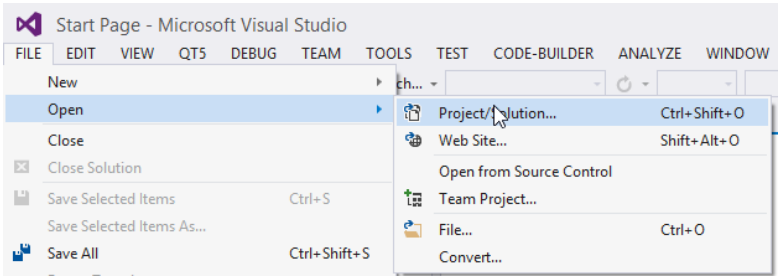
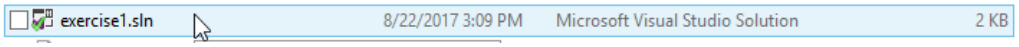
For this Tutorial, we will use Microsoft Visual Studio* as a primary C++ development environment with Intel® Parallel Studio XE extensions, using Intel® Compiler and Intel TBB as an Intel Performance Library option. We provide Makefiles if you prefer to run the exercises in a different environment (e.g. Linux or MacOSX*). All exercises depend on Intel® TBB 2018, some exercises require an existing OpenCL* installation on your system.

**IMPORTANT INFORMATION FOR THE PRACTICE:**

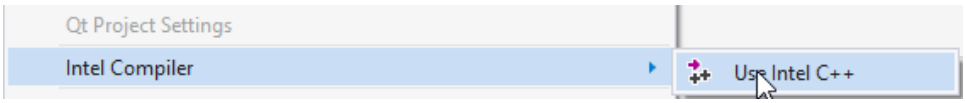
- Click on the Windows icon and type "Visual Studio" to start Visual Studio.
- You can open each exercise by clicking on File->Open->Project/Solution and the go to Documents\examples\exerciseN\msvs\exerciseN.sln



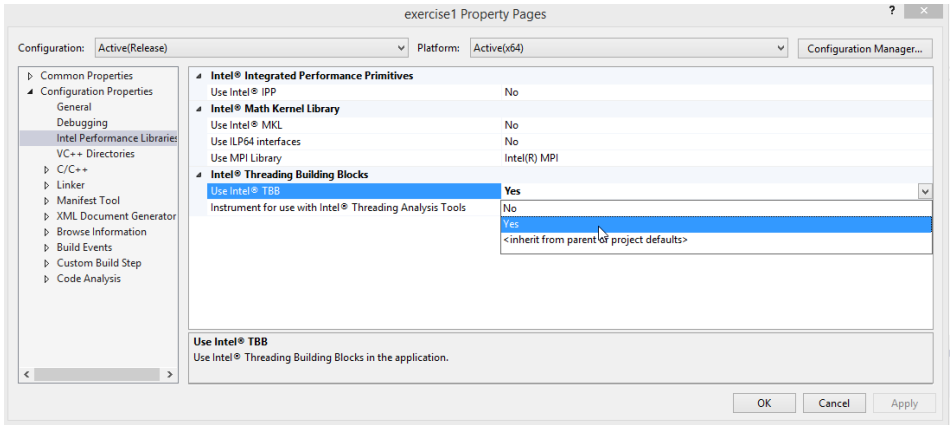

Select the file of Type "Microsoft Visual Studio Solution". The extension ".sln" may not be shown, by the Type will be shown.




- After opening the solution file, right-click on the project in the Solution Explorer and select Intel Compiler->Use Intel Compiler



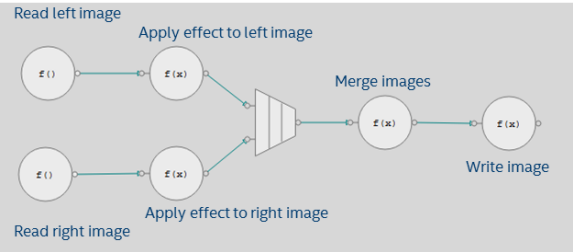

- Turn on Intel TBB so that the compiler can find the headers, .lib and .dlls. Again open the project properties and traverse to Configuration Properties -> Intel Performance Libraries -> Intel® Threading Building Blocks -> Use Intel® TBB as shown below. Select "Yes".



## Exercise 0: Inspect and execute the serial stereoscopic 3D example

1. Open the file exercise0.sln in Microsoft Visual Studio* from the working directory msvs in the exercise0 project directory
2. Open and inspect the stereo-serial.cpp file. The main function executes the following steps sequentially:
    a. Read left image
    b. Read right image
    c. Apply effect to left image
    d. Apply effect to right image
    e. Merge left and right images
    f. Write out resulting image
3. Build and run to generate an output.png file. Note the execution time.
4. The output image can be found in the working directory

## Exercise 1: Convert the stereo example in to an Intel TBB flow graph




1. Open the exercise1 project in Microsoft Visual Studio
2. Open exercise01.cpp, search for the 2 comments that start with TODO, and follow the instructions provided in those comments. You will create the right_transform node and add 4 missing edges to the graph.

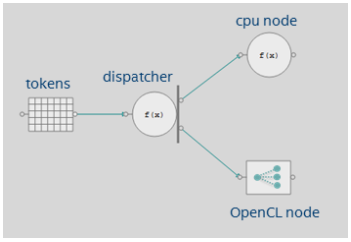
    *NOTE: A solution is provided in a separate solution01 project if you want to refer to it.*
3. When your modifications are complete, build and run the exercise to generate an output.png file. Note the execution time. The execution time should be smaller (on average) than that of the serial-stereo in exercise 0.
4. View the output image to confirm that the output has not changed.

## Exercise 2: Encapsulate stereo in a composite_node


1. Open the exercise2 project in Microsoft Visual Studio
2. Open exercise02.cpp, search for the 2 comments that start with TODO, and follow the instructions provided in those comments. You will complete the lambda expressions that are passed to the constructors of the `right_transform` and `image_merge` objects. You can refer to exercise01.cpp if you want to copy statements from the lambda expressions in that file.

   *NOTE: A solution is provided in a separate solution02 project if you want to refer to it.*
3. When your modifications are complete, build and run the exercise to generate an output.png file. Note the execution time. The execution time should similar to exercise 01.
4. Inspect again the output image to confirm that the output has not changed.

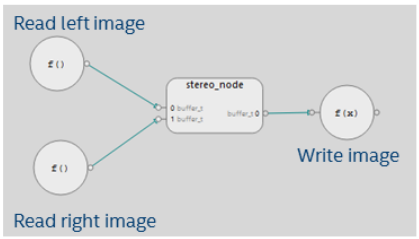
## Exercise 3: Hello OpenCL*



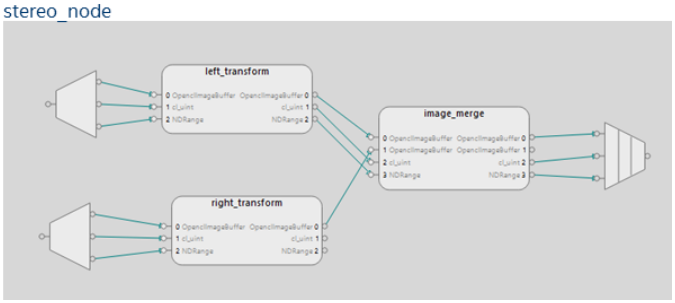

1. Go to the exercise3 directory
2. Open exercise03.cpp. Note the opencl_program and opencl_node in the source code. These refer to the OpenCL program file hello_world.cl and the print kernel found in that file, respectively.
3. As written, the graph in exercise03.cpp has an empty tokens buffer so no nodes will execute. Before the g.wait_for_all() in the main function, add calls to put "tokens" in to the tokens buffer to start the execution of the graph. A token of "0" will cause the dispatcher to invoke the cpu_node on the CPU and a token value of 1 will cause the dispatcher to invoke the OpenCL node on the integrated graphics. A completed solution is provided in solution03.cpp.

   ```
   token_buffer.try_put(0);
   token_buffer.try_put(1);
   ```
4. When your modifications are complete, build and run the exercise. Notice that now the OpenCL library must also be linked in to the executable
5. If you wish, you can add additional tokens to the token buffer to force additional executions on each device.

## Exercise 4: Execute the OpenCL* stereoscopic 3D example

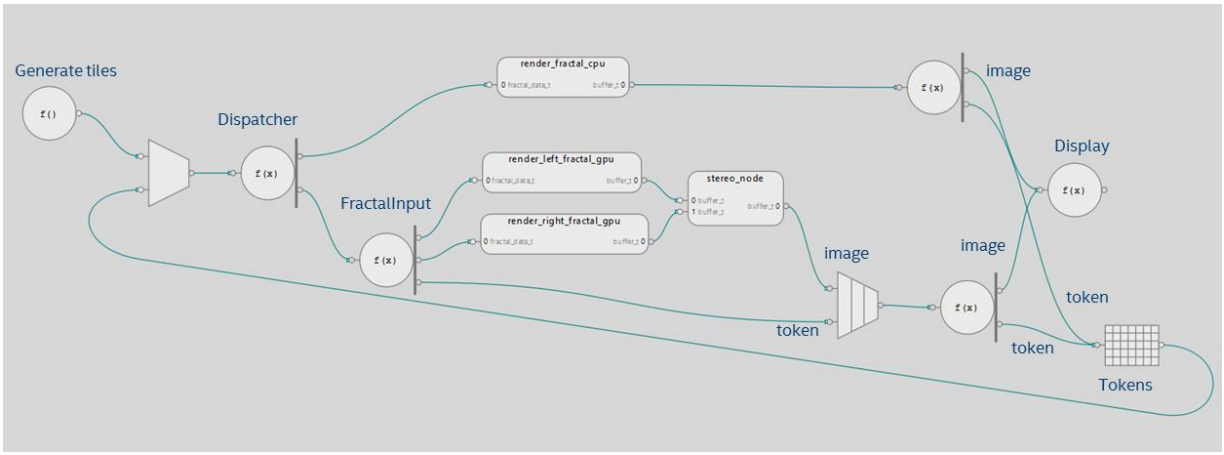



stereo_node




1. Open the exercise4 project
2. Open exercise04.cpp. The topology of the graph built in the main function is the same as in exercise02.cpp, but now the stereo effect is implemented using OpenCL*.
3. Build and run the exercise. Note the execution time. The execution time should be smaller (on average) than the previous implementations in exercise 0, 1 and 2.

## Exercise 5: Run a Stereoscopic 3D Fractal Generator that uses Tokens




1. Open to the exercise5 project
2. Open exercise05.cpp. This graph plugs the OpenCL* implementation of stereo in to the token-passing verison of Fractal shown in the slides.
3. Build and run the exercise.
4. The final output image, found in output3.png, will show the four tiles of fractal. The tiles executed on the CPU will be in color and with no stereoscopic 3D effect applied, while the tiles executed on the integrated graphics will be grayscale with the stereoscopic 3D effect applied.

## TBB installation in Linux

Open a terminal window (ctrl-alt-t), and follow these steps:

```
1.- git clone https://github.com/01org/tbb.git
2.- cd tbb
3.- git checkout tbb_2018
4.- make tbb stdver=c++11 tbb_cpf=1
5.- source build/linux_intel64_gcc_*****_kernel***_preview_release/tbbvars.sh
6.- cd examples
```

Now to compile, for example, exercise5, do:

- `cd exercise5`
- `g++ -o sol solution05_final.cpp ../common/lodepng.cpp -std=c++11 -lOpenCL -ltbb_preview`

## TBB installation on Mac OS

1. First install gcc on your Mac
    a. Install homebrew: ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/master/install)"
    b. brew install homebrew/versions/gcc6
2. Now install tbb:
    a. git clone https://github.com/01org/tbb.git
    b. `cd tbb`
    c. `git checkout tbb_2018`
    d. cd build
    e. cp macos.gcc.inc macos.gcc6.inc
    f. vi macos.gcc6.inc, and change this:
        i. CPLUS = g++-6
        ii. CONLY = gcc-6
    g. cd ..
    h. make tbb compiler=gcc6 stdver=c++11 tbb_cpf=1
3. Use g++-6 to compile the exercises and –framework OpenCL when OpenCL is needed.



| **Optimization Notice** |
| --- |
| Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2®, SSE3, |

and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice.

Notice revision #20110804