# Fortran と OpenMP* でヘテロジニアス・プログラミングの課題を解決

## オープンな標準プログラミング言語でヘテロジニアス並列処理を表現

**Shiquan Su 博士 インテル コーポレーション ソフトウェア・テクニカル・コンサルティング・エンジニア**
**Ron Green インテル コーポレーション コンパイラー・エンジニアリング・マネージャー**
**Barbara Perz インテル コーポレーション ソフトウェア・テクニカル・コンサルティング・エンジニア**
**Pamela Harrison インテル コーポレーション ソフトウェア・テクニカル・コンサルティング・エンジニア**

急速な技術革新は、ヘテロジニアス・コンピューティングの新しい時代をもたらしています。ハードウェアの多様性と増大する計算能力の要求により、ヘテロジニアス並列処理を利用できるプログラミング・モデルが求められています。これらのモデルは、さまざまなベンダーのハードウェアでプログラムを実行できるように、オープンで、移植性に優れていなければなりません。Fortran は古い言語ですが、今でも科学や工学の分野で活発に利用されている重要なプログラミング言語です。同様に、1997 年にリリースされたコンパイラー主導の並列処理のオープンな標準である OpenMP* も、ヘテロジニアス・コンピューティングをサポートするように進化し、計算をアクセラレーター・デバイスにオフロードして不連続メモリー間でデータを移動するディレクティブが含まれるようになりました。開発者は、ホストとデバイスメモリーの概念、テクスチャー / サーフェスや定数メモリーなどのより繊細なメモリータイプを、OpenMP* ディレクティブを通じて利用できます。

タスクをアクセラレーターにオフロードすると、一部の計算はより効率的になります。例えば、高度なデータ並列計算では、GPU の多くの処理要素を活用できます。この記事では、Fortran + OpenMP* がヘテロジニアス・コンピューティングの 3 つの主な課題である、アクセラレーターへの計算のオフロード、不連続メモリーの管理、ターゲットデバイス上での既存の API の呼び出し、をどのように解決するかを示します。

## アクセラレーターへの計算のオフロード

最初に、簡単な例から始めましょう。**リスト 1** は、OpenMP* の `TARGET`、`TEAMS`, および `DISTRIBUTE PARALLEL DO` 構文を使用して入れ子のループを実行する方法を示しています。`TARGET` 構文はターゲットデバイス上の並列領域を作成します。`TEAMS` 構文はチームのリーグ（つまり、スレッドのグループ）を作成します。この例では、チームの数は `num_blocks` パラメーター以下です。各チームには、`block_threads` 変数以下の数のスレッドがあります。各チームのプライマリー・スレッドは、`teams` 領域でコードを実行します。外部ループの反復は、各チームのプライマリー・スレッドに分散されます。チームのプライマリー・スレッドが `DISTRIBUTE PARALLEL DO` 構文を検出すると、チーム内のほかのスレッドがアクティブになります。チームは並列領域を実行した後、内部ループの実行をワークシェアします。この一連の処理を図式化したのが**図 1** です。

```
PROGRAM target_teams_distribute
   EXTERNAL saxpy

   INTEGER, PARAMETER :: n = 2048, num_blocks = 64
   REAL,  ALLOCATABLE :: A(:), B(:), C(:)
   REAL               :: d_sum = 0.0
   INTEGER            :: i, block_size = n / num_blocks
   INTEGER            :: block_threads = 128

   ALLOCATE(A(n), B(n), C(n))
   A = 1.0
   B = 2.0
   C = 0.0

   CALL saxpy(A, B, C, n, block_size, num_blocks, block_threads)

   DO i = 1, n
      d_sum = d_sum + C(i)
   ENDDO

   PRINT '("sum = 2048 x 2 saxpy sum:"(f))', d_sum

   DEALLOCATE(A, B, C)
END PROGRAM target_teams_distribute

SUBROUTINE saxpy(B, C, D, n, block_size, num_teams, block_threads)
   REAL    :: B(n), C(n), D(n)
   INTEGER :: n, block_size, num_teams, block_threads, i, i0

   !$OMP TARGET MAP(TO: B, C) MAP(TOFROM: D)
   !$OMP TEAMS num_teams(num_teams) thread_limit(block_threads)
   DO i0 = 1, n, block_size
      !$OMP DISTRIBUTE PARALLEL DO
      DO i = i0, min(i0 + block_size - 1, n)
         D(i) = D(i) + B(i) * C(i)
      enddo
   ENDDO
   !$OMP END TEAMS
   !$OMP END TARGET
END SUBROUTINE
```

リスト 1. OpenMP* ディレクティブ（青で表示）を使用して入れ子のループをアクセラレーターにオフロード

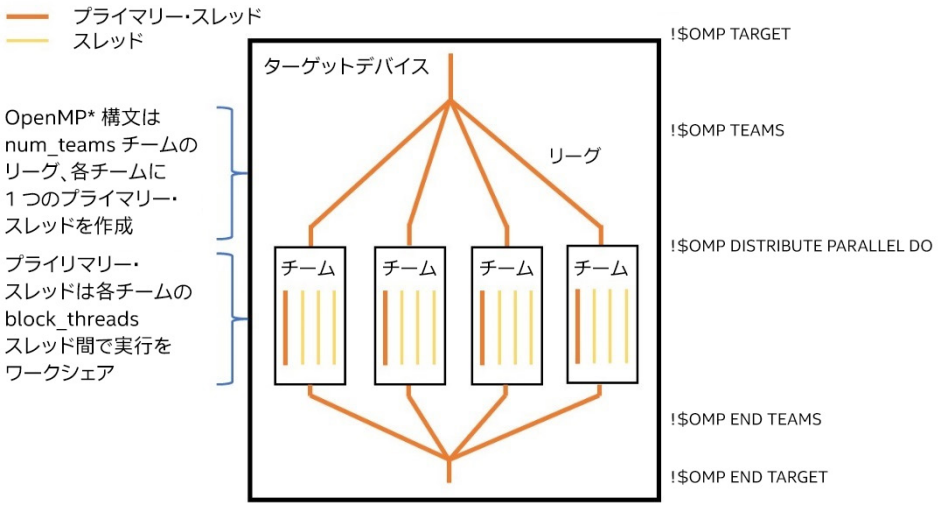


図 1. OpenMP* TARGET、TEAMS、および DISTRIBUTE PARALLEL DO 領域の概念図

## ホストとデバイス間のデータ転送

次に、ホストとデバイス間のメモリー管理とデータ移動について説明します。OpenMP* には 2 つのアプローチが用意されています。1 つ目のアプローチは、DATA 構文を使用して、不連続メモリー間のデータをマップします。例えば、**リスト 1** は、TARGET ディレクティブの MAP(TO: B, C) 節と MAP(TOFROM: D) 節で配列 B、C、および D をデバイスにコピーし、デバイスから D の最終値を取得します。2 つ目のアプローチは、OpenMP* ランタイム・ライブラリー・ルーチンの 1 つ、デバイス・メモリー・アロケーターの呼び出しです。この記事では、2 つ目のアプローチは説明しません。

**リスト 2** は、TARGET DATA 構文で新しいデバイスデータ環境（ターゲットデータ領域とも呼ばれる）を作成し、配列 A、B、および C をマップしています。TARGET DATA 領域は、2 つの TARGET 領域を囲んでいます。最初の領域は、MAP(TO: A, B) および MAP(FROM: C) データモーション節に従って囲んでいるデバイスデータ環境から A、B、および C を継承する新しいデバイスデータ環境を作成します。ホストは、1 つ目のターゲット領域が完了するのを待ってから、データ環境の A と B に新しい値を割り当てます。TARGET UPDATE 構文は、デバイスデータ環境で A と B を更新します。2 つ目のターゲット領域が終了すると、デバイスデータ環境の終了時に C の結果がデバイスからホストメモリーにコピーされます。この一連の処理を図式化したのが**図 2** です。

```
PROGRAM target_data_update
   INTEGER           :: i, n = 2048
   REAL, ALLOCATABLE :: A(:), B(:) ,C(:)
   REAL              :: d_sum = 0.0

   ALLOCATE(A(n), B(n), C(n))
   A = 1.0
   B = 2.0
   C = 0.0

   !$OMP TARGET DATA MAP(TO: A, B) MAP(FROM: C)
   !$OMP TARGET
   !$OMP PARALLEL DO
   DO i = 1, n
      C(i) = A(i) * B(i)
   ENDDO
   !$OMP END TARGET

   A = 2.0
   B = 4.0

   !$OMP TARGET UPDATE TO(A, B)

   !$OMP TARGET
   !$OMP PARALLEL DO
   DO i = 1, n
      C(i) = C(i) + A(i) * B(i)
   ENDDO
   !$OMP END TARGET
   !$OMP END TARGET DATA

   DO i = 1, n
      d_sum = d_sum + C(i)
   ENDDO

   PRINT '("sum = 2048 x (2 + 8) sum:"(f))', d_sum

   DEALLOCATE(A, B, C)
END PROGRAM target_data_update
```

リスト 2. デバイスデータ環境の作成

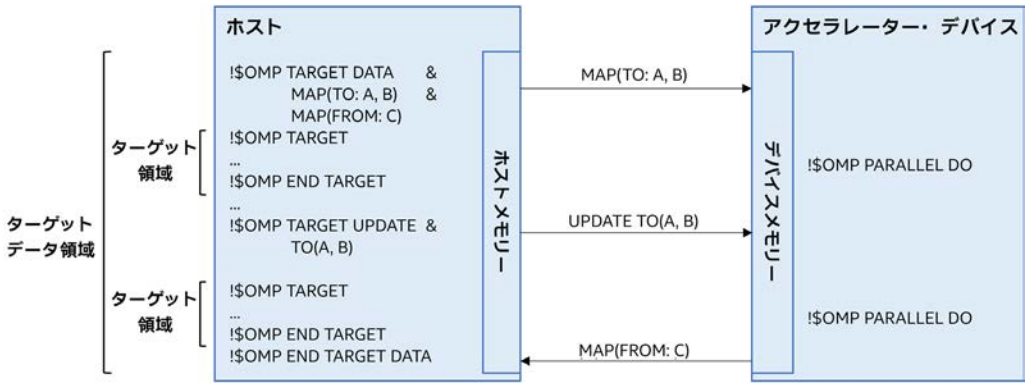


図 2. リスト 2 の OpenMP* プログラムにおけるホストとデバイス間のデータ転送。
各矢印は、ホストとデバイスのメモリー間のデータ転送を示します。

Linux* でインテル® Fortran コンパイラーと OpenMP* ターゲットオフロードを使用してサンプルプログラムをコンパイルするコマンドを次に示します。

```
$ ifx -xhost -qopenmp -fopenmp-targets=spir64 source_file.f90
```

## OpenMP* ターゲット領域から既存の API を使用する

OpenMP* ターゲット領域から外部関数を呼び出す方法は、前号の記事、「Fortran、oneMKL、OpenMP* を使用して LU 因数分解を高速化」で説明しました。`DISPATCH` ディレクティブで、関連するサブルーチンまたは関数呼び出しの周囲で条件付きディスパッチ・コードを出力するようにコンパイラーに指示します。

```
!$OMP TARGET DATA
!$OMP DISPATCH
CALL external_subroutine_on_device
!$OMP END TARGET DATA
```

ターゲットデバイスが利用可能な場合、構造化ブロックのバリアントバージョンが呼び出されます。

## インテル® Fortran のサポート

インテル® Fortran コンパイラー（`ifx`）は、インテル® Fortran コンパイラー・クラシック（`ifort`）のフロントエンドとランタイム・ライブラリーをベースに、LLVM バックエンドを使用した新しいコンパイラーです。詳細は、『インテル® Fortran コンパイラー・クラシックおよびインテル® Fortran コンパイラー・デベロッパー・ガイドおよびリファレンス』（英語）を参照してください。バイナリー（`.o`/`.obj`）およびモジュール（`.mod`）は `ifort` と互換性があり、さまざまな Fortran 標準規格（95、2003、2018）と OpenMP* を利用したヘテロジニアス・コンピューティングをサポートしています。Fortran を利用したヘテロジニアス並列処理の別のアプローチ（英語）は、`DO CONCURRENT` ループです。

```
PROGRAM test_auto_offload
   INTEGER, PARAMETER :: N = 100000
   REAL               :: a(N), b(N), c(N), sumc

   a = 1.0
   b = 2.0
   c = 0.0
   sumc = 0.0

   CALL add_vec

   DO i = 1, N
      sumc = sumc + c(i)
   ENDDO

   PRINT *,' sumc = 300,000 =', sumc

   CONTAINS
      SUBROUTINE add_vec
         DO CONCURRENT (i = 1:N)
            c(i) = a(i) + b(i)
         ENDDO
      END SUBROUTINE add_vec

END PROGRAM test_auto_offload
```

このコードを次のようにコンパイルすると、OpenMP* ランタイム・ライブラリーがデバイス・カーネル・コードを生成します。

```
$ ifx -xhost -qopenmp -fopenmp-targets=spir64 -fopenmp-target-do-concurrent source_file.f90
```

`-fopenmp-target-do-concurrent` オプションは、`DO CONCURRENT` のデバイスカーネルを自動的に生成するようにコンパイラーに指示します［**翻訳者注：**このコードには OpenMP* のディレクティブはありませんが、インテル® Fortran コンパイラーは `DO CONCURRENT` ループの並列処理に OpenMP* ランタイムを使用するため、`-qopenmp` と `-fopenmp-targets=spir64` オプションが必要です］。

次の環境変数を設定することにより、OpenMP* ランタイムは、カーネル・アクティビティーのプロファイルを提供できます。

```
$ export LIBOMPTARGET_PLUGIN_PROFILE=T
```

実行ファイルを実行すると、出力が得られます。

プログラムを実行したときの出力で、サブルーチン名 `add vec` を探します。

```
Kernel 0 : __omp_offloading_3b_dd004710_test_auto_offload_IP_add_vec__l10
```

Fortran 言語委員会は、Fortran 2023 標準規格に `DO CONCURRENT` のリダクションを追加する提案に取り組んでいます。

```
DO CONCURRENT(i = 1:N) REDUCE(+:sum)
```

## 最後に

Fortran と OpenMP* を使用したヘテロジニアス並列プログラミングの概要を説明しました。上記のコード例から分かるように、OpenMP* は、一般にほかへの影響がない並列処理を表現するための記述的なアプローチです。つまり、OpenMP* ディレクティブが有効でない場合でも、シーケンシャル・プログラムはそのまま残ります。このコードは、アクセラレーター・デバイスがない、ホモジニアス・プラットフォームでも動作します。Fortran + OpenMP* は、ヘテロジニアス並列処理のための、強力で、オープンな、標準ベースのアプローチです。