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

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

2021 年 12 月 20 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。

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。

2022.0 リリース

新機能と改善点

- OpenMP* SIMD のベクトル化は、OpenMP* 言語機能が有効な場合(`-qopenmp`、`-qopenmp-simd` など)、以前は `O2` 以上のオプションでサポートされていましたが、`O0` 以上のオプションでサポートされるようになりました。
- `-fopenmp-target-simd` は、GPU での OpenMP* SIMD サポートを有効にします。
- `-fopenmp-target-simdlen=n` は、OpenMP* SIMD ループの GPU ベクトル長を指定します。
- OpenMP* 5.0 標準のターゲット `in_reduction` 節をサポートしました。
- OpenMP* 5.1 標準の `masked` 構造と `tile` 構造をサポートしました。
- 非同期オフロードの `nowait` をサポートしました。
- 新しい SYCL* 2020 機能 `sycl::logical_and` および `sycl::logical_or` をサポートしました。また、ホストタスクを完全にサポートしました。サポートされている SYCL* 2020 機能のリストは、こちら(英語)を参照してください。
- 次の DPC++ 拡張機能を追加しました。
    - 特定のハードウェア・メディアに一致するように色を調整する RGB 色の値の線形化を提供する `sRGBA`(英語)のサポート
    - DPC++ での行列プログラミング拡張機能(英語)のプレビュー実装
    - `SYCL_EXT_INTEL_BF16_CONVERSION`(英語)のサポート
- こちら(英語)にリストされているように、非推奨の SYCL* 1.2.1 API のサポートを終了しました。
- ユーザー定義型との潜在的な競合状態を回避するため、グローバル名前空間での SYCL* `half` 型のサポートを削除しました。以前は `sycl::half` 型のエイリアスでした。`::half` 型がないことによるコンパイルエラーを解決するには、`sycl::half` 型を直接使用する必要があります。

- DPC++ アプリケーションのインクリメンタル・ビルドの時間を高速化する試験的な機能を追加しました。この機能を有効にするには、`-fsycl-max-parallel-link-jobs=<N>` コンパイラー・オプションを指定します。このオプションは、DPC++ アプリケーションのリンクに必要なアクションを実行するため、指定された数までのプロセスを同時に生成できることをコンパイラーに伝えます。
- 以前のコンパイラー・リリースでは、`bin` ディレクトリーにすべての LLVM ツールが含まれていました。このディレクトリーを `PATH` に追加すると、一部のバイナリーがシステムのほかの LLVM インストールと競合することが判明したため、`bin-llvm` ディレクトリーに移動しました。コンパイラー・ドライバー (`dpcpp`/`icx`/`icpx`/`ifx`) は、ユーザーがこれらの内部ツールを見つけることができるように、必要に応じて調整されます。しかし、`PATH` に含まれなくなったツールが一部のアプリケーションの Makefile (または cmake 設定) で直接呼び出されている場合、調整が必要になります。詳細は、`<…/bin/>../bin-llvm/README` を参照してください。
- コンパイラーは、Windows* レジストリーをデフォルトのメカニズムとして使用して、Windows* のバックエンド OpenCL* ICD を検出するようになりました。`OCL_ICD_FILENAMES` 環境変数はデバッグ専用で、Windows の管理者権限では動作しません。
- FPGA のストールフリー・クラスターの出口 FIFO レイテンシーをグローバルに制御する `-Xssfcexit-fifo-type=`*`<value>`* オプションをサポートしました。
- FPGA の隣接ループとの融合を防ぐ `nofusion` ループ属性をサポートしました。
- FPGA の読み取り専用のアクセサーで読み取り専用のキャッシュを有効にする `-Xsread-only-cache-size=`*`<N>`* オプションをサポートしました。
- FPGA 向けの `hls_float` データ型のサポートを終了し、`ap_float` データ型に置換しました。
- FPGA 向けのオープンソース・ランタイム環境をサポートしました。
- FPGA 向けの高速 BSP カスタマイズ・フローをサポートしました。
- Microsoft* Visual Studio* 2022 をサポートしました。

問題の修正

- `dpcpp` コンパイラーによって生成されるホストコンパイル中に使用される一時ソースファイルが、コンパイル中に生成されるファイルを追跡するビルド環境を壊す可能性があるソースの依存関係として表示されていた問題を修正しました。
- 入力カーネルバンドルに複数のデバイスイメージが含まれ、特殊化定数が使用されている場合、`sycl::link` API がユーザーコードの JIT コンパイルに失敗する可能性がある問題を修正しました。
- FPGA 向けにコンパイルするときに、カーネル名をローカルで宣言すると、FPGA 最適化レポートでカーネル名が正しく復号化されます。
- oneAPI 固有の GPU プラットフォームを併せてインストールした場合にコンパイルに失敗する FPGA エミュレーターの問題を修正しました。

既知の問題と制限事項

- https://dgpu-docs.intel.com/ (英語) から利用可能な最新の GPU ドライバーは、インテル® oneAPI コンパイラーを使用している場合、第 9 世代インテル® インテグレーテッド・グラフィックスで実行する OpenMP* オフロード・アプリケーションで Ahead-Of-Time (AOT) ビルドの問題を引き起こします。この問題は、今後のドライバーリリースで修正される予定です。
  この問題が発生しないバージョンのドライバーへのダウングレードについては、Graphics - インテル・コミュニティー (英語) までお問い合わせください。

- 以下の Linux* ディストリビューションでは、マルチスレッドを多用する(3 スレッド以上)GPU オフロード・アプリケーションで、ハードリセットまたはシステムの電源サイクルによってのみ回復可能なハングアップまたはタイムアウトが発生する場合があります。この問題は、古い Linux* カーネルの不具合により、マルチスレッドを多用してインテル® GPU へデータを読み書きする際に発生します。

カーネル/ ディストリビューション	**問題あり**	**問題なし**
RedHat* Enterprise Linux*	RHEL 8.4(カーネル 4.18.0-305)以前	RHEL 8.5(カーネル 4.18.0-348)
SUSE* Linux*	SLES* 15 SP3 以前	SLES* 15 SP4 beta
Ubuntu* Linux*	Ubuntu* 20.04.03 以前	Ubuntu* 20.04.03(カーネル 5.11.0-40-generic #44~20.04.2-ubuntu)

  推奨される回避方法:不具合を解決済みの Linux* ディストリビューションにアップグレードします。ソフトウェアは動作しますが、カーネルログに警告メッセージが出力されます。
  Ubuntu* 20.04.03 用の GPU ソフトウェアは、https://dgpu-docs.intel.com(英語)から入手できます。ソフトウェアは動作しますが、カーネルログに警告メッセージが出力されます。
  RHEL 8.5 用の GPU ソフトウェアは、上記の場所で 2022 年第一四半期に提供される予定です。
  SLES* 15 SP4 用の GPU ソフトウェアは、SLES* 15 SP4 の一般提供開始後、間もなく提供される予定です。
  別の回避方法:GPU 対応アプリケーションではマルチスレッドを多用しないようにします(スレッド数を 2 以下にします)。例えば、インテル® MPI ライブラリーを使用するアプリケーションでは、マルチスレッド・バージョンの代わりに、シングルスレッド・バージョンのインテル® MPI ランタイム・ライブラリーを使用します。環境変数 `I_MPI_THREAD_SPLIT=0` を設定して、シングルスレッド・バージョンのインテル® MPI ライブラリーを使用します。
- OpenMP* 5.0 標準に準拠するように、スケジュール種別が `dynamic` または `guided` の場合のワークシェアリング・ループ構文の OpenMP* デフォルト・ループ・スケジュール修飾子を `nonmonotonic` に変更しました。`monotonic` 動作を想定しているユーザーコードは、この変更により正しく動作しなくなる可能性があります。以前のコードの動作を維持するには、`schedule` 節に `monotonic` スケジュール修飾子を追加してください。
- SYCL* 2020 の `barriers` は SYCL* 1.2.1 の `barriers` よりもパフォーマンスが低下します。この問題は現在調査中で、将来のリリースで修正される予定です。
- カーネル内から devicelib 関数を少なくとも 1 回呼び出し、2 ステップの Ahead-Of-Time(AOT)コンパイルを使用すると、デバイスのバイナリーイメージが破損する可能性があります。
- レベルゼロランタイムによるサポートの制限により、割り当て要求のアライメントは 64KB に制限されています。
- SYCL* 2020 の特殊化定数機能には、次の制限があります。
  - JIT ターゲットと AOT ターゲットの両方で特殊化定数を同時に使用するプログラムを作成すると、次のメッセージが出力され、例外がスローされる可能性があります。

    ```
    Native API failed. Native API returns: -49 (CL_INVALID_ARG_INDEX)
    -49 (CL_INVALID_ARG_INDEX).
    ```
  - 非 AOT シナリオでは(`-fsycl-targets` コマンドライン・オプションが渡されない場合、または `spir64` がターゲットの場合)、特殊化定数値のゼロへの設定は DPC++ ランタイムに

より無視されます。以下に、問題を再現するサンプルコードを示します。現在、この問題の回避方法はありません。

```
specialization_id<int> spec_id(42);
// ...
queue q;
q.submit(handler &cgh) {
  cgh.set_specialization_constant<spec_id>(0);
  // spec_id will still have value 42
  cgh.set_specialization_constant<spec_id>(41);
  // spec_id value will be changed to 41
  cgh.set_specialization_constant<spec_id>(0);
  // spec_id will still have value 41
}
```

- AOT モードで、パディングされたオブジェクトにデフォルト値を設定すると、ほかのデフォルト値が正しくアライメントされず、特殊化定数のデフォルト値が正しくなくなる可能性があります。次に例を示します。

```
struct PaddedStruct {
  uint32_t a;
  char b;

  constexpr PaddedStruct() : a(0), b('a') {}
  constexpr PaddedStruct(uint32_t _a, char _b) : a(_a), b(_b) {}
};

constexpr specialization_id<PaddedStruct>
padded_struct_spec_id{20, 'c'};
constexpr specialization_id<bool> bool_spec_id{true};
```

`PaddedStruct` のサイズは 8 バイトで、3 バイトがパディングのため、`bool_spec_id` 識別される特殊化定数のデフォルト値が `true` にならない可能性があります。この問題の既知の回避方法は、クラスまたは構造体に `__attribute__((packed))` を追加して、パディングされたオブジェクトからパディングを削除することです。つまり、`PaddedStruct` を次のようにします。

```
struct __attribute__((packed)) PaddedStruct {
  uint32_t a;
  char b;

  constexpr PaddedStruct() : a(0), b('a') {}
  constexpr PaddedStruct(uint32_t _a, char _b) : a(_a), b(_b) {}
};
```

- Windows* で最新の Microsoft* Visual Studio* リリースを使用している場合、`-Qlong-double` コンパイラー・オプションの使用には制限があります。詳細は、こちら(英語)を参照してください。
- アプリケーション・コードで `sinpif` 関数や `cospif` 関数を使用していないにも関わらず、コンパイラーの最適化フェーズが原因で、「`Compilation from IR - skipping loading of FCL error: undefined reference to `sinpif'` (IR からのコンパイル - FCL のロードスキップ エラー: `sinpif' の未定義参照)」のようなエラーが発生することがあります。回避方法は、`-mllvm -enable-transform-sin-cos=0` コンパイラー・オプションを使用して、誤った最適化を無効にすることです。
- メンバー・テンプレートでの `#pragma omp declare simd` の使用は現在サポートされていません。使用すると、エラー「`error: function declaration is expected after 'declare simd' directive`(エラー: 関数宣言は 'declare simd' ディレクティブの後に行ってください)」が発生します。非テンプレート・メンバー関数とクラスのメンバーでないテンプレート関数は影響を受けません。

- C++17 を有効にして DPC++ のホスト・コンパイラーとして Microsoft* Visual Studio*を使用すると、エラー「`C:\Program Files (x86)\Intel\oneAPI\compiler\latest\windows\include\sycl\CL/sycl/ONEAPI/accessor_property_list.hpp(199): error C2686: cannot overload static and non-static member functions with the same parameter types`(... エラー C2686: 同じパラメーター型の静的および非静的メンバー関数をオーバーロードすることはできません)」が発生します。この問題の回避方法は、こちらの記事(英語)を参照してください。
- デバイスとホスト間の共有割り当ての暗黙的な移行に対する USM のサポートは、現在、ホストからのアクセスを識別するアクセス違反メカニズム(SIGSEV など)を使用してソフトウェアで実装されています。アプリケーションが同様のアクセス違反メカニズムに依存している場合、または GPU ドライバーによりホストに移行する前にシステムコールを使用して共有メモリー割り当てにアクセスする場合、未定義の動作が発生する可能性があります。
- `icx` コンパイラーは、ターゲット・オフロード・コードを含むライブラリーで `-l` オプションを使用したライブラリー・アーカイブのリンクをサポートしていません。この問題の詳細および回避方法は、「既知の問題:スタティック・ライブラリーとターゲットオフロード」(英語)を参照してください。
- リンク時の最適化(LTO)を使用しようとすると、一部の Linux* OS でリンクエラーが発生することがあります。正常にリンクするには、「インテル® oneAPI DPC++/C++ コンパイラーの動作環境」(英語)にリストされている、OS で推奨するバージョンの `binutils` を使用してください。
- OpenCL* C 組込み関数と同じ名前およびシグネチャー(戻り値の型は関係なく、引数が完全に一致している)のユーザー定義関数を使用すると、未定義の動作を引き起こすことがあります。この問題の詳細は、「既知の問題:OpenCL* 組込み関数と同じシグネチャーのユーザー定義関数」(英語)を参照してください。
- ファイルスコープで発生する `#pragma float_control` は、クラス定義内の入れ子のステートメント・ブロックでは正しく動作しません。`#pragma clang fp` にも同じ問題が存在します。
- Windows* システムで Microsoft* Visual Studio* で FPGA エミュレーター・コードをデバッグすると、デバッガーはカーネルコードに設定されたブレークポイントで停止しません。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。
- 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することはできません。コンパイルに成功しても、コンパイルされたバイナリーが実行時に失敗する可能性があります。現在、この問題の回避方法はありません。
- 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。

- FPGA 最適化レポートで、ループビューアー(アルファ)は、現在 100 回以下の反復でのみループを処理できます。デザインのループが 100 回を超える場合、最適化レポートがハングします。この問題の既知の回避方法はありません。
- このリリースでは FPGA で `modulefiles-setup.sh` スクリプト(英語)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(英語)を使用してください。
- FPGA 最適化レポートは Windows* の Microsoft* Visual Studio* 内で正しく表示されません。レポートを表示するには、プロジェクト・ディレクトリーに生成された `report.html` ファイルを開いてください。
- 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 向けにコンパイルするとき、Windows* エミュレーター・フローで -c を使用してオブジェクト・ファイルを作成し、アーカイブファイルにリンクした後、そのアーカイブから実行ファイルを生成すると、実行ファイルがデバイスカーネルの呼び出しに失敗することがあります。この問題を回避するには、次に示すように、アーカイブステップに `-fsycl-device-code-split=none` オプションを追加します。

  ```
  # generate .obj files
  dpcpp /EHsc -fintelfpga -c host.cpp device.cpp device_adder.cpp -DFPGA_EMULATOR
  # generate host.a
  dpcpp -fintelfpga -fsycl-link=image -fsycl-device-code-split=none host.obj device.obj device_adder.obj
  # generate .exe
  dpcpp -fintelfpga host.a /link /wholearchive
  # emulator executable
  host.exe
  ```
- 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。
- Linux* システムで FPGA 向けにコンパイルするときに、多くの Linux* ディストリビューションに標準で含まれる `zlib` ライブラリーをコンパイラーが検出できない場合、「`Unable to open zlib library!`(`zlib` ライブラリーを開けません!)」エラーメッセージが表示されます。コンパイラーがこのライブラリーを検出できるようにするには、次の OS 固有のコマンドのいずれかを実行して、ライブラリーの開発バージョンをインストールします。
  - **Ubuntu* 18**:`sudo apt install zlib1g-dev`
  - **RHEL 7/CentOS* 7**:`sudo yum install zlib-devel`
- FPGA 最適化レポートを起動するときに、コンパイラーがソースファイルに含まれている特定のテキスト文字のレンダリングに失敗することがあります。レポートがクラッシュした場合、`reports/lib/` ディレクトリーの `report_data.js` ファイル内の `fileJSON` オブジェクトのコンテンツセクションにエスケープされたバックスラッシュ(\\)で終了している文字列リテラルが含まれているかどうかを

確認します。この問題を回避するには、`report_data.js` ファイルを変更して、エスケープされていない文字をエスケープするようにします。例えば、`"hello\\"` を `"hello\\\"` に変更します。

動作環境

- インテル® oneAPI DPC++/C++ コンパイラーの動作環境(英語)

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(英語)
- 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(英語)

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 2021.1 リリースノート(PDF)

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。

製品とパフォーマンス情報

[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。