SYCL* によるデバイス検出

システムに搭載されているアクセラレーターを知る

Henry A. Gabb インテル コーポレーション シニア主席エンジニア兼 The Parallel Universe 編集長
John Pennycook インテル コーポレーション ソフトウェア・イネーブリング & 最適化アーキテクト

デバイス検出は、SYCL* やクロスアーキテクチャーのヘテロジニアス並列プログラミング・アプローチの重要な一面です。以前の oneAPI の記事では、SYCL* と oneMKL および oneDPL ライブラリーを使用して計算をアクセラレーター・デバイスにオフロードすること（つまり、コードを実行する場所の制御）に焦点を当てました。ヘテロジニアス・システム向けの移植可能なコードを記述するには利用可能なハードウェアに関する情報をシステムに照会する機能が必要になるため、この記事ではデバイス検出に焦点を当てています。例えば、GPU を使用するように SYCL* デバイスセレクターを指定しても、システムに GPU がない場合、次の文は失敗します。

```
...
    sycl::queue Q(sycl::gpu_selector_v);
...
terminate called after throwing an instance of 'sycl::_V1::runtime_error'
  what():  No device of requested type available. Please check https://software.intel.com/
content/www/us/en/develop/articles/intel-oneapi-dpcpp-system-requirements.html -1 (PI_ERROR_
DEVICE_NOT_FOUND)
Aborted
```

このコードは、GPU がないシステムには移植できません。GPU セレクターの代わりにデフォルトセレクターを使用して SYCL* キューをインスタンス化すれば動作することは保証されますが、キューがどこにワークを送信するか制御できなくなります。SYCL* ランタイムがデバイスを選択します。

```
...
    sycl::queue Q(sycl::default_selector_v);

    std::cout << "Running on: "
              << Q.get_device().get_info<sycl::info::device::name>()
              << std::endl;
...
Running on: Intel(R) Xeon(R) Gold 6128 CPU @ 3.40GHz
```

確実なヘテロジニアス並列プログラムを作成するため、SYCL* デバイス検出を詳しく見て、次の質問に答えてみましょう。

- どのアクセラレーター・デバイスを使用できますか？
- SYCL* キューはどのデバイスを使用していますか？
- oneDPL 実行ポリシーはどのデバイスを使用していますか？

堅牢なデバイス検出

さまざまなインテル® ハードウェアのオプションがあり、最新のインテル® oneAPI ソフトウェアがすでにインストールされているインテル® DevCloud for oneAPI（英語）で、いくつかの例を実行してみましょう。ハードウェアは定期的に更新されますが、この記事の執筆時点（2022 年 12 月 2 日）では次の計算ノードが利用可能でした。

```
$ pbsnodes | grep properties | sort | uniq -c | sort -nr

     79      properties = xeon,skl,gold6128,ram192gb,net1gbe,jupyter,batch
     78      properties = xeon,cfl,e-2176g,ram64gb,net1gbe,gpu,gen9
     26      properties = xeon,skl,gold6128,ram192gb,net1gbe,jupyter,batch,fpga_compile
     25      properties = core,tgl,i9-11900kb,ram32gb,netgbe,gpu,gen11
     12      properties = xeon,skl,ram384gb,net1gbe,renderkit
     12      properties = xeon,skl,gold6128,ram192gb,net1gbe,fpga_runtime,fpga,arria10
      6      properties = xeon,icx,gold6348,ramgb,netgbe,jupyter,batch
      4      properties = xeon,icx,plat8380,ram2tb,net1gbe,batch
      4      properties = xeon,clx,ram192gb,net1gbe,batch,extended,fpga,stratix10,fpga_runtime
```

ご覧のように、さまざまな CPU、GPU、および FPGA オプションがあります（機密保持契約ユーザーは、インテル® DevCloud for oneAPI の NDA パーティション内のリリース前のハードウェアにアクセスできます）。ノードをリクエストして、利用可能なデバイスを確認します。

```
$ qsub -I -l nodes=1:gen11:ppn=2
```

このコマンドは、第 11 世代インテル® プロセッサー・グラフィックスを含む単一ノードへの対話型アクセスをリクエストします。SYCL* では、すでに説明した 2 つのセレクターに加えて、ビルトインセレクターとして `default_selector_v`、`gpu_selector_v`、`cpu_selector_v`、および `accelerator_selector_v` を提供しています。インテルは、FPGA 開発向けに、`fpga_selector` および `fpga_emulator_selector` の 2 つの拡張機能も提供しています。これらの拡張機能は `sycl/ext/intel/fpga_device_selector.hpp` ヘッダーに含まれています。FPGA 上での SYCL* の使用に関する詳細は、『インテル® oneAPI プログラミング・ガイド』の「FPGA フロー」を参照してください。

ビルトインセレクターは主に利便性を目的としていますが、例外処理と組み合わせると堅牢になります。

```
sycl::device d;
try {
    d = sycl::device(sycl::gpu_selector_v);
}
catch (sycl::exception const &e) {
    d = sycl::device(sycl::cpu_selector_v);
}
```

しかし、まだ SYCL* ランタイムがデバイスを選択しています。特に複数のデバイスが利用可能な場合は、さらなる制御が必要になります。

次のプログラムは、計算ノードで利用可能なプラットフォームとデバイスをリストします（この情報は、`sycl-ls` コマンドライン・ユーティリティーを使用して取得することもできます）。

```
#include <sycl/sycl.hpp>

int main()
{
    for (auto platform : sycl::platform::get_platforms())
    {
        std::cout << "Platform: "
                  << platform.get_info<sycl::info::platform::name>()
                  << std::endl;

        for (auto device : platform.get_devices())
        {
            std::cout << "\tDevice: "
                      << device.get_info<sycl::info::device::name>()
                      << std::endl;
        }
    }
}
```

```
$ icpx -fsycl show_platforms.cpp -o show_platforms
$ ./show_platforms

Platform: Intel(R) FPGA Emulation Platform for OpenCL(TM)
        Device: Intel(R) FPGA Emulation Device
Platform: Intel(R) OpenCL
        Device: 11th Gen Intel(R) Core(TM) i9-11900KB @ 3.30GHz
Platform: Intel(R) OpenCL HD Graphics
        Device: Intel(R) UHD Graphics [0x9a60]
Platform: Intel(R) Level-Zero
        Device: Intel(R) UHD Graphics [0x9a60]
```

SYCL* プラットフォームは、ホストがアクセラレーター・デバイスに接続されている OpenCL* プラットフォーム・モデルに基づいています。これは、上記の出力例で明白です。このシステムには OpenCL* プラットフォームと oneAPI レベルゼロ・プラットフォームがあります。各プラットフォームには、SYCL* プログラムがワークを送信できるデバイスがあります。OpenCL* または oneAPI レベルゼロ・バックエンドを使用するかどうかに応じて、2 つの GPU プラットフォームがあります。CPU および FPGA エミュレーション・プラットフォームもあります。この情報を利用して、これらのデバイスのいずれかにワークを送信するキューを作成できます。

```
#include <sycl/sycl.hpp>

int main()
{
    auto platforms = sycl::platform::get_platforms();

    sycl::queue Q1(platforms[1].get_devices()[0]);
    sycl::queue Q2(platforms[3].get_devices()[0]);

    std::cout << "Q1 mapped to "
              << Q1.get_device().get_info<sycl::info::device::name>()
              << std::endl;

    std::cout << "Q2 mapped to "
              << Q2.get_device().get_info<sycl::info::device::name>()
              << std::endl;
}
$ ipcx -fsycl map_queues.cpp -o map_queues
$ ./map_queues

Q1 mapped to 11th Gen Intel(R) Core(TM) i9-11900KB @ 3.30GHz
Q2 mapped to Intel(R) UHD Graphics [0x9a60]
```

前の例では明示的にデバイスが指定されているため、各自のシステムでこのプログラムを試す場合は、必ずプラットフォーム・インデックスを更新してください。

SYCL* キューとデバイスの照会

前のサンプルコードではキューの作成を確認できますが、常に確認できるとは限りません。例えば、SYCL* キューは通常、oneAPI ライブラリー関数に渡されます。そのため、場合によっては、次のような情報をキューに照会する必要があります。

- キューがマップされているターゲットデバイスは何か？
- デバイスのバックエンド API は何か？
- それはインオーダー・キューか（つまり、カーネルを送信した順序で実行する必要があるか）？
- ターゲットデバイスのベクトル幅または最大ワークアイテム数はいくつか？

ライブラリー開発者は、照会した情報を使用して最適なコードパスを選択できます。そのため、SYCL* `queue` クラスでは、`get_backend()`、`get_context()`、`get_device()`、`is_in_order()` などの情報を照会するメンバー関数を提供しています。同様に、SYCL* device クラスでは、`is_cpu()`、`is_gpu()`、`get_info()` などのデバイス特性を照会するメンバー関数を提供しています。特に、`get_info()` 関数は、ベンダー、ベクトル幅、最大ワークアイテム数とイメージサイズ、メモリー特性など、ターゲットデバイスに関する詳細な情報を収集するために使用できます。SYCL* 2020 仕様には、照会できるデバイス情報記述子とデバイスアスペクトの完全なリストが含まれています。

これらの情報を使用したコードの最適化はここでは取り上げませんが、将来の記事の候補の 1 つです。

カスタムセレクター

これまで見てきたデバイスセレクターは、SYCL* 実装により提供されるビルトインセレクターでした。これらのデバイスセレクターは、デバイスを受け入れてスコアを返す C++ の呼び出し可能オブジェクトとして実装されます。SYCL* 実装は、デバイスセレクターを呼び出してシステム内で利用可能なすべてのデバイスにスコアを付け、最終的にスコアが最も高いデバイスを選択します。

同じ形式の呼び出し可能オブジェクトを作成することで、独自のカスタム・デバイス・セレクターを作成できます。簡単な最初の例として、すべての CPU デバイスに正のスコアを付け、他のすべてのデバイスに負のスコアを付けることで、ビルトイン CPU セレクターと同じ動作を行うデバイスセレクターを作成します。

```
...
    auto my_cpu_selector = [](const sycl::device& d)
    {
        if (d.is_cpu())
        {
            return 1;
        }
        else
        {
            return -1;
        }
    };
    sycl::queue Q(my_cpu_selector);
...
Running on: Intel(R) Xeon(R) Gold 6128 CPU @ 3.40GHz
```

デバイスセレクターは単なる関数であるため、デバイスの任意のプロパティー（アスペクトやデバイス情報記述子など）とプログラムのほかの変数（コマンドライン引数など）を組み合わせて、デバイスのスコアを付けることができます。デバイスのスコア付けと選択を完全に制御できるため、選択したデバイスがアプリケーションの要件を満たしていることを確認できます。以下の例は、倍精度浮動小数点演算をサポートしていないデバイスを無視し、ブール変数を使用して GPU を優先するように構成するデバイスセレクターを示しています。

```
...
    bool prefer_gpus = true;  // コマンドラインまたは設定ファイルから
    auto my_selector = [=](const sycl::device& d)
    {
        // 倍精度をサポートしないデバイスを無視
        if (not d.has(sycl::aspect::fp64))
        {
            return -1;
        }

        // オプションで GPU を選択
        if (prefer_gpus and d.is_gpu())
        {
            return 1;
        }
        else
        {
            return 0;
        }
    };
    sycl::queue Q(my_selector);
...
```

プログラマーの要件を満たすデバイスを選択できるように支援する `aspect_selector` 関数が最近 SYCL* に追加されました。例えば、次の文は、エミュレートされた固定機能デバイスを除外して、半精度をサポートする GPU デバイスを選択します。

```
auto dev = sycl::device{sycl::aspect_selector(
  std::vector{sycl::aspect::fp16, sycl::aspect::gpu},        // 対象のアスペクト
  std::vector{sycl::aspect::custom, sycl::aspect::emulated}  // 対象でないアスペクト
)};
```

この記事の執筆時点では `aspect_selector` はまだインテル® oneAPI DPC++/C++ コンパイラーでサポートされていませんが、間もなく利用可能になるはずです［**翻訳者注：** バージョン 2023.1 では `aspect_selector` はサポートされていますが、`sycl::aspect::emulated` は未実装です］。

oneDPL の実行ポリシーの変更

The Parallel Universe 48 号の記事「oneAPI の maxloc リダクション」では、oneDPL の実行ポリシーを使用して関数をアクセラレーターにオフロードする方法を紹介しました。コード例では `oneapi::dpl::execution::dpcpp_default` ポリシーを単純に使用しています。SYCL* キューを使用して実行ポリシーを変更し、oneDPL 関数を実行する場所を明示的に制御する方法を見てみましょう。

```
#include <oneapi/dpl/execution>

int main()
{
    sycl::queue Q1(sycl::gpu_selector_v);
    auto gpu_policy = oneapi::dpl::execution::make_device_policy(Q1);

    std::cout << "GPU execution policy runs oneDPL functions on "
              << gpu_policy.queue().get_device().
                                  get_info<sycl::info::device::name>()
              << std::endl;

    sycl::queue Q2(sycl::cpu_selector_v);
    auto cpu_policy = oneapi::dpl::execution::make_device_policy(Q2);

    std::cout << "CPU execution policy runs oneDPL functions on "
              << cpu_policy.queue().get_device().
                                  get_info<sycl::info::device::name>()
              << std::endl;
}
$ icpx -fsycl onedpl_policy_example.cpp -o onedpl_example
$ ./onedpl_example

GPU policy runs oneDPL functions on Intel(R) UHD Graphics [0x9a60]
CPU policy runs oneDPL functions on 11th Gen Intel(R) Core(TM) i9-11900KB @ 3.30GHz
```

以前のプログラムは、ビルトイン CPU セレクターとビルトイン GPU セレクターを使用してキューを作成し、これらのキューを使用して oneDPL 実行ポリシーを設定しています。前に示したように、プラットフォームとデバイスを照会して、キューをインスタンス化することもできます。

```
...
    auto platforms = sycl::platform::get_platforms();

    sycl::queue Q1(platforms[3].get_devices()[0]);
    auto gpu_policy = oneapi::dpl::execution::make_device_policy(Q1);

    sycl::queue Q2(platforms[1].get_devices()[0]);
    auto cpu_policy = oneapi::dpl::execution::make_device_policy(Q2);
...
```

繰り返しますが、前の例ではデバイスが直接指定されているため、必ずシステムのプラットフォーム・インデックスを更新してください。

この記事では、SYCL* が提供するデバイス検出向けの機能と、プログラムでプラットフォームとデバイス情報を使用する方法の概要を説明しました。マルチデバイス・システムの普及とともに、プログラマーが特定のデバイス向けのアルゴリズムをターゲットにする機会は増えています。この続きは、今後の The Parallel Universe で詳しく説明する予定です。

関連情報

- インテル® デベロッパー・クラウド（英語）– さまざまなインテルのハードウェアで oneAPI をテストするために必要なものがすべてまとめられており、無料で利用できます。
- SYCL* 2020 リファレンス・ガイド（PDF）– SYCL* 2020 仕様が要約されたリファレンスです。この記事では、デバイスの選択、プラットフォーム・クラス、コンテキスト・クラス、およびデバイスクラスの説明を参照しました。
- 『Data Parallel C++ : Mastering DPC++ for Programming Heterogeneous Systems using C++ and SYCL』（英語）の「第 12 章 : デバイス情報」– 一部のコード例の構文は古くなっていますが、デバイス検出の概要がよく分かります。
- oneAPI-samples（英語）リポジトリーには、SYCL* と oneAPI ライブラリーを使用したプログラミングを説明する数多くのサンプルコードがあります。
- 『インテル® oneAPI プログラミング・ガイド』は、インテルの oneAPI ツールの基本的な概要を示します。
- 『oneAPI GPU 最適化ガイド』は、oneAPI を使用して最高のパフォーマンスを達成するためのコーディングに関するアドバイスを含むドキュメントです。