インテル® oneAPI DPC++/C++ コンパイラー 2024.0 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。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。

バージョン: 2024.0.2
2023 年 12 月 21 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。

コンパイラー・リリース 2024.0.2

- 問題の修正
- 2024.0.1 の一部となる修正を含みます。

コンパイラー・リリース 2024.0.1

- SYCL* バインドレス・テクスチャーが修正され、Codeplay の NVIDIA* GPU プラグインを利用して NVIDIA のハードウェア上で正しく動作するようになりました。
- インテル® Core™ Ultra プロセッサーを搭載したデバイスをサポートするように OpenMP* ランタイムが更新されました。

oneAPI 2024.0、コンパイラー・リリース 2024.0

新機能と改善点

- 2024.0 でレベルゼロドライバーのインテル® データセンター GPU マックス・シリーズのデフォルトを「cards-as-devices」から「tiles-as-devices」に変更しました。
- このリリース以降、インテルのレベルゼロおよび OpenCL* GPU ドライバーは、インテル® データセンター GPU マックス・シリーズなどのマルチタイル・デバイスを異なる方法で公開します。この変更は、これらのデバイスの SYCL* および OpenMP* での公開方法にも影響します。この変更の前は、各カードはルートデバイスとして公開され、タイルはサブデバイスとして公開されていました。

現在、各タイルはデフォルトでルートデバイスとして公開されます。この変更は、ルートデバイスをサブデバイスに分割する方法にも影響します。古い動作は、ZE_FLAT_DEVICE_HIERARCHY (英語) 環境変数で有効にすることができます。

ルートデバイスの数とサブデバイスの利用が以前のリリースとは異なるため、ONEAPI_DEVICE_SELECTOR (英語) および ZE_AFFINITY_MASK (英語) 環境変数の変更が必要になることがあります。

- `-fsycl` オプションを指定してホストコンパイルを行う際にプリコンパイル済みヘッダー (PCH) をインクルードできるようになりました。
- AOT のシンボルをエクスポートできる `-ftarget-export-symbols` オプションをサポートしました。
- `reqd_sub_group_size` のサポートを `sycl::is_compatible` に追加し、`reqd-sub-group-size` カーネル機能に基づくデバイスコード分割を実装しました。
- spir64_gen および JIT モードで `-ftarget-compile-fast` をサポートしました。
- `-fsycl` オプションを指定してリンカーに C++ ライブラリーを追加する機能を実装しました。
- カーネル・ファンクターで複数の呼び出し演算子をサポートしました。
- デバイスのバックエンド・コンパイラーへのコンパイルオプションの継承をサポートしました。
- 新しい FPGA ループ属性 `enable_loop_pipelining` を追加しました。
- `sycl_ext_oneapi_annotated_arg` および `sycl_ext_oneapi_annotated_ptr` 試験的拡張機能を有効にしました。
- `sycl_ext_intel_queue_immediate_command_list` 拡張機能を実装しました。
- `sycl_ext_oneapi_copy_optimize` 試験的拡張機能を実装しました。
- 試験的 SYCL* Graph 拡張機能 `sycl_ext_oneapi_graph` の初期実装を追加しました。
- item/rangelike 型に `dimensions` メンバーを追加しました。
- OpenCL* バックエンドで `queue::priority_*` プロパティーをサポートしました。
- `SYCLNativeCPU` で設計された SYCL* Native CPU プラグインの初期バージョンを実装しました。
- imf デバイス・ライブラリーに `__imf_max/min/hadd/fast_*` 関数を追加しました。
- `sycl_ext_oneapi_device_architecture` 拡張機能の一部として、新しい `sycl::ext::oneapi::experimental::info::device::architecture` デバイス記述子と `device::ext_oneapi_architecture_is(ext::oneapi::experimental::architecture)` ホスト API を導入および実装しました (レベルゼロおよび OpenCL*)。
- `sycl_ext_intel_grf_size` の試験的実装を追加しました。
- 試験的 `sycl_ext_oneapi_device_global` 拡張機能をサポートしました。
- ESIMD のイメージアクセサーと連携するメディア API を有効にしました。これらのアクセサーはステートフル/ステートレス・モードに依存しません。
- `std::experimental::simd_mask` から `ESIMD::simd_mask` への暗黙的な変換ができるようになりました。
- `host_task` イメージアクセサーをサポートしました。
- ESIMD API のステートレス・モードのアクセサーで 64 ビット・オフセットをサポートしました。
- imf デバイス・ライブラリーに `__imf_llabs` を追加しました。
- `multi_ptr<T>` から `multi_ptr<const T>` への明示的な変換を追加しました。
- グループ・アルゴリズムでスカラー論理演算子をサポートしました。

- ESIMD エミュレーターでの 64 ビット・アトミックをサポートするデバイスクエリーを実装しました。
- `lsc_atomic_update` および `lsc_slm_atomic_update` ESIMD API で 16 ビット・データをサポートしました。
- `sycl_ext_oneapi_root_group` の初期実装を追加しました。いくつかの制限があります。`max_num_work_group_sync` クエリーは常に 1 を返します。ルートグループ内のすべてのワークアイテムは現在同じワークグループ内にあるため、実装されたバリアにはワークグループのスコープが含まれます。
- SYCL* Matrix 拡張機能の統合インターフェイスを使用した tf32 型をサポートしました。
- `sycl_ext_intel_dataflow_pipes` 拡張機能で説明されているホストパイプを実装しました。
- 拡張仕様 `sycl_ext_oneapi_bfloat16_math_functions` を更新し、bfloat16 向けの多くの数学組込み関数をサポートしました。
- 修飾された async_work_group_copy オーバーロードを追加しました。
- 統合ランタイムプラグインの初期実装を追加しました。SYCL_PREFER_UR を使用してプラグインを経由します。
- `atomic_update` ESIMD API でアクセサーをサポートしました。
- GDB による参照オブジェクトのプリティープリントを有効にしました。
- GDB で SLM から GPU 上の `local_accessors` の読み取りをサポートする Xmethod を追加しました。
- `invoke_simd()` API 経由でローカルアクセサーを ESIMD カーネルに渡せるようにして、ESIMD カーネルでのアクセサーへの get_pointer() と operator[] の使用法を有効にしました。accessor と local_accessor のサポートは、SYCL* と比較してまだ制限されています。
- 2 次元ブロックロード/ストア ESIMD API にアンパディングを実装しました。
- SYCL* 2020 イメージクラスを追加しました。
- SYCL* 2020 イメージアクセサーのインターフェイスを追加しました。ホストアクセサーのインターフェイスのみ実際に機能します。
- SYCL* 2020 イメージおよび関連するアクセサークラスの XPTI 通知を追加しました。
- `sycls verbose` でデバイスアスペクトを出力するようにしました。
- `tangle_group` および `opportunistic_group` 引数でグループ・アルゴリズムを使用できるようにしました。
- `info::device::backend_version` クエリーを実装しました。
- アルゴリズムで `fixed_size_group` をサポートしました。
- imf libdevice に単純な abs(int) を追加しました。
- アルゴリズムで ballot_group をサポートしました。
- レベルゼロ・バックエンドのイメージの相互運用性のサポートを実装しました。
- 数学組込み関数で `marray` をサポートしました。
- ESIMD カーネルでのインライン・アセンブリーのサポートを有効にしました。
- 標準または即時コマンドリストを選択できるように、相互運用性キュー・インターフェイスを強化しました。
- `atomic_update()` ESIMD API で double 型を有効にしました。
- ESIMD で `addc` および `subb` 操作をサポートしました。

- ゼロサイズの 3D アクセサーが使用できるようになりました。
- `fsycllinkhugedevicecode` は非推奨になりました。新しいオプション `flinkhugedevicecode` を使用してください。新しいオプションは機能的に同じで、`fopenmptargets` とともに使用できます。
- `device_global` を使用したときに生成されるバイナリーのサイズを最適化しました。
- SYCL* モードで PCH がトリガーされたときにエラーを出力するようにしました。
- FPGA アーカイブデバイスと AOCO のアンバンドルを改善しました。
- imf `abs` をディープラーニング向けの別のデバイス・ライブラリーに移動しました。
- DPC++ のバグレポート URL を修正しました。
- bfloat16 関連のデバイス・ライブラリーを使用する場合のみライブラリーとリンクするようにしました。
- `fsyclesimdforcestatelessmem` をホストコードのコンパイルに適切に渡すようにしました。
- ADLS と RPLS デバイス・アーキテクチャーを結合するようにしました。
- SYCL* 1.2.1 でのコード分割を回避するため、multi_ptr のデフォルトがレガシーになるように変更しました。
- 更新された `sycl_ext_oneapi_local_memory` に従って `group_local_memory` 配列で集約の初期化を使用するようにしました。
- ESIMD エミュレーターは非推奨になりました。
- `ext::oneapi::sub_group` は非推奨になりました。
- `ext_intel_free_memory` アスペクトに関連するエラーメッセージを改善しました。
- レベルゼロ・バックエンドの領域コピー後の不要なバリアを削除しました。
- `get_info<device::free_memory>` をアスペクトをチェックするように修正しました。
- 補助バッファーの解放の回避策を削除しました。
- コンテキスト内の複数のデバイスからの読み取り専用バッファーアクセスの最適化を有効にしました。
- 古い特殊化定数の拡張機能と実装を削除しました。
- 特定のターゲットが定義されているかどうかを `fsycltargets` で確認して結果を変更できるように `is_compatible` を改善しました。
- ホスト `sycl::remquo` の商の近似を改善しました。
- ゼロサイズのバッファーで `accessor` を構築できるようになりました。
- インオーダー・キュー待機時のイベントをクリーンアップすることにより、レベルゼロ・バックエンドのリソースリサイクルを改善しました。
- エラーレポートを改善するため、`code_location` パラメーターを残りの `sycl::queue` メソッドに追加しました。
- グラフをバイパスする parallel_for の `xpti::node_create` シグナルの出力を有効にしました。
- ホスト `sycl::cospi` の精度を向上しました。
- 古い memset OpenCL* API `clEnqueueMemsetINTEL` の使用を `clEnqueueMemFillINTEL` に置き換えました。
- 有益な XPTI メモリートレースを可能にするため、XPTI メモリー割り当てイベントメタデータへのメモリーポインターを追加しました。
- レベルゼロ・バックエンドのコンテキスト内のキュー向けに即時コマンドリストのリサイクルを実装しました。

- レベルゼロ・バックエンドのインオーダー・キュー向けに ext_oneapi_submit_barrier() を最適化しました。
- SYCL_PI_LEVEL_ZERO_USM_RESIDENT のデフォルトをデバイス割り当てのみを強制するように変更しました。
- すべてのデバイスで aspect::image の false をレポートするようにしました。
- `sycl_ext_oneapi_device_architecture` に従って「if_architecture_is」からラムダ・パラメーターを削除しました。
- `reqd_work_group_size` がデバイスでサポートされていない場合のエラーレポートを改善しました。
- `block_2d` API の静的制限チェックを調整しました。
- SYCL* 2020 に従ってローカルアクセサーの誤使用を禁止しました。
- SPIR-V* ジョイント行列ロード/ストア組込み関数にアドレス空間情報を渡す機能を実装しました。
- レベルゼロ・バックエンドの即時コマンドリストをデフォルトで有効にしました。
- キャッシュされたカーネルの追加中に削除されたカーネル引数の余分なマップ・ルックアップを削除することにより、SYCL* RT のパフォーマンスが向上しました。
- SYCL_PI_LEVEL_ZERO_USM_RESIDENT のデフォルトを 2 に変更しました。
- レベルゼロ・バックエンドの即時コマンドリストのクリーンアップのオーバーヘッドを削減するヒューリスティックを追加しました。
- `cluster_group` の名前を `fixed_size_group` に変更しました。
- InvokeSIMD の無効な統一引数のエラーを追加しました。
- piGetDeviceAndHostTimer はオーバーヘッドが大きいため、`steady_clock::now()` を使用してホスト時間を取得することによりキュー・プロファイルのオーバーヘッドを削減しました。
- レベルゼロランタイムがクリーンアップ時にすでにアンロードされている場合の適切な処理を実装しました。
- InvokeSIMD のエラーメッセージを改善しました。
- 特殊化定数が存在しない場合のために `native_specialization_constant()` を更新しました。
- レベルゼロ・バックエンドのコマンドリストの再利用を最適化しました。
- `std::max` および `std::min` と一致するように `sycl::maximum` および `sycl::minimum` の動作を変更しました。
- 大部分の SYCL* デバイスヘッダーの `sycl::runtime_error` を `sycl::exception` に変更しました。`sycl::runtime_error` は SYCL* 2020 で非推奨になりました。
- USM での無効なデバイスメモリーへのアクセスに関する診断を改善するため、`sycl-trace` に USM 呼び出しパラメーター検証レイヤーを追加しました。
- `sycllls --verbose` でサポートされている SG サイズを出力するようにしました。
- 次の FPGA 可変精度データ型変換をサポートしました。
    - `ac_int` から `ap_float`
    - `ap_float` から `ac_int`
    - `ac_fixed` から `ap_float`
    - `ap_float` から `ac_fixed`
- マルチアーキテクチャー・バイナリーから FPGA ハードウェア構成ファイル (`.aocx`) を抽出するコマンドを追加しました。

- `-Xsfp-relaxed` FPGA オプションを削除しました。このオプションを今後サポートする予定はありません。`-fp-model=precise` を使用している場合でもアキュムレーターで推論したい場合は、累積を利用するようにコードを書き直してください。
- `protocol_name::avalon_mm` を使用した FPGA ホストパイプのシミュレーションの制限を削除しました。
- `-Xsoptimize=throughput` オプションを `-Xsoptimize=throughput-area-balanced` オプションに置き換えました。
- `annotated_arg` クラスを追加しました。FPGA カーネルへのメモリーマップ・ホスト・インターフェイスを作成するには、このクラスを使用してください。
- `[[intel::max_global_work_dim(0)]]` カーネル属性は非推奨になりました。コンパイラーはこの属性をシングルタスク・カーネルに自動的に追加するため、この属性を明示的に追加する必要はなくなりました。
- FPGA カーネル向けの `register_map_interface` マクロは非推奨になりました。メモリーマップ・インターフェイスがデフォルトになったため、明示的な宣言は不要になりました。
- `streaming_interface` マクロを置換するため `streaming_interface` FPGA カーネル・プロパティーを追加しました。このマクロは非推奨になりました。
- `streaming_pipelined_interface` マクロを置換するため `pipelined` FPGA カーネル・プロパティーを追加しました。このマクロは非推奨になりました。

問題の修正

- `uses_aspects` が関数定義のみではなく関数宣言にも適用されるように修正しました。
- テンプレート・コンテキストの `ivdep` 属性を修正しました。
- Windows* デバイスのオプション制限を修正しました。
- ESIMD のサポートされていない属性の処理を修正しました。
- デバイスオプションの最適化オプションの処理を修正しました。
- SYCL_EXTERNAL を使用したオプションのカーネル機能が適切に動作するようにしました。
- -O0 オプションを指定してコンパイルした場合の、ESIMD カーネル内の slm_init() の「SLM init call is supported only in kernels (SLM init call はカーネル内でのみサポートしています)」メッセージを修正しました。
- `Base64::decode` のメモリーリークを修正しました。
- `-fsycl` を指定してライブラリーを使用したときにクラッシュする問題を修正しました。
- Windows* でリダクションを使用したときの特定のビルドでの問題を修正するために `/MDd` の事前定義を追加しました。
- カーネル名が「`final`」キーワードで定義された `class` の場合に発生するコンパイル問題を修正しました。
- ESIMD カーネルで slm_init() を使用すると JIT コンパイルが失敗する VCSLMSize 属性の設定の問題を修正しました。
- hostdep リンクの未定義シンボルのオプションが gold リンカーで動作するように修正しました。
- `DIFile` のディレクトリー・フィールドを修正しました。

- `ext::oneapi::level_zero::ownership::keep` が `make_kernel_bundle` に渡されているにもかかわらず、`zeModuleDestroy` が呼び出されて二重解放が発生するレベルゼロ・バックエンドの問題を修正しました。
- `multi_ptr` の仕様の不一致に対処しました。
- 排他的スキャンで `short` または `char` を使用するときの例外的なケースを修正しました。
- ほかのアクセサーで使用される構築手法に準拠するため、`local_accessor` の従来の `multi_ptr` 構築を更新しました。
- SYCL* 2020 に準拠するため、`get_pointer`、`noexcept` を追加しました。
- `sycl::reqd_work_group_size` が無効な場合に正しい `sycl::errc` を返すようにしました。
- `sycl::remquo` の切り捨てエラーを修正しました。
- `mem_channel` プロパティーを取得しようとしたとき、または `has_property()` を使用してチェックしようとしたときのリンクエラーを修正しました。
- online_compiler の ocloc ツールの動的ロードを修正しました。
- Arc のグローバル・メモリー・レポートを修正しました。
- SYCL* 2020 に従って byvalue セマンティクスに沿うように `sycl::sub_group` を修正しました。
- OpenCL* バックエンドの `event.get_info<sycl::info::event::command_execution_status>()` で返される無効な値を修正しました。
- ESIMD グローバルの代入演算子を修正しました。
- プログラムのビルドに失敗したときにビルドログが破壊されないようにレベルゼロ・バックエンドを修正しました。
- 検証レイヤーが `ZE_DEBUG=6` でレベルゼロに対して有効になっているときに表示されるエラーを修正しました。
- グループ・アルゴリズムで明示的な型を含む関数オブジェクトを使用できるようになりました。
- カーネル ID のベクトルが空の場合の `sycl::is_compatible()` を修正しました。
- 拡張アドレス空間の multi_ptr ctor を修正しました。
- `fill` と `copy` の trivially_copyable を device_copyable に修正しました。
- リダクション parallel_for の範囲推論を修正しました。
- SYCL* 2020 に準拠するため、multi_ptr 推論ガイドを調整しました。
- ESIMD バックエンドにスカラーを渡すときのアクセサーを使用したギャザー/スキャッターを修正しました。
- ローカルおよびホストアクセサーでの暗黙的な変換中にデータが失われる問題を修正しました。
- 不要なカーネル保持に伴うメモリーリークを修正しました。
- 読み取り専用アクセサーに含まれるポインター型を const に修正しました。
- サブグループのないデバイスの `max_sub_group_size` クエリーを修正しました。
- ホスト側の `abs_diff` の未定義の動作を削除しました。
- vec::as<vec<bool, N>>() を修正しました。
- ZE_DEBUG を使用したときのセグメンテーション・フォルトおよびデバイス選択フェーズで例外がスローされる問題を修正しました。
- 既存の変換との競合により暗黙的な変換が曖昧になる不要な `multi_ptr` 変換を削除しました。

- アクセサーの暗黙的な変換でデータが失われる問題を修正しました。
- OpenCL* バックエンドの一部のデバイス情報クエリーで発生していたスタック破壊を修正しました。
- DPCPP ツールチェーンのインストールに伴うレベルゼロローダーとヘッダーの出力を停止しました。
- レベルゼロ・バックエンドのインテル® データセンター GPU マックス・シリーズのデバイス ID チェックを修正しました。
- `bit_cast` の曖昧さを修正しました。
- `fastmath` モードでの `sycl::stream` の nan/inf 処理を修正しました。
- グループ・アルゴリズム `reduce`、`exclusive_scan`、`inclusive_scan` で異なる型を使用できるようになりました。
- 2D ブロック・ステートレス・ロード/ストア API の `config_2d_mem_access` クラスの get メソッドの問題を修正しました。
- `fastmath` モードでの `sycl::fabs` 組込み関数のコンパイルエラーを修正しました。
- プレースホルダー・アクセサーで host_task 推論タグを使用できるようになりました。
- SYCL* 2020 に準拠するため、ゼロ次元アクセサーのイテレーター操作を追加しました。
- __imf_vavgs の丸め問題を修正しました。
- ゼロ次元アクセサーの `fill` 操作の問題を修正しました。
- `ftargetcompilefast` 経由で ocloc に渡される引数を修正しました。
- SYCL* 2020 に準拠するため、ローカルアクセサーとホストアクセサーの暗黙的な変換を実装しました。
- SYCL* 2020 に準拠するため、空のアクセサーが require() を呼び出したときに例外をスローするようにしました。
- `sycl::vec<bool, N>` の `operator~` を修正しました。
- `long long` と `half` のベクトルのサブグループのシャッフルを修正しました。
- OpenCL* 拡張 fptr キャッシュの静的制限順序の問題を修正しました。
- レベルゼロ・バックエンドのアクティブバリアのイベントのリークを修正しました。
- `aspect::ext_oneapi_srgb` の適切なクエリーを実装しました。
- SYCL* 2020 に準拠するため、スカラーからの `vec` 割り当てと `vec` モジュロのオーバーロードを追加しました。
- mem_channel バッファー・プロパティーの処理を修正しました。
- local_accessor<const T> の operator& と operator[] を修正しました。
- `select` 組込み関数の 3 番目の引数の型を修正しました。
- `native_specialization_constant()` API の実装を修正しました。
- レベルゼロドライバーが使用できないケースをサポートするため、レベルゼロ固有のコレクターを sycltrace ツールの要求によりロードされるダイナミック・ライブラリーに移動しました。
- 不足していた marray リレーショナル関数 `any`、`all`、`bitselect` を追加しました。スカラー `select` リレーショナル関数を修正しました。スカラー `abs` 整数関数を SYCL* 2020 に準拠するように変更しました。`multi_ptr` 引数を受け取る数学関数を修正しました。
- SYCL* 2020 に準拠するため、swizzle vec に operator[] と element_type を追加しました。
- `atomic_memory_order_acq_rel` のバッファー範囲を修正しました。
- 無効な `global_work_size` クエリーをスローするようにしました。

- イベントが完了するまで、実行中ではなく送信済みとしてイベントをレポートするように、レベルゼロ・バックエンドを修正しました。
- raw 送信 ESIMD API で `sycl::half` などの非標準の型を使用できるようになりました。
- SYCL* 2020 に準拠するため、無効な `info::kernel::num_args query` クエリーをスローするようにしました。
- `vec` 型をサポートするように group_broadcast を更新しました。
- 暗黙的な atomic64 要件を回避するようにリダクションを修正しました。
- コマンド送信プロファイル情報に関する最近の変更後、OpenCL* バージョン 2.1 未満でキュー・プロファイルがサポートされなくなったため、acc デバイスの部分的なプロファイル回避策を追加しました。
- プログラムのビルドに失敗したときにビルドログを保持するように、レベルゼロ・バックエンドのプログラム ビルド API を修正しました。
- SYCL* 2020 に準拠するため、local_accessor および host_accessor に不足していた std::hash 特殊化を実装しました。
- global_work_size カーネルクエリーを修正しました。
- `sycl::vec<bool, N>` コンストラクターの不正な動作を修正しました。
- `host_accessor` と `stream` の weak_object を修正しました。
- `sycl::vec<bool, N>` を使用した一部の操作での不正な動作を修正しました。
- specialization_id の統合フッターを修正しました。
- `bfloat16` コンストラクターをカーネルで使用したときに発生していたコンパイルの中断を修正しました。
- サブグループをサポートしていない OpenCL* バックエンドで実行しているときにサブグループ情報クエリーでクラッシュする問題を修正しました。
- mode_target タグを使用した target::host_tas 特殊アクセサー・コンストラクターに不足していたサポートを追加しました。
- 書かれていないレデューサーによる単位元のないリダクションを修正しました。
- xpti トレースにおけるダングリング・ポインターの問題を修正しました。
- memcpy2d デバイスホスト・フォールバックの PI イベントリークを修正しました。
- `device` オブジェクトの `weak_object` と `owner_less` を修正しました。
- SYCL* 2020 に準拠するため、vec::byte_size に `noexcept` 指定子を追加しました。
- ベクトル printf 指定子の未定義の動作を修正しました。
- パターンがバックエンドでネイティブにサポートされていない場合でも機能するように handler::fill を修正しました。
- プレースホルダー・アクセサーがコマンドに渡されたときに例外をスローするメカニズムを修正しました。
- 空のゼロ次元アクセサーのアクセス範囲を修正しました。
- `sycl::buffer` がホストデータとして `const T*` で構築されている場合の不正なライトバックを修正しました。
- SYCL* 2020 に準拠するため、空のアクセサーのデフォルト・コンストラクターがプレースホルダーを作成しないように修正しました。

- 非推奨の `piclCreateProgramWithSource` を削除しました。
- 非推奨のバリア API を削除しました。
- 非推奨の `interop_task` を削除しました。
- 非推奨の `sycl::group_local_memory` を削除しました。
- 非推奨の `sycl::detail::bitcast` を削除しました。
- 非推奨の `piEnqueueNativeKernel` を削除しました。
- 非推奨のバックエンド enum 値 (level_zero、cuda、esimd_cpu、hip) を削除しました。
- 補助バッファーの解放の回避策を削除しました。
- 古い特殊化定数の拡張機能と実装を削除しました。
- sycl 名前空間から非標準の RT 名前空間を削除しました。
- 非推奨の ESIMD API を削除しました。
- API から DISABLE_SYCL_INSTRUMENTATION_METADATA マクロおよび _CODELOC* マクロの使用法を削除しました。
- sycl 名前空間から非標準の RT 名前空間を削除しました。
- getOSModuleHandle の使用法を削除しました。
- 非推奨の sycldevice トリプルのサポートを終了しました。
- `sycl_ext_oneapi_extended_atomics` および `sycl_ext_oneapi_group_algorithms` 拡張機能のサポートを終了しました。
- make_queue および get_native の不要な下位互換性を削除しました。
- 「sycldevice」環境コンポーネントを使用してトリプル向けに生成されたバイナリーのサポートを終了しました。
- `sycl_ext_oneapi_device_architecture` に従って「if_architecture_is」からラムダ・パラメーターを削除しました。
- SYCL* 2020 に準拠するように `sycl::exception` を更新しました。
- 非推奨の sycl::runtime_error を SYCL* 2020 準拠の `sycl::exception` に置き換えました。
- win_proxy_loader の名前を `pi_win_proxy_loader` に変更しました。
- SYCL* 1.2.1 と SYCL* 2020 の間に変更された SYCL* リレーショナル組込み関数の戻り型の変更を、SYCL2020_CONFORMANT_APIS のガードから外してプロモートしました。
- 仕様に従って target::device 特殊アクセサーには `T*` を返すように `get_pointer` を修正しました。
- SYCL* 2020 に従って `max_work_item_sizes` の戻り値の型を id から `range` に修正しました。
- 非推奨の試験的な `set_kernel_properties` API と `use_double_grf/use_large_grf` プロパティーを削除しました。`sycl_ext_intel_grf_size` 拡張機能で提供される新しい API を使用してください。
- 戻り値 `struct` の FPGA タスクシーケンス関数のサポートを妨げていた問題を修正しました。
- `[[intel::max_reinvocation_delay]]` FPGA ループ属性を使用して高最適化ループをコンパイルしたときに、ループ伝播メモリー依存により、ループの II クリティカル・パス上にローカルメモリー LSU があるループでアサートメッセージが出力されていた FPGA 問題を修正しました。
- FPGA RTL ライブラリーが原因で、コンパイルプロセスの後半で SYCL* HLS IP コアのインテル® Quartus® Prime 開発ソフトウェアのコンパイルが失敗していた問題を修正しました。

- SYCL* HLS フローの一部として生成される `hw.tcl` ファイルがシグナルを正しくマップしない FPGA 問題を修正しました。
- カーネル関数内のラムダ内で使用するカーネル引数で FPGA `mmhost` マクロを使用したときに発生していたエラーを修正しました。このエラーの修正に加えて、`mmhost` マクロは非推奨になりました。
- `modulefiles-setup.sh` スクリプトが FPGA でサポートされていなかった FPGA 問題を修正しました。

既知の問題と制限事項

- 現在、SYCL* にはバインドレス・テクスチャーに関する既知の問題があります。チームはこの問題の解決に取り組んでおり、問題に対処するパッチリリース (2024.0.1) をポストする予定です。
- インテル® Arc™ グラフィックスのグラフノード内のコピー操作は、アクセサーにも影響する同期問題により失敗することがあります。
- 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。
- 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。
- 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。
- ESIMD の accessor と local_accessor のサポートは、SYCL* と比較してまだ制限されています。
- `sycl_ext_oneapi_root_group` 実装には次の制限があります。max_num_work_group_sync クエリーは常に 1 を返します。ルートグループ内のすべてのワークアイテムは現在同じワークグループ内にあるため、実装されたバリアにはワークグループのスコープが含まれます。
- デバイスの不要な処理を行うと、オーバーヘッドが発生し、バッファーとの意図しない処理が行われることがあります。SYCL* Graph は、特定のユーザー定義デバイスのみファイナライズするのではなく、コンテキストに関連付けられたデバイスごとにファイナライズします。
- 「インテル® データセンター GPU マックス・シリーズ以前の」GPU で 4GB を超える割り当てを使用するアプリケーションを実行する場合は、インテル® グラフィックス・コンパイラー (IGC) が正しいコンパイル単位を作成できるように、`SYCL_PROGRAM_COMPILE_OPTIONS=-ze-intel-greater-than-4GB-buffer-required` 環境変数を設定する必要があります。

この環境変数を使用しないで 4GB を超える割り当てを使用するアプリケーションをコンパイルすると、予期しない動作が発生することがあります。「インテル® データセンター GPU マックス・シリーズ以前の」GPU で 4GB を超える割り当てを使用すると、グラフィックス・コンパイラーにより追加された特定のポインター演算の最適化が無効になるため、アプリケーションのパフォーマンスが異なることがあります。

- `-g` オプションを指定して SYCL* プログラムをビルドし、CPU デバイス上の gcc バージョン 13.1.0 以降で実行すると、シャットダウン段階でプログラムがクラッシュすることがあります。この問題はインテルの製品の欠陥ではありませんが、使用している gcc のバージョンが 13.1.0 以降の場合、CPU デバイスで `-g` オプションを指定してビルドした SYCL* プログラムを実行しないでください。
- 2024.0 リリースでは、アドレス、リーク、スレッド・サニタイザー (-fsanitize コンパイラー・オプションを指定) を微調整し、DPC++ ランタイムが依存するさまざまなランタイム・ライブラリーで正当であるかどうかではなく、ユーザーのプログラムで問題を検出することに注目しました。将来のリリースでこれらの問題を修正または抑制する作業に取り組んでいる間、スレッド・サニタイザーで、SYCL* ヘッダーファイルの問題が検出される場合があることに注意してください。一方、メモリー・サニタイザーは、ユーザーコードに達する前にランタイム・ライブラリーの問題をレポートします。これらのサニタイザーは、DPC++ プログラム内のホスト側のコードのみサニタイズすることに注意してください。カーネルはサニタイズしません。
- DPC++ 2024.0 コンパイラーは SYCL* 2020 仕様の変更に準拠するように更新されましたが、この更新により、DPC++ 2023.0 では正常に動作していた一部の SYCL* プログラムが動作しなくなることがあります。特に、`sycl::minimum` または `sycl::maximum` を使用するプログラムが影響を受けます。これは、以前は `operator>` を使用して `sycl::maximum` を定義していましたが、現在は `std::max` と一致するように `operator<` を使用して定義しているためです。
- `-O0` を指定してコンパイルするときに `sycl::image` L0 相互運用を使用すると、問題が発生することがあります。この問題を回避するには、異なる最適化レベルを使用してください。
- CMake* 3.27 以降、インテル® oneAPI DPC++/C++ コンパイラーとインテル® Fortran コンパイラーのサポートが更新され、リンカーの代わりにコンパイラー・ドライバーをリンクに使用するようになりました。この変更により、SYCL* アプリケーションとライブラリーの作成のユースケースが有効になり、Windows* 上でプロシージャー間の最適化 (IPO) が有効になります。CMake* 3.25 以降では、CMake* がリンカーフラグを追跡する方法の制限により、CMake* プロジェクトに LLVM ベースのコンパイラー (icx または ifx) とクラシックバージョンのコンパイラー (icc または ifort) の C/C++ コードと Fortran コードが混在していると、無効なリンクフラグが生成されます。例えば、CMake* は、C/C++ コンパイラーとして「icx」を使用し、Fortran コンパイラーとして「ifort」を使用したプロジェクトをビルドできません。この問題を回避するには、混在言語アプリケーションを作成するときに LLVM ベースのコンパイラーのみを使用します。
- 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。
- 1 つ以上の FPGA デバイスでカーネルを実行する DPC++ プログラムはマルチスレッド実行をサポートしていません。
- FPGA エミュレーション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。
- 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:
'…\oneAPI\compiler\latest\windows\bin\dpcpp.EXE' : 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。また、この関数の使用は診断されません。コンパイラーは、`memory_scope::system` 条件を FPGA でサポートされる最も広いスコープである `memory_scope::device` 条件として扱います。
- FPGA 向けにコンパイルするときに、コンパイラーは Windows* と Linux* で異なる方法で構造体をパックします。この違いにより、メモリーアクセスが適切に調整されていないメンバーを含む構造体が生成されます。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];
```

- FPGA 最適化レポートは、インテル® Quartus® Prime 開発ソフトウェア・プロ・エディションのバージョン 23.1 以降の Quartus コンパイルから不正な領域使用データをレポートします。現在、この問題の既知の回避方法はありません。
- Windows* で、スタンドアロンのインテル® oneAPI FPGA レポート・ツール・アプリケーションがマップされたネットワーク・ドライブ上で実行できず、コンソールに「`GPU process launch failed (GPU プロセスの起動に失敗)`」エラーメッセージが表示されることがあります。この問題を回避するには、インテル® oneAPI FPGA レポートツールをマップされたネットワーク・ドライブからローカル・コンピューターにコピーし、ローカルで実行します。
- Jupyter* Notebook 内の HTML ファイルに関連する既知の問題により、Jupyter* Notebook 内では FPGA 最適化レポートを起動できません。この問題を回避するには、インテル® oneAPI FPGA レポートツールを使用するか、FPGA 最適化レポート・ディレクトリーをローカル・ファイル・システムにコピーし、サポートされているブラウザーを使用して起動します。
- `-g0` オプションを使用した場合など、特定の状況下で、FPGA レポート・サマリー・ページにコンパイルに使用した最適化フラグのリストが正しく表示されないことがあります。この問題を回避するには、コンパイルコマンドで `-g0` オプションを使用しないようにします。また、`-ghdl` オプションを使用する場合は、コマンドの最後の引数であることを確認してください。
- FPGA 最適化レポートを生成するときに、容量 0 のパイプを含むデザインでコンパイラーがクラッシュすることがあります。デザインの (すべてのパイプではなく) 一部のパイプのみ容量 0 の場合、容量 0 でないパイプがエリアレポートに表示されます。コンパイラーのクラッシュを回避するには、容量 0 のパイプの 1 つに容量 (1 など) を割り当てます。

- FPGA 向けにコンパイルするときに、数字のみまたは数字で始まる出力ターゲット名を指定すると、次の例で示すように、コンパイラーでエラーが発生し、エラーメッセージが表示されます。

  ```
  icpx -fsycl -fintelfpga -Xssimulation basic.cpp -o 2
  aoc: Compiling for Simulator. (aoc: シミュレーター向けにコンパイルしています)
  Error: Simulation system generation FAILED. (エラー: シミュレーション・システムの生成に失敗しました)
  Refer to 2.prj/2.log for details. (詳細は、2.prj/2.log を参照してください)

  llvm-foreach:
  cpx: error: fpga compiler command failed with exit code 1 (use -v to see invocation) (cpx: 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))
  ```

  エラーメッセージは代表的なものではありません。この問題を回避するには、数字のみまたは数字で始まる出力ターゲット名を指定しないようにします。
- FPGA 向けにコンパイルするときに、コンパイラーは `sycl::property::buffer::mem_channel` バッファー・プロパティーを無視します。プロパティーを指定するかどうかに関係なく、すべてのバッファー割り当ては最初のメモリーチャネルに割り当てられます。現在、この問題の既知の回避方法はありません。
- FPGA シミュレーション・フローでパイプライン化カーネルを実行するとき、シミュレーション・ランタイムが、生成された RTL で達成可能な最小のカーネル開始間隔を達成するために十分な速さでカーネル呼び出しを起動できないことがあります。現在、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、ライブラリー・アーカイブ・ファイルに RTL ソース・オブジェクトが含まれている場合、コンパイラーは非 RTL ソース・ライブラリー関数を無視して、次のエラーメッセージを出力することがあります。

  ```
  Compiler Error: undefined reference to <non-RTL source library function> (コンパイラー・エラー: <非 RTL ソース・ライブラリー関数> の未定義の参照)
  ```

  この問題を回避するには、RTL ソース・ライブラリー・オブジェクトと非 RTL ソース・ライブラリー・オブジェクトを同じアーカイブファイルに配置しないようにします。
- コンパイラーは、struct データ型の FPGA LSU コントロールを使用して特定の LSU スタイルを要求する場合、指定された LSU スタイルに制約されません。代わりに、アクセスパターンに最適な LSU スタイルが選択されます。この問題を回避するには、struct データ型の LSU コントロールを使用することを避け、代わりに単純なデータ型を使用します。
- FPGA SYCL* HLS 暗号化フローは Windows* システムでは完全にはサポートされていません。
- FPGA 向けにコンパイルするときに、`board_spec.xml` ファイル内のグローバルメモリーに名前フィールドがない場合、コンパイラーがクラッシュすることがあります。`board_spec.xml` ファイル内のすべてのグローバルメモリーに名前フィールドがあることを確認してください。例えば、`<global_mem name="DDR" ... >`
- FPGA 向けにコンパイルし、(`-fsycl-link=image` オプションを指定して生成した) デバイスコードを含む複数のファット・スタティック・ライブラリーをリンクすると、最初のライブラリーのデバイスコードのみファット実行ファイルに含まれ、次のエラーメッセージが出力されます。

  ```
  > what(): native api failed. native api returns: -46 (pi_error_invalid_kernel_name) (what(): ネイティブ API に失敗。ネイティブ API の戻り値: -46 (pi_error_invalid_kernel_name))
  > terminate called after throwing an instance of 'sycl::_v1::exception' ('sycl::_v1::exception' のインスタンスをスローした後の呼び出しで終了)
  ```

この問題を回避するには、ホストコードを静的にリンクする代わりに動的にリンクします。
コンパイルコマンドの例:

```
icpx -fsycl main.cpp -c -o main.o
icpx -fsycl -fintelfpga -fpic -shared add_kernel.cpp -o
libadd_kernel.so
icpx -fsycl -fintelfpga -fpic -shared sub_kernel.cpp -o
libsub_kernel.so
icpx -fsycl -fintelfpga main.o -L. -ladd_kernel -lsub_kernel -o
hot_swapper
LD_LIBRARY_PATH=$LD_LIBRARY_PATH:. ./hot_swapper
```

- サイズが 1024 ビットを超える device_global メモリーがデザインに含まれていて、カーネルで初期化していない場合、シミュレーター向けにコンパイルするときに不正な動作が発生することがあります。メモリーサイズが 1024 ビットを超えるのは、次のような場合です。
  - `device_global` が、1024 ビットを超えるサイズの配列である。
  - `device_global` が、1024 ビットを超えるサイズの ac_int (またはほかの大きな型) を使用するスカラーである。

    インテル® Quartus® Prime 開発ソフトウェア・プロ・エディションの不具合が原因で、MIF ファイルを使用してメモリーを初期化するときに発生するします。この問題を回避するには、メモリーにアクセスする前に `device_global` メモリーの内容をゼロに初期化します。
- `device_global` をリセットし、新しい `device_image` を `device_image` スコープ・プロパティーなしでロードすると、FPGA エミュレーション・フローで機能に問題が発生することがあります。現在、この問題の既知の回避方法はありません。
- FPGA フローで `atomic_ref` を使用してメモリーデータにアクセスするには、データ型の幅よりも大きなインターフェイスの幅が必要です。
- FPGA `mmhost` マクロはカーネルラムダでは機能しません。`mmhost` マクロは非推奨で、将来のリリースで削除される予定です。
- FPGA 向けのマルチステップ・コンパイルを実行しているときに、マルチアーキテクチャー・バイナリーのコンパイルに使用する FPGA 初期イメージファイル (`.a`) を作成し、2 つのコンパイルの間にファイル名を変更すると、2 回目のコンパイルは失敗します。
  例えば、次のコンパイルシーケンスは失敗します。

  ```
  icpx -fsycl -fintelfpga -fsycl-link=early -Xshardware -Xsboard=pac_a10
  vector_add.cpp *.cpp -o A.a
  icpx -fsycl -fintelfpga -Xshardware -Xsboard=pac_a10 A.a -o B.exe
  ```

  この問題を防ぐには、両方のコンパイルで同じファイル名を使用します。

  ```
  icpx -fsycl -fintelfpga -fsycl-link=early -Xshardware -Xsboard=pac_a10
  vector_add.cpp *.cpp -o B.a
  icpx -fsycl -fintelfpga -Xshardware -Xsboard=pac_a10 B.a -o B.exe
  ```
- FPGA コンパイルでは、SYCL* HLS カーネルで `annotated_arg` クラスを使用する場合、最小番号のバッファー位置のアドレス幅は 11 ビット以上でなければなりません。10 ビット未満の場合、コンパイラーはエラーを出力します。
- FPGA コンパイルでは、`buffer_location` への SYCL* 割り当て API の 1 回の呼び出しで割り当てられるメモリーの量は、最小番号のバッファー位置の合計サイズによって制限されます。

  この制限により SYCL* 割り当て API で null ポインターが返されるのを回避するには、最小番号の `buffer_location` のアドレス幅を `ceil(log2(the largest allocation size made + 1024))` ビット以上に設定します。ハードウェアでは、入力の場合は未使用アドレスのビットを 0 にして、出力の場合はそのままにしておきます。

次の割り当て API が影響を受けます。

- `sycl::malloc_shared`
- `sycl::malloc_host`
- `sycl::malloc_device`
- `sycl::aligned_alloc_shared`
- `sycl::aligned_alloc_host`
- `sycl::aligned_alloc_device`

- シミュレーション・フローで FPGA パイプライン・カーネルを実行すると、波形で最低の II が達成されないことがあります。
- FPGA アクセラレーション・フローで、`protocol<protocol_name::avalon_mm_uses_ready>` を使用するパイプによりコンパイラー・エラーが発生します。このエラーを防ぐには、ホスト USM などの異なる手法により、(実行中に) ホストとカーネル間でデータを渡します。
- インテル® FPGA PAC D5005 (旧称インテル® PAC インテル® Stratix® 10 SX FPGA 搭載版) では、FPGA カーネルで一連のストア操作の後にロード操作を行うと正しい結果が得られないという既知の問題があります。PAC BSP の提供は終了しました。インテル® FPGA PAC D5005 の交換用 BSP の入手については、インテルの担当者までお問い合わせください。
  例えば、次のコードでは、`data[1] = data[0];` 操作は、前の行のストア操作の前に発生します。この誤った操作順序により、`data[1]` の値が不正になります。

```
int main() {
  sycl::queue q{sycl::ext::intel::fpga_selector_v};
  volatile int *data = sycl::malloc_host<int>(2, q);
  data[0] = 0;
  q.parallel_for(sycl::range<1>(128), [=] (sycl::id<1> index) {
    if (index == 0)
      #pragma unroll
      for (int i = 0; i < 100; ++i)
        *data = i;
      data[1] = data[0];
    }
  }).wait();
  std::cout << "Result: " << data[1] << "\n";
}
```

- インテル® Stratix® 10 FPGA リファレンス・ボードでは、内部メモリーの初期化時にまれにエラーが発生し、メモリーが不明な状態に初期化されて、予期しない動作が発生します。
  この問題を回避するには、`-Xsbsp-flow=flat` コンパイラー・オプションを指定してデザインをコンパイルします。
- Intel Agilex® 7 ボードをターゲットにする場合など、大規模な FPGA デザインのシミュレーションでは、`PC-relative offset overflow (PC` 相対オフセット・オーバーフロー) メッセージを含むリンカーエラーが発生することがあります。
  このメッセージが出力された場合は、`-fsycl-link-huge-device-code` コンパイラー・コマンド・オプションを指定してシミュレーションをコンパイルします。
- FPGA カーネルでは、`no-interleaving=default` プロパティーを指定すると、バッファーはバーストインターリーブされません。これらのアドレスは、引き続き `mem_channel` プロパティーに従って割り当てられます。
  no-interleaving フラグを指定しない場合、`mem_channel` プロパティーは無視され、バッファーはバーストインターリーブされます。
  現在、この問題の回避方法はありません。

- FPGA デバイスでは、インテル® Quartus® Prime 開発ソフトウェアのバージョン 21.4 以前ではチャネル幅が 4096 ビットに制限されています。チャネル幅の制限を超えると、次のメッセージのようなエラーが表示されます。

  ```
  <name>_pipe_channel_read: dataBitsPerSymbol 5120 is out of range: 1-4096
  ```

  このエラーを回避するには、インテル® Quartus® Prime 開発ソフトウェアのバージョン 22.1 以降を使用してください。
- Microsoft* Windows* システムで FPGA `aocl` コマンドを使用すると、次のエラーが発生することがあります。

  ```
  aocl.exe: Unable to determine the execution environment of the Intel(R) FPGA SDK for OpenCL(TM). (aocl.exe: Unable to determine the execution environment of the インテル® FPGA SDK for OpenCL* の実行環境を決定できません。)
  aocl.exe: Detailed error: Could not determine the path to SDK internal Perl executable (aocl.exe: 詳細エラー: SDK 内部の Perl 実行ファイルのパスを決定できませんでした)
  ```

  このエラーが発生した場合は、次の手順を実行します。

  1. `aocl.exe` を <oneAPI-install-location>`\oneAPI\2024.0\bin` から <oneAPI-install-location>`\oneAPI\compiler\2024.0\opt\oclfpga\bin` にコピーします。
  2. `PATH` 環境変数の <oneAPI-install-location>`\oneAPI\2024.0\bin` の前に <oneAPI-install-location>`\oneAPI\compiler\2024.0\opt\oclfpga\bin` を追加します。
- FPGA 最適化レポートでは、すべてのラムダカーネルに一意の名前が付けられていない限り、複数のラムダカーネルを含むデザインは不正確な結果をレポートします。ラムダカーネルに名前を付ける方法の詳細は、『インテル® oneAPI FPGA ハンドブック』の「推奨コーディング・スタイル」(https://www.intel.com/content/www/us/en/docs/oneapi-fpga-add-on/developer-guide/current/suggested-kernel-coding-styles.html (英語)) を参照してください。
  ループ内で生成されたラムダカーネルには、テンプレート・クラスを使用してカーネルで手続き的に生成された名前を付けます。
- FPGA `ap_float` データ型では、`-fp-model=fast compiler` コンパイラー・コマンド・オプションでドット積推論が有効になりません。
  `ap_float` データ型でドット積推論を有効にするには、`-Xsffp-reassociate` コンパイラー・コマンド・オプションを使用します。
- SYCL* HLS フローの一部としてインテル® Quartus® Prime 開発ソフトウェアのバージョン 23.2 以降を使用する場合、デザインに次の条件を満たすメモリー・インターフェイスが含まれているとシミュレーションのコンパイルが失敗します。
  - メモリー・インターフェイスがグローバル・メモリー・リング・インターコネクトを使用する。
  - メモリー・インターフェイス・ポートが読み取り/書き込みまたは読み取り専用である。
  - メモリー・インターフェイスのレイテンシーが固定 (つまり、レイテンシーが明示的に 0 に設定されていない)。

    次のエラーメッセージが表示されます。

    ```
    Error: mm_agent_ks_mem0_rw.mm_agent_ks_mem0_rw.s0: Agent with readdatavalid must use waitrequest. (エラー: mm_agent_ks_mem0_rw.mm_agent_ks_mem0_rw.s0: readdatavalid のエージェントは waitrequest を使用しなければなりません。)
    ```

- FPGA アクセラレーション・フローのシミュレーションで、<global_mem_name> に割り当てる USM ポインターを使用するにも関わらず、割り当て先をコンパイラーに通知するアノテーションやバッファー位置プロパティーがないデザインは、ターゲットボードの `board_spec.xml` ファイルが次の条件を満たす場合、実行時にエラーになります。
    - 複数のデバイスのグローバルメモリーを定義する。
    - すべてのグローバルメモリーの最小アドレスと最大アドレスの差が、グローバルメモリー <global_mem_name> のサイズよりも大きい。
    - <global_mem_name> の最小アドレスがサイズより大きい。

      次のようなエラーが出力されます。

      `Error: Out of bounds memory write attempted at address <some_address>, for <size> bytes, max_size = <global_mem_name_size>` (エラー: アドレス `<some_address>`、`<size>` バイト、`max_size = <global_mem_name_size>` で範囲外のメモリー書き込みが試みられました)

      このエラーを回避するには、アノテーションやバッファー位置プロパティーを USM ポインターに追加して、割り当て先を指定します。
- FPGA で、次の例のようなループを使用して先頭のゼロなしで符号なし整数のビット数をカウントすると、「`Compiler Error: undefined reference to 'llvm.ctlz.iN'` (コンパイラー・エラー: `'llvm.ctlz.iN'` の未定義の参照)」などのコンパイラー・エラーが発生することがあります。

```
unsigned int leading_zeros = 0;
while (number) {
  leading_zeros += 1;
  number >>= 1;
}
```

  この問題を回避するには、最初に組込み関数 __builtin_clz(unsigned) または __builtin_clzll(unsigned long long) を使用して先頭のゼロの数を取得します。組込み関数を使用して unsigned char または unsigned short の先頭のゼロをカウントする場合、戻り値から型変換中に拡張されたビット数を引きます。
- FPGA SYCL* HLS フローでは、カーネル内のバッファー位置「X」を含む `annotated_arg<>` により定義された「書き込み専用」ポインター型のカーネル引数を含むデザインは、次の条件をすべて満たす場合、シミュレーションのコンパイルに失敗します。
    - バッファー位置「X」を含むポインターがすべてのカーネル本体でアクセスされない。
    - デザイン内のすべてのカーネルで、アノテーションのないポインター型のカーネル引数が定義または使用されていない。

      例えば、次のデザインをシミュレーション向けにコンパイルすると、コンパイルに失敗します。

```
struct kernelA {
  annotated_arg<int*, properties{buffer_location<0>,
readwrite_mode_write}> a;
  void operator()() {}  // a は使用されず、この操作は失敗します
}
```

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)
- インテル® oneAPI プログラミング・ガイド
- 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

以前のリリース

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

バージョン: 2024.0
2023 年 11 月 1 日

はじめに

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

- GPU
  - インテル® UHD グラフィックス (第 11 世代以降のインテル® Core™ プロセッサーに搭載)
  - インテル® Iris® Xᵉ グラフィックス
  - インテル® Arc™ グラフィックス
  - インテル® データセンター GPU フレックス・シリーズ
  - インテル® データセンター GPU マックス・シリーズ
- FPGA (次のいずれか):
  - FPGA カスタム・プラットフォーム (英語)
  - 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
• Intel Agilex® 7 • インテル® Arria® 10 • インテル® Stratix® 10 • インテル® Cyclone® 10 GX	インテル® Quartus® Prime 開発ソフトウェア・プロ・エディション (英語) バージョン 20.3 から 23.2

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
• Cyclone® V	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) バージョン 22.1

注: インテル® Quartus® Prime 開発ソフトウェアのオペレーティング・システム (OS) のサポートについての詳細は、インテル® FPGA Pro コンプリート・デザイン・スイートの OS サポートを参照してください。

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
• Red Hat* Enterprise Linux* 8.x、9.x • SUSE* Linux* Enterprise Server (SLES*) 15 SP3、SP4、SP5 • Ubuntu* 20.04 LTS、22.04 • Fedora* 37、38	• RHEL (Red Hat*) 8.6、9.2 • SLES* 15 SP5 • Ubuntu* 22.04 • Linux* カーネル 4.11 以降	• RHEL 8.9 • Ubuntu* 20.04 LTS • SLES* 15 SP3

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 開発用ソフトウェアのインストールのフロー (英語) を参照してください。

開発ツール

- コンパイラーがビルドされる分散ライブラリーの最小サポートバージョン: GCC - 7.5.0、BINUTILS-2.30、GLIBC-2.28、G++ - 7.5.0。使用している GNU* gcc の最新バージョンに同等の g++ パッケージがインストールされていない場合、エラーが発生します。詳細は、「Fatal Error: 'C++ Header' File Not Found with Intel® oneAPI DPC++/C++ Compiler (致命的なエラー: インテル® oneAPI DPC++/C++ コンパイラーで「C++ ヘッダー」ファイルが見つかりません)」 (英語) を参照してください。
- Eclipse* 開発者: Eclipse* 4.22
- 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 (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 デバイスまたはカスタム・プラットフォーム (英語)。
    - GCC 7.4.0 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

- インテル® ソフトウェア開発ツールは、第 11 世代以降のインテル® Core™ プロセッサーに内蔵されているグラフィックス・プロセッサーをサポートします。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・プロセッサーのサポートは 2023.1 リリースで終了しました。
- 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
- コンパイラーがビルドされる分散ライブラリーの最小サポートバージョン: GCC - 7.5.0、BINUTILS-2.30、GLIBC-2.28、G++ - 7.5.0。使用している GNU* gcc の最新バージョンに同等の g++ パッケージがインストールされていない場合、エラーが発生します。詳細は、「Fatal Error: 'C++ Header' File Not Found with Intel® oneAPI DPC++/C++ Compiler (致命的なエラー: インテル® oneAPI DPC++/C++ コンパイラーで「C++ ヘッダー」ファイルが見つかりません)」 (英語) を参照してください。
- 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* 2019、2022	• Windows* 10、11 (64 ビット) • Microsoft* Windows Server* 2019、2022	• Windows* 10 (64 ビット) • Microsoft* Windows Server* 2019

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
- インテル® ソフトウェア開発ツールは、第 11 世代以降のインテル® Core™ プロセッサーに内蔵されているグラフィックス・プロセッサーをサポートします。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・プロセッサーのサポートは 2023.1 リリースで終了しました。
- 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 2023.1 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 (開発コード名 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
- インテル® Arc™ A シリーズ・グラフィックス (開発コード名 DG2)
- インテル® データセンター GPU フレックス・シリーズ (開発コード名 Arctic Sound-M、略称 ATS-M)

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。

その他の開発ツール

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・ツールキット & インテル® HPC ツールキット (macOS* 版) 導入ガイド (英語)

Codeplay のプラグイン

- インテル® oneAPI DPC++ C++ コンパイラーで AMD* GPU を使用する場合は、oneAPI for AMD* GPU プラグインをインストールします。
- インテル® oneAPI DPC++ C++ コンパイラーで NVIDIA* GPU を使用する場合は、oneAPI for NVIDIA* GPU プラグインをインストールします。

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 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。

既知の問題

- インテル® oneAPI ベース・ツールキット、インテル® oneAPI ベース & HPC ツールキット、インテル® oneAPI ベース & IoT ツールキットのバージョン 2023.2 の一部として、または oneAPI スタンドアロン・コンポーネント・ページからインテル® コンパイラーをインストールした場合、その環境向けの適切なパッチをインストールしてください。

  oneAPI 2023.2 の一部として公開されたインテル® C++ コンパイラー向けに 1 つ、インテル® Fortran コンパイラー向けに 1 つ、合計 2 つのパッチが利用できます。

  * インテル® oneAPI DPC++/C++ コンパイラーおよびインテル® コンパイラー・クラシック
  * インテル® Fortran コンパイラー・クラシックおよびインテル® Fortran コンパイラー

  パッチのバージョンは 2023.2.1 です。

  これらのパッチは Linux* および Windows* に適用されます。

  これらのパッチは、環境モジュール・ユーティリティーのモジュールファイルが見つからない問題やその他の問題を解決します。

  パッチは、インテル® レジストレーション・センター、APT や YUM などのディストリビューション・チャネル、およびスタンドアロン・コンポーネント・ページから入手できます。
- 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。

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 2023 の動作環境 (英語)
- インテル® oneAPI DPC++/C++ コンパイラー 2022 の動作環境 (英語)
- インテル® oneAPI DPC++/C++ コンパイラー 2021 の動作環境 (英語)

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。