C++ AMP Conformance Test Suite Readme

1. Introduction

C++ AMP is a C++ language extension with an accompanying library enabling offloading data-parallel algorithms to hardware accelerators, such as GPUs. C++ AMP open specification (called *open* spec later in this document) is available free of charge [1]. C++ AMP open spec conformance test suite is a set of functional tests validating C++ AMP implementation against the specification. This set of tests is in no mean exhaustive nor constitutes any certification.

2. License

3. Tests hierarchy

Most of the tests in the suite are referring to a particular section of the open spec and are organized accordingly. E.g. tests under path *2_Cxx_Language_Extensions\2_1_Syntax\2_1_3_Type_Specifiers* are targeting primarily open spec section *2.1.3 Type Specifiers*.

One notable exception is the set of tests under path *2_Cxx_Language_Extensions\2_x_general* which consist of tests targeted at general C++ constructs used in the C++ AMP restricted context, not described explicitly in the open spec.

4. Test case structure

Each test case consists of a single *test.cpp* file, possibly including multiple header files. Most of the tests are statically linked with the test library (described below).

Each test case source file is preceded with meta-information.

Tags

Tags describing the test case which may be used for automated processing.

```
/// <tags>tag[,tag[...]]</tags>
```

Summary

Descriptive summary of the test case. May contain line breaks.

```
/// <summary>Text</summary>
```

Expects

Marks negative tests. Handling of Expects lines is described in the Test runner section. Expects lines may be placed anywhere in the file.

```
//#Expects: Error: ...
```

5. Test library

The C++ AMP Conformance Test Suite contains a small library of common test helper functions and device management.

Building the Library

A nmake makefile has been included for building the test library with the Microsoft(R) Visual C++(R) compiler (MSVC).   The file is straight forward and should be easy to translate to other build systems or platforms. Let <amp_test_lib> be the full path to the amp_test_lib directory.

- set INCLUDE=%INCLUDE%;<amp_test_lib>/inc
- cd <amp_test_lib>
- nmake /f nmakefile

By default, the library will be built to <amp_test_lib>\lib\<arch>\ libamptest-MTd.lib.

/MT, /MD, or /MDd can be specified using the LIBTYPE parameter. libamptest-MT.lib can be built as follows:

- nmake /f nmakefile LIBTYPE=MT

Directory Structure

The test library is contained within the amp_test_lib directory and has the following structure.

amp_test_lib

- nmakefile : Makefile for use with nmake.

amp_test_lib/inc

This directory should be added as an include path. All tests and test library will specify include files relative to this path.

File Name	Description
amptest_minimal.h	Includes a minimal set of test library header files
amptest.h	Includes amptest_minimal.h and additional test library header files
dpctest.h	Alias for amptest.h
amptest_main.h	Defines a main() function providing exception handling capabilities for runtime tests calling the user-defined test method with the following function signature: int amptest_main(amptest_context_t& context);

amp_test_lib/inc/amptest

This directory contains header files for C++ AMP conformance test library and other utility functions. Files in this directory will be included using a relative path from the inc directory. For example:

#include <amptest/restrict.h>

amp_test_lib/inc/amptest/msvc

Header files specific to usage of the MSVC compiler.

amp_test_lib/src

This directory contains C++ AMP test library implementation files.

6. Test runner

Test runner is a simple command line utility responsible for compiling and running tests, interpreting results and providing report.

Test runner is implemented as a Perl program and located in the conformance suite root directory (file *run_tests.pl*). It requires Perl runtime >= 5.6.0 to be installed.

Configuration

The program has to be configured for a particular compiler and optionally for the environment. The persistent configuration is embedded in the *run_tests.pl* itself, and additional options are read from environmental variables.

Compiler command

Compiler command read from the environmental variable “CC” is used.

Compiler flags

Compiler flags set in the environmental variable “CFLAGS” are used. Note that they might be supplemented with after-mentioned compile only and preprocessor definition flags.

Compile only flag

```
my $cflag_compile_only = '/c';
```

Defines a compiler flag used to compile the test source file without linking, used as in the following command:

```
system(“$CC $cflag_compile_only test.cpp”);
```

Preprocessor definition flag

```
my $cflag_define = '/D%s#"%s"';
```

Defines syntax of a compiler flag used to enforce a particular C preprocessor definition. Multiple definitions must be allowed with simple concatenation. Used as in the following snippet:

```
$actual_flag_1 = sprint($cflag_define, 'name', 'value, possibly with whitespaces');
$actual_flag_2 = sprint($cflag_define, 'name_2', ''); # empty definition
system(“$CC $actual_flag_1 $actual_flag_2 test.cpp”);
```

Test executable

```
my $test_exec = 'test.exe';
```

Name of the resulting runnable test executable, used as in the following snippet:

```
system(“$CC test.cpp”);
system(“$test_exec”);
```

Test configuration files

Test case may optionally provide a configuration file *test.conf*, located in the same directory as the test case source file.

Supported parameters:

- compile_only – value: 0/1, 0 is default – when set to 1 the test case will be only compiled (i.e. compile only flag will be used) and the runner will not try to execute it.
- definitions – set of key -> value mapping sets, empty is default – when non-empty the test case will be separately compiled and run (unless it is compile only) for every element of the set with each key-value pair used as preprocessor definition name and value; when empty the test case will be compiled and run once without additional preprocessor definitions.

For example:

```
%config = (
   'compile_only' => 1,
   'definitions' => [
           { 'AMP_RESTRICTION' => 'amp' },
           { 'AMP_RESTRICTION' => 'cpu,amp', 'EXTRA' => '' }
   ]
);
```

Using this configuration file, the test case will be compiled twice with different set of definitions and not executed in either case. If the settings where entered for MSVC compiler, it would result in two commands:

```
cl.exe /c /DAMP_RESTRICTION#”amp” test.cpp
cl.exe /c /DAMP_RESTRICTION#”cpu,amp” /DEXTRA#”” test.cpp
```

Expected return codes

The compiler command must return 0 error code in the case of success, any other error code is interpreted as compilation failure.

The test case executable must return one of the following exit codes:

- 0 – successful execution;
- 2 – execution skipped (e.g. required accelerator type was not available);

- any other – execution failure.

Negative tests

The test case is regarded negative whenever an “Expects” line (as described in the Test structure section) is found within the source file. Negative test cases are expected to result in a compilation error, therefore only non-zero compiler error code is treated as success.

Note: the actual text in an “Expects” line is currently ignored, any compilation error is treated as success, you may want to fine tune this mechanism for your compiler.

7. Bibliography

[1] Microsoft, "C++ AMP open spec published," May 2012. [Online]. Available: http://blogs.msdn.com/b/nativeconcurrency/archive/2012/02/03/c-amp-open-spec-published.aspx. [Accessed May 2012].