PGI Compiler Option 一覧

PGI コンパイラのコンパイル・オプション (2019年2月版 PGI 2019対応)

PGI の F77, F2003, C, C++ のコンパイラを使用する際のオプションを以下に示しました。以下は、pgfortran を使用した場合の例ですが、コンパイラのオプションの設定方法は、他の言語コンパイラでも同じです。なお、各オプションの詳細は、PGI User's Guide をお読みください。

```
コマンド名 -[options] [path] filename

(Fortran言語 : 例)  pgf95/pgf90/pgfortran は全て同じコンパイラです。
  pgf95 -fastsse -Minfo=all -L/opt/lib -lmylib test.f
  pgf90 -fastsse -Minfo=all -L/opt/lib -lmylib test.f
  pgfortran -fastsse -Minfo=all -L/opt/lib -lmylib test.f

(C11言語 : 例)
  pgcc  -fastsse -Mipa=fast,inline -L/opt/lib -lmylib test.cpp

(PGI C++03 for Windows) PGI 2016以降、Windows 版の C++ コンパイラは廃止されました（終息）。
  pgCC(pgcpp)  -fastsse -Mipa=fast,inline -L/opt/lib -lmylib test.cpp
  PGI 16.1 以降、Windows 版の C++ コンパイラ処理系の提供はありません。

(PGI C++17 言語 for GNU 4.8 ABI 互換、Linux/macOS/OpenPOWER 版) 一例
  pgc++ -fastsse -Mipa=fast,inline  test.cpp
    PGI 18.7 以降 GNU 4.8～8.1 ABI互換
    PGI 18.1 以降 GNU 4.8～7.2 ABI互換
    PGI 17.7 以降 GNU 4.8～6.3 ABI互換
    PGI 17.1 以降 GNU 4.8～6.2 ABI互換
    PGI 16.1 以降 GNU 4.8～5.1 ABI互換
    PGI 15.4 以降 GNU 4.8～4.9 ABI互換
    PGI 15.1 以降 Apple OS X でもサポート開始
    PGI 14.7 以降 GNU 4.8 ABI互換
    PGI 13.2～14.7以降 GNU ABI互換 (Linux版)
```

必要とする各オプションを *-[option]* 形式でブランクを空けて指定します。また、-M オプションは、最適化オプションを詳細に指定するものであり、-M に引き続きブランクを空けずにフラグを指定します。なお、-M にさらに、サブ・フラグがある場合は、-M[flag]={subflag} の形式で指定します。**サブ・フラグを指定しない場合は、コンパイラの default 設定のサブ・フラグが使用されます。**

[options]　各コンパイル・オプションを指定する。指定順序は基本的に制約はない
　　但し、ライブラリパス等の順序は重要であり、その順位で反映される
[path]　　リンカへのライブラリ等のパスを指定する
[filename] ソースファイル、オブジェクト・ファイル、アセンブリ・ファイル等を指定する

コンパイル・オプションの概要について以下の表に纏めました。表の中の**「白抜き」の行は、最適化において、よく使用されるオプション**を表しています。

- PGI コンパイル・オプションの説明
- -M オプション（最適化詳細オプション）の各種フラグについての説明
- C、C++ 特有のコンパイル・オプション

PGI Compiler のコンパイル・オプション

オプション	記述
-#	コンパイラ手続きの呼出し情報を表示します。
-###	ドライバコマンドを表示しますが、実行しません（-dryrunと同じ）。
-**acc**[=strict|verystrict]	OpenACC用ディレクティブを認識し、GPU用のコードの生成を行います。(リンク時オプションにも必要)　PGI 12.6 以降 -acc=[no]autopar は、OpenACC parallel構文内の自動並列化を行う[行わない] (PGI 13.6 以降)。 -acc=[no]required はアクセラレータ・コードを生成出来なかった場合、コンパイ

	ルエラーとする(default) （PGI 14.1 以降）、(PGI 15.1 以降廃止) -acc=strict は、non-OpenACC accelerator ディレクティブが見つかった場合、warning を出す。-acc=verystrict は、non-OpenACC accelerator ディレクティブが見つかった場合、エラーメッセージを出し、コンパイルを終了する。 -acc=sync は、async cluase を無視します。 -acc=[no]wait は各デバイス kernel の終了を待つか待たないかを指示します。
-Bdynamic	(Linux) 明示的にコンパイラのドライバが、shared object library をリンクするように指示する。 (PGI 7.1 以降) Windows にも対応。 PGI のランタイムライブラリの DLL をリンクして実行バイナリを生成します。このオプションは、コンパイル時とリンク時の両方で必要です。
-Bstatic	(Linux) 明示的に、PGI runtime static libraries を使用して静的リンクを行うように指示する。 （PGI 7.1 以降） Windows にも対応。Windows 上では、実行形式モジュールにリンクされる全てのファイルは、同じオプションでコンパイル/リンクされなければなりません。また、本オプションは、コンパイル時にも必要です。Windows のデフォルトは、-Bstatic となりました。
-Bstatic_pgi	(Linux only) PGI 用の share library を静的にリンクし、システム依存のライブラリは、ダイナミック・ローディングする形式の実行モジュールを生成する（PGI 6.2 以降）。 このオプションは、-Mnorpath 機能を含む。
-byteswapio	（Fortran only) アンフォーマット Fortran データ･ファイルの入出力時にビッグエンディアンからリトルエンディアンにあるいはその逆に、バイトをスワップします。生成された実行モジュールは、自動的に read/write 処理中にこのエンディアン変換を行います。
-C	実行時の配列の境界チェックを有効にする実行モジュールを作成するように指示する。
-c	アセンブリフェーズの後で止まり、オブジェクトコードを filename.o にセーブ。
-cudalibs	リンク時に、CUDA ランタイム API ライブラリ群をリンクする
-D<arg>	プリプロセッサマクロを定義します。
-d[D|I|M|N]	**【PGI 7.0 以降　新設】** プリプロセッサからの追加情報を出力させるためのものです。 　-dD：ソースファイルからマクロと値をプリントします。 　-dI：インクルードファイル名をプリントします。 　-dM：前以って定義された、コマンドラインマクロを含むマクロと値をプリントします。 　-dN：ソースファイルからマクロ名をプリントします。
-dryrun	コンパイル手続き上のドライバコマンドを表示しますが,、実行しません。
-drystdin	(PGI 7.2新設）標準インクルード・ディレクトリを出力して終了します。
-E	プリプロセスフェーズの後で止まり、標準出力にプリプロセスされたファイルを表示。 (PGI 7.0 以降) pgcc -E は、.h ファイルの前処理を行うようになりました。
-F	（pgf77、pgfortranとpghpfのみ）プリプロセスフェーズの後で止まり、プリプロセスされたファイルを　filename.f　にセーブ。
-f	無視されます。
-fast	一般的に最適化セット・フラグ。 x86 並びに AMD64 ターゲットに対するフラグ -O2 -Munroll -Mnoframe -Mlre と 同等。PGI のバージョンによって異なるため、pgf90 -fast -help でオプションの内容を確認すること。 **【PGI 7.0 以降の C/C++ 環境】** C/C++コンパイラは、-fast あるいは -fastsse の複合オプションの中に、-Mautoinline 機能も有効になるように変更されました。 **【PGI 7.0 以降の 64 ビット環境】** 64 ビットシステムのターゲットに対して、-fast オプションは、従来の -fastsse オプションと同じ機能を有するオプションに変更しました。新しい -fast オプションは、SSE 命令を伴うベクトル化、キャッシュ整列、flushz（SSEのflush-to-zeroモード）機能を有効にします。従来の -fast と等価な機能として、**-nfast** と言うオプションが新設されました。
-fastsse	SSE/SSE2 インストラクションを有するマシンターゲットへの一般的な最適化フラグセット。 x86 並びに AMD64 ターゲットに対するフラグ、-O2 -Munroll -Mnoframe -Mscalarsse -Mvect=sse -Mchache_align -Mflushz と同等 **【PGI 7.0 以降の C/C++ 環境】** C/C++コンパイラは、-fast あるいは -fastsse の複合オプションの中に、-Mautoinline 機能も有効になるように変更されました。
-flags	有効なドライバオプションとその内容を表示します。この場合は、コンパイルの実行は行われません。

-fpic	（Linux only) 他のコンパイラとの互換性を有するポジション独立のコードを生成します。Dynamic Shared Library を作成する際に使用することができる。-mcmodel=medium と共には使用できません。
-fPIC	（Linux only) -fpicと同じ。
-G	リンカに共有オブジェクトファイル（ダイナミック・リンク・ライブラリ）を生成するように指示する
-g	オブジェクトモジュールにデバッグ情報を含ませます。
-gopt	オブジェクトモジュールにデバッグ情報を含ませます。最適化されたコードのデバッギングが可能とするような方法でモジュールが生成されます。-goptはシンボリックデバッグ情報をオブジェクトモジュールの中に付加し、さらに、-gが指定されない時と同じ最適化コードを生成するように、コンパイラに対して指示するものです。
-g77libs	（Linux only) g77 によって生成されたオブジェクトファイルを pgfortran を使用してコンパイルされたメインプログラムにリンクする場合、このオプションを指定することで、g77 でコンパイルされたプログラム内で使用している未解決な g77 サポートライブラリを検索できるようにします。
-help	ドライバが認識する全てのオプションを標準出力に表示します。また、オプションとサブオプションの内容も表示します。 -help と他のコンパイルオプションを同時に付けた場合、そのオプションの意味・内容を表示します。
-I<dirname>	ディレクトリを #include ファイルのためのサーチパスに加えます。
-i	リンカに渡されます。
-i2	2 バイトとしてINTEGER変数を扱います。（明示的にバイト数を指定しない整数宣言変数）
-i4	4 バイトとしてINTEGER変数を扱います。（明示的にバイト数を指定しない整数宣言変数）
-i8	8 バイトとしてINTEGER変数を扱い、INTEGER*8 オペレーションに 64ビットを使います。
-i8storage	INTEGER変数を 4 バイトとして扱うが、ストアする際に 8 バイトワード(64bit)としてストアする。
-K<flag>	特別なセマンテイックをコンパイルに指示します。<flag> は多種類あるため、詳細は、User's Guide を参照。例えば、IEEE 754 に準拠するように浮動小数点演算を行う、あるいは、**浮動小数点演算の例外処理の方式等の指定**が可能です(default は例外が起きても実行続行する)。 ieee / noieee ：厳密なIEEE 754に準拠する浮動小数点演算 pic ：ポジション独立のコードを生成 trap=[subflag] ：例外が生じた場合、実行を停止させます。 trap=none ： 全てのトラップを抑止（PGI 6.2) （例） **pgfortran -Ktrap=fp test.f**
-L<dirname>	ライブラリ・ディレクトリを指定します。これをライブラリ・サーチ・パスに加えます。
-l<library>	指定された<library>ライブラリをロードします。
-M<pgflag>	コード生成と最適化の各種のフラグ<pgflag>を指定します。フラグの指定方法は、-M<pglag>,<pgflag>, ... or -M<pgflag>=xxxx
-m	標準出力にリンクマップを表示します。
-m32	デフォルトのプロセッサタイプとして、32ビットコンパイラを使用することをコンパイラに指示する。（PGI 10.3 新設）
-m64	デフォルトのプロセッサタイプとして、64ビットコンパイラを使用することをコンパイラに指示する。（PGI 10.3 新設）
-module <moduledir>	(F90/F95/HPF only) ディレクトリ<moduledir>にモジュールファイル（.mod）を保存/検索します。
-mcmodel=medium	（linux86-64のみ） linux86-64 環境において、medium memory model をサポートするコードを生成します。（2GB 超えのプログラム） このオプションは、 **linux86-64（64bit Linux) のみ**となります。 Win64/osx86-64 では使用できません。
-mp[=align,[no]numa allcores,bind]	ユーザーによって挿入された共有メモリ並列プログラミングディレクティブを解釈、処理します。 **align** サブオプションは、並列化と SSE によるベクトル化の両方が適用されるループにおいて、ベクトル化のためのアライメント（整列）を最大化するようなアルゴリズムを使用して、OpenMP スレッドにループ回数を割り当てるようにするものです。この機能は、このような特性を帯びた多くのループがプログラムに存在する時に性能が向上します。しかし、一方、各ループの中で、非常に大きなタスク・ワークを含むループで、そのループ長が相対的に短いプログラムにおいては、結果としてロードバランスの問題が生じて大きく性能を落とす場合がありますので注意が必

	要です。このオプションを適用し性能を確認してから使用してください。（このalign サブオプションは、**PGI 6.1** 以降のオプションです） **-mp=nonuma** (libnumaライブラリをリンクしない) PGI 6.1 以降 PGI 6.2 から libnumaを有しないシステムには、 その stub（仲介）ライブラリを提供する。 （**PGI 8.0 以降のサブオプション**） allcores　環境変数OMP_NUM_THREADS あるいは NCPUSにセットしていない場合、すべての有効なコアを使用する（リンク時に指定すること） bind　スレッドをコアあるいはプロセッサにバインドする（リンク時に指定すること）
-noswitcherror	コマンドライン上に、コンパイラに有効なオプションではないものが指定された場合、エラーで終了させる代わりに警告レベルに変更する。この挙動は、コンパイラのサイト初期設定ファイル　siterc ファイルに set NOSWITCHERROR=1 を指定することでも可能となります。（PGI 7.0-4 以降）　siterc ファイルは、一般に $PGI/linux86{-64}/{version}/bin の配下にあります。 PGI 7.1 以降は、未知のオプションが指定された場合、コンパイル・エラーとなります。
-**O***level*	コード最適化レベルを指定します。level は 0、1、2、 3 あるいは 4。 0：各ステートメントに対し基本ブロックを生成します。しかし、スケジューリング並びにグローバルな最適化は行いません。 1：基本ブロック内でのスケジューリング並びにいくつかのレジスタ・割当の関する最適化を行います。しかし、グローバルな最適化は行いません。 2：全ての上記　レベル 1 の最適化を行います。さらに、基本ブロック間の制御フローとデータフロー解析を実施し、グローバル最適化を行います。導入変数の削除や問題のないループの移動、グローバルレジスタの割り当て等のグローバル最適化を行います。 3：アグレッシブなグローバル最適化を行います。全てのレベル 1, 2 の最適化だけでなく、効果のあるなしに関わらず、スカラの置き換え、より積極的な最適化を行います。 4：4 レベルの最適化は、浮動小数点演算式の中で不変変数に対する巻上げ最適化を行うようになります。(PGI 7.0 以降で新設) (PGI 7.1) algebraic transformations と レジスタ・アロケーション最適化を追加しました。 (PGI 2013以降)　上記 -O2 の機能の中に SIMD ベクトルコード生成（-Mvect=simd)、キャッシュアラインメント、冗長性の排除等の最適化機能も含まれました。これらの追加最適化は -O3、-O4 でも引き継がれます。 また、-O のみ指定した場合は、上記、レベル 2 の最適化であるが、SIMD ベクトル最適化は行わない形態となります(PGI 2012 以前の -O2 と同等な最適化となります）。
-o	オブジェクトファイルの名前を指定します。
-nomp	(PGI 11.0以降）PGI 11.0 から、リンク時のオプションとして、常に-mpオプション（マルチスレッドライブラリ）がデフォルトして付加されます。これは、リンケージの時の動作ですので、コンパイル時の動作ではありません。もし、このデフォルトを変更したい場合は、新しいオプション -nomp をリンク時に指定して下さい。
-pgc++libs	PGF77 あるいは pgfortran あるいは、 pgcc でオブジェクトをビルドする際に、PGC++ ランタイムライブラリ群をリンクするために使用します。(pgf77 あるいは pgfortran、 pgccで指定する)
-pgcpplibs	PGF77 あるいは pgfortran あるいは、 pgcc でオブジェクトをビルドする際に、PGCPP ランタイムライブラリ群をリンクするために使用します。(pgf77 あるいは pgfortran、 pgccで指定する) (PGI 16.1以降廃止)
-pgf77libs	PGF77 でコンパイルされたオブジェクトを C あるいは C++ のメインプログラムにリンクする際に、PGF77 ランタイムライブラリをリンクするために使用します。（ pgcc あるいは pgCC で指定する） (PGI 6.0～)
-pgf90libs	pgfortran でコンパイルされたオブジェクトを F77 あるいは、C、C++ のメインプログラムにリンクする際に、pgfortran ランタイムライブラリをリンクするために使用します。（pgf77 あるいは、pgcc、pgc++/pgCC で指定する） (PGI 6.0～)
-P	（pgccとpgc++）プリプロセスフェーズの後で止まり、プリプロセスされたファイルをfilename.iにセーブします。
-pc	(CPU target が、px/p5/p6/piii のみ)　浮動小数点計算時の x86 アーキテクチャ上のレジスタビット長の使用精度の制御を行います。プログラムの誤差感度の評価に有効です。 -pc 32　:　単精度 (32bit) -pc 64　:　倍精度 （64bit) -pc 80　:　x87 natice （80bit) このモードがデフォルトです

<table>
<tr><td></td><td>-Kieee も参照のこと（厳密の IEEE 754 準拠）</td></tr>
<tr><td>-pg</td><td>gprof-style のサンプルベースのプロファイルデータを生成する。生成されたプロファイルデータ gmon.out ファイルは、pgprof で分析可能となる。</td></tr>
<tr><td>-Q</td><td>コンパイラステップの変化を選択します。</td></tr>
<tr><td>-R<directory></td><td>(Linux only) リンカへ渡されます。リンク時の共有オブジェクトファイルのサーチパスの中に<directory> を入れます。これは、環境変数 LD_LIBRARY_PATH の内容を変えるものではありません。</td></tr>
<tr><td>-r</td><td>リロケータブルなオブジェクトファイルを作成します。</td></tr>
<tr><td>-r4</td><td>DOUBLE PRECISION 変数を REAL と解釈します。</td></tr>
<tr><td>-r8</td><td>REAL 変数を DOUBLE PRECISION と解釈します。</td></tr>
<tr><td>-rc file</td><td>ドライバのスタートアップファイルの名前を指定します。</td></tr>
<tr><td>-rdynamic</td><td>pgc++コンパイラにリンカーへのオプションとして -export-dynamicを適用するように指示するスイッチ (PGI 16.1以降)</td></tr>
<tr><td>-S</td><td>コンパイルフェーズの後で止まり、アセンブリ言語コードを filename.s にセーブします。</td></tr>
<tr><td>-s</td><td>オブジェクトファイルからシンボルテーブル情報を除去します。</td></tr>
<tr><td>-shared</td><td>(Linux only) リンカへ渡されます。共有オブジェクトファイルを生成するようにリンカに指示します。</td></tr>
<tr><td>-show</td><td>コンパイラ起動時の各ドライバの設定パラメータ、引数の詳細を表示します。</td></tr>
<tr><td>-silent</td><td>警告メッセージをプリントしません。</td></tr>
<tr><td>-soname　<library.so></td><td>(Linux only) shared オブジェクトを生成する時、library.so（一例） と言うシェアードライブラリを内部の DT_SONAME フィールドへセットするようにリンカーに指示します。</td></tr>
<tr><td>-stack=nocheck</td><td>(PGI 7.1 以降) (Windows only) -stack オプションは、Windows 上で自動ランタイムスタック拡張を行わないようにすることができるように変更されました。もし、researve と commit サブオプションが、十分なスタック量を確保できるようにセットされたなら、自動的な拡張チェックは必要ありませんし、スタックのチェックを避けることができます。デフォルトは、-stack=checkです。Win64 では、デフォルトの researve 値あるいは commit 値はありません。Win32 では、researve、commit それぞれのデフォルト値は、2,097,152byteです。</td></tr>
<tr><td>-time</td><td>様々なコンパイルステップの実行時間を表示します。</td></tr>
<tr><td>-ta=nvidia(,suboption),host

-ta=radeon
(,suboptions),host

-ta=multicore</td><td>(PGI 2010以降、pgfortranとpgcc に有効)
(PGI 13.1 pgcpp でも利用可能)
OpenACC 用のターゲット・アーキテクチャを意味します。PGI 13.10 以前は、-ta=nvidia でしたが、PGI 14.1 以降 AMD の Radeon GPU ボードも OpenACC 対応となったため、以下のように、NVIDIA 社と AMD 社の二つのメーカの通称ボード名で、OpenACC コンパイルの「ターゲットの識別」を行います。さらに、各ターゲットに対する細かなオプションを指定できます。(デフォルトは -ta=tesla,host です)
コンパイル方法の詳細はこちらへ
-ta=tesla ： NVIDIA　アクセラレータをターゲットとして選択します。さらに、以下の nvidia 用のサブオプションがあります。
<table>
<tr><th>サブ
オプション</th><th>NVIDIA -ta=tesla(nvidia) のサブオプション</th></tr>
<tr><td>analysis</td><td>ループの解析のみ行い、コードの生成を行いません。）(PGI 13.10以降廃止)</td></tr>
<tr><td>cc10</td><td>compute capability 1.0 のコードを生成 (PGI 14.1以降廃止)</td></tr>
<tr><td>cc11</td><td>compute capability 1.1 のコードを生成 (PGI 14.1以降廃止)</td></tr>
<tr><td>cc12</td><td>compute capability 1.2 のコードを生成 (PGI 14.1以降廃止)</td></tr>
<tr><td>cc13</td><td>compute capability 1.3 のコードを生成 (PGI 14.1以降廃止)</td></tr>
<tr><td>cc1x</td><td>compute capability 1.x のコードを生成 (PGI 15.1以降廃止)</td></tr>
<tr><td>cc1+</td><td>compute capability 1.x, 2.x, 3.x のコードを生成 (PGI 14.1以降)、 (PGI 15.1以降廃止)</td></tr>
<tr><td>tesla</td><td>cc1x と同じ(PGI 13.1以降)、 (PGI 15.1以降廃止)</td></tr>
<tr><td>tesla+</td><td>cc1+ と同じ (PGI 14.1以降)、 (PGI 15.1以降廃止)</td></tr>
<tr><td>cc20</td><td>compute capability 2.0 のコードを生成 (PGI 10.4以降) (PGI 14.1以降廃止) (PGI15.5以降復活)</td></tr>
<tr><td>cc2x</td><td>compute capability 2.x のコードを生成 (PGI 10.4以降)</td></tr>
</table>
</td></tr>
</table>

cc2+	compute capability 2.x, 3.x のコードを生成 (PGI 14.1以降)
fermi	cc2xと同じ (PGI 13.1以降)
felmi+	cc2+と同じ (PGI 14.1以降)
cc30	compute capability 3.0 のコードを生成 (PGI 12.8以降) (PGI 14.1以降廃止) (PGI15.5以降復活)
cc35	compute capability 3.5 のコードを生成 (PGI 13.1以降) (PGI 14.1以降廃止) (PGI15.5以降復活)
cc3x	compute capability 3.x のコードを生成 (PGI 12.8以降)
cc3+	compute capability 3.x (=cc3x) 以上のコードを生成 (PGI 14.1以降)
kepler	cc3xと同じ (PGI 13.1以降)
kepler+	cc3+と同じ (PGI 14.1以降)
cc50	compute capability 5.0 のコードを生成 (PGI 15.7以降)
cc60	compute capability 6.0 のコードを生成 (PGI 16.10以降)
cc70	compute capability 7.0 のコードを生成 (PGI 17.7 以降)
charstring	GPUカーネル内で文字列の使用を制限付きで使用する (PGI 15.1以降)
cuda2.3 or 2.3	PGIにバンドルされた CUDA toolkit 2.3 バージョンを使用 (PGI 10.4以降)
cuda3.0 or 3.0	PGIにバンドルされた CUDA toolkit 3.0 バージョンを使用 (PGI 10.4以降)
cuda3.1 or 3.1	PGIにバンドルされた CUDA toolkit 3.1 バージョンを使用 (PGI 10.8以降)
cuda3.2 or 3.2	PGIにバンドルされた CUDA toolkit 3.2 バージョンを使用 (PGI 11.0以降)
cuda4.0 or 4.0	PGIにバンドルされた CUDA toolkit 4.0 バージョンを使用 (PGI 11.6以降)
cuda4.1 or 4.1	PGIにバンドルされた CUDA toolkit 4.1 バージョンを使用 (PGI 12.2以降)
cuda4.2 or 4.2	PGIにバンドルされた CUDA toolkit 4.2 バージョンを使用 (PGI 12.6以降)
cuda5.0 or 5.0	PGIにバンドルされた CUDA toolkit 5.0 バージョンを使用 (PGI 13.1以降)
cuda5.5 or 5.5	PGIにバンドルされた CUDA toolkit 5.5 バージョンを使用 (PGI 13.9以降)
cuda6.0 or 6.0	PGIにバンドルされた CUDA toolkit 6.0 バージョンを使用 (PGI 14.4以降)
cuda6.5 or 6.5	PGIにバンドルされた CUDA toolkit 6.5 バージョンを使用 (PGI 14.9以降)
cuda7.0 or 7.0	PGIにバンドルされた CUDA toolkit 7.0 バージョンを使用 (PGI 15.4以降)
cuda7.5 or 7.5	PGIにバンドルされた CUDA toolkit 7.5 バージョンを使用 (PGI 15.9以降)
cuda8.0 or 8.0	PGIにバンドルされた CUDA toolkit 8.0 バージョンを使用 (PGI 16.10以降)
cuda9.0 or 9.0	PGIにバンドルされた CUDA toolkit 9.0 バージョンを使用 (PGI 17.9以降)
cuda9.1 or 9.1	PGIにバンドルされた CUDA toolkit 9.1 バージョンを使用 (PGI 18.1以降)
cuda9.2 or 9.2	PGIにバンドルされた CUDA toolkit 9.2 バージョンを使用 (PGI 18.5以降)
[no]debug	デバイスコード内にデバッグ情報を生成する[しない] (PGI 14.1 以降)

deepcopy	OpenACC Fortran における full deep copyを有効にする(PGI 17.7以降ベータサポート)
fastmath	fast mathライブラリを使用
[no]flushz	GPU上の浮動小数点演算の flush-to-zero モードを制御。デフォルトはnoflushz。 (PGI 11.5以降)
[no]fma	fused-multiply-add命令を生成する[しない] (default at -O3)
keep	kernelバイナリファイル(.bin)、kernelソースファイル(.gpu)、portable assembly(.ptx)ファイルを保持し、各々ファイルとして出力する (PGI 13.10以降)
keepbin	kernelバイナリファイルを保持し、ファイル(.bin)として出力する）(PGI 13.10以降廃止)
keepgpu	kernelソースファイルを保持し、ファイル(.gpu)として出力する (PGI 13.10以降廃止)
keepptx	GPUコードのためのportable assembly(.ptx)ファイルを保持し、ファイルとして出力する）(PGI 13.10以降廃止)
[no]lineinfo	GPU line informationを生成する (PGI 15.1 以降)
[no]llvm	llvmベースのバックエンドを使用してコードを生成する。PGI 15.1 以降は、64-bit上ではLLVMバックエンドをデフォルトとして使う [使わない]
managed	OpenACCの 割り当て可能なデータが CUDA Unified Memory に配置されている場合、明示的なデータ移動またはデータ指示を必要としない機能を有効にする（PGI 17.7以降）
maxregcount:n	GPU上で使用するレジスタの最大数を指定。ブランクの場合は、制約が無いと解釈する
mul24	添字計算に、24ビット乗算を使用 (GT200系、CC 1.3のみ) (PGI 13.10以降廃止)
noL1	グローバル変数をキャッシュするためのハードウェア L1 データキャッシュの使用を抑止する (PGI 13.10以降)
loadcache:L1 loadcache:L2	グローバル変数をキャッシュするためのハードウェア L1 or L2 データキャッシュを使用する (PGI 14.4以降)但し、アーキテクチャ上、有効とならない GPU がある
pin	デフォルトを pin ホストメモリ(割付）としてセットする(PGI 14.1以降、PGI16.1廃止)
pinned	プログラムのアロケート時に pinned メモリを割り付けるよう（PGI 16.1以降)
time	アクセラレータ領域の単純な時間情報を集積するためにプロファイル・ライブラリをリンクする。このオプションは、PGI 13.1 以降廃止されました。この代わりに、プロファイルを環境変数 PGI_ACC_TIME に 1 をセットすることにより実行後プロファイル情報が出力されます。
[no]required	アクセラレータ・コードを生成出来なかった場合、コンパイルエラーとする [しない] (default) (PGI 14.1 以降）、 (PGI 15.1以降廃止)
[no]rdc	異なるファイルに配置されたデバイスルーチンをそれぞれ分割コンパイルし、リンクが出来るようにする。cc2x以降、CUDA 5.0 以降の機能を使用する。(PGI 13.1以降 + CUDA 5.0 以降) (PGI 14.1 は以降デフォルト)
[no]unroll	自動的に最内側ループのアンローリングを行う (default at -O3) (PGI 14.9以降)
managed	CUDA managed Memory を使用する
beta	ベータ版機能のコード生成（生成コード内の 128-bit ロード・ストアオペレーションを有効化）(PGI 15.7以降)
[no]wait	ホスト側での実行継続を行う際に、各カーネルが終了するまで待つ。nowaitは待たない。）(PGI 13.10以降廃止)

safecache	cache directive 内での可変長の配列セクションの使用を許す。但し、そのサイズは CUDA shared memory 内に収まるものでなければならない。(PGI 16.5以降)

-ta=tesla**,host** ： host は、アクセラレータがターゲットとして存在しないコード生成も行う。アクセラレータ領域をホスト側で実行するようにコンパイルする。PGI Unified Binaryコードを生成する。

-ta=radeon - AMD アクセラレータをターゲットとして選択します。さらに、以下の radeon 用のサブオプションがあります。このサブオプションは、カンマ（,）で区切って複数のものを指定することができます） (PGI 14.1 以降)。

サブオプション	AMD -ta=radeon のサブオプション
buffercount:n	データをアロケートする際のOpenCLバッファの最大数をセットする
capeverde	Radeon Cape Verde アーキテクチャ用のコードを生成
keep	kernel ファイルを保持する
[no]lineinfo	GPU line informationを生成する (PGI 15.1 以降)
[mo]llvm	llvm/SPIRベースのバックエンドを使用してコードを生成する。PGI 15.1 以降は、64-bit上ではLLVM/SPIRバックエンドをデフォルトとして使う [使わない]
[no]required	アクセラレータ・コードが生成出来なかった場合、コンパイルエラーとする [しない] (default)、(PGI 15.1 以降廃止)
[no]unroll	自動的に最内側ループのアンローリングを行う (default at -O3) (PGI 14.9以降)
capeverde	Radeon capeverdeアーキテクチャ用コードを生成
spectre	Radeon Spectre アーキテクチャ用コードを生成
tahiti	Radeon Tahiti アーキテクチャ用コードを生成
spir	LLVM/SPIRバックエンドをデフォルトとして使う (PGI 15.1 以降)

-ta=radeon**,host** ： host は、アクセラレータがターゲットとして存在しないコード生成も行う。アクセラレータ領域をホスト側で実行するようにコンパイルする。PGI Unified Binaryコードを生成する。

-ta=multicore - ホスト上のマルチコアCPU上で並列動作するように OpenACC プログラムをコンパイルします。（PGI 15.10 以降）

-tp <target>

ターゲットプロセッサのタイプを指定し、そのアーキテクチャに沿ったコードを生成します。 **ターゲットの default は、コンパイルを実行するシステムの「プロセッサ・タイプ」にターゲットが設定されます。コンパイルの方法は、こちらへ**

amd64	AMD64 Processor	PGI6.0以前
athlon	AMD Athlon Processor	PGI6.0以前
athlonxp	AMD Athlon XP Processor	PGI6.0以前
k8-32	AMD Athlon64/Opteron 32-bit mode	
k8-64	AMD Athlon64/Opteron 64-bit mode	
k8-64e	AMD Opteron Rev.E/F Turion 64-bit mode	PGI6.1以降
barcelona	AMD barcelona/Quad-Core AMD64	PGI 7.0-3 以降
barcelona-32	AMD barcelona/Quad-Core AMD64 32-bit mode	PGI 7.0-3 以降
barcelona-64	AMD barcelona/Quad-Core AMD64 64-bit mode	PGI 7.0-3 以降
shanghai	AMD shanghai/Quad-Core AMD64	PGI 8.0以降
shanghai-32	AMD shanghai/Quad-Core AMD64 32-bit mode	PGI 8.0以降

shanghai-64	AMD shanghai/Quad-Core AMD64 64-bit mode	PGI 8.0以降
istanbul	AMD istanbul/six-Core AMD64	PGI 9.0以降
istanbul-32	AMD istanbul/six-Core AMD64 32-bit mode	PGI 9.0以降
istanbul-64	AMD istanbul/six-Core AMD64 64-bit mode	PGI 9.0以降
bulldozer	AMD bulldozer AMD64	PGI 11.9以降
bulldozer-32	AMD bulldozer AMD64 32-bit mode	PGI 11.9以降
bulldozer-64	AMD bulldozer AMD64 64-bit mode	PGI 11.9以降
piledriver	AMD Piledriver AMD64	PGI 13.1以降
piledriver-32	AMD Piledriver AMD64 32-bit mode	PGI 13.1以降
piledriver-64	AMD Piledriver AMD64 64-bit mode	PGI 13.1以降
zen	AMD Zen AMD64 64-bit mode	PGI 18.1以降
piii	Intel PentiumIII with SSE1 only	
p6	Intel Pentium Pro, II, III, AthlonXP	
p7	Intel Pentium 4/Xeon 32-bit mode	
px	Intel generic x86 mode	
p7-64	Intel Xeon/Pentium4 EM64T 64-bit mode	PGI5.2以降
core2	Intel Core 2 (Duo) 32-bit mode	PGI6.2以降
core2-64	Intel Core 2 (Duo) EM64T 64-bit mode	PGI6.2以降
penryn	Intel Penryn 32-bit mode	PGI7.2以降
penryn-64	Intel Penryn 64-bit mode	PGI7.2以降
nehalem	Intel Core i7/i5/i3(Nehalem)	PGI9.0以降
nehalem-32	Intel Core i7/i5/i3(Nehalem) 32-bit mode	PGI9.0以降
nehalem-64	Intel Core i7/i5/i3(Nehalem) 64-bit mode	PGI9.0以降
sandybridge	Intel Core i7/i5/i3(SandyBridge)	PGI11.6以降
sandybridge-32	Intel Core i7/i5/i3(SandyBridge) 32-bit mode	PGI11.6以降
sandybridge-64	Intel Core i7(SandyBridge) 64-bit mode	PGI11.6以降
ivybridge	Intel Core i7/i5/i3(IvyBridge)	PGI14.1以降
ivybridge-32	Intel Core i7/i5/i3(IvyBridge) 32-bit mode	PGI14.1以降
ivybridge-64	Intel Core i7(IvyBridge) 64-bit mode	PGI14.1以降
haswell	Intel Core i7/i5/i3(Haswell)	PGI14.1以降
haswell-32	Intel Core i7/i5/i3(Haswell) 32-bit mode	PGI14.1以降
haswell-64	Intel Core i7(Haswell)以降 64-bit mode	PGI14.1以降
skylake	Intel Core i7(Skylake)以降 64-bit mode	PGI18.1以降
x64	AMD64/EM64Tの両方に対応可能とした最適化を施こす Unified Bynary の生成（-tp p7-64,k8-64 と同意）	PGI6.1以降

【PGI 7.0 以降の 64 ビット環境】
-tp オプションは、コンマ（、）で区切られた複数の64ビット・ターゲットを記述する方式を採用しました。以前のバージョンでは、これは一つのターゲットのみの記述方式でした。もし、複数のターゲットが指定された場合、Unified binary は、

	各ターゲットに対して最適化されたコードを生成します。
-[no]traceback	環境変数 PGI_TERM のスイッチにより異常終了時のトレースバックの処理を制御できますが、その際に必要なデバッグ情報を加えます。なお、Fortranコンパイラのデフォルトは**ON**であり、C/C++ のデフォルトは OFF として設定されています。
-U symbol	プリプロセッサマクロを #undef します。
-u symbol	リンカーにとって未定義なものとしてシンボルテーブルを symbol で初期化します。未定義シンボルは、アーカイブライブラリ上の最初のメンバーのローディングを引きおこします。
-V{Release_Number}	バージョンメッセージ、及び、他の情報を表示します。-V に続けてシステムにインストールしている過去のバージョンを指定した場合、デフォルトバージョンではなく、そのバージョンのコンパイラを使用してコンパイルされます。 （例）　pgfortran -V5.2 test.f (PGI 7.1 以降)　プロセッサ名をプリントするようになりました。例えば、Core 2 Duo上でコンパイルすると-Vオプションは、-tp core2-64と表示します。
-v	コンパイラ、アセンブラ、及び、リンカフェーズ呼出しを表示します。
-W	引数を特定のフェーズ（コンパイル、アセンブラ、リンカ）に渡します。 -W{0,1,l}, <option>,<option> 形式：0　はコンパイラ、1 はアセンブラ、l はリンカ
-w	警告メッセージを表示しません。

-M オプションの各種フラグ

pgflag	記述	カテゴリ
allocatable=[95/03]	【PGI 7.0 以降 pgfortranで　新設】 -Mallocatable= オプションは、コンパイラがメモリ割付 (allocatable) に係る意味合いを制御します。デフォルトは Fortran 95 に準拠します。 =03 オプションは、Fortran 2003 に準拠します。	Fortran95 言語
anno	アセンブリコードと共にソースコードを注釈する。-Manno -S の指定により、アセンブラ・リスティング・ファイル xxxx.s の中にソース・リストとそれに対するアセンブラアセンブラ・リストが両方表示される。	その他
[no]asmkeyword	（pgccとpgc++）コンパイラが　C/C++ ソースファイルにの中に asm キーワードの挿入を許すかどうかを指定。asm キーワードの構文は以下のとおり。 　asm("statement") ; statement はアセンブラ言語による文であり、ダブル・クォツで囲むことが必要。	C / C++言語
[no]autoinline[=levels:n | maxsize:n | totalsize:n]	（PGI 6.2 以降）最適化オプション -O2 以上において、C/C++ コンパイラはインラインキーワードで定義されたもの、あるいはクラス実体(class body)で定義された関数をインライン化する。-Mnoautoinline は、このインライン化を抑止する。levels:n は、インラインの段数（レベル）の数の制約値を指定します。そのデフォルトは4です。 （PGI 2010 以降の C/C++ コンパイラ）　-O2 オプション時にインライン化をコンパイラに指示する。 levels:n　：インラインを行うレベル階層を最大 n まで行うことを指示。デフォルトは10 です。 maxsize:n ： nサイズを超える関数のインラインを行わない。デフォルトは100。 totalsize:n ： インライン対象が、n サイズ時にインラインを止めることを指示。 デフォルトは800。	インライン C / C++言語
[no]backslash	（pgf77、pgfortranとpghpfのみ） backslash キャラクタが quote された文字列において　escape キャラクタとして扱うかを決定。	Fortran言語
[no]**bounds**	実行時の配列の境界チェックを有効にするか、無効するかを指定。プログラムのデバッグ時に非常に有効である。例えば、配列境界外のアクセスを行った場合、以下のような形式で出力される。 　PGFTN-F-Subscript out of range for array a (a.f: 2) 　subscript=3, lower bound=1, upper bound=2, dimension=2	その他

<table>
<tr><td>[no]builtin</td><td>（pgccとpgc++）数学サブルーチンのビルトインサポート（選択された数学ライブラリルーチンをインライン化する）を用いてコンパイルする[しない]。</td><td>最適化</td></tr>
<tr><td>byteswapio</td><td>FortranアンフォーマットデータのI/O時にバイトオーダをスワップ（リトルエンディアンからビッグエンディアンに、またその逆）する。</td><td>その他</td></tr>
<tr><td>cache_align</td><td>可能な限り、16バイト以上のデータオブジェクトをキャッシュラインに整列させる。特に、SSE/SSE2 のベクトル化を行う際に有効（必須）である。</td><td>最適化</td></tr>
<tr><td>chkfpstk (32bit only)</td><td>関数の開始時と、関数またはサブルーチン呼び出しから戻った後での x86 FPスタックの内部の一貫性についてチェック。
実行時に環境変数 PGI_CONTINUE=verbose のセットを行うと警告メッセージが出る。32bit のみに有効で、64ビット環境では無視される。</td><td>その他</td></tr>
<tr><td>chkptr</td><td>（pgfortranとpghpfのみ） NULLポインタについてチェック。</td><td>その他</td></tr>
<tr><td>chkstk</td><td>パラレル領域のエントリー時と、パラレル領域の開始前にエントリ上のスタックの利用可能なスペースをチェック。多くのプライベートな変数が宣言されるときに有益。
（PGI 7.1 以降） -Mchkstk オプションでコンパイルされたプログラムは、スタック high-water mark の情報を収集できるように指示できます（Windows版のみ）。もし、環境変数 PGI_STACK_USAGE が実行時にセットされた場合、スタックの high-water mark が実行終了時に印字されます。</td><td>その他</td></tr>
<tr><td>concur[=flag[,flag,...]]</td><td>ループの自動並行化を有効にします。複数のプロセッサにより並列化できるループの並列性を確認し、可能な限り並列化する（共有メモリマルチCPUシステムのみで有効）。以下のサブ・フラグがありますので、詳細は User's Guide を参照のこと。
altcode:n / noaltcode
dist:block / dist:cyclic
cncall
assoc/noassoc
[no]innermost (最内側ループの並列化） PGI 6.1以降
nonuma (libnumaライブラリをリンクしない) PGI 6.1 以降
（PGI 8.0 以降）
allcores 環境変数OMP_NUM_THREADS あるいは NCPUSにセットしていない場合、すべての有効なコアを使用する （リンク時に指定すること）
bind スレッドをコアあるいはプロセッサにバインドする（リンク時に指定すること）</td><td>最適化</td></tr>
<tr><td>cpp=[option]</td><td>後続のコンパイル手続きを行わずに、PGI cppライクのプリプロセッサを実行する。このオプションは、makefileの中に含む各ルーチンの依存情報を生成する際に有効です。optionは、以下に示す一つあるいは複数の文字列（m, md, mm, mmd）からなる。もし、これらの複数のオプションが指定された場合は、最後にリストされたオプションのみが有効となる。
<table>
<tr><td>m</td><td>: makefile dependenciesをstdoutに出力する。</td></tr>
<tr><td>md</td><td>: makefile dependenciesをfilename.dと言うファイルに出力します。ここでfilenem.dとは、コンパイルする入力ファイル名のルート部分の名前が採用される。</td></tr>
<tr><td>mm</td><td>: makefile dependenciesをstdoutに出力しますが、システムincludeファイルは無視する。</td></tr>
<tr><td>mmd</td><td>: makefile dependenciesをfilename.dと言うファイルに出力します。ここでfilenem.dとは、コンパイルする入力ファイル名のルート部分の名前が採用される。なお、ステムincludeファイルは無視する。</td></tr>
<tr><td>[no]comment</td><td>: プリプロセス処理の出力のコメントを残す（さない）。</td></tr>
<tr><td>[suffix:]
<suff></td><td>: makefile dependenciesを含むファイルの添字として<suff>を使用する</td></tr>
</table>
</td><td>その他</td></tr>
</table>

<table>
<tr><td>cray</td><td>（pgf77、pgfortranとpghpfのみ）Cray Fortran（CF77）互換性を強制。</td><td>最適化</td></tr>
<tr><td>cuda=[option]</td><td>（pgfortranのみ、PGI 2010以降、アクセラレータ製品のみ） コンパイラは、一般的な Fortran 構文だけでなく CUDA Fortran 構文を解釈するコンパイラモードとなる。CUDA Fortran プログラムをコンパイルし、必要なライブラリをリンクします。なお、リンク時においてもこのオプションが必要です。以下のサブオプションを有する。このサブオプションは、カンマ（,）で区切って複数のものを指定する。
<table>
<tr><th>サブ
オプション</th><th>nvidia用 機能</th></tr>
<tr><td>emu</td><td>エミュレーションモードでコンパイルします。これは、GPU 用のコード生成は行わず、ホスト側でエミュレーション実行可能なコードを生成します。一般に、デバッグ時に使用します。CUDA Fortran の " device code (kernel)" は、ホスト上で実行出来るコードで生成され、ホスト側の pgdbg デバッガを使用できます。</td></tr>
<tr><td>cc10</td><td>compute capability 1.0 のコードを生成 (PGI 13.10以降廃止)</td></tr>
<tr><td>cc11</td><td>compute capability 1.1 のコードを生成 (PGI 13.10以降廃止)</td></tr>
<tr><td>cc12</td><td>compute capability 1.2 のコードを生成 (PGI 13.10以降廃止)</td></tr>
<tr><td>cc13</td><td>compute capability 1.3 のコードを生成 (PGI 13.10以降廃止)</td></tr>
<tr><td>cc1x</td><td>compute capability 1.x のコードを生成 (PGI 15.1以降廃止)</td></tr>
<tr><td>cc1+</td><td>compute capability 1.x, 2.x, 3.x のコードを生成 (PGI 14.1以降), (PGI 15.1以降廃止)</td></tr>
<tr><td>tesla</td><td>compute capability 1.x (=cc1x) のコードを生成 (PGI 13.1以降), (PGI 15.1以降廃止)</td></tr>
<tr><td>cc20</td><td>compute capability 2.0 のコードを生成 (PGI 10.4以降) (PGI 13.10以降廃止) (PGI15.5以降復活)</td></tr>
<tr><td>cc2x</td><td>compute capability 2.x のコードを生成 (PGI 10.4以降)</td></tr>
<tr><td>cc2+</td><td>compute capability 2.x, 3.x のコードを生成 (PGI 14.1以降)</td></tr>
<tr><td>felmi</td><td>compute capability 2.x (=cc2x) のコードを生成 (PGI 13.1以降)</td></tr>
<tr><td>felmi+</td><td>cc2+と同じ (PGI 14.1以降)</td></tr>
<tr><td>cc30</td><td>compute capability 3.0 のコードを生成 (PGI 12.8以降) (PGI 13.10以降廃止) (PGI15.5以降復活)</td></tr>
<tr><td>cc35</td><td>compute capability 3.5 のコードを生成 (PGI 13.1以降) (PGI 13.10以降廃止) (PGI15.5以降復活)</td></tr>
<tr><td>cc3x</td><td>compute capability 3.x のコードを生成 (PGI 12.8以降)</td></tr>
<tr><td>kepler</td><td>compute capability 3.x (=cc3x) のコードを生成 (PGI 13.1以降)</td></tr>
<tr><td>cc50</td><td>compute capability 5.0 のコードを生成 (PGI 15.7以降)</td></tr>
<tr><td>cc60</td><td>compute capability 6.0 のコードを生成 (PGI 16.10以降)</td></tr>
</table></td><td>CUDA Fortran言語</td></tr>
</table>

cc70	compute capability 7.0 のコードを生成 (PGI 17.7以降)
charstring	GPUカーネル内で文字列の使用を制限付きで使用する(PGI 15.1 以降)
cuda2.3 or 2.3	PGIにバンドルされた CUDA toolkit 2.3 バージョンを使用 (PGI 10.4以降)
cuda3.0 or 3.0	PGIにバンドルされた CUDA toolkit 3.0 バージョンを使用 (PGI 10.4以降)
cuda3.1 or 3.1	PGIにバンドルされたCUDA toolkit 3.1 バージョンを使用 (PGI 10.8以降)
cuda3.1 or 3.1	PGIにバンドルされたCUDA toolkit 3.2 バージョンを使用 (PGI 11.0以降)
cuda4.0 or 4.0	PGIにバンドルされた CUDA toolkit 4.0 バージョンを使用 (PGI 11.6以降)
cuda4.1 or 4.1	PGIにバンドルされた CUDA toolkit 4.1 バージョンを使用 (PGI 12.2以降)
cuda4.2 or 4.2	PGIにバンドルされた CUDA toolkit 4.2 バージョンを使用 (PGI 12.6以降)
cuda5.0 or 5.0	PGIにバンドルされた CUDA toolkit 5.0 バージョンを使用 (PGI 13.1以降)
cuda5.5 or 5.5	PGIにバンドルされた CUDA toolkit 5.5 バージョンを使用 (PGI 13.9以降)
cuda6.0 or 6.0	PGIにバンドルされた CUDA toolkit 6.0 バージョンを使用 (PGI 14.4以降)
cuda6.5 or 6.5	PGIにバンドルされた CUDA toolkit 6.5 バージョンを使用 (PGI 14.9以降)
cuda7.0 or 7.0	PGIにバンドルされた CUDA toolkit 7.0 バージョンを使用 (PGI 15.4以降)
cuda7.5 or 7.5	PGIにバンドルされた CUDA toolkit 7.5 バージョンを使用 (PGI 15.9以降)
cuda8.0 or 8.0	PGIにバンドルされた CUDA toolkit 8.0 バージョンを使用 (PGI 16.10以降)
cuda9.0 or 9.0	PGIにバンドルされた CUDA toolkit 9.0 バージョンを使用 (PGI 17.9以降)
cuda9.1 or 9.1	PGIにバンドルされた CUDA toolkit 9.1 バージョンを使用 (PGI 18.1以降)
cuda9.2 or 9.2	PGIにバンドルされた CUDA toolkit 9.2 バージョンを使用 (PGI 18.5以降)
fastmath	fast mathライブラリを使用 (PGI 10.4以降)
[no]flushz	GPU上の浮動小数点演算の flush-to-zero モードを制御。デフォルトはnoflushz。 (PGI 11.5以降)
keepbin	kernelバイナリファイルを保持し、ファイル(.bin)として出力する
keepgpu	kernelソースファイルを保持し、ファイル(.gpu)として出力する (PGI 10.3新設)
keepptx	GPUコードのためのportable assembly(.ptx)ファイルを保持し、ファイルとして出力する
maxregcount:n	GPU上で使用するレジスタの最大数を指定。ブランクの場合は、制約が無いと解釈する
[no]lineinfo	GPU line informationを生成する(PGI 15.1 以降)

	[no]llvm: 64-bit上ではLLVMバックエンドをデフォルトとして使う [使わない] nofma: fused-multiply-add命令を生成しない (PGI 10.4以降) noL1 noL1cache: グローバル変数をキャッシュするためのハードウェア L1 データキャッシュの使用を抑止する (PGI 13.10以降) loadcache:L1 loadcache:L2: グローバル変数をキャッシュするためのキャッシュを選択する（Kepler K40以降） (PGI 14.4以降) ptxinfo: コンパイル時にPTXAS情報メッセージを表示する(PGI 11.0以降) [no]rdc: Fortran Module 内の device routine など、異なるファイルに配置されたデバイスルーチンをそれぞれ分割コンパイルし、リンクが出来るようにする。CUDA 5.0 以降の機能を使用します。(PGI 13.1以降 + CUDA 5.0 以降)(PGI 14.1 以降デフォルト) [no]unroll: 自動的に最内側ループのアンローリングを行う (default at -O3) (PGI 14.9以降)	
cudalib[=cublas|cufft|curand|cusparse|cusolver]	指定した NVIDIA CUDAライブラリをリンクする。	リンク
cudax86	(PGI 11.5新設、pgc++/pgcpp のみ) CUDA C++ プログラムを PGI C++ コンパイラでコンパイルして、この実行バイナリをインテルやAMDの x86 プロセッサ上で実行できる PGI CUDA C for Multi-core x86 機能を有効にする。	C++言語
[no]daz	IEEE 754 正規化されていない数字（内部表現）に対して、zero セットすることを許可する（しない）オプション。(PGI6.0) このオプションは、メインプログラムに対して適用しなければ有効とならない。 PGI 6.2 以降、64ビット EM64T の場合は、-Mdazがデフォルトとし、AMD64 の場合は、-Mnodaz がデフォルトとなる。	最適化
[no]dclchk	（pgf77、pgfortranとpghpfのみ）全てのプログラム変数が宣言されていることを前提としてチェックする（しない）。	Fortran言語
[no]defaultunit	（pgf77、pgfortranとpghpfのみ）どのようにアステリスクキャラクタ“*”が（I/O ユニット 5 と 6 の状態に関係なく）標準入力、および、標準出力と関連して扱われるかを決定。	Fortran言語
[no]depchk	潜在的なデータ依存性が実際に存在することをコンパイラに指示してチェックを行う。一方、nodepchk は依存性がないことをコンパイラに指示する（もし、この場合、存在した場合は不正確な結果となる）。	最適化
[no]dlines	（pgf77、pgfortranとpghpfのみ）コンパイラが実行可能なステートメントとしてカラム1に“D”を含む行を扱うかどうかを決定。	Fortran言語
dll	（Windows only) ランタイムライブラリの DLL バージョンとリンクする。 (PGI 7.0以降）-Mdll オプションは、-D_DLL 機能を含意します。-D_DLL は、プリプロセッサ・シンボル _DLL を定義するものです。 ((PGI 7.1以降）-Mdll オプションが削除されました。その代わり、-Mdynamic オプションを使用します。	その他
dollar,*char*	コンパイラがドル記号コードをマップする際の文字(*char*)を指定。ドル記号を名前として使用することを許す。ANSI C は許さない。	Fortran言語
[no]dse	【PGI 7.0 以降】 参照しない変数の保存を排除（dead store eliminations）する最適化を有効[無効]にするオプションです。これは、C++プログラムのようなパフォーマンス向上のために、関数呼び出しをインライン化することが多い場合に有効となります。	最適化
dwarf1 | dwarf2 | dwarf3	DWARF1 あるいは DWARF2、DWARF3 フォーマットのいずれかのデバッグ情報を生成する。デフォルトは、 DWARF2 である。-g とともに使用する。	コード生成

nodwar	(PGI 8.0以降）デバッグ情報を生成バイナリに付け加えないように指示する。	コード生成
eh_frame, noeh_frame	(PGI 2010以降）リンカーに、executable内のen_frameのcall frame を保持/非保持することを指示する。 （注意） このオプションは、システムunwindライブラリを持つ、最新のLinux、Windowsシステムでのみ有効です。	コード生成
extend	（pgf77、pgfortranとpghpfのみ）コンパイラは、132 カラムソースコードを受け付けます。デフォルトでは 72 カラムコードを受け付けます。	Fortran言語
extract[=*flag[, flag,...]]*	関数エキストラクタを起動。コマンドライン上で指定されたファイルから関数を抽出し、指定した外部 directory へその関数ファイルを生成、追加する。 インライン（-Minline）とともに使用する場合が多い。以下のフラグがありますので、詳細は User's Guide を参照のこと。 name:func size:number lib:dirname	インライン化
fcon	（pgccとpgCCのみ）浮動小数点定数を倍精度型の代わりに、 float 型として扱うようにコンパイラに指示。	C / C++言語
fixed	（pgfortranとpghpfのみ）F77スタイルの固定フォーマットのソースであると認識する	Fortran言語
[no]flushz	SSE/SSE2 を flush-to-zero モードにセットする。浮動小数点のアンダーフローが生じた場合、これを 0 にセットする。このオプションは、メインプログラムに対して適用しなければ有効とならない。	最適化
free	（pgfortranとpghpfのみ）コンパイラは F90 形式のフリーフォーマットのソースコードであると仮定する。	コード生成
[no]func32	32 Byte 上で全ての関数をアライン（整列）させる。	その他
[no]fpapprox[=div|sqrt|rsqrt]	(PGI 7.1 以降) 特定の単精度浮動小数点演算を低精度近似方を使用して実行します。のオプションは結果の差異が生じる可能性がありますので、十分注意して使用してください。 div：浮動小数点除算近似 sqrt：浮動小数点平方根近似 rsqrt：浮動小数点逆数平方根近似 デフォルトでは、-Mfpapprox は使用されません。もし、サブ・オプションを指定しない -Mfpapprox のみの場合は、上記の全てのサブ・オプションが指定されたものとして扱います。 (PGI 8.0 追加) -Mnofpapprox ：低い精度の浮動小数点演算を使用しないように指示する。	最適化
[no]fpmisalign	(PGI 7.1 以降) AMD barcelonaプロセッサに対して、16-byte境界に整列されていないアドレスを持つメモリ・オペランドのベクトル演算命令の使用を許可します。デフォルト設定は、Barcelonaを含めて、全てのプロセッサにおいて-Mnofpmisalignです。本オプションは、-tp barcelona-64あるいは、-tp barcelonaの設定時、あるいは、barceona上でコンパイルされたときにのみ効果があります。また、このオプションでコンパイルされたコードは、barcelonaプロセッサ上だけで実行できるものとなりますのでご注意下さい。	最適化
[no]fprelaxed= [div,order,rsqrt,sqrt:recip, intrinsic]	いくつかの内部組込み関数 (div/sqrt/rsqrt) の計算において緩い精度で行うことをコンパイラに指示する。性能は向上するが、計算精度は劣る。 (PGI 6.1 以降) デフォルトは、-Mnofprelaxed。 PGI 6.2 以降、細かな制御を行うためのサブオプションを導入。サブオプションは以下のとおりです。 -Mfprelaxed=[div,rsqrt,sqrt] div：緩い精度で除算処理を行う。 order：a*b+a*cをa*(b+c)と変換する方式も含め、演算の順序の変更 noorder：上記 order を行わない。 rsqrt：緩い精度でsqrtの逆数近似（1/sqrt）の処理を行う sqrt：緩い精度でsqrtの処理を行う。 なお、サブオプションを付加しない場合（-Mfprelaxedのみ）は、そのターゲットプロセッサに応じて、顕著な性能向上が行える処理に緩い精度での処理を行うかを選択し適用される。 （PGI 9.0 新設） recip：緩和した精度で逆数近似 （PGI 13.1 新設）	最適化

	intrinsic：緩和した精度の組み込み関数を使用	
[no]i4	（pgf77、pgfortranとpghpfのみ）どのようにコンパイラがINTEGER 変数を扱うかを決定。i4 の場合、INTEGER*4、noi4 の場合は、INTEGER*2 として扱う。	最適化
iface=unix | cref | mixed_str_len_arg | nomixed_str_len_arg	（PGI 7.2 新設　Windowsのみ）サブオプション-MifacはFortranのための呼び出しルール（コンベンション）を調整するものです。 unix (32bit only) - Use UNIX calling conventions＋語末のアンダースコアがあるタイプ cref - Use CREF calling conventions＋ 語末のアンダースコアがないタイプ mixed_str_len_arg -文字列の長さをその対応する引数の直後に置くタイプ nomixed_str_len_arg - 文字列の長さを引数リストの最後に置くタイプ。	Fortran言語
[no]idiom	ループ内でidiom認識（パターン認識）を行う [抑止する] (PGI 15.1以降)	最適化
info[=*flag[,flag,...]*]	コンパイル時に最適化並びにコード生成に関するコンパイル・メッセージを標準出力に表示する。以下のサブ・フラグがありますので、詳細は User's Guide を参照のこと。 all inline ipa loop mp opt time unroll （PGI 7.2新設） intensity –ループ内の「演算密度」(Computational Intensity)を表示します。デフォルトは、最内側ループの情報が表示されます。演算密度とは、一般にループ内の演算数とメモリのロード・ストア数との比率を表し、演算とメモリ参照のバランスを見るための指標です。このような情報はパフォーマンス・チューニングにおいて特に重視されます。 ・ ループ内の演算が浮動小数点演算である場合、演算密度は、浮動小数点演算総数を浮動小数点データのメモリロードとストアの総和で割った比率として定義します。 ・ ループ内の演算が整数演算である場合、演算密度は、整数演算総数を整数データのメモリロードとストアの総和で割った比率として定義します。 （PGI 8.0 以降新設） all　以下のサブオプションをすべて指定したものと解釈します。 -Minfo=accel,inline,ipa,loop,lre,mp,opt,par,unified,vect accel　アクセレータ情報の有効化 ccff　オブジェクトファイルに最適化情報を追加します ftn　Fortran特有な情報の有効化 hpf　HPF特融な情報の有効化 information inline　インライン情報の有効化 lre　LRE情報の有効化 par　並列化の情報の有効化 pfo　プロファイル・フィードバックに関する情報の有効化 vect　ベクトル化の情報の有効化 （PGI 9.0 新設） accel　アクセラレータ領域をGPU Kernel に翻訳することが成功したかどうかの情報を示す	その他
inform*[=level]*	指定した *level* 以上のエラー・メッセージをを表示するように指示。 fatal ： fatal error messages. severe ： severe and fatal error messages. warn ： warning, severe and fatal error messages inform ： all error messages (inform, warn, severe and fatal)	その他
inline [=*func* | *filename.ext* | *number* | *levels:number*],...	関数のインライン展開を行う。以下のサブ・フラグがありますので、詳細は User's Guide を参照のこと。 except:func ： IPA(-Mipa) のインライン機能にも影響する。 [name:]func filename.ext Number	インライン化

	levels:<n> (PGI 17.1廃止) totalsize:<n>、maxsize:<n>　(PGI 17.1以降) smallsize:<n>　(PGI 17.7以降) PGI 7.1以降)　配列の形態(Array shape) が一致しない場合でもFortran におけるインライン処理を許可（抑止）する。-Mconcur あるいは -mp の場合を除いたデフォルトは、-Minline=noreshape。-Mconcur あるいは -mp の場合のデフォルトは、Minline=reshape。	
instrument [=functions]	(PGI 9.0以降、linux86-64 にみ) Common Compiler Feedback Format (CCFF)を使用して、 PGIコンパイラは、どのようにプログラムの最適化を行ったら良いか、あるいは、特定の最適化がなされないのか等の関数レベルのinstrument 情報をオブジェクトに保持することを可能とする。-Minstrument=functions の指定も -Minstrumen tと同じ意味なる。このオプションは、-Minfo=ccff -Mframe の二つを指定したことと同意です。	その他
ipa[=*flags*]	関数、サブルーチン間のグローバルな最適化を行うために、内部手続き間の最適化を行うように指示。version 5.2 から 1パスで行うことが可能。この ipa は同時に -O2 のレベルで行うことを前提にしている。以下のサブ・フラグ *flags* の詳細は、User's Guide を参照のこと。**一般的には、-Mipa=fast を指定すると良い**。 [no]align [no]arg [no]const Interprocedural constant propagation [no]cg except:<func> [no]f90ptr fast force [no]globals inline:<n> inline ipofile [no]keepobj [no]libc [no]libinline [no]libopt [no]localarg main:<func> noerror [no]ptr [no]pure required safe:[<function>|<library>] [no]safeall [no]shape summary [no]vestigial -Mipa Default enables constant propagation **複合フラグ fast の意味は** -Mipa=align,arg,const,f90ptr,shape,globals,localarg,ptr **PGI6.0 New flag:** -Mipa[=…,safe:<libname>,safeall,…] –　IPA機能を使用してコンパイルして生成していない、ライブラリ名 libname の中のプログラムユニットへの呼び出しが安全であると仮定する、あるいは呼び出し側におけるIPA最適化を禁止しないことをコンパイラに指示するためのオプションです。-Mipa=safeall は実行モジュールの中にリンクされる全てのライブラリが安全であることコンパイラに指示します。 **PGI 6.1 New flag:** -Mipa=cg スイッチを設定することで、プログラムのコール・グラフ情報を出力できるようになりました。これは、新規に提供されたpgicgコマンド・ユーティリティを使用して、出力可能です。 -Mipa=except:<func> –　IPA最適化において、インラインすべきでない関数funcを指定します。-Mipa=inlineと共に指定します。デフォルトは、内部的に検出されたすべての関数がインライン対象となります。 **PGI 6.2 New flag:** -Mipa=[no]libc は、システム標準Cライブラリ内で、あるルーチン	最適化

	への呼び出しを最適化するために使用します。-fastオプション時のデフォルトは-Mipa=libcです。 nolibcは、その機能を抑止します。 **PGI 7.1 New flag:** -Mipa=[no]reshape は、配列の形態(Array shape) が一致しない場合でも Fortran におけるインライン処理を許可（抑止）します。 **PGI 7.2 New flag:** -Mipa=jobs:<n>－　jobs:[n] サブオプションを指定できるようになりました。このサブオプションは、並列に n ジョブで再コンパイルを行うように指示するものです。 **PGI 9.0 New flag:** -Mipa=nopfo は、プロファイル・フィードバック情報の引用回数情報を無視する。このサブオプションは、inlineサブオプションの次に指定されているときのみ有効。-Mipa=inline,nopfo　は、IPA手続きに対して、PFO情報が有効な状態において、インラインされる関数を決める際に、PFO情報を無視するように伝えます。 **PGI 13.1 New flag:** -Mipa=reaggregation IPA guided structure reaggregation を行います。自動的に struct の要素の並べ替え、あるいは、メモリやキャッシュの利用向上のために struct を substruct に分離する等の処理を行います。	
noipa	内部手続き間解析と最適化機能を抑制します。機能複合オプションの後に、このオプションを指定した場合、他の機能に関しては影響せず、IPA最適化のみを抑制することができます。(PGI 6.0)	最適化
[no]iomutex	（pgf77、pgfortranとpghpfのみ）クリティカルセクションがFortran I/Oコールの周辺で生成されるかどうかを決定。	Fortran言語
[no]large_arrays	(**64bit環境**,)　配列添え字（インデックス）を 64 ビット整数で扱えるように変更します。この意味は、必要に応じて、64 ビット整数変数あるいは定数が、インデックスの計算において使用されることを意味します。但し、コンパイラが暗黙に 32 ビット整数から 64 ビット整数に変更することによって、思わぬところで副作用が現れるかもしれないことに注意してください。一般に、64 ビット・アドレッシングが必要なインデックス変数は、明示的に 64 ビット整数宣言をすることが最も安全な方法で、これ行ってかつ、このオプションを指定することを推奨します。 さらに、Linux 環境下では 2GB を超える「単一の静的なデータオブジェクト」を扱うことができるコードを生成します。PGI 5.2 の場合は、pgfortran, PGF77, PGCC でサポートします。一般的には、-mcmodel=medeium と同時に使用します。**PG 6.0 以降**では、2GB以上の単一の静的なデータオブジェクトをサポートするために指定する有効化（無効化）フラグです。pgfortran, PGF77, PGCC, PGC++ の言語でサポートします。なお、このオプションは、PGI 6.0 から　Linux の -mcmodel=medium の複合オプションの中に加えられました。2GB以上の単一の静的なデータオブジェクトを使用するアプリケーションでは必要とされるオプションです。	コード生成
largeaddressaware[=no]	(PGI 7.2新設：Windowsのみ)　Windows x64 用に 2GB 以上のアドレス・インデックシングを Windows のリンカーへ指示します。（RIP-relative　addressingを使用する）。デフォルトは、noで、direct addressing 形式となります。	その他
[no]loop32	(PGI 7.1 以降) barcelona上での 32-byte 境界上にある最内側ループを整列します。barcelona上で 32-byte 境界で整列されている場合、小さなループは性能が向上する可能性があります。しかし、実際には、ほとんどのアセンブラが、まだ効果的なパッディング(padding)を実装していません。その結果、このオプションで遅くなる可能性もあります。Barcelonaに対して最適化されたアセンブラを有するシステム上でこのオプションを使用してください。デフォルトは、-Mnoloop32 です。	最適化
lsf	(32-bit Linux) 32ビットシステム上で 2GB 以上のファイル　I/O を扱うためのライブラリをリンクする。	環境
lre[=array | assoc | noassoc] [no]lre	ループ内での冗長性を削除する最適化の有効化 [無効化]。 array　：個々の配列要素の参照を冗長性削減の対象として扱う。デフォルトは、2以上のオペランドを含む冗長式のみが対象となる。 assoc　：冗長性削減の対象を増やすことができる、演算式の再結合を許す最適化。結果の差異が生じる可能性がある。	最適化

<table>
<tr><td></td><td>noassoc：上記を許さない最適化</td><td></td></tr>
<tr><td>keepasm</td><td>アセンブリファイルを保持するようにコンパイラに命令。ファイル名は、<filenema>.s 。</td><td>その他</td></tr>
<tr><td>[no]list</td><td>コンパイラがリスティング・ファイルを作成するかどうかを指定。ファイル名は、<filenema>.lst 。</td><td>その他</td></tr>
<tr><td>[no]m128</td><td>(PGI 9.0 新設 pgcc のみ)
__m128, __m128d, __m128iデータ型を認識するためオプション</td><td>その他</td></tr>
<tr><td>makedll[=export_all]</td><td>(Windows only) Dynamic Link Library (DLL) を生成する。=export_all は、DLL内の全ての関数をエクスポートする。
Winodws 上での DLL の作成に関しては、PGI User's Guide の 8 章を参考のこと。
(PGI 7.1 以降) -Mmakedll オプションは、-Mdynamic オプションを内包します。</td><td>その他</td></tr>
<tr><td>makeimpdll[=export_all]</td><td>(Windows only) DLL を生成することなしに、import ライブラリを生成する。=export_all は、DLL内の全ての関数をエクスポートする。</td><td>その他</td></tr>
<tr><td>makeimplib</td><td>(Windows only：PGI 7.0 以降) DLLを生成することなしに、インポートライブラリを生成します。これは、まだ、それ自身のDLLライブラリが構築される前に、DLLのためにインポートライブラリを生成したい時に使用します。</td><td>その他</td></tr>
<tr><td>mpi[=option]</td><td>(MPI 使用可能ライセンスのみ：PGI 7.1 以降)
プログラムのビルドに使用する MPI ライブラリの指定を行う。
<table>
<tr><th>使用するライブラリ</th><th>コンパイル・リンクに必要なオプション</th></tr>
<tr><td>MPICH1</td><td>-Mmpi=mpich1 は PGI 13.10 以前有効、PGI 14.1以降廃止、必要な場合は、MPIDIR環境変数にそのディレクトリをセット</td></tr>
<tr><td>MPICH2</td><td>-Mmpi=mpich1 は PGI 13.10 以前有効、PGI 14.1以降廃止、必要な場合は、MPIDIR環境変数にそのディレクトリをセット</td></tr>
<tr><td>MPICH v3</td><td>-Mmpi=mpich (PGI 14.1 以降)</td></tr>
<tr><td>MS-MPI</td><td>-Mmpi=msmpi （Windows)</td></tr>
<tr><td>MVAPICH1 (CDK)</td><td>-Mmpi=mvapich1 は PGI 13.10 以前有効、PGI 14.1以降廃止、必要な場合は、MPIDIR環境変数にそのディレクトリをセット</td></tr>
<tr><td>MVAPICH2 (CDK)</td><td>MVAPICH2 の mpif90,mpicc等のラッパーを使用する</td></tr>
<tr><td>Open MPI (CDK)</td><td>Open MPI の mpif90,mpicc等のラッパーを使用する</td></tr>
<tr><td>SGI MPI</td><td>-Mmpi=sgimpi （PGI 13.5 以降）</td></tr>
</table></td><td>コード生成</td></tr>
<tr><td>neginfo[=flags]</td><td>なぜ、最適化が行われないかに関する情報を生成するようにコンパイラに指示。
all : 全てのメッセージ出力
concur ：自動並列化できない理由
loop ：メモリ階層型の最適化ができない理由
(PGI 8.0 以降)
all 以下のサブオプションをすべて指定したものと解釈します。
-Mneginfo=accel,inline,ipa,loop,lre,mp,opt,par,vect
accel アクセレータ情報の有効化
ftn Fortran特有な情報の有効化
hpf HPF特有な情報の有効化 information
inline インライン情報の有効化
ipa IPA i情報の有効化
lre LRE情報の有効化
mp OpenMP情報の有効化
opt 最適化の情報の有効化
par 並列化の情報の有効化
pfo プロファイル・フィードバックに関する情報の有効化
vect ベクトル化の情報の有効化</td><td>その他</td></tr>
</table>

names=lowercase|uppercase	Fortran外部関数名の大文字/小文字を指定する。Lowercaseの場合は、小文字を使用すると言う意味となり、upperecaseは大文字を使用すると言う意味となる。（PGI 7.2新設）	その他
noframe	関数の真のスタック・フレームポインタのセットアップ処理を消去するように指示。このオプションを有効化すると traceback 機能を使用することができない。	最適化
nomain	（pgf77、pgfortranとpghpfのみ）リンクステップ時に、Fortran のメインプログラムを呼ぶオブジェクトファイルを含めない形でリンクする。C プログラムと Fortran プログラムのオブジェクトの混在したものをリンクする時、C プログラムにメインプログラムが存在している場合で、かつ pgf77, pgfortran でリンクする時に使用する。	コード生成
[no]movnt	non-temporal ストア並びにプリフェッチの生成を強制するオプション。これまで使用してきた -Mnontemporal を置き換えるものです。(PGI 6.1)	コード生成
nontemporal	non-temporal ストア並びにプリフェッチの生成を強制するオプション。	最適化
noopenmp	-mp オプションと同時に使用した場合、強制的に OpenMP directives を無視するようにコンパイラに指示する。但し、SGI スタイルの並列 directive は解釈する。	その他
[no]prefetch　（PGI5.2まで） [no]prefetch [=d:<m>[,n:<p>[,{nta | t0 |w}]]] (PGI6.0以降)	prefetch インストラクションの生成を有効化/無効化する。-Mvect（−fastsse) オプションと共に使用する。 **PG 6.0 以降**では、メモリデータのプリフェッチ命令を生成することを有効化（無効化）します。このオプションは、-Mvect あるいは、-Mfastsse（-Mvectを含む複合オプション）と組み合わせて使用する場合のみ有効です。新たなサブオプションである、d:<m> **距離サブフラグ**は、アクセスしようとするデータの先にある m キャッシュラインの長さをプリフェッチするようにコンパイラに指示します。n:<p> **数サブフラグ**は、プリフェッチが使用されている場所において、最大 p プリフェッチ命令まで出すことができるようにコンパイラに指示するものです。また、新たな nta | t0 | w の各サブオプションは、プリフェッチのために、prefetchnta、prefetch0、prefetchw 命令を使うようにコンパイラに指示するものです。なお、prefetchw は、IA32 あるいは EM64T プロセッサではサポートしません。	最適化
nopgdllmain	(Windows only) デフォルトの DllMain() を DLL の中に含んでいるモジュールをリンクしない。このフラグは、pgfortran による DLL の構築に対して適用される。	その他
norpath	(Linux only) PGI の 共有ライブラリ・オブジェクトを含むディレクトリパス名を -rpath オプションをリンク時のコマンド行に付加しない。（デフォルトは -Mrpath で含む）	その他
nosgimp	-mp オプションと同時に使用した場合、強制的に SGI スタイルの並列 directive を無視するようにコンパイラに指示する。但し、OpenMP directives は解釈する。	その他
nostartup	（pgf77、pgfortranとpghpfのみ）標準のスタートアップルーチンをリンクしない。	環境
nostddef	標準のプリプロセッサマクロを認識しないようにコンパイラに指示。	環境
nostdinc	インクルードファイルの標準の場所を検索しないようにコンパイラに指示。	環境
nostdlib	標準のライブラリをリンクしないようにリンカに指示。	環境
[no]onetrip	（pgf77、pgfortranとpghpfのみ）各 DOループが少なくとも 1 回実行させるかさせないかの指示。	言語
novintr	イディオム認識を抑制し、最適化されたベクトル関数の呼び出しを行う。	最適化
pfi	-Mpfo 最適化オプションを含む後続のコンパイル時において使用されるプロファイルとデータ・フィードバック情報を集めるための実行モジュールを生成するためのオプションです。-Mpfi を伴う実行モジュールはこの情報を集積するためのオーバヘッドが発生するため、実行時間が多く掛かります。(PGI 6.0) （PGI 7.2 新設） -Mpfi[=indirect] -Mpfiオプションは、間接的(indirect)な関数呼び出しターゲットを保持することを指示する	最適化

<table>
<tr><td>pfo</td><td>強化されたブロック・リオーダリング機能を含む特定の性能最適化を有効にするために、pgfi.outプロファイル・フィードバック・トレースファイルのデータを使用して最適化を行います。(PGI 6.0)
（PGI 7.2 新設）
-Mpfo[=indirect|nolayout]　Indirect サブオプションは、間接的な関数呼び出しのインライン化を有効にするもので、nolayout は、動的なコード配置を抑止する</td><td>最適化</td></tr>
<tr><td>pre[=all]、nopre</td><td>（PGI 7.2新設）サブオプションを付けない -Mpre オプションは、一部の冗長部削除を有効にする。サブオプション all を付けた場合、よりアグレッシブな pre 処理を行う。
（PGI 9.0 以降）
=all サブオプションが廃止された。
(PGI 2010以降)
-Mnopre 冗長部削除の最適化を抑止する。</td><td>最適化</td></tr>
<tr><td>preprocess</td><td>cpp 形式の前処理をアセンブラ言語と Fortran ソースファイル上で行う。
代わりとして -cpp オプションを新設 （PGI 17.1）</td><td>その他</td></tr>
<tr><td>prof[=flags[,flags,.]]</td><td>プロファイルオプションをセット。関数レベルと、行レベルのプロファイリングがサポートされます。-Mprof=func、あるいは -Mprof=lines を指定する。これは、リンク時にも指定が必要である（特に Makefile 等でコンパイルとリンク処理を別々に行う際に注意）
PGI 6.0 New feature:
プロファイル・オプションをセットします。-ql, -qp, -pg スイッチは、通常、プロファイルのために使用されますが、プロファイリングのデフォルトの方法を再セット（上書き）するために以下のオプションを指定します。詳細は、PGI User’s Guide をご覧ください。
dwarf：サードパーティのプロファイリング・ツールによって、ソース相関を有効にするため、DWARF 情報を生成する。
func：PGI スタイルの関数レベルのプロファイリングを実行する
hwcts：ハードウェア・カウンタを用いた PAPI ベースのプロファイリングを使用する場合に指定する （linux x86-64ベースのシステムのみ）
lines：PGI スタイルのソースレベルのプロファイリングを実行する
time：サンプリングベースのインストラクション・ベースのプロファイリングを実行
（PGI 7.1 以降）
プロファイルするアプリケーションにリンクするための MPI ライブラリ名を、-Mprof オプションに指定する。

<table>
<tr><th>使用するライブラリ</th><th>コンパイル・リンクに必要なオプション</th></tr>
<tr><td>MPICH1</td><td>-Mmpi=mpich1 は　PGI 13.10 以前有効、PGI 14.1以降廃止、-Mprof=mpich1,{func|lines|time}</td></tr>
<tr><td>MPICH2</td><td>-Mmpi=mpich1 は　PGI 13.10 以前有効、PGI 14.1以降廃止、-Mprof =mpich2,{func|lines|time}</td></tr>
<tr><td>MPICH v3</td><td>-Mprof=mpich,{func|lines|time} (PGI 14.1 以降)</td></tr>
<tr><td>MS-MPI</td><td>-Mprof =msmpi,{func|lines}　（Windows）</td></tr>
<tr><td>MVAPICH1 (CDK)</td><td>-Mmpi=mvapich1 は　PGI 13.10 以前有効、PGI 14.1以降廃止、-Mprof=mvapich1,{func|lines|time}</td></tr>
<tr><td>MVAPICH2 (CDK)</td><td>MVAPICH2 の mpif90,mpicc等のラッパーを使用する
-profile={profcc|proffer}
-Mprof ={func|lines|time}</td></tr>
<tr><td>Open MPI (CDK)</td><td>Open MPI の mpif90,mpicc等のラッパーを使用する
-Mprof ={func|lines|time}</td></tr>
</table>
</td><td>コード生成</td></tr>
</table>

	SGI MPI | -Mprof =sgimpi,{func|lines|time}（PGI 13.5 以降） （PGI 8.0以降） [no]ccff：CCFF情報の有効化 [無効化]	
[no]propcond	(PGI 7.1 新設) equality conditionalsから派生するassertions からのconstant propagation 最適化を有効にします。これは、デフォルトで有効となります。	最適化
[no]r8	（pgf77、pgfortranとpghpfのみ）コンパイラが REAL 変数と定数をDOUBLE PRECISION に変換する（しない）。	最適化
[no]r8intrinsics	（pgf77、pgfortranとpghpfのみ）コンパイラが 組込み関数の CMPLX and　REAL 型を DCMPLX and DBLEとして扱す（扱わない）	最適化
[no]recursive	（pgf77、pgfortranとpghpfのみ）ローカル変数をスタックに割当てます（割当てません）。これは再帰を可能にします。SAVEされた、データ初期化された、または、namelist メンバは、このスイッチの設定に関係なく常にスタティックに割当てられます。	コード生成
[no]reentrant	コンパイラがコードをリエントラントとしない最適化を回避するかどうかを指定。	コード生成
[no]ref_externals	（pgf77、pgfortranとpghpfのみ） EXTERNAL 文に現れる名前の参照を強制（強制しない）。	コード生成
safeptr[=*flags*]	（pgcc と pgCC のみ）ポインタと配列の間のデータ依存関係を以下のサブフラグの内容でオーバライドするようにコンパイラに指示します。以下のサブ・フラグ *flags* の詳細は、User's Guide を参照のこと。 all　all is safe arg　Argument pointers are safe auto　Local pointers are safe dummy　Argument pointers are safe local　Local pointers are safe static　Static local pointers are safe global Global pointers are safe -Msafeptr All pointers are safe	最適化
safe_lastval	スカラがループの後で使用され、しかし、ループの全ての反復に関しては定義されない場合、コンパイラはデフォルトではループを並列化しません。しかし、このオプションは、コンパイラにループを並列化することが安全であると告げます。特定のループについて、全てのスカラの最後に計算された値がループの並列化を安全にします。	コード生成
[no]**save**	（pgf77、pgfortranとpghpfのみ）コンパイラが全てのローカルな変数がSAVEステートメントと同等な状況に強いるように仮定するかどうかを決定。	Fortran言語
[no]**scalarsse**	スカラの浮動小数点演算において、xmm レジスタを使用した SSE/SSE2 のインストラクションを使用するか否かを指示。このオプションは、-tp { p7 / p7-64 / k8-32 / k8-64 以降の target } 時に有効。	最適化
scalapack	(PGI 14.1 以降 Linux / OS X 版のみ）バンドルされた MPICH 3.0.4 と共に使用し、分散メモリ用の LAPACK ライブラリ（ScaLapak) のリンクを有効にする。	ライブラリ
schar	（pgccとpgc++） "plain" character を signed char ととして扱う。--uchar を参照。	C / C++言語
[no]second_underscore	（pgf77、pgfortranとpghpfのみ） Fortran のグローバルなシンボル名が、既にその名前の suffix にアンダースコアを有しているものが存在している場合に、内部シンボル名として 2 つめのアンダースコアを加えます（加えません）。Fortran Module 間のシンボル名の競合が起きる場合にも便利です。また、g77プログラムとのリンク時に有効です。	コード生成
[no]signextend	コンパイラがサインビットを拡張するかどうかを指定します。	コード生成
[no]single	（pgccとpgc++） float パラメータを double パラメータキャラクタに変換するかどうかを指示。	C / C++言語

nosizelimit sizelimit:n	ベクトライザにループ中のステートメント数に拘らず、全てのループに対してベクトル化最適化の対象とするように指示する。 PGI 6.2 から nosizelimit が、デフォルトとなった。一方、そのステートメントのサイズは、-Mvect=sizelimit:n （nはループ内のステートメントの数）によって制限される。	最適化
[no]smart	AMD64専用　post-pass instruction スケジューリングを行うか否かのスイッチ。(デフォルトは no)	最適化
[no]smartalloc[=option]	メインルーチン中に最適化された mallopt ルーチンのコールを加えます。これを有効にするためには、Fortran、C、C++のメインプログラムを含むファイルをコンパイルする際に、このオプションを付する必要がある。デフォルトは、-Mnosmartalloc。（PGI 6.2 以降） **PGI 7.1 New feature:** -Msmartalloc オプションは、Linux 並びに Windows 上での large TLBs をサポートするために強化されました。このオプションは、最適な malloc ルーチンを有効にするために、メイン・プログラムをコンパイルする際に使用することが必要です。サブ・オプション huge は、シングルプログラムで使用される大きな 2MB ページを有効にするために指定します。これは、実行するために必要な TLB エントリ数を削減する効果があります。このオプションは、AMDの Barcelona やインテル(r)の Core2 システムで特に有効です。古いプロセッサ・アーキテクチャでは、TLB エントリの数が少ないため、大きな効果は期待できない可能性もあります。サポートするサブ・オプションは、以下のとおりです。 　huge　　：huge page のランタイムライブラリをリンクします。 　huge:<n>　：使用されるページの数の限度を n に設定します。 　hugebss　：huge page の中に BSS セクションを置きます。 huge　サブ・オプションは、それ自身、必要とされる huge page をアロケートしようとします。Huge page の数は、:n サブ・オプションで制限を設けることができ、あるいは、環境変数 PGI_HUGE_SIZE でも設定できます。Hugebss は、プログラムの初期化されていないデータセクションを huge page の中に置きます。 (PGI 8.0 以降) 　hugebss　：　hugeページの中にBSSセクションを置く (PGI 9.0 新設) 　nohuge：　-Msmartalloc=hugeを上書き（無効化）するサブオプション	環境
[no]stack_arrays	自動配列(Automatic Array)をスタック上に配置します。(PGI 13.1 以降) -Mnostack_arrays は、従来通り自動配列をヒープ上に配置します。従来からの互換性を維持するために-Mnostack_arrays がデフォルトです。	Fortran言語
standard	（pgf77、pgfortranとpghpfのみ） ANSI 標準に適合しないソースコードを検出します。 （PGI 7.1 以降）　-Mstandard は、-Mbackslash を内包しました。これは、-Mstandard が現れたときにバックスラッシュ・エスケープ・シーケンスを認識することを禁止します。例えば、バックスラッシュは標準的なキャラクタとして扱います。	Fortran言語
[no]stride0	（pgf77、pgfortranとpghpfのみ）コンパイラは、増分がゼロであるかもしれない誘導変数を含むループのために代替のコードを生成します（生成しません）。	コード生成
summary[=<file>]	プログラム解析ツールのためのl プログラム分析サマリ出力（PASO）を指定されたfile ファイルに記録する。（PGI 17.7以降）	その他
[no]traceback	環境変数 $PGI_TERMを使用することにより、ランタイム traceback のためにデバッグ情報が追加されました。また、デフォルトでのトレースバック機能は、f77、f90/f95では有効となっておりますが、C/C++では無効となっています。コンパイラへの初期設定ファイル siterc あるいは、.mypg*rc ファイルに TRACEBACK=OFF をセットすることで、デフォルトのレースバック機能を無効にすることができます。反対に OFF の代わりに ON と指定することによって、有効にすることができます。	最適化制御

uchar	（pgccとpgc++） "plain character" を unsigned char として扱う。-- scharも参照。	C/C++言語
unix	（pgf77、pgfortran for Win32) Fortran サブプログラムに対して、UNIX の呼び出し、名前のコンベンションを使用することを指示。	コード生成
[no]unixlogical	（pgf77、pgfortranとpghpfのみ）論理値 .TRUE. と .FALSE. が、unixlogical 非ゼロ（TRUE）、ゼロ（FALSE）と決定されるかどうかを決定します。デフォルトの unixlogical では、none-zero 値がTRUE で、0 の値が FALSE です。nounixlogical は、VMS convention スタイルを使用する。	Fortran言語
[no]**unroll**[=*flags*]	アンロール展開を制御。-Munroll=flags と言う形態でサブフラグを設定できる。以下のサブ・フラグ *flags* の詳細は、User's Guide を参照のこと。 c：***m*** n：***u*** **PGI 7.1 New feature:** -M[no]unroll[=c:<n>|n:<n>|m:<n>] ： マルチ・ブロックを持ったループをアンロールする機能を追加しました。特に、条件文を伴ったこのようなループで、アンロールできるようになりました。新しいオプション -Munroll="m" は、この機能を制御するために導入されました。 n:<n>：シングル・ブロックを n 回アンロール m:<n>：マルチ・ブロックを n 回アンロール デフォルトでは、-Munroll=m は有効となっておりません。また、-Munroll=m の場合のデフォルトの n 値は 4 です。	最適化
[no]upcase	（pgf77、pgfortranとpghpfのみ）コンパイラがプログラム識別子に大文字を許すかどうかを決定します。upcase の場合、大文字も識別されます。デフォルトは、noupcase で全てが小文字として識別されます。特に、リンク時のモジュール名の識別において重要です。	Fortran言語
unsafe_par_align	並列化ループでの配列の参照において、その配列の最初の要素が「整列」されている限り、「整列移動（aligned moves)」を行うことは安全であるとみなします。 NOTE: このオプションは、コンパイラがその安全性を疑った場合でも、「整列移動」で行うコードを生成します。このオプションは、特に、 STREAM Benchmark やメモリ・インテンシブなメモリアクセスを含むループブロックの並列化で効果を発揮します。	最適化
vect	コードベクタライザを起動。プログラムのベクトル化を行います。-Mvect の指示だけでも良い。以下のサブ・フラグ *flags* の詳細は、User's Guide を参照のこと。 altcode:***n*** / noaltcode ： 代替スカラコードの生成 assoc / noassoc ： ループの結合の許可 cachesize:***n*** ： cache tileing の最適化における cache size の仮定 nosizelimit ： 全てのループに対して、そのソースコード数の制限なしで、ベクトル化の適用を行うように指示する diom / noidiom ：イディオム認識の許可 levels:<n> ：最適化対象とするループネストの最大数 nocond ：条件文を有するループへのベクトル化の抑止 prefetch ： ベクトル可能なコードの可能な限りの prefetch操作 smallvect[:n] ： 最大のベクトル長の定義 sse ： SSE/SSE2 インストラクションの使用によるベクトル化 tile / notile ：ループタイリングの有効化、無効化 (PGI 7.1 以降) gether ： 配列のgather（ギャザー）間接参照を有するループのベクトル化ができるようになりました。コンパイラのデフォルトは、-Mvect=gatherです。 (PGI 7.2 新設) partial ： 最内側ループの分離によるループのベクトル化を有効にするように指示するサブオプション （PGI 8.0 以降） [no]short ： 短いベクトル演算を有効化[無効化] -Mvect=short は、ループ外のスカラコードから生じる、あるいは、ループ・イテレーションの中から生じる短ベクトル演算のためのパックSSE演算の生成を有効化します。 (PGI 11.6 新設) simd:{128|256} ： SIMD命令とデータを使用してベクトル化する際、そのデータ幅を 128bit / 256bit のどちらを使用するかを	最適化

	選択する。256bit を使用できるかはプロセッサに依存する。 (PGI 18.1以降) simd:{128|256|512}　：512bit-SIMD命令を使用するオプション追加 [no]simdresidual ：ベクトル化されたループの残り部分のベクトル化を行うかどうか	
novector	ベクトル化を抑制します。-fastsse のような機能複合オプションの後に、このオプションを指定した場合、他の機能に関しては影響せず、ベクトル化のみを抑制することができます。(PGI 6.0)	最適化
novintr	コンパイラに、イディオム認識を実行しないように指示する、あるいは、手製の最適化ベクトル関数を導入することを指示する。	最適化
varargs	(pgf77 と pgfortran のみ) Fortran ユニットに対して、C ルーチンが vararg 型のインタフェースを有すると仮定する場合に指定します。	コード生成
writable-strings	(pgcc/pgc++/pgCC: PGI 7.2新設）書き込み可能なデータセグメント内に　string constant　をストアできるようにします。（注意）既存の-Xt並びに-Xsは、本オプションを含む	

-C と C++ 特有のオプション

オプション	記述
-alias=[ansi|traditional]	(PGI 7.1 以降) C、C++プログラムにおける、「型」ベースのポインタ・エイリアス規則に基づき、最適化方法を選択します。 ansi：ANSI C型ベースのポインタの一義化(disambiguation)を使用した最適化を有効化 traditional：型ベースのポインター義化を無効にする C コンパイラでは、デフォルトは -alias=ansi で、C++ においては、-alias=traditionalとしています。
-A	(pgc++) プログラムが Proposed ANSI C++ に合致していることを指示する
--no_alternative_tokens	(pgc++) 代替トークンの認識を Enable/disable する。 These are tokens that make it possible to write C++　without the use of the , , [,], #, &, and ^ and characters. The alternative tokens include the operator keywords (e.g., and, bitand, etc.) and digraphs. デフォルトは、..no_alternative_tokens.
-B	C ソース内における // を使用したC++ 形式のコメントを許可する。
-b	(pgc++) cfront2.1 互換でコンパイルを行う
-b3	(pgc++) cfront3.0 互換でコンパイルを行う。 See -babove.
-c11	C11言語を使用する (PGI 15.1以降)
-c1x	C11言語を使用する (PGI 15.1以降)
-c89	(pgccのみ） C ソース言語として、 C89 standard (C89) を使用する（PGI 6.2 以降）　PGI 6.1 以前のデフォルト
-c8x	(pgccのみ）　-c89 と同じ機能
-c99	(pgccのみ） C ソース言語として、 C99 standard (C99) を使用する（PGI 6.2 以降 のデフォルト）
-c9x	(pgccのみ） -c99 と同じ機能
--c++11	(pgc++のみ） C++11 言語を認識する(-std=c++11と同じ)
-[no]compress_names	(PGI 7.1 以降）　C++ マングル名を 1024 キャラクタにフィットするように圧縮します。高度にネストされたテンプレート・パラメータは、非常時長い関数名が作成されるようになります。これらの長い名前は、古いアセンブラでは問題を引き起こす原因になります。現在のデフォルトは、-no_compress_names です。全ての C++ユーザコードは、このスイッチを使用する際に、再度コンパイルされなければなりません。PGI によって提供されるライブラリは、-compress_namesと共に動作します。
--[no]bool	(pgc++) bool の認識をするかどうかを指示する。デフォルトは、 --bool.
--[no]builtin	数学関数ルーチンをビルトインでコンパイルするかどうかを指示する。選択された数学ライブラリルーチンをコンパイル時にインライン化する。デフォルトは、--builtin.　コンパイルオプションは -M[no]builtin

--cfron t _2 . 1	(pgc++) cfront version 2.1互換でコンパイルするかどうかを指示する。
--cfront_3.0	(pgc++) cfront version 3.0 互換でコンパイルするかどうかを指示する。
--c++[arg]	(PGI 2013以降、 pgc++のみ)C++ の規格を指定する。c++**14**、c++**11**、C++**0x**、C++**03**のいずれかを指定。例えば、c++11 は C++11 機能を解釈する。
--create_pch filename	(pgc++) filename を伴った プリコンパイルされたヘッダーファイルを生成する。
--dependencies	(pgc++)makefile 依存性を標準出力に出力する　(-M を参照）。
--dependencies_to_ file *filename*	(pgc++)makefile 依存性を *filename* ファイルに出力する。
--diag_error tag	(pgc++)指定されたダイアグ・メッセージの標準的なエラーレベルの内容を tag を使用して上書きする。
--diag_remark tag	(pgc++) 指定されたダイアグ・メッセージの標準的なエラーレベルの内容を tag を使用して上書きする。
--diag_suppress tag	(pgc++) 指定されたダイアグ・メッセージの標準的なエラーレベルの内容を tag を使用して上書きする。
--diag_warning tag	(pgc++) 指定されたダイアグ・メッセージの標準的なエラーレベルの内容を tag を使用して上書きする。
--display_error_number	(pgc++) 生成されたダイアグ・メッセージの中にエラーメッセージ番号を表示する。
--enumber	(pgc++) C++ front-end error の数の上限を指定した数にセットする。
--[no_]exceptions	(pgc++) Disable/enable例外処理のサポートを許可するかどうかを指示する。 デフォルトは、 --exceptions　このオプションは PGI 17.1 以降廃止
--gnu	(pgc++、PGI 2013以降) GNU 互換 C++ コンパイルモード。GNU C++ コンパイラとの互換性を維持するために、gnu ライブラリをリンクする。PGI 13.1 より、この機能を提供するために GNU互換 C++コンパイラ（コマンド名 pgc++)も別提供した。 PGI 17.1 以降廃止
--gnu_version	(pgc++、PGI 2010以降) コンパイル時に使用するGNU C++互換性をセットする。デフォルトは、最新のバージョン番号がセットされる。使用例　gnu 4.8.2 の場合： --gnu_version 040802。
--gnu_extensions	(pgc++) Linux system header files をコンパイルする必要がある “include next” のような GNU 拡張を許す。
--instantiation_dir	(pgc++) If --one_instantiation_per_object is used, define dirnameas the instantiation directory.
--[no]llalign	(pgc++) 整数の境界で、 long long integersの整列を行うかどうかを指示する。デフォルトは --llalign.
-M	make 依存性リストを生成する。
-MD	make 依存性リストを生成する。
-MD,filename	(pgc++) make 依存性リストを生成して、それらを filename へ出力する。
--microsoft_version	(pgCCのみ、PGI 2010以降) コンパイル時に使用するMicrosoft C++互換性をセットする。デフォルトは、最新のバージョン番号がセットされる。使用例： --microsoft_version 1.5。
--one_instantiation_per_object	(pgc++) 各 template instantiation (function or static data member) を個々のオブジェクトファイル上に置く。
--optk_allow_dollar_in_id_chars	(pgc++) 識別子としてドル記号を許す。
--pch	(pgc++) 自動的にプリコンパイルされたヘッダファイルを使用する、あるいは生成することを指示する。
--pch_dir directoryname	(pgc++) プリコンパイルされたヘッダファイルの置かれたディレクトリをサーチパスに加える。
--[no_]pch_messages	(pgc++) 現在のコンパイルフェーズで、プリコンパイルされたヘッダファイルが生成され/使用されたかと言うメッセージを表示するかどうかを指示する。
--pedantic	(pgc++)　PGI 8.0 新設　含まれたシステムヘッダーファイルに関わる警告メッセージを印刷
+p	(pgc++) 全ての anachronistic construct を抑制する。
-P	プリプロセスフェーズの後で止まり、プリプロセスされたファイルを filename.i　にセーブ。

--preinclude=<filename>	(pgc++) コンパイル時の始めにインクルードされるファイルの名前を指定する。このオプションは、システム依存のマクロ、型をセットする時に使われる。
--prelink_objects	(pgc++) このオプションが指定された場合、テンプレート・ライブラリにしようとするオブジェクト・セットのために、template instantiations を作成する。
-**std=c++11**	(pgc++のみ) C++11 言語を認識する(--c++11と同じ)
-t [arg]	テンプレート関数の instantiation を制御する。[arg] は以下の引数が存在する。 all local none used
--use_pch filename	(pgc++) 現在のコンパイルフェーズで、指定された名前のプリコンパイルされたヘッダファイルを使用する。
--[no_]using_std	(pgc++) 標準ヘッダーファイルがインクルードされた時に、std namespace の使用を暗黙に使用するか否かを指示する。
-X	(pgc++) クロス・リファレンス情報を生成し、指定されたファイルに書き込む。
-Xm	(pgc++) 名前として $ を許す。 PGI 7.1 以降、このオプションは削除されました。現在、ほとんどの場合、ドルサインは許可されております。
-Xs	(PGI 7.1以降) C/C++ において、レガシーな標準モードを使用する。 これは、-alias=traditional オプションを内包します。
-Xt	(PGI 7.1以降) C/C++ において、レガシーな移行モードを使用する。 これは、-alias=traditional オプションを内包します。
-xh	(pgc++) 例外処理を enable にする。
--zc_eh	(PGI 7.1以降) ゼロ・オーバーヘッド例外領域を生成します。本オプションは、実際の例外処理が起こるまで、例外ハンドリングのコストを遅らせる措置を行います。多くの例外領域を有しながら、あまり例外が起こらないプログラムでは、このためのコンパイル・オプションによって、ランタイム性能の向上に繋がるかもしれません。デフォルトは、--zc_eh を使用しませんが、その代わり、setjmp と longjmp と共に例外ハンドリングを実装する -sjlj_eh を使用します。このオプションは、PGI C++ の以前のバージョンでコンパイルされた C++ コードにも互換性があります。--zx_eh オプションは、libgcc_eh 内のシステム unwind ラ イブラリを提供している新しい Linux システムと Windows 上でのみ有効です。 このオプションは、PGI 2011(11.0)以降、C++コンパイラのデフォルトとなりました。 PGI 17.1 以降廃止
-*suffix* (see -P)	(pgc++) -E、-F、-P の機能で指定された中間ファイルをセーブする。

PGI 6.0 以降でのC++のテンプレートのインスタント化の変更について

C++ テンプレートのインスタント化は、32-bit 並びに 64-bit Linux システムにおいて変更されました。新しい方法では、全てのテンプレート参照を解決するためにGNUリンカーを使用し、重複を回避することでテンプレートの使用の単純化に大きな効果を発揮します。この新しい方法は、PGI コンパイラの前のバージョンと互換性はありません。 C++ プログラムを PGC++ 6.0 用に移行するためには、全ての C++ プログラムを再コンパイルすることが必要です、また、makefile 上の全てのテンプレート・インスタント化フラグを削除することが必要です。次のテンプレートのインスタント化に関係するコマンドオプションが削除する対象となります。

-one_instantiation_per_object
-instantiation_dir
-instantiate
-[no]auto_instantiation
-prelink_objects
-Wc, -tlocal
-Wc, -tused
-Wc, -tall