NEC

輸出する際の注意事項

本製品（ソフトウェアを含む）は、外国為替および外国貿易法で規定される規制貨物（または役務）に該当することがあります。

その場合、日本国外へ輸出する場合には日本国政府の輸出許可が必要です。

なお、輸出許可申請手続きにあたり資料等が必要な場合には、お買い上げの販売店またはお近くの当社営業拠点にご相談ください。

は し が き

本書は、Vector Engine 用の NEC C/C++コンパイラの使用法について説明したものです。

備考

(1) 本書は 2022 年 6 月発行の第 27 版です。

(2) NEC C/C++コンパイラは、以下の言語仕様標準に準拠しています。

- ISO/IEC 9899:2011 Programming languages - C
- ISO/IEC 14882:2014 Programming languages - C++
- ISO/IEC 14882:2017 Programming languages - C++
- OpenMP Application Program Interface Version 4.5

(3) ISO/IEC 14882:2020 Programming languages – C++、OpenMP Application Program Interface Version 5.0 の一部機能にも対応しています。

(4) 本書では Vector Engine を略して VE と表記しています。

(5) 本書の読者は、Linux 上での Fortran/C/C++言語でのソフトウェア開発の知識を有していることを前提としています。

(6) 本書内の製品名、ブランド名、社名などは、一般に各社の表示、商標または登録商標です。

(7) 本製品の一部で、Apache License version 2.0 with LLVM Exceptions の製品を使用しています。

目　次

第1章　C/C++コンパイラ

1.1　概要

NEC C/C++コンパイラは、C/C++プログラムをコンパイル・リンクし、VEのCPUで実行するためのバイナリを作成するコンパイラです。本コンパイラは、VEのハードウェア性能を限界まで容易に引き出せるよう、次の最適化機能を実装しています。

- 自動ベクトル化機能
- 自動並列化機能、OpenMP並列化機能
- 自動インライン展開機能
- 性能情報取得機能

各種コンパイラオプションの指定により、これらの機能を選択しながら、その能力を最大限に利用することができます。最適化機能の詳細及びコンパイラオプションについては、2章以降を参照してください。

1.2　コンパイラの起動

(1)　環境変数の設定

NEC C/C++コンパイラ起動時にパスの指定を省略したいときに、環境変数**PATH**を設定してください。NEC C/C++コンパイラは、標準で/opt/nec/ve配下にインストールされます。環境変数**PATH**に/opt/nec/ve/binを追加してください。

NEC C/C++コンパイラには、ヘッダファイルやライブラリなどのパスを設定する環境変数が用意されていますが、NEC C/C++コンパイラは、既定のパスを自動的にサーチするため、これらの環境変数を設定しなくてもご利用いただけます。環境変数は、コンパイラに付属しないOSSのヘッダやライブラリのパスを常に追加したい場合など、標準以外のディレクトリをサーチする必要があるときに設定してください。

環境変数については、「2.2 実行時に参照される環境変数」を参照してください。

(2)　起動例

コンパイラの起動例を説明します。指定しているコンパイラオプションの詳細については、「第3章 コンパイラオプション」を参照してください。

- Cソースファイル(a.c)のコンパイル、リンク

```
$ ncc a.c
```

- 複数のソースファイルのコンパイル、リンク

```
$ ncc a.c b.c
```

- 作成する実行ファイルの名前(prog.out)を指定するコンパイル、リンク

```
$ ncc -o prog.out a.c
```

- 最大限のベクトル化、最適化を行うコンパイル、リンク

```
$ ncc -O4 a.c
```

- 副作用を伴わないレベルでベクトル化、最適化を行うコンパイル、リンク

```
$ ncc -O1 a.c
```

- ベクトル化、最適化を行わないコンパイル、リンク

```
$ ncc -O0 a.c
```

- 自動並列化を行うコンパイル、リンク

```
$ ncc -mparallel a.c
```

- 自動インライン展開を行うコンパイル、リンク

```
$ ncc -finline-functions a.c
```

- 特定のバージョンのコンパイラを使用してコンパイル、リンク

```
$ /opt/nec/ve/bin/ncc-X.X.X a.c（X.X.Xはコンパイラのバージョン番号）
```

1.3　プログラムの実行

プログラムを実行するときの例を以下に示します。

- プログラムを実行

```
$ ./a.out
```

- 利用するVEを指定してプログラムを実行

```
$ env VE_NODE_NUMBER=1 ./a.out　　（ノード番号1のVEで実行）
```

- 入力ファイル、入力パラメタを指定して実行

```
$ ./a.out data1.in 10                    （ファイルdata1.in、値10を入力）
```

- 入力ファイルをリダイレクトして実行

```
$ ./a.out < data2.in
```

- スレッド数を指定して並列プログラムを実行

```
$ ncc -mparallel -O3 a.c b.c
$ export OMP_NUM_THREADS=4
$ ./a.out
```

- プロファイラ(ngprof)の使用
  コンパイル、リンク時に**-pg**を指定し、コンパイルしたプログラムを実行すると性能測定結果がファイルgmon.outに出力されます。gmon.outをngprofコマンドで解析、表示できます。

```
$ ncc -pg a.c
$ ./a.out
$ ls gmon.out
gmon.out
$ ngprof
（性能情報が表示される）
```

1.4　コマンドラインの指定形式

コンパイラのコマンドラインの指定形式は以下のとおりです。

```
ncc   [ コンパイラオプション | ファイル ] ...
nc++  [ コンパイラオプション | ファイル ] ...
```

1.5　コンパイラオプションの指定

- コンパイラオプションはハイフン(-)に続けて指定します。複数のコンパイラオプションを指定するとき、空白で区切って指定します。

  例

```
$ ncc -v -c a.c                          （正）
$ ncc -vc a.c                              （誤）
```

- コンパイラはファイルとして以下の拡張子を認識します。その他の拡張子は、オブジェクトファイルとして扱われます。

拡張子	ファイル種別
.c　.i	Cソースファイル
.h	Cヘッダファイル
.C　.cc　.cpp　.cp　.cxx　.c++　.ii	C++ソースファイル
.H　.hh　.hpp　.hp　.hxx　.h++　.tcc	C++ヘッダファイル
.S　.s	アセンブラソース

- コンパイラオプションとファイルは、オプションファイルに記述することができます。オプションファイルは、コンパイラの起動時に常に指定するコンパイラオプションやファイルを指定するファイルです。コンパイラオプション、ファイルはコマンドラインと同じ規則で指定します。オプションファイルは、環境変数**HOME**に設定されたホームディレクトリに作成します。

コンパイラ種別	オプションファイル名
ncc	$HOME/.nccinit
nc++	$HOME/.nc++init

例

```
$ cat ~/.nccinit
-O3 -finline-functions
$ ncc -v a.c
/opt/nec/ve/libexec/ccom … -O3 -finline-functions … a.c
```

1.6　#includeで取り込まれるファイルのサーチ

#include <*ファイル名*>で取り込まれるファイルがサーチされるディレクトリの順番は以下のとおりです。

備考　　**#include** "*ファイル名*"で取り込まれるファイルは、ソースファイルのあるディレクトリからもサーチされます。そのディレクトリは最初にサーチされます。

(1)　**-I** で指定されたディレクトリ

(2)　**-B** で指定されたディレクトリ直下の include ディレクトリ

(3)　環境変数 **NCC_INCLUDE_PATH** で指定されたディレクトリ

(4)　**-isystem** で指定されたディレクトリ

(5)　/opt/nec/ve/ncc/*バージョン番号*/include

(6)　**-isysroot** が指定されているとき、**-isysroot** で指定されたディレクトリ直下の include、**-isysroot** が指定されていないとき、/opt/nec/ve/include

1.7　ライブラリのサーチ

ライブラリがサーチされるディレクトリの順番は以下のとおりです。

(1)　**-L** で指定されたディレクトリ

(2)　**-B** で指定されたディレクトリ

(3)　環境変数 **NCC_LIBRARY_PATH** で指定されたディレクトリ

(4)　/opt/nec/ve/ncc/*バージョン番号*/lib

(5)　環境変数 **VE_LIBRARY_PATH** で指定されたディレクトリ

(6)　/opt/nec/ve/lib/gcc

(7)　/opt/nec/ve/lib

1.8　演算例外

1.8.1　演算例外発生時の演算結果

浮動小数点数や整数の演算中にオーバーフロー、アンダーフロー、ゼロ除算、無効演算、あるいは、精度落ちが発生したときの処置について説明します。

(1)　ゼロ除算

整数型の演算において、除数がゼロの除算が行われたとき、演算結果は不定になります。

整数型でない演算において、除数がゼロの除算が行われたとき、被除数が正なら正の無限大、負なら負の無限大の演算結果になります。

環境変数**VE_FPE_ENABLE**に“DIV”を設定したとき、例外が発生し、エラーメッセージが標準エラー出力に出力されます。環境変数**VE_FPE_ENABLE**に”DIV”を設定しなかったとき、ゼロ除算例外は発生しません。

(2)　浮動小数点オーバーフロー

実数型、複素数型の演算において、オーバーフローが発生したとき、値が正なら正の有限の最大値を、負なら負の無限大が演算結果になります。

環境変数**VE_FPE_ENABLE**に”FOF”を設定したとき、例外が発生し、エラーメッセージが標準エラー出力に出力されます。環境変数**VE_FPE_ENABLE**に”FOF”を設定しなかったとき、浮動小数点オーバーフロー例外は発生しません。

(3)　浮動小数点アンダーフロー

実数型、複素数型の演算において、アンダーフローが発生したとき、ゼロが演算結果になります。

環境変数**VE_FPE_ENABLE**に”FUF”を設定したとき、例外が発生し、エラーメッセージが標準エラー出力に出力されます。環境変数**VE_FPE_ENABLE**に”FUF”を設定しなかったとき、浮動小数点アンダーフロー例外は発生しません。

(4)　無効演算

実数型、複素数型の演算において、無効演算が発生したとき、演算結果は不定、または、**NaN**になります。

環境変数**VE_FPE_ENABLE**に”INV”を設定したとき、例外が発生し、エラーメッセージが標準エラー出力に出力されます。環境変数**VE_FPE_ENABLE**に”INV”を設定しなかったとき、無効演算例外は発生しません。

(5)　精度落ち

実数型、複素数型の演算において、精度落ちが発生したとき、丸めた値が演算結果となります。

環境変数**VE_FPE_ENABLE**に”INE”を設定したとき、例外が発生し、エラーメッセージが標準エラー出力に出力されます。環境変数**VE_FPE_ENABLE**に”INE”を設定しなかったとき、精度落ち例外は発生しません。

(6)　ベクトル命令実行中の演算例外

ベクトル命令の実行中にオーバーフロー、アンダーフローあるいはゼロ除算が発生したときの演算結果は、スカラ命令のときと同じです。ただし、一つのベクトル命令の実行中に複数個の演算例外が発生するときでも、例外は1回しか発生しません。

1.8.2　演算例外マスクの変更

Cプログラムでは、演算例外マスクを変更することにより、演算例外発生時に例外を発生させるかどうかを制御できます。

環境変数**VE_FPE_ENABLE**により、演算例外マスクを変更できます。環境変数**VE_FPE_E**

NABLEの指定では、どの演算例外を発生させるかを指定します。

例

```
$ export VE_FPE_ENABLE=FOF,DIV
$ ./a.out
```

浮動小数点オーバーフロー時(FOF)とゼロ除算時(DIV)に例外を発生させるように、演算例外マスクを変更します。

1.8.3　トレースバック機能との連携

演算例外マスクを変更し、演算例外を発生させることで、トレースバック機能で例外発生箇所を容易に特定できます。

例

```
$ ncc -traceback=verbose below.c out.c watch.c hey.c ovf.c
...
$ export VE_TRACEBACK=VERBOSE
$ export VE_FPE_ENABLE=DIV
$ ./a.out
Runtime Error: Divide by zero at 0x600008001088
[ 0] 0x600008001088 below_           below.c:3
[ 1] 0x600018001168 out_             out.c:3
[ 2] 0x600020001168 watch_           watch.c:3
[ 3] 0x600010001168 hey_             hey.c:3
[ 4] 0x60000001cab8 MAIN__           ovf.c:5
```

例ではbelow.cの3行目付近でDivide by zeroの例外が発生したことがわかります。

1.8.4　演算例外マスクを変更するときの注意事項

演算例外マスクの変更は関数から呼び出しているシステムライブラリ関数にも適用されます。したがって、システムライブラリ関数で精度落ちなどの演算例外が発生したときも、エラーが発生することがあります。

第2章　　環境変数

2.1　コンパイル時に参照される環境変数

HOME

コンパイラは、オプションファイルを取得するために環境変数**HOME**を参照し、ユーザのホームディレクトリを取得します。環境変数**HOME**に値が設定されていないとき、オプションファイルは参照されません。

NCC_COMPILER_PATH

ncc/nc++から起動するC/C++コンパイラ(ccom)をサーチするディレクトリを指定します。複数指定するときはコロン(:)で区切って指定します。先に指定したディレクトリが優先されます。C/C++コンパイラがみつからなかったとき、ncc/nc++は標準のディレクトリにあるC/C++コンパイラを自動的に起動します。この環境変数は、標準以外のディレクトリを常にサーチしたい場合に設定します。

例

```
$ export NCC_COMPILER_PATH="$HOME/libexec:$HOME/wk/libexec"
```

NCC_INCLUDE_PATH

#includeで取り込まれるファイルをサーチするディレクトリを指定します。複数指定するときはコロン(:)で区切って指定します。先に指定したディレクトリが優先されます。この環境変数は、標準以外のディレクトリを常にサーチしたい場合に設定します。例えば、C/C++コンパイラに付属しないOSSのヘッダファイルのディレクトリを常にサーチしたいときに設定します。

例

```
$ export NCC_INCLUDE_PATH="$HOME/include:$HOME/wk/include"
```

NCC_LIBRARY_PATH

ライブラリをサーチするディレクトリを指定します。複数指定するときはコロン(:)で区切って指定します。先に指定したディレクトリが優先されます。この環境変数は、標準以外のディレクトリを常にサーチしたい場合に設定します。例えば、C/C++コンパイラに付属しないOSSのライブラリのディレクトリを常にサーチしたいときに設定します。

例

```
$ export NCC_LIBRARY_PATH="$HOME/lib:$HOME/wk/lib"
```

NCC_PROGRAM_PATH

C/C++コンパイラから起動するVE用のアセンブラおよびリンカをサーチするディレクトリを指定します。複数指定するときはコロン(:)で区切って指定します。先に指定したディレクトリが優先されます。アセンブラおよびリンカがみつからなかったとき、C/C++コンパイラは標準のディレクトリにあるアセンブラおよびリンカを自動的に起動します。この環境変数は、標準以外のディレクトリを常にサーチしたい場合に設定します。

例

```
$ export NCC_PROGRAM_PATH="$HOME/bin:$HOME/wk/bin"
```

PATH

ncc/nc++をサーチするディレクトリを追加します。複数指定するときはコロン(:)で区切って指定します。先に指定したディレクトリが優先されます。C/C++コンパイラをインストールしたディレクトリ配下のbinディレクトリを追加してください。この環境変数を設定すると、ncc/nc++を起動するとき、パスの指定を省略できます。標準のディレクトリにインストールしたときは、/opt/nec/ve/binを追加します。環境変数**PATH**はC/C++コンパイラ以外のアプリケーションにも影響を与えます。既存の環境変数**PATH**に追加する形で設定してください。

例

```
$ export PATH="/opt/nec/ve/bin:$PATH"
```

TMPDIR

コンパイラとコマンドが一時ファイルを作成するディレクトリを指定します。

(既定値: /tmp)

VE_LIBRARY_PATH

システムライブラリをサーチするディレクトリを指定します。複数指定するときはコロン(:)で区切って指定します。先に指定したディレクトリが優先されます。この環境変数は、標準以外のディレクトリを常にサーチしたい場合に設定します。

例

```
$ export VE_LIBRARY_PATH="$HOME/lib:$HOME/wk/lib"
```

2.2　実行時に参照される環境変数

OMP_NUM_THREADS / VE_OMP_NUM_THREADS

OpenMPと自動並列化プログラムで使用するスレッド数を設定します。値を設定しないと

き、スレッド数はVEのコア数と同じです。

例

```
$ export OMP_NUM_THREADS=4
```

OMP_STACKSIZE / VE_OMP_STACKSIZE

OpenMPと自動並列化プログラムで各スレッドが使用するスタックサイズの上限をキロバイト単位で指定します。単位としてMを使用するとメガバイト、Gを使用するとギガバイトとして指定することができます。値を設定しないとき、各スレッドが使用するスタックサイズは4メガバイトとなります。

例

```
$ export OMP_STACKSIZE=1G
```

OMP_TOOL / VE_OMP_TOOL

OMPT interface機能を有効/無効を制御します。有効にするとき、”enabled”を指定します。既定値は”disabled”(無効)です。

例

```
$ export OMP_TOOL=enabled
```

OMP_TOOL_LIBRARIES / VE_OMP_TOOL_LIBRARIES

OMPT interface機能のための動的リンクライブラリを指定します。複数指定するときはコロン(:)で区切って指定します。

例

```
$ export OMP_TOOL_LIBRARIES=libomptool.so:/usr/myhome/libompt.so
```

VE_ADVANCEOFF

Advance-offモードを制御します。”YES”を設定するとAdvance-offモードが有効になります。他の値を設定したとき、Advance-offモードは無効になります。Advance-offモードを有効にしたとき、実行時間が大幅に長くなる場合があります。

例

```
$ export VE_ADVANCEOFF=YES
```

VE_FPE_ENABLE

実行時の浮動小数点例外を制御します。この変数が設定されているとき、特定の例外が有効になります。以下の値を指定できます。値はコンマで区切って複数指定できます。

DIV

ゼロ除算例外

FOF

浮動小数点オーバーフロー

FUF

浮動小数点アンダーフロー

INV

無効演算

INE

精度落ち

例

```
$ export VE_FPE_ENABLE=DIV
```

VE_INIT_STACK

実行時にスタックに割り付ける変数を初期化する値を設定します。値が設定されなかったとき、0(ゼロ)で初期化されます。コンパイル時には**-minit-stack=runtime**を指定してください。

以下の値を指定できます。

ZERO

0(ゼロ)で初期化します。

NAN

double型のQuiet NaN(0x7fffffff7fffffff)で初期化します。

NANF

float型のQuiet NaN(0x7fffffff)で初期化します。

SNAN

double型のSignaling NaN(0x7ff4000000000000)で初期化します。

SNANF

float型のSignaling NaN(0x7fa00000)で初期化します。

0x*XXXX*

最大16桁の16進数で指定された値で初期化します。指定された値が8桁以上の16進数であるとき、8バイト単位で初期化します。そうでないときは4バイト単位で初期化します。

例

```
$ ncc -minit-stack=runtime a.c
```

```
$ export VE_INIT_STACK=SNAN
$ ./a.out
```

VE_LD_LIBRARY_PATH

実行時に動的リンカが共有ライブラリをサーチするディレクトリを指定します。複数指定するときはコロン(:)で区切って指定します。先に指定したディレクトリが優先されます。動的リンカは標準のディレクトリを自動的にサーチします。この環境変数は、標準以外のディレクトリを常にサーチしたい場合に設定します。例えば、C/C++コンパイラに付属しないOSSの共有ライブラリのディレクトリを常にサーチしたいときに設定します。

例

```
$ export VE_LD_LIBRARY_PATH="${HOME}/lib:$VE_LD_LIBRARY_PATH"
```

VE_NODE_NUMBER

プログラムが実行される VEノード番号を指定します。

VE_PROGINF

“YES”、または、”DETAIL”が設定されているとき、プログラムの実行終了時に標準エラー出力にプログラム実行解析情報が出力されます。プログラム実行解析情報の詳細については「PROGINF/FTRACE ユーザーズガイド」を参照してください。

VE_TRACEBACK

実行時に致命的エラーが発生したときのトレースバック情報の出力を制御します。トレースバック情報を出力するとき、プログラムのコンパイル時とリンク時に**-traceback**を指定する必要があります。この変数に”FULL”、または、”ALL”が設定されたとき、環境変数**VE_TRACEBACK_DEPTH**で指定した値を上限にトレースバック情報が出力されます。その他の値が設定されたとき、エラーが発生した関数のトレースバック情報のみ出力されます。値が設定されなかったとき、トレースバック情報は出力されません。

トレースバック情報に表示されたアドレス情報から、致命的エラーの発生個所がわかります。

例

```
$ export VE_TRACEBACK=FULL
$ ./a.out
Runtime Error: Divide by zero at 0x600000000cc0
[ 1] Called from 0x7f5ca0062f60
[ 2] Called from 0x600000000b70
Floating point exception
```

naddr2lineコマンドを使って表示アドレスのファイル位置を求めます。例ではa.cの3行目付近でDivide by zeroの例外が発生したことがわかります。

例

```
$ naddr2line -e ./a.out -a 0x600000000cc0
0x0000600000000cc0
a.c:3
```

また、コンパイル時とリンク時に**-traceback=verbose**を指定して作成したプログラムの実行時に、環境変数**VE_TRACEBACK**に"VERBOSE"を指定すると、ファイル名や行番号情報が出力されます。

例

```
$ export VE_TRACEBACK=VERBOSE
$ ./a.out
Segmentation fault: Address not mapped to object at 0x600008001078
[ 0] 0x600008001078 below            below.c:8
[ 1] 0x600018001170 out              out.c:3
[ 2] 0x600020001170 watch            watch.c:3
[ 3] 0x600010001170 hey              hey.c:3
[ 4] 0x600000001500 main             ovf.c:10
```

VE_TRACEBACK_DEPTH

環境変数**VE_TRACEBACK**にてトレースバック情報を出力する際に、トレースバック情報の上限の値を設定します。値を指定しない場合は、50が設定されます。この環境変数に0を設定した場合、上限なしとなります。

第3章　　コンパイラオプション

本章では、C/C++コンパイラのコンパイラオプションについて説明します。
コンパイラオプションは、次のカテゴリに分類できます。

- 全体オプション
  コンパイラシステム全体を制御するオプションです。
- 最適化・ベクトル化オプション
  最適化・ベクトル化を制御するためのコンパイラオプションです。
- 並列化オプション
  自動並列化機能を制御するためのコンパイラオプションです。
- インライン展開オプション
  インライン展開機能を制御するためのコンパイラオプションです。
- コード生成オプション
  性能測定やスタック領域の初期化を行う付加的なコードを生成するためのコンパイラオプションです。
- デバッグオプション
  デバッグをサポートするオブジェクトファイルを生成するためのコンパイラオプションです。
- 言語仕様制御オプション
  C/C++の言語仕様を制御するためのコンパイラオプションです。
- メッセージオプション
  コンパイルメッセージの出力を制御するためのコンパイラオプションです。
- リスト出力オプション
  コンパイルリストの出力を制御するためのコンパイラオプションです。
- プリプロセッサオプション
  プリプロセスを制御するためのコンパイラオプションです。
- アセンブラオプション
  コンパイラのコマンドラインで指定できるアセンブラを制御するためのオプションです。
- リンカオプション
  コンパイラのコマンドラインで指定できるリンカを制御するためのオプションです。

- ディレクトリオプション
  コンパイラシステムを構成するコマンドや、リンカ、システムヘッダファイル、システムライブラリのあるディレクトリを指定するためのコンパイラオプションです。

3.1　全体オプション

-S

コンパイルのみ行いアセンブラソースを出力する。

-c

コンパイルのみ行いオブジェクトファイルを出力する。

-cf=*conf*

コンパイル、リンクするとき*conf*で指定されたコンフィギュレーションファイルを適用する。

-clear

本オプションより前に指定されたコンパイラオプション、ファイルをすべて無視する。

-fsyntax-only

文法チェックのみ行う。

-o *ファイル名*

出力するプリプロセス結果、アセンブラソース、オブジェクトファイル、実行ファイルの名前を指定する。複数のソースファイルを指定し、**-S**、**-c**、または、**-E**を指定するとき、指定できない。

-x *language*

ファイルの言語の種類を指定する。このオプションはファイル拡張子に従う自動選択より優先される。仕様は、コマンドライン上のこのオプションに続く(もし次の**-x**があればそれまで)すべてのファイルに適用される。*language*に指定できる言語の種類は以下である。

c

Cソースファイルとしてコンパイルする。

c++

C++ソースファイルとしてコンパイルする。

assembler

アセンブラソースファイルとしてアセンブルする。

assembler-with-cpp

プリプロセスし、プリプロセスされたファイルをアセンブルする。

@*file-name*

コンパイラオプションを*file-name*で指定されたファイルから読み込む。読み込んだコンパイラオプションは、**@*file-name***を指定した位置に挿入される。

3.2　最適化・ベクトル化オプション

-O[*n*]

最適化レベル(*n*)を指定する。*n*に指定できる値は以下である。

4

C/C++言語仕様を逸脱した副作用を伴う最大限の最適化・自動ベクトル化を適用する。

3

副作用を伴う最適化・自動ベクトル化、および、多重ループの最適化を適用する。

2

副作用を伴う最適化・自動ベクトル化を適用する。(既定値)

1

副作用を伴わない最適化・自動ベクトル化を適用する。

0

最適化、自動ベクトル化、並列化、インライン展開を適用しない。

-fargument-alias

引数が互いに、または、非局所オブジェクトへのエイリアスであると仮定して最適化・自動ベクトル化を適用する。(既定値)

-fargument-noalias

引数が互いに、または、非局所オブジェクトへのエイリアスでないと仮定して最適化・自動ベクトル化を適用する。

-f[no-]associative-math

演算順序の変更を伴う最適化・自動ベクトル化を適用する[しない]。**-fno-associative-math**を指定したとき、**-fmatrix-multiply**による行列積の高速なベクトルライブラリ呼出しへの変換を行わない。

(既定値: **-fassociative-math**)

-f[no-]aggressive-associative-math

より激しく演算順序の変更を伴う最適化・自動ベクトル化を適用する[しない]。

(既定値: **-fno-aggressive-associative-math**)

-f[no-]check-noexcept-violation

C++のnoexcept指定を違反しているかの実行時チェックを有効にする[しない]。本オプションが有効でないとき、noexcept指定を違反してもstd::terminate関数は呼び出されずプログラムの実行を継続する。

(既定値: **-fno-check-noexcept-violation**)

-f[no-]cse-after-vectorization

ベクトル化後にも共通式削除の最適化を適用する[しない]。

(既定値: **-fno-cse-after-vectorization**)

-f[no-]fast-math

ベクトル化されたループ外でスカラ高速バージョンの数学関数を使用する[しない]。

(既定値: **-ffast-math**)

-f[no-]ignore-induction-variable-overflow

最適化を行うためにインダクション変数のオーバーフローを無視する[しない]。

(既定値: **-fno-ignore-induction-variable-overflow**)

-f[no-]ignore-volatile

volatile修飾子を無視する[しない]。(既定値: **-fno-ignore-volatile**)

-fivdep

すべてのループに**ivdep**指示行を指定する。

-fivdep-omp-worksharing-loop

OpenMP機能により並列化されたループに**ivdep**指示行を指定する。ただし、**safelen**、**simdlen**句の指定された**simd**が指定されたループを除く。

-f[no-]loop-collapse

多重ループの一重化を許可する[しない]。**-O***n* (*n*=2,3,4)が有効でなければならない。

(既定値: **-fno-loop-collapse**)

-floop-count=*n*

繰返し数に合った最適化を行うため、コンパイル時に繰返し数を算定できないループの繰返し数を*n*回と仮定する。 (既定値: **-floop-count**=5000)

-f[no-]loop-fusion

ループ融合を許可する[しない]。**-O***n*（*n*=2,3,4)が有効でなければならない。
(既定値: **-fno-loop-fusion**)

-f[no-]loop-interchange

ループ入れ換えを許可する[しない]。**-O***n*（*n*=2,3,4)が有効でなければならない。
(既定値: **-fno-loop-interchange**)

-f[no-]loop-normalize

ループの正規化を許可する[しない]。このとき、コンパイラは、ループの繰返し数がループ本体で変更されないと仮定する。(既定値: **-fno-loop-normalize**)

-f[no-]loop-split

関数の呼出し前後でのループ分割を許可する[しない]。**-O***n*（*n*=2,3,4)が有効でなければならない。(既定値: **-fno-loop-split**)

-f[no-]loop-strip-mine

ループストリップマイニングを許可する[しない]。**-O***n*（*n*=2,3,4)が有効でなければならない。(既定値: **-fno-loop-strip-mine**)

-f[no-]loop-unroll

ループアンローリングを許可する[しない]。**-O***n*（*n*=2,3,4)が有効でなければならない。
(既定値: **-floop-unroll**)

-**floop-unroll-complete**=*m*

ループの繰返し数が定数でコンパイル時に算定でき*m*回以下であるとき、そのループのループ展開(完全ループアンローリング)を許可する。**-O***n*（*n*=2,3,4)が有効でなければならない。(既定値: **-floop-unroll-complete**=4)

備考

別名オプションとして-**floop-unroll-completely**=*m*も使用できる。

-floop-unroll-max-times=*n*

最大アンロール段数を*n*段とする。本オプションが有効でないとき、コンパイラは適切なアンロール段数を自動的に決定する。

-f[no-]matrix-multiply

行列積の高速なベクトルライブラリ呼出しへの変換を許可する[しない]。**-O***n*（*n*=2,3,4)、かつ、**-fassociative-math**が有効でなければならない。
(既定値: **-fno-matrix-multiply**)

-f[no-]move-loop-invariants

条件下の不変式のループ外への移動を許可する[しない]。

(既定値: **-fmove-loop-invariants**)

-f[no-]move-loop-invariants-if

ループ内不変のIF構造のループ外への移動を許可する[しない]。-**O***n* (*n*=2,3,4)が有効でなければならない。(既定値: **-fno-move-loop-invariants-if**)

-f[no-]move-loop-invariants-unsafe

副作用を伴う可能性のあるコードのループ外への移動を許可する[しない]。副作用を伴う可能性のあるコードの例は以下のとおり。

- 除算
- 1バイト、または、2バイトの記憶領域へのメモリ参照

(既定値: **-fno-move-loop-invariants-unsafe**)

-f[no-]move-nested-loop-invariants-outer

多重ループ内の不変式の外側ループ外への移動を許可する[しない]。このオプションが指定されたとき、不変式は当該ループ外に移動される。

(既定値: -**fmove-nested-loop-invariants-outer**)

-fnaked-ivdep

#pragma ivdepを**#pragma _NEC ivdep**として扱う。

-fnamed-alias

ポインタの指示先がエイリアスをもつと仮定して最適化・自動ベクトル化を適用する。

-fnamed-noalias

ポインタの指示先がエイリアスをもたないと仮定して最適化・自動ベクトル化を適用する。(既定値)

-f[no-]outerloop-unroll

外側ループのアンローリングを許可する[しない]。**-O***n* (*n*=2,3,4)が有効でなければならない。(既定値: **-fno-outerloop-unroll**)

-**fouterloop-unroll-max-size**=*n*

外側ループアンローリングの対象とするループの最大サイズを*n*とする。

(既定値: **-fouterloop-unroll-max-size=4**)

-fouterloop-unroll-max-times=*n*

外側ループの最大アンロール段数を*n*段とする。*n*は2のべき乗数でなければならない。本オプションが有効でないとき、コンパイラは適切なアンロール段数を自動的に決定する。

-f[no-]precise-math

pow(3C)、powf(3C)のベクトル演算において、べき指数(exponent)が整数値であるときの計算精度を高めたべき乗算アルゴリズムを適用する[しない]。計算精度は高まるが、計算速度は低下する。(既定値: **-fno-precise-math**)

-f[no-]reciprocal-math

“x/y”の“x * (1/y)”への変更を許可する[しない]。(既定値: **-freciprocal-math**)

-f[no-]replace-loop-equation

ループ繰返し条件の「!=」と「==」を「<=」、または、「>=」に置換する[しない]。
(既定値: **-fno-replace-loop-equation**)

-f[no-]strict-aliasing

すべての最適化において言語規格のエイリアシング規則を満たしていると仮定することを許可する[しない]。(既定値: **-fstrict-aliasing**)

本オプションが有効でないとき、コンパイラは格納された値が以下のタイプの内の一つであると仮定する。

- オブジェクトの実際の型と対応する型
- オブジェクトの実際の型と対応すると識別される型
- オブジェクトの実際の型と対応する符号あり、または、符号なしの型
- オブジェクトの実際の型と対応すると識別される符号あり、または、符号なしの型
- 前述の型の一つをメンバに含む集成体、共用体 (再帰的に集成体、共用体に含まれるメンバも含む)
- 文字型

-fthis-pointer-alias

thisポインタがエイリアスをもつと仮定して最適化・自動ベクトル化を適用する。

-fthis-pointer-noalias

thisポインタがエイリアスをもたないと仮定して最適化・自動ベクトル化を適用する。(既定値)

-m[no-]list-vector

同一の非線形添字をもつ同一配列が左辺と右辺に現れる代入文のベクトル化を許可する[しない]。

(既定値: **-mno-list-vector**)

-mretain-*keyword*

優先的にLLC(Last-Level Cache)に保持する配列、または、ポインタのベクトルメモリア

クセスの種類を指定する。*keyword*に指定できるメモリアクセスの種類は以下である。

all

すべてのベクトルメモリアクセス。(既定値)

list-vector

非線形添字をもつベクトルメモリアクセス。

none

優先度は指定しない。

-msched-*keyword*

命令の並べ換えのレベルを指定する。*keyword*で指定できる命令の並べ換えレベルは以下である。

none

命令の並べ換えを行わない。

insns

基本ブロック内での命令の並べ換えを行う。

block

基本ブロック内での命令の並べ換えを行う。**-msched-insns**より命令の並べ換えを適用する範囲を広くし、より強力に命令を並べ換える。(既定値)

interblock

基本ブロックをまたがって命令の並べ換えを行う。

-m[no-]vector

自動ベクトル化を適用する[しない]。(既定値: **-mvector**)

-m[no-]vector-advance-gather

ベクトル収集命令をループ本体の前方に移動する[しない]。前方に移動することにより、後続の演算命令などとオーバーラップさせ、計算時間を短縮できることがある。

(既定値: **-mvector-advance-gather**)

-mvector-advance-gather-limit=*n*

前方に移動するベクトル収集命令の個数の上限を指定する。

(既定値: **-mvector-advance-gather-limit**=56)

-m[no-]vector-dependency-test

依存関係の判定を用いた条件ベクトル化を適用する[しない]。**-O***n* (*n*=2,3,4)が有効でなければならない。(既定値: **-mvector-dependency-test**)

-m[no-]vector-floating-divide-instruction

ベクトル浮動小数点除算において、ベクトル浮動小数点除算命令を使用する[しない]。使用しないとき、逆数近似命令を用いてベクトル浮動小数点除算する。

(既定値: **-mno-vector-floating-divide-instruction**)

-m[no-]vector-fma

ベクトル積和演算命令の使用を許可する[しない]。(既定値: **-mvector-fma**)

-m[no]vector-intrinsic-check

ベクトル化された部分から呼び出された数学関数の引数の値の範囲を実行時に検査する[しない]。(既定値: **-mno-vector-intrinsic-check**)

本機能の対象となる数学関数は以下のとおり。

acos、acosh、asin、atan、atan2、atanh、cos、cosh、cotan、exp、exp10、exp2、expm1、log10、log2、log、pow、sin、sinh、sqrt、tan、tanh

-m[no-]vector-iteration

ベクトル漸化式命令の使用を許可する[しない]。(既定値: **-mvector-iteration**)

-m[no-]vector-iteration-unsafe

結果不正を伴うことがあるベクトル漸化式命令の使用を許可する[しない]。

(既定値: -**mvector-iteration-unsafe**)

-m[no-]vector-loop-count-test

ループの繰返し数判定を用いた条件ベクトル化を適用する[しない]。-**O***n* (*n*=2,3,4)が有効でなければならない。(既定値: **-mno-vector-loop-count-test**)

-m[no-]vector-low-precise-divide-function

低精度のベクトル浮動小数点数除算を使用する[しない]。通常精度版と比較して高速に処理されるが、除算結果の仮数部に最大1ビットの誤差が含まれることがある。

(既定値: **-mno-vector-low-precise-divide-function**)

-m[no-]vector-merge-conditional

THEN節、**ELSE IF**節、**ELSE**節のベクトルロードとストアをマージすることを許可する[しない]。

(既定値: **-mno-vector-merge-conditional**)

-m[no-]vector-packed

パックドベクトル命令の使用を許可する[しない]。(既定値: **-mno-vector-packed**)

-m[no-]vector-power-to-explog

ベクトル化されたループ中のpow(R1,R2)の呼び出しをexp(R2*log(R1))の呼び出しに置

き換えることを許可する[しない]。powf(3C)に対しても同様の最適化が適用される。pow(3C)、powf(3C)で計算した場合に比べて実行時間は短縮されるが、計算結果が誤差レベルで変わることがある。

(既定値: **-mno-vector-power-to-explog**)

-m[no-]vector-power-to-sqrt

ベクトル化されたループ中のpow(R1,R2)の呼び出しにおいて、R2の値が0.5、または、1.0/3.0などの特別な値のとき、pow(3C)の呼び出しをsqrt(3C)、cbrt(3C)を使用した計算に置き換えることを許可する[しない]。powf(3C)に対しても同様の最適化が適用される。sqrt(3C)、cbrt(3C)を使用した計算のとき、pow(3C)、powf(3C)で計算した場合に比べて実行時間は短縮されるが、計算結果が誤差レベルで変わることがある。

(既定値: **-mvector-power-to-sqrt**)

-m[no-]vector-reduction

ベクトルリダクション命令の使用を許可する[しない]。(既定値: **-mvector-reduction**)

-m[no-]vector-shortloop-reduction

リダクション演算を含むループにおいてループの繰返し数判定を用いた条件ベクトル化を適用する[しない]。**-O***n* (*n*=2,3,4)が有効でなければならない。

(既定値: **-mno-vector-shortloop-reduction)**

-m[no-]vector-sqrt-instruction

ベクトルSQRTにおいて、ベクトルSQRT命令を使用する[しない]。使用しないとき、逆数近似命令を用いてベクトルSQRT演算する。

(既定値: **-mno-vector-sqrt-instruction**)

-mvector-threshold=*n*

ベクトル化の対象とする最小繰返し数(*n*)を指定する。

(既定値: **-mvector-threshold**=5)

-mwork-vector-kind=none

部分ベクトル化(ループ分割によるベクトル化)を許可しない。

3.3　並列化オプション

-fopenmp

OpenMP機能を使用する。**-pthread**は暗黙的に有効となる。

-fopenmp-tools

OMPT interface機能を使用する。(既定値: **-fno-openmp-tools**)

-m[no-]create-threads-at-startup

最初に実行されるParallelリージョン、または、並列ループの開始時に、OpenMP・自動並列化のためのスレッドを生成する[しない]。既定値では、プログラムの実行開始時に生成する。(既定値: **-mcreate-threads-at-startup**)

備考

本オプションを指定するとき、**-static-nec**、または、**-static**を指定してください。

-mparallel

自動並列化を適用する。**-pthread**は暗黙的に有効となる。

-mparallel-innerloop

内側ループの並列化を許可する。

-m[no-]parallel-omp-routine

-fopenmpと**-mparallel**が同時に指定されたとき、OpenMPディレクティブを含む関数を自動並列化する[しない]。(既定値: **-mparallel-omp-routine**)

-mparallel-outerloop-strip-mine

外側ループストリップマイニングで得られる多重ループの並列化を許可する。

-mparallel-sections

並列化されたセクションの生成を許可する。

-mparallel-threshold=*n*

自動並列化を適用するループを選択する際のしきい値(*n*)を指定する。

しきい値以上の作業量を持つループに自動並列化を適用する。

(既定値: **-mparallel-threshold**=2000)

-mschedule-dynamic

-mschedule-runtime

-mschedule-static

-mschedule-chunk-size=*n*

OpenMP並列化、自動並列化において、schedule句によりスレッドのスケジューリング種別、サイズの指定が行われなかった場合のスケジューリング種別、サイズを指定する。

-pthread

pthreadライブラリを用いたマルチスレッドのサポートを有効にする。

3.4　インライン展開オプション

-**finline-abort-at-error**

サーチ対象のソースファイル内で定義された手続の生成に失敗した際、コンパイルを終了する。本オプションが指定されなかったとき、エラーのあったファイルをサーチせず、コンパイルを継続する。(既定値: **-fno-inline-abort-at-error**)

-f[no-]inline

インライン関数のインライン展開を行う[行わない]。(既定値: **-finline**)

-f[no-]inline-copy-arguments

自動インライン展開する関数の実引数のコピーを生成する[しない]。コピーを生成しないとき、関数の仮引数が対応する実引数に置き換えられる。

(既定値: -f**inline-copy-arguments**)

-**finline-directory**=*ディレクトリ名*

インライン展開する関数をサーチするとき、指定されたディレクトリにあるすべてのソースファイルをサーチする。複数指定するときにコロン(:)で区切って指定する。

-**fno-inline-directory**=*ディレクトリ名*

インライン展開する関数をサーチするとき、指定されたディレクトリにあるすべてのソースファイルをサーチしない。複数指定するときにコロン(:)で区切って指定する。**-finline-file**、または、**-finline-directory**でサーチ対象となったソースファイルをサーチしないとき指定する。

-**finline-file**=*文字列*

インライン展開する関数をサーチするとき、指定されたソースファイルをサーチする。複数指定するときにコロン(:)で区切って指定する。**all**が指定されたとき、コマンドラインで指定されたコンパイル対象のすべてのソースファイルもサーチする。

-**fno-inline-file**=*文字列*

インライン展開する関数をサーチするとき、指定されたソースファイルをサーチしない。複数指定するときにコロン(:)で区切って指定する。**-finline-file**、または、**-finline-directory**でサーチ対象となったソースファイルをサーチしないとき指定する。

-finline-functions

自動インライン展開を適用する。

-finline-max-depth=*n*

自動インライン展開する関数の深さを指定する。(既定値: **-finline-max-depth**=2)

-finline-max-function-size=*n*

自動インライン展開する関数の大きさ(関数の中間言語の量)を指定する。

(既定値: **-finline-max-function-size=**50)

-finline-max-times=*n*

自動インライン展開後の関数の大きさ(関数の中間言語の量)の上限を「インライン展開前の関数の大きさ×*n*」とする。(既定値: **-finline-max-times**=6)

-**f[no-]inline-suppress-diagnostics**

サーチ対象のソースファイル内で定義された関数の生成に失敗した際、エラーメッセージを出力しない[する]。-finline-file、または、-finline-directoryでサーチ対象となったソースファイルの内、正常にサーチされているものを確認したいときに**-fno-inline-suppress-diagnostics**を指定する。

(既定値: **-finline-suppress-diagnostics**)

-**mgenerate-il-file**

クロスファイルインライニングのためのILファイルをカレントディレクトリに出力する。ファイル名は、「ソースファイル名.cil」である。**-o** ファイル名で名前を変更できる。

-**mread-il-file** *ILファイル名*

インライン展開する関数をサーチするとき、指定されたILファイルをサーチする。複数指定するときにコロン(:)で区切って指定する。**-finline-file**、**-finline-directory**、または、**-mgenerate-il-file**のいずれかと同時に指定されたとき無視される。

3.5　コード生成オプション

-finstrument-functions

関数の入口と出口にトレース用の関数呼出しを挿入する。挿入する関数は以下。

```
void __cyg_profile_func_enter(void *this_fn, void *call_site);
void __cyg_profile_func_exit(void *this_fn,void *call_site);
```

-fpic

-fPIC

位置独立コードを生成する。

-ftrace

ftrace機能用のオブジェクトファイル、および、実行ファイルを生成する。

(既定値: **-no-ftrace**)

-**p**

-pg

プロファイラ情報(ngprof)を出力するオブジェクトファイル、実行ファイルを生成する。

-[no-]proginf

PROGINF機能用の実行ファイルを生成する[しない]。(既定値: **-proginf**)

3.6　デバッグオプション

-g

DWARFフォーマットでデバッグ情報を生成する。

-minit-stack=*value*

実行時にスタックに割り付ける領域を指定された値で初期化する。*value*に指定できる値は以下である。

zero

0(ゼロ)で初期化する。

nan

double型のQuiet NaN(0x7fffffff7fffffff)で初期化する。

nanf

float型のQuiet NaN(0x7fffffff)で初期化する。

snan

double型のSignaling NaN(0x7ff4000000000000)で初期化する。

snanf

float型のSignaling NaN(0x7fa00000)で初期化する。

runtime

環境変数**VE_INIT_STACK**に設定された値で初期化する。

0x*XXXX*

最大16桁の16進数で指定された値で初期化する。指定された値が8桁以上の16進数であるとき、8バイト単位で初期化する。そうでないときは4バイト単位で初期化する。

-traceback[=verbose]

実行時に環境変数**VE_TRACEBACK**がセットされているとき、トレースバック情報を出力するオブジェクトファイル、実行ファイルを生成する。

verboseを指定した場合、トレースバック情報を出力する際に、ファイル名や行番号情報を出力するための情報を追加したオブジェクトファイル、実行ファイルを生成する。トレースバック情報を出力する際に、これらの情報を出力するためには、実行時に環境変数**VE_TRACEBACK=VERBOSE**を指定する必要がある。

3.7　言語仕様制御オプション

3.7.1　C 言語仕様制御

-fno-allow-keyword-macros

いかなるキーワードで定義されるマクロも許可しない。

-fgnu89-inline

GCC拡張のgnu89のルールにしたがってインライン関数の関数定義を出力する。

-f[no-]restrict

restrictをC言語のキーワードとして扱う[扱わない]。(既定値: **-frestrict**)

備考

C言語とC++言語で既定値が異なる。C++言語での既定値は**-fno-restrict**である。

-fsigned-char | **-funsigned-char**

charで宣言されたデータを**signed char**、または、**unsigned char**とみなす。

(既定値: **-fsigned-char**)

-std=*standard*

C言語仕様を指定する。*standard*として指定できるキーワードはgnu89、gnu99、gnu11、c99、c11のいずれかである。 (既定値: **-std**=gnu11)

-traditional

K&R C言語仕様にしたがってCソースファイルをプリプロセスする。**-E**と同時に指定しなければならない。

-traditional-cpp

K&R C言語仕様にしたがってCソースファイルをプリプロセスする。

-trigraphs

3文字表記の使用を許可する。

3.7.2　C++言語仕様制御

-fdefer-inline-template-instantiation

インライン関数テンプレートのインスタンス化を関数呼出し位置で行わず、適切なタイミングに延期して行う。(既定値)

-fno-defer-inline-template-instantiation

インライン関数テンプレートのインスタンス化を関数呼出し位置で行う。

(既定値: **-fdefer-inline-template-instantiation**)

-f[no-]exceptions

C++例外処理機能の使用を有効にする[無効にする]。(既定値: **-fexceptions**)

-fext-numeric-literals

サフィックス I、i、J、jをもつ定数式を、複素数型定数として扱う。

(**-std**=gnu++11、**-std**=gnu++14、**-std**=gnu++17のいずれかが有効であるときの既定値)

-fno-ext-numeric-literals

サフィックス I、i、J、jをもつ定数式を、ユーザ定義リテラルとして扱う。

(**-std**=c++11、c++14、c++17、c++20のいずれかが有効であるときの既定値)

-ffor-scope

for文の初期化文で宣言されたデータのスコープの範囲をfor文で定義されるスコープにする。(既定値)

-fno-for-scope

for文の初期化文で宣言されたデータのスコープの範囲をfor文の属するブロック全体のスコープにする。(既定値: **-ffor-scope**)

-fimplicit-include

テンプレートをヘッダファイルで宣言し、ソースファイルで定義しているとき、ヘッダファイルと同じ名前のソースファイルを暗黙に取り込む。

-f[no-]restrict

restrictをC++言語のキーワードとして扱う[扱わない]。(既定値: **-fno-restrict**)

備考

C言語とC++言語で既定値が異なる。C言語での既定値は**-frestrict**である。

-f[no-]rtti

実行時型同定(Run-Time Type Identification)を利用する[しない]。(既定値: **-frtti**)

-ftemplate-depth=*n*

あるテンプレートのインスタンシエーションの最大個数を指定する。無制限に再帰的にインスタンシエートされてしまうことを避けるために使用する。*n*に指定できる値は0～1024である。*n*に0を指定したとき、無制限にインスタンシエートする。

(既定値: **-ftemplate-depth=**256)

-std=*standard*

C++言語仕様を指定する。*standard*として指定できるキーワードはgnu++11、gnu++14、gnu++17、gnu++20、c++11、c++14、c++17、c++20のいずれかである。

(既定値: **-std**=gnu++14)

3.8　メッセージオプション

-Wall

すべての警告レベルの文法診断メッセージを出力する。

-Wcomment

/* */ コメントの内部に /* があるとき警告メッセージを出力する。

-Werror

すべての警告レベルの文法診断メッセージを致命的エラーとして扱う。

-Wno-div-by-zero

コンパイル時に検出する整数ゼロ除算の警告メッセージを抑止する。

-Wunknown-pragma

コンパイラが認識できない**#pragma**を検出したとき、警告メッセージを出力する。

-Wunused

-Wunused-variableと同じ。

-Wunused-but-set-parameter

設定されるが未使用の仮引数に関する警告メッセージを出力する。

-Wunused-but-set-variable

設定されるが未使用のローカル変数に関する警告メッセージを出力する。

-Wunused-parameter

未使用の仮引数に関する警告メッセージを出力する。

-Wunused-value

計算結果が使われない式に関する警告メッセージを出力する。

-Wunused-variable

未使用のローカル変数、関数に関する警告メッセージを出力する。

-fdiag-inline=*n*

自動インライン展開に関する診断メッセージのレベル*n*を指定する。 (0: 出力しない、1: 通常レベル、2: 詳細レベル) (既定値: **-fdiag-inline**=1)

-fdiag-parallel=*n*

自動並列化に関する診断メッセージのレベル*n*を指定する。 (0: 出力しない、1: 通常レベル、2: 詳細レベル) (既定値: **-fdiag-parallel**=1)

-fdiag-vector=*n*

ベクトル化に関する診断メッセージのレベル*n*を指定する。 (0: 出力しない、1: 通常レベル、2: 詳細レベル) (既定値: **-fdiag-vector**=1)

-fdiag-system-header

システムヘッダファイル中で定義された関数に対する最適化診断メッセージを出力する。

-pedantic

拡張言語仕様に対して警告メッセージを出力する。

-pedantic-errors

拡張言語仕様に対して致命的エラーを出力する。

-w

すべての警告メッセージを抑止する。

3.9　リスト出力オプション

-report-file=*ファイル名*

既定のファイルの代わりに指定されたファイルにリスト結果を出力する。

-report-append-mode

「上書きモード」の代わりに「追加モード」で出力ファイルを開く。このオプションは**-report-file**の指定が無いと使用できない。

-report-all

コード生成リスト、診断メッセージリスト、編集リスト、インラインリスト、オプションリスト、ベクトルリストを出力する。

-[no-]report-cg

コード生成モジュールの最適化情報リストを出力する。

(既定値: **-no-report-cg**)

-[no-]report-diagnostics

診断メッセージリストを出力する[しない]。

(既定値: **-no-report-diagnostics**)

-[no-]report-format

編集リストを出力する[しない]。

(既定値: **-no-report-format**)

-[no-]report-inline

インライン展開モジュールの最適化情報リストを出力する[しない]。

(既定値: **-no-report-inline**)

-[no-]report-option

オプションリストを出力する[しない]。

(既定値: **-no-report-option**)

-[no-]report-system-header

システムヘッダファイル中で定義された関数に対するコンパイルリストを出力する。

(既定値: **-no-report-system-header**)

-[no-]report-vector

ベクトル化モジュールの最適化情報リストを出力する[しない]。

(既定値: **-no-report-vector**)

3.10 プリプロセッサオプション

-C

プリプロセス結果からコメントを取り除かない。

-dD

プリプロセス結果の出力に、**#define**で定義されたマクロ定義、および、**#undef**で未定義にされたマクロ定義を挿入する。**-E**と同時に指定されていないとき無視される。

-dI

入力ファイル内で**#include**が使用されているとき、プリプロセス結果の出力に、その**#include**を挿入する。**-E**と同時に指定されていないとき無視される。**-dM**と同時に指定されたとき無視される。

-dM

プリプロセス結果の代わりに、**#define**、**-D**で定義されたマクロ定義、および、定義済みマクロを標準出力に出力する。**-E**と同時に指定されていないとき無視される。

-dN

プリプロセス結果の出力に、**#define**、**-D**で定義されたマクロ名を挿入する。**-E**と同時に指定されていないとき無視される。

-D*macro***[=***defn***]**

#defineと同様に*macro*を値*defn*で定義する。=*defn*が指定されなかったとき*macro*は10進整数1に定義される。

-E

プリプロセスのみ行いプリプロセス結果を標準出力に出力する。

-H

#includeで取り込まれるファイルの名前を標準エラー出力に出力する。

-I*ディレクトリ名*

#includeで指定されたファイルを*ディレクトリ名*で指定されたディレクトリからサーチする。

-I-

このオプションより前に指定する**-I**で指定するディレクトリは、**#include** "..."で指定されるファイルのみサーチし、**#include** <...>で指定されるファイルはサーチしない。

-include *ファイル名*

コンパイルの開始時に*ファイル名*で指定されたファイルを取り込む。

-isysroot *ディレクトリ名*

#includeで指定されたヘッダファイルを*ディレクトリ名*で指定されたディレクトリ配下のincludeディレクトリからサーチする。

-isystem *ディレクトリ名*

#includeで指定されたヘッダファイルを**-I**で指定されるディレクトリの後、かつ、標準のシステムディレクトリより前に、*ディレクトリ名*で指定されたディレクトリからサーチする。

-M

プリプロセス結果の代わりにファイルの依存情報を出力する。

-MD

-M -MF *ファイル名* と同じ。出力する*ファイル名*は、入力ファイル、または、**-o**で指定する名前の拡張子を「.d」に変更したものとなる。

-MF *ファイル名*

ファイルの依存情報を*ファイル名*で指定するファイルに出力する。このオプションは**-M**と同時に指定されていないとき無視される。

-MP

ファイルの依存情報だけでなく、依存するファイルを仮想ターゲットとして出力する。このオプションは**-M**と同時に指定されていないとき無視される。

-MT *target*

依存情報のターゲットを*target*に変更して出力する。このオプションは**-M**と同時に指定されていないとき無視される。

-nostdinc

ヘッダファイルについて標準のシステムディレクトリはサーチしない。

-P

行ディレクティブをプリプロセス結果に出力しない。

-U*macro*

*macro*の定義を削除する。

-undef

システム固有の既定義マクロを定義しない。

-Wp,*option*

プリプロセッサ(cpp)のオプションを指定する。複数のオプションや引数はコンマ(,)で区切って指定する。

3.11 アセンブラオプション

-Wa, *option*

アセンブラ(nas)のオプションを指定する。複数のオプションや引数はコンマ(,)で区切って指定する。

-Xassembler *option*

アセンブラ(nas)のオプションを指定する。引数を持ったオプションのとき、オプションと引数をそれぞれ本オプションで指定する。

-assembly-list

アセンブリリストを出力する。出力*ファイル名*は、入力*ファイル名*の拡張子を「.O」に変更したものとなる。

3.12 リンカオプション

-Bdynamic

実行時にダイナミックリンクライブラリをリンクする。**-Bstatic**を指定しない場合の既定値である。

-Bstatic

利用者のライブラリを静的リンクする。

-L*ディレクトリ名*

ライブラリを既定のディレクトリより先に*ディレクトリ名*で指定したディレクトリからサーチする。

-l*ライブラリ名*

既定のディレクトリから名前が"lib*ライブラリ名*"であるライブラリをサーチする。

-nostartfiles

リンク時に標準のシステムのスタートアップファイルを使用しない。

-nostdlib

リンク時に標準のシステムライブラリとスタートアップファイルを使用しない。

-rdynamic

リンク時に未使用のシンボルを含むすべてのシンボルをダイナミックなシンボルテーブルに追加する。

-static

ライブラリを静的リンクする。

-**static-nec**

NEC SDKのライブラリを静的リンクする。

-shared

共有オブジェクトを生成する。

-Wl,*option*

リンカ(nld)のオプションを指定する。複数のオプションや引数はコンマ(,)で区切って指定する。

-Xlinker *option*

リンカ(nld)のオプションを指定する。引数を持ったオプションのとき、引数それぞれを本オプションで指定する。

-z *keyword*

リンカ(nld)の-**z**と同じ。

3.13 ディレクトリオプション

--sysroot= *ディレクトリ名*

ヘッダファイルとライブラリをサーチするディレクトリを指定する。ヘッダファイルは*ディレクトリ名*で指定されたディレクトリ配下の**include**ディレクトリ配下、ライブラリは*ディ*

*レクトリ名*で指定されたディレクトリ配下の**lib**ディレクトリ配下からサーチする。

-B *ディレクトリ名*

コマンド、ヘッダファイル、ライブラリをサーチするディレクトリを指定する。コマンドとライブラリは*ディレクトリ名*で指定されたディレクトリ、ヘッダファイルは*ディレクトリ名*で指定されたディレクトリ配下の**include**ディレクトリ配下からサーチする。

3.14 その他オプション

--help

コンパイラの用法を表示する。

-print-file-name=*library*

リンク時に使用されるライブラリファイル(*library*)の絶対パスを出力する。このオプションを指定したとき、コンパイル、および、リンクを行わない。ライブラリファイルが存在しないときは*library*に指定した文字列を出力する。

-print-prog-name=*program*

コンパイル中に呼びだされるコンパイラシステムのコマンド名(*program*)を表示する。このオプションを指定したとき、コンパイル、および、リンクを行わない。指定されたコマンドが存在しないときは*program*で指定した名前を表示する。

-noqueue

コンパイラのライセンス数が使用制限に達しているとき、使用可能になるまで待ち合わせずに終了する。

-v

コンパイルの各ステージで起動されたコマンドを表示する。

--version

コンパイラのバージョンと著作権を表示する。

3.15 オプション指示行で指定できないコンパイラオプション

次のコンパイラオプションは、オプション指示行で指定できません。

- 全体オプション

  -S、**-c**、**-cf**=*conf*、**-fsyntax-only**、**-o** *ファイル名*、**-x** *language*、@*file-name*

- 並列化オプション

  -mno-create-threads-at-startup、**-pthread**

- コード生成オプション
  -no-proginf
- デバッグオプション
  -traceback
- 言語仕様制御オプション
  -traditional、**-cpp**、**-trigraphs**
- メッセージオプション
  -Werror
- プリプロセッサオプション
  -C、**-dD**、**-dI**、**-dM**、**-dN**、**-D***macro*[=*defn*]、**-E**、**-H**、**-include** ファイル名、**-M**、**-MD**、**-MF** ファイル名、**-MP**、**-MT** *target*、**-P**、**-U***macro*、**-undef**
- アセンブラオプション
  -Wa,*option*、**-Xassembler** *option*、**-assembly-list**
- リンカオプション
  -Bdynamic、**-Bstatic**、**-L**ディレクトリ名、**-l**ライブラリ名、**-nostartfiles**、**-nostdlib**、
  -rdynamic、**-static**、**-static-nec**、**-shared**、**-Wl**,*option*、**-Xlinker** *option*、
  -z keyword
- ディレクトリオプション
  --sysroot=ディレクトリ名、**-B**ディレクトリ名
- その他オプション
  --help、**-print-file-name**=*library*、**-print-prog-name**=*program*、**-noqueue**、**-v**、**--version**

3.16 最適化レベルとオプションの既定値

-Onと最適化を個別に制御するオプションの対応は以下のとおりです。
ただし、-Onは最適化全体のレベルを制御するもので、最適化を個別に制御するオプションの有効、無効を同等にしても同じ命令コードが作成されるとは限りませんので注意してください。ある最適化を効果的に適用するには、別の補助的な最適化も適用するなど最適化同士が相互に関連しており、それらが連携して動作するよう-Onで制御しています。例えば-O0に対して、-O1 指定時に既定値として設定される最適化オプションを指定しても -O1 と同じにはなりません。

オプション名	-O4	-O3	-O2	-O1	-O0
-fargument-alias	-	✓	✓	✓	✓
-fargument-noalias	✓	-	-	-	-
-fassociative-math	✓	✓	✓	-	-
-ffast-math	✓	✓	✓	✓	-
-fignore-induction-variable-overflow	✓	-	-	-	-
-fignore-volatile	✓	-	-	-	-
-finline-copy-arguments	-	✓	✓	✓	✓
-floop-collapse	✓	✓	-	-	-
-floop-fusion	✓	✓	-	-	-
-floop-interchange	✓	✓	-	-	-
-floop-normalize	✓	✓	-	-	-
-floop-strip-mine	✓	✓	-	-	-
-floop-unroll	✓	✓	✓	-	-
-floop-unroll-complete=4	✓	✓	✓	-	-
-fmatrix-multiply	✓	✓	-	-	-
-fmove-loop-invariants	✓	✓	✓	✓	-
-fmove-loop-invariants-if	✓	✓	-	-	-
-fmove-loop-invariants-unsafe	✓	-	-	-	-
-fmove-nested-loop-invariants-outer	✓	✓	✓	✓	-
-fnamed-alias	-	-	-	✓	✓
-fnamed-noalias	✓	✓	✓	-	-
-fouterloop-unroll	✓	✓	-	-	-
-freciprocal-math	✓	✓	✓	-	-
-freplace-loop-equation	✓	-	-	-	-
-fstrict-aliasing	✓	✓	✓	-	-

オプション名	-O4	-O3	-O2	-O1	-O0
-fthis-pointer-alias	-	-	-	✓	✓
-fthis-pointer-noalias	✓	✓	✓	-	-
-msched-none	-	-	-	-	✓
-msched-block	✓	✓	✓	✓	-
-mvector	✓	✓	✓	✓	-
-mvector-dependency-test	✓	✓	✓	-	-
-mvector-fma	✓	✓	✓	-	-
-mvector-merge-conditional	✓	✓	-	-	-

第4章　　コンパイラ指示行

本章では、C/C++コンパイラのコンパイラ指示行について説明します。
コンパイラ指示行は以下の形式で記述します。

```
#pragma _NEC コンパイラ指示オプション
```

利用可能なコンパイラ指示オプションについて以降で説明します。

[no]advance_gather

直後のベクトルループ中のベクトル収集命令をループ本体の前方に移動する[しない]。前方に移動することにより、後続の演算命令などとオーバーラップさせ、計算時間を短縮できることがある。

always_inline

本指示行の指定された関数を常にインライン展開の対象とする。呼び出される関数に指定する。ただし、本関数の関数呼出しに**noinline**が有効であるとき、インライン展開されない。**-O***n*[*n*=2,3,4]、**-finline-functions**、**-fopenmp**、または、**-mparallel**が指定されたときのみ有効である。

[no]assoc

演算順序の変更を伴うループ変形を許可する[しない]。

[no]assume

直後の**for**ループのループ長を仮定するとき、配列宣言の利用を許可する[しない]。

atomic

直後の文が、総和、累積などのマクロ演算の式であることを示す。式は、*x* binop=*expr*、*x*++、++*x*、*x*--、--*x*のいずれかの形式でなければならない。詳細については、「7.1.5 ループの強制並列化」を参照してください。

cncall

ループ中に関数の呼出しがあるときでも並列化を許可する。

collapse

多重ループの一重化を許可する。

[no]concurrent

ループの並列化を許可する[しない]。**-mparallel**が有効であるときのみ効果がある。

concurrentに続けて、ループの分割方法を指定するOpenMPと同じ**schedule**句を指定できる。指定できるschedule種別は以下である。

schedule(static [,*chunk-size*])

schedule(dynamic [,*chunk-size*])

schedule(runtime)

[no]dependency_test

データ依存関係の判定をベクトルコード、スカラコードの選択の条件とする条件ベクトル化を許可する[しない]。

forced_collapse

直後の多重ループが一重化可能かどうか不明な場合でも強制的に一重化する。一重化しても予期しない結果、結果不正とならないことはプログラマが保証しなければならない。

gather_reorder

直後の**for**ループ中に現れる非線形添字をもつベクトルロード、ベクトルストアは互いに重なることが無いものと仮定して命令の並べ換えを行うことを許可する。

[no]inline

本指示行の直後の文、ブロック({}で囲まれた文の集まり)、ループ、**if**文およびその複文、**switch**文とそのスイッチ本体に含まれる関数呼出しをインライン展開の対象とする[しない]。**-O***n*[*n*=2,3,4]、**-finline-functions**、**-fopenmp**、または、**-mparallel**が指定されたときのみ有効である。

inline_complete

#pragma _NEC inlineと同様であるが、インライン展開される関数がさらに関数を呼び出しているときその関数もインライン展開の対象とし、関数呼出しがなくなるまでインライン展開を試みる。**-O***n*[*n*=2,3,4]、**-finline-functions**、**-fopenmp**、または、**-mparallel**が指定されたときのみ有効である。

[no]inner

最内側ループの自動並列化を許可する[しない]。最内側ループに指定したときのみ効果がある。

[no]interchange

ループ入れ換えを許可する[しない]。

ivdep

配列の依存関係が不明なとき、ベクトル化不可の依存関係ではないと仮定してベクトル化することを許可する。ベクトル化不可のループをベクトル化し、結果が不正となることがある。

[no]list_vector

同一の非線形添字をもつ同一配列が左辺と右辺に現れる代入文のベクトル化を許可する[しない]。加算、減算のみ含まれる代入文を対象とする。

loop_count(*n*)

ループの繰返し数が不明なとき、繰返し数を整定数*n*仮定する。

[no]loop_count_test

ループ長による条件ベクトル化を許可する[しない]。

[no]lstval

ループ内で定義され、ループの後で値が参照されるかどうかが不明な変数の終値を保証する[しない]。

move / move_unsafe / nomove

move

不変式のループ外への移動を許可する。

move_unsafe

副作用を伴う場合でも、不変式のループ外への移動を許可する。

nomove

不変式のループ外への移動を許可しない。

nofma

ベクトル積和命令を使用したベクトル化を許可しない。

nofuse

直前のループとのループ融合を許可しない。

nosync

配列要素間、ポインタ式の指示先間の依存関係が不明であるときでも、依存関係がないものと仮定して並列化する。

options “*compiler-option* [*compiler-option*]...”

オプション指示行を指定することによりコマンドラインと同じようにコンパイラオプションを指定できる。

指定方法

- オプション指示行は、ソースファイルの先頭に記述する。
- オプション指示行は、複数続けて指定できる。
- 空行、コメント行、**#line**、**#ident**に限り、オプション指示行の直前、間に記述できる。
- オプション指示行は、ソースファイルの先頭の**#include**により取り込まれるファイルの先頭に記述できる。

注意事項

- オプション指示行は継続できない。
- オプション指示行で指定された**-I**で指定されたディレクトリは、オプション指示行の記載されたヘッダファイルのサーチの対象外となる。
- オプション指示行を読み取る際、**#include**により取り込まれるファイルのネスト数の上限値は1000となる。
- 「3.15 オプション指示行で指定できないコンパイラオプション」で示すリンクやコンパイラの動作環境を制御するオプションはオプション指示行で指定できない。
- オプション指示行で **-fopenmp**、**-mparallel**、または、**-ftrace** を指定したとき、リンク時にもそれらのオプションを指定しなければならない。

outerloop_unroll(*n*) / noouterloop_unroll

outerloop_unroll(*n*)

外側ループのアンローリングを許可する。アンロール段数は*n*を超えない最大の2のべき乗数となる。

noouterloop_unroll

外側ループのアンローリングを許可しない。

[no]packed_vector

直後の実行文中に現れる**for**ループ中でパックドベクトル命令の使用を許可する[しない]。

parallel for

ループを強制並列化する。ループが並列実行できることは、プログラマが保証しなければな

らない。詳細については、「7.1.5 ループの強制並列化」を参照してください。

pvreg(*array-name*)

配列*array-name*にメモリの代わりにベクトルレジスタを割り当てる。

コンパイラはこの指定行を含む関数内のすべての**pvreg**指定された配列の参照を、ロード・ストアではなく、ベクトルレジスタの参照として翻訳する。

pvreg指定される配列は以下の条件を満たさなければならない。

- ローカル配列
- 配列のタイプは**int**、**unsigned int**、**float**のいずれかでなければならない
- 1次元配列
- 配列要素の数は最大のパックドベクトル長(=512)以下
- パックドベクトル化されるループ中でのみ定義／参照されること
- すべてのループにおいて添字が同じであること
- **vreg**指定される配列を指定してはならない

retain(*array-name*)

*array-name*の配列、または、ポインタの指示先を可能なかぎりLLC(Last-Level Cache)に保持することを指定する。

備考　この指示行を有効にするとき、**-mretain-list-vector**、または、**-mretain-none**を指定してください。

select_concurrent

多重ループのループのうち、直後のループに対して他のループより優先して自動並列化を適用する。

select_vector

多重ループのループのうち、直後のループに対して他のループより優先して自動ベクトル化を適用する。

shortloop

ループの繰返し数が、システムの最大ベクトルレジスタ長（=256）を超えないものとしてベクトル化することを許可する。

[no]shortloop_reduction

リダクション演算を含むループの繰返し数による条件ベクトル化を許可する[しない]。

-fassociative-mathが有効であるときのみ効果がある。

[no]sparse

sparse

ループ中の条件下の数学関数の実行回数が、ループの繰返し数より非常に小さいものとして、ベクトル化する。

nosparse

ループ中の条件文下の数学関数の実行回数が、ループの繰返し数に近いものとして、ベクトル化する。

unroll(*n*) / nounroll

unroll(*n*)

ループを*n*段にアンロールすることを許可する。

nounroll

ループのアンローリングを許可しない。

unroll_complete

直後のループの繰返し数が定数でコンパイル時に算定できるとき、そのループをループ展開(完全ループアンローリング)することを許可する。

なお、コンパイル時に繰返し数を算定するため、ループは次の形式でなければならない。

var*N*(*N*=1,2,..)は変数とする。また、それらはループ内で定義されてはならない。さらにvar*N*、定数*N*の型は同じでなければなりません。

定数*3*の値が1であるとき、最右側のvar1を更新する式では++、--演算子を使用してもよい。

- for (var1 = 定数1; var1 <= 定数2; var1 = var1 + 定数3)
- for (var1 = 定数1; var1 < 定数2; var1 = var1 + 定数3)
- for (var1 = 定数1; var1 > 定数2; var1 = var1 - 定数3)
- for (var1 = 定数1; var1 >= 定数2; var1 = var1 - 定数3)
- for (var1 = var2 + 定数1; var1 <= var2 + 定数2; var1 = var1 + 定数3)
- for (var1 = var2 + 定数1; var1 < var2 + 定数2; var1 = var1 + 定数3)
- for (var1 = var2 + 定数1; var1 > var2 + 定数2; var1 = var1 - 定数3)
- for (var1 = var2 + 定数1; var1 >= var2 + 定数2; var1 = var1 - 定数3)
- for (var1 = var2 - 定数1; var1 <= var2 + 定数2; var1 = var1 + 定数3)

- for (var1 = var2 - 定数1; var1 < var2 - 定数2; var1 = var1 + 定数3)
- for (var1 = var2 - 定数1; var1 > var2 - 定数2; var1 = var1 - 定数3)
- for (var1 = var2 - 定数1; var1 >= var2 - 定数2; var1 = var1 - 定数3)

  備考　別名としてunroll_completelyも使用できる。

[no]vector

自動ベクトル化を許可する[しない]。

vector_threshold

直後のループのベクトル化を行う最小の繰返し数を*n*回とする。

[no]verror_check

ベクトル化された部分から呼び出されたベクトルバージョンの数学関数の引数の値の範囲を実行時に検査する[しない]。

[no]vob

直後の実行文中に現れる**for**ループの終了後に実行されるスカラロード、スカラストア、ベクトルロードが、その**for**ループ中のベクトルストアを追い越すことを許可しない[する]。

多重ループの外側ループに指定したとき、その内側ループには効果が無い。

[no]vovertake

直後の実行文中に現れる**for**ループ中のベクトルストアを、後続のスカラロード、スカラストア、ベクトルロードが追い越して実行することを許可する[しない]。

- **for**ループ中のベクトルストアと、ループ中またはループ後のスカラロード、スカラストア、ベクトルロードの領域に重なりがあるとき、実行結果が不正になることがある。
- 多重ループの外側ループに指定したとき、その内側ループには効果が無い。

vreg(*array-name*)

配列*array-name*にメモリの代わりにベクトルレジスタを割り当てる。

コンパイラはこの指定行を含む関数内のすべての**vreg**指定された配列の参照を、ロード／ストアではなく、ベクトルレジスタの参照として翻訳する。

vreg指定される配列は以下の条件を満たさなければならない。

- ローカル配列
- 配列のタイプはint、unsigned int、long、unsigned long、long long、unsigned long long、float、doubleのいずれかでなければならない

- 1次元配列
- 配列要素の数は最大のベクトル長(=256)以下
- ベクトル化されるループ中でのみ定義／参照されること
- すべてのループにおいて添字式の値が同じであること
- **pvreg**指定される配列を指定してはならない

[no]vwork

部分ベクトル化(ループ分割によるベクトル化)を許可する[しない]。**novwork**を指定した場合、外側ループ、およびベクトル化不可の部分を含むループは、ループ全体がベクトル化されなくなる。

第5章　　最適化・ベクトル化

本章では、コンパイラのもつ最適化、自動ベクトル化によるプログラムの高速化にかかわる機能を説明します。

5.1　コードの最適化

コードの最適化は、プログラム中の制御の流れやデータの流れを解析することによって、不要な演算を可能なかぎり削除し、また、ループ中の演算を必要最小限のものとします。さらに可能ならば、演算をそれと等価なより高速な演算と置き換えることにより、プログラム実行時間の短縮を図ります。

5.1.1　コンパイラの適用する最適化

C/C++コンパイラは次の最適化を適用します。()内でそれらを有効にするオプションを示します。

- 共通式の削除（**-O**[*n*](*n*=1,2,3,4)）
- 条件下の不変式のループ外への移動（**-O**[*n*](*n*=1,2,3,4)，**-fmove-loop-invariants**，**-fmove-loop-invariants-unsafe**）
- 単純代入の削除（**-O**[*n*](*n*=1,2,3,4)）
- 不要コードの削除（**-O**[*n*](*n*=1,2,3,4)）
- べき乗の最適化（**-O**[*n*](*n*=1,2,3,4)）
- 除算の乗算化（**-O**[*n*](*n*=2,3,4)，**-freciprocal-math**）
- ループ融合（**-O**[*n*](*n*=3,4)）
- volatile修飾子の無視（-O[*n*](*n*=4)，**-fignore-volatile**）
- 定数の演算・型変換のコンパイル時計算（**-O**[*n*](*n*=1,2,3,4)）
- 複素数演算の最適化（**-O**[*n*](*n*=1,2,3,4)）
- 単項演算子のマイナスの除去（**-O**[*n*](*n*=1,2,3,4)）
- 分岐に関する最適化（**-O**[*n*](*n*=1,2,3,4)）
- 演算の強度縮小とテスト変数の置換（**-O**[*n*](*n*=1,2,3,4)）
- 不要終値保証の除去（**-O**[*n*]**-O**[*n*](*n*=1,2,3,4)）
- 組込み関数のインライン展開（**-O**[*n*](*n*=1,2,3,4)）

- 命令の並べ換えによる最適化（**-msched**-*keyword*）

5.1.2　最適化による副作用

- 式、コードの削除により、計算を行う場所、回数が変わり、エラーの発生位置、回数が最適化を適用しなかった場合と比べて変わることがあります。
- 条件下の不変式のループ外への移動により、実行されないはずの式が実行され、本来発生しないはずのエラーや演算例外が発生することがあります。
- べき乗の最適化を適用したとき、アンダーフローが発生しても例外を検出しません。
- 除算の乗算化により、演算結果にわずかの差異が生じます。浮動小数点数の演算では、この差異はほとんど無視してかまいませんが、無視したくないときコンパイラオプションで最適化を抑止してください。
- 命令の並べ換えによる最適化により、ある条件が成立したときのみ実行される計算が、基本ブロックをまたがって移動され、常に実行されるようになると、発生しないはずの実行時エラーが検出されることがあります。また、コンパイル時間やコンパイラが使用するメモリ量を著しく増加させることがあります。

5.2　ベクトル化機能

5.2.1　ベクトル化

変数や配列の各要素のことをスカラデータと呼びます。これに対し、行列の行要素、列要素、あるいは対角要素など、規則的に並んだデータ列をベクトルデータと呼びます。ベクトルデータを処理するスカラ命令列を、等価な処理を行うベクトル命令で置き換えることをベクトル化といいます。コンパイラがプログラムを解析してベクトル命令で実行可能な部分を自動的に検出し、その部分に対してベクトル命令を生成することを自動ベクトル化といいます。**-O**[*n*](*n*=1,2,3,4)が有効であるとき、この最適化が有効になります。
この最適化を制御するコンパイラオプションは、**-mvector**です。
この最適化を制御するコンパイラ指示オプションは、**[no]vector**です。

5.2.2　部分ベクトル化

ループにベクトル化可能な部分とベクトル化不可の部分が混在しているとき、そのループをベクトル化可能な部分(ベクトルコード部分)とベクトル化不可の部分(スカラコード部分)に分割し、ベクトル化可能な部分のみをベクトル化します。このベクトル化方法は、部分ベクトル化と呼ばれます。**-O**[*n*](*n*=1,2,3,4)が有効であるとき、この最適化が有効になります。
この最適化を制御するコンパイラオプションは、**-mwork-vector-kind**=noneです。

この最適化を制御するコンパイラ指示オプションは、**[no]vwork**です。

5.2.3　マクロ演算

以下のようなパターンの式はベクトル化の定義・参照の条件を満たしていませんが、コンパイラは特別なパターンと認識し、専用のベクトル命令を用いてベクトル化します。**-O**[*n*] (*n*=1,2,3,4)が有効であるとき、この最適化が有効になります。

(1)　総和/内積

```
S =   S   ±   exp                                   (exp:式)
```

次のような、複数の文から構成されている総和/内積もベクトル化されます。

```
t1=   S   ±   exp1                                  (expi:式)
t2=   t1  ±   exp2
      ...
S =   tn  ±   expn
```

-mvector-reductionで制御します。

(2)　累積

```
S =   S   *   exp                                   (exp:式)
```

次のような、複数の文から構成されている累積もベクトル化されます。

```
t1=   S   *   exp1                                  (expi:式)
t2=   t1  *   exp2
      ...
S =   tn  *   expn
```

-mvector-reductionで制御します。

(3)　漸化式

```
a[i]      =   exp1     ±   a[i-1];                  (expi:式)
a[i]      =   exp1     *   a[i-1];
a[i]      =   exp1     ±   a[i-1]      *   exp2;
a[i]      =   (exp1    ±   a[i-1])     *   exp2;
```

次のような複数の文から構成される漸化式もベクトル化されます。

```
t = exp1 ± a[i-1];                                    (expi:式)
a[i] = t * exp2
```

-mvector-iteration、および、-mvector-iteration-unsafeで制御します。

(4)　最大値/最小値

(a)　最大値/最小値のみを求める

例

```
for (i = 0; i < n; i++) {
    if (a[i] > amx)
        amx = a[i];
}
```

(b)　最大値/最小値とインデックスを求める

例

```
for (i = 0; i < n; i++) {
    if (a[i] > amx) {
        amx = a[i];
        ix = i;
    }
}
```

(c)　最大値/最小値とインデックス、および、その他の値を求める

例

```
for (j = 0; j < n; j++) {
    for (i = 0; i < n; i++) {
        if (a[i][j] > amx) {
            amx = a[i][j];
            ix = i;
            iy = j;
        }
    }
}
```

(5)　サーチ

ある条件を満たす要素をサーチするループをベクトル化します。

例

```
for (i = 0; i < n; i++) {
    if (a[i] == 0)
        break;
}
```

このとき、ループは以下の条件を満たさなければなりません。

- 最内側のループである。
- ループの外への分岐はただ一つである。
- ループの外への分岐条件はループの繰返しに依存する。
- ループの外への分岐の前に配列要素、ポインタの指示先への代入がない。
- ループの外への分岐以外は、ベクトル化の条件をすべて満たしている。

(6)　圧縮

ある条件を満たす要素を圧縮するループをベクトル化します。

例

```
j = 0;
for (i = 0; i < N; i++) {
    if (x[i] >= 0.0) {
        j = j + 1;
        y[j] = z[i];
    }
}
```

(7)　伸長

ある条件を満たす要素に値を伸長するループをベクトル化します。

例

```
j = 0;
for (i = 0; i < N; i++) {
    if (x[i] >= 0.0) {
        j = j + 1;
        z[i] = y[j];
    }
}
```

5.2.4　条件ベクトル化

条件ベクトル化とは、一つのループに対してベクトル化したコードとスカラのコード、特定のパターンのみ高速に実行できるコードなど、数種類のコードを用意し、実行時に条件を調べて、適切なコードを選択して実行するようなループの変形のことをいいます。実行時に調べる条件は以下があります。

- 依存関係
- ループ長
- リダクション演算を含むループのループ長

  -O[*n*](*n*=2,3,4)が有効であるとき、この最適化が有効になります。この最適化を制御するコンパイラオプションは、実行時に調べる条件ごとに以下です。

- 依存関係による条件ベクトル化の制御は、**-mvector-dependency-test**です。
- ループ長による条件ベクトル化の制御は、**-mvector-loop-count-test**です。
- リダクション演算を含むループのループ長による条件ベクトル化の制御は、**-mvector-shortloop-reduction**です。

  この最適化を制御するコンパイラ指示オプションは、実行時に調べる条件ごとに以下です。

  - 依存関係による条件ベクトル化の制御は、**dependency_test**です。
  - ループ長による条件ベクトル化の制御は、**loop_count_test**です。
  - リダクション演算を含むループのループ長による条件ベクトル化の制御は、**[no]shortloop_reduction**です。

5.2.5　外側ループストリップマイニング

繰返し数が最大ベクトルレジスタ長(=256)より長いループをベクトル化するとき、コンパイラは内部的に繰返し数を最大ベクトルレジスタ長に収まるように分割しています。これをストリップマイニングと呼びます。密な多重ループの内側ループに、外側ループのインダクション変数を添字式に含まない配列要素があるとき、コンパイラはストリップループを外側に移動します。**-O**[*n*](*n*=3,4)が有効であるとき、この最適化が有効になります。

この最適化を制御するコンパイラオプションは、**-floop-strip-mine**です。

備考　密な多重ループとは、外側ループと内側ループの間に実行文が現れない多重ループのことです。

例　　密な多重ループ

```
for (k = 0; k < 10; k++) {
    for (j = 0; j < 20; j++) {
        for (i = 0; i < 30; i++) {
            a[k][j][i] = b[k][j][i] * c[k][j][i];
        }
    }
}
```

例　　密でない多重ループ

```
for (k = 0; k < 10; k++) {
    for (j = 0; j < 20; j++) {
        for (i = 0; i < 30; i++) {
            a[k][j][i] = b[k][j][i] * c[k][j][i];
        }
        x[j][k] = y[j][k] + z[j][k];
    }
}
for (k = 0; k < 10; k++) {
    for (j = 0; j < 20; j++) {
        for (i = 0; i < 30; i++) {
            a[k][j][i] = b[k][j][i] * c[k][j][i];
        }
        for (i = 0; i < 30; i++) {
            s[k][j][i] = t[k][j][i] * u[k][j][i];
        }
    }
}
```

5.2.6　ショートループ

ベクトル化されたループのうち、繰返し数が最大ベクトルレジスタ長(=256)以下のループに対して、ループの繰返しの終了判定が省略された命令コードが生成されます。このループはショートループと呼ばれます。**-O**[*n*](*n*=1,2,3,4)が有効であるとき、この最適化が有効になります。

この最適化を制御するコンパイラ指示オプションは、**shortloop**です。

5.2.7　パックドベクトル命令

ベクトルレジスタの各要素を32ビットで二つに分割して、ベクトルレジスタの各要素に2つ

のデータを格納することをパックといいます。パックしたデータに対して演算を行う命令をパックドベクトル命令といいます。パックドベクトル命令はベクトル命令の2倍のデータを1命令で処理することができます。

パックドベクトル命令の使用を制御するコンパイラオプションは、**-mvector-packed**です。
パックドベクトル命令の使用を制御するコンパイラ指示オプションは、**[no]packed_vector**です。

5.2.8　その他のベクトル化コードに適用する最適化

共通式の削除、単純代入の削除、不要コードの削除、除算の乗算化、不要終値保証の除去は、ベクトル化されたコードに対しても行います。その他に、C/C++コンパイラはベクトル化したコードに対して、次の最適化を適用します。()内でそれらを有効にするオプションを示します。

- スカラ演算のくくりだし（**-O**[*n*](*n*=1,2,3,4)）
- 文の入れ換えによるベクトル化（**-O**[*n*](*n*=1,2,3,4)）
- ループの一重化（**-O**[*n*](*n*=3,4)，**-floop-collapse**）
- 外側ループのアンローリング（**-O**[*n*](*n*=3,4)，**-fouterloop-unroll**）
- ループのリローリング（**-O**[*n*](*n*=3,4)）
- 行列積ループの認識（-O[*n*](*n*=3,4)，-fassociative-math，-fmatrix-multiply）
- ループ展開（**-O**[*n*](*n*=2,3,4)，**-floop-unroll-complete**=*m*）

5.2.9　ベクトル化機能使用時の注意事項

- 総和演算は、演算順序がベクトル化したときとしないときとで違うため、演算結果が異なることがあります。累積、漸化式、内積演算も同様です。
- 8バイト整数型の漸化式は浮動小数点型のベクトル漸化式命令を用いてベクトル化されます。このため結果の精度が52bitを超えるときや、実数型のオーバーフローが発生するとき、実行結果はベクトル化しないときと異なります。
- ベクトル版の数学関数で使用している計算方法は、高速化のため、スカラ版と必ずしも同じ結果にはなりません。
- ベクトル化したときとしないときとで、除算の乗算化などの最適化の適用のされ方が異なります。
- ベクトル化したときとしないときとで、演算順序の変更などの最適化の適用のされ方が異

なります。

- ベクトル化すると、数学関数で検出されるエラーや演算例外の検出のされ方が、ベクトル化しないときと異なります。
- コンパイラは、配列の定義・参照関係がベクトル化により正しく保たれるか否かを検査する際、個々の添字式の値が配列宣言の対応する次元の上限、下限の間に収まっていることを前提とします。したがって、この条件に反するループをベクトル化した場合、その実行結果は保証されません。
- **if**文、**switch**文、条件演算子(?)を含むループがベクトル化されると、条件付きで実行される部分に関し、演算は必要な部分しか行われませんが、配列要素、ポインタの指示先は、ループ構造で決まる繰返し数分だけ参照されます。すなわち、本来参照されない配列要素、ポインタの指示先が参照されます。したがって、配列やポインタの指示先については繰返し数分だけの領域を用意しておかないと、メモリアクセス例外を起こすことがあります。
- 飛び出しを含むループがベクトル化されると、飛び出しが起こる繰返し回より後の繰返しも実行されます。したがって、本来実行されない演算が実行されたり、本来参照されないデータが参照されることにより、エラーや例外が発生することがあります。
- ベクトル処理できるデータのアラインメントは、そのデータの型のサイズと同じ（4バイト、または、8バイト）でなければなりません。アラインメントの条件を満たさない配列要素、ポインタの指示先の参照、定義を含むループがベクトル化されたとき、実行時に例外となることがあります。そのとき、コンパイル時に**-mno-vector**を指定してプログラム全体のベクトル化をやめるか、**#pragma _NEC novector**を指定し、問題のループのベクトル化を抑止してください。ベクトル処理できるアラインメントを満たさなくなる可能性のあるデータは以下です。
  - 仮引数
  - ポインタで指示されるデータ
    コンパイラは、これらのデータをアラインメントの条件を満たすものと仮定してベクトル化します。

第6章　　インライン展開機能

6.1　自動インライン展開

コンパイラがソースファイルを解析、サーチして、インライン展開すべき関数呼出しを選択し、インライン展開します。
この最適化を制御するコンパイラオプションは、**-finline-functions**です。

6.2　明示的インライン展開

6.2.1　説明

明示的インライン展開では、利用者がインライン展開を指示する指示行(インライン展開指示行)をソースファイルに指定し、インライン展開します。このとき**-finline-functions**の指定は必要ありません。ただし、インライン展開指示行は**-O***n*[*n*=2,3,4]、**-finline-functions**、**-fopenmp**、または、**-mparallel**が指定されたときのみ有効です。
インライン展開を指示する指示行には次のものがあります。

- **always_inline**
  本指示行の指定された関数を常にインライン展開の対象とする。呼び出される関数に指定する。ただし、本関数の関数呼出しに**noinline**が有効であるとき、インライン展開されない。
- **inline**
  本指示行の直後の文、ブロック({}で囲まれた文の集まり)、ループ、**if**文およびその複文、**switch**文とそのスイッチ本体に含まれる関数呼出しをインライン展開の対象とする。
- **inline_complete**
  inlineと同様であるが、インライン展開される関数がさらに関数を呼び出しているときその関数もインライン展開の対象とし、関数呼出しがなくなるまでインライン展開を試みる。
- **noinline**
  本指示行の直後の文、ブロック({}で囲まれた文の集まり)、ループ、**if**文およびその複文、**switch**文とそのスイッチ本体に含まれる関数呼出しをインライン展開しない。**always_inline**の指定された関数もインライン展開しない。

6.2.2　インライン展開指示行の指定

(1)　呼び出される関数

always_inlineは呼び出される関数に指定します。

例

```
double small_func(double a)
{
#pragma _NEC always_inline
return sqrt(a);
}
```

(2)　直後の文

inline / **inline_complete** / **noinline**は、直後の文中のすべての関数呼出しに作用します。

例

```
#pragma _NEC inline
    x = func1(a) + func2(a);
    x += func3(a);
```

func1()、func2()の呼び出しをインライン展開対象とする。func3()は対象ではない。

(3)　ブロック

inline / **inline_complete** / **noinline**は、直後のブロック中のすべての関数呼出しに作用します。

例

```
#pragma _NEC inline
    {
        func1();
        func2();
    }
```

func1()、func2()の呼び出しをインライン展開対象とする。

(4)　ループ

inline / inline_complete / noinlineは、直後のfor、while、do-while文、および、ループ本体中のすべての関数呼出しに作用します。

例

```
#pragma _NEC inline
    for (i = ifunc(); i < 1000; i++) {
        z[i] = func1();
        w[i] = func2();
    }
```

ifunc()、func1()、func2()の呼び出しをインライン展開対象とする。

(5)　if 文、switch 文

inline / **inline_complete** / **noinline**は、直後の**if**、**switch**文、および、それらの副文中のすべての関数呼出しに作用します。

例

```
#pragma _NEC inline
    if (ifunc1()) {
        x = func1();
    }
    else if (ifunc2()) {
        x = func2();
    }
    else {
        x = func3();
    }
```

ifunc1()、ifunc2()、func1()、func2()、func3()の呼び出しをインライン展開対象とする。

6.2.3　注意事項

- always_inline / inline / inline_complete / noinlineは、-O*n*[*n*=2,3,4]、-finline-functions、-fopenmp、または、-mparallelが指定されたときのみ有効です。
- __attribute__((noinline))と**always_inline**が指定されたとき、**always_inline**の指定は無視されます。
- **always_inline**を指定した関数の定義は削除されません。_attribute__((always_inline))を指定したとき、関数定義が削除されることに注意してください。
- **noinline**指定された関数呼出しから**always_inline**を持つ関数を呼び出しているとき、その関数はインライン展開しない。

- ブロックがネストしており、それぞれに相反する指示行が指定されているとき、内側のブロックについてはそのブロックに指定された指示行が優先適用されます。

  例

```
#pragma _NEC inline
    {
        x = func1();              // インライン展開の対象
#pragma _NEC noinline
        {
            y = func2();          // インライン展開しない
        }
    }
```

- 宣言の初期化子に現れた関数呼出しは**inline**を指定してもインライン展開されない。

  例

```
#pragma _NEC inline
    double x = func1();      // インライン展開しない
```

6.3　クロスファイルインライニング

コンパイル対象のソースファイルとは別のソースファイルに含まれる関数をインライン展開することをクロスファイルインライニングと呼びます。
C/C++コンパイラでは、インライン展開する関数をサーチする別のソースファイルを指定することでクロスファイルインライニングできます。
以下にサーチするソースファイルの指定例を示します。

(1)　ソースファイルを指定

```
$ ncc -c -finline-functions -finline-file=sub.c call.c
```

(2)　ソースファイルとコマンドラインで指定されたすべてのソースファイルを指定

```
$ ncc -c -finline-functions -finline-file=sub2.c:all call.c sub.c
```

(3)　ディレクトリにあるすべてのソースファイルを指定

```
$ ls dir
sub.c sub2.c sub3.c
$ ncc -c -finline-functions -finline-directory=dir sub.c
```

(4)　ディレクトリにあるすべてのソースファイルを指定したが、あるソースファイルを除外

```
$ ls dir
sub.c sub2.c sub3.c
$ ncc -c -finline-functions -finline-directory=dir -fno-inline-file=sub2.c call.
c
```

サーチ対象にソースファイルを指定する他にILファイルを指定する方法もあります。コンパイルするソースファイルが多いとき、この方法の方がコンパイル時間を短縮できる場合があります。

(5)　IL ファイルを生成後、IL ファイルを読み込んで関数をサーチする。

```
$ ncc -mgenerate-il-file sub.c
$ ncc -c -finline-functions -mread-il-file sub.cil main.c
```

6.4　インライン展開の阻害要因

インライン展開の阻害要因として次の項目があります。

- インライン展開される関数が見つからない。
- 呼出し元の関数の引数がインライン展開される関数の引数に一致しない。
- 呼出し元の関数とインライン展開される関数に、同じ名前でサイズが異なる**union**が含まれる。
- インライン展開される関数で引用される関数名が、呼出し元の関数で使用される非関数名と衝突する。
- インライン展開される関数にOpenMPディレクティブが指定されている。
- インライン展開される関数に再帰的な関数呼出しが含まれる。

6.5　インライン展開機能使用時の注意事項

- インライン展開機能を利用するとき、インライン展開後のオブジェクトファイルの大きさに注意してください。多数の関数をインライン展開してしまうと、プログラムのコードサイズが肥大化し、命令キャッシュからコードがあふれてしまい、プログラム全体の実行性能が低下することがあります。
- 関数呼出しが再帰的に行われる(リカーシブな関数呼出しを含む)とき、その関数呼出しはインライン展開されません。

- クロスファイルインライニングで、大きなプログラムや多数のプログラムをサーチする場合、それ以降の処理量が増えて、コンパイル時間が延びたり、コンパイル時に使用するメモリ量が増加したりすることがあります。
- クロスファイルインライニングは、C言語でのみ利用可能です。

第7章　　自動並列化・OpenMP 並列化機能

本章では、自動並列化、OpenMP並列化機能、それらの利用にあたって留意すべき項目について説明します。

7.1　自動並列化機能

7.1.1　自動並列化

自動並列化機能は、プログラム内の並列実行できるループや文の集まりを抽出し、並列処理できるようにプログラムを変形、および、並列処理制御のための処理の挿入を自動的に行います。

この最適化を制御するコンパイラオプションは、**-mparallel**です。

7.1.2　作業量による条件並列化

一般に、並列処理はオーバーヘッドを伴うため、分割された各ループの繰返しが、それぞれに十分な作業量をもっていなければ、プログラムの実行時間を増大させることになります。繰返し数がコンパイル時に計算できないとき、その多重ループについて、並列化したコードとそうでないコードの両方を生成しておき、実行時に繰返し数の大小によって適切な方が実行されるようにします。これを「作業量による条件並列化」と呼びます。作業量による条件並列化では、ループ中の演算の数などから、その多重ループが並列化に適している繰返し数の下限値(しきい値)が計算されます。実行時には、このしきい値と、多重ループの繰返し数が比較され、繰返し数がしきい値以上ならば並列化されたコードが実行され、小さければ並列化されていないコードが実行されるようにIF 文などが生成されます。

この最適化を制御するコンパイラオプションは、**-mparallel-threshold**=*n*です。

7.1.3　依存関係による条件並列化

ループ中に、コンパイル時に不明なデータ依存関係が存在して並列化を妨げているとき、自動並列化機能は作業量による条件ベクトル化と同様に、依存関係を実行時にテストするためのコードを生成し、依存関係により条件並列化します。1重、または、2重ループで、しきい値のテストが行われるとき、依存関係のテストも同時に行われます。

7.1.4　最内側ループの並列化

外側ループによる適切な並列化ができないとき、最内側のループも並列化の対象とします。本最適化を有効にしない限り、最内側ループは、繰返し数がしきい値を超えることが明らかなときのみ並列化されます。

この最適化を制御するコンパイラオプションは、**-mparallel-innerloop**です。

7.1.5　ループの強制並列化

プログラマはある**for**ループが並列化可能なことを知っているが、コンパイラには並列化可能であることが認識できないため自動並列化が行われないようなとき、強制並列化指示行の**#pragma _NEC parallel for**によりループを並列化できます。このとき、ループが並列実行できることは、プログラマが保証しなければなりません。

ループは、「**for** (*init-expr*; *var relational-op*. b; *incr-expr*)」の形式の**for**文によるもので、各項は次の条件を満たさなければならなりません。

- *init-expr*は、*var*=lb、または、integer-type *var*=lbのいずれか
- *incr-expr*は、++*var*、*var*++、--*var*、*var*--、*var*+=*incr*、*var*-=*incr*、*var*=*var*+*incr*、*var*=*var*-*incr*のいずれか
- *var*は、**int**、**long**、**long int**、**long long**、**long long int**型のスカラ変数
- *relational-op*は、**<**、**<=**、**>**、**>=**、**!=**のいずれか
- *lb*、*b*、*incr*はループ内不変式

parallel forに続けて、ループの分割方法を指定するOpenMPと同じ**schedule**句を指定できます。指定できるスケジューリング種別は以下です。

- **schedule**(static [,*chunk-size*])
- **schedule**(dynamic [,*chunk-size*])
- **schedule**(runtime)

また、OpenMPと同じ**private**句も指定できます。

- **private**(*スカラ変数名*[,*スカラ変数名*]...)

ループが、総和、累積などのマクロ演算の文を含むとき、**#pragma _NEC atomic**をその文の直前に指定します。式は、x binop=expr、x++、++x、x--、--xのいずれかの形式でなければならず、さらに、各項は次の条件をすべて満たさなければなりません。

- xは、代入可能なスカラ変数
- exprは、xの引用を含まないスカラ式
- binopは、+、*、-、/、&、^、|、<<、>>のうちのいずれか一つの演算子、また、その演算子はオーバロードされていてはならない。

強制並列化指示行の指定例を以下に示します。

例

```
double sub (double *a, int n)
{
    int i, j;
    double b[n];
    double sum = 1.0;
    ...
#pragma _NEC parallel for schedule(dynamic,16)
    for (j = 0; j < n; j++) {
        for (i = 0; i < n; i++) {
#pragma _NEC atomic
            sum += a[j] + b[i];
        }
    }
    ...
    return sum;
}
```

7.1.6　並列化処理機能使用時の注意事項

- 並列処理時の総CPU時間は、並列処理のオーバーヘッドにより増加します。
- 関数呼出しを含む関数を並列化するとき、共有データの定義、参照が不正にならないかどうか、呼び出される関数の中まで調べなければなりません。
- **-fopenmp**と**-mparallel**が同時に指定されたとき、ループがOpenMPの並列区間の外側にある場合は外側のループが自動並列化の対象となります。OpenMPディレクティブを含むルーチンを自動化したくないとき、**-mno-parallel-omp-routine**を指定してください。

7.2　OpenMP並列化

7.2.1　OpenMP 並列化の利用

OpenMPを利用するには、コンパイル、リンク時に**-fopenmp**を指定します。OpenMPディレクティブや注意事項の詳細については、OpenMP仕様を参照してください。

例　　OpenMPディレクティブの挿入

```
double sub (double *a, int n)
{
    int i, j;
    double b[n];
    double sum = 1.0;
    ...
#pragma omp parallel for reduction(+:sum)
    for (j = 0; j < n; j++) {
        for (i = 0; i < n; i++) {
            sum += a[j] + b[i];
        }
    }
    ...
    return sum;
}
```

7.2.2　OpenMP Version 5.0

OpenMP Version 5.0で追加された構文のうち、以下の構文、機能が利用できます。

- **loop**構文
- **parallel loop**構文
- **parallel master**構文
- OMPT interface

7.2.3　OpenMP 並列化に対する拡張機能

OpenMP Version 4.5で規定されている環境変数について、環境変数名に接頭子”VE_”をつけた環境変数も利用できます。接頭子”VE_”のあり・なし両方の環境変数が指定されている場合には、接頭子”VE_”ありの環境変数で指定した値が有効となります。

例　　環境変数の指定(**VE_OMP_NUM_THREADS** が有効)

```
$ export OMP_NUM_THREADS=4
$ export VE_OMP_NUM_THREADS=8
```

7.2.4　OpenMP 並列化に対する制限事項項

以下の機能は利用できません。

- “Device Constructs”で定義されている機能

コンパイラはデバイスに対するコードを一切生成せず、TARGETリージョンはホスト上で実行されます。

- **reduction**句の中以外で現れる、“Array Sections”で定義されている構文
- “Cancellation Constructs”で定義されている機能
- “Controlling OpenMP Thread Affinity”で定義されている機能
- **distribute**、**target**、**teams**
  複合構文のための指示行中の**distribute**、**target**、**teams**、および、それらに関わる指示句は無視されます。
  例：”**target parallel for**”は“**parallel for**”としてコード生成します。
- **taskloop**構文
- **parallel for simd**構文、および**for simd**構文
  それぞれ**parallel for**構文、**for**構文としてコード生成します。
- **simd**構文
  safelen句、または、**simdlen**句が指定されていない場合、ivdep指示行が指定されているものとみなします。
- **declare reduction**構文
- **allocate**句
- **bind**句
- **if**句でのdirective-name-modifier指定
- **in_reduction**句、**task_reduction**句
- **ordered**句での引数指定
- **schedule**句での*modifier*指定
- 配列が指定された**depend**句
- **depend**句での*dependence-type*指定の**source**、**sink**
- **critical**構文での**hint**指定
- **atomic**構文での**seq_cst**指定
- **linear**句での*modifier*指定
- ネスト並列性

7.3　スレッド制御

7.3.1　スレッド数の指定・取得

自動並列化されたプログラムでは、OpenMP並列機能をベースに並列処理を実現しています。このため、自動並列化されたプログラム、OpenMP並列化されたプログラムの実行では、環境変数**OMP_NUM_THREADS**、または、**VE_OMP_NUM_THREADS**により実行に使用するスレッド数を指定できます。

また、自動並列化されたプログラムにおいても、OpenMPの実行時ルーチンで、スレッド数を指定、取得できます。

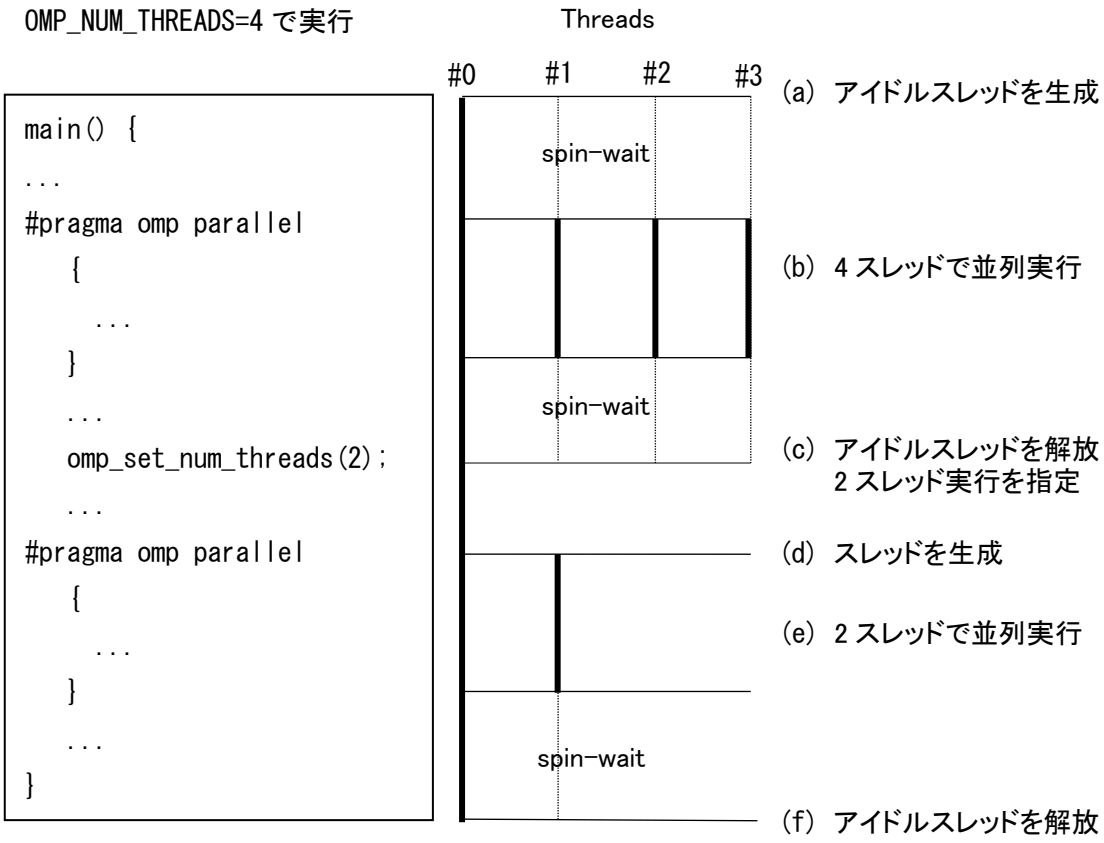
```
extern void omp_set_num_threads(int num_threads);    // スレッド数の設定
extern int omp_get_num_threads(void);                // スレッド数の取得
extern int omp_get_max_threads(void);                // 利用可能スレッド数の取得
extern int omp_get_thread_num(void);                 // スレッド番号の取得
```

プログラムの実行開始前に環境変数**OMP_NUM_THREADS**、または、**VE_OMP_NUM_THREADS**によりスレッド数が指定されなかったとき、プログラムで利用可能なVEコア数と同じスレッド数でプログラムの実行が開始されます。

7.3.2　スレッドの生成・解放

自動並列化されたプログラム、OpenMP並列化されたプログラムの実行では、main()の実行開始前にスレッドが自動生成され、プログラムの終了時に解放されます。

次の図を参照してスレッドの生成、解放について説明します。環境変数**OMP_NUM_THREADS**に4が設定されているものとします。



(a) main()の開始前に、マスタースレッド(#0)により三つのアイドルスレッドが生成されます。アイドルスレッドは、スピンウェイトしながら仕事が割り当てられるのを待ちます。

(b) 並列リージョンに到達すると仕事が割り当てられ、計算が並列実行されます。並列リージョンが終了すると、スレッドはスピンウェイトを開始し、再度仕事が割り当てられるのを待ちます。

(c) 実行時ルーチンomp_set_num_threads(2)で、以降の並列リージョンを2スレッドで実行するよう指定しています(ICV値を2に変更)。ここですべてのアイドルスレッドが解放されます。

(d) 並列リージョンの開始直前で一つのスレッドが生成され、二つのスレッドで並列リージョンの実行が開始されます。

(e) 並列リージョンを2スレッドで並列実行します。

(f) プログラム終了直前に、すべてのアイドルスレッドが解放され、プログラムを終了します。

7.3.3　スレッド生成の遅延

既定の動作ではmain()の開始前にスレッドが生成されますが、リンク時に次のコンパイラオ

プションを指定することで、最初に実行される並列リージョンの直前まで、スレッドの生成を遅延させることができます。

```
$ ncc -fopenmp -mno-create-threads-at-startup -static-nec a.o
$ ncc -mparallel -mno-create-threads-at-startup -static-nec b.o
```

7.4　注意事項

- 並列処理時の総CPU時間は、並列処理のオーバーヘッドにより増加します。
- 関数呼出しを含む関数を並列化するとき、共有データの定義、参照が不正にならないかどうか、呼び出される関数の中まで調べなければなりません。
- **-fopenmp**と**-mparallel**が同時に指定されたとき、ループがOpenMPの並列区間の外側にある場合は外側のループが自動並列化の対象となります。OpenMPディレクティブを含むルーチンを自動並列化したくないとき、**-mno-parallel-omp-routine**を指定してください。
- MPIプログラムのとき、MPIプロセスごとにその開始直前でスレッドが生成されます。たとえば、2MPIプロセス実行で環境変数**OMP_NUM_THREADS**に4が指定されたとすると、2MPI×4スレッドでの実行となるので、プログラムの実行には8コアを使用します。VEにMPIプロセスを配置するとき、実行に必要なコア数が足りなくならないよう注意してください。

第8章　　コンパイルリスト

本章ではC/C++コンパイラの出力するコンパイルリストについて説明します。
コンパイルリストは、「*ソースファイル名*.L」という名前でカレントディレクトリに出力されます。

8.1　オプションリスト

オプションリストは、**-report-option**、または、**-report-all**が指定されたとき出力されます。
形式:

```
NEC C/C++ Compiler (3.0.7) for Vector Engine      Thu Jun 18 10:18:05 2020    (a)

FILE NAME: fft.c          (b)

  COMPILER OPTIONS : -report-option     (c)

OPTIONS DIRECTIVE: -O4    (d)

PARAMETER :

Optimization Options :
   (e)                                                (f)
   -On                                               : 4
-fargument-alias                                   : disable
-fargument-noalias                                    : enable
-fassociative-math                                    : enable
```

(a)　オプションリストを作成したコンパイラ、および、リストの作成時刻
(b)　対応するソースファイルの名前
(c)　コマンドラインで指定されたコンパイラオプション
(d)　オプション指示行で指定されたコンパイラオプション
(e)　コンパイラオプション
(f)　コンパイラオプションの値

8.2　診断メッセージリスト

診断メッセージリストは、**-report-diagnostics**、または、**-report-all**が指定されたとき出力されます。

8.2.1　形式

診断メッセージリストの形式は以下のとおりです。

形式:

```
NEC C/C++ Compiler (1.0.0) for Vector Engine     Wed Jan 17 14:55:20 2018   (a)

FILE NAME: fft.c        (b)

FUNCTION NAME: fft3d    (c)
DIAGNOSTIC LIST

 LINE              DIAGNOSTIC MESSAGE
   (d)       (e)        (f)
    13: vec( 101): Vectorized loop.
    15: inl(1222): Inlined: fft3d
    18: vec( 101): Vectorized loop.
```

(a)　診断メッセージを作成したコンパイラ、および、リストの作成時刻

(b)　対応するソースファイルの名前

(c)　診断メッセージに対応するループ、文が含まれる手続の名前

(d)　行番号

(e)　診断メッセージの種別とメッセージ番号

メッセージの種別は以下のとおり。

vec : ベクトル化診断メッセージ

opt : 最適化診断メッセージ

inl　: インライン展開診断メッセージ

par : 並列化診断メッセージ

(f)　診断メッセージ

8.2.2　注意事項

- ある関数にインライン展開された関数の文、ループ対する診断メッセージは、それを呼び出した関数の診断メッセージリストには出力されません。インライン展開された関数の診断メッセージについては、その関数自身の診断メッセージリストを参照してください。

8.3　編集リスト

編集リストは、**-report-format**、または、**-report-all**が指定されたとき出力されます。
リストにはソース行とともに、次の情報が関数ごとに出力されます。

- ループのベクトル化情報
- ループの並列化情報
- 関数呼出しのインライン展開情報

8.3.1　形式

編集リストの形式は以下のとおりです。

```
NEC C/C++ Compiler (1.0.0) for Vector Engine     Wed Jan 17 14:55:16 2018    (a)

FILE NAME: a.c       (b)

FUNCTION NAME: func  (c)
FORMAT LIST

 LINE   LOOP      STATEMENT
  (d)    (e)       (f)
    1:           int func(int m, int n)
    2:           {
    3:               int i,j, a[m][n], b[m][n];
    4: +------>      for (i = 0; i < m; i++) {
    5: |V----->          for (j = 0; j < n; j++) {
    6: ||                    a[i][j] = a[i][j] + b[i][j];
    7: |V-----           }
    8: +------       }
    9:               return a[0][0];
   10:           }
```

(a)　編集リストを作成したコンパイラ、および、リストの作成時刻
(b)　対応するソースファイルの名前
(c)　以降のリスト中のソースコードの含まれる手続の名前
(d)　行番号
(e)　ループのベクトル化、並列化、インライン展開に関する情報
(f)　対応するコード

8.3.2　ループのベクトル化、並列化、インライン展開に関する情報

ループの構造、および、ベクトル化、並列化、インライン展開に関する情報を記号で出力します。出力例を以下に示します。

- ループ全体がベクトル化されたとき

```
V------> for (i = 0; i < n; i++) {
|
V------  }
```

- ループが部分ベクトル化されたとき

```
S------> for (i = 0; i < n; i++) {
|
S------  }
```

- ループが条件ベクトル化されたとき

```
C------> for (i = 0; i < n; i++) {
|
C------  }
```

- ループが並列化されたとき

```
P------> for (i = 0; i < n; i++) {
|
P------  }
```

- ループが並列化、かつ、ベクトル化されたとき

```
Y------> for (i = 0; i < n; i++) {
|
Y------  }
```

- ループがベクトル化されなかったとき

```
+------> for (i = 0; i < n; i++) {
|
+------  }
```

- ループが一重化され、ベクトル化されたとき

```
W------> for (i = 0; i < n; i++) {
|*----->     for (j = 0; j < m; j++)
||
|*-----      }
W------  }
```

- ループが入れ換えられ、ベクトル化されたとき

```
X------> for (i = 0; i < n; i++) {
|*----->     for (j = 0; j < m; j++) {
||
|*-----      }
X------  }
```

- 外側ループがアンロールされ、内側ループがベクトル化されたとき

```
U------> for (i = 0; i < n; i++) {
|V----->     for (j = 0; j < m; j++) {
||
|V-----      }
U------  }
```

- ループが融合されたとき

```
V------> for (i = 0; i < n; i++) {
|
|        }
|        for (i = 0; i < n; i++) {
|
V------  }
```

- ループが展開されたとき

```
*------> for (i = 0; i < 4; i++) {
|
*------  }
```

- 17カラム目に表示される文字は行がどう最適化されたかを表す
  - “I”　関数呼出しがインライン展開された
  - “M”この行を含む多重ループがベクトル行列積ライブラリ呼び出しに置き換えられた

- “F”　式に対してベクトル積和命令が生成された
- “R”　配列に**retain**指示行が適用された
- “G”　ベクトル収集命令が生成された
- “C”　ベクトル拡散命令が生成された
- “V”　配列に**vreg**指示行、または、**pvreg**指示行が適用された

8.3.3　注意事項

- ループの一部がヘッダファイルに含まれているとき、ループの構造、情報が正しく出力されないことがあります。
- 一つの行にループが複数存在するとき、ループの構造、情報が正しく出力されないことがあります。
- 関数の入口がヘッダファイルに含まれているとき、その関数の編集リストは出力されません。
- ループの直後の行が、プリプロセッサディレクティブのとき、それらの行がループの終端であるかのように出力されます。

```
V------> for (i = 0; i < n; i++) {
|
|        }
V------  #if 0
```

8.4　モジュール別最適化情報リスト

インライン展開モジュール、ベクトル化モジュール、コード生成モジュールの最適化情報を含んだリストを出力できます。

8.4.1　インライン展開モジュール

インライン展開モジュールの最適化情報リストは、**-report-inline**、または、**-report-all**が指定されたとき出力されます。

形式:

```
NEC C/C++ Compiler (3.1.0) for Vector Engine     Thu Sep 17 07:33:16 2020    (a)

FILE NAME: fft.c         (b)

FUNCTION NAME: func3     (c)
```

```
INLINE LIST

  INLINE REPORT: func3 (fft.c:17)
  (d)
  -> INLINE: func2 (fft.c:19)                (e)
     -> NOINLINE: func0 (fft.c:12)           (e)
        *** Source for routine not found.    (f)
     -> INLINE: func1 (fft.c:13)             (e)
```

(a)　インライン展開リストを作成したコンパイラ、および、リストの作成時刻

(b)　対応するソースファイルの名前

(c)　インライン展開状況を表示する手続の名前

(d)　インライン展開する手続の深さ

(e)　インライン展開の結果

(f)　診断メッセージ

8.4.2　ベクトル化モジュール

ベクトル化モジュールの最適化情報リストは、**-report-vector**、または、**-report-all**が指定されたとき出力されます。

形式:

```
NEC C/C++ Compiler (3.1.0) for Vector Engine     Thu Sep 17 08:10:39 2020    (a)

FILE NAME: vec.c         (b)

FUNCTION NAME: func      (c)

VECTORIZATION LIST

  LOOP BEGIN: (vec.c:3)
    <Unvectorized loop.>                 (d)

    LOOP BEGIN: (vec.c:4)
      <Vectorized loop.>                 (d)
      *** The number of VGT,    VSC.     :  0,  0. (vec.c:4)          (e)
      *** The number of VLOAD, VSTORE. :  1,  1. (vec.c:4)          (e)
    LOOP END
  LOOP END
```

(a)　ベクトルリストを作成したコンパイラ、および、リストの作成時刻
(b)　対応するソースファイルの名前
(c)　ベクトル化状況を表示する手続の名前
(d)　ベクトル化の結果
(e)　診断メッセージ

8.4.3　コード生成モジュール

コード生成モジュールの最適化情報リストは、**-report-cg**、または、**-report-all**が指定されたとき出力されます。

形式:

```
NEC C/C++ Compiler (3.1.0) for Vector Engine     Thu Sep 17 08:10:39 2020  (a)

FILE NAME: vec.c        (b)

FUNCTION NAME: func     (c)

CODE GENERATION LIST

  Hardware registers    (d)
    Reserved                  : 10 [sl fp lr sp s12 s13 tp got plt s17]
    Callee-saved              : 16 [s18-s33]
    Assigned
      Scalar registers        : 32 [s0-s12 s15-s16 s18-s21 s23-s32 s61-s63]
      Vector registers        : 35 [v0 v30-v63]
      Vector mask registers   :  0
      VREG directive          :  2 [v18-v19]

  Routine stack         (e)
    Total size                : 256 bytes
      Register spill area     :  16 bytes
      Parameter area          :  40 bytes
      Register save area      : 176 bytes
      User data area          :  16 bytes
      Others                  :   8 bytes

    Note: Total size of Routine stack does not include
          the size extended by alloca() and so on.
```

```
LOOP BEGIN: (vec.c:3)
  LOOP BEGIN: (vec.c:4)
    *** The number of VECTOR REGISTER SPILL  (f)
          Total                                    : 14
            Across calls                           : 11
            Not enough registers                   : 1
            Over basic blocks                      : 1
            Others                                 : 1
    *** The number of VECTOR REGISTER RESTORE
          Total                                    : 14
            Across calls                           : 11
            Not enough registers                   : 1
            Over basic blocks                      : 1
            Others                                 : 1
    *** The number of VECTOR REGISTER TRANSFER     : 12

    *** The number of SCALAR REGISTER RESTORE
          Total                                    : 14
            Across calls                           : 11
            Not enough registers                   : 1
            Over basic blocks                      : 1
            Others                                 : 1
    *** The number of SCALAR REGISTER RESTORE
          Total                                    : 14
            Across calls                           : 11
            Not enough registers                   : 1
            Over basic blocks                      : 1
            Others                                 : 1
    *** The number of SCALAR REGISTER TRANSFER     : 21
  LOOP END
LOOP END
```

(a)　コード生成リストを作成したコンパイラ、および、リストの作成時刻

(b)　対応するソースファイルの名前

(c)　コード生成状況を表示する手続の名前

(d)　レジスタの種類ごとのレジスタ使用数

Reserved　　：システムで予約済のレジスタ

Callee-saved　　：手続呼出しを跨いでセーブするレジスタ

Assigned　　：計算、ユーザデータに割り当てられたレジスタ

(e)　スタック情報

Register spill area　：レジスタ退避領域

Parameter area　：引数領域

Register save area　：レジスタセーブ領域

User data area　：ユーザデータ

Others　　：その他

(f)　ループ毎のレジスタスピル・リストア・転送の原因

Across calls　　：手続呼出しを跨いでいるため

Not enough registers　　：レジスタが枯渇しているため

Over basic blocks　：ベーシックブロックを跨いで使用されているため

Others　　：その他

第9章　　言語仕様に関する補足

9.1　組込み関数

9.1.1　性能チューニング

void __builtin_vprefetch(const void ***target*****, size_t** ***size*****)**

*target*で指定されたアドレスからデータをプリフェッチする。プリフェッチするサイズは*size*バイトで指定する。

9.1.2　デバッグ支援

void __builtin_traceback(unsignd long ***framepointer*****)**

実行時に環境変数**VE_TRACEBACK**に値が設定されているとき、*framepointer*で指定されたフレームポインタからのトレースバック情報を出力する。

例

```
__builtin_traceback((unsigned long *)__builtin_frame_address(0));
abort();
```

9.2　処理系定義

9.2.1　型

9.2.1.1　サイズとアラインメント

C/C++コンパイラで利用できる型のサイズ、アラインメントは以下のとおりです。単位はバイトです。1バイトは8ビットで構成されます。

型名	サイズ	アラインメント	備考
_Bool	1	1	Cでのみ利用可能。
bool	1	1	C++でのみ利用可能。
char **signed char** **unsigned char**	1	1	単に**char**と宣言したときは、**signed char**型。-**funsigned-char**で**unsigned char**型に変更可。
short **short int** **unsigned short** **unsigned short int**	2	2	

型名	サイズ	アラインメント	備考
int **unsigned int**	4	4	
long **long int** **unsigned long** **unsigned long int**	8	8	
long long **long long int** **unsigned long long** **unsigned long long int**	8	8	
float	4	4	4バイトの浮動小数点型。単精度実数型とも呼ぶ。
double	8	8	8バイトの浮動小数点型。倍精度実数型とも呼ぶ。
long double	16	16	16バイトの浮動小数点型。4倍精度実数型とも呼ぶ。
float _Complex	8	4	8バイトの複素数型。単精度複素数型。
double _Complex	16	8	16バイトの複素数型。倍精度複素数型。
long double _Complex	32	16	32バイトの複素数型。4倍精度複素数型。
ポインタ	8	8	
enum	4	4	**int**型に相当。
	8	8	**long**型に相当。

9.2.1.2 派生型のサイズとアラインメント

派生型は、基本型から派生して作られた型で、C言語では、配列型、構造体型、共用体型、ポインタ型、関数型に分類されます。

(1) 配列型

- **サイズ**
  配列要素のサイズ×配列の要素数
- **アラインメント**
  配列要素のアラインメントと同じ

(2)　構造体型、共用体型(含むクラス型)

- **サイズ**
  データメンバのサイズの合計値にデータメンバのアラインメントのための領域のサイズを加算した値。ただし、共用体のとき、領域の重複部分を除きます。
- **アラインメント**
  データメンバのアラインメントサイズの最大値か、4バイトの大きい方の値
- **注意事項**
  C++言語でのクラス、構造体、共用体のとき、コンパイラが言語仕様を実現するため、内部データをデータメンバに追加することがあります。そのとき、これらのサイズ、アラインメントサイズは、上記のとおりにはならないことがあります。

9.2.1.3 その他の型

上記の型のほかに、さらに以下の型が定義されます。

(1)　**size_t** 型

unsigned long型に相当します。

(2)　**ptrdiff_t** 型

long型に相当します。**ptrdiff_t**型は、ヘッダ<stddef.h>、および、<cstddef>で定義されます。

(3)　**wchar_t** 型

int型に相当します。

(4)　ビットフィールド

構造体、または、共用体のメンバのビット数（符号ビットがあれば符号ビットを含む）を指定します。

9.2.2　型変換

この節では、C/C++コンパイラでの次の型変換について説明します。

- 整数の上位変換
- 整数の変換
- 浮動小数点数の変換
- 複素数の変換
- 浮動小数点数と整数の変換
- 複素数と整数の変換

- 複素数と浮動小数点数の変換
- 算術変換

9.2.2.1 整数の上位変換

整数は、より広い値を表現できる整数型に変換できます。このようなサイズを大きくする変換を「整数の上位変換」といいます。整数の上位変換は、次のように行われます。

- **値の範囲が**int**型で表現できるとき、**int**型に変換**
- **int**型で表現できず**unsigned int**型で表現できるとき、**unsigned int**型に変換**
- **unsigned int**型で表現できず**long**型で表現できるとき、**long**型に変換**
- **long**型で表現できず**unsigned long**型で表現できるとき、**unsigned long**型に変換**

9.2.2.2 整数の変換

ここでは、次の整数の変換について説明します。

- 符号付き整数から符号なし整数への変換
- 符号なし整数から符号付き整数への変換

整数の変換は、整数型の間で行います。整数型とは、**char**、**short**、**int**、**long**、**long long**型と各型の**unsigned**のものです。

(1)　符号付き整数から符号なし整数への変換

符号付き整数は、対応する符号なし整数型に変換できます。この変換では、値のビットパターンは変わりませんが、値の解釈方法が変わります。

例

```
#include <iostream>
main()
{
      short                s = -1;
      unsigned short    u;
      std::cout << s << std::endl;
      u = s;
      std::cout << u << std::endl;
}
```

このプログラムを実行すると-1と65535が表示されます。

この例では、**signed short**型の変数sを定義して負の数に初期化しています。u = sという式によって、sは**unsigned short**に変換されてから、uに代入されます。

(2)　符号なし整数から符号付き整数への変換

符号なし整数は、対応する符号付き整数型に変換できます。ただし、この変換では、符号なし整数型の値が符号付きの型で表現できる範囲を超えると、値の解釈を誤ることもあります。

例

```
#include <iostream>
main()
{
	short			s;
	unsigned short	u = 65535;

	std::cout << u << std::endl;
	s = u;
	std::cout << s << std::endl;
}
```

このプログラムを実行すると65535と-1が表示されます。

この例では、u は**unsigned short**型の整数ですが、式s = uを評価するには符号付きの値に変換しなければなりません。しかし、この値は**signed short**型で正しく表現できないので、データは誤って解釈されます。

9.2.2.3 浮動小数点数の変換

浮動小数点数は、より高い精度の浮動小数点型に変換できます。より高い精度への変換では、その変換により有効桁数が失われません。たとえば、**float**型から**double**型への変換や、**double**型から**long double**型への変換では、値は変化しません。

浮動小数点数は、より低い精度の型にも変換できます。ただし、元の値が変換先の型で表現できる範囲にある場合に限ります。元の値を正確に表現できないとき、表現できる範囲の最も近い値に変換されます。最も近い値が存在しないときの結果は未定義です。

例

```
#include <iostream>
main()
{
    double           d = 1e+100;
    long double  ld;
    float            f;
    ld = d;
    std::cout << ld << std::endl;
    f = d;
    std::cout << f  << std::endl;
}
```

このプログラムを実行すると1e+100とinfが表示されます。

最初の式ld = dでは、**double**型から**long double**型に安全に変換されます。しかし、次の式f = dでは**double**型から**float**型へ正しく変換されません。**float**型で表現できる最大値が3.40282347e+38であるためです。したがって、1e+100は無限大に変換され、**inf**と表示されます。

9.2.2.4 複素数の変換

複素数は、より高い精度の複素数型、および、より低い精度の複素数型に変換できます。複素数を変換するとき、複素数の実部と虚部の両方に浮動小数点数の変換と同様の規則が適用されます。浮動小数点数の変換の規則については、「浮動小数点数の変換」を参照してください。

9.2.2.5 浮動小数点数と整数の変換

浮動小数点数を整数に変換できます。逆に、整数を浮動小数点数に変換することもできます。

(1)　浮動小数点数から整数への変換

浮動小数点数を整数型に変換すると、小数部は切り捨てられます。変換の過程で値は丸められません。切り捨てとは、たとえば1.3を1に、-1.2を-1に変換することを意味します。

(2)　整数から浮動小数点数への変換

整数を浮動小数点型に変換した場合に元の値を厳密に表現できないとき、表現できる範囲の最も近い値に変換されます。

9.2.2.6 複素数と整数の変換

整数を複素数型に変換できます。逆に、複素数を整数型に変換することもできます。

(1)　複素数から整数への変換

複素数を整数型に変換すると、複素数の虚部を捨て、実部から小数部が切り捨てられた値になります。変換の過程で値は丸められません。たとえば、複素数の実部が1.3で虚部が2.0であるとき、1に変換されます。

(2)　整数から複素数への変換

整数を複素数型に変換すると、複素数の実部は元の値を厳密に表現できないとき、表現できる範囲の最も近い値になり、虚部は符号付きの0になります。

9.2.2.7　複素数と浮動小数点数の変換

浮動小数点数を複素数型に変換できます。逆に、複素数を浮動小数点型に変換することもできます。

(1)　複素数から浮動小数点数への変換

複素数を浮動小数点型に変換すると、複素数の虚部を捨て、実部は浮動小数点数の変換と同様の規則に基づいて変換されます。浮動小数点数の変換の規則については、「浮動小数点数の変換」を参照してください。

(2)　浮動小数点数から複素数への変換

浮動小数点数を複素数型に変換すると、複素数の実部は浮動小数点数の変換と同様の規則に基づいて変換され、虚部は符号付きの0になります。浮動小数点数の変換の規則については、「浮動小数点数の変換」を参照してください。

9.2.2.8　算術変換

2項演算子の多くは、オペランドの型を変換し、その演算結果もそれに従った型になります。このような変換を「算術変換」といいます。この変換により、各オペランドは共通の型となり、演算結果もまた同じ型になります。

この変換は、以下の規則に従います。

- 一方のオペランドが**long double _Complex**型のとき、他方のオペランドは**long double _Complex**型に変換されます。
  - 上の条件に該当せず、一方のオペランドが**long double**型のとき、他方のオペランドは**long double**型に変換されます。
- 上の条件に該当せず、一方のオペランドが**double _Complex**型のとき、他方のオペランドは**double _Complex**型に変換されます。
  - 上の条件に該当せず、一方のオペランドが**double**型のとき、他方のオペランドは**double**型に変換されます。

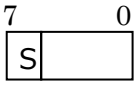
- 上の条件に該当せず、一方のオペランドが**float _Complex**型のとき、他方のオペランドは**float _Complex**型に変換されます。
  - 上の条件のどちらにも該当せず、一方のオペランドが**float**型のとき、他方のオペランドは**float**型に変換されます。
- 上の条件のどれにも該当しない(どのオペランドも浮動小数点型、複素数型でない)とき、オペランドに対して以下の上位変換が行われます。
  - 一方のオペランドが**unsigned long**型のとき、他方のオペランドは**unsigned long**型に変換されます。
  - 上の条件に該当せず、一方のオペランドが**long**型で他方が**unsigned int**型のとき、両方のオペランドは**unsigned long**型に変換されます。
  - 上の二つの条件に該当せず、一方のオペランドが**long**型のとき、他方のオペランドは**long**型に変換されます。
  - 上の三つの条件に該当せず、一方のオペランドが**unsigned int**型のとき、他方のオペランドは**unsigned int**型に変換されます。
  - 上の条件のどれにも該当しないとき、両方のオペランドは**int**型に変換されます。

9.2.3　データの内部表現

9.2.3.1 整数型

(1) **signed char** (1 バイト符号付き整数型)

形式



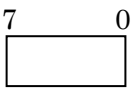
S:符号ビット

表現可能な値

-128 ～ 127

(2) **unsigned char** (1 バイト符号なし整数型)

形式

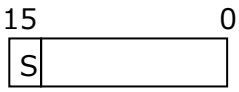


表現可能な値

0 ～ 255

(3) **short** / **short int** (2バイト符号付き整数型)

形式



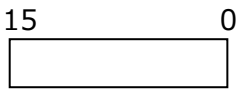
S:符号ビット(0:正 1:負)

表現可能な値

-32768 ～ 32767 (-2^{15} ～ 2^{15-1})

(4) **unsigned short** / **unsigned short int** (2バイト符号なし整数型)

形式

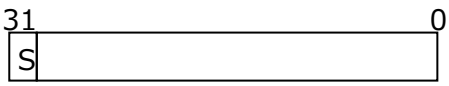


表現可能な値

0 ～ 65535 (0 ～ 2^{16-1})

(5) **int** (4バイト符号付き整数型)

形式



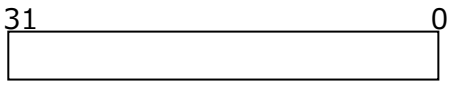
S:符号ビット(0:正 1:負)

表現可能な値

-2147483648 ～ 2147483647 (-2^{31} ～ 2^{31-1})

(6) **unsigned int** (4バイト符号なし整数型)

形式

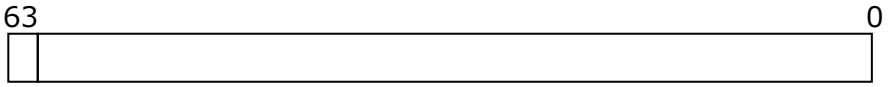


表現可能な値

0 ～ 4294967295 (0 ～ 2^{32-1})

(7) **long** / **long int** / **long long** / **long long int** (8バイト符号付き整数型)

形式



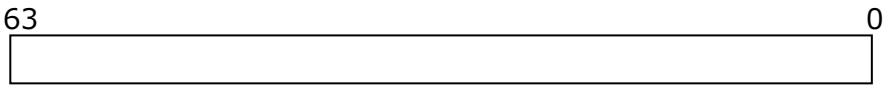
S:符号ビット(0:正 1:負)

表現可能な値

-9223372036854775808 ～ 9223372036854775807 (-2^{63} ～ 2^{63-1})

(8) **unsigned long** / **unsigned long int** / **unsigned long long** / **unsigned long long int** (8バイト符号なし整数型)

形式



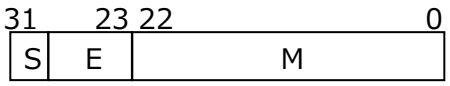
表現可能な値

0 ～ 18446744073709551615 (0 ～ 2^{64-1})

9.2.3.2 浮動小数点型

(1) **float** (単精度浮動小数点型)

形式



S:仮数部の符号ビット(0:正 1:負)

E:指数部 (0≦E≦255)

M:仮数部 (0≦M＜1)

表現可能な値

$(-1)^{S}*2^{E-127}*1.M$。有効桁数は10進約7桁で、絶対値が0、または、約10^{-38}～10^{37}の範囲の値。

特別な値

NaN　　E==255、かつ、M!=0

無限大　　E==255、かつ、M==0

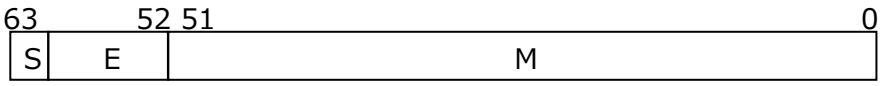
符号付き0　E==0

注意事項

非正規化数(E=0、かつ、M!=0)をサポートしていません。このような値はすべて符号付き0として扱われます。

(2) **double** (倍精度浮動小数点型)

形式



S:仮数部の符号ビット(0:正 1:負)

E:指数部 (0≦E≦2047)

M:仮数部 (0≦M＜1)

表現可能な値

$(-1)^{S}*2^{E-1023}*1.M$。有効桁数は10進約16桁で、絶対値が0、または、約10^{-308}～10^{308}の範囲の値。

特別な値

NaN　　E==2047、かつ、M!=0

無限大　　E==2047、かつ、M==0

符号付き0　E==0

注意事項

非正規化数(E=0、かつ、M!=0)をサポートしていません。このような値はすべて符号付き0として扱われます。

(3) **long double** (4 倍精度浮動小数点型)

形式

S:仮数部の符号ビット(0:正 1:負)

E:指数部 (0≦E≦32767)

M:仮数部 (0≦M＜1)

表現可能な値

$(-1)^{S}*2^{E-16383}*1.M$。有効桁数は10進約34桁で、絶対値が0、または、約10^{-4932}～10^{4932}の範囲の値。

特別な値

NaN　　E==32767、かつ、M!=0

無限大　　E==32767、かつ、M==0

符号付き0　E==0

注意事項

非正規化数(E=0、かつ、M!=0)をサポートしていません。このような値はすべて符号付き0として扱われます。

9.2.3.3 複素数型

(1) **float _Complex**（単精度複素数型）

形式

63　　55 54　　32

RS	RE	RM
IS	IE	IM

31　　23 22　　0

実部 RS:仮数部の符号ビット(0:正 1:負)、RE:指数部(0≦RE≦255)、RM:仮数部

虚部 IS:指数部の符号ビット(0:正 1:負)、IE:指数部(0≦IE≦255)、IM:仮数部

表現可能な値

$(-1)^{RS}*2^{RE-127}*1.RM$および$(-1)^{IS}*2^{IE-127}*1.IM$。有効桁数は10進約7桁で、絶対値が0、または、約10^{-38}～10^{37}の範囲の値。

特別な値

NaN	RE==255、かつ、RM!=0、または、IE==255、かつ、IM!=0
無限大	RE==255、かつ、RM==0、または、IE==255、かつ、IM==0
符号付き0	RE==0、かつ、RI==0

注意事項

複素数型の実数部または虚数部における非正規化数(RE=0、かつ、RM!=0、および、IE=0、かつ、IM!=0)をサポートしていません。このような値はすべて符号付き0として扱われます。

(2) **double _Complex**（倍精度複素数型）

形式

127　　116 115　　64

RS	RE	RM
IS	IE	IM

63　　52 51　　0

実部 RS:仮数部の符号ビット(0:正 1:負)、RE:指数部(0≦RE≦2047)、RM:仮数部

虚部 IS:仮数部の符号ビット(0:正 1:負)、IE:指数部(0≦IE≦2047)、IM:仮数部

表現可能な値

$(-1)^{RS}*2^{RE-1023}*1.RM$、および、$(-1)^{IS}*2^{IE-1023}*1.IM$。有効桁数は10進約16桁で、絶対値が0、または、約10-308～10308の範囲の値。

特別な値

NaN　　RE==2047、かつ、RM!=0、または、IE==2047、かつ、IM!=0

無限大　　RE==2047、かつ、RM==0、または、IE==2047、かつ、IM==0

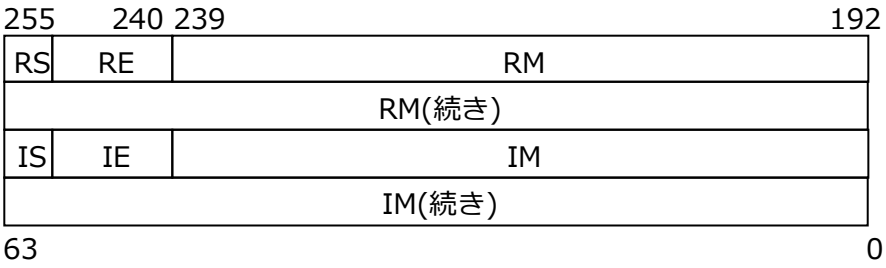
符号付き0　RE==0、かつ、IE==0

注意事項

複素数型の実数部または虚数部における非正規化数(RE==0、かつ、RM!=0、および、IE==0、かつ、IM!=0)をサポートしていません。このような値はすべて符号付き0として扱われます。

(3) **long double _Complex** (4 倍精度複素数型)

形式



実部RS:仮数部の符号ビット(0:正　1:負)、RE:指数部(0≦RE≦32767)、RM:仮数部

虚部IS:仮数部の符号ビット(0:正　1:負)、IE:指数部(0≦IE≦32767)、IM:仮数部

表現可能な値

$(-1)^{RS}*2^{RE-16383}*1.RM$、および、$(-1)^{IS}*2^{IE-16383}*1.IM$。有効桁数は10進約34桁で、絶対値が0、または、約10^{-4932}～10^{4932}の範囲の値。

特別な値

NaN　　RE==32767、かつ、RM!=0、または、IE==32767、かつ、IM!=0

無限大　　RE==32767、かつ、RM==0、または、IE==32767、かつ、IM==0

符号付き0　RE==0、かつ、IE==0

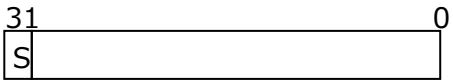
注意事項

複素数型の実数部または虚数部における非正規化数(RE=0、かつ、RM!=0、および、IE=0、かつ、IM!=0)をサポートしていません。このような値はすべて符号付き0として扱われます。

9.2.3.4 enum 型

(1) ４バイト型

形式



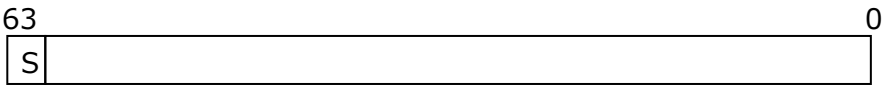
S:符号ビット(0:正 1:負)

表現可能な値

-2147483648 ～ 2147483647 (-2^{31} ～ 2^{31-1})

(2) ８バイト型

形式



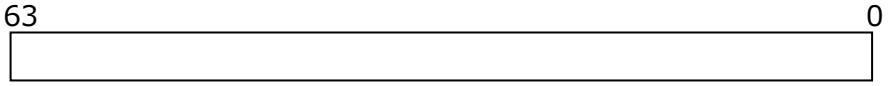
S:符号ビット(0:正 1:負)

表現可能な値

-9223372036854775808 ～ 9223372036854775807 (-2^{63} ～ 2^{63-1})

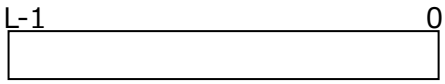
9.2.3.5 ポインタ型

形式



9.2.3.6 ビットフィールド

形式



L:ビットフィールドの長さ

説明

- 符号付きのとき、最左端ビットが符号ビットとなる。
- ビットフィールドは、そのビットフィールドの型の境界をまたがないように左詰めで割り付けられる。隣接している複数のビットフィールドの割り付けにおいて、あるビットフィールドがそのビットフィールドの型の境界をまたぐとき、そのビットフィールドはその型の境界をもつ次のアドレスの先頭から割り付けられる。

9.2.4　定義済みマクロ

定義済みマクロは**-dM -E**を指定することによって出力できます。

例

```
$ ncc -dM -E a.c | sort
```

主な定義済みマクロは以下のとおりです。

_LP64、__LP64__

常に1に定義される。

unix、__unix、__unix__

常に1に定義される。

linux、__linux、__linux__

常に1に定義される。

__ve、__ve__

常に1に定義される。

__ELF__

常に1に定義される。

__STDC__

常に1に定義される。

__STDC_HOSTED__

常に1に定義される。

__STDC_NO_ATOMICS__

常に定義されない。

__NEC__

常に1に定義される。

__FAST_MATH__

-**ffast-math**が有効であるとき1に定義され、そうでないとき定義されない。

_FTRACE

-ftraceが有効であるとき1に定義され、そうでないとき定義されない。

__NEC_VERSION__

コンパイラのバージョンが*X.Y.Z*であるとき、以下の計算式で得られる値が定義される。

*X**10000 + *Y**100 + *Z*

__OPTIMIZE__

コンパイル時に有効となった**-O***n*の最適化レベル(*n*)の値に定義される。

__VECTOR__

自動ベクトル化が有効であるとき1に定義され、そうでないとき定義されない。

__VERSION__

常に使用しているコンパイラのバージョンを示す文字列定数に定義される。

9.3　インラインアセンブラ

9.3.1　基本 asm 文

形式

asm [**volatile**] (Assembler-instructions)

説明

基本**asm**文はオペランドを持ちません。基本**asm**文には暗黙的に**volatile**が指定されるため、**volatile**指定はプログラムの動作に影響を与えません。

9.3.2　拡張 asm 文

形式

```
asm [ volatile ]   ( アセンブラテンプレート
                        : 出力オペランド
                        [ : 入力オペランド
                        [ : 破壊されるレジスタ] ])

アセンブラテンプレート:
        アセンブラ命令のテキスト文字列
出力オペランド:
        [ [シンボル名] ] 制約（変数名）, [ 出力オペランド ] ...
入力オペランド:
        [ [シンボル名] ] 制約（式）, [ 入力オペランド ] ...
破壊レジスタ:
        レジスタ名、または、”memory” , [ 破壊レジスタ ] ...
```

説明

拡張**asm**文はオペランドを持つことができます。C言語の変数は、拡張**asm**文のオペランドとして読み込み、書き込みできます。

volatile指定は、**asm**文実行中に副作用が発生する可能性のある最適化を無効にできます。

*アセンブラテンプレート*はアセンブラ命令のテキスト文字列です。コンパイラは入力/出力オペランドを参照して、アセンブラテンプレート中のトークンを置換し、テキスト文字列をアセンブラコードとして出力します。出力オペランドと入力オペランドN番目のオペランドは%Nのように記述できます。アセンブラテンプレート中の%%は%に置き換えられます。

*出力オペランド*はコンマ区切りのリストです。*出力オペランド*は制約（詳細は後述）を持ち、アセンブラテンプレート中のアセンブラ命令によって値が更新される変数名を持ちます。リストは空でも構いません。

*入力オペランド*はコンマ区切りのリストです。*入力オペランド*は制約(詳細は後述)を持ち、アセンブラテンプレート中のアセンブラ命令によって値が参照される変数名、または、式を持ちます。リストは空でも構いません。

*破壊レジスタ*はアセンブラテンプレート中のアセンブラ命令によって破壊されるレジスタのコンマ区切りのリストです。”memory”を指定したとき、出力オペランドと入力オペランドに指定されたオペランド以外のメモリ上の領域が暗黙的に読み書きされる可能性があることをコンパイラに知らせます。

利用可能な制約

制約	機能
m	AS、ASX形式のメモリオペランドを許可する。
r	スカラレジスタオペランドを許可する。
i	即値を許可する。値がアセンブルされない限り内容がわからない記号定数を含む。
0, 1, 2, …9	入力オペランドと出力オペランドに同じレジスタを割り当てることを指定する。(この指定がなければ、Cソース上で同じ式でもコンパイラは同じレジスタを割り当てることを保証しない。)この制約はマッチング制約と呼ばれる。
[シンボル名]	アセンブラテンプレート中でこの制約が指定されたオペランドを%[シンボル名]として参照することができる。
f	スカラレジスタオペランドを許可する。オペランドに割り当てられたレジスタに浮動小数点形式のデータを格納する。

2つ以上の制約を指定するとき(例えば”=rm0”)、コンパイラは優先順位に沿って制約を選択します。優先順位は以下のとおりです。

ｉ＞ｒ＞ｍ＞ **マッチング制約**

“#”か”*”の制約修飾文字によってすべての制約が無視されるとき、コンパイラはrが指定されたとみなします。

制約修飾文字

修飾文字	機能
=	オペランドに書き込みのみを行う。
+	オペランドに読み込み、書き込み両方を行う。
&	入力オペランドと出力オペランドに異なるレジスタを割り当てる。
%	このオペランドと次のオペランドが入れ替え可能であることを指定する。これによって、全ての制約を満たすためのコストが最も少ない場合にコンパイラは2つのオペランドを入れ替えることができる。
#	次のコンマまでのすべての制約を無視する。
*	次の文字を無視する。

例

```
#define FIVE 5
int main(void)
{
    int in=10, out;
    asm ("ldl.sx %%s50, %1¥n¥t"
         "adds.w.zx %0, %2, %%s50¥n¥t"
         : "=r"(out)         /* Constraint "r" */
         : "m"(in), "i"(FIVE)    /* Constraint "m", "i" */
         : "%s50"
    );
    printf("out=%d¥n", out);
}
```

9.3.3　名前指定

形式

宣言子 **asm** (*Name*)

説明

アセンブラコード中で使用する関数名や変数名を指定できます。宣言子は、C言語の構文に準拠した宣言子です。*Name*は、アセンブラコード内の関数名または、変数名の文字列です。

9.3.4　注意事項

- 代替のキーワードとして、__asm、__asm__、__volatile、__volatile__が利用できます。

- 拡張**asm**文の*破壊レジスタ*に指定できるのはスカラレジスタ、ベクトルレジスタ、ベクトルマスクレジスタ、ベクトルインデックスレジスタのみです。
- M形式の即値はアセンブラテンプレート中に直接指定しなければなりません。拡張**asm**文のオペランドに指定することはできません。
- “br[cf].*.[bp]” (“*”は”l”、”w”、”d”、”s”のいずれか)命令のASオペランドとして指定されたdispには制約文字”i”を指定しなければなりません。
- lhm.*、shm.*(“*”は”b”、”h”、”w”、”l”のいずれか)命令のHMオペランドには制約文字”r”か”i”を指定しなければなりません。
- 入力/出力オペランドの式に指定される即値が-2147483648から2147483647の範囲に収まらないとき、そのオペランドには制約文字”r”、または、”m”が指定しなければなりません。
- **asm**文の外に飛び出す命令は利用できません。
- アセンブラテンプレート中の%指定の番号が、入力/出力オペランドの合計数以下であるとき、%指定はそのままアセンブラファイルに出力されます。
- FAQ、FSQ、FMQ、FCQのような4倍精度浮動小数点型命令がアセンブラテンプレート中で利用されるとき、拡張**asm**文は利用できません。
- 自動変数、レジスタ変数、即値は入力/出力オペランドの式にのみ指定できます。

9.4　注意事項

9.4.1　C 言語に関する注意事項

- **_Atomic**修飾子は利用できません。
- **CX_LIMITED_RANGE**マクロは無視されます。
- **FENV_ACCESS**、および、**FP_CONTRACT**マクロは無視されます。
- 次のCヘッダを使用するとき、**-pthread**を指定してください。
  <stdatomic.h>、<threads.h>
- 以下のGCC拡張機能は利用できません。
  - 関数パラメタの前方宣言 (可変長配列のパラメタとして指定する変数の前方宣言)
  - GCC形式の複素数型
  - ネストされた関数

- 可変長配列をフィールドとして持つローカルな構造体

※そのようなフィールドはGNU C モードでは長さ0として扱われますが、GCCとは異なります。

- 値としてのラベル

9.4.2　C++言語に関する注意事項

- 指定されたサイズの領域解放は利用できません。
- 次のC++ヘッダを使用するとき、**-pthread**を指定してください。
  <atomic>、<condition_variable>、<mutex>、<thread>
- C++17言語規格で追加、拡張された以下の機能には対応していません。
  - クラステンプレートscoped_lock
  - クラステンプレートowner_lessのvoid特殊化版
  - ヘッダ<map>、<set>、<unordered_map>、<unordered_set>の連想コンテナクラスのテンプレートメンバ関数merge()
  - クラステンプレートshared_ptrの配列型サポート

第10章　多言語プログラミング

異なるプログラミング言語のオブジェクトファイルをリンクし、一つの実行ファイルを作成することを多言語プログラミングと呼びます。この章ではCプログラム、C++プログラム、Fortranプログラムを使用した多言語プログラミングの手法について説明します。

10.1 多言語プログラミングのポイント

次の例は、Cプログラム、Fortranプログラムをリンクし、一つの実行ファイルを作成する多言語プログラミングの例です。

C プログラム（ファイル名: a.c）

```
#include <stdlib.h>
#define N 1024
#define SIZE sizeof(double)
main()
{
  double *x = (double *)malloc(SIZE*N);
  double *y = (double *)malloc(SIZE*N);
  double *z = (double *)malloc(SIZE*N);
  int n;
  n = read_data(x, y);
  compute_(x, y, z, &n);
  write_data(z, n);
}
```

C プログラム（ファイル名: b.c）

```
#include <stdio.h>
int read_data(double *x, double *y)
{ ... }
```

Fortran プログラム（ファイル名: c.f90）

```
SUBROUTINE COMPUTE (X, Y, Z, N)
REAL*8 X(N),Y(N),Z(N)
!!! 計算
I = CHECK_VALUE(Z(N))
IF (I.EQ.0) RETURN
END SUBROUTINE
```

C プログラム（ファイル名: d.c）

```
int check_value_(double *x)
{ ... }
```

この例では、CプログラムからFortranプログラム、FortranプログラムからCプログラムを呼び出しています。呼び出す際には関数名、手続名を実際にプログラミングしている名前から外部シンボル名に変更し、引数、返却値でデータをやりとりし、言語間でデータを共有しています。

多言語プログラミングのポイントは以下です。

- C関数、C++関数、Fortran手続の名前の対応
- C/C++のデータ、Fortranのデータの型の対応

- C/C++ - Fortran間の返却値の受け渡し
- C/C++ - Fortran間の引数による値の受け渡し
- プログラムのコンパイル、リンクによる実行ファイルの作成

以降で、これらのポイントについて説明します。

10.2 C/C++関数名・Fortran手続名の対応

ソースファイルに記述されたC++関数名、Fortran手続名は、コンパイラによって名前が変更され、オブジェクトファイルに外部シンボル名として登録されます。このため、それらの関数、手続を参照するとき、変更後の外部シンボル名で呼び出すようにします。

10.2.1 Fortran 手続の外部シンボル名

(1)　手続の結合ラベルを使用するとき

Fortran 2003言語仕様の「Cとの相互利用」で手続の結合ラベルを使用するとき、Fortranプログラムの手続名は、結合ラベルと同じ文字列の外部シンボル名になります。すなわち、**NAME**指定があるとき指定された名前、省略されたときすべて小文字に変更された外部シンボル名になります。

例

```
SUBROUTINE SUB1(X) BIND(C, NAME="Fortran_Sub1")
...
END SUBROUTINE

SUBROUTINE SUB2(Y) BIND(C)
...
END SUBROUTINE
```

このFortranプログラムでは手続名が以下の外部シンボル名に変更されます。

（手続名）		（外部シンボル名）
SUB1	→	Fortran_Sub1
SUB2	→	sub2

(2)　手続の結合ラベルを使用しないとき

Fortranプログラムの手続名は、次の規則で外部シンボル名に変更されます。

- 手続名はすべて小文字に変更
- 名前の末尾に下線(_)が付加

例

```
SUBROUTINE COMPUTE (X, Y, Z, N)
REAL*8 X(N),Y(N),Z(N)
! 計算
I = CHECK_VALUE(Z(N))
IF (I.EQ.0) RETURN
END SUBROUTINE
```

このFortranプログラムでは手続名が以下の外部シンボル名に変更されます。

（手続名）		（外部シンボル名）
COMPUTE	→	compute_
CHECK_VALUE	→	check_value_

10.2.2 C++関数名の外部シンボル名

C++コンパイラは、ソースファイルに記述された関数名に返却値、引数の型を表す文字列が付加します。この操作を関数名のマングル(mangle)と呼び、C++コンパイラは、この仕組みを利用して引数の型が違う同じ名前の関数の定義を可能としています。

例

（ソースファイルでの表記）		（マングル名）
void func(double *x)	→	_Z4funcPd
void func(float *x)	→	_Z4funcPf

備考　　マングル名をソースファイルでの表記に戻すことを「デマングル」(demangle)と呼びます。

C関数やFortran手続から呼び出すC++関数は、C結合で宣言し、マングルされないようにします。そして、ソースファイルに記述された関数名で呼び出すようにします。同じように、C++関数から呼び出すC関数やFortran手続のプロトタイプ宣言もC結合で宣言し、マングルされないようにします。

例

```
extern "C" {
    void func(double *x);
    void func1(float *x);
};
```

結合指定はC++言語でのみ利用できます。プロトタイプ宣言などをC言語でも利用するときは、結合指定を条件コーディングします。

例

```
#ifdef __cplusplus          // __cplusplusはC++コンパイラによって
                            // 自動的に定義される
extern "C" {
#endif
    void func(double *x);
    void func1(float *x);
#ifdef __cplusplus
};
#endif
```

10.2.3 C/C++関数名、Fortran 手続の対応ルール

- C関数からFortran手続を呼び出すとき、Fortran手続の外部シンボル名でC関数から呼び出します。
- Fortran手続から呼び出されるC関数は、Fortran手続の外部シンボル名で定義します。
- C関数やFortran手続から呼び出されるC++関数は、C結合で宣言します。
- C++関数から呼び出すC関数やFortran手続のプロトタイプ宣言は、C結合で宣言します。

10.2.4 呼出し例

例　　C 関数からの Fortran 手続の呼出し(**BIND** 属性の指定あり)

呼出し側 (C 関数)

```
void cfunc() {
    ...
    sub1();
    ...
}
```

呼び出される側 (Fortran 手続)

```
SUBROUTINE SUB1() BIND(C)
...
END SUBROUTINE SUB1
```

すべて小文字の名前でプロトタイプ宣言し、呼び出します。

例　　C 関数からの Fortran 手続の呼出し(**BIND** 属性の指定なし)

呼出し側 (C 関数)

```
extern int sub_();
void cfunc() {
    ...
    sub_();
    ...
}
```

呼び出される側 (Fortran 手続)

```
SUBROUTINE SUB
...
END SUBROUTINE SUB
```

Fortran手続を下線付きのすべて小文字の名前でプロトタイプ宣言し、呼び出します。

例　　Fortran 手続からの C 関数の呼出し(**BIND** 属性の指定あり)

呼出し側 (Fortran 手続)

```
SUBROUTINE SUB
    USE, INTRINSIC :: ISO_C_BINDING
    INTERFACE
        SUBROUTINE CFUNC() BIND(C)
        END SUBROUTINE CFUNC
    END INTERFACE
    ...
    CALL CFUNC
    ...
END SUBROUTINE SUB
```

呼び出される側 (C 関数)

```
void cfunc() {
...
}
```

Cプログラムにおいて、すべて小文字の名前でC関数を宣言、定義し、Fortranプログラムにおいてすべて大文字の名前で引用仕様の定義、参照します。

例　　Fortran 手続からの C 関数の呼出し(**BIND** 属性の指定なし)

呼出し側 (Fortran 手続)

```
SUBROUTINE SUB
...
CALL CFUNC
...
END SUBROUTINE SUB
```

呼び出される側 (C 関数)

```
int cfunc_() {
...
}
```

下線付きのすべて小文字の名前で、C関数を宣言、定義します。

例　　C++関数からの Fortran 手続の呼出し

呼出し側 (C++関数)

```
extern "C" {
    int sub_(void);
};
void cfunc() {
    ...
    sub_();
    ...
}
```

呼び出される側 (Fortran 手続)

```
SUBROUTINE SUB
...
END SUBROUTINE SUB
```

Fortran手続を下線付きのすべて小文字の名前で、C結合でプロトタイプ宣言し、呼び出します。

例　　Fortran 手続からの C++関数の呼出し

呼出し側 (Fortran 手続)

```
SUBROUTINE SUB
...
CALL CFUNC
...
END SUBROUTINE SUB
```

呼び出される側 (C++関数)

```
extern "C" {
    int cfunc_(void);
};
int cfunc_(void) {
...
}
```

下線付きのすべて小文字の名前、C結合で、C++関数を宣言、定義します。

10.3 データ型

FortranプログラムとC/C++プログラムのデータ型の対応は以下のとおりです。

10.3.1 Fortran の整数型/論理型

型	Fortranの型	C/C++の型
整数型	**INTEGER**	**int** (*1)
	INTEGER(KIND=1) **INTEGER*1**	**signed char**
	INTEGER(KIND=2) **INTEGER*2**	**short**
	INTEGER(KIND=4) **INTEGER*4**	**int**
	INTEGER(KIND=8) **INTEGER*8**	**long**、**long int**、**long long**、 または、**long long int**
論理型	**LOGICAL**	**int** (*1)
	LOGICAL(KIND=1)	**signed char**
	LOGICAL(KIND=2)	**short**
	LOGICAL(KIND=4)	**int**

LOGICAL(KIND=8)	**long**、**long int**、**long long**、 または、**long long int**

(*1) **-fdefault-integer**=8 指定時は、**long**、**long int**、**long long**、または、**long long int**

10.3.2 Fortran の実数型/複素数型

型	Fortranの型	C/C++の型
実数型	**REAL**	**float** (*1)
	REAL(KIND=4) **REAL*4**	**float**
倍精度実数型	**DOUBLE PRECISION**	**double** (*2)
	REAL(KIND=8) **REAL*8**	**double**
4倍精度実数型	**QUADRUPLE PRECISION** **REAL(KIND=16)** **REAL*16**	**long double**
複素数型	**COMPLEX**	**float __complex__** (*3)
	COMPLEX(KIND=4) **COMPLEX*8**	**float __complex__**
倍精度複素数型	**COMPLEX(KIND=8)** **COMPLEX*16**	**double __complex__**
4倍精度複素数型	**COMPLEX(KIND=16)** **COMPLEX*32**	**long double __complex__**

(*1)　**-fdefault-real**=8指定時は、**double**

(*2)　**-fdefault-double**=16指定時は、**long double**

(*3)　**-fdefault-real**=8指定時は、**double __complex__**

10.3.3 Fortran の文字型

文字列chを宣言する例で示します。

型	Fortranの型	C/C++の型
文字型	**CHARACTER(LEN=*n*) ch**	**char ch[*n*];**

10.3.4 Fortran の派生型

(1) 説明

Fortranプログラムの派生型をCプログラムの構造体と結合するとき、派生型定義と宣言部に**BIND**属性（**BIND**(C)）を指定します。派生型の成分と構造体のメンバの数は同じでなければならず、対応する成分・メンバの型が同じでなければなりません。

例

Fortranプログラム

```
USE, INTRINSIC :: ISO_C_BINDING
TYPE, BIND(C) :: STR_TYPE        ! 結合する派生型定義にBIND属性を指定
   REAL(C_DOUBLE) :: S1, S2
END TYPE STR_TYPE

INTERFACE
   SUBROUTINE FUNC(X) BIND(C)
      USE, INTRINSIC :: ISO_C_BINDING
      TYPE(C_PTR) :: X
   END SUBROUTINE FUNC
END INTERFACE

TYPE(C_PTR) :: P
TYPE(STR_TYPE),TARGET :: F_STR

P=C_LOC(F_STR)              ! 派生型変数F_STRのCアドレス取得
CALL FUNC(P)                    ! C関数の関数呼出しでF_STRのCアドレスPを渡す
...
```

Cプログラム

```
struct str_type {               // Fortranと結合する構造体定義
    double s1, s2;
} *c_str;

void func(struct str_type **x) {
    c_str = *x;                 // c_strの指示先はF_STRになる
    ...
}
```

(2)　注意事項

- 派生型の成分と構造体のメンバの名前は同じである必要はありません。
- ビットフィールドあるいは可変長配列を含んでいるCプログラムの構造体は結合できません。
- 4倍精度実数型、複素数型を含む派生型と構造体は結合できません。

10.3.5 C のポインタ

CポインタをFortranのデータと結合するとき、派生型**C_PTR**を使用します。

(1)　Fortran プログラムと C プログラムのポインタの結合

FortranプログラムでCプログラムのポインタを使用するとき、派生型**C_PTR**を使用します。

例

Fortranプログラム

```
USE, INTRINSIC :: ISO_C_BINDING
   INTERFACE
   SUBROUTINE FUNC(X) BIND(C)
      USE, INTRINSIC :: ISO_C_BINDING
      TYPE(C_PTR) :: X
   END SUBROUTINE FUNC
END INTERFACE

TYPE(C_PTR) :: P
...
CALL FUNC(P)                                    ! C関数の関数呼出し
...
```

Cプログラム

```
int *a;

void func(int **p) {
    *p = a;                                     // FortranのPの指示先は変数aにな
る
}
```

(2)　Fortran プログラムでの変数の C アドレスの取得

Fortranプログラムの割り付けられた変数のCアドレスを取得するには、組込み手続**C_LOC**を使用します。

例

```
USE, INTRINSIC :: ISO_C_BINDING
INTEGER(C_INT),TARGET :: N
TYPE(C_PTR) :: N_ADDR
N_ADDR = C_LOC(N)                          ! 変数NのCアドレスの取得
```

(3)　C ポインタの比較

Fortranの組込み手続**C_ASSOCIATED**で二つのCポインタの指示先が同じであるかどうかを比較できます。組込み手続**C_ASSOCIATED**は、第1引数と第2引数が指す実体が同じとき真、そうでないとき偽を返却します。第2引数が省略されたとき、第1引数がNULLなら偽、それ以外なら真を返却します。

例

Fortranプログラム

```
MODULE MOD
USE, INTRINSIC :: ISO_C_BINDING
...
INTEGER(C_INT), BIND(C) :: X, Y
TYPE(C_PTR) :: P1, P2
...
END MODULE
PROGRAM MAIN
USE MOD
...
CALL FUNC(P1, P2)                          ! C関数の関数呼出し
  IF ( C_ASSOCIATED(P1, P2) ) THEN         ! 組込み手続C_ASSOCIATEDは、
   ...                                     ! P1とP2が同じ実体を指すとき.true.、
END IF                                     ! そうでないとき.false.を返却する
...
END
```

Cプログラム

```
int x, y;
void func_(int **px, int **py) {
    *px = &x;           // 関数func()をFortranプログラムから呼び出したとき、
    *py = &y;           // FortranのP1、P2の指示先はそれぞれ変数x、yになる
}
```

(4)　C ポインタと Fortran のデータポインタの結合

Fortranの組込み手続**C_F_POINTER**でCポインタをFortranのデータポインタに結合できます。組込み手続**C_F_POINTER**は、第1引数の**C_PTR**を第2引数のデータポインタに結合します。

例

Fortranプログラム

```
MODULE MOD
USE, INTRINSIC :: ISO_C_BINDING
...
INTEGER(C_INT), BIND(C) :: X
TYPE(C_PTR), BIND(C) :: CP
INTEGER(C_INT), POINTER :: FP
...
END MODULE
PROGRAM MAIN
USE MOD
...
CALL FUNC(CP)                   ! C関数の関数呼出し
CALL C_F_POINTER(CP, FP)        ! CポインタCPをデータポインタFPに結合する
...
END
```

Cプログラム

```
int x;
void func_(int **px) {
    *px = &x;               // 関数func()をFortranプログラムから呼び出したと
き、
}                           // FortranのCPの指示先は変数xになる
```

10.3.6 Fortran の共通ブロック

(1)　説明

Fortranプログラムの共通ブロックをCプログラムと結合するとき、共通ブロックに**BIND**属性（BIND(C)）を指定します。**BIND**属性が指定されたFortranプログラムの共通ブロックの成分が1個であるとき、Cプログラムの外部結合をもつ変数と結合できます。また、共通ブロックの成分が複数個あるとき、Cプログラムの外部結合をもつ構造体と結合できます。ただし、共通ブロックの成分と構造体のメンバの数は同じでなければならず、対応する成分・メンバの型が同じでなければなりません。

例

Fortranプログラム

```
USE, INTRINSIC :: ISO_C_BINDING
COMMON /COM1/ F1, F2
COMMON /COM2/ F3
REAL(C_FLOAT) :: F1, F2, F3
BIND(C) :: /COM1/, /COM2/        ! 結合する共通ブロックにBIND(C)を指定する
...
```

Cプログラム

```
// 共通ブロックに複数個の変数が定義されている
// とき、Cプログラムの構造体と結合できる
struct { float f1, f2; } com1;


...
// 共通ブロックに一つだけ変数が定義されている
// とき、Cプログラムの変数と結合できる
float com2;
...
```

(2)　注意事項

- 共通ブロックの成分と構造体のメンバの名前は同じである必要はありません。
- ビットフィールドあるいは可変長配列を含んでいるCプログラムの構造体は結合できません。
- 4倍精度実数型、複素数型を含む共通ブロックと構造体は結合できません。

10.3.7 注意事項

Fortranの複素数型、倍精度複素数型、4倍精度複素数型を、Cの**_Complex**キーワードを使用して宣言された複素数型、倍精度複素数型、4倍精度複素数型に対応させることはできませ

ん。

10.4 関数・手続の型と返却値

ここではC関数、Fortran手続間の返却値の受渡しについて説明します。C++関数は、C関数、Fortran手続から呼び出される、または、それらを呼び出すとき、C結合で定義、宣言されるため、C関数と同じとみなせます。

(1)　整数、論理型、実数型、倍精度実数型、および、4 倍精度実数型の Fortran 手続

「10.3 データ型」で示した対応と同じです。

例　　倍精度実数型のFortran手続の呼出し

呼出し側 (C関数)

```
extern double func_();
...
double a;
a = func_();                    // Fortran手続の呼出し
...
```

呼び出される側 (Fortran手続)

```
REAL(KIND=8) FUNCTION FUNC()
...
FUNC=10.0
...
END FUNCTION FUNC
```

例　　**double**型のC++関数の呼出し

呼出し側 (Fortran手続)

```
REAL(KIND=8) A
...
A = CFUNC()      ! C++関数の呼出し
...
```

呼び出される側 (C++関数)

```
extern "C" {
    double cfunc_();
};
double cfunc_()
{
    double a;
    ...
    return a;
}
```

(2)　複素数型、倍精度複素数型、4 倍精度複素数型

C/C++関数とFortran手続間では、複素数型、倍精度複素数型、4倍精度複素数型に相当する関数を参照できません。

(3)　文字型の関数

Fortranの文字型の関数では、返却値を返却するための二つの引数が追加されます。二つの引数は、返却値である文字列の設定される領域のアドレスとその長さ(バイト数)です。

例　　文字型のFortran手続の呼出し

呼出し側 (C++関数)

```
extern "C" {
    int chfunc_(char *res_p, long res_l);
};
char a[17];                                  // 16バイト+終端1バイト分確保
...
chfunc_(a, 16L);                             // Fortran手続の呼出し
a[16] = '¥0';
...
```

呼び出される側 (Fortran手続)

```
CHARACTER*16 FUNCTION CHFUNC
CHFUNC="THIS IS FORTRAN."
RETURN
END FUNCTION CHFUNC
```

文字列を格納するための領域は、C/C++関数側で確保します。また、Fortranの文字列長は終端のヌル文字を含まないので、C/C++関数側で領域を確保するときは1バイト余計に確保します。

例　　文字型の関数としてC関数を呼び出す

呼出し側 (Fortran手続)

```
SUBROUTINE SUB
CHARACTER*20 CFUNC, CH
INTEGER M
...
CH = CFUNC(M)                                    ! C関数の呼出し
...
END SUBROUTINE SUB
```

呼び出される側 (C関数)

```
extern int cfunc_(char *a, long b, int *p);

int cfunc_(char *a, long b, int *p)
{
    strcpy(a, "THIS IS C++.");
}
```

Fortran手続の第1引数は、C/C++関数の第3引数になります。

(4)　Fortran 手続のサブルーチン

Fortran手続のサブルーチンは、**int**型のC関数に相当します。

10.5 関数・手続間の引数の受け渡し

10.5.1 Fortran 手続の引数

Fortran手続の引数は、すべてアドレスで渡されます。このため、C/C++関数で引数を受け取るとき、引数の領域を指すポインタ値で受け取ります。逆に、Fortran手続に引数を渡すとき、引数のアドレスをFortran手続に渡します。

(1)　Fortran 手続の **VALUE** 属性をもたない引数を渡す

引数として渡す変数のアドレスを渡します。定数は、1度変数に代入してから渡します。

例

呼出し側 (C++関数)

```
extern "C" {
    int func_(int *i, int *j);
}
void cfunc()
{
    int a, b, ret;
    ...
    b = 100;                                    // 定数は変数に代入して渡す
    ret = func_(&a, &b);                        // Fortran手続の呼出し
    ...
}
```

呼び出される側 (Fortran手続)

```
INTEGER FUNCTION FUNC(I, J)
INTEGER I, J
...
END FUNCTION FUNC
```

(2)　Fortran 手続の **VALUE** 属性をもつ引数を渡す

変数の値を渡します。定数は、引数にして渡します。

例

渡す側 (C++関数)

```
extern "C" {
    int func_(int i, int j);
}
void cfunc()
{
    int a, ret;
    ...
    ret = func_(a, 100);                // Fortran手続の呼出し
    ...
}
```

受け取る側 (Fortran手続)

```
INTEGER FUNCTION FUNC(I, J)
INTEGER, VALUE :: I, J                    ! VALUE属性を指定
...
END FUNCTION FUNC
```

(3)　Fortran 手続の **VALUE** 属性をもたない引数を受け取る

引数をポインタで受け取ります。

例

渡す側 (Fortran手続)

```
SUBROUTINE SUB
INTEGER K, I, J
...
CALL C_FUNC(I, J)
...
END SUBROUTINE SUB
```

受け取る側 (C++関数)

```
extern "C" {
    int c_func_(int *a, int *b);
}
int c_func_(int *a, int *b)       // 引数をポインタで受け取る
{
    ...
}
```

(4)　Fortran 手続の **VALUE** 属性をもつ引数を受け取る

変数の値を渡します。定数は、引数に指定して渡します。

例

渡す側 (Fortran手続)

```
SUBROUTINE SUB
  INTERFACE
    INTEGER(C_INT) FUNCTION C_FUNC(A, B)
        USE, INTRINSIC :: ISO_C_BINDING
        INTEGER(C_INT), VALUE :: A, B                    ! VALUE属性を指定
      END FUNCTION C_FUNC
  END INTERFACE
INTEGER K, I, J
...
CALL C_FUNC(I, J)
...
END SUBROUTINE SUB
```

受け取る側 (C++関数)

```
extern "C" {
    int c_func_(int a, int b);
}
int c_func_(int a, int b)                              // 引数を値で受け取る
{
    ...
}
```

10.5.2 留意事項

10.5.2.1暗黙に追加される引数

Fortran手続ではソースファイルに記述された引数のほかに、暗黙的に引数が追加されることがあります。次の引数が追加されます。

- 呼び出すFortran手続が文字型の関数のとき、関数の返却値を格納する領域のアドレスとその返却値の長さ(単位:バイト)
- 引数が文字型の要素であるとき、その引数の長さ(単位:バイト)
- 引数が手続名のとき、手続の返却値のサイズ(単位:バイト)。
  ただし、文字型の関数以外は0

上記の引数が渡される順番は以下のとおりです。

(1)　返却値を格納する領域のアドレス(呼び出す手続が文字型のときのみ)

(2)　返却値のサイズ(呼び出す手続が文字型のときのみ)

(3)　ソースファイルに記述された引数

引数が文字型のときその長さ、手続名のとき返却値のサイズが、それぞれの引数の直後に渡されます。

10.6 プログラムのリンク

10.6.1 C プログラムと Fortran プログラムのリンク

CプログラムとFortranプログラムをリンクするとき、Fortranコンパイラ(nfort)でリンクします。

例

```
$ nfort -c a.f90          (Fortranプログラムのコンパイル)
$ ncc -c b.c              (Cプログラムのコンパイル)
$ nfort a.o b.o           (Fortranコンパイラによるリンク)
```

10.6.2 C++プログラムと Fortran プログラムのリンク

C++プログラムとFortranプログラムをリンクするとき、Fortranコンパイラ(nfort)でリンクします。リンクするとき、C++コンパイラの実行時ライブラリ(-**cxxlib**)を指定します。

例

```
$ nfort-c a.f90           (Fortranプログラムのコンパイル)
$ nc++ -c b.cpp           (C++プログラムのコンパイル)
$ nfort a.o b.o -cxxlib   (Fortranコンパイラによるリンク)
```

10.7 実行時の注意事項

C/C++プログラムとFortranプログラムをリンクしたとき、C/C++関数で**stdin**、**stdout**、**stderr**をクローズしないでください。クローズしたとき、Fortran手続の動作は保証されません。

第11章　メッセージ

11.1 診断メッセージ

コンパイラは、プログラムの最適化状況を示すメッセージを標準エラー出力、診断メッセージリストに出力します。本セクションでは、それらの形式、主なメッセージについて説明します。

11.1.1 メッセージの形式

診断メッセージは次の形式で出力されます。

種別（番号）: 位置情報: メッセージ本文［: ヒント］

種別 (番号):

メッセージの種別とメッセージ本文に割り当てられた番号が表示されます。種別には以下があります。

vec:　ベクトル化

opt:　最適化・ベクトル化

dtl:　最適化・ベクトル化のより詳細な情報

inl:　インライン展開

par:　OpenMP並列化・自動並列化

err:　主にOpenMP指示行指定時の問題

位置情報:

診断メッセージ対応するソースコードの行番号が出力されます。標準エラー出力に出力されたとき、行番号の行を含むファイル名も出力されます。

メッセージ本文:

診断メッセージの本文が出力されます。

ヒント:

診断メッセージによっては、手続名、変数名、配列名などが出力されます。

- 変数名、配列名が不明であるとき、型名が出力されることがあります。
- コンパイラが最適化のために生成した関数名、変数名としてベースとなった関数名、変数名に「$数値」が付加されて出力されることがあります。

11.1.2 診断メッセージ一覧

vec(101): Vectorized loop.

ループ全体がベクトル化された。

vec(102): Partially vectorized loop.

ループが部分ベクトル化された。

vec(103): Unvectorized loop.

ループがベクトル化されなかった。

vec(107): Iteration count is too small.

ループの繰返し数がベクトル化の閾値より小さいためベクトル化しない。ベクトル化の閾値は**-mvector-threshold=***n*で変更できる。

vec(108): Unvectorizable loop structure.

ループの繰返しを制御する変数やループ構造がベクトル化に適していないため、ベクトル化できない。主に次のような場合に出力される。

- ループの繰返しを制御する変数が別の型に型変換されている。**-mreplace-loop-induction**でベクトル化できる場合がある。
- ループの繰返しの終了判定式が、ループの繰返しを制御する変数とループ内不変の式の比較でない。
- ループ終了の条件式に、||、&&が現れている。
- ループ終了の条件式に、!=、==が現れている。**-mreplace-loop-equation**でベクトル化できる場合がある。
- ループの中から外への分岐(飛出し)が2個以上含まれる。
- ループ外からループ内への分岐(飛込み)がある。**if**文と**goto**文からなるループの場合に考えられる。
- 部分ベクトル化しようとしたが、そのための作業ベクトルが作成できない。次の例では、部分ベクトル化するためにa[0]の作業ベクトルが必要だが、その型はベクトル化できない型であり作業ベクトルを作成できない。

```
void func(
    int n,
    long double _Complex a[n],
    double _Complex b[n],
```

```
    double _Complex c[n],
    double _Complex d[n]
)
{
    for (int i = 0; i < n; i++) {
        a[0] = b[i] + d[i] + c[i];
        c[i] = a[0];
    }
}
```

vec(109): Vectorization obstructive statement.

ベクトル化できない文である。

vec(110): Vectorization obstructive function reference : 関数名

ベクトル化できない関数の呼出しがある。

vec(111): “novector” is specified.

novector指示行が指定されたためベクトル化しない。

vec(112): “novwork” is specified.

novwork指示行が指定されたため部分ベクトル化しない。

vec(113): Overhead of loop division is too large.

ループ分割によるオーバーヘッドが大きくベクトル化の効果がないため、部分ベクトル化しない。

vec(115): Internal table overflow.

ベクトル化処理中に内部テーブルの大きさが足りなくなったため、ベクトル化できない。

vec(116): Unvectorizable function reference. : 関数名

ベクトル化できないユーザ関数の呼出しがある。関数ポインタを介して関数を呼び出しているとき関数名は出力されない。

vec(117): Unvectorizable statement.

ベクトル化できない文がある。

vec(118): Unvectorizable data type.

ベクトル化できないデータ型の参照がある。

vec(119): Array is not aligned. : 変数名

配列、または、ポインタの指示先が、ベクトル化できるメモリ境界に整列されていない。

vec(120): Unvectorizable dependency. : 変数名

ベクトル化不可の依存関係がある。

vec(121): Unvectorizable dependency.

ベクトル化不可の依存関係がある。

vec(122): Dependency unknown. Unvectorizable dependency is assumed. : 変数名

依存関係がわからないためベクトル化不可の依存関係が存在すると仮定する。変数名が不明であるとき、変数名は出力しない。ループに**ivdep**指示行を指定すると本依存関係をベクトル化できるものとみなしベクトル化を適用する。

vec(124): Iteration count is assumed. Iteration count=n

ループの繰返し数を最大*n*回であると仮定した。

vec(126): Idiom detected. : マクロ演算の種別

ベクトルマクロ演算が検出された。マクロ演算の種別は以下である。

Max/Min、List Vector、Sum、Product、Bit-op、Iteration、Search

vec(128): Fused multiply-add operation applied.

ベクトルFMA命令を使用した。

vec(129): Array is retained. : 配列名

retain指示行が適用された。

vec(130): Vector register is assigned.: 配列名

vreg指示行により、配列名にベクトルレジスタが割り当てられた。

vec(131): Too many statements.

文の数が多すぎるため、ベクトル化できない。

vec(132): Too many function calls.

関数呼出しの数が多すぎるためベクトル化できない。

vec(133): Too many memory refereneces.

メモリ参照の数が多すぎるためベクトル化できない。

vec(134): Too many branches.

分岐の数が多すぎるためベクトル化できない。

vec(139): Packed loop.

packed-vector命令を使用してベクトル化した。

vec(140): Unpacked loop. : 理由

-mvector-packed、または、**packed_vector**指示行が指定されたが、ループがpacked-vector命令を使用してベクトル化できなかった。

vec(141): “nopacked_vector” is specified.

nopacked_vector指示行が指定された。

vec(142): pvreg is used in vector loop.

pvreg指示行で指定された配列がpacked-vector命令を使用されずにベクトル化されたループに含まれる。

vec(143): vreg is used in packed vector loop.

vreg指示行で指定された配列がpacked-vector命令を使用しベクトル化されたループに含まれる。

vec(161): Structure assignment obstructs vectorization.

構造体、共用体、クラスの代入が含まれるためベクトル化できない。

vec(163): Exception handling obstructs vectorization.

C++例外処理に関わる処理が含まれるためベクトル化できない。

vec(184): Division obstructs vectorization.

ベクトル化できない型の除算が含まれるためベクトル化できない。

vec(185): Exponentiation obstructs vectorization.

ベクトル化できない型のべき乗算が含まれるためベクトル化できない。

opt(1011): Too large to optimize -- reduce program or loop size.

ループ、または、ルーチンが大きすぎるため最適化できない。ループ、ルーチンの分割を検討する。

opt(1019): Feedback of scalar value from one loop pass to another.

スカラ変数が異なる繰返しで定義された値を参照しているため最適化できない。

opt(1025): Reference to this function inhibits optimization.

最適化を阻害する関数呼出しがあるため最適化できない。

opt(1034): Multiple store conflict.

同一の配列要素が複数回定義されるため最適化できない。

opt(1037): Feedback of array elements.

異なる繰返しで同一の配列要素を定義・参照しているため最適化できない。

opt(1038): Loop too complex -- optimization of this loop halted.

ループが複雑すぎるため最適化できない。

opt(1056): Loop nest too deep for optimization.

ループのネストが深すぎるため最適化できない。

opt(1057): Complicated use of variable inhibits loop optimization.

変数の定義/参照が複雑で最適化できない。

opt(1059): Unable to determine last value of scalar temporary.

スカラ変数の終値が確定できないため最適化できない。

opt(1061): Use of scalar under different condition causes feedback.

スカラ変数が異なる条件下で参照されているため最適化できない。

opt(1062): Too many data dependency problems.

データ依存が多すぎるため最適化できない。

opt(1082): Backward transfers inhibit loop optimization.

逆方向分岐があるため最適化できない。

opt(1083): Last value of promoted scalar required.

作業配列化されたスカラ変数の終値が保証できないため最適化できない。

opt(1084): Branch out of the loop inhibits optimization.

ループからの飛出しがあるため最適化できない。

opt(1097): This statement prevents loop optimization.

最適化を阻害する文があり最適化できない。

opt(1117): Indirect branch inhibits to optimization of loop.

間接分岐があるため最適化できない。

opt(1128): Branching too complex to optimize at this optimization level.

分岐が複雑で最適化できない。

opt(1130): Conditional scalar inhibits optimization of outer loop.

条件下で定義されるスカラ変数が外側ループの最適化を阻害している。

opt(1131): Function references in iteration count inhibits optimization.

ループの繰返し制御に現われる関数参照があり最適化できない。

opt(1166): Potential dependency due to pointer -- use restrict qualifier if ok.

ポインタの参照先の定義・参照関係に依存がある可能性があり最適化しない。ループに**ivdep**指示行を指定すると本依存関係を最適化できるものとみなしベクトル化を適用する。

inl(1214): Expansion routine is too big for automatic expansion.: ルーチン名

ルーチンが大きすぎるので自動インライン展開しない。**-finline-max-function-size**=*n*、**-finline-max-times**=*n*で展開するルーチンの大きさを調整するとインライン展開できる場合がある。

inl(1219): Nesting level too deep for automatic expansion. : ルーチン名

インライン展開するルーチン呼のネストが深すぎるためインライン展開しない。

-finline-max-depth=*n*で展開するルーチンの呼出しの深さを調整するとインライン展開できる場合がある。

inl(1222): Inlined.: ルーチン名

ルーチン名がインライン展開された。

opt(1268): Use of pointer variable inhibits optimization.

ポインタ変数があるため最適化できない。

opt(1282): This store into array inhibits optimization of outer loop.

配列への代入が外側ループの最適化を阻害している。

opt(1285): Not enough work to justify concurrency optimization.

ループ中の作業量が少ないため自動並列化しない。

opt(1298): Use of induction variable outside the loop inhibits optimization.

インダクション変数がループ外で参照されているため最適化できない。

opt(1299): Redefinition of induction variable in loop inhibits optimization.

インダクション変数がループ内で再定義されているため最適化できない。

opt(1315): Iterations peeled from loop in order to avoid dependence.

ベクトル化不可の依存関係を除去するため、ループの前方/後方展開を行った。

opt(1376): User function reference inhibits optimization.

関数参照があるため最適化できない。

opt(1377): Must synchronize to preserve order of accesses.

同期制御が必要であるため最適化できない。

opt(1378): Many synchronizations needed.

同期制御が多数必要なため並列化しない。

opt(1380): User function references not ok without "cncall".

関数参照があるため並列化できない。**cncall**指示行を指定すると並列化できる場合がある。

inl(1388): Inlining inhibited: OpenMP or parallel directive.

OpenMP指示行、自動並列化の指示行があるためインライン展開できなかった。

opt(1395): Inner loop stripped and strip loop moved outside outer loop.

外側ループストリップマイニングを適用した。

opt(1408): Loop interchanged.

ループを入れ換えた。

opt(1409): Alternate code is generated.

条件ベクトル化を適用した。

opt(1589): Outer loop moved inside inner loop(s).

外側ループを内側ループと入れ換えた。

opt(1590): Inner loop moved outside outer loop(s).

内側ループを外側ループと入れ換えた。

opt(1592): Outer loop unrolled inside inner loop.

外側ループをアンローリングした。

opt(1593): Loop nest collapsed into one loop.

ループを一重化した。

opt(1772): Loop nest fused with following nest(s).

後続のループと融合した。

opt(1800): Idiom detected (matrix multiply).

多重ループを行列積ライブラリ呼出しに置換した。

11.2 実行時メッセージ

コンパイラの実行時ルーチンは、プログラムの実行時に検出したプログラムの問題を標準エラー出力に出力します。本セクションでは、それらの主なメッセージについて説明します。

C++ runtime abort: terminate() called by the exception handling mechanism.

C++ exception handling機構によってteminate()関数が呼び出された。

C++ runtime abort: returned from a user-defined terminate() routine.

ユーザ定義のterminate()関数からreturnしてきた。

C++ runtime abort: internal error: static object marked for destruction more than once.

デストラクタで破壊されるべきオブジェクトが二度以上破壊された。

C++ runtime abort: a pure virtual function was called.

純粋仮想関数が呼び出された。

C++ runtime abort: invalid dynamic cast.

サブオブジェクトへのdynamic_cast演算が不正である。

C++ runtime abort: invalid typeid operation.

typeid演算が不正である。

C++ runtime abort: freeing array not allocated by an array new operation.

new演算子でアローケーとされていない領域が解放された。

C++ runtime abort: terminate() called itself recursively.

terminate()関数が再帰的に呼び出された。

C++ runtime abort: a deleted virtual function was called.

delete仮想関数が呼び出された。

Compatibility Error: veos (older than v2.6.0) and ve_exec (vVEOS-verision) are not compatible

veosが古くve_execと互換がない。コンテナ上でVEプログラムを実行している場合は、ホストマシンに入っているveosのバージョンが古い可能性がある。最新のveosパッケージをホストマシンにインストールしてください。

Compatibility Error: veos (vVEOS-version-A) and ve_exec (vVEOS-verision-B) are not compatible

veosが古くve_execと互換がない。コンテナ上でVEプログラムを実行している場合は、ホストマシンに入っているveosのバージョンが古い可能性がある。最新のveosパッケージをホストマシンにインストールしてください。

Failed to load EXEC DATA (fixed): Error Message

実行ファイルのデータ領域の読み込みに失敗した。VEメモリ不足の可能性がある。実行中の他のVEプロセスがあればそれを終了させるか、データ領域のサイズを減らしてください。なお、VE搭載メモリ量と現在のVEメモリ使用量は/opt/nec/ve/bin/free -hで取得できる。

Failed to load EXEC DATA (fixed, fileback): Error Message

実行ファイルのデータ領域の読み込みに失敗した。VEメモリ不足の可能性がある。実行中の他のVEプロセスがあればそれを終了させるか、データ領域のサイズを減らしてください。なお、VE搭載メモリ量と現在のVEメモリ使用量は/opt/nec/ve/bin/free -hで取得できる。

Unable to grow stack

スタックのサイズが足りない。環境変数**VE_LIMIT_OPT**を利用して、次の例のように利

用可能なスタックのサイズ上限を増やしてください。

```
export VE_LIMIT_OPT="-s 8192"
```

現在のスタックサイズの上限は、ve_execコマンドの--show-limitで確認できる。

```
$ ve_exec --show-limit
core file size          (blocks, -c) 0                    0
data seg size           (kbytes, -d) unlimited            unlimited
pending signals                 (-i) 379523               379523
max memory size         (kbytes, -m) unlimited            unlimited
stack size              (kbytes, -s) unlimited            unlimited  <--
cpu time               (seconds, -t) unlimited            unlimited
virtual memory          (kbytes, -v) unlimited            unlimited
```

VE Node node-number is UNAVAILABLE

*node-number*のVEカードに障害が発生した。他のVEノードを利用して、ジョブを実行してください。

第12章　トラブルシューティング

12.1 プログラムのコンパイルに関するトラブルシューティング

“Fatal: License: Unknown host.” というコンパイルエラーが発生する。

コンパイラのライセンスチェック時にライセンスサーバーにアクセスできない問題が発生している可能性があります。以下のHPC ソフトウェアライセンス発行のページに記載されているFAQを参照してください。解決しなければ、同ページよりお問い合わせください。

https://www.hpc-license.nec.com/aurora/

指示行に対して、”Syntax error”というコンパイルエラーが発生する。

指示行の綴り、使い方が間違っていないかを確認してください。SXシリーズ向けコンパイラの指示行に対してエラーとなっているときは、コンパイラ指示行変換ツールなどでVEコンパイラの指示行に変換してください。コンパイラ指示行変換ツールについては「付録 C コンパイラ指示行変換ツール」を参照してください。

”Error: Invalid suffix”というアセンブルエラーが発生する。

binutils-veパッケージが古い可能性があります。binutils-veパッケージが最新版であるか確認してください。

ヘッダファイル、ライブラリを利用するときにコンパイラ、リンカが参照するディレクトリを確認したい

「1.6 #includeで取り込まれるファイルのサーチ」、「1.7 ライブラリのサーチ」を参照してください。

”undefined reference to `ftrace_region_begin' / `ftrace_region_end'”というリンクエラーが発生する。

FTRACE機能が使用されています。**-ftrace**を指定してリンクしてください。FTRACE機能については「PROGINF/FTRACE ユーザーズガイド」を参照してください。

```
$ ncc a.o b.o -ftrace
```

"undefined reference to '__vthr$_barrier'"というリンクエラーが発生する。

-mparallel、または、**-fopenmp**を指定してリンクしてください。

"undefined reference to '__vthr$_pcall_va"というリンクエラーが発生する。

-mparallel、または、**-fopenmp**を指定してリンクしてください。

"cannot find -lveproginf"、および、"cannot find -lveperfcnt"というリンクエラーが発生する。

nec-veperfパッケージがインストールされていません。nec-veperfパッケージをインストールしてください。

VE向けの実行ファイルであることを確認したい。

「/opt/nec/ve/bin/nreadelf -h」に実行ファイルを指定して実行してください。「Machine:」の行に「NEC VE architecture」と出力されていれば、VE向けの実行ファイルであることを示します。

```
$ /opt/nec/ve/bin/nreadelf -h a.out
ELF Header:
  Magic:   7f 45 4c 46 02 01 01 00 00 00 00 00 00 00 00 00
  Class:                             ELF64
  Data:                              2's complement, little endian
  Version:                           1 (current)
  OS/ABI:                            UNIX - System V
  ABI Version:                       0
  Type:                              EXEC (Executable file)
  Machine:                           NEC VE architecture
  (...)
```

自動並列とOpenMP並列を混ぜてリンクするとき、-fopenmp、-mparallelのどちらを指定すればよいか。

-fopenmp、**-mparallel**のどちらかを指定してリンクしてください。

```
$ ncc -c -mparallel a.c
$ ncc -c -fopenmp b.c
```

```
$ ncc -fopenmp a.o b.o
```

-ftraceを指定すると、実行時間が異常に長くなる。

性能情報を取得するルーチンが実行されるため長くなります。このルーチンは関数の入口、出口、ユーザ指定リージョンの前後で呼び出されます。

性能情報を取得したい関数を含むソースファイルのみ**-ftrace**を指定してください。

OMP_NUM_THREADSで8より大きい値を指定しても、その数のスレッドが生成されない。

VEのコア数は8個であるため、8スレッドが上限です。

定義済みマクロの名前、値を知りたい。

「9.2.4 定義済みマクロ」を参照してください。

C++プログラムのリンク時に次のようなエラーが発生した。

```
/opt/nec/ve/bin/nld: __curr_eh_stack_entry: TLS reference in /tmp/nccwvkaaa.o mismatches non-TLS reference in /opt/nec/ve/ncc/2.x.x/lib/libnc++.a(iostream.o)
/opt/nec/ve/ncc/2.x.x/lib/libnc++.a: error adding symbols: Bad value
```

バージョン2.2.1以降のコンパイラでプログラムを再コンパイルしてください。

コードサイズの大きいプログラムをコンパイルしたときにSIGSEGVでアボートする。

コンパイラが必要とするスタック領域の容量が設定の上限を超えている可能性があります。ulimitコマンドでスタックサイズの上限を拡大することで解消することがあります。

以下のように「ulimit –s」でスタックサイズの上限値を確認できます。「ulimit –s (スタックサイズの上限値)」で上限値を変更できますので、「ulimit -s」で出力された上限値よりも大きな値を設定し、再度コンパイルしてください。

```
$ ulimit -s              (値の確認)
8192
$ ulimit -s 16384        (値の変更)
```

コンパイル時にSIGKILLが発生する。

コンパイルを行ったマシンのメモリが不足している可能性があります。**-O0**、**-O1**により最適化レベルを落とすことでメモリ使用量をある程度少なくすることができます。

FortranプログラムとC/C++プログラムのオブジェクトファイルを混在リンクしたい。

「10.6 プログラムのリンク」を参照してください。

SXシリーズで利用していたプログラムのオプションをVector Engine向けに変更したい。

「付録 B SXシリーズ向けコンパイラとの対応」を参照し、SXのコンパイラオプションに対応するVEのコンパイラオプションに変更してください。

SXシリーズで利用していたプログラムの指示行をVector Engine向けに変更したい。

SXのコンパイラ指示行をVEのコンパイラ指示行に変換するツールが利用できます。

「付録 C コンパイラ指示行変換ツール」を参照してください。

または、「付録 B SXシリーズ向けコンパイラとの対応」を参照し、SXのコンパイラオプションに対応するVEのコンパイラ指示行に変更してください。

診断メッセージに、名前$1など、'$'の後に数字のある変数、ルーチン名が表示される。これらは何か？

ベクトル化、並列化のためにコンパイラが作成した変数、ルーチンです。

診断メッセージに、変数名ではなくDOUBLE、floatなどの型名のみが表示されることがある。これらは何か？

ベクトル化、並列化のためにコンパイラが作成した名前なし変数で、名前の代わりに型名を表示しています。

コンパイル時に指定した覚えのないコンパイラオプションが有効になっている。

コンパイラオプションがオプションファイルに指定されている可能性があります。オプションファイルの詳細については「1.5 コンパイラオプションの指定」を参照してください。

コンパイラのバージョンを確認したい。

--versionを指定してください。

12.2 プログラムの実行に関するトラブル

実行時に“Node 'N' is Offline”というエラーが発生する。

ノード番号NのVEがOFFLINE状態になっています。VEをONLINE状態にしてください。以下は0番のVEをONLINE状態にするときの例です。

```
# /opt/nec/ve/bin/vecmd -N 0 state set on
...
Result: Success
# /opt/nec/ve/bin/vecmd state get
...
------------------------------------------------------------------
VE0 [03:00.0] [ ONLINE ] Last Modif:2017/11/29 10:18:00
------------------------------------------------------------------
Result: Success
```

実行時に使用されているノードを確認したい。

/opt/nec/ve/bin/psを実行してください。psコマンドを実行するとVEのノードごとに現在実行されているプロセスのスナップショットを出力します。以下の例では、2番のVEノードでa.outというプログラムが実行中であることが確認できます。

```
/opt/nec/ve/bin/ps -a
VE Node: 3
  PID TTY          TIME CMD
VE Node: 1
  PID TTY          TIME CMD
VE Node: 2
  PID TTY          TIME CMD
50727 pts/1    00:01:36 a.out
VE Node: 0
  PID TTY          TIME CMD
```

実行時に"./a.out: error while loading shared libraries: libncc.so.2: cannot open shared object file: No such file or directory"というエラーが出力される。

実行環境にパッケージnec-nc++-shared、nec-nc++-shared-instをインストールしてください。手順は「インストレーションガイド」を参照してください。

実行時にダイナミックリンクライブラリが見つからないというエラーが発生する。

共有ライブラリを配置しているディレクトリを環境変数**VE_LD_LIBRARY_PATH**に設定してください。環境変数**VE_LD_LIBRARY_PATH**については「2.2 実行時に参照される環境変数」を参照してください。

例外発生時に例外発生個所がソースファイルの何行目に対応しているか確認したい。

トレースバック情報から調べることができます。手順は「1.8.3 トレースバック機能との連携」を参照してください。

例外発生時にトレースバック機能で表示される例外発生個所が正しくない。

HWによる命令の先行制御によってトレースバック機能で出力される例外発生個所が正しく表示されないことがあります。環境変数**VE_ADVANCEOFF**=YESを設定することで先行制御を無効にできます。先行制御を無効にすることで実行時間が大幅に長くなることがありますのでご注意ください。

```
$ export VE_ADVANCEOFF=YES
```

未初期化ローカル変数を使用していないか確認したい。

double型の未初期化ローカル変数については**-minit-stack**=snanを指定してコンパイル、実行し、例外を検出することで確認できることがあります。float型の未初期化ローカル変数について確認するときは、snanの代わりにsnanfをそれぞれ指定します。ローカル変数がdouble、float型でないとき、この方法で確認することはできません。

未初期化ローカル変数の不定値を参照して、プログラムが異常終了するのを回避したい。

-minit-stack=zeroを指定してコンパイル、実行してください。未初期化の領域をゼロクリアすることで異常終了を回避できることがあります。潜在的な問題を解決するためにもプログラムの修正を推奨します。

自動並列化、OpenMP機能を使用したプログラムを実行したとき、"Unable to grow stack"、または、SIGSEGVでプログラムが異常終了する。

スタックの使用量が上限を超えている可能性があります。以下の方法でスタックの上限を拡張してください。

- 環境変数**OMP_STACKSIZE**によって各スレッドが使用するスタックサイズの上限を拡張することができます。

```
$ export OMP_STACKSIZE=2G
```

プログラムが実行時に何スレッドで動作したかを確認したい。

PROGINFのMax Active Threadsを参照してください。Max Active Threadsは環境変数**VE_PROGINF**にDETAILが設定されているとき、プログラムの実行終了時に標準エラー出力に出力されます。詳細は「PROGINF/FTRACE ユーザーズガイド」を参照してください。

以下の例では、Max Active Threadsが4であるため、4スレッドで動作したことが確認できます。

```
******** Program Information ********
(...)
Power Throttling (sec)          :        0.000000
Thermal Throttling (sec)        :        0.000000
Max Active Threads          :        4
Available CPU Cores         :        8
Average CPU Cores Used          :        3.323850
Memory Size Used (MB)           :        7884.000000
Start Time (date)               :        Mon Feb 19 04:43:34 2018 JST
End Time (date)             :        Mon Feb 19 04:44:08 2018 JST
```

自動並列化、OpenMP機能を使用したプログラムを実行したとき、スレッドがいつ生成、解放されるのか知りたい。

既定の動作では、環境変数**OMP_NUM_THREADS**、または、**VE_OMP_NUM_THREADS**によりスレッド数が指定されたときその個数、そうでないときプログラムで利用可能なVEコア数と同じ個数のスレッドが、プログラムの実行開始前に生成され、終了時に解放されます。

詳細については、「7.3.2 スレッドの生成・解放」を参照してください。

ベクトル化するとバスエラーが発生する。

4バイトアラインされた配列を8バイト要素のベクトル命令でロード/ストアしている可能性があります。

以下の例では、引数で渡された**float**型(4バイトアライン)の配列a、bが**uint64_t**型にキャストされているため、8バイト要素のベクトル命令でロード/ストアされます。

```
void func1(){
    float a[511],c[511];
    ...
    func2(a,b);
}

void func2( void* a, void* b ){
    for(int i=0; i<255; ++i){ //!!!<---vectorized loop
        ((uint64_t*)b)[i] = ((uint64_t*)a)[i];
    }
}
```

次のように配列を8バイトアラインするか、**novector**指示行でベクトル化を抑止してください。

```
float a[511] __attribute__((aligned(8)));
float b[511] __attribute__((aligned(8)));
```

12.3 プログラムのチューニングに関するトラブルシューティング

プログラムにどの最適化が適用されたかを確認したい

コンパイル時に出力される診断メッセージ、および、編集リストを参照してください。診断メッセージリストは**-report-diagnostics**、編集リストは**-report-format**を指定したとき出力されます。

詳細は、「第8章 コンパイルリスト」を参照してください。

ベクトル化を促進したにもかかわらず、性能が低下する。

ベクトル化対象のループの繰り返し数が少ない場合、ベクトル化のためのオーバーヘッドにより性能が低下する場合があります。このようなループは**novector**指示行で自動ベクトル化を抑止してください。

自動並列化、OpenMP機能を使用したプログラムを実行したとき、PROGINFとFTRACEの同項目で表示される値が異なる。

主プログラムの実行開始前に自動生成されるスレッドのスピンウェイト分の演算数等はPROGINFでは加算されますが、FTRACEでは加算されません。

!$omp parallel num_threads(4)を指定して、環境変数OMP_NUM_THREADS=4、OMP_NUM_THREADS=5でそれぞれ実行した場合、OMP_NUM_THREADS=5のほうが並列数が多いにもかかわらず、実行時間が長くなる。

num_threads句で渡される値が、環境変数OMP_NUM_THREADSで指定している値と違う場合には、スレッドの再生成による実行時間の増加が発生します。

主プログラムの実行開始前にスレッドが自動生成されます。この時生成されるスレッド数は、環境変数OMP_NUM_THREADSの値で決まります。プログラム中で、OpenMPのomp_set_thread_num()関数やnum_threads句の指定により、スレッド数が変更される場合、実行開始時に自動生成されたスレッドは解放され、新たにスレッドが再生成されます。

12.4 インストールに関するトラブルシューティング

正しくインストールできているかを確認したい

--versionを指定し、バージョンを確認します。表示されたバージョン番号がインストールした物件と同じであれば正しくインストールできています。以下の*X.X.X*にバージョン番号が表示されます。

```
$ /opt/nec/ve/bin/ncc --version

ncc (NCC) X.X.X (Build 14:10:47 Apr 23 2020)

Copyright (C) 2018,2020 NEC Corporation.
```

過去のバージョンのコンパイラをインストールしたい。

過去のバージョンのコンパイラのインストール手順は、SX-Aurora TSUBASA インストレーションガイドの「A.1.1 特定バージョンのコンパイラのインストール」を参照してください。

過去のバージョンのコンパイラを利用したい。

コンパイル時に /opt/nec/ve/bin/nfort-*X.X.X*、ncc-*X.X.X*、または、nc++-*X.X.X*(*X.X.X* はコンパイラのバージョン番号)を起動してください。

詳細については「1.2 コンパイラの起動」を参照してください。

過去のバージョンのコンパイラをデフォルトにしたい。

各バージョンのncc/nc++/nfortコマンドの実体は以下のようにインストールされています。

ここでは、*X.X.X*はバージョン番号です。

```
/opt/nec/ve/ncc/X.X.X/bin/ncc
/opt/nec/ve/ncc/X.X.X/bin/nc++
/opt/nec/ve/nfort/X.X.X/bin/nfort
```

デフォルトにしたいバージョンの**bin**ディレクトリをコマンドサーチパス(環境変数**PATH**)に設定してください。

第13章　旧バージョンとの互換に関する注意事項

(1) NEC C/C++ コンパイラのバージョン 1.X.X とバージョン 2.0.0 以降 には、互換性がありません。そのため、バージョン 1.X.X で作成したオブジェクトファイルはバージョン 2.0.0 以降で作成したオブジェクトファイルとリンクできません。

(2) バージョン 2.2.2 以降でコンパイラの実行時ルーチンを共有ライブラリでも提供しています。このため、バージョン 2.1.2 以前のコンパイラで作成した共有ライブラリは、2.2.2 以降のコンパイラで再コンパイルしてください。

(3) C++プログラムでリンク時に次のエラーが発生したとき、バージョン 2.2.2 以降のコンパイラで、オブジェクトファイルを再コンパイルしてください。

```
/opt/nec/ve/bin/nld: __curr_eh_stack_entry: TLS reference in /tmp/nccwvkaa
a.o mismatches non-TLS reference in /opt/nec/ve/ncc/2.X.X/lib/libnc++.a(ios
tream.o)
/opt/nec/ve/ncc/2.X.X/lib/libnc++.a: error adding symbols: Bad value
```

(4) バージョン 2.2.2 以降のコンパイラで作成した実行ファイルを実行するには、バージョン 2.21-4 以降の glibc-ve パッケージに含まれるダイナミックリンカが必要です。実行がうまくいかないとき、glibc-ve パッケージのバージョンをご確認ください。

```
$ rpm -q glibc-ve
glibc-ve-2.21-4.el7.x86_64
```

(5) バージョン 2.2.2 以降のコンパイラは既定値で共有ライブラリをリンクするため、バージョン 2.1.2 以前に比べ、ダイナミックリンク処理のためのオーバーヘッドにより実行性能が低下することがあります。このオーバーヘッドによる性能低下を回避するには、**-static**、または、**-static-nec** を指定して静的ライブラリをリンクしてください。

備考　**-static**、または、**-static-nec**を指定して作成した実行ファイルを実行するとき、結果不正や異常終了などにより正しく実行できないことがあります。

付録 A　コンフィギュレーションファイル

A.1　概要

コンフィギュレーションファイルを使用することで、コンパイラの既定の設定を変更できます。コンフィギュレーションファイルは、**-cf**で指定します。

コンフィギュレーションファイルは、以下の形式で記述します。

```
キーワード:値
```

コンフィギュレーションファイルには以下のキーワードを指定できます。

キーワード	説明
veroot	VEのルートディレクトリを指定します。 (既定値：/opt/nec/ve)
system	コンパイラシステムのルートディレクトリを指定します。 (既定値：/opt/nec/ve/ncc/version)
as	アセンブラを指定します。 (既定値：<veroot>/bin/nas)
ccom	C/C++コンパイラを指定します。 (既定値：<system>/libexec/ccom)
ld	リンカを指定します。 (既定値：<veroot>/bin/nld)
cc_pre_options cc_post_options	コンパイラオプションを指定します。コンパイラへは以下の順で指定されます。 <cc_pre_options><user-specified-cc-options><cc_post_options>
as_pre_options as_post_options	アセンブラオプションを指定します。アセンブラへは以下の順で指定されます。 <as_pre_options><user-specified-ld-options><as_post_options>
ld_pre_options ld_post_options	リンカオプションを指定します。リンカへは以下の順で指定されます。 <ld_pre_options><user-specified-ld-options><ld_post_options>
startfile	スタートアップファイルを指定します。
endfile	スタートアップファイルを指定します。リンカオプションの末尾に指定されます。

A.2　記述方法

- キーワードと値は、コロン(:)で区切ります。
- キーワードを記述しないときは既定値が使用されます。
- 区切り文字のコロンの前後は空白を指定できます。
- 値は複数行に指定でき、空白行または次のキーワードまでを値とみなします。複数行にわたり値を指定するときは改行部分に'¥'を指定します。

例

```
cc_pre_options: -I /tmp  ¥
-I /tmp2
```

- 同じキーワードを複数回指定したときは、最後のキーワードが有効になります。

A.3　使用例

コンフィギュレーションファイルの使用例を説明します。

- VEのルートディレクトリとコンパイラシステムのルートディレクトリを変更する。
  verootキーワードとsystemキーワードに変更後の値を指定します。

```
veroot: /foo/ve
system: /foo/ve/ncc/X.X.X
```

- 作成したコンフィギュレーションファイルを**-cf**で指定すると、指定した環境でコンパイルされます。ここでは、コンフィギュレーションファイル名をve.confとします。

```
$ ncc -cf=ve.conf  test.c
```

- 使用するコンパイラだけを変更する。
  先の例は、コンパイル環境全体を変更しますが、使用するコンパイラだけを変更することもできます。この場合、ccomキーワードに値を指定します。

```
ccom: /foo/ve/ncc/X.X.X/libexec/ccom
```

作成したコンフィギュレーションファイルを**-cf**で指定すると、指定したC/C++コンパイラでコンパイルします。アセンブラやリンカなども同様に変更することができます。

付録 B　SX シリーズ向けコンパイラとの対応

本節ではSXシリーズ向けコンパイラとVector Engine向けコンパイラの主なコンパイラオプション、指示行環境変数の対応について説明します。

B.1　コンパイラオプション

SXシリーズ向けコンパイラのコンパイラオプションとVector Engine向けコンパイラのコンパイラオプションの対応表を示します。Vector Engineコンパイラ列の「なし」は対応するコンパイラがないことを表します。

B.1.1 全体オプション

SXシリーズ向けコンパイラ	Vector Engineコンパイラ
-Caopt	**-O4**
-Chopt	**-O3**
-Cvopt	**-O2**
-Csopt	**-O2 –mno-vector**
-Cvsafe	**-O1**
-Cssafe	**-O1 –mno-vector**
-Cnoopt	**-O0**
-Cdebug	**-O0 -g**
-S	**-S**
-NS	なし
-V (注) バージョン表示後にコンパイル処理を続けます。	**--version** (注) コンパイル処理を行わずにバージョン表示のみ行います。
-NV	なし
-c	**-c**
-Nc	なし
-cf *文字列*	**-cf=***文字列*
-clear	**-clear**

SXシリーズ向けコンパイラ	Vector Engineコンパイラ
-continst | -Ncontinst	なし
-dir { opt | noopt }	なし
-f03lib	なし
-f90lib [{ dw | dW | ew | eW }]	なし
-o *ファイル名*	**-o** *ファイル名*
-prelink	なし
-size_t32	なし
-size_t64	なし (注) 常に**-size_t64**相当で動作します。
-syntax	**-fsyntax-only**
-Nsyntax	**-fno-syntax-only**
-to *ディレクトリ名*	なし
-verbose	**-v**
-Nverbose	なし

B.1.2 最適化・ベクトル化オプション

SXシリーズ向けコンパイラ	Vector Engineコンパイラ
-Ochg	**-fassociative-math** または、 **-faggressive-associative-math** (注) **-faggressive-associative-math**は、**-fassociative-math**より激しく最適化します。
-Onochg	**-fno-associative-math**
-Odiv	**-freciprocal-math**
-Onodiv	**-fno-reciprocal-math**
-Oextendreorder	**-msched-interblock**
-Oignore_volatile	**-fignore-volatile**
-Onoignore_volatile	**-fno-ignore-volatile**
-Omove	**-fmove-loop-invariants-unsafe**

SXシリーズ向けコンパイラ	Vector Engineコンパイラ
-Onomovediv	**-fmove-loop-invariants**
-Onomove	-**fno-move-loop-invariants**
-Ooverlap	**-fnamed-alias**
-Onooverlap	**-fnamed-noalias**
-Orestrict=arg	**-fargument-noalias**
-Orestrict=this	**-fthis-pointer-noalias**
-Orestrict=type	**-fstrict-aliasing**
-Orestrict=no	**-fargument-alias** **-fthis-pointer-alias** **-fno-strict-aliasing**
-Osafe_longjmp	なし
-Onosafe_longjmp	なし
-Ounroll	**-floop-unroll**
-Ounroll=*nlevel*	**-floop-unroll** **-floop-unroll-max-times**=*n* (注) 2つのオプションを同時に指定します。
-Onounroll	**-fno-loop-unroll**
-alias { pointer | nopointer }	なし
-alias { type | notype }	なし
-alias { variable | novariable }	なし
-dir { vec | novec }	なし
-ipa	**-fipa**
-Nipa	**-fno-ipa**
-math,scalar	**-ffast-math**
-math,vector	なし
-math,nofast=*数学関数名*	なし
-math { inline | noinline }	なし
-math,round=tonearest	なし
-math,round=towardzero	なし

SXシリーズ向けコンパイラ	Vector Engineコンパイラ
-math,round=upward	なし
-math,round=downward	なし
-math,strict_prototype	なし
-math,nostrict_prototype	なし
-Nmath	なし
-pvctl,altcode	**-mvector-dependency-test** **-mvector-loop-count-test** **-mvector-shortloop-reduction** (注) 3つのオプションを同時に指定します。
-pvctl,altcode=dep	**-mvector-dependency-test**
-pvctl,altcode=nodep	**-mno-vector-dependency-test**
-pvctl,altcode=loopcnt	**-mvector-loop-count-test**
-pvctl,altcode=noloopcnt	**-mno-vector-loop-count-test**
-pvctl,altcode=shortloop	**-mvector-shortloop-reduction**
-pvctl,altcode=noshortloop	**-mno-vector-shortloop-reduction**
-pvctl,noaltcode	**-mno-vector-dependency-test** **-mno-vector-loop-count-test** **-mno-vector-shortloop-reduction** (注) 3つのオプションを同時に指定します。
-pvctl,assoc	**-fassociative-math**
-pvctl,noassoc	**-fno-associative-math**
-pvctl { assume | noassume }	なし
-pvctl,collapse	**-floop-collapse**
-pvctl,nocollapse	**-fno-loop-collapse**
-pvctl { compress | nocompress }	なし
-pvctl { conflict | noconflict }	なし
-pvctl { delinearize | nodelinearize }	なし
-pvctl,divloop	なし
-pvctl,nodivloop	**-mwork-vector-kind**=none
-pvctl,expand=*n*	**-floop-unroll-complete**=*n*

SXシリーズ向けコンパイラ	Vector Engineコンパイラ
-pvctl,noexpand	**-fno-loop-unroll-complete**
-pvctl,listvec	**-mlist-vector**
-pvctl,nolistvec	**-mno-list-vector**
-pvctl,loopchg	**-floop-interchange**
-pvctl,noloopchg	**-fno-loop-interchange**
-pvctl,loopcnt=*n*	**-floop-count**=*n*
-pvctl,loop_eq	**-freplace-loop-equation**
-pvctl,noloop_eq	**-fno-replace-loop-equation**
-pvctl,lstval	なし
-pvctl,nolstval	なし
-pvctl,matmul	**-fmatrix-multiply**
-pvctl,nomatmul	**-fno-matrix-multiply**
-pvctl { neighbors | noneighbors }	なし
-pvctl,nodep	**-fivdep**
-pvct,on_adb	なし
-pvctl,outerunroll=*n*	**-fouterloop-unroll** **-fouterloop-unroll-max-times**=*n* (注) 2つのオプションを同時に指定します。
-pvctl,outerunroll_lim=*n*	なし
-pvctl,replace_induction	なし
-pvctl,noreplace_induction	なし
-pvctl,split	**-floop-split**
-pvctl,nosplit	**-fno-loop-split**
-pvctl { vchg | novchg }	なし
-pvctl,vecthreshold=*n*	**-mvector-threshold**=*n*
-pvctl,verrchk	**-mvector-intrinsic-check**
-pvctl,noverrchk	**-mno-vector-intrinsic-check**
-pvctl { vlchk | novlchk }	なし
-pvctl,vregs=*n*	なし

SXシリーズ向けコンパイラ	Vector Engineコンパイラ
-pvctl,vwork={hybrid|stack|static}	なし
-pvctl,vworksz=*n*	なし
-struct,loop=*n*	なし
-v	**-mvector**
-Nv	**-mno-vector**
-xint	**-mno-vector-iteration**
-Nxint	**-mvector-iteration**

B.1.3 インライン展開オプション

SXシリーズ向けコンパイラ	Vector Engineコンパイラ
-dir { inline | noinline }	なし
-pi,auto	**-finline-functions**
-pi,copy_arg	**-finline-copy-arguments**
-pi,nocopy_arg	**-fno-inline-copy-arguments**
-pi,directory=*ディレクトリ名*	なし
-pi,file=*ファイル名*	なし
-pi,func_size=*n*	なし
-pi,inline	**-finline**
-pi,noinline	**-fno-inline**
-pi,max_depth=*n*	**-finline-max-depth**=*n*
-pi,max_size=*n*	**-finline-max-function-size**=*n*
-pi,search_all	なし
-pi,times=*n*	**-finline-max-times**=*n*

B.1.4 並列化オプション

SXシリーズ向けコンパイラ	Vector Engineコンパイラ
-dir { par | nopar }	なし

SXシリーズ向けコンパイラ	Vector Engineコンパイラ
-Pauto	**-mparallel**
-Pmulti	なし
-Popenmp	**-fopenmp**
-Pstack	なし
-pvctl,for[=*n***]**	なし (注) 並列化のスケジュールは、**-mschedule-static**などで制御できます。「3.3 並列化オプション」を参照してください。
-pvctl,by=*m*	なし (注) 並列化のスケジュールは、**-mschedule-static**などで制御できます。「3.3 並列化オプション」を参照してください。
-pvctl,inner	**-mparallel-innerloop**
-pvctl,noinner	**-mno-parallel-innerloop**
-pvctl,outerstrip	**-mparallel-outerloop-strip-mine**
-pvctl,noouterstrip	**-mno-parallel-outerloop-strip-mine**
-pvctl,parcase	**-mparallel-sections**
-pvctl,noparcase	**-mno-parallel-sections**
-pvctl,parthreshold=*n*	**-mparallel-threshold**=*n*
-pvctl,noparthreshold	**-mno-parallel-threshold**
-pvctl,res={ whole | parunit | no }	なし
-reserve=*n*	なし

B.1.5 コード生成オプション

SXシリーズ向けコンパイラ	Vector Engineコンパイラ
-mask { nosetall | setall | setmain }	なし
-mask { flovf | flunf | fxovf | inv | inexact | zdiv }	なし (注) 実行時の環境変数**VE_FPE_ENABLE**で制御できます。「1.8 演算例外」を参照してください。

SXシリーズ向けコンパイラ	Vector Engineコンパイラ
-stkchk | -Nstkchk	なし
-sx9 | -sxace	なし

B.1.6　言語仕様制御オプション

SXシリーズ向けコンパイラ	Vector Engineコンパイラ
-Xa	なし
-Xc	なし
-Xkr	なし
-Xp	なし
-Xs	なし
-K { align8 | noalign8 }	なし
-K { complex | nocomplex }	なし
-Kcompound_literals	なし
-Knocompound_literals	なし
-Kconst_string_literals	なし
-Knoconst_string_literals	なし
-K { designators | nodesignators }	なし
-Kexceptions	**-fexceptions** (注) 既定で有効です。
-Knoexceptions (注) 既定で有効です。	**-fno-exceptions**
-K { gcc | nogcc }	**-std=***keyword*
-Kgnu89_inline	**-fgnu89-inline**
-Knognu89_inline	なし
-Kmultibyte_chars	なし
-Knomultibyte_chars	なし
-Knew_for_init	**-ffor-scope**
-Kold_for_init	**-fno-for-scope**

SXシリーズ向けコンパイラ	Vector Engineコンパイラ
-Knonstd_gnu_keywords	なし
-Knononstd_gnu_keywords	なし
-K { nullptr | nonullptr }	なし
-Kopenmp_fatal	なし
-Kopenmp_warning	なし
-Krestrict	**-frestrict**
-Knorestrict	**-fno-restrict**
-Kstd=*keyword*	**-std**=*keyword*
-Ktrigraphs (注) 既定で有効です。	**-trigraphs**
-Knotrigraphs	なし (注) 既定では**-Knotrigraphs**相当で動作します。
-Kunsigned_char (注) 既定で有効です。	**-funsigned-char**
-Ksigned_char	**-fsigned-char** (注) 既定で有効です。
-K { using_std | nousing_std }	なし
-Kvariadic_templates	なし
-Knovariadic_templates	なし
-K { vla | novla }	なし
-T { auto | noauto }	なし
-T { none | all | used | local }	なし
-Timplicit_include	**-fimplicit-include**

B.1.7　性能測定オプション

SXシリーズ向けコンパイラ	Vector Engineコンパイラ
-acct	**-proginf**
-Nacct	**-no-proginf**

SXシリーズ向けコンパイラ	Vector Engineコンパイラ
-ftrace	**-ftrace**
-ftrace { simple | demangled }	なし
-Nftrace	**-no-ftrace**
-p	**-p**
-Np	なし

B.1.8 デバッグオプション

SXシリーズ向けコンパイラ	Vector Engineコンパイラ
-dir { debug | nodebug }	なし
-g	**-g**
-gv	なし
-Ng	**-g0**
-init,stack={ zero | nan | *0xXXXX* **}**	**-minit-stack={ zero | snan | snanf |** *0xXXXX* **}**
-traceback	**-traceback**
-traceback { simple | demangled }	なし
-Ntraceback	なし

B.1.9 プリプロセッサオプション

SXシリーズ向けコンパイラ	Vector Engineコンパイラ
-D*name*[=*def*]	**-D*name*[=*def*]**
-E	**-E**
-EP	なし
-H	**-H**
-Iディレクトリ名	***-Iディレクトリ名***
-K gcc_predefines	なし (注) マクロは既定で定義されます。
-K nogcc_predefines	なし
-Kkeep_comments	**-C**

SXシリーズ向けコンパイラ	Vector Engineコンパイラ
-Knokeep_comments	なし
-Kkeep_line_dirs	なし
-Knokeep_line_dirs	なし
-Knew_preprocessing	なし
-Kold_preprocessing	**-traditional-cpp** (注) –Eと同時に指定する必要があります。
-Kvariadic_macros	なし
-Knovariadic_macros	なし
-M	**-M**
-U*name*	**-U*name***
-dD	**-dD**
-dI	**-dI**
-dM	**-dM**
-dN	**-dN**

B.1.10リスト出力オプション

SXシリーズ向けコンパイラ	Vector Engineコンパイラ
-Rappend	**-report-append-mode**
-Rnoappend	なし
-Rdiaglist	**-report-diagnostics**
-Rnodiaglist	なし
-Rfile={ *ファイル名* **| stdout }**	**-report-file={** *ファイル名* **| stdout }**
-Rfmtlist	**-report-format**
-Rnofmtlist	なし
-Robjlist	**-assembly-list**
-Rnoobjlist	なし
-R { summary | nosummary }	なし
-Rsystem_header	**-fdiag-system-header**

SXシリーズ向けコンパイラ	Vector Engineコンパイラ
-R { transform | notransform }	なし

B.1.11メッセージオプション

SXシリーズ向けコンパイラ	Vector Engineコンパイラ
-O { fullmsg | infomsg | nomsg }	なし
-pi { fullmsg | infomsg | nomsg }	**-fdiag-inline={ 2 | 1 | 0 }**
-pvctl { fullmsg | infomsg | nomsg }	**-fdiag-parallel={ 2 | 1 | 0 }** **-fdiag-vector={ 2 | 1 | 0 }**
-wall	**-Wall**
-wno_unset_use	なし
-wnone	**-w**
-wfatal=*n*	なし
-woff=*n*	なし
-wlongjmp	なし

B.1.12アセンブラオプション

SXシリーズ向けコンパイラ	Vector Engineコンパイラ
-Wa,"*オプション列***"**	**-Wa,"***オプション列***"**

B.1.13リンカオプション

SXシリーズ向けコンパイラ	Vector Engineコンパイラ
-G	なし
-L*ディレクトリ名*	**-L***ディレクトリ名*
-l*ライブラリ名*	**-l***ライブラリ名*
-Wl,"*オプション列***"**	**-Wl,"***オプション列***"**

B.1.14ディレクトリオプション

SXシリーズ向けコンパイラ	Vector Engineコンパイラ
-YI, *ディレクトリ名*	なし
-YL, *ディレクトリ名*	なし
-YS, *ディレクトリ名*	なし
-Ya, *ディレクトリ名*	なし
-Yc, *ディレクトリ名*	なし
-Yl, *ディレクトリ名*	なし
-Ys, *ディレクトリ名*	なし
-Yt, *ディレクトリ名*	なし

B.2　コンパイラ指示行

SXシリーズ向けコンパイラとVector Engine向けのコンパイラの指示行の対応表は「C.3 コンパイラ指示行」を参照してください。SXシリーズ向けコンパイラの指示行をVector Engine向けのコンパイラに変換するツールを用意しています。詳細は「付録 C コンパイラ指示行変換ツール」を参照してください。

B.3　環境変数

SXシリーズ向けコンパイラの主要な実行時に参照される環境変数とほぼ同等の機能をもつVector Engine向けコンパイラの環境変数を以下に示します。

SXシリーズ向けコンパイラ	Vector Engineコンパイラ
C_PROGINF	**VE_PROGINF**
C_TRACEBACK	**VE_TRACEBACK**

B.4　処理系定義

SXシリーズ向けコンパイラの処理系定義とVector Engine向けコンパイラの処理系定義の対応表を以下に示します

B.4.1 型

型名	SXシリーズ向けコンパイラ		Vector Engineコンパイラ	
	サイズ	アライメント	サイズ	アライメント
_Bool	4	4	1	1
bool	4	4	1	1
char **signed char** **unsigned char**	1	1	1	1
short **short int** **unsigned short** **unsigined short int**	2	2	2	2
int **unsigned int**	4	4	4	4
long **long int** **unsigned long** **unsigned long int**	8	8	8	8
long long **long long int** **unsigned long long** **unsigned long long int**	8	8	8	8
float	4	4	4	4
double	8	8	8	8
long double	16	16	16	16

型名	SXシリーズ向け コンパイラ		Vector Engine コンパイラ	
	サイズ	アライメント	サイズ	アライメント
float _Complex	8	4	8	4
double _Complex	16	8	16	8
long double _Complex	32	16	32	16
float _Imaginary	4	4	なし	なし
double _Imaginary	8	8	なし	なし
long double _Imaginary	16	16	なし	なし
ポインタ	8	8	8	8
enum	4	4	4	4
配列型	(*1)	(*2)	(*1)	(*3)
構造体型 共用体型 クラス型	(*4)	(*5)	(*4)	(*5)
ビットフィールド	4 (*6)	4 (*6)	(*7)	(*7)

(*1) 配列の要素サイズ×配列の要素数。

(*2) **char** 型の配列の時 16 バイト、**char** 型の配列以外は配列要素のアライメントと同じ。

(*3) 配列要素のアライメントと同じ。

(*4) データメンバのサイズの合計値にデータメンバのアライメントのための領域のサイズを加算した値。ただし、共用体のとき、領域の重複部分を除く。

(*5) データメンバのアライメントサイズの最大値か、4 バイトの大きい方の値。

(*6) **int** 型に相当する。

(*7) ビットフィールドのサイズやアライメントは構造型や共用型のルールに従う。

B.4.2　定義済みマクロ

以下のSXシリーズ向けコンパイラの定義済みマクロは、Vector Engine向けコンパイラでは定義されません。

名前
__BUILTIN_ABS
_C99
_C99_COMPLEX
_C99LIB
_EXCEPTION_ENABLE
_FLOAT0
_LONG64
_RESTRICT
_SIGNED_CHAR
_SIZE_T64
__STDC_NO_THREADS__
SX
_SX
__SXCXX_EXTENSIONS
__SXCXX_REVISION
_VECLIB

付録 C　コンパイラ指示行変換ツール

本節ではsxf90/sxf03/sxcc/sxc++のコンパイラ指示行をnfort/ncc/nc++のコンパイラ指示行に変換するツールについて説明します。

C.1　ncdirconv

コマンド名

ncdirconv

書式

ncdirconv [*オプション...*] [*ファイル* | *ディレクトリ*]...

説明

ソースファイルに含まれるsxf90/sxf03/sxcc/sxc++のコンパイラ指示行をnfort/ncc/nc++のコンパイラ指示行に変換します。ソースファイルのかわりにディレクトリを指定することで、そのディレクトリの次の拡張子を持つファイルをまとめて変換できます。

```
.c  .i  .h  .C  .cc  .cpp  .cp  .cxx  .c++  .ii  .H  .hh  .hpp
.hp  .hxx  .h++  .tcc .F  .FOR  .FTN  .FPP  .F90  .F95  .F03  .f
.for  .ftn  .fpp  .f90  .f95  .f03  .i90
```

変換前のファイルは「*ファイル名*.bak」として保存されます。

sxf90/sxf03/sxcc/sxc++の指示行はオプションの指定により、変換後も残したり、削除したりできます。

オプション

オプション名	説明
-a,　--append	sxf90/sxf03/sxcc/sxc++のコンパイラ指示行を削除せずに、nfort/ncc/nc++のコンパイラ指示行を追加します。
-d,　--delete	sxf90/sxf03/sxcc/sxc++のコンパイラ指示行に対応するnfort/ncc/nc++のコンパイラ指示行が無い場合、sxf90/sxf03/sxcc/sxc++のコンパイラ指示行を削除します。
-f,　--force	入力ファイルの拡張子をチェックしません。
-h,　--help	本コマンドの利用方法を出力して終了します。
-o *ファイル名*, **--output** *ファイル名*	出力ファイル名を指定します。入力ファイルを複数指定したとき、または、ディレクトリを指定したとき、この指定は無視されます。
-p,　--preserve	sxf90/sxf03/sxcc/sxc++のコンパイラ指示行に対応するnfort/ncc/nc++のコンパイラ指示行が無い場合、sxf90/sxf03/sxcc/sxc++のコンパイラ指示行を削除しません。

オプション名	説明
	(既定の動作)
-q,　-quiet	コンパイラ指示行の変換に関するメッセージを出力しません。
-r,　--recursive	ディレクトリと同時に指定したとき、サブディレクトリを再帰的に走査します。ディレクトリのシンボリックリンクは無視します。
-v,　--version	バージョン情報を出力して終了します。

出力メッセージ

指示行を変換した場合やnfort/ncc/nc++で指示行が未サポートの場合、標準エラー出力に次の形式のメッセージを出力します。

ファイル名: line *行番号*: メッセージ

ファイル名　：入力ファイル名

行番号　　：変換前のファイルの行番号

メッセージ　：

- converted "*SX用指示行*" to "*VE用指示行*" (Converted|Substitute)

コンパイラ指示行を変換したことを表します。SXとVEのコンパイラ指示行が同等の機能を持つ場合、"Converted"が出力されます。完全に同じではないが、ほぼ同等の機能を持つ場合は"Substitute"が出力されます。

- "*SX用指示行*" is not supported [(Remained| Removed/Obsolescent)]

sxf90/sxf03/sxcc/sxc++のコンパイラ指示行がVEではサポートしていないことを表します。将来実装予定のコンパイラ指示行は"Remained"が出力されます。サポート予定のないコンパイラ指示行は、"Removed/Obsolescent"が出力されます。

返却値

すべての変換に成功したときは0を返し、エラーが発生したときは0以外を返します。

注意事項

本コマンドは作業用の一時ファイルを/tmpに作成します。この一時ファイルは、コマンド終了時に自動的に削除されます。一時ファイルを作成するディレクトリは、環境変数**TMPDIR**で変更できます。

C.2　使用例

指示行変換ツールの使用例を示します。ここでは、環境変数**PATH**に/opt/nec/ve/binが追加されていることを前提とします。

ファイルを指定した場合

ファイルに含まれるsxf90/sxf03/sxcc/sxc++のコンパイラ指示行をnfort/ncc/nc++のコンパイラ指示行に変換します。

```
$ cat sample.c
int func(int max)
{
    int i;
    int sum = 0;

#pragma cdir novector
    for (i=0; i<max; i++) {
        sum += i;
    }
    return sum;
}
```

```
$ ncdirconv sample.c
sample.c: line   6: converted 'novector' to 'novector' (Converted)
```

```
$ cat sample.c
int func(int max)
{
    int i;
    int sum = 0;

#pragma _NEC novector
    for (i=0; i<max; i++) {
        sum += i;
    }
    return sum;
}
```

ディレクトリを指定した場合

以下のようなディレクトリ構成とします。

```
dir/
    ├ Makefile
    ├ sample1.c
    ├ sample2.c
    └ subdir/
          ├ Makefile
          └ sample3.c

$ ncdirconv dir
dir/sample1.c: line   5: converted 'loopcnt=5' to 'loop_count(5)' (Converted)
dir/sample2.c: line  16: converted 'nodep' to 'ivdep' (Substitute)
```

この場合、dir直下のsample1.cとsample2.cが変換対象となります。Makefileは拡張子が無いため変換の対象外です。また、ディレクトリ直下のファイルが対象となるため、サブディレクトリsubdir配下のファイルも変換対象外です。

```
$ ncdirconv -r dir
dir/sample2.c: line   5: converted 'nodep' to 'ivdep' (Substitute)
dir/sample1.c: line  16: converted 'loopcnt=5' to 'loop_count(5)' (Converted)
dir/subdir/sample3.c: line  12: converted 'loopcnt=5' to 'loop_count(5)' (Conv
erted)
```

サブディレクトリのファイルも含めて変換したい場合は、**-r**を指定します。**-r**を指定するとディレクトリを再帰的に処理するようになり、サブディレクトリにあるファイルも変換されるようになります。

C.3　コンパイラ指示行

sxf90/sxf03/sxcc/sxc++のコンパイラ指示行と変換後のコンパイラ指示行を以下に示します。「変換後」列の(Remained)は将来実装予定のコンパイラ指示行、(Removed/Obsolescent)はサポートしないコンパイラ指示行を表します。

SXシリーズ向けコンパイラ	変換後
alias	(Removed/Obsolescent)
alloc_on_vreg(*identifier*, *n***)**	**vreg(***identifier***)**
altcode	**dependency_test** **loop_count_test** **shortloop_reduction**
altcode=dep	**dependency_test**

SXシリーズ向けコンパイラ	変換後
altcode=loopcnt	**loop_count_test**
altcode=nodep	**nodependency_test**
altcode=noshort	**noshortloop_reduction**
altcode=short	**shortloop_reduction**
noaltcode	**nodependency_test** **noloop_count_test** **noshort_loop_reduction**
assoc	**assoc**
noassoc	**noassoc**
assume	**assume**
noassume	**noassume**
atomic	**atomic**
cncall	**cncall**
collapse	**collapse**
compress	(Removed/Obsolescent)
nocompress	(Removed/Obsolescent)
concur	**concurrent**
concur(by=*m***)**	**concurrent schedule(***dynamic***, *m*)**
concur(for=*n***)**	**concurrent**
noconcur	**noconcurrent**
data_prefetch	(Removed/Obsolescent)
delinearize	(Removed/Obsolescent)
nodelinearize	(Removed/Obsolescent)
divloop	**vwork**
nodivloop	**novwork**
expand	**unroll_complete**
expand=*n*	(Removed/Obsolescent)
noexpand	**nounroll**
gthreorder	**gather_reorder**
nogthreorder	(Removed/Obsolescent)
iexpand(*function***)**	**inline**
noiexpand(*function***)**	**noinline**
inline(*function***)**	**inline**

SXシリーズ向けコンパイラ	変換後
inline(*function***) complete**	**inline_complete**
noinline(*function***)**	**noinline**
inner	**inner**
noinner	**noinner**
listvec	**list_vector**
nolistvec	**nolist_vector**
loop_eq	(Removed/Obsolescent)
noloop_eq	(Removed/Obsolescent)
loopchg	**interchange**
noloopchg	**nointerchange**
loopcnt=n	**loop_count(***n***)**
lstval	**lstval**
nolstval	**nolstval**
move	**move_unsafe**
nomove	**nomove**
nomovediv	**move**
neighbors	(Removed/Obsolescent)
noneighbors	(Removed/Obsolescent)
nexpand	**inline_complete**
noconflict(*identifier***)**	(Removed/Obsolescent)
nodep	**ivdep**
on_adb(*identifier***)**	(Removed/Obsolescent)
outerunroll=*n*	**outerloop_unroll(***n***)**
noouterunroll	**noouterloop_unroll**
overlap	(Removed/Obsolescent)
nooverlap	(Removed/Obsolescent)
parallel for	**parallel for**
parallel for private(*identifier***)**	**Parallel for private(***identifier***)**
parallel sections	(Removed/Obsolescent)
section	(Removed/Obsolescent)
select(*keyword***)**	(Remained)
shape	(Removed/Obsolescent)
shortloop	**shortloop**

SXシリーズ向けコンパイラ	変換後
skip	(Removed/Obsolescent)
sparse	**sparse**
nosparse	**nosparse**
split	(Remained)
nosplit	(Remained)
sync	(Remained)
nosync	**nosync**
threshold	(Removed/Obsolescent)
nothreshold	(Removed/Obsolescent)
traceback	(Remained)
unroll=n	**unroll(***n***)**
nounroll	**nounroll**
Vecthreshold	**vector_threshold(***n***)**
Vector	**vector**
Novector	**novector**
Verrchk	**verror_check**
Noverrchk	**noverror_check**
Vlchk	(Removed/Obsolescent)
Novlchk	(Removed/Obsolescent)
Vob	**vob**
Novob	**novob**
vovertake(*identifier***)**	**vovertake**
Novovertake	**novovertake**
Vprefetch	(Remained)
Novprefetch	(Removed/Obsolescent)
vreg(*identifier***)**	**vreg(***identifier***)**
vwork=*keyword*	(Removed/Obsolescent)
vworksz=*n*	(Removed/Obsolescent)

C.4　留意事項

- **-a**や**-p**を指定するとsxf90/sxf03/sxcc/sxc++のコンパイラ指示行が残るため、コンパイル時に警告が出力されます。

```
$ ncc -c sample.c
"sample.c", line 6: warning: unrecognized #pragma
  #pragma cdir novector
           ^

ncc: vec( 103): sample.c, line 8: Unvectorized loop.
```

- ツール実行時に変換前のファイルが保存されます。「*ファイル名*.bak」が存在する場合は、".bak"を".bak2"にリネームし、新しいファイルを".bak"として保存します。*ファイル名*は最大5つまで保存されます。必要に応じてファイルを削除してください。
- 本ツールでは入力ファイルの書式をチェックしません。sxf90/sxf03/sxcc/sxc++のコンパイラ指示行の書式が誤っているときは正しく変換できない場合があります。
- 入力ファイルがシンボリックリンクファイルの場合は、シンボリックリンク先のファイルが更新されます。保存用の*ファイル名*.bakは通常のファイルとして作成されます。

付録 D　追加・変更点詳細

前版(第26版、2022年3月発行)からの更新内容は以下のとおりです。

- 第2章の環境変数の説明に次を追加しました。
  - **OMP_TOOL** / **VE_OMP_TOOL**
  - **OMP_TOOL_LIBRARIES** / **VE_OMP_TOOL_LIBRARIES**
- 第3章の次のコンパイラオプションの説明を追加、変更しました。
  - **-fopenmp-tools**を追加
  - **-fnamed-alias**、および、**-fnamed-noalias**の説明を変更
  - **-ftemplate-depth**の既定値を変更し、指定できる値の説明を追加
- 7.2.2の利用できるOpenMP 5.0機能にOMPT interfaceを追加しました。

索　引

G

H

I

L

M

N

O

P

R

S

T

U

V

W

X

Z

あ

い

え

お

か

き

く

こ

さ

し

す

せ

そ

た

て

と

は

ひ

ふ

へ

ほ

ま

む

め

る

SX-Aurora TSUBASA システムソフトウェア

C/C++コンパイラ　ユーザーズガイド

2022年6月 27版

日本電気株式会社

東京都港区芝五丁目７番１号

TEL(03)3454-1111（大代表）