) 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。

指定しない場合のデフォルトは、-xia=widestneed です。

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。詳細は、3-80 ページの「-xinterval[={widestneed | strict | no}]」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 ild を使用可能/不可能にします。

-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。

デフォルトは -xildoff です。以前のリリースのコンパイラでは、-g を指定したオブジェクトファイルのみのコンパイルでのデフォルトは -xildon でした。-g を指定したリンク専用のコンパイルで強制的に ild を使用するには、明示的に -xildon オプションを含めます。

3-32 ページの「-g」および『C ユーザーズガイド』の ild に関する節を参照してください。

-xinline=*list*

-inline と同義です。

-xinterval[={*widestneed*|*strict*|*no*}]

(SPARC) 区間演算処理を有効化します。

オプションの値には、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。詳細は、3-79 ページの「-xia[={widestneed|strict}]」を参照してください。

-xipo[={**0**|**1**|**2**}]

(SPARC) 内部手続きの最適化を実行します。

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。-xipo=2 は、キャッシュのパフォーマンスを向上させるために、手続き間の別名付けの解析、および記憶域割り当てとレイアウトの最適化を追加します。デフォルトは -xipo=0 で、-xipo が値を伴わずに指定された場合は、-xipo=1 が使用されます。

-xipo=2 を付けてコンパイルすると、-xipo=2 を付けずにコンパイルされた関数やサブルーチン (たとえばライブラリ) から -xipo=2 を付けてコンパイルされた関数やサブルーチンへの呼び出しがあってはいけません。

一例として、malloc() の置き換えとして、-xipo=2 を付けてコンパイル化した独自のバージョンの malloc() を使用する場合は、作成したコードとリンクするライブラリ内の malloc() を参照するすべての関数も -xipo=2 を付けてコンパイルする

必要があります。しかしながら、システムライブラリに対してこういったことを行うのは不可能な場合があるため、この独自のバージョンの malloc のコンパイルに -xipo=2 を使うべきではありません。

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 のルーチンの間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 -xO4 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。
- .s のアセンブリ言語ソースファイルは、内部手続き解析には関係しません。
- -S を付けたコンパイルでは、-xipo フラグは無視されます。

コンパイルに -xipo を使用すべきでないケース

内部手続き解析では、コンパイラは、リンクステップでオブジェクトファイル群を操作しながら、プログラム全体の解析と最適化を試みます。このとき、コンパイラは、このオブジェクトファイル群に定義されているすべての foo() 関数 (またはサブルーチン) に関して次の 2 つのことを仮定します。

(1) 実行時、このオブジェクトファイル群の外部で定義されている別のルーチンによって foo() が明示的に呼び出されない。

(2) オブジェクトファイル群内のルーチンから呼び出される foo() が、そのオブジェクトファイル群の外部に定義されている別のバージョンの foo() によって置き換えられることがない。

アプリケーションに対して仮定 (1) が当てはまらない場合は、コンパイルで -xipo=2 を使わないでください。
仮定 (2) が当てはまらない場合は、コンパイルで -xipo=1 および -xipo=2 を使わないでください。

1 例として、独自のソースバージョンの malloc() で関数 malloc() を置き換えるケースを考えてみましょう。-xipo=2 を使ってコンパイルする場合、その独自のコードとリンクされる malloc() を参照する、あらゆるライブラリのあらゆる関数のコンパイルで -xipo=2 を使用する必要があるとともに、リンクステップでそれらのオブジェクトファイルが必要になります。しかしながら、システムライブラリでは、このことが不可能なことがあり、このため、独自のバージョンの malloc のコンパイルに -xipo=2 を使うべきではありません。

もう 1 つの例として、別々のソースファイルにある foo() および bar() という 2 つの外部呼び出しを含む共有ライブラリを構築するケースを考えてみましょう。bar() はその本体内で foo() を呼び出します。関数呼び出し foo() が実行時に異なるバージョンの foo() に置き換えられる可能性がある場合、foo() および bar() のソースファイルのコンパイルで -xipo=1 や -xipo=2 を使ってはいけません。foo() が bar() 内にインライン化され、不正な結果になる可能性があります。

-xipo_archive[={none|readonly|writeback}]

(SPARC) ファイル相互の最適化でアーカイブ (.a) ライブラリを取り込むことを可能にします。

値には、次のいずれかを指定します。

writeback	実行可能ファイルを生成する前に、.a アーカイブライブラリに存在するオブジェクトファイル (-xipo でコンパイルしたファイル) を使ってリンカーに渡すオブジェクトファイルを最適化します。コンパイル中に最適化されたライブラリに含まれるオブジェクトファイルはすべて、その最適化されたバージョンに置き換えられます。
readonly	実行可能ファイルを生成する前に、.a アーカイブライブラリに存在するオブジェクトファイル (-xipo でコンパイルしたファイル) を使ってリンカーに渡すオブジェクトファイルを最適化します。
none	アーカイブファイルを処理しません。

-xipo の値が指定されていない場合、コンパイラは -xipo_archive=none に設定します。

-xjobs=*n*

複数のプロセッサを使用してコンパイルします。

コンパイラが処理を行うために生成するプロセスの数を設定するには、-xjobs オプションを指定します。このオプションを使用すると、マルチ CPU マシン上での構築時間を短縮できます。このリリースの f95 コンパイラでは、-xjobs とともに使用できるのは -xipo オプションだけです。-xjobs=*n* を指定すると、内部手続きオプティマイザは、異なるファイルをコンパイルするために呼び出すことができるコードジェネレータの最大インスタンス数として *n* を使用します。

一般に、*n* に指定する確実な値は、使用できるプロセッサ数に 1.5 を掛けた数です。生成されたジョブ間のコンテキスト切り替えにより生じるオーバーヘッドのため、使用できるプロセッサ数の何倍もの値を指定すると、パフォーマンスが低下することがあります。また、あまり大きな数を使用すると、スワップ領域などシステムリソースの限界を超える場合があります。

-xjobs には必ず値を指定する必要があります。値を指定しないと、エラーという診断が表示され、コンパイルは中止します。

コマンド行で複数の -xjobs の指定がある場合、一番右にあるインスタンスの指定によって上書きされます。

次の例に示すコマンドは 2 つのプロセッサを持つシステム上で、-xjobs オプションを指定しないで実行された同じコマンドよりも高速にコンパイルを実行します。

```
example% f95 -xipo -xO4 -xjobs=3 t1.f t2.f t3.f
```

-xknown_lib=*library_list*

既知のライブラリの呼び出しを認識します。

指定された場合は、既知のライブラリの参照をイントリンシクスとして扱い、ユーザー定義のバージョンを無視します。これによって、コンパイラは、ライブラリに関する情報に基づき、ライブラリルーチンの呼び出しを最適化します。

library_list には、現時点では blas、blas1、blas2、blas3、および intrinsics に対する、コンマで区切られたキーワードのリストを指定します。コンパイラは次の BLAS1、BLAS2、および BLAS3 ライブラリルーチンを認識し、サンのパフォーマンスライブラリの実装に適するように自由に最適化します。コンパイラは、これらのライブラリルーチンのユーザー定義バージョン無視し、サンのパフォーマンスライブラリ中の BLAS ルーチンとリンクします。

-xknown_lib=	機能
blas1	コンパイラは次の BLAS1 ライブラリルーチンを認識します。 caxpy ccopy cdotc cdotu crotg cscal csrot csscal cswap dasum daxpy dcopy ddot drot drotg drotm drotmg dscal dsdot dswap dnrm2 dzasum dznrm2 icamax idamax isamax izamax sasum saxpy scasum scnrm2 scopy sdot sdsdot snrm2 srot srotg srotm srotmg sscal sswap zaxpy zcopy zdotc zdotu zdrot zdscal zrotg zscal zswap
blas2	コンパイラは次の BLAS2 ライブラリルーチンを認識します。 cgemv cgerc cgeru ctrmv ctrsv dgemv dger dsymv dsyr dsyr2 dtrmv dtrsv sgemv sger ssymv ssyr ssyr2 strmv strsv zgemv zgerc zgeru ztrmv ztrsv
blas3	コンパイラは次の BLAS2 ライブラリルーチンを認識します。 cgemm csymm csyr2k csyrk ctrmm ctrsm dgemm dsymm dsyr2k dsyrk dtrmm dtrsm sgemm ssymm ssyr2k ssyrk strmm strsm zgemm zsymm zsyr2k zsyrk ztrmm ztrsm
blas	すべての BLAS ルーチンを選択します。 -xknown_lib=blas1,blas2,blas3 と同義です。
intrinsics	コンパイラは、Fortran 95 の明示的な EXTERNAL 宣言を無視します。このため、ユーザー定義組み込み関数ルーチンも無視されます。

-xlang=f77

(SPARC) f77 でコンパイルされた実行ライブラリとのリンクの準備をします。

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。このフラグを使用してコンパイルすることによって、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。

-xlang を付けたコンパイルでは、次のことに注意してください。

- コンパイルで -xnolib および -xlang の両方を使わないでください。
- Fortran オブジェクトファイルと C++ が混在する場合は、C++ コンパイラを使用し、CC コマンド行で -xlang=f95 を指定してください。
- C++ オブジェクトと並列オプションを付けてコンパイルした Fortran オブジェクトファイルが混在する場合は、リンク用の CC コマンド行で -mt も指定する必要があります。

-xlibmil

-libmil と同義です。

-xlibmopt

最適化された数学ルーチンを使用します。

速度の最適化のために選択された数学ルーチンを使用します。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

Sun のパフォーマンスライブラリとリンクします。

例 :

```
f95 -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、Sun 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)。

-xlicinfo

ライセンス情報を表示します。

インストールしたコンパイラのシリアル番号情報を取得するには、このオプションを使用します。

-xlinkopt[={1|2|0}]

(SPARC) 再配置可能なオブジェクトファイルのリンク時の最適化を実行します。

ポストオプティマイザは、リンク時にバイナリオブジェクトコードに対して高度なパフォーマンス最適化を多数実行します。オプションの値には、実行する最適化のレベルを 0、1、2 のいずれかで設定します。

0　ポストオプティマイザは無効になっています。(デフォルト値)

1　リンク時の命令キャッシュカラーリングと分岐の最適化を含む、制御フロー解析に基づき最適化を実行します。

2　リンク時のデッドコードの除去とアドレス演算の簡素化を含む、追加のデータフロー解析を実行します。

値なしで -xlinkopt フラグを指定すると、-xlinkopt=1 とみなされます。

このような最適化、はリンク時にオブジェクトのバイナリコードを解析することによって実行されます。オブジェクトファイルは書き換えられませんが、最適化された実行可能コードは元のオブジェクトコードとは異なる場合があります。

このオプションは、プログラム全体をコンパイルし、実行時プロファイルフィードバックと共に使用されるともっとも効果的です。

コンパイルとリンクを個別に実行する場合、-xlinkopt は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。

```
demo% f95 -c -xlinkopt a.f95 b.f95
demo% f95 -o myprog -xlinkopt=2 a.o b.o
```

レベルパラメータは、コンパイラのリンク時にだけ使用されます。上記の例では、オブジェクトバイナリが指定された 1 のレベルでコンパイルされていても、使用される最適化後のレベルは 2 です。

リンク時のポストオプティマイザは、インクリメンタルリンカー ild とともに使用することはできません。-xlinkopt フラグは、デフォルトリンカーを ld に設定します。-xildon フラグを使用してインクリメンタルリンカーを明示的に有効にしたときに -xlinkopt オプションも指定していると、-xlinkopt オプションは無効になります。

-xlinkopt オプションを有効にするには、プログラム内のルーチンの少なくとも一部は、こ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。-xlinkopt を指定しないでコンパイルされたオブジェクトバイナリについても、オプティマイザは限定的な最適化を実行できます。

-xlinkopt オプションは、コンパイラのコマンド行にある静的ライブラリのコードは最適化しますが、コマンド行にある共有 (動的) ライブラリのコードは最適化しません。共有ライブラリを構築 (-G でコンパイル) する場合は、-xlinkopt も使用できます。

リンク時のポストオプティマイザは、実行時のプロファイルフィードバックとともに使用するのがもっとも効果的です。プロファイリングによって、コードでもっともよく使用される部分ともっとも使用されない部分が明らかになるので、オプティマイザはそれにもとづき処理を集中するよう指示されます。これは、リンク時に実行されるコードの最適な配置が命令のキャッシュミスを低減できるような、大きなアプリケーションにとって特に重要です。このようなコンパイルの例を次に示します。

```
demo% f95 -o progt -xO5 -xprofile=collect:prog file.f95
demo% progt
demo% f95 -o prog -xO5 -xprofile=use:prog -xlinkopt file.95
```

プロファイルフィードバックの使用方法についての詳細は、-xprofile を参照してください。

このオプションを指定してコンパイルすると、リンク時間がわずかに増えます。オブジェクトファイルも大きくなりますが、実行可能ファイルのサイズは変わりません。-xlinkopt フラグと -g フラグを指定してコンパイルすると、デバッグ情報が取り込まれるので、実行可能ファイルのサイズが増えます。

-xloopinfo

-loopinfo と同義です。

-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。

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。2-13 ページの 2.3.1.5 節「OPT 指令」を参照してください。

このプラグマ指令が -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*>]

(SPARC) メモリー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動作を指定します。

境界整列がコンパイル時に決定できるメモリーアクセスの場合、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ーアクセスの場合、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ー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ーアクセスが行われた場合の実行時エラーの動作も指定します。

指定する値は、2 種類です。すなわち、数値の境界整列値 <*a*> と、英数字の動作フラグ <*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動作を表わす <*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コンパイル時のデフォルト値は、次のようになります。

- -xarch=generic,v7,v8,v8a,v8plus,v8plusa の場合は、8i
- 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。

-xmemalign そのものは、特定のデータ整列を強制的に行わせるわけではないことに注意してください。強制的にデータ境界整列を行わせるには、-dalign または -aligncommon を使用してください。

-dalign オプションはマクロです。

-dalign は -xmemalign=8s -aligncommon=16 のマクロです。

詳細は、3-12 ページの「-aligncommon[={1|2|4|8|16}]」を参照してください。

-xnolib

-nolib と同義です。

-xnolibmil

-nolibmil と同義です。

-xnolibmopt

高速数学ライブラリを使用しません。

最適化された数学ライブラリとのリンクを無効にする場合に -fast と組み合わせ使用します。

```
f95 -fast -xnolibmopt …
```

-xO*n*

-O*n* と同義です。

-xopenmp

(SPARC) -openmp と同義です。

-xpad

-pad と同義です。

-xpagesize=*size*

(SPARC) スタックとヒープ用に優先ページサイズを設定します。

size には以下のいずれかを指定します。

8K 64K 512K 4M 32M 256M 2G 16G または default

例 : -xpagesize=4M

これらすべてのページサイズが、あらゆるプラットフォームでサポートされているわけではなく、アーキテクチャと Solaris 環境に依存します。指定するページサイズは、ターゲットプラットフォーム上で Solaris オペレーティング環境に有効なページサイズでなければなりません。有効なページサイズは getpagesizes(3C) によって返される値です。有効なページサイズを指定しないと、要求は実行時に暗黙的に無視されます。Solaris オペレーティング環境では、ページサイズ要求に従うという保証はありません。

実行中のプログラムが要求したページサイズを受け取ったかどうかを判断するには、pmap(1) または meminfo(2) を使用します。

-xpagesize=default を指定すると、フラグは無視されます。*size* 値を指定しないで -xpagesize を指定することは、-xpagesize=default と同義です。

このオプションは、-xpagesize_heap=*size* -xpagesize_stack=*size* のマクロです。これら 2 つのオプションは、-xpagesize と同じ次の引数を使用します。8K、64K、512K、4M、32M、256M、2G、16G、default のいずれか。両方に同じ値を設定するには -xpagesize=*size* を指定します。別々の値を指定するには個々に指定します。

このフラグを指定してコンパイルするのは、LD_PRELOAD 環境変数を同等のオプションで mpss.so.1 に設定するか、またはプログラムを実行する前に同等のオプションを指定して Solaris 9 コマンドの ppgsz(1) を実行するのと同じことです。詳細は、Solaris 9 マニュアルページを参照してください。

この機能は Solaris 7 および Solaris 8 オペレーティング環境では使用できません。このオプションを指定してコンパイルされたプログラムは、Solaris 7 および Solaris 8 オペレーティング環境ではリンクしません。

-xpagesize_heap=*size*

(SPARC) ヒープ用に優先ページサイズを設定します。

size には以下のいずれかを指定します。

```
8K 64K 512K 4M 32M 256M 2G 16G または default
```

例 : -xpagesize_heap=4M

詳細は、-xpagesize を参照してください。

-xpagesize_stack=*size*

(SPARC) スタック用に優先ページサイズを設定します。

size には以下のいずれかを指定します。

```
8K 64K 512K 4M 32M 256M 2G 16G または default
```

例 : -xpagesize_stack=4M

詳細は、-xpagesize を参照してください。

-xparallel

-parallel と同義です。

-xpg

-pg と同義です。

-xpp={fpp|cpp}

ソースファイルプリプロセッサを選択します。

デフォルトは -xpp=fpp です。

コンパイラは fpp(1) を使用して、.F、.f95、または .F03 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 や UltraSPARC III、Pentium 3、Pentium 4、AMD Opteron 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、v9b、sse、sse2、generic64、または amd64)。

Fortran PREFETCH 指令については、2-14 ページの 2.3.1.8 節「PREFETCH 指令」を参照してください。

a には以下のいずれかを指定します。

a	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。メモリー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もっと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化がなく、突然変化し、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、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。

-xprefetch_auto_type=[no%]indirect_array_access

間接アクセスされるデータ配列に対して間接先読み命令を生成します。

直接メモリーアクセスに対して先読み命令が生成されるのと同じ方法で、-xprefetch_level={1|2|3} オプションが指示するループに対して間接先読み命令を生成します (または生成しません)。接頭辞 no% は、この宣言を否定します。

-xprefetch_auto_type の値が指定されていない場合は、-xprefetch_auto_type=no%indirect_array_access に設定されます。

このオプションを使用するには、-xprefetch=auto および最適化レベル -xO3 以上が必須です。

-xdepend などのオプションは、メモリー別名のあいまいさを排除する情報の生成に役立つため、間接先読み候補の計算の積極性に影響し、このため、自動的な間接先読みの挿入が促進されることがあります。

-xprefetch_level={1|2|3}

先読み命令の自動生成をコントロールします。

このオプションは、次の設定でコンパイルしたときのみ有効です。

- -xprefetch=auto
- 最適化レベル 3 以上
- 先読みをサポートするプラットフォーム上 (-xarch=v8plus、v8plusa、v8plusb、v9、v9a、v9b、generic64、native64)。

-xprefetch_level を指定しない場合の -xprefetch=auto のデフォルトは レベル 2 です。

先読みレベル 2 は、レベル 1 よりも多くの先読み命令を生成します。先読みレベル 3 は、レベル 2 よりも多くの先読み命令を生成します。

先読み命令レベル 2 および 3 は、UltraSPARC III プラットフォーム (-xarch=v8plusb または v9b) や x86 Pentium 4、AMD Opteron (-xarch=sse2 または amd64) でのみ有効です。

-xprofile={collect[:*name*]|use[:*name*]|tcov}

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うか、基本ブロックのカバレージ分析を行います。

高い最適化レベルでのコンパイル (-xO5) が拡張されて、コンパイラには実行時パフォーマンスのフィードバックが提供されます。コンパイラが最良の最適化を実行するために必要なプロファイルフィードバックを生成するには、最初に -xprofile=collect を指定してコンパイルし、標準的なデータセットに対して実行可能ファイルを実行し、次に最高の最適化レベルと -xprofile=use を指定して再コンパイルします。

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ときに、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。

プロファイルの収集は「マルチスレッド (MT) 環境で安全」です。すなわち、-mt を指定して独自のマルチタスクを実行し、マルチタスクライブラリを直接呼び出すプログラムをプロファイル処理すると、正確な結果が得られます。

コンパイルとリンクを個別に実行する場合、コンパイルで -xprofile=collect を指定する場合は、リンクでも指定する必要があります。

use[:*nm*]

最適化レベル -xO5 で計画的に最適化するために、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、プロファイルデータ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 *nm* は同一のものを指定する必要があります。

収集 (collect) 段階と使用 (use) 段階の間のコンパイル速度を高める方法については、-xprofile_ircache も参照してください。

コンパイラがプロファイルデータファイルを検索するディレクトリを制御する方法については、-xprofile_pathmap も参照してください。

tcov

「新しい」形式の tcov を使用して、基本ブロックのカバレージ分析を行います。最適化レベルは、-O2 以上を指定してください。

コード計測方法は -a と似ていますが、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「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。

-xprofile_ircache[=*path*]

(SPARC) プロファイルの収集段階と使用段階の間、コンパイルデータを保存および再利用します。

収集段階で保存したコンパイルデータを再利用することによって使用段階のコンパイル時間を短縮するには、-xprofile=collect|use とともに使用します。

指定すると、*path* はキャッシュファイルが保存されているディレクトリを上書きします。デフォルトでは、これらのファイルはオブジェクトファイルと同じディレクトリに保存されます。収集段階と使用段階が 2 つの別のディレクトリで実行される場合は、パスを指定しておくと便利です。

一般的なコマンドシーケンスを次に示します。

```
demo% f95 -xO5 -xprofile=collect -xprofile_ircache t1.c t2.c
demo% a.out       collects feedback data
demo% f95 -xO5 -xprofile=use -xprofile_ircache t1.c t2.c
```

大きなプログラムでは、中間データが保存されるため、使用段階のコンパイル時間を大幅に向上させることができます。ただし、データを保存するために必要なディスク容量が増大します。

-xprofile_pathmap=*collect_prefix*:*use_prefix*

(SPARC) プロファイルデータファイル用のパスマッピングを設定します。

-xprofile_pathmap オプションは -xprofile=use オプションとともに使用します。

コンパイラが -xprofile=use でコンパイルされたオブジェクトファイルのプロファイルデータを見つけられず、以下の点に該当する場合は、-xprofile_pathmap を使用します。

- 前回 -xprofile=collect でコンパイルしたときに使用されたディレクトリとは異なるディレクトリで、-xprofile=use を指定してコンパイルしている。
- オブジェクトファイルはプロファイルで共通ベース名を共有しているが、異なるディレクトリのそれぞれの位置で相互に識別されている。

collect-prefix は、オブジェクトファイルが -xprofile=collect でコンパイルされたディレクトリツリーの UNIX パス名の接頭辞です。

use-prefix は、オブジェクトファイルが -xprofile=use を指定してコンパイルされたディレクトリツリーの UNIX パス名の接頭辞です。

-xprofile_pathmap の複数のインスタンスを指定すると、コンパイラは指定した順序でインスタンスを処理します。-xprofile_pathmap のインスタンスで指定された各 *use-prefix* は、一致する *use-prefix* が識別されるか、最後に指定された *use-prefix* がオブジェクトファイルのパス名と一致しないことが確認されるまで、オブジェクトファイルのパス名と比較されます。

-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。

通常、RECURSIVE 属性によって定義された副プログラムのみが再帰的に自分自身を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、RECURSIVE 属性で定義されていない副プログラムも、再帰的に自分自身を呼び出すことができます。ただし、RECURSIVE で定義されたサブルーチンとは異なり、このフラグを使用しても、デフォルトで局所変数がスタックに割り当てられることはありません。副プログラムの再帰的な呼び出しごとに異なる局所変数を持つ場合は、-stackvar を使用してコンパイルし、局所変数をスタックに設定します。

-xO2 以上の最適化レベルで間接的な再帰 (ルーチン A がルーチン B を呼び出し、その後にルーチン B がルーチン A を呼び出す) を実行すると、得られる結果に一貫性がない場合があります。-xrecursive フラグを指定してコンパイルすると、-xO2 以上の最適化レベルであっても、間接的な再帰を実行した場合の正確さが保証されます。

-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。

-xreduction

-reduction と同義です。

-xregs=*r*

(SPARC) 使用するレジスタを指定します。

r 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。

[no%]appl、[no%]float

no には % を付けてください。

例 : -xregs=appl,no%float

- appl : コンパイラがアプリケーションレジスタをスクラッチレジスタとして使用することを許可します。

  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=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 : コンパイラが浮動小数点レジスタを整数値用のスクラッチレジスタとして使用することを許可します。このオプションは、コンパイラが浮動小数点値用に浮動小数点レジスタを使用することには影響しません。
- 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、ソースプログラムには浮動小数点のコードが含まれなくなります。

コンパイラのデフォルトは次のとおりです。-xregs=appl,float。

-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。

-xs を指定すると、すべてのデバッグ情報が実行可能ファイルにコピーされます。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 ファイルを維持できない場合に使用します。

-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。

xsafe=mem-xsafe=mem

(SPARC) コンパイラは、メモリー保護の違反が発生していないことを想定できます。

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 -O5 で使用する場合に限り有効です : v8plus、v8plusa、v8plusb、v9、v9a、または v9b。

注意 – 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

(廃止) -sb と同義です。

-xsbfast

(廃止) -sbfast と同義です。

-xspace

コードのサイズが増大するような最適化は行いません。

例 : 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*

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、*platform–name* 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 に拡張されます。

platform–name : 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。

SPARC プラットフォーム

実行中のシステムで -xtarget=native の展開を調べるには、fpversion(1) コマンドを使用します。

特定のホストプラットフォームで -xtarget を展開した場合、そのプラットフォームでコンパイルすると -xtarget=native と同じ -xarch、-xchip、または -xcache 設定にならない場合があります。

次の表は、コンパイラが認識できる、一般に使用されているシステムプラットフォーム名の一覧です。付録 C は、古くてあまり使用されていないシステムプラットフォーム名の一覧です。

表 3-18　一般に使用されている -xtarget システムプラットフォームの展開

-xtarget= プラットフォーム名	-xarch	-xchip	-xcache
generic	generic	generic	generic
generic64	v9	generic	generic
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
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
urtra2i	v8plusa	urtra2i	16/32/1:512/64/1
ultra3	v8plusa	ultra3	64/32/4:8192/512/1

表 3-18　一般に使用されている -xtarget システムプラットフォームの展開 (続き)

-xtarget= プラットフォーム名	-xarch	-xchip	-xcache
ultra3cu	v8plusa	ultra3cu	64/32/4:8192/512/2
ultra3i	v8plusa	ultra3i	64/32/4:1024/64/4
ultra4	v8plusa	ultra4	64/32/4:8192/128/2

UltraSPARC V9 での 64 ビット Solaris 向けのコンパイルは、-xarch=v9 または -xarch=v9a フラグで指示します。-xtarget=ultra または ultra2 の指定は必要でなく、十分でもありません。-xtarget を指定する場合は、次に示すように、そのフラグの後に -xarch=v9 または v9a オプションも指定する必要があります。

```
-xtarget=ultra2 ...-xarch=v9
```

この指定がないと、-xtarget の設定によって、-xarch が v8plusa に戻されます。

x86 プラットフォーム

x86 システムで有効な -xtarget プラットフォーム名は次のとおりです。

generic、native、386、486、pentium、pentium_pro、pentium3、pentium4、opteron

64 ビット x86 AMD Opteron プラットフォームでの 64 ビット Solaris 向けのコンパイルは、-xarch=amd64 フラグで指示します。-xtarget=opteron でのコンパイルは、必要でもなく、十分でもありません。 -xtarget を指定する場合は、次に示すように、そのフラグの後に -xarch=amd64 オプションも指定する必要があります。

```
-xtarget=opteron -xarch=amd64
```

この指定がないと、32 ビット x86 用のコンパイルに戻ります。

-xtime

-time と同義です。

-xtypemap=*spec*

デフォルトのデータサイズを指定します。

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。このオプションは、デフォルトのサイズの変数および定数に適用されます。

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: サイズ
double: サイズ
integer: サイズ
```

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

例：

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

デフォルトの REAL および DOUBLE を 8 バイトにマップします。

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。

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。

-xvector[={yes|no}]

ベクトルライブラリ関数を自動呼び出しします。

-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。

-ztext

再配置を伴わない純粋なライブラリだけを生成します。

-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 とともに使用します。

-ztext を指定すると、*text* 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。

第4章

Fortran 95 の機能と相違点

この章では、標準の Fortran 95 と Fortran 95 コンパイラ `f95` の機能の主な相違点について説明します。

4.1 ソース言語の機能

Fortran 95 コンパイラは、Fortran 95 標準規則に対して、以下のソース言語の機能および拡張機能を提供します。

4.1.1 継続行の制限

`f95` 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。

4.1.2 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 桁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。

f95 は、タブがあると以降その行の 72 桁目までパディングします。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.
```

4.1.3 想定するソースの書式

f95 が想定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 4-1 F95 ソース書式のコマンド行のオプション

オプション	コンパイラの動作
-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。また、!DIR$ FREE 指令または !DIR$ FIXED 指令は、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。

4.1.3.1 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 のコマンド内で、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。
- 1 つのファイル内で、!DIR$ FREE 指令または !DIR$ FIXED 指令を使用すると、自由形式と固定形式を混在させることができます。

4.1.3.2 大文字・小文字の区別

Sun Fortran 95 では、デフォルトでは大文字と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。

4.1.4 制限とデフォルト

- Fortran 95 のプログラム単位には、最大 65,535 個の構造型、および最大 16,777,215 個の定数を定義できます。
- 変数および他のオブジェクトの名前は最大 127 文字です。31 文字が標準です。

4.2 データ型

ここでは、Fortran 95 データ型の特徴と拡張機能について説明します。

4.2.1 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。

4.2.1.1 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。
- 論理要素は、算式、関係式、またはビット単位の論理式において無効です。ブール要素はいずれにおいても有効です。

4.2.1.2 ブール型定数の代替書式

f95 では、ブール型定数 (8 進、16 進、ホレリス) を、以下のような書式 (2 進ではありません) で使用することができます。ただし変数はブール型として宣言できません。標準の Fortran では、このような書式は許されていません。

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、値は右揃えになります。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 0 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、値は右揃えになります。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 0 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。

```
nH...          '...'H          "..."H
nL...          '...'L          "..."L
nR...          '...'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	1 ワード 32 ビットでの内部の 8 進数
0B	00000000000
77740B	00000077740
X"ABE"	00000005276
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。

4.2.1.3 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 文でのみ使用できます。

4.2.2 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 4-2　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
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 倍精度浮動小数点数 (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)

4.2.3 データ型のサイズおよび整列

記憶領域および整列は常にバイト単位で表されます。シングルバイトに収まる値は、バイト整列です。

データ型のサイズおよび整列は、コンパイラのオプションとプラットフォーム、および変数の宣言方法に依存します。COMMON ブロック内のデフォルトの最大の整列は、4 バイトの境界整列です。

デフォルトのデータ整列および記憶領域の割り当ては、`-aligncommon`、`-f`、`-dalign`、`-dbl_align_all`、`-xmemalign`、および `-xtypemap` などの特別なオプションを指定してコンパイルすることにより、変更できます。このマニュアルは、これらのオプションが有効でないものとして記述されています。

いくつかプラットフォームにおける特殊ケースのデータ型および整列については、『Fortran プログラミングガイド』の第 11 章に追加の説明があります。

デフォルトのサイズおよび整列を次の表にまとめます (データ型のその他の点およびオプションは考慮していません)。

表 4-3 デフォルトのデータサイズおよび整列 (バイト)

Fortran 95 データ型	サイズ	デフォルトの整列	COMMON 内の整列
`BYTE X`	1	1	1
`CHARACTER X`	1	1	1
`CHARACTER*`*n* `X`	n	1	1
`COMPLEX X`	8	4	4
`COMPLEX*8 X`	8	4	4
`DOUBLE COMPLEX X`	16	8	4
`COMPLEX*16 X`	16	8	4
`COMPLEX*32 X`	32	8/16	4
`DOUBLE PRECISION X`	8	8	4
`REAL X`	4	4	4
`REAL*4 X`	4	4	4
`REAL*8 X`	8	8	4
`REAL*16 X`	16	8/16	4
`INTEGER X`	4	4	4
`INTEGER*2 X`	2	2	2
`INTEGER*4 X`	4	4	4
`INTEGER*8 X`	8	8	4

表 4-3　デフォルトのデータサイズおよび整列 (バイト) (続き)

Fortran 95 データ型	サイズ	デフォルトの整列	COMMON 内の整列
LOGICAL X	4	4	4
LOGICAL*1 X	1	1	1
LOGICAL*2 X	2	2	2
LOGICAL*4 X	4	4	4
LOGICAL*8 X	8	8	4

次の点に注意してください。

- REAL*16 および COMPLEX*32：64 ビット環境 (-xarch=v9 または v9a を指定してコンパイル) では、デフォルトの整列は、表で 8/16 と示されるように、8 バイトではなく 16 バイト境界整列になります。x86 プラットフォームでは、この「4 倍精度」データ型は使用できません。
- 配列および構造体は、その要素または欄に従って整列します。配列は、配列要素と同じように整列します。構造体は、もっとも広い整列で整列する欄と同じように整列します。

オプション -f または -dalign は、8、16、または 32 バイトのすべてデータを、強制的に 8 バイト境界で整列させます。オプション -dbl_align_all の場合は、すべてのデータが 8 バイト境界で整列します。これらのオプションを使用するプログラムには、移植性がない場合があります。

4.3 Cray ポインタ

Cray ポインタとは、別の言語要素 (変数や配列など) のアドレスを値に持つ変数のことです。この別の言語要素のことを、「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が、標準の Fortran 95 はサポートしていません。

4.3.1 構文

Cray ポインタの POINTER 文は以下の形式で記述します。

```
POINTER  ( pointer_name, pointee_name [array_spec] ), …
```

pointer_name、*pointee_name*、*array_spec* のそれぞれの意味は、以下のとおりです。

pointer_name 対応する *pointee_name* へのポインタです。
pointer_name には *pointee_name* アドレスが含まれます。

制限事項 : *pointer_name* にはスカラー変数名を指定してください (構造型は指定できません)。
禁止事項 : 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

pointee_name 対応する *pointer_name* 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。

array_spec *array_spec* 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持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` × `m+1` 次元の配列であることを宣言しています。

4.3.2 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付け、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、メモリーの絶対アドレスにアクセスすることができます。

4.3.3 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 (*pointer_name*, *pointee_name* [*array_spec*])

Fortran 95 のポインタは次のように宣言します。

POINTER *object_name*

この 2 種類のポインタを混在させることはできません。

4.3.4 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
  - 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時ではなく、副プログラムの処理が始まる時に認識されます。

4.3.5 Cray ポインタの制限事項

- *pointer_name* は、CHARACTER*(*) で型宣言された変数であってはなりません。
- *pointer_name* 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持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 ポインタの指示先にする
- 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

4.3.6 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構造型にすることはできません。

4.3.7 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 + 4000
      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 は無名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。

4.4 STRUCTURE および 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」に相当します。構文についての詳細は、『FORTRAN 77 言語リファレンス』を参照してください。

STRUCTURE の欄宣言は、次のいずれかになります。

- 副構造体 - 別の STRUCTURE 宣言、または事前に定義された記録。
- 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 (これは特に f95 の場合のみです)。

f77 と同様に、POINTER 文を欄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。

- 構造体の欄宣言の記号として、「.」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。

4.5 符号なし整数

Fortran 95 コンパイラでは言語が拡張され、新しいデータ型である UNSIGNED が使用可能になりました。UNSIGNED では、KIND (種別) パラメータに対して指定できる値は 4 つです。パラメータ値、1、2、4、8 はそれぞれ 1、2、4、8 バイトの符号なし整数に対応します。

符号なし整定数は、数字列の後に大文字または小文字の U が付き、場合によっては下線と種別パラメータが続くという形式です。次の例では、符号なし整定数の最大値が示されています。

```
255u_1
65535u_2
4294967295U_4
18446744073709551615U_8
```

種別パラメータが付いていない場合 (12345U) は、デフォルトは基本整数の場合と同じです。この場合、デフォルトは U_4 ですが、-xtypemap オプションを使うとデフォルトの符号なし整数の種別が変更されます。

UNSIGNED 種別指定子を使って、符号なし整変数または配列を宣言します。

```
UNSIGNED U
UNSIGNED(KIND=2) ::A
UNSIGNED*8 ::B
```

4.5.1 演算式

- + - * / といった 2 項演算子は、符号あり、符号なしの演算対象をともに指定することはできません。つまり、U が UNSIGNED として宣言され、N が符号ありの INTEGER である場合、U*N は不正です。

- 2 項演算に符号あり、符号なしの演算対象を混在させる場合は、U*UNSIGNED(N) のように、UNSIGNED 組み込み関数を使用します。
- ただし、一方の演算対象が符号なし整数で、もう一方が符号あり整定数式で値が正かゼロである場合は例外で、結果は符号なし整数となります。
- このような混在した式の結果の種別は、演算対象の最大種別となります。

- 符号ありの値のべき乗は符合ありに、符号なしの値のべき乗は符号なしになります。
- 符号なしの値の単項マイナスは符号なしになります。
- 符号なし演算対象は、実数と複素数を自由に混在させることができます (符号なし演算対象は、区間演算対象と混在させることはできません)。

4.5.2 関係式

関係組み込み演算を使用して、符号あり整数と符号なし整数の演算対象を比較できます。結果は演算対象の変更されない値に基づいて決まります。

4.5.3 制御構文

- CASE 構文では符号なし整数が場合式として使用可能です。
- 符号なし整数は、DO ループ制御変数としても、また、算術 IF 文の制御式中でも使用できません。

4.5.4 入出力構文

- 符号なし整数は I、B、O、Z の各編集記述子を使用して読み取り、書き込みが可能です。
- また、並び入出力、変数群入出力を使っても読み取り、書き込みが可能です。並び入出力あるいは変数群入出力による符号なし整数の書き込み形式は、正の符号あり整数の場合と同じです。
- 符号なし整数は、書式なし入出力によっても読み取り、書き込みできます。

4.5.5 組み込み関数

- 符号なし整数は組み込み関数内で使用できますが、例外として SIGN および ABS 関数では使用できません。
- 新しい組み込み関数、UNSIGNED は、INT と似ていますが、符号なし型を結果として生成します。形式は次のとおりです。

UNSIGNED(*v* [,*kind*])

- もう 1 つの新しい組み込み関数、SELECTED_UNSIGNED_KIND(*var*) は、*var* の種別パラメータを返します。
- MIN 関数と MAX 関数では、REAL 型の演算対象が少なくとも 1 つ存在しないと、符号あり整数演算対象も符号なし整数演算対象も使用できません。
- 符号なし配列は、配列組み込み関数の引数として使えません。

4.6 Fortran 2003 の機能

今回の f95 コンパイラのリリースには、Fortran 2003 の規格草稿で提案された多数の機能が含まれています。

4.6.1 C 関数との相互運用性

Fortran の新しい規格草稿には次のものが含まれています。

- C 言語手続きを参照する方法、および反対に C 関数から Fortran 副プログラムを参照できるよう指定する方法
- 外部 C 変数とリンクする大域変数を宣言する方法

ISO_C_BINDING モジュールは、C の型と互換のデータを表す種別パラメータである名前付き定数へのアクセスを可能にします。

この規格草稿は、BIND(C) 属性も取り入れています。Fortran の構造型は、BIND 属性を持つものならば、C と相互に利用できます。

Fortran 95 コンパイラの今回のリリースでは、規格草稿の第 15 章に記述されている機能を実現します。また、規格草稿第 4 章に述べられている、C の型に対応する構造型、リスト、型別名を定義する機能を備えます。

4.6.2 IEEE 浮動小数点の例外処理

新しい組み込みモジュール、IEEE_ARITHMETIC および IEEE_FEATURES は、Fortran 言語における例外と IEEE 演算をサポートします。次のように指定すると、これらの機能がすべてサポートされます。

```
USE, INTRINSIC ::IEEE_ARITHMETIC
USE, INTRINSIC ::IEEE_FEATURES
```

これらのモジュールは、一連の構造型、定数、丸めモード、要素別処理関数、種別関数、要素別処理サブルーチン、非要素別処理サブルーチンを定義します。詳細は Fortran 2000 規格草稿の第 14 章に説明があります。

4.6.3 コマンド行引数用組み込み関数

Fortran 2003 規格草稿では、コマンド行引数および環境変数を処理するための組み込み関数が新しく導入されています。それら組み込み関数は次の 3 つです。

- GET_COMMAND(*command*, *length*, *status*)

  command で、プログラムを呼び出したコマンド行全体を返します。

- GET_COMMAND_ARGUMENT(*number*, *value*, *length*, *status*)

  value でコマンド行引数を返します。

- GET_ENVIRONMENT_VARIABLE(*name*, *value*, *length*, *status*, *trim_name*)

  環境変数の値を返します。

4.6.4 PROTECTED 属性

Fortran 95 コンパイラでは、新たに Fortran 2003 の PROTECTED 属性が使えるようになりました。PROTECTED はモジュール要素の使用に制限を設けます。PROTECTED 属性を持つオブジェクトは、それ自身が宣言されるモジュール内でのみ定義可能です。

4.6.5 Fortran 2003 非同期入出力

コンパイラは入出力文中の ASYNCHRONOUS 指定子を認識します。

```
ASYNCHRONOUS=['YES' | 'NO']
```

この構文は Fortran 2000 規格草稿の第 9 章で提案されているものです。WAIT 文とともに使うことで、コンピューティングで重複する可能性のある入出力処理を指定することができます。このコンパイラは ASYNCHRONOUS='YES' を認識しますが、規格草稿は実際の非同期入出力を要求しません。今回のコンパイラのリリースでは、入出力は常に同期します。

4.6.6 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で使用が禁止されています。COMMON ブロックと EQUIVALENCE 文。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。

4.6.7 VALUE 属性

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 2003 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
```

4.6.8 Fortran 2003 ストリーム入出力

新しい「ストリーム」入出力スキーマは、Fortran 2003 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。

4.6.9 Fortran 2003 の書式付き入出力機能

3 つの新しい Fortran 2003 書式付き入出力指定子が、f95 に実装されています。これらの指定子は、OPEN、READ、WRITE、PRINT、および INQUIRE 文で指定されます。

- DECIMAL=['POINT'|'COMMA']

  デフォルトの小数部分編集モードを変更します。デフォルトでは、ピリオドによって、D、E、EN、ES、F、および G 編集によって書式付けされた数値全体と、浮動小数点の少数部分が分離されます。'COMMA' は、123,456 のように、印刷の際に、ピリオドの代わりにコンマを使用するようにデフォルトを変更します。デフォルトの設定は、123.456 のように、印刷の際にピリオドを使用する 'POINT' です。

- ROUND=['PROCESSOR_DEFINED' | 'COMPATIBLE']

  書式付き入出力 D、E、EN、ES、F、および G 編集のデフォルトの丸めモードを設定します。'COMPATIBLE' と指定する場合は、データ変換によって得られる値は、2 つのもっとも近い表示値のうち、より近い方の表示値になります。値が表示値のちょうど中間である場合は、0 から離れている方の表示値になります。'PROCESSOR_DEFINED' を指定する場合は、丸めモードはプロセッサのデフォルトのモードに依存します。ROUND が指定されていない場合は、丸めモードはコンパイラのデフォルトになります。

  たとえば、WRITE(*,'(f11.4)') 0.11115 は、デフォルトのモードでは 0.1111、'COMPATIBLE' モードでは 0.1112 になります。

- IOMSG=*character-variable*

  指定された文字変数に文字列としてエラーメッセージを返します。これは、標準の出力で表示されるエラーメッセージと同じです。最長メッセージが保持可能な大きさの文字バッファーを割り当ててください (CHARACTER*256 で十分です)。

INQUIRE 文で使用する場合は、これらの指定子は、現在の値を返すための文字変数を宣言します。

新しい編集記述子 DP、DC、RP、および RC は、単一の FORMAT 文内のデフォルトの設定を、それぞれ、小数点、少数部のコンマ、プロセッサ定義の丸め、および互換性のある丸めに変更します。たとえば、次のとおりです。

```
WRITE(*,'(I5,DC,F10.3)') N, W
```

F10.3 出力項目のピリオドの代わりにコンマが使用されます。

書式付き入出力の浮動小数点丸めモードの変更については、-iorounding コンパイラコマンド行オプションも参照してください (3-35 ページの「-iorounding[={compatible|processor-defined}]」)。

4.7 新しい入出力拡張機能

ここでは、Fortran 2003 規格草稿には含まれていませんが、f95 コンパイラで使用可能な Fortran 95 入出力処理の拡張機能について説明します。FORTRAN 77 コンパイラ、f77 の入出力拡張機能の一部は、Fortran 95 コンパイラに組み込まれました。

4.7.1 入出力エラー処理ルーチン

2 つの新機能によって、ユーザーは独自に論理ユニットの書式付き入力のエラー処理ルーチンを指定できます。書式エラーが検出されると、実行時入出力ライブラリが、エラーの原因となった入力中の文字を表すデータを付けて、指定されたユーザー定義のハンドラを呼び出します。ハンドラルーチンは、代わりの文字を提供してエラーが検出された時点から入出力処理を続けるか、デフォルトの Fortran エラー処理を実行することができます。

この新しいルーチン、SET_IO_ERR_HANDLER(3f) と GET_IO_ERR_HANDLER(3f) はモジュールサブルーチンであり、これらを呼び出すルーチンの中には USE SUN_IO_HANDLERS が必要です。これらのルーチンの詳細については、マニュアルページを参照してください。

4.7.2 可変フォーマット式

FORTRAN 77 では、ある書式の整定数を、山カッコで囲まれた任意の式に置き換えることができました。

```
1  FORMAT( ... < expr > ... )
```

*n*H... 編集記述子の *n* として、あるいは ASSIGN 文の参照する FORMAT 文、または並列化領域内の FORMAT 文では、可変フォーマット式は使えません。

この機能は f95 で可能になり、-f77 互換性オプションフラグは不要です。

4.7.3 NAMELIST 入力形式

- 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、記録の最初の桁から開始することができます。

4.7.4 書式なしバイナリ入出力

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、記録長がファイルに埋め込まれません。このデータがないと、どこで記録が開始し、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。ファイル上に記録マークがないので、「記録の終わり」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" を使用して開かれたファイルに対して INQUIRE を実行すると、次のものが返されます。

  ```
  FORM="BINARY"
  ACCESS="SEQUENTIAL"
  DIRECT="NO"
  FORMATTED="NO"
  UNFORMATTED="YES"
  RECL= および 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。

4.7.5 その他の入出力拡張機能

- 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記録長です。
- 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。
- 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
```

4.8 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。

Sun 形式の並列化指令では、f95 -explicitpar がデフォルトです。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。OpenMP 指令を使用した明示的な並列化では、-openmp を指定してコンパイルしてください。

Fortran の指令については、付録 D にまとめられています。

4.8.1 f95 の特殊な指令行の書式

f95 は、第 2 章で説明した指令に加え、独自の特別な指令を認識します。これらの指令は、次のような構文になります。

```
!DIR$ d1, d2, ...
```

4.8.1.1 ソースが固定形式の場合

- CDIR$ または !DIR$ を 1 桁目から 5 桁目に記述します。
- 指令を 7 桁目以降に記述します。
- 73 桁目以降は無視されます。
- 新たに始まる指令行では、6 桁目を空白とします。
- 指令の継続行では、6 桁目に空白以外の文字を記述します。

4.8.1.2 ソースが自由形式の場合

- !DIR$ の後に空白を 1 つ付けて、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 桁目から 5 桁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。

4.8.2 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。

4.8.2.1 スコープ

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。

4.8.2.2 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します。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。

4.8.2.3 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
```

4.8.3 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 D と『Fortran プログラミングガイド』に記載されています。f95 は、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。OpenMP 指令の並列化については、『OpenMP API ユーザーズガイド』を参照してください。

4.9 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、-moddir=*dir* フラグまたは MODDIR 環境変数で指定されたディレクトリに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。指定されていない場合は、現在の作業ディレクトリにある .mod ファイル に書き込みます。

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。ただし、これは必須ではありません。

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。

```
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
```

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。

4.9.1 モジュールの検索

Sun Studio Fortran 95 コンパイラでは、.mod ファイルはアーカイブ (.a) ファイルに格納できます。アーカイブファイルは、モジュールを検索するコマンド行で、-M*path* フラグによって明示的に指定する必要があります。デフォルトでは、コンパイラはアーカイブファイルを検索しません。

USE 文にある .mod ファイルのみが検索されます。たとえば、Fortran 95 の USE mymod によって、コンパイラは、デフォルトでモジュールファイル mymod.mod を検索します。

検索時には、モジュールファイルが記述されるディレクトリが優先されます。これは、-moddir=*dir* オプションフラグおよび MODDIR 環境変数によってコントロールできます。つまり、-M*path* オプションのみが指定されている場合は、モジュールに対して、-M フラグに示されたディレクトリおよびファイルよりも先に、現在のディレクトリが検索されます。

4.9.2 -use=*list* オプションフラグ

-use=*list* フラグによって、1 つ以上の暗黙的な USE 文がこのフラグを指定してコンパイルされる副プログラムまたはモジュールの副プログラムに挿入されます。このフラグを使用すると、モジュールまたはモジュールファイルが、ライブラリまたはアプリケーションの機能のために要求された場合に、ソースプログラムを修正する必要がなくなります。

-use=*module_name* を使用してコンパイルすると、USE *module_name* をコンパイルされる各副プログラムまたはモジュールに追加する効果があります。-use=*module_file_name* を使用してコンパイルすると、*module_file_name* ファイルに含まれる各モジュールに USE *module_name* を追加する効果があります。

4.9.3 fdumpmod コマンド

fdumpmod(1) コマンドを使用すると、モジュール情報ファイルの内容を表示できます。

```
demo% fdumpmod x.mod group.mod
x 1.0 v8,i4,r4,d8,n16,a4 x.mod
group 1.0 v8,i4,r4,d8,n16,a4 group.mod
```

fdumpmod コマンドによって、単一の .mod ファイル、連結される .mod ファイルによって形成されるファイル、.mod ファイルの .a アーカイブにあるモジュールについての情報が表示されます。表示には、モジュール名、バージョン番号、対象のアーキテクチャ、およびそのモジュールと互換性のあるコンパイルオプションを示すフラグが含まれます。詳細は、fdumpmod(1) マニュアルページを参照してください。

4.10 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 4-4 非標準の組み込み関数

名前	定義	関数の型	引数の型	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。

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 95 が認識可能な FORTRAN 77 の組み込み関数など、組み込み関数についての詳細は、『Fortran ライブラリ・リファレンス』を参照してください。

4.11 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。

4.12 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。これは、C と Fortran では呼び出し規則が共通なためです。C と Fortran のルーチンの相互運用についての詳細は、『Fortran プログラミングガイド』の「C と Fortran のインタフェース」の章を参照してください。

第5章

FORTRAN 77 の互換性：Fortran 95 への移行

Fortran 95 コンパイラ `f95` は、ほとんどの従来の FORTRAN 77 プログラムをコンパイルできます。これには、`f77` コンパイラによって以前にコンパイルされた非標準拡張機能を利用するプログラムも含まれます。

`f95` は、これらの FORTRAN 77 の機能の多くを直接的に受け入れます。その他の機能は、FORTRAN 77 互換モード (`f95 -f77`) でコンパイルする必要があります。

この章では、`f95` で使用可能な FORTRAN 77 の機能について説明し、`f95` との互換性がない `f77` の機能のリストを示します。`f77` コンパイラで使用可能な非標準 FORTRAN 77 拡張機能についての詳細は、`http://docs.sun.com/source/806-4841/index.html` の『FORTRAN 77 言語リファレンス』を参照してください。

`f95` コンパイラで使用可能な Fortran 95 言語のその他の拡張機能については、第 4 章を参照してください。

`f95` は、標準規則に準拠する FORTRAN 77 プログラムをコンパイルできます。移植性を確保するためには、非標準 FORTRAN 77 機能を利用しているプログラムを標準に準拠する Fortran 95 に移行する必要があります。`-ansi` オプションを指定してコンパイルすると、プログラム中で使用されているすべての非標準機能にフラグが立てられます。

5.1 互換性のある `f77` 機能

`f95` では、次の FORTRAN 77 コンパイラ `f77` の非標準機能を、直接的でも、`-f77` 互換性モードでも使用できます。

- **ソースの書式**
  - 継続行は、カラム 1 に「`&`」を設定して始めることができます。[`-f77=misc`]
  - インクルードファイルの最初の行は、継続行の場合があります。[`-f77=misc`]

- f77 タブ書式を使用します。[-f77=tab]
- タブ書式は、ソース行を 72 桁以上に拡張できます。[-f77=tab]
- f95 のタブ書式では、文字列が継続行に及ぶ場合、72 桁目で途切れることはありません。[-f77]

■ 入出力

- Fortran 95 では、ACCESS='APPEND' でファイルを開くことができます。
- 並び出力は f77 コンパイラと類似する形式を使用します。[-f77=output]
- f95 は、直接探査ファイルの BACKSPACE を許可しますが、ENDFILE は許可しません。
- f95 では、形式編集記述子で欄幅を暗黙的に指定できます。たとえば、FORMAT(I) が許可されます。
- f95 は、出力形式で f77 のエスケープシーケンス (\n \t \) を認識します。[-f77=backslash]
- f95 は、OPEN 文の FILEOPT= を認識します。
- f95 では、STATUS='KEEP' を使用して、SCRATCH ファイルを閉じることができます [-f77]。プログラムが終了しても、検索ファイルは削除されません。SCRATCH ファイルは、-f77 を指定してコンパイルすれば、FILE=*name* を使用して開くこともできます。
- 内部ファイルの直接的な入出力を実行できます。[-f77]
- f95 は、FORTRAN 77 形式編集記述子 A、$、および SU を認識します。[-f77]
- FORM='PRINT' を、OPEN 文で使用できます。[-f77]
- f95 は、従来の FORTRAN 入出力文 ACCEPT および TYPE を認識します。
- FORTRAN 77 形式の NAMELIST 出力を記述するには、-f77=output を指定してコンパイルします。
- ERR= のみを指定した READ (IOSTAT= も END= 分岐もない場合) は、EOF が検出されると、ERR= 分岐を END= として取り扱います。[-f77]
- VMS Fortran NAME=*filename* を、OPEN 文で使用できます。[-f77]
- f95 では、READ() または WRITE() の後ろの余分なコンマを受け入れます。[-f77]
- END= 分岐は、REC= による直接探査 READ で使用できます。[-f77=input]
- 形式編集記述子 E*w*.*d*.*e* が使用でき、これは E*w*.*d*.E*e* として取り扱われます。[-f77]
- 入力文 FORMAT で文字列を使用できます。[-f77=input]
- IOSTAT= 指定子を、ENCODE/DECODE 文で使用できます。
- ENCODE/DECODE 文で、並び入出力が使用できます。
- 入出力文の論理ユニットとして使用される場合、アスタリスク (*) を STDIN および STDOUT の代わりに使用できます。

- FMT= 指定子で配列を使用できます。[-f77=misc]
- PRINT 文で変数群名を使用できます。[-f77=output]
- コンパイラは、FORMAT 文の余分なコンマを受け付けます。
- NAMELIST 入力実行中に疑問符 (?) を入力すると、読み込まれた変数群の名前が返されます。[-f77=input]

- **データ型、宣言、および用法**
  - プログラム単位において、別の宣言文の後ろに IMPLICIT 文が記述される場合もあります。
  - f95 では、IMPLICIT UNDEFINED 文が使用できます。
  - f95 では、FORTRAN 77 拡張機能 AUTOMATIC 文を使用できます。
  - f95 では、STATIC 文が使用でき、これは SAVE 文のように取り扱われます。
  - f95 では、VAX STRUCTURE、UNION、および MAP 文が使用できます (4-12 ページの 4.4 節「STRUCTURE および UNION (VAX Fortran)」を参照)。
  - Fortran 95 では、LOGICAL 変数と INTEGER 変数を置き換えて使用できます。[-f77=logical]
  - INTEGER 変数は、DO WHILE などの条件式で使用できます。[-f77=logical]
  - Cray ポインタは、組み込み関数の呼び出しに使用できます。
  - f95 では、型宣言で、スラッシュを使用したデータ初期化を実行できます。例 : REAL MHW/100.101/, ICOMX/32.223/
  - f95 では、Cray 文字ポインタを、非ポインタ変数および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ができます。
  - f95 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できます。
  - POINTER として宣言されたものと同じプログラム単位で Cray ポインタを INTEGER として宣言できます。INTEGER 宣言は無視されます。[-f77=misc]
  - Cray ポインタは、割り算や掛け算の演算で使用できます。[-f77=misc]
  - ASSIGN 文の変数の型を INTEGER*2 にすることができます。[-f77=misc]
  - 代替 RETURN 文の表現を非整数型にすることができます。[-f77=misc]
  - SAVE 属性を保持する変数は、COMMON ブロックの要素と同等化できます。
  - 同じ配列の初期化指定子に異なる型を使用できます。例 : REAL*8 ARR(5) /12.3 1, 3, 5.D0, 9/
  - 名前リスト項目の型宣言は、NAMELIST 文に後続できます。
  - f95 では、BYTE データ型が使用できます。
  - f95 では、非整数を配列添字として使用できます。[-f77=subscript]
  - 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す。[-f77=logical]

- f95 では、従来の f77 VIRTUAL 文が使用でき、これは DIMENSION 文のように取り扱われます。
- 異なるデータ構造は、f77 コンパイラと互換性のある方法で等価にされます。[-f77=misc]
- f77 コンパイラと同様に、f95 では、PARAMETER 文の初期化式で、多くの組み込み関数が使用できます。
- f95 では、整数値を CHARACTER*1 変数に割り当てることができます。[-f77=misc]
- 指数として BOZ が使用できます。[-f77=misc]
- BOZ 定数は文字変数に割り当てることができます。
  例：character*8 ch
  ch ="12345678"X
- BOZ 定数は、組み込み関数呼び出しの引数として使用できます。[-f77=misc]
- 文字変数は、DATA 文の整数値で初期化できます。変数の先頭文字は整数値に設定され、残りの文字列 (文字列が 2 文字以上の場合) は空白になります。
- ホレリス文字の整数配列を形式記述子として使用できます。[-f77]
- 浮動小数点の例外が生成される場合、定数の折りたたみは実行されません。[-f77=misc]
- -f77=misc を指定してコンパイルすると、f95 は、f77 コンパイラの方法で、自動的に REAL 定数を、引数、データ、およびパラメータ文に適切な種類 (REAL*8 または REAL*16) にします。[-f77=misc]
- 割り当てられた GOTO で、等価にされた変数を使用できます。[-f77]
- 非定数文字式を数値変数に割り当てることができます。
- -f77=misc でコンパイルを実行すると、型宣言の変数名の後に **kind* を配置できます。[-f77=misc]
  例：REAL Y*4, X*8(21))
  INTEGER FUNCTION FOO*8(J)
- 部分文字列を、DATA 文の DO 形並びの対象として使用できます。
  [-f77=misc] 例：DATA (a(i:i), i=1,n) /n*'+'/
- 括弧で囲まれた整数式は、型サイズとして配置できます。
  例：PARAMETER (N=2)
  INTEGER*(N+2) K

- **プログラム、サブルーチン、関数、および実行文**
  - f95 では、**名前**を設定するために PROGRAM 文は必要ありません。
  - 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す。[-f77]
  - 関数は、定義された戻り値を持つ必要はありません。[-f77]
  - 選択戻り指定子 (**label* または &*label*) を実際のパラメータリストおよび別の位置で使用できます。[-f77=misc]
  - %VAL を COMPLEX 型の引数とともに使用できます。[-f77=misc]

- %REF および %LOC を利用できます。[-f77=misc]
- サブルーチンは、RECURSIVE キーワードを使用して自分自身を宣言しないでも、自分自身を再帰的に呼び出すことができます。[-f77=misc] ただし、間接的な再帰を実行するプログラム (ルーチン A がルーチン B を呼び出し、その後にルーチン B がルーチン A を呼び出す) は、正しく動作させるために -xrecursive フラグでコンパイルする必要があります。
- 代替リターンを保持するサブルーチンは、ダミー引数のリストに代替リターンのリストがない場合でも呼び出すことができます。
- -f77=misc を指定してコンパイルすると、INTEGER または REAL 型以外の引数を使用して文関数を定義でき、実際の引数は文関数で定義された型に変換されます。[-f77=misc]
- ヌルの実引数を許可します。
  例 : CALL FOO(I,,,J) には、先頭の I と末尾の J 引数の間に 2 つの null 引数があります。
- f95 では、関数 %LOC() の呼び出しは LOC() として取り扱われます。[-f77=misc]
- **、* など別の演算子の後に単項プラスや単項マイナスを配置できます。
- 最初の引数が COMPLEX 型であっても、CMPLX() 組み込み関数を保持する第 2 引数を使用できます。この場合、最初の引数の実数部が使用されます。[-f77=misc]
- CHAR() 組み込み関数の引数が 255 文字を超過しても、警告が発せられるだけで、エラーにはなりません。[-f77=misc]
- 負のシフトカウントに対し、警告が発せられるだけでエラーにはなりません。
- 現在のディレクトリに配置された INCLUDE ファイルと -I オプションで指定された INCLUDE ファイルを検索します。[-f77=misc]
- 連続的な .NOT. 演算子 (.NOT..NOT..NOT.(I.EQ.J) など) を許可します。[-f77=misc]

- **その他**
  - f95 は、通常、標準出力に対する進捗メッセージを発行しません。f77 コンパイラは、進捗メッセージを発行し、コンパイルしているルーチン名を表示します。この規則は、-f77 を指定してコンパイルすると維持されます。
  - f77 コンパイラでコンパイルされたプログラムは、算術例外でトラップされることはなく、自動的に ieee_retrospective を終了に呼び出し、実行中に起こった例外をレポートします。-f77 フラグを使用したコンパイルは、f77 コンパイラのこの動作を模倣します。デフォルトでは、f95 コンパイラは最初の算術例外でトラップされますが、ieee_retrospective は呼び出しません。
  - f77 コンパイラは、高い精度が必要な場合に、REAL*4 定数がコンテキスト中でより高い精度を保持しているように取り扱います。-f77 フラグを使用してコンパイルする場合、f95 コンパイラは、倍精度または 4 倍精度の演算対象に対し、REAL*4 定数がそれぞれ倍精度または 4 倍精度を保持することを許可します。

- DO ループ変数をループ内で再定義できます。[-f77=misc]
- コンパイルするプログラム単位の名前を表示します。[-f77=misc]
- DIMENSION 文で使用される変数の型を DIMENSION 文の後に宣言できます。
  例：
```
SUBROUTINE FOO(ARR,G)
  DIMENSION ARR(G)
  INTEGER G
  RETURN
  END
```

非標準の言語拡張機能の構文および意味についての詳細は、http://docs.sun.com/source/806-4841/index.html の『FORTRAN 77 言語リファレンス』を参照してください。

5.2 非互換性の問題

現行リリースの f95 で、従来の f77 プログラム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を次に示します。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。

- **ソースの書式**
  - -f77 オプションを指定すると、6 文字を超える名前に対し ANSI 警告が発せられます。
- **入出力**
  - f95 は、直接探査ファイルで ENDFILE を許可しません。
  - f95 は、直接探査入出力でレコード番号を指定する '*n* 書式 (例： READ (2 '13) X,Y,Z) を認識しません。
  - f95 は、従来の f77 "R" 書式編集記述子を認識しません。
  - f95 では、CLOSE 文における DISP= 指定子を許可しません。
  - WRITE 文でのビット定数は許可されません。
  - Fortran 95 NAMELIST は、可変長の配列および文字列を許可しません。
  - RECL=1 を使用して直接探査ファイルを開くことは、「ストリーム」ファイルとしては使用できません。代わりに FORMAT='STREAM' を使用してください。
  - Fortran 95 は、不当な入出力指定子をエラーとしてレポートします。f77 では警告のみです。
- **データ型、宣言、および用法**

- 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。
- f95 は、PARAMETER 文での非定数を許可しません。
- CHARACTER 型宣言の初期化子では整数値は使用できません。
- REAL() 組み込み関数は、引数を REAL*4 に変換する代わりに、複素引数の実数部を返します。これにより、引数が DOUBLE、COMPLEX、または COMPLEX*32 の場合に、異なる結果が返されます。
- Fortran 95 は、配列が宣言される前に、境界式の配列要素を許可しません。例 :

```
subroutine s(i1,i2)
integer i1(i2(1):10)
dimension i2(10)
...ERROR:"I2" は関数として使用されているため、形状明示
DIMENSION 属性を付けて宣言してはなりません。

end
```

- **プログラム、サブルーチン、関数、文**
  - 名前の最大長は 127 文字です。
- **コマンド行オプション**
  - f95 は、f77 コンパイラオプション -dbl、-oldstruct、-i2、-i4、および -vax の一部のサブルーチンを認識しません。
- **f95 がサポートしていない f77 ライブラリルーチン**
  - POSIX ライブラリ
  - IOINIT() ライブラリルーチン
  - テープ入出力ルーチン topen、tclose、twrite、tread、trewin、tskipf、tstate
  - start_iostats および end_iostats ライブラリルーチン
  - f77_init() 関数
  - f95 では、同じ名前を持つユーザー独自のルーチンを定義することによってバイパスされる IEEE_RETROSPECTIVE サブルーチンを許可していません。

5.3 f77 でコンパイルしたルーチンとのリンク

- f77 および f95 オブジェクトバイナリを混在させるには、-xlang=f77 オプションを指定して f95 コンパイルとリンクします。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 : f77 オブジェクトファイルで f95 主プログラムをコンパイルします。

```
demo% cat m.f95
CHARACTER*74 :: c = 'テストです。'
   CALL echo1( c )
END
demo% f95 -xlang=f77 m.f95 sub77.o
demo% a.out
 テストです。
demo%
```

- f95 プログラムに対して FORTRAN 77 ライブラリおよび組み込み関数が使用できます。『Fortran ライブラリ・リファレンス』を参照してください。

  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
elapsed: 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。

5.3.1 Fortran 95 組み込み関数

Fortran 95 の標準機能として、FORTRAN 77 にはない組み込み関数がサポートされています。Fortarn 95 の非標準組み込み関数を含むすべての組み込み関数は、『Fortran ライブラリ・リファレンス』に記載されています。

『Fortran ライブラリ・リファレンス』に記載された組み込み関数名をプログラムの関数名として使用する場合は、組み込みではないルーチンを使用するために、f95 の EXTERNAL 文を追加する必要があります。

『Fortran ライブラリ・リファレンス』には、以前の f77 コンパイラ が認識可能な組み込み関数も記載されています。f95 コンパイラは、これらの名前を組み込み関数と同様に認識できます。

-f77=intrinsics を指定してコンパイルすると、認識可能な組み込み関数は f77 コンパイラで知られるものだけに制限され、Fortran 95 組み込み関数は無視されます。

5.4 f95 への移行についてのその他の問題

- floatingpoint.h ヘッダーファイルは、f77_floatingpoint.h を置き換え、次のソースプログラムで使用される必要があります。

  ```
  #include "floatingpoint.h"
  ```

- f77/*filename* 書式のヘッダーファイルの参照は、f77/ ディレクトリパスを削除するように変更する必要があります。
- 非標準の名前付け手法を使用しているプログラム (配列のオーバーインデックス、Cray または Fortran ポインタのオーバーラップによる) の場合は、適切な -xalias フラグを指定してコンパイルするとよいでしょう。3-58 ページの「-xalias[=keywords]」を参照してください。また、『Fortran プログラミングガイド』では、「dusty deck (互換性または保守のために残さざるを得ない)」プログラムの移植についての章で、例を挙げて検討されています。

付録 A

実行時のエラーメッセージ

この付録では、Fortran 95 の実行時入出力ライブラリ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。

A.1 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: cannot access afile
 status =  512
```

A.2 f95 の実行時入出力メッセージ

f95 入出力ライブラリは、実行時にエラーを検出すると、診断メッセージを出力します。Fortran95 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。

表 A-1 に f95 で出力される実行時入出力メッセージの一部の日本語訳を示します。.

表 A-1 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** (format error)
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** (formatted 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)
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)

表 A-1　f95 の実行時入出力メッセージ (続き)

エラー	メッセージ
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	長すぎる記録 (record too long)
1011	切り捨てエラー (truncation failed)
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取り (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 (reentrant I/O)
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)
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)

表 A-1　f95 の実行時入出力メッセージ (続き)

エラー	メッセージ
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')
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)

表 A-1　f95 の実行時入出力メッセージ (続き)

エラー	メッセージ
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)
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)
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)

表 A-1　f95 の実行時入出力メッセージ (続き)

エラー	メッセージ
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)
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')

表 A-1　f95 の実行時入出力メッセージ (続き)

エラー	メッセージ
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 specifier)
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')
1138	RECL 指定子がゼロまたは負数です (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された 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)
1152	デフォルト装置に対する指定子 ACCESS='STREAM' (specifier ACCESS= 'STREAM' for default unit)
1153	デフォルト装置へのストリーム探査 (stream-access to default unit)

表 A-1　f95 の実行時入出力メッセージ (続き)

エラー	メッセージ
1161	装置は REWIND をサポートしません (device does not support REWIND)
1162	BACKSPACE には読み取り権が必要です (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	順番捜査装置または直接探査装置へのストリーム探査 (stream-access to sequential or direct-access unit)
1169	接続されていない装置へのストリーム探査 (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	POS 指定子の不正な値 (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)
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	ストリーム探査装置に対する OPEN 文では RECL 指定子を使用できません (RECL specifier is not allowed in OPEN statement for stream-access unit)

表 A-1　f95 の実行時入出力メッセージ (続き)

エラー	メッセージ
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解放 (deallocating an unassociated pointer)
1183	結合していない割り付け配列の解放 (deallocating an unallocated allocatable array)
1184	ポインタを通して割り付け配列の解放 (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解放 (deallocating an object not allocated by an ALLOCATE statement)
1186	実体の一部の解放 (deallocating a part of an object)
1187	割り付けより大きな実体の解放 (deallocating a larger object than was allocated)
1191	配列組み込み関数に渡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)
1213	接続されていない装置に対する非同期入出力 (asynchronous I/O on unconnected unit)
1214	同期装置に対する非同期入出力 (asynchronous I/O on synchronous unit)
1215	データ編集記述子と入出力リスト項目の型に互換性がありません (a data edit descriptor and I/O list item type are incompatible)
1216	現在の入出力リスト項目はデータ編集記述子と一致しません (current I/O list item doesn't match with any data edit descriptor)
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)

表 A-1　f95 の実行時入出力メッセージ (続き)

エラー	メッセージ
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)
2021	初期化されていないロック (lock not initialized) (OpenMP)
2122	ロック変数の使用におけるデッドロック (deadlock in using lock variable) (OpenMP)
2123	設定されていないロック (lock not set) (OpenMP)

付録B

各リリースにおける機能変更

この付録では、Fortran 95 コンパイラの今回のリリースおよびこれまでのリリースで、追加された機能、変更された機能について説明します。

Fortran 95 コンパイラ、バージョン 8.1 は Sun Studio 10 でリリースされるコンポーネントの 1 つです。

B.1 Sun Studio 10 Fortran リリース

- **AMD-64 プロセッサ向けコンパイル**

  このリリースでは、64 ビット x86 プラットフォームで動作するようにアプリケーションをコンパイルするためのオプションとして、`-xarch=amd64` および `-xtarget=opteron` が導入されています。

- **ビッグエンディアンとリトルエンディアン式プラットフォーム間のファイルの共有**

  新しいコンパイラフラグの `-xfilebyteorder` は、プラットフォームにまたがるバイナリ入出力ファイルのサポートを提供します。

- **Solaris x86 プラットフォームでの OpenMP のサポート**

  このリリースの Sun Studio では、Solaris SPARC プラットフォームばかりでなく、Solaris x86 プラットフォームでも、共有メモリー並列化のための OpenMP API を利用できます。両方のプラットフォームで、同じ内容の機能を利用できます。

- **OpenMP オプション `-openmp=stubs` のサポート廃止**

  ユーザーの便宜のため、OpenMP のスタブライブラリは提供されます。OpenMP ライブラリ関数を呼び出すだけで、OpenMP プログラムを無視する OpenMP プログラムをコンパイルする場合は、-openmp オプションを付けてプログラムをコン

パイルし、オブジェクトファイルを libompstubs.a ライブラリとリンクします。以下に例を示します。

```
% f95 omp_ignore.c -lompstubs
```

libompstubs.a と OpenMP 実行時ライブラリの libmtsk.so 両方とのリンクはサポートされていません。両方とリンクすると、予期しない動作になることがあります。

B.2 Sun Studio 9 Fortran リリース

- **x86 Solaris プラットフォーム向け Fortran 95 リリース**

  このリリースの Sun Studio では、x86 プラットフォーム版 Solaris で Fortran 95 コンパイラが使用できるようになっています。x86 Solaris プラットフォームで実行可能なファイルを生成するには、-xtarget 値として generic、native、386、486、pentium, pentium_pro、pentium3、pentium4 のいずれかをコンパイル時に指定します。x86 プラットフォームでのデフォルトは -xtarget=generic です。

  x86 プラットフォームの場合、次の f95 機能はまだ実装されていません。使用できるのは、SPARC プラットフォーム上のみです。

  - 区間演算 (コンパイラオプション -xia および -xinterval)
  - Quad (128 ビット) 演算 (REAL*16 など)
  - IEEE 組み込みモジュール の IEEE_EXCEPTIONS、IEEE_ARITHMETIC、および IEEE_FEATURES
  - sun_io_handler モジュール
  - -autopar、-parallel、-explitipar、-openmp などの並列化オプション

  次の f95 コマンド行オプションは、x86 プラットフォームでのみ使用できます。SPARC プラットフォーム では使用できません。
  -fprecision、-fstore、および -nofstore

  次の f95 コマンド行オプションは、SPARC プラットフォームでのみ使用できます。x86 プラットフォーム では使用できません。
  -xcode、-xmemalign、-xprefetch、-xcheck、-xia、-xinterval、-xipo、-xjobs、-xlang、-xlinkopt、-xloopinfo、-xpagesize、-xprofile_ircache、-xreduction、-xvector、-depend、-openmp、-parallel、-autopar、-explicitpar、-vpara、-XlistMP
  また、x86 プラットフォーム版の場合、-fast は -nofstore を追加します。

- **実行時のパフォーマンスの向上**

  今回のリリースでは、多くのアプリケーションの実行時のパフォーマンスが向上するとみられます。最良の結果を得るには、最適化レベルを高くして (-x04 または -x05) コンパイルしてください。これらのレベルでは、コンパイラが内部手続きや、形状引き継ぎ、割り付け、あるいはポインタ引数を持つ手続きをインライン化することができます。

■ Fortran 2003 のコマンド行組み込み関数

Fortran 2003 規格草案では、コマンド行引数および環境変数を処理するための新しい組み込み関数が紹介されています。今回のリリースの f95 コンパイラには、これらの組み込み関数が実装されています。新しい組み込み関数は以下のとおりです。

- GET_COMMAND(*command, length, status*)

  command でプログラムを呼び出すコマンド行全体を返します。

- GET_COMMAND_ARGUMENT(*number, value, length, status*)

  value でコマンド行引数を返します。

- GET_ENVIRONMENT_VARIABLE(*name, value, length, status, trim_name*)

  環境変数の値を返します。

■ f95 コマンド行オプションの追加および変更

このリリースの f95 では、次のコマンド行オプションが新しく追加されています。詳細は 第 3 章を参照してください。

- -xipo_archive={ none | readonly | writeback }

  クロスファイル最適化でアーカイブ (.a) ライブラリを取り込むことができます。(SPARC のみ)

- -xprefetch_auto_type=[no%]indirect_array_access

  間接アクセスされるデータ配列に対して間接先読み命令を生成します。(SPARC のみ)

- -xprofile_pathmap=*collect_prefix*:*use_prefix*

  プロファイルデータファイルのパスマッピングを設定します。以前に -xprofile=collect を使ってコンパイルしたときに使用ディレクトリとは異なるディレクトリにプロファイリングする場合は、-xprofile_pathmap オプションを -xprofile=use オプションと併用してください。

このリリースの f95 では、次のコマンド行オプションのデフォルト値が変更されています。

- -xprefetch のデフォルト値は -xprefetch=no%auto,explicit です。
- -xmemalign のデフォルト値は -xmemalign=8i です。ただし、-xarch=v9 オプションのいずれかを付けたコンパイルでは、デフォルト値は -xmemalign=8f になります。
- -xarch=v9 オプションのいずれかを付けたコンパイルでの -xcode のデフォルト値は abs44 になります。

  以前のリリースのコンパイラで使用されていたデフォルト値でコンパイルにするには、次のオプションを明示的に指定します。

  32 ビットコンパイルの場合 : -xarch=v8 -xmemalign=4s -xprefetch=no
  64 ビットコンパイルの場合 : -xcode=abs64 -xprefetch=no

■ **デフォルトの SPARC アーキテクチャを V8PLUS に変更**

デフォルトの SPARC アーキテクチャは V7 でなくなりました。この Sun Studio 9 リリースでは、`-xarch=v7` のサポートに制限があります。新しいデフォルトは V8PLUS (UltraSPARC) です。`-xarch=v8` 以上をサポートしているのは Solaris 8 だけであるため、`-xarch=v7` によるコンパイルは、`-xarch=v8` として扱われます。

SPARC V8 システム (SPARCStation 10 など) に配備するには、明示的に `-xarch=v8` を使ってコンパイルします。提供のシステムライブラリは、SPARC V8 アーキテクチャで動作します。

SPARC V7 システム (SPARCStation 1 など) に配備するには、明示的に `-xarch=v7` を使ってコンパイルします。提供のシステムライブラリは、SPARC V8 命令セットを利用します。この Sun Studio 9 リリースでは、SPARC V7 アーキテクチャをサポートするのは、Solaris 8 だけです。SPARC V8 命令が検出されると、OS はソフトウェアでその命令を解釈します。このためプログラムは実行されますが、パフォーマンスは低下します。

■ **OpenMP：最大スレッド数を増加**

`OMP_NUM_THREADS` およびマルチタスクライブラリの最大スレッド数が 128 から 256 に増加しました。

■ **OpenMP：変数の自動スコープ**

このリリースの Fortran 95 コンパイラに実装されている、共有メモリー並列プログラミング用の OpenMP API には、並列領域における変数の自動スコープ機能があります。詳細は、『OpenMP API ユーザーズガイド』を参照してください。(このリリースでは、OpenMP は SPARC プラットフォームにのみ実装されます。)

B.3 Sun Studio 8 Fortran リリース

■ **`-openmp`** オプションの拡張

`-openmp` オプションフラグは、OpenMP プログラムのデバッグが容易にできるように強化されました。OpenMP アプリケーションのデバッグに dbx を使用するには、次の指定をしてコンパイルします。

```
-openmp=noopt -g
```

その後 dbx を使用することによって、並列化領域内のブレークポイントで停止し、変数の中身を表示できます。3-44 ページの「`-openmp[=`{`parallel`|`noopt`|`none`}`]`」を参照してください。

■ **マルチプロセスのコンパイル**

`-xipo` とともに `-xjobs=`*n* を指定すると、内部手続き間最適化が最大 *n* 個のコード生成インスタンスを起動して、コマンド行に列挙されたファイルをコンパイルします。このオプションによって、マルチ CPU を持つマシン上の大きなアプリケーションを構築するための時間が大幅に削減されます。3-83 ページの「`-xjobs=n`」を参照してください。

■ **`PRAGMA ASSUME` を使った表明**

`ASSUME` プラグマはこのコンパイラに今回新しく追加された機能です。このプラグマは、手続き内のある個所において真であることをプログラマが知っている条件について、コンパイラにヒントを与えます。このことによって、コンパイラのコードの最適化機能がさらに向上します。また、プログラマはこの表明を使って、実行時にプログラムの妥当性をチェックできます。詳細は、2-14 ページの 2.3.1.9 節「`ASSUME` 指令」、および 3-66 ページの「`-xassume_control[=keywords]`」を参照してください。

■ **Fortran 2000 機能の追加**

Fortran 2000 の規格草稿に記述されている以下の機能が、Fortran 95 コンパイラの今回のリリースに導入されました。これらは第 4 章で説明されています。

- 例外処理と IEEE 演算

  新しい組み込みモジュールの EEE_ARITHMETIC と IEEE_FEATURES によって、Fortran 言語での例外処理と IEEE 演算がサポートされます。4-15 ページの 4.6.2 節「IEEE 浮動小数点の例外処理」を参照してください。

- C との相互運用性

  Fortran のための新しい規格草稿では、C 言語手続きを参照する方法、および反対に C 関数から Fortran 副プログラムを参照できるように指定する方法を定めています。また、外部 C 変数とリンクする大域変数を宣言する方法も定めています。4-15 ページの 4.6.1 節「C 関数との相互運用性」を参照してください。

- `PROTECTED` 属性

  Fortran 95 コンパイラでは、新たに Fortran 2000 の `PROTECTED` 属性が使えるようになりました。`PROTECTED` はモジュール要素の使用に制限を設けます。`PROTECTED` 属性を持つオブジェクトは、それ自身が宣言されるモジュール内でのみ定義可能です。4-16 ページの 4.6.4 節「`PROTECTED` 属性」

- `ASYNCHRONOUS` 入出力指定子

  コンパイラは入出力文中の `ASYNCHRONOUS` 指定子を認識します。

  ```
  ASYNCHRONOUS=['YES' | 'NO']
  ```

  この構文は Fortran 2000 規格草稿で提案されているものです。4-16 ページの 4.6.5 節「Fortran 2003 非同期入出力」を参照してください。

■ **従来の f77 との互換性の強化**

多数の機能の追加によって、Fortran 95 コンパイラでは従来の FORTRAN 77 コンパイラである f77 との互換性が向上しました。その機能とは、可変フォーマット式 (VFE)、long 識別子、コンパイルオプションの -arg=local と -vax などです。第 3 章および第 4 章を参照してください。

■ **入出力エラーハンドラ**

2 つの新機能によって、ユーザーは独自に論理ユニットの書式付き入力のエラー処理ルーチンを指定できます。このルーチンについては 4-19 ページの 4.7.1 節「入出力エラー処理ルーチン」と『Fortran ライブラリ・リファレンス』のマニュアルページを参照してください。

■ **符号なし整数**

今回のリリースによって Fortran 95 コンパイラでは言語が拡張され、新しいデータ型である UNSIGNED が使用可能になりました。4-13 ページの 4.5 節「符号なし整数」を参照してください。

■ **優先スタックサイズ、ヒープページサイズの設定**

新しいコマンド行オプション、-xpagesize を使用すれば、実行プログラムがプログラム開始時に優先スタックサイズおよびヒープページサイズを設定できるようになります。3-89 ページの「-xpagesize=size」を参照してください。

■ **プロファイル処理の高速化と機能強化**

今回のリリースで、新しいコマンド行オプション、-xprofile_ircache= *path* が導入され、プロファイルフィードバック中のコンパイルフェーズ "use" がスピードアップされました。3-96 ページの「-xprofile_ircache[=path]」を参照してください。詳細は、3-96 ページの「-xprofile_pathmap= collect_prefix : use_prefix」を参照してください。

■ **既知のライブラリの拡張**

-xknown_lib オプションが強化され、Basic Linear Algebra ライブラリ、BLAS より多くのルーチンが取り入れられました。3-83 ページの「-xknown_lib= library_list」を参照してください。

■ **リンク時の最適化**

新たに追加された -xlinkopt フラグを使ってコンパイル、リンクすると、ポストオプティマイザが起動され、生成されたバイナリコードに対し、リンク時にパフォーマンス面で各種の高度な最適化が施されます。3-86 ページの「-xlinkopt[={1|2|0}]」を参照してください。

■ **局所変数の初期化**

-xcheck オプションフラグが強化され、局所変数の特別な初期化が可能になりました。-xcheck=init_local を指定してコンパイルすると、プログラムによる割り当て前に使用された場合に算術例外を引き起こす可能性のある局所変数を、ある値に初期化できます。3-68 ページの「-xcheck=keyword」を参照してください。

B.4 Sun ONE Studio 7, Compiler Collection (Forte Developer 7) リリース

- **Fortran 95 コンパイラに組み込まれた FORTRAN 77 の機能**

  このリリースでは、`f77` コンパイラは `f95` コンパイラの追加機能に置き換わりました。`f77` のコマンドは、`f95` を呼び出すスクリプトです。

> **コマンド**:
> `f77` **オプション ファイル ライブラリ**
> **は、次の** `f95` **コンパイラの呼び出しになります**:
> `f95 -f77=%all -ftrap=%none` **オプション ファイル** `-lf77compat` **ライブラリ**

  FORTRAN 77 の互換性および非互換性についての詳細は、第 5 章 を参照してください。

- **FORTRAN 77 互換性モード**

  一般的には、Fortran 95 とは互換性のない FORTRAN 77 構造構文および規則を、コンパイラが受け付けるようにするためのさまざまな互換性機能を、新しい `-f77` フラグを使用することによって選択できます。詳細は、3-23 ページの「`-f77`[= list]」および第 5 章を参照してください。

- **非標準別名付けを採用している「Dusty Deck (互換性または保守のために残さざるを得ない) 」プログラムのコンパイル**

  `f95` コンパイラは、コンパイルするプログラムが、副プログラムの呼び出し、大域変数、ポインタ、オーバーインデックスを使用した変数の別名付けに関して、Fortran 95 標準規則に従っていると仮定する必要があります。従来の多くのプログラムは、Fortran 言語の旧バージョンの欠点を避けるために、意図的に別名付け手法を使用しています。`-xalias` フラグを使用すると、コンパイラにプログラムの標準からのずれと、予想される別名付けの問題を知らせることができます。場合によっては、適切な `-xalias` サブオプションを指定したときにのみ、正しいコードが生成されることもあります。厳格に標準に準拠しているプログラムの場合は、コンパイラに別名付けを考慮しないように助言すると、パフォーマンスが向上する場合もあります。3-58 ページの「`-xalias`[=keywords]」および『Fortran プログラミングガイド』の移植に関する章を参照してください。

- **MODULE 機能の向上**
  - 新しいフラグ `-use=`*list* を使用すると、1 つ以上の暗黙的な `USE` 文が各サブプログラムに挿入されます。3-54 ページの「`-use=list`」を参照してください。
  - 新しいフラグ `-moddir=`*path* によって、コンパイルした `MODULE` サブプログラム (`.mod` ファイル) をどこに書き込むかをコントロールできます。3-39 ページの「`-moddir=path`」を参照してください。新しい環境変数 `MODDIR` によっても、どこに .mod ファイルを記述するかをコントロールできます。

- -M*path* フラグは、ディレクトリパス、アーカイブ (.a) ファイル、または (.mod) ファイルを、MODULE 副プログラムを検索するために受け取ることができます。コンパイラは、ファイルの内容を検査してファイルの型を決めます。実際のファイルの拡張子は無視されます。3-38 ページの「-Mpath」を参照してください。
- モジュールを検索する際に、コンパイラは、モジュールファイルが書き込まれるディレクトリを最初に探します。

詳細は、4-23 ページの 4.9 節「モジュールファイル」を参照してください。

■ -Xlist を使用した大域的なプログラム検査の向上

f95 コンパイラのこのリリースでは、大域的なプログラム検査のために、-Xlist フラグによる多くの新しい検査機能が加わりました。新しい -XlistMP サブオプションは、静的プログラム解析の新しいドメイン (OpenMP 並列化指令の検証) を開きます。詳細は、3-56 ページの「-Xlist[x]」、Forte Developer『OpenMP API ユーザーズガイド』、『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。

■ -xknown_lib=library による既知のライブラリの識別

新しいオプション -xknown_lib=*library* は、既知のライブラリへの参照を組み込み関数として扱い、ユーザー定義のバージョンを無視するように、コンパイラに指示します。これによって、コンパイラは、ライブラリに関する情報に基づき、ライブラリの呼び出しを最適化できます。このリリースでは、Fortaran 95 標準組み込み関数に対する明示的な EXTERNAL 宣言とこれらのルーチンのユーザー定義バージョンを無視するため、既知のライブラリ名は、サンのパフォーマンスライブラリにある BLAS ルーチンのサブセット blas および intrinsics に限定されます。3-83 ページの「-xknown_lib=library_list」を参照してください。

■ インタフェース仮引数型の無視

新しい指令 !$PRAGMA IGNORE_TKR {*list_of_variables*} は、特定の呼び出しを解釈するときに、一般的な手続きのインタフェースで使用される仮引数名の型、種類、ランクを、指定されたものに関して無視するように、コンパイラに指示します。この指令を使用することによって、引数の型、種類、ランクに基づいてライブラリルーチンを呼び出すラッパーの一般的なインタフェースの記述を単純化できます。詳細は、2-10 ページの 2.3.1.2 節「IGNORE_TKR 指令」を参照してください。

■ -C 配列検査の向上

この f95 コンパイラのリリースでは、-C による実行時の配列添字範囲の検査が向上し、配列の準拠検査もできるようになりました。配列のセクションが準拠していない配列構文が実行されるとエラーになります。3-15 ページの「-C」を参照してください。

■ **Fortran 2000 の機能の導入**

この f95 リリースでは、次の Fortran 標準に提案されている新しい書式付き入出力機能が実装されています。これらは、DECIMAL=、ROUND=、および IOMSG= 指定子で、OPEN、READ、WRITE、PRINT、および INQUIRE 文で使用されます。また、DP、DC、RP、および RC 編集編記述子も実装されています。詳細は、4-18 ページの 4.6.9 節「Fortran 2003 の書式付き入出力機能」を参照してください。

■ **書式付き入出力の丸め**

新しいオプションフラグ -iorounding は、書式付き入出力のデフォルトの丸めモードを設定します。モード (プロセッサ定義または互換性がある) は、Fortran 2000 の機能として実装された ROUND= 指定子に対応して決まります。3-35 ページの「-iorounding[={compatible|processor-defined}]」を参照してください。

■ **旧オプションの削除**

次のフラグは f95 コマンド行から削除されました。

```
-db     -dbl
```

次の f77 コンパイラフラグは、f95 コンパイラに実装されず、旧オプションとして扱われます。

```
-arg=local -i2 -i4 -misalign -oldldo -r8 -vax-xl -xvpara
-xtypemap=integer:mixed
```

■ **スタックオーバーフローの検査**

新しい -xcheck=stkovf フラグを指定してコンパイルすると、エントリのスタックオーバーフロー状態に対する実行時の検査が副プログラムに加わります。スタックオーバーフローが検出されると、SIGSEGV セグメントフォルトが発生します。スタックオーバーフローは、スタックに大きな配列が割り当てられるマルチスレッドアプリケーションで、近傍のスレッドスタックのデータを警告なしに破壊する可能性があります。スタックオーバーフローの可能性がある場合は、-xcheck=stkovf を使用してすべてのルーチンをコンパイルします。3-68 ページの「-xcheck=keyword」を参照してください。

■ **新しいデフォルトのスレッドスタックのサイズ**

このリリースでは、デフォルトのスレーブスレッドのスタックサイズが、SPARC V8 プラットフォームでは 4M バイトに、SPARC V9 プラットフォームでは 8M バイトに増加されました。詳細は、『Fortran プログラミングガイド』の並列化の章で、スタックおよびスタックサイズに関する内容を参照してください。

■ **内部手続きの最適化の向上**

-xipo=1 を使用すると、コンパイラはすべてのソースファイルに対してインライン化を実行しません。このリリースでは、キャッシュのパフォーマンス向上を目的として、手続き間の別名付けの解析を向上し、記憶域割り当てとレイアウトを最適化するために、-xipo=2 が追加されました。3-80 ページの「-xipo[={0|1|2}]」を参照してください。

- **-xprefetch_level=n による先読み命令のコントロール**

  新しいフラグ `-xprefetch_level=`*n* を使用すると、`-xprefetch=auto` による先読み命令の自動挿入をコントロールできます。使用の際は、`-xO3` 以上の最適化レベルを指定し、先読み命令をサポートするターゲットプラットフォーム (`-xarch` プラットフォーム `v8plus`、`v8plusa`、`v8plusb`、`v9`、`v9a`、`v9b`、`generic64`、または `native64`) が必要です。3-94 ページの「`-xprefetch_level={1|2|3}`」を参照してください。

Forte Developer 7 以前の機能の履歴は、Web サイト `http://docs.sun.com` にある以前のリリースのマニュアルを参照してください。

付録C

従来の -xtarget プラットフォームの展開

この付録では、古くてあまり使われなくなった -xtarget オプションのプラットフォームシステムの名前とその展開について説明します。これは参考として紹介するものです。UltraSPARC プラットフォームで使う値は、第 3 章の -xtarget オプションの説明に基づいて指定されます。ここで取り上げているシステムプラットフォームには、新しい Solaris オペレーティング環境ではサポートされなくなったものもあります。

次の表に示すように、-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。

たとえば次のように指定すると、

```
-xtarget=sun4/15
```

は、次の意味です。

```
-xarch=v8a -xchip=micro -xcache=2/16/1
```

表 C-1 従来の -xtarget の展開

-xtarget=	-xarch	-xchip	-xcache
cs6400	v8	super	16/32/4:2048/64/1
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

表 C-1　従来の -xtarget の展開 (続き)

-xtarget=	-xarch	-xchip	-xcache
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
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

表 C-1　従来の -xtarget の展開 (続き)

-xtarget=	-xarch	-xchip	-xcache
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
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

表 C-1　従来の -xtarget の展開 (続き)

-xtarget=	-xarch	-xchip	-xcache
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
sun4/670	v7	old	64/32/1
sun4/690	v7	old	64/32/1
sun4/75	v7	old	64/32/1

付録 D

Fortran 指令の要約

この付録では、f95 Fortran コンパイラで認識可能な指令について示します。

- 一般的な Fortran 指令
- 特殊な Fortran 指令
- Fortran 95 の OpenMP 指令
- Sun の並列化指令
- Cray の並列化指令

D.1 一般的な Fortran 指令

f95 で使用可能な一般的な指令については、第 2 章で説明します。

表 D-1 一般的な Fortran 指令の要約

書式	
C$PRAGMA *keyword* (*a* [, *a*] ...) [, *keyword* (*a* [, *a*] ...)] ,...	
C$PRAGMA SUN *keyword* (*a* [, *a*] ...) [, *keyword* (*a* [, *a*] ...)] ,...	
C$PRAGMA SPARC *keyword* (*a* [, *a*] ...) [, *keyword* (*a* [, *a*] ...)] ,...	
1 桁目に指定する注釈指示子は、c、C、!、または * です (ここでの例では注釈指示子として C を使用していますが、f95 の自由形式では ! を使わなければなりません)。	
C 指令	C$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。
IGNORE_TKR 指令	C$PRAGMA IGNORE_TKR {*name* {, *name*} ...} コンパイラは、特定の呼び出しを解釈するとき、一般的な手続きのインタフェースで表示される仮引数名の型、種類、ランクを無視します。

表 D-1　一般的な Fortran 指令の要約 (続き)

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* を優先順位の低いシンボル (weak symbol) または *name2* 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$PRAGMA SUN PIPELOOP=*n* ループの *n* 離れた反復間の依存性を宣言します。
PREFETCH 指令	C$PRAGMA SPARC_PREFETCH_READ_ONCE (*name*) C$PRAGMA SPARC_PREFETCH_READ_MANY (*name*) C$PRAGMA SPARC_PREFETCH_WRITE_ONCE (*name*) C$PRAGMA SPARC_PREFETCH_WRITE_MANY (*name*) 名前の参照のために、先読み命令を生成するようにコンパイラに要求します。-xprefetch オプションを指定する必要があります。
ASSUME *Directives*	C$PRAGMA [BEGIN] ASSUME (*expression* [, *probability*]) C$PRAGMA END ASSUME プログラム内の特定の個所において、コンパイラが真であると想定できる条件について表明を行います。

D.2 特殊な Fortran 95 指令

次の指令は、f95 でのみ使用できます。詳細は、4-22 ページの 4.8.2 節「FIXED 指令と FREE 指令」を参照してください。

表 D-2 特殊な Fortran 95 指令

書式	!DIR$ **指令** : 最初の行 !DIR$& ...: 継続行 固定書式の場合、C は CDIR$ *directive...* のように指令指示子としても受け取られます。行は第 1 桁から始める必要があります。自由書式の場合は、行の前に空白がある場合があります。
FIXED/FREE 指令	!DIR$ FREE !DIR$ FIXED 指令の後に続くソース行の書式を指定します。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。

D.3 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。-openmp コンパイラフラグは、これらの指令を有効にします (3-44 ページの「-openmp[={parallel|noopt|none}]」を参照)。

詳細は、『OpenMP API ユーザーズガイド』を参照してください。

D.4 Sun の並列化指令

Fortran 95 で推奨する並列化モデルは、OpenMP による並列化です。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 D-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 [*qualifiers*] 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) 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。

D.5 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 D-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。修飾子は、次のとおりです。 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 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 と等価です。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 D-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 以降のループの並列化を無効にします。

索引

G

I

L

M

N

O

P

R

S

か

き

く

け

こ

さ

し

す

せ

そ

た

ち

て

へ

ほ

ま

む

め

も

ゆ