



# Fortran ユーザーズガイド

---

Sun WorkShop 6  
Fortran 77 および Fortran 95

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

Part No. 806-4842-01  
2000 年 6 月 Revision A

本製品およびそれに関連する文書は、その使用、複製、頒布および逆コンパイルを制限するライセンスのもとにおいて頒布されます。サン・マイクロシステムズ株式会社の書面による事前の許可なく、本製品および関連する文書のいかなる部分も、いかなる方法によっても複製することが禁じられます。フォント技術を含む第三者のソフトウェアは、著作権法により保護されており、提供者からライセンスを受けているものです。

本製品の一部は、カリフォルニア大学からライセンスされている Berkeley BSD システムに基づいていることがあります。UNIX は、X/Open Company Limited が独占的にライセンスしている米国ならびに他の国における登録商標です。Netscape™、Netscape Navigator™、および Netscape Communications Corporation のロゴは、次の著作権で保護されています。

© 1995 Netscape Communications Corporation.

Sun、Sun Microsystems、docs.sun.com、AnswerBook2、SunOS、JavaScript、SunExpress、Sun WorkShop、Sun WorkShop Professional、Sun Performance Library、Sun Performance WorkShop、Sun Visual WorkShop、Forte は、米国およびその他の国における米国 Sun Microsystems, Inc. (以下、米国 Sun Microsystems 社とします) の商標もしくは登録商標です。

サンのロゴマークおよび Solaris は、米国 Sun Microsystems 社の登録商標です。

すべての SPARC 商標は、米国 SPARC International, Inc. のライセンスを受けて使用している同社の米国およびその他の国における商標または登録商標です。SPARC 商標が付いた製品は、米国 Sun Microsystems 社が開発したアーキテクチャに基づくものです。

本書で参照されている製品やサービスに関しては、該当する会社または組織に直接お問い合わせください。

OPEN LOOK および Sun Graphical User Interface は、米国 Sun Microsystems 社が自社のユーザおよびライセンス実施権者向けに開発しました。米国 Sun Microsystems 社は、コンピュータ産業用のビジュアルまたはグラフィカル・ユーザインタフェースの概念の研究開発における米国 Xerox 社の先駆者としての成果を認めるものです。米国 Sun Microsystems 社は米国 Xerox 社から Xerox Graphical User Interface の非独占的ライセンスを取得しており、このライセンスは米国 Sun Microsystems 社のライセンス実施権者にも適用されます。

Sun f90 / f95 は、米国 Silicon Graphics, Inc. の Cray CF90™ に基づいています。

Federal Acquisitions: Commercial Software -- Government Users Subject to Standard License Terms and Conditions

本書は、「現状のまま」をベースとして提供され、商品性、特定目的への適合性または第三者の権利の非侵害の黙示の保証を含み、明示的であるか黙示的であるかを問わず、あらゆる説明および保証は、法的に無効である限り、拒否されるものとします。

本製品が、外国為替および外国貿易管理法(外為法)に定められる戦略物資等(貨物または役務)に該当する場合、本製品を輸出または日本国外へ持ち出す際には、サン・マイクロシステムズ株式会社の事前の書面による承諾を得ることのほか、外為法および関連法規に基づく輸出手続き、また場合によっては、米国商務省または米国所轄官庁の許可を得ることが必要です。

原典 : *Fortran User's Guide (Sun WorkShop 6 Fortran 95 - Fortran 77)*  
Part No: 806-3591-10  
Revision A

© 2000 by Sun Microsystems, Inc.



## 製品名の変更について

---

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

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

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

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

製品名の変更に加えて、次の 2 つの製品について大きな変更があります。

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

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

# 目次

---

製品名の変更について	iii
はじめに	xi
1. ご使用になる前に	1
規格への準拠	1
Sun Fortran コンパイラの特徴	2
その他の Fortran ユーティリティ	3
デバッグユーティリティ	4
Sun Performance Library	4
区間演算	5
マニュアルページ	5
README ファイル	6
コマンド行ヘルプ	8
2. Sun Fortran コンパイラの使用	9
クイックスタート	10
コンパイラの起動	12
コンパイルとリンク処理の流れ	12
ファイル名の拡張子	13

ソースファイル	14
ソースファイルプリプロセッサ	14
別々に実行するコンパイルとリンク	14
コンパイルとリンクの整合性	15
Fortran 95 と FORTRAN 77 のコンパイルが混在している場合のリンク	16
認識されないコマンド行引数	16
モジュール (Fortran 95)	17
指令	17
一般的な指令	18
並列化指令	22
OpenMP 指令	24
コンパイラの利用方法	24
ハードウェアプラットフォームの特定	25
メモリーサイズ	26
<b>3. f77 / f95 コンパイラオプション</b>	<b>31</b>
コマンド構文	31
オプションの構文	32
オプションのまとめ	33
頻繁に利用するオプション	38
下位互換オプションと旧オプション	39
現在使用されていないオプション	39
オプションリファレンス	40
SPARC プラットフォームでの <code>-xarch</code> の使用	96
<b>A. 実行時のエラーメッセージ</b>	<b>125</b>
オペレーティングシステムのエラーメッセージ	125
シグナルハンドラのエラーメッセージ	126

入出力のエラーメッセージ (f77)	126
入出力のエラーメッセージ (f95)	130
<b>B. 各リリースにおける機能変更</b>	<b>139</b>
Fortran 95 の新機能と変更	139
Sun WorkShop 6 における f95 の新機能	139
f90 2.0 の新機能	140
Fortran 77 の新機能と変更点	145
Sun WorkShop 6 での f77 の新機能	145
f77 5.0 の新機能	145
f77 4.2 の新機能	146
FORTRAN 77 の上位互換	147
Fortran 3.0/3.0.1 と 4.0	147
Solaris 1 アプリケーションの実行	147
技術用語について	148
<b>C. Fortran 95 の機能と相違点</b>	<b>151</b>
機能	151
継続行の制限	151
固定形式固定形式のソースの行	151
指令	151
タブ書式	152
使用するソースの書式	153
既知の制限	153
ブール (Boolean) 型	154
数値データ型のサイズの略記法	157
Cray ポインタ	158
組み込み関数	163

入出力拡張機能	164
指令	166
f95 の特殊な指令行の書式	166
FIXED 指令と FREE 指令	167
並列化の指令	168
Fortran 77 との互換性	168
ソース	168
制約	168
f77 でコンパイルしたルーチンとのリンク	169
入出力	170
組み込み関数	171
将来のバージョンとの互換性	172
言語の混在	172
モジュールファイル	172
D. -xtarget プラットフォームの展開	173
E. Fortran 指令の要約	179
一般的な Fortran 指令	180
特殊な Fortran 95 指令	181
Sun の並列化指令	182
Cray の並列化指令	183
Fortran 95 の OpenMP 指令	184
OpenMP ライブラリルーチン	192
OpenMP 環境変数	194
索引	195



# 表目次

---

表 1-1	重要なREADMEs	7
表 2-1	Fortran コンパイラが認識するファイル名の拡張子	13
表 2-2	一般的な Fortran 指令	19
表 3-1	オプションの構文	32
表 3-2	オプションの表記規則	32
表 3-3	機能別コンパイラオプション	33
表 3-4	頻繁に利用するオプション	38
表 3-5	下方互換オプション	39
表 3-6	現在使用されていないオプション	39
表 3-7	デフォルトのデータサイズ (バイト) と -dbl オプション	49
表 3-8	非正規数 REAL と DOUBLE	59
表 3-9	デフォルトのデータサイズ (バイト) と -r8	85
表 3-10	-vax のサブオプション	93
表 3-11	-xarch ISA キーワード	95
表 3-12	SPARC プラットフォーム上でのもっとも一般的な -xarch オプションのまとめ	96
表 3-13	SPARC プラットフォーム上で使用できる各 -xarch 値	97
表 3-14	-xcache の値	101
表 3-15	-xchip の値	102
表 3-16	-Xlist サブオプション	110
表 A-1	<a href="#">f77</a> 実行時入出力メッセージ	127
表 A-2	<a href="#">f95</a> の実行時入出力メッセージ	130

表 B-1	JIS X 3001-1994 の訳と本書の訳の違い	148
表 C-1	<a href="#">F95</a> ソース書式のコマンド行のオプション	153
表 C-2	数値データ型のサイズの表記法	157
表 C-3	非標準の組み込み関数	163
表 D-1	<code>-xtarget</code> の展開	173
表 E-1	一般的な Fortran 指令の要約	180
表 E-2	特殊な Fortran 95 指令	181
表 E-3	Sun 形式の並列化指令の要約	182
表 E-4	Cray の並列化指令の要約	183
表 E-5	Fortran 95 における OpenMP 指令の要約	184
表 E-6	Fortran 95 OpenMP ライブラリルーチンの要約	192
表 E-7	OpenMP Fortran 環境変数の要約	194

# はじめに

---

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

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

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

---

## マルチプラットフォーム対応

この Sun WorkShop リリースは、Solaris 2.6、7、および 8 のオペレーティング環境 (SPARC™ プラットフォームおよび Intel プラットフォーム) をサポートしています。

特定のプラットフォームの [f77](#) と [f95](#) コンパイラの利点については、Sun WorkShop README ディレクトリの `fortran_77` および `fortran_95` の README ファイルを参照してください (6 ページも参照してください)。

---

## Sun WorkShop 開発ツールへのアクセス方法

Sun WorkShop 製品コンポーネントとマニュアルページは標準ディレクトリ `/usr/bin` および `/usr/share/man` にはインストールされません。そのため `PATH` および `MANPATH` 環境変数を変更して Sun WorkShop コンパイラとツールにアクセスできるようにする必要があります。

`PATH` 環境変数を設定する必要があるかどうか判断するには以下を実行します。

1. 次のように入力して、`PATH` 変数の現在値を表示します。

```
% echo $PATH
```

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

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

`MANPATH` 環境変数を設定する必要があるかどうか判断するには以下を実行します。

1. 次のように入力して、`workshop` マニュアルページを表示します。

```
% man workshop
```

2. 出力された場合、内容を確認します。

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

---

注 – この節に記載されている情報は Sun WorkShop 6 製品が `/opt` ディレクトリにインストールされていることを想定しています。Sun WorkShop ソフトウェアが `/opt` ディレクトリにインストールされていない場合は、システム管理者に連絡してください。

---

`PATH` 変数および `MANPATH` 変数は、C シェルを使用している場合はホームディレクトリの下での `.cshrc` ファイルに設定する必要があります。Bourne シェルか Korn シェルを使用している場合は、ホームディレクトリの下での `.profile` ファイルに設定する必要があります。

- Sun WorkShop コマンドを使用するには、`PATH` 変数に以下を追加してください。

```
/opt/SUNWspro/bin
```

- `man` コマンドで、Sun WorkShop マニュアルページにアクセスするには、`MANPATH` 変数に以下を追加してください。

```
/opt/SUNWspro/man
```

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

---

## 内容の紹介

このマニュアルは次の章と付録から構成されています。

第 1 章「ご使用になる前に」では、このコンパイラの機能について簡単に説明します。

第 2 章「Sun Fortran コンパイラの使用」では、コンパイラの使用環境について説明します。

第 3 章「f77 / f95 コンパイラオプション」では、すべてのコマンド行オプションについて詳しく説明します。

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

付録 B 「各リリースにおける機能変更」では、今回のリリースで追加された機能と変更された機能について説明します。

付録 C 「Fortran 95 の機能と相違点」では、標準の Fortran 95 と Sun Fortran 95 との相違点について説明します。

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

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

---

## 書体と記号について

このマニュアルで使用している書体と記号について説明します。

表 P-1 このマニュアルで使用している書体と記号

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

表 P-1 このマニュアルで使用している書体と記号 (続き)

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

- 小さい三角 (Δ) は意味のある空白を示します。

ΔΔ36.001

- FORTRAN 77 の例はタブ書式で、Fortran 95 の例は自由書式で示します。  
FORTRAN 77 と Fortran 95 に共通の例は、特に明示がない限りタブ書式で示します。
- FORTRAN 77 規格では、「FORTRAN」とすべて大文字で表記する旧表記規則を使用しています。サンのマニュアルでは FORTRAN と Fortran の両方を使用しています。現在の表記規則では、「Fortran 95」と小文字を使用しています。。
- オンラインマニュアル (man) ページへの参照は、トピック名とセクション番号とともに表示されます。たとえば、GETENV への参照は、[getenv\(3F\)](#) と表示されます。[getenv\(3F\)](#) とは、このページにアクセスするためのコマンドが `man -s 3F getenv` であるという意味です。
- システム管理者は Sun Fortran コンパイラとそのサポート機器を、  
`<install_point>/SUNWspr0/SC5.0/` にインストールできます。`<install_point>` には、通常、標準のインストールを示す `/opt` を指定します。本書では `/opt` をインストールポイントとして使用します。

---

## シェルプロンプトについて

シェルプロンプトの例を以下に示します。

表 P-2 シェルプロンプト

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

---

## 関連マニュアル

以下の方法で、関連マニュアルにアクセスすることができます。

- インターネットの [docs.sun.com](http://docs.sun.com) の Web サイトからアクセスできます。特定の本のタイトルで検索するか、主題、マニュアルコレクションまたは製品別にブラウズすることができます。

<http://docs.sun.com>

- ローカルシステムまたはローカルネットワークにインストールされた Sun WorkShop 製品からアクセスできます。Sun WorkShop 6 HTML 文書 (マニュアル、オンラインヘルプ、マニュアルページ、各コンポーネントの README ファイル、リリースノート) が、インストールした Sun WorkShop 6 製品から参照可能です。HTML 文書にアクセスするには、次のいずれかを実行します。
  - Sun WorkShop または Sun WorkShop™ TeamWare ウィンドウで、「ヘルプ」
    - ▶ 「オンラインマニュアルについて」を選択します。
  - Netscape™ Communicator 4.0 またはその互換バージョンのブラウザで、以下のファイルを開きます。

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

参照できる Sun WorkShop 6 HTML 文書の一覧がブラウザに表示されます。一覧にあるマニュアルを開くには、マニュアルのタイトルをクリックしてください。



表 P-3 は、Sun WorkShop 6 関連マニュアルをマニュアルコレクション別に一覧にしたものです。

表 P-3 マニュアルコレクション別 Sun WorkShop 6 関連マニュアル

マニュアルコレクション	マニュアルタイトル	内容の説明
Forte Developer 6 / Sun WorkShop 6 リリース マニュアル	Sun WorkShop 6 マニュアルの概要	Sun WorkShop 6 で使用可能なマニュアルとそのアクセス方法について説明しています。
	Sun WorkShop の新機能	Sun WorkShop 6 の現在のリリースと以前のリリースでの新機能についての情報を記載しています。
	Sun WorkShop 6 リリースノート	インストールの詳細と Sun WorkShop 6 最終リリースの直前に判明した情報を記載しています。このマニュアルはコンポーネントごとの README ファイルにある情報を補足するものです。
Forte Developer 6 / Sun WorkShop 6	プログラムのパフォーマンス解析	新しい標本コレクタと標本アナライザの使い方について説明しています ( 上級者向けのプロファイリング事例と説明付き )。コマンド行解析ツール <code>er_print</code> 、ループツール、ループレポートユーティリティおよび UNIX プロファイルツール <code>prof</code> 、 <code>gprof</code> 、 <code>tcov</code> についての情報も含んでいます。
	dbx コマンドによるデバッグ	dbx コマンドを使ってプログラムをデバッグする方法について説明しています。参考情報として、同じデバッグ処理を Sun WorkShop デバッグウィンドウを使って実行する方法も記載しています。
	Sun WorkShop の概要	Sun WorkShop 統合プログラミング環境の基本的なプログラム開発機能について説明しています。

表 P-3 マニュアルコレクション別 Sun WorkShop 6 関連マニュアル

マニュアルコレクション	マニュアルタイトル	内容の説明
Forte C 6 / Sun WorkShop 6 Compilers C	C ユーザーズガイド	C コンパイラオプション、サン固有の機能 (プラグマ、 <a href="#">lint</a> ツール、並列化、64 ビットオペレーティングシステムへの移行および ANSI/ISO 準拠 C) について説明しています。
Forte C++ 6 / Sun WorkShop 6 Compilers C++	C++ ライブラリ・リファレンス	C++ ライブラリについて説明しています。C++ 標準ライブラリ、Tools.h++ クラスライブラリ、Sun WorkShop Memory Monitor、 <a href="#">Iostream</a> および複素数の情報も含まれます。
	C++ 移行ガイド	コードを本バージョンの Sun WorkShop C++ コンパイラに移行する方法について説明しています。
	C++ プログラミングガイド	新しい機能を使ってより効率的なプログラムを記述する方法について説明しています。テンプレート、例外処理、実行時の型識別、キャスト演算、パフォーマンス、およびマルチスレッド対応のプログラムに関する情報も記載されています。
	C++ ユーザーズガイド	コマンド行オプションとコンパイラの使い方についての情報を記載しています。
	Sun WorkShop Memory Monitor ユーザーズガイド	C および C++ のメモリー管理で生じた問題を Sun WorkShop Memory Monitor で解決する方法について説明しています。このマニュアルはインストールした製品 ( <a href="#">/opt/SUNWspro/docs/ja/index.html</a> ) からのみ参照可能で、 <a href="#">docs.sun.com</a> Web サイトで参照することはできません。
Forte for High Performance Computing 6 / Sun WorkShop 6 Compilers Fortran 77/95	Fortran ライブラリ・リファレンス	Fortran コンパイラによって提供されるライブラリルーチンの詳細について説明しています。

表 P-3 マニュアルコレクション別 Sun WorkShop 6 関連マニュアル

マニュアルコレクション	マニュアルタイトル	内容の説明
	Fortran プログラミングガイド	入出力、ライブラリ、プログラム分析、デバッグおよびパフォーマンスに関連する内容を記述しています。
	Fortran ユーザーズガイド	コマンド行オプションとコンパイラの使い方についての情報を記載しています。
	FORTTRAN 77 言語リファレンス	Fortran 77 言語の包括的な参照情報を記載しています。
	Fortran 95 区間演算プログラミングリファレンス	Fortran 95 コンパイラによってサポートされる組み込み <a href="#">INTERVAL</a> データについて説明しています。
Forte TeamWare 6 / Sun WorkShop TeamWare 6	Sun WorkShop TeamWare ユーザーズガイド	Sun WorkShop TeamWare コード管理ツールの使用方法について説明しています。
Forte Developer 6 / Sun WorkShop Visual 6	Sun WorkShop Visual ユーザーズガイド	C++ と Java™ の GUI (グラフィカルユーザーインターフェース) を Sun WorkShop Visual を使用して作成する方法について説明しています。このマニュアルには、旧リリース (Sun WorkShop Visual 5.0) から変更のない機能が記載されています。
	Sun WorkShop Visual の新機能	Sun WorkShop Visual 6.0 で追加または変更された機能について説明しています。
Forte / Sun Performance Library 6	Sun Performance Library Reference (英語のみ)	コンピュータによる線形代数および高速フーリエ変換を実行するサブルーチンと関数の最適化ライブラリについて説明しています。
	Sun Performance Library User's Guide (英語のみ)	線形代数で発生した問題の解決に使用されるサブルーチンと関数のコレクションである Sun Performance Library のサン固有の機能の使用方法について説明しています。

表 P-3 マニュアルコレクション別 Sun WorkShop 6 関連マニュアル

マニュアルコレクション	マニュアルタイトル	内容の説明
数値計算ガイド	数値計算ガイド	浮動小数点演算における数値の精度に関する問題について説明しています。
標準ライブラリ 2	Standard C++ Library Class Reference (英語のみ)	標準 C++ の詳細について説明しています。
	標準 C++ ライブラリ・ユーザーズガイド	標準 C++ ライブラリの使用方法について説明しています。
Tools.h++ 7	Tools.h++ 7.0 ユーザーズガイド	Tools.h++ クラスライブラリの詳細について説明しています。
	Tools.h++ 7.0 クラスライブラリ・リファレンスマニュアル	C++ クラスを使用して、プログラム効率を向上させる方法について説明しています。

表 P-4 は、[docs.sun.com](http://docs.sun.com) の Web サイトからアクセスできる Solaris 関連マニュアルの一覧です。

表 P-4 Solaris 関連マニュアル

マニュアルコレクション	マニュアルタイトル	内容の説明
Solaris ソフトウェア開発	リンカーとライブラリ	Solaris リンクエディタと実行時リンカーの操作およびそれらが操作するオブジェクトについて説明しています。
	プログラミングユーティリティ	Solaris オペレーティング環境で使用可能な特殊組み込みプログラミングツールに関する開発者向けの情報を記載しています。

# 第1章

## ご使用になる前に

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

Fortran コンパイラは、Sun Performance WorkShop™ の構成要素です。Fortran 90 のコンパイラである f90 は、f95 に名前が変更されました。今回、f90 コマンドは f95 の別名になりました。どちらを使用しても、Sun Performance WorkShop 6 の Fortran 95 コンパイラを起動できます。

## 規格への準拠

- f77 は、ANSI X3.9-1978 Fortran 規格およびこれに対応する国際標準化機構 (ISO)1539-1980 規格に準拠しています。そのほか、FIPS 69-1、BS 6832、MIL-STD-1753 にも準拠しています。
- f95 は、ANSI X3.198-1992、ISO/IEC 1539:1991 および ISO/IEC 1539:1997 規格に準拠するように設計されました。
- 両コンパイラの浮動小数点演算は、IEEE 754-1985 規格および国際規格の IEC 60559:1989 に準拠しています。
- SPARC プラットフォームでは、UltraSPARC™ の実装を含む SPARC V8 および SPARC V9 の機能を利用した最適化を f77 と f90 の両方のコンパイラがサポートしています。これらの機能は『SPARC アーキテクチャマニュアル バージョン 8』(トッパン刊) および バージョン 9 (ISBN 0-13-099227-5) で定義されています。

- このマニュアルでは、「規格」とは前述の規格のバージョンに準拠していることを意味します。また、これらの規格のバージョン以降の機能は、「規格外」や「拡張機能」と呼ばれます。

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

---

## Sun Fortran コンパイラの特徴

Sun Fortran コンパイラには以下の拡張機能があります。

- **f77** : 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
- SPARC のみ : 自動的および明示的にループを並列化する機能など、マルチプロセッサシステムに対するサポートが最適化に統合されている。

---

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

---

- **f77** : VAX/VMS Fortran 5.0 拡張機能
  - NAMELIST
  - DO WHILE
  - 構造体、記録、共用体、マップ
  - 変数形式の式
  - 再帰
  - ポインタ
  - 倍精度複素数
  - SPARC : 4 倍精度実数
  - SPARC : 4 倍精度複素数
- TASK COMMON を含む、**f95** 用に拡張された CRAY 形式の並列化指令
- OpenMP 並列化指令は、**f95** で受け付けられます。

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

---

## その他の Fortran ユーティリティ

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

- Sun WorkShop Performance Analyzer - シングルスレッドアプリケーションおよびマルチスレッドアプリケーション用の詳細なパフォーマンス解析ツール。[analyzer\(1\)](#) を参照してください。
- `asa` - この Solaris ユーティリティは、1 カラム目に Fortran の改行制御文字のあるファイルを印刷する際の Fortran の出力フィルタです。UNIX システムは改行制御を使用していないため、Fortran の改行制御規則で書式化されたファイルを UNIX のラインプリンタの規則にしたがった書式に変換するときに、`asa` を使用します。詳細は [asa\(1\)](#) を参照してください。
- `fpp` - Fortran ソースコードプロセッサ。[fpp\(1\)](#) を参照してください。
- `fsplit` - このユーティリティは、複数のルーチンで構成される 1 つの Fortran ファイルを、1 ルーチンが 1 ファイルに対応するように複数のファイルに分割します。FORTRAN 77 または Fortran 95 のソースファイルには `fsplit` を使用します。詳細は、[fsplit\(1\)](#) を参照してください。

---

## デバッグユーティリティ

次のデバッグ用ユーティリティを利用することができます。

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

---

## Sun Performance Library

Sun Performance Library™ は、線形代数やフーリエ変換の演算に使用できる最適化サブルーチンと関数のライブラリです。このライブラリは、LAPACK、BLAS1、BLAS2、BLAS3、FFTPACK、VFFTPACK、および LINPACK といった標準ライブラリを基盤として構築されており、通常 [NetLib \(www.netlib.org\)](http://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、マルチプロセッシング環境で実行したほうが、実行速度も速く、精度もより正確で、使いやすくなります。

詳細は、[Performance\\_library](#) の README ファイルおよび『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)。



---

## 区間演算

Fortran 95 のコンパイラの今回のリリースでは、新しい二つのコンパイラフラグである `-xia` および `-xinterval` が導入されました。これによって、コンパイラは、新しい言語拡張を認識し、適切なコードを生成して、区間演算計算を実行します。

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

---

## マニュアルページ

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

マニュアルページは、次のコマンドによって表示することができます。

```
demo$ man topic
```

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

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

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

以下に Fortran を使用する場合の、関連マニュアルページを示します。

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

---

## README ファイル

[READMEs](#) ディレクトリには、新機能、ソフトウェアの互換性、バグについての説明、マニュアルが印刷された後に判明した情報を記載したファイルが置かれています。

[READMEs](#) ディレクトリの場所は、Solaris のバージョンやソフトウェアがインストールされた場所によって異なります。パスは、[install\\_directory/SUNWspro/READMEs/ja](#) です。通常インストールの場合、[install\\_directory](#) は、[/opt](#) になります。

標準インストーラの READMEs は `/opt/SUNWspro/READMEs/ja` にあります。

表 1-1 重要なREADMEs

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

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

```
f95 xhelp=readme
```

と指定すると、[fortran\\_95](#) README ファイルが直接表示されます。

## コマンド行ヘルプ

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

```
%f77 -help または
%95 -help
```

---

[ ] 中の項目は省略可能。< > 中の項目は変数パラメータ。

縦棒 | はリテラル値の選択を意味します。例：

`-someoption[=<yes|no>]` とある場合、`-someoption` は `-someoption=yes` を意味します。

---

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

## 第2章

# Sun Fortran コンパイラの使用

---

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

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

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

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

---

## クイックスタート

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

---

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

---

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

例：画面上にメッセージを出力するプログラムの作成例を示します。

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

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

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

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

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

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

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

以下に、 `f95` を使用した同じ例を示します。

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

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

---

## コンパイラの起動

単純なコンパイラコマンドのシェルプロンプトでの起動方法は次のとおりです。

<code>f77 &lt;オプション&gt; &lt;ファイル名&gt; ...</code>	Fortran 77 コンパイラ起動
<code>f95 &lt;オプション&gt; &lt;ファイル名&gt; ...</code>	Fortran 95 コンパイラ起動

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

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

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

---

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

---

## コンパイルとリンク処理の流れ

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

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

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



## ファイル名の拡張子

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

表 2-1 Fortran コンパイラが認識するファイル名の拡張子

拡張子	言語	処理
<code>.f</code>	Fortran 77 または Fortran 95 固定形式	ソースファイルをコンパイルし、オブジェクトファイ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拡張子 <code>.o</code> を付けたもの。
<code>.f95</code> <code>.f90</code>	Fortran 95 自由形式	<code>.f</code> と同じ( <code>f95</code> のみ)。
<code>.for</code>	Fortran 77	<code>.f</code> と同じ。
<code>.F</code>	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。
<code>.F95</code> <code>.F90</code>	Fortran 95 自由形式	Fortran コンパイラでコンパイルする前に、Fortran 95 の自由形式のソースファイルを Fortran または C のプリプロセッサで処理する ( <code>f95</code> のみ)。
<code>.S</code>	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。
<code>.s</code>	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。
<code>.il</code>	インライン展開	インライン展開コードのテンプレートファイルを処理する。コンパイラは、テンプレートを使用して、インライン呼び出しを指定したルーチンに展開する。
<code>.o</code>	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。
<code>.a,</code> <code>.so,</code> <code>.so.n</code>	ライブラリ	ライブラリの名前をリンカーに渡す。 <code>.a</code> ファイルは静的ライブラリ、 <code>.so</code> と <code>.so.n</code> ファイルは動的ライブラリ。

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

## ソースファイル

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

## ソースファイルプリプロセッサ

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

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

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

<http://www.netlib.org/fortran/>

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

## 別々に実行するコンパイルとリンク

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

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

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

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

## コンパイルとリンクの整合性

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

```
-a, -autopar, -Bx, -dbl, -fast, -G, -Lpath, -lname, -mt,
-xmemalign, -nolib, -norunpath, -p, -pg, -xlibmopt,
-xlic_lib=name, -xprofile=p
```

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

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

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

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

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

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

## 認識されないコマンド行引数

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

基本的には次のように区別されます。

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

例：

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

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

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

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

## モジュール (Fortran 95)

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

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

---

## 指令

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

`f95` に固有の指令については付録 C で説明します。

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

---

注 - 指令は Fortran 規格には含まれていません。

---

## 一般的な指令

一般的な Sun の Fortran 指令は次のような書式で使します。

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

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

一般的な指令の構文は、次のとおりです。

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

制限事項は、次のとおりです。

- 最初の 8 文字の後では、空白は無視され、大文字と小文字は区別されません。
- 指令は注釈なので行をまたがって継続することはできません。ただし、1 行で完結している `C$PRAGMA` 行を複数使うことができます。
- 注釈が上記の構文条件を満たしていると、コンパイラが認識できる指令が 1 つまたは複数含まれていることになります。上記の構文条件を満たしていない場合は、警告メッセージが出力されます。
- C プリプロセッサの `cpp` は注釈行または指令行の中でマクロシンボル定義を展開します。Fortran プリプロセッサの `fpp` は注釈行の中でマクロの展開は行いませんが、正当な `f77` と `f95` の指令は認識し、指令キーワード外の置換を制限します。ただし、キーワード `SUN` が必要な指令には注意してください。`cpp` は小文字の `sun`

を事前定義した値で置き換えます。また、`cpp` マクロ `SUN` を定義すると、`SUN` 指令キーワードが干渉されます。一般的な規則では、次のようにソースが `cpp` または `fpp` で処理される場合、プリAGMAは大文字と小文字を混在させて指定します。

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

Fortran のコンパイラは、次の一般的な指令を認識します。

表 2-2 一般的な Fortran 指令

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

## C 指令

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

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

例：C で ABC と XYZ をコンパイルします。

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

## UNROLL 指令

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

C\$PRAGMA SUN UNROLL=*n* 指令は、ループを *n* 回展開するように最適化に指示します。(コンパイラは、解析の結果、ループの展開が適切であると判断した場合のみ展開します)。

*n* には正の整数を指定します。以下の選択肢があります。

- *n*=1 の場合、最適化は、どのループも展開できません。
- *n*>1 の場合、最適化は、ループを *n* 回展開できます。

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

例：ループを 2 回展開するときは、次のように指定します。

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

## WEAK 指令

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

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

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



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

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

## OPT 指令

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

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

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

**-O4** を指定する **f77** コマンドでコンパイルする場合、指令はこのレベルを上書きして **-O2** でサブルーチンをコンパイルします。このルーチンの後に別の指令がない限り、次の副プログラムは **-O4** でコンパイルされます。

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

## PIPELOOP=n 指令 (SPARC のみ)

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

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

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

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

## (SPARC のみ) PREFETCH 指令

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

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

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

## 並列化指令

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

---

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

---

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

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

一方、Cray 形式の並列化指令を使用するには、`-mp=cray` というコンパイラオプションを指定します。これには、`MIC$` というセンチネルが付きます。Sun と Cray では、同じ指令でも解釈の仕方が異なります。詳細は、『Fortran プログラミングガイド』の並列化の章を参照してください。

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

SUN/Cray の並列化指令の構文は以下のとおりです。

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

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

並列化指令では、キーワードの後にオプション修飾子を指定します。

例：共有変数でループを指定：

<code>C\$PAR DOALL SHARED(yvalue)</code>	<i>Sun</i> 形式
<code>CMIC\$ DOALL SHARED(yvalue)</code>	<i>Cray</i> 形式

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

## OpenMP 指令

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

OpenMP 指令を使用可能にするには、`-mp=openmp` または `-openmp` というコマンド行オプションを指定してコンパイルしなければなりません。

OpenMP 指令の概要は、付録 E に記載されています。

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

---

## コンパイラの利用方法

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

## ハードウェアプラットフォームの特定

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

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

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

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

Hostid = hardware_host_id.
```

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

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

## 環境変数の使用

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

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

例: `FLAGS` を設定します (C シェル)。

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

例 : `FFLAGS` を明示的に使用します。

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

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

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

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

---

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

---

## メモリーサイズ

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

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

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

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

## スワップ領域の制限

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

例：`swap` コマンドを使用します。

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

実際の実メモリーの容量は、次のコマンドで確認できます。

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

## スワップ領域の拡大

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

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

## 仮想メモリの制限

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

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

例：仮想メモリーを 16MB に制限します。

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

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

例：仮想メモリーを 16MB に制限します。

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

いずれの場合も、最適化は 16MB 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：32MB のスワップ領域のあるマシンでは、次のコマンドを使用します。

`sh` の場合:

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

`csch` の場合:

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



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

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

```
demo% limit datasize unlimited
```

`sh`、`ksh` の場合：

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

詳細は、『Solaris 64 ビット開発ガイド』を参照してください。



## 第3章

### f77 / f95 コンパイラオプション

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

- コンパイラオプションフラグに使用する構文の説明: 31 ページ
- 機能別のオプションのまとめ: 31 ページ
- 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: 40 ページ

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

## コマンド構文

コンパイラのコマンドの構文は次のとおりです。

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

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

## オプションの構文

オプションの一般的な書式を以下に示します。

表 3-1 オプションの構文

構文の形式	例
<code>-flag</code>	<code>-g</code>
<code>-flagvalue</code>	<code>-Dnostep</code>
<code>-flag=value</code>	<code>-xunroll=4</code>
<code>-flag value</code>	<code>-o outfile</code>

次の表記規則に従って、オプションを説明しています。

表 3-2 オプションの表記規則

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

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

オプションの一般的な規則を以下に示します。

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

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

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

## オプションのまとめ

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

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

表 3-3 機能別コンパイラオプション

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

表 3-3 機能別コンパイラオプション

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

表 3-3 機能別コンパイラオプション

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

表 3-3 機能別コンパイラオプション

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



表 3-3 機能別コンパイラオプション

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

## 頻繁に利用するオプション

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

表 3-4 頻繁に利用するオプション

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

## 下位互換オプションと旧オプション

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

表 3-5 下方互換オプション

目的	オプション
デフォルトのデータサイズを倍長にする。 <code>-xtypemap</code> を使用する。	<code>-r8</code> または <code>-dbl</code>
定数の引数への代入を可能にする。	<code>-copyargs</code>
外部名に下線を使用しない。	<code>-ext_names=e</code>
非標準の算術演算を使用可能にする。	<code>-fnostd</code>
ホストシステムに合わせて最適化を行う。	<code>-native</code>
並びによる旧式の出力を行う。	<code>-olddo</code>
少なくとも 1 回は <code>DO</code> ループを実行する。	<code>-onetrip</code>
SPARC V7 アーキテクチャ用のコンパイル	<code>-cg89</code>
SPARC V8 アーキテクチャ用のコンパイル	<code>-cg92</code>

これらのオプションフラグは推奨していませんので使用しないでください。

## 現在使用されていないオプション

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

表 3-6 現在使用されていないオプション

元の意味	オプション
スレッドアナライザ用にコンパイルします。	<code>-ztha</code>
例外トラップを無効にします (f95)。	<code>-fnonstop</code>

## オプションリファレンス

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

次の表に、オプションの利用の可能性について示します。

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

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

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

- a

`tcov` を使用する、旧式の基本ブロックごとのプロファイリングを行います。

### ● SPARC: [f77/f95](#)

`tcov` を使用する旧式の基本ブロックごとのプロファイルを行います。新しい形式のプロファイルについては、115 ページの「`-xprofile=p`」の `tcov` の説明を参照してください。詳細は、[tcov\(1\)](#) のマニュアルページおよび『プログラムのパフォーマンス解析』を参照してください。

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

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

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

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

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

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

## **-aligncommon [=n]**

COMMON ブロック内のデータの境界整列を指定します。

### ● SPARC: f77/f95

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

たとえば、-aligncommon=4 と指定すると、4 バイト以上の共通ブロック内の全データ要素が、4 バイト境界に整列します。指定のサイズより小さいブロック内のデータは、このオプションの影響を受けず、本来の境界上に位置したままになります。

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

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

## -ansi

規格以外の拡張機能を使用すると、警告メッセージを出力します。

- **SPARC: f77/f95**

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

## -arg=local

ENTRY 文の実引数を保存します。

- **SPARC:f77**

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

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

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

## -autopar

ループの自動並列化を使用可能にします。

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

- **SPARC: f77/f95**

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

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

プログラム中に `libthread` スレッドライブラリへの明示的な呼び出しがある場合は、`-autopar` は使用しないでください。

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

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

`-autopar` を使用してコンパイルとリンクを一度に行う場合、マルチスレッド処理ライブラリとスレッド対応の Fortran 実行時ライブラリが自動的にリンクされます。

`-autopar` を使用してコンパイルとリンクを別々に行う場合は、適切なライブラリにリンクするために、`-autopar` を使用してリンクを行う必要があります。

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

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

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

## `-B{static | dynamic}`

動的または静的のどちらかのライブラリリンクを指定します。

### ● SPARC: f77/f95

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

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

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

以下の点にも注意してください。

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

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

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

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

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

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

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

## -C

配列の引用で範囲を超えていないかを検査します。

### ● SPARC:f77/f95

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

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



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

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

## `-c`

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

### ● SPARC: f77/f95

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

## `-cg89`

一般的な SPARC アーキテクチャ用にコンパイルを行います。(廃止)

### ● SPARC: f77/f95

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

## `-cg92`

SPARC V8 アーキテクチャ用にコンパイルを行います。(廃止)

### ● SPARC: f77/f95

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

## `-copyargs`

定数の引数へ代入を行えるようにします。

## ● SPARC:f77/f95

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

- `-copyargs` を指定しない場合、定数の引数をサブルーチンに渡し、そのサブルーチン内でその定数を変更しようとする、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。

## -Dname [=def]

プリプロセッサのシンボル名を定義します。

## ● SPARC: f77/f95

このオプションは `.F` と `.f95` ソースファイルだけに適用します。

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

`-Dname` : シンボル `name` を 1 と定義します。

コマンド行では、ソースファイル中に

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

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

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

次は事前定義された値です (シンボルの前に二重の下線を付けます) :

- 製品バージョンは、`__SUNPRO_F77`、`__SUNPRO_F90`、`__SUNPRO_F95` に 16 進数で事前定義されています。たとえば、Sun WorkShop 6 のリリースの場合、`__SUNPRO_F77` は、`0x600` となります。
- 次の値は、該当するシステム上でそれぞれ事前定義されています。

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

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

```
#ifdef __sparc
```

- `sparc`、`unix`、`sun` は、下線なしで事前定義されていますが、将来のリリースで削除される可能性があります。
- SPARC V9システムでは、`__sparcv9` マクロも定義されています。

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

## **-dalign**

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

### ● **SPARC: f77/f95**

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

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

---

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

---

`-dalign` は、`-xmemalign=8s -aligncommon=8` と等価のマクロです。112 ページの「`-xmemalign[=<a><b>]`」を参照してください。

`-dbl` と `-dalign` の両方を使用すると、デフォルトの `INTEGER` 変数が 8 バイトに境界整列され 64 ビットになります。これは、次のオプションと同義になります。

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

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

## `-db`

オプションの CIF ファイルを生成します。

- **SPARC:f95**

拡張子 `.T` を付けて、オプションのコンパイラ情報ファイル (CIF) を生成します。このファイルは、Sun WorkShop ソースブラウザで必要になることがあります。CIF ファイルは、`-xlist` オプションを使用すると、自動的に生成されます。`-db` オプションは、`-xlist` を指定せずに CIF ファイルを生成し直す場合に使用できます。

## `-dbl`

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

- **SPARC:f77**

---

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

---

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

表 3-7 デフォルトのデータサイズ (バイト) と `-dbl` オプション

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

このオプションは、変数、パラメータ、定数、および関数に適用されます。

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

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

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

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

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

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

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

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

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

### **-dbl\_align\_all={yes|no}**

8 バイトの境界上ですべてのデータを強制的に整列します。

- **SPARC: f77/f95**

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

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

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

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

使用した場合、すべてのルーチンをこのフラグでコンパイルする必要があります。

### **-depend**

ループを解析してデータ依存を調べます。

- **SPARC: f77/f95**

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。このオプションを指定すると、最適化レベルが指定されていない場合または O3 以下の場合、自動的に最適化レベルが O3 に設定されます。`-depend` は、`-fasti`、`-autopar` および

`-parallel` オプションでも行われます。(詳細は、『Fortran プログラミングガイド』を参照してください)。

### **-dryrun**

コンパイラ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、実行はしません。

- **SPARC: f77/f95**

デバッグ時に便利です。このオプションにより、コンパイル時に実行されるコマンドが表示されます。

## `-d{y|n}`

実行可能形式全体を生成する時のライブラリのリンク形式 (動的または静的)。

### ● SPARC: f77/f95

- `-dy` : 実行形式は動的にリンクされたライブラリを使用します (共有ライブラリを使用できます)。
- `-dn` : 実行形式は静的にリンクされたライブラリを使用します (共有ライブラリを使用しません)。

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

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

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

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

## `-e`

拡張された入力ソース行を受け付けます。

### ● SPARC: f77/f95

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

## **-erroff=taglist**

taglist 名で指定した警告メッセージの表示を抑制します。

- **SPARC:f77/f95**

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

例：

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

-errtags オプションを使用して、警告メッセージに関連付けられているタグ名を表示します。

## **-errtags**

各警告メッセージがメッセージタグ付きで表示されます。

- **SPARC:f77/f95**

このオプションを付けると、コンパイラの内部エラータグ名が警告メッセージとともに表示されます。デフォルトでは、タグは表示されません。

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

## **-explicitpar**

指令で明示的に示されたループを並列化します。

- **SPARC: f77/f95**

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



このオプションを使用すると、Sun、Cray、OpenMP といった明示的並列化指令が有効になります。並列化指令の直前にある DO ループには、スレッド化されたコードが生成されます。並列化はマルチプロセッサシステムの場合のみ適しています。このオプションは、すでに `libthread` ライブラリへの呼び出しによって、独自にマルチスレッド処理を行なったプログラムをコンパイルする場合には使用できません。

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

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

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

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

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

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

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

## `-ext_names=e`

外部名に下線を付けるかどうかを指定します。

### ● SPARC:f77/f95

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

`-ext_names=plain`: 下線を付けません。

`-ext_names=underscore`: 下線を付けます。

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

## -F

ソースファイルプリプロセッサを起動します。ただしコンパイルは行いません。

### ● SPARC: f77/f95

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

例：

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

を実行すると、ソースファイルが `source.f` に書き込まれます。

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

## -f

COMMON ブロックのデータを整列します。

### ● SPARC: f77/f95

COMMON ブロックの倍精度および 4 倍精度のデータを境界整列します。

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

---

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

---

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

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

このオプションを単独で使用すると、コンパイラで倍精度および 4 倍精度のデータに対して高速のマルチワードのフェッチ/ストア命令を生成することはできません。

`-dalign` がこれを実行し、`-f` も呼び出します。`-f` よりも `-dalign` を使用することをお勧めします。詳細は、47 ページの「`-dalign`」を参照してください。これは、`-dalign` が `-f` も同様に `-fast` オプションの一部であるからです。

## `-fast`

最適なオプションを選択して、実行可能ファイルの実行速度を上げます。

### ● SPARC: f77/f95

過度にコンパイルに時間をかけずに、実行速度が速くなるように最適化を行うオプションを選択します。これにより、多くのアプリケーションのパフォーマンスが最大に近くなります。

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

`-fast` では、次のオプションを選択します。

- `-dalign`
- `-depend`
- `-fns`
- `-fsimple=2`
- `-ftrap=%none` (f77) または `-ftrap=common` (f95)
- `-libmil`
- `-f` (f95 のみ)
- `-xtarget=native`
- `-O5`
- `-xlibmopt`

- `-pad=common`
- `-xvector=yes`

---

注 – このオプションは、他のオプションをいくつか組み合わせたもので、その組み合わせは、各リリースやコンパイラの種類によって異なります。`-fast` で選択されたオプションは、プラットフォームによっては動作しないことがあります。

---

`-fast` が、選択するオプションの詳細は以下のとおりです。

- `-xtarget=native` オプション

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

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

- `-O5` 最適化レベルオプション (旧コンパイラリリースでは、`-fast` に `-O3` または `-O4` を設定していました)。

- `-depend` オプション (SPARC のみ)

- システムが提供するインライン展開テンプレート用の `-libmil` オプション

例外処理を使用する C モジュールでは、`-fast` の後に `-nolibmil` 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。61 ページを参照してください (旧リリースでは、`-fast` は `-fsimple=1` を設定していました)。

- 倍長ロードとストアを生成する `-dalign` オプション (SPARC のみ)

このオプションを使用すると、標準外の形式で 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 (SPARC のみ)

- `-pad=common` は、キャッシュの利用률을改善するために、適宜共通ブロック内の変数の間にパディングを挿入します (旧リリースの `-fast` では `-pad=common` を設定していませんでした)。

- `-xvector=yes` は、DO ループ内のある特定の数学ライブラリ呼び出しを、同等のベクトル化されたライブラリルーチンの単一呼び出しに変換します (旧リリースの `-fast` では、`-xvector=yes` を設定していませんでした)。
- `-fns -ftrap=%none` は、Fortran 77 の全トラッピングをオフにします。共通の浮動小数点例外のトラッピングは、Fortran 95 のデフォルトです。
- `f95` では、`-f` オプションが追加されました。

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

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

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

## `-fixed`

固定書式の Fortran 95 ソース入力ファイルを指定します。

- **SPARC: f95**

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

## `-flags`

`-help` と同義です。

- **SPARC: f77/f95**

## `-fnonstd`

浮動小数点算術ハードウェアの非標準の初期化を行います。

- **SPARC: f77/f95**

このオプションは、以下のオプションフラグの組み合わせと同義です。

- SPARC: `-fns -ftrap=common`

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

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

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

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

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

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

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

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

## `-fns [= {no | yes}]`

SPARC の非標準の浮動小数点モードを選択します。

### ● SPARC: f77/f95

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

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

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

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

表 3-8 非正規数 REAL と DOUBLE

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

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

デフォルトでは、浮動小数点は標準の設定に初期化されます。

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

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

### `-fpovery [= {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。

#### ● SPARC: f77/f95

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

## -fpp

`fpp` を使用して、入力の前処理を強制的に行います。

- **SPARC: f95**

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

## -free

自由書式のソース入力ファイルを指定します。

- **SPARC:f95**

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

## -fround=*r*

起動時に IEEE の丸めモードを有効にします。

- **SPARC: f77/f95**

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

デフォルトは `-fround=nearest` です。

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

このオプションは、IEEE 754 の丸めモードを以下のように設定します。

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

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



## `-fsimple [=n]`

浮動小数点の最適化の設定を選択します。

### ● SPARC: f77/f95

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

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

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

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

別の浮動小数点単純化レベルは次のとおりです。

`-fsimple=0`

前提を単純化しません。IEEE 754 に厳密に準拠してください。

`-fsimple=1`

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

`-fsimple=1` を指定すると、最適化は以下のことを前提とします。

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

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

`-fsimple=2`

より広範囲の浮動小数点の最適化を行います。丸めが変わると、多くのプログラムにおいて、生成される数値結果が異なることがあります。

たとえば、`z=1/y` が 1 回は計算され一時ファイルに保存されている場合は、`-fsimple=2` を指定すると最適化は `x/y` の反復演算を `x*y` で置き換えることができ、除算は削除されます。

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

`-fast` では、`fsimple=2` を設定します。

## `-fttrap=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。

### ● SPARC: f77/f95

`t` には、以下のうち 1 つまたは複数の項目をコンマで区切って指定します。

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

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

`%` は省略できません。

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

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

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

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

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

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

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

## -G

実行可能ファイルの代わりに、動的共有ライブラリを構築します。

### ● SPARC: f77/f95

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

## -g

デバッガを使用できるようにコンパイルします。

### ● SPARC: f77/f95

`dbx(1)` と Sun WorkShop のデバッグユーティリティを使用してデバッグできるように、シンボルテーブル情報を生成します。

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

`-g` とともに指定したオプションの機能が制限される場合があります。

詳細に関しては、『`dbx` コマンドによるデバッグ』を参照してください。

SPARC の場合:

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

## **-hname**

生成する動的共有ライブラリの名前を指定します。

- **SPARC: f77/f95**

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

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

**-hname** 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、実行時リンクの際に行うライブラリの検索が、より柔軟になります。このオプションは、共有ライブラリのバージョンを指定する場合にも使用できます。

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

## **-help**

コンパイルオプションの一覧を表示します。

- **SPARC: f77/f95**

オプションの要約リストを表示します。105 ページの「`-xhelp=h`」も参照してください。

## **-Idir**

インクルードファイルの検索パスに `dir` を追加します。

- **SPARC: f77/f95**

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

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

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

```
demo% f77 -I/usr/app/include growth.F
```

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

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

1. ソースファイルがあるディレクトリ
2. `-I dir` オプションで指定したディレクトリ
3. デフォルトのリストにあるディレクトリ

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

`f77` の場合

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

`f95` の場合

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

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

-i2

デフォルトの整数サイズを 2 バイトに設定します。

- SPARC: `f77`

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

## `-i4`

デフォルトの整数サイズを 4 バイトに設定します。

### ● SPARC:f77

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

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

```
demo% f77 -dbl -i4 *.f
f77: 警告: 廃止予定の -r8、-dbl には -xtypemap を使います; f77 マニュアルページを参照
コマンド行の警告: -i4 により -dbl の整数部が上書きされます。
```

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

指定のルーチンのインライン化を有効または無効にします。

### ● SPARC: f77/f95

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

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

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

`-inline` と `-O4` を同時に指定すると、コンパイラが通常実行する自動インライン化が無効になります。ただし、これは、`%auto` に `-inline` を指定していない場合に限ります。

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を示します。ただし、条件が満たされていなくても、警告メッセージは出力されません。

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

なお、`-O4` を指定すると、コンパイラは通常、ユーザー作成のサブルーチンや関数をすべてインライン化しようとします。`-O4` に `-inline` を追加すると、最適化はリスト中にあるルーチンに限ってインライン化を行うため、実際にはパフォーマンスが低下します。この場合、`%auto` サブオプションを使用して、`-O4` および `-O5` で自動インライン化を有効にします。

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

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

## **-Kpic**

`-pic` と同義です。

- SPARC: f77/f95

## **-KPIC**

`-PIC` と同義です。

- SPARC: f77/f95

## **-Ldir**

ライブラリ検索ディレクトリのリストに `dir` を追加します。

- **SPARC: f77/f95**

オブジェクトライブラリの検索ディレクトリのリストの先頭にディレクトリ `dir` を追加します。`-L` と `dir` の間の空白文字はあってもなくてもかまいません。このオプションはリンカーに渡されます。68 ページの「`-lx`」を参照してください。

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

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

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

---

注 - `-Ldir` 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。

---

## **-lx**

リンカー検索ライブラリのリストに、ライブラリ `libx.a` を追加します。

- **SPARC: f77/f95**

`-lx` をリンカーに渡して、`ld` が未解決の参照を検索するためのライブラリを追加指定します。オブジェクトライブラリ `libx` をリンクします。共有ライブラリ `libx.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 `libx.a` を使用します。共有ライブラリを使用する場合は、名前は `a.out` に組み込まれます。`-l` と `x` の間には空白文字を入れないでください。



例: ライブラリ `libV77` をリンクします。

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

複数のライブラリとリンクするには、`-lx` を再度使用してください。

例: ライブラリ `liby` と `libz` をリンクします。

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

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

## `-libmil`

最適化として `libm` ライブラリルーチンをインライン化します。

- **SPARC: f77/f95**

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

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

## `-loopinfo`

並列化の結果を表示します。

- **SPARC: f77/f95**

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

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

`-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% f77 -o shalow -fast -parallel -loopinfo shalow.f
shalow.f:
  MAIN shalow:
    inital:
    calc1:
    ...etc
"shalow.f", 78 行目: 並列化されません、安全性が疑わしい呼び出し
"shalow.f", 172 行目: 並列化されず
"shalow.f", 173 行目: 並列化されません、利得なし
"shalow.f", 181 行目: 並列化されず、融合
"shalow.f", 182 行目: 並列化されません、利得なし
"shalow.f", 226 行目: 並列化されず、逐次版が生成されました
"shalow.f", 227 行目: 並列化されません、利得なし
...etc
```

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

例: 標準エラーを `error` ユーティリティに渡します。

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

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

## **-mdir**

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

### ● SPARC:f95

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

`-M` で指定したディレクトリは、現在のディレクトリの後に検索されます。モジュールのあるソースファイルをコンパイルすると、検出された `MODULE` ごとに `.mod` モジュールファイルが生成されます。Fortran 95 モジュールについての詳細は、172 ページの「モジュールファイル」を参照してください。

## `-misalign`

境界整列に失敗したデータを受け付けます。

### ● SPARC: f77

`-misalign` オプションは、通常であればエラーになるようなメモリー境界整列に失敗したデータを受け付けます。COMMON 文および EQUIVALENCE 文の使用法によっては、データの境界整列が失敗する場合があります (コンパイラの診断時)。

`-misalign` オプションを使用すれば、コンパイラは作為的な境界整列の失敗を受け入れ、COMMON ブロックに本来のデータの境界整列を保証するためのパディングを挿入しません。しかし、これによってパフォーマンスはかなり低下します。このオプションを使用する代わりに、データが正しく整列されるようにコードを書き直すことをお勧めします。

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

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

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

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

並列化指令の形式を選択します。

### ● SPARC: f77/f95

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

---

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

---

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

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

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

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

78 ページの「`-openmp`」も参照してください。

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

これらの [f77/f95](#) 指令については、このマニュアルの付録 E にまとめられています。

詳細は、『Fortran プログラミングガイド』の並列化に関する説明を参照してください。

## `-mt`

マルチスレッド環境で使用しても安全なライブラリが必要になります。

### ● SPARC: [f77/f95](#)

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

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

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

次の点に注意してください。

- `-mt` を使用するときは、入出力を伴う関数の副プログラム自体を入出力文の一部として引用することはできません。このような入出力は再帰入出力と呼ばれ、デッドロックの原因になることがあります。

- 一般的な注意として、ユーザー独自でマルチスレッド化したコードを `-autopar`、`-explicitpar`、または `-parallel` オプションを付けてコンパイルしないでください。コンパイラが生成するスレッドライブラリへの呼び出しとプログラム自体の呼び出しが衝突して、予測できない結果になることがあります。
- シングルプロセッサシステムの場合に、`-mt` オプションを使用すると、パフォーマンスが低下することがあります。

## `-native`

使用中のマシンに最適なパフォーマンスにします。(廃止)

- **SPARC: f77/f95**

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

## `-noautopar`

自動並列化を行いません。

- **SPARC: f77/f95**

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

## `-nodepend`

コマンド行で指定した `-depend` を取り消します。

- **SPARC: f77/f95**

コマンド行で先に指定された `-depend` を取り消します。

## `-noexplicitpar`

明示的な並列化を行いません。

- **SPARC: f77/f95**

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

## **-nolib**

システムライブラリとリンクしません。

- **SPARC: f77/f95**

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

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

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

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

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

`f95` では、`libm` を静的にリンクし、`libc` を動的にリンクします。

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

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

## **-nolibmil**

コマンド行の `-libmil` を取り消します。

- **SPARC: f77/f95**

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

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

## `-noqueue`

ライセンスのキューイングを使用しません。

- **SPARC: f77/f95**

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

## `-noreduction`

コマンド行の `-reduction` を取り消します。

- **SPARC: f77/f95**

`-reduction` は、別の並列化オプションとともに使用します。このオプションにより、`-reduction` オプションが取り消されます。

## `-norunpath`

実行可能ファイル中に、実行時共有ライブラリのパスを設定しません。

- **SPARC: f77/f95**

コンパイラは通常、実行時リンカーが共有ライブラリを検索する位置を示すパスを実行可能ファイル中に設定します。このパスはインストールの形式によって異なります。

`-norunpath` オプションは、実行可能ファイルにパスが組み込まれないようにします。

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

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

## -O [n]

実行時間を最適化します。

### ● SPARC: f77/f95

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

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

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

-g オプションは -On を抑制しませんが、-On 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。

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

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

## -O

-O3 と同義です。

## -O1

文レベルの最小限の最適化を行います。

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



## -O2

基本ブロックレベルの最適化を行います。

通常、生成されるコードのサイズが最も小さくなります (119 ページの「-xspace」を参照してください)。

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

## -O3

関数レベルで、ループを展開し大域的に最適化を行います。

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

## -O4

同一ファイル内にあるルーチンを自動的にインライン化します。

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

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

## -O5

最高レベルの最適化を行います。

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

プロファイルのフィードバックと併せて使用すると、最適化がパフォーマンスの向上につながる可能性が高まります。115 ページの「-xprofile=p」を参照してください。

## -o *name*

書き込み先の実行可能ファイルの名前を指定します。

### ● SPARC: f77/f95

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

## `-oldldo`

旧式のリスト形式の出力を選択します。

- **SPARC: f77/f95**

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

## `-onetrip`

`DO` ループを 1 回だけ実行します。

- **SPARC: f77/f95**

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

## `-openmp`

Fortran 95 の OpenMP 指令で明示的な並列化を有効にします。

- **SPARC: f77/f95**

このマクロは、次に示すオプションを組み合わせたマクロです。

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

OpenMP 指令については、付録 E にまとめられています。

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

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

## -p

`prof` プロファイラを使用するプロファイル用にコンパイルします。

### ● SPARC: f77/f95

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

## -pad [=p]

キャッシュを効率よく利用するためにパディングを挿入します。

### ● SPARC:f77/f95

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

`-pad[=p]` で `p` を指定する場合は、以下のいずれか (または両方) を指定します。

---

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

---

`-pad` のデフォルトは以下のとおりです。

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

以下に示すコマンド行は同義です。

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

`-pad[=p]` オプションは、以下の条件を満たす項目に適用されます。

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

局所変数、または静的変数については、88 ページの「`-stackvar`」の説明を参照してください。

`-pad=common` に関する制限事項

- 配列と文字列のどちらに対しても `EQUIVALENCE` 文を適用できません。
- ある共通ブロックを引用するファイルのコンパイルで `-pad=common` を指定するときは、その共通ブロックを引用するすべてのファイルのコンパイルで `-pad=common` を指定する必要があります。このオプションは、共通ブロック内の変数の配置を変更します。あるプログラム単位をこのオプション付きでコンパイルし、別のプログラム単位をこのオプションなしでコンパイルすると、共通ブロック内の同じ位置への引用が、別の位置を引用してしまう可能性が生じます。
- `-pad=common` を指定する場合、別のプログラム単位にある共通ブロックの変数宣言を名前を除いて同じにする必要があります。共通ブロックの変数の間に挿入されるパディングの量は、このような変数の宣言内容に応じて異なります。別のプログラム単位にある変数のサイズやランクが異なる場合は、同じファイル内でも変数の位置が異なることがあります。
- `-pad=common` を指定する場合、共通ブロック変数を伴う `EQUIVALENCE` を宣言すると、エラーになります。

## `-parallel`

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

- **SPARC: f77/f95**

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

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

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

ユーザー独自のスレッド管理を行なっている場合は、`-parallel` は使用しないでください。71 ページの `-mp` の説明を参照してください。

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

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

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

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

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

-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。

- **SPARC: f77/f95**

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

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

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

## `-pic`

共有ライブラリ用の位置独立コードをコンパイルします。

### ● SPARC: f77/f95

このオプションでコンパイルしたコードは動的共有ライブラリ用です。大域的なデータへの引用はそれぞれ、大域的なオフセットテーブル内でポインタの間接引用として生成されます。関数呼び出しは、プロシージャリンクージテーブルを通して `pc` 相対アドレッシングモードで生成されます。

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

`-pic` は `-xcode=pic13` と同義です。

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

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

以上のパフォーマンス上の犠牲を比較した場合、`-pic` と `-PIC` を使用すると、ライブラリコードが共有されるため、必要なシステムメモリーが大幅に少なくなります。`-pic` または `-PIC` でコンパイルした共有ライブラリ中のコードの各ページは、そのライブラリを使用する各プロセスどうしで共有することができます。共有ライブラリ

中にあるコードのページに 1 つでも `-pic` でコンパイルされていないメモリ参照 (直接メモリ参照) があると、このページは共有できなくなるため、ライブラリを使用したプログラムを実行するたびに、そのページのコピーが作成されます。

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

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

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

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

64 ビットの Solaris 7 で `-xarch=v9` または `v9a` を使用して共有動的ライブラリを構築する場合、`-pic` または `-PIC` オプション、あるいはこれらと同義の `-xcode` オプションを指定する必要があります。

## -PIC

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

- **SPARC: f77/f95**

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

`-PIC` は `-xcode=pic32` と同義です。

64 ビットの Solaris 7 で `-xarch=v9` または `v9a` を使用して共有動的ライブラリを構築する場合、`-pic` または `-PIC` オプション、あるいはこれらと同義の `-xcode` オプションを指定する必要があります。

## -Qoption *pr ls*

オプションをコンパイル段階 *pr* に渡します。

- SPARC: f77/f95

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

このオプションは主に、サポートスタッフによる内部デバッグ用として用意されています。LD\_OPTIONS 環境変数を使用してリンカーにオプションを渡します。

『Fortran プログラミングガイド』のリンクとライブラリに関する章を参照してください。

## -qp

-p と同義です。

- SPARC: f77/f95

## -R *ls*

動的ライブラリの検索パスを実行可能ファイルに設定します。

- SPARC: f77/f95

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

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

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

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

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



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

標準のインストール: `/opt/SUNWspr/lib`

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

## -r8

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

### ● SPARC:f77/

---

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

---

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

表 3-9 デフォルトのデータサイズ (バイト) と `-r8`

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

このオプションは、変数、パラメータ、定数、および関数に適用されます。

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

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

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

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

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

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

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

`-r8` と `-i2` を両方指定した場合の結果は予測できません。

## `-r8const`

単精度の定数を `REAL * 8` の定数に変換します。

- **SPARC: f77/f95**

単精度の `REAL` 定数はすべて `REAL * 8` に変換されます。倍精度 (`Real * 8`) 定数は変更されません。このオプションは、定数にだけ適用されます。定数と変数の両方を変換する場合は、121 ページの「`-xtypemap=spec`」を使用してください。

## `-reduction`

ループ中にある縮約演算を識別します。

- **SPARC: f77/f95**

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

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

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

例：`-reduction` オプションを付けた自動並列化処理

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

**-s**

コンパイルし、アセンブリのソースコードだけを生成します。

- **SPARC: f77/f95**

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

**-s**

実行可能ファイルからシンボルテーブルを取り除きます。

- **SPARC: f77/f95**

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

**-sb**

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

- **SPARC:f77/f90**

詳細は『Sun WorkShop の概要』を参照してください。

---

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

---

## `-sbfast`

ソースコードブラウザ用のテーブル情報のみを生成します。

- **SPARC:f77/f95**

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

## `-silent`

コンパイラメッセージの出力を抑制します。

- **SPARC:f77/f95**

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

## `-stackvar`

局所変数すべてを強制的にメモリースタックに割り当てます。

- **SPARC: f77/f95**

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

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

次の条件に該当しない場合は、変数と配列は局所的となります。

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

REAL X/8.0/ または DATA X/8.0/

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

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

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

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

例: 現在のメインスタックのサイズを表示します。

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

例：メインスタックのサイズを 64MB 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 8MB に設定します。

```
demo% setenv STACKSIZE 8192
```

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

## **-stop\_status=yn**

`STOP` 文により整数のステータス値を返します。

### ● SPARC:f77/f95

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

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

```
STOP 123
```

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

```
STOP 'stop string'
```

は受け付けられません。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 (`cs`) では `$status`、また Bourne (`sh`) シェルと Korn (`ksh`) シェルでは `$?` です。

## **-temp=dir**

一時ファイルのディレクトリを設定します。

### ● SPARC: f77/f95

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

#### **-time**

各コンパイル段階の経過時間を表示します。

- **SPARC: f77/f95**

各コンパイル段階で費された時間とリソースが表示されます。

#### **-U**

ソースファイル中の大文字と小文字を区別します。

- **SPARC:f77/f95**

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

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

#### **-u**

未宣言の変数に対してメッセージを出力します。

- **SPARC:f77/f95**

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

#### **-unroll=n**

`DO` ループの展開が可能な箇所、使用可能にします。

- **SPARC:f77/f95**

`n` は正の整数です。次の選択が可能です。

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

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

## -v

各コンパイラパスの名前とバージョンを表示します。

- **SPARC: f77/f95**

コンパイラの実行時に、各パスの名前とバージョンを表示します。

上記の情報は、問題が発生した場合にご購入先に問い合わせる時に役立ちます。

## -v

各コンパイラパスの詳細情報を表示します。

- **SPARC: f77/f95**

`-v` と同様に、コンパイラの実行時にそれぞれのパス名を表示し、ドライバが使用したオプションと環境変数を詳細に表示します。

## -vax=v

有効にする VMS Fortran の拡張機能を指定します。

- **SPARC:f77/**

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

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

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



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

以下の表に選択できるサブオプションを示します。

表 3-10 `-vax` のサブオプション

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

`%all` と `%none` を使用して、これらのサブオプションを全部選択する、またはまったく選択しない、のいずれかを指定できます。

サブオプションは、リスト `v` が左から右に読み込まれるに従って蓄積されます。たとえば、1 つの機能を除いてすべてを使用可能にするには次のように指定します。

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

71 ページの「`-misalign`」を参照してください。

## `-vpara`

並列化に関する詳細な警告メッセージを表示します。

### ● SPARC: f77/f95

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

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

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

**-w**

警告メッセージを抑制します。

- **SPARC: f77/f95**

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

例：`-w` オプションを使用しても警告メッセージが出力される場合

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

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

**-xa**

`-a`と同義です。

- **SPARC: f77/f95**

## **-xarch=isa**

命令セットアーキテクチャ (ISA) を指定します。

表 3-11 -xarch ISA キーワード

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

表 3-11 に、-xarch キーワード *isa* で受け付けられるアーキテクチャを示します。

-xarch は単独でも使用できますが、-xtarget オプションが展開される一部であり、これを使用すると、特定の -xtarget オプションで設定した -xarch 値を無効にすることができます。たとえば、次のように指定すると、

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

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

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

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

## SPARC プラットフォームでの `-xarch` の使用

表3-12 に、もっとも一般的な `-xarch` オプションについてまとめます。

表 3-12 SPARC プラットフォーム上でのもっとも一般的な `-xarch` オプションのまとめ

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

次の点にも注意してください。

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

- `v9`、`v9a` および `v9b` を指定してコンパイルしたオブジェクトバイナリファイル (`.o`) は、リンクし、まとめて実行することができますが、実行は SPARC V9b 互換プラットフォーム上だけに限ります。

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

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

表 3-13 SPARC プラットフォーム上で使用できる各 `-xarch` 値

<code>-xarch</code>	意味
<code>generic</code>	たいてい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 これは、デフォルトです。このオプションは、ほとんどのプロセッサ上で良好なパフォーマンスを得るために最善の命令セットを使用します、このとき、どのプロセッサ上でも大きなパフォーマンスの低下は見られません。「最善」の命令セットの定義は、新リリースごとに適宜調整できます。
<code>native</code>	当該システム上で良好なパフォーマンスを得るためのコンパイル これは、 <code>-fast</code> オプションのデフォルトです。コンパイラは、コンパイルが実行されている現在のシステムプロセッサに適した設定を選択します。
<code>v7</code>	SPARC-V7 ISA 用のコンパイル コンパイラは、V7 ISA 上で良好なパフォーマンスが得られるようにコードを生成することができます。これは、V8 ISA で良好なパフォーマンスを得るために最善の命令セットを使用するのに相当します。なお、V7 ISA には <code>mul</code> および <code>div</code> という整数命令、および <code>fsmuld</code> 命令は含まれません。 例：SPARCstation1、SPARCstation2
<code>v8a</code>	SPARC-V8 ISA のV8a バージョン用のコンパイル 定義によれば、V8a は、V8 ISA を意味しますが、 <code>fsmuld</code> 命令は含まれていません。このオプションを使用すると、コンパイラは、V8a ISA 上で良好なパフォーマンスが得られるようにコードを生成することができます。 例：microSPARC Iチップアーキテクチャに基づく任意のシステム

表 3-13 SPARC プラットフォーム上で使用できる各 `-xarch` 値 (続き)

<code>-xarch</code>	意味
<code>v8</code>	<p>SPARC-V8 ISA 用のコンパイラ</p> <p>コンパイラは、V8 上で良好なパフォーマンスが得られるようにコードを生成することができます。</p> <p>例：SPARCstation10</p>
<code>v8plus</code>	<p>SPARC-V9 ISA のV8plus バージョン用のコンパイラ</p> <p>定義によれば、V8plus は、V9 ISA を意味しますが、V8plus ISA アーキテクチャで定義されている 32 ビットサブセットだけに限定され、Visual 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。</p> <ul style="list-style-type: none"> <li>このオプションを使用すると、コンパイラは、V8plus ISA 上で良好なパフォーマンスが得られるようにコードを生成することができます。</li> <li>作成されるオブジェクトコードは、SPARC -V8 + ELF32 形式であり、Solaris UltraSPARC 環境でだけ動作します。すなわち、V7 または V8 プロセッサ上では動作しません。</li> </ul> <p>例：UltraSPARC チップアーキテクチャに基づく任意のシステム</p>
<code>v8ppplusa</code>	<p>SPARC-V9 ISA のV8ppplusa バージョン用のコンパイラ</p> <p>定義によれば、V8ppplusa は、V8plus アーキテクチャを意味しますが、Visual Instruction Set (VIS) バージョン 1.0 および UltraSPARC 拡張機能も含まれます。</p> <ul style="list-style-type: none"> <li>このオプションを使用すると、コンパイラは、UltraSPARC アーキテクチャ上で良好なパフォーマンスが得られるようにコードを生成できますが、V8plus の仕様で定義されている32 ビットのサブセットに限定されます。</li> <li>作成されるオブジェクトコードは、SPARC -V8 + ELF32 形式であり、Solaris UltraSPARC 環境でだけ動作します。すなわち、V7 または V8 プロセッサ上では動作しません。</li> </ul> <p>例：UltraSPARC チップアーキテクチャに基づく任意のシステム</p>

表 3-13 SPARC プラットフォーム上で使用できる各 `-xarch` 値 (続き)

<code>-xarch</code>	意味
<code>v8plusb</code>	<p>UltraSPARC-III 拡張機能付きの SPARC-V8plus ISA の V8plusb バージョン用のコンパイラ</p> <p>コンパイラは、Visual Instruction Set (VIS) バージョン 2.0 および UltraSPARC-III 拡張機能付きの UltraSPARC アーキテクチャで良好なパフォーマンスが得られるようにコードを生成することができます。</p> <ul style="list-style-type: none"> <li>作成されるオブジェクトコードは、SPARC -V8 + ELF32 形式であり、Solaris UltraSPARC-III 環境でしか動作しません。</li> <li>このオプションを使用してコンパイルすると、UltraSPARC-III アーキテクチャで良好なパフォーマンスが得られるように、最善の命令セットが使用されます。</li> </ul>
<code>v9</code>	<p>SPARC-V9 ISA 用のコンパイラ</p> <p>コンパイラは、V9 SPARC アーキテクチャ上で良好なパフォーマンスが得られるようにコードを生成することができます。</p> <ul style="list-style-type: none"> <li>作成される <code>.o</code> オブジェクトファイルは、ELF64 形式であり、同じ形式の他の SPARC -V9 オブジェクトファイルとだけリンクできます。</li> <li>作成される実行可能プログラムは、64 ビットカーネルの 64 ビット目がオンにされた Solaris オペレーティング環境を実行している UltraSPARC プロセッサ上でだけ実行できます。</li> <li><code>-xarch=v9</code> が使用できるのは、64 ビット目がオンの Solaris 環境でコンパイルする場合だけです。</li> </ul>
<code>v9a</code>	<p>UltraSPARC 拡張機能付きの SPARC-V9 ISA 用のコンパイラ</p> <p>SPARC -V9 ISA に Visual Instruction Set (VIS) および UltraSPARC プロセッサに固有の拡張機能を追加し、V9 SPARC アーキテクチャ上で良好なパフォーマンスが得られるように、コンパイラがコードを生成できるようにします。</p> <ul style="list-style-type: none"> <li>作成される <code>.o</code> オブジェクトファイルは、ELF64 形式であり、同じ形式の他の SPARC -V9 オブジェクトファイルとだけリンクできます。</li> <li>作成される実行可能プログラムは、64 ビットカーネルの 64 ビット目がオンにされた Solaris オペレーティング環境を実行している UltraSPARC プロセッサ上でだけ実行できます。</li> <li><code>-xarch=v9a</code> が使用できるのは、64 ビット目がオンの Solaris 環境でコンパイルする場合だけです。</li> </ul>

表 3-13 SPARC プラットフォーム上で使用できる各 `-xarch` 値 (続き)

<code>-xarch</code>	意味
<code>v9b</code>	<p>UltraSPARC-III 拡張機能付きの SPARC-V9 ISA 用のコンパイル SPARC -V9 ISA の V9a バージョンに UltraSPARC-III の拡張機能および VIS バージョン 2.0 を追加します。このオプションを指定してコンパイルすると、Solaris UltraSPARC-III 環境で良好なパフォーマンスが得られるように、最善の命令セットが使用されます。</p> <ul style="list-style-type: none"> <li>• 作成される <code>.o</code> オブジェクトファイルは、SPARC-V9 ELF64 形式であり、同じ形式の他の SPARC -V9 オブジェクトファイルとだけリンクできます。</li> <li>• 作成される実行可能プログラムは、64 ビットカーネルの64 ビット目がオンにされた Solaris オペレーティング環境を実行している UltraSPARC-III プロセッサ上でだけ実行できます。</li> <li>• <code>-xarch=v9b</code> が使用できるのは、64 ビット目がオンの Solaris オペレーティング環境でコンパイルする場合だけです。</li> </ul>

## `-xautopar`

`-autopar` と同義のオプションです。

- **SPARC: f77/f95**

## `-xcache=c`

オブティマイザ用のキャッシュ特性を定義します。

- **SPARC: f77/f95**

`c` には以下のいずれかを指定します。

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

`si`、`li`、`ai` の定義は以下のとおりです。

`si` レベル  $i$  のデータキャッシュのサイズ (キロバイト単位)

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

`ai` レベル  $i$  のデータキャッシュの結合性



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

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

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

表 3-14 `-xcache` の値

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

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

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

### `-xcg89`

`-cg89` と同義です。

- SPARC: f77/f95

### `-xcg92`

`-cg92` と同義です。

- SPARC: f77/f95

### `-xchip=c`

最適化用のターゲットプロセッサを指定します。

- SPARC: f77/f95

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

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

`-xchip=c` は以下のものに影響を与えます。

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

表 3-15 に、`-xchip` の有効な値をまとめてあります。

表 3-15 `-xchip` の値

	値	最適化の対象
SPARC	<code>generic</code>	ほとんどの SPARC プロセッサで良好なパフォーマンス
	<code>old</code>	SuperSPARC' 以前のプロセッサ
	<code>super</code>	SuperSPARC チップ
	<code>super2</code>	SuperSPARC II チップ
	<code>micro</code>	MicroSPARC' チップ
	<code>micro2</code>	MicroSPARC II チップ
	<code>hyper</code>	HyperSPARC' チップ
	<code>hyper2</code>	HyperSPARC II チップ
	<code>powerup</code>	Weitek' PowerUP' チップ
	<code>ultra</code>	UltraSPARC' チップ
	<code>ultra2</code>	UltraSPARC II' チップ
	<code>ultra2i</code>	UltraSPARC III' チップ
	<code>ultra3</code>	UltraSPARC III™ チップ

### `-xcode=code`

SPARC プラットフォームのコードアドレス空間を指定します。

- **SPARC: f77/f95**

`code` の値は以下のとおりです。

---

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

---

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

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

`-xcode=abs64` SPARC および UltraSPARC V9 (`-arch=v9` または `v9a`)

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

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

共通ブロック不一致の実行時検査を行います。

- **SPARC: f77/f95**

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

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

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

## `-xcrossfile [=n]`

最適化とソースファイル間のインライン化を有効にします。

- **SPARC:f77/f95**

指定する場合、`n` には `0`、または `1` を指定できます。

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

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

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

ファイル間のインライン化を行うと、ソースファイルどうしの相互依存関係が生まれます。`-xcrossfile` を指定してコンパイルしたファイルセット中のいずれかのファイルを変更した場合は、新しいコードが正しくインライン化されるように、全ファイルを再コンパイルする必要があります。

インライン化については 66 ページの「`-inline=[%auto][,][no%]f1,...[no%]fn`」の説明を参照してください。

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

## `-xdepend`

`-depend` と同義です。

- **SPARC: f77/f95**

## **-xexplicitpar**

`-explicitpar` と同義です。

- **SPARC: f77/f95**

## **-xF**

Sun WorkShop アナライザにより、関数レベルの並べ替えを行います。

- **SPARC: f77/f95**

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

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

## **-xhelp=h**

オプションに関するヘルプ情報または `README` ファイルを表示します。

- **SPARC: f77/f95**

`h` には `readme` か `flags` のいずれかを指定します。

`-xhelp=readme` : 本リリースのコンパイラに関する `README` ファイルの内容を表示します。

`-xhelp=flags` : コンパイラフラグ (オプション) を表示します。

`-xhelp=flags` は `-help` と同義です。

## **-xia [=v]**

区間演算処理を有効化し、適切な浮動小数点環境を設定します。

- **SPARC: f95**

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

Fortran 95 で拡張された区間演算の詳細は、『Fortran 95 区間演算プログラミングリファレンス』に記載されています。107 ページの「`-xinterval[=v]`」を参照してください。

`-xia` フラグは、次のように展開されるマクロです。

---

<code>-xia</code> または	<code>-xinterval=widestneed -ftrap=%none -fns=no -fsimple=0</code>
<code>-xia=widestneed</code>	
<code>-xia=strict</code>	<code>-xinterval=strict -ftrap=%none -fns=no -fsimple=0</code>

---

## **-xildoff**

インクリメンタルリンカーを使用しません。

- **SPARC: f77/f95**

標準のリンカー `ld` を使用するように指定します。

このオプションは、`-g` オプションを使用しない場合のデフォルトです。また、`-G` を使用する場合や、コマンド行でソースファイル名を指定する場合のデフォルトでもあります。

このデフォルトは、`-xildon` オプションによって解除されます。

## **-xildon**

インクリメンタルリンカーを使用します。

- **SPARC: f77/f95**

インクリメンタルリンカーをオンにし、インクリメンタルモードで `ild` を使用するようにします。

このオプションは、`-G`でなく、`-g`を指定し、コマンド行にソースファイルを指定しない場合 (オブジェクトファイルやライブラリだけを指定する場合) のデフォルトです。

このデフォルトは、`-xildoff` オプションによって解除されます。

『C ユーザーズガイド』の `ild` のセクションを参照してください。

### `-xinline=list`

`-inline` と同義です。

- SPARC: f77/f95

### `-xinterval [=v]`

区間演算処理を有効化します。

- SPARC: f95

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

---

<code>no</code>	区間演算処理を有効にしません。
<code>widestneed</code>	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。
<code>strict</code>	型や長さが混在した間隔式の使用を禁止します。間隔型および長さの変換はすべて明示的に行わなければなりません。

---

Fortran 95 で拡張された区間演算の詳細は、『Fortran 95 区間演算プログラミングリファレンス』に記載されています。106 ページの「`-xia[=v]`」も参照してください。

### `-xl [d]`

使用できる VMS Fortran 機能を追加します。

- SPARC:f77/

`-xl` : コンパイラがより多くの VMS Fortran 機能を使用できるようになります。これは、`-vax=misalign` をマクロ化したもので、以下のような言語機能を提供します。92 ページの「`-vax=v`」の説明を参照してください。

特別なオプションを指定しなくても、ほとんどの VMS 機能は f77 で自動的に有効になりますが、VMS 機能によっては -x1 オプションを指定する必要があります。

通常は、ソース文が VMS、f77、または f95 のいずれにも解釈できるときに、VMS として解釈させたい場合に -x1 オプションを指定します。-x1 オプションを指定すると、コンパイラは VMS として解釈します。

以下に示す VMS の言語機能は、このオプションを指定して有効にします。

- 書式なし記録のバイトではなくワードでのサイズ (-x1)
- VMS 形式の論理ファイル名 (-x1)
- 引用符文字 (") で始まる 8 進定数 (-x1)
- 文字定数内の本来の文字としてのバックスラッシュ (\) 文字 (-x1)
- PARAMETER 文の非標準書式 (-x1)
- VMS としての構造体の境界整列 (-x1)
- 注釈行あるいは Fortran 文としてのデバッグ文 (-x1d)

VMS の構造体がどのように実装されているかを、プログラムが認識できる場合に、-x1 を使用して VMS の境界整列を行なってください。

-x1d を使用してデバッグ用の注釈 (D または d が 1 カラム目にある) をコンパイルさせます。-x1d オプションを省略すると、注釈として扱われます。  
(-x1 と d の間には空白文字を入れないでください。)

C ルーチンと構造体を共有するプログラムでは、-x1 は使用しません。

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

## **-xlibmil**

-libmil と同義です。

- SPARC: f77/f95

## **-xlibmopt**

最適化された数学ルーチンを使用します。

- SPARC: f77/f95



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

### **-xlic\_lib=sunperf**

サンのパフォーマンスライブラリとリンクします。

- **SPARC:f77/f95**

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

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

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

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

### **-xlicinfo**

ライセンスサーバーの情報を表示します。

- **SPARC: f77/f95**

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

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

### **-xlist [x]**

大域的プログラム検査 (GPC) を行ないます。

- **SPARC: f77/f95**

このオプションは、潜在的なプログラミングバグを検出するときに使用します。もう1つ別のコンパイラパスを起動して、プログラムを大域的に検査することにより、副プログラム呼び出しの引数、共通ブロック、およびパラメータに整合性があるかどうかを調べます。このオプションにより、クロスリファレンステーブルを含む、行番号の付いたソースコードが表示されます。-Xlist オプションによって生成されるエラーメッセージは警告であるため、プログラムのコンパイルやリンクは停止しません。

---

注 - 大域的プログラム検査を行う -Xlist オプションでコンパイルを実行する前に、必ずソースコードのすべての構文エラーを修正してください。構文エラーを修正しないままソースコードで GPC を実行すると、予期せぬレポートが出力される可能性があります。

---

例：整合性をルーチン間で検査します。

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

上記の例では、出力ファイル `fil.lst` に次の情報が書き込まれます。

- 行番号の付いたソースコードのリスト (デフォルト)
- ルーチン間の不整合に関するエラーメッセージ (リストの該当箇所に埋め込まれる)
- 識別子のクロスリファレンステーブル (デフォルト)

デフォルトでは、リストは `name.lst` というファイルに書き込まれます。name は、コマンド行で最初に指定したソースファイルの名前から拡張子を除いたものです。

以下に示すサブオプションによって、さまざまな動作を選択することができます。サブオプションは、-Xlist メインオプションの後に続けて指定します。

表 3-16 -Xlist サブオプション

オプション	機能
-Xlist	エラー、リスト、クロスリファレンステーブルを表示する。
-Xlistc	コールグラフとエラーを表示する (f77のみ)。
-XlistE	エラーを表示する。
-Xlisterr[nnn]	エラーメッセージ nnn を抑制する。

表 3-16 `-Xlist` サブオプション (続き)

---

<code>-Xlistf</code>	エラー、リスト、クロスリファレンスを表示する。ただしオブジェクトファイルは除く。
<code>-Xlistflndir</code>	すでに存在しているディレクトリ <code>dir</code> に <code>.fln</code> ファイルを置く ( <a href="#">f77</a> のみ)。
<code>-Xlisth</code>	エラーが検出された場合は、コンパイルを終了する ( <a href="#">f77</a> のみ)。
<code>-XlistI</code>	ソースファイルとともに <code>#include</code> と インクルードのファイルを分析する。
<code>-XlistL</code>	リストとエラーのみを表示表示する。
<code>-Xlistln</code>	ページ長を <code>n</code> 行に設定する。
<code>-Xlisto name</code>	レポートファイル名を <code>name.lst</code> に変更する。
<code>-Xlists</code>	クロスリファレンステーブルから参照されていない名前を抑制する ( <a href="#">f77</a> のみ)。
<code>-Xlistvn</code>	検査レベルを <code>n</code> (1、2、3、4) に設定表示する (デフォルトは 2) ( <a href="#">f77</a> のみ)。
<code>-Xlistw[nnn]</code>	出力行の幅を <code>nnn</code> 桁に設定する (デフォルトは 79) ( <a href="#">f77</a> のみ)。
<code>-Xlistwar[nnn]</code>	警告メッセージ <code>nnn</code> を抑制する。
<code>-XlistX</code>	クロスリファレンステーブルとエラーを表示する。

---

`-Xlistc`、`-Xlistf`、`-Xlistflndir`、`-Xlisth`、`-Xlists`、`-Xlistvn` の各オプションは、[f95](#) では使用できません。

詳細は、『Fortran プログラミングガイド』の第 5 章「プログラムの解析とデバッグ」を参照してください。

## `-xloopinfo`

`-loopinfo` と同義です。

- SPARC: [f77](#)/[f95](#)

## `-xmaxopt [=n]`

最適化プラグマを有効にして、最大最適化レベルを設定します。

- SPARC: [f77](#)/[f95](#)

$n$ には1～5の値を指定でき、それぞれ最適化レベル `-01`～`-05` に対応しています。指定しない場合、コンパイラは5を使用します。

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

このようなプリAGMA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` レベルを使用します。

## `-xmemalign [=<a><b>]`

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

### ● SPARC: f77/f95

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

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

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

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

境界整列値 `<a>` に指定できる値は、次のとおりです。

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

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

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

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

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

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

`-dalign` (47 ページ参照) および `-misalign` (71 ページ) のオプションは、それぞれ次のマクロです。

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

## **-xnolib**

`-nolib` と同義です。

- SPARC: `f77/f95`

## **-xnolibmil**

`-nolibmil` と同義です。

- SPARC: `f77/f95`

## **-xnolibmopt**

高速数学ライブラリを使用しません。

- SPARC: `f77/f95`

`-fast` と併用すると、最適化済みの数学ライブラリとリンクされません。

`f77 -fast -xnolibmopt ...`

## **-xOn**

`-On` と同義です。

- SPARC: f77/f95

## -xpad

-pad と同義です。

- SPARC:f77

## -xparallel

-parallel と同義です。

- SPARC: f77/f95

## -xpg

-pg と同義です。

- SPARC: f77/f95

## -xpp={fpp | cpp}

ソースファイルプリプロセッサを選択します。

- SPARC: f77/f95

デフォルトは `-xpp=fpp` です。

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

## -xprefetch [v]

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

- SPARC: f77/f95

`-xprefetch=yes` と指定すると、コンパイラは適切であると判断したときに先読み命令を挿入します。その結果、UltraSPARC II プロセッサ (`-xarch=v8plus`, `v9plusa`, `v9`, `v9a`) のパフォーマンスが改善されます。

v を指定する場合は、次に示すサブオプションのうち、一つを指定するか、またはそのいくつかをコンマで区切ったリストを指定します。

xprefetch=	意味
auto	先読み命令の自動生成を有効にします。
no%auto	先読み命令の自動生成を無効にします。
explicit	明示的な先読み命令を有効にします。
no%explicit	明示的な先読み命令を無効にします。
auto, explicit	自動先読みモードおよび明示的先読みモードを有効にします。
yes	<code>-xprefetch=auto</code> 、 <code>explicit</code> と同じ
no	<code>-xprefetch=no%auto</code> 、 <code>no%explicit</code> と同じ

指定しない場合、コンパイラは `-xprefetch=no` とみなします。`-xprefetch` だけを指定すると、デフォルトで `-xprefetch=yes` となります。

Fortran の `PREFETCH` 命令については、31 ページを参照してください。

## **-xprofile=p**

実行時のプロファイル用データを収集、またはそのデータを使用して最適化を行います。

### ● SPARC: f77/f95

p には `collect[:nm]`、`use[:nm]`、`tcov` のいずれかを指定します。最適化レベルは、`-O2` 以上を指定してください。

`collect[:nm]`

後に `-xprofile=use` オプションを使用してプログラムをコンパイルする時に、オプティマイザが使用する実行頻度データを収集し保存します。コンパイラは、文の実行頻度を測定するコードを生成します。

nm には、解析対象の実行可能ファイル名を指定します。この名前の指定は省略することができます。nm が指定されていない場合は、`a.out` が実行可能ファイル名とみなされます。

`-xprofile=collect`: `nm` でコンパイルしたプログラムは、実行時に `nm.profile` サブディレクトリを作成します。データは、このサブディレクトリにある `feedback` ファイルに書き込まれます。プログラムを複数回実行すると、実行頻度データは `feedback` ファイルに蓄積されていくので、以前の実行頻度データは失われません。

`use[:nm]`

実行頻度データを使用します。

`collect:nm` の場合と同じように `nm` には、実行可能ファイル名を指定します。名前の指定は省略することもできます。

`-xprofile=collect` オプションを付けてコンパイルした時に生成され、`feedback` ファイルに保存された実行頻度データを使用して、プログラムが最適化されます。

ソースファイル、その他のコンパイルオプションは、`feedback` ファイルが生成されるプログラムをコンパイルした時に使用したのと同じものを指定してください。

`-xprofile=collect:nm` でコンパイルするとき、`-xprofile=use:nm` で最適化コンパイルをするときのプログラム名は同一のものを指定する必要があります。

`tcov`

新しい形式の `tcov` を使用して、基本的なカバレッジ分析を行います。

コード計測方法は `-a` と似ていますが、各ソースファイルごとに `.d` ファイルは生成されません。最終的な実行可能ファイルに基づく名前のファイルが 1 つ生成されます。たとえば、実行可能ファイルが `stuff` である場合、`stuff.profile/tcovd` がデータファイルとなります。

`tcov` を実行する場合、`-x` オプションを付けて、新しい形式のデータを使用するよう指示する必要があります。`-x` オプションを指定しないと、`tcov` はデフォルトでは古い `.d` ファイルがあればそれをデータとして使用し、予測外の結果が生成されます。

`-a` とは異なり、`TCOVDIR` 環境変数はコンパイル時には影響しませんが、プログラムの実行時に使用され、プロファイルデータを置くサブディレクトリの位置が決まりません。

詳細は、`tcov(1)` マニュアルページ、『Fortran プログラミングガイド』の第 8 章「パフォーマンスプロファイリング」および『プログラムのパフォーマンス解析』を参照してください。



---

注 - `tcov` で作成されたレポートは、`-O4` または `-inline` によって副プログラムがインライン化されている場合、信頼性がない可能性があります。インライン化されているルーチンへの呼び出しの範囲は記録されません。

---

## **-xrecursive**

`RECURSIVE` 属性をもたないルーチンが自分自身を再帰的に呼び出せるようにします。

- **SPARC: f95**

`RECURSIVE` 属性で定義された副プログラムは、`-xrecursive` を使用してコンパイルされていない場合に限って、自分自身を再帰的に呼び出すことができます。

`-xrecursive` を使用してコンパイルすると、パフォーマンスが低下する可能性があります。`-xrecursive` は単独ではメモリスタック上に局所変数を割り当てないの  
で、`-xrecursive` と `-stackvar` の併用も検討してください。

## **-xreduction**

`-reduction` と同義です。

- **SPARC: f77/f95**

## **-xregs=r**

使用するレジスタを指定します。

- **SPARC: f77/f95**

`r` には、以下の 1 つまたは複数の項目をコンマで区切って指定します。

`[no%]appl, [no%]float`

`no` には `%` を付けてください。

例: `-xregs=appl, no%float`

デフォルトは `-xregs=appl, float` です。

- `appl` : アプリケーションレジスタを使用できるようにします。

SPARC システムでは、ある特定のレジスタをアプリケーションレジスタと呼びます。これらのレジスタを使用すると必要なロードおよびストア命令が少なくてすむため、パフォーマンスが向上します。ただし、アセンブリコードで記述された古いライブラリプログラムとの間で衝突が起きることがあります。

アプリケーションレジスタの組み合わせは、SPARC プラットフォームによって異なります。

- `-xarch=v8` または `v8a` の場合 - レジスタ `%g2`、`%g3` および `%g4`
  - `-xarch=v8` または `v8a` の場合 - レジスタ `%g2`、`%g3` および `%g4`
  - `-xarch=v8plus` または `v8plusa` の場合 - レジスタ `%g2`、`%g3` および `%g4`
  - `-xarch=v9` または `v9a` の場合 - レジスタ `%g2` および `%g3`
- `no%appl :appl` レジスタを使用しません。
  - `float :SPARC ABI` に指定されているように浮動小数点レジスタを使用できるようにします。  
これらのレジスタは、プログラムに浮動小数点のコードがない場合でも使用することができます。
  - `no%float` : 浮動小数点レジスタを使用しません。  
このオプションを使用すると、コンパイラは浮動小数点レジスタを浮動小数点演算の用途にのみ使用し、共通ブロックや構造体のコピーのためには使用しません。

デフォルトは `-xregs=appl, float` です。

## **-xs**

オブジェクトファイル (`.o`) がなくても `dbx` によってデバッグを実行できるようにします。

### ● **SPARC: f77/f95**

実行可能ファイルを別のディレクトリに移動した場合でも、オブジェクトファイル (`.o`) を無視してそのまま `dbx` を使用することができます。このオプションは、`.o` ファイルを残しておくことができない場合に使用します。

- コンパイラは、`-s` オプションをアセンブラに渡し、リンカーは `dbx` 用のシンボルテーブルをすべて実行可能ファイル内に配置します。
- このようにシンボルテーブルを処理する方法は旧式の方法です。この方法を非自動読み込みと呼ぶことがあります。
- リンカーによるリンクや `dbx` による初期化が遅くなります。

`-xs` を付けずに実行可能ファイルを移動する場合は、ソースファイルとオブジェクト (`.o`) ファイルの両方を移動するか、あるいは `dbx` の `pathmap` コマンドか `use` コマンドのいずれかでパスを設定する必要があります。

- これはシンボルテーブルを読み込む新しいデフォルトの方法です。この方法を自動読み込みと呼ぶことがあります。
- シンボルテーブルは、必要な場合にのみ `dbx` がシンボルテーブル情報を読み込むように、`.o` ファイル中に配置されます。したがって、リンク処理と、`dbx` による初期化が高速で実行されます。

### **-xsafe=mem**

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: f77/f95**

このオプションを使用する場合、コンパイラはメモリーに関するトラップが発生しないことを前提とします。V9 マシン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、`-05` と `-xarch=v8plus` がともに指定されている場合に限り、有効になります。

### **-xsb**

`-sb` と同義です。

- **SPARC: f77/f95**

### **-xsbfast**

`-sbfast` と同義です。

- **SPARC:f77/f95**

### **-xspace**

コードサイズが増大するような最適化は行いません。

- **SPARC: f77/f95**

コードのサイズが増大するような最適化は行いません。

たとえば、コードのサイズが増大する場合は、ループの展開や並列化は行いません。

## `-xtarget=t`

最適化の対象とするプラットフォームを指定します。

### ● SPARC: f77/f95

命令セットと最適化の対象とするプラットフォームを指定します。

`t`には `native`、`generic`、プラットフォーム名のいずれかを指定します。

`-xtarget` オプションは、実際のプラットフォームで発生する、`-xarch`、`-xchip`、`-xcache` をまとめて指定することができます。`-xtarget` の意味は = の後に指定した値を展開したものにあります。

対象となるハードウェア (コンピュータ) を正確にコンパイラに指定すると、パフォーマンスが向上するプログラムもあります。プログラムのパフォーマンスが重要な場合は、対象となるハードウェアを正確に指定してください。これは特に、新しい SPARC プロセッサ上でプログラムを実行する場合に当てはまります。ただし、ほとんどのプログラムおよびより旧式の SPARC プロセッサでは、パフォーマンス向上はごくわずかなので、`generic` を指定することで十分です。

`native` : ホストプラットフォームに対してパフォーマンスを最適化します。

コンパイラは、ホストプラットフォームに対して最適化されたコードを生成します。コンパイラが動作しているマシンで利用できるアーキテクチャ、チップ、キャッシュ特性が選択されます。

`generic` : 一般的なアーキテクチャ、チップ、キャッシュに対して最高のパフォーマンスが得られるようにします。

コンパイラは `-xtarget=generic` を次のように展開します。

```
-xarch=generic -xchip=generic -xcache=generic
```

これはデフォルトの値です。

プラットフォーム名 : 指定したプラットフォームに対して最高のパフォーマンスが得られるようにします。

コンパイラで使用できる現在の SPARC プラットフォームの名前

(`-xtarget=ultra2i` など) については、付録 D 「`-xtarget` プラットフォームの展開」を参照してください。

## `-xtime`

`-time` と同義です。

- SPARC: f77/f95

## `-xtypemap=spec`

デフォルトのデータサイズを指定します。

- SPARC:f77/f95

デフォルトのデータ型に対するバイトサイズを指定することができます。 `-dbl` と `-r8` よりもこのオプションを使用することをお勧めします。このオプションは、デフォルトサイズの変数および定数の両方に割り当てられます。

指定する文字列 `spec` には、次の全部またはいずれかをコンマで区切ったリストとして指定します。

```
real:size  
double:size  
integer:size
```

指定できるデータのサイズ `size` は、`real` に対しては 64、`double` に対しては 128、`integer` に対しては 32、64、`mixed` です。

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

このオプションは `REAL XYZ` (64 ビットの `XYZ` になる) のように明示的にバイトサイズを指定しないで宣言されたすべての変数に適用されます。また、単精度 `REAL` 定数はすべて、`REAL*8` に変換されます。

各プラットフォームで使用できる組み合わせは次のとおりです。

- `real:32`
- `real:64`
- `double:64`
- `double:128`
- `integer:32`
- `integer:64`
- `integer:mixed` (f77 のみ)

`integer:mixed` のマッピングは、8 バイトデータを指定しますが、4 バイト演算だけを行います。これは、f77 でだけ使用できます。`integer:64` の使用をお勧めしません。

f77 フラグの `-dbl` および `-r8` のオプションは、`-xtypemap` で次のように表すことができます。

- `-dbl` *same as:* `-xtypemap=real:64,double:128,integer:64`
- `-r8` *same as:* `-xtypemap=real:64,double:128,integer:mixed`

SPARC にはさらに 2 つの可能性があります。

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

これらは、デフォルトの `REAL` と `DOUBLE` を 8 バイトにします。また、`-dbl` や `-r8` では `DOUBLE PRECISION` から `QUAD PRECISION` にできないので、これらを優先して使用します。

`INTEGER` と `LOGICAL` は等価として扱われます。また、`COMPLEX` は 2 つの `REAL` として扱われます。`DOUBLE COMPLEX` は `DOUBLE` と同じように扱われます。

## `-xunroll=n`

`-unroll=n` と同義です。

- **SPARC: f77/f95**

## `-xvector [= {yes | no}]`

SPARC ベクトルライブラリ関数を自動呼び出しします。

- **SPARC: f77/f95**

`-xvector=yes` と指定すると、コンパイラは、必要に応じて、DO ループ内の数学ライブラリ呼び出しを特定して、同等のベクトル化されたライブラリルーチンの単一呼び出しに変換できます。その結果、ループカウンターの大きいループのパフォーマンスが改善されます。

デフォルトは `-xvector=no` です。`-xvector` だけを指定すると、デフォルトで `-xvector=yes` になります。

このオプションは `-depend` もトリガーします。(コマンド行で `-xvector` に続けて `-nodepend` を指定すると、依存解析を取り消すことができます。)

`-xvector` が指定されている場合は、`libmvec` と `libc` ライブラリをロードステップに含めるように、コンパイラはリンカーに自動通知します。ただし、コンパイルとリンクを別々に行う場合は、必要なライブラリを正確に選択するためにリンクステップで `-xvector` を指定する必要があります。

## `-xvpara`

`-vpara` と同義です。

- SPARC: f77

## `-zlp`

ループツールを使用するループパフォーマンスのプロファイル用にコンパイルします。

- SPARC: f77/f95

このオプションはループのプロファイラであるループツール用にオブジェクトファイルを準備します。次に `looptool(1)` コーティリティを実行すると、プログラムに関するループ統計情報が生成されます。

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

1つの副プログラムに `-zlp` を付けてコンパイルしたとしても、そのプログラムのすべての副プログラムに `-zlp` を付けてコンパイルする必要はありません。ただし、ループ情報が得られるのは `-zlp` を付けてコンパイルしたファイルからだけです。そのプログラムに含まれている他のファイルの情報は得られません。

詳細は、『プログラムのパフォーマンス解析』を参照してください。

## `-ztext`

再配置を伴わない純粋なライブラリを生成します。

- SPARC: f77/f95

再配置が残っている場合はライブラリを作成しません。

`-ztext` の主な目的は、生成されたライブラリが純粋なテキストであるかどうか、すべての命令が位置独立コードであるかどうかを確認することです。したがって、通常は `-G` および `-pic` と共に使用します。

`-ztext` を指定すると、テキストセグメントに不完全な再配置がある場合、`ld` はライブラリを構築しません。データセグメントに不完全な再配置がある場合は、`ld` はライブラリを構築しますが、そのデータセグメントは書き込み可能となります。

`-ztext` を指定しない場合、`ld` は再配置の状況とは無関係にライブラリを構築します。

このオプションは主に、オブジェクトファイルが `-pic` を付けて作成されたかどうか不明な場合に、ソースファイルとオブジェクトファイルの両方からライブラリを作成するときに使用します。

例：ソースファイルとオブジェクトファイルの両方からライブラリを作成します。

```
demo% f95 -G -pic -ztext -o MyLib -hMyLib a.f b.f x.o y.o
```

また、コードが位置独立コードであるかどうかを確認するためにも、このオプションを使用します。`-pic` を付けずにコンパイルすると、純粋なテキストであるかどうかを確認できます。

例：`-pic` を付けない場合は、純粋なテキストであるかどうかを確認します。

```
demo% f95 -G -ztext -o MyLib -hMyLib a.f b.f x.o y.o
```

`-ztext` を付けてコンパイルしても `ld` によってライブラリが構築されなかった場合は、`-ztext` を付けずにコンパイルし直すと `ld` によってライブラリが構築されます。`-ztext` を指定した場合に構築が失敗するということは、ライブラリ中に共有不可能な成分があることを示します。ただし、この場合でもその他の成分は共有できるはずですが、パフォーマンスが最高でない可能性もあります。



## 付録 A

---

### 実行時のエラーメッセージ

---

この付録では Fortran 入出力ライブラリ、シグナルハンドラ、およびオペレーティングシステムが生成するエラーメッセージについて説明します。

---

#### オペレーティングシステムのエラーメッセージ

オペレーティングシステムのエラーメッセージには、システムコールの失敗、C ライブラリのエラー、シェルの診断などがあります。システムコールのエラーメッセージは、[intro\(2\)](#)に記載されています。Fortran ライブラリを介して行われたシステムコールから、直接エラーメッセージが生成されることはありません。Fortran ライブラリの中にある次に示すシステムルーチンが、C のライブラリルーチンを呼び出し、それがエラーメッセージを生成します。

```
CALL SYSTEM("rm /")
END
```

このようにすると、次のメッセージが表示されます。

```
rm: / はディレクトリです
```

---

## シグナルハンドラのエラーメッセージ

プログラムの実行を開始する前に、Fortran ライブラリは、プログラムを終了させるシグナルとしてシグナルハンドラ (`sigdie`) を設定します。`sigdie` は、シグナルの内容を示すメッセージを表示し、待機中の出力をフラッシュし、コアイメージを生成して、トレースバックを行います。

現在のところ、エラーメッセージを生成する算術例外は、ゼロによる `INTEGER*2` の除算です。その他の算術例外は無視されます。

次の例のように、`SUB` サブルーチンに渡されていないパラメータを `SUB` サブルーチンがアクセスしようとすると、シグナルハンドラのエラーとなります。

```
CALL SUB()  
END  
SUBROUTINE SUB(I, J, K)  
  I=J+K  
  RETURN  
END
```

この場合、次のエラーメッセージが表示されます。

```
*** 停止 a.out  
*** シグナル 10 SIGBUS を受け取りました  
バスエラー
```

---

## 入出力のエラーメッセージ (f77)

Fortran の入出力ライブラリは、この後に列挙するエラーメッセージを生成します。`ERR` が発生すると、`IOSTAT` 変数にエラー番号が返されます。

たとえば、次のプログラムでは、書式付き出力用にオープンされたファイルに書式なしの書き込みを行おうとしています。

```
WRITE( 6 ) 1  
END
```

この場合、次のエラーメッセージが表示されます。

```
sue: [1003] unformatted io not allowed
logical unit 6, name 'stdout'
lately: writing sequential unformatted external IO
```

生成されるエラーメッセージは以下のとおりです。 [perror\(3F\)](#) のマニュアルページにも、これらのエラーメッセージが記載されています。

エラー番号が 1000 より小さい場合は、システムエラーです。 [intro\(2\)](#) を参照してください。

表 A-1 [f77](#) 実行時入出力メッセージ

---

エラー	メッセージ
-----	-------

---

- |      |   |
|------|---|
| 1000 | <a href="#">format</a> のエラー ( <a href="#">error in format</a> )<br>エラーメッセージを調べて、書式エラーの箇所を確認してください。このエラーは、10 レベルを超えて括弧を入れ子にした場合や、極端に長い書式文を使用した場合に発生する可能性があります。                       |
| 1001 | 不正な装置番号 ( <a href="#">illegal unit number</a> )<br>論理ユニット番号 0 を閉じるのは不正です。また、負の装置番号は使用できません。装置番号の上限は 231 - 1 です。   |
| 1002 | 書式付き入出力は許可されません ( <a href="#">formatted io not allowed</a> )<br>論理ユニットが、書式なし入出力用に開かれています。   |
| 1003 | 書式なし入出力は許可されません ( <a href="#">unformatted io not allowed</a> )<br>論理ユニットが、書式付き入出力用に開かれています。   |
| 1004 | 直接入出力は許可されません ( <a href="#">direct io not allowed</a> )<br>論理ユニットが、順番探査用に開かれています。または、論理記録長が 0 に指定されています。  |
| 1005 | 順次入出力は許可されません ( <a href="#">sequential io not allowed</a> )<br>論理ユニットが、直接探査入出力用に開かれています。  |
| 1006 | ファイルの <a href="#">backspace</a> ができません ( <a href="#">can't backspace file</a> )<br>論理ユニットに結合されたファイルをシークすることはできず、したがってバックスペースも行えません。そのファイルは <a href="#">tty</a> デバイスまたはパイプの可能性もあります。 |
| 1007 | レコードの先頭を越えています ( <a href="#">off beginning of record</a> )<br>内部入力記録の先頭より前に左タブを設定しようとしています。   |
-

表 A-1 f77 実行時入出力メッセージ (続き)

エラー	メッセージ
1008	ファイルの <code>stat</code> ができません ( <code>can't stat file</code> ) 当該のファイルに関して、システムがステータス情報を返すことができません。ディレクトリを読み込めない可能性があります。
1009	反復数の後に <code>*</code> がありません ( <code>no * after repeat count</code> ) 並び指定の入出力の繰り返し回数の中には、空白入れずに <code>*</code> が続いていなければなりません。
1010	レコードの終わりを越えています ( <code>off end of record</code> ) 書式付きの書き込み、または書式なしの読み込みか書き込みで、論理的な記録の終端を越えようとしていました。
1011	<使用せず>
1012	不完全な並び入力 ( <code>incomprehensible list input</code> ) 並び入力は、宣言で指定したとおりでなければなりません。
1013	空き領域の不足 ( <code>out of free space</code> ) ライブラリが内部使用のために動的に作成するバッファ用のメモリーが不足しています (プログラムが大きすぎます)。
1014	接続されていない装置 ( <code>unit not connected</code> ) 論理ユニットが開かれていません。
1015	予期しない文字の読みとり ( <code>read unexpected character</code> ) ある種の書式変換では、非数値データを処理できません。
1016	不正な論理入力フィールド ( <code>illegal logical input field</code> ) <code>logical</code> データは <code>T</code> か <code>F</code> でなければなりません。
1017	'new' ファイルが存在します ( <code>'new' file exists</code> ) 既存のファイルを <code>status='new'</code> で開こうとしています。
1018	'old' ファイルが見つかりません ( <code>can't find 'old' file</code> ) 存在しないファイルを <code>status='old'</code> で開こうとしています。
1019	認識できないシステムエラー ( <code>unknown system error</code> ) 発生するはずのない何らかの事態が発生しました。
1020	シーク可能な条件が必要です ( <code>requires seek ability</code> ) シークを許可していないファイルに対してシークしようとしています。直接探査、書式なしの順番探査入出力、左タブの設定を伴う入出力処理を行うときは、シーク可能な条件が必要です。

表 A-1 f77 実行時入出力メッセージ (続き)

エラー	メッセージ
1021	不正な引数 ( <i>illegal argument</i> ) open および関連する関数に対する引数の一部は、正当性を検査されます。デフォルトではない形式のものがしばしば検査対象となります。
1022	負数の反復数 ( <i>negative repeat count</i> ) 並びによる入力の繰り返し回数は、正の整数でなければなりません。
1023	装置に対する不正な操作 ( <i>illegal operation for unit</i> ) 論理ユニットに結合されているデバイスに対しては実行できない入出力処理を行おうとしています。テープの終端やファイルの終端を越えて読み込もうとした場合には、このエラーになります。
1024	<使用せず>
1025	オープン時の指定子が矛盾している ( <i>incompatible specifiers in open</i> ) 'new' オプションと <i>access='append'</i> を付けるなど、不正なオプションの組み合わせを指定してファイルを開こうとしています。
1026	<i>namelist</i> への不正な入力 ( <i>illegal input for namelis</i> ) 名前並びの読み込みで、不正なデータ項目があります。
1027	<i>FILEOPT</i> パラメータのエラー ( <i>error in FILEOPT parameter</i> ) <i>OPEN</i> の使用で、 <i>FILEOPT</i> の文字列の構文に誤りがあります。
1028	読み取り専用ファイルに対する <i>WRITE</i> ( <i>WRITE to readonly file</i> ) 論理ユニットが読み取り専用として開かれています。
1029	書き込み専用ファイルに対する <i>READ</i> ( <i>READ from writeonly file</i> ) 論理ユニットが書き込み専用として開かれています。
1030	数値入力変換中のオーバーフロー ( <i>overflow converting numeric input</i> ) 対応する入力変数に対し、整数入力データが大きすぎます。
1032	数値入力での指数オーバーフロー ( <i>exponent overflow on numeric input</i> ) 浮動小数点入力データが大きすぎて、対応する入力変数で表せません。

## 入出力のエラーメッセージ (f95)

次の表に f95 で出力される実行時入出力メッセージの一部を示します。

表 A-2 f95 の実行時入出力メッセージ

エラー	メッセージ
1000	書式エラー (error in format)
1001	不正な装置番号 (illegal unit number)
1002	書式なし装置に対する書式付き入出力 (formatted I/O on unformatted unit)
1003	書式付き装置に対する書式なし入出力 (unformatted I/O on formatted unit)
1004	順番探査装置に対する直接探査入出力 (direct-access I/O on sequential-access unit)
1005	直接探査装置に対する順番探査入出力 (sequential-access I/O on direct-access unit)
1006	装置は BACKSPACE をサポートしません (device does not support BACKSPACE)
1007	レコードの先頭を越えています (off beginning of record)
1008	ファイルの stat できません (can't stat file)
1009	反復数の後に * がありません (no * after repeat count)
1010	長過ぎる記録
1011	切り捨てエラー
1012	不完全な並び入力 (incomprehensible list input)
1013	空き領域の不足 (out of free space)
1014	接続されていない装置 (unit not connected)
1015	予期しない文字の読みとり (read unexpected character)
1016	不正な論理入力コード (illegal logical input field)
1017	'new' ファイルが存在します ('new' file exists)

表 A-2 f95 の実行時入出力メッセージ (続き)

エラー	メッセージ
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	チャンネルやデバイスに対する不正な操作 (illegal operation for channel or device)
1024	再入可能入出力
1025	オープン時の指定子が矛盾している (incompatible specifiers in open)
1026	namelist への不正な入力 (illegal input for namelist)
1027	FILEOPT パラメータのエラー (error in FILEOPT parameter)
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)

表 A-2 f95 の実行時入出力メッセージ (続き)

エラー	メッセージ
1056	内部装置の無効な引用 (null reference to internal unit)
1057	空の内部ファイル (empty internal file)
1058	書式なし装置に対する並び入出力 (list-directed I/O on unformatted unit)
1059	書式なし装置に対する変数群入出力 (namelist I/O on unformatted unit)
1060	内部ファイルの終端を越えて書き出ししようとした (tried to write past end of internal file)
1061	結合していない ADVANCE 指定子 (unassociated ADVANCE specifier)
1062	ADVANCE 指定子が 'YES' または 'NO' ではありません (ADVANCE specifier is not 'YES' or 'NO')
1063	EOR 指定子が前進入力に対して指定されています (EOR specifier present for advancing input)
1064	SIZE 指定子が前進入力に対して指定されています (SIZE specifier present for advancing input)
1065	負数またはゼロの記録番号 (negative or zero record number)
1066	ファイルに存在しない記録 (record not in file)
1067	破壊された書式 (corrupted format)
1068	結合していない入力変数 (unassociated input variable)
1069	データ編集記述子より多い入出力項目 (more I/O-list items than data edit descriptors)
1070	添字三つ組にゼロの刻み幅 (zero stride in subscript triplet)
1071	DO 形ループにゼロの増分値 (zero step in implied DO-loop)
1072	負数の欄幅 (negative field width)
1073	ゼロ幅の欄 (zero-width field)
1074	文字列編集記述子が入力に用いられています (character string edit descriptor reached on input)



表 A-2 f95 の実行時入出力メッセージ (続き)

---

エラー	メッセージ
1075	ホレリス編集記述子が入力に用いられています (Hollerith edit descriptor reached on input)
1076	数字列に数字がありません (no digits found in digit string)
1077	指数に数字がありません (no digits found in exponent)
1078	範囲外の桁移動数 (scale factor out of range)
1079	数字が基数と等しいかまたは、基数を越えています (digit equals or exceeds radix)
1080	整数欄に予期しない文字 (unexpected character in integer field)
1081	実数欄に予期しない文字 (unexpected character in real field)
1082	論理欄に予期しない文字 (unexpected character in logical field)
1083	整数値に予期しない文字 (unexpected character in integer value)
1084	実数値に予期しない文字 (unexpected character in real value)
1085	複素数値に予期しない文字 (unexpected character in complex value)
1086	論理値に予期しない文字 (unexpected character in logical value)
1087	文字値に予期しない文字 (unexpected character in character value)
1088	変数群名の前に予期しない文字 (unexpected character before NAMELIST group name)
1089	変数群名がプログラム中の名前と一致しません (NAMELIST group name does not match the name in the program)
1090	変数群の項目に予期しない文字 (unexpected character in NAMELIST item)

---

表 A-2 f95 の実行時入出力メッセージ (続き)

---

エラー	メッセージ
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)

---

表 A-2 f95 の実行時入出力メッセージ (続き)

エラー	メッセージ
1117	結合していない FILE 指定子 (unassociated FILE specifier)
1118	結合していない FMT 指定子 (unassociated FMT specifier)
1119	結合していない FORM 指定子 (unassociated FORM specifier)
1120	結合していない FORMATTED 指定子 (unassociated FORMATTED specifier)
1121	結合していない NAME 指定子 (unassociated NAME specifier)
1122	結合していない PAD 指定子 (unassociated PAD specifier)
1123	結合していない POSITION 指定子 (unassociated POSITION specifier)
1124	結合していない READ 指定子 (unassociated READ specifier)
1125	結合していない READWRITE 指定子 (unassociated READWRITE specifier)
1126	結合していない SEQUENTIAL 指定子 (unassociated SEQUENTIAL specifier)
1127	結合していない STATUS 指定子 (unassociated STATUS specifier)
1128	結合していない UNFORMATTED 指定子 (unassociated UNFORMATTED specifier)
1129	結合していない WRITE 指定子 (unassociated WRITE specifier)
1130	長さゼロのファイル名 (zero length file name)
1131	ACCESS 指定子が 'SEQUENTIAL' または 'DIRECT' ではありません (ACCESS specifier is not 'SEQUENTIAL' or 'DIRECT')
1132	ACTION 指定子が 'READ', 'WRITE' または 'READWRITE' ではありません (ACTION specifier is not 'READ', 'WRITE' or 'READWRITE')

表 A-2 f95 の実行時入出力メッセージ (続き)

エラー	メッセージ
1133	BLANK 指定子が 'ZERO' または 'NULL' ではありません (BLANK specifier is not 'ZERO' or 'NULL')
1134	DELIM 指定子が 'APOSTROPHE', 'QUOTE', または 'NONE' で はありません (DELIM specifier is not 'APOSTROPHE', 'QUOTE', or 'NONE')
1135	予期しない FORM 指定子 (unexpected FORM specifie)
1136	PAD 指定子が 'YES' または 'NO' ではありません (PAD specifier is not 'YES' or 'NO')
1137	POSITION 指定子が 'APPEND', 'ASIS', または 'REWIND' ではありません (POSITION specifier is not 'APPEND', 'ASIS', or 'REWIND')
1138	RECL 指定子が zero または negative です (RECL specifier is zero or negative)
1139	直接探査ファイルに対して記録長が指定されていません (no record length specified for direct-access file)
1140	予期しない STATUS 指定子 (unexpected STATUS specifier)
1141	接続されている装置に対して 'OLD' でない status が指定され ています (status is specified and not 'OLD' for connected unit)
1142	STATUS 指定子が 'KEEP' または 'DELETE' ではありません (STATUS specifier is not 'KEEP' or 'DELETE')
1143	一時ファイルに対して指定した status 'KEEP' ( status 'KEEP' specified for a scratch file)
1144	不当な status の値 (impossible status value)
1145	一時ファイルに対してファイル名が指定されました (a file name has been specified for a scratch file)
1146	読み取り中または書き出し中の装置を開こうとしています (attempting to open a unit that is being read from or written to)

表 A-2 f95 の実行時入出力メッセージ (続き)

---

エラー	メッセージ
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)
1161	装置は REWIND をサポートしていません (device does not support REWIND)
1162	BACKSPACE には読みとり権が必要です (read permission required for BACKSPACE)
1163	直接探査装置に対する BACKSPACE (BACKSPACE on direct-access unit)
1164	バイナリ装置に対する BACKSPACE (BACKSPACE on binary unit)
1165	backspace 中にファイルの終わりになりました (end-of-file seen while backspacing)
1166	ENDFILE には書き込み権が必要です (write permission required for ENDFILE)
1167	直接探査装置に対する ENDFILE (ENDFILE on direct-access unit)
1181	割り付けされている配列を割り付けしようとしています (attempting to allocate an allocated array)
1182	結合していないポインタの割り付け解放 (deallocating an unassociated pointer)
1183	結合していない割り付け済み配列の割り付け解放 (deallocating an unallocated allocatable array)

---

表 A-2 f95 の実行時入出力メッセージ (続き)

エラー	メッセージ
1184	ポインタにより割り付け配列の割り付け解放 (deallocating an allocatable array through a pointer)
1185	ALLOCATE 文により割り付けされていない実体の割り付け解放 (deallocating an object not allocated by an ALLOCATE statement)
1186	実体の一部の割り付け解放 (deallocating a part of an object)
1187	割り付けより大きな実体の割り付け解放 (deallocating a larger object than was allocated)
1191	配列組み込み関数にわたされた割り付けされていない配列 (unallocated array passed to array intrinsic function)
1192	不正な次元数 (illegal rank)
1193	小さなソースサイズ (small source size)
1194	ゼロの配列サイズ (zero array size)
1195	形状に負の要素 (negative elements in shape)
1196	不正な種別 (illegal kind)
1197	形状不適合の配列 (nonconformable array)
2001	無効な定数、構造体、または名前 (invalid constant, structure, or component name)
2002	生成されていないハンドル (handle not created)
2003	短過ぎる文字引数 (character argument too short)
2004	長過ぎる、または短過ぎる配列引数 (array argument too long or too short)
2005	ファイル、記録、またはディレクトリストリームの終わり (end of file, record, or directory stream)

# 各リリースにおける機能変更

---

この付録では、f77 と f90 の今回のリリースで追加された機能と、旧リリースから変更された機能について説明します。

---

## Fortran 95 の新機能と変更

この節では、f95 の Sun WorkShop 6 とそれ以前のリリースに固有の新機能および動作上の変更点についてまとめてあります。

### Sun WorkShop 6 における f95 の新機能

以下に、Sun Performance WorkShop 6 でリリースされた Fortran 95 コンパイラにおける新機能と変更された機能を示します。

- 準拠：f95 は、Fortran 95 規格に完全に準拠しています。
- 新コマンド：Fortran 95 のコンパイラは、f95 または f90 のいずれかのコマンドで起動できます。
- 最適化コードのデバック：-g を他のオプションと併用することに関する制約が緩和され、並列化コードおよび -O4 または -O5 を指定した最適化コードを、dbx および Sun WorkShop デバッカを使用してデバックできるようになりました。
- ソースファイル名の拡張子：コンパイラは、.f95 および .f90 というファイル拡張子に加え、.F95 および .F90 という拡張子も受け付けるようになりました。
- 区間演算：今回のリリースでは、区間演算計算を可能にするいくつかの拡張機能を実装しました。詳細は、『Fortran 95 区間演算プログラミングリファレンス』および区間演算の REAME ファイルを参照してください。

- 配列の最適化の向上：コンパイラは、`-O4` および `-O5` のレベルで配列に関して積極的な最適化を行うようになりました。
- ハイパーリンク診断メッセージ：Sun WorkShop オンラインヘルプは、「構築」ウィンドウに表示される `f95` エラー診断の解釈についてオンラインヘルプ情報を提供するようになり、エラーメッセージから書斎なオンラインヘルプへのハイパーテキストリンクを作成します。
- OpenMP：コンパイラは、OpenMP の明示的な並列化指令を受け入れます。OpenMP 仕様は、<http://www.openmp.org/> で表示できます。
- Cray 形式の `DOALL` 並列化指令に `AUTOSCOPE` が追加されました。
- 新コマンド行オプションおよび変更されたコマンド行オプション
  - `-aligncommon` は、COMMONブロック内の要素を指定のバイト境界に整列させます。
  - `-r8const` は、単精度データ定数を `REAL*8` に変換します。
  - `-xinterval` および `-xia` を使用すると、区間演算展開が有効になります。
  - `-xmemalign` は、データ要素のメモリ内での一般的な境界整列を指定します。
  - `-om=openmp` および `-openmp` を使用すると、OpenMP の明示的な並列化指令の固有のコンパイルが可能になります。
  - UltraSPARC 先読み命令を有効にするために `-xprefetch` の機能拡張が行われ、サブオプションが追加されました。
  - `-xrecursive` を使用すると、`RECURSIVE` 属性が指定されていない副プログラムからの再帰的呼び出しが可能になります。
  - `-xtypemap` では、可能なデータ型指定のセットが拡張されました。
  - `-fast` が機能拡張され、`-O5`、`-fsimple=2`、`-xvector=yes` および `-pad=common` を設定できます。
- `f95` の並列化機能を使用するには、Sun WorkShop HPC ライセンスが必要です。

## f90 2.0 の新機能

Sun WorkShop 5.0 でリリースされた `f90 2.0` のコンパイラには、`f90 1.2` リリースに対する次の新機能と変更された機能が含まれています。

- 新しいオプション：
  - ほとんどの `f77` オプションが `f90` で認識されます。
  - `-fprover` が入出力処理で浮動小数点のオーバーフローを検出します。
  - `-xcode=code` で SPARC プラットフォームのメモリーアドレスモデルを指定します。



- `-xcommonchk` で COMMON ブロック宣言に不一致がないか実行時検査を行います。
  - `-xprefetch` を使用すると、コンパイラは UltraSPARC II プラットフォームで先読み命令を生成できます。
  - `-xvector` を使用すると、コンパイラは DO ループ内の特定の数学ライブラリ呼び出しをベクトル化された数学ルーチンの単一呼び出しで置き換えることができます。
- 変更されたオプション：
- `-xcrossfile [=n]` オプションのレベル番号が追加されました。
  - `-fns [= {yes|no}]` オプションの yes/no が追加されました。
  - `-Ztha` 現在ではこのオプションは無視されます。
- 新しい機能：
- `-xarch=v9` または `v9a` を使用して 64 ビットの SPARC プラットフォームで 64 ビットの Solaris 7 環境用にコンパイルします。
  - 入出力ライブラリで大型ファイル (2GB 以上) をサポートします。
  - 64 ビットの Solaris オペレーティング環境で大きな配列をサポートします。
  - サン形式の指令をデフォルトとして使用できます。
  - `REDUCTION` 指令に変数リストの配列を使用できます。
  - SPARC の場合：`TASK COMMON` 指令が `COMMON` の変数を非公開として宣言します。
  - 新しい最適化プラグマでコンパイラの最適化レベルをルーチンごとに設定できます。
- 入出力の相違点 (f90 1.2 と比較した場合)
- `NAMELIST` 出力書式：
    - 1.2 の場合：
 

1つの PRINT 文に含まれる変数はすべて改行なしで 1 行に書き込まれます。
    - 2.0 の場合：
 

各変数が別々の行に出力されます。
    - 1.2 の場合：
 

値を区切るのにコンマを使用します。
    - 2.0 の場合：
 

値を区切るのに空白文字を 1 つ使用します。

1.2 の場合 :

繰り返される値は `r*` 書式 (`3*8.22`) を使用して出力されます。

2.0 の場合 :

繰り返される値はすべて明示的に (`8.22 8.22 8.22`) 出力されます。

1.2 の場合 :

整数の浮動小数点の末尾がゼロなしで出力されます (`1.`)。

2.0 の場合 :

浮動小数点の整数に末尾のゼロを付けて出力されます (`1.0`)。

1.2 の場合 :

出力される値が、同じ型の変数として読み取られた値と異なることがあります。読み取り時に `0.1` であった値が `0.100000001` として出力されます。

2.0 の場合 :

書き込まれた値と読み取られた値が同じになるように、必要最小限の桁数で出力されます。`0.1` は `0.1` として出力されます。

1.2 の場合 :

標準の要件に従うと、ゼロの値は指数形式で出力されます。ただし、1.2 では `0.E+0` と出力されます。

2.0 の場合 :

ゼロを `0.0E+0` として出力します。

1.2 の場合 :

複素数値のコンマと虚数部との間に空白を出力します (`1., 0.E+0`)。

2.0 の場合 :

コンマは使用しません (`1.0,0.0E+0`)。

■ **NAMELIST** 入力書式 :

2.0 では、グループ名の前に `$` や `&` を付けて入力できます。`&` は Fortran 90 の規格で認められている唯一の書式で、NAMELIST 出力で書き込まれます。

2.0 では、グループの最終データ項目が CHARACTER の場合を除いて、`$` を入力の終わりを示すシンボルとして使用できます。これは入力として処理されます。

2.0 では、NAMELIST 入力をレコードの最初の桁から開始することができます。

■ **PRINT \*** コンマで区切った出力は廃止しました。

- `OPEN FORM='BINARY'` は、レコードマークなしで規格外の raw テキストの入出力を行うことができます。`FORM='BINARY'` でファイルを開いた場合、ファイルにレコード長が埋め込まれないことを除いて、`FORM='UNFORMATTED'` を使用した場合とほぼ同じ効果が得られます。このデータなしでは、レコードの開始場所と終了場所を特定することはできません。したがって、どの位置まで後退させるのか分からないので、`FORM='BINARY'` ファイルを `BACKSPACE` 文で移動することはできません。`'BINARY'` ファイルに対して読み取り (`READ`) を実行すると、入力並び上にある変数を満たすのに必要なだけのデータが読み取られます。詳細は、付録 C または『FORTRAN 77 言語リファレンス』を参照してください。
- 異なる装置に対する再帰的入出力が可能です (これは、`f90` I/O ライブラリが「MT-Warm」だからです)。
- 順次書式付き出力、並びによる出力、および `NAMelist` 出力でのデフォルトのレコード長は `RECL=2147483646` ( $2^{31}-2$  上文字) です (以前のデフォルトは 267)。
- `ENCODE` と `DECODE` が、認識、実装されています。方法については、『FORTRAN 77 言語リファレンス』に記載されています。
- 一時ファイルの命名規則は `f77` の場合と同じです。
- 以下に示すように、`ADVANCE='NO'` で非前進入出力を有効にできます。
 

```
write(*,'(a)',ADVANCE='NO') 'n= '
read(*,*) n
```
- 内部ファイルの入出力の処理は、`f90` 1.2 よりも Fortran 90 標準に近くなりました。また、内部入出力を行うルーチンを入出力リストで呼び出すことができます。これは、1.2 (または `f77`) では許可されていませんでした。
- 操作上の相違点：
  - モジュールの扱い方が異なります。1 つ以上の `MODULE` 単位を含むソースコードをコンパイルすると、情報ファイル (名前 `.mod`) がモジュールごとに生成されます。この情報ファイルの名前はモジュールの名前で、`.mod` 拡張子を付けて小文字で生成されます。`.mod` ファイルは、モジュールが `USE` 文で指定される前に使用できなければなりません。つまり、`MODULE` を `USE` 文で参照するファイルをコンパイルする前に、すべての `MODULE` ファイルがコンパイルされている (およびモジュール情報ファイルも作成されている) 必要があります。
  - `-fttrap=common` はデフォルトのトラップモードです。
  - Sun Performance Library のルーチンが配列処理を行うように自動リンクされます。

■ 新しい言語要素：

- Fortran 95 の言語要素がいくつか実装されています。

属性 PURE と ELEMENTAL

MAXVAL と MINVAL の拡張形式

■ 新しいデータ型が認識されます。

COMPLEX\*32 REAL\*16

INTEGER\*8 (および \*1、\*2)

LOGICAL\*8 (および \*1、\*2)

■ いくつかのデータ表現が f90 1.2 以降に変更されています。

INTEGER\*2 は、現在では、4 バイトではなく 2 バイトです。

INTEGER\*1 は、現在では、4 バイトではなく 1 バイトです。

LOGICAL\*2 は、現在では、4 バイトではなく 2 バイトです。

LOGICAL\*1 は、現在では、4 バイトではなく 1 バイトです。

これにより、1.2 コンパイラでコンパイルした f90 プログラムで書き込まれたこれらのデータ項目を含むバイナリデータファイルを読み取るプログラムに影響します。これに対処するには、2.0 でコンパイルするときに \*1 や \*2 ではなく INTEGER\*4 または LOGICAL\*4 になるように宣言を変更します。

- 値による呼び出し %VAL が f77 と同様に実装されています。ただし、f90 2.0 では、REAL\*8 と REAL\*16 を double および long double として C ルーチンに渡すことができます。

■ f77 および C の f90 2.0 との互換性：

- f77 と f90 のオブジェクトバイナリを混在させるには、libF77 ではなく f77 互換ライブラリ libf77compat とリンクします。たとえば、メインプログラムが f77 プログラムの場合でも、f90 ..files.. -lf77compat としてリンクを行います。
- f90 COMMON の構造は、現在では、f77 と互換性があります。
- f90 スカラーポインタは、C ポインタと互換性があります。

## Fortran 77 の新機能と変更点

ここでは、今回の Sun WorkShop 6 のリリースと旧リリースで f77 固有新たに追加された機能と変更点について説明します。

### Sun WorkShop 6 での f77 の新機能

次に、Sun WorkShop 6 の Fortran 77 には、次の新機能および変更された機能が含まれています。

- 入出力拡張機能：OPEN (FORM= 'BINARY') を指定したファイルのオープンは、レコードマークがついていないシーケンシャルバイナリ (未書式化) ファイルとして扱われます。詳細は、『FORTRAN 77 言語リファレンス』を参照してください。
- 最適化コードのデバック：-g を他のオプションと併用することに関する制約が緩和され、並列化コードおよび -O4 または -O5 を指定した最適化コードを、dbx および Sun WorkShop デバッカーを使用してデバックできるようになりました。
- 新コマンド行オプションと変更されたコマンド行オプション
  - -aligncommon は、COMMONブロック内の要素を指定のバイト境界に整列させます。
  - -r8const は、単精度データ定数を REAL\*8 に変換します。
  - -xmemalign は、データ要素のメモリ内で一般的な境界整列を指定します。
  - UltraSPARC 先読み命令を有効にするために、-xprefetch の機能拡張が行われ、サブオプションが追加されました。
  - -xtypemap では、可能なデータ型指定のセットが拡張されました。
  - -fast が機能拡張され、-O5、-fsimple=2、-xvector=yes および -pad=common を設定できます。
- f77 の並列化機能を使用するには、Sun WorkShop HPC ライセンスが必要です。
- ハイパーリンク診断メッセージ：Sun WorkShop オンラインヘルプは、「構築」ウィンドウに表示される f77 エラー診断の解釈についてオンラインヘルプ情報を提供できるようになり、エラーメッセージから詳細なオンラインヘルプへのハイパーテキストリンクを作成します。

### f77 5.0 の新機能

f77 5.0 の新機能と変更点は次のとおりです。

- 新しいオプション：
  - `-fpover` が入出力処理で浮動小数点のオーバーフローを検出します。
  - `-xcode=code` で SPARC プラットフォームのメモリアドレスモデルを指定します。
  - `-xcommonchk` で COMMON ブロック宣言に不一致がないか実行時検査を行います。
  - `-xmaxopt` は `OPT=n` プラグマを有効にし、ソースコードで `OPT` プラグマが使用できる最大最適化レベルを制御します。
  - `-xprefetch` を使用すると、コンパイラは UltraSPARC II プラットフォームで先取り命令を生成できます。
  - `-xvector` を使用すると、コンパイラは DO ループ内の特定の数学ライブラリ呼び出しをベクトル化された数学ルーチンの単一呼び出しで置き換えることができます。
- 変更されたオプション：
  - `-xcrossfile [=n]` オプションのレベル番号が追加されました。
  - `-fns[={yes|no}]` オプションの `yes/no` が追加されました。
  - `-Ztha` 現在ではこのオプションは無視されます。
- 新しい機能：
  - `-xarch=v9` または `v9a` を使用して 64 ビットの SPARC プラットフォームで 64 ビットの Solaris 7 環境用にコンパイルします。
  - 入出力ライブラリで大型ファイル (2GB 以上) をサポートします。
  - 64 ビットの Solaris 7 環境で大規模配列をサポートします。
  - 動的配列 (動的サイズのローカル配列) が実装されています (『FORTRAN 77 言語リファレンス』を参照)。
  - `REDUCTION` 指令に変数リストの配列を使用できます。
  - SPARC の場合：`TASK COMMON` 指令が `COMMON` の変数を非公開として宣言します。
  - Fortran 90 スタイルの定数でバイトサイズ (64 ビット、8 バイト、定数に対して `12345678_8` など) を指定できます。
  - 新しい最適化プラグマでコンパイラの最適化レベルをルーチンごとに設定できます。
  - 2000 年以降も正しく処理される `date_AND_time( )` ライブラリルーチン

## f77 4.2 の新機能

f77 4.0 リリース以降に追加、または変更された f77 4.2 の機能は、次のとおりです。

- 新しいオプション：
  - `-dbl_align_all`
  - `-errtags=yes|no` および `-errofs=taglist`
  - `-stop_status=no|yes`
  - `-xcrossfile`
  - `-xlic_lib=libs`
  - `-xpp=fpp|cpp`
  - `-xtypemap=type:spec, .`
- 変更されたオプション：
  - `-fround`、`-fsimple`、`-ftrap`、`-xprofile=tcov`、`-xspace`、`-xunroll` の各オプションが Intel の各プラットフォームで使用できます。
  - `-xtarget`、`-xarch`、`-xchip` の各オプションは、SPARC Ultra、Intel の各プラットフォームでも使用できます。
  - `-vax=` オプションによって、各 VAX/VMS Fortran 機能を有効または無効にすることができます。
  - デフォルトのソースファイルプリプロセッサが、`cpp(1)` から `fpp(1)` に変更されています。

---

## FORTRAN 77 の上位互換

FORTRAN 77 5.0 のソースは、旧リリースと互換性があります。ただし、オペレーティングシステムの変更やバグ修正のために、多少の変更が加えられています。

### Fortran 3.0/3.0.1 と 4.0

Solaris 2 上で Fortran 3.0/3.0.1 を使用してコンパイル/リンクした実行可能ファイル (`a.out`)、ライブラリ (`.a`)、およびオブジェクトファイル (`.o`) は、Solaris 2 上で実行する Fortran 5.0 と互換性があります。

### Solaris 1 アプリケーションの実行

Solaris 1 上で生成した実行可能ファイルを Solaris 2 上で実行するには、バイナリ互換パッケージ (BCP) をインストールする必要があります。

Solaris 1 でコンパイル/リンクした実行可能ファイルは、Solaris 2 でも実行できません。ただし、Solaris 2.x でコンパイル/リンクした場合に比べ、パフォーマンスは低下します。

Solaris 1 で Fortran 2.0.1 を使用してコンパイル/リンクしたライブラリ (.a) とオブジェクトファイル (.o) は、Fortran 5.0 とは互換性が保証されません。

## 技術用語について

本書では、Fortran 特有の用語の和訳については、日本標準規格 Fortran (JIS X 3001-1994) で使用されている用語に準じるよう努めました。ただし、以下に挙げるものについては、他言語との関連性を考慮し、また、WorkShop 製品全体との整合性を優先し、日本標準規格とは異なる訳語を用いています。

表 B-1 JIS X 3001-1994 の訳と本書の訳の違い

英語	JIS X 3001-1994 での和訳	本書での和訳
allocate	割付け	割り付け
ampersand	アンド記号	アンパサンド
assign	割当て	割り当て
assumed	引継ぎ	引き継ぎ
assumed-seize dummy array	擬寸法仮配列	大きさ引き継ぎ仮配列
call	呼出し	呼び出し
error condition	誤り条件	エラー条件
inquiry	問合せ	問い合わせ
intrinsic	組込み	組み込み
ising model	イジング模型	イジングモデル
optionally signed integer constant	符号付き又は符号なしの 整定数	符号付き、または符号なしの 整定数
parameter	パラメタ	パラメータ
procedure	手続き	手続き
reading	読込み	読み取り
scalar	スカラ	スカラー



表 B-1 JIS X 3001-1994 の訳と本書の訳の違い (続き)

英語	JIS X 3001-1994 での和訳	本書での和訳
scale factor	けた移動数	桁移動数
scope	有効範囲	スコープ
scoping unit	有効域	スコープ単位

なお、JIS X 3001-1994 で用いられる用語の調査に際しては、駒沢大学の西村和夫先生、ならびに、江戸川大学の高田正之先生にご協力いただきましたことを、感謝いたします。



## 付録 C

---

### Fortran 95 の機能と相違点

---

この付録では、以下の Fortran の機能の主な相違点について説明します。

- 標準の Fortran 95 と Sun Fortran 95
- FORTRAN 77 と Fortran 95

---

#### 機能

Sun WorkShop 6 Fortran 95 は、以下に示す機能を備えています。

#### 継続行の制限

f95 と f77 では 99 行まで行を継続することができます (開始行が 1 行とその後の継続行が 98 行ということです)。標準の Fortran 95 では、固定形式の場合で 19 行まで、自由形式の場合で 39 行までです。

#### 固定形式固定形式のソースの行

固定形式のソースの場合、1 行に 73 文字以上使用できます。ただし、73 カラム目以降はすべて無視されます。標準の Fortran 95 では、行の長さは 72 文字までです。

#### 指令

f95 では、指令行が `CDIR$`、`!DIR$`、`CMIS$`、`C$PRAGMA`、`C$OMP` または `!MIC$` で開始できるようになりました。指令の要約については、付録 E を参照してください。標準 Fortran 95 には、指令の機能はありません。

## タブ書式

f95 の固定形式のタブ書式のソーステキストは以下のように定義されます。

- 1 カラム目 ~ 6 カラム目にタブがある場合、行はタブ書式のソース行になります。
- 注釈インジケータまたは文番号をタブの前に挿入できます。
- タブが最初の空白文字でない場合：
  - タブの次の文字がゼロ以外の数字でない場合は、タブの次のテキストが開始行になります。
  - 最初のタブの次の文字がゼロ以外の数字の場合は、行は継続行になります。
- f95 のデフォルトの行の最大長は、固定形式の場合は 72 カラムで、自由形式の場合は 132 カラムです。-e コンパイラオプションを使用すると、固定形式のソース内の行を 132 カラムまで拡張できます。

例：左側のタブ書式のソースは右側のようにになります。

<pre>!^IUses of tabs ^I CHARACTER *3 A = 'A' ^I INTEGER B = 2 ^I REAL C = 3.0 ^I WRITE(*,9) A, B, C 9^I FORMAT(1X, A3, ^I1 I3, ^I2 F9.1 ) ^IEND</pre>	<pre>! Uses of tabs CHARACTER *3 A = 'A' INTEGER B = 2 REAL C = 3.0 WRITE(*,9) A, B, C 9 FORMAT(1X, A3 1 I3, 2 F9.1) END</pre>
---	--

この例では、タブ文字は「^I」で示されており、「1」または「2」で始まっている行が継続行です。また、1つの書式ではなく、さまざまなタブ書式がコーディングで示されています。

## 使用するソースの書式

f95 で使用するソースの書式は、オプション、指令、拡張子によって異なります。

表 C-1 F95 ソース書式のコマンド行のオプション

オプション	作用
-fixed	すべてのソースファイルが Fortran の固定形式で記述されていると解釈します。
-free	すべてのソースファイルが Fortran の自由形式で記述されていると解釈します。

-free および -fixed オプションは、ファイル名の拡張子よりも優先されます。FREE 指令または FIXED 指令を使用すると、オプションおよびファイル名の拡張子よりも優先されます。

## 書式の混在

以下のように、異なるソースの書式を混在させてもかまいません。

- 1 つの f95 のコマンドで、固定形式のソースファイルと自由形式のソースファイルを混在させることができます。
- 指令を使用することによって、1 つのファイル内で、自由形式と固定形式を混在させることができます。
- 1 つのプログラム単位で、タブ書式を自由形式または固定形式と混在させることができます。

## 大文字・小文字の区別

Sun Fortran 95 では、デフォルトでは大文字・小文字が区別されません。すなわち、AbcDeF という変数は、abcdef と同じ文字列として扱われます。-u オプションを付けてコンパイルすると、コンパイラは大文字と小文字を区別します。

## 既知の制限

Fortran 95 のプログラム単位には、最高 65,535 個の導出型、および最高 16,777,215 個の定数を定義できます。

## ブール (Boolean) 型

f95 では、ブール型の定数と式をサポートしています。ただし、ブール型の変数、配列、文はサポートしていません。

### ブール型に関する規則

- マスク処理 -- ビット単位の論理式ではブール型の結果が生成されます。個々のビットは、対応する演算対象のビットで行われた 1 つまたは複数の論理演算の結果を表します。
- 2 進の算術演算子および関係演算子では、以下のように処理されます。
  - 一方の演算対象がブール型の場合は、そのまま演算が実行されます。
  - 両方の演算対象がブール型の場合は、両者を整数であるとみなして演算が実行されます。
- ユーザー定義の関数によってブール型の結果を生成することはできません。ただし、一部の (標準でない) 組み込み関数では可能です。
- ブール型と論理型には、以下のような相違点があります。
  - 変数、配列、関数は論理型にできますが、ブール型にはできません。
  - LOGICAL 文はありますが、BOOLEAN 文はありません。
  - 論理変数や論理定数が表せるのは 1 つの値だけですが、ブール型の定数は 32 個まで値を表すことができます。
  - 論理式は 1 つの値を生成し、ブール型の式は最大 32 個の値を生成することができます。
  - 論理型は、算術式、関係式、ビット単位の論理式で使用できませんが、ブール型はこれらすべてにおいて有効です。

### ブール型定数の別の書式

f95 では、ブール型定数 (8 進、16 進、ホレリス) を、以下に示すような書式 (2 進ではありません) で使用することができます。標準の Fortran では、変数をブール型と宣言することはできません。

## 8 進

書式は `dddddB` です。 *d* は任意の 8 進数です。

- B または b のどちらの文字を使用してもかまいません。
- 1 桁から 11 桁までの 8 進数 (0 から 7) を使用できます。
- 11 桁の 8 進数は 32 ビットの完全なワードを表します。左端の数字は常に 0、1、2、3 のいずれかです。
- 8 進の個々の数字は 3 ビットの値を表します。
- 右端の桁は、右 3 ビット (29、30、31 ビット) の内容を表します。
- 11 桁未満の場合は、値は右揃えされます。ワードの右端にある *n* ビットから 31 ビットまでが使用され、それ以外のビットはゼロになります。
- 空白は無視されます。

入出力の書式指定では B という文字は 2 進数であることを示しますが、それ以外の場合は 8 進数であることを表します。

## 16 進

*d* が任意の 16 進の数字である `X'ddd'` または `X"ddd"` の書式です。

- 1 桁から 8 桁までの 16 進数 (0 から 9、A から F) を使用できます。
- 文字は大文字でも小文字でもかまいません (X、x、A から F、a から f)。
- 数字は引用符 (アポストロフィ) または二重引用符で囲まなければなりません。
- 空白は無視されます。
- 16 進数の始めに + か - の記号を付けてもかまいません。
- 8 桁の 16 進数は 32 ビットの完全なワードを表しています。この 32 ビットワードの各ビットの内容は、同じ値を表す 2 進数に対応しています。
- 8 桁未満の場合は、値は右揃えされます。ワードの右端にある *n* ビットから 31 ビットまでが使用され、それ以外のビットはゼロになります。

## ホレリス

ホレリスデータには、以下の書式を使用できます。

$nH \dots ' \dots 'H" \dots "H$

$nL \dots ' \dots 'L" \dots "L$

$nR \dots ' \dots 'R" \dots "R$

上記の  $\dots$  は文字列を表し、 $n$  は文字数を表します。

- ホレリス定数はブール型です。
- ビット単位の論理式に文字定数がある場合は、その式はホレリスとみなされます。
- ホレリス定数には 4 文字まで使用することができます。

例：8 進と 16 進の定数の表現例を示します。

ブール型定数	内部の 8 進数
0B	00000000000
77740B	0000077740
X"ABE"	0000005276
X"-340"	37777776300
X'1 2 3'	0000000443
X'FFFFFFFFFFFFFFF'	37777777777

例：代入文での 8 進と 16 進の使用例を示します。

$i = 1357B$

$j = X"28FF"$

$k = X'-5A'$

算術式の中で 8 進数または 16 進数の定数を使用すると、結果が未定義になることがあります。ただし、構文エラーにはなりません。



## 別の場所におけるブール型定数の使用

f95 では、DATA 文以外の場所で BOZ 定数を使用することができます。

```
B' bbb' O' ooo' Z' zzz'
```

```
B" bbb" O" ooo" Z" zzz"
```

このような BOZ 定数が実数変数に代入されている場合には、型は変換されません。

標準の Fortran では、BOZ 定数は DATA 文でしか使用できません。

## 数値データ型のサイズの略記法

f95 では、宣言文、関数文、IMPLICIT 文において、以下のような非標準の書式で型を宣言することができます。

表 C-2 数値データ型のサイズの表記法

非標準	宣言子	短縮書式	意味
INTEGER*1	INTEGER (KIND=1)	INTEGER (1)	1 バイトの符号付き整数
INTEGER*2	INTEGER (KIND=2)	INTEGER (2)	2 バイトの符号付き整数
INTEGER*4	INTEGER (KIND=4)	INTEGER (4)	4 バイトの符号付き整数
LOGICAL*1	LOGICAL (KIND=1)	LOGICAL (1)	1 バイト論理型
LOGICAL*2	LOGICAL (KIND=2)	LOGICAL (2)	2 バイト論理型
LOGICAL*4	LOGICAL (KIND=4)	LOGICAL (4)	4 バイト論理型
REAL*4	REAL (KIND=4)	REAL (4)	IEEE の単精度浮動小数点数 (4 バイト)
REAL*8	REAL (KIND=8)	REAL (8)	IEEE の倍精度浮動小数点数 (8 バイト)
REAL*16	REAL (KIND=16)	REAL (16)	IEEE の 4 倍精度浮動小数点 (16 バイト)
COMPLEX*8	COMPLEX (KIND=4)	COMPLEX (4)	単精度複素数 (各部に 4 バイト)
COMPLEX*16	COMPLEX (KIND=8)	COMPLEX (8)	倍精度複素数 (各部に 8 バイト)
COMPLEX*32	COMPLEX (KIND=16)	COMPLEX (16)	4 倍精度複素数 (各部に 16 バイト)

1 列目の形式は一般に使用されていますが、非標準の Fortran 95 です。2 列目の種別番号はベンダーにより変わります。

## Cray ポインタ

Cray ポインタとは、別の変数や配列などのアドレスを値にもつ変数のことです。この別のもののことを、指示先と呼びます。

f95 は Cray ポインタをサポートしています。標準の Fortran 95 ではサポートされていません。

### 構文

Cray ポインタの POINTER 文は以下の形式で記述します。

```
POINTER ( ポインタ名, 指示先名 [配列の仕様] ), ...
```

ポインタ名、指示先名、配列の仕様のそれぞれの意味は、以下のとおりです。

ポインタ名	対応する <指示先名> へのポインタです。ポインタ名には指示先のアドレスが含まれます。 制限事項: ポインタ名にはスカラー変数名を指定してください (派生型は指定できません)。 定数、構造体の名前、配列、関数を指定することはできません。
指示先名	対応する <ポインタ名> の指示先です。 制限事項: 変数名、配列の宣言子、配列名を指定してください。
配列の仕様	配列の仕様を指定する場合は、明示的な実体があるもの (定数または非定数のサイズを持つもの)、または仮のサイズをもつものを指定してください。

例: 2 つの指示先に対して Cray ポインタを宣言します。

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

上記の例では、Cray ポインタ p とその指示先 b、Cray ポインタ q とその指示先 c を宣言しています。

例：配列に対して Cray ポインタを宣言します。

```
POINTER ( ix, x(n, 0:m) )
```

この例では、Cray ポインタ `ix` とその指示先 `x` を宣言しています。同時に、`x` は `n` と `m-1` 次元の配列であることを宣言しています。

## Cray ポインタの目的

ポインタを使用すると、記憶領域の特定の場所に変数を動的に対応づけ、ユーザーが管理する記憶領域にアクセスすることができます。

Cray ポインタでは、メモリーの絶対的な場所にアクセスすることができます。

最適化のために同じ値をもつポインタは存在しないと仮定しているため、Cray ポインタではリンクをリストして操作することはできません。

## Cray ポインタと Fortran のポインタ

Cray ポインタは次のように宣言します。

```
POINTER ( ポインタ名, 指示先名 [配列の仕様] )
```

Fortran のポインタは次のように宣言します。

```
POINTER オブジェクト名
```

この 2 種類のポインタを混在させることはできません。

## Cray ポインタの機能

- 指示先が引用されるたびに、`%95` 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、領域の 1 単位を占めます。値が占める領域は使用中のマシンのメモリーサイズによって異なります。

- Cray ポインタは COMMON の並びまたは仮引数で使用することができます。
- Cray ポインタの値が定義されるまでは、指示先にアドレスはありません。
- 指示先として配列が指定されている場合、その配列を指示先配列と呼びます。

この場合の配列の宣言子は以下の場所に指定することができます。

- 独立した型宣言文
- 独立した DIMENSION 文
- ポインタ文自体
- 配列の宣言子が副プログラムにある場合、次元の設定は以下の場所で確認できます。
  - 共通ブロックにある変数、または
  - 仮引数である変数
- 各次元のサイズは、指示先が引用されるときではなく、副プログラムの処理が始まるときに認識されます。

## Cray ポインタの制限事項

- 指示先名は、CHARACTER\*(\*) で型宣言された変数であってはなりません。
- 指示先名が配列の宣言子である場合は、明示的な実体があるもの、(定数または非定数のサイズを持つもの)、または仮のサイズをもつものでなければなりません。
- Cray ポインタを配列にすることはできません。
- Cray ポインタを以下のように扱うことはできません。
  - 別の Cray ポインタまたは Fortran のポインタの引用先にする
  - 構造体の成分にする
  - 他のデータ型で宣言する
- Cray ポインタを以下の場所で使用することはできません。
  - PARAMETER 文または PARAMETER 属性を含む型宣言文
  - DATA 文

## Cray ポインタの指示先の制限事項

- Cray ポインタの指示先を、SAVE、DATA、EQUIVALENCE、COMMON、PARAMETER 文で使用することはできません。
- Cray ポインタの指示先を仮引数にすることはできません。
- Cray ポインタの指示先を関数値にすることはできません。
- Cray ポインタの指示先を構造体または構造体の成分にすることはできません。
- Cray ポインタの指示先を派生型にすることはできません。

## Cray ポインタの使用方法

Cray ポインタには以下のようにして値を割り当てることができます。

- 絶対アドレスに設定します。

例： $q = 0$

- 整数変数の加減式によって割り当てます。

例： $p = q + 100$

- Cray ポインタは整数ではありません。Cray ポインタを実数変数に割り当ててことはできません。
- 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： $p = \text{LOC} ( x )$

## 例 : Cray ポインタの使用例

```
SUBROUTINE sub ( n )
COMMON pool(100000)
INTEGER blk(128), word64
REAL a(1000), b(n), c(100000-n-1000)
POINTER ( pblk, blk ), ( ia, a ), ( ib, b ), &
        ( ic, c ), ( address, word64 )
DATA address / 64 /
pblk = 0
ia = LOC( pool )
ib = ia + 1000
ic = ib + n
...
```

上記の例を説明します。

- word64 は絶対アドレス 64 の内容を参照します。
- blk はメモリーの最初の 128 ワードを占める配列です。
- a は空白共通ブロックにある配列で、長さは 1,000 です。
- b は a の後に位置し、長さは n です。
- c は b の後に位置します。
- a、b、c は pool 領域に関連付けられています。
- word64 は blk(17) と同じです。Cray ポインタはバイトアドレスであり、blk の整数要素はそれぞれ 4 バイトの長さがあるためです。

## 最適化と Cray ポインタ

最適化を図るため、f95 では指示先の領域は他の変数の領域とは重ならないようにしています。つまり、指示先は他の変数とは関連づけられていないとみなされます。

ただし、以下の 2 つの場合には関連が生じる可能性がありますので注意してください。

- 1 つの Cray ポインタに 2 つの指示先がある、または

- 2つの Cray ポインタが同じ値をもっている

---

注 – このような関連を防ぐのはプログラマの責任です。

---

このような関連付けは、配列の等値化などで意図的に行われる場合もあります。ただし、このときの実行結果は最適化を実行するかどうかによって異なる可能性があります。

例 : b と c のポインタは同じです。

```

    POINTER ( p, b ), ( p, c )
    REAL x, b, c
    p = LOC( x )
    b = 1.0
    c = 2.0
    PRINT *, b
    ...

```

上記の例では、b と c のポインタが同じであるため、c に 2.0 を代入すると同じ値が b にも割り当てられます。このため、b に 1.0 がすでに代入されていても、出力結果は 2.0 になります。

---

## 組み込み関数

f95 は標準の処理を拡張した組み込み関数をサポートしています。

表 C-3 非標準の組み込み関数

名前	定義	関数の型	引数	引数	備考
COT	余接関数	実数	実数	( [X] =x )	P, E
DDIM	正差	倍精度実数	倍精度実数	( [X] =x, [Y=] y )	P, E

表 C-3 非標準の組み込み関数 (続き)

名前	定義	関数の型	引数	引数	備考
LEADZ	先行の 0 ビットの数を調べる	整数	ブール型、整数、実数、ポインタ	([I=] i)	NP, I
POPCNT	設定されたビットの数を調べる	整数	ブール型、整数、実数、ポインタ	([I=] i)	NP, I
POPPAR	ビットの設定数のパリティを演算する	整数	ブール型、整数、実数、ポインタ	([X=] x)	NP, I

上記の表の備考欄の意味は以下のとおりです。

備考	意味
P	名前を引数として渡すことができる
NP	名前を引数として渡すことはできない
E	組み込み関数の外部コードは実行時に呼び出される
I	f95 が組み込み関数のインラインコードを生成する

## 入出力拡張機能

Sun FORTRAN 77 の一部の入出力拡張機能が Fortran 95 のコンパイラに追加されました。

### ■ NAMELIST 入出力形式

入力時にグループ名の先頭に \$ または & が付きます。& は、Fortran 95 の規格だけが受け付ける形式であり、NAMELIST 出力によっても書き込まれます。

入力の終了を表す記号として \$ を受け付けます。ただし、グループ内の最後のデータ項目が CHARACTER データである場合は別です。その場合、\$ は、入力データとして扱われます。

NAMELIST 入力は、レコードの最初のコラムから開始することができます。

- OPEN (... , FORM='BINARY') は、レコードマークなしのバイナリデータとしてファイルを扱います。



FORM=' BINARY' が指定されたファイルのオープン、FORM=' UNFORMATTED' を指定した場合と同じ効果があります。ただし、レコード長がファイルに埋め込まれません。このデータがないと、どこでレコードが開始し、どこで終了するかを示す手段がありません。したがって、FORM=' BINARY' が指定されたファイルに BACKSPACE 操作を行うことはできません。なぜなら、どこまで現在記録を前に位置付けするかを指示する方法がないからです。' BINARY' ファイルに対する READ は、入力リスト上の変数を満たすのに必要なだけのデータを読み込みます。

- WRITE 文：データは、バイナリ形式のファイルに書き込まれます。出力リストで指定したバイト数が転送されます。
- READ 文：データは、入力リスト上の変数に読み込まれます。リストに必要なバイト数が転送されます。ファイル上にレコードマークがないので、「レコードの終わり」というエラーが検出される可能性はありません。検出されるエラーは、「ファイルの終わり」またはシステムの異常だけです。
- INQUIRE 文：FORM="BINARY" を指定してオープンされたファイルに対する INQUIRE では、次のものが返されます。

```
FORM=" BINARY"  
ACCESS=" SEQUENTIAL"  
DIRECT=" NO"  
FORMATTED=" NO"  
UNFORMATTED=" YES"  
RECL= AND NEXTREC= は、未定義。
```

- BACKSPACE 文：使用できません。エラーが返されます。
- ENDFILE 文：通常どおり、現在の位置でファイルが打ち切られます。
- REWIND 文：通常どおり、ファイルをデータの開始に位置付けます。
- さまざまな装置に対し再帰的な入出力が可能です (これは、f95 の入出力ライブラリが「MT-Warm」だからです)。
- RECL=2147483646 (2<sup>31</sup> -2) は、順番に書式化された、並びによる変数群 出力上のデフォルトのレコード長です。
- ENCODE および DECODE は、『FORTRAN 77 言語リファレンス』で説明するように認識され、実装されています。
- スクラッチファイルの命名方法は、f77 と同じです。
- 次に示すように、ADVANCE=' NO' で非前進入出力が可能になります。

```
write(*,'(a)',ADVANCE='NO') 'n= '  
read(*,*) n
```

---

## 指令

コンパイラ指令は、特別な動作をするようにコンパイラに指示します。プラグマとも呼ばれます。

コンパイラ指令は 1 行または複数行のテキストとしてソースプログラムに挿入されます。コンパイラ指令は一見注釈に似ていますが、注釈にはない特別な文字が付加されています。Fortran 95 以外のほとんどのコンパイラでは指令を注釈として扱うので、コードの一定の移植性は保たれます。

f95 (および f77) では、サン形式の指令がデフォルトとして設定されています。Cray 形式の指令に切り換えるには、コンパイラコマンド行フラグの `-mp=cray` を使用してください。

Fortran 指令については、付録 E にまとめられています。

### f95 の特殊な指令行の書式

f95 は、第 2 章に示すような f95/f77 の一般的な指令に加え、独自の特殊な指令を認識します。構文は次のようになります。

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

### ソースが固定形式の場合

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

## ソースが自由形式の場合

- `!DIR$` の後に空白を 1 つつけて、行の任意の位置に記述できます。`!DIR$` 文字は、その行の空白でない最初の文字となります。
- 指令は空白の後に記述します。
- 新たに始まる指令行では、`!DIR$` の直後に空白、タブ、または改行が続きます。
- 指令の継続行では、`!DIR$` の直後に空白、タブ、改行以外の文字が続きます。

これらのことから、`!DIR$` を 1 カラム目から 5 カラム目に記述しておけば、自由形式または固定形式のどちらのソースでも機能することがわかります。

## FIXED 指令と FREE 指令

指令行の後に続くソース行の書式を指定します。

### 範囲

指令が適用される範囲は、ファイル内に指令が出現してから最後までの部分、または次に `FREE` あるいは `FIXED` が出現するまでの部分です。

### 使用法

- 1 つのソースファイル内でソースの書式を切り換えることができます。
- `INCLUDE` ファイルのソースの書式を切り換えることができます。`INCLUDE` ファイルの先頭に指令を挿入すると、`INCLUDE` ファイルが処理された後に、ソースの書式が `INCLUDE` ファイルの処理前の書式に戻ります。

### 制限事項

`FREE` 指令と `FIXED` 指令には以下の制限事項があります。

- どちらの指令もコンパイラの指令行に単独で指定します (継続行にしないでください)。
- どちらの指令もソースコードの任意の位置に指定できます。その他の指令は作用するプログラム中に指定する必要があります。

例：FREE 指令を指定します。

```
!DIR$ FREE
  DO i = 1, n
    a(i) = b(i) * c(i)
  END DO
```

## 並列化の指令

並列化の指令は、コンパイラに次の DO ループの並列化処理を指示する特別な注釈です。これらに関する概要は、付録 E と『Fortran プログラミングガイド』に記載されています。[f95](#) は、[f77](#) の Sun および Cray 形式の並列化指令だけでなく、OpenMP Fortran API 指令も認識します。

---

注 – Fortran の並列化機能には、Sun WorkShop HPCのライセンスが必要です。

---

## Fortran 77 との互換性

### ソース

標準に準拠した固定形式 (`filename.F`) の FORTRAN 77 のソースコードにはサンの Fortran 95 との互換性が保証されます。たとえば、VMS Fortran の機能などの標準以外の拡張機能を使用すると、互換性がなくなり、サンの Fortran 95 でのコンパイルはできません。

### 制約

[f77](#) コンパイラでは、配列添え字を最高 20 個まで使用できますが、[f90](#) では、7 個しか使用できません。

## f77 でコンパイルしたルーチンとのリンク

- f77 と f95 のオブジェクトバイナリを混在させるには、f95 および f77 互換性ライブラリ libf77compat (libF77 ではない) とリンクします。たとえば、メインプログラムが f77 プログラムの場合でも、f95 `..files. -lf77compat` でリンクを行います。

例：f95 のメインと f77 のサブルーチンです。

```
demo$ cat m.f95
CHARACTER*74 :: c = 'テストです。'
      CALL echo1( c )
END
demo$ cat s.f
      SUBROUTINE echo1( a )
      CHARACTER*74 a
      PRINT*, a
      RETURN
      END
demo$ f77 -c -silent s.f
demo$ f95 m.f95 s.o -lf77compat
demo$ a.out
      テストです。
demo$
```

- 通常、FORTRAN 77 ライブラリは f95 と互換性があります。

例：FORTRAN 77 のライブラリからルーチン呼び出す f95 のメインです。

```
demo$ cat tdttime.f95
      REAL e, dtime, t(2)
      e = dtime( t )
      DO i = 1, 100000
          as = as + cos(sqrt(float(i)))
      END DO
      e = dtime( t )
      PRINT *, '経過:', e, ', ユーザー:', t(1), ', システム:', t(2)
      END
demo$ f95 tdttime.f95
demo$ a.out
経過:0.14, ユーザー:0.14, システム:0.0E+0
demo$
```

[dtime\(3F\)](#) を参照してください。

## 入出力

f95 が f77 の互換ライブラリにリンクされるので、f77 と f95 のバイナリ入出力には互換性があります。

バイナリ入出力に互換性があるので、以下のような状況でも実行することができます。

- 同じプログラムで、ある記録を f95 で書き込み、f77 で読み込むことができます。
- f95 のプログラムでファイルを書き込み、f77 のプログラムで読み込むことができます。

読み込まれた数字と書き込まれた数字が一致しない場合があります。

- 書式なしの場合  
読み込まれた数字と書き込まれた数字は同じです。

- 浮動小数点の書式付きの場合

読み込まれた数字と書き込まれた数字が一致しないことがあります。これは、基数変換ルーチンが若干異なることや、大文字・小文字、空白、プラス・マイナス記号などの規則が異なるためです。

例 : 1.0e12、1.0E12、1.0E+12

- 並びによる入出力の場合

読み込まれた数字と書き込まれた数字とが異なることがあります。これは、コンマ、空白、ゼロ、繰り返し因子などの配置規則が異なるためです。

例 : '0.0' と '.0'

例 : '7' と '7'

例 : '3,4,5' と '3 4 5'

例 : '3\*0' と '0 0 0'

上記の問題は以下のような式で発生します。

```
integer::v(3)=(/0,0,0/); print *,v
```

例 : '0.333333343' と '0.333333'

上記の問題は以下のような式で発生します。

```
PRINT *, 1.0/3.0
```

## 組み込み関数

Fortran 95 の標準機能として、以下に示すような組み込み関数が新たにサポートされました。これらの関数は FORTRAN 77 にはありません。

以下の名前をユーザープログラムで使用する場合は、EXTERNAL 文を追加し、組み込み関数ではなくユーザー関数を使用するように f95 に指示する必要があります。

Fortran 95 の組み込み関数には、以下のものがあります。

```
ADJUSTL, ADJUSTR, ALL, ALLOCATED, ANY, BIT_SIZE, COUNT, CSHIFT,  
DIGITS, DOT_PRODUCT, EOSHIFT, EPSILON, EXPONENT, HUGE, KIND,  
LBOUND, LEN_TRIM, MATMUL, MAXEXPONENT, MAXLOC, MAXVAL, MERGE,  
MINEXPONENT, MINLOC, MINVAL, NEAREST, PACK, PRECISION,  
PRESENT, PRODUCT, RADIX, RANGE, REPEAT, RESHAPE, RRSPACING,
```

SCALE, SCAN, SELECTED\_INT\_KIND, SELECTED\_REAL\_KIND,  
SET\_EXPONENT, SHAPE, SIZE, SPACING, SPREAD, SUM, TINY,  
TRANSFER, TRANSPOSE, UBOUND, UNPACK, VERIFY

---

## 将来のバージョンとの互換性

ソースコードは、f95 の本リリースと将来のリリースで互換となる予定です。

f95 の本リリースでモジュール情報ファイルを作成する場合、そのファイルが将来のリリースと互換性があるかどうかは保証されません。

---

## 言語の混在

Solaris システムでは、C で書かれたルーチンを Fortran のプログラムと組み合わせることが可能です。C と Fortran では呼び出し規則が共通なためです。

---

## モジュールファイル

Fortran 95 の MODULE を含むファイルをコンパイルすると、ソースで検出された MODULE ごとにモジュールファイル (.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。

デフォルトでは、このようなファイルは現在作業中のディレクトリで検索されます。-mdir オプションを指定すると、別の場所も検索するように f95 に指示することができます。

.mod ファイルをアーカイブファイルに格納したり、1 つのファイルに結合したりすることはできません。



## 付録 D

### -xtarget プラットフォームの展開

この付録では、`-xtarget` オプションのプラットフォームシステム名とその展開について説明します。

表 D-1 に示すように、`-xtarget` の値は `-xarch`、`-xchip`、`-xcache` オプションのそれぞれの値に展開されます。`fpversion(1)` を実行すると、対象システムの定義を確認できます。

たとえば、`-xtarget=sun4/15` は、`-xarch=v8a` `-xchip=micro` `-xcache=2/16/1` を意味します。

表 D-1 `-xtarget` の展開

<code>-xtarget=</code>	<code>-xarch</code>	<code>-xchip</code>	<code>-xcache</code>
generic	generic	generic	generic
cs6400	v8	super	16/32/4:2048/64/1
entr150	v8	ultra	16/32/1:512/64/1
entr2	v8	ultra	16/32/1:512/64/1
entr2/1170	v8	ultra	16/32/1:512/64/1
entr2/1200	v8	ultra	16/32/1:512/64/1
entr2/2170	v8	ultra	16/32/1:512/64/1
entr2/2200	v8	ultra	16/32/1:512/64/1
entr3000	v8	ultra	16/32/1:512/64/1
entr4000	v8	ultra	16/32/1:512/64/1
entr5000	v8	ultra	16/32/1:512/64/1
entr6000	v8	ultra	16/32/1:512/64/1
sc2000	v8	super	16/32/4:2048/64/1

表 D-1 -xtarget の展開 (続き)

-xtarget=	-xarch	-xchip	-xcache
solb5	v7	old	128/32/1
solb6	v8	super	16/32/4:1024/32/1
ss1	v7	old	64/16/1
ss10	v8	super	16/32/4
ss10/20	v8	super	16/32/4
ss10/30	v8	super	16/32/4
ss10/40	v8	super	16/32/4
ss10/402	v8	super	16/32/4
ss10/41	v8	super	16/32/4:1024/32/1
ss10/412	v8	super	16/32/4:1024/32/1
ss10/50	v8	super	16/32/4
ss10/51	v8	super	16/32/4:1024/32/1
ss10/512	v8	super	16/32/4:1024/32/1
ss10/514	v8	super	16/32/4:1024/32/1
ss10/61	v8	super	16/32/4:1024/32/1
ss10/612	v8	super	16/32/4:1024/32/1
ss10/71	v8	super2	16/32/4:1024/32/1
ss10/712	v8	super2	16/32/4:1024/32/1
ss10/hs11	v8	hyper	256/64/1
ss10/hs12	v8	hyper	256/64/1
ss10/hs14	v8	hyper	256/64/1
ss10/hs21	v8	hyper	256/64/1
ss10/hs22	v8	hyper	256/64/1
ss1000	v8	super	16/32/4:1024/32/1
ss1plus	v7	old	64/16/1
ss2	v7	old	64/32/1
ss20	v8	super	16/32/4:1024/32/1
ss20/151	v8	hyper	512/64/1
ss20/152	v8	hyper	512/64/1

表 D-1 -xtarget の展開 (続き)

-xtarget=	-xarch	-xchip	-xcache
ss20/50	v8	super	16/32/4
ss20/502	v8	super	16/32/4
ss20/51	v8	super	16/32/4:1024/32/1
ss20/512	v8	super	16/32/4:1024/32/1
ss20/514	v8	super	16/32/4:1024/32/1
ss20/61	v8	super	16/32/4:1024/32/1
ss20/612	v8	super	16/32/4:1024/32/1
ss20/71	v8	super2	16/32/4:1024/32/1
ss20/712	v8	super2	16/32/4:1024/32/1
ss20/hs11	v8	hyper	256/64/1
ss20/hs12	v8	hyper	256/64/1
ss20/hs14	v8	hyper	256/64/1
ss20/hs21	v8	hyper	256/64/1
ss20/hs22	v8	hyper	256/64/1
ss2p	v7	powerup	64/32/1
ss4	v8a	micro2	8/16/1
ss4/110	v8a	micro2	8/16/1
ss4/85	v8a	micro2	8/16/1
ss5	v8a	micro2	8/16/1
ss5/110	v8a	micro2	8/16/1
ss5/85	v8a	micro2	8/16/1
ss600/120	v7	old	64/32/1
ss600/140	v7	old	64/32/1
ss600/41	v8	super	16/32/4:1024/32/1
ss600/412	v8	super	16/32/4:1024/32/1
ss600/51	v8	super	16/32/4:1024/32/1
ss600/512	v8	super	16/32/4:1024/32/1
ss600/514	v8	super	16/32/4:1024/32/1
ss600/61	v8	super	16/32/4:1024/32/1

表 D-1 -xtarget の展開 (続き)

<b>-xtarget=</b>	<b>-xarch</b>	<b>-xchip</b>	<b>-xcache</b>
ss600/612	v8	super	16/32/4:1024/32/1
sselc	v7	old	64/32/1
ssipc	v7	old	64/16/1
ssipx	v7	old	64/32/1
sslc	v8a	micro	2/16/1
sslt	v7	old	64/32/1
sslx	v8a	micro	2/16/1
sslx2	v8a	micro2	8/16/1
ssslc	v7	old	64/16/1
ssvyger	v8a	micro2	8/16/1
sun4/110	v7	old	2/16/1
sun4/15	v8a	micro	2/16/1
sun4/150	v7	old	2/16/1
sun4/20	v7	old	64/16/1
sun4/25	v7	old	64/32/1
sun4/260	v7	old	128/16/1
sun4/280	v7	old	128/16/1
sun4/30	v8a	micro	2/16/1
sun4/330	v7	old	128/16/1
sun4/370	v7	old	128/16/1
sun4/390	v7	old	128/16/1
sun4/40	v7	old	64/16/1
sun4/470	v7	old	128/32/1
sun4/490	v7	old	128/32/1
sun4/50	v7	old	64/32/1
sun4/60	v7	old	64/16/1
sun4/630	v7	old	64/32/1
sun4/65	v7	old	64/16/1
sun4/670	v7	old	64/32/1

表 D-1 -xtarget の展開 (続き)

<b>-xtarget=</b>	<b>-xarch</b>	<b>-xchip</b>	<b>-xcache</b>
sun4/690	v7	old	64/32/1
sun4/75	v7	old	64/32/1
ultra	v8	ultra	16/32/1:512/64/1
ultra1/140	v8	ultra	16/32/1:512/64/1
ultra1/170	v8	ultra	16/32/1:512/64/1
ultra1/200	v8	ultra	16/32/1:512/64/1
ultra2	v8	ultra2	16/32/1:512/64/1
ultra2/1170	v8	ultra	16/32/1:512/64/1
ultra2/1200	v8	ultra	16/32/1:1024/64/1
ultra2/1300	v8	ultra2	16/32/1:2048/64/1
ultra2/2170	v8	ultra	16/32/1:512/64/1
ultra2/2200	v8	ultra	16/32/1:1024/64/1
ultra2/2300	v8	ultra2	16/32/1:2048/64/1
ultra2i	v8	urtra2i	16/32/1:i512/64/1
ultra3	v8	ultra3	64/32/4:8192/256/1



## 付録 E

---

### Fortran 指令の要約

---

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

- 一般的な Fortran 指令
- Sun の並列化指令
- Cray の並列化指令
- OpenMP の Fortran 95 指令、ライブラリルーチン、および環境

---

注 – Fortran 並列化機能を使用するには、Sun WorkShop HPC ライセンスが必要です。

---

## 一般的な Fortran 指令

f77 でも f95 でも適用される一般的な指令については、第 2 章で説明しています。

表 E-1 一般的な Fortran 指令の要約

---

### Format

```
C$PRAGMA キーワード ( a [ , a ] ... ) [ , keyword ( a [ , a ] ... ) ] , ...  
C$PRAGMA SUN キーワード ( a [ , a ] ... ) [ , keyword ( a [ , a ] ... ) ] , ...  
C$PRAGMA SPARC キーワード ( a [ , a ] ... ) [ , keyword ( a [ , a ] ... ) ] , ...
```

1 桁目のコメント指示子には、`c`、`C`、`!`、または `*` を使用できます (上記の例では、`C` を使用しています。f95 自由書式では `!` を使用しなければなりません)。

---

C 指令	C\$PRAGMA C ( <i>list</i> )
	外部関数の名前リストを C 言語のルーチンとして宣言します。
UNROLL 指令	C\$PRAGMA SUN UNROLL= <i>n</i>
	コンパイラに、次のループは長さ <i>n</i> に展開できることを伝えます。
WEAK 指令	C\$PRAGMA WEAK ( <i>name</i> [= <i>name2</i> ])
	<i>name</i> を弱い記号 (weak symbol) または <i>name2</i> のエイリアスとして宣言します。
OPT 指令	C\$PRAGMA SUN OPT= <i>n</i>
	サブプログラムの最適化レベルを <i>n</i> に設定します。
PIPELOOP 指令	C\$PRAGMA SUN PIPELOOP= <i>n</i>
	反復 <i>n</i> 間のループの依存性を別に宣言します。
PREFETCH 指令	C\$PRAGMA SPARC_PREFETCH_READ_ONCE ( <i>name</i> ) C\$PRAGMA SPARC_PREFETCH_READ_MANY ( <i>name</i> ) C\$PRAGMA SPARC_PREFETCH_WRITE_ONCE ( <i>name</i> ) C\$PRAGMA SPARC_PREFETCH_WRITE_MANY ( <i>name</i> )
	コンパイラに <i>name</i> へ参照したときに先読み命令を生成するように要求します ( <code>-xprefetch</code> オプションを指定する必要があります)。

---



---

## 特殊な Fortran 95 指令

次の指令は、f95 でのみ使用できます。詳細については、付録 C を参照してください。

表 E-2 特殊な Fortran 95 指令

---

書式	!DIR\$ <i>directive</i> : 最初の行 !DIR\$& ... : 継続行
	固定書式の場合、C は CDIR\$ 指令 ... のように指令指示子としても受け取られます。その行は、1 桁目から始まる必要があります。 自由書式ソースの場合、その行の前でも空白を使用できます。
FIXED/FREE 指令	!DIR\$ FREE !DIR\$ FIXED
	これらの指令では、それ以降の行のソースフォーマットを指定します。これらは、次に FREE 指令または FIXED 指令が出現するまで、残りのソースファイルに適用されます。

---

## Sun の並列化指令

Sun 形式の並列化指令は、デフォルト (`-mp=sun` コンパイラオプション)であり、「Fortran プログラミングガイド」の並列化に関する章で詳しく説明しています。

表 E-3 Sun 形式の並列化指令の要約

書式	<code>C\$PAR directive [optional_qualifiers]</code> : 最初の行 <code>C\$PAR&amp; [more_qualifiers]</code> : 継続行
	固定書式。指令指示子には、 <code>C</code> (上記の例)、 <code>c</code> 、*、または <code>!</code> を使用できません。 修飾子が複数ある場合は、コンマで区切ります。72 桁目を越える文字は、 <code>-e</code> コンパイラオプションを指定していない限り無視されません。
TASKCOMMON 指令	<code>C\$PAR TASKCOMMON block_name</code>  共通ブロック <code>block_name</code> の変数をスレッド非公開として宣言します。これは、スレッドに対しては非公開ですが、スレッド内ではグローバルとなります。共通ブロック <code>TASKCOMMON</code> を宣言するには、そのブロックの全共通宣言の後にこの指令を指定する必要があります。
DOALL 指令	<code>C\$PAR DOALL [修飾子]</code>  それ以降の DO ループを並列化します。修飾子は、次のとおりです。 <code>PRIVATE(list)</code> リストの名前を PRIVATE として宣言します。 <code>SHARED(list)</code> リストの名前を SHARED として宣言します。 <code>MAXCPUS(n)</code> 多くても使用するスレッドは <code>n</code> 個です。 <code>READONLY(list)</code> リストの変数は、ループ内で変更されません。 <code>SAVELAST</code> すべての非公開変数の最終的な値を保存します。 <code>STOREBACK(list)</code> リストの変数の最終的な値を保存します。 <code>REDUCTION(list)</code> リストの変数は、縮約変数です。 <code>SCHEDTYPE(type)</code> スケジューリング型を使用します (デフォルトは STATIC です)。 <code>STATIC</code> <code>SELF(nchunk)</code> <code>FACTORING[(m)]</code> <code>GSS[(m)]</code>
DOSERIAL 指令	<code>C\$PAR DOSERIAL</code>  以降のループの並列化を無効にします。
DOSERIAL* 指令	<code>C\$PAR DOSERIAL*</code>  以降のループのネストの並列化を無効にします。

## Cray の並列化指令

Cray 形式の並列化指令については、『Fortran プログラミングガイド』の並列化に関する章で詳しく説明しています。

`-mp=cray` コンパイラオプションを指定する必要があります。

表 E-4 Cray の並列化指令の要約

---

書式	<code>CMIC\$ directive qualifiers</code> : 最初の行 <code>CMIC\$&amp; [more_qualifiers]</code> : 継続行
	固定書式。指令指示子には、 <code>C</code> (上記の例)、 <code>c</code> 、 <code>*</code> 、または <code>!</code> を使用できます。 <code>f95</code> 自由書式の場合、 <code>!MIC\$</code> より前に空白を指定できます。
DOALL 指令	<code>CMIC\$ DOALL SHARED(list), PRIVATE(list) [, more_qualifiers]</code>  以降のループの並列化を無効にします。修飾子は、次のとおりです。  スコーピング修飾子が必要です ( <code>list</code> が空でない場合) ループ内の変数はすべて <code>PRIVATE</code> 句または <code>SHARED</code> 句で宣言されていなければなりません。 <code>PRIVATE(list)</code> リストの名前を <code>PRIVATE</code> として宣言します。 <code>SHARED(list)</code> リストの名前を <code>SHARED</code> として宣言します。 <code>AUTOSCOPE</code> 変数の範囲を自動的に判別します。  次にオプションを示します。 <code>MAXCPUS(n)</code> 多くても使用するスレッドは <code>n</code> 個です。 <code>SAVELAST</code> すべての非公開変数の最終的な値を保存します。  スケジューリング修飾子は 1 つだけ指定できます。 <code>GUIDED</code> Sun 形式の <code>GSS(64)</code> と等価です。 <code>SINGLE</code> Sun 形式の <code>SELF(1)</code> と等価です。 <code>CHUNKSIZE(n)</code> Sun 形式の <code>SELF(n)</code> と等価です。 <code>NUMCHUNKS(m)</code> Sun 形式の <code>SELF(n/m)</code> と等価です。 デフォルトのスケジューリング型は、Sun 形式の <code>STATIC</code> です。これと等価の Cray 形式のスケジューリング型はありません。これらのスケジューリング型の解釈方法は、Sun 形式と Cray 形式で異なります。詳細については、『Fortran プログラミングガイド』を参照してください。

---

表 E-4 Cray の並列化指令の要約 (続き)

TASKCOMMON 指令	CMIC\$ TASKCOMMON <i>block_name</i>  指定した共通ブロックの変数をスレッド非公開として宣言します。これは、スレッドに対しては非公開ですが、スレッド内ではグローバルとなります。共通ブロック <code>TASKCOMMON</code> を宣言するには、そのブロックの全共通宣言の直前または直後にこの指令を指定する必要があります。
DOSERIAL 指令	CMIC\$ DOSERIAL  以降のループの並列化を無効にします。
DOSERIAL* 指令	CMIC\$ DOSERIAL*  以降のループのネストの並列化を無効にします。

## Fortran 95 の OpenMP 指令

Sun の Fortran 95 コンパイラでは、OpenMP バージョン 1.1 の Fortran API がサポートされています。f95 コンパイラフラグの `-mp=openmp` および `-openmp` を使用すれば、これらの指令を使用できるようになります。

ここでは、f95 でサポートされている OpenMP 指令、ライブラリルーチン、環境変数を示します。OpenMP の詳細については、<http://www.openmp.org/> の OpenMP Fortran 仕様を参照してください。

次の表に、f95 でサポートされている OpenMP 指令を要約します。角括弧 ([...]) で囲まれている項目は、オプションです。

表 E-5 Fortran 95 における OpenMP 指令の要約

指令の形式 (固定)	C\$OMP <i>directive optional_clauses...</i> !\$OMP <i>directive optional_clauses...</i> *\$OMP <i>directive optional_clauses...</i>  1桁目から始まらなければなりません。継続行については、6桁メニュー空白以外の文字化ゼロ以外の文字が指定されていなければなりません。
------------	--

表 E-5 Fortran 95 における OpenMP 指令の要約 (続き)

指令の形式 (自由)	!\$OMP <i>directive optional_clauses...</i>
	<p>任意の場所で指定でき、その前に空白があってもかまいません。継続行は、アンパースンドにより識別されます (!\$OMP&amp;)。</p>
条件付きコンパイル	<p>1 桁目と 2 桁目が !\$、C\$, *\$ のいずれかで始まるソース行 (固定書式)、または前に空白が指定されたソース行 (自由書式) は、コンパイラオプション <code>-openmp</code> または <code>-mp=openmp</code> が指定されている場合にのみコンパイルされます。</p>
PARALLEL 指令	<p>!\$OMP PARALLEL [<i>clause</i>[:,] <i>clause</i>]...            ブロックに対する転送のない Fortran 文のブロック            !\$OMP END PARALLEL</p> <p>並列化の範囲を定義します。この範囲は、複数のスレッドにより並列で実行されるコードのブロックです。<i>clause</i> には PRIVATE(<i>list</i>), SHARED(<i>list</i>), DEFAULT(<i>option</i>), FIRSTPRIVATE(<i>list</i>), REDUCTION(<i>list</i>), IF(<i>expression</i>), COPYIN(<i>list</i>) のいずれかを指定できます。</p>
DO 指令	<p>!\$OMP DO [<i>clause</i>[:,] <i>clause</i>]...  <i>do_loop</i> 文のブロック            [!\$OMP END DO [NOWAIT]]</p> <p>DO 指令は、直後の DO ループの反復を並列で実行するように指定します。この指令は、並列化範囲内に指定しなければなりません。<i>clause</i> には PRIVATE(<i>list</i>), FIRSTPRIVATE(<i>list</i>), LASTPRIVATE(<i>list</i>), REDUCTION(<i>list</i>), SCHEDULE(<i>type</i>), ORDERED のいずれかを指定できます。</p>

表 E-5 Fortran 95 における OpenMP 指令の要約 (続き)

---

SECTIONS 指令	<pre>!\$OMP SECTIONS [<i>clause</i>[:,] <i>clause</i>]... [!\$OMP SECTION] 転送のない文のブロック [!\$OMP SECTION 文のオプションブロック ] ... !\$OMP END SECTIONS [NOWAIT]</pre>
	<p>チーム内のスレッド間で分割されるコードの非反復セクションを囲みます。各セクションは、チーム内のスレッドにより 1 度実行されます。 <i>clause</i> には PRIVATE(<i>list</i>), FIRSTPRIVATE(<i>list</i>), LASTPRIVATE(<i>list</i>), REDUCTION(<i>list</i>) のいずれかを指定できます。</p> <p>各セクションの前には SECTION 指令があり、これは最初のセクションで選択できます。</p>
<hr/>	
SINGLE 指令	<pre>!\$OMP SINGLE [<i>clause</i>[:,] <i>clause</i>]... 転送のない文のブロック !\$OMP END SINGLE [NOWAIT]</pre>
	<p>SINGLE 指令により囲まれた文はチーム内の 1 スレッドによってのみ実行されます。文の SINGLE ブロックを実行しないチームのスレッドは、NOWAIT が指定されない限り、END SINGLE 指令で待機しています。 <i>clause</i> には、PRIVATE(<i>list</i>) または FIRSTPRIVATE(<i>list</i>) のいずれかを指定できます。</p>
<hr/>	
PARALLEL DO 指令	<pre>!\$OMP PARALLEL DO [<i>clause</i>[:,] <i>clause</i>]... <i>do_loop</i> 文のブロック [!\$OMP END PARALLEL DO [NOWAIT]]</pre>
	<p>1つの DO ループに含まれている並列領域を指定する簡単な方法です。PARALLEL 指令のすぐ後に DO 指令を指定します。 <i>clause</i> には、PARALLEL 指令や DO 指令で指定可能な句のいずれかを使用できます。</p>

---

表 E-5 Fortran 95 における OpenMP 指令の要約 (続き)

---

PARALLEL SECTIONS 指令	<pre>!\$OMP PARALLEL SECTIONS [<i>clause</i>[,] <i>clause</i>...] [!\$OMP SECTION] ブロックに対する転送のない文のブロック [!\$OMP SECTION 文のオプションブロック ] ... !\$OMP END PARALLEL SECTIONS [NOWAIT]</pre>
-------------------------	--

1 つの SECTIONS 指令に含まれている並列領域を指定する簡単な方法です。PARALLEL 指令のすぐ後に SECTIONS 指令を指定します。clause には、PARALLEL 指令や SECTIONS 指令で指定可能な句のいずれかを使用できます。

---

同期指令

MASTER 指令	<pre>!\$OMP MASTER ブロックに対する転送のない文のブロック !\$OMP END MASTER</pre>
-----------	--

これらの指令で囲まれている文のブロックは、チームのマスタースレッドだけが実行します。その他のスレッドは、このブロックをスキップして、処理を継続します。マスターセクションの開始または終了では、暗黙的な境界はありません。

CRITICAL 指令	<pre>!\$OMP CRITICAL [(<i>name</i>)] ブロックに対する転送のない文のブロック !\$OMP END CRITICAL [(<i>name</i>)]</pre>
-------------	--

これらの指令で囲まれている文のブロックへのアクセスを、一度に 1 スレッドだけに制限します。オプション *name* の引数には識別のための領域名を指定します。指定のない CRITICAL 指令では、すべて同じ名前にマッピングされます。クリティカルなセクション名とは、プログラムの大域要素を指します。名前がほかの要素と競合する場合は、プログラムの動作が定まらなくなります。CRITICAL 指令で名前を指定した場合は、END CRITICAL 指令にもその名前を指定しなければなりません。

---

表 E-5 Fortran 95 における OpenMP 指令の要約 (続き)

BARRIER 指令	<p>!\$OMP BARRIER</p> <p>チーム内のすべてのスレッドの同期をとります。各スレッドは、チーム内のその他すべてのスレッドがこの点に到達するまで待機します。</p>
ATOMIC 指令	<p>!\$OMP ATOMIC</p> <p>特定のメモリ位置を自動的に更新するようにし、複数の同時書き込みスレッドが生じないようにします。</p> <p>指令は、そのすぐ後の文にのみ適用されます。その文は、次の形式のいずれかでなければなりません。</p> <p><math>x = x \text{ operator expression}</math>  <math>x = \text{expression operator } x</math>  <math>x = \text{intrinsic}(x, \text{expression})</math>  <math>x = \text{intrinsic}(\text{expression}, x)</math></p> <p>ただし、</p> <ul style="list-style-type: none"> <li>• <math>x</math> は、intrinsic 型のスカラーです。</li> <li>• <math>\text{expression}</math> は、<math>x</math> を参照しないスカラー式です。</li> <li>• <math>\text{intrinsic}</math> は、MAX、MIN、IAND、IOR、IEOR のいずれかです。</li> <li>• <math>\text{operator}</math> は、+、-、*、/、.AND.、.OR.、.EQV.、.NEQV. のいずれかです。</li> </ul>
FLUSH 指令	<p>!\$OMP FLUSH [(list)]</p> <p>スレッド可視変数は、この指令が出現したところでメモリに書き戻されます。これには、大域変数(共通ブロックとモジュール)、(SAVE 属性を指定しないで) 副プログラムに渡されるかまたは副プログラムの並列領域で SHARED として宣言された局所変数、仮引数、そしてすべてのポインタ非参照が含まれます。オプションのリストには、フラッシュしなければならない変数をコンマで区切って指定します。</p>



表 E-5 Fortran 95 における OpenMP 指令の要約 (続き)

ORDERED 指令	<pre>!\$OMP ORDERED ブロックに対する転送のない文のブロック !\$OMP END ORDERED</pre>
<p>この指令により囲まれた文のブロックは、反復がループの実行順に実行されるときの順序で実行されます。これは、<code>DO</code> 指令または <code>PARALLEL DO</code> 指令の動的拡張機能でのみ指定できます。<code>ORDERED</code> 句は、ブロックを囲んでいる一番近い <code>DO</code> 指令で指定しなければなりません。</p>	
データ環境指令	<pre>THREADPRIVATE 指令 !\$OMP THREADPRIVATE (/cb/[<i>cb</i>]/...)</pre>
<p>指定した共通ブロックをスレッドに対して非公開としますが、そのスレッド内では大域となります。<code>cb</code> は共通ブロックの名前です。共通ブロックを <code>THREADPRIVATE</code> にするには、そのブロックのすべての <code>COMMON</code> 宣言の後でこの指令を指定しなければなりません。</p>	
データスコープ (有効範囲) 句	<p>上記のいくつかの指令は、指令により閉じた変数のスコープ属性を制御するための句を受け取ります。データスコープ句が指令に指定されていない場合は、その指令に関係のある変数のデフォルトスコープは <code>SHARED</code> となります。<code>list</code> には、スコープ単位でアクセス可能な指定された変数や共通ブロックのリストをコンマで区切って指定します。共通ブロック名は、スラッシュで囲まなければなりません (<code>/ABLOCK/</code> など)。</p>
PRIVATE 句	<pre>PRIVATE(<i>list</i>)</pre>
<p>チーム内の各スレッドに対して非公開となるように、変数のリストをコンマで区切って宣言します。</p>	
SHARED 句	<pre>SHARED(<i>list</i>)</pre>
<p>チーム内のすべてのスレッドは、<code>list</code> に指定された変数を共有し、同じ記憶領域にアクセスします。</p>	

表 E-5 Fortran 95 における OpenMP 指令の要約 (続き)

DEFAULT 句	<p>DEFAULT (PRIVATE   SHARED   NONE)</p> <p>並列領域内のすべての変数にスコーピング属性を指定します。  <b>THREADPRIVATE</b> 変数は、この句の影響を受けません。この句を指定しない場合は、<b>DEFAULT (SHARED)</b> であるとみなされます。</p>
FIRSTPRIVATE 句	<p>FIRSTPRIVATE(<i>list</i>)</p> <p><i>list</i> の変数は、<b>PRIVATE</b> です。また、変数の非公開コピーは、構成前から存在している元のオブジェクトから初期化されず。</p>
LASTPRIVATE 句	<p>LASTPRIVATE(<i>list</i>)</p> <p><i>list</i> の変数は、<b>PRIVATE</b> です。また、<b>DO</b> 指令で <b>LASTPRIVATE</b> 句が指定されていると、順序が最後の反復を実行するスレッドにより変数のバージョンが構成前に更新されます。<b>SECTIONS</b> 指令では、意味的に最後の <b>SECTION</b> を実行するスレッドにより、それが持っているオブジェクトのバージョンが構成前に更新されます。</p>
REDUCTION 句	<p>REDUCTION({<i>operator</i>   <i>intrinsic</i>): <i>list</i>)</p> <p><i>operator</i> は、 + * - .AND. .OR. .EQV. .NEQV. のいずれかです。  <i>intrinsic</i> は、 MAX MIN IAND IOR IEXOR のいずれかです。  <i>list</i> の変数は、 <i>intrinsic</i> タイプの名前付きスカラー変数でなければなりません。</p> <p><b>REDUCTION</b> 句は、前述の <b>ATOMIC</b> 指令で示したフォームの縮約文でのみ縮約変数が使用される領域で使用することを目的としています。<i>list</i> の変数は、囲んだコンテキストの中で <b>SHARED</b> でなければなりません。各変数の非公開コピーは、それが <b>PRIVATE</b> の場合、スレッドごとに作成されます。縮約の終わりに、各非公開コピーの最終値を元の値と組み合わせ、共有変数が更新されます。</p>

表 E-5 Fortran 95 における OpenMP 指令の要約 (続き)

COPYIN 句	COPYIN( <i>list</i> )	<p>COPYIN 句は、THREADPRIVATE として宣言された共通ブロックに対してのみ適用されます。並列領域で、COPYIN 句はチームのマスタースレッドのデータが並列領域の先頭で共有ブロックのスレッド非公開コピーにコピーされるように指定します。</p>
DO 指令と PARALLEL DO 指令のスケジューリング句		
SCHEDULE 句	SCHEDULE( <i>type</i> [, <i>chunk</i> ])	<p>DO ループの反復がチームのスレッド間でどのように分割されるかを指定します。<i>type</i> には、次のいずれかを指定します。SCHEDULE 句を指定しなければ、STATIC スケジューリングが使用されます。</p>
STATIC スケジューリング	SCHEDULE (STATIC, <i>chunk</i> )	<p>反復は、チャンクにより指定されたサイズに分割されます。分割された反復は、スレッド番号の順序で総当り形式でチームのスレッドに割り当てられます。<i>chunk</i> は、スカラー整数式でなければなりません。</p>
DYNAMIC スケジューリング	SCHEDULE (DYNAMIC, <i>chunk</i> )	<p>反復は、チャンクにより指定されたサイズに分割されます。各スレッドは、反復空間を終了すると、次の反復セットを動的に取得します。</p>
GUIDED スケジューリング	SCHEDULE (GUIDED, <i>chunk</i> )	<p>GUIDED を指定すると、チャンクサイズは実行された各反復セットによりべき乗的に縮約されます。<i>chunk</i> は、その都度実行する反復の最小数を指定します。</p>
RUNTIME スケジューリング	SCHEDULE (RUNTIME)	<p>スケジューリングが、実行時まで遅延されます。スケジューリング型やチャンクサイズは、OMP_SCHEDULE 環境変数の設定値をもとに決定されます。</p>

## OpenMP ライブラリルーチン

OpenMP Fortran API ライブラリルーチンは、外部手続きです。次の要約で、*int\_expr* はデフォルトでスカラー整数式であり、*logical\_expr* はデフォルトのスカラー論理式です。これらのルーチンの戻り値は、デフォルトのようなものです。詳細については、OpenMP 仕様を参照してください。

表 E-6 Fortran 95 OpenMP ライブラリルーチンの要約

---

### 実行環境ルーチン

---

#### OMP\_SET\_NUM\_THREADS サブルーチン

```
SUBROUTINE OMP_SET_NUM_THREADS(int_expr)
```

次の並列領域で使用するためにスレッド数を設定します。

---

#### OMP\_GET\_NUM\_THREADS 関数

```
INTEGER FUNCTION OMP_GET_NUM_THREADS()
```

呼び出し元の並列領域を実行しているチーム内に現在あるスレッドの数を返します。

---

#### OMP\_GET\_MAX\_THREADS 関数

```
INTEGER FUNCTION OMP_GET_NUM_THREADS()
```

OMP\_GET\_NUM\_THREADS 関数への呼び出しで返すことが可能な最大値を返します。

---

#### OMP\_GET\_THREAD\_NUM 関数

```
INTEGER FUNCTION OMP_GET_THREAD_NUM()
```

チーム内のスレッド番号を返します。これは、0 から OMP\_GET\_NUM\_THREADS() - 1 までの数値です。マスタースレッドはスレッド 0 です。

---

#### OMP\_GET\_NUM\_PROCS 関数

```
INTEGER FUNCTION OMP_GET_NUM_PROCS()
```

プログラムで使用可能なプロセッサの数を返します。

---

#### OMP\_IN\_PARALLEL 関数

表 E-6 Fortran 95 OpenMP ライブラリルーチンの要約

---

LOGICAL FUNCTION OMP\_IN\_PARALLEL()

並列で実行している領域の動的拡張内から呼び出された場合、`.TRUE.` を返します。それ以外の場合は、`.FALSE.` を返します。

---

OMP\_SET\_DYNAMIC サブルーチン

SUBROUTINE OMP\_SET\_DYNAMIC(*logical\_expr*)

プログラムの並列実行で使用可能なスレッド数の動的調整を有効または無効にします。

---

OMP\_GET\_DYNAMIC 関数

LOGICAL FUNCTION OMP\_GET\_DYNAMIC()

動的スレッド調整が有効な場合、`.TRUE.` を返します。それ以外の場合は、`.FALSE.` を返します。

---

OMP\_SET\_NESTED サブルーチン

SUBROUTINE OMP\_SET\_NESTED(*logical\_expr*)

ネストされた並列化機能を有効または無効にします。

---

OMP\_GET\_NESTED 関数

FUNCTION OMP\_GET\_NESTED()

ネストされた並列化機能が有効な場合、`.TRUE.` を返します。それ以外の場合は、`.FALSE.` を返します。

---

ロックルーチン

ロック変数 *var* は、次のルーチンからしかアクセスできません。*var* は、integer 型で、またアドレスを保持できる大きさの `KIND` でなければなりません。たとえば、64 ビットシステムで *var* は `INTEGER (KIND=8)` として宣言できます。

---

OMP\_INIT\_LOCK サブルーチン

SUBROUTINE OMP\_INIT\_LOCK(*var*)

以降の呼び出しで使用するために、ロック変数 *var* に関連付けられているロックを初期化します。初期の状態ではロックは解除されています。

---

OMP\_DESTROY\_LOCK サブルーチン

SUBROUTINE OMP\_DESTROY\_LOCK(*var*)

指定したロック変数 *var* が関連付けられているロックがあれば、それから *var* との関連付けを解除します。

---

OMP\_SET\_LOCK サブルーチン

---

表 E-6 Fortran 95 OpenMP ライブラリルーチンの要約

---

<pre>SUBROUTINE OMP_SET_LOCK(var)</pre>
指定したロックが使用可能な状態になるまで、実行スレッドを待機させます。ロックが使用可能になると、スレッドにロックの所有権が与えられます。

---

<b>OMP_UNSET_LOCK サブルーチン</b>
<pre>SUBROUTINE OMP_UNSET_LOCK(var)</pre>
実行スレッドから、ロックの所有権を解放します。スレッドがそのロックを所有していない場合は、動作は不確定となります。

---

<b>OMP_TEST_LOCK 関数</b>
<pre>LOGICAL FUNCTION OMP_TEST_LOCK(var)</pre>
ロック変数 <i>var</i> に関連付けられているロックを設定しようと試みます。ロックが正常に設定されると、 <code>.TRUE.</code> が返されます。それ以外の場合は、 <code>.FALSE.</code> が返されます。

---

## OpenMP 環境変数

次の表は、OpenMP プログラムの実行を制御する OpenMP Fortran API 環境変数を要約したものです。

表 E-7 OpenMP Fortran 環境変数の要約

---

<b>OMP_SCHEDULE</b>
スケジュール型が <code>RUNTIME</code> として指定された <code>DO</code> 指令や <code>PARALLEL DO</code> 指令のスケジュール型を設定します。たとえば、 <code>setenv OMP_SCHEDULE "GUIDED,4"</code> とします。指定しない場合、 <code>STATIC</code> のデフォルト値が使用されます。

---

<b>OMP_NUM_THREADS</b>
<code>OMP_SET_NUM_THREADS()</code> サブルーチンの呼び出しにより設定されていない場合に、実行時に使用するスレッド数を設定します。たとえば、 <code>setenv OMP_NUM_THREADS 16</code> とします。設定しない場合、デフォルト値 1 が使用されます。

---

<b>OMP_DYNAMIC</b>
並列領域の実行で使用可能なスレッド数の動的調整機能を有効または無効にします。たとえば、 <code>setenv OMP_DYNAMIC FALSE</code> とします。設定しない場合、デフォルト値 <code>TRUE</code> が使用されます。

---

<b>OMP_NESTED</b>
ネストされた並列機能を有効または無効にします。たとえば、 <code>setenv OMP_NESTED TRUE</code> とします。

---

# 索引

---

## 記号

Δ, xv

!DIR\$ 指令, 166

## 数字

1 カラム目の D (VMS Fortran), 108

16 進, 155

8 進, 155

## A

-a による基本ブロックのプロファイル, 40

abrupt\_underflow, 58

asa、Fortran 印刷ユーティリティ, 3

## C

CSPAR DOALL 指令, 23

CSPAR DOSERIAL 指令, 23

CSPRAGMA C(.) 指令, 19

CSPRAGMA SUN UNROLL, 20

CSPRAGMA WEAK 指令, 20

C(...) 指令, 19

CALL

ENTRY 文の引数の保存, 42

ループを並列化, 88

CDIR\$ 指令, 166

CDIR\$FREE 指令, 167

CIF、コンパイラ情報ファイル, 48

COMMON

consistency checking with

-xcommoncheck, 103

パディング, 79

COMMON

境界整列, 41

cpp、C プリプロセッサ, 14, 46, 54

cpp、-Dname のシンボルの定義, 46

Cray

ポインタ, 158

ポインタと Fortran 90 ポインタ, 159

ポインタと最適化, 162

## D

dbx

f77、-g, 63

初期化の高速実行, 119

DO ループの 1 回実行, 78

DOALL 指令, 23

DOSERIAL 指令, 23

## E

error、エラーメッセージの表示, 4

## F

f77、f90 コマンド行, 12, 31

FFLAGS 環境変数, 25

### Fortran

f77 と f90 の混在, 16

SunSoft の特徴と拡張機能, 2

プリプロセッサ, 47

-F による起動, 54

ユーティリティ, 3

### Fortran 90

大文字と小文字の区別, 153

サンの機能, 151

モジュール, 172

fpp、Fortran プリプロセッサ, 14, 47, 54

fpversion、浮動小数点バージョンの表示, 25

fsplit、Fortran ユーティリティ, 3

## G

### gprof

-pg、手続きごとのプロファイル, 81

## I

### CALL

-inline で副プログラムの引用をインライン化する, 66

## L

### libm

デフォルトによる検索, 68

### limit

コマンド, 28

スタックサイズ, 89

### loop

parallelizing a CALL in a loop, 88

looptool、MP のループプロファイラ, 123

## N

nonstandard\_arithmetic(), 58

## O

### OpenMP

指令の要約, 184

ライブラリルーチン、要約, 192

### options

-inline, 66

OPTIONS 環境変数, 25

## P

### parallelization

OpenMP directives summarized, 184

prof、-p, 79

## R

README ファイル, 6, 105

-reduction、縮約して自動的に並列化処理を行う, 86

## S

SIGFPE、浮動小数点例外, 58

STOP 文、ステータスの返し, 90

SWAP コマンド, 27

## T

### tcov

-a、文ごとのプロファイル, 40

-xprofile による新しい形式, 116

.T ファイル, 48

## U

ulimit コマンド, 28



UNROLL 指令, 20

## V

VAX VMS Fortran

-vax による機能, 92

-xl による機能, 108

## X

xild、インクリメンタルリンカー, 106

## あ

アセンブリコード, 87

アナライザコンパイルオプション、-xF, 105

アプリケーションレジスタ (SPARC), 118

アンダーフロー

ゼロへの強制指定, 59

暗黙のオプション, 40

## い

一時ファイル、ディレクトリ, 91

位置独立コード, 82, 83

インクルードファイル, 65

印刷

asa, 3

インストール, 6

インストールパス, 65

インライン, 56

テンプレート、-libmil, 69

インラインテンプレート, 69

## え

エラーメッセージ

-erroff による抑制, 52

error, 4

f90, 130

入出力, 126

メッセージタグ, 52

## お

大型のファイル, 26

オーバーフロー, 62

スタック, 89

大文字と小文字の区別, 91

オブジェクトライブラリの検索ディレクトリ, 68

オブジェクトファイル

コンパイルのみ, 45

名前, 78

オプション

-a, 40

-ansi 拡張機能, 42

-arg=local、ENTRY 引数の保存, 42

-autopar、自動並列化, 42

-Bdynamic, 43

-Bstatic, 44

-cg89, 45

-cg92, 45

-copyargs、定数の引数への代入, 45

-c、コンパイルのみ, 45

-C、引数の検査, 44

-dalign

-fast による, 56

-dalign, 47

-dbl

-xtypemap, 49

倍長のデフォルトデータサイズ, 48

-dbl\_align\_all、強制的なデータの整列, 50

-depend

-fast による, 56

データ依存の解析, 50

-dn, 51

-Dname、事前定義されたシンボル, 46

-dryrun, 50

-dy, 51

-erroff、警告の抑制, 52

-errtags、警告によるメッセージタグの表示, 52

-explicitpar、明示的並列化, 52

- ext\_names、下線なしの外部名, 53
- e、拡張されたソース行, 51
- F, 54
- fast
  - 高速実行, 55
- fixed, 57
- flags, 57
- fnonstd, 57
- fns, 58
- free, 60
- fround=*r*, 60
- fsimple
  - 浮動小数点モデルの単純化, 61
- ftrap, 62
- f、8 バイトでの境界整列, 54
- G, 63
- g, 63
- help, 64
- hname, 64
- i2、短い整数用, 65
- i4, 66
- I*dir*, 64
- KPIC, 67
- Kpic, 67
- L*dir*, 68
- libmil
  - fast による, 56
- libmil, 69
- llibrary, 68
- loopinfo、並列化の表示, 69
- M*dir*、f90 モジュール, 70
- misalign, 71
- mp=cray、Cray MP 指令, 71
- mp=sun、Sun MP 指令, 71
- mt、マルチスレッド環境で安全なライブラリ, 72
- native, 73
- noautopar, 73
- nodepend, 73
- noexplicitpar, 73
- nolib, 74
- nolibmil, 74
- noqueue, 75
- noreduction, 75
- norunpath, 75
- O
  - g による, 76
  - O, 76
  - O4, 77
  - O5, 77
  - oldldo, 78
  - onetrip, 78
  - o、出力ファイル, 77
  - pad=*p*, 79
  - parallel、ループの並列化, 80
  - pg、手続きのプロファイル, 81
  - PIC, 83
  - pic, 82
  - p、手続きごとのプロファイル, 79
  - Qoption, 84
  - R *ls*, 84
  - r8, 85
  - reduction, 86
  - S, 87
  - s, 87
  - sbfast, 88
  - sb、ソースブラウザ, 87
  - silent, 88
  - stackvar, 88
  - stop\_status=*yn*, 90
  - temp, 90
  - time, 91
  - u, 91
  - unroll、ループの展開, 91
  - U、小文字に変換しない, 91
  - V, 92
  - v, 92
  - vax=*v*, 92, 107
  - vpara, 93
  - w, 94
  - xa, 94
  - xarch=*a*, 95
  - xautopar, 100
  - xcache=*c*, 100
  - xcg[89|92], 101
  - xchip=*c*, 101
  - xcrossfile, 104
  - xdepend, 104
  - xexplicitpar, 105
  - xF, 105
  - xhelp=*h*, 105
  - xildoff, 106

-xildon, 106  
-xinline, 107  
-xld, 107  
-xlibmil, 108  
-xlibmopt  
  -fast による, 56  
-xlibmopt, 108  
-xlicinfo, 109  
-xlic\_lib=libs, 109  
-Xlist  
  サブオプション, 110  
-Xlist、プログラムの大域的検査, 109  
-xloopinfo, 111  
-xnolib, 113  
-xnolibmil, 113  
-xnolibmopt, 113  
-xOn, 113  
-xparallel, 114  
-xpg, 114  
-xpp=p, 114  
-xprofile=p, 115  
-xreduction, 117  
-xregs=r, 117  
-xs, 118  
-xsafe=men, 119  
-xsb, 119  
-xsbfast, 119  
-xspace, 119  
-xtarget, 173  
-xtarget=t, 120  
-xtime, 121  
-xtypemap  
  -dbl, 49  
-xtypemap=spec, 121  
-xunroll=n, 122  
-xvpara, 123  
-Zlp、ループプロファイラ、MP, 123  
-ztext, 123  
暗黙の, 40  
オプションフラグのリファレンス, 124  
機能別の分類, 33  
旧, 39  
現在使用されていないオプション, 39  
コマンド行の構文, 32  
コンパイル段階への引き渡し, 84  
コンパイルとリンクの整合性, 15

処理の順序, 33  
認識されない, 16  
頻繁に利用する, 38  
リストの表示、-help, 64  
オプションのリスト, 64

## か

下位互換オプション, 39

### 拡張機能

ANSI 規格以外、-ansi フラグ, 42  
-xl による VAX VMS Fortran の機能, 108

### 拡張子

コンパイラが認識するファイル名, 13  
コンパイラ (f90) が認識するファイル名, 153  
コンパイラが認識するファイル名の, 13

各リリースにおける機能変更, 139

### 下線, 53

外部名につけない, 19

型宣言の別の書式, 157

### 環境

STOP によるプログラムの終了, 90

### 環境変数

使用, 25

関数レベルの並べ替え, 105

外部名, 53

## き

### 規格

準拠, 1

### 機能

Fortran 90, 151  
各リリースにおける機能変更, 139

### キャッシュ

ハードウェアキャッシュの指定, 100  
パディング, 79

### 境界整列

VMS Fortran における構造体, 71, 108  
データを参照してください

共有ライブラリ

位置独立コード, 82  
共有ライブラリの名前, 64  
構築、-G, 63  
純粋な、再配置なし, 124  
大域オフセットテーブル, 82  
リンクの使用不可、-dn, 51  
局所変数  
メモリースタックへの割り当て, 88  
境界整列  
-dalign, 47

く

クイックスタート, 10  
空白文字, xv  
組み込み手続き、拡張, 163  
クロスリファレンステーブル、-Xlist, 110

け

警告  
ANSI 規格以外の拡張機能, 42  
-erroff による抑制, 52  
未宣言の変数, 91  
メッセージタグ, 52  
メッセージの抑制, 94  
継続行, 51, 151  
検索  
オブジェクトライブラリディレクトリ, 68  
モジュール, 71  
現在使用されていないオプション, 39

こ

構文  
f77、f90 コマンド, 12, 31  
コンパイラコマンド行, 31  
コンパイラコマンド行のオプション, 32  
コードのサイズ, 119  
互換性  
C, 172

Fortran 90 と Fortran 77, 168  
コンパイラリリース間の, 147  
コピー ストア, 42  
コマンド行  
認識されないオプション, 16  
コマンド行ヘルプ, 8  
コメント  
指令として, 166  
小文字、変換なし, 91  
コンパイラ  
経過時間, 91  
コマンド行, 12  
詳細情報, 92  
指令, 17  
ドライバ、-dryrun によるコマンドの表示, 50  
バージョンの表示, 92  
頻繁に利用するオプション, 33  
コンパイル済みコードのサイズ, 119  
コンパイルとリンク, 12, 14  
-B, 44  
コンパイルのみ, 45  
整合性, 15  
動的共有ライブラリ, 51  
動的共有ライブラリの構築, 63  
抑制, 45  
コンパイルのパス, 92

さ

最適化  
Cray ポインタ, 162  
-fast による, 55  
キャッシュの指定, 100  
指令によるループの展開, 20  
数学ライブラリ, 108  
ソースファイル, 104  
ターゲットハードウェア, 73, 120  
デバッグによる, 63  
浮動小数点, 61  
プロセッサの指定, 102  
ユーザー作成ルーチンのインライン化, 66

- ループの展開, 92
- ループのプロファイル, 92
- レベル, 76

最適化のメモリー不足, 26

算術、浮動小数点を参照してください, 58

## し

- シエル
  - 制限, 28
- シグナルハンドラ, 126
- 指示先, 158
- 実行時間
  - 最適化, 76
- 自動変数, 88
- 処理の順序、オプション, 33
- 使用
  - コンパイラ, 24
- 指令
  - FORTRAN 77, 17
  - Fortran 90, 166
  - OpenMP (Fortran 95), 184
  - 全指令の要約, 179
  - 並列化 (f77), 22
  - 並列化 (f90), 168
  - 並列化、Cray または Sun, 71
  - 弱いシンボルのリンク, 20
  - ループの展開, 20
- 指令のリスト, 179
- シンボルテーブル
  - dbx, 63, 118
- 実行可能ファイル
  - シンボルテーブルの取り除き, 87
  - 動的ライブラリのパスの埋め込み, 84
  - 名前, 78
- 実行可能ファイルからのシンボルテーブルの取り除き、-s, 87
- 自動インライン化
  - O4 による, 77
- 従来のコンパイラオプション, 39
- 順序
  - 関数, 105

- 自由書式のソース, 60
- 情報ファイル, 6

## す

- 数学ライブラリ
  - Ldir オプション, 68
  - 最適化バージョン, 108
- スタック
  - オーバーフロー, 89
  - サイズの増大, 90
- スレッド安全ライブラリ, 72
- スワップ領域
  - 拡大, 27
  - 実際のスワップ領域の表示, 27
  - ディスクのスワップ領域の制限, 26

## せ

- 整数、4 バイトと 8 バイト, 66
- 静的
  - リンク, 51
- 境界整列
  - COMMON ブロック内のデータ
    - aligncommon, 41
- 設定
  - #include パス, 64

## そ

- 相違点
  - Sun Fortran 90, 151
- 添字の範囲, 44
- ソース行
  - 大文字と小文字の保持, 91
  - 拡張, 51
  - 行の長さ, 151
  - 自由書式, 60
  - プリプロセッサ, 114
- ソースの書式
  - オプション (f90), 153

拡張子 (f90), 153  
指令 (f90), 153  
ソース行の書式の混在 (f90), 153  
ソースファイル  
  プリプロセス, 14  
ソースブラウザ, 87

## た

大域オフセットテーブル, 82  
大域シンボル  
  弱い, 20  
大域プログラムの検査、-Xlist, 109  
タブフォーマット  
  フリーフォーマット, xv

## ち

注釈  
  デバッグ、VMS, 108

## て

定数の引数、-copyargs, 46  
ディレクトリ  
  一時ファイル, 91  
データ  
  -dbl\_align\_all による境界整列, 50  
  -f による境界整列, 54  
  REAL を DOUBLE PRECISION として解  
  積, 85  
  -xtypemap のマッピング, 121  
  境界整列に失敗したデータの受付、  
  -misalign, 71  
  デフォルトサイズと -dbl, 49  
  デフォルトサイズと -r8, 85  
データ依存  
  -depend, 50  
デバッグ  
  -C による配列添字のチェック, 45  
  -dryrun によるコマンドの表示, 50

-g オプション, 63  
-Xlist, 4  
-Xlist によるプログラムの大域的検査, 110  
オブジェクトファイルなし, 118  
クロスリファレンステーブル, 110  
最適化, 63  
ユーティリティ, 4  
デフォルト  
  インクルードファイルのパス, 65

## と

特徴と拡張機能, 2  
トラップ  
  浮動小数点例外, 62  
  メモリー, 119  
動的  
  ライブラリ  
  構築、-G, 63  
  動的ライブラリの名前, 64  
  リンク, 51

## な

名前  
  オブジェクト、実行可能ファイル, 78  
  引数、下線を付けない, 19

## に

入出力  
  f77 と f90 の互換性, 170  
  エラーメッセージ, 126  
認識されないオプション, 16

## は

ハードウェア  
  浮動小数点の非標準の初期化, 57  
ハードウェアアーキテクチャ, 102, 120  
配列範囲のチェック, 44

パフォーマンス  
  Sun Performance Library, 4  
  最適化, 55  
バージョン  
  各コンパイラパスの ID, 92  
パス  
  #include, 64  
  実行可能ファイル中の動的ライブラリ, 84  
  標準インクルードファイルへの, 65  
  モジュールの検索, 71  
  ライブラリの検索, 68  
パディング, 79

## ひ

非自動読み込み, dbx, 118  
標準  
  ANSI 規格以外の識別, -ansi フラグ, 42  
  インクルードファイル, 65

## ふ

ファイル  
  .M、モジュールファイルを参照してください  
  オブジェクト, 12  
  サイズが大きすぎる場合, 26  
  実行可能, 12  
ファイル名  
  コンパイラ (f90) で認識される, 153  
  コンパイラで認識される, 13  
浮動小数点  
  fpversion、ハードウェアの表示, 25  
  『数値計算ガイド』を参照してください  
  設定、-fsimple, 61  
  トラップモード, 62  
  非標準, 58  
  丸め, 60  
プロファイル  
  MP のループごとの、-Zlp、ループツ  
    ル, 123  
  基本ブロックごとの, 40  
  手続きごとの、-pg、gprof, 81

プロファイル、-xprofile, 115  
文  
  -a および -tcov によるプロファイル, 40  
ブール型  
  型、定数, 154  
  定数の別の書式, 154  
ブラウザ, 87  
プラグマ、指令を参照してください  
プリプロセッサ、ソースファイル  
  fpp、cpp, 14  
  シンボルの定義, 46  
プロセッサ  
  ターゲットプロセッサの指定, 102, 120

## へ

並列化  
  Cray 指令と Sun 指令の設定, 71  
  『Fortran プログラマーズガイド』を参照して  
    ください  
  -stackvar, 88  
  縮約演算, 86  
  指令 (f77), 22  
  自動, 42  
  自動と明示的、-parallel, 80  
  スレッド環境で安全なライブラリ, 72  
  明示的, 53  
  メッセージ, 93  
  ループ情報, 69  
ヘルプ  
  README 情報, 105  
ヘルプ、コマンド行, 8  
変換  
  ファイル名の拡張子, 13  
変数  
  未宣言の, 91

## ほ

ポインタ, 158  
ホレリス, 156

## ま

- マニュアルページ, 5
- マルチスレッド
  - 並列化を参照してください, 22
- 丸め, 60, 62

## む

- 無効な浮動小数点, 62

## め

- 明示的
  - 型指定, 91
- 明示的な並列化指令, 22
- メッセージ
  - silent による抑制, 88
  - シグナルハンドラ, 126
  - 詳細情報, 92
  - 実行時, 125
  - 入出力エラー, 126
  - 並列化, 69
- メモリー
  - 仮想メモリーの制限, 28
  - 最適化のメモリー不足, 26
  - 実際の実メモリー、表示, 27

## も

- 文字定数中でバックスラッシュ, 93, 108
- モジュール
  - 検索パス, 70
  - 作成と使用, 17

## ゆ

- ユーティリティ, 4

## よ

- 抑制
  - dbx の自動読み込み, 118
  - 暗黙の型, 91
  - 大文字から小文字への変換, 91
  - 書き込み先の並びを空白文字で区切る, 78
  - 警告, 94
  - タグ名による警告、-erroff, 52
  - ライセンスキュー, 75
  - リンク, 45
- 弱いリンカーシンボル, 20

## ら

- ライセンス情報, 109
- ライブラリ
  - l によるリンク, 68
  - Sun Performance Library, 4
  - 位置独立コードと純粋な, 124
  - 共有ライブラリの名前, 64
  - 構築、-G, 63
  - システムライブラリの使用不可, 74
  - 実行可能ファイル中の共有ライブラリのパス, 75
  - 実行可能ファイルの動的検索パス, 84
  - スレッドに対して安全, 72
  - ライセンスによって使用できる, 109

## り

- リンク
  - FORTRAN 77 と Fortran 90 コンパイルの混在, 16
  - l によるライブラリの指定, 68
  - WEAK 指令, 20
  - インクリメンタルリンカーの使用不可, 106
  - インクリメンタルリンカーを使用可能にする, 106
  - コンパイル, 12
  - コンパイルからの分離, 14
  - コンパイルとの整合性, 15



- システムライブラリの使用不可, 74
- 自動並列化、-autopar, 43
- 静的, 44
- 動的, 43, 51
- 動的リンク、共有ライブラリを使用可能にする, 51
- ライセンスによって使用できるライブラリ, 109
- リンカー -Mmapfile オプション, 105

## る

### ループ

- 1 回実行、-onetrip, 78
- unroll による展開, 92
- 依存解析、-depend, 50
- プロファイル, 123
- 並列化メッセージ, 69
- 明示的並列化, 52
- ループ中の CALL の並列化, 88
- 自動並列化, 42

ループの展開

- 指令, 19

## れ

- 例外、浮動小数点, 62
- トラップ, 62
- レジスタの使用, 117

