インテル® oneAPI DPC++/C++ コンパイラー 2023.0 リリースノート

本書は、英文「Intel® oneAPI DPC++/C++ Compiler Release Notes」(英語)の日本語参考訳です。

バージョン: 2023.0
2022 年 12 月 6 日

このドキュメントでは、新機能、変更された機能、注意事項、および製品ドキュメントに記述されていない既知の問題について説明します。

動作環境

「インテル® oneAPI DPC++/C++ コンパイラーの動作環境」を参照してください。

リリースの入手方法

このサイト (英語) から手順に従ってツールキットをダウンロードし、インストール手順に従ってツールキットをインストールします。

インテル® oneAPI 2023.0、コンパイラー・リリース 2023.0

新機能と改善点

- コンパイラーは C++17 をデフォルトの C++ 言語として使用するようになりました。古いバージョンを使用したい場合は、コンパイラー・オプションでバージョンを指定する必要があります。例えば、C++14 を使用したい場合は、`-std=c++14` と指定します。
- FPGA IP オーサリング・フローをサポートしました。SYCL* コードをターゲットにして、異なるターゲットでスタンドアロン IP コンポーネントを生成し、インテル® Quartus® Prime 開発ソフトウェアのカスタム・プロジェクトに統合できます。特定のアクセラレーション・プラットフォームではなく、サポートされているインテル® FPGA デバイスファミリー (英語) または製品番号をコンパイルのターゲットにすることができます。
- FPGA 最適化レポートは、システムで生成されるループラベルの代わりにユーザー定義のループラベルをサポートするようになりました。次に例を示します。

```
LOOP1: for( int i = 0; i < 12; i++ ) {
   ...
}
```

- スタンドアロンのインテル® oneAPI FPGA レポートツールをサポートしました。
- FPGA でストールフリー・ループを利用したレイテンシー制御をサポートしました。
- FPGA で `max_reinvocation_delay` 属性をサポートしました。
- FPGA でサポートされているシミュレーターでのシミュレーション波形の表示をサポートしました。

- ESIMD のステートレス・メモリー・アクセスを強制する機能を追加しました。
- `-fsycl-force-target` コンパイラー・オプションをサポートしました。
- `[[intel::max_reinvocation_delay]]` ループ属性をサポートしました。
- 2GB を超えるオブジェクト・ファイルをリンクできる `-fsycl-huge-device-code` コンパイラー・オプションをサポートしました。
- SYCL* コンパイラーでの `.cu` ファイルのコンパイルをサポートしました。
- 整数型のグループ・コレクティブ組込み関数を実装しました。
- SYCL* 2020 の呼び出し可能デバイスセレクターを実装しました。
- SYCL* 2020 のスタンドアロン・デバイス・セレクターを実装しました。
- SYCL* 2020 のプロパティー・インターフェイスを `local_accessor`、`usm_allocator`、`accessor` および `host_accessor` クラスに追加しました。
- `fpga_simulator_selector` をサポートしました。
- `local_accessor` をサポートしました。`target::local` は非推奨になりました。
- レベルゼロ・バックエンドのデバイスメモリーを解放するクエリーをサポートしました。
- ホストで `bfloat16` と `float` 間の変換を実装しました。
- レベルゼロ PI プラグインの `ext::oneapi::property::queue::discard_events` をサポートしました。
- ESIMD エミュレーターで `lsc_atomic` をサポートしました。
- ESIMD エミュレーターで `dpas` をサポートしました。
- `imf` libdevice 組込み関数の C++ API を追加しました。
- ESIMD `lsc_block_store/load` にプレディケートを追加しました。
- ESIMD に試験的な `set_kernel_properties` API と `use_double_grf` プロパティーを追加しました。
- レベルゼロ PI プラグインに「Eager 初期化」モードを追加しました。プラグインにより不要な作業が行われる可能性がありますが、ホットなパスとレポート可能なパスで最も高速な実行が保証されます。
- `group::get_linear_id(int)` メソッドを実装しました。
- 関連付けられていないプレースホルダー・アクセサーに正しい `errc` がスローされるようになりました。
- ランタイムから OpenCL* ICD ローダーへの依存関係を削除しました。
- パーシステント・キャッシュ・メカニズムで `ZEBIN` 形式をサポートしました。
- 新しい `ZEBIN` 形式のバイナリーの識別メカニズムを追加しました。
- SYCL* 2020 準拠の `struct` 情報記述子を使用するように変更しました。一部の非推奨の情報クエリーを削除しました。
- `kernel_device_specific::max_sub_group_size` クエリーを SYCL* 2020 仕様と一致するように更新しました。古いバリアントは非推奨になりました。
- SYCL* 1.2.1 デバイスセレクターは非推奨になりました。
- サポートされていないデバイスのパーティショニングについて報告するエラーメッセージを改良しました。
- `device` と `platform` のデフォルトを `default_selector_v` にしました。
- `address_space::constant_space` は非推奨になりました。

- `sycl::exception::has_context` を `noexcept` としてマークしました。
- CPU の範囲リダクションのパフォーマンスが向上しました。
- `sycl::exception` `nothrow` コピーを構築可能にしました。
- `has_property` メソッドを `noexcept` としてマークしました。
- `event` がデフォルトで構築されている場合の `sycl::event::get_profiling_info` 例外メッセージを改良しました。
- ホストとデバイス間のカーネル・ラムダ・サイズの不一致に関する診断を (`static_assert` 形式で) 追加しました。
- ホストで使用された場合に例外をスローするように `pipes` クラスを更新しました。
- `cl_khr_fp64` 拡張機能のサポートをレポートするように ESIMD エミュレーター PI プラグインを更新しました。
- メモリー読み取り/書き込み操作で copy engine を優先するようにレベルゼロプラグインを更新しました。
- 一部のメモリー転送を最適化しました。
- レベルゼロ PI プラグインでイベントキャッシュを有効にしました。
- ディスクリート GPU で `sycl::range` を受け入れる `parallel_for` の一部のリダクションを最適化しました。
- コンテキスト内でコンテキスト・メンバーの派生デバイスを使用する機能を追加しました。OpenCL* バックエンドではまだサポートしていません。
- HW の機能を適切に表すように、`rol/ror` ESIMD 関数で使用可能な引数の型を制限しました。
- `default-constructed` イベントのコンテキストを設定する遅延メカニズムを実装しました。
- カーネルで複数のアクセスを行う多次元アクセサーのパフォーマンスが向上しました。
- FPGA ターゲットで最大 `_Bitint` のサイズを 4096 に増やしました。
- `[[intel::disable_loop_pipelining]]` 属性の非推奨メッセージを削除しました。
- デバイスコードから `__builtin_assume_aligned` を呼び出せるようになりました。
- `per_kernel` デバイスコード分割が使用されている場合のリンクステップのパフォーマンスを向上しました。
- `device_global` 変数で `SYCL_EXTERNAL` をサポートしました。
- 多くのパラメーターが使用できるように `__builtin_intel_fpga_mem` を更新しました。
- `safelen = 0` が使用できるように `ivdep` 属性を更新しました。
- Windows* の `sycl.lib` のリンクを改良しました。
- 正しくない `device_global` の使用法に対する多くの診断を実装しました。
- `libsycl.so` のライブラリー・レゾリューションを改良しました。
- 一致しないオブジェクトとリンクした場合の診断を改良しました。
- 暗黙的な変換後の浮動小数点サイズの変更に対する警告を追加しました。
- `invoke_simd` が引数を適切な型に変換するようにしました。

問題の修正

- 非推奨の `kernel::get_work_group_info` を削除しました。
- 非推奨の `get_native` クラスメソッドを削除しました。

- `intel::fpga_pipeline` 属性のサポートを終了しました。
- Windows* の SYCL* ライブラリーの名前に `MAJOR_VERSION` を追加しました。
- `sycl::program` クラスを削除しました。
- `ext::oneapi::reduction` を削除しました。
- 非推奨の `address_space` enum 値を削除しました。
- `event::get` メソッドを削除しました。
- `ext::intel` 内の `using namespace experimental` を削除しました。
- インテル固有のデバイス情報記述子を名前空間で修飾しました。
- 非推奨の `make_queue` API を削除しました。
- `sycl::get_native` および `interop::get_native_mem` 関数の戻り値の型を SYCL* 2020 仕様に準拠するように変更しました。
- `sycl::buffer_allocator` インターフェイスを SYCL* 2020 仕様に準拠するように変更しました。
- `sycl/sycl.hpp` ヘッダーから `cl` 名前空間を削除しました。
- C++17 以前のモードでの SYCL* のコンパイルのサポートを廃止しました。
- 内部リファクタリングに起因する多くの ABI-breaking を変更しました。
- FPGA 向けにコンパイルするときに、インテル® FPGA PAC D5005 がインストールされたシステムを使用してインテル® PAC インテル® Arria® 10 GX FPGA 搭載版をターゲットとする SYCL* アプリケーションをコンパイルできるようになりました。
- Windows* システムで FPGA エミュレーター・フロー向けにコンパイルするときにデバイスカーネルの呼び出しに失敗する問題を修正しました。
- `queue` ショートカットで `offset` パラメーターを使用して依存関係イベントベクトルの初期化リストを渡すことができなかったコンパイルの問題を修正しました。
- ルートデバイスではなく派生デバイス (サブデバイス) を渡すと `sycl::get_pointer_device` が例外をスローする問題を修正しました。
- カーネルバンドルをリンクした場合に発生していたメモリーリークを修正しました。
- 派生デバイス向けに作成したコンテキストを渡すと、USM フリーが例外をスローする問題を修正しました。
- 多次元 `accessor` の添字演算子を使用した場合のコンパイルの問題を修正しました。
- カーネルで複数のバッファー・リダクションを使用したときに発生していた "definition with the same mangled name" エラーを修正しました。
- GCC 11.1 以前をホスト・コンパイラーとして使用している場合の SYCL* 数学組込み関数のコンパイルの問題を修正しました。
- SYCL* 数学組込み関数 (`sycl::modf` など) で `half` のポインターが使用できないコンパイルの問題を修正しました。
- MSVC をホスト・コンパイラーとして使用している場合の `reduction` の問題を修正しました。
- 特殊な `sycl::span` を配列から初期化する場合のコンパイルの問題を修正しました。
- 特殊化定数をデバイス側で使用していないにも関わらず、プログラムに存在することにより引き起こされていたレベルゼロ PI プラグインのクラッシュを修正しました。
- レベルゼロプラグインのイベントリークを修正しました。

- レベルゼロプラグインのサブ-サブ-デバイスの問題を修正しました。
- ESIMD エミュレーターの正しくない `half` 変換の問題を修正しました。
- `abs` ESIMD 関数のコンパイルの問題を修正しました。
- C++20 モードでコンパイルしたときに出力されていた SYCL* ヘッダーの警告を修正しました。
- ESIMD で複数のビット単位のシフト操作を使用した場合のコンパイルの問題を修正しました。
- 同期フェンスが関連付けられていないコマンドリストをランタイムがリセットしようとしたときに発生するレベルゼロプラグインのクラッシュを修正しました。
- `sycl::get_native<sycl::backend::ext_oneapi_cuda>(sycl::device)` free 関数のコンパイルの問題を修正しました (#6653 (英語))。
- ホストタスクまたはホストアクセサーによりブロックされる明示的な依存関係 (`depends_on` を使用) の同期の問題を修正しました。
- バリアがキュー全体に正しく適用されないレベルゼロプラグインの問題を修正しました。
- gdb がテンプレート・パラメーターを正しく解釈できるように `accessor` を修正しました。
- 実装のヘッダーファイルでの共通のマクロ名の使用が修正されました。
- レベルゼロ・バックエンドのコマンドリストに関連するパフォーマンスの低下を修正しました。
- アーカイブのアンバンドルにより生成される一時ファイルのクリーンアップを修正しました。
- 内部リンクを使用した `device_global` 変数の最適化を修正しました。
- 異なる最適化レベルでコンパイルおよびリンクするとランタイムエラーが発生する問題を修正しました。
- `-f[no-]sycl-unnamed-lambda` コンパイラー・オプションの説明を修正しました。
- `Windows-Clang.cmake` を使用してデバッグモードで SYCL* プログラムをビルドしたときの問題を修正しました。
- ESIMD で符号なしの型を含む正しくない変換が行われる問題を修正しました。
- 名前なし ESIMD カーネルと非 ESIMD カーネルが混在するアプリケーションでのクラッシュを修正しました。
- gdb で typedef 引数を指定して `op[]` を呼び出したときの問題を修正しました。

既知の問題と制限事項

- このリリースは、以前のリリースと下位互換性がありません。このため、既存の SYCL* アプリケーションは再コンパイルしないと新しいランタイムで動作しません。
- TEAM 構造内部の入れ子の並列ループが REDUCTION 節で変数を使用していて、TEAM 構造に同じ REDUCTION 節がなく、`num_teams` 値が 1024 を超えている場合 (環境変数で明示的に設定した場合、または `thread_limit` が低くカーネルが値を超える場合)、OpenMP* プラグマを使用してインテル® GPU にオフロードすると、正しい結果が得られない可能性があります。正しくない結果を回避するには、`-mllvm -vpo-paropt-atomic-free-reduction-slm=true` を指定してコンパイルし、グローバル・メモリー・バッファーを無効にします。
- 「schedule(dynamic)」を含む OpenMP* ループ構造を含むプログラムで `opt-report` を使用すると、コンパイラーがエラーを出力する既知の問題があります。この場合、`-qopt-report` を削除してコンパイルすることを推奨します。

- 次の条件がすべて満たされている場合、SYCL* の組込みグループ・アルゴリズムが CPU または FPGA エミュレーター・デバイスで正しくない結果を生成することがあります。
    - 最も高い次元の work-group サイズが sub-group サイズより大きい。
    - グループ・アルゴリズムが work-group に適用されている。
    - グループ・アルゴリズムが、ワークグループ内のすべてのワークアイテムで同じ結果を生成する (`all_of_group`、`any_of_group`、`group_broadcast`、`reduce_over_group`)。
    - グループ・アルゴリズムがループ内で使用され、入力の変更により結果が変わることがある。例えば、次のカーネルコードは正しくない結果を生成します (既知の問題により while ループが終了しない、またはすべてのワークアイテムで acc[gid] が設定されない)。

```
cgh.parallel_for(
  sycl::nd_range<1>(8, 8),
  [=](sycl::nd_item<1> item) [[intel::reqd_sub_group_size(4)]] {
    // work-group サイズ > sub-group サイズ
    bool predicate = true;
    int gid = item.get_global_id(0);
    while (sycl::all_of_group(item.get_group(), predicate)) {
      // all_of_group を work-group に適用
      // all_of_group はグループのすべての work-items で同じ結果を生成し、
      // ループ内で使用される
      acc[gid] = 1;
      predicate = false;
      // predicate の変更により 2 番目のループ反復では all_of_group の結果が変わる
    }
  });
```

この問題を回避するには、work-group サイズを sub-group サイズと同じに設定します。

- SYCL* 2020 のバリアは SYCL* 1.2.1 よりもパフォーマンスが低下します。
- 別のコンパイルフローでフォールバック・アサートを使用している場合、`lib/libsycl-fallback-cassert.o` または `lib/libsycl-fallback-cassert.spv` に対して明示的にリンクする必要があります。
- 割り当て要求のアライメントをレベルゼロでサポートしている 64KB に制限します。
- レベルゼロ・バックエンドの次のシナリオの場合:
    1. バッファー A を使用するカーネル A はキュー A に送られます。
    2. バッファー B を使用するカーネル B がキュー B に送られます。
    3. `queueA.wait()`。
    4. `queueB.wait()`。
       DPCPP ランタイムは、バッファー A/B の割り当て解除/書き込みコマンドをホストの依存関係 (依存するコマンドをキューに入れる前に待機していた) として処理するために使用されます。これにより、これまでレベルゼロプラグインはステップ 1/2 で各キューがアイドルであることを検出して、コマンドリストをすぐに送信できました。しかし、イベント待機リストでこれらの依存関係を渡すようになったため、この処理は行われなくなりました。レベルゼロプラグインはこれらのコマンドをバッチで処理しようとするため、カーネル B の実行はステップ 4 でのみ開始されます。この問題が解決するまで、回避策として以前の動作を復元します。

- 名前とシグネチャーが OpenCL* C 組込み関数と一致するユーザー定義関数 (引数が完全に一致する場合で、戻り値の型は影響しません) は、未定義の動作を引き起こすことがあります。
- FPGA がインストールされている DPC++ システムはマルチプロセス実行をサポートしていません。コンテキストを作成すると、コンテキストに関連付けられているデバイスが開き、そのプロセスのためにデバイスがロックされます。ほかのプロセスはそのデバイスを使用できません。ランタイムが実際のデバイスを照会してその情報を取得する必要があるため、`device.get_info<>()` によるデバイスに関する一部のクエリーも、デバイスを開き、そのプロセスのためにデバイスをロックします。
- コンパイラーにより生成されるオブジェクト・ファイルの形式は、バージョン間で変わることがあります。回避策は、アプリケーションをリビルドすることです。
- `sycl::program`/`sycl::kernel_bundle` API を使用して別の翻訳単位で定義されたカーネルを参照すると、未定義の動作を引き起こします。
- バージョン 16.3.0 以前の Microsoft* Visual Studio* 2019 を使用して SYCL* アプリケーションをビルドし、ユーザーが `-std=c++14` または `/std:c++14` を指定すると、リンクエラー「`error LNK2005: "bool const std::_Is_integral<bool>" (??$_Is_integral@_N@std@@3_NB) already defined`」が発生することがあります。
- 内部定義の出力は Windows* ではサポートしていません。
- `-fpic` オプションを指定してライブラリーをビルドする場合、新しい `-ax` (自動 cpu ディスパッチ) オプションは現時点では使用できません。
- `/Fo` オプションの引数としてディレクトリーを指定できません。このオプションを使用すると、エラーメッセージ (clang-offload-bundler command failed with exit code 1) が表示されます。このリリースでは修正は利用できません。
- FPGA 向けにコンパイルするときに非常に広い struct に読み取り専用アクセサーを使用すると、コンパイル時間が長くなることがあります。長いコンパイル時間を回避するには、代わりに読み取り/書き込みアクセサーを使用します。
- 単一の dpcpp コマンド (例えば、`dpcpp -fintelfpga <other arguments> -Xshardware src/kernel.cpp`) を使用して FPGA コンパイルおよびリンクステージを実行するときに、ソースコードが現在のディレクトリーに含まれていないと、生成される FPGA 最適化レポートにソースコード・ブラウザーが表示されないことがあります。この問題を回避するには、次のように、実行ファイルを別のステージでコンパイルおよびリンクします。
  ```
  dpcpp -fintelfpga <other arguments> -Xshardware -c src/kernel.cpp -o kernel.o
  dpcpp -fintelfpga <other arguments> -Xshardware -kernel.o
  ```
- FPGA 向けにコンパイルするときに、デバイス側のライブラリーを使用すると、Windows* のデバッグサポートは利用できません。この問題を回避するには、Windows* のエミュレーター・プラットフォームでデバッガーを実行しないでください。
- このリリースでは FPGA で `modulefiles-setup.sh` スクリプトはサポートされていません。回避策として、setvars.sh スクリプト を使用してください。
- Windows* で、パス名が長いディレクトリーで FPGA デザインをコンパイルすると、失敗して次のエラーが表示される場合があります。
  dpcpp: error: fpga compiler command failed with exit code 1 (use -v to see invocation)
  (dpcpp: エラー: fpga コマンドが終了コード 1 で失敗しました (呼び出しを確認するには -v を使用します))
  NMAKE : fatal error U1077: '…\oneAPI\compiler\latest\windows\bin\dpcpp.EXE' : return code '0x1' (NMAKE : 致命的なエラー U1077: … : リターンコード '0x1')

この問題を回避するには、パス名が短いディレクトリーでデザインをコンパイルするか、TMP および TEMP 環境変数の値を短いパス (C:\temp など) に変更します。

- FPGA で `atomic_fence` 関数を使用する場合、`memory_scope::system` 条件はサポートされません。サポートされる最も広いスコープは `memory_scope::device` 条件です。現在、この問題の回避方法はありません。
- FPGA 向けにコンパイルするときに、コンパイラーが Windows* と Linux* で異なる中間表現 (IR) を生成します。この問題は、structs がアライメントされていないために発生します。例えば、Linux* で II=1 でコンパイルするデザインが、Windows* で II=10 になります。この問題を回避するには、次の例で示すように、アライメントされていない構造体を強制的にアライメントさせます。

```
// struct がアライメントされていないコード
struct Item {
  bool valid;
  int value1;
  unsigned char value2;
};

// struct を強制的にアライメント
struct Item {
  bool valid;
  bool __empty__[3];
  int value1;
  unsigned char value2;
  unsigned char __empty2__[3];
}
```

- ホストパイプを定義している場合、FPGA エミュレーターは異なる Avalon インターフェイスを認識しません。そのため、Avalon インターフェイス・タイプを指定したときに未定義の動作を引き起こすことがあります。この問題の既知の回避方法はありません。
- FPGA 向けにコンパイルするときに II クリティカル・パスの II をレデュースしようとすると、スケジューラーが正しくない II クリティカル・パスを返すことがあります。これは、コンパイラーが正しくないパスの II をレデュースし、II の目標が達成されないためです。この問題は、LSU のクリティカル・パスに複数のネガティブサイクルがある場合にのみ発生します。この問題の既知の回避方法はありません。ただし、デザインの機能は影響を受けません。パフォーマンス (QoR) は多少低下することがあります。
- FPGA デザインをシミュレートする場合、ホストチャネルを含むデザインで 2 つの信号不一致エラー (dataBitsPerSymbol および firstSymbolInHigh OrderBits) が発生することがあります。
    - 8 以外の dataBitsPerSymbol 値を指定すると、FPGA IP オーサリング・フローで dataBitsPerSymbol エラーが発生することがあります。このエラーを回避するには、dataBitsPerSymbol を 8 に設定します。
    - firstSymbolInHigh OrderBits を false に設定すると、FPGA IP オーサリング・フローで firstSymbolInHigh OrderBits エラーが発生することがあります。このエラーを回避するには、firstSymbolInHigh OrderBits を true に設定します。
- FPGA IP オーサリング・フローを使用して、生成された .prj フォルダーをインテル® Quartus® Prime 開発ソフトウェアのプロジェクト・ディレクトリーにコピーすることで、デザインを Platform Designer に直感的に統合できます。Platform Designer はプロジェクトを自動的に検出します。しかし、生成される hw.tcl ファイルには既知の問題があり、信号が正しくマッピングされません。この問題を回避するには、Linux* システムと Windows* システムの両方で、次の手順に従います。
- `$ cd <kernel_name>.prj`

- `$ python <kernel-name>_di_hw_tcl_adjustment_script.py`
    1. コマンドラインから実行できるように、python を PATH 環境変数に追加します。
    2. IP オーサリング・カーネルを Platform Designer に統合する前に、上記のコマンドを実行して、.prj ディレクトリーに生成された *<kernel-name>*_di_hw_tcl_adjustment_script.py スクリプトを実行します。
- `sycl::ext::oneapi::experimental::printf()` 関数を呼び出している FPGA カーネルをコンパイルすると、コンパイラーは次の警告メッセージを出力します。
  compiler warning: argument 'llvm_fpga_printf_buffer_start' on component '<your kernel name>' is never used by the component. Note that the compiler may optimize it away.
  この問題の既知の回避方法はありません。カーネルの機能には影響しないため、この警告は無視してかまいません。
- FPGA 向けにコンパイルするとき、SYCL* コードの固定サイズのループ (ビット幅 8、16、32、64 以外) 内に `std::popcount` 関数が含まれる場合、直接 `llvm.ctpop` にマップされてコンパイルに失敗し、エラーメッセージが表示されます。この問題の既知の回避方法はありません。ループ内で `std::popcount` 関数を使用しないことを推奨します。
- FPGA の高最適化ループでは、最小再起動遅延はループ II と同じにする必要があります。コンパイラーが指定された `max_reinvocation_delay` 値に適切な II を使用してループを実装できない場合、コンパイラーは次のエラーメッセージを出力します。
  Compiler Error: Couldn't achieve specified max_reinvocation_delay for loop, try setting -Xshyper-optimized-handshaking=off.
  指定された `max_reinvocation_delay` を達成するために高最適化ループに `-Xshyper-optimizedhandshaking=off` コマンドオプションを使用するという提案は無視してください。現在、指定された `max_reinvocation_delay` 値を達成する既知の回避方法はありません。
- FPGA IP オーサリング・フローで、サンプルコードの unrolled_loop.hpp (英語) ヘッダーファイルで定義されている `fpga_tools::UnrolledLoop` ユーティリティーはカーネル引数インターフェイス・マクロ (`mmhost`、`conduit_mmhost`、および `register_map_mmhost`) をサポートしていません。次に例を示します。

  ```
  fpga_tools::UnrolledLoop<ROWS>([&](auto row) {
    #pragma unroll
    for (int i = COLS - 1; i > 0; i--) {
      shift_reg[row][i] = shift_reg[row][i - 1];
    }
    shift_reg[row][0] = MA[col * ROWS + row];
  });
  ```

  この問題を回避するには、次の例で示すように、`for` ループの前に `#pragma unroll` を使用します。

  ```
  #pragma unroll
  for (int row = 0; row < ROWS; row++) {
    #pragma unroll
    for (int i = COLS - 1; i > 0; i--) {
      shift_reg[row][i] = shift_reg[row][i - 1];
    }
    shift_reg[row][0] = MA[col * ROWS + row];
  }
  ```

- システムにデバイスを提供しない MESA OpenCL* を実装すると、デバイスが正しく検出されないことがあります。この問題を回避するには、`/etc/OpenCL/vendor/mesa.icd` を削除して OpenCL*実装を無効にします。

- カーネルが `std::array` を使用している場合、デバッグモードの Windows* でコンパイルに失敗することがあります。この問題は、Microsoft* STL C++ ヘッダーの `std::array` のデバッグバージョンがデバイスコードに対して不正な関数を呼び出すために発生します。この問題を回避するには、次の操作を行ってください。
    1. `-###` オプションを指定して、コンパイラー・パイプライン実行文字列をダンプします。コンパイラーは、コンパイルツールの内部実行文字列を出力します。コンパイルは行われません。
    2. 文字列の最後に `-D_CONTAINER_DEBUG_LEVEL=0 -D_ITERATOR_DEBUG_LEVEL=0` オプションを追加して、(通常は) 最初の実行文字列 (`-fsycl-is-device` オプションで指定します) を変更します。すべての文字列を 1 つずつ実行します。
- オフセットを指定しないでアクセサーを作成した場合でも、`-fsycl-dead-args-optimization` でアクセサーのオフセットを削除することはできません。
- SYCL* 2020 のバリアは SYCL* 1.2.1 よりもパフォーマンスが低下します。
- 別のコンパイルフローでフォールバック・アサートを使用して、`lib/libsycl-fallback-cassert.o` または `lib/libsycl-fallback-cassert.spv` に対して明示的にリンクする必要があります。
- 割り当て要求のアライメントをレベルゼロでサポートしている 64KB に制限します。
- レベルゼロ・バックエンドの次のシナリオの場合:
    1. バッファー A を使用するカーネル A はキュー A に送られます。
    2. バッファー B を使用するカーネル B がキュー B に送られます。
    3. `queueA.wait()`。
    4. `queueB.wait()`。
       DPCPP ランタイムは、バッファー A/B の割り当て解除/書き込みコマンドをホストの依存関係 (依存するコマンドをキューに入れる前に待機していた) として処理するために使用されます。これにより、これまでレベルゼロプラグインはステップ 1/2 で各キューがアイドルであることを検出して、コマンドリストをすぐに送信できました。しかし、イベント待機リストでこれらの依存関係を渡すようになったため、この処理は行われなくなりました。レベルゼロプラグインはこれらのコマンドをバッチで処理しようとするため、カーネル B の実行はステップ 4 でのみ開始されます。この問題が解決するまで、回避策として以前の動作を復元します。
- 名前とシグネチャーが OpenCL* C 組込み関数と一致するユーザー定義関数 (引数が完全に一致する場合で、戻り値の型は影響しません) は、未定義の動作を引き起こすことがあります。
- FPGA がインストールされている DPC++ システムはマルチプロセス実行をサポートしていません。コンテキストを作成すると、コンテキストに関連付けられているデバイスが開き、そのプロセスのためにデバイスがロックされます。ほかのプロセスはそのデバイスを使用できません。ランタイムが実際のデバイスを照会してその情報を取得する必要があるため、`device.get_info<>()` によるデバイスに関する一部のクエリーも、デバイスを開き、そのプロセスのためにデバイスをロックします。
- コンパイラーにより生成されるオブジェクト・ファイルの形式は、バージョン間で変わることがあります。回避策は、アプリケーションをリビルドすることです。
- `sycl::kernel_bundle` API を使用して別の翻訳単位で定義されたカーネルを参照すると、未定義の動作を引き起こします。
- バージョン 16.3.0 以前の Microsoft* Visual Studio* 2019 を使用して SYCL* アプリケーションをビルドし、ユーザーが `-std=c++14` または `/std:c++14` を指定すると、リンクエラー「`error LNK2005: "bool const std::_Is_integral<bool>" (??$_Is_integral@_N@std@@3_NB) already defined`」が発生することがあります。

- 内部定義の出力は Windows* ではサポートしていません。

追加ドキュメント

- インテル® oneAPI ツールキット (Linux* 版) 導入ガイド (英語)
- インテル® oneAPI ツールキット (Windows* 版) 導入ガイド (英語)
- セマンティック・バージョニングに基づく oneAPI バージョン管理スキーマ (英語)
- 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 (英語)
- サポートされている SYCL* 2020 仕様の機能と DPC++ 言語拡張機能
- インテル® oneAPI DPC++/C++ コンパイラーでサポートされている OpenMP* の機能と拡張機能

以前のインテル® oneAPI リリース

- インテル® oneAPI DPC++/C++ コンパイラー 2022.1 リリースノート (PDF)
- インテル® oneAPI DPC++/C++ コンパイラー 2021.1 リリースノート (PDF)

インテル® oneAPI DPC++/C++ コンパイラーの動作環境

本書は、英文「Intel® oneAPI DPC++/C++ Compiler System Requirements」(英語)の日本語参考訳です。

バージョン: 2023.0
2022 年 11 月 23 日

はじめに

このコンパイラーおよびライブラリーのハードウェア、オペレーティング・システム、ソフトウェア要件を説明します。

ハードウェア要件

CPU (プロセッサー) の要件

次のインテル® 64 アーキテクチャー・ベースのシステムは、ホスト・プラットフォームとターゲット・プラットフォームの両方としてサポートされています。

- インテル® Core™ プロセッサー・ファミリー
- インテル® Xeon® プロセッサー・ファミリー
- インテル® Xeon® スケーラブル・プロセッサー・ファミリー

アクセラレーターの要件

- GEN9 (以降の) GPU (英語)
- FPGA (次のいずれか):
    - FPGA カスタム・プラットフォーム (英語)
    - インテル® FPGA PAC カード
        - インテル® プログラマブル・アクセラレーション・カード (インテル® PAC) インテル® Arria® 10 GX FPGA 搭載版 (英語) (サポート終了のお知らせ (英語) を参照)
        - インテル® FPGA プログラマブル・アクセラレーション・カード D5005 (インテル® FPGA PAC D5005) (英語) (サポート終了のお知らせ (英語) を参照)

- FPGA デバイス

ターゲット - インテル® FPGA デバイス	インテル® Quartus® Prime 開発ソフトウェアのエディション	注
• インテル® Agilex™ (英語) • インテル® Arria® 10 (英語) • インテル® Stratix® 10 (英語) • インテル® Cyclone® 10 GX (英語)	インテル® Quartus® Prime 開発ソフトウェア・プロ・エディション (英語)	インテル® oneAPI の各リリースは、最大 3 年間のインテル® Quartus® Prime 開発ソフトウェアのバージョンをサポートします。
• Cyclone® V (英語)	インテル® Quartus® Prime 開発ソフトウェア・スタンダード・エディション (英語)	

メモリー要件

- 16GB (CPU および GPU 開発)
- 64GB (インテル® FPGA 開発)

注: インテル® FPGA 開発 (英語) では、デザインを処理するために必要な (推奨) 物理メモリーと同等の仮想メモリーを提供するようにシステムを構成することを推奨します。

ソフトウェア要件

注: これらの OS ディストリビューションはインテルによってテストされたもの、または動作が確認されているものです。その他のディストリビューションは、動作する場合としない場合があり、推奨されません。質問がある場合は、インテル・コミュニティー・フォーラム (英語) でサポートを受けることができます。商用サポート (英語) を利用可能な場合は、サポートチケットを作成してください。

Linux*

オペレーティング・システム

CPU ホスト/ターゲット	インテル® インテグレーテッド・グラフィックス (GPU)	FPGA
• Ubuntu* 20.04 LTS、22.04 • Red Hat* Enterprise Linux* 8.x、9.x • SUSE* Linux* Enterprise Server (SLES*) 15 SP3、SP4 • Fedora* 36、37	• Ubuntu* 20.04、22.04 • RHEL (Red Hat*) 8.x、9x • SLES* 15 SP2、SP3、SP4 • Linux* カーネル 4.11 以降 • Rocky Linux* 8、9	• RHEL 9 および 8.x (注: v8.6 はサポートしていません) • Ubuntu* 22.04 LTS、20.04 LTS • SLES* 15 SP3、SP4

注 (FPGA):
- FPGA でハードウェアをコンパイルするには、インテル® Quartus® Prime 開発ソフトウェア・プロ・エディションのソフトウェアと BSP を個別にダウンロードする必要があります。詳細は、インテル® FPGA 開発フロー (英語) を参照してください。

- HWE (Hardware Enablement) カーネルを使用する Ubuntu* Server 18.04 はサポートされていません。また、このリリースでサポートする予定はありません。

開発ツール

- コンパイラーが構築されている分散ライブラリーのサポートされている最小バージョン: GCC - 7.5.0、BINUTILS- 2.30、GLIBC-2.28
- Eclipse* 開発者: Eclipse* 4.20 および 4.21
- GPU 開発:
    - Linux* オペレーティング・システム向け汎用 GPU ドライバー (英語) の最新のインテル® GPU ドライバー
- インテル® FPGA 開発 (英語):
    - インテル® Quartus® Prime 開発ソフトウェア・プロ・エディションのソフトウェア (英語) (インテル® oneAPI ツールキット (Linux* 版) 導入ガイド (英語) も参照してください)
    - ハードウェア・コンパイルに利用する FPGA デバイス (英語) またはカスタム・プラットフォーム (英語)。
    - GCC 7.5.0 以降

インテル® oneAPI DPC++ ライブラリーを使用した開発:

    - 範囲ベースの API を使用するには、C++17 および GCC 8.1 以降または Clang 7 以降に含まれる C++ 標準ライブラリーが必要です。
- RHEL で LTO (Link Time Optimization、リンク時最適化) を使用しているときにリンカーエラーが発生する既知の問題があります。この問題を解決するには、次の最小バージョンの binutils がインストールされていることを確認してください。
    - RHEL 8: binutils 2.30-82.el8
- OpenMP* 開発:
    - `libffi.so.6` (Ubuntu* 18.04 のデフォルトバージョン)。新しい OS バージョンのユーザーは、`libffi.so.6` を個別にインストールする必要があります。

インテル® インテグレーテッド・グラフィックス (GPU) 開発のサマリー

- サポートしている Linux* カーネル: 4.11 以降のデプロイメントが必要
- サポートしている Linux* OS: Ubuntu* 20.04.3 LTS、RHEL 8.x、SUSE* 15.x
- コンパイラーが構築されている分散ライブラリーのサポートされている最小バージョン: GCC - 7.5.0、BINUTILS- 2.30、GLIBC-2.28
- Linux* オペレーティング・システム向け汎用 GPU ドライバー (英語) の最新のインテル® GPU ドライバー
- ハードウェアにアクセスするには、「video」グループに属している必要があります。次のコマンドを使用して追加します。
  ```
  $ usermod -a -G video $USER
  ```

Windows*

オペレーティング・システム

CPU ホスト/ターゲット	GPU	FPGA
• Windows* 10、11 (64 ビット) • Microsoft* Windows Server* 2022、2019	• Windows* 10、11 (64 ビット) • Microsoft* Windows Server* 2019、2022	• Windows* 10、11 (64 ビット) • Microsoft* Windows Server* 2019、2022

注: FPGA でハードウェアをコンパイルするには、インテル® Quartus® Prime 開発ソフトウェア・プロ・エディションのソフトウェアと BSP を個別にダウンロードする必要があります。詳細は、インテル® FPGA 開発フロー (英語) を参照してください。

開発ツール

- GPU 開発: インテル® グラフィックス Windows* DCH ドライバー (英語) の最新のドライバー
- Microsoft* Visual Studio* 開発環境またはコマンドライン・ツールを使用して IA-32 またはインテル® 64 アーキテクチャー・アプリケーションをビルドする場合:
    - Microsoft* Visual Studio* 2019 または 2022 の Community、Enterprise、および Professional エディション (「C++ によるデスクトップ開発」コンポーネントがインストールされていること)
    - 詳細は、インテル® コンパイラーと Microsoft* Visual Studio* および Xcode* の互換性 (英語) を参照してください。

インテル® oneAPI 2023.0 は、Windows* および Linux* で検証を行っています。

Windows* インテル® グラフィックス・ドライバー

ドライバーをインストールするには、次の手順に従ってください。

インテル® Iris® Xᵉ MAX グラフィックス (DG1) および第 10 世代から第 13 世代インテル® Core™ プロセッサー・グラフィックス (英語)

インテル® Arc™ A シリーズ・グラフィックス (DG2) (英語)

インテル® データセンター GPU フレックス・シリーズ (ATS-M) (英語)

Linux* 汎用インテル® GPU (GPGPU) ドライバー

すべてのインテル® GPU は、この記事 (英語) の手順に従ってください。

インテル® レジストレーション・センターのアクセス方法は、インテル製品の担当者までお問い合わせください。

汎用インテル® GPU (GPGPU) は、この記事 (英語) を参照してください。メニューの **20220830** をクリックし、指示に従ってダウンロードおよびインストールしてください。

インテル® oneAPI ツールキット向け Visual Studio* Code (VS Code) 拡張

インテル® oneAPI ツールキット向け VS Code 拡張 (英語) は、oneAPI アプリケーションを作成、デバッグ、およびプロファイルする開発者を支援します。詳細は、「Visual Studio* Code とインテル® oneAPI ツールキットの使用ユーザーガイド」(英語) を参照してください。

VS Code Marketplace (英語) から以下の VS Code 拡張を利用できます。

- インテル® oneAPI ツールキット向けサンプルブラウザー
- インテル® oneAPI ツールキット向け環境コンフィグレーター
- インテル® oneAPI ツールキット向け解析コンフィグレーター
- インテル® oneAPI ツールキット向け GDB GPU サポート
- インテル® oneAPI ツールキット向けインテル® DevCloud コネクター

関連情報:

- インテル® oneAPI ベース・ツールキット (Linux* 版) 導入ガイド (英語)
- インテル® oneAPI ベース・ツールキット (Windows* 版) 導入ガイド (英語)
- インテル® oneAPI ベース & HPC ツールキット (macOS* 版) 導入ガイド (英語)

終了予定のサポート

- 2022.2 リリースから、第 6 世代から第 10 世代インテル® Core™ プロセッサー、および同世代の Intel Atom® プロセッサー、インテル® Pentium® プロセッサー、インテル® Celeron® プロセッサーに内蔵されているグラフィックス・プロセッサー向け Windows* ドライバーのサポートはメンテナンス・モードに移行しました。セキュリティー問題および重大な問題の修正のみ行われます。前述のプロセッサーの既存の統合グラフィックス・プロセッサー機能を使用するインテル® oneAPI のツールは引き続き動作する可能性はありますが、サポートされなくなります。これらのプロセッサーの CPU 機能は引き続きサポートされます。詳細は、oneAPI フォーラム (英語) およびリリースノートを参照してください。
- Microsoft* Visual Studio* 2017 統合は古い機能 (非推奨) で、将来のリリースで削除される予定です。
- これらのオペレーティング・システムはインテル® oneAPI 2022.1 リリースでは非推奨で、将来のリリースではサポートされなくなる予定です。
    - Windows Server* 2016
    - Ubuntu* 18.04 LTS
    - CentOS* 7
    - Fedora* 34

既知の問題

インテル® oneAPI ツールキット 2022.1.3 以前およびインテル® Parallel Studio XE (すべてのバージョン) は Microsoft* Visual Studio* 2022 をサポートしていません。Microsoft* Visual Studio* 2022 がインストールされているシステムでインテル® oneAPI およびインテル® Parallel Studio XE のインストーラーを実行すると、インストール、アップグレード、変更、アンインストールに失敗します。詳細は、「この記事」(英語) を参照してください。

法務上の注意書き

インテルのテクノロジーを使用するには、対応したハードウェア、ソフトウェア、またはサービスの有効化が必要となる場合があります。

絶対的なセキュリティーを提供できる製品またはコンポーネントはありません。

実際の費用と結果は異なる場合があります。

本資料は、(明示されているか否かにかかわらず、また禁反言によるとよらずにかかわらず) いかなる知的財産権のライセンスも許諾するものではありません。

本資料で説明されている製品には、エラッタと呼ばれる設計上の不具合が含まれている可能性があり、公表されている仕様とは異なる動作をする場合があります。現在確認済みのエラッタについては、インテルまでお問い合わせください。

インテルは、明示されているか否かにかかわらず、いかなる保証もいたしません。ここにいう保証には、商品適格性、特定目的への適合性、および非侵害性の黙示の保証、ならびに履行の過程、取引の過程、または取引での使用から生じるあらゆる保証を含みますが、これらに限定されるわけではありません。

製品および性能に関する情報

[1] 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。