Oracle® Solaris Studio 12.2: Fortran
ユーザーズガイド

Part No: 821–2494
2010 年 9 月

ORACLE®

101019@24999

目次

はじめに

『Oracle Solaris Fortran ユーザーズガイド』では、Oracle Solaris Studio Fortran コンパイラ **f95** の環境と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 に関する実用的な知識を持ち、Oracle Solaris Studio Fortran コンパイラの効率的な使用法を学ぼうとしている、科学者、技術者、プログラマを対象に書かれています。また、Solaris オペレーティング環境や UNIX® の一般的な知識を持つ読者を対象としています。

入出力、アプリケーション開発、ライブラリの作成とその使用、プログラム解析、移植、最適化、並列化などの、Oracle Solaris オペレーティング環境での Fortran プログラミングについては、『Fortran プログラミングガイド』を参照してください。

サポートされるプラットフォーム

この Oracle Solaris Studio のリリースは、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ーを使用するシステムをサポートしています。ご使用の Oracle Solaris オペレーティングシステムのバージョンに対するシステムのサポート状況は、ハードウェア互換性リスト `http://www.sun.com/bigadmin/hcl` をご参照ください。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 関連の用語は次のものを指します。

- 「x86」は、64 ビットおよび 32 ビットの x86 互換製品を指します。
- 「x64」は、AMD 64 または EM64T システムで、特定の 64 ビット情報を指します。
- 「32 ビット x86」は、x86 ベースシステムで特定の 32 ビット情報を指します。

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。

Solaris Studio マニュアルへのアクセス方法

マニュアルには、次の場所からアクセスできます。

- マニュアルは、次に示すマニュアル索引のページからアクセスできます。`http://www.oracle.com/technetwork/server-storage/solarisstudio/documentation`。
- IDE、パフォーマンスアナライザ、dbxtool、および DLight の全コンポーネントのオンラインヘルプは、これらのツール内の「ヘルプ」メニューだけでなく、F1 キー、および多くのウィンドウやダイアログにある「ヘルプ」ボタンを使用してアクセスできます。

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次の表に示す場所から参照することができます。

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	HTML 形式。docs.sun.com にある Oracle Solaris Studio 12.2 Collection - Japanese から選択
『Oracle Solaris Studio 12.2 リリースの新機能』(以前はコンポーネントの README ファイル)	HTML 形式。docs.sun.com にある Oracle Solaris Studio 12.2 Collection - Japanese から選択
マニュアルページ	`man` コマンドを使用して Oracle Solaris 端末に表示されます。
オンラインヘルプ	HTML 形式。IDE、dbxtool、DLight、およびパフォーマンスアナライザの「ヘルプ」メニュー、「ヘルプ」ボタン、および F1 キーを使用して表示。
リリースノート	HTML 形式。docs.sun.com にある Oracle Solaris Studio 12.2 Collection - Japanese から選択

関連するサードパーティの Web サイトリファレンス

このマニュアルには、詳細な関連情報を提供するサードパーティの URL が記載されています。

注 – このマニュアルで紹介する Oracle 以外の Web サイトが使用可能かどうかについては、Oracle は責任を負いません。このようなサイトやリソース上、またはこれらを経由して利用できるコンテンツ、広告、製品、またはその他の資料についても、Oracle は保証しておらず、法的責任を負いません。また、このようなサイトやリソースから直接あるいは経由することで利用できるコンテンツ、商品、サービスの使用または依存が直接のあるいは関連する要因となり実際に発生した、あるいは発生するとされる損害や損失についても、Oracle は一切の法的責任を負いません。

開発者向けのリソース

http://www.oracle.com/technetwork/server-storage/solarisstudio を参照して、次の頻繁に更新されるリソースを確認してください。

- リソースは頻繁に更新されます。
- ソフトウェアのマニュアル、およびソフトウェアとともにインストールされる一連のマニュアル
- Oracle Solaris Studio のツールを使用して、順を追って開発タスク全体を説明するチュートリアル
- サポートレベルに関する情報
- http://forums.sun.com/category.jspa?categoryID=113 のユーザーフォーラム

表記上の規則

このマニュアルでは、次のような字体や記号を特別な意味を持つものとして使用します。

表 P–1 表記上の規則

字体または記号	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、コード例を示します。	.login ファイルを編集します。 ls -a を使用してすべてのファイルを表示します。 system%
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示します。	system% **su** password:
AaBbCc123	変数を示します。実際に使用する特定の名前または値で置き換えます。	ファイルを削除するには、rm *filename* と入力します。

表 P–1 表記上の規則 (続き)

字体または記号	意味	例
『 』	参照する書名を示します。	『コードマネージャ・ユーザーズガイド』を参照してください。
「 」	参照する章、節、ボタンやメニュー名、強調する単語を示します。	第 5 章「衝突の回避」を参照してください。 この操作ができるのは、「スーパーユーザー」だけです。
\	枠で囲まれたコード例で、テキストがページ行幅を超える場合に、継続を示します。	`sun% grep '^#define \` `XV_VERSION_STRING'`

コード例は次のように表示されます。

- C シェル

  machine_name% **command y|n** [*filename*]

- C シェルのスーパーユーザー

  machine_name# **command y|n** [*filename*]

- Bourne シェルおよび Korn シェル

  $ **command y|n** [*filename*]

- Bourne シェルおよび Korn シェルのスーパーユーザー

  # **command y|n** [*filename*]

[ ] は省略可能な項目を示します。上記の例は、*filename* は省略してもよいことを示しています。

| は区切り文字 (セパレータ) です。この文字で分割されている引数のうち 1 つだけを指定します。

キーボードのキー名は英文で、頭文字を大文字で示します (例: Shift キーを押します)。ただし、キーボードによっては Enter キーが Return キーの動作をします。

ダッシュ (-) は 2 つのキーを同時に押すことを示します。たとえば、Ctrl-D は Control キーを押したまま D キーを押すことを意味します。

マニュアル、サポート、およびトレーニング

追加リソースについては、次の Web サイトを参照してください。

- マニュアル (http://docs.sun.com)
- サポート (http://www.oracle.com/us/support/systems/index.html)
- トレーニング (http://education.oracle.com) – 左側のナビゲーションバーで Sun へのリンクをクリックしてください。

ご意見の送付先

マニュアルの品質や使いやすさに関するご意見やご提案をお待ちしています。間違いやその他の改善すべき箇所がありましたら、http://docs.sun.com で「Feedback」をクリックしてお知らせください。ドキュメント名とドキュメントの Part No.、および、可能な場合は章、節、ページ番号を記載してください。返答が必要な場合はお知らせください。

Oracle 技術ネットワーク (http://www.oracle.com/technetwork/index.html) では、Oracle ソフトウェアに関するさまざまなリソースを提供しています。

- 技術上の問題やソリューションについては、ディスカッションフォーラム (http://forums.oracle.com) を参照してください。
- 実践的なステップ・バイ・ステップのチュートリアルについては、Oracle By Example (http://www.oracle.com/technology/obe/start/index.html) を参照してください。
- サンプルコードのダウンロードについては、サンプルコード (http://www.oracle.com/technology/sample_code/index.html) を参照してください。

◆ ◆ ◆ 第 1 章

1 概要

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 Solaris Studio Fortran コンパイラ **f95** は、SPARC、UltraSPARC、および x64/x86 プラットフォーム上の Solaris オペレーティング環境と、x86/x64 プラットフォーム上の Linux 環境で使用可能です。このコンパイラは、公開されている Fortran 言語規格に準拠しています。また、マルチプロセッサ並列化、最適化されたコードコンパイル、C と Fortran 言語の混在のサポートなど、さまざまな拡張機能を提供します。

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 準拠規格

- **f95** は、ISO/IEC 1539-1:1997 Fortran 規格ドキュメントの第 1 部に準拠しています。
- 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 および Linux (x86) プラットフォームでの SPARC および x86 ファミリ (UltraSPARC、SPARC64、AMD64、Pentium Pro、および Xeon Intel®64) プロセッサアーキテクチャーの機能を利用した最適化をサポートしています。
- Solaris Studio コンパイラは、OpenMP 3.0 共有メモリー並列化 API 仕様に準拠しています。詳細は、『OpenMP API ユーザーズガイド』を参照してください。
- このマニュアルでは、「標準」は、前述の規格の各バージョンに準拠していることを意味します。これらの規格の範囲外の機能を「非標準」または「拡張機能」と呼んでいます。

これらの標準は、それぞれの標準を策定する組織によって改訂されることがあります。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、ほかの

バージョンと交換されることがあります。その結果、Solaris Studio 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 コンパイラの機能

Solaris Studio の Fortran コンパイラ **f95** は、次の機能と拡張機能を提供します。

- 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(Solaris プラットフォームでのみ使用可能)
- 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 3.0 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。ベンチマークによると、最適化されたアプリケーションは、最適化していないコードに比べると、はるかに高速に実行できます。
- 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。
- 64 ビット対応の Solaris および Linux 環境のサポート
- **%VAL** を使用した値による呼び出し
- FORTRAN 77 と Fortran 95/Fortran 2003 プログラム、およびオブジェクトバイナリの間の互換性
- 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の詳細は、付録 B を参照してください。

このリリースのコンパイラとツールの新機能と拡張機能、および既知の問題、解決策、および制限事項の最新情報については、『Oracle Solaris Studio 12.2 リリースの新機能』ガイドも参照してください。『新機能』ガイドには、このリリースのマニュアルの索引 (http://www.oracle.com/technetwork/server-storage/solarisstudio/documentation) からアクセスできます。

1.3 そのほかの Fortran ユーティリティー

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ーを利用できます。

- **Solaris 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やマルチスレッドアプリケーションの強力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 ユーティリティーは、1 桁目に Fortran のキャリッジ制御文字が入っているファイルを印刷するための Fortran 出力フィルタです。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より書式化されたファイルに変換するときに、**asa** を使用します。**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ているモジュールの名前を表示するユーティリティー。**fdumpmod**(1) を参照してください。
- **fpp** - Fortran ソースコードプリプロセッサ。**fpp**(1) を参照してください。
- **fsplit** このユーティリティーは、複数のルーチンが含まれる Fortran ファイルを複数のファイルに分割して、各ファイルにルーチン 1 つ設定します。**fsplit**(1) を参照してください。

1.4 デバッグユーティリティー

次のデバッグ用ユーティリティーを利用することができます。

- **-Xlist** - 引数、COMMON ブロックなどの整合性を複数のルーチンについてチェックするコンパイラオプション。(Solaris プラットフォームのみ)
- **Solaris Studio dbx** - 安定した機能豊富な実行時および静的デバッガ。パフォーマンスデータコレクタを含んでいます。

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

f95 コンパイラでは、コンパイラフラグである **-xia** および **-xinterval** を提供します。これによって、コンパイラは新しい言語拡張を認識し、適切なコードを生成して、区間演算計算を実行します。詳細は、『Fortran 95 区画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

オンラインのマニュアル (**man**) ページで、コマンド、関数、サブルーチン、またはそれらの情報に関する説明を簡単に参照できます。Solaris Studio のマニュアルページにアクセスするには、**MANPATH** 環境変数を、インストールされている Solaris Studio の **man** ディレクトリへのパスに設定する必要があります。

マニュアルページを表示するには、次のように入力してください。

```
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。たとえば **ieee_flags**(3M) など、ほかのセクションを表示するには、**man** コマンドに **-s** オプションを使用します。

```
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。

次に 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)	Solaris Studio 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_flags(3M)	浮動小数点の例外ビットを検証、設定、解除する
ieee_handler(3M)	浮動小数点例外を処理する
matherr(3M)	数学ライブラリエラー処理ルーチン
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 コマンド行ヘルプ

次のようにコンパイラの **-help** オプションを起動すると、**f95** のコマンド行オプションの要約を表示できます。

```
%f95 -help=flags
Items within [ ] are optional. Items within < > are variable parameters.
Bar | indicates choice of literal values.
-someoption[={yes|no}] implies -someoption is equivalent to -someoption=yes

_____________________________________________________________________
-a                  Collect data for tcov basic block profiling
-aligncommon[=<a>] Align common block elements to the specified
                    boundary requirement; <a>={1|2|4|8|16}
-ansi               Report non-ANSI extensions.
-autopar            Enable automatic loop parallelization
-Bdynamic           Allow dynamic linking
-Bstatic            Require static linking
-C                  Enable runtime subscript range checking
-c                  Compile only; produce .o files but suppress
                    linking
...etc.
```

◆ ◆ ◆ 第 2 章

Solaris Studio Fortran の使用

この章では、Fortran 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。

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**)。
- ソースファイルを次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。あと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コンパイルします (**-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コンパイルします (**-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**、**.F95**、**.f03**、または**.F03**という接尾辞の付いたFortranのソースファイルを作成します(表 表2–1 を参照)。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起動します。

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、デフォルトで実行可能プログラムを**a.out**というファイルにリンクします。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。しかし、すべてのコンパイルで同じファイルを使用するのは不都合な場合があります。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%
```

前述の例で、**-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。

この章の次の項では、 **f95** コマンドで使用する表記法、コンパイラのソース行指令、これらのコンパイラの使用上の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

シェルプロンプトで起動される単純なコンパイラコマンドの構文は次のとおりです。

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 つ以上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「自由形式」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、実行時デバッグ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 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入れます。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、28 ページの「2.2.2 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の流れ

前述の例では、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。

コンパイルが失敗すると、エラーごとにメッセージが返されます。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接尾辞によって、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。次の表に示されていない接尾辞の付いたファイル名、および拡張子のないファイル名は、リンカーに渡されます。

表 2–1 Fortran コンパイラが認識可能なファイル名の接尾辞

接尾辞	言語	処理
.f	Fortran 77 または Fortran 95 固定形式	Fortran ソースファイルをコンパイルし、オブジェクトファイル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接尾辞 **.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 固定形式	**.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って、選択されたルーチンのインライン呼び出しを展開します(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、159 ページの「4.1 ソース言語の機能」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。コンパイラでは、デフォルトで **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、**fpp** よりも **cpp** を使用することをお勧めします。

2.2.5 コンパイルとリンクの分離

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が生成されますが、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、別の手順でほかのオブジェクトファイルとリンクします。

```
demo% f95 -c file1.f                          (Make new object file)
demo% f95 -o prgrm file1.o file2.o file3.o        (Make executable file)
```

リンク時には (2 行目)、完全なプログラムを作成するのに必要なすべてのオブジェクトファイルを必ず 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のステップで行う場合は、整合性のあるコンパイルとリンクのオプションを選択することが重要です。オプションによっては、プログラムの一部をコンパイルするときに使用したら、リンクするときにも同じオプションを使用する必要があります。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数多くあります。

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: **-fast** を使用して **sbr.f** をコンパイルし、C ルーチンをコンパイルしてから、別のステップでリンクします。

```
demo% f95 -c -fast sbr.f
demo% cc -c -fast simm.c
demo% f95 -fast sbr.o simm.o          link step; passes  -fast  to the linker
```

2.2.7 認識されないコマンド行引数

コンパイラで認識されないコマンド行の引数はすべて、リンカーオプション、オブジェクトプログラムファイル名、またはライブラリ名である可能性があるとみなされます。

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。ただし、これらのオプションがリンカーに渡されて、リンカーに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。

次に例を示します。

```
demo% f95 -bit move.f                 <- -bit is not a recognized f95 option
f95: Warning: Option -bit passed to ld, if ld is invoked, ignored otherwise
demo% f95 fast move.f                 <- The user meant to type -fast
ld: fatal: file fast: cannot open file; errno=2
ld: fatal: File processing errors.  No output written to a.out
```

最初の例では、**-bit** は **f95** では認識されず、このオプションはこれを解釈しようとするリンカー (**ld**) に渡されます。**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。

前述の例から、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 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。検出されたモジュール (**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、183 ページの「4.9 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。**f95** も OpenMP 共有メモリー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、181 ページの「4.8 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 C「Fortran 指令の要約」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 指令は次のような書式で使用します。

!$PRAGMA *keyword* **(** *a* [, *a*] … **)** [, *keyword* **(** *a* [, *a*] … **)**] ,…

!$PRAGMA SUN *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、前述に示す追加のキーワード **SUN** を指定する必要があります。

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** の自由形式では、**!** は認識される唯一の注釈指示子 (**!$PRAGMA**) です。この章の例では、Fortran 95 自由形式を想定しています。
- 次の 7 文字は空白文字を入れず **$PRAGMA** とします。大文字でも小文字でもかまいません。
- 自由形式ソースプログラムでは、**!** という注釈指示子文字を使用する指令は、行のどの位置にも記述できます。

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 文字のあとでは、空白は無視され、大文字と小文字は区別されません。
- これは注釈であるため、指令は継続できませんが、必要に応じて多くの **!$PRAGMA** 行を順番に使用できます。
- 注釈が前述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。前述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、**fpp** は正当な **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。

  ```
  !$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	**!$PRAGMA C(***list* **)** 外部関数の名前リストを C 言語のルーチンとして宣言します。
IGNORE_TKR 指令	**!$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	**!$PRAGMA SUN UNROLL=***n* コンパイラに、次のループは長さ *n* に展開できることを伝えます。
WEAK 指令	**!$PRAGMA WEAK(***name*[**=***name2*]**)** *name* を弱いシンボル (weak symbol) または *name2* の別名として宣言します。
OPT 指令	**!$PRAGMA SUN OPT=***n* 副プログラムの最適化レベルを *n* に設定します。
PIPELOOP 指令	**!$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。
PREFETCH 指令	**!$PRAGMA SUN_PREFETCH_READ_ONCE(***name***)** **!$PRAGMA SUN_PREFETCH_READ_MANY(***name***)** **!$PRAGMA SUN_PREFETCH_WRITE_ONCE(***name***)** **!$PRAGMA SUN_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。このオプションはデフォルトで有効になっています。PREFETCH 指令は、**–xprefetch=no** でコンパイルし無効にします。ターゲットアーキテクチャーも PREFETCH 指令をサポートしている必要があり、コンパイラ最適化レベルは **-xO2** より上である必要があります)。
ASSUME 指令	**!$PRAGMA** [**BEGIN**} **ASSUME (***expression* [**,***probability*]**)** **!$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。**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 のインタフェース」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
!$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、次の手続きのインタフェースでは、**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可能です。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、Fortran 95 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
```

The subroutine call:

```
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、Fortran 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、**!$PRAGMA** のあとに **SUN** と指定する必要があります。

!$PRAGMA SUN UNROLL=*n* 指令は、最適化パス中に、次のループを *n* 回展開するようにコンパイラに指示します。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。詳細については、『Fortran プログラミングガイド』のパフォーマンスと最適化に関する章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。

```
!$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。リンカーは弱いシンボルを解決できなくてもエラーメッセージを表示しません。

```
!$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* が優先順位の低いシンボルとして、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、**!$PRAGMA** のあとに **SUN** と指定する必要があります。

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

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

上記の例を、**-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。このルーチンのあとに別の指令がないかぎり、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 PIPELOOP[= *n*] 指令

PIPELOOP=*n* 指令では、**!$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。*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。(*n* が指定されない場合のデフォルトは 0 です)

```
C    We know that the value of K is such that there can be no
C    cross-iteration dependencies (E.g. K>N)
!$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。

2.3.1.7 PREFETCH 指令

-xprefetchオプションフラグを使用すると (140 ページの「3.4.161 **–xprefetch**[= *a*[,*a*]]」 を参照)、コンパイラに指示した一連の **PREFETCH** 指令は、先読みをサポートするプロセッサで指定のデータ要素について先読み命令を生成できます。

```
!$PRAGMA SUN_PREFETCH_READ_ONCE(name)
!$PRAGMA SUN_PREFETCH_READ_MANY(name)
!$PRAGMA SUN_PREFETCH_WRITE_ONCE(name)
!$PRAGMA SUN_PREFETCH_WRITE_MANY(name)
```

先読み命令についての詳細は、『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。

2.3.1.8 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、次のようになります。

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

また、「範囲表明」**ASSUME** は、次のようになります。

```
!$PRAGMA BEGIN ASSUME [expression [, probability)
      block of statements
!$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、コンパイラにこれを示しておくと、より良いコードを生成できます。通常、次のループは、**ASSUME** プラグマがある場合の方がすばやく実行されます。

```
!$PRAGMA BEGIN ASSUME(__tripcount().GE.10000,1) !! a big loop
        do i = j, n
           a(i) = a(j) + 1
        end do
!$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、もっとも外側の閉じ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。(110 ページの「3.4.111 –xassume_control[=*keywords*]」 を参照) 。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、プログラムの終了時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

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

-xassume_control=check を使用して前述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 並列化の指令

並列化 指令は、コンパイラに対して、指令のあとに続く **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** を使用してコンパイルされる場合にのみ認識されます。Fortran の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 3.0 共有メモリー並列化モデルをサポートします。従来の Sun および Cray の並列化指令は現在推奨されていないため、使用すべきではありません。

2.3.2.1 OpenMP 並列化指令

Fortran コンパイラでは、並列プログラミングモデルとして OpenMP Fortran 共有メモリーマルチプロセッシング 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) によって仕様が定められています。

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する必要があります (137 ページの「3.4.153 **–xopenmp**[={**parallel**|**noopt**|**none**}]」 を参照)。

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

注 – 従来の Sun および Cray 形式の並列化指令は非推奨になりました。Open MP 並列化 API が推奨されます。従来の Sun/Cray 指令から OpenMP モデルへの移行方法については、『OpenMP API ユーザーズガイド』を参照してください。

2.3.3 IVDEP 指令

!DIR$ IVDEP 指令は、ループ内で検出された一部またはすべての配列参照のループがもたらす依存関係を無視し、特にほかの方法では実行できないマイクロベクトル化、配布、ソフトウェアパイプラインなどのさまざまなループの最適化を実行するように、コンパイラに指示します。これは、依存関係が重要ではない、または依存関係が実際に発生しないことをユーザーが把握している状況で使用されます。

次に例を示します。

```
DO I = 1, N
  A(V(I)) = A(V(I)) + C(I)
END DO
```

このループでは、`A(V(I))` へのいくつかのループがもたらす依存関係があります。`V(I)` には、インデックス `A` への複製値が含まれている可能性があり、ループの順序を変更すると、異なる結果になる可能性があります。ただし、`V` に固有の値のみ含まれていることがわかっている場合は、ループの順序を安全に変更でき、**IVDEP** 指令を使用して最適化を実行できます。

-xivdep コンパイラオプション (128 ページの「3.4.133 **–xivdep**[=*p*]」 を参照) を使用して、**IVDEP** 指令の解釈を無効にするか、指定することができます。

IVDEP 指令の従来の解釈の中には、後方へのループがもたらす依存関係がないことを表明するだけのものがあります。Fortran コンパイラのデフォルトは **–xivdep=loop** です。これは、**IVDEP** 指令で推測されるループがもたらす依存関係がないことを表明することを示します。

次に、後方および前方への依存関係の例を示します。

```
 do i = 1, n            ! BACKWARD LOOP-CARRIED DEPENDENCE
   ... = a(i-1)         ! S1
   a(i) = ...          ! S2
end do
```

```
     do i = 1, n             ! FORWARD LOOP-CARRIED DEPENDENCE
        a(i) = ...         ! S3
        ... = a(i-1)         ! S4
     end do
```

最初のループは S2 から S1 への後方へのループがもたらす依存関係で、2 つ目のループには、S3 から S4 への前方へのループがもたらす依存関係があります。2 つ目のループには前方への依存関係しかないため、このループは安全に配布やマイクロベクトル化が行えますが、最初のループは行えません。

次に、デフォルト値 **-xivdep=loop** を指定した **IVDEP** の使用例を示します。

```
integer target a(n)
integer, pointer :: p(:), q(:)
!DIR$ IVDEP
do i = 1, n
     p(i) = q(i)
     a(i) = a(i-1)
end do
```

p(i) と q(i) 間、および p(i) と a(*) 間の推測される依存関係は無視されますが、a(i) と a(i-1) 間の明確な依存関係は無視されません。ループは 2 つのループに分割でき、その結果の p(i) = q(i) ループはマイクロベクトル化できます。

IVDEP 指令は、DO ループの直後に適用されます。この指令とループとの間にほかのコードを使用することはできません。**!DIR$ IVDEP** は、配列代入、**FORALL**、または **WHERE** 構文にも適用できます。特定のループに対して複数の指令が存在する場合 (**IVDEP** や **UNROLL** など)、コンパイラは可能な限りそれらすべての指令に従います。

2.4 ライブラリインタフェースと system.inc

Fortran 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、関数の明示的な型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、**getuid()** 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 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 (詳細は、『Fortran ライブラリリファレンス』を参照)。

2.5 コンパイラの利用方法

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

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。コンパイラの **-dryrun** オプションを使用して、ネイティブプロセッサを特定できます。

```
<sparc>%f95 -dryrun -xtarget=native
###     command line files and options (expanded):
### -dryrun -xarch=sparcvis2 -xcache=64/32/4:1024/64/4 -xchip=ultra3i

<x64>%f95 -dryrun -xtarget=native
###     command line files and options (expanded):
### -dryrun -xarch=sse2a -xcache=64/64/2:1024/64/16 -xchip=opteron
```

2.5.2 環境変数の使用

FFLAGS または **OPTIONS** 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できます。**make** 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** 変数が前述のように設定されており、メイクファイルの「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 Sun 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」の章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** (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 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。最適化でメモリーが不足した場合、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

Solaris オペレーティングシステムのコマンドである **swap -s** コマンドは、利用可能なスワップ領域を表示します。**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
To determine the actual real memory:

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

2.5.3.2 スワップ領域の増加

ワークステーションのスワップ領域を増やすには、**mkfile**(1M) と **swap**(1M) コマンドを使用します。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。これを制御するには、1 つのプロセスで使用できる仮想メモリーの量を制限します。

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。データ領域の追加割り当てが必要な場合は、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

3 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: 45 ページの「3.1 コマンド構文」
- 機能別のオプションのまとめ: 47 ページの「3.3 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: 55 ページの「3.4 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行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あと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追加ファイルリストが含まれる可能性がありました。

3.2 オプションの構文

標準のコンパイラオプション形式は次のとおりです。

表 3–1 オプションの構文

構文形式	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

記法	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**
|	「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**

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次に示します。

- **-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あとに指定して、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。これはリンカーオプションには当てはまりません。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、コマンド行に指定した順にコンパイルとリンクが行われます。

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、概略を説明します。詳細は、以降の節のページを参照してください。

SPARC および x64/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** コンパイラオプションの概略を機能別に示します。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

関数		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**
	コンパイル中に作成された一時ファイルを保持します。	**-keeptmp**
	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*
	コンパイルし、アセンブリコードだけを生成します。	**-S**

表 3–3 機能別コンパイラオプション (続き)

関数		オプションフラグ
	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*
	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、マルチワードのフェッチ/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)

関数		オプションフラグ
デバッグ		
	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**
	未宣言変数の検査を行います。	**-u**
	!$PRAGMA ASSUME 表明を確認します。	**-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グ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**
	指定された警告メッセージを抑制します。	**-erroff=**
	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**
	動的 (共有オブジェクト) ライブラリを作成します。	**-G**

表 3–3 機能別コンパイラオプション (続き)

関数		オプションフラグ
	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;*
	ライブラリ検索パスを実行可能プログラムに組み込みません。	**-norunpath**
	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**
	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*

表 3–3 機能別コンパイラオプション (続き)

関数		オプションフラグ
	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*]**
	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*]**
	相互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*]**
	コンパイル後の最適化用にコンパイルします。	**-xbinopt=prepare**
	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**
	ループの並列化情報を表示します。	**-loopinfo**
	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。	**-xopenmp[=*keyword*]**
	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;*

表 3–3 機能別コンパイラオプション (続き)

関数		オプションフラグ
	拡張 (132 文字) ソース行を受け入れます。	**-e**
	.F、**.F90**、および **.F95** のファイルにプリプロセッサを適用するが、コンパイルは行いません。	**-F**
	固定形式の 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		
	32 または 64 ビットのメモリーモデルを指定します。	**-m32 | –m64**
	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ーを指定します。	**-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。次の表に、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

処理	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プログラムを起動します。	**–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生成します。	**-fast**
動的 (**-Bdynamic**) または静的 (**-Bstatic**) ライブラリと結合します。	**-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	現在の展開内容の詳細は、**–fast** を参照してください。
-fnonstd	**-fns -ftrap=common**
-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あと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または追加されます。

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

処理	オプション
ENTRY文に対する実際の引数を保持します。	**-arg=local**
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**
非標準の演算 - 非標準の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。

表 3–7 旧 **f95** オプション (続き)

オプションフラグ	同等なオプションフラグ
-pic	**-xcode=pic13**
-PIC	**-xcode=pic32**
-sb	無視されます。
-sbfast	無視されます。
-silent	無視されます。
-xarch={**v7**、**v8**、**v8a**}	**–m32**を使用してください。

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 x64/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。x64/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いるため、使用しないでください。多くの場合、代わりに使用すべきほかのオプションまたはフラグに置き換えられています。

3.4.1 –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** などのほか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、すべて 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において **-aligncommon=8** に戻ります。

-aligncommon=1 を **-xmemalign** とともに使用しないでください。これらの指令は衝突するため、同じプラットフォームや構成でセグメント例外が発生する場合があります。

SPARC プラットフォームで **–aligncommon=1** を使用すると、不正な整列によるバスエラーが発生する可能性があります。この場合は、**–xmemalign** オプションの使用を選択する必要があります。アプリケーションに応じて、**–xmemalign=1s**、**–xmemalign=4i**、または **–xmemalign=8i** を指定すると、パフォーマンスを最適化するとともに、セグメント例外を回避できます。

–xmemalign も参照してください。

3.4.2 -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 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。

3.4.3 -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副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 FORTRAN 77 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。

3.4.4 -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 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。**-autopar** を使用している場合は、**-stackvar** オプションを使用した方がパフォーマンスが改善される場合があります。これは、オプティマイザが並列化する機会をより多く検出できるようになるためです。メインスレッドおよびスレーブスレッドのスタックサイズを設定する方法については、**-stackvar** オプションの説明を参照してください。

プログラム中に **libthread** スレッドライブラリへの明示的な呼び出しがある場合は、**-autopar** は使用しないでください。83 ページの「3.4.56 **-mt**[={ **yes**|**no**}]」 の注釈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。これにより、プログラムで作成できる最大スレッド数を実行時システムに設定します。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仮想プロセッサ数を設定します。この数は、Solaris の **psrinfo**(1) コマンドを使用して特定することができます。

-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。

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。明示的にユーザーが制御して並列化を行う場合は、OpenMP 指令および **–xopenmp** オプションを使用します。

3.4.5 –B{static| dynamic}

動的または静的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)。

次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 -Bx オプションを指定する必要があります。

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

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。これらのシステムライブラリには、**libm.so** および **libc.so** があります。**libm.a** と **libc.a** は提供していません。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。

Fortran 実行時システムの静的ライブラリと動的ライブラリを組み合わせて使用することは推奨しません。リンカーエラーが発生したり、データが警告なしに破壊されたりする可能性があります。必ず、Fortran 実行時システムの最新の共有動的ライブラリとリンクさせてください。

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。

3.4.6 -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、セグメント例外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。

3.4.7 -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。

3.4.8 -copyargs

定数の引数への代入を可能にします。

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。

3.4.9 –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 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。マイクロシンボル *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で使用してください (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、Solaris Studio 12 リリースでは **0x850** 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** フラグを使用すると、コンパイラは **fpp** ではなく **cpp** を使用します。

3.4.10 -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 **-m64** 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。

-dalign を **-xtypemap=real:64,double:64,integer:64** とともに使用すると、SPARC プロセッサで 64 ビット整数変数がダブルワードで境界整列されます。

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場合があります。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、次と同等なマクロです。

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。55 ページの「3.4.1 **–aligncommon**[={ **1**|**2**|**4**|**8**| **16**}]」 を参照してください。

3.4.11 –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 バイトの境界に整列されます。デフォルトは、**-dbl_align_all=no** です。

64 ビット環境で **-m64** を使用してコンパイルした場合、4 倍精度のデータは 16 バイト境界に整列されます。

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。

3.4.12 –depend[={ yes|no}]

反復間の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。ループの再構築には、ループ交換、ループ融合、およびスカラー置換が含まれます。

-depend を指定しない場合、デフォルトは **-depend=yes** です。**-depend** を指定しても、引数を指定しない場合、コンパイラは **-depend=yes** を使用します。

依存解析をオフにするには、**-depend=no** でコンパイルします。

-xdepend は **-depend** と同義です。

3.4.13 -dn

動的ライブラリを使用不可能にします。62 ページの「3.4.15 **–d{ y|n}**」を参照してください。

3.4.14 -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。

3.4.15 –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**: はい、 動的/共有ライブラリを使用できます。
- **-dn**: いいえ、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 回だけ使用します。

-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。これらのシステムライブラリには、**libm.so** および **libc.so** があります。**libm.a** と **libc.a** は提供していません。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 release 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。

3.4.16 -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に挿入されてしまいます。

3.4.17 –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、警告は抑制されません。引数なしで **–erroff** を指定した場合は、**–erroff=%all** を指定するのと同じです。

次に例を示します。

```
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。

3.4.18 –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。**-errtags** だけの場合は **-errtags=yes** と同じです。

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", line 11: Warning: local variable "i" never used (WDECL_LOCAL_NOTUSED)
```

3.4.19 –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。

3.4.20 -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**、**underscores**、または **fsecond-underscore** のいずれかを指定します。デフォルトは **underscores** です。

-ext_names=plain: 下線を付けません。

-ext_names=underscores: 下線を付けます。

–ext_names=fsecond-underscore: 下線を含む外部名に二重下線を付け、下線を含まない外部名に一重下線を付けます。

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 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す、または呼び出しを受けることができます。

fsecond-underscore は、**gfortran** との互換性のために用意されています。

3.4.21 -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。

コマンド行に表示された **.F**、**.F90**、**.F95**、および **.F03** ソースファイルに **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。

次に例を示します。

```
f95 -F source.F
```

このコマンドを実行すると、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。

3.4.22 -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** と同義です。**-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 **-m64** 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** オプションがこれを実行し、**-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。60 ページの「3.4.10 **-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。

3.4.23 –f77[= *list*]

FORTRAN 77 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、Sun WorkShop **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 コンパイラへの移植が可能になります(FORTRAN 77 コンパイラは存在しません)。

list は、次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。

キーワード	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 組み込み関数のみに制限します。
logical	次に示す論理変数の 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。
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します。
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 のトラップモードが変更されず、**-ftrap=common** になります。**f95** と FORTRAN 77 コンパイラは、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あと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あとに **-ftrap=%none** を指定すると、元の FORTRAN 77 の動作を真似することができます。

f77 の互換性および FORTRAN 77 から Fortran 95 への移行の詳細は、186 ページの「4.12 言語の混在」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。

3.4.24 -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ほかのオプションを選択する機能として定義されています。**-fast** により選択されるいくつかのオプションはすべてのプラットフォームで使用できない可能性があります。**-fast** の展開を表示するには、**-dryrun** 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、オプションによっては、アプリケーションで使用できない場合があります。**-fast** を使用して、最大のパフォーマンスを得るためにアプリケーションをコンパイルしてください。しかし、さらに調整が必要な場合があります。**-fast** を指定してコンパイルしたプログラムが正しく動作しない場合、**-fast** を形成している個々のオプションを調査して、プログラムを正しく動作させる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ーに依存しているプログラムは、**-fast** を使用してコンパイルしないでください。

-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、リンク時も必ず **-fast** を指定してください。

-fast では次のオプションが選択されます。

- **-xtarget=native** ハードウェアターゲット。
  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** のあとにコード生成オプションを付けます。例: **f95 -fast -xtarget=ultraT2 ...**
- **-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築についてループを解析します。
- システムが提供するインライン展開テンプレート用の **-libmil** オプション。
  例外処理を使用するCモジュールでは、**-fast** のあとに **-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。73 ページの「3.4.35 **–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=lib** 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。**(SPARC)**
- **–fns** は、標準外の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。70 ページの「3.4.29 **–fns**[**=**{**yes**| **no**}]」 を参照してください。
- **–xvector** および **–xlibmopt** で必要なため、**-fround=nearest** が選択されます。**(Solaris)**
- 共通の浮動小数点例外のトラッピング **-ftrap=common** は、**f95** で有効です。
- **-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)**
- x86 で **-xregs=frameptr** を使用すると、コンパイラは汎用レジスタとしてフレームポインタレジスタを使用できます。特に C、C++、Fortran が混在するソースコードをコンパイルする場合は、詳細について **–xregs=frameptr** の説明を参照してください。**-fast** のあとに **-xregs=no%frameptr** を指定すると、フレームポインタレジスタは通常の用途でのレジスタとして使用されません。**(x86)**

次に示すように、**-fast** オプションのあと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 フラグで選択する一連のオプションは、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。**-dryrun** を指定してコンパイラを呼び出すと、**-fast** の展開値が表示されます。

```
<sparc>%f95 -dryrun -fast |& grep ###
          ###     command line files and options (expanded):
          ### -dryrun -xO5 -xarch=sparcvis2 -xcache=64/32/4:1024/64/4
```

```
-xchip=ultra3i -xdepend=yes -xpad=local -xvector=lib
-dalign -fsimple=2 -fns=yes -ftrap=common -xlibmil
-xlibmopt -fround=nearest
```

3.4.25 -fixed

固定形式の 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形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形式として解釈し、**.f95** ファイルを自由形式として解釈します。

3.4.26 -flags

-help と同義です。

3.4.27 –fma={none| fused}

(SPARC) 浮動小数点演算、融合演算、積和演算命令の自動生成を有効にします。**-fma=none** は、これらの命令の生成を無効にします。**-fma=fused** は、コンパイラが浮動小数点演算、融合演算、積和演算命令を使用して、コードのパフォーマンスを改善する機会を見つけようとすることを許可します。デフォルトは **-fma=none** です。

融合した積和演算命令を生成するには、コンパイラに対して **-xarch=sparcfmaf** および最適化レベルが **-xO2** 以上に設定されていることが最低限必要です。融合した積和演算命令が生成された場合は、これらの命令をサポートしていないプラットフォームでプログラムが実行されないように、コンパイラはバイナリプログラムにマークを付けます。

積和演算 (FMA) により、積と和 (乗算と加算) の間で中間の丸め手順が排除されます。その結果、**-fma=fused** を指定してコンパイルしたプログラムは、精度は減少ではなく増加する傾向にありますが、異なる結果になることがあります。

3.4.28 -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、次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。

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。

このオプションを使用すると、浮動小数点ハードウェアが初期化されて次の処理が実行されます。

- 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』、および『Fortran プログラミングガイド』を参照してください。

3.4.29 –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」の章を参照してください。

-fast などの **-fns** フラグが含まれるマクロフラグのあとに **=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 プラットフォームで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」が無効になります。つまり、小さな結果は、非正規数にはならず、ゼロに切り捨てられます。さらに、このモードでは、非正規のオペランドが報告なしに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場合、|x| が次の範囲にある数であるときにのみ、*x* は非正規数になります。

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」の章を参照してください。(演算によっては、「非正規数」を表すのに「指数が最小の非正規化数」という用語を使用している場合があります)。

デフォルトでは、浮動小数点は標準の設定に初期化されます。

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

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 プロセッサ (sse または sse2 命令セット) でのみ有効です。

x86 では、**-fns** は SSE flush-to-zero モードを選択します。利用可能な場合には、denormals-are-zero モードが選択されます。このフラグは、非正規数の結果をゼロに切り捨てます。また、利用可能な場合には、非正規数オペランドもゼロとして扱われます。このフラグは、SSE または SSE2 命令セットを利用しない従来の x87 浮動小数点演算には影響しません。

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。

3.4.30　–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** と同等です。**–ftrap** とともに使用すると、完全な診断情報が表示されます。

3.4.31 -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** のファイルのみです。140 ページの「3.4.160 –xpp={fpp| cpp}」も参照してください。

3.4.32 –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 プラットフォームで、浮動小数点精度モードを **single**、**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 システムでメインプログラムのコンパイル時に使用する場合にのみ有効で、64 ビット (**-m64**) または SSE2 対応 (**-xarch=sse2**) プロセッサでコンパイルする場合は無視されます。SPARC システムでも無視されます。

3.4.33 -free

自由形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** 自由形式と解釈されます。通常、**f95** は **.f** のファイルだけを固定形式として解釈し、**.f95** ファイルを自由形式として解釈します。

3.4.34 –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、次に示す IEEE 754 丸めモードを設定します。

- 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」の章を参照してください。

3.4.35 –fsimple[={ 1|2|0}]

浮動小数点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」の章を参照してください。

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

デフォルトは次のとおりです。

- **-fsimple** フラグが指定されていない場合、コンパイラは **-fsimple=0** とみなします。
- 値なしで **-fsimple** が指定されている場合、コンパイラは **-fsimple=1** を使用します。

別の浮動小数点単純化レベルは次のとおりです。

-fsimple=0 仮定の設定を許可しません。IEEE 754 に厳密に準拠します。

-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 の場合、次に示す内容を前提とした最適化が行われます。

- IEEE 754 のデフォルトの丸めとトラップモードが、プロセスの初期化以後も変わらない。
- 浮動小数点例外以外には、目に見える結果が生じない演算は削除できる。
- 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 を指定すると、オプティマイザは必ず丸めまたは例外に応じた、完全な最適化を行います。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 **–fsimple=1** に加えて、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ーに依存するプログラムは、**-fsimple=2** でコンパイルしないでください。

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。

3.4.36 -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** のあとに **-fstore** を続けてください。

3.4.37 -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、次の 1 つまたは複数の項目をコンマで区切って指定します。

%all、**%none**、**common**、**[no%]invalid**、**[no%]overflow**、**[no%]underflow**、**[no%]division**、**[no%]inexact**。

-ftrap=common は、**-ftrap=invalid,overflow,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 754 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 ハンドラの組み込みを同時に行うには、**ieee_handler**(3M) か **fex_set_handling**(3M) を使用します。複数の値を指定すると、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。

次の点を除いて、**-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」の章を参照してください。

3.4.38 -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」の章を参照してください。

3.4.39 -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ーによるデバッグ、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** とデバッガ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、ほかのオプションの機能の一部が制限される場合があります。詳細は、『**dbx** コマンドによるデバッグ』を参照してください。

パフォーマンスアナライザの機能を最大限に利用するには、**-g** オプションを指定してコンパイルします。一部のパフォーマンス解析機能では、 **-g** オプションを必要としませんが、注釈付きのソース、一部の関数レベル情報、およびコンパイラの注釈メッセージを表示するには **-g** を指定してコンパイルする必要があります。詳細は、**analyzer**(1) マニュアルページおよびマニュアル『Solaris Studio パフォーマンスアナライザ』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。

3.4.40 -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ガイド』および『Fortran プログラミングガイド』の「ライブラリ」の章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** のあと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。

3.4.41 -help

コンパイルオプションの一覧を表示します。

122 ページの「3.4.125 **–xhelp=**{**readme**| **flags**}」を参照してください。

3.4.42 -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。

検索パスは、**MODULE** ファイルの検索にも使用されます。

例: **/usr/app/include** で **INCLUDE** ファイルを検索するには、次のようにします。

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

コマンド行で複数回 -I**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**、**.F90**、**.F95**、または **.F03** の接尾辞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。

3.4.43 -i8

-i8 オプションはありません。

このコンパイラで 8 バイト **INTEGER** を指定するには、**–xtypemap=integer:64** を使用します。

3.4.44 –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%** という接頭辞を付けると、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 **-O4** または **-O5** 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** も指定した場合は除く)。なお、**-O4** を指定すると、コンパイラは通常、ユーザー作成の適切な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
```

前述の例では、**-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。

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

書式付き入出力の浮動小数点の丸め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。

3.4.46 -keeptmp

コンパイル中に作成された一時ファイルを保持します。

3.4.47 -Kpic

*(*廃止*)* **-pic** と同義です。

3.4.48 -KPIC

*(*廃止*)* **-PIC** と同義です。

3.4.49 -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。80 ページの「3.4.50 **-l***x*」も参照してください。

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」の章を参照してください。**LD_LIBRARY_PATH** および **-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
```

3.4.50 -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。**ld** は、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」の章を参照してください。

3.4.51 -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。

一部の **libm** ライブラリルーチンには、インラインテンプレートがあります。このオプションを指定すると、これらのテンプレートが選択され、現在選択されている浮動小数点オプションとプラットフォームに対してもっとも高速な実行可能コードが生成されます。

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。

3.4.52 -loopinfo

ループの並列化結果を表示します。

–autopar オプションで並列化されたループと並列化されなかったループを表示します。

-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 -c -fast -autopar -loopinfo shalow.f
...
"shalow.f", line 172: PARALLELIZED, and serial version generated
"shalow.f", line 173: not parallelized, not profitable
"shalow.f", line 181: PARALLELIZED, fused
"shalow.f", line 182: not parallelized, not profitable
...
...etc
```

3.4.53 -M*path*

MODULE ディレクトリ、アーカイブ、またはファイルを指定します。

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

path には、ディレクトリ、**.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)、または **.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、モジュールが検索されることが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。

これは、**-M** フラグのみが表示されている場合は、**-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。

検索対象の場所でファイルが見つからない場合は、**–I** *path* で指定されたディレクトリでモジュールファイルが検索されます。

-M とパスの間に空白文字を入れてもかまいません。たとえば、**-M /home/siri/PK15/Modules** のようにします。

Solaris で、アーカイブやモジュールファイル以外の通常ファイルをパスに指定した場合は、コンパイラは **ld** オプションをリンカーに渡し、リンカーマップファイルとしてファイルを処理します。これは C および C++ コンパイラと同様の便利な機能です。

Fortran モジュールについての詳細は、183 ページの「4.9 モジュールファイル」を参照してください。

3.4.54 –m32 | –m64

コンパイルされたバイナリオブジェクトのメモリーモデルを指定します。

32 ビット実行可能ファイルおよび共有ライブラリを作成するには、**-m32** を使用します。64 ビット実行可能ファイルおよび共有ライブラリを作成するには、**-m64** を使用します。

ILP32 メモリーモデル (32 ビット int、long、ポインタデータ型) は 64 ビット対応ではないすべての Solaris プラットフォームおよび Linux プラットフォームのデフォルトです。LP64 メモリーモデル (64 ビット long、ポインタデータ型) は 64 ビット対応の Linux プラットフォームのデフォルトです。**-m64** は、LP64 モデルが使用可能なプラットフォームでのみ許可されます。

-m32 でコンパイルされたオブジェクトファイルまたはライブラリは、**-m64** でコンパイルされたオブジェクトファイルまたはライブラリとリンクできません。

x64 プラットフォームで大量の静的データを持つアプリケーションを **-m64** を使用してコンパイルするときは、**-xmodel=medium** も必要になることがあります。

一部の Linux プラットフォームは、ミディアムモデルをサポートしていません。

旧バージョンのコンパイラでは、**-xarch** で命令セットを選択することで、メモリーモデルの ILP32 または LP64 が暗黙に指定されていました。Solaris Studio 12 以降のコンパイラでは、このようなことはありません。ほとんどのプラットフォームでは、64 ビットオブジェクトを作成するのに、コマンド行に **-m64** を追加するだけです。

Solaris では、**-m32** がデフォルト値です。64 ビットプログラムをサポートしている Linux システムでは、**-m64 -xarch=sse2** がデフォルト値です。

3.4.55 -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します。

Fortran モジュールについての詳細は、183 ページの「4.9 モジュールファイル」を参照してください。

3.4.56 -mt[={ yes|no}]

このオプションを使用して、Solaris スレッドまたは POSIX スレッドの API を使用しているマルチスレッド化コードをコンパイルおよびリンクします。**-mt=yes** オプションにより、ライブラリが適切な順序でリンクされることが保証されます。

このオプションは **-D_REENTRANT** をプリプロセッサに渡します。

Linux プラットフォーム上では、POSIX スレッドの API のみが使用できます (Linux プラットフォームには `libthread` はありません)。したがって、Linux プラットフォームで **-mt=yes** を使用すると、**–lthread** の代わりに **–lpthread** が追加されます。Linux プラットフォームで POSIX スレッドを使用するには、**–mt** を使用してコンパイルします。

–G を使用してコンパイルする場合は、**-mt=yes** を指定しても、**–lthread** と **–lpthread** のどちらも自動的には含められません。共有ライブラリを構築する場合は、これらのライブラリを明示的にリストする必要があります。

(OpenMP 共有メモリー並列化 API を使用するための) **–xopenmp** オプションには、**-mt=yes** が自動的に含まれます。

-mt=yes を指定してコンパイルを実行し、リンクを個別の手順でリンクする場合は、コンパイル手順と同様にリンク手順でも **-mt=yes** オプションを使用する必要があります。**-mt=yes** を使用して 1 つの変換ユニットをコンパイルおよびリンクする場合は、**-mt=yes** を指定してプログラムのすべてのユニットをコンパイルおよびリンクする必要があります。

-mt=yes は、コンパイラのデフォルトの動作です。この動作が望ましくない場合は、**-mt=no** でコンパイルします。

オプション **-mt** は、**-mt=yes** と同じです。

3.4.57 -native

(廃止) ホストシステムに対してパフォーマンスを最適化します。

このオプションは、**-xtarget=native** と同義です。-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。

3.4.58 -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。

3.4.59 -nodepend

コマンド行で先に指定された **-depend** を取り消します。**-depend=no** は、**-nodepend** よりも優先して使用されます。

3.4.60 -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。

3.4.61 -nolib

システムライブラリとリンクしません。

どの システムライブラリや言語ライブラリとも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なってください。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。

3.4.62 -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** オプションのあとに使用して、**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 ...
```

3.4.63 -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。

3.4.64 -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***paths* 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」の章を参照してください。

3.4.65 –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** (レベル **-O3**) または **-fast** (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 を使用するデバッグは **-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」の章を参照してください。

3.4.66 -0

-03 と同義です。

3.4.67 -01

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を超えている場合に使用します。

3.4.68 -02

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** も参照)。

-03 を使用すると、コンパイル時間が長すぎる場合、スワップ領域を超えている場合、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 **-02** を使用します。これ以外の場合は、**-03** を使用してください。

3.4.69 -03

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、**-03** では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。

3.4.70 -04

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、**-04** 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、前に説明した **-04** 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、**-04** によるインライン化の範囲が拡張されます。

3.4.71 -O5

最高レベルの最適化を行います。

プログラムの中で、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する場合もあります。

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。**-xprofile=***p* を参照してください。

3.4.72 -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。

3.4.73 -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。

3.4.74 -openmp

-xopenmp と同義です。

3.4.75 -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に使用します。**gprof** を使用した **-pg** プロファイリングの方をお勧めします。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。

3.4.76 –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)。

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、次のとおりです。

- 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、次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、94 ページの「3.4.91 **-stackvar**」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も等値化されません。
- 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、予想外の形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。

3.4.77 -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** ファイルのあとに指定してください (**-pg** ライブラリは静的)。

注 – **-pg** を指定した場合、**-xprofile** でコンパイルする利点はありません。これら 2 つの機能は、他方で使用できるデータを生成せず、他方で生成されたデータを使用できません。

---

64 ビット Solaris プラットフォームで **prof**(1) または **gprof**(1)、32 ビット Solaris プラットフォームで **gprof** を使用して生成されたプロファイルには、おおよそのユーザー CPU 時間が含まれます。これらの時間は、メインの実行可能ファイルのルーチンと、実行可能ファイルをリンクするときにリンカー引数として指定した共有ライブラリのルーチンの PC サンプルデータ (**pcsample**(2) を参照) から導出されま

す。そのほかの共有ライブラリ (**dlopen**(3DL) を使用してプロセスの起動後に開かれたライブラリ) のプロファイルは作成されません。

32 ビット Solaris システムの場合、**prof**(1) を使用して生成されたプロファイルには、実行可能ファイルのルーチンだけが含まれます。32 ビット共有ライブラリのプロファイルは、**-pg** で実行可能ファイルをリンクし、**gprof**(1) を使用することで作成できます。

Solaris 10 ソフトウェアには、 **-p** でコンパイルされたシステムライブラリが含まれません。その結果、Solaris 10 プラットフォームで収集されたプロファイルには、システムライブラリルーチンの呼び出し回数が含まれません。

コンパイラオプション **-p**、**-pg**、または **-xpg** の実行時サポートは、スレッドに対して安全ではありません。そのため、マルチスレッドプログラムのコンパイルには使用しないでください。マルチスレッドを使用するプログラムをこれらのオプションを付けてコンパイルすると、実行時に、不正な結果やセグメント例外が発生する可能性があります。

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。

3.4.78 -pic

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。

SPARC では、**-pic** は **-xcode=pic13** と同等です。位置独立コードの詳細は、115 ページの「3.4.117 **-xcode=***keyword*」 を参照してください。

x86 では、位置独立コードを生成します。このオプションは、共有ライブラリを構築するためにソースファイルをコンパイルするときに使用します。大域データへの各参照は、大域オフセットテーブルにおけるポインタの間接参照として生成されます。各関数呼び出しは、手続きリンケージテーブルを通して PC 相対アドレス指定モードで生成されます。

3.4.79 -PIC

32 ビットアドレスで位置独立コードをコンパイルします。

SPARC では、**-PIC** は **-xcode=pic32** と同等です。位置独立コードの詳細は、115 ページの「3.4.117 **-xcode=***keyword*」 を参照してください。

x86 では、**–PIC** は **–pic** と同等です。

3.4.80 -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。リストには、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。

3.4.81 -qp

-p と同義です。

3.4.82 -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」の章および Solaris の『リンカーとライブラリガイド』を参照してください。

3.4.83 -r8const

単精度の定数を **REAL*8** 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、155 ページの「3.4.177 **-xtypemap=** *spec*」を参照してください。

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。

3.4.84 –recl=a[,b]

デフォルトの出力記録長を設定します。

接続済みの装置の出力 (標準の出力) と エラー (標準のエラー) のいずれかまたは両方に対するデフォルトの記録長 (文字数単位) を設定します。このオプションは、次のいずれかの書式で指定する必要があります。

- **–recl=out:***N*
- **–recl=error:***N*
- **–recl=out:***N1* **,error:***N2*
- **–recl=error:***N1* **,out:***N2*
- **–recl=all:***N*

ここで *N*、*N1*、*N2* は、72 ～ 2147483646 の範囲のすべての正の整数です。**out** は標準の出力を、**error** は標準のエラーを指し、**all** によってデフォルトの記録長が両方に設定されます。デフォルトは **-recl=all:80** です。このオプションは、コンパイルされるプログラムが Fortran 主プログラムを持つ場合にのみ有効です。

3.4.85 -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」の章を参照してください。

このオプションは、自動並列化オプション **–autopar** 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。

3.4.86 -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。

3.4.87 -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。

3.4.88 -sb

(廃止 - このオプションは無視されます)

3.4.89 -sbfast

(廃止 - このオプションは無視されます。)

3.4.90 -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。

3.4.91 -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** を持たないかぎり、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が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かぎり、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、セグメント例外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。

メインスタックのデフォルトのサイズは、約 8M バイトです。デフォルトのスレッドスタックサイズは、32 ビットシステムで 4M バイト、64 ビットシステムで 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** を使用したときにセグメント例外が発生する場合は、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         unlimited
filesize        unlimited
datasize        523256 kbytes
stacksize       8192 kbytes      <——
coredumpsize    unlimited
descriptors     64
memorysize      unlimited
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
```

各スレーブスレッドで使用されるスタックサイズは、**STACKSIZE** 環境変数に値 (キロバイト単位) を指定することで設定できます。

```
% setenv STACKSIZE 8192
```

この例は、各スレーブスレッドのスタックサイズを 8M バイトに設定します。

STACKSIZE 環境変数は、接尾辞 **B**、**K**、**M**、または **G** の付いた数値も受け付けます。これらの接尾辞はそれぞれ、バイト、キロバイト、メガバイト、ギガバイトを表します。デフォルトはキロバイトです。

STACKSIZE 環境変数は、**–xopenmp** または **–xautopar** オプションを使用してコンパイルされたプログラムにのみ影響し、Solaris システムで pthread インタフェースを使用するプログラムには影響しません。

並列化と **-stackvar** を併用する方法の詳細は、『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。**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。113 ページの「3.4.115 **-xcheck=***keyword*」を参照してください。

3.4.92 –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
```

0 ～ 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。

3.4.93 -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。このオプション文字列の中にはスペースを入れてはいけません。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。

このオプションは、**TMPDIR** 環境変数の値に優先されます。

3.4.94 -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。

3.4.95 –traceback[={ %none|common|*signals_list*}]

実行時に重大エラーが発生した場合にスタックトレースを発行します。

-traceback オプションを指定すると、プログラムによって特定のシグナルが生成された場合に、実行可能ファイルで **stderr** へのスタックトレースが発行されて、コアダンプが実行され、終了します。複数のスレッドが 1 つのシグナルを生成すると、スタックトレースは最初のスレッドに対してのみ生成されます。

追跡表示を使用するには、リンク時に **-traceback** オプションをコンパイラコマンド行に追加します。このオプションはコンパイル時にも使用できますが、実行可能バイナリが生成されない場合無視されます。**-traceback** を **-G** とともに使用して共有ライブラリを作成すると、エラーが発生します。

表 **3–9** -traceback オプション

オプション	意味
common	**sigill**、**sigfpe**、**sigbus**、**sigsegv**、または **sigabrt** の共通シグナルのいずれかのセットが発生した場合にスタックトレースを発行することを指定します。
signals_list	スタックトレースを生成するシグナルの名前を小文字で入力してコンマで区切ったリストを指定します。sigquit、sigill、sigtrap、sigabrt、sigemt、sigfpe、sigbus、sigsegv、sigsys、sigxcpu、sigxfsz のシグナル (コアファイルが生成されるシグナル) をキャッチできます。 これらのシグナルの前に **no%** を付けると、シグナルのキャッチは無効になります。 たとえば、**-traceback=sigsegv,sigfpe** と指定すると、sigsegv または sigfpe が発生した場合にスタックトレースとコアダンプが生成されます。
%none または **none**	追跡表示を無効にします。

このオプションを指定しない場合、デフォルトは **-traceback=%none** になります。

値を指定せずに、**-traceback** だけを指定すると、**-traceback=common** と同義になります。

注: コアダンプが不要な場合は、次を使用して coredumpsize 制限を 0 に設定できます。

```
% limit coredumpsize 0
```

-traceback オプションは、実行時のパフォーマンスに影響しません。

3.4.96 -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。組み込み関数の呼び出しは、このオプションによる影響を受けません。

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。『Fortran プログラミングガイド』の Solaris Studio Fortran への移植に関する章を参照してください。

3.4.97 -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 -Dname も対象となります。ソースファイルのマクロ定義には影響しません。コマンド行に複数の **-U***name* フラグを配置できます。**-U** とマクロ *name* の間に空白文字を入れることはできません。

3.4.98 -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。これは、各コンパイル単位で **IMPLICIT NONE** が指定されていることと同じです。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。

3.4.99 -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*>1 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」の章を参照してください。35 ページの「2.3.1.3 **UNROLL** 指令」も参照してください。

3.4.100 -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副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 モジュールについての詳細は、183 ページの「4.9 モジュールファイル」を参照してください。

3.4.101 -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。

前述の情報は、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。

3.4.102 -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。

3.4.103 –vax=キーワード

従来の 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	文字「D」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。このようなエラーを回避するには、**–xmemalign** と併せて使用してください。
%all	前述すべての VAX VMS 機能を有効にします。
%none	前述すべての 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。

次に例を示します。

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

デフォルトは **–vax=%none** です。サブオプションなしで —vax を指定すると、**–vax=%all** と同じ結果になります。

3.4.104 -vpara

並列化に関する詳細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。

-xopenmp および OpenMP API 指令とともに使用します。

警告は、コンパイラが次の状態を検出したときに表示されます。

- OpenMP 並列領域でのアクセスによってデータの競合が起きる可能性がある共有変数の宣言、並列領域に値があって並列領域のあとで使用されるスレッド固有変数の宣言など、OpenMP のデータ共有属性節の問題のある使用

すべての並列化指令が問題なく処理される場合、警告は表示されません。

注 – Solaris Studio のコンパイラは OpenMP API の並列化モデルをサポートします。そのため、従来の **C$MIC** 並列化指令は非推奨で、無視されます。OpenMP API への移植については、『OpenMP API ユーザーズガイド』を参照してください。

3.4.105 -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** と同義です。**-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。

3.4.106 -Xlist[*x*]

(*Solaris* のみ) 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、大域プログラムを通して、副プログラムの引数、共通ブロック、およびパラメータの一貫性をチェックします。また、このオプションは、相互参照表などの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
```

前述の例により、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る)
- 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 3–10 **-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。

Linux システムでは、このオプションはありません。

3.4.107 –xaddr32[={ yes|no}]

(x86/x64 のみ) **-xaddr32=yes** コンパイルフラグは、生成される実行可能ファイルまたは共有オブジェクトを 32 ビットアドレス空間に制限します。

この方法で実行可能ファイルをコンパイルすると、32 ビットアドレス空間に制限されたプロセスが生成されます。**-xaddr32=no** を指定すると、通常の 64 ビットバイナリが生成されます。**-xaddr32** オプションを指定しないと、**-xaddr32=no** が想定されます。**-xaddr32** だけを指定すると、**-xaddr32=yes** が使用されます。

このオプションは **-m64** コンパイルだけに適用され、**SF1_SUNW_ADDR32** ソフトウェア機能をサポートする Solaris プラットフォームでのみ適用されます。Linux カーネルはアドレス空間制限をサポートしないため、このオプションは Linux では使用できません。**-xaddr32** オプションは Linux では無視されます。

リンクするときには、単一のオブジェクトファイルが **-xaddr32=yes** を使用してコンパイルされていると、出力ファイル全体が **-xaddr32=yes** を使用してコンパイルされていると見なされます。32 ビットアドレス空間に制限された共有オブジェクトは、制限された 32 ビットモードアドレス空間内で実行されるプロセスによってロードされる必要があります。詳細は、『リンカーとライブラリ』で説明されている **SF1_SUNW_ADDR32** ソフトウェア機能の定義を参照してください。

3.4.108 –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に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します。

別名付けのキーワードは、次のとおりです。

表 3–11 **-xalias** オプションキーワード

キーワード	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、副プログラムの実引数を大域的な変数として扱います。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。
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。これらの構文で配列の境界を越えた添字付けが発生する場合は、**DO** ループとして構文を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。配列の参照がほかの変数を参照することもありません。
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 **-xO3** 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 の標準に準拠しているとみなされます。

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。

3.4.109 –xannotate[={ yes|no}]

(Solaris のみ) あとで **binopt**(1) などのバイナリ変更ツールで変換できるバイナリを作成するようにコンパイラに指示します。

将来のバイナリ解析、コードガバレージ、およびメモリーエラー検出ツールでも、このオプションを使用して構築されたバイナリを使用できます。

-xannotate=no オプションを使用すると、これらのツールでバイナリファイルを変更できなくなります。**-xannotate=yes** オプションは最適化レベル **-xO1** 以上で使用しないと有効になりません。また、新しいリンカーサポートライブラリインタフェース **-ld_open()** を備えたシステムでのみ有効になります。このリンカーインタフェースを備えていないシステム (Solaris 9 や 旧バージョンの Solaris 10 など) でコンパイラを使用すると、暗黙的に **-xannotate=no** に戻ります。

デフォルトは **-xannotate=yes** ですが、上記のいずれかの条件が満たされていないと、デフォルトは **-xannotate=no** に戻ります。

Linux システムでは、このオプションはありません。

3.4.110 -xarch=*isa*

命令セットアーキテクチャー (ISA) を指定します。

表 3–12 に、**-xarch** キーワード *isa* で受け付けられるアーキテクチャーを示します。

表 3–12 **-xarch** ISA キーワード

プラットフォーム	有効な **-xarch** キーワード
SPARC	**generic**、**generic64**、 **native**、**native64**、**sparc**、**sparcvis**、**sparcvis2**、**sparcvis3**、**sparcfmaf**、**v9**、**v9a**、**v9b**
x86	**generic**、**native**、**386**、**pentium_pro**、**sse**、**sse2**、**amd64**、**pentium_proa**、**ssea**、**sse2a**、**amd64a**、

-xarch は単独で使用できますが、**-xtarget** オプションの展開の一部です。特定の **-xtarget** オプションで設定されている **-xarch** の値を上書きするために使用することもできます。次に例を示します。

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

-xtarget=ultra2 で設定した **-xarch** 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ー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。このオプションは、すべてのターゲットを対象とするような命令としての使用は保証しません。

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ーを選択すると、そのアーキテクチャー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。不適切なアーキテクチャーを選択すると、バイナリプログラムがその対象プラットフォーム上で実行できなくなることがあります。

次の点に注意してください。

- generic、sparc、sparcvis2、sparcvis3、sparcfmaf、sparcima でコンパイルされたオブジェクトライブラリファイル (.o) をリンクして、一度に実行できます。ただし、実行できるのは、リンクされているすべての命令セットをサポートしているプロセッサのみです。
- 特定の設定で、生成された実行可能ファイルが実行されなかったり、従来のアーキテクチャーよりも実行速度が遅くなったりする場合があります。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ーのいずれにも実装されないため、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。

-xarch が指定されない場合のデフォルトは、**generic** 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プラットフォームの詳細を説明します。

表 3–13 SPARC プラットフォーム上の **-xarch** の値

-xarch=	意味 (SPARC)
generic	たいていのプロセッサに共通の命令セットを使用したコンパイルを行います。
generic64	たいていの **64** ビットプラットフォーム用のコンパイルを行います。 (Solaris のみ) このオプションは **-m64 -xarch=generic** と同等で、旧バージョンとの互換を提供します。64 ビットコンパイルを指定する場合は、**-xarch=generic64** ではなく **-m64** を使用します。
native	この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を行います。 現在コンパイルしているシステムプロセッサにもっとも適した設定を選択します。これは、**-fast** オプションのデフォルトです。
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を行います。 (Solaris のみ) このオプションは **-m64 -xarch=native** と同等で、旧バージョンとの互換を提供します。
sparc	**SPARC V9-ISA** 用のコンパイルを行います。 V9 ISA 用のコンパイルですが、Visual Instruction 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して、コンパイラは、V9 ISA 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。
sparcvis	**UltraSPARC** 拡張機能付きの **SPARC-V9 ISA** 用のコンパイルを行います。 SPARC-V9 + VIS (Visual Instruction Set) version 1.0 + UltraSPARC 拡張機能用のコンパイルを実行します。このオプションを使用すると、コンパイラは、UltraSPARC アーキテクチャー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。
sparcvis2	**UltraSPARC-III** 拡張機能付きの **SPARC-V9 ISA** 用のコンパイルを行います。 UltraSPARC アーキテクチャー + VIS (Visual Instruction Set) version 2.0 + UltraSPARC-III 拡張機能用のオブジェクトコードを生成します。
sparcvis3	**SPARC VIS version 3** の **SPARC-V9 ISA** 用にコンパイルします。 SPARC-V9 命令セット、VIS (Visual Instruction Set) version 1.0 を含む UltraSPARC 拡張機能、VIS (Visual Instruction Set) version 2.0、積和演算 (FMA) 命令、および VIS (Visual Instruction Set) version 3.0 を含む UltraSPARC-III 拡張機能の命令をコンパイラが使用できるようになります。

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

-xarch=	意味 **(SPARC)**
sparcfmaf	**SPARC-V9 ISA** の **sparcfmaf** バージョン用のコンパイルを行います。 SPARC-V9 命令セット、VIS (Visual Instruction Set) version 1.0 を含む UltraSPARC 拡張機能、VIS (Visual Instruction Set) version 2.0 を含む UltraSPARC-III 拡張機能、および浮動小数点積和演算用の SPARC64 VI 拡張機能の命令をコンパイラが使用できるようになります。 コンパイラが自動的に積和演算命令を使用する機会を見つけられるようにするには、**-xarch=sparcfmaf** および **-fma=fused** と最適化レベルを組み合わせて使用する必要があります。
sparcima	sparcima 版の SPARC-V9 ISA 用にコンパイルします。コンパイラが、SPARC-V9 命令セットに加えて、VIS (Visual Instruction Set) Version 1.0 を含む UltraSPARC 拡張機能、VIS (Visual Instruction Set) Version 2.0 を含む UltraSPARC-III 拡張機能、浮動小数点の積和演算用の SPARC64 VI 拡張機能、整数の積和演算用の SPARC64 VII 拡張機能からの命令を使用できるようにします。
v9	**-m64 -xarch=sparc** と同義です。 64 ビットのメモリーモデルを取得するために **-xarch=v9** を使用している従来のメイクファイルおよびスクリプトでは、**-m64** のみを使用する必要があります。
v9a	**-m64 -xarch=sparcvis** と同義です。旧バージョンとの互換を提供します。
v9b	**-m64 -xarch=sparcvis2** と同義です。旧バージョンとの互換を提供します。

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を説明します。x86 で **-xarch** が指定されなかった場合のデフォルトは **generic** です (または **–m64** が指定された場合は **generic64**)。

表 3–14 x86 プラットフォーム上の **-xarch** の値

-xarch=	意味 **(x86)**
generic	たいていの 32 ビット x86 プラットフォームで良好なパフォーマンスを得るために、コンパイルを行います。これはデフォルトで、**-xarch=pentium_pro** と同義です。
generic64	たいていの 64 ビット x86 プラットフォームで良好なパフォーマンスを得るために、コンパイルを行います。**sse2** と同義です。
native	x86 アーキテクチャーで良好なパフォーマンスを得るために、コンパイルを行います。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ために、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、必要に応じて新しいリリースごとに調整される可能性があります。
native64	64 ビット x86 アーキテクチャーで良好なパフォーマンスを得るために、コンパイルを行います。

表 3–14　x86 プラットフォーム上の **-xarch** の値　　*(続き)*

-xarch=	意味 **(x86)**
386	命令セットを Intel 386/486 アーキテクチャーに限定します。
pentium_pro	命令セットを Pentium Pro アーキテクチャーに制限します。
pentium_proa	AMD 拡張機能 (3DNow!、3DNow! 拡張機能、および MMX 拡張機能) を 32 ビット Pentium Pro アーキテクチャーに追加します。
sse	**pentium_pro** に SSE 命令セットを追加します(次を参照)。
ssea	AMD 拡張機能 (3DNow!、3DNow! 拡張機能、および MMX 拡張機能) を 32 ビット SSE アーキテクチャーに追加します。
sse2	**pentium_pro** に SSE2 命令セットを追加します (次を参照)。
sse2a	AMD 拡張機能 (3DNow!、3DNow! 拡張機能、および MMX 拡張機能) を 32 ビット SSE2 アーキテクチャーに追加します。
sse3	SSE2 命令セットに SSE3 命令セットを追加します。
amd64	Solaris プラットフォームでは、**-m64 -xarch=sse2** と同義です。 64 ビットのメモリーモデルを取得するために **-xarch=amd64** を使用している従来のメイクファイルおよびスクリプトでは、**-m64** を使用する必要があります。
amd64a	Solaris プラットフォームでは、**–m64 –xarch=sse2a** と同義です。
sse3a	AMD 拡張命令 (3DNow! など) をSSE3 命令セットに追加します。
ssse3	SSE3 命令セットに SSSE3 命令を追加します。
sse4_1	SSSE3 命令セットに SSE4.1 命令を追加します。
sse4_2	SSE4.1 命令セットに SSE4.2 命令を追加します。
amdsse4a	AMD 命令セットに SSE4a 命令を追加します。

3.4.110.1　x86/x64 プラットフォームでの特別な注意

x86 Solaris プラットフォームでコンパイルを行う場合は、次の点が重要です。

- **-xarch** を **sse**、**sse2**、**sse2a**、または **sse3** 以降に設定してコンパイルしたプログラムは、これらの機能および拡張機能をサポートするプラットフォームで実行する必要があります。
- Pentium 4 互換プラットフォームの場合、Solaris 9 4/04 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 OS は SSE/SSE2 に対応していません。
- 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、同じ **-xarch** 設定で正しい起動ルーチンがリンクされるようにしてください。

- x86 の 80 バイト浮動小数点レジスタが原因で、x86 での演算結果が SPARC の結果と異なる場合があります。この差を最小にするには、 **-fstore** オプションを使用するか、ハードウェアが SSE2 をサポートしている場合は **-xarch=sse2** でコンパイルします。
- Solaris Studio 11 と Solaris 10 OS から、これらの特殊化された **-xarch** ハードウェアフラグを使用してコンパイルし、構築されたプログラムバイナリは、適切なプラットフォームで実行されることが確認されます。
- Solaris 10 以前のシステムでは確認が行われないため、これらのフラグを使用して構築したオブジェクトが適切なハードウェアに配備されることをユーザーが確認する必要があります。
- これらの **-xarch** オプションでコンパイルしたプログラムを、適切な機能または命令セット拡張に対応していないプラットフォームで実行すると、セグメント例外や明示的な警告メッセージなしの不正な結果が発生することがあります。
- 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、SSE2a、SSE3 の命令、および拡張機能を利用している **__asm()** アセンブラコードにも当てはまります。

3.4.111 –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、37 ページの「2.3.1.8 **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できます。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** は次と同義です。

-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。

3.4.112 -xautopar

-autopar と同義です。

3.4.113 –xbinopt={prepare | off}

(SPARC) コンパイル後の最適化用にバイナリを準備します。

コンパイル済みのバイナリファイルは、あとで **binopt**(1) を使用して、最適化、変換、分析できます。このオプションは、実行可能ファイルまたは共有オブジェクトを構築するときに使用できます。また、有効にするには、最適化レベルを **-O1** 以上にする必要があります。

このオプションを使用して構築すると、バイナリファイルのサイズが約 5% 増加します。

コンパイルとリンクを個別に実行する場合、コンパイルとリンクの両方で **-xbinopt** を指定する必要があります。

アプリケーションのすべてのソースコードを **-xbinopt** でコンパイルしなかった場合でも、次のように、プログラムバイナリを構築する最後のリンク手順で **-xbinopt** フラグを使用します。

```
example% f95 -0 program -xbinopt=prepare a.o b.o c.f95
```

-xbinopt を使用してコンパイルしたコードだけが、**binopt**(1) で最適化できます。

デフォルトは **-xbinopt=off** です。

3.4.114 -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 には次のいずれかを指定します。

- **generic**
- native
- *s1*/*l1*/*a1*[/*t1*]
- *s1*/*l1*/*a1*[/*t1*] **:***s2*/*l2* **/***a2*[/*t2*]
- *s1*/*l1*/*a1*[/*t1*] **:***s2*/*l2* **/***a2*[/*t2*] **:***s3*/*l3* **/***a3*[/*t3*]

si/*li* **/***ai*/*ti* の定義は次のとおりです。

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 *ti* レベル *i* でキャッシュを共有するハードウェアスレッドの数 (省略可能)

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。

表 3–15 **-xcache** の値　(続き)

値	意味
s1/l1 /a1[/t1}	レベル 1 のキャッシュ特性を定義します。
s1/l1 /a1[/t1}: *s2/l2/ a2[/t2}*	レベル 1 と 2 のキャッシュ特性を定義します。
s1/l1 /a1[/t1}: *s2/l2/ a2[/t2}*: *s3/l3/ a3[/t3}*	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** では、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

3.4.115 -xcheck=*keyword*

実行時の特別な検査を生成します。

キーワードには次のいずれかを指定します。

キーワード	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** セグメント例外が発生します
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
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、**STATIC** ローカル変数と **SAVE** ローカル変数、および **COMMON** ブロックの変数は初期化されません。 詳細は、『C ユーザーズガイド』を参照してください。
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。

3.4.116 -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。

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

-xchip=*c* は次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 プロセッサこれがデフォルトです。
native	このホストプラットフォーム
sparc64vi	SPARC64 VI プロセッサ
sparc64vii	SPARC64 VII プロセッサ
ultra	UltraSPARC プロセッサ
ultra2	UltraSPARC II プロセッサ
ultra2e	UltraSPARC IIe プロセッサ
ultra2i	UltraSPARC IIi プロセッサ
ultra3	UltraSPARC III プロセッサ
ultra3cu	UltraSPARC IIIcu プロセッサ
ultra3i	UltraSPARC IIIi プロセッサ
ultra4	UltraSPARC IV プロセッサ

表 3–16 **-xchip** でよく使われる SPARC プロセッサ名　*(続き)*

-xchip=	最適化の対象
ultra4plus	UltraSPARC IV+ プロセッサ
ultraT1	UltraSPARC T1 プロセッサ
ultraT2	UltraSPARC T2 プロセッサ
ultraT2plus	UltraSPARC T2+ プロセッサ
ultraT3	UltraSPARC T3 プロセッサ

x86 プラットフォーム: **-xchip** 値
は、**pentium**、**pentium_pro**、**pentium3**、**pentium4**、**generic**、**opteron**、**core2**、**penryn**、**nehalem**、**amdfam10**、および **native** のいずれかです。

3.4.117 -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 の値は次のとおりです。

キーワード	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、および bss を合計したサイズは 2**32 バイトに制限されます。次の 32 ビットのプラットフォームでのデフォルトです。
abs44	44 ビットの絶対アドレスを生成します。コード、データ、および bss を合計したサイズは 2**44 バイトに制限されます。次の 64 ビットのプラットフォームだけで使用できます。
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。
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。

32 ビットのプラットフォームの場合は、**–xcode=abs32**です。64 ビットのプラットフォームの場合は、**–xcode=abs44** です。

3.4.117.1 位置独立コード

実行時のパフォーマンスを向上するために動的共有ライブラリを作成するときには、**-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。頻繁にアクセスされる共有オブジェクトの項目は、**-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行われる多数の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。

こうした影響があるとしても、**-xcode=pic13** または **-xcode=pic32** を使用すると、ライブラリコードを共有できるため、必要となるシステムメモリーを大幅に減らすことができます。**-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** のどちらを使用するか決定するときは、**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** の使用方法の決定に際しては、次のガイドラインに従ってください。

- 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** から開始し、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。

3.4.118 –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** 指令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、製品版の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** と同等です。

3.4.119 –xcrossfile[={ 1|0}]

ソースファイル間での最適化とインライン化を有効にします。(廃止)

このオプションは廃止され、使用できません。このオプションは、**-xipo** と同義です。代わりに **xipo** を使用してください。

3.4.120 –xdebugformat={dwarf |stabs}

Solaris 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。このリリースのデフォルト設定は、**-xdebugformat=dwarf** 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=dwarf 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=dwarf** 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。また、この情報の一部の形式はこのオプションで制御されます。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。

コンパイルされたオブジェクトまたは実行可能ファイルのデバッグ情報の形式を判断するには、**dwarfdump**(1) コマンドを使用します。

3.4.121 -xdepend

-depend と同義です。

3.4.122 -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、およびリンカーを使用して、コアイメージで関数 (副プログラム) の並べ替えを再度実行できます。**-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。そのあと、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。**f95 –M** *path* オプションも、リンカーに通常ファイルを渡します。**f95 –M***path* オプションの説明を参照してください。

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。その他の場合の並べ替えでは、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ない場合があります。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。

3.4.123 -xfilebyteorder= *options*

リトルエンディアン式プラットフォームとビッグエンディアン式プラットフォーム間のファイルの共有をサポートします。

このフラグは、書式なし入出力ファイル内のデータのバイト順序とバイト列を特定します。*options* には、次のフラグを任意に組み合わせたものを指定しますが、少なくとも 1 つのフラグを指定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、および 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 派生型に適用されます。

最大バイト列が *max_align* の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 ファイルを表すのに対し、**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** が 32 ビット SPARC ファイルを表すのに対し、**big16** は 64 ビット SPARC 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。次は、「ネイティブ」とみなされます。

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	**big8**
64 ビット SPARC	**big16**
32 ビット x86	**little4**
64 ビット x86	**little16**

spec には、コンマ区切りのリストで次を指定します。

%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、その他の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。

3.4.123.1 次に例を示します。

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

3.4.123.2 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は、常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、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ていないファイルは、ネイティブファイルとみなされます。たとえば、**-xfilebyteorder=little4:zork.out** を付けて **zork.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、ほか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 **m64** 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。(ただし、Solaris Studio 12 Update 1 のリリース以降では、**double:128** は x64 プロセッサで受け入れられます)。

VAX 構造体の成分の整列を変更するオプション (**–vax=struct_align** など) または構造型の成分の整列を変更するオプション (**–aligncommon** や **–dalign** など) をあるプラットフォームで使用する場合、同じ整列オプションを、その整列オプションの影響を受ける内容を持つ同じ書式なしデータファイルを共有する他のプラットフォームでも使用する必要があります。

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。

3.4.124 –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は、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。

次に例を示します。

```
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     <-- should be 7
demo% f95 -o has1 -xhasc=no hasc.f
demo% has1
 string length =   7   <-- now correct length for  s
```

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

このフラグは、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。

3.4.125 –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。

3.4.126 -xhwcprof[={enable | disable}]

(SPARC) コンパイラのデータ空間プロファイリングのサポートを有効にします。

-xhwcprof を有効にすると、コンパイラは、プロファイル対象のロード命令およびストア命令と、それらが参照するデータ型および構造体メンバーをツールが関連付けるのに役立つ情報を、**-g** で生成されたシンボル情報と組み合わせて生成します。プロファイルデータは、ターゲットの命令空間ではなく、データ空間と関連付けられ、命令のプロファイリングだけでは入手の容易でない、動作に関する詳細情報が提供されます。

指定した一連のオブジェクトファイルは、**-xhwcprof** を使用してコンパイルできます。ただし、このオプションがもっとも役立つのは、アプリケーション内のすべてのオブジェクトファイルに適用したときです。このオプションによって、アプリ

ケーションのオブジェクトファイルに分散しているすべてのメモリー参照を識別したり、関連付けたりするカバレージが提供されます。

コンパイルとリンクを別々に行う場合は、**-xhwcprof** をリンク時にも使用してください。

-xhwcprof=enable または **-xhwcprof=disable** のインスタンスは、同じコマンド行にある以前の **-xhwcprof** のインスタンスをすべて無効にします。

-xhwcprof はデフォルトでは無効です。引数を指定せずに **-xhwcprof** と指定することは、**-xhwcprof=enable** と指定することと同じです。

-xhwcprof を使用する場合は最適化を有効にし、デバッグのデータ形式を dwarf (**-xdebugformat=dwarf**) に設定しておく必要があります。これは、この Solaris Studio のリリースのデフォルトです。

-xhwcprof と **-g** を組み合わせて使用すると、コンパイラに必要な一時ファイル記憶領域は、**-xhwcprof** と **-g** を単独で指定することによって増える量の合計を超えて大きくなります。

次のコマンドは **example.f** をコンパイルし、ハードウェアカウンタによるプロファイリングのサポートを指定し、DWARF シンボルを使用してデータ型と構造体メンバーのシンボリック解析を指定します。

```
f95 -c -O -xhwcprof -g  example.f
```

ハードウェアカウンタによるプロファイリングの詳細は、『Solaris Studio パフォーマンスアナライザ』を参照してください。

3.4.127 –xia[={ widestneed|strict}]

(Solaris) 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 で拡張された区間演算の詳細は、『区画演算プログラミングリファレンス』に記載されています。124 ページの「3.4.130 **–xinterval**[={ *widestneed*|*strict*|*no*}]」も参照してください。

-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**

3.4.128 -xinline= *list*

-inline と同義です。

3.4.129 –xinstrument=[%no]datarace

スレッドアナライザで分析するためにプログラムをコンパイルして計測するには、このオプションを指定します。

スレッドアナライザについての詳細は、**tha**(1) を参照してください。

このオプションを使用してコンパイルすることにより、パフォーマンスアナライザを使用して **collect -r races** オプションを付けて計測されるプログラムを実行し、データ競合検出実験を作成できます。計測されたコードをスタンドアロンで実行できますが、低速で実行されます。

この機能を無効にするには、**-xinstrument=no%datarace** と指定します。これはデフォルト値です。

-xinstrument には、引数を 1 つ指定する必要があります。

コンパイルとリンクを別々に行う場合は、両方の手順で **-xinstrument=datarace** を指定してください。

このオプションは、プリプロセッサトークン **__THA_NOTIFY** を定義します。**#ifdef __THA_NOTIFY** を指定して、**libtha**(3) ルーチンへの呼び出しを保護することができます。

このオプションでは、**-g** も設定します。

3.4.130 –xinterval[={ *widestneed*|*strict*|*no*}]

(Solaris) 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 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。123 ページの「3.4.127 **–xia**[={ **widestneed**|**strict**}]」 も参照してください。

3.4.131 –xipo[={ 0|1|2}]

相互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** が値なしで指定された場合は、**-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るべきではありません。

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。ただし、システムライブラリに対してこのような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
```

オプティマイザは 3 つのすべてのソースファイル間でファイル間のインライン化を実行します。これは最終的なリンク手順で実行されるため、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** の間で相互手続きの最適化が実行されますが、**main.f** または **four.f** と、**mylib.a** のルーチンの間では相互手続きの最適化が実行されません。最初のコンパイルは未定義のシンボルに関する警告を生成する場合がありますが、相互手続きの最適化は、コンパイル手順でありしかもリンク手順であるために実行されます。

-xipo に関するそのほかの重要な情報を次に示します。

- 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** を付けてコンパイルされた実行可能プログラムを、並列 make ツールを使用して構築する場合、並列実行するリンク手順に共通のオブジェクトファイルが存在すると、問題が発生する可能性があります。リンク処理の前に、最適化されるオブジェクトファイルのコピーをリンク手順ごとに作成してください。
- **-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、ファイルを介して最適化を実行する際に必要な情報を追加するため、非常に大きなオブジェクトファイルを生成します。ただし、この補足情報は最終的な実行可能バイナリ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相互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、**-xipo=2** でコンパイルしないでください。仮定 (2) が当てはまらない場合は、**-xipo=1** および **-xipo=2** でコンパイルしないでください。

一例として、独自のソースバージョンの **malloc()** で関数 **-xipo=2** を置き換えるケースを考えてみましょう。その独自のコードとリンクされる **malloc()** を参照するライブラリのすべての関数も **-xipo=2** でコンパイルする必要があり、リンク手順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()** が実行時に割り込み処理される場合、**foo()** または **bar()** いずれのソースファイルも **-xipo=1** または **-xipo=2** でコンパイルしません。**foo()** が **bar()** 内にインライン化され、**-xipo** でコンパイルした場合に不正な結果になる可能性があります。

3.4.132 –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。

none	アーカイブファイルを処理しません。コンパイラは、**-xipo** を使用してコンパイルされたオブジェクトファイル、およびリンク時にアーカイブライブラリから抽出されたオブジェクトファイルに対して、モジュール相互のインライン化または最適化を適用しません。これを適用するには、**-xipo** と、**-xipo_archive=readonly** または **-xipo_archive=writeback** のいずれかをリンク時に指定する必要があります。

readonly	実行可能ファイルを生成する前に、アーカイブライブラリ (**.a**) に存在する **-xipo** でコンパイルしたオブジェクト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。 **-xipo_archive=readonly** オプションによって、リンク時に指定されたアーカイブライブラリのオブジェクトファイルに対する、モジュール相互のインライン化および内部手続きデータフロー解析が有効になります。ただし、モジュール間のインライン化によってほかのモジュールに挿入されたコード以外のアーカイブライブラリのコードに対する、モジュール間の最適化は有効になりません。 アーカイブライブラリ内のコードにモジュール相互の最適化を適用するには、**-xipo_archive=writeback** を指定する必要があります。これを行うと、コードが抽出されたアーカイブライブラリの内容が変更されることがあります。
writeback	実行可能ファイルを生成する前に、アーカイブライブラリ (**.a**) に存在する **-xipo** でコンパイルしたオブジェクト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。 アーカイブライブラリの共通セットを使用する並列リンクでは、最適化されるアーカイブライブラリの独自のコピーを、各リンクでリンク前に作成する必要があります。

-xipo_archive 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。

3.4.133 –xivdep[= *p*]

!DIR$ IVDEP 指令の解釈を無効または設定します。

IVDEP 指令は、ループ内で検出された一部またはすべての配列参照のループがもたらす依存関係を無視し、特にほかの方法では実行できないマイクロベクトル化、配布、ソフトウェアパイプラインなどのさまざまなループの最適化を実行するように、コンパイラに指示します。これは、依存関係が重要ではない、または依存関係が実際に発生しないことをユーザーが把握している状況で使用されます。

!DIR$ IVDEP 指令の解釈は、**-xivdep** オプションの値に応じて異なります。*p* の次の値は、次のように解釈されます。

loop — 推測されるループがもたらすベクトル依存関係を無視します。
loop_any — すべてのループがもたらすベクトル依存関係を無視します。
back — 推測される後方へのループがもたらすベクトル依存関係を無視します。
back_any — すべての後方へのループがもたらすベクトル依存関係を無視します。
none — 依存関係を無視しません (**IVDEP** 指令を無効にします)。

これらの解釈は、ほかのベンダーの **IVDEP** 指令の解釈と互換性があります。

–xivdep が指定されていない場合、および引数を使用せずに **–xivdep** が指定されている場合のデフォルトはどちらも **–xivdep=loop** です。これは、**!DIR$ IVDEP** 指令がデフォルトで有効であることを示します。

詳細は、39 ページの「2.3.3 **IVDEP** 指令」を参照してください。

3.4.134 -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、さまざまなファイルをコンパイルするために呼び出せるコードジェネレータインスタンスの最大数として *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、使用できる仮想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
```

3.4.135 –xkeepframe[=[%all,%none,*name*,no% *name*]]

指定した機能 (*name*) のスタック関連の最適化を禁止します。

%all - すべてのコードのスタック関連の最適化を禁止します。

%none - すべてのコードのスタック関連の最適化を許可します。

このオプションがコマンド行で指定されていないと、コンパイラはデフォルトの **-xkeepframe=%none** を使用します。このオプションが値なしで指定されると、コンパイラは **-xkeepframe=%all** を使用します。

3.4.136 -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、Sun 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、Sun 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 BLAS3 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 BLAS3 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 組み込みの明示的な **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。組み込み関数の名前のリストは、『Fortran ライブラリ・リファレンス』を参照してください。

3.4.137 –xl

(廃止) この旧バージョンの **f77** オプションはサポートされていません。現在の Fortran コンパイラの同等オプションとして次を使用してください。**-f77=%all,no%backslash -vax=$all,no%debug**

3.4.138 -xlang=f77

(SPARC)旧バージョンの **f77** コンパイラで作成されたオブジェクトと互換性のある実行時ライブラリを伴うリンクを作成します。

f95 -xlang=f77 は、**f77compat** ライブラリを伴うリンクを暗黙に定義し、**f95** 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。

3.4.139 –xld

(廃止) この (**f77**) オプションはサポートされていません。現在の Fortran コンパイラの同等オプションとして次を使用してください。**-f77=%all,no%backslash -vax=$all,no%debug**

3.4.140 -xlibmil

-libmil と同義です。

3.4.141 -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に指定する順序は重要ではありません。

3.4.142 -xlic_lib=sunperf

Sun Performance Library とリンクします。

次に例を示します。

```
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あとに指定します。

このオプションを使用して、Sun Performance Library とリンクさせる必要があります (『Sun Performance Library User's Guide 』を参照)。

3.4.143 -xlicinfo

(廃止) コンパイラによって無視されます。

3.4.144 –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です。これがデフォルトです。
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とも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する必要があります。

```
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。前述の例では、オブジェクトバイナリが暗黙的に指定された 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基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の詳細は、**-xprofile** オプション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ため、実行可能ファイルのサイズが増えます。

3.4.145 -xloopinfo

-loopinfo と同義です。

3.4.146 –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。

このオプションを指定すると、**!$PRAGMA SUN OPT=***n* 指令がソース入力に表示されている場合に有効になります。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。35 ページの「2.3.1.5 **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。

3.4.147 –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。

コンパイル時に境界整列を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、このようなあいまい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 64 ビットのプラットフォームでは、4 バイト以下の境界整列にだけ SIGBUS 信号を発生させます。それ以外ではアクセスを解釈して実行を継続します。その他のプラットフォームでは、**f** は **i** と等価です。

-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。

- 32 ビットのプラットフォームの場合は、**8i**
- C および C++ の 64 ビットのプラットフォームの場合は、**8s**
- Fortran の 64 ビットのプラットフォームの場合は、**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ことはありません。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。

また、**b** の値に **i** または **f** を指定してコンパイルしたオブジェクトファイルにリンクする場合は、必ず、**-xmemalign** を指定してください。

-dalign オプションはマクロです。

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

-aligncommon=1 を **-xmemalign** とともに使用しないでください。これらの指令は衝突するため、同じプラットフォームや構成でセグメント例外が発生する場合があります。

詳細は、55 ページの「3.4.1 **–aligncommon**[={ **1**|**2**|**4**|**8**| **16**}]」を参照してください。

3.4.148 –xmodel=[small | kernel | medium]

(x86) Solaris x64 プラットフォームで共有オブジェクトのデータアドレスモデルを指定します。

-xmodel オプションを使用すると、コンパイラで Solaris x64 プラットフォーム用の 64 ビット共有オブジェクトを作成できます。このオプションは、そのようなオブジェクトのコンパイル時にのみ指定してください。

このオプションは、64 ビットに対応した x86 プラットフォーム ("x64") で **-m64** が指定されている場合にのみ有効です。

small このオプションは、実行されるコードの仮想アドレスがリンク時にわかっていて、すべてのシンボルが 0 ～ 2^{31} - 2^{24} - 1 の範囲の仮想アドレスに配置されることがわかっているスモールモデルのコードを生成します。

kernel すべてのシンボルが 2^{64} - 2^{31} ～ 2^{64} - 2^{24} の範囲で定義されるカーネルモデルのコードを生成します。

medium データセクションへのシンボリック参照の範囲に関する前提がないミディアムモデルのコードを生成します。テキストセクションのサイズとアドレスは、スモールコードモデルの場合と同じように制限されます。静的データのサイズが大きいアプリケーションでは、**–m64** を指定してコンパイルする際に **–xmodel=medium** の指定が必要な場合があります。

-xmodel を指定しない場合、コンパイラは **-xmodel=small** とみなします。引数を指定せずに **-xmodel** を指定すると、エラーになります。

この範囲内でオブジェクトにアクセスすることが確実であれば、必ずしもすべてのルーチンをこのオプションでコンパイルする必要はありません。

3.4.149 -xnolib

-nolib と同義です。

3.4.150 -xnolibmil

-nolibmil と同義です。

3.4.151 -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** と組み合わせて使用します。

```
f95 -fast -xnolibmopt …
```

3.4.152 -x0*n*

-O*n* と同義です。

3.4.153 –xopenmp[={ parallel|noopt|none}]

Fortran の OpenMP Version 3.0 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、そのプラグマに基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、警告を出力します。
- プリプロセッサトークン **_OPENMP** を定義します。
- **-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、そのプラグマに基づいてプログラムが並列化されます。
- 最適化のレベルが **-xO3** より低い場合でも、コンパイラは最適化のレベルを上げません。最適化レベルを **-xO2 -xopenmp=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。**-xopenmp=noopt** で最適化レベルを指定しなかった場合、OpenMP プラグマが認識され、その結果プログラムが並列化されますが、最適化は行われません。
- プリプロセッサトークン **_OPENMP** を定義します。
- **-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コンパイラのデフォルトです。

サブオプションキーワードなしで指定した **-xopenmp** は、**-xopenmp=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。これにより、プログラムで作成できる最大スレッド数を実行時システムに設定します。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で利用可能な仮想プロセッサ数を設定します。

入れ子並列を有効にするには、**OMP_NESTED** 環境変数を **TRUE** に設定する必要があります。入れ子並列は、デフォルトでは無効です。入れ子並列についての詳細は、Solaris Studio の『OpenMP API ユーザーズガイド』を参照してください。

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。現在の Solaris Studio リリースでは、この値は OpenMP Version 3.0 を表す 200830 です。

コンパイルとリンクを分けて行う場合は、リンク時にも **-xopenmp** 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。

3.4.154 -xpad

-pad と同義です。

3.4.155 -xpagesize=*size*

スタックとヒープ用に優先ページサイズを設定します。

SPARC プラットフォームでは、*size* 値には次のいずれかを指定します。

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

x86 プラットフォームでは、*size* 値には次のいずれかを指定します。

4K 2M 4M 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、アーキテクチャーと Solaris 環境に依存します。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を指定する必要があります。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* と **-xpagesize_stack=***size* を組み合わせたマクロです。これら 2 つのオプションは **-xpagesize** と同じ次の引数を受け入れます。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。

3.4.156 -xpagesize_heap= *size*

ヒープ用に優先ページサイズを設定します。

size の値は、**-xpagesize** の説明と同じです。

詳細は、**-xpagesize** を参照してください。

3.4.157 -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 の値は、**-xpagesize** の説明と同じです。

詳細は、**-xpagesize** を参照してください。

3.4.158 –xpec[={ yes|no}]

PEC (Portable Executable Code) バイナリを生成します。

PEC バイナリは、自動チューニングシステム（Automatic Tuning System、ATS）とともに使用することができます。ATS についての詳細は、`http://cooltools.sunsource.net/ats/index.html` を参照してください。

-xpec を使用して構築したバイナリは、-xpec を使用しないで構築したバイナリの通常 5 〜 10 倍の大きさになります。デフォルトは **-xpec=no** です。

引数がない場合、**-xpec** は **-xpec=yes** と同義です。

3.4.159 -xpg

-pg と同義です。

3.4.160 –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。このプリプロセッサは Fortran 用に適しています。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** と指定します。

3.4.161 –xprefetch[= *a*[,*a*]]

先読みをサポートするアーキテクチャーで先読み命令を有効にします。

Fortran **PREFETCH** 指令の詳細は、36 ページの「2.3.1.7 **PREFETCH** 指令」を参照してください。

a には次のいずれかを指定します。

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* (**SPARC**) 指定の係数により、コンパイラの先読みからロード、および先読みからストアまでの応答時間を調整します。係数には必ず正の浮動小数点または整数を指定します。

大型の SPARC 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* を使用すると役立つことがあります。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かぎりません。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 よりも大きい係数を使用します。.5 と 2.0 の間の値は、おそらく最高のパフォーマンスを提供します。

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 に近い係数の値から始め、アプリケーションに対してパフォーマンステストを実施します。そのあと、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。

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ーでパフォーマンスが向上します。

3.4.161.1 デフォルト

-xprefetch を指定しないと、**-xprefetch=auto,explicit** が使用されます。

-xprefetch だけを指定すると、**-xprefetch=auto,explicit** が使用されます。

-xprefetch または **-xprefetch=yes** などで自動先読みを有効にしても、応答時間係数を指定しないと、**-xprefetch=latx:1.0** が使用されます。

3.4.161.2 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。

!$PRAGMA SUN_PREFETCH_READ_ONCE (*name* **)**
!$PRAGMA SUN_PREFETCH_READ_MANY (*name* **)**
!$PRAGMA SUN_PREFETCH_WRITE_ONCE (*name* **)**
!$PRAGMA SUN_PREFETCH_WRITE_MANY (*name* **)**

-xchip 設定は、想定した応答時間、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、SPARC プロセッサで自動先読み (**auto**) が実行可能な場合のみ有効です。

3.4.161.3 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、広範囲なマシンやアプリケーション間で最適なパフォーマンスを得るために先読み機構を調整しますが、**-xprefetch=latx:***factor* は、パフォーマンステストで明らかに利点があることが確認された場合にかぎり使用してください。使用先読み応答時間は、リリースごとに変わる可能性があります。したがって、別のリリースに切り替えたら、その都度応答時間係数の影響を再テストすることを推奨します。

3.4.162 -xprefetch_auto_type=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。接頭辞 **no%** を付けると、この宣言を無効にできます。

デフォルトは **-xprefetch_auto_type=no%indirect_array_access** です。

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。

3.4.163 –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

-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の機会を生成します。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。

先読みレベル 2 および 3 は、旧バージョンの SPARC または x86 プラットフォームでは無効な場合があります。

3.4.164 –xprofile=*p*

プロファイルのデータを収集したり、プロファイルを使用して最適化したりします。

p には、**collect**[**:***profdir*]、**use**[**:***profdir*]、または **tcov**[**:***profdir*] を指定する必要があります。

このオプションを指定すると、実行頻度のデータが収集されて実行中に保存されます。このデータを以降の実行で使用すると、パフォーマンスを向上させることができます。プロファイルの収集は、マルチスレッド対応のアプリケーションにとって安全です。すなわち、独自のマルチタスク (**-mt**) を実行するプログラムをプロファイリングすることで、正確な結果が得られます。このオプションは、**-xO2** またはそれ以上のレベルの最適化を指定するときにのみ有効になります。コンパイルとリンクを別々の手順で実行する場合は、リンク手順とコンパイル手順の両方で同じ **-xprofile** オプションを指定する必要があります。

collect[**:** *profdir*] 実行頻度のデータを集めて保存します。のちに **-xprofile=use** を指定した場合にオプティマイザがこれを使用します。コンパイラによって文の実行頻度を測定するためのコードが生成されます。

-xMerge、**-ztext**、および **-xprofile=collect** を一緒に使用しないでください。**-xMerge** を指定すると、静的に初期化されたデータを読み取り専用記憶領域に強制的に配置します。 **-ztext** を指定すると、位置に依存するシンボルを読み取り専用記憶領域内で再配置することを禁止します。**-xprofile=collect** を指定すると、書き込み可能記憶領域内で、静的に初期化された、位置に依存するシンボルの再配置を生成します。

プロファイルディレクトリ名として *profdir* を指定すると、この名前が、プロファイル化されたオブジェクトコードを含むプログラムまたは共有ライブラリの実行時にプロファイルデータが保存されるディレクトリのパス名になります。*profdir* パス名が絶対パスではない場合、プログラムがオプション -**xprofile=use:***profdir* でコンパイルされるときの現在の作業用ディレクトリの相対パスとみなされます。プロファイルディレクトリ名を指定しないと、プロファイルデータは、*program*`.profile` という名前のディレクトリに保存されます (*program* はプロファイル化されたプロセスのメインプログラムのベース名)。

例 [1]: プログラムが構築されたディレクトリと同じディレクトリにある myprof.profile ディレクトリでプロファイルデータを収集して使用するには、次のように指定します。

```
demo: f95 -xprofile=collect:myprof.profile -xO5 prog.f95 -o prog
demo: ./prog
demo: f95 -xprofile=use:myprof.profile    -xO5 prog.f95 -o prog
```

例 [2]: ディレクトリ `/bench/myprof.profile` にプロファイルデータを収集し、収集したプロファイルデータをあとから最適化レベル **-xO5** のフィードバックコンパイルで使用するには、次のように指定します。

```
demo: f95 -xprofile=collect:/bench/myprof.profile
\    -xO5 prog.f95 -o prog
...run prog from multiple locations..
demo: f95 -xprofile=use:/bench/myprof.profile
\    -xO5 prog.f95 -o prog
```

環境変数の **SUN_PROFDATA** と **SUN_PROFDATA_DIR** を設定して、**-xprofile=collect** を指定してコンパイルされたプログラムがどこにプロファイルデータを入れるかを制御できますこれらの環境変数を設定すると、**-xprofile=collect** データが **$SUN_PROFDATA_DIR/$SUN_PROFDATA** に書き込まれます。

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を **tcov**(1) マニュアルページの説明どおり、同様に制御指定します。これらの環境変数をまだ設定して

いない場合、プロファイルデータは現在のディレクトリの *profdir* **.profile** に書き込まれます (*profdir* は実行ファイルの名前または **-xprofile=collect:** *profdir* フラグで指定された名前)。*profdir* が **.profile** ですでに終了している場合、**-xprofile** では、**profile** が *profdir* に追加されません。プログラムを複数回実行すると、実行頻度データは *profdir***.profile** ディレクトリ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。

別々の手順でコンパイルしてリンクする場合は、**-xprofile=collect** を指定してコンパイルしたオブジェクトファイルは、リンクでも必ず **-xprofile=collect** を指定してください。

use[**:** *profdir*]

-xprofile=collect[**:** *profdir*] でコンパイルされたコードから収集された実行頻度データを使用して、プロファイル化されたコードが実行されたときに実行された作業用の最適化が行えます。*profdir* は、**-xprofile=collect**[**:** *profdir*] でコンパイルされたプログラムを実行して収集されたプロファイルデータを含むディレクトリのパス名です。

profdir パス名は省略可能です。*profdir* が指定されていない場合、実行可能バイナリの名前が使用されます。**-o** が指定されていない場合、`a.out` が使用されます。 *profdir* が指定されていない場合、コンパイラは、*profdir* **.profile/feedback**、または **a.out.profile/feedback** を探します。次に例を示します。

```
demo: f95 -xprofile=collect    -o myexe prog.f95
demo: f95 -xprofile=use:myexe -xO5 -o myexe    prog.f95
```

-xprofile=collect オプションを付けてコンパイルしたときに生成され、プログラムの前の実行で作成されたフィードバック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。

-xprofile オプションを除き、ソースファイルおよびコンパイラのほかのオプションは、フィードバックファイルを生成したコンパイル済みプログラムのコンパイルに使用したものと完全に同一のものを指定する必要があります。同じバージョンのコンパイラは、収集構築と使用構築の両方に使用する必要があります。

-xprofile=collect:*profdir* を付けてコンパイルした場合は、**-xprofile=use:** *profdir* のコンパイルの最適化に同じプロファイルディレクトリ名 *profdir* を使用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。

tcov[**:** *profdir*]

tcov(1) を使用する基本のブロックカバレッジ分析用の命令オブジェクトファイル。

オプションの *profdir* 引数を指定すると、コンパイラは指定された場所にプロファイルディレクトリを作成します。プロファイルディレクトリに保存されたデータは、**tcov**(1) または **-xprofile=use:***profdir* を付けたコンパイラで使用できます。オプションの *profdir* パス名を省略すると、プロファイル化されたプログラムの実行時にプロファイルディレクトリが作成されます。プロファイルディレクトリに保存されたデータは、**tcov**(1) でのみ使用できます。プロファイルディレクトリの場所は、環境変数 **SUN_PROFDATA** および **SUN_PROFDATA_DIR** を使用して指定できます。

profdir で指定された場所が絶対パス名ではない場合、コンパイル時に、現在のオブジェクトファイルが作成されたディレクトリの相対パスとみなされます。いずれかのオブジェクトファイルに *profdir* を指定する場合は、同じプログラムのすべてのオブジェクトファイルに対して同じ場所を指定する必要があります。場所が *profdir* で指定されているディレクトリには、プロファイル化されたプログラムを実行するときにすべてのマシンからアクセスできる必要があります。プロファイルディレクトリはその内容が必要なくなるまで削除できません。コンパイラでプロファイルディレクトリに保存されたデータは、再コンパイルする以外復元できません。

例 [1]: 1 つ以上のプログラムのオブジェクトファイルが **-xprofile=tcov:/test/profdata** でコンパイルされる場合、**/test/profdata.profile** という名前のディレクトリがコンパイラによって作成されて、プロファイル化されたオブジェクトファイルを表すデータの保存に使用されます。実行時に同じディレクトリを使用して、プロファイル化されたオブジェクトファイルに関連付けられた実行データを保存できます。

例 [2]: **myprog** という名前のプログラムが **-xprofile=tcov** でコンパイルされ、ディレクトリ `/home/joe` で実行されると、実行時にディレクトリ **/home/joe/myprog.profile** が作成されて、実行時プロファイルデータの保存に使用されます。

3.4.165 –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。

3.4.166 –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、次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。

3.4.167 -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 より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、そのあとにルーチン 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 より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。

3.4.168 -xreduction

-reduction と同義です。

3.4.169 -xregs=*r*

生成されたコードのレジスタの使用法を指定します。

r には、`appl`、`float`、`frameptr` サブオプションのいずれか 1 つ以上をコンマで区切って指定します。

サブオプションの前に `no%` を付けるとそのサブオプションは無効になります。

–xregs サブオプションは、特定のハードウェアプラットフォームでしか使用できません。

例: `-xregs=appl,no%float`

表 3–17 `-xregs` サブオプション

値	意味
`appl`	(SPARC) 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してコードを生成することを許可します。アプリケーションレジスタは次のとおりです。 g2、g3、g4 (32 ビットプラットフォーム) g2、g3 (64 ビットプラットフォーム) すべてのシステムソフトウェアおよびライブラリは、`-xregs=no%appl` を指定してコンパイルすることをお勧めします。システムソフトウェア (共有ライブラリを含む) は、アプリケーション用のレジスタの値を保持する必要があります。これらの値は、コンパイルシステムによって制御されるもので、アプリケーション全体で整合性が確保されている必要があります。 SPARC ABI では、これらのレジスタはアプリケーションレジスタと記述されています。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。
`float`	(SPARC) 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してコードを生成することを許可します。浮動小数点値を使用する場合は、このオプションとは関係なくこれらのレジスタを使用します。浮動小数点レジスタに対するすべての参照をコードから排除する場合は、`-xregs=no%float` を使用するとともに、決して浮動小数点型をコードで使わないようにする必要があります。

表 3–17 -xregs サブオプション (続き)

値	意味
frameptr	(x86) フレームポインタレジスタ (IA32 の場合 %ebp、AMD64 の場合 %rbp) を汎用レジスタとして使用することを許可します。 デフォルトは -xregs=no%frameptr です。 -xregs=framptr を使用すると、コンパイラは浮動小数点レジスタを自由に使用できるので、プログラムのパフォーマンスが向上します。ただし、この結果としてデバッガおよびパフォーマンス測定ツールの一部の機能が制限される場合があります。スタックトレース、デバッガ、およびパフォーマンスアナライザは、–xregs=frameptr を使用してコンパイルされた機能についてレポートできません。 C、Fortran、C++ が混在しているコードで、C または Fortran 関数から直接または間接的に呼び出された C++ 関数が例外をスローする可能性がある場合、このコードは –xregs=frameptr でコンパイルできません。このような言語が混在するソースコードを –fast でコンパイルする場合は、コマンド行の –fast オプションのあとに –xregs=no%frameptr を追加します。 64 ビットのプラットフォームで使用できる多くのレジスタでは、–xregs=frameptr でコンパイルすると、64 ビットコードよりも 32 ビットコードのパフォーマンスが向上する可能性が高くなります。 -xpg も指定されている場合、コンパイラは -xregs=frameptr を無視し、警告を表示します。

SPARC のデフォルトは -xregs=appl,float です。

x86 のデフォルトは -xregs=no%frameptr です。-fast の展開に含まれる場合は -xregs=frameptr です。

アプリケーションにリンクする共有ライブラリ用のコードは、-xregs=no%appl,float を指定してコンパイルすることをお勧めします。少なくとも、共有ライブラリとリンクするアプリケーションがこれらのレジスタの割り当てを認識するように、共有ライブラリがアプリケーションレジスタを使用する方法を明示的に示す必要があります。

たとえば、大局的な方法で (重要なデータ構造体を示すためにレジスタを使用するなど) レジスタを使用するアプリケーションは、ライブラリと確実にリンクするため、-xregs=no%appl なしでコンパイルされたコードを含むライブラリがアプリケーションレジスタをどのように使用するかを正確に特定する必要があります。

3.4.170 -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ファイル (**.o**) 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。

3.4.171 -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、最適化レベルの `-xO5` と、次のいずれかの値の `-xarch` を組み合わせた場合にだけ有効です。`-m32` と `-m64` の両方で `sparc`、`sparcvis`、`sparcvis2`、または `sparcvis3`。

注意 – アドレスの位置合わせが合わない、またはセグメンテーション侵害などの違反が発生した場合は違反のないロードはトラップを引き起こさないので、このオプションはこのような違反が起こる可能性のないプログラムでしか使用しないでください。ほとんどのプログラムではメモリーに関するトラップは起こらないので、大多数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。

3.4.172 -xsb

(廃止) **-sb** と同義です。

3.4.173 -xsbfast

(廃止) **-sbfast** と同義です。

3.4.174 -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。

3.4.175 -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 = のあと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) の正式な名前をコンパイラに指定した方がパフォーマンスが優れてい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** の指定で十分です。

-xtarget の実際の展開値は、リリースによって異なる可能性があります。コンパイラが使用する展開値は、**-dryrun** フラグを使用して判断できます。

```
demo% f95 -dryrun -xtarget=ultra4plus
###     command line files and options (expanded):
### -dryrun -xarch=sparcvis
    -xcache=64/32/4/1:2048/64/4/2:32768/64/4/2 -xchip=ultra4plus
```

特定の指定プラットフォームでの **–xtarget** 展開は、同じプラットフォームでの **–xtarget=native** 指定の展開と異なる場合があることに注意してください。

3.4.175.1 一般的なプラットフォームとネイティブプラットフォーム

native ホストプラットフォーム (32 ビット) で、パフォーマンスを最適化します。

–m32 –xarch=native –xchip=native –xcache=native に展開します。

native64 廃止。代わりに、**–xtarget=native –m64** を使用してください。

generic たいていの 32 ビットプラットフォームで最高のパフォーマンスが得られるようにします。

これがデフォルトで、次のように展開します。**-m32 -xarch=generic -xchip=generic -xcache=generic**

generic64 廃止。代わりに **-xtarget=generic -m64** を使用してください。

platform-name 指定したプラットフォームで、最高のパフォーマンスが得られるようにします。次に一覧で表示します。

3.4.175.2 SPARC プラットフォーム

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。

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
sparc64vi	**sparcfmaf**	**sparc64vi**	**128/64/2:5120/64/10**
sparc64vii	**sparcima**	**sparc64vii**	**64/64/2:5120/256/10**
ultra	**sparcvis**	**ultra**	**16/32/1:512/64/1**
ultra1/140	**sparcvis**	**ultra**	**16/32/1:512/64/1**
ultra1/170	**sparcvis**	**ultra**	**16/32/1:512/64/1**
ultra1/200	**sparcvis**	**ultra**	**16/32/1:512/64/1**
ultra2	**sparcvis**	**ultra2**	**16/32/1:512/64/1**
ultra2/1170	**sparcvis**	**ultra**	**16/32/1:512/64/1**
ultra2/1200	**sparcvis**	**ultra**	**16/32/1:1024/64/1**
ultra2/1300	**sparcvis**	**ultra2**	**16/32/1:2048/64/1**
ultra2/2170	**sparcvis**	**ultra**	**16/32/1:512/64/1**
ultra2/2200	**sparcvis**	**ultra**	**16/32/1:1024/64/1**
ultra2/2300	**sparcvis**	**ultra2**	**16/32/1:2048/64/1**
ultra2e	**sparcvis**	**ultra2e**	**16/32/1:256/64/4**
ultra2i	**sparcvis**	**ultra2i**	**16/32/1:512/64/1**
ultra3	**sparcvis2**	**ultra3**	**64/32/4:8192/512/1**
ultra3cu	**sparcvis2**	**ultra3cu**	**64/32/4:8192/512/2**
ultra3i	**sparcvis2**	**ultra3i**	**64/32/4:1024/64/4**
ultra4	**sparcvis2**	**ultra4**	**64/32/4:8192/128/2**
ultra4plus	**sparcvis2**	**ultra4plus**	**64/32/4/1:2048/64/4/2:32768/64/4/2**
ultraT1	**sparc**	**ultraT1**	**8/16/4/4:3072/64/12/32**

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
ultraT2	**sparcvis2**	**ultraT2**	**8/16/4:4096/64/16**
ultraT2plus	**sparcvis2**	**ultraT2plus**	**8/16/4:4096/64/16**
ultraT3	**sparcvis3**	**ultraT3**	**8/16/4:6144/64/24**

64 ビット対応のプラットフォームでの 64 ビット Solaris OS 向けのコンパイルは、**-m64** フラグで指示します。**-xtarget** を指定する場合は、次に示すように **-xtarget** フラグのあとに **-m64** を表示する必要があります。

```
-xtarget=ultra2   ...   -m64
```

この指定がないと、デフォルトの 32 ビットメモリーモデルが使用されます。

3.4.175.3 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**、**pentium**、**pentium_pro**、**pentium3**、**pentium4**、**woodcrest**、**penryn**、**nehalem**、**barcelona**、**opteron**

表 3–19　x86 プラットフォームでの -xtarget の値

-xtarget=	-xarch	-xchip	-xcache
generic	generic	generic	generic
opteron	sse2	opteron	64/64/2:1024/64/16
pentium	386	pentium	generic
pentium_pro	pentium_pro	pentium_pro	generic
pentium3	sse	pentium3	16/32/4:256/32/4
pentium4	sse2	pentium4	8/64/4:256/128/8
nehalem	sse4_2	nehalem	32/64/8:256/64/8:8192/64/16
penryn	sse4_1	penryn	2/64/8:4096/64/16
woodcrest	ssse3	core2	32/64/8:4096/64/16
barcelona	amdsse4a	amdfam10	64/64/2:512/64/16

64 ビット対応の x86 プラットフォームでの 64 ビット Solaris OS 向けのコンパイルは、**-m64** フラグで指示します。たとえば、**-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 **-xtarget** フラグのあとに **-m64** オプションを表示する必要があります。

-xtarget=opteron -m64

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。

3.4.176 -xtime

-time と同義です。

3.4.177 -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で指定します。

real:*size***double:** *size***integer:** *size*

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:16**
- **integer:32**
- **integer:64**

次に例を示します。

- **-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。単精度の **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、**COMPLEX** は 2つの **REAL** としてマップされます。また、**DOUBLE COMPLEX** は、**DOUBLE** と同じようにマップされます。

3.4.178 -xunroll=*n*

-unroll=*n* と同義です。

3.4.179 –xvector[= [[no%]lib, [no%] simd, %none]]

ベクトルライブラリ関数を自動呼び出しします。

–xvector オプションを指定するには、最適化レベルが **–x03** またはそれ以上であることが必要です。最適化レベルが指定されていない場合や **–x03** よりも低い場合はコンパイルは続行されず、メッセージが表示されます。

このオプションを使用する場合、**-xvector** を使用してコンパイルするときに、デフォルトの丸めモードである **-fround=nearest** でコンパイルする必要があります。

-xvector=lib (Solaris プラットフォームのみ) が指定されると、可能な場合は、コンパイラはループ内の数学ライブラリへの呼び出しを、同等のベクトル数学ルーチンへの単一の呼び出しに変換します。大きなループカウントを持つループでは、これによりパフォーマンスが向上します。この機能は、**-xvector=no%lib** で無効になります。

-xvector=simd を使用すると、コンパイラはネイティブ x86 SSE SIMD 命令を使用して特定のループのパフォーマンスを向上させます。ストリーミング拡張機能は、x86 で最適化レベルが 3 かそれ以上に設定されている場合にデフォルトで使用されます。サブオプション `no%simd` を使用すると、この機能を無効にできます。

コンパイラは、ストリーミング拡張機能がターゲットのアーキテクチャーに存在する場合、つまりターゲットの ISA が SSE2 以上である場合にのみ SIMD を使用します。たとえば、最新のプラットフォームで `-xtarget=woodcrest`、`–xarch=generic64`、`-xarch=sse2`、`-xarch=sse3`、または `-fast` を指定して使用できます。ターゲットの ISA にストリーミング拡張機能がない場合、このサブオプションは無効です。

-xvector=simd と **-fsimple=2** の両方を指定すると、**-xvector=simd** だけを指定した場合よりもパフォーマンスが向上します。ただし、**-fsimple=2** では浮動小数点演算の並べ替えが許可されるため、浮動小数点の結果がわずかに異なる場合があります。

デフォルトは、x86 では `-xvector=simd` で、SPARC プラットフォームでは `-xvector=%none` です。サブオプションなしで `-xvector` を指定すると、コンパイラでは、x86 では `-xvector=simd,lib`、SPARC (Solaris) では `-xvector=lib`、および `-xvector=simd` (Linux) が使用されます。

コンパイラは、ロード時に libmvec ライブラリを取り込みます。コンパイル時に **-xvector=lib** を指定した場合は、リンク時にも指定する必要があります。

このオプションは以前のインスタンスを上書きするため、**-xvector=%none** は、それ以前に指定した **-xvector=lib** よりも優先されます。

3.4.180 -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 オプションと **-xprofile=collect** オプションを一緒に使用しないでください。**-ztext** が読み取り専用記憶領域での位置依存シンボルの再配置を禁止する一方で、**-xprofile=collect** は書き込み可能記憶領域での静的に初期化された位置依存シンボルの再配置を生成します。

-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

4 Solaris Studio Fortran の機能と相違点

この付録では、標準のFortran と Solaris Studio Fortran 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

f95 コンパイラは、Fortran 標準規則に対して、次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、999 行まで行を継続することができます (開始行が 1 行とそのあとの継続行が 999)。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。

4.1.3 タブ形式

f95 の固定形式のソーステキストは、次のように定義されています。

- 1 から 6 の任意の列にあるタブによって、その行がタブ形式のソース行になります。
- タブより前に、コメントインジケータまたは文番号がある場合があります。
- タブが最初の文字で空白以外の場合は、次のようになります。

- タブのあとの文字がゼロでない数字以外の場合、タブに続くテキストが最初の行です。
- 最初のタブのあとがゼロでない数字の場合、その行は継続行です。ゼロでない数字に続くテキストは、その文の次の部分です。

f95 のデフォルト最大行の長さは、固定形式で 72 列および自由形式で 132 列です。**-e** コンパイラオプションを使用すると、固定形式のソースの行を 132 列に拡張できます。

例: 左のタブの形式のソースは、右に表示されます。

```
!^IUses of tabs                                 !       Uses of tabs
^ICHARACTER *3 A = 'A'                                  CHARACTER *3 A = 'A'
^IINTEGER B = 2                                         INTEGER B = 2
^IREAL C = 3.0                                          REAL C = 3.0
^IWRITE(*,9) A, B, C                                    WRITE(*,9) A, B, C
9^IFORMAT(1X, A3,                               9       FORMAT(1X, A3,
^I1 I3,                                                1 I3,
^I2 F9.1 )                                             2 F9.1 )
^IEND                                                   END
```

前述の例で、「**^I**」はタブ文字を表し、「1」および「2」で始まる行は継続行を表しています。コードはさまざまなタブの状態を示しますが、いずれの形式も推奨しません。

f95 は、タブがあると以降その行の72行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表示される場合、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.
```

–f77 オプションを使用したタブ書式も可能です。

4.1.4 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、および接尾辞によって異なります。

接尾辞が **.f** または **.F** のファイルは、固定形式とみなされます。接尾辞が **.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	処理
-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、ファイル名の接尾辞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接尾辞よりも優先されます。

4.1.4.1 書式の混在

次のように、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。
- 同じプログラム単位内では、タブ形式と自由形式または固定形式を混在させることができます。

4.1.4.2 大文字・小文字の区別

Solaris Studio Fortran 95 では、デフォルトで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。

4.1.5 制限とデフォルト

- Fortran 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。
- 変数およびほか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 データ型の機能と拡張子について説明します。

4.2.1 ブール型

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

- マスク処理の場合、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、次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、次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、次のような書式 (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

X'*ddd***'** または **X"***ddd***"**、*d* が任意の 16 進数である 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む必要があります。
- 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、次の書式を使用できます。

*n***H**...	**'**...**'H**	**"**..."**H**
*n***L**...	**'**...**'L**	**"**..."**L**
*n***R**...	**'**...**'R**	**"**..."**R**

前述の「...」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、次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 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

表 4–2 数値データ型のサイズの表記法 (続き)

非標準	宣言子	短縮書式	意味
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。

いくつかの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』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 のデータ型	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
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 (**-m64** 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。このデータ型は、*4* 倍精度とも呼ばれます。
- 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 バイトのデータすべてを、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、別のエンティティー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** 文は次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、次のとおりです。

pointer_name 対応する *pointee_name* へのポインタです。

pointer_name には *pointee_name* のアドレスが含まれます。pointer_name にはスカラー変数名を指定してください (派生型は指定できません)。禁止事項: 定数、構造体の名前、配列、または関数。

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 ポインタを宣言できます。

```
   POINTER ( p, b ),  ( q, c )
```

前述の例では、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 ポインタを宣言することもできます。

```
    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次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、次元の設定は次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

- *pointee_name* は、**CHARACTER*(*)** で型宣言された変数であってはいけません。
- *pointee_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いけません。
- Cray ポインタを配列にすることはできません。
- Cray ポインタを次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
  - ほかのデータ型で宣言する

  Cray ポインタを次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 ポインタには次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
...
```

前述の例を説明します。

- **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** のあとに位置し、長さは **n** です。
- **c** は **b** のあとに位置します。
- **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)

従来の FORTRAN 77 からプログラムを移行しやすくするため、**f95** は、Fortran 95 の「構造型」のプレカーソルである、VAX Fortran の **STRUCTURE** 文と **UNION** 文を受け入れます。構文についての詳細は、『FORTRAN 77 言語リファレンスマニュアル』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)。

従来の **f77** コンパイラと同様に、**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 コンパイラは言語への拡張子として、新しいデータ型である **UNSIGNED** を受け入れます。**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、数字列のあと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。並び入出力また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。
- 組み込み関数は、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。ただし、**MAX** 関数と **MIN** 関数では、**REAL** 型の演算対象が少なくとも 1 つ存在していれば、符号あり整数演算対象と符号なし整数演算対象を使用できます。
- 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 200x の機能

今回の Solaris Studio Fortran コンパイラのリリースには、Fortran 2003 規格の多数の新機能が含まれています。詳細は、Fortran 2003 規格を参照してください。また、Fortran 2008 ドラフト規格で提案されている機能もいくつか含まれています。これらの機能の詳細は、該当するドラフト出版物を参照してください。

4.6.1 C 関数との相互運用性

Fortran の新しい規格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 コンパイラの今回のリリースでは、規格の第 15 章に記述されている機能を実現します。また、規格第 4 章に述べられている、C の型に対応する構造型およびリスト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
```

INTRINSIC キーワードが Fortran 2003 で新しく追加されました。これらのモジュールは、一連の構造型、定数、丸めモード、照会関数、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 2003 規格の第 14 章を参照してください。

4.6.3 コマンド行引数用組み込み関数

Fortran 2003 規格では、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。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 コンパイラでは、Fortran 2003 の **PROTECTED** 属性が受け入れられています。**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 2003 規格の第 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、入出力は常に同期化します。

4.6.6 ALLOCATABLE 属性の拡張機能

Fortran 2003 で、**ALLOCATABLE** 属性に使用できるデータエンティティーが拡張されました。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、Fortran 2003 **VALUE** 型の宣言属性を受け入れます。

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 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 code:
#include <stdlib.h>
int main(int ac, char *av[])
{
    to_fortran(2);
}

Fortran code:
       subroutine to_fortran(i)
       integer, value :: i
       print *, i
       end
```

4.6.8 Fortran 2003 ストリーム入出力

Fortran 2003 規格では、新しい「ストリーム」入出力方式が定義されています。ストリーム入出力アクセスは、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。データファイルは、書式付きアクセスまたは書式なしアクセス用に結合できます。

OPEN 文で **ACCESS='STREAM'** 指定子を使用して、ストリーム入出力ファイルを宣言します。バイトアドレスにファイルを位置付けるには、**READ** または **WRITE** 文に **POS=***scalar_integer_expression* 指定子が必要です。**INQUIRE** 文は、**ACCESS='STREAM'**、指定子 **STREAM=***scalar_character_variable*、および **POS=***scalar_integer_variable* を受け入れます。

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、浮動小数点の小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、小数部のコンマ、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。次に例を示します。

```
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 (79 ページの「3.4.45 **–iorounding**[={**compatible**|**processor-defined**}]」）。

4.6.10 Fortran 2003 の IMPORT 文

IMPORT 文は、親子結合によってアクセス可能な親有効域の要素を指定します。この文は、インタフェース本体でのみ使用できます。

4.6.11 Fortran 2003 の FLUSH 入出力文

f95 コンパイラは、Fortran 2003 の **FLUSH** 文を受け入れます。**FLUSH** 文を使用すると、外部ファイルに書き込まれたデータをほかのプロセスで利用したり、Fortran 以外の方法で外部ファイルに配置されたデータを **READ** 文で利用したりすることができるようになります。

4.6.12 Fortran 2003 POINTER INTENT 機能

Fortran コンパイラは、**POINTER** 仮引数の **INTENT** 属性をサポートするようになりました。ポインタ仮引数として **INTENT(IN)**、**INTENT(OUT)**、または **INTENT(INOUT)** を指定できます。

たとえば、次を見てください。

```
subroutine sub(P)
integer, pointer, intent(in) :: p
...

end
```

ポインタの **INTENT** 属性はポインタに適用され、指示先には適用されません。したがって、**INTENT(IN)** ポインタの場合、次のものはポインタを変更するため無効です。

```
p => t
allocate(p)
deallocate(p)
```

ただし、**INTENT(IN)** ポインタの場合、次のものは指示先を変更するため有効です。

```
p = 400
```

4.6.13 Fortran 2003 拡張配列構成子

配列構成子内の (/ と /) に角括弧を使用できるようになりました。

```
X = [ 3.2, 4.01, 6.5 ]
```

Fortran 2003 規格では、配列構成子としての角括弧の使用が許可されます。これによって、区間定数との間で衝突が起こる可能性があります。**-xia** オプション (または区間演算を有効にするための同様のオプション) を指定せずに角括弧を使用すると、配列構成子として処理されます。**-xia** オプションを使用すると、角括弧は定数として処理されます。区間ユーザーは、コンパイルエラーを回避するために、(/ および /) 配列構成子を継続して使用する必要があります。

4.6.14 その他の Fortran 2003 および Fortran 2008 機能

次に示す Fortran 2003 機能についての詳細は、公開されている Fortran 2003 規格を参照してください。Fortran 2008 機能については、公開されている Fortran 200x ドラフトドキュメントを参照してください。

- 割り付け配列の 2003 拡張 — 配列の再割り付け、および割り付けスカラー
- **ALLOCATE/DEALLOCATE** 文の 2003 拡張 — **ERRMSG** および **SOURCE**
- 2003 拡張の **MOVE_ALLOC** 組み込み関数
- 2003 拡張のポインタ代入と再マッピング
- 2003 拡張の **MIN/MAX**、**MIN/MAXVAL**、および **MIN/MAXLOC** と文字引数
- 2003 組み込み関数 **IS_IOSTAT_END**、**IS_IOSTAT_EOR**、**NEW_LINE**
- 2003 組み込み関数 **SELECTED_CHAR_KIND**
- 組み込み関数 **SYSTEM_CLOCK** の引数 **COUNT_RATE** の 2003 **REAL** 型
- 複素 **SQRT** 組み込み関数の結果に関する 2003 の新規制限
- 2008: 欠如しているオプション引数としての null ポインタの使用

- x86 プラットフォームでの IEEE 組み込みモジュールのサポート
- 2008 ビット組み込み関数:
  `BGE、 BGT、 BLE、 BLT、 DSHIFTL、 DSHIFTR、 LEADZ、 POPCNT、 POPPAR、 TRAILZ、 MASKL、 MASKR`

4.7 新しい入出力拡張機能

ここでは、Fortran 2003 規格には含まれていませんが、**f95** コンパイラで受け入れら 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 コンパイラに組み込まれています。

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 **$** を受け入れます。ただし、グループ内の最後のデータ項目が **CHARACTER** データである場合は別です。その場合、**$** 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

ファイルを開くときに **FORM='BINARY'** と指定すると、レコード長がファイルに組み込まれないことを除いて、**FORM='UNFORMATTED'** とほぼ同じ結果になります。このデータがなければ、1 レコードの開始点と終了点を示す方法がありません。このように、後退する場所を知らせることができないので、**FORM='BINARY'** ファイルに対して **BACKSPACE** を実行できません。**'BINARY'** ファイルに対して **READ** を実行すると、入力リストの変数を設定するために必要な量のデータが読み込まれます。

- **WRITE** 文。データはバイナリでファイルに書き込まれ、出力リストで指定された量のバイトが転送されます。
- **READ** 文。入力リストの変数にデータが読み込まれ、リストで必要なだけのバイトが転送されます。ファイルにはレコードマークがないので、「レコードの終端」エラーは検出されません。検出されるエラーは、「ファイルの終端」または異常システムエラーだけです。
- **INQUIRE** 文。**FORM='BINARY'** で開いたファイル上の **INQUIRE** は、次の結果を返します。

  FORM="BINARY"ACCESS="SEQUENTIAL"DIRECT="NO"FORMATTED="NO"
  UNFORMATTED="YES"RECL=AND NEXTREC= は未定義です。
- **BACKSPACE** 文。許可されていません。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を切り捨てます。
- **REWIND** 文。通常どおり、ファイルの位置をデータの先頭に変更します。

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 言語リファレンスマニュアル』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。

すべての Fortran 指令についての概要は、付録 C 「Fortran 指令の要約」を参照してください。

4.8.1 f95 の特殊な指令行の書式

f95 は、23 ページの「1.8 コマンド行ヘルプ」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。
- 最初の指令行の 6 桁目は空白です。
- 継続指令行の 6 桁目は空白以外の文字です。

4.8.1.2 ソースが自由形式の場合

- **!DIR$** のあとに空白を 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。
- 指令は空白のあとに記述します。
- 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

指令行のあと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あとに、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 指令には次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。Sun および Cray 形式の並列化指令は、非推奨になり、廃止されました。OpenMP の Fortran API 指令および並列化モデルを使用してください。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する必要があります。

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。

前述の例では、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。

```
mainmain.oone.modmod_one.otwo.modmod_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。そのあとの手順で、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

Version 7.0 の Fortran 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 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ー、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	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	倍精度	**([X=]***x***,[Y=]***y***)**	P、E

備考**:** P: 名前を引数として渡すことができる。E: 組み込み関数の外部コードは実行時に呼び出される。

Fortran 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

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: Solaris Studio Fortran への移行

単体の FORTRAN 77 コンパイラの提供はありません。Solaris Studio Fortran コンパイラ **f95** は、Sun WorkShop **f77** コンパイラで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を含む多くのレガシー FORTRAN 77 プログラムをコンパイルします。

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** で受け入れられる 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。Sun WorkShop **f77** コンパイラで受け入れられる非標準 FORTRAN 77 拡張機能についての詳細は、`http://docs.sun.com/source/806-3594/index.html` の『FORTRAN 77 言語リファレンスマニュアル』を参照してください。

f95 コンパイラで使用可能な Fortran 言語のその他の拡張機能については、第 4 章「Solaris Studio Fortran の機能と相違点」 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します。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 95/2003 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、レガシー FORTRAN 77 コンパイラ **f77** の非標準機能を、直接、または **-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'** は、**OPEN** 文で表示できます。[**-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 (170 ページの「4.4 **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、型宣言の変数名のあとに ***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、そのあとにルーチン 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**]
- null 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()** の呼び出しとして取り扱われます。[**-f77=misc**]
- ******、***** など別の演算子のあと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** コンパイラは、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** 互換フラグ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** 文のあとに宣言できます。次に例を示します。

```
SUBROUTINE FOO(ARR,G)
DIMENSION ARR(G)
INTEGER G
RETURN
END
```

レガシー Sun WorkShop FORTRAN 77 コンパイラの非標準言語拡張機能の構文と意味についての詳細は、`http://docs.sun.com/source/806-3594/index.html` にある『FORTRAN 77 言語リファレンスマニュアル』(アーカイブファイル) 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、レガシー **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。これらの項目は、レガシー Sun WorkShop FORTRAN 77 コンパイラでサポートされる FORTRAN 77 の非標準拡張機能ですが、現在の **f95** ではサポートされません。

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、直接アクセス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。次に例を示します。

```
subroutine s(i1,i2)
integer i1(i2(1):10)
dimension i2(10)
...ERROR: "I2" has been used as a function,
therefore it must not be declared with the explicit-shape DIMENSION attribute.


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** コンパイラの **-dbl**、**-oldstruct**、**-i2**、**-i4**の各オプション、および **-vax** の一部のサブルーチンを認識しません。

f95 でサポートされていない **FORTRAN 77** 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、**IEEE_RETROSPECTIVE** サブルーチンが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ことを許可していません。

5.3 レガシー FORTRAN 77 でコンパイルされたルーチンでのリンク

- **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** コンパイラでリンクします。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 = 'This is a test.'
```

```
   CALL echo1( c )
END
demo% f95 -xlang=f77 m.f95 sub77.o
demo% a.out
 This is a test.
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 *, 'elapsed:', e, ', user:', t(1), ', sys:', t(2)
        END
demo% f95 tdtime.f95
demo% a.out
elapsed: 0.14 , user: 0.14 , sys: 0.0E+0
demo%
```

  dtime(3F) を参照してください。

5.3.1 Fortran 組み込み関数

Fortran 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 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マニュアル』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、組み込みではないユーザー指定の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 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ため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。103 ページの「3.4.108 **–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。

5.5 f77 コマンド

Solaris Studio ソフトウェアには、単独の FORTRAN 77 コンパイラ **f77** は含まれていません。最近のリリースでは、FORTRAN 77 の機能の多くが Fortran 95 コンパイラである **f95** に盛り込まれているため、レガシー FORTRAN 77 コンパイラの機能の多くは Fortran 95 コンパイラで利用できます。現在の Solaris Studio コンパイラリリースには、**f77** スクリプトが用意されています。このスクリプトは、対応するデフォルトオプションのセットを含む **f95** コンパイラを呼び出します。**f77** の呼び出しは次と同じです。

```
f95 -f77 -ftrap=%none
```

以前のリリースの **f77** コンパイラでコンパイルされたライブラリルーチンにリンクする必要がある場合は、**-xlang=f77** をコマンド行に追加します。ただし、コンパイルとリンクをそれぞれ別の手順で実行し、**-xlang=f77**、**-lM77**、**-lF77**、または **-lsunmath** を明示的に指定している場合は、**cc** または **CC** ではなく、**f95** (または **f77** スクリプト) でリンクする必要があります。また、**-fast** フラグを使用してコンパイルしている場合は、**-fast** がトラップモードを「common」に設定するため、**-fast** の後に **-ftrap=%none** を追加して、FORTRAN 77 の算術例外におけるトラップの動作を保持します。

```
f77 -fast -ftrap=%none
```

f77 スクリプトを呼び出すと、**f95** コンパイラを **-f77** 互換モードで使用していることを警告するメッセージが表示されます。このメッセージを表示しないようにするには、コマンド行に **-errtags=INVOKE** を追加します。詳細は、65 ページの「3.4.23 **–f77**[**=** *list*]」 を参照してください。

◆ ◆ ◆ 付 録 A

実行時のエラーメッセージ

この付録では、Fortran 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 の実行時入出力エラーメッセージ

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。**f95** 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 は **f95** で発行される実行時入出力メッセージを一覧表示します。

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	レコードの先頭を超えています **(off beginning of record)**
1008	ファイルの **stat** ができません **(can't stat file)**
1009	反復数のあとに ***** 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)**

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
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)
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超えて書き出ししようとしました (tried to write past end of internal file)

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
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)
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、または基数を超えています (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)

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
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)
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)

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
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')
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)

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
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)
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)

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
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)
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)

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
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)
1217	不正な CORR_ACCTION 値 (illegal CORR_ACCTION value)
1218	有効になっている入出力ハンドラが原因で、無限ループが発生しました (infinite loop occurred due to I/O handler enabled)
1220	必要なバイト数が、ターゲットプラットフォームでサポートされているバイト数を超えています (the number of requested bytes is greater than is supported on the target platform)
1221	互換性のない種類のファイルとの間で UNION 内のデータの読み書きを行うことはできません (data in a UNION cannot be read from or written to an incompatible file type)
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	短か過ぎる文字引数 (character argument too short)
2004	長過ぎる、または短か過ぎる配列引数 (array argument too long or too short)
2005	ファイル、記録、またはディレクトリストリームの終わり (end of file, record, or directory stream)
2021	初期化されていないロック (lock not initialized) (OpenMP)
2022	ロック変数の使用におけるデッドロック (deadlock in using lock variable) (OpenMP)
2023	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

この付録には、今回のリリースおよび以前のリリースの Fortran コンパイラの新機能と変更された機能を記載します。

B.1 Oracle Solaris Studio 12.2 Fortran リリース

Solaris Studio Fortran 95 コンパイラバージョン 8.5 は、Oracle Solaris Studio 12.2 リリースのコンポーネントです。

- SPARC-V9 命令セットの SPARC VIS3 バージョンをサポートします。**-xarch=sparcvis3** オプションでコンパイルすると、SPARC-V9 命令セット、VIS (Visual Instruction Set) version 1.0 を含む UltraSPARC 拡張機能、VIS (Visual Instruction Set) version 2.0、積和演算 (FMA) 命令、および VIS (Visual Instruction Set) version 3.0 を含む UltraSPARC-III 拡張機能の命令をコンパイラが使用できるようになります。
- x86 ベースのシステムに基づく **-xvector** オプションのデフォルト値が **-xvector=simd** に変更されました。x86 ベースのシステムでは、最適化レベル 3 およびそれ以上の場合にストリーミング拡張機能がデフォルトで使用されます。サブオプション **no%simd** を使用すると、この機能を無効にできます。SPARC ベースのシステムのデフォルトは **-xvector=%none** です。156 ページの「3.4.179 **–xvector**[**=** [[**no%**]**lib**, [**no%**] **simd**, **%none**]]」 を参照してください。
- AMD SSE4a 命令セットがサポートされるようになりました。**-xarch=amdsse4a** オプションでコンパイルします。
- 新しい **-traceback** オプションを使用すると、重大なエラーが発生した場合に実行可能ファイルでスタックトレースを出力できます。このオプションを指定すると、実行可能ファイルは、シグナルのセットをトラップし、スタックトレースとコアダンプを出力してから終了します。複数のスレッドが 1 つのシグナルを生成すると、スタックトレースは最初のスレッドに対してのみ生成されます。追跡表示を使用するには、**f95**、**cc**、または **CC** でプログラムをリンクするときに、**-traceback** オプションを追加します。便宜上、このオプションはコンパイル

時にも受け入れられますが、無視されます。**-traceback** オプションを **-G** オプションとともに使用して共有ライブラリを作成すると、エラーになります。97 ページの「3.4.95 **–traceback**[={ **%none**|**common**|*signals_list*}]」 を参照してください。

- **-mt** オプションが **-mt=yes** または **-mt=no** に変更されています。**-mt=yes** オプションにより、ライブラリが適切な順序でリンクされることが保証されます。83 ページの「3.4.56 **-mt**[={ **yes**|**no**}]」 を参照してください。
- **-xprofile=tcov** オプションが拡張されて、オプションのプロファイルディレクトリパス名がサポートされるようになりました。また、tcov 互換のフィードバックデータも生成できます。143 ページの「3.4.164 **–xprofile=***p*」 を参照してください。
- 新しい **-xkeepframe**[=[**%all,%none**] オプションでは、指定した機能のスタック関連の最適化を禁止できます。**%all** を指定すると、すべてのコードのスタック関連の最適化が禁止されます。**%none** を指定すると、すべてのコードのスタック関連の最適化が許可されます。デフォルトは **-xkeepframe=%none** です。129 ページの「3.4.135 **–xkeepframe**[=[**%all,%none**,*name*,**no%** *name*]]」 を参照してください。
- 追加の F2003 機能が実装されています。173 ページの「4.6 Fortran 200x の機能」 を参照してください。
- **IVDEP** 指令は、最適化の目的でループ内で検出された一部またはすべての配列参照のループがもたらす依存関係を無視するように、コンパイラに指示します。これにより、コンパイラはほかの方法で実行できないさまざまなループの最適化を実行できます。**-xivdep** オプションを使用して **IVDEP** 指令を無効にしたり、指令の解釈の方法を指定したりできます。39 ページの「2.3.3 **IVDEP** 指令」 を参照してください。

B.2 Sun Studio 12 Update 1 Fortran リリース

- Solaris OS (x86 プラットフォーム) または Linux OS 上のコンパイラで生成されるオブジェクトファイルは、アプリケーションコードに _m128/_m64 データ型を使用するパラメータまたは戻り値を持つ関数が含まれる場合、旧バージョンのコンパイラと互換性を持ちません。.il インライン関数ファイル、アセンブラコード、またはをこれらの関数を呼び出す asm インライン文を使用するユーザーも、この非互換性に注意する必要があります。
- 新しい x86 **–xtarget** 値の **woodcrest**、**penryn**、**nehalem**。
- 新しい SPARC **–xtarget** 値の **ultraT2plus**、**sparc64vii**。
- 新しい x86 **–xarch** 値および **–xchip** 値の **ssse3**、**sse4_1**、**sse4_2**、**core2**、**penryn**、**nehalem**、**barcelona**。
- 新しい SPARC **–xarch** 値および **–xchip** 値の **sparcima**、**sparc64vii**、**ultraT2plus**。
- **-xprofile=collect** と **-xprofile=use** の各オプションは、マルチスレッド化された動的リンクアプリケーションのサポートを改善します。

- **-xcrossfile=1** オプションは、**-xipo=1** オプションの別名になりました。
- Solaris プラットフォームでは、**-xpec**[= **yes**|**no**] オプションにより、自動チューニングシステム (Automatic Tuning System、ATS) とともに使用するために再コンパイルできる PEC バイナリが生成されます。
- **-xdepend** オプションが最適化レベル **-x03** 以上に対して暗黙的に有効になり、**-fast** オプションの展開に含まれません。
- OpenMP 3.0 タスクのサポート。
- **-xannotate**[=**yes**| **no**] (SPARC プラットフォームのみ) は、あとで **binopt** (1) などのバイナリ変更ツールで変換できるバイナリを作成するようにコンパイラに指示します。
- 4 倍精度 (REAL*16) が x86 プラットフォームで実装されます。REAL*16 は 128 ビット IEEE 浮動小数点です。
- コンパイラは、通常、一時ファイルを **/tmp** ディレクトリに作成します。**TMPDIR** 環境変数を設定することにより、別のディレクトリを指定できます。
- **cpu_time**() Fortran 組み込みルーチンの動作が、Solaris プラットフォームと Linux プラットフォームで異なります。
- Fortran 2003 の **IMPORT** 文が実装されます。

B.3 Sun Studio 12 Fortran リリース

- Fortran コンパイラは、次の Linux (x86 および x64) ディストリビューションで利用できるようになりました。SUSE Linux Enterprise Server 9 (Service Pack 3 以降)、Red Hat Enterprise Linux 4、および 2.6 カーネルを基にしたその他の Linux ディストリビューション (ただし正式なサポートはなし)。
- -**m64** を使用して、64 ビットの実行可能ファイルおよび共有ライブラリを作成できます。
- **-xarch** の新しいフラグにより、古いフラグが置き換えられました。
- **-xtarget** および **-xchip** の新しい値により、UltraSPARC T2 および SPARC64vi プロセッサ用にコードが生成されるようになりました。
- 新しいフラグ **-fma=fused** を使用することにより、FMA (fused multiply-add) 命令をサポートするプロセッサで、この命令を生成できるようになりました。
- 新しいフラグ **-xhwcprof** を使用すると、データ空間のプロファイリングがコンパイラでサポートされます。
- 新しいフラグ **-xinstrument** を使用すると、スレッドアナライザによるパフォーマンス分析が有効になります。
- x86 で、**-xregs=frameptr** が **-fast** に追加されました。
- Solaris x86 プラットフォームで、**-xarch=sse2** および **-xia** オプションを使用することにより、区間演算がサポートされます。

- 明示的な先取り指令が、SPARC プラットフォームだけでなく、x86 プラットフォームで使用できるようになりました。(**–xprefetch=explicit**)
- デバッグ情報のデフォルトの形式が「stabs」標準形式から「dwarf」標準形式に変更されました。(**-xdebugformat=dwarf**)

B.4 Sun Studio 11 Fortran リリース

- 新しい **-xmodel** オプション。新しい -xmodel オプションでは、64 ビット AMD アーキテクチャーでカーネル、スモール、ミディアムのメモリーモデルを指定できます。大域変数および静的変数のサイズが 2G バイトを超える場合は、**-xmodel=medium** を指定します。そうでない場合は、デフォルトの **-xmodel=small** 設定を使用します。135 ページの「3.4.148 **–xmodel=**[**small** | **kernel** | **medium**]」を参照してください。
- **x86 SSE2** プラットフォーム用に拡張された **-xvector** オプション。**-xvector** オプションでは、ベクトルライブラリ関数の呼び出しの自動生成や、SIMD (Single Instruction Multiple Data) 命令の生成が可能です。このオプションは、x86 SSE2 プラットフォームの拡張構文を提供します。156 ページの「3.4.179 **–xvector**[**=** [[**no%**]**lib**, [**no%**] **simd**, **%none**]]」 を参照してください。
- **STACKSIZE** 環境変数の拡張。**STACKSIZE** 環境変数の構文が拡張され、単位キーワードを含めることができるようになりました。
- **x86** プラットフォームで利用できる **-xpagesize** オプション。SPARC のほかに x86 プラットフォームでも、オプション **-xpagesize**、**-xpagesize_heap**、-xpagesize_stack を使用できます。138 ページの「3.4.155 **-xpagesize=** *size*」 を参照してください。
- 新しい **UltraSPARC T1** および **UltraSPARC IV+** への対応。**-xarch**、**-xchip**、-xcache、-xtarget の値で、新しい UltraSPARC プロセッサがサポートされます。152 ページの「3.4.175 **-xtarget=***t*」を参照してください。

B.5 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 Solaris 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。次に例を示します。**% 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.6 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 Solaris Studio では、x86 プラットフォーム版 Solaris で Fortran コンパイラが使用できるようになっています。Solaris x86 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**、**-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**、**-xloopinfo**、**-xpagesize**、**-xprofile_ircache**、**-xreduction**、**-xvector**、**-depend**、**-openmp**、**-autopar**、**-vpara**、**-XlistMP**。また、x86 プラットフォームの場合、**-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 (**-xO4** または **-xO5**) 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次のとおりです。

  - **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した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=abs 64 -xprefetch=no**

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

デフォルトの SPARC アーキテクチャーは V7 でなくなりました。この Solaris 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 OS 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 アーキテクチャーで動作します。

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 Solaris Studio リリースでは、SPARC V7 アーキテクチャー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 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.7 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
  ```

そのあと **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。

- 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、相互手続き最適化が最大 *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。129 ページの「3.4.134 **-xjobs=** *n*」を参照してください。

- **PRAGMA ASSUME** を使った表名

  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。37 ページの「2.3.1.8 **ASSUME** 指令」および 110 ページの「3.4.111 **–xassume_control**[**=***keywords*]」を参照してください。

- **Fortran 2003** 機能の追加

  Fortran 2003 の規格に記述されている次の機能が、Fortran コンパイラの今回のリリースで実装されました。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 IEEE_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。174 ページの「4.6.2 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 規格では、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。173 ページの「4.6.1 C 関数との相互運用性」を参照してください。

  - **PROTECTED** 属性

    Fortran コンパイラでは、Fortran 2003 の **PROTECTED** 属性が受け入れられています。**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。174 ページの「4.6.4 **PROTECTED** 属性」を参照してください。

  - **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'**]

    174 ページの「4.6.5 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 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す。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、179 ページの「4.7.1 入出力エラー処理ルーチン」、マニュアルページ、および『Fortran ライブラリリファレンス』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 コンパイラは言語への拡張子として、新しいデータ型である **UNSIGNED** を受け入れます。171 ページの「4.5 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。138 ページの「3.4.155 **-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。147 ページの「3.4.165 **–xprofile_ircache**[=*path*]」 を参照してください。147 ページの「3.4.166 **–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。130 ページの「3.4.136 **-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。132 ページの「3.4.144 **–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、局所変数を、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値に初期化します。113 ページの「3.4.115 **-xcheck=***keyword*」 を参照してください。

B.8 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。

the command:
`f77` *options files libraries*
becomes a call to the `f95` *compiler::*
`f95 -f77=%all -ftrap=%none` *options files* `-lf77compat` *libraries*

FORTRAN 77 の互換性および非互換性についての詳細は、186 ページの「4.12 言語の混在」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。65 ページの「3.4.23 **–f77**[=*list*]」 および 186 ページの「4.12 言語の混在」 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。103 ページの「3.4.108 **–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** 文が各副プログラムに挿入されます。99 ページの「3.4.100 **-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** 副プログラム (**.mod** ファイル) をどこに書き込むかを制御できます。83 ページの「3.4.55 **-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制御できます。

- -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。81 ページの「3.4.53 **-M***path*」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、183 ページの「4.9 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、101 ページの「3.4.106 **-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、Sun 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。130 ページの「3.4.136 **-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指定された仮引数名の型、種類、ランクを無視するように、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、34 ページの「2.3.1.2 **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。58 ページの「3.4.6 **-C**」を参照してください。
- **Fortran 2003** 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、176 ページの「4.6.9 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。(プロセッサで定義されたまたは互換性がある) モードは、Fortran 2003 の機能として実装された **ROUND=** 指定子に対応します。79 ページの「3.4.45 **–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** セグメント例外が発生します。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。113 ページの「3.4.115 **-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。

- 相互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す。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。125 ページの「3.4.131 **–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。143 ページの「3.4.163 **–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

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
- Sun の並列化指令
- Cray の並列化指令
- OpenMP Fortran 95 指令、ライブラリルーチン、および環境

C.1 一般的な Fortran 指令

f95 で受け入れられる一般的な指令ては、31 ページの「2.3 指令」で説明します。

表 C–1 一般的な Fortran 指令の要約

書式	
!$PRAGMA *keyword* **(** a **[,]** a **] …)** [, *keyword* (a [, a **] …)] ,…** **!$PRAGMA SUN** *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。(これらの例では注釈指示子として **!** を使用しています。**f95** の自由形式では **!** を使用する必要があります。	
C 指令	**!$PRAGMA C(***list* **)** 外部関数の名前リストを C 言語のルーチンとして宣言します。
IGNORE_TKR 指令	**!$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	**!$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。

表 C–1 一般的な Fortran 指令の要約 (続き)

WEAK 指令	**!$PRAGMA WEAK(***name*[**=***name2*]**)** *name* を弱いシンボル (weak symbol) または *name2* の別名として宣言します。
OPT 指令	**!$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。
PIPELOOP 指令	**!$PRAGMA SUN PIPELOOP**[**=** *n*] ループの *n* 離れた反復間の依存性を宣言します。
PREFETCH 指令	**!$PRAGMA SUN_PREFETCH_READ_ONCE (***name* **)** **!$PRAGMA SUN_PREFETCH_READ_MANY (***name* **)** **!$PRAGMA SUN_PREFETCH_WRITE_ONCE (***name* **)** **!$PRAGMA SUN_PREFETCH_WRITE_MANY (***name* **)** 名前の参照のために、先読み命令を生成するようにコンパイラに要求します。(**-xprefetch** オプションを指定する必要があります。このオプションはデフォルトで有効になっています。PREFETCH 指令は、**–xprefetch=no** でコンパイルし無効にします。ターゲットアーキテクチャーも PREFETCH 指令をサポートしている必要があり、コンパイラ最適化レベルは **-xO2** より上に設定されている必要があります)
ASSUME 指令	**!$PRAGMA** [**BEGIN**} **ASSUME (***expression* [**,***probability*]**)** **!$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。

C.2 特殊な Fortran 指令

次の指令は、**f95** でのみ使用できます。詳細は、182 ページの「4.8.2 **FIXED** 指令と **FREE** 指令」を参照してください。

表 C–2 特殊な Fortran 指令

書式	**!DIR$** *directive* : 最初の行 **!DIR$&** ... : 継続行 固定形式の場合、**C** は CDIR$ directive... のように指令指示子としても **CDIR$** *directive*... 行は第 1 行から始める必要があります。自由形式の場合は、行の前に空白がある場合があります。

表 C–2 特殊な Fortran 指令 (続き)

FIXED/FREE 指令	**!DIR$ FREE!DIR$ FIXED** 指令のあと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。
IVDEP	**!DIR$ IVDEP** 次の **DO**、**FORALL**、または **WHERE** ループにループがもたらす依存関係がないことを表明します。これでループを最適化できます。**-xivdep** オプションによって解釈が決まります。39 ページの「2.3.3 **IVDEP** 指令」を参照してください。

C.3 Fortran の OpenMP 指令

Solaris Studio Fortran のコンパイラでは、OpenMP 3.0 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 (137 ページの「3.4.153 **–xopenmp**[={ **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。

索引

G

I

L

M

N

O

P

R

S

き

く

け

こ

さ

し