インテル® oneAPI ポーティング・ガイド

ifx へ移行する ifort ユーザー向け

Intel Corporation
www.intel.com (英語)
著作権と商標について

注意事項:

インテル® Fortran コンパイラー・クラシック (ifort) の開発は終了しました。インテル® oneAPI 2024.2.0 リリースに含まれる ifort 2021.13.0 は、ifort の計画されている最後のバージョンであり、インテル® oneAPI 2025.0.0 で削除される予定です。
インテルでは、ifort のすべての言語機能、レガシー機能、および Fortran 拡張機能をサポートするインテル® Fortran コンパイラー (ifx) への移行を推奨しています。今後、ifx はこれらを継承しながら、同時に新しい Fortran 標準サポートと OpenMP* 標準サポートを実装していく予定です。詳しくは、「インテル® Fortran コンパイラー・クラシック (ifort) の廃止」と「A Historic Moment for The Intel® Fortran Compiler Classic (ifort)」(英語) をご覧ください。

本ドキュメントはレイアウト調整および校閲を行っておりません。誤字脱字、製品名や用語の表記、レイアウト等の不具合が含まれる可能性があることを予めご了承ください。

この日本語マニュアルは、インテル コーポレーションのウェブサイトで公開されている『Porting Guide for Intel® Fortran Compiler』(**更新日 2024/5/17**) を iSUS 編集長が翻訳した参考訳です。原文は更新される可能性があります。原文と翻訳文の内容が異なる場合は原文を優先してください。

目次

著作権と商標について

インテルのテクノロジーを使用するには、対応したハードウェア、特定のソフトウェア、またはサービスの有効化が必要となる場合があります。

絶対的なセキュリティーを提供できるコンピューター・システムはありません。

実際の結果は異なる場合があります。

* その他の社名、製品名などは、一般に各社の表示、商標または登録商標です。

最適化に関する注意事項
インテル® コンパイラーでは、インテル® マイクロプロセッサーに限定されない最適化に関して、他社製マイクロプロセッサー用に同等の最適化を行えないことがあります。これには、インテル® ストリーミング SIMD 拡張命令 2、インテル® ストリーミング SIMD 拡張命令 3、インテル® ストリーミング SIMD 拡張命令 3 補足命令などの最適化が該当します。インテルは、他社製マイクロプロセッサーに関して、いかなる最適化の利用、機能、または効果も保証いたしません。本製品のマイクロプロセッサー依存の最適化は、インテル® マイクロプロセッサーでの使用を前提としています。インテル® マイクロアーキテクチャーに限定されない最適化のなかにも、インテル® マイクロプロセッサー用のものがあります。この注意事項で言及した命令セットの詳細については、該当する製品のユーザー・リファレンス・ガイドを参照してください。 注意事項の改訂 #20110804

製品とパフォーマンス情報

[1] パフォーマンスは、利用、構成、およびその他の要因によって異なります。詳細については、www.Intel.com/PerformanceIndex（英語）をご覧ください。

1 はじめに

このポーティング・ガイドは、新しい LLVM ベースのインテル® Fortran コンパイラー (ifx) に移行するインテル® Fortran コンパイラー・クラシック (ifort) ユーザー向けに情報と提案を提供します。**このガイドは、**インテル® oneAPI DPC++/C++ コンパイラーへ移行するインテル® C++ コンパイラー・クラシック・ユーザー向けポーティング・ガイドと共通の内容が含まれます。

> このポーティング・ガイドは、今後も更新されます。2–3 か月単位で更新されるので確認をお願いします。

改訂履歴	
2024年5月17日	• コンパイラー・オプション `-fp-model=source` (`/fp:source`) の更新
2024年4月2日	• コンパイラー・オプション `fstrict-overflow` と `parallel`
2024年2月5日	• ifx を使用したインテル® Fortran ライブラリーの静的および動的リンク
2024年1月18日	• コンパイラー・オプション `fp-model (fp)` の機能を明確にしました
2023年12月5日	• `fimf` ファミリーのコンパイラー・オプションが更新されました
2023年4月13日	• 最初に使用する際のコンパイラー・オプションに関するヒント
2023年2月06日	• 機能、コンパイラー・オプション、既知の問題の更新
2022年10月03日	• oneAPI 2022.3 リリース、ハウスキーピング処理の更新
2022年6月23日	• ブルータスまたはバイセクション最適化のサポート
2022年5月24日	• `ifx` を使用した `dpcpp` オブジェクト・ファイルのリンク

2 インテル® Fortran コンパイラーに対する指針

インテル® oneAPI 製品パッケージには、2 つの Fortran コンパイラーが含まれます。インテル® Fortran コンパイラー・クラシック (ifort) は、業界最高の Fortran 言語機能と CPU パフォーマンスを提供します。新しいインテル® Fortran コンパイラー (ifx) を使用すると、OpenMP* でインテル® GPU にオフロードする機能が利用できます。`ifx` がサポートする OpenMP* 5.0 および 5.1 の GPU オフロード機能は、`ifort` では利用できません。`ifort` は、GPU オフロードの機能を必要としない開発者向けの業界最高の Fortran コンパイラーであり続けます。Microsoft* Visual Studio* 環境におけるデフォルト Fortran コンパイラーは `ifort` です。

最新のコンパイラーであるインテル® Fortran コンパイラー（`ifx`）は、CPU および GPU に対応しています。`ifx` は、インテル® Fortran コンパイラー・クラシック（`ifort`）のフロントエンド、およびランタイム・ライブラリーをベースとする、LLVM バックエンド・コンパイラー・テクノロジーを採用しています。2023.0.0 のリリースでは、`ifx` は FORTRAN 77、Fortran 90/95、Fortran 2003、Fortran 2008、および Fortran 2018 の言語標準を完全に実装し、ほとんどの OpenMP* 4.5 および OpenMP* 5.0/5.1 ディレクティブとオフロード機能を実装しています。`ifx` は、バイナリー (.o/.obj) とモジュールファイル (.mod) の互換性があります。`ifort` で生成されたバイナリーとライブラリーは、ifx で生成されたバイナリーおよびライブラリーとリンクでき、一方のコンパイラーで生成された `.mod` ファイルは他方のコンパイラーで使用できます (64 ビット・ターゲットのみ)。どちらのコンパイラーも同じランタイム・ライブラリーを使用します。`ifx` でコンパイルしたアプリケーションのパフォーマンスは、`ifort` でコンパイルしたアプリケーションのパフォーマンスに匹敵する場合も、しない場合もあります。`ifx` のパフォーマンスの改善は、2023年の各リリースで引き続き行われる予定です。

注: インテル® Fortran コンパイラー・クラシック (ifort) は非推奨となり、2024 年後半に廃止される予定です。インテルでは、Windows* および Linux* のサポート、新しい言語サポート、新しい言語機能、および最適化を継続するために、今すぐ LLVM ベースのインテル® Fortran コンパイラー (ifx) に移行することをお勧めします。ifx の詳細については、『インテル® Fortran コンパイラー開発者ガイドおよびリファレンス』(英語) およびこの『ifort ユーザー向け ifx 移植ガイド』を参照してください。

インテル® ツールキットのバージョンとコンパイラーのバージョンのマッピングについては、この表を参照 (英語) してください。

3 コンパイラー・オプションデのフォルトに関する大きな変更

- インテル® Fortran コンパイラー・クラシックのユーザーは、引き続き `ifort` ドライバーを使用できます。
- インテル® Fortran コンパイラーのユーザーが、インテル® Fortran コンパイラー・クラシックの機能や、OpenMP* TARGET ディレクティブによりインテル® GPU へオフロードする追加のコンパイル機能を利用するには、`ifx` ドライバーを使用します。

3.1 IA-32 バイナリーの作成

インテル® Fortran コンパイラー (`ifx`) は、Windows* または Linux* 向けの 64 ビットバイナリーを生成できますが、IA-32 はサポートしていません。また、`-m32` や `/Qm32` もサポートされません。

3.2 インテル® Fortran コンパイラーを最初に使用する際のコンパイラー・オプション

安全なオプションを検討してください。最適化されたコードのテストを行う前に、次のオプションを使用してアプリケーションの正当性を検証します。

	Linux*	Windows*
最適化を無効化	-O0	/O0
コンパイル時の警告を確認	-warn all	/warn:all
ランタイムチェック	-check all	/check:all
クラッシュ時にスタックのトレースバックを出力	-g -traceback	/debug:full /tracdback
デバッグ用のシンボルを生成	-g	/debug:full
デフォルトの OpenMP* SIMD または !dir$ simd を無効化	-qno-openmp-simd -no-simd	/Qopenmp-simd- /simd-
式の Fortran セマンティクスに従う	-standard-semantics	/standard-semantics
浮動小数点精度に precise を使用	-fpmodel=precise	/fp:precise

3.3 浮動小数点の動作

- 浮動小数点数の比較で `ifort` と同じ動作になるようにするには、`ifx` ユーザーは、`assume [no]ieee_compares` や `assume nan_compares` を指定する必要があります。
  - Fortran 2018 標準で要求される、浮動小数点比較向けの IEEE compareSignaling 操作を生成するようにコンパイラーに指示します。デフォルトでは、`ifx` は NaN が浮動小数点比較のオペランドにならないことを前提としており、NaN チェックを行いません。
    - Linux*:
      `-assume [no]ieee_compares`
    - Windows*:
      `/assume:[no]ieee_compares`
- `fp-model | fp` および `fast`
  - `fp-model | fp fast=1` または 2 オプションの浮動小数点比較の動作は、`ifort` と `ifx` で異なります。`ifx` で `fp-model` または `fp fast` オプションを使用する際に `ifort` と同じ動作を期待するには、`ifx` コマンドラインに `assume nan_compares` を指定する必要があります。
    - Linux*:
      `-fp-model fast=1` または `-fp-model fast=2 -assume nan_compares`
    - Windows*:
      `/fp:fast 1` または `/fp:fast 2 /assume:nan_compares`

3.4 CPU のパフォーマンス

最適化	ifx (Linux* 構文)	ifort (Linux* 構文)
最適化を無効にする	`-O0`	`-O0`
スピードの最適化 (コードサイズの増加なし)	`-O1`	`-O1`
スピードの最適化 (デフォルト)	`-O2` `-x<code>` を追加すると、利用可能なすべての最適化とベクトル化が有効になります。 `-xHost` を追加すると、コンパイルするプラットフォームで利用可能な最適化が有効になります。	`-O2`

最適化	ifx (Linux* 構文)	ifort (Linux* 構文)
高レベルのループ最適化	-O3 -x<code> を追加すると、利用可能なすべての最適化とベクトル化が有効になります。 -xHost を追加すると、コンパイルするプラットフォームで利用可能な最適化が有効になります。	-O3
デバッグ用のシンボルを生成	-g	-g
複数ファイルのプロシージャー間の最適化	-ipo	-ipo
プロファイル・ガイドに基づく最適化	-fprofile-generate -fprofile-use	-prof-gen -prof-use
プログラム全体でスピードを最適化	-fast (-ipo -O3 -static -fp-model fast)**	-fast (-ipo -O3 -no-prec-div -static -fp-model=fast=2 -xHost)
OpenMP* のサポート	-fiopenmp または -qopenmp	-qopenmp

** 最終的ではありません。

3.5 ifx の OpenMP* オプションと OpenMP* ディレクティブ

- OpenMP* 4.5/5.0/5.1 の TARGET ディレクティブは、インテル® oneAPI HPC ツールキットに含まれる ifx のみが認識できます。ifx はまた、OpenMP* PARALLEL と SIMD ディレクティブも認識します。
- -fiopenmp と -qopenmp は等価です。
  - OpenMP* parallel と SIMD ディレクティブと節を認識してコンパイルし、インテルの OpenMP* ランタイム・ライブラリーを使用します。
- -fopenmp (非推奨) 将来のリリースで削除されます。
- -fopenmp-targets=spir64
  - OpenMP* 4.5/5.0/5.1 の TARGET ディレクティブを使用してインテル® GPU で実行する場合、このオプションを指定する必要があります。
  - spir64 は、64 ビットの標準ポータブル中間表現です。
- 上記のコンパイラー・オプションを併用して、CPU とインテル® GPU 両方のファットバイナリーを生成します。
  - ifx -fiopenmp -fopenmp-targets=spir64 code.F90
  - ifx -qopenmp -fopenmp-targets=spir64 code.F90

3.6 重要なコンパイラー・オプションの対応付け

- `ifx` では多くの `ifort` コンパイラー・オプションを使用できます。
- すべての `ifort` オプションが `ifx` で許可/実装されるわけではないことに注意してください。
- `ifort` コンパイラーのドキュメントに明記されていないオプションは実装されません。また、今後も実装される予定はありません。`ifx` コンパイラーと `ifort` コンパイラーは大きく異なります。古い内部オプションやドキュメント化されていないオプションは作用せず、対応する `ifx` オプションはありません。ドキュメント化されていない内部オプションに必要と思われる機能がある場合は、想定される動作と `ifx` でそれがサポートされていないことをオンライン・サービス・センター (OSC) (英語) に報告してください。「`Makefile` でこのオプションを使用しており、`ifort` ではこのオプションが使用できた」という理由は正当な理由とは認められません。`ifx` コンパイラーの最適化と動作は、`ifort` コンパイラー・クラシックとは異なります。最初はオプションを指定せずに `ifx` を試してみてください。
- 現時点で `ifx` で実装される予定がない `ifort` コンパイラー・オプションに対して、診断の警告が出力されます。以下に例を示します。

```
command line warning #10148: option '-ip' not supported.
```

- `ifx` のオプション `-qnextgen-diag` を指定すると、`ifx` で使用できない `ifort` コンパイラー・オプションのリストを表示します。
- サポートされていないオプションがアプリケーションにとって重要であり、そのオプションを `ifx` のビルドオプションから除くとエラーになる場合は、オンライン・サービス・センター (OSC) (英語) に問題として報告してください。`ifx` に実装する必要があるオプションについて、皆様のご意見をお聞かせください。
- 実装済みまたは間もなく実装される予定の `ifort` コンパイラー・オプションに対してメッセージは表示されません。
- GNU* および Microsoft* 互換オプションは `ifort` と `ifx` で認識されます。

3.7 fp-model と fp

2024 年 5 月 17 日更新

`ifx` で `-fp-model=precise (/fp:presice)` がサポートされました。

`-fp-model=fast (/fp:fast)` と `fp-model=fast=2 (/fp:fast=2)` の動作は、`ifx` と `ifort` では異なります。

- `ifx` の `-fp-model=fast (/fp:fast)` ではオプション `-fimf-precision=medium` が設定されます
  - `ifx` 2024.0.0 では以下のようになります。
    - fast=1 と fast=2 は同じです
    - NaN オペランドのチェックは生成されません。

- ifort では、-fp-model=fast=2 (/fp:fast2) はオプション -fimf-precision=medium (/Qimf-precision:medium) および -fimf-domain-exclusion=15 (/Qimf-domain-exclusion=15) を設定します。ifort では、浮動小数点の比較は IEEE 浮動小数点標準で指定されるように行われ、比較用に生成されたコードシーケンスでは、比較に NaN が含まれる可能性があることが想定されます。

-fp-model=consistent (/fp:consistent) は、ifx 2022.0.0 以降でサポートされています。Fortran 開発者ガイドおよびリファレンスは、2024.0.0 時点ではまだ反映されていません。これは、-fp-model precise -no-fma -fimf-arch-consistency=true (/fp:precise /Qfma- /Qimf-arch-consistency:true) と同等です。

-fp-model=source (/fp:source) は ifx でサポートされています。このオプションは -fp-model=precise (/fp:precise) と同じ精度を提供しますが、括弧を含む式に対して Fortran 式評価ルールを使用するようにコンパイラーに強制するには -fprotect-parens (/Qprotect-parens) を有効にします。

-fp-model=except (/fp:except) は ifx ではサポートされていません。使用しても警告メッセージは表示されません。

3.8 fimf ファミリーのコンパイラー・オプション

2023年12月5日更新: funclist オプションをサポート

ifx の fimf コンパイラー・オプションでは、ifort で利用可能なオプションの機能リストはサポートされていません。fimf コンパイラー・オプションの一覧を以下に示します。

```
fimf-absolute-error=value
fimf-accuracy-bits=bits
fimf-arch-consistency=value
fimf-max-error=ulps
fimf-precision=value
fimf-domain-exclusion=classlist
```

3.9 -parallel

ifort では、-parallel コンパイラー・オプションで自動並列化が有効になりますが、ifx では自動並列化機能はありません。

3.10 -fstrict-overflow (Linux*) /Qstrict-overflow (Windows*)

算術式の整数オーバーフローは、Fortran 標準では許可されていません。ただし、一部のレガシープログラムは整数オーバーフローに依存しています。オーバーフローが発生すると、結果に収まる限りの値が割り当てられ、符号ビットが変更されることもあります。

デフォルトでは、ifx コンパイラーは整数演算がオーバーフローしないと想定します。ifx のデフォルトは -fstrict-overflow (Linux*) または /Qstrict-overflow (Windows*) です。strict-overflow を有効にすると、コンパイラーは整数演算がオーバーフローしないことを想定し、より優れた最適化が可能になります。ただし、オーバーフローが発生すると、結果の動作は未定義となり、この動作はデフォルトの ifort 動作と互換性がない可能性があります。

ifort は整数オーバーフローを許可します。したがって、整数オーバーフローの ifort 動作に依存するプログラムは、ifx で -fno-strict-overflow (Linux*) または /Qstrict-overflow- (Windows*) オプションを使用する必要があります。ifx の -fno-strict-overflow (Linux*) または /Qstrict-overflow- (Windows*) により、コンパイラーは整数がオーバーフローする可能性があると想定し、オーバーフローを許可します。ifx でオーバーフローを許可すると、コードの最適化が低下する可能性があります。

3.11 最適化レポート

最適化レポートは、ifx で実装が進行中の機能です。各リリースで多くの情報が追加されていきます。

以下の点に留意してください。

- 最適化レポートのコンパイラー・オプション -qopt-report および /Qopt-report には変更はありません。
- 詳細レベルには、1、2、3 を指定できます。
- Linux* ではレポートは stdout に出力されます。Windows* ではコマンドプロンプトでコンパイルすると、そのウィンドウに出力されます。
  - stdout またはターミナルに出力される最適化レポートは、インテル固有のベクタライザーによって実行された最適化が報告されます。
- Linux* では、-qopt-report は .yaml 拡張子付きのファイルを生成します。これは、LLVM コミュニティーで作成された最適化レポートです。opt-viewer.py ツールを使用して表示します。現時点では、インテルの最適化サポートを使用することを推奨します。

3.12 インテル® アドバンスト・ベクトル・エクステンション 512 (インテル® AVX-512) の注意事項

- -x または -ax を使用したコンパイルは、コンパイラーへの提案や要求であり、強制ではありません。
- ifx はインテル® AVX-512 の代わりに インテル® AVX2 を使用することがあります。
- –mprefer-vector-width=512 を使用します。
  - ifx –mprefer-vector-width=512
  - ifort コンパイラーでの同様のオプションは、–qopt-zmm-usage=high です。

4 コンパイラー・バージョン

- `ifx` では新しいバージョンマクロがサポートされます。
  - `__INTEL_LLVM_COMPILER`
- バージョン文字列

  2 つのコンパイラー `ifort` と `ifx` のバージョン文字列が新しくなりました。インテル® oneAPI のバージョン管理セマンティクスが使用されます。詳細は、こちら (英語) を参照してください。

  例:
  ```
  ifx -version
  ifx (IFORT) 2022.0.0 20211123
  ```

  形式は次のとおりです。
  ```
  MAJOR.MINOR.PATCH （ビルド番号）
  ```

  説明:
  - `MAJOR` は、製品バージョン (年)です。必ずしも暦年と一致するとは限りません。
  - `MINOR` は、"1" から始まる 1 桁のマイナーバージョン番号です。"1" は最初のリリースで、新しいマイナーリリースが提供されるたびに 1 つ増えます。
  - `PATCH` は "0" から始まり、特定のバグとセキュリティーの問題が解決されるごとに数値が増加します。
  - ビルド番号は `YYYMMDD` 形式です。

5 オブジェクト・ファイルと .mod ファイル

ifx のバイナリー (.o/.obj) およびモジュール (.mod) ファイルは ifort と互換性があります。ifort で生成されたバイナリーとライブラリーは、ifx で生成されたバイナリーやライブラリーとリンクが可能であり、ほかのコンパイラーで生成された .mod ファイルを使用できます (64 ビット・ターゲットのみ)。icc、icl、icpc、icx、icpx、dpcpp で作成されたオブジェクト・ファイルはバイナリー互換であり、相互にリンクできます。

ただし、-ipo コンパイラー・オプションを使用してコンパイルする場合、ifort と ifx で生成されたオブジェクト・ファイルには互換性がありません。

5.1 インテル固有のディレクティブのサポート

一般的な非 OpenMP* コンパイラー・ディレクティブ (!DIR$) は、リリースごとに追加されています。

!DIR$ ディレクティブは、コンパイラーのフロントエンドで認識されますが、すべてが実装されているわけではありません。ifx 2022.0.0 では以下の !DIR$ ディレクティブをサポートしています。

- IVDEP、VECTOR [NO]DYNAMIC_ALIGN、VECTOR [NO]REMAINDER、DISTRIBUTE_POINT、NOFUSION、[NO]UNROLL、[NO]UNROLL_AND_JAM

5.2 OpenMP* オフロードに関連する基本的な環境変数

ここで示す環境変数は、オフロードされる計算に関連する詳細情報を制御および取得するのに役立ちます。詳細は、『インテル® Fortran コンパイラー・クラシックおよびインテル® Fortran コンパイラー・デベロッパー・ガイドおよびリファレンス』のこちらのセクション (英語) を参照してください。

- ターゲットデバイスを選択します: OMP_TARGET_OFFLOAD= mandatory | disabled | default
  - mandatory: TARGET 領域のコードは、GPU またはほかのアクセラレーターで実行されます。
  - disabled: TARGET 領域のコードは CPU で実行されます。
  - default: デバイスが利用可能であれば TARGET 領域のコードは GPU で実行され、そうでなければ CPU にフォールバックして実行されます。
- プラグイン (デバイスドライバー) を選択します: LIBOMPTARGET_PLUGIN=OPENCL | LEVEL0
  - デフォルトは LEVEL0 です。
  - コンパニオン環境変数: LIBOMPTARGET_DEVICE_TYPE=GPU | CPU
    - デフォルトは GPU です。
    - CPU は OPENCL のみに実装されています。

- プログラムの終了時に GPU カーネルのランタイム・プロファイルを出力します。

  LIBOMPTARGET_PLUGIN_PROFILE = T

  - プロファイルには GPU カーネルの開始時間と終了時間が含まれます。
  - データ転送時間も出力されます。
- オフロード実行時のデバッグ情報をダンプします。
  - LIBOMPTARGET_DEBUG = 0 | 1 | 2
    - デフォルトは 0 (無効) です。
  - LIBOMPTARGET_INFO = 0 | 1 | 2 | 4 | 8 | 32
    - デフォルトは 0(無効)です。
    - 詳細は、こちらの (英語) LLVM ドキュメントを参照してください。

5.3 ifx を使用した dpcpp オブジェクト・ファイルのリンク

一部の開発者は、DPC++ を使用して Fortran プログラムで計算カーネルをオフロードする方法を選択することがあります。

正しくリンクするには、次の例のように `ifx` に追加のコンパイラー・オプションが必要です。

```
$ dpcpp -c device.cpp
$ ifx -qopenmp -fsycl host.f90 device.o -lstdc++ -lsycl
```

これは、`ifx` の 2022.1.0 以前のバージョンで機能します。将来のバージョンでは `-qopenmp` は不要になります。

5.4 ifx を使用したインテル® Fortran ライブラリーの静的および動的リンク

Linux*

Linux* の `ifort` は、デフォルトで 2 つのインテル® ライブラリーを除くすべてのライブラリーを静的にリンクしていました。これは、`-static-intel` オプションを使用するのと同じです。デフォルトで静的リンクされない 2 つのインテル® ライブラリーは、`libiomp5` (OpenMP ランタイム・ライブラリー) と `libicaf` (Coarray Fortran ランタイム・ライブラリー) です。OpenMP ランタイム・ライブラリーを静的にリンクするには、コンパイラー・オプション `-qopenmp-link=static` (デフォルトは `-qopenmp-link=dynamic`) を使用します。`libicaf` の静的バージョンは存在しないため、動的にリンクする必要があります。

Linux* の `ifx` では、デフォルトは `ifort` と同様です。次の例外を除き、ほとんどのインテル® ライブラリーは静的にリンクされます。

1. `libicaf` (Coarray Fortran ライブラリー)
2. `libiomp5` (OpenMP ランタイム・ライブラリー)
3. `libimf` (インテルの `libm` 置き換えイントリンジクス)
4. `libintlc` (`libc` 置き換え関数)

`ifx` で `libimf` と `libintlc` を静的にリンクするには、`-static-intel` コンパイラー・オプションを使用します。OpenMP ランタイム・ライブラリーを静的にリンクするには、`-qopenmp-link=static` (デフォルトは `-qopenmp-link=dynamic`) コンパイラー・オプションを使用します。`libicaf` の静的バージョンは存在しないため、動的にリンクする必要があります。

また、静的リンクを使用している場合は、コンパイラーのコマンドラインの `-l<library name>` にライブラリーが指定されていないことを確認してください。`-l<library name>` を追加すると、`-static*` オプションが上書きされ、そのライブラリーが動的にリンクされます。たとえば、

```
ifx -static-intel -limf foo.f90
```

では `libimf` が動的にリンクされるようになります。

Windows*

Windows* では、両方のコンパイラーはデフォルトで常に動的リンクを行います。Windows* では静的リンクは使用できません。

5.5 既知の問題

- `ifx` 2023.x では、OpenMP* TARGET 領域で許可される I/O は、`print *` のみであり、数値、論理、複素数、文字列に制限されています。訳者注: 日本語文字列は表示できないことがあります。
- `ifx` 2022.3.0 でサポートされた `ATTRIBUTES DLLEXPORT` と `DLLIMPORT` は、Windows* ではまだサポートされていません。
- ifx 2023.0 以降では、Windows* QuickWin アプリケーションがサポートされました。
- `COMPLEX` データタイプのパフォーマンス
  - インテル® Fortran コンパイラー (バージョン 2023.x 以前) の COMPLEX データタイプのパフォーマンスを示します。影響を受けるデータタイプは以下です。

    `COMPLEX(KIND=4)` (`COMPLEX*8` と同様)
    `COMPLEX(KIND=8)` (`COMPLEX*16` または `DOUBLE COMPLEX` と同様)
    `COMPLEX(KIND=16)` (`COMPLEX*32` と同様)

  - これらのデータタイプを使用するアプリケーションでは、インテル® Fortran コンパイラーはインテル® Fortran コンパイラー・クラシックと比較してパフォーマンスが低下します。今後のリリースでは、これらの Fortran 組込みデータタイプのパフォーマンス改善に取り組んでいきます。

5.6 PGO、IPO、リンクに関する変更

`ifx` コンパイラーは、プロシージャー間の最適化 (IPO) とプロファイルに基づく最適化 (PGO) に関して、クラシック・コンパイラーとは異なるアプローチを採用しています。

- **PGO**
  - `ifort` で実装されている PGO (プロファイルに基づく最適化) はサポートされていません。プロファイルは、Clang の `-fprofile` コンパイラー・オプションを使用して行うことができます。詳細は、Multi-Stage PGO (英語) を参照してください。
- **IPO**
  - LLVM は、リンク時の最適化 (LTO) テクノロジーを採用しています。`ifort` では、“プロシージャー間の最適化” (IPO) と呼ばれます。

LLVM LTO の詳細については、LLVM リンク時の最適化を参照してください。

- `-ipo` は、`ifx` ドライバーによって `-flto` に自動変換されます。
- `-ipo` コンパイラー・オプションのバリエーションは削除されました。
- ただし、`-ipo` コンパイラー・オプションを使用してコンパイルすると、`ifort` と `ifx` で生成されたオブジェクト・ファイル間には互換性がありません。
- `-ipo` でコンパイルされたオブジェクトを使用して静的ライブラリーを作成するには、`llvm-ar` を使用します。このユーティリティーは、インテル® コンパイラーに含まれます。

```
1. ifx -ipo -c issue.f90
2. llvm-ar rc ifx_ipo.a issue.o
```

- インテルの `xi*` リンカーツール (`xilink`、`xild`、`xiar`) は削除されました。
  - インテルのツール `xilink`、`xild`、および `xiar` は、インテル® コンパイラーが使用する独自のオブジェクト・ファイル形式向けであるため、`ifx` では削除されました。
  - Makefile やプロジェクトの設定で `xilink` や `xild` を使用している場合、同等のネイティブリンカーに置き換えてください。同様に、`xia` は `ar` などのアーカイバーに置き換えます。

5.7 ifx でサポートされない ifort の機能

- ガイド付き自動並列化
  - `ifort` で非推奨
- プロファイルに基づく最適化 (PGO) は、Clang の `-fprofile` コンパイラー・オプションを使用して実装されます。
- 投機実行の副作用の可能性があるコードの識別

5.8 ブルータスまたはバイセクション最適化のサポート

インテル® Fortran コンパイラーの「Brutus(ブルータス)」または Clang/LLVM の「Bisectional(バイセクション)最適化」に興味のない方は、このセクションをスキップしてください。

`ifx` は、Clang/LLVM のバイセクション最適化デバッグ用の `-opt-bisect-limit=N` オプションをサポートしています。これは、`ifort` のブルータスオプションに類似しています。オープンソース・コミュニティーは、現在 Clang/LLVM の最適化デバッグ機能を強化するため取り組んでいます。

使用例:

```
$ ifx -mllvm -opt-bisect-limit=300 t.f90
```

詳細は、「`-opt-bisect-limit` を使用して最適化のエラーをデバッグ」(英語)を参照してください。

6 付録

6.1 参考資料

次のリファレンスをご覧ください。

- インテル® Fortran コンパイラー・クラシックおよびインテル® Fortran コンパイラー・デベロッパー・ガイドおよびリファレンス(英語)
- インテル® Fortran コンパイラー・クラシックおよびインテル® Fortran コンパイラー・デベロッパー・ガイドおよびリファレンスの「`ifx` の新機能」(英語)
- インテル® oneAPI プログラミング・ガイド:環境設定からアプリケーションのコンパイルと実行まで、プログラミングを始めるのに役立つ一般的なガイド
- `ifx` の Fortran 言語と OpenMP* 機能:コンパイラーのリリースごとのアップデート
- oneAPI 向けインテル® Fortran コンパイラー・リリースノート(英語)
- インテル® Fortran コンパイラー・クラシックの Fortran 言語標準のサポート
- インテル® Fortran コンパイラーの情報とよくある問い合わせ(英語)
- DPCPP または ICX へ移行する ICC ユーザー向けポーティング・ガイド