SYCLomatic：新しい CUDA* から SYCL* へのコード移行ツール

このオープンソース・プロジェクトにぜひ貢献してください

Kent Moffat インテル コーポレーション シニア製品マネージャー

CPU、GPU、FPGA、およびその他のアーキテクチャーでハイパフォーマンスと効率的な開発生産性を実現するには、開発者が目の前のタスクに最適なハードウェアを選択できる統一されたプログラミング・モデルが必要です。それには、標準化され、拡張可能な、高水準でオープンスタンダードのヘテロジニアス・プログラミング言語が不可欠です。また、開発者の生産性を向上させるとともに、どのアーキテクチャーでも一貫したパフォーマンスを提供する必要があります。Khronos Group の C++ ベースの SYCL* 標準は、C++ の機能を拡張し、マルチアーキテクチャーとディスジョイント・メモリー構成をサポートすることで、これらの課題に対処しています。

SYCL* の導入を容易にするため、開発者は SYCL* の開発をゼロから始めるのではなく、既存の CUDA* GPU コードを移行したいと考えるかもしれません。以前、インテル® DPC++ 互換性ツール（インテル® DPCT）を使用して CUDA* から SYCL* へ移行する記事を掲載しました。このツールは、インテル® oneAPI ベース・ツールキットに含まれており、インテルのテクニカル・コンサルティング・エンジニアによってサポートされています。

SYCLomatic オープンソース・プロジェクト

この互換性ツールは、開発者の要望に応え、「SYCLomatic」という名前でオープンソース・プロジェクトとしてリリースされています。多くの組織がこのツールを使用しており、一部の組織ではその機能を強化し、カスタマイズして自分たちのニーズに合わせてチューニングしています。その 1 つがアルゴンヌ国立研究所です。

> 「CRK-HACC は、現在活発に開発が行われている宇宙論的 N 体シミュレーション・コードです。Aurora の準備のため、インテル® DPC++ 互換性ツールを使用して、20 以上の（CUDA*）カーネルを SYCL* に素早く移行できました。コード移行ツールの現在のバージョンはファンクターへの移行をサポートしていないため、簡単な Clang ツールを記述して、生成された SYCL* ソースコードをニーズに合うようにリファクタリングしています。オープンソースの SYCLomatic プロジェクトでは、私たちのこれまでの研究を統合して、より堅牢なソリューションを提供し、ファンクターを利用可能な移行オプションにすることを計画しています。」
>
> アルゴンヌ国立研究所
> 宇宙物理学 & 先進コンピューティング
> HACC（Hardware/Hybrid Accelerated Cosmology Code）
> Steve（Esteban）Rangel 氏

LLVM 例外付き Apache* 2.0 ライセンスを利用し、GitHub* でホストされている SYCLomatic プロジェクトは、CPU、GPU、FPGA 向けのオープンなヘテロジニアス開発を推進するため、開発者が貢献したり、フィードバックを送ることができるコミュニティーを用意しています。GitHub* ポータルにある「CONTRIBUTING.md」というガイドで、プロジェクトへの技術的貢献の手順が説明されています。開発者の皆さん、ぜひ本ツールを使用し、ツールを進化させるためフィードバックや貢献をお寄せください。このオープンソース・プロジェクトは、SYCL* 標準の採用を推進するコミュニティーに協力でき、開発者を単一ベンダーの独占的エコシステムから解放する重要なステップとなります。SYCLomatic に加えられた改善点は、インテル® DPC++ 互換性ツール製品にもフィードバックされます。

SYCLomatic ツールの仕組み

SYCLomatic は、CUDA* から SYCL* への移行を支援し、通常、CUDA* コードの 90 ~ 95% を自動的に SYCL* コードに移行します。[1] 開発者は残りのコーディングを手動で行い、ターゲット・アーキテクチャーで求められるパフォーマンスを達成できるようにチューニングします（**図 1**）。

[1] 出典：2021 年 9 月現在のインテルによる推定。Rodinia、SHOC、PENNANT など、70 種類の HPC ベンチマークとサンプルの測定結果に基づいています。結果は異なることがあります。

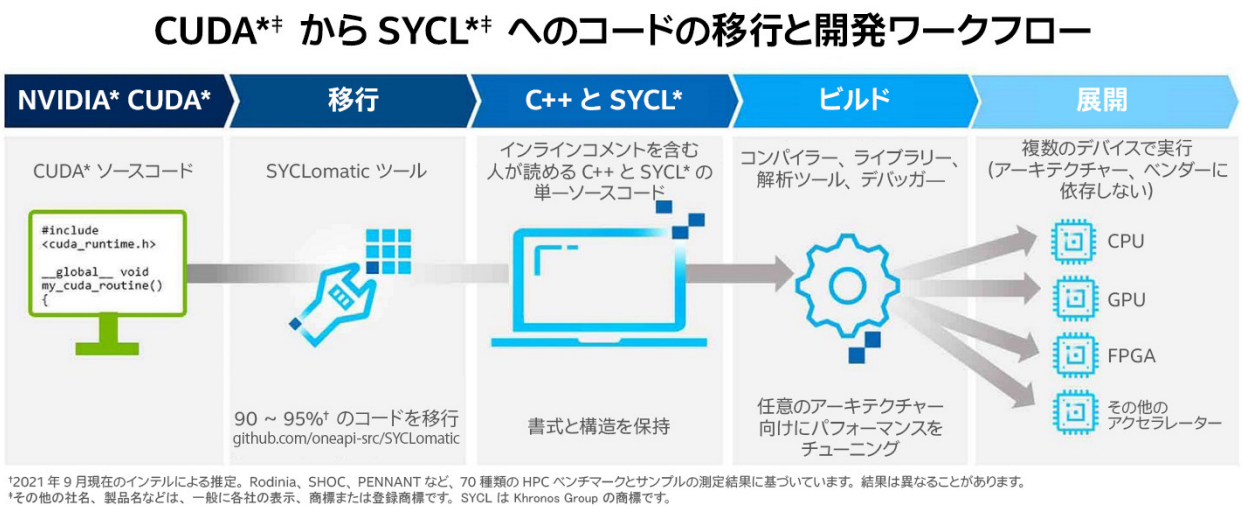

図 1. SYCLomatic のワークフロー

コード移行の成功例

多くの研究機関やインテルのお客様が、SYCLomatic と同じ技術を持つインテル® DPC++ 互換性ツールを使用して、複数のベンダーのアーキテクチャー上で CUDA* コードから SYCL*（または oneAPI の SYCL* 実装であるデータ並列 C++）への移行に成功しています。例として、ストックホルム大学の GROMACS 2022（英語）、Zuse Institute Berlin（ZIB）（英語）の easyWave、Samsung Medison（英語）、Bittware（英語）などが挙げられます（その他の例は、oneAPI DevSummit のコンテンツ（英語）を参照してください）。また、アルゴンヌ国立研究所の Aurora スーパーコンピューター（英語）、ライプニッツ・スーパーコンピューティング・センター（LRZ）（英語）、GE Healthcare（英語）など、複数のお客様が現在および将来のインテル® Iris® Xe アーキテクチャー・ベースの GPU でコードをテストしています。

例：CUDA* ベクトル加算の SYCL* への移行

移行プロセスの実用的な概要を説明するため、ここではベクトル加算の簡単な CUDA* 実装を使用します。そして、SYCLomatic が生成するコードを詳しく見ていきます。主に、CUDA* と SYCL* が大きく異なるコードセクションに注目します。

このタスクには、インテル® oneAPI ベース・ツールキットの SYCLomatic とインテル® oneAPI DPC++/C++ コンパイラーを使用します。ツールキットをインストールするには、「インテル® oneAPI インストール・ガイド」（英語）の手順に従ってください。次のワークフローを使用して、既存の CUDA* アプリケーションを SYCL* へ移行します。

1. intercept-build ユーティリティーを使用して Makefile が実行するコマンドをインターセプトし、JSON 形式のコンパイル・データベース・ファイルに保存します。このステップは、シングルソースのプロジェクトではオプションです。
2. SYCLomatic を使用して CUDA* コードを SYCL* へ移行します。
3. 生成されたコードの正当性を検証し、警告メッセージで明示的に示された場合は、手動で移行を完了します。『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』（英語）を確認して警告を修正します。
4. インテル® oneAPI DPC++/C++ コンパイラーでコードをコンパイルし、プログラムを実行して、出力を確認します。

その後、インテル® VTune™ プロファイラーを含むインテル® oneAPI の解析およびデバッグツールを使用して、コードをさらに最適化できます。

ベクトル加算の例について考えてみます。ベクトル加算では、ベクトル `A` と `B` の要素を加算してベクトル `C` に格納します。CUDA* カーネルは、これを次のように計算します。

```
__global__ void vector_sum(const float *A,
                           const float *B,
                           float *C,
                           const int num_elements)
{
    int idx = blockDim.x * blockIdx.x + threadIdx.x;
    if (idx < num_elements) C[idx] = A[idx] + B[idx];
}
```

CUDA* のスレッドブロック（スレッドのグループ）は SYCL* のワークグループに相当しますが、スレッドのインデックスの計算方法は異なります。CUDA* では、組込み変数を使用してスレッドを識別します（上記のコードで `idx` 変数をどのように計算しているかを見てください）。SYCL* に移行すると、同じカーネルは次のようになります。

```
void vector_sum(const float *A,
                const float *B,
                float *C,
                const int num_elements,
                sycl::nd_item<3> item_ct1)
{
    int idx = item_ct1.get_local_range().get(2) *
              item_ct1.get_group(2) +
              item_ct1.get_local_id(2);
    if (idx < num_elements) C[idx] = A[idx] + B[idx];
}
```

CUDA* のスレッドと同様に、SYCL* のワークアイテムは、グローバル空間でのグローバル識別子、またはワークグループ内のローカル識別子を持ちます。これらの識別子は `nd_item` 変数から得られます。そのため、グローバル識別子を明示的に計算する必要はありません。しかし、この例では SYCL* でこれを行う方法を示しているため、CUDA* の組込み変数と類似していることが分かります。`nd_item` は、CUDA* では `dim3` 型なので、3 次元であることに注意してください。この例では、`nd_item` を 1 次元にすることができます。これにより、ワークアイテムがベクトルの各要素にマップされます。CUDA* カーネルを実行するには、ブロックサイズと必要なブロック数を設定しなければなりません。SYCL* では、実行範囲を定義する必要があります。以下のコードに示すように、グローバル範囲とローカル範囲を組み合わせた `nd_range` 変数でこれを行います。グローバル範囲はワークアイテムの総数を表し、ローカル範囲はワークグループのサイズを表します。

```
const int num_elements = 512;
dpct::device_info prop;
dpct::dev_mgr::instance().get_device(0).get_device_info(prop);
const size_t max_block_size = prop.get_max_work_group_size();
const size_t block_size = std::min<size_t>(max_block_size, num_elements);

range<1> global_rng(num_elements);
range<1> local_rng(block_size);
nd_range<1> kernel_rng(global_rng, local_rng);
```

SYCL* カーネルを呼び出すため、`parallel_for` と実行範囲を使用して、カーネルをキューに送信します。各ワークアイテムはカーネルを一度呼び出します。この例では、各ベクトル要素に対して同じ数のワークアイテムがあります。以下のようなコードになります。

```
dpct::get_default_queue().parallel_for(kernel_rng, [=](nd_item<1> item_ct1)
{
    vector_sum(d_A, d_B, d_C, num_elements, item_ct1);
});
```

ここまで、カーネルを実装して実行する方法を見てきましたが、カーネルを実行する前に、メモリー割り当てと、データのデバイスへのコピーについて考える必要があります。

1. まず、ホスト側でオペランドベクトル用のメモリーを割り当てて初期化します。
2. その後、デバイス上でも同じことを行います。CUDA* は `cudaMalloc` ルーチンを使用します。デフォルトでは、インテル® DPCT はこのルーチンを統合共有メモリー（USM）を使用する `malloc_device` に移行します。
3. 次に、`memcpy` コマンドを使用して、ホストメモリーからデバイスにベクトルをコピーします。

これらのステップが完了したら、カーネルを実行します。実行が完了したら、結果をホストへコピーバックして、結果の正当性を確認します。最後に、ホストとデバイスで `free` と `sycl::free` をそれぞれ呼び出して、メモリーを解放します。

まとめ

Khronos の SYCL* C++ 標準は、複数のアーキテクチャーで動作するヘテロジニアス・コードを開発するオープンなパスです。新しいオープンソース・プロジェクトである SYCLomatic は、以前のインテル® DPC++ 互換性ツールと同様に、CUDA* から SYCL* へのコード移行の利点を提供します。そして、誰でもこのツールの改善やチューニングに貢献できます。ぜひお試しください。

開発者向けのリソース

- プログラミング・モデルの比較 : CUDA* と SYCL*（英語）
- SYCL* について（英語）
- oneAPI 仕様（英語）
- インテル® oneAPI ツールキット
- GitHub* の SYCLomatic プロジェクト（英語）| CONTRIBUTING.md ガイド（英語）
- 初心者向け:書籍:『C++ と SYCL* によるヘテロジニアス・システムのプログラミングの習得』（英語）| トレーニング: SYCL* の基本（英語）
- コード・プロジェクト : oneAPI を使用して CUDA* コードを SYCL* へ移行する（英語）
- インテル® DevCloud（英語）:インテル® oneAPI ツールを利用して、さまざまなインテル® アーキテクチャー（CPU、GPU、FPGA）でコードを開発およびテストできる無償環境です。iSUS で翻訳した日本語版はこちら。