インテル® DPC++ 互換性ツールのベスト・プラクティス

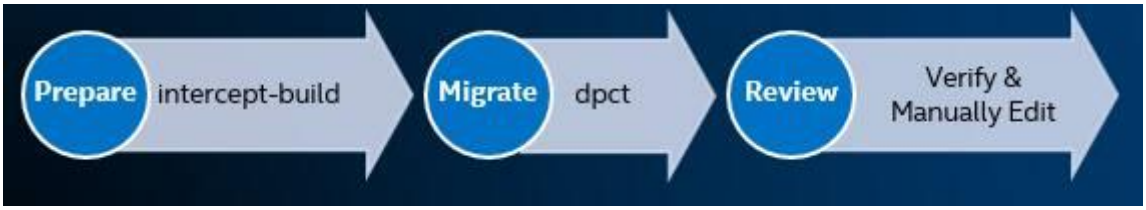
この記事は、インテル® デベロッパー・ゾーンに公開されている「Intel® DPC++ Compatibility Tool Best Practices」の日本語参考訳です。原文は更新される可能性があります。原文と翻訳文の内容が異なる場合は原文を優先してください。

このドキュメントでは、インテル® DPC++ 互換性ツール (インテル® DPCT) を使用して、CUDA* コードをデータ並列 C++ (DPC++) コードへ移行する際のヒントやコツ、一般的なベスト・プラクティスを紹介します。

このドキュメントは、「インテル® DPC++ 互換性ツール」(英語) と「インテル® DPC++ 互換性ツール・ユーザーガイド」(英語) を補完します。特に明記されている場合を除いて、ここで使用するサンプルコードは MIT ライセンスの下で提供され、説明目的で掲載されています。

移行ワークフローの概要

ほとんどの場合、インテル® DPC++ 互換性ツールを使用した CUDA* ソースコードから DPC++ コードへの移行は、準備、移行、レビューの 3 ステージに分けることができます。



準備ステージでは、プロジェクト・ディレクトリーを整理し、コンパイルオプションに注意し、場合によってはソースファイルの変更が必要になります。ほとんどの Makefile ベースのプロジェクトでは、コンパイルコマンド、コンパイルフラグ、オプションを自動的に追跡し、JSON ファイルに保存する `intercept-build` スクリプトを実行することを推奨します。Microsoft* Visual Studio* プロジェクトの場合、`.vcxproj` ファイルが存在することを確認し、これを `dpct` 移行ツールに渡すことで、プロジェクトのオプションを追跡できます。単純なプロジェクトの場合、コンパイルオプションとマクロは、`dpct` の実行時に手動で指定できます。コマンドラインで `intercept-build` を実行する場合、ビルドコマンドを指定します。

```
intercept-build make
```

移行ステージでは、インテル® DPC++ 互換性ツールの実行ファイルである `dpct` が実行されます。入力としてオリジナルのアプリケーションを受け取り、そのヘッダとソース、および生成された `compile_commands.json` があればそれを解析し、DPC++ のコードとレポートを出力します。

```
dpct -p ./ --in-root=./ --out-root=output *.cu
```

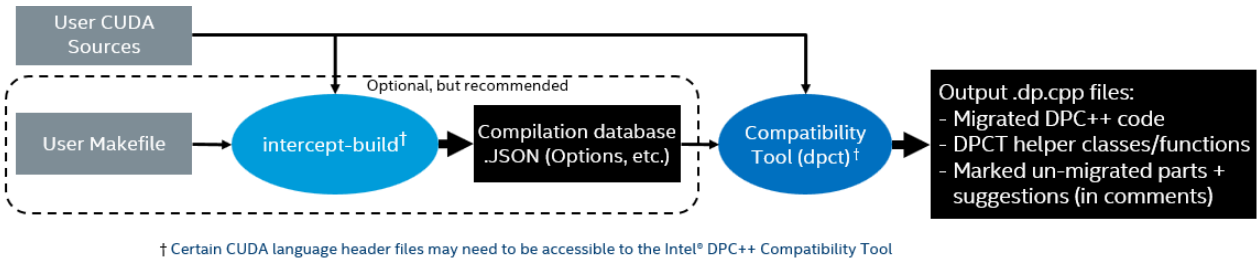
`intercept-build` が実行されなかった場合、コンパイルオプションは `dpct` の引数として手動で指定することもできます。

```
dpct --out-root=output source.cu -extra-arg="-I./include" --extra-arg="-
DBUILD_CUDA"
```

インテル® DPC++ 互換性ツールは、Microsoft* Visual Studio* または Eclipse* IDE 内で呼び出すことも可能です。最終レビューステージでは、手動で検証して編集する必要があります。インテル® DPC++ 互換性ツールで移行できない部分については、ユーザーが移行したコードを修正し、正当性を確認する必要があります。手作業が必要なコード部分については、簡単に見つけられるように、インテル® DPCT メッセージが移行されたソースファイルにコメントとして記録されます。以下の `dpct` の出力例では、オリジナルの CUDA* 呼び出し `cudaMemcpy` (`DPCT_ORIG` コメントで示される) が DPC++ に移行されています。しかし、DPC++ が例外を使用してエラーを処理するのに対し、CUDA* はエラーコードを使用するため、`dpct` ツールはコメントにメッセージ DPCT1003 を追加して、手動で追加の編集が必要であることを示しています。

```
/* DPCT_ORIG status = cudaMemcpy(Result, d_C, dcsize,
* cudaMemcpyDeviceToHost);*/
/*
DPCT1003:0: Migrated API does not return error code. (*, 0) is
inserted. You may need to rewrite this code.
*/
status = (q_ct1.memcpy(Result, d_C, dcsize).wait(), 0);
```

次の図は、インテル® DPC++ 互換性ツールを使用した場合のワークフローと生成されるファイルを示します。



ベスト・プラクティスの準備

インテル® DPC++ 互換性ツールを実行する前に、次の操作を行うことを推奨します。

プロジェクトのソースファイルが構文的に正しいことを確認する

インテル® DPC++ 互換性ツールは Clang を使用して CUDA* ソースを解析するため、オリジナルソースに構文エラーが含まれる場合、移行が成功しない可能性が高くなります。

「intercept-build make」を実行する前に「make clean」を実行する

intercept-build スクリプトを実行する前に make clean を実行して不要なファイルを削除します。これにより、干渉を受けずにコンパイル用データベースを作成できます。

複雑なプロジェクトの場合 intercept-build コマンドを使用してコンパイル・データベースを作成する

Make や CMake* を使用するプロジェクトの場合、インテル® DPC++ 互換性ツールを実行する前にコンパイルオプション、設定、マクロ定義、およびインクルード・パスを追跡するのは困難です。intercept-build <build command> で、インテル® DPC++ 互換性ツールが使用するビルドコマンドを含む JSON ファイル形式のコンパイル・データベースを自動生成します。

Clang と nvcc の違いにより移行前にコード変更が必要な場合

`dpct` が使用する Clang パーサーは `nvcc` と必ずしも互換性がないため、移行前に CUDA* ソースの手動編集が必要な場合があります。次に例を示します。

1. Clang パーサーでは特定の使用シナリオで名前空間修飾が必要になることがありますが、`nvcc` では必要ありません。
2. Clang パーサーでは追加の前方クラス宣言が必要になることがありますが、nvcc では必要ありません。
3. カーネル呼び出しの三重括弧内のスペースは、`nvcc` では許容されますが、Clang では許容されません。例えば、`cuda_kernel<< <num_blocks, threads_per_block>> >(args…)` は nvcc では問題ありませんが、Clang パーサーではこれらのスペースを削除する必要があります。

Clang と nvcc の方言の違いについては、llvm.org の「clang を使用した CUDA* のコンパイル」(英語) を参照してください。

移行のベスト・プラクティス

`dpct` 実行ファイルを使用して CUDA* プロジェクトを DPC++ へ移行する場合、多くのコマンドライン・オプションを利用できます。利用可能なコマンドライン・オプションの一覧は、『インテル® DPC++ 互換性ツール・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。以下は、さまざまなシナリオで使用可能な便利なオプションです。

プロジェクトのソースファイルを一度に移行できない場合、1 ファイルずつ段階的に移行することが有用な場合があります。

インテル® DPCT の基本オプション	
--in-root	移行するソースツリーのルートへのパス。
--out-root	生成されるファイルのルートへのパス。
-p	コンパイル・データベース JSON ファイルへのパス。
--process-all	--in-root ディレクトリーから --out-root ディレクトリーへすべてのファイルを移行/コピーして、.cu ファイルを 1 つずつ指定する必要を省きます。
--extra-arg	Clang コンパイラー・オプションを指定します。 例: dpct --extra-arg="-std=c++14" –extra-arg="-I…"
--format-style	出力ファイルの書式を設定します。 例: =llvm、=google、=custom (.clang-format ファイルを使用)
--format-range	コードに書式設定を適用します: 適用しない (=none)、移行したコードに適用する (=migrated)、またはすべてのコードに適用 (=all) する。

以下は、移行とデバッグを容易にする推奨オプションです。

移行/デバッグを容易にするインテル® DPCT オプション	
--keep-original-code	オリジナルの CUDA* コードを生成した DPC++ ファイルのコメントに保持します。 オリジナルの CUDA* コードと生成した DPC++ コードを簡単に比較できるようにします。
--comments	生成したコードを説明するコメントを挿入します。
---always-use-async-handler	常に非同期例外ハンドラーで cl::sycl::queue を作成します。

統合共有メモリー (USM) の使用

DPC++ でサポートされる統合共有メモリー (USM) は、ポインターベースのアプローチでホストとデバイスのメモリーを管理できる機能です。インテル® DPC++ 互換性ツールを使用すると、移行した DPC++ コードはデフォルトのメモリー管理方法として USM を使用します。SYCL* バッファーを使用する場合と比較すると、USM はコード量が少なく、`dpct` はより多くのメモリー関連の API をサポートできます。

しかし、一部のコンパイラー、特にインテル以外のハードウェアをターゲットとするインテル以外のコンパイラーでは、USM をうまく処理できず、浮動小数点例外や不正なメモリーアクセスなどのランタイムエラーが発生する可能性があります。これらのエラーが発生した場合は、回避策として「dpct --usm-level=none」を使用します。

インテル® DPCT USM オプション	
--usm-level	統合共有メモリー (USM) のレベルを設定します。 =Restricted: USM を使用する (デフォルト) =none: ヘルパー関数と SYCL* バッファーを使用する

インテル® DPCT ヘルパー関数

インテル® DPC++ 互換性ツールは、移行した DPC++ コードでヘルパー関数とクラスを使用します。ユーティリティー関数の例として、メモリー管理タスク用の `dpct_malloc`、`dpct_memcpy`、`get_buffer` や、デバイス管理タスク用の `get_default_queue`、`get_default_context` があります。関連ファイルは `<dpcpp-ct installation directory>/latest/include/dpct` にあります。メイン・ヘッダー・ファイルは `dpct.hpp` で、名前空間は `dpct::` です。

これらのインテル® DPCT ヘルパー関数は、移行したコードを対象としており、その他の目的には使用できません。新しい DPC++ コードを記述する場合、これらのインテル® DPCT ヘルパーを使用することは推奨されません。

レビューと編集のベスト・プラクティス

`dpct` を実行後、移行した DPC++ コードをコンパイルする前に、通常手動で編集が必要になります。ほとんどの場合、出力ファイルには残りのコードの移行に役立つヒントやコメントが含まれています。これらのコメントを確認し、移行したコードが論理的に相反しないように変更します。『インテル® DPC++ 互換性ツール・デベ

ロッパー・ガイドおよびリファレンス』の「診断リファレンス」(英語) で、問題を修正するための詳細なヘルプや推奨事項を含むコメントの説明が得られます。

タイミングの問題

時間の計算は実装に依存するため、タイミング関連のブロックは手動での編集が必要な場合があります。言語固有の機能やライブラリーへの依存を含むコードを書き換えて、同等の効果が得られるようにします。

潜在的な問題の例として、プロファイル関連のタイマーの呼び出しがあります。`sdkCreateTimer` や `sdkStartTimer` などの CUDA* サンプルのタイマー関数を使用している場合、これらの呼び出しを実装し直す必要があります。

一般的なランタイムの問題

コードを DPC++ に移行して、コンパイルの問題を修正した後、ランタイムの問題が発生する場合があります、ここでは、一般的なエラーとその対処法を紹介します。

「OpenCL API failed. OpenCL API returns: -52 (CL_INVALID_KERNEL_ARGS)」: このエラーは、デバイス上で実行する前に一部のポインターが適切に設定されていないことを示します。parallel_for デバイス実行で使用する前に、すべてのポインターが有効なメモリー割り当てまたは NULL に初期化されていることを確認してください。

「OpenCL API failed. OpenCL API returns: -54 (CL_INVALID_WORKGROUP_SIZE)」: アクセラレーターに応じて、ハードウェアの違いにより、ワークグループの最大ワークアイテム数が制限されます。例えば、NVIDIA* ハードウェアではワークグループのサイズが 512 に制限されることが多く、第 9 世代インテル® グラフィックでは 256 に制限されます。このエラーが発生した場合、カーネル起動時に設定されるワークグループ・サイズをハードウェアの制限に応じて調整する必要があります。

「Caught asynchronous SYCL exception」: エラーメッセージに従って、エラーが発生した場所を確認し、必要な変更を行います。

まとめ

インテル® DPC++ 互換性ツールは、CUDA* コードから DPC++ へ移行を効率良く支援し、コードの移行時間を大幅に短縮します。インテル® DPC++ 互換性ツールを使用する 3 つのステージで、ここで紹介したベスト・プラクティスに従って、発生する可能性がある問題に素早く対応してください。

関連情報

- インテル® DevCloud (英語) で DPC++ と oneAPI 製品を評価
- インテル® oneAPI 製品ページ
- インテル® oneAPI プログラミング・ガイド
- インテル® DPC++ 互換性ツール導入ガイド (英語)

製品および性能に関する情報

[1] 性能は、使用状況、構成、その他の要因によって異なります。詳細については、www.Intel.com/PerformanceIndex/ (英語) を参照してください。