



SYCL* ベースの oneAPI プログラミング

2021 年 5 月

IA Software User Society (iSUS)

編集長 すがわら きよふみ

内容

- oneAPI 概要
- SYCL* の概要
- コンパイラーの仕組み
- oneAPI ライブラリーの例
- 関連ドキュメント

oneAPI 業界イニシアチブ

独自のアーキテクチャーに代わる選択肢

C++ および SYCL* 標準ベースのクロスアーキテクチャー言語
ドメイン固有の関数を高速化するように設計された強力なライブラリー
低水準ハードウェア抽象化レイヤー
コミュニティと業界の協力を促進するオープンな環境
複数のアーキテクチャーとベンダーでコードを再利用可能



経済的および技術的な負担を伴う独自のプログラミング・モデルに代わって、高速なコンピューティングを実現する生産性に優れたスマートなパス



データ並列 C++ (DPC++)

標準ベースのクロスアーキテクチャー言語

DPC++ = ISO C++ と Khronos SYCL*

CPU とアクセラレーターに並列処理、生産性、パフォーマンスを提供

- ハードウェアの機能を活用して高速なコンピューティングを実現
- ターゲット・ハードウェア間でコードの再利用が可能、特定のアクセラレーター向けのカスタム・チューニングを行うことが可能
- 単一アーキテクチャー専用のソリューションに代わる、オープンな業界共通のソリューションを提供

C++ および SYCL* ベース

- C++ の生産性の利点を提供、一般的で使い慣れた C および C++ 構造を使用
- The Khronos Group の SYCL* を継承し、データ並列処理とヘテロジニアス・プログラミングをサポート

コミュニティ・プロジェクトを通じて言語の拡張を推進

- データ並列プログラミングを簡素化する拡張機能を提供
- オープンな共同開発により継続的に進化

ダイレクト・プログラミング:
データ並列 C++

コミュニティによる拡張

Khronos SYCL*

ISO C++

次のハードウェア・プラットフォーム向けのソフトウェア
変更ではなく、次のイノベーションにスキルを活用できる

強力な oneAPI ライブラリー

- 主要なドメイン固有の関数を高速化するように設計
- 最高のパフォーマンスが得られるように各プラットフォーム向けに事前に最適化

oneAPI マス・カーネル・
ライブラリー
(oneMKL)

oneAPI ディープ・ニューラル・
ネットワーク・ライブラリー
(oneDNN)

oneAPI ビデオ・プロセッシング・
ライブラリー
(oneVPL)

oneAPI データ・アナリティクス・
ライブラリー
(oneDAL)

oneAPI スレッディング・
ビルディング・ブロック
(oneTBB)

oneAPI コレクティブ・コミュ
ニケーション・ライブラリー
(oneCCL)

oneAPI DPC++ ライブラリー
(oneDPL)

SYCL* 2020

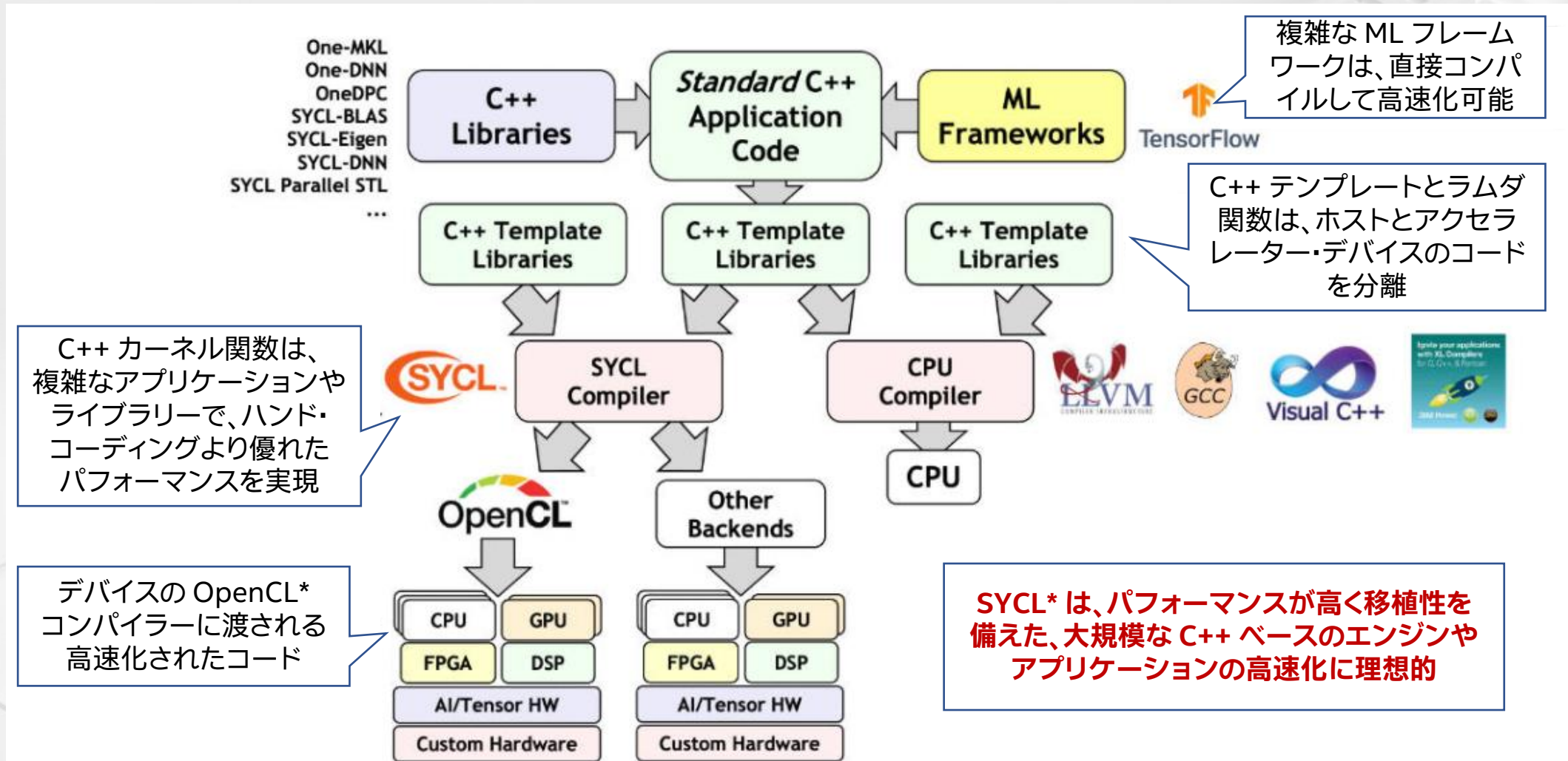


SYCL* 2020 は、2020 年後半にワーキンググループによって承認され、2021 年初頭に The Khronos Group によって公開された SYCL* 仕様の最新リリースです

OpenCL* に基づく最後のバージョンである SYCL* 1.2.1 に続くものです。より一般化されたバックエンド・モデルへの移行と、進化する ISO C++ のリリースに合わせて、プロジェクトは年ベースで更新されます

- 統合共有メモリー (USM)、ポインターを含むコードがバッファやアクセサなしで機能できるようにします
- 並列リダクション、ビルトイン・リダクション操作の追加、定形コードセクションの回避を支援、ビルトイン操作を備えたハードウェアに最大限のパフォーマンスを提供
- ワークグループとサブグループのアルゴリズムは、ワーク項目間の効率的な操作を可能にします
- クラス・テンプレートのインスタンス化を可能にするクラス・テンプレート引数控除 (CTAD) および控除ガイド・ビルトイン・リダクション操作を追加するアクセサの簡素化により、定形コードセクションの負担が軽減され、C++ パターンの簡素化が可能になります
- さまざまなバックエンドとの相互運用性が拡張され、OpenCL* 以外のバックエンドのサポートが可能になりました
- 並列プログラミングをより自由にする C++ アトミックに類似したアトミック操作

SYCL* の概要

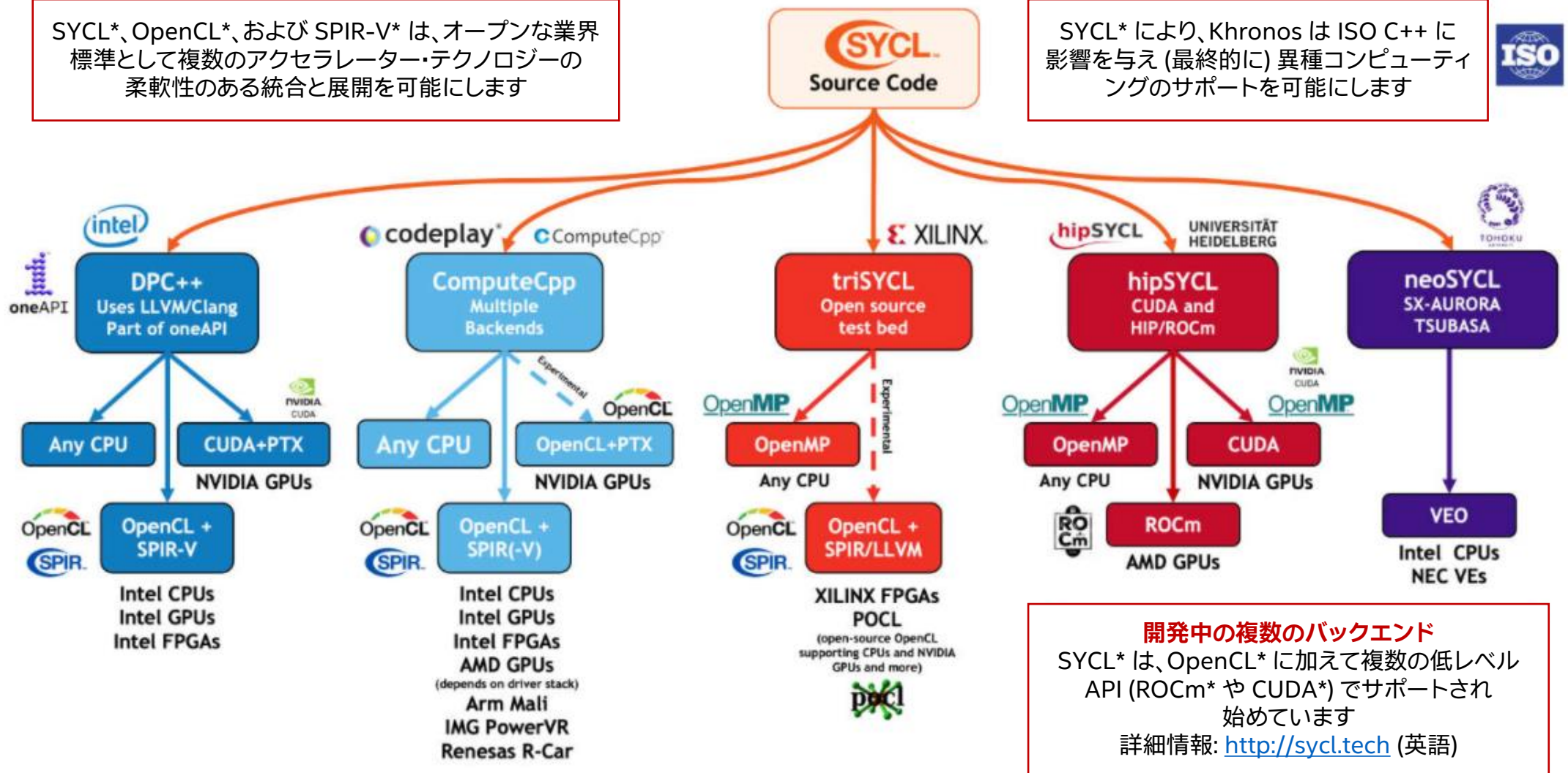


SYCL* 単一ソース C++ 並列プログラミング

SYCL* の実装

SYCL*, OpenCL*, および SPIR-V* は、オープンな業界標準として複数のアクセラレーター・テクノロジーの柔軟性のある統合と展開を可能にします

SYCL* により、Khronos は ISO C++ に影響を与え (最終的に) 異種コンピューティングのサポートを可能にします



開発中の複数のバックエンド
 SYCL* は、OpenCL* に加えて複数の低レベル API (ROCm* や CUDA*) でサポートされ始めています
 詳細情報: <http://sycl.tech> (英語)

コンパイラーの仕組み

```
$ dpcpp sample.cpp -o sample
```

```
#include <CL/sycl.hpp>
#include <iostream>
constexpr int num=16;
using namespace cl::sycl;

int main() {
    auto R = range<1>{ num };
    buffer<int> A{ R };

    queue{}.submit([&](handler& h) {
        auto out =
            A.get_access<access::mode::write>(h);
        h.parallel_for(R, [=](id<1> idx) {
            out[idx] = idx[0];
        });
    });
    auto result =
        A.get_access<access::mode::read>();
    for (int i=0; i<num; ++i)
        std::cout << result[i] << "¥n";

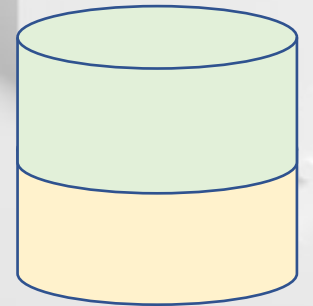
    return 0;
}
```

ホストコード

デバイスコード

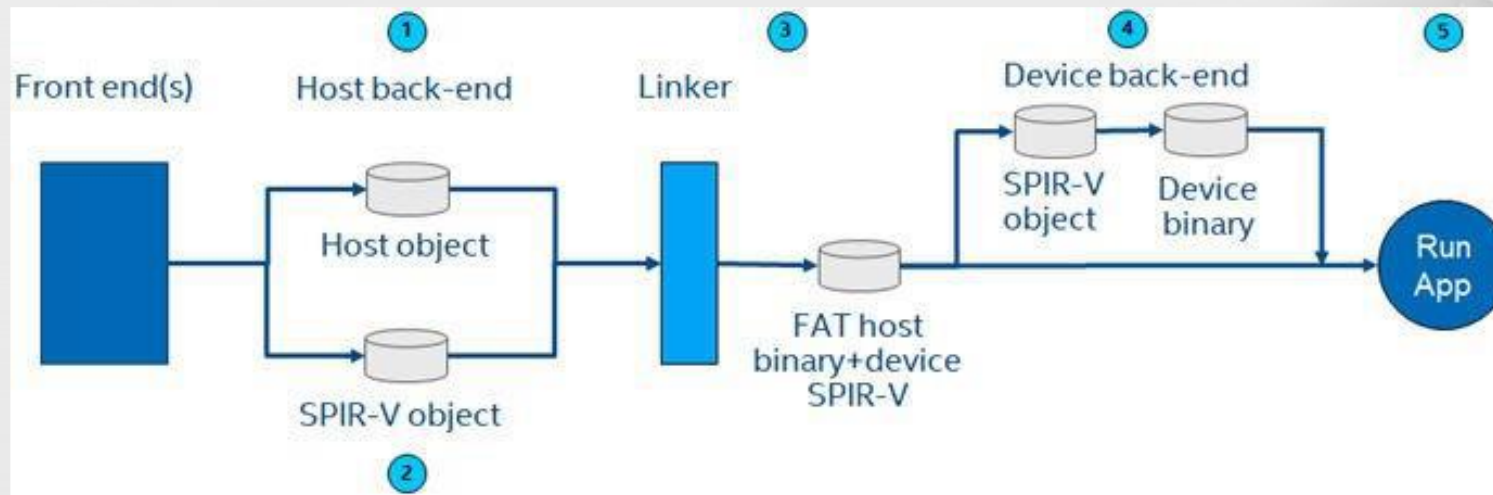
ホストコード

FAT バイナリー
.exe または a.out



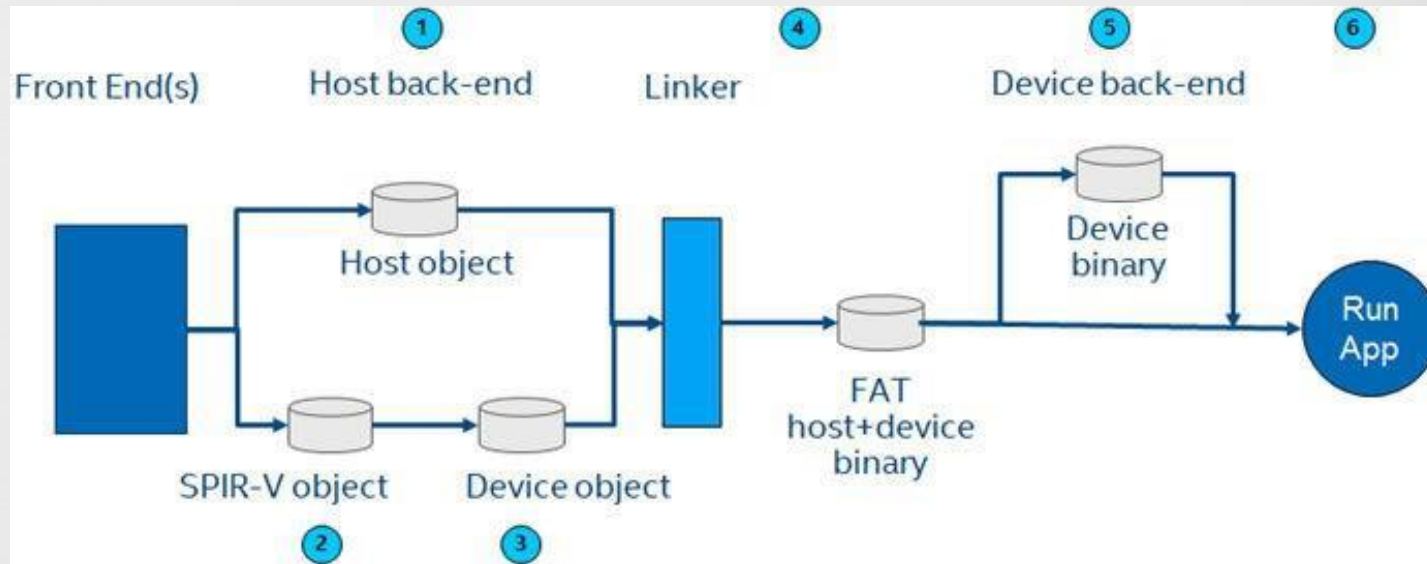
JIT コンパイル
AOT コンパイル

ジャストインタイム (JIT) コンパイル



1. ホストコードは、バックエンドでオブジェクト・コードに変換される
2. デバイスコードは SPIR-V* 形式に変換される
3. リンカーは、ホスト・オブジェクト・コードとデバイス SPIR-V* を組み合わせた、(SPIR-V* が埋め込まれた) ホスト実行コードを含むファットバイナリーを生成
4. **実行されると次のように処理される**
 - a. ホスト上のデバイスランタイムは、デバイスの SPIR-V* をデバイスのバイナリーに変換
 - b. 変換されたデバイスバイナリーはデバイスへロードされる

事前 (AOT) コンパイル



1. ホストコードは、バックエンドでオブジェクト・コードに変換される
2. デバイスコードは SPIR-V* 形式 (SPIR-V.obj) に変換される
3. デバイスの SPIR-V* は、ユーザーが指定したデバイス向けのデバイス・コード・オブジェクトに変換される
4. リンカーは、ホスト・オブジェクト・コードとデバイス・オブジェクト・コードを組み合わせた、デバイスバイナリーが埋め込まれたホスト実行コードを含むファットバイナリーを生成
5. 実行時に、デバイスバイナリーはデバイスへロードされる

oneMKL の例: 行列乗算

バッファ API

```
using mkl::blas::gemm;
int64_t n = 32;

device dev({host, cpu, gpu}_selector());
queue Q(dev);

double *A = ..., *B = ..., *C = ...;

buffer<double, 1> A_buf{A, range<1>(n * n)},
                 B_buf{B, range<1>(n * n)},
                 C_buf{C, range<1>(n * n)};

gemm(Q, mkl::transpose::N, mkl::transpose::N,
      n, n, n, 1.0, A_buf, n, B_buf, n,
      0.0, C_buf, n);
```

デバイス・
セットアップ

行列の準備

$C = A * B$

USM API

```
using mkl::blas::gemm;
int64_t n = 32;

device dev({host, cpu, gpu}_selector());
queue Q(dev);

size_t bytes = n * n * sizeof(double);
double *A = malloc_shared(bytes, Q,
                           Q.get_context());
double *B = malloc_shared(...);
double *C = malloc_shared(...);

gemm(Q, mkl::transpose::N, mkl::transpose::N,
      n, n, n, 1.0, A, n, B, n,
      0.0, C, n);

Q.wait_and_throw();
```

oneAPI の利点: まとめ


- 1つのソースコードから CPU、GPU、FPGA、アクセラレーターに対応するバイナリーを生成することが可能
- 既存のインテル® ライブラリーを使用するアプリケーションは、容易に oneAPI 環境への移行が可能
- インテル® Advisor、インテル® VTune™ プロファイラー、インテル® Inspector など使い慣れたインテル® ソフトウェア開発製品を利用可能

ドキュメント

• SYCL* 2020 API リファレンス・ガイド

<https://www.isus.jp/isus-translated-documents/>

SYCL 2020 API リファレンス・ガイド (日本語 翻訳版) ページ 1



SYCL™ (「シクル」と読みます) は、汎用プログラミングにより、さまざまなデバイスでカーネルコードの最適化を加速し、高レベルのアプリケーション・ソフトウェアを開発者にコーディングすることを可能にします。

開発者はネイティブ・アクセラレーション API よりも高レベルでプログラミングを行います。高レベルのアクセラレーション API とシミュレーションに統合することで、常に低レベルのコードにアクセスできます。

このリファレンス・ガイドのすべての定義は `sycl` 名前空間にあります。
[n.n] は、kronos.org/registry/sycl にある SYCL 2020 (リビジョン 2) 仕様のセクション番号を示します。

共通インターフェイス [4.5.2]

7 は、`accessor`、`buffer`、`context`、`device`、`image`、`event`、`host_accessor`、`host_sampler`、`image_accessor`、`kernel`、`kernel_bundle`、`local_accessor`、`platform`、`queue`、`[un]sampled_image`、`sampled_image_accessor` です。

```
using T&rh;
T&rh;
T&operator=(const T&rh);
T&operator(T&rh);
~T();
friend bool operator==(const T&rh, const T&rh);
friend bool operator!=(const T&rh, const T&rh);
```

共通値セマンティクス [4.5.2]

7 は、`id`、`range`、`item`、`nd_item`、`h_item`、`group`、`sub_group` または `nd_range` です。

```
friend bool operator==(const T&rh, const T&rh);
friend bool operator!=(const T&rh, const T&rh);
```

プロパティ [4.5.4]

次に示す SYCL ランタイムの各コンストラクターには、プロパティ名を各 `property` `list` を提供する (ラスタラー) があります: `accessor`、`buffer`、`host_accessor`、`host_sampler`、`image_accessor`、`context`、`local_accessor`、`queue`、`[un]sampled_image`、`[un]sampled_image_accessor`、`stream`、および `usm_allocator`。

```
template <typename propertyT>
struct is_property;
template <typename propertyT>
inline constexpr bool is_property_v = is_property<propertyT>::value;
template <typename propertyT, typename syclObjectT>
struct is_property_of_template <typename propertyT,
typename syclObjectT>;
inline constexpr bool is_property_of_v = is_property_of<propertyT, syclObjectT>::value;
class T {
template <typename propertyT>
bool has_property() const;
template <typename propertyT>
propertyT get_property() const;
public:
template <typename... propertyTN>
propertyT listPropertyTN...();
};
```

デバイス選択 [4.6.1]

デバイスの選択は、すでにデバイスの特定のインスタンスを保持していない場合は、デバイスセレクターを使用して実行されます。デバイスセレクターの実際のインターフェイスは、呼び出し可能なデバイス定数多項式を返し、数値的に `int` に変換可能な値を返します。システムは各デバイスの定数を呼び出し、最も高い値のデバイスを選択します。

事前定義 SYCL デバイスセレクター

default_selector_v	ヒューリスティックによって選択されたデバイス。
gpu_selector_v	Info:device:device_type:gpu デバイスタイプによってデバイスを選択。
cpu_selector_v	Info:device:device_type:cpu デバイスタイプによってデバイスを選択。
accelerator_selector_v	アクセラレーターデバイスを選択。

SYCL アプリケーションの構造 [3.2]

以下は、OpenCL アクセラレーターで実行されるジョブをスケジューリングする典型的な SYCL アプリケーションの例です。この例の USM パージョンは、このリファレンス・ガイドの 15 ページ目にあります。

```
#include <iostream>
#include <sycl/sycl.hpp>
using namespace sycl; // (オプション) SYCL 名の前に "sycl:" を // 記述する必要はない
int main() {
int data[1024]; // 処理するデータを割り当て
queue myQueue; // ワークを送信するデフォルト・キューを作成
// すべての SYCL ワークを 1 ブロックで実行することで、 // resultBuf のデストロイヤーが待機するため、ブロックを終了する前に // すべての SYCL タスクを完了させる必要がある
// データ定数をバッファーで覆う
buffer<int, 1> resultBuf ( data, range<1> { 1024 } );
// コマンドグループを作成して、キューにコマンドを発行
myQueue.submit([&handler & cgh] {
// 初期化してバッファーへのアクセスを要求
accessor writeResult { resultBuf, cgh, write_only, no_init };
// 1024 個のワーク項目を持つ parallel_for タスクをキューに送信
cgh.parallel_for(1024, [=]auto idx) {
// 0 から始まるランク番号でそれぞれのバッファー要素を初期化
writeResult[idx] = idx;
}; // カーネル初期化の終わり
}); // キューコマンドの終わり
// スコープの終了、キューに送信されたワークの完了を待機
// 結果を出力
for (int i = 0; i < 1024; i++) {
std::cout << "data[" << i << "] = " << data[i] << std::endl;
return 0;
}
```

ヘッダーファイル
この例で示す SYCL の機能を利用するには、SYCL プログラムで `<sycl/sycl.hpp>` ヘッダーファイルを含める必要があります。

名前空間
SYCL 名は `sycl` 名前空間で定義されています。

キュー
この行は、実行の基礎となるデバイスを暗黙的に選択します。キュークラス定数 [4.6.5] については、2 ページ目をご覧ください。

バッファー
カーネルが使用するすべてのデータは、バッファーまたはイメージに格納しなければなりません。そうでなければ USM を使用します。バッファークラス定数 [4.7.2] については、3 ページ目をご覧ください。

アクセラレーター
アクセラレータークラス定数 [4.7.6.4] については、4 および 5 ページ目をご覧ください。

ハンドラー
ハンドラークラス定数 [4.9.4] については、9 ページ目をご覧ください。

スコープ
カーネルスコープは、デバイスコンパイラーによってコンパイルされ、デバイス上で実行される単一のカーネル関数を指定します。

コマンドグループ・スコープ
コマンドグループ・スコープは、コマンドグループ・スコープ外のすべてのコードを指定します。

アプリケーション・スコープ
アプリケーション・スコープは、コマンドグループ・スコープ外のすべてのコードを指定します。

プラットフォーム・クラス [4.6.2]

プラットフォーム・クラスは、カーネル関数を実行する単一のプラットフォームをカプセル化します。プラットフォームは、単一のコンパイル・ユニットに実行されます。

```
platform();
template <typename DeviceSelector>
explicit platform(const DeviceSelector & deviceSelector);
backend get_backend() const noexcept;
std::vector<device> get_devices();
info:device_type = info:device_type::all() const;
template <typename param>
typename param::return_type get_info() const;
template <typename param>
typename param::return_type get_backend_info() const;
bool has_aspect(aspect) const;
static std::vector<platform> get_platforms();
```

事前定義 SYCL デバイスセレクター

version	戻り値: std::string
platformname	
platformvendor	

デバイスクラス [4.6.2]

デバイスクラスは、カーネル関数を実行する単一のデバイスをカプセル化します。デバイスクラスは、メンバ関数で初期化されます。

```
device();
template <typename DeviceSelector>
explicit device(const DeviceSelector & deviceSelector);
backend get_backend() const noexcept;
platform get_platform() const;
bool is_cpu() const;
bool is_gpu() const;
bool is_accelerator() const;
template <typename param>
typename param::return_type get_info() const;
template <typename param>
typename param::return_type get_backend_info() const;
bool has_aspect(aspect) const;
template <info:partition, property prop>
std::vector<device> create_sub_devices(size_t count) const;
```

また、9 ページ目のリダクション・カーネルの作成方法の例と、16 ページ目のカーネルの呼び出し方法の例も参照してください。

©2021 Khronos Group - Rev. 0221 www.khronos.org/sycl

ドキュメント

- SYCL* 2020 API リファレンス・ガイド
<https://www.isus.jp/isus-translated-documents/>
- SYCL* 2020 仕様 (リビジョン 3) 翻訳中
<https://www.khronos.org/registry/SYCL/> (英語)



SYCL™ 2020 仕様 (レビジョン 2)

Khronos® SYCL™ ワーキンググループ

2021-02-09 04:56:42Z: git SYCL-2020/final-rev2 commit から:
9bc5a11106cc1e17e97f2abd5cde73828b2c6bb8

この日本語マニュアルは、Khronos Group のウェブサイト:

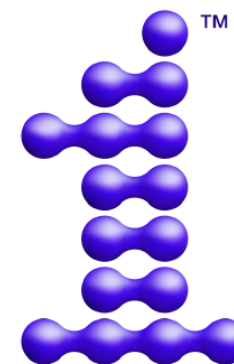
<https://www.khronos.org/registry/SYCL/> (英語) で公開されている『SYCL™ 2020 Specification (revision 2)』の参考訳です。

Khronos Group の許可を得て iSUS (IA Software User Society) が翻訳版を作成した iSUS の著作物です。

原文は Khronos Group の Copyright であり、日本語参考訳版にも適用されます

ドキュメント

- SYCL* 2020 API リファレンス・ガイド
<https://www.isus.jp/isus-translated-documents/>
- SYCL* 2020 仕様 (リビジョン 3) 翻訳中
<https://www.khronos.org/registry/SYCL/> (英語)
- oneAPI 仕様リリース 1.0 Rev3 Part1
<https://www.isus.jp/products/oneapi/oneapi-spec-japanese/>



oneAPI

oneAPI 仕様

リリース 1.0 リビジョン 3 日本語参考訳 Part 1 (1 - 311 ページ)

インテル

2020年11月23日

注意事項:

1. この日本語マニュアルは、インテル コーポレーションのウェブサイト:
<https://spec.oneapi.com/versions/latest/versions.html> (英語) で公開されている『oneAPI Specification Release 1.0-rev-3』の参考訳です。インテル社の許可を得て ISUS (IA Software User Society) が翻訳版を作成した ISUS の著作物です。原文はIntel Corporation の Copyright であり、日本語参考訳版にも適用されます。
2. 英語版と日本語版のページ番号は必ずしも同一ではありません。

ドキュメント

- SYCL* 2020 API リファレンス・ガイド
<https://www.isus.jp/isus-translated-documents/>
- SYCL* 2020 仕様 (リビジョン 3) 翻訳中
<https://www.khronos.org/registry/SYCL/> (英語)
- oneAPI 仕様リリース 1.0 Rev3 Part1
<https://www.isus.jp/products/oneapi/oneapi-spec-japanese/>
- インテル® oneAPI プログラミング・ガイド
<https://www.isus.jp/products/oneapi/oneapi-programming-guide-released/>



インテル® oneAPI プログラミング・ガイド

Intel Corporation

www.intel.com

著作権と商標について

注意事項:

この日本語マニュアルは、インテル コーポレーションのウェブサイト
<https://software.intel.com/content/www/us/en/develop/documentation/oneapi-programming-guide/top.html> (英語) で
公開されている「Intel® oneAPI Programming Guide」(バージョン 2021.1、更新日 2020/12/04) の参考訳です。

インテル社の許可を得て iSUS (IA Software User Society) が翻訳版を作成した iSUS の著作物です。

原文は Intel Corporation の Copyright であり、日本語参考訳版にも適用されます。

ドキュメント

- SYCL* 2020 API リファレンス・ガイド
<https://www.isus.jp/isus-translated-documents/>
- SYCL* 2020 仕様 (リビジョン 3) 翻訳中
<https://www.khronos.org/registry/SYCL/> (英語)
- oneAPI 仕様リリース 1.0 Rev3 Part1
<https://www.isus.jp/products/oneapi/oneapi-spec-japanese/>
- インテル® oneAPI プログラミング・ガイド
<https://www.isus.jp/products/oneapi/oneapi-programming-guide-released/>
- インテル® oneAPI ポーティング・ガイド
<https://www.isus.jp/products/c-compilers/oneapi-porting-guide-japanese/>



インテル® oneAPI ポーティング・ガイド DPCPP または ICX へ移行する ICC ユーザー向け

IntelCorporation

www.intel.com (英語)

著作権と商標について

注意事項:

この日本語マニュアルは、インテルコーポレーションのウェブサイト <https://software.intel.com/content/www/us/en/develop/articles/porting-guide-for-icc-users-to-dpcpp-or-icx.html> (英語) で公開されている『Porting Guide for ICC Users to DPCPP or ICX』(更新日: 2020/12/09) の参考訳です。

インテル社の許可を得て ISUS (IA Software User Society) が翻訳版を作成した ISUS の著作物です。

原文は Intel Corporation の Copyright であり、日本語参考訳版にも適用されます。

ドキュメント

- SYCL* 2020 API リファレンス・ガイド
<https://www.isus.jp/isus-translated-documents/>
- SYCL* 2020 仕様 (リビジョン3) 翻訳中
<https://www.khronos.org/registry/SYCL/> (英語)
- oneAPI 仕様リリース 1.0 Rev3 Part1
<https://www.isus.jp/products/oneapi/oneapi-spec-japanese/>
- インテル® oneAPI プログラミング・ガイド
<https://www.isus.jp/products/oneapi/oneapi-programming-guide-released/>
- インテル® oneAPI ポーティング・ガイド
<https://www.isus.jp/products/c-compilers/oneapi-porting-guide-japanese/>
- oneAPI GPU 最適化ガイド
<https://www.isus.jp/products/oneapi/oneapi-gpu-optimization-guide-released/>



intel.

oneAPI GPU 最適化ガイド

2021年3月



ハイパフォーマンス・ソフトウェア・ カンファレンス・オンライン 2021【春】

Intel、インテル、Intel ロゴ、その他のインテルの名称やロゴは、Intel Corporation またはその子会社の商標です。

*その他の社名、製品名などは、一般に各社の商標または登録商標です。

製品および性能に関する情報: 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。注意事項の改訂 #20201201

© 2021 Intel Corporation. 無断での引用、転載を禁じます。