ClearSpeed Visual Profiler

CSAPI - Example 1

Viewing a simple trace containing CSAPI function profiling information

Advance
Accelerate
Optimize

ClearSpeed Visual Profiler

CSAPI - Example 1

Introduction

This is the first example in the set of examples demonstrating the use of the visual profiler for investigating information relating to the use of the CSAPI when communicating between a host system and a CSX processor card. The ClearSpeed host runtime contains a built-in profiling mode which can be enabled simply and will output a file in the format required for use with the visual profiler. Each time a CSAPI call is made the results of the call are captured and time stamped to allow a post execution review of the performance using csvprof. The example assumes that the user is familiar with the use of csvprof from the GUI overview workshop. The CSAPI examples assume that a user had the ClearSpeed runtime installed and either a simulator or CSX700 based accelerator board installed. Without these the example will not run.

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/example1
```

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

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
```


Advance
Accelerate
Optimize

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_example1.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_example1.cst
```

You can now run the `example` executable, it should output some text regarding the memory transfers and then exit.

Once this has completed you should be left with a trace file called `csapi_example1.cst`. This file contains the profiling information for the CSAPI example that has just been run.

1. Starting the csvprof GUI

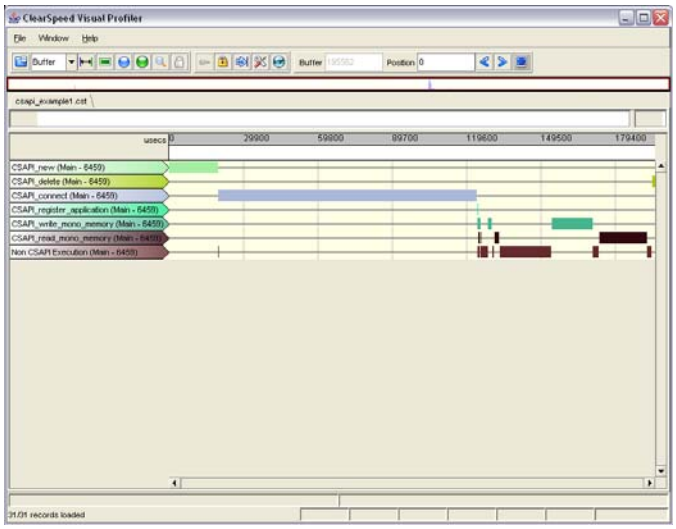
The csvprof GUI is started from the command line using the following command. The example trace file is passed on the command line. You can also start up the GUI and navigate to the current working directory and load the file via the file menu.

```
csvprof csapi_example1.cst
```

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

The GUI should start up and appear as below.

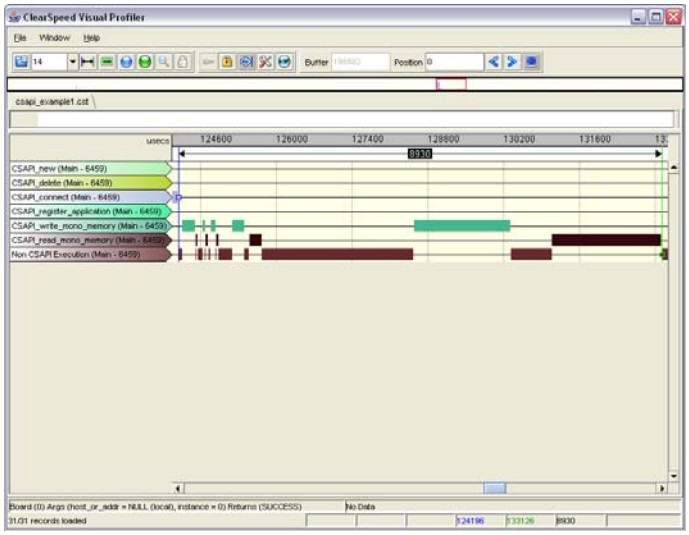
Advance
Accelerate
Optimize



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. As we are now profiling the CSAPI what we see are a list of all the CSAPI calls made during the execution run and an event for time spent outside of the CSAPI. This allows a visual representation of the flow of code execution to be displayed. The scale for CSAPI profiling traces is in microseconds and the minimum an event can be is one microsecond.

In this example you can see the CSAPI calls that have been made. The interesting ones for this example are the read and write memory calls. We can now look at getting more information about these operations.

 # 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 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. The tooltips which are displayed give information regarding the parameters passed to each CSAPI function.

There are several things to note from this simple example.

1. This example is single threaded, the events listed down the side of the trace all contain the thread id and in this example then are all the same number. The very first thread seen is marked as the main thread in a CSAPI application, this is also visible from the event label.
2. Everything is completely synchronous as everything is running on a single processor core there is no overlap of any executing code.
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.
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.
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.

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

In this example you have seen how a simple trace of CSAPI calls allows you to see the profile of how the host system is interacting with the CSX700 processor card. We have looked at a single threaded example which just performs simple memory transfers to and from the board. This method of profiling the CSAPI can be used with any application which is using the board no matter how complicated and allows a user much more insight into the operation of software between the host and the CSX700 based processor card. You should now be familiar with enabling the CSAPI trace and analysing the file in the visual profiler GUI.