OpenCL* Code Builder で SPIR* を使用する

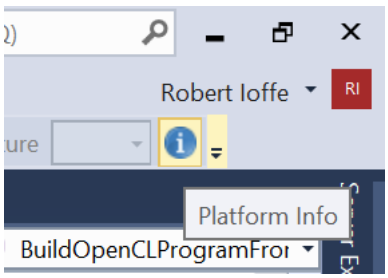
この記事は、インテル® デベロッパー・ゾーンに公開されている「Using SPIR for fun and profit with Intel® OpenCL™ Code Builder」の日本語参考訳です。

はじめに

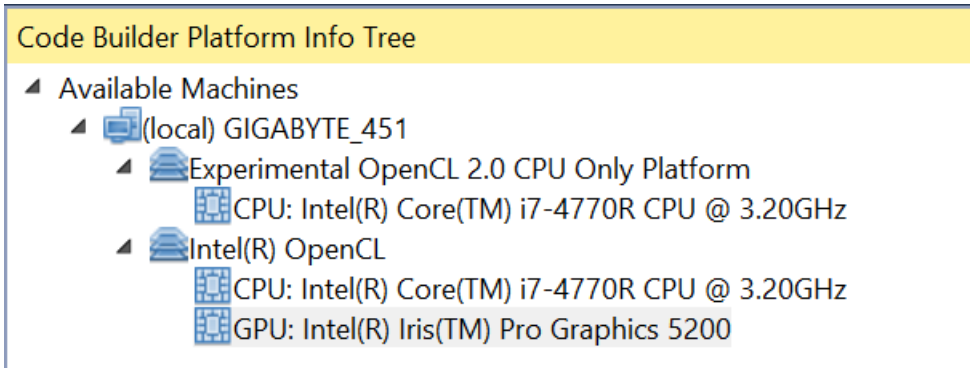
この記事では、Khronos* の SPIR* について簡単に紹介し、SPIR* バイナリーとインテルの中間バイナリーの違いに触れます。そして、インテル® INDE のツールを使って SPIR* バイナリーを生成するいくつかの方法と OpenCL* プログラムで SPIR* バイナリーを使用する方法を示します。ここで紹介するツールのインストール方法は、こちらの記事 (英語) をご覧ください。

SPIR* とは?

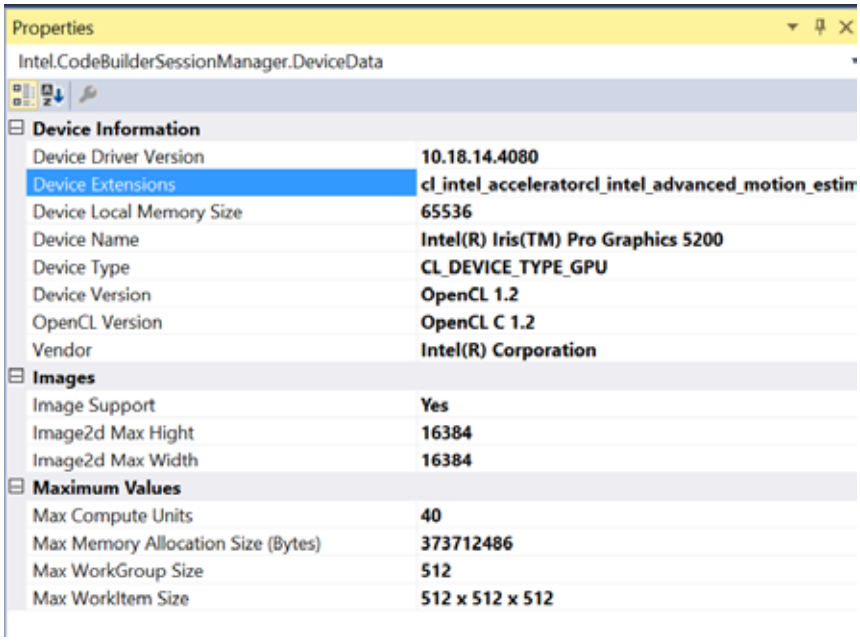
SPIR* は、The Standard Portable Intermediate Representation の略で、LLVM IR ベースの OpenCL* C デバイスプログラムの移植可能なバイナリー・エンコーディングです。SPIR* の主な目的は、ベンダーとデバイス間の移植性を損なうことなく、アプリケーション開発者がカーネルをソース形式で配布しなくても済むようにすることです。SPIR* は Khronos* 拡張なので、ターゲットデバイスが cl_khr_spir 拡張をサポートしているかどうか確認する必要があります。第 4 世代およびそれ以降のインテル® Core™ プロセッサーを搭載し、最新のドライバーがインストールされた Windows* および Android*システムはすべて SPIR* をサポートしています。インテル® INDE とインテル® Media Server Studio に含まれる OpenCL* Code Builder がインストールされている場合、Microsoft* Visual Studio* のプラットフォームおよびデバイス情報で確認できます。

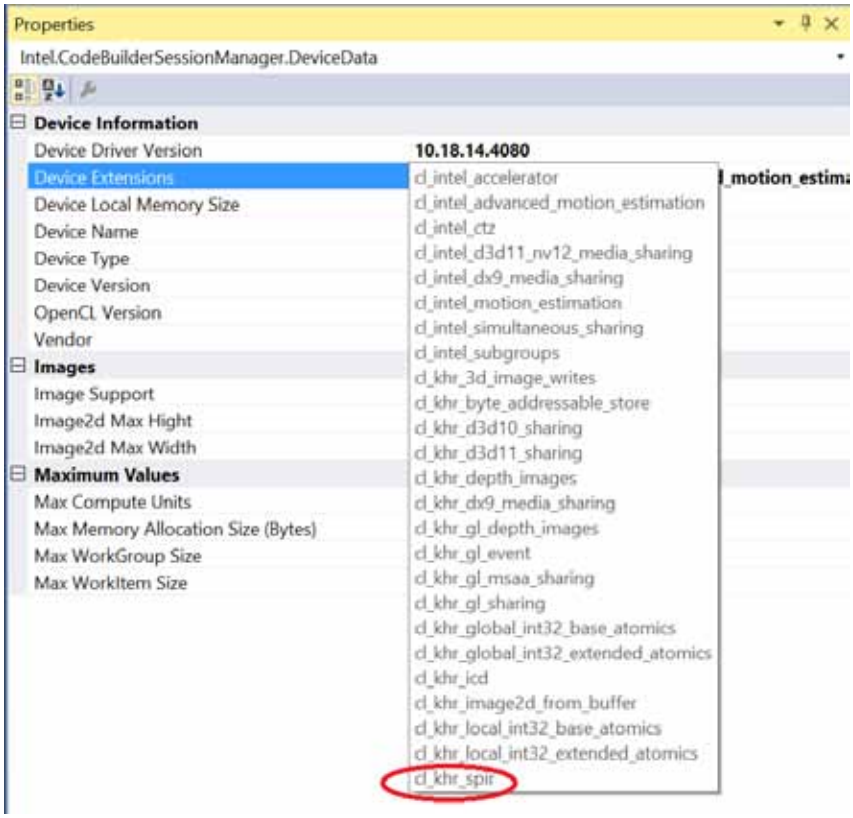


例えば、第 4 世代インテル® Core™ プロセッサー以上を搭載した Windows* システム向けの最新のインテル® グラフィックス・ドライバー (10.18.14.4080) は、3 つの OpenCL* デバイスをサポートします: OpenCL* 1.2 CPU デバイス、OpenCL* 1.2 GPU デバイス、および OpenCL* 2.0 CPU デバイス (試験的にサポート)。



これらのデバイスはすべて cl_khr_spir 拡張をサポートします。





SPIR* バイナリーと中間バイナリーの違いは?

SPIR バイナリーは、同一ベンダーの異なるデバイス間でも、異なるベンダーのさまざまなデバイス間でも、移植性があります。例えば、NVidia* または AMD* 製のグラフィックス・カードが装着された、インテル® プロセッサー・グラフィックス搭載のインテル® プロセッサー・ベースのマシンの場合、少なくとも次の 3 つのデバイスが表示されます: インテル CPU デバイス、インテル GPU デバイス、および NVidia* または AMD* デバイス。SPIR* バイナリーは、これらすべてで動作します。

インテルのコマンドライン・コンパイラーで SPIR* バイナリーを生成する方法

OpenCL* Code Builder には、OpenCL* C 用のコマンドライン・コンパイラーが含まれています。コマンドラインで、次のコマンドを入力します。

```
ioc64 -cmd=build -input=SobelKernels.cl -device=gpu -spir64=SobelKernels_x64.spir -bo="-cl-std=CL1.2"
```

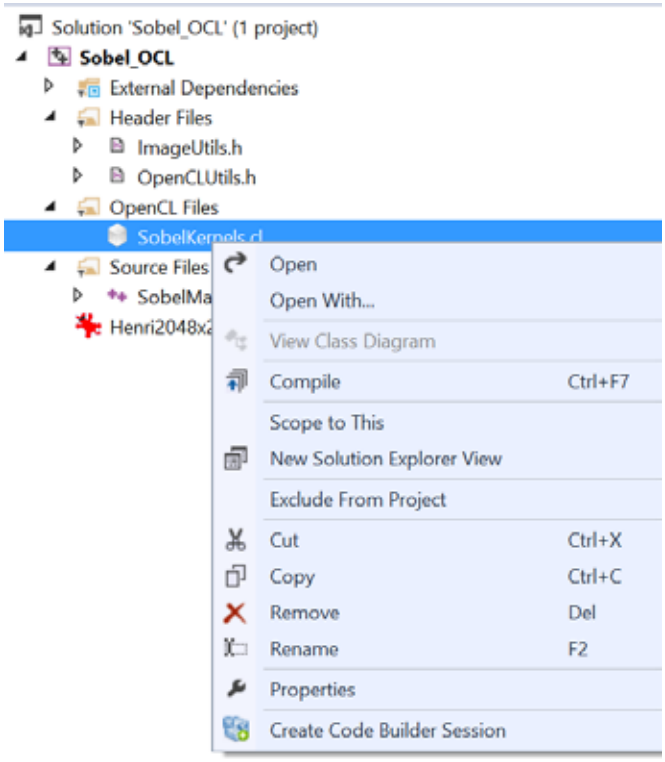
または、次のようにデバイスを省略して入力することもできます。

```
ioc64 -cmd=build -input=SobelKernels.cl -spir64=SobelKernels_x64.spir -bo="-cl-std=CL1.2"
```

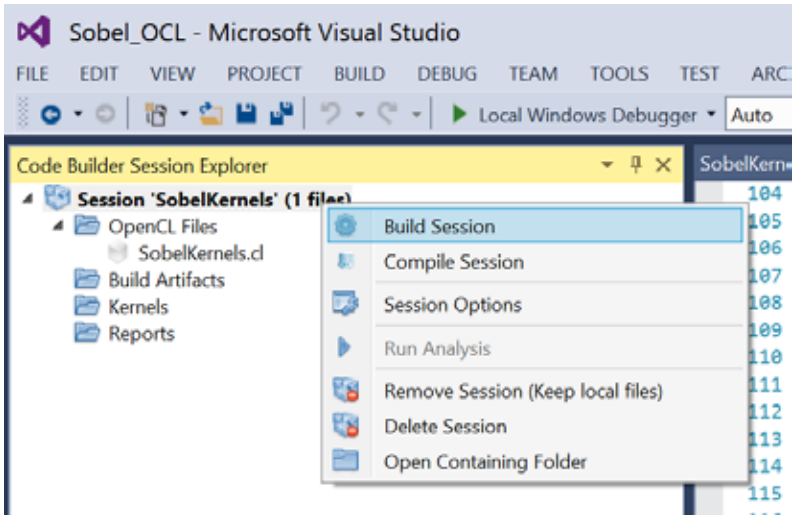
デバイスの指定は、そのデバイス向けにカーネルがコンパイルされること保証するだけであって、コンパイル結果には影響しません。1 つ目のコマンドではカーネルは GPU 向けにコンパイルされ、2 つ目のコマンドではカーネルはデフォルトの CPU 向けにビルドされますが、どちらの場合も生成される SPIR* ファイルは同じです。さらに、開発マシンが SPIR* をサポートしていなくても SPIR* ファイルを生成することができます。インテル® Media Server Studio に含まれる OpenCL* Code Builder では、Linux* でも SPIR* の生成がサポートされています: サポートされている Linux* プラットフォームで、スタンドアロンの Kernel Builder または Eclipse* プラグインを使用して SPIR* バイナリーを生成することができます。

インテル® INDE の Kernel Builder で SPIR* バイナリーを生成する方法

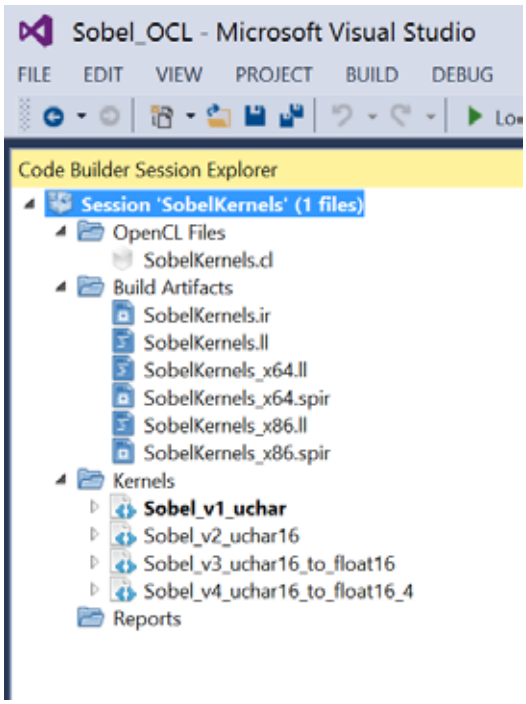
OpenCL* ファイルを含むソリューションを開きます。OpenCL* ファイルを右クリックして、ポップアップ・メニューから [Create Code Builder Session] を選択します。



Code Builder Session Explorer で作成されたセッションを右クリックして、[Build Session] を選択します。



ビルドに成功すると、[Build Artifacts] 以下に .spir ファイルが生成されます。



ここでは、_x64.spir バージョンを使用します。SobelKernels_x86.ll と SobelKernels_x64.ll ファイルは、SPIR* バイナリー SobelKernels_x86.spir と SobelKernels_x64.spir のテキスト形式の表記です。解析時に、必要に応じて、これらのファイルを検証することができます。Win32 モードでアプリケーションをコンパイルすると、CPU デバイスは _x86.spir ファイルを使用し、GPU デバイスは _x64.spir ファイルを使用します。x64 モードでアプリケーションをコンパイルすると、CPU デバイスと GPU デバイスは、どちらも同じ _x64.spir ファイルを使用します。

OpenCL* プログラムで SPIR* バイナリーを使用する方法

SPIR* バイナリーからプログラムをビルドする前に、ターゲット・プラットフォームとターゲットデバイスが cl_khr_spir 拡張をサポートしているかどうか確認する必要があります。CL_PLATFORM_EXTENSIONS フラグとともに clGetPlatformInfo 呼び出しを使用して、cl_khr_spir 文字列を探します。また、CL_DEVICE_EXTENSIONS フラグとともに clGetDeviceInfo 呼び出しを使用して、プラットフォームの各デバイスの SPIR* サポートも確認します。通常の C/C++ API を利用して、バイナリーファイル (SobelKernels_x64.spir) を文字配列に読み込みます。そして、clCreateProgramWithBinary 呼び出しでプログラムを作成します。その後、"-cl-mad-enable" などの通常の最適化フラグとともに "-x spir" を指定して、clBuildProgram でプログラムをビルドします。これで、通常どおりカーネルを作成することができます。

SPIR* バイナリーの長所

SPIR* バイナリーは、さまざまなデバイスおよびベンダー間で移植性があります。

SPIR* バイナリーは、ネイティブ中間バイナリーよりも小さいです。

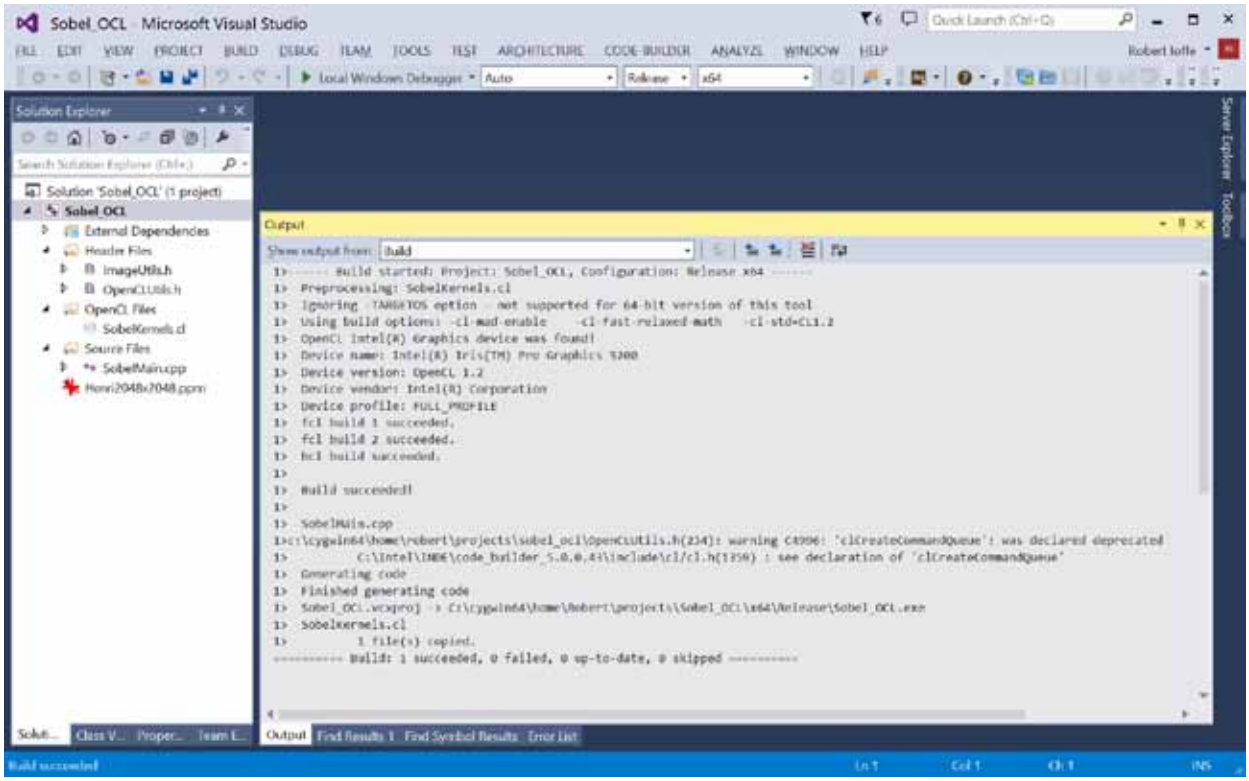
SPIR* バイナリーの短所

SPIR* バイナリーからプログラムをビルドする場合、追加の変換および最適化ステップが含まれるため、中間バイナリーからプログラムをビルドする場合よりも時間がかかります。

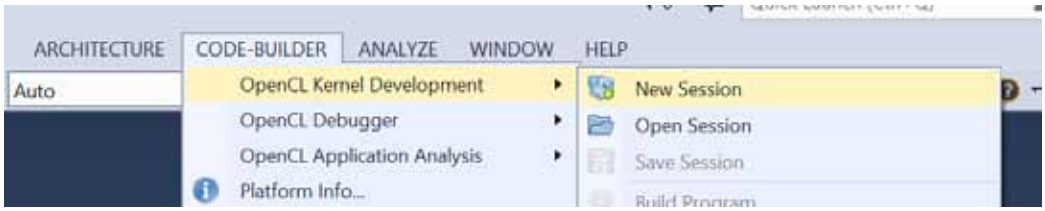
SPIR* バイナリーからプログラムをビルドする場合、フラグによって指定される最適化の一部が後のコンパイルフェーズで実施されるため、ビルドフラグを提供する必要があります。

SPIR* サンプルのビルドと実行

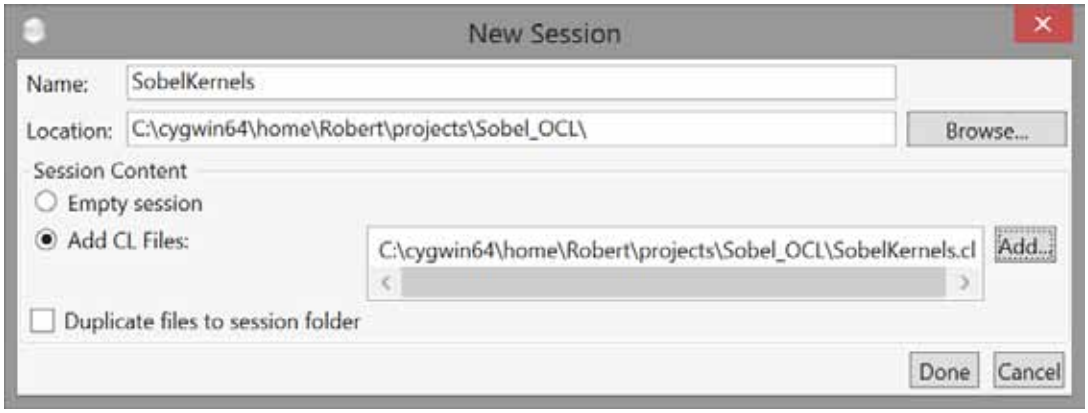
インテル® INDE またはインテル® Media Server Studio に含まれる OpenCL* Code Builder をインストールします (参考文献 を参照)。Sobel_OCL ソリューションを開き、ビルドします (ここで使用するサンプルコードは、記事の最後にあるリンクからダウンロードできます)。



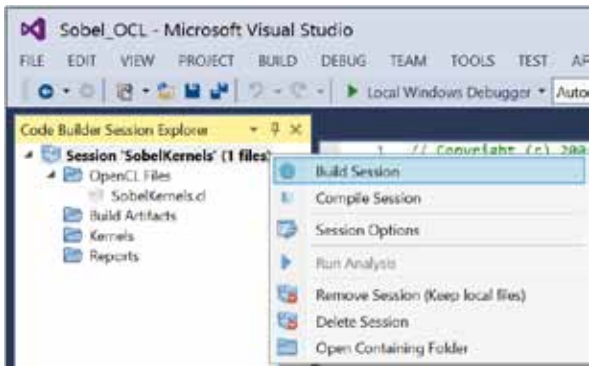
次のように、Code Builder Kernel Development セッションを作成します。



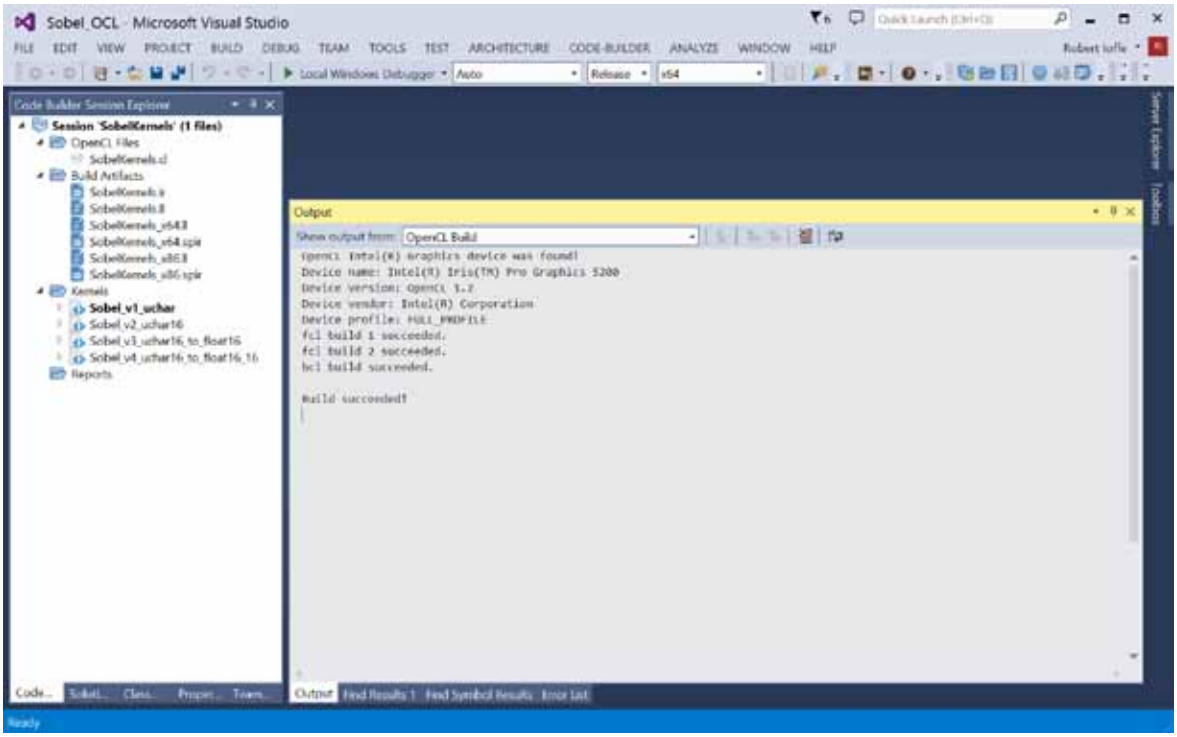
[New Session] ダイアログで、セッション名を SobelKernels、場所をソリューションがあるディレクトリーにし、[Session Content] の [Add CL Files] をオンにしてソリューションの SobelKernels.cl ファイルを指定します。そして、[Duplicate files to session folder] チェックボックスをオフにし、[Done] ボタンをクリックします。



Session 'SobelKernels' を選択して、ビルドします。



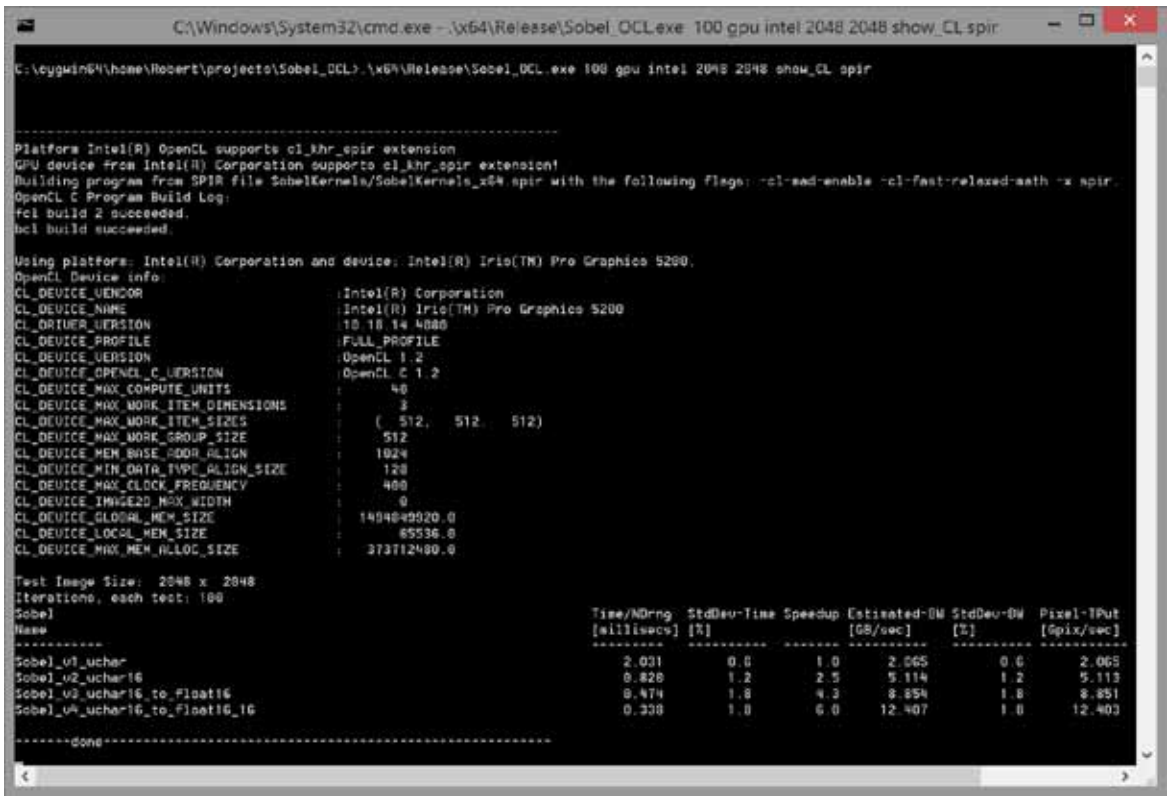
Build Artifacts フォルダーと Kernels フォルダーにファイルが生成されます。



これで、ワークロードを実行する準備ができました。コマンドウィンドウを開いて、ソリューション・ファイルのあるディレクトリーに移動し、次のコマンドを実行します。

```
.\x64\Release\Sobel_OCL.exe 100 gpu intel 2048 2048 show_CL spir
```

次のような出力が表示されます。



ディレクトリーに 4 つの *_validation.ppm ファイルが生成されます。これらのファイルには、次のような Sobel 処理された犬のイメージが含まれています。

次のコマンドを使用して、CPU OpenCL* デバイスでサンプルプログラムを実行します。

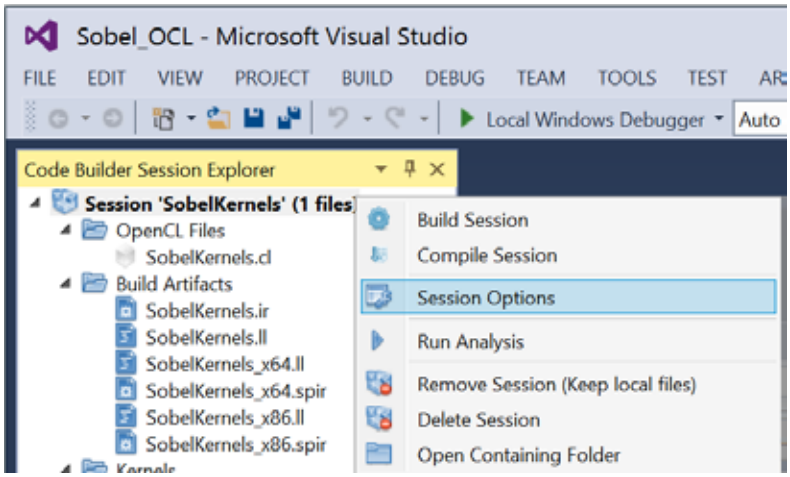
```
.\x64\Release\Sobel_OCL.exe 100 cpu intel 2048 2048 show_CL spir
```

GPU デバイスで **openclc** と **ir** オプションを指定してサンプルプログラムを実行することもできます。その場合、次のコマンドを使用します。

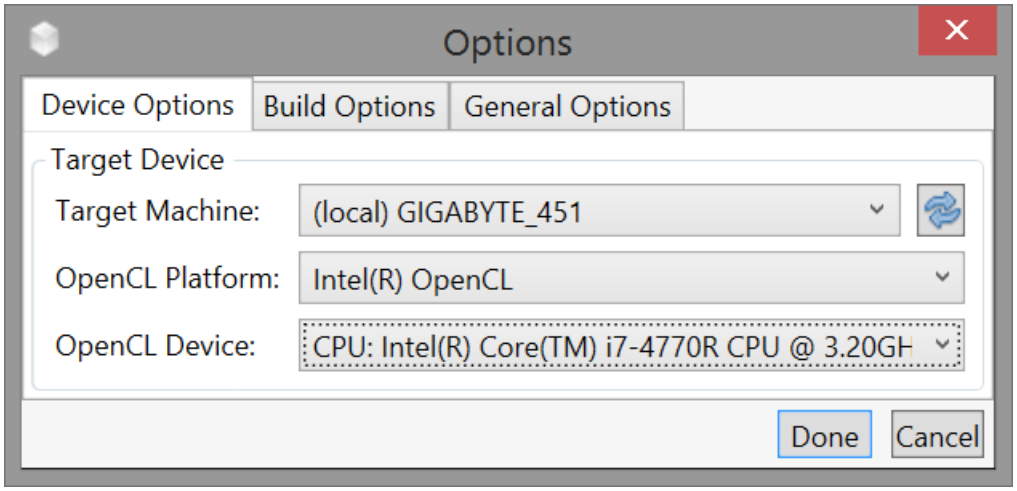
```
.\x64\Release\Sobel_OCL.exe 100 gpu intel 2048 2048 show_CL
openclc .\x64\Release\Sobel_OCL.exe 100 gpu intel 2048 2048 show_CL ir
```

各実行の最初に表示される OpenCL* C Program Build Log を確認し、CPU デバイスで SPIR* を実行した際の出力と比較してみましょう。中間バイナリーは完全なビルドでそのまま使用されるため、**ir** オプションを指定した場合は何も出力されません。

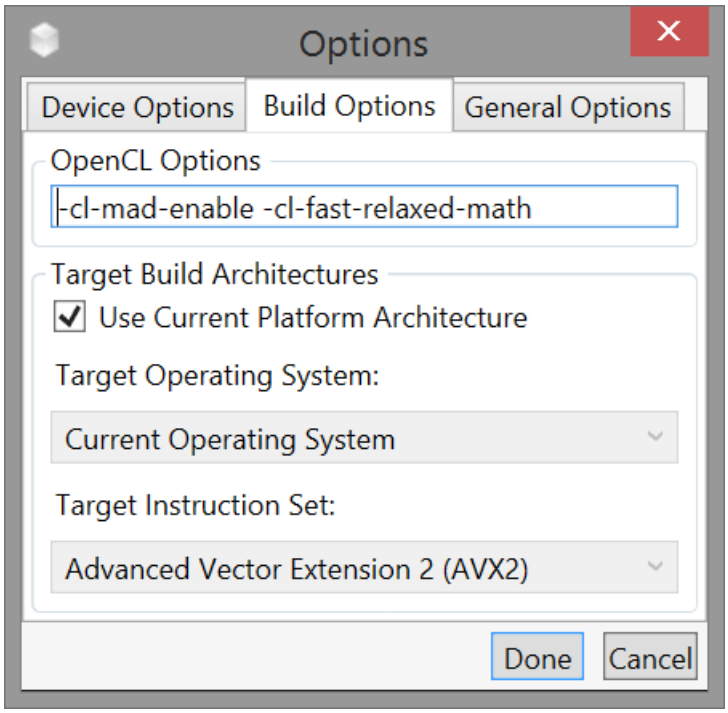
CPU デバイスで openclc と ir オプションを指定して実行することもできます。その場合、ir オプションを指定して実行する前にセッションのオプションを変更するのを忘れないでください。



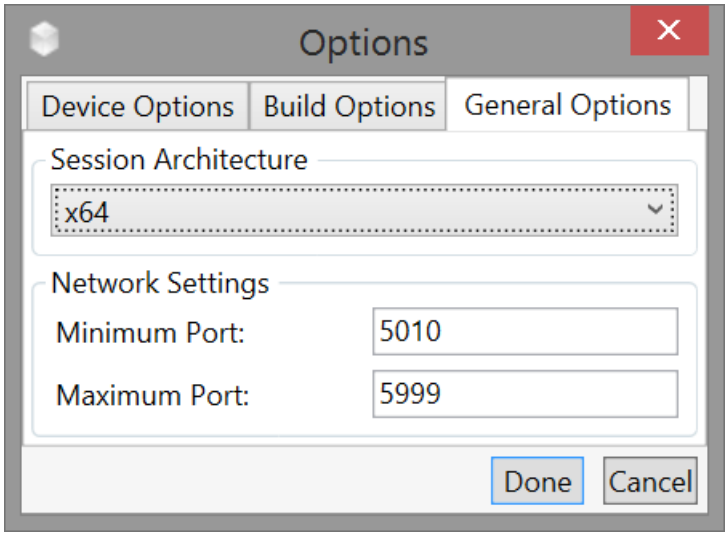
CPU デバイスを指定します。



適切なビルドオプションを指定します。



適切な Session Architecture を選択します。



セッションをリビルドします。Build Artifacts フォルダーに SobelKernels.asm が生成されます。SobelKernels.ir は、CPU デバイスの中間表現バイナリーで上書きされます。次のコマンドで、実行ファイルを実行します。

```
.\x64\Release\Sobel_OCL.exe 100 cpu intel 2048 2048 show_CL ir
```

結論

この記事では、SPIR* について簡単に紹介し、SPIR* バイナリーとインテルの中間バイナリーの違いに触れました。また、インテル® INDE に含まれるインテルのコマンドライン・コンパイラーと Kernel Builder の Microsoft* Visual Studio* アドオンを使用して SPIR* バイナリーを生成するいくつかの方法と、生成した SPIR* バイナリーを OpenCL* プログラムで使用する方法を示しました。

謝辞

この記事をレビューし、素晴らしいフィードバックを提供してくれた Uri Levy 氏に感謝します。

参考文献

1. Khronos* SPIR* Web サイト: https://www.khronos.org/spir (英語)
2. Khronos* SPIR* FAQ: https://www.khronos.org/faq/spir (英語)
3. インテル® INDE: http://www.isus.jp/article/intel-software-dev-products/intel-inde
4. インテル® Media Server Studio: http://www.isus.jp/article/idz/vc/media-server-studio
5. インテル® Code Builder for OpenCL* API for Linux*: https://software.intel.com/en-us/articles/intel-code-builder-for-opencl-api (英語)
6. OpenCL* Code Builder ユーザーマニュアル: https://software.intel.com/en-us/code-builder-user-manual (英語)
7. Quick Installation Guide for OpenCL™ Development on Windows* with Intel® INDE: https://software.intel.com/en-us/articles/getting-started-with-opencl-development-on-windows-with-intel-inde (英語)

著者紹介

Robert Ioffe は、インテル コーポレーションのソフトウェア & ソリューション・グループのテクニカル・コンサルティング・エンジニアです。OpenCL* プログラミングとインテル® Iris™ グラフィックスまたはインテル® Iris™ Pro グラフィックスにおける OpenCL* ワークロードの最適化のエキスパートで、インテル® グラフィックス・ハードウェアを熟知しています。Khronos* の標準化作業に深くかかわっており、これまでに最新機能のプロトタイプの作成やインテル® アーキテクチャーでの動作検証を行ってきました。最近では、OpenCL* 2.0 の入れ子構造の並列処理 (enqueue_kernel functions) 機能のプロトタイプの作成に取り組み、OpenCL* 2.0 用の GPU クイックソートを含む、いくつかの入れ子構造の並列処理サンプルコードを作成しました。また、以下の単純な OpenCL* カーネルの最適化に関する動画 2 つと OpenCL* 2.0 用の GPU クイックソートと シェルピンスキーのカーペットに関する動画を公開しています。

Optimizing Simple OpenCL Kernels: Modulate Kernel Optimization
Optimizing Simple OpenCL Kernels: Sobel Kernel Optimization
GPU-Quicksort in OpenCL 2.0: Nested Parallelism and Work-Group Scan Functions
Sierpiński Carpet in OpenCL 2.0

著作権と商標について

サンプルのダウンロード

ここで使用したサンプルコードは、インテル・サンプル・ソース・コード使用許諾契約書 (英語) の下で公開されています。
サンプルコードを今すぐダウンロード

コンパイラーの最適化に関する詳細は、最適化に関する注意事項を参照してください