OpenMP API ユーザーズガイド

Sun™ ONE Studio 8

Sun Microsystems, Inc.
4150 Network Circle
Santa Clara, CA 95054 U.S.A.
650-960-1300

Part No. 817-2923-10
2003 年 5 月 , Revision A

原典： OpenMP API User's Guide
Part No: 817-0933-10
Revision A

目次

表目次

はじめに

『OpenMP API ユーザーズガイド』では、多重処理アプリケーションを構築するための OpenMP Fortran 95、C、C++ アプリケーションプログラムインタフェース (API) について解説します。Sun™ ONE Studio コンパイラは OpenMP API をサポートします。

このマニュアルは、Fortran、C、C++ 言語、および OpenMP 並列プログラミングモデルの知識を有する科学者、エンジニア、プログラマを対象としています。さらに、Solaris™ オペレーティング環境または UNIX® について一般的な知識を有することを前提とします。

書体と記号について

次の表と記述は、このマニュアルで使用している書体と記号について説明しています。

書体または記号	意味	例
AaBbCc123	コマンド名、ファイル名、ディレクトリ名、画面上のコンピュータ出力、コーディング例。	.login ファイルを編集します。 ls -a を使用してすべてのファイルを表示します。 machine_name% You have mail.
AaBbCc123	ユーザーが入力する文字を、画面上のコンピュータ出力と区別して表わします。	machine_name% **su** Password:
AaBbCc123 または**ゴシック**	コマンド行の可変部分。実際の名前または実際の値と置き換えてください。	rm filename と入力します。 rm **ファイル名** と入力します。
『 』	参照する書名を示します。	『SPARCstorage Array ユーザーマニュアル』
「 」	参照する章、節、または、強調する語を示します。	第 6 章「データの管理」を参照してください。 この操作ができるのは、「スーパーユーザー」だけです。
\	枠で囲まれたコード例で、テキストがページ行幅を超える場合、バックスラッシュは、継続を示します。	machinename% grep '^#define \ XV_VERSION_STRING'
➤	階層メニューのサブメニューを選択することを示します。	作成: 「返信」➤「送信者へ」

コードの記号	意味	記法	コード例
[ ]	角括弧にはオプションの引数が含まれます。	O[n]	-O4, -O
{ }	中括弧には、必須オプションの選択肢が含まれます。	d{y|n}	-dy
|	「パイプ」または「バー」と呼ばれる記号は、その中から 1 つだけを選択可能な複数の引数を区切ります。	B{dynamic|static}	-Bstatic
:	コロンは、コンマ同様に複数の引数を区切るために使用されることがあります。	Rdir[:dir]	-R/local/libs:/U/a
...	省略記号は、連続するものの一部が省略されていることを示します。	-xinline=f1[,...fn]	-xinline=alpha,dos

シェルプロンプトについて

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

コンパイラコレクションのツールとマニュアルページへのアクセス

コンパイラコレクションのコンポーネントとマニュアルページは、標準の /usr/bin/ と /usr/share/man の各ディレクトリにインストールされません。コンパイラとツールにアクセスするには、PATH 環境変数にコンパイラコレクションのコンポーネントディレクトリを必要とします。マニュアルページにアクセスするには、PATH 環境変数にコンパイラコレクションのマニュアルページディレクトリが必要です。

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

注 – この節に記載されている情報は Sun ONE Studio コンパイラコレクションコンポーネントが /opt ディレクトリにインストールされていることを想定しています。ソフトウェアが /opt 以外のディレクトリにインストールされている場合は、システム管理者に実際のパスをお尋ねください。

コンパイラとツールへのアクセス方法

PATH 環境変数を変更して、コンパイラとツールにアクセスできるようにする必要があるかどうか判断するには以下を実行します。

▼ PATH 環境変数を設定する必要があるかどうか判断するには

1. **次のように入力して、PATH 変数の現在値を表示します。**

```
% echo $PATH
```

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

パスがある場合は、PATH 変数はコンパイラとツールにアクセスできるように設定されています。パスがない場合は、次の指示に従って、PATH 環境変数を設定してください。

▼ PATH 環境変数を設定してコンパイラとツールにアクセスする

1. C シェルを使用している場合は、ホームの .cshrc ファイルを編集します。Bourne シェルまたは Korn シェルを使用している場合は、ホームの .profile ファイルを編集します。
2. 次のパスを PATH 環境変数に追加します。

   /opt/SUNWspro/bin

マニュアルページへのアクセス方法

マニュアルページにアクセスするために MANPATH 変数を変更する必要があるかどうかを判断するには以下を実行します。

▼ MANPATH 環境変数を設定する必要があるかどうか判断するには

1. 次のように入力して、dbx マニュアルページを表示します。

```
% man dbx
```

2. 出力された場合、内容を確認します。

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

▼ MANPATH 変数を設定してマニュアルページにアクセスする

1. C シェルを使用している場合は、ホームの .cshrc ファイルを編集します。Bourne シェルまたは Korn シェルを使用している場合は、ホームの .profile ファイルを編集します。
2. 次のパスを PATH 環境変数に追加します。

   /opt/SUNWspro/man

コンパイラコレクションのマニュアルへのアクセス

マニュアルには、以下からアクセスできます。

- 製品マニュアルは、ご使用のローカルシステムまたはネットワークの製品にインストールされているマニュアルの索引から入手できます。
  `/opt/SUNWspro/docs/ja/index.html`

製品ソフトウェアが `/opt` 以外のディレクトリにインストールされている場合は、システム管理者に実際のパスをお尋ねください。

- マニュアルは、`docs.sun.com` の Web サイトで入手できます。以下に示すマニュアルは、インストールされている製品のマニュアルの索引から入手できます (`docs.sun.com` Web サイトでは入手できません)。
  - 『Standard C++ Library Class Reference』
  - 『標準 C++ ライブラリ・ユーザーズガイド』
  - 『Tools.h++ クラスライブラリ・リファレンスマニュアル』
  - 『Tools.h++ ユーザーズガイド』
- リリースノートは、docs.sun.com で入手できます。

インターネットの `docs.sun.com` Web サイト (`http://docs.sun.com`) から、サンのマニュアルを参照したり、印刷したり、購入することができます。マニュアルが見つからない場合はローカルシステムまたはネットワークの製品とともにインストールされているマニュアルの索引を参照してください。

注 - Sun では、本マニュアルに掲載した第三者の Web サイトのご利用に関しましては責任はなく、保証するものでもありません。また、これらのサイトあるいはリソースに関する、あるいはこれらのサイト、リソースから利用可能であるコンテンツ、広告、製品、あるいは資料に関して一切の責任を負いません。Sun は、これらのサイトあるいはリソースに関する、あるいはこれらのサイトから利用可能であるコンテンツ、製品、サービスのご利用あるいは信頼によって、あるいはそれに関連して発生するいかなる損害、損失、申し立てに対する一切の責任を負いません。

アクセシブルな製品マニュアル

マニュアルは、技術的な補足をすることで、ご不自由なユーザーの方々にとって読みやすい形式のマニュアルを提供しております。アクセシブルなマニュアルは以下の表に示す場所から参照することができます。製品ソフトウェアが /opt 以外のディレクトリにインストールされている場合は、システム管理者に実際のパスをお尋ねください。

マニュアルの種類	アクセシブルな形式と格納場所
マニュアル (サードパーティ製マニュアルは除く)	形式 : HTML (日本語版は PDF のみ) 場所 : http://docs.sun.com
サードパーティ製マニュアル: 『Standard C++ Library Class Reference』 『標準 C++ ライブラリ・ユーザーズガイド』 『Tools.h++ クラスライブラリ・リファレンスマニュアル』 『Tools.h++ ユーザーズガイド』	形式 : HTML 場所 : file:/opt/SUNWspro/docs/ja/index.html のマニュアル索引
Readme および マニュアルページ	形式 : HTML 場所 : file:/opt/SUNWspro/docs/ja/index.html のマニュアル索引
リリースノート	製品 CD 内の HTML ファイル

関連するコンパイラコレクションマニュアル

以下の表は、file:/opt/SUNWspro/docs/ja/index.html および http://docs.sun.com から参照できるマニュアルの一覧です。製品ソフトウェアが /opt 以外のディレクトリにインストールされている場合は、システム管理者に実際のパスをお尋ねください。

マニュアルタイトル	内容の説明
Fortran ユーザーズガイド	f95 コンパイラのコンパイル時環境とコマンド行オプションについて説明しています。従来の f77 のプログラムを f95 に移行するためのガイドラインも記載されています。
Fortran ライブラリ・リファレンス	Fortran ライブラリと組み込みルーチンについて詳しく説明しています。
OpenMP API ユーザーズガイド	OpenMP 多重処理 API の概要とその Forte Developer 実装の詳細について説明します。
数値計算ガイド	浮動小数点演算における数値の精度に関する問題について説明しています。

関連する Solaris マニュアル

次の表では、docs.sun.com の Web サイトで参照できる関連マニュアルについて説明します。

マニュアルコレクション	マニュアルタイトル	内容の説明
Solaris 8 Reference Manual Collection	マニュアルページの節を参照。	Solaris のオペレーティング環境に関する情報を提供しています。
Solaris 8 Software Developer Collection	リンカーとライブラリ	Solaris のリンクエディタと実行時リンカーの操作について説明しています。
Solaris 8 Software Developer Collection	マルチスレッドのプログラミング	POSIX と Solaris スレッド API、同期オブジェクトのプログラミング、マルチスレッド化したプログラムのコンパイル、およびマルチスレッド化したプログラムのツール検索について説明します。

開発者向けのリソース

`http://www.sun.com/developers/studio` にアクセスし、Compiler Collection というリンクをクリックして、以下のようなリソースを利用できます。リソースは頻繁に更新されます。

- プログラミング技術と最適な演習に関する技術文書
- プログラミングに関する簡単なヒントを集めた知識ベース
- Compiler Collection のコンポーネントのマニュアル、ソフトウェアと共にインストールされるマニュアルの訂正
- サポートレベルに関する情報
- ユーザーフォーラム

- ダウンロード可能なサンプルコード
- 新しい技術の紹介
- `http://www.sun.co.jp/developers/` でも開発者向けのリソースが提供されています。

技術サポートへの問い合わせ

製品についての技術的なご質問がございましたら、以下のサイトからお問い合わせください (このマニュアルで回答されていないものに限ります)。

http://sun.co.jp/service/contacting

第1章

OpenMP API の概要

OpenMP™ Application Program Interface (API) は、共有メモリー型マルチプロセッサアーキテクチャ用の移植性のある並列プログラミングモデルで、多数のコンピュータベンダーと共同で開発されました。仕様書は OpenMP Architecture Review Board で作成され、発行されています。説明およびその他のリソースを含む OpenMP 開発者コミュニティの詳細については、`http://www.openmp.org` の Web サイトを参照してください。

OpenMP API は、SPARC® 、UltraSPARC® プラットフォームで動作するすべての Sun ONE Studio コンパイラの推奨並列プログラミングモデルです。従来の Fortran および C の並列化指令を OpenMP 指令に変換する方法については、第 4 章を参照してください。

この章では、OpenMP Version 2.0 API を構成する指令、実行時ライブラリルーチン、環境変数について説明します。この API は、Sun ONE Studio の Fortran 95、C、C++ のコンパイラで実装されています。

OpenMP 仕様の参照先

この章では、簡潔にするため、OpenMP の概要だけを説明しており、詳細情報の多くを省略しています。詳細については、『OpenMP 仕様書』を参照してください。

Fortran、C、C++ の OpenMP 2.0 仕様 については、OpenMP の公式 Web サイト、`http://www.openmp.org/` を参照してください。

このマニュアルで使用している特別な表記

後述の表および例では、Fortran の指令およびソースコードは大文字で表記されていますが、実際には大文字と小文字は区別されません。

structured-block は、ブロックの内外への転送を行わない Fortran 文または C/C++ 文のブロックを指します。

[...] (角括弧) 内の要素は省略可能です。

このマニュアルでは、「Fortran」は Fortran 95 言語およびそのコンパイラである **`f95`** を示します。

「指令」および「プラグマ」は、このマニュアルでは同義で使用されています。

指令の書式

指令行では指令名を 1 つだけ指定することができ、指定した指令は後続のプログラム文に適用されます。

Fortran:

Fortran の固定書式では 3 つ、自由書式では 1 つだけ、「標識」を使用することができます。後述の Fortran の表および例では、自由書式が使用されています。

C/C++:

C および C++ は、**#pragma omp** で始まる標準のプリプロセッサ指令を使用します。

OpenMP 2.0 Fortran

固定書式

C$OMP directive-name optional_clauses...
!$OMP directive-name optional_clauses...
*$OMP directive-name optional_clauses...

標識は、カラム 1 から開始する必要があります。継続行については、カラム 6 で空白およびゼロ以外の文字が指定されている必要があります。
コメントは、指令行のカラム 6 以降に、感嘆符 (**!**) に続けて入力することができます。行中の **!** 以降の部分は無視されます。

自由書式

!$OMP directive-name optional_clauses...

任意の場所で指定でき、その前に空白があってもかまいません。継続行は、行の最後にあるアンパサンド (**&**) により識別されます。
コメントは、指令行で感嘆符 (!) に続けて入力することができます。以降の部分は無視されます。

OpenMP 2.0 C/C++

#pragma omp directive-name optional_clauses...
各プラグマは、改行文字で終了する必要があります。また、標準の C および C++ のコンパイラプラグマの表記に従う必要があります。
プラグマでは、大文字と小文字が区別されます。句の表記順序には意味はありません。**#** の前後および単語の間には空白文字を入力することができます。
指令は、後続の文 (構造ブロックでなければならない) に適用されます。

条件付きコンパイル

OpenMP API では、条件付きコンパイルに使用するプリプロセッサ記号 **_OPENMP** が定義されています。また、OpenMP Fortran API では、条件付きコンパイル標識を使用できます。

OpenMP 2.0 Fortran

固定書式

```
!$      fortran_95_statement
C$      fortran_95_statement
*$      fortran_95_statement
c$      fortran_95_statement
```

標識は、空白文字を入れずにカラム 1 から開始する必要があります。OpenMP コンパイルが有効な場合は、標識は 2 つの空白文字に置き換えられます。行のそれ以外の部分は、標準の Fortran の固定書式の表記に従って入力する必要があります。例:

```
C23456789
!$ 10 iam = OMP_GET_THREAD_NUM() +
!$   1       index
```

自由書式

```
!$ fortran_95_statement
```

この標識は、任意のカラムで指定できます。標識の前には空白文字だけを入力し、1 語で表記する必要があります。行のそれ以外の部分には、Fortran の自由書式の表記を使用します。例:

```
C23456789
 !$ iam = OMP_GET_THREAD_NUM() +     &
 !$&         index
```

Fortran プリプロセッサ:

OpenMP を有効にしたコンパイルでは、プリプロセッサ記号 **_OPENMP** が定義されます。

```
#ifdef _OPENMP
     iam = OMP_GET_THREAD_NUM()+index
#endif
```

OpenMP 2.0 C/C++

C/C++ プリプロセッサ:
OpenMP を有効にしたコンパイルでは、マクロ **_OPENMP** が定義されます。

```
#ifdef _OPENMP
     iam = omp_get_thread_num() + index;
#endif
```

PARALLEL - 並列領域構造

PARALLEL 指令は、並列領域を定義します。並列領域は、複数のスレッドで並列で実行されるプログラムの領域です。

OpenMP 2.0 Fortran

```
!$OMP PARALLEL [clause[[,]clause]...]
  structured-block
!$OMP END PARALLEL
```

OpenMP 2.0 C/C++

```
#pragma omp parallel [clause[[,]clause]...]
   structured-block
```

特別な条件や制限が多数存在します。詳細については、OpenMP 仕様書を参照してください。

表 1-1 は、この構造とともに記述できる句を示しています。

ワークシェアリング構造

ワークシェアリング構造は、構造内のコード領域を、検出したスレッドのチームのメンバー間で分割して実行します。ワークシェアリング構造を並列で実行するには、構造を並列領域内に記述する必要があります。

これらの指令およびそれらが適用されるコードについては特別な条件と制限が多数あります。詳細については、『OpenMP 仕様書』を参照してください。

DO、for 構造

DO または **for** ループの反復が並列に実行されるよう指定します。

OpenMP 2.0 Fortran

```
!$OMP  DO [clause[[,] clause]...]
  do_loop
[!$OMP  END  DO  [NOWAIT]]
```

DO 指令は、直後の **DO** ループの反復を並列で実行するように指定します。この指令は、並列領域内に記述する必要があります。

OpenMP 2.0 C/C++

```
#pragma  omp  for  [clause[[,]clause]...]
  for-loop
```

for プラグマは、直後のfor-**ループ**の反復を並列で実行するように指定します。このプラグマは、並列領域内に記述する必要があります。**for** プラグマでは、対応する **for** ループの構造を次のような正規の形式で記述する必要があります。

```
for （initexpr； var logicop b； increxpr）
```

ここで、各要素の意味は以下のとおりです。

- initexpr　には、以下のいずれかを指定します。

  var = lb　　integer_type var = lb

- increxpr には、以下のいずれかの形式の式を指定します。

  ++var　var++　　--var　var--　　var += incr　var -= incr　var = var + incr　　var = incr + var　　var = var - incr

- var　は符号付き整数型の変数で、**for** の範囲で暗黙に非公開となります。var は、**for** 文の本体内では変更することはできません。値は、**lastprivate** を指定した場合を除き、ループ以降は不定になります。
- logicop　には、以下のいずれかの論理演算子を指定します。

  <　<=　>　>=

- lb、b、incr　は、ループの整数の不変式です。

< または <= と > または >= を **for** 文の logicop として使用する場合、さらに制限があります。詳細については、OpenMP C/C++ の仕様を参照してください。

SECTIONS 構造

SECTIONS 構造には、チーム内のスレッド間で分割する、非反復コードブロックが含まれます。各ブロックは、チーム内のスレッドによって 1 回実行されます。

各セクションの前に **SECTION** 指令を記述します。この指令は、最初のセクションについては省略可能です。

OpenMP 2.0 Fortran

```
!$OMP SECTIONS [clause[[,] clause]...]
[!$OMP SECTION]
  structured-block
[!$OMP SECTION
 structured-block ]
...
!$OMP END SECTIONS [NOWAIT]
```

OpenMP 2.0 C/C++

```
#pragma omp sections [clause[[,]clause]...]
  {
   [#pragma omp section ]
           structured-block
   [#pragma omp section
           structured-block]
    ...
  }
```

表 1-1 は、この構造とともに記述できる句を示しています。

SINGLE 構造

SINGLE で囲まれた構造ブロックは、チーム内の 1 つのスレッドだけによって実行されます。チーム内で **SINGLE** ブロックを実行していないスレッドは、**NOWAIT** を指定した場合を除き、ブロックの最後で待機します。

OpenMP 2.0 Fortran

```
!$OMP SINGLE [clause[[,] clause]...] structured-block
!$OMP END SINGLE [end-modifier]
```

OpenMP 2.0 C/C++

```
#pragma omp single [clause[[,] clause]...]
   structured-block
```

表 1-1 は、この構造とともに記述できる句を示しています。

Fortran の WORKSHARE 構造

WORKSHARE 構造では、コードブロックを実行する処理が複数の作業単位に分割されます。各単位が 1 回ずつだけ実行されるように、チームのスレッド間で作業が分割されます。

OpenMP 2.0 Fortran

```
!$OMP WORKSHARE
  structured-block
!$OMP END WORKSHARE [NOWAIT]
```

C/C++ で Fortran の WORKSHARE に相当する構造はありません。

並列/ワークシェアリング複合構造

並列/ワークシェアリング複合構造は、ワークシェアリング構造を 1 つ含む並列領域を指定するためのショートカットです。

これらの指令およびそれらが適用されるコードについては特別な条件と制限が多数あります。詳細については、適切な OpenMP の仕様書を参照してください。以下の説明は概要であり、完全なものではありません。

表 1-1 は、この構造とともに記述できる句を示しています。

PARALLEL DO、parallel for 構造

DO ループまたは **for** ループを 1 つ含む並列領域を指定するためのショートカットです。**PARALLEL** 指令の直後に **DO** 指令または **for** 指令を続けた場合と同じ意味になります。clause には、**PARALLEL** と **DO** または **for** の指令で使用可能な句を指定できますが、**NOWAIT** 修飾子は指定できません。

OpenMP 2.0 Fortran

```
!$OMP PARALLEL DO [clause[[,] clause]...]
   do_loop
[!$OMP END PARALLEL DO ]
```

OpenMP 2.0 C/C++

```
#pragma omp parallel for [clause[[,] clause]...]
      for-loop
```

PARALLEL SECTIONS 構造

SECTIONS 指令を 1 つ含む並列領域を指定するためのショートカットです。**PARALLEL** 指令の直後に **SECTIONS** 指令を続けた場合と同じ意味になります。clause には、**PARALLEL** および **SECTIONS** 指令で使用可能な句を指定できますが、**NOWAIT** 修飾子は指定できません。

OpenMP 2.0 Fortran

```
!$OMP PARALLEL SECTIONS [clause[[,] clause]...]
[!$OMP SECTION]
  structured-block
[!$OMP SECTION
  structured-block ]
...
!$OMP END PARALLEL SECTIONS
```

OpenMP 2.0 C/C++

```
#pragma omp parallel sections [clause[[,] clause]...]
  {
  [#pragma omp section]
       structured-block
  [#pragma omp section
       structured-block]
  ...
  }
```

PARALLEL WORKSHARE 構造

Fortran の **PARALLEL WORKSHARE** 構造は、**WORKSHARE** 指令を 1 つ含む並列領域を指定するためのショートカットです。clause には、**PARALLEL** または **WORKSHARE** 指令で使用可能な句を指定できます。

OpenMP 2.0 Fortran

```
!$OMP PARALLEL WORKSHARE [clause[[,] clause]...]
  structured-block
!$OMP END PARALLEL WORKSHARE
```

C/C++ には対応する構造はありません。

同期構造

次の構造は、スレッド同期化を指定します。これらの構造については特別な条件および制限がありますが、その数が多すぎるため、ここでは説明しません。詳細については、『OpenMP 仕様書』を参照してください。

MASTER 構造

チームのマスタースレッドだけが、この指令で囲まれたブロックを実行します。他のスレッドは、このブロックをスキップして続行します。マスター構造の入り口や出口には、暗黙のバリアはありません。

OpenMP 2.0 Fortran

```
!$OMP MASTER
  structured-block
!$OMP END MASTER
```

OpenMP 2.0 C/C++

```
#pragma omp master
     structured-block
```

CRITICAL 構造

構造ブロックへのアクセスを、同時に 1 スレッドだけに制限します。省略可能な name 引数には、危険領域の名前を指定します。名前指定のない **CRITICAL** 指令は、すべて同一名にマッピングされます。危険領域の名前はプログラムの大域要素であり、一意の名前を指定する必要があります。Fortran では、**CRITICAL** 指令で name に名前を指定した場合は、**END CRITICAL** 指令でもその名前を記述する必要があります。C/C++ の場合は、危険領域を指定する識別子は外部にリンクされていて、ラベル、タグ、メンバー、通常の識別子で使用される名前空間とは別の名前空間に含まれています。

OpenMP 2.0 Fortran

```
!$OMP CRITICAL [(name)]
  structured-block
!$OMP END CRITICAL [(name)]
```

OpenMP 2.0 C/C++

```
#pragma omp critical [(name)]
     structured-block
```

BARRIER 構造

チーム内のすべてのスレッドの同期をとります。各スレッドは、チーム内の他のすべてのスレッドがこの地点に到達するまで待機します。

OpenMP 2.0 Fortran

```
!$OMP BARRIER
```

OpenMP 2.0 C/C++

```
#pragma omp barrier
```

チーム内のすべてのスレッドがこのポイントに到達すると、チーム内の各スレッドは、**BARRIER** 指令の後ろの文の並列実行を開始します。

barrier プラグマの構文には C/C++ の文が含まれていないため、このプラグマをプログラム内に配置する際には制限があります。詳細については、『OpenMP C/C++ の仕様』を参照してください。

ATOMIC 構造

特定のメモリー位置をアトミックに更新するようにし、複数のスレッドによる同時書き込みが生じないようにします。

この実装では、危険領域内に expression-statement を配置することで、すべての ATOMIC 指令が置き換えられます。

OpenMP 2.0 Fortran

```
!$OMP ATOMIC
  expression-statement
```

指令は直後の文だけに適用されます。直後の文は、以下のいずれかの書式で指定します。

x = x operator expression
x = expression operator x
x = intrinsic(x, expr-list)
x = intrinsic(expr-list, x)

ここで、各要素の意味は以下のとおりです。

- x は、組み込み型のスカラーです。
- expression は、x を参照しないスカラー式です。
- expr-list は、x を参照しないスカラー式のリストです。空でない、コンマで区切ったリストを指定します (詳細については、OpenMP Fortran 2.0 仕様を参照)。
- intrinsic には、**MAX**、**MIN**、**IAND**、**IOR**、**IEOR** のいずれかを指定します。
- operator には、**+**、**-**、*****、**/**、**.AND.**、**.OR.**、**.EQV.**、**.NEQV.** のいずれかを指定します。

OpenMP 2.0 C/C++

```
#pragma omp atomic
    expression-statement
```

プラグマは直後の文だけに適用されます。直後の文は、以下のいずれかの書式で指定します。

x binop = expr
x++
++x
x--
--x

ここで、各要素の意味は以下のとおりです。

- x は、スカラー型の lvalue 式です。
- expr は、x を参照しないスカラー型の式です。
- binop は、多重定義された演算子以外の演算子で、**+**、*****、**-**、**/**、**&**、**^**、**|**、**<<**、**>>** のいずれかです。

FLUSH 構造

スレッド可視の Fortran 変数および C オブジェクトは、この指令の記述されている地点でメモリーに書き戻されます。**FLUSH** 指令は、実行中のスレッドおよび全体のメモリー内の処理間でだけ整合性を保証します。省略可能な variable-list は、フラッシュする必要のある変数またはオブジェクトをコンマで区切ったリストです。variable-list を指定せずに**FLUSH** 指令を記述した場合は、すべてのスレッド可視の共有変数またはオブジェクトの同期をとります。

OpenMP 2.0 Fortran

```
!$OMP FLUSH  [(variable-list)]
```

OpenMP 2.0 C/C++

```
#pragma omp flush [(variable-list)]
```

flush プラグマの構文には C/C++ の文が含まれていないため、このプラグマをプログラム内に配置する際には制限があります。詳細については、OpenMP C/C++ の仕様を参照してください。

ORDERED 構造

ORDERED 指令で囲まれた部分のブロックは、ループで逐次実行された場合の順序で実行されます。

OpenMP 2.0 Fortran

```
!$OMP ORDERED
  structured-block
!$OMP END ORDERED
```

ORDERED 指令で囲まれた部分のブロックは、ループで逐次実行された場合の順序で実行されます。**DO** または **PARALLEL DO** 指令の動的な範囲内でだけ指定することができます。**ORDERED** 句は、ブロックを囲むもっとも近い **DO** 指令で指定する必要があります。**DO** 指令が適用されるループでは、同一の **ordered** 指令を 1 回のループで複数回実行することはできません。また、複数の **ordered** 指令を実行することもできません。

OpenMP 2.0 C/C++

```
#pragma omp ordered
    structured-block
```

ordered 指令で囲まれた部分のブロックは、ループで逐次実行された場合の順序で実行されます。このブロックは、**ordered** 句が指定されている **for** または **parallel for** 指令の動的な範囲内でのみ指定できます。**for** 構造のあるループでは、同一の **ordered** 指令を 1 回のループで複数回実行することはできません。また、複数の **ordered** 指令を実行することもできません。

データ環境指令

以下の指令は、並列構造の実行時のデータ環境を設定します。

THREADPRIVATE 指令

オブジェクト (Fortran の共通ブロックと名前付き変数、C と C++ の名前付き変数) の list を、スレッドに対しては非公開に、スレッド内では広域に設定します。

詳細および制限については、OpenMP 仕様 (Fortran 仕様の 2.6.1 節、C/C++ 仕様の 2.7.1 節) を参照してください。

OpenMP 2.0 Fortran

```
!$OMP THREADPRIVATE(list)
```

共通ブロック名は、スラッシュで囲む必要があります。共通ブロックを **THREADPRIVATE** に設定するには、この指令をそのブロックのすべての **COMMON** 宣言の後に記述する必要があります。

OpenMP 2.0 C/C++

```
#pragma omp threadprivate (list)
```

list に指定した、ファイル、名前空間、ブロックスコープ内の変数は、字句的にこのプラグマの前に指定されているファイル、名前空間、ブロックスコープ内の変数宣言を参照する必要があります。

OpenMP 指令の句

ここでは、OpenMP 指令で記述可能な、データスコープおよびスケジュールを指定する句について説明します。

データスコープを指定する句

指令によっては、構造の範囲内で変数のスコープを設定できる句を使用できます。データスコープを指定する句を指令で使用していない場合は、指令が適用される変数のデフォルトのスコープは **SHARED** になります。

Fortran: list は、有効範囲内のアクセス可能な名前付き変数または共通ブロックを、コンマで区切ったリストです。共通ブロック名は、スラッシュで囲む必要があります (例 **/ABLOCK/**)。

スコープを指定する句の使用については、重要な制限があります。完全な情報については、Fortran 仕様の 2.6.2 節と C/C++ 仕様の 2.7.2 節を参照してください。

表 1-1 は、この構造とともに記述できる句を示しています。

PRIVATE 句

private(list)

list に指定した変数 (コンマで区切って複数指定可能) を、チーム内の各スレッドに対して非公開として宣言します。

SHARED 句

shared(list)

チーム内のすべてのスレッドは、list の変数を共有し、同一の記憶領域を使用します。

DEFAULT 句

Fortran

DEFAULT(PRIVATE | SHARED | NONE)

C/C++

default(shared | none)

並列領域内のすべての変数のスコープを指定します。**THREADPRIVATE** 変数は、この句の影響を受けません。指定しない場合は、**DEFAULT(SHARED)** が使用されます。変数のデフォルトのデータ共有属性は、**private**、**firstprivate**、**lastprivate**、**reduction**、**shared** クラスを使用して無効にすることができます。

FIRSTPRIVATE 句

firstprivate(list)

list に指定した変数が **PRIVATE** になります。また、変数の非公開コピーは、この構造の前に存在している元のオブジェクトで初期化されます。

LASTPRIVATE 句

lastprivate(list)

list に指定した変数が **PRIVATE** になります。また、**LASTPRIVATE** 句を **DO** または **for** 指令で記述している場合は、順序的に最後に反復を実行するスレッドが、構造の前にあったバージョンの変数を更新します。**SECTIONS** 指令では、字句的に最後の **SECTION** を実行するスレッドが、構造の前にあったバージョンのオブジェクトを更新します。

COPYIN 句

Fortran

COPYIN(list)

COPYIN 句は、**THREADPRIVATE** として宣言された変数、共通ブロック、共通ブロック内の変数だけに適用されます。並列領域では、**COPYIN** は、並列領域の最初に、チームのマスタースレッド内のデータをスレッドの共通ブロックの非公開の複製にコピーするように指定します。

C/C++

copyin(list)

COPYIN 句は、**THREADPRIVATE** として宣言された変数だけに適用されます。並列領域では、**COPYIN** は、並列領域の最初で、チームのマスタースレッド内のデータをスレッドの非公開の複製にコピーするように指定します。

COPYPRIVATE 句

Fortran

COPYPRIVATE(list)

チームの特定メンバーから他のメンバーに値をブロードキャスト通信するために、非公開変数 (共有オブジェクトへのポインタ) を使用します。**COPYPRIVATE** 句は、**END SINGLE** 指令でのみ指定できます。ブロードキャスト通信は、**single** 句に関連付けられた構造ブロックの実行後、構造の終わりでバリアを離れたチーム内のスレッドの前で発生します。list に指定した変数は、**COPYPRIVATE** を指定する **SINGLE** 構造の **PRIVATE** または **FIRSTPRIVATE** 句で使用できません。

C/C++

copyprivate(list)

チームの特定メンバーから他のメンバーに値をブロードキャスト通信するために、非公開変数を使用します。**copyprivate** 句は、**single** 指令でのみ指定できます。ブロードキャスト通信は、**single** 句に関連付けられた構造ブロックの実行後、構造の終わりでバリアを離れたチーム内のスレッドの前で発生します。list に指定した変数は、この **copyprivate** 句を指定した **single** 指令の **private** または **firstprivate** 句で使用することはできません。

PRIVATE 句

Fortran

REDUCTION(operator | intrinsic**:**list)

operator には、**+**、*****、**-**、**.AND.**、**.OR.**、**.EQV.**、**.NEQV.** のいずれかを指定します。

intrinsic には、**MAX**、**MIN**、**IAND**、**IOR**、**IEOR** のいずれかを指定します。

list の変数には、組み込み型の名前付き変数を指定する必要があります。

C/C++

reduction(operator:list)

operator には、**+**、*****、**-**、**&**、**^**、**|**、**&&**、**||**のいずれかを指定します。

REDUCTION 句は、縮約変数が縮約文でだけ使用されている領域で使用します。list の変数は、構造内のコンテキストでは **SHARED** に設定する必要があります。各変数の非公開の複製が、スレッドごとに **PRIVATE** であるものとして作成されます。縮約の最後に、元の値と非公開の複製の最終的な値とを結合することで、共有変数が更新されます。

詳細と **REDUCTION** 句、構造の制限については、Fortran OpenMP 仕様の 2.6.2.6 節と C/C++ 仕様の 2.7.2.6 節を参照してください。

スケジュールを指定する句

SCHEDULE 句は、Fortran の **DO** ループまたは C/C++ の **for** ループでの反復をチーム内のスレッド間で分割する方法を指定します。表 1-1 は、どの指令で **SCHEDULE** 句を記述できるかを示しています。

スケジュールを指定する句を使用する場合は、重要な制限があります。完全な情報については、Fortran 仕様の 2.3.1 節、C/C++ 仕様の 2.4.1 節を参照してください。

schedule(type [,chunk])

DO または **for** ループでの反復をチーム内のスレッド間で分割する方法を指定します。type には、**STATIC**、**DYNAMIC**、**GUIDED**、**RUNTIME** のいずれかを指定します。**SCHEDULE** 句がない場合、Sun ONE Studio のコンパイラは、**STATIC** を使用します。chunk には、整数式を指定する必要があります。

STATIC スケジューリング

schedule(static[,chunk])

反復は、chunk で指定したサイズに分割されます。分割された部分は、スレッドの番号順でラウンドロビン形式でチーム内のスレッドに静的に割り当てられます。chunk を指定していない場合は、ほぼ同サイズの連続チャンクに分割され、スレッドごとに 1 つずつチャンクが割り当てられます。

DYNAMIC スケジューリング

schedule(dynamic[,chunk])

反復は、chunk で指定したサイズに分割され、待機中のスレッドに割り当てられます。各スレッドは割り当てられた反復の部分を終了すると、反復の次のセットが動的に割り当てられます。chunk を指定していない場合は、デフォルトで 1 に設定されます。

GUIDED スケジューリング

schedule(guided[,chunk**])**

GUIDED を指定した場合、チャンクのサイズは反復のディスパッチごとに指数関数的に減少します。chunk は、ディスパッチごとの最小反復回数を指定します。(反復の最初のチャンクのサイズは、実装によって異なります。第 2 章を参照)。chunk を指定していない場合は、デフォルトで 1 に設定されます。

RUNTIME スケジューリング

schedule(runtime)

スケジュール指定は実行時まで遅延されます。スケジュールの type および chunk は、**OMP_SCHEDULE** 環境変数の設定によって決定されます。デフォルトでは **SCHEDULE(STATIC)** が指定されます。

NUM_THREADS 句

NUM_THREADS 句は、**PARALLEL**、**PARALLEL SECTIONS**、**PARALLEL DO**、**PARALLEL for**、**PARALLEL WORKSHARE** 指令で使用できます。

num_threads(scalar_integer_expression**)**

スレッドが並列領域に入ったときにチーム内で作成されるスレッドの数を指定します。scalar_integer_expression には、作成する必要のあるスレッドの数を指定します。指定を行うと、**OMP_SET_NUM_THREADS** ライブラリ関数の呼び出しによって決定されたスレッドの数、または、**OMP_NUM_THREADS** 環境変数の値が無効になります。動的なスレッド管理が有効な場合、指定した数がスレッドの最大数として使用されます。

num_threads は、以降の領域には適用されないことに注意してください。

指令での句の記述

表 1-1 は、以下の指令およびプラグマで記述できる句を示しています。

- **PARALLEL**
- **DO**
- **for**
- **SECTIONS**
- **SINGLE**
- **PARALLEL DO**
- **parallel for**
- **PARALLEL SECTIONS**
- **PARALLEL WORKSHARE**

表 1-1　　句とともに記述できるプラグマ

句/プラグマ	PARALLEL	DO/for	SECTIONS	SINGLE	PARALLEL DO/for	PARALLEL SECTIONS	PARALLELWORKSHARE[3]
IF	•				•	•	•
PRIVATE	•	•	•	•	•	•	•
SHARED	•				•	•	•
FIRSTPRIVATE	•	•	•	•	•	•	•
LASTPRIVATE		•	•		•	•	
DEFAULT	•				•	•	•
REDUCTION	•	•	•		•	•	•
COPYIN	•				•	•	•
COPYPRIVATE				•[1]			
ORDERED		•			•		
SCHEDULE		•			•		
NOWAIT		•[2]	•[2]	•[2]			
NUM_THREADS	•				•	•	•

1. Fortran のみ:**COPYPRIVATE** を **END SINGLE** 指令で指定できます。

2. Fortran では、**END DO**、**END SECTIONS**、**END SINGLE**、**END WORKSHARE** の各指令で **NOWAIT** 修飾子を使用することができます。

3. **WORKSHARE** および **PARALLEL WORKSHARE** は、Fortran でだけサポートされています。

OpenMP 実行時ライブラリルーチン

OpenMP は、並列実行環境の設定および照会を実行する呼び出し可能なライブラリルーチン、汎用ロックルーチン、2 つのポータブルタイマールーチンを提供します。詳細については、OpenMP Fortran 仕様、OpenMP C/C++ 仕様を参照してください。

Fortran の OpenMP ルーチン

Fortran の実行時ライブラリルーチンは、外部手続きです。以下の概要では、int_expr はスカラー整数式、logical_expr はスカラー論理式を示します。

INTEGER(4) および **LOGICAL(4)** を返す **OMP_** 関数は組み込み関数ではないため、正しく宣言する必要があります。宣言しない場合は、コンパイラでは **REAL** を返すものとして処理されます。以下で説明する OpenMP Fortran 実行時ライブラリルーチンのインタフェース宣言は、Fortran の インクルードファイルである **omp_lib.h** および Fortran **MODULE omp_lib** で提供されています。これについては、Fortran OpenMP 仕様で説明されています。

これらのライブラリルーチンを参照するすべてのプログラム単位で、**INCLUDE 'omp_lib.h'** 文、**#include "omp_lib.h"** プリプロセッサ指令、**USE omp_lib** 文のいずれかを記述してください。

-Xlist を指定してコンパイルを実行すると、あらゆる型の不一致が報告されます。

整数パラメータ **omp_lock_kind** は、**OMP_*_LOCK** ルーチンでの単純ロック変数で使用される **KIND** 型のパラメータを定義します。

整数パラメータ **omp_nest_lock_kind** は、**OMP_*_NEST_LOCK** ルーチンでの入れ子可能なロック変数で使用される **KIND** 型のパラメータを定義します。

整数パラメータ **openmp_version** は、**YYYYMM** という書式のプリプロセッサマクロ **_OPENMP** として定義されています。ここで、**YYYY** および **MM** は、OpenMP Fortran API のバージョンを年と月で示したものになります。

C/C++ の OpenMP ルーチン

C/C++ の実行時ライブラリ関数は、外部関数です。

ヘッダー **<omp.h>** では、2 つの型、並列実行環境の設定および照会に使用する複数の関数、データアクセスの同期をとるのに使用するロック関数が定義されています。

omp_lock_t 型は、ロックが使用可能であるか、スレッドがロックを所有しているかのいずれかを示すことができるオブジェクト型です。これらのロックを、単純ロックと呼びます。

omp_nest_lock_t 型は、ロックが使用可能であるか、スレッドがロックを所有しているかのいずれかを示すことができるオブジェクト型です。これらのロックを、ネストロックと呼びます。

実行時スレッド管理ルーチン

詳細については、それぞれの言語の OpenMP 仕様を参照してください。

OMP_SET_NUM_THREADS ルーチン

それ以降の並列領域で使用するスレッド数を設定します。

Fortran

```
SUBROUTINE OMP_SET_NUM_THREADS(int_expr)
```

C/C++

```
#include <omp.h>void omp_set_num_threads(int num_threads);
```

OMP_GET_NUM_THREADS ルーチン

チーム内で、呼び出し元の並列領域を実行しているスレッドの個数を返します。

Fortran

```
INTEGER(4) FUNCTION OMP_GET_NUM_THREADS()
```

C/C++

```
#include <omp.h>int omp_get_num_threads(void);
```

OMP_GET_MAX_THREADS ルーチン

OMP_GET_NUM_THREADS 関数の呼び出しが返す最大値を返します。

Fortran

```
INTEGER(4) FUNCTION OMP_GET_MAX_THREADS()
```

C/C++

```
#include <omp.h>int omp_get_max_threads(void);
```

OMP_GET_THREAD_NUM ルーチン

チーム内で、この関数の呼び出しを実行しているスレッドの個数を返します。値の範囲は、0 ～ OMP_GET_NUM_THREADS()-1 になります。0 はマスタースレッドを示します。

Fortran

```
INTEGER(4) FUNCTION OMP_GET_THREAD_NUM()
```

C/C++

```
#include <omp.h>int omp_get_thread_num(void);
```

OMP_GET_NUM_PROCS ルーチン

プログラムで使用可能なプロセッサ数を返します。

Fortran

```
INTEGER(4) FUNCTION OMP_GET_NUM_PROCS()
```

C/C++

```
#include <omp.h>int omp_get_num_procs(void);
```

OMP_IN_PARALLEL ルーチン

並列領域の動的な範囲内でスレッドを実行するかどうかを決定します。

Fortran

```
LOGICAL(4) FUNCTION OMP_IN_PARALLEL()
```

並列領域の動的な範囲内から呼び出された場合は .TRUE.、そうでない場合は .FALSE. を返します。

C/C++

```
#include <omp.h>int omp_in_parallel(void);
```

並列領域の動的な範囲内から呼び出された場合はゼロ以外の値、そうでない場合はゼロを返します。

OMP_SET_DYNAMIC ルーチン

使用可能なスレッドの数の動的調整を有効または無効にします。(デフォルトでは、動的調整が有効に設定されます。)

Fortran

```
SUBROUTINE OMP_SET_DYNAMIC(logical_expr)
```

logical_expr が .TRUE. の場合は動的調整が有効になり、それ以外の値の場合は無効になります。

C/C++

```
#include <omp.h>void omp_set_dynamic(int dynamic);
```

dynamic がゼロ以外の場合は、動的調整が有効になります。それ以外の場合は無効になります。

OMP_GET_DYNAMIC ルーチン

動的なスレッド調整が有効かどうかを特定します。

Fortran

```
LOGICAL(4) FUNCTION OMP_GET_DYNAMIC()
```

動的調整が有効な場合は .TRUE.、そうでない場合は .FALSE. を返します。

C/C++

```
#include <omp.h>int omp_get_dynamic(void);
```

動的調整が有効な場合はゼロ以外の値、そうでない場合はゼロを返します。

OMP_SET_NESTED ルーチン

ネストされた並列化機能を有効または無効にします。(ネストされた並列化機能はサポートされていないため、デフォルトで無効に設定されます。)

Fortran

```
SUBROUTINE OMP_SET_NESTED(logical_expr)
```

C/C++

```
#include <omp.h>void omp_set_nested(int nested);
```

OMP_GET_NESTED ルーチン

ネストされた並列化機能が有効かどうかを特定します。(ネストされた並列化機能はサポートされていないため、デフォルトで無効に設定されます。)

Fortran

```
LOGICAL(4) FUNCTION OMP_GET_NESTED()
```

.FALSE. を返します。ネストされた並列化機能はサポートされていません。

C/C++

```
#include <omp.h>int omp_get_nested(void);
```

ゼロを返します。ネストされた並列化機能はサポートされていません。

同期ロックを操作するルーチン

単純ロックとネストロックの 2 種類のロックがサポートされています。ネストロックは、ロック解除前に同一スレッドで複数回ロックできます。単純ロックは、すでにロック状態の場合はロックできません。単純ロック変数は、単純ロックルーチンにだけ渡すことができます。ネストロック変数は、ネストロックルーチンにだけ渡すことができます。

Fortran:

ロック変数 var にアクセスするには、後述のルーチンを使用する必要があります。このとき、**OMP_LOCK_KIND** および **OMP_NEST_LOCK_KIND** というパラメータを使用します (**omp_lib.h INCLUDE** ファイルおよび **omp_lib MODULE** で定義)。たとえば、次のように指定します。

```
INTEGER(KIND=OMP_LOCK_KIND)    ::
var          INTEGER(KIND=OMP_NEST_LOCK_KIND) :: nvar
```

C/C++:

単純ロック変数には、**omp_lock_t** 型を使用する必要があります。また、この変数にアクセスするには、後述のロックルーチンを使用する必要があります。すべての単純ロック関数では、**omp_lock_t** 型へのポインタを引数として指定する必要があります。

ネストロック変数には、**omp_nest_lock_t** 型を使用する必要があります。同様に、すべての入れ子ロック関数では、**omp_nest_lock_t** 型へのポインタを引数として指定する必要があります。

OMP_INIT_LOCK、OMP_INIT_NEST_LOCK ルーチン

それ以降の呼び出し用にロック変数を初期化します。

Fortran

```
SUBROUTINE OMP_INIT_LOCK(var)

SUBROUTINE OMP_INIT_NEST_LOCK(nvar)
```

C/C++

```
#include <omp.h>void omp_init_lock(omp_lock_t *lock);
```

```
void omp_init_nest_lock(omp_nest_lock_t *lock);
```

OMP_DESTROY_LOCK、OMP_DESTROY_NEST_LOCK ルーチン

ロック変数を削除します。

Fortran

```
SUBROUTINE OMP_DESTROY_LOCK(var)

SUBROUTINE OMP_DESTROY_NEST_LOCK(nvar)
```

C/C++

```
#include <omp.h>

void omp_destroy_lock(omp_lock_t *lock);

void omp_destroy_nest_lock(omp_nest_lock_t *lock);
```

OMP_SET_LOCK、OMP_SET_NEST_LOCK ルーチン

実行中のスレッドを、指定したロックが使用可能になるまで待機させます。指定したロックが使用可能になると、スレッドはそのロックの所有者になります。

Fortran

```
SUBROUTINE OMP_SET_LOCK(var)

SUBROUTINE OMP_SET_NEST_LOCK(nvar)
```

C/C++

```
#include <omp.h>

void omp_set_lock(omp_lock_t *lock);

void omp_set_nest_lock(omp_nest_lock_t *lock);
```

OMP_UNSET_LOCK、OMP_UNSET_NEST_LOCK ルーチン

実行中のスレッドから、ロックの所有権を解放します。スレッドがそのロックを所有していない場合の動作は未定義です。

Fortran

```
SUBROUTINE OMP_UNSET_LOCK(var)

SUBROUTINE OMP_UNSET_NEST_LOCK(nvar)
```

C/C++

```
#include <omp.h>

void omp_unset_lock(omp_lock_t *lock);

void omp_unset_nest_lock(omp_nest_lock_t *lock);
```

OMP_TEST_LOCK、OMP_TEST_NEST_LOCK ルーチン

OMP_TEST_LOCK ロック変数に関連付けられたロックを設定します。この呼び出しにより、スレッドの実行がブロックされることはありません。

OMP_TEST_NEST_LOCK ロックが正常に設定された場合は、最新のネスト数を返します。それ以外の場合は 0 を返します。この呼び出しにより、スレッドの実行がブロックされることはありません。

Fortran

```
LOGICAL(4) FUNCTION OMP_TEST_LOCK(var)
```

ロックが設定された場合は **.TRUE.**、そうでない場合は **.FALSE.** を返します。

```
INTEGER(4) FUNCTION OMP_TEST_NEST_LOCK(nvar)
```

ロックが正常に設定された場合は、最新のネスト数を返します。それ以外の場合は 0 を返します。

C/C++

```
#include <omp.h>int omp_test_lock(omp_lock_t *lock);
```

ロックが正常に設定された場合は、ゼロ以外の値を返します。それ以外の場合は 0 を返します。

```
int omp_test_nest_lock(omp_nest_lock_t *lock);
```

ロックが正常に設定された場合は、ロックの最新のネスト数を返します。それ以外の場合は 0 を返します。

タイミングルーチン

2 つの関数でポータブル時計時間タイマーをサポートしています。

OMP_GET_WTIME ルーチン

過去のある時点から経過した時計時間を秒数で返します。

Fortran

```
REAL(8) FUNCTION OMP_GET_WTIME()
```

C/C++

```
#include <omp.h>double omp_get_wtime(void);
```

OMP_GET_WTICK ルーチン

連続するクロック刻みの間隔の秒数を返します。

Fortran

```
REAL(8) FUNCTION OMP_GET_WTICK()
```

C/C++

```
#include <omp.h>double omp_get_wtick(void);
```

第2章

実装に関わる問題

この章では、OpenMP 2.0 Fortran および OpenMP 2.0 C/C++ の各仕様に固有の問題について説明します。最新のコンパイラリリースの最新の情報については、C、C++、Fortran の readme ファイルを参照してください。

スケジュール指定

- **OMP_SCHEDULE** 環境変数または **SCHEDULE** 句を明示的に設定していない場合は、**static** スケジュール指定がデフォルトで使用されます。

スレッド数

- **num_threads()** 句、**omp_set_num_threads**() 関数の呼び出し、**OMP_NUM_THREADS** 環境変数の定義が明示的に行われていない場合は、チームのスレッド数はデフォルトで 1 に設定されます。

動的スレッド

- **omp_set_dynamic()** 関数の呼び出しまたは **OMP_DYNAMIC** 環境変数の定義が明示的に行われていない場合は、動的スレッド調整がデフォルトで有効になります。動的スレッド調整が有効な場合、スレッドの数は使用可能なプロセッサの数に限られます。

ネストされた並列化機能

- ネストされた並列化機能はこの実装ではサポートされていないため、デフォルトにより無効になります。ネストされた並列領域は、1 つのスレッドによって実行されます。

ATOMIC 指令

- この実装では、危険領域内に文を含めることで、すべての **ATOMIC** 指令およびプラグマが置き換えられます。

GUIDED のチャンクの最初のサイズと最小サイズ

- **SCHEDULE(GUIDED,** chunk**)** のデフォルトの最小チャンクサイズは 1 です。デフォルトの最初のチャンクサイズは、ループ内の反復数をループを実行するスレッド数で割った値になります。

明示的にスレッド化されたプログラム

- スレッドを使用するプログラムでは、Open MP の指令が含まれるルーチンを呼び出すことができます。

第3章

OpenMP 用のコンパイル

この章では、OpenMP API を使用するプログラムをコンパイルする方法を説明します。

並列化プログラムをマルチスレッド環境で実行するには、**OMP_NUM_THREADS** 環境変数をプログラム実行前に設定する必要があります。これにより、プログラムで作成される最大スレッド数を実行時システムに設定します。デフォルトは 1 です。通常は、**OMP_NUM_THREADS** を対象プラットフォームで使用可能なプロセッサ数に設定します。

コンパイラの readme ファイルで、OpenMP の実装に関する制限および既知の問題を説明しています。readme ファイルは、`-xhelp=readme` フラグを指定してコンパイラを起動するか、HTML ブラウザで以下のマニュアル索引を指定することで表示できます。

```
file:/opt/SUNWspro/docs/index.html
```

使用するコンパイラオプション

OpenMP の指令を使用して明示的に並列化を有効にするには、**cc**、**CC**、または **f95** のオプションフラグ **-xopenmp** を指定してプログラムをコンパイルします。このフラグには、キーワードの引数を 1 つ指定することができます(**f95** コンパイラでは、**-xopenmp** と **-openmp** を同義語として使用することができます)。

-xopenmp フラグには、以下のキーワードを指定することができます。

-xopenmp=parallel	OpenMP プラグマが認識されるように設定します。-xopenmp=parallel の最適化レベルは -xO3 です。コンパイラは、必要に応じて、最適化のレベルを -xO3 に上げ、警告を出力します。
-xopenmp=noopt	OpenMP プラグマが認識されるように設定します。最適化のレベルが -xO3 より低い場合でも、コンパイラは最適化のレベルを上げません。-xO2 -openmp=noopt のように最適化レベルを明示的に -xO3 より下げると、コンパイラはエラーを出力します。-openmp=noopt を使用して最適化レベルを指定しない場合、OpenMP プラグマが認識され、並列化されますが、最適化は行われません。(このオプションは、cc と f95 でのみ使用できます。CC でこのオプションを指定すると、警告が出力され、OpneMP の並列化は行われません。)
-xopenmp=stubs	OpenMP プラグマの認識を無効にし、スタブライブラリルーチンに接続し、最適化レベルを変更しません。OpenMP の実行時ライブラリルーチンを明示的に呼び出すアプリケーションを逐次実行するようにコンパイルする必要がある場合、このオプションを使用します。
-xopenmp=none	OpenMP プラグマの認識を無効にし、最適化レベルを変更しません (デフォルト)。

コマンド行で **-xopenmp** を指定しないと、**-xopenmp=none** (OpenMP プラグマの認識を無効にする) を指定したと見なされます。

-xopenmp をキーワードなしで指定した場合は、コンパイラでは **-xopenmp=parallel** が指定されます。

コマンド行で、**-xopenmp** を **-xparallel** または **-xexplicitpar** と共に指定しないでください。

-xopenmp= に **parallel**、**noopt**、または **stubs** を指定すると、**_OPENMP** プリプロセッサトークンが YYYYMM (C/C++ では **200203L**、Fortran 95 では **200011**) として定義されます。

dbx を使用して OpenMP プログラムをデバッグする場合、**-xopenmp=noopt -g** を使用してコンパイルします。

-xopenmp のデフォルトの最適化レベルは将来のリリースで変更される可能性があります。適切な最適化レベルを明示的に指定することによって、警告メッセージの表示を防止することができます。

Fortran 95 では、**-xopenmp**、**-xopenmp=parallel**、**-xopenmp=noopt** を指定すると、**-stackvar** が自動的に追加されます。

Fortran 95 OpenMP の妥当性検査

f95 コンパイラのプログラム全体のチェック機能を使用して、Fortran 95 プログラムの OpenMP 指令の手続き間の妥当性検査を静的に実行することができます。OpenMP のチェックは、**-XlistMP** フラグを指定してコンパイルを行うことによって有効になります。(**-XlistMP** を指定した場合の診断メッセージは、ソースファイル名に **.lst** という拡張子の付いた名前の別ファイルの形で出力されます。) コンパイラは、以下の違反と並列化の阻害要因を診断します。

- 並列化指令の仕様の違反 (不正なネストを含む)
- 手続き間の依存性解析により検出される、データの使用が原因である並列化の阻害要因
- 手続き間のポインタ解析により検出される並列化の阻害要因

たとえば、ord.f というソースファイルを -XlistMP を指定してコンパイルすると、ord.lst という診断ファイルが生成されます。

```
FILE  "ord.f"
     1  !$OMP PARALLEL
     2  !$OMP DO ORDERED
     3                  do i=1,100
     4                          call work(i)
     5                  end do
     6  !$OMP END DO
     7  !$OMP END PARALLEL
     8
     9  !$OMP PARALLEL
    10  !$OMP DO
    11                  do i=1,100
    12                          call work(i)
    13                  end do
    14  !$OMP END DO
    15  !$OMP END PARALLEL
    16                  end
    17                  subroutine work(k)
    18  !$OMP ORDERED
         ^
**** ERR-OMP:  It is illegal for an ORDERED directive to bind to a
directive (ord.f, line 10, column 2) that does not have the
ORDERED clause specified.
    19                  write(*,*) k
    20  !$OMP END ORDERED
    21                  return
    22                  end
```

この例では、サブルーチン **WORK** 内の **ORDERED** 指令は、**ORDERED** 句がないため、2 番目の **DO** 指令を参照しているという診断が出力されています。

OpenMP 環境変数

OpenMP 仕様では、OpenMP プログラムの実行を制御する環境変数が 4 つ定義されています。これらの環境変数を下表に示します。

表 3-1　OpenMP 環境変数

環境変数	機能
OMP_SCHEDULE	スケジュール型が**RUNTIME**として指定された **DO**、**PARALLEL DO**、**parallel for**、**for** の指令またはプラグマのスケジュール型を設定します。定義しない場合は、デフォルト値の **STATIC** が使用されます。value は ";type[,chunk]" という書式で指定します。 例: `setenv OMP_SCHEDULE "GUIDED,4"`
OMP_NUM_THREADS または **PARALLEL**	**NUM_THREADS** 句または **OMP_SET_NUM_THREADS()** の呼び出しで設定した場合を除き、実行時に使用するスレッド数を設定します。設定しない場合は、デフォルト値の 1 が使用されます。value には、正の整数を指定します (現在のバージョンの最大値は 128)。従来のプログラムとの互換性のため、**PARALLEL** 環境変数を設定すると **OMP_NUM_THREADS** を設定するのと同じ効果が得られます。ただし、それらが共に異なる値に設定されると、実行時ライブラリはエラーメッセージを発行します。 例: `setenv OMP_NUM_THREADS 16`
OMP_DYNAMIC	並列領域の実行で使用可能なスレッド数の動的調整を有効または無効にします。設定しない場合は、デフォルト値の **TRUE** が使用されます。value には、**TRUE** または **FALSE** を指定します。 例: `setenv OMP_DYNAMIC FALSE`
OMP_NESTED	ネストされた並列化機能を有効または無効にします (ネストされた並列化機能はサポートされていません)。 value には、**TRUE** または **FALSE** を指定します (この変数は現在のバージョンでは効果がありません)。 例: `setenv OMP_NESTED FALSE`

これ以外にも、OpenMP プログラムの実行に影響を与える多重処理に関する環境変数がありますが、OpenMP 仕様には含まれていません。これらの環境変数を下表に示します。

表 3-2　多重処理に関する環境変数

環境変数	機能
SUNW_MP_WARN	OpenMP の実行時ライブラリで出力される警告メッセージを制御します。**TRUE** に設定した場合は、実行時ライブラリの警告メッセージが stderr に出力されます。**FALSE** に設定した場合は、警告メッセージが無効になります。デフォルトは **FALSE** です。 例: **setenv SUNW_MP_WARN FALSE**
SUNW_MP_THR_IDLE	プログラムの並列部分を実行する各ヘルパースレッドのタスク終了時点の状態を制御します。**spin**、**sleep ns**、**sleep nms** のいずれかに設定できます。デフォルトは **SPIN** です。この場合、スレッドは並列タスクの完了後は、新しい並列タスクが到着するまでスピン (busy-waits) します。 **SLEEP** time を指定した場合は、並列タスクの完了後にヘルパースレッドがスピンを継続する時間を指定します。スレッドのスピン中にそのスレッド用の新しいタスクが到着した場合は、スレッドは新しいタスクをすぐに実行します。それ以外の場合は、スレッドはスリープし、新しいタスクの到着時に動作を再開します。time は、秒数（**ns**）または（n）またはミリ秒（n**ms**）で指定できます。 引数なしで **SLEEP** を指定すると、スレッドは並列タスクの完了直後にスリープします。**SLEEP**、**SLEEP (0)**、**SLEEP (0s)**、**SLEEP (0ms)** はすべて同義です。 例:　**setenv SUNW_MP_THR_IDLE SLEEP(50ms)**
STACKSIZE	各スレッドのスタックサイズを設定します。値はキロバイト単位で指定します。デフォルトのスレッドスタックサイズは、32 ビット SPARC V8 プラットフォームで 4M バイト、64 ビット SPARC V9 プラットフォームで 8M バイトです。 例: **setenv STACKSIZE 8192**　スレッドのスタックサイズを 8M バイトに設定します

スタックとスタックサイズ

実行プログラムは、各ヘルパースレッド用の個別スタックのほか、プログラムを実行する初期スレッド用のメインメモリースタックを保持します。スタックは、サブプログラムまたは関数参照で引数および自動変数を保持するために使用される一時的なメモリーアドレス空間です。

デフォルトのメインスタックは約 8M バイトです。`f95` オプション **`-stackvar`** を指定して Fortran プログラムをコンパイルすると、自動変数であるかのようにスタック上にローカル変数と配列が割り当てられます。OpenMP プログラムでの **`-stackvar`** 指定は、明示的に並列化されたプログラムで必要になります。これは、オプティマイザのループでの呼び出しの並列化機能を向上させるためです(**`-stackvar`** フラグについては、『Fortran ユーザーズガイド』を参照)。ただし、スタックに十分なメモリーが割り当てられていない場合は、スタックのオーバーフローが発生する可能性があります。

メインスタックのサイズを表示または設定するには、C シェルの **`limit`** コマンド、または ksh、sh の `ulimit` コマンドを使用します。

マルチスレッドプログラムの各ヘルパースレッドは、それぞれスレッドスタックを持ちます。このスタックは最初の (メイン) スレッドスタックに似ていますが、そのスレッドに固有のものです。スレッドの **`PRIVATE`** 配列および変数 (スレッドにローカル) は、スレッドスタックに割り当てられます。デフォルトのサイズは、32 ビットシステムで 4M バイト、64 ビットシステムで 8M バイトです。ヘルパースレッドスタックのサイズは、**`STACKSIZE`** 環境変数で設定されます。

```
demo% setenv STACKSIZE 16384    <-スレッドのスタックサイズを 16 Mb に設定
(C シェル)

demo% STACKSIZE=16384           <-同上 (Bourne/Korn シェル)
demo% export STACKSIZE
```

最適なスタックサイズを判定するには、試行とエラーを経る必要があるかもしれません。スタックサイズがスレッドに対して小さすぎて実行できない場合、エラーメッセージが出力されないまま、隣接するスレッドでデータ破壊やセグメントエラーが発生する可能性があります。スタックオーバーフローが発生するかどうか不確かな場

合、**-xcheck=stkovf** フラグを指定して Fortran または C プログラムをコンパイルすると、スタックオーバーフローのセグメント例外を発生させることができます。この場合、データ破壊が発生する前にプログラムの実行が停止します。

第4章

OpenMP への変換

この章では、Sun または Cray の指令およびプラグマを使用する従来のプログラムを OpenMP に変換するための指針を説明します。

従来の Fortran 指令の変換

従来の Fortran プログラムでは、Sun または Cray 形式の並列化指令が使用されています。これらの指令の詳細については、『Fortran プログラミングガイド』の並列化に関する章を参照してください。

Sun 形式の Fortran の指令の変換

次の表は、Sun の並列化指令およびその従属句と、それに相当する OpenMP の指令の概要です。これらは、変換の一例です。

表 4-1 Sun の並列化指令を OpenMP の指令に変換する

Sun の指令	OpenMP の指令
`C$PAR DOALL` [qualifiers]	`!$omp parallel do` [qualifiers]

`C$PAR DOSERIAL`	完全に相当する句はありません。以下で代用することができます。
	`!$omp master`
	loop
	`!$omp end master`

`C$PAR DOSERIAL*`	完全に相当する句はありません。以下で代用することができます。
	`!$omp master`
	loopnest
	`!$omp end master`

`C$PAR TASKCOMMON` block[,...]	`!$omp threadprivate (/block/[,...])`

DOALL 指令では、以下の修飾句を指定することができます。

表 4-2 DOALL 修飾句とそれに相当する OpenMP の句

Sun の DOALL 句	OpenMP の PARALLEL DO に相当する句
`PRIVATE(v1,v2,...)`	`private(v1,v2,...)`
`SHARED(v1,v2,...)`	`shared(v1,v2,...)`
`MAXCPUS(n)`	`num_threads(n)` 完全に相当する句はありません。
`READONLY(v1,v2,...)`	完全に相当する句はありません。`firstprivate(v1,v2,...)` を使用して同じ効果を得ることができます。
`STOREBACK(v1,v2,...)`	完全に相当する句はありません。`lastprivate(v1,v2,...)` を使用して同じ効果を得ることができます。

表 4-2　DOALL 修飾句とそれに相当する OpenMP の句 (続き)

Sun の DOALL 句	OpenMP の PARALLEL DO に相当する句
SAVELAST	完全に相当する句はありません。lastprivate(v1,v2,...) を使用して同じ効果を得ることができます。
REDUCTION(v1,v2,...)	reduction(operator:v1,v2,...) 縮約演算子および変数リストを指定する必要があります。
SCHEDTYPE(spec)	schedule(spec) (表 4-3 を参照)

SCHEDTYPE(spec) 句では、以下のスケジュール指定を使用することができます。

表 4-3　SCHEDTYPE のスケジュール指定とそれに相当する OpenMP の schedule

SCHEDTYPE(spec)	OpenMP の schedule(spec) 句
SCHEDTYPE(STATIC)	schedule(static)
SCHEDTYPE(SELF(chunksize))	schedule(dynamic,chunksize) デフォルトの chunksize の値は 1 です。
SCHEDTYPE(FACTORING(m))	OpenMP で完全に相当する句はありません。
SCHEDTYPE(GSS(m))	schedule(guided, m) デフォルトの m の値は 1 です。

Sun 形式の Fortran の指令と OpenMP の変換の問題

- OpenMP では、変数のスコープ (共有または非公開) を明示的に宣言する必要があります。Sun の指令では、PRIVATE または SHARED 句で明示的にスコープが指定されていない変数の場合は、コンパイラは専用のデフォルトのスコープ規則を使用します。つまり、すべてのスカラーは PRIVATE、すべての配列参照は SHARED として処理されます。OpenMP では、DEFAULT(PRIVATE) 句を PARALLEL DO 指令で使用している場合を除き、デフォルトのデータスコープは SHARED です。DEFAULT(NONE) 句を使用すると、コンパイラで変数の有効範囲が明示的に設定されません。

- DOSERIAL 指令がないため、自動と明示的な OpenMP の並列化を混在させると異なる結果になることがあります。Sun の指令では並列化されていなかったループが、自動的に並列化されることがあります。

- OpenMP では並列領域と並列セクションを用意しているため、並列化モデルが豊富です。したがって、Sun の指令を使用するプログラムの並列化戦略を再設計し、OpenMP の機能を利用するようにすることで性能の向上を実現することができます。

Cary 形式の Fortran の指令の変換

Cray 形式の Fortran 並列化指令は、指令を示す標識が `!MIC$` である点を除き、Sun 形式のものと同一です。また、`!MIC$ DOALL` の修飾句も異なります。

表 4-4　Cray 形式の `DOALL` 修飾句とそれに相当する Open MP の句

Cray の DOALL 句	OpenMP の PARALLEL DO に相当する句
`SHARED(v1,v2,...)`	`SHARED(v1,v2,...)`
`PRIVATE(v1,v2,...)`	`PRIVATE(v1,v2,...)`
`AUTOSCOPE`	相当する句はありません。スコープは明示的に指定するか、`DEFAULT` 句を使用する必要があります。
`SAVELAST`	完全に相当する句はありません。`lastprivate` を使用して同じ効果を得ることができます。
`MAXCPUS(n)`	`num_threads(n)`.完全に相当する句はありません。
`GUIDED`	`schedule(guided, m)` デフォルトの m の値は 1 です。
`SINGLE`	`schedule(dynamic,1)`
`CHUNKSIZE(n)`	`schedule(dynamic,n)`
`NUMCHUNKS(m)`	`schedule(dynamic,n/m)`　n には反復数を指定します。

Cray 形式の Fortran の指令と OpenMP の指令の変換の問題

両者の違いは、Cray の `AUTOSCOPE` に相当するものがない点を除き、Sun 形式の指令の場合と同様です。

従来の C プラグマの変換

C コンパイラでは、明示的な並列化用の従来のプラグマを使用することができます。これらのプラグマについては、『C ユーザーズガイド』を参照してください。Fortran の指令の場合と同様に、これらは一例です。

従来の並列化プラグマは、以下のとおりです。

表 4-5 C の並列化プラグマを OpenMP に変換する

従来の C プラグマ	相当する OpenMP プラグマ
`#pragma MP taskloop` [clauses]	`#pragma omp parallel for` [clauses]
`#pragma MP serial_loop`	完全に相当する句はありません。以下で代用することができます。 `#pragma omp master` loop
`#pragma MP serial_loop_nested`	完全に相当する句はありません。以下で代用することができます。 `#pragma omp master` loopnest

`taskloop` プラグマでは、以下の句を指定できます。

表 4-6 `taskloop` の句とそれに相当する OpenMP の句

`taskloop` の句	OpenMP の `parallel for` に相当する句
`maxcpus(n)`	相当する句はありません。`num_threads(n)` を使用します。
`private(v1,v2,...)`	`private(v1,v2,...)`
`shared(v1,v2,...)`	`shared(v1,v2,...)`
`readonly(v1,v2,...)`	完全に相当する句はありません。`firstprivate(v1,v2,...)` を使用して同じ効果を得ることができます。
`storeback(v1,v2,...)`	完全に相当する句はありません。`lastprivate(v1,v2,...)` を使用して同じ効果を得ることができます。

表 4-6　taskloop の句とそれに相当する OpenMP の句 (続き)

taskloop の句	OpenMP の parallel for に相当する句
savelast(v1,v2,...)	完全に相当する句はありません。lastprivate(v1,v2,...) を使用して同じ効果を得ることができます。
reduction(v1,v2,...)	reduction(operator:v1,v2,...)縮約演算子および変数リストを指定する必要があります。
schedtype(spec)	schedule(spec) (表 4-7 を参照)

schedtype(spec) 句では、以下のスケジュール指定を使用することができます。

表 4-7　SCHEDTYPE のスケジュール指定とそれに相当する OpenMP の schedule

schedtype(spec)	OpenMP の schedule(spec) 句
SCHEDTYPE(STATIC)	schedule(static)
SCHEDTYPE(SELF(chunksize))	schedule(dynamic,chunksize) 注:デフォルトのchunksize の値は 1 です。
SCHEDTYPE(FACTORING(m))	OpenMP で完全に相当する句はありません。
SCHEDTYPE(GSS(m))	schedule(guided, m) デフォルトの m の値は 1 です。

従来の C のプラグマと OpenMP の変換の問題

- 並列構造内で宣言された変数は、スコープが非公開になります。#pragma omp parallel for 指令で default(none) 句を使用すると、コンパイラで変数の有効範囲が明示的に設定されません。
- serial_loop 指令がないため、自動と明示的な OpenMP の並列化を混在させると異なる結果になることがあります。従来の C の指令では並列化されていなかったループが、自動的に並列化されることがあります。
- OpenMP の方が並列化モデルが豊富なため、従来の C の指令を使用するプログラムの並列化戦略を再設計し、OpenMP の機能を利用することで、多くの場合は性能を向上できます。

索引

と

ね

は

ふ

へ

ま

め

わ