ClearSpeed Visual Profiler

CSAPI – Example 2

Understanding CSAPI trace information when profiling multi-threaded code

Advance
Accelerate
Optimize

ClearSpeed Visual Profiler

CSAPI – Example 2

Introduction

In this second example for the CSAPI profiling we will look at analysing multi threaded code which interacts with the CSX700 processor board. In modern multi core x86 systems multiple threads are often executed on multiple processors by the operating system allowing operations to occur in parallel. This adds an extra degree of complexity to developing applications as the code needs to be written to take account of this fact in order to ensure correct operation and performance. We will see in this example how the visual profiler in conjunction with the CSAPI profiling allows a user to visually see the operation of their multi threaded code and understand its operation. The example used is similar to the first example except for the fact that the memory transfers now operate in their own threads. This makes the execution of the application more complex as there are now multiple threads attempting in parallel to access a serial resource by performing memory transfers via the PCI bus to the card. We will see from the profiler how this is visible and how different multi threaded code looks to the previous CSAPI example.

Building the example

The source file for this simple CSAPI example can be found in the directory, it is recommended to copy the source for this example into a user directory before building and running so as not to create unnecessary files in the installation directory.

```
csvprof_examples/csapi/example2
```

This example needs to be compiled and linked against the CSAPI host library. Below are the command lines for both windows and linux depending on which platform you are running the example on. You could build the example with other tools but the workshop has only been tested with those described below.

Windows

The windows example needs to be build with the Visual Studio tools and the cl compiler. The command line is as follows and is all one line.

```
cl /I"%CSHOME%\include\cs_api" example.c
“%CSHOME%\lib\cleard_stub_lib.lib" /MT
```

Linux

Advance
Accelerate
Optimize

The linux example is build in this instance using gcc. The command line is as follows and is all one line.

```
gcc -o example -I$CSHOME/include/cs_api example.c
$CSHOME/lib/libcleard_stub_lib.a -ldl -lpthread
```

Running the example

Once the example executable is build you can run it to see the output. If running on simulator you need a –s option passed on the command line. The simple CSAPI code transfers a series of increasing sized memory buffers to the board and read them back. This allows us to view a simple example of the CSAPI profiling. In order to use the profiling mechanism built into the ClearSpeed runtime you need to set the environment variables described in the user guide. These variables need to be set to have the following values in your native command prompt shell.

Windows

On windows the environment variables are set as follows in a standard command prompt.

```
set CS_CSAPI_TRACE=1
set CS_CSAPI_TRACE_CSVPROF=1
set CS_CSAPI_TRACE_CSVPROF_FILE=csapi_example2.cst
```

Linux

On a linux system running a bash shell the variables are set as follows.

```
export CS_CSAPI_TRACE=1
export CS_CSAPI_TRACE_CSVPROF=1
export CS_CSAPI_TRACE_CSVPROF_FILE=csapi_example2.cst
```

You can now run the `example -t` executable, it should output some text regarding the memory transfers and then exit. The –t switch makes the code run with multiple threads.

Once this has completed you should be left with a trace file called `csapi_example2.cst`. This file contains the profiling information for the CSAPI example that has just been run.

1. Starting the csvprof GUI

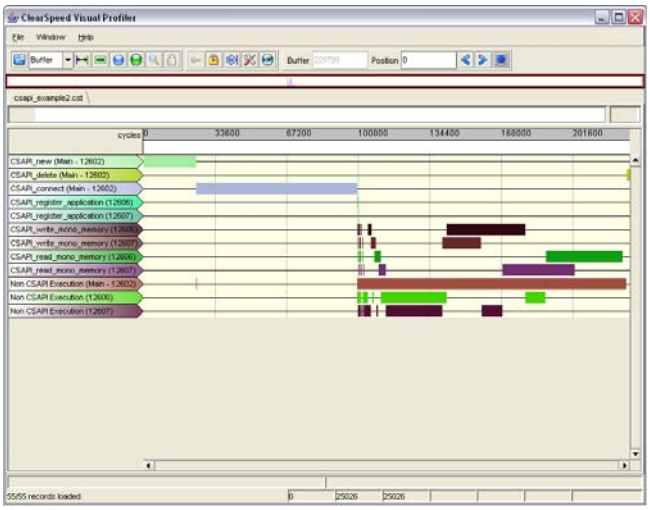
The csvprof GUI is started from the command line using the following command. The example trace file is passed on the command line. You can also start up the GUI and navigate to the current working directory and load the file via the file menu.

```
csvprof csapi_example2.cst
```


Advance
Accelerate
Optimize

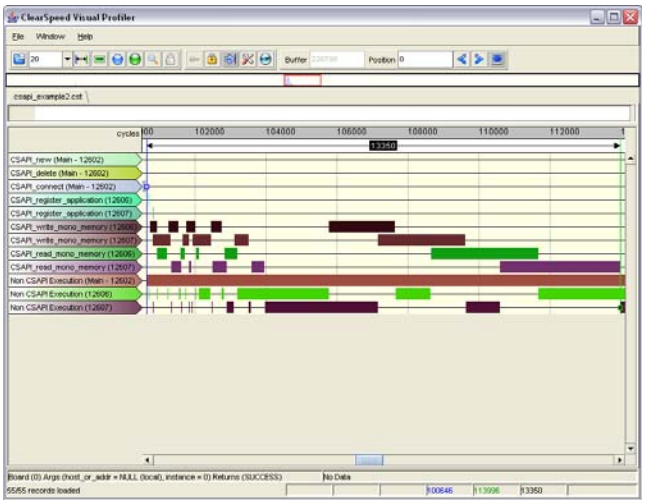
 # Start up the visual profiler with the above command line.

The GUI should start up and appear as below.



The visual profiler has started up and you can see the events that have been traced during this session listed down the left hand side of the GUI. In this example we have 2 extra threads that have been started up during execution of the code. There are 3 threads in total which are visible, the main thread which as performed the house keeping functions to initialise the CSAPI and 2 extra threads which are each performing the same set of memory transfers from the first example. The trace should look as above on start up but due to differences in number of processor cores and operating systems you may have a slightly different view.

 # Select the snap cursors button from the control panel, select the end of the CSAPI_connect event with the left button, the start of the longest visible Non CSAPI Execution event on the bottom event and then press fit to markers.



Advance
Accelerate
Optimize

The view has now zoomed in to encompass the first 5 memory transfers to and from the board. By moving the mouse over the individual events you can get more information regarding them. It is clear from this example that there are more memory transfers in operation due to the 2 threads. When moving the mouse over the different threaded memory transfers you can see that one thread is writing and reading from CSX processor 0 and the other from processor 1.

There are several things to note from this simple example.

1. As this example is multi-threaded you can see there is a main thread and 2 memory transfer threads active during the code execution from the events listed in the visual profiler. The main thread is marked as such and the others simply have a different thread id associated.
2. In this example as it had been run on a multi core machine some of the events overlap as there are sections of code which can operate independently.
3. Moving the mouse over the individual memory transfer events you can see that they are increasing in size the amount of data transferred to and from the board. In this example you can also see that each thread is transferring data to a different memory location.
4. When the profile is showing “Non CSAPI Execution” it means that the time is spent in code outside of the CSAPI. In this example that will be the code to malloc and set up the buffer to be transferred to the card and to handle the checking of the results when data is read back. There are 3 of these events visible in this profile, one for each thread active during execution.
5. The tooltips for the CSAPI events gives you all the arguments passed to the particular function and the return value. This makes it visible when incorrect values are passed or the function fails.
6. The PCI bus in the machine is a point of serialization and as such if 2 processors attempt to access board memory at the same time the will access it serially. This is visible in the trace in the fact that although the start of a memory transfer event can overlap another no 2 events finish at exactly the same time. You can see events where they have taken longer to complete due to the fact that another thread has access to the board.

Feel free to experiment more with the GUI and the data present in this example. Once finished exit the GUI with the exit option from the file menu.

Conclusion

We have had a look in this example at an application that communicates with the CSX700 processor board and uses multiple threads. The features of the visual profiler for displaying multiple threads have been demonstrated and you should now be able to move onto profiling more complicated threaded applications using the CSAPI. The example demonstrated is fairly simple but should have given you enough of an insight into using the visual profiler for CSAPI applications that you can now use it for analysing your own applications.