CUDA* と oneAPI 間の変換を可能にする SYCL* の相互運用性を深掘り

この記事は、インテルのサイトで公開されている「SYCL* Interoperability: A Deep Dive into Bridging CUDA* and oneAPI」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。

インテル® DPC++ 互換性ツール (英語) または SYCLomatic (英語) を使用して、CUDA* コードを C++ with SYCL* に移行すると、既存の CUDA* ベースのアクセラレーテッド・コンピューティング・ワークロードを、ベンダーに依存しない、幅広いマルチアーキテクチャー・プラットフォームで実行できるようになります。

ただし、移行ツールは、コードベースを移行する最初のステップを支援するに過ぎません。通常、移行ツールを実行すると、解決すべきいくつかの課題が残ります。開発者は、移行中や移行後に発生した警告やエラーに対処するため、SYCLomatic の実行後に回避策を特定して適用する必要があります。

また、CUDA* と SYCL* の API アーキテクチャーの違いにより、ワークロードの一部でパフォーマンスに影響が生じる可能性があります。これは、理解して解決したほうが良いでしょう。

CUDA* から SYCL* へ移行する主な目的は、異なるプラットフォーム構成にわたるソフトウェアの移植性です。パフォーマンスの移植性も目標の 1 つであるため、移行を完了する上でこれを確保することが重要です。

ただし、一部の CUDA* API では、SYCL* API および関連する oneAPI エコシステム・ライブラリー・ソリューションに直接一致するものが明確ではない場合があります。

SYCL* とベンダー固有のバックエンド間の変換メカニズムを想像してみてください。

このブログでは、SYCL* の相互運用性がどのようにして変換を可能し、oneAPI エコシステムがさまざまな API 間のギャップを埋めてシームレスなソフトウェアの移行を実現する方法について説明します。

課題: 移行されない CUDA* API

最も一般的で複雑な課題の 1 つは、移行されない CUDA* API への対応策を見つけることです。多くの場合、コンパイルと検証のため、SYCL* と CUDA* でサポートされる機能の違いを考慮した代替ロジックの再設計/書き換えが必要になります。

ただし、成熟した CUDA* 環境から、急速に進化する新しい oneAPI エコシステムに移行する場合、特に、CUDA* と oneAPI エコシステムの機能に大きな違いがあると、パフォーマンス、コーディング・セマンティクス、および 2 つのエコシステムのパラダイムにおいて、困難な作業が生じることがあります。

一例として、cuSparse API を使用する CUDA* アプリケーションの移行が挙げられます。この API は、NVIDIA* プラットフォーム上で実行できる完全な代替 SYCL* API がまだありません。

しかし、SYCL* が提供する相互運用機能により、これは移行の障壁にはなりません。

相互運用性: SYCL* 内からの CUDA* API 呼び出し

インテルは SYCL* 作業部会と協力して、Codeplay などの企業からの貢献により、SYCL* 仕様の強化に取り組んできました。SYCL* の際立った特徴の 1 つは、ベンダーへの非依存性を重視していることです。

これにより、SYCL* コード内から CUDA* API を直接呼び出せる相互運用機能を SYCL* に導入する道が開かれました。これは理論上の概念ではなく、NVIDIA* および AMD* プラットフォームをサポートするため oneMKL や oneDNN などの oneAPI ライブラリーで使用されている実用的で成熟したソリューションです。

また、開発者はパフォーマンスを低下させることなく、oneAPI フレームワーク内から CUDA* の確立された機能を利用できるため、パフォーマンスの観点からも革新的です。負荷の高い作業、複雑な変換、調整はすべて、LLVM ベースのインテル® oneAPI DPC++/C++ コンパイラーによってバックグラウンドで適切に管理されます。

SYCL* ランタイムクラスは、ハードウェア・ベンダーの SYCL* バックエンドに固有のオブジェクトをカプセル化できます。ハードウェア固有のバックエンドはそのクラスの機能を実現するため、SYCL* ランタイム・オブジェクトとネイティブ・バックエンド・オブジェクト (CUDA*/HIP) 間のインターフェイスを提供します。これにより、アプリケーション内から SYCL* と関連する SYCL* バックエンド API 間の相互運用性を可能にします。

したがって、SYCL* の相互運用性は、SYCL* とバックエンド (CUDA* など) 間の変換メカニズムを提供し、ユーザーが SYCL* オブジェクト (キュー、デバイス、コンテキスト、メモリーなど) を通じて基礎となるバックエンドのオブジェクトにアクセスして操作し、関連するプラットフォーム (NVIDIA*、AMD など) 上でネイティブ・バックエンド API (ランタイム、ドライバー、ライブラリー) を実行できるようにします。

SYCL* の相互運用性の使用例

SYCL* の相互運用性は、すべての SYCL* オブジェクト (コンテキスト、デバイス、メモリー、キューなど) のネイティブ・バックエンド・オブジェクトを派生する一連の **get_native** フリー関数を提供します。

さらに、SYCL* の **host_task** クラスと **interop_handle** クラスを使用して、SYCL* タスクグラフ内からバックエンド固有の操作を実行することもできます。これらは、プラットフォーム/デバイスの基礎となるネイティブ・オブジェクトを利用できるようにします。**host_task** の実行は、SYCL* ランタイムによってスケジュールされます。

SYCL* の相互運用性は、部分的に移行された SYCL* アプリケーションで容易に使用できます。

1. 移行されない CUDA* API のすべてのパラメーターをリストします。
2. SYCLomatic によって部分的に移行された値渡しパラメーター (列挙型など) の代わりに、CUDA* と同等のものを直接使用します。
3. デバイスメモリーを使用するほかのパラメーターについては、**get_native** 関数または **interop_handle** を使用して、SYCL* によって管理されるメモリー空間のネイティブ CUDA* オブジェクトを取得します。

以下は、CUDA* バックエンドをサポートする同等の oneAPI ライブラリー API がまだ存在しない cuSparse API を使用して、行列ベクトル乗算を行う単純な CUDA* アプリケーション (完全なソース コードはこちら (英語)) の移行で SYCL* の相互運用性を使用するコードの一部です。

spMV.cu

```
#include <iostream>
#include <cuda_runtime.h>
#include <cusparse.h>

int main() {

    // cuSPARSE ライブラリーの初期化
    ...

    // メモリー管理
    ...

    // 行列記述子とベクトル記述子の作成
    cusparseCreateCsr(&matA, 3, 3, 4, d_csrRowPtr, d_csrColInd, d_csrVal,
CUSPARSE_INDEX_32I, CUSPARSE_INDEX_32I, CUSPARSE_INDEX_BASE_ZERO, CUDA_R_32F);
    cusparseCreateDnVec(&vecX, 3, d_x, CUDA_R_32F);
    cusparseCreateDnVec(&vecY, 3, d_y, CUDA_R_32F);

    float alpha = 1.0f;
    float beta = 0.0f;

    // バッファーの作成と割り当て
    ...

    // 行列-ベクトル乗算の実行: y = A*x
    cusparseSpMV(handle, CUSPARSE_OPERATION_NON_TRANSPOSE, &alpha, matA, vecX,
&beta, vecY, CUDA_R_32F, CUSPARSE_MV_ALG_DEFAULT, dBuffer);

    // 結果をホストにコピーバック
    ...

    // クリーンアップ
    ...
```

```
    return 0;
}
```

SYCLomatic を使用してこのアプリケーションを移行すると、次のコードが生成され、cuSparse CUDA* API の移行がサポートされていないことを示す **DPCT1007** 警告が出力されます。

spMV.dp.cpp

```
#include <sycl/sycl.hpp>
#include <dpct/dpct.hpp>
#include <iostream>
#include <dpct/sparse_utils.hpp>

#include <dpct/blas_utils.hpp>

#include <dpct/lib_common_utils.hpp>

int main() {

    // cuSPARSE ライブラリーの初期化
    ...

    // メモリー管理
    ...

    // 行列記述子とベクトル記述子の作成
    /*
    DPCT1007:0: Migration of cusparseCreateCsr is not supported.
    */
    cusparseCreateCsr(&matA, 3, 3, 4, d_csrRowPtr, d_csrColInd, d_csrVal,
                      CUSPARSE_INDEX_32I, CUSPARSE_INDEX_32I,
                      oneapi::mkl::index_base::zero,
                      dpct::library_data_t::real_float);
    /*
    DPCT1007:1: Migration of cusparseCreateDnVec is not supported.
    */
    cusparseCreateDnVec(&vecX, 3, d_x, dpct::library_data_t::real_float);
    /*
    DPCT1007:2: Migration of cusparseCreateDnVec is not supported.
    */
    cusparseCreateDnVec(&vecY, 3, d_y, dpct::library_data_t::real_float);

    float alpha = 1.0f;
    float beta = 0.0f;

    // バッファーの作成と割り当て
    ...

    // 行列-ベクトル乗算の実行: y = A*x
    /*
    DPCT1007:4: Migration of cusparseSpMV is not supported.
    */
    cusparseSpMV(handle, oneapi::mkl::transpose::nontrans, &alpha, matA, vecX,
                 &beta, vecY, dpct::library_data_t::real_float,
                 CUSPARSE_MV_ALG_DEFAULT, dBuffer);
```

```
    // 結果をホストにコピーバック
    ...

    // クリーンアップ
    ...

    return 0;
}
```

移行が完了すると、SYCLomatic は移行されなかった cuSparse API に関する **DPCT1007** 警告を出力します。このコードをコンパイルするとコンパイルエラーとなり、NVIDIA* GPU では実行できません。しかし、SYCL* の相互運用性により、以下のように部分的に移行されたアプリケーションにわずかな変更を加えることで、NVIDIA* GPU 上で実行できるようになります。

spMV_with_SYCL_Interop.dp.cpp

```
#include <sycl/sycl.hpp>
#include <dpct/dpct.hpp>
#include <iostream>
#include <dpct/sparse_utils.hpp>

#include <dpct/blas_utils.hpp>

#include <dpct/lib_common_utils.hpp>

#include <cusparse.h>

int main() {

    // cuSPARSE ライブラリーの初期化
    ...

    // メモリー管理
    ...

    // 行列記述子とベクトル記述子の作成
    /*
    DPCT1007:0: Migration of cusparseCreateCsr is not supported.
    */
    /*
    // SYCL* 相互運用性
    cusparseCreateCsr(&matA, 3, 3, 4, d_csrRowPtr, d_csrColInd, d_csrVal,
                      CUSPARSE_INDEX_32I, CUSPARSE_INDEX_32I,
                      CUSPARSE_INDEX_BASE_ZERO,
                      CUDA_R_32F);
    /*
    DPCT1007:1: Migration of cusparseCreateDnVec is not supported.
    */
    // SYCL* 相互運用性
    cusparseCreateDnVec(&vecX, 3, d_x, CUDA_R_32F);

    /*
    DPCT1007:2: Migration of cusparseCreateDnVec is not supported.
    */
    // SYCL* 相互運用性
```

```
    cusparseCreateDnVec(&vecY, 3, d_y, CUDA_R_32F);

    float alpha = 1.0f;
    float beta = 0.0f;

    // cusparseSpMV 操作に必要なバッファーサイズの特定
    size_t bufferSize = 0;
    /*
    DPCT1007:3: Migration of cusparseSpMV_bufferSize is not supported.
    */
    // SYCL* 相互運用性
    cusparseSpMV_bufferSize(handle, CUSPARSE_OPERATION_NON_TRANSPOSE, &alpha,
        matA, vecX, &beta, vecY,
        CUDA_R_32F,
        CUSPARSE_MV_ALG_DEFAULT, &bufferSize);

    // バッファーの作成と割り当て
    ...

    // 行列-ベクトル乗算の実行: y = A*x
    /*
    DPCT1007:4: Migration of cusparseSpMV is not supported.
    */
    // SYCL* 相互運用性
    cudaStream_t nativeStream =
sycl::get_native<sycl::backend::ext_oneapi_cuda>(q_ct1);
    cudaStreamCreate(&handle);
    cusparseSetStream(handle, stream);

    cusparseSpMV(handle, CUSPARSE_OPERATION_NON_TRANSPOSE, &alpha, matA, vecX,
        &beta, vecY, CUDA_R_32F,
        CUSPARSE_MV_ALG_DEFAULT, dBuffer);

    // 結果をホストにコピーバック
    ...

    // クリーンアップ
    ...

    return 0;
}
```

ここでは、**sycl::get_native** 関数と要求されたバックエンドがテンプレート引数として渡される SYCL* キューを使用して、cusparseSpMV API のネイティブ・ストリーム・ハンドルを派生することで、SYCL* の相互運用性を最もよく示しています。

注: 相互運用性を使用して SYCL* コードをコンパイルするには、ユーザーがネイティブ CUDA* ライブラリー・ヘッダーとライブラリー・ファイルをコンパイラーのインクルード・パスに追加する必要があります。

SYCL* を介した完全な CUDA* 機能

インテルとエコシステム・パートナーが oneAPI 製品を継続的に強化することで、SYCL* の相互運用性はマルチベンダー・ハードウェアをサポートするソリューションとなります。

これにより、CUDA* と oneAPI 間に明確かつ効率的なパフォーマンス指向のパスが提供され、両方のエコシステムの長所を組み合わせて、SYCL* ベースのプロジェクトを効率良く運用できる状態にします。

開発者は、パフォーマンスを低下させることなく、oneAPI フレームワーク内から CUDA* の機能を利用できます。

関連情報

- spMV を使用して CUDA*-SYCL* の相互運用性を示すサンプル (英語)
- SYCL* 2020 仕様 (英語)
    - バックエンドの相互運用性 (英語)
    - ホスト・タスク・インターフェイス (英語)
    - get_native テンプレート・メンバー関数 (英語)

ソフトウェアを入手

CUDA* から SYCL* へ効率良くコードを移行するため、インテル® DPC++ 互換性ツール (英語) を使用することを推奨します。

詳細なガイダンスと移行前/後の比較を含む、移行されたサンプルコードの一覧を確認してください。これらはすべて、CUDA* から C++ with SYCL* への移行ポータルにあります。

CUDA* から SYCL* への移行に役立つ追加のリソース:

- インテル® DPC++ 互換性ツールの製品ページ (英語)
- 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
- インテル® DPC++ 互換性ツールのセルフガイドの Jupyter* Notebook チュートリアル (英語)

インテル® DPC++ 互換性ツールは、インテル® oneAPI ベース・ツールキットに含まれています。

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、http://www.intel.com/PerformanceIndex/ (英語) を参照してください。