パート 5: DPC++ と Visual Studio* Code を使用した C++ プロジェクトの SYCL* への移行

この記事は 2023 年 6 月 30 日に Codeplay のウェブサイトで公開された「Porting C++ projects to SYCL with DPC++ and Visual Studio Code」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。

本シリーズのこれまでの記事では、Ubuntu* 20.04 プラットフォーム上で Visual Studio* Code (VSCode) 開発環境をゼロからセットアップしました。

本シリーズの記事:

- パート 1: DPC++ と Visual Studio* Code で SYCL* コードをデバッグ
- パート 2: Ubuntu* で C++ 開発向けに Visual Studio* Code を設定
- パート 3: Ubuntu* で SYCL* 開発向けに oneAPI、DPC++、Visual Studio* Code を設定
- パート 4: Ubuntu* で Visual Studio* Code を使用して DPC++ をデバッグ

はじめに

この記事では、VSCode IDE に慣れていない方向けに、Visual Studio* Code で DPC++ を使用するガイドを引き続き提供します。シェルベースの開発環境から (大規模な) C++ プロジェクトを移行して、VSCode の C/C++ ビルドおよび構成システムを使用するいくつかのアプローチについて簡単に説明します。これは通常、IDE 機能を利用してコード開発とデバッグを迅速化するために行われます。**CMake** や **Makefile** ビルド構成を使用するプロジェクトなど、ほとんどの C++ プロジェクトを、VSCode 用の Microsoft* C/C++ 拡張機能によって提供されるビルド構成システムに置き換える手順を紹介します。IDE と拡張機能を使用することで、コマンドライン・インターフェイスだけで開発する場合に比べて、以下のような利点が得られます。

- すべての構成とデバッグ機能 (ウォッチ変数とブレークポイントの場所など) は、VSCode の C/C++ プロジェクトのインスタンス間で一貫性があります。
- IntelliSense を使用して、コードベースのナビゲーションを迅速化し、暗黙的なコード定義を明らかにし、コード補完によりコードを素早く作成できます。
- **CMake** と同様に、1 つのプロジェクト内の複数のコンパイル・プロファイルを利用して、さまざまな種類のプログラムビルド (デバッグビルド、リーン・リリース・ビルド、デバッグ・リリース・ビルドなど) を構成できます。
- さまざまなデバッグ・セッション・タイプ (デバッグビルドやデバッグ・リリース・ビルド) を GUI のドロップダウン・リストから選択して、デバッグセッションを呼び出すことができます。
- デバッグセッションの開始時に、GUI プロンプトやリストからプログラムの引数を入力することで、さまざまなケースを迅速にデバッグできます。
- コード内に任意の数のブレークポイントを設定し、関数のステップイン、ステップオーバー、ステップアウト、ステップアップを行うことができます (図 2)。
- デバッグセッション中に、デバッガーからの出力メッセージを表示したり、より高度なデバッグのため命令を直接実行するようにデバッガーに指示できます。

- Microsoft* C/C++ 拡張機能のデバッグビューや追加の拡張機能を使用して、プログラムやリソースの状態をリアルタイムで表示して操作できます (図 1)。

本ガイドは、パート 2 の内容を拡張し、DPC++ プロジェクトの依存関係を取り込むコンパイラー・オプションを特定する方法を説明します。これらの依存関係には、oneAPI マス・カーネル・ライブラリー (oneMKL) などのサポート・ライブラリーや、その他のサードパーティー製ライブラリーが含まれます。必要なインクルード・パスとライブラリー・パスを VSCode プロジェクトのビルド設定ファイルに追加することで、ビルドを成功させることができます。

> **注:** プラットフォームに依存しないプラットフォーム構成ツールである CMake は、CMake Tools 拡張機能により VSCode と連携し、すべての機能を提供します。VSCode のコマンドパレットからプログラムをビルドしたり、IDE を使用してプログラムをデバッグすることはできますが、IntelliSense 機能は利用できません。

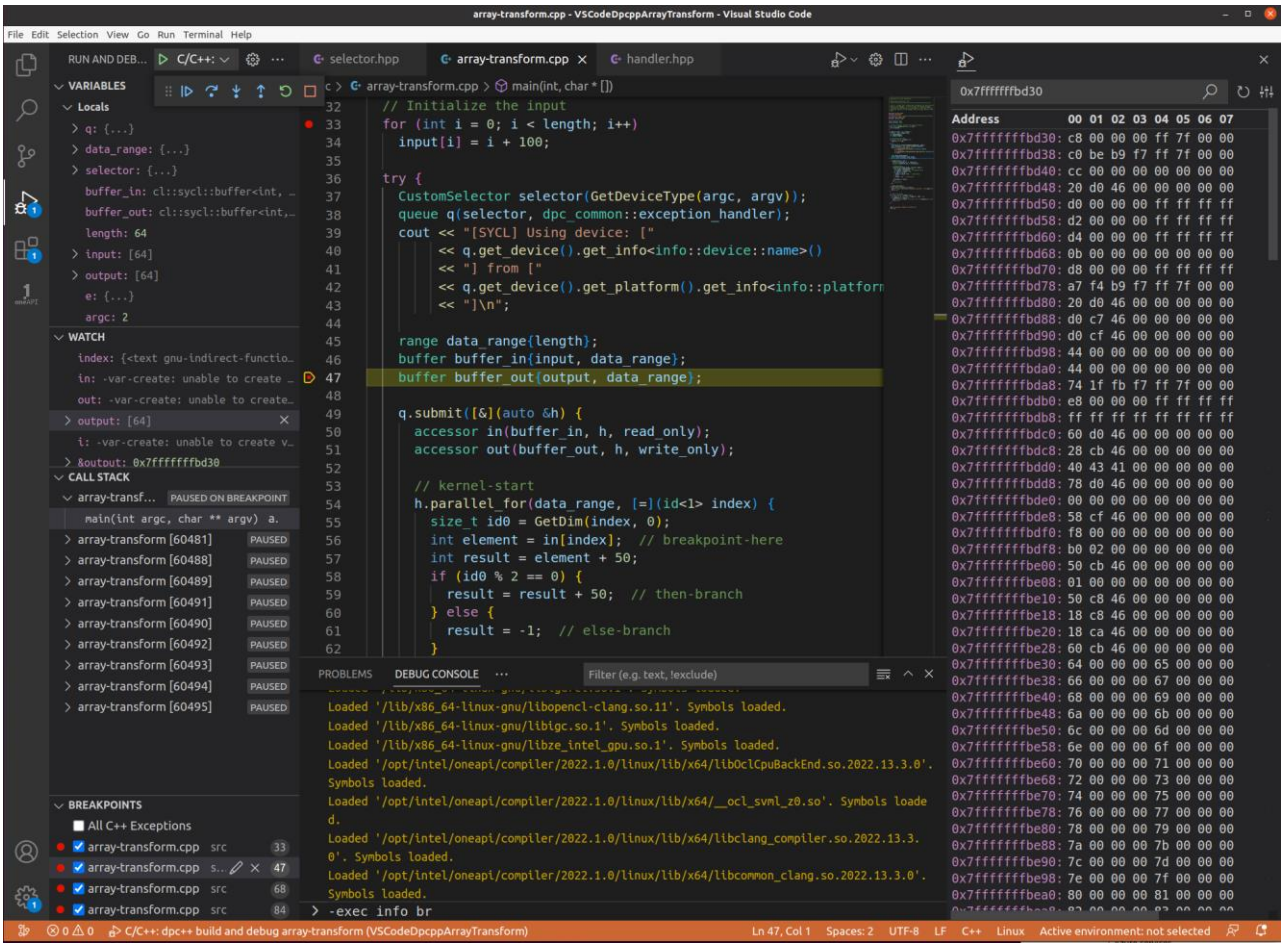

図 1: DPC++ プログラムのデバッグセッション

VSCode はビルドシステムをネイティブにサポートしていないため、VSCode を新規インストールしただけではデバッグできません。さまざまなプラグインや拡張機能を追加することで、必要な機能を利用できるようになります。上記の利点を活用して C/C++ プログラムを開発するには、Microsoft* C/C++ 拡張機能が必要です。

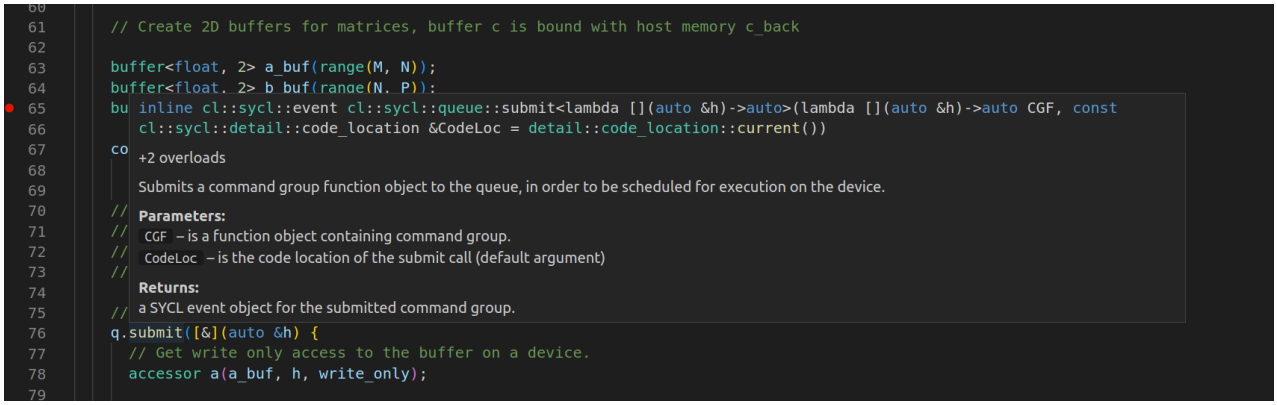

図 2: DPC++ プログラムのデバッグセッション

Microsoft* C/C++ 拡張機能は、コンパイルタスクとデバッグセッションの両方を解析する、各種 JSON 設定ファイルを提供します。これらの設定ファイルは、拡張機能が提供するさまざまな VSCode コマンドを使用して編集できます。拡張機能は、利用可能なコンパイラーやデバッガーをプラットフォームから検索して、基本的な設定を作成する「ヘルパー」を提供します。ほとんどの場合、これによって **HelloWorld** のような標準的なシングルソースの C++ ファイルのプログラムをビルドしてデバッグできるようになります。しかし、移行プロジェクトのファイル構造に合わせたり、データ並列 C++ (DPC++) コンパイラーや **gdb-oneapi** のような非標準のコンパイラーやデバッガーを使用する場合は、これらの設定を編集する必要があります。幸いなことに、ほとんどの場合、同じコンパイラーとデバッグターゲットが使用されるため、必要な情報の多くは既存のビルドシステム構成スクリプト (つまり、移行プロジェクトの Makefile または CMake ファイル) から得られます。

開始方法

開始するには、VSCode でアプリケーションを開き、Extensions ビューに移動します。拡張機能の検索ボックスに **Microsoft C/C++ extensions…** と入力すると、拡張機能が表示されます。**Microsoft C/C++ Extension Pack** を選択します。

VSCode の設定ファイルの概要

Microsoft C/C++ 拡張機能の設定ファイルの基本セットを表 1 に示します。

設定ファイル	説明
tasks.json	1 つ以上のビルド構成を含み、各構成には実行形式ごとのビルド設定が含まれます。構成ごとに特定のコンパイラーを指定できます。
launch.json	1 つ以上のデバッグ構成を含み、各構成には実行形式ごとのデバッグ設定が含まれます。
c_cpp_properties.json	コンパイラーのパス、プログラムのインクルード・パス、IntelliSense の定義が含まれます。

表 1: Microsoft* C/C++ 拡張プロジェクトの設定ファイル

すべてのプロジェクトには、プロジェクト・フォルダーの直下に `.vscode` フォルダー、または VSCode の**フォルダー**があります。このフォルダーは隠しフォルダーである可能性があります。表 1 にリストされているすべてのファイルは、`.vscode` フォルダーにあります。

ニーズに応じて、追加の設定ファイルをプロジェクトに追加して、多くのパスとファイルの依存関係があるプロジェクトの自動化や利便性を高めることができます。一例として、後述するオプションの **compile_commands.json** 設定ファイルが挙げられます。

c_cpp_properties.json ファイルは、IntelliSense を必要なインクルード・パスと定義に誘導するように設定します。IntelliSense が適切に設定されていない新しいプロジェクトをセットアップすると、図 3 に示すように、IntelliSense は認識できないものにフラグを立てます。

```
16
17  #include <CL/sycl.hpp>
18  #include <iostream>
19  #include <limits>
20
21  // dpc_common.hpp can be found in the dev-utilities include folder.
22  // e.g., $ONEAPI_ROOT/dev-utilities/<version>/include/dpc_common.hpp
23  #include "dpc_common.hpp"
24  #include "device_selector.hpp"
25
```

図 3: 波線の下線は IntelliSense が認識できないインクルード

プロジェクトの移行アプローチ

> **注:** 初めて VSCode IDE を使用して DPC++ プロジェクトをセットアップする方法や注意すべき点については、このシリーズのパート 1 を参照してください。

Microsoft* C/C++ 拡張機能を使用するため、既存のプロジェクトを移行するいくつかのアプローチがあります。

1. プロジェクトを作成して手動で設定する
2. 既存の Microsoft* C/C++ 拡張プロジェクトの設定ファイルをコピーして、新しいプロジェクトに合わせて編集する
3. 既存のビルド構成で **Bear** ツールを使用して、プロジェクトに追加できるコンパイル・データベースを作成する

1. 手動でゼロから設定する

以下は、**gcc** コンパイラーとデバッガーを使用して、単純な標準 Microsoft* C/C++ 拡張プロジェクトを作成する手順の概要です。gcc コンパイラーはデフォルトで提供されるため、拡張機能はプラットフォーム上の gcc コンパイラーとその使用方法を認識しています。しかし、インテルのコンパイラーとデバッガーは認識されないため、設定ファイルの場所を指定する必要があります。インテル® DPC++ コンパイラーとデバッガーは gcc と互換性があるため、設定ツールが定義するオプションのほとんどは両方で同じです。

以下の手順に従って、標準 C++ プロジェクトを作成します。

1. プロジェクトの名前でプロジェクト・フォルダーを作成します。
2. 既存のプロジェクトのディレクトリー構造をコピーするか、新しいファイル構造を定義します。
3. コードファイルを適切なフォルダーにコピーします。

4. ターミナルウィンドウでプロジェクトのトップフォルダーに移動し、`code .` と入力して VSCode アプリケーションを実行します。
5. ビルドタスクを生成します。VSCode の [Explorer] ペインで、プロジェクトのメインの .cpp ファイルを選択します。**Ctrl + Shift + P** で VSCode コマンドパレットを選択し、**C/C++:** と入力して、リストから **Select a configuration** を選択し、次にコンパイラーを選択します。
6. プロジェクトのコードファイル構造に合わせるため、新しいビルドタスク .json ファイルを開き、コンパイラーへの引数を以下のように編集します。

```
"-I${workspaceFolder}/include"
"${workspaceFolder}/src/*.cpp"
```

7. ビルドタスクが正しく設定されていることを確認するため、IDE のデバッグ・ターミナル・ペインを開いて、コンパイラーの起動後の進行状況を確認します。
8. プロジェクトをビルドするには、コマンドパレット (**Ctrl + Shift + P**) を使用し、ビルド構成 (例: Debug ビルド) を選択します。
9. コンパイルエラーが出力された場合は修正し、そうでない場合は次のステップに進みます。
10. IntelliSense 設定ファイルを生成します。
    a. **[Explorer]** ペインでファイルの選択を解除します。
    b. **Ctrl + Shift + P** を押して、**C/C++:** と入力し、リストから **Edit configuration (UI)** を選択します。
11. デバッグセッションの launch.json ファイルを生成します。VSCode の **[Explorer]** ペインでメインの .cpp ファイルを選択し、**Ctrl + Shift + P** を押して、**C/C++:** と入力し、リストから **Add debug configuration** を選択し、次にデバッガーを選択します。
12. デバッグセッションを開始するには、**[Explorer]** ペインでメインの .cpp ファイルを選択し、**Ctrl + Shift + Alt + D** を押します。
13. デバッグモードで、main の後のコード行を選択し、**F9** を押してブレークポイントを設定します。
14. **F5** を押してプログラムの実行を開始します。
15. プログラムがビルドされ、実行されると、ブレークポイントで停止するはずです。

注:
VSCode に GUI から利用可能なデバッグ機能がない場合、デバッグ・コンソール・ウィンドウのコマンドプロンプトにデバッガーのコマンドを入力することで、デバッガーを直接使用できます。コマンドの前に -exec を追加してください。

C++ プロジェクトを DPC++ プログラムのコンパイルとデバッグ用に変換するには、表 1 に示す基本設定ファイルを編集し、gcc コンパイラーとデバッガーを DPC++ コンパイラーとデバッガーのパスに置き換えます。

注:
DPC++ プログラムのデバッグ中に VSCode IDE が応答し続けるようにするには、プロジェクトのデバッグ起動設定ファイルに次の .json テキストを追加します。このコマンドは、gdb-oneapi デバッグセッションが SYCL* 関数 **queue(DeviceSelector.select_device(),...)** のステップインまたはステップオーバーを要求されたときにハングするのを防ぎます。

```
"setupCommands": [
    {
        "description": "Intel gdb-oneapi disable target async",
        "text": "set target-async off",
        "ignoreFailures": true
    }]
```

2. 既存のプロジェクトの設定をコピーする

1 つの DPC++ プロジェクトが、必要なコンパイラー設定とデバッグシナリオで動作していれば、以下の手順に従って、既存の DPC++ プロジェクトをテンプレートとして使用し、新しい DPC++ プロジェクトを迅速かつ簡単に作成できます。

1. プロジェクトの名前でプロジェクト・フォルダーを作成します。
2. プロジェクトのトップフォルダー直下に既存のファイル構造をコピーするか、新しいファイル構造を定義します。
3. 希望するビルド構成とデバッグ構成に最も近い「テンプレート」プロジェクトから .vscode フォルダーとその内容をコピーし、プロジェクトのトップフォルダーにペーストします。
4. 新しいプロジェクトのファイル構造、プロジェクト名、ビルド構成とデバッグ構成に合わせて、.vscode フォルダー内の各設定ファイルを編集します。
5. VSCode IDE を開き、GUI から **[Open Folder…]** を選択し、プロジェクト・フォルダーを指定します。
6. これで、新しいプロジェクトは、設定を使用してコードをビルドし、デバッグできるようになるはずです。

   注:
   設定ファイルの一部オプションは、同じプロジェクトの別の設定ファイルのオプションを名前で参照している可能性があるため、古いプロジェクトのテキスト参照をすべて新しいプロジェクトのものに変更します。

3. Bear を使用してコンパイラー・パスとコマンドを検索して作成する

プロジェクトを移行するため前述の 2 つのアプローチのいずれかを使用することに加えて、特定のビルドに使用されている設定を検出できる Bear ツールを使用して、プロジェクトのコンパイラー・パスの依存関係や、(場合によっては暗黙の) コンパイラー設定を明らかにすることができます。

Ubuntu* で利用できる Bear ツールコマンドは、コンパイル・データベースを生成します (図 4)。JSON コンパイル・データベースには、単一のコンパイル単位がどのように処理されたかに関する情報が含まれています。その主な用途は、検出された情報を使用して、異なるプログラムで同じコンパイル構成を再実行することですが、同じ情報を使用して、新しいプロジェクトの構成を定義することもできます。これを使用して、プロジェクトの**提案なし**に追加の検索パスを追加し、追加のファイルと定義を IntelliSense に認識させることができます。

```
{
"arguments": [
"/usr/bin/x86_64-linux-gnu-g++-10",
"-c",
"-pipe",
"-O2",
"-std=gnu++11",
"-Wall",
"-Wextra",
"-D_REENTRANT",
"-fPIC",
"-DQT_NO_DEBUG",
"-DQT_WIDGETS_LIB",
"-DQT_GUI_LIB",
"-DQT_CORE_LIB",
"-I../TestQtApp",
"-I.",
"-I/usr/include/glib-2.0",
"-I/usr/lib/x86_64-linux-gnu/glib-2.0/include",
"-I../../../Qt/5.15.2/gcc_64/include",
"-I../../../Qt/5.15.2/gcc_64/include/QtWidgets",
"-I../../../Qt/5.15.2/gcc_64/include/QtGui",
"-I../../../Qt/5.15.2/gcc_64/include/QtCore",
"-I.",
"-I/usr/include/libdrm",
"-I.",
"-I../../../Qt/5.15.2/gcc_64/mkspecs/linux-g++",
"-o",
"main.o",
"../TestQtApp/main.cpp"
],
"directory": "/home/milosz/Documents/projects/build-TestQtApp",
"file": "/home/milosz/Documents/projects/build-TestQtApp/../TestQtApp/main.cpp'
"output": "/home/milosz/Documents/projects/build-TestQtApp/main.o"
}
```


図 4: QT プロジェクトの qmake ビルドシステムで Bear を使用して生成した compile_commands.json ファイルの例

注:
compile_commands.json の例の出展:
https://www.kdab.com/improving-cpp-dev-in-vs-code/ (英語)

JSON コンパイル・データベースの定義は、以下を参照してください。
http://clang.llvm.org/docs/JSONCompilationDatabase.html (英語)

c_pp_properties.json ファイルにデータを設定する方法については、以下を参照してください。
https://code.visualstudio.com/docs/cpp/customize-default-settings-cpp (英語)

Bear ツールの使用方法:

1. ターミナルウィンドウでプロジェクトの make コマンドを実行可能なフォルダーに移動します。
2. ターミナルに `Bear – <make コマンド>` を入力します。
3. 新しいプロジェクトの設定に必要なオプション、パス、引数を取得します。

注:
インストールされている Bear ツールのバージョンによっては、失敗する場合があります。その場合は、コマンドの「–」を省略してみてください。

Bear ツールは以下から入手できます。
https://github.com/rizsotto/Bear (英語)

Microsoft* C/C++ 拡張機能は、コンパイル・データベース・ファイルを直接参照できるため、c_cpp_properties.json ファイルにすべてのオプションを記述する必要がなくなります。

```
[
    {
        "arguments": [
            "/opt/intel/oneapi/compiler/2022.1.0/linux/bin-llvm/clang++",
            "-c",
            "--dpcpp",
            "-fsycl",
            "-g",
            "-O0",
            "-std=gnu++17",
            "-o",
            "CMakeFiles/array-transform.dir/src/array-transform.cpp.o",
            "../src/array-transform.cpp"
        ],
        "directory": "/home/illya/Dev/Intel-oneAPI-examples/array-transform/build",
        "file": "../src/array-transform.cpp"
    }
]
```

図 5: 既存の DPC++ プロジェクトの CMake ファイルのビルド設定を Bear ツールで実行し、コンパイル・データベースを生成した例

Microsoft* C/C++ 拡張プロジェクトにコンパイル・データベースの使用を指示するには、コマンドパレットに **C/C++** と入力して、リストから **Edit configuration (UI)** を選択し、**[Advanced]** セクションまでスクロールします。**[Advanced]** セクション以下にある **[Compile Commands]** にデータベース・ファイルのフルパスと名前を入力します。

まとめ

紹介した 3 つのアプローチのうち、別の既存のプロジェクトから設定ファイルをコピーするアプローチが、圧倒的に簡単で、数分しかかかりません。

Bear ツールは、複雑なプロジェクトのビルドを分解して、暗黙を含むコンパイラー・オプションやインクルード・パスに関する貴重なヒントを提供してくれます。

次のステップ

次のパート 6 では、本パートで紹介した手順を使用して、oneAPI の多数のユーティリティー・ライブラリーの 1 つである oneMKL 数学ライブラリーにリンクする DPC++ VSCode プロジェクトを作成します。
