ベータ版 oneAPI for AMD* GPU 2023.0.0 ガイド

この記事は、CodePlay 社の許可を得て iSUS (IA Software User Society) が作成した 2023 年 1 月 19 日時点の『oneAPI for AMD* GPUs (beta) 2023.0.0』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。

ベータ版 oneAPI for AMD* GPU は、開発者が DPC++/SYCL* を利用して oneAPI アプリケーションを作成し、それらを AMD* GPU 上で実行できるようにするインテル® oneAPI ツールキット向けのプラグインです。

注意
これはベータ品質のソフトウェアであり、主要機能のほとんどが含まれていますが、まだ完全ではなく、既知および未確認のバグがあることに留意してください。サポートされる機能の詳細については、「機能」を参照してください。

このプラグインは、HIP バックエンドを DPC++ 環境に追加します。このドキュメントでは、「ベータ版 oneAPI for AMD* GPU」と「DPC++ HIP プラグイン」が同じ意味で使われています。

oneAPI の詳細については、インテル® oneAPI の概要 (英語) を参照してください。

ベータ版 oneAPI for AMD* GPU の使用を開始するには、「導入ガイド」を参照してください。

導入ガイド

- 導入ガイド
- SYCL アプリケーションのデバッグ

サポート

- 機能
- 更新履歴
- トラブルシューティング
- 使用許諾契約書 (英語)

導入ガイド

このガイドには、DPC++ と DPC++ HIP プラグインを使用して、AMD* GPU で SYCL* アプリケーションを実行する方法を説明します。

これはベータ品質のソフトウェアであり、主要機能のほとんどが含まれていますが、まだ完全ではなく、既知および未確認のバグがあることに留意してください。サポートされる機能の詳細については、「機能」を参照してください。

DPC++ に関連する一般的な情報は、「DPC++ のリソース」の節を参照してください。

ベータ版 oneAPI for AMD* GPU のインストール

サポートされるプラットフォーム

このリリースは、次のプラットフォームで検証されています。

GPU ハードウェア	アーキテクチャー	オペレーティング・システム	HIP	GPU ドライバー
AMD Radeon* Pro W6800	gfx1030	Ubuntu* 20.04.5 LTS	4.5.2	21.40.2.40502

- このリリースは各種 AMD* GPU と HIP バージョンで動作するはずですが、CodePlay は評価されていないプラットフォームでの正常な動作を保証するものではありません。
- このパッケージは Ubuntu* 20.04 でのみテストされていますが、一般的な Linux* システムにインストールできます。
- プラグインは、システムにインストールされている HIP のバージョンに依存します。HIP は Windows* と macOS* をサポートしていないため、これらのオペレーティング・システムでは ベータ版 oneAPI for AMD* GPU パッケージは利用できません。

要件

1. C++ 開発ツールインストールします。

   oneAPI アプリケーションをビルドして実行するには、C++ 開発ツールの `cmake`、`gcc`、`g++`、`make` および `pkg-config` をインストールする必要があります。

   次のコンソールコマンドは、一般的な Linux* ディストリビューションに上記のツールをインストールします。

   Ubuntu*

   ```
   $ sudo apt update
   $ sudo apt -y install cmake pkg-config build-essential
   ```

Red Hat* と Fedora*

```
$ sudo yum update
$ sudo yum -y install cmake pkgconfig
$ sudo yum groupinstall "Development Tools"
```

SUSE*

```
$ sudo zypper update
$ sudo zypper --non-interactive install cmake pkg-config
$ sudo zypper --non-interactive install pattern devel_C_C++
```

次のコマンドで、ツールがインストールされていることを確認します。

```
$ which cmake pkg-config make gcc g++
```

次のような出力が得られるはずです。

```
/usr/bin/cmake
/usr/bin/pkg-config
/usr/bin/make
/usr/bin/gcc
/usr/bin/g++
```

2. DPC++/C++ コンパイラーを含むインテル® oneAPI ツールキット 2023.0.0 をインストールします。
    - インテル® oneAPI ベース・ツールキットは、多くの利用環境に適用できます。
    - oneAPI for AMD* GPU をインストールするには、インテル® oneAPI ツールキットのバージョン 2023.0.0 が必要です。これよりも古いバージョンにはインストールできません。
3. AMD* GPU 向けの GPU ドライバーと ROCm* ソフトウェア・スタックをインストールします。
    - 例えば、ROCm* 4.5 の場合、『ROCm* インストール・ガイド v4.5』(英語) の手順に従ってください。
    - `--usecase="dkms,graphics,opencl,hip,hiplibsdk` 引数を指定して `amdgpu-install` インストーラーを起動し、必要となるすべてのコンポーネントを確実にインストールすることを推奨します。

インストール

1. ベータ版 oneAPI for AMD* GPU のインストーラー (英語) をダウンロードします。
2. ダウンロードした自己展開型インストーラーを実行します。

```
$ sh oneapi-for-amd-gpus-2023.0.0-linux.sh
```

    - インストーラーは、デフォルトの場所にあるインテル® oneAPI ツールキット 2023.0.0 のインストールを検索します。インテル® oneAPI ツールキットが独自の場所にインストールされている場合、`--install-dir /path/to/intel/oneapi` でパスを指定します。
    - インテル® oneAPI ツールキットが home ディレクトリー外にある場合、`sudo` を使用してコマンドを実行する必要があります。

環境を設定

1. 実行中のセッションで oneAPI 環境を設定するには、インテルが提供する `setvars.sh` スクリプトを source します。

   システム全体へのインストールの場合:

   ```
   $ . /opt/intel/oneapi/setvars.sh --include-intel-llvm
   ```

   プライベート・インストールの場合 (デフォルトの場所):

   ```
   $ . ~/intel/oneapi/setvars.sh --include-intel-llvm
   ```

   - `clang++` などの LLVM ツールにパスを追加するには、`--include-intel-llvm` オプションを使用します。
   - ターミナルを開くたびにこのスクリプトを実行する必要があります。セッションごとに設定を自動化する方法については、「CLI 開発向けの環境変数を設定する」(英語) など、関連するインテル® oneAPI ツールキットのドキュメントを参照してください。
2. HIP ライブラリーとツールが環境内にあることを確認します。
   - `rocminfo` を実行します。実行時の表示に明らかなエラーが認められなければ、環境は正しく設定されています。
   - 問題があれば、環境変数を手動で設定します。

     ```
     $ export PATH=/PATH_TO_ROCM_ROOT/bin:$PATH
     $ export LD_LIBRARY_PATH=/PATH_TO_ROCM_ROOT/lib:$LD_LIBRARY_PATH
     ```

     ROCm* は通常 `/opt/rocm-x.x.x/` にインストールされます。

インストールの確認

DPC++ HIP プラグインのインストールを確認するには、DPC++ の `sycl-ls` ツールを使用して、SYCL* で利用可能な AMD* GPU があることを確認します。AMD* GPU が利用できる場合、`sycl-ls` の出力に次のような情報が表示されます。

```
[ext_oneapi_hip:gpu:0] AMD HIP BACKEND, AMD Radeon PRO W6800 0.0 [HIP 40421.43]
```

- 上記のように利用可能な AMD* GPU が表示されていれば、DPC++ HIP プラグインが適切にインストールされ、設定されていることが確認できます。
- インストールや設定に問題がある場合、トラブルシューティングの「sycl-ls の出力でデバイスが見つからない場合」を確認してください。
- 利用可能なハードウェアとインストールされている DPC++ プラグインに応じて、OpenCL* デバイス、インテル® GPU、または NVIDIA* GPU など、ほかのデバイスもリストされることがあります。

サンプル・アプリケーションを実行

1. 次の C++/SYCL* コードで構成される `simple-sycl-app.cpp` ファイルを作成します。

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

int main() {
// カーネルコード内で使用する 4 つの int バッファーを作成
sycl::buffer<sycl::cl_int, 1> Buffer(4);

// SYCL* キューを作成
sycl::queue Queue;

// カーネルのインデックス空間サイズ
sycl::range<1> NumOfWorkItems{Buffer.size()};

// キューへコマンドグループ（ワーク）を送信
Queue.submit([&](sycl::handler &cgh) {

// デバイス上のバッファーへの書き込み専用アクセサーを作成
auto Accessor = Buffer.get_access<sycl::access::mode::write>(cgh);

// カーネルを実行
cgh.parallel_for<class FillBuffer>(
  NumOfWorkItems, [=](sycl::id<1> WIid) {
          // インデックスでバッファーを埋めます
          Accessor[WIid] = (sycl::cl_int)WIid.get(0);
  });
});

// ホスト上のバッファーへの読み取り専用アクセサーを作成。
// キューのワークが完了するのを待機する暗黙のバリア
const auto HostAccessor = Buffer.get_access<sycl::access::mode::read>();

// 結果をチェック
bool MismatchFound = false;
for (size_t I = 0; I < Buffer.size(); ++I) {
  if (HostAccessor[I] != I) {
  std::cout << "The result is incorrect for element: " << I
                 << " , expected: " << I << " , got: " << HostAccessor[I]
                 << std::endl;
  MismatchFound = true;
}
}

if (!MismatchFound) {
  std::cout << "The results are correct!" << std::endl;
}

return MismatchFound;
}
```

2. アプリケーションをコンパイルします。

   ```
   $ clang++ -fsycl -fsycl-targets=amdgcn-amd-amdhsa -Xsycl-target-backend --
   offload-arch=<ARCH> simple-sycl-app.cpp -o simple-sycl-app
   ```

   ARCH には GPU のアーキテクチャー (例えば `gfx1030`) を指定します。次のコマンドで確認できます。

   ```
   $ rocminfo | grep 'Name: *gfx.*'
   ```

   出力に GPU アーキテクチャーが表示されます。例えば、次のようになります。

   ```
   Name: gfx1030
   ```

3. アプリケーションを実行します。

   ```
   $ SYCL_DEVICE_FILTER=hip SYCL_PI_TRACE=1 ./simple-sycl-app
   ```

   次のような出力が得られます。

   ```
   SYCL_PI_TRACE[basic]: Plugin found and successfully loaded: libpi_hip.so
   [ PluginVersion: 11.15.1 ]
   SYCL_PI_TRACE[all]: Selected device: -> final score = 1500
   SYCL_PI_TRACE[all]: platform: AMD HIP BACKEND
   SYCL_PI_TRACE[all]: device: AMD Radeon PRO W6800
   The results are correct!
   ```

   これで、oneAPI for AMD* GPU の環境設定が確認でき、oneAPI アプリケーションの開発を開始できます。

   以降では、AMD* GPU で oneAPI アプリケーションをコンパイルして実行するための一般的な情報を説明します。

DPC++ を使用して AMD* GPU をターゲットにする

AMD* GPU 向けのコンパイル

AMD* GPU 対応の SYCL* アプリケーションをコンパイルするには、DPC++ に含まれる `clang++` コンパイラーを使用します。

例:

```
$ clang++ -fsycl -fsycl-targets=amdgcn-amd-amdhsa -Xsycl-target-backend=amdgcn-
amd-amdhsa --offload-arch=gfx1030 sycl-app.cpp -o sycl-app
```

次のフラグが必要です。

- `-fsycl`: C++ ソースファイルを SYCL* モードでコンパイルするようにコンパイラーに指示します。このフラグは暗黙的に C++ 17 を有効にし、SYCL* ランタイム・ライブラリーを自動でリンクします。
- `-fsycl-targets=amdgcn-amd-amdhsa`: AMD* GPU をターゲットに SYCL* カーネルをビルドすることをコンパイラーに指示します。

- `-Xsycl-target-backend=amdgcn-amd-amdhsa --offload-arch=gfx1030`: `gfx1030` AMD* GPU をターゲットに SYCL* カーネルをビルドすることをコンパイラーに指示します。

AMD* GPU をターゲットにする場合、GPU の特定のアーキテクチャーを指定する必要があることに注意してください。

利用できる SYCL* コンパイルフラグの詳細は、『DPC++ コンパイラー・ユーザーズ・マニュアル』(英語) を参照してください。すべての DPC++ コンパイラー・オプションの詳細は、『インテル® oneAPI DPC++/C++ コンパイラー・デベロッパー・ガイドおよびリファレンス』の「コンパイラー・オプション」(英語) を参照してください。

AMD* GPU でアプリケーションを実行

AMD ターゲットの SYCL* アプリケーションをコンパイルしたら、ランタイムが SYCL* デバイスとして AMD* GPU を選択しているか確認する必要があります。

通常、デフォルトのデバイスセレクターを使用するだけで、利用可能な AMD* GPU の 1 つが選択されます。しかし、場合によっては、SYCL* アプリケーションを変更して、GPU セレクターやカスタムセレクターなど、より正確な SYCL* デバイスセレクターを設定することもあります。

環境変数 `SYCL_DEVICE_FILTER` を設定して、利用可能なデバイスセットを限定することで SYCL* デバイスセレクターを支援できます。例えば、DPC++ HIP プラグインでサポートされるデバイスのみを許可するには、次のように設定します。

```
$ export SYCL_DEVICE_FILTER=hip
```

この環境変数の詳細については、インテル® oneAPI DPC++ コンパイラーのドキュメントで「環境変数」(英語) 参照してください。
注意: この環境変数は、今後のリリースで廃止される予定です。

DPC++ のリソース

- インテル® DPC++ の概要 (英語)
- DPC++ 導入ガイド
- DPC++ コンパイラー・ユーザーズ・マニュアル (英語)
- DPC++ コンパイラーとランタイムのアーキテクチャー設計
- DPC++ 環境変数 (英語)

SYCL* のリソース

- SYCL* 2020 仕様
- SYCL* アカデミー学習教材 (英語)
- Codingame インタラクティブ SYCL* チュートリアル (英語)
- IWOCL SYCL* トーク (英語)
- 無料の DPC++ 電子書籍 (英語)
- SYCL* の最新ニュース、学習教材、プロジェクトの紹介 (英語)

SYCL* アプリケーションのデバッグ

この節では、さまざまなデバイスで SYCL* アプリケーションをデバッグするための情報、ヒント、およびポインターについて説明します。

SYCL* アプリケーションのホストコードは、単純に C++ アプリケーションとしてデバッグできますが、カーネルデバッグのサポートやツールは、ターゲットデバイスによって異なる可能性があります。

注意
SYCL* アプリケーションに汎用性がある場合、実際のターゲットデバイスではなく、インテルの OpenCL* CPU デバイスなど、豊富なデバッグサポートとツールを備えたデバイスでデバッグしたほうが有用なことがあります。

インテルの OpenCL* CPU デバイスでのデバッグ

インテルの OpenCL* CPU デバイスを使用した DPC++ アプリケーションのデバッグについては、『インテル® oneAPI プログラミング・ガイド』の「DPC++ と OpenMP* オフロードプロセスのデバッグ」の節を参照してください。

ROCm* デバッガーのサポート

ROCm* SDK には `rocgdb` デバッガーが付属しており、HIP アプリケーションの AMD* GPU 上のカーネルをデバッグできます。

ただし、DPC++ では現在、AMD* GPU ターゲットの SYCL* カーネルに対し、適切なデバッグ情報を生成することができません。そのため、`rocgdb` を使用して SYCL* カーネルをデバッグすると、次のようなエラーが表示されれることがあります。

```
Thread 5 "dbg" hit Breakpoint 1, with lanes [0-63], main::
{lambda(sycl::_V1::handler&)#1}::operator()(sycl::_V1::handler&) const::
{lambda(sycl::_V1::id<1>)#1}::operator()(sycl::_V1::id<1>) const
(/long_pathname_so_that_rpms_can_package_the_debug_info/src/rocm-
gdb/gdb/dwarf2/frame.c:1032:
internal-error: Unknown CFA rule.
```

デバッグ情報を生成せずにアプリケーションをビルドしても、デバッガーは役立ちます。例えば、カーネルが無効なメモリーアドバイスなどのエラーをスローする場合、rocgdb を使用してプログラムを実行することができます。エラー発生時にブレークして、`disas` コマンドを使用してエラーを引き起こした場所のカーネル・アセンブリー行を確認できます。

機能

コア機能

機能	サポート
コンテキスト内の複数デバイス	いいえ
サブグループ (sub-group)	部分的 [1]
グループ関数/アルゴリズム	部分的 [1]
整数関数	はい
数学関数 (スカラー)	はい
数学関数 (ベクトル)	部分的 [1]
数学関数 (marray)	いいえ
共通関数	はい
ジオメトリー関数	はい
リレーショナル関数	はい
atomic ref	はい
オペレーティング・システム	Linux*
バッファーの再解釈	はい
stream	いいえ
デバイスイベント	いいえ
グループの非同期コピー	はい
プラットフォームの get info	はい
カーネルの get info	はい
`sycl::nan` と `sycl::isnan`	はい
デバイスセレクター	いいえ
階層的並列化	はい
ホストタスク	はい
インオーダー・キュー	はい
リダクション	部分的 [1]
キューのショートカット	はい
vec	はい
marray	はい
errc	はい
匿名カーネルラムダ	はい
機能を評価するマクロ	はい
sycl::span	はい
sycl::dynamic_extent	いいえ [2]
sycl::bit_cast	はい

機能	サポート
aspect_selector	いいえ
カーネルバンドル	いいえ
特殊化定数	いいえ

非コア機能

機能	サポート
image	いいえ
fp16 データタイプ	いいえ
fp64 データタイプ	はい
prefetch	はい
USM	ホスト、デバイス、共有
USM アトミックホスト割り当て	いいえ
USM アトミック共有割り当て	いいえ
USM システムに割り当て	はい
SYCL_EXTERNAL	いいえ
アトミックメモリーの順序付け	relaxed
アトミック・フェンス・メモリーの順序付け	いいえ
アトミック・メモリー・スコープ	work_group
アトミック・フェンス・メモリーのスコープ	いいえ
64 ビット・アトミック	いいえ
バイナリー形式	AMDGCN
デバイスのパーティション化	いいえ
ホストデバッグ可能デバイス	いいえ
オンラインコンパイル	いいえ
オンラインリンカー	いいえ
キューのプロファイル	はい
mem_advise	いいえ
バックエンド仕様	いいえ
アプリケーション・バックエンドの相互運用	いいえ
カーネル・バックエンドの相互運用	いいえ
ホストタスク (ハンドルと相互運用)	いいえ
reqd_work_group_size	いいえ
キャッシュビルド結果	いいえ
ビルドログ	いいえ
ビルトインカーネル関数	なし

拡張機能

機能	サポート
uniform	いいえ
USM アドレス空間 (デバイス、ホスト)	部分的 [3]
固定ホストメモリーの使用	はい
サブグループ・マスク (+ グループ投票)	いいえ
静的ローカルメモリー使用量照会	いいえ
sRGB イメージ	いいえ
デフォルト・プラットフォーム・コンテキスト	はい
メモリーチャネル	いいえ
最大ワークグループ照会	一部分
結合行列	いいえ
すべて制限 (restrict all)	いいえ
プロパティー・リスト (property list)	いいえ
カーネル・プロパティー (kernel properties)	いいえ
SIMD 呼び出し	いいえ
低レベルデバイス情報	いいえ
カーネルキャッシュ設定	いいえ
FPGA lsu	いいえ
FPGA reg	いいえ
データ・フロー・パイプ	いいえ
キューに投入されたバリア	いいえ
フィルターセレクター	はい
グループソート	はい
フリー関数の照会	はい
明示的な SIMD	いいえ
discard_queue_events	部分的 [1]
device_if	いいえ
device_global	いいえ
C と C++ 標準ライブラリーのサポート	いいえ
カーネルでの assert	いいえ
buffer_location	いいえ
accessor_property_list (+ no_offset、no_alias)	はい
グループ・ローカル・メモリー	はい
printf	いいえ
ext_oneapi_bfloat16	いいえ
拡張デバイス情報	いいえ

1 (1、2、3、4、5)	一部のテストで失敗
2	numeric_limits<size_t>::max() の使用
3	https://github.com/intel/llvm/pull/6289 に追加 (未テスト)

更新履歴

2023.0.0

oneAPI for AMD* GPU の最初の**ベータ**リリースです。

このリリースは、intel/llvm repository at commit 0f579ba (英語) から作成されました。

新機能

- HIP バックエンドのベータサポート

SYCL* コンパイラー

- デバイスでの `assert` をサポート
- ローカル・メモリー・アクセサーのサポート
- グループ集合関数のサポート
- `sycl::ext::oneapi::sub_group::get_local_id` のサポート

SYCL* ライブラリー

- `atomic64` デバイス機能の照会をサポート
- SYCL* キューごとに複数の HIP ストリームをサポート
- 相互運用のサポート
- `sycl::queue::submit_barrier` のサポート

トラブルシューティング

この節では、トラブルシューティングのヒントと一般的な問題の解決方法について説明します。ここで説明する方法で問題が解決しない場合は、Codeplay のコミュニティー・サポート・ウェブサイト (英語) からサポートリクエストをお送りください。完全なサポートは保証できませんが、できる限り支援させていただきます。サポートリクエストを送信する前に、ソフトウェアが最新バージョンであることを確認してください。

問題は、oneAPI DPC++ コンパイラーのオープンソース・リポジトリー (英語) からも報告できます。

icpx でビットコード出力が無効になる

現在のバージョンの `icpx` には、NVIDIA* または AMD* ターゲット向けにコンパイルする際に既知の問題があり、次のエラーが出力される場合があります。

```
LLVM ERROR: Bitcode output disabled because proprietary optimizations have been
performed.
```

これは次のリリースで修正される予定ですが、それまでは NVIDIA* または AMD* GPU をターゲットとする場合、`icpx` 実行ファイルの代わりに、このガイドで説明されている `clang++` 実行ファイルを直接使用してください。

sycl-ls の出力にデバイスが表示されない

`sycl-ls` がシステム上の期待されるデバイスを報告しない場合:

1. システムに互換性のあるバージョンの CUDA* または ROCm* ツールキット (それぞれ CUDA* と HIP プラグイン向け)、および互換性のあるドライバーがインストールされていることを確認してください。
2. `nvidia-smi` または `rocm-smi` がデバイスを正しく認識できることを確認します。
3. プラグインが正しくロードされていることを確認します。これは、環境変数 `SYCL_PI_TRACE` に `1` を設定して、`sycl-ls` を再度実行することで分かります。

   例:

   ```
   $ SYCL_PI_TRACE=1 sycl-ls
   ```

   次のような出力が得られるはずです。

   ```
   SYCL_PI_TRACE[basic]: Plugin found and successfully loaded: libpi_opencl.so
   [ PluginVersion: 11.15.1 ]
   SYCL_PI_TRACE[basic]: Plugin found and successfully loaded:
   libpi_level_zero.so [ PluginVersion: 11.15.1 ]
   SYCL_PI_TRACE[basic]: Plugin found and successfully loaded: libpi_cuda.so
   [ PluginVersion: 11.15.1 ]
   [ext_oneapi_cuda:gpu:0] NVIDIA CUDA BACKEND, NVIDIA A100-PCIE-40GB 0.0
   [CUDA 11.7]
   ```

   インストールしたプラグインが `sycl-ls` の出力に表示されない場合、`SYCL_PI_TRACE` に `-1` を設定して再度実行することで、詳細なエラー情報を取得できます。

   ```
   $ SYCL_PI_TRACE=-1 sycl-ls
   ```

   大量の出力が得られますが、次のようなエラーが表示されているか確認してください。

   ```
   SYCL_PI_TRACE[-1]:
   dlopen(/opt/intel/oneapi/compiler/2023.0.0/linux/lib/libpi_hip.so) failed
   with <libamdhip64.so.4: cannot open shared object file: No such file or
   directory>
   SYCL_PI_TRACE[all]: Check if plugin is present.Failed to load plugin:
   libpi_hip.so
   ```

- CUDA* プラグインには、CUDA* SDK で提供される `libcuda.so` と `libcupti.so` が必要です。
- HIP プラグインには、ROCm* の `libamdhip64.so` が必要です。

CUDA* または ROCm* のインストールと、環境が適切に設定されていることを確認してください。また、`LD_LIBRARY_PATH` が上記のライブラリーを検出できる場所を指しているか確認してください。

4. `SYCL_DEVICE_FILTER` または `SYCL_DEVICE_ALLOWLIST` などのデバイスフィルター環境変数が設定されていないことを確認します (SYCL_DEVICE_FILTER が設定されていると、`sycl-ls` は警告を表示します)。

不正バイナリーエラーの扱い

CUDA* または HIP をターゲットにする SYCL* アプリケーションを実行すると、特定の状況でアプリケーションが失敗し、無効なバイナリーであることを示すエラーが報告されることがあります。例えば、CUDA* の場合は `CUDA_ERROR_NO_BINARY_FOR_GPU` がレポートされる場合があります。

これは、選択された SYCL* デバイスに適切でないアーキテクチャーのバイナリーが送信されたことを意味します。この場合、次の点を確認してください。

1. アプリケーションが、利用するハードウェアのアーキテクチャーと一致するようにビルドされていることを確認してください。
   - CUDA* 向けのフラグ: `-Xsycl-target-backend=nvptx64-nvidia-cuda --cuda-gpu-arch=<arch>`
   - HIP 向けのフラグ: `-Xsycl-target-backend=amdgcn-amd-amdhsa --offload-arch=<arch>`
2. 実行時に適切な SYCL* デバイス (ビルドされたアプリケーションのアーキテクチャーに一致するもの) が選択されていることを確認します。環境変数 `SYCL_PI_TRACE=1` を設定すると、選択されたデバイスに関連するトレース情報を表示できます。以下に例を示します。

```
SYCL_PI_TRACE[basic]: Plugin found and successfully loaded: libpi_opencl.so
[ PluginVersion: 11.16.1 ]
SYCL_PI_TRACE[basic]: Plugin found and successfully loaded:
libpi_level_zero.so [ PluginVersion: 11.16.1 ]
SYCL_PI_TRACE[basic]: Plugin found and successfully loaded: libpi_cuda.so
[ PluginVersion: 11.16.1 ]
SYCL_PI_TRACE[all]: Requested device_type: info::device_type::automatic
SYCL_PI_TRACE[all]: Requested device_type: info::device_type::automatic
SYCL_PI_TRACE[all]: Selected device: -> final score = 1500
SYCL_PI_TRACE[all]: platform: NVIDIA CUDA BACKEND
SYCL_PI_TRACE[all]: device: NVIDIA GeForce GTX 1050 Ti
```

3. 誤ったデバイスが選択されいる場合、環境変数 `SYCL_DEVICE_FILTER` を使用して SYCL* デバイスセレクターが選択するデバイスを変更できます。インテル® oneAPI DPC++/C++ コンパイラーのドキュメントにある「環境変数」(英語) の節を参照してください。

   注意
   `SYCL_DEVICE_FILTER` 環境変数は、今後のリリースで廃止される予定です。

oneAPI for AMD* GPU 使用許諾契約書

重要 - ソフトウェアを複製、インストール、または使用する前に使用許諾契約書 (英語) をお読みになり、同意する必要があります。