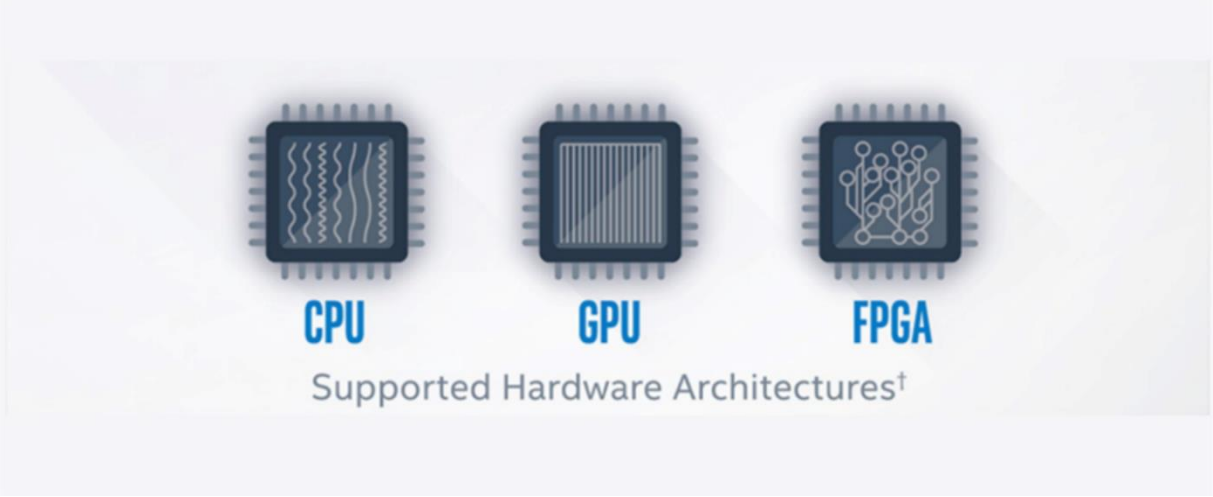
DPC++ 基礎サンプルコード

この記事は、インテル® デベロッパー・ゾーンで公開されている「DPC++ Foundations Code Sample Walk-Through」の日本語参考訳です。

データ並列 C++(DPC++)の基礎を紹介するサンプルコード

この記事では、サンプルコード vector_add を使用して、データ並列 C++(DPC++)プログラミング言語における oneAPI の概念と機能を示します。このプログラムは、ハードウェア・アクセラレーションにより、2 つの整数配列を加算します。ここでは、次の項目について学ぶことができます。

- DPC++ ヘッダー
- カーネルからの非同期例外
- 異なるアクセラレーター用のデバイスセレクター
- バッファーとアクセサー
- キュー
- parallel_for カーネル



サンプルコード vector_add は、GitHub*(英語)からダウンロードできます。

DPC++(SYCL*)ヘッダー

DPC++ は、使い慣れた業界標準の C++ をベースに、Khronos Group の SYCL* 1.2.1 仕様を取り入れ、オープン・コミュニティーにより開発された言語拡張機能を採用しています。また、SYCL* 仕様で定義されているヘッダーファイル sycl.hpp は、インテル® oneAPI DPC++/C++コンパイラーでも提供されています。FPGA サポートは、DPC++ 拡張とともに fpga_extensions.hpp ヘッダーファイルに含まれています。

以下の vector_add のコードスニペットは、異なるアクセラレーターをサポートする場合に必要なヘッダーの違いを示しています。

```
// CPU または GPU の場合

#include <CL/sycl.hpp>
#include <array>
#include <iostream>
using namespace sycl;

// FPGA の場合

#include <CL/sycl.hpp>
#include <array>
#include <iostream>
#if FPGA || FPGA_EMULATOR
#include <CL/sycl/INTEL/fpga_extensions.hpp>
#endif
using namespace sycl;
```

DPC++ カーネルからの async 例外のキャッチ

DPC++ カーネルは、異なるスタックフレームのアクセラレーター上で非同期に実行します。カーネルでは、スタックに伝搬できない非同期エラーが発生する場合があります。非同期例外をキャッチするため、SYCL* キュークラスはエラーハンドラー関数を提供しています。次の vector_add サンプルのコードは、エラーハンドラー関数の使用法を示します。

```
// async 例外をキャッチする例外ハンドラーを作成

static auto exception_handler = [](cl::sycl::exception_list eList) {
    for (std::exception_ptr const &e : eList) {
        try {
            std::rethrow_exception(e);
        }
        catch (std::exception const &e) {
#if _DEBUG
            std::cout << "Failure" << std::endl;
#endif
            std::terminate();
        }
    }
};
… …
try {
    queue q(d_selector, exception_handler);
    … …
} catch (exception const &e) {
    … …
}
```

アクセラレーターのデフォルトのセレクターの使用

オフロードカーネル用のアクセラレーターの選択は簡単です。SYCL* と oneAPI には、実行環境で利用可能なハードウェアを検出してアクセス可能なセレクターがあります。default_selector は、すべての利用可能なアクセラレーターを列挙して、最もパフォーマンスが高いものを選択します。

DPC++ は、FPGA アクセラレーター用に fpga_selector と fpga_emulator_selector クラスを追加で提供しており、これらは fpga_extensions.hpp にあります。以下の vector_add のコードスニペットは、その一例です。

```
#if FPGA || FPGA_EMULATOR
#include <CL/sycl/INTEL/fpga_extensions.hpp>
#endif
… …
#if FPGA_EMULATOR
  // DPC++ 拡張:FPGA カードが搭載されていないシステムの FPGA エミュレーター・セレクター
  INTEL::fpga_emulator_selector d_selector;
#elif FPGA
  // DPC++ 拡張:FPGA カードが搭載システムの FPGA セレクター
  INTEL::fpga_selector d_selector;
#else
  // デフォルトのデバイスセレクターは最もパフォーマンスの良いものを選択
  default_selector d_selector;
#endif
```

データ、バッファー、アクセサー

DPC++は、大規模なデータや計算を処理するアクセラレーター上で動作するカーネルを使用します。ホスト上で宣言されたデータはバッファーにラップされ DPC++ ランタイムによって暗黙的にアクセラレーターに転送されます。アクセラレーターは、アクセサーを介してデータを読み書きします。ランタイムはアクセサーからカーネルの依存関係を検出して、最も効率的な順序でカーネルをディスパッチして実行します。

- a_array、b_array、および sum_parallel はホストからの配列オブジェクトです。
- a_buf、b_buf、および sum_buf はバッファーラッパーです。
- a と b は読み取り専用のアクセサーで、sum は書き込み専用のアクセサーです。

```
  buffer a_buf(a_array);
  buffer b_buf(b_array);
  buffer sum_buf(sum_parallel.data(), num_items);
… …
  q.submit([&](handler &h) {

// 読み取り、書き込み、または読み書き権限で各バッファーのアクセサーを作成
// アクセサーはバッファー内のメモリーアクセスに使用される

    accessor a(a_buf, h, read_only);
    accessor b(b_buf, h, read_only);

// sum_アクセサー(書き込み権限)は合計データのストアに使用される

    accessor sum(sum_buf, h, write_only);
… …
  });
```

キューと parallel_for カーネル

DPC++ キューは、カーネル実行に必要なすべてのコンテキストと状態をカプセル化します。パラメーターが指定されない場合、デフォルトのセレクターを介してキューが作成され、アクセラレーターに関連付けられます。また、特定のデバイスセレクターと vector_add で使用される非同期例外ハンドラーを受け取ることもできます。

カーネルは、キューを介してデバイスに送信され実行されます。カーネルには、単一タスクカーネル、基本データ並列カーネル、階層並列カーネルなどの種類があります。vector_add では、基本データ並列カーネル parallel_for が使用されます。

```
try {
     queue q(d_selector, exception_handler);
     … …
     q.submit([&](handler &h) {
     … …
          h.parallel_for(num_items, [=](auto i) { sum[i] = a[i] + b[i]; });
   });
} catch (exception const &e) {
     … …
}
```

カーネル本体は、ラムダ関数でキャプチャーされた 2 つの配列を加算します。

```
sum[i] = a[i] + b[i];
```

カーネルが処理可能なデータ範囲は、h.parallel_for の第 1 引数 num_items で指定されています(例: サイズ num_items の 1D 範囲)。2 つの読み取り専用データ a_array と b_array はランタイムによってアクセラレーターに転送されます。カーネルが完了し、sum_buf バッファーがスコープ外になるときに、sum_buf 内のデータの合計がホストにコピーされます。

まとめ

デバイスセレクター、バッファー、アクセサー、キュー、カーネルは、oneAPI プログラミングのビルディング・ブロックです。DPC++ は、データ並列プログラミングを容易にするため、SYCL* とコミュニティー拡張を取り入れた ISO C++ のオープンな標準ベースの進化版です。DPC++は、ハードウェア・ターゲット間でコードを再利用し、CPU、GPU、FPGA アーキテクチャー間で高い生産性とパフォーマンスを実現し、アクセラレーター固有のチューニングを可能にします。

製品とパフォーマンス情報

[1] 実際の性能は利用法、構成、その他の要因によって異なります。詳細は、www.Intel.com/PerformanceIndex(英語)を参照してください。