OpenACC* API から OpenMP* API への移行

Harald Servat、Shiquan Su、および Tobias Kloeffel インテル コーポレーション
Ron Caplan Predictive Science, Inc.
Junyi Cheng コロラド大学ボルダー校

はじめに

OpenACC* と OpenMP* はどちらもアクセラレーターへのオフロードをサポートしています。OpenACC* が早期に利用可能になったことは計算ワークロードに GPU デバイスを使用するユーザーにとって良いニュースと言えますが、OpenACC* がサポートするデバイスベンダーはまだ限られています。現在は OpenMP* オフロードをサポートするアクセラレーター・ベンダーの数が増えていることから、アクセラレーターへのオフロードに OpenMP* を採用する需要が高まっています。

OpenACC* から OpenMP* API へのインテル® アプリケーション移行ツール（英語）は、OpenACC* ベースのアプリケーションの OpenMP* への移行を支援するオープンソース・プロジェクトです（IWOMP の論文（英語）を参照）。このツールは、C/C++ および Fortran アプリケーションのソースを解析し、OpenACC* 構造を識別して、可能な場合は意味的に同等の OpenMP*（5.0 以降）構造を提案します。構造間の記述的 / 規範的な違いを含む、さまざまな理由により、すべての OpenACC* 構造を OpenMP* に変換できるわけではありません。

特に、パフォーマンス・チューニングは考慮されないので注意が必要です。このツールは意味的に正しい移行コードを提供することのみに焦点を当てており、パフォーマンスが最適化されたバージョンの移行コードを提供することには焦点を当てていないため、パフォーマンスの最適化は後の段階で行うことになります。OpenACC* 構造を変換できない場合、ツールは移行レポートにメッセージを生成します。2 つの言語間のマッピングの詳細は、IWOMP の論文で説明されています。アプリケーションのソースが移行されたら、開発者は OpenMP* 5.0 準拠のコンパイラーを選択し、ターゲット・アーキテクチャー向けにアプリケーションをコンパイルするだけです。

次のセクションで、POT3D と GEM という 2 つの移行の成功事例について説明します。その後、いくつかのコメントを述べて、この記事のまとめとします。

アプリケーション移行の例

移行ツールを実証するため、2 つのアプリケーション、POT3D と GEM を選択しました。詳細は、**表 1** を参照してください。これらのアプリケーションを移行した後、インテル® HPC ツールキット 2024.0.1 のコンパイラーとライブラリーを使用してコンパイルしました。両方のアプリケーションを、2 ソケットのインテル® Xeon® Platinum 8480+ プロセッサーと 4 つのインテル® データセンター GPU Max 1550 を含む単一ノード上で実行しました。各 GPU は 2 つの計算タイルで構成され、ノードの計算タイルは合計 8 つになります。対応するリポジトリーで提供されているリファレンス出力と比較して実行結果を検証しました。

			OpenACC* 構造 / 節の数				
アプリケーション	**言語**	**行数**	計算	データ	アトミック	非同期	API 呼び出し
POT3D	Fortran	~11k	33	33	2	23	0
GEM	Fortran	~18k	34	69	118	0	1

表 1. アプリケーションの特性

注：
対象となる計算構造：`kernels`、`loop`、`parallel` および `loop`。
対象となるデータ構造：`enter/exit data`、`host_data`、および update。
対象となるアトミック構造：`atomic`。
対象となる非同期構造：`async` および `wait`。

POT3D

POT3D（英語）（commit-id: 5e8ee69f92860a372d5746462199ddfa702bbac2）は、観測された光球磁場を境界条件として使用して太陽コロナ磁場を近似するポテンシャル場の解を計算する Fortran コードです。コードは MPI を使用して並列化されますが、GPU アクセラレーションには OpenACC* と Fortran 標準 `do concurrent` 構文の 2 つの代替手段があります。この記事では、OpenACC* バージョンを使用します。

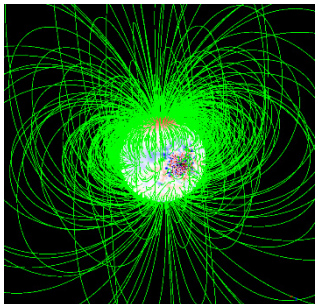

図 1. POT3D はポテンシャル場ソリューションにより磁力線をトレースします。表面の放射状の磁場が赤と青のカラーマップとして示されています。

次のコマンドを実行してコードを変換しました。

```
${PATH_TO_TRANSLATOR}/intel-application-migration-tool-for-openacc-to-openmp src -async=none
```

ここで、`src` はアプリケーションのソース・ディレクトリーを指します。`-async=none` はソースコード内に存在する非同期 OpenACC* ステートメント（`wait` および `async` を含む）を無視してコードを変換します。OpenMP* は、これらのステートメントの正確なマッピングを提供しません。幸いなことに、開発者によれば、非同期はこのアプリケーションにほとんど利益をもたらしません。

移行に関する注意事項

移行レポートに表示されるコメントと警告のうち、アプリケーションが `!$acc set device_num(iprocsh)` を使って、使用する OpenACC* デバイスを指定していることに注目してください。OpenMP* は、この動作を模倣する API 呼び出し（`omp_set_default_device`）と環境変数（`OMP_DEFAULT_DEVICE`）を提供します。インテル® MPI は、MPI ランクを GPU デバイス（またはタイル）にピニングする環境変数（`I_MPI_OFFLOAD_PIN`）も提供します。そのため、MPI ランク（GPU タイルのバインド）に関しては、構造の変換を無視して、インテル® MPI のインフラストラクチャーに依存しました。

変換に関して言えば、多くのデータ割り当てに `!$acc data create` という注釈が付いていることが分かります。これは、対応するデータをアクセラレーターのアドレス空間に割り当てる指示であり、`!$omp target enter data` を使用して変換されます。最も時間のかかる領域（`cgsolve`）には、`!$omp target teams loop` に変換された `!$acc parallel loop` 節が含まれていることも分かりました。興味深いことに、元の構造では `present(X)` 節または `default(present)` 節を明示的に使用して、オフロードランタイムでアクセラレーターのアドレス空間に指定された変数が存在するかどうかをチェックしています。これらの節はそれぞれ、`present` マップタイプ修飾子または `defaultmap` 節を使用して `map` に変換されます。ホストとアクセラレーターのアドレス空間の内容を同期するため、境界交換や I/O の実行前後に `!$acc update` などの節もあります。

対応する Makefile は、IFX 向けの MPI コンパイラー・ラッパー（`mpiifx`）を使用するように手動で調整され、OpenMP* オフロード構造（`-fiopenmp -fopenmp-targets=spir64`）と HDF5 の依存関係を処理するために必要なオプションが追加されました。

アプリケーションの実行に関しては、アプリケーション・リポジトリーに含まれる small テストを使用し、環境変数 `I_MPI_ADJUST_BARRIER=1 I_MPI_OFFLOAD_PIN=1 I_MPI_OFFLOAD_PRINT_TOPOLOGY=1 I_MPI_DEBUG=3` を設定してバイナリーを（`mpirun` 経由で）呼び出しました。達成されたパフォーマンスを**図 2** に示します。アプリケーションは 4 MPI ランク（各 GPU で 1 つのコンピューティング・タイルを使用）までは理想に近いスケーラビリティーを示しますが、8 MPI ランク（各 GPU で 2 つの計算タイルを使用）になるとパフォーマンスが低下します。このパフォーマンス低下の根本的な理由を考察するため、アプリケーション開発者といくつかのアイデアを議論しました。

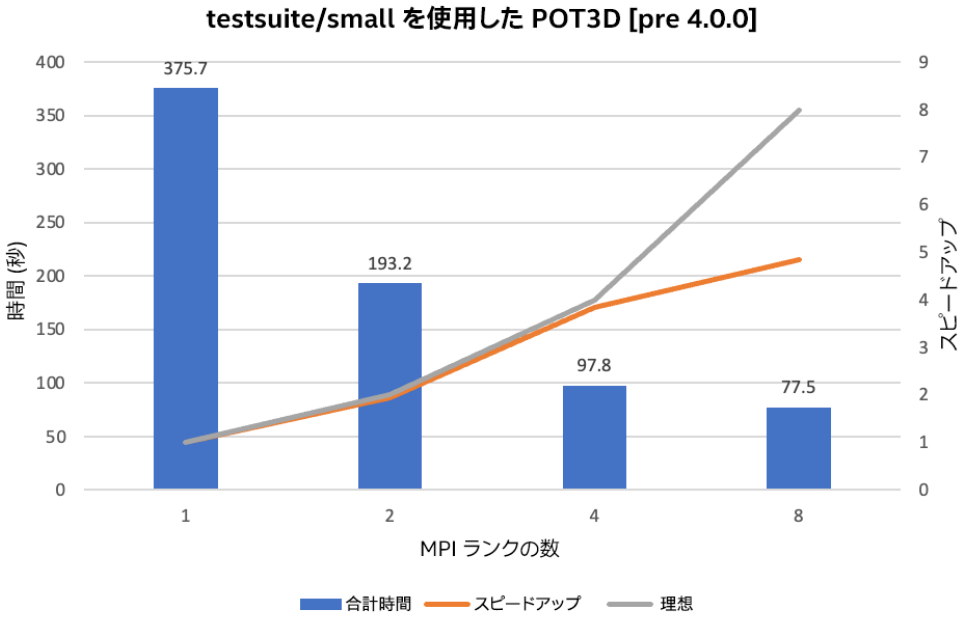

図 2. ターゲットシステム上で POT3D を実行したときに達成されたパフォーマンス

GEM

GEM（英語）は、1990 年代から Fortran で記述された、物理場効果を考慮したオープンソースの Particle-In-Cell（PIC）法のシミュレーション・コードです。ITER（英語）のような、磁気的に閉じ込められた核融合装置における輸送を研究することを目的としています。

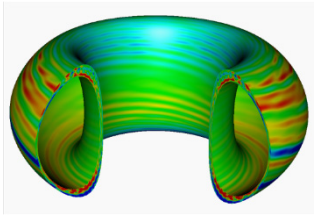

図 3. GEM は、磁気平衡でドーナツ状に移動するプラズマ粒子をシミュレートします。

GEM には時間のかかるタスクが 2 つあります。

1. 入れ子のループ（リダクションあり / なし）
2. 物理場の行列ソルバー

どちらも GPU で高速化できる可能性がありますが、開発者はコードの保守性を高める標準 LAPACK を使用してソルバーを実装することを好むため、高速化に OpenACC* を使用していたのは入れ子のループのみです。まず、移行ツールを利用して OpenACC* 構造を OpenMP* に変換します。次に、`!$omp dispatch` 構造により並列かつ高速な LAPACK 実装を提供する oneAPI マス・カーネル・ライブラリー（oneMKL）を使用するように、変換されたコードを手動で変更します。後者の変換はこの記事では取り上げません。

アプリケーションのファイル構造は単純で、すべてのソースファイルが同じフォルダーに含まれています。コードを変換するコマンドは次のとおりです。

```
${PATH_TO_TRANSLATOR}/intel-application-migration-tool-for-openacc-to-openmp *.f90
```

変換後、IFX Fortran コンパイラーで OpenMP* オフロード機能を有効にするため、OpenACC* のコンパイラー・オプションを `-DOPENACC2OPENMP_ORIGINAL_OPENMP -fiopenmp -fopenmp-targets=spir64` に置換しました。オリジナルコードでは、GPU に送るには小さすぎるいくつかのループを OpenACC* が処理するときに複数のスレッドを起動する OpenMP* 構文も適用しているため、`-DOPENACC2OPENMP_ORIGINAL_OPENMP` コンパイラー・オプションを追加しました。これらは移行ツールにより保持されますが、このプリプロセッサー変数で保護されます。

移行に関する注意事項

移行中に見つかった特異点を述べておきます。

1. データ管理
2. ループ構造での並列処理

データ管理

GPU と CPU 間のデータ移動に関して、GEM は非構造化データアプローチを使用します。このアプローチにより、コードがホストメモリー上にデータを割り当てた（または割り当て解除した）直後に、ホストとデバイス間のデータマッピングを確立する柔軟性が提供されます（データ割り当て時に `!$acc enter data create(X)` 構造を使用します）。この構造は、移行ツールにより `!$omp target enter data(X)` として変換されます。

ほかのアプリケーション・モジュールがデバイスデータにアクセスする必要がある場合、OpenACC* の `present(X)` 節に依存します。この節はマップタイプ修飾子（`map(present,alloc:X)`）に変換されます。見た目は同じではありませんが、この 2 つは同じセマンティクスを表現します。ただし、OpenMP* では、割り当てが行われない場合でも、存在チェックをマップタイプ（この場合は `alloc`）で行う必要があります。

さらに、アプリケーション・コードは `acc_is_present(X)` API を利用して、GPU アドレス空間にデータが存在しない潜在的なケースを識別します。この API ルーチンは移行ツールで変換されず、ツールはこの誤変換に関する警告を生成します。

ループ構造での並列処理

`!$acc loop` で高速化される 2 つの主な並列領域を特定します。1 つ目の領域は、`!$acc atomic` 構造を使用していくつかのリダクションを実装します。これらの実装は対応する OpenMP* に変換されます。2 つ目の領域は、以下のコードで表されます。内部に `control-flow (if)` 構造があることに注意してください。このループには入れ子のループ構造が含まれていますが、完全な入れ子のループではないため、`collapse` 節を使用することはできません。

```
!$acc parallel present(xp,s_buf,ipsend,s_displ,s_counts)
!$omp target teams map(present,alloc:xp,s_buf,ipsend,s_displ,s_counts)
!$acc loop independent  private(i,isrt,iend,isbuf)
!$omp loop order(concurrent) private(i,isrt,iend,isbuf)
    DO i=0,nvp-1
       IF( s_counts(i) .GT. 0 ) THEN
          isrt = s_displ(i)+1
          iend = s_displ(i)+s_counts(i)
          !$acc loop independent
          !$omp loop order(concurrent)
          DO isbuf=isrt,iend
             s_buf(isbuf) = xp(ipsend(isbuf))
          END DO
          !$acc end loop
          !$omp end loop
       END IF
    END DO
!$omp end loop
!$acc end parallel
!$omp end target teams
```

移行の結果として得られる最初の OpenMP* 実装は、いくつかの手動の変更を加えてコンパイルして実行できます。ランクごとに 1 つの GPU タイルを使用して 4 MPI ランクで実行した場合のパフォーマンスを**図 4** に示します。この結果は開発者の予想と一致しています。

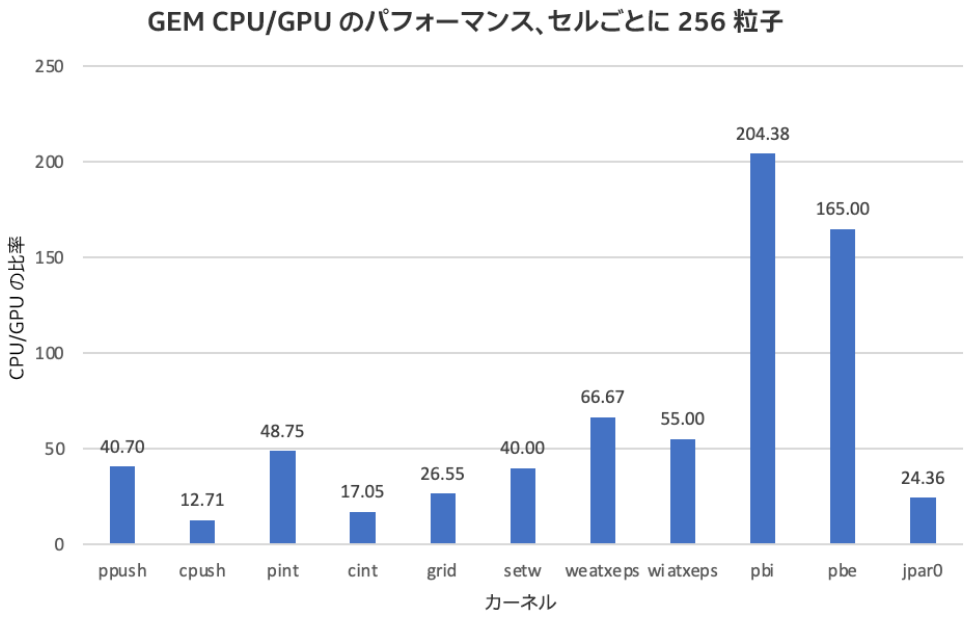

図 4. 主要なオフロードされたカーネルでの GEM アプリケーションの CPU と GPU の比率

まとめ

OpenACC* から OpenMP* API へのインテル® アプリケーション移行ツールは、OpenACC* コードの OpenMP* への移行に役立つこと、OpenMP* 5.0 をサポートしている場合、ユーザーはさまざまなハードウェア / ソフトウェア上でこれらのコードを実行できることを示しました。2 つのプログラミング・モデル間の暗黙的な違いにより、すべての OpenACC* アプリケーションを完璧に変換することはできませんが、このツールは移行作業の優れた開始点となるでしょう。現在、GEM 開発者は OpenACC* 移行済みコードのリポジトリー・フォーク（英語）を作成しています。この記事の公開時点で、リポジトリーに変更を加えることについて、POT3D 開発者と話し合いを行っています。

可能性のある将来の機能として、ユーザーのフィードバックに基づいて、対応する OpenMP*（存在する場合）への OpenACC* API 呼び出しの変換を追加しています。また、OpenACC* と OpenMP* の異なる非同期メカニズムに代わるマッピングを提供することにも取り組んでいます。最後に、オリジナルコード内の CPU OpenMP* ディレクティブを特定するためのいくつかの取り組みが行われており、CPU と GPU の両方をサポートする変換に OpenMP* メタディレクティブを使用することを検討しています。パフォーマンスの点では、多種多様なハードウェアとソフトウェアが利用可能であるため、パフォーマンスが最も高い変換を保証することはできません。パフォーマンス・ツールを使用してソースコードをチューニングすることを強く推奨します。