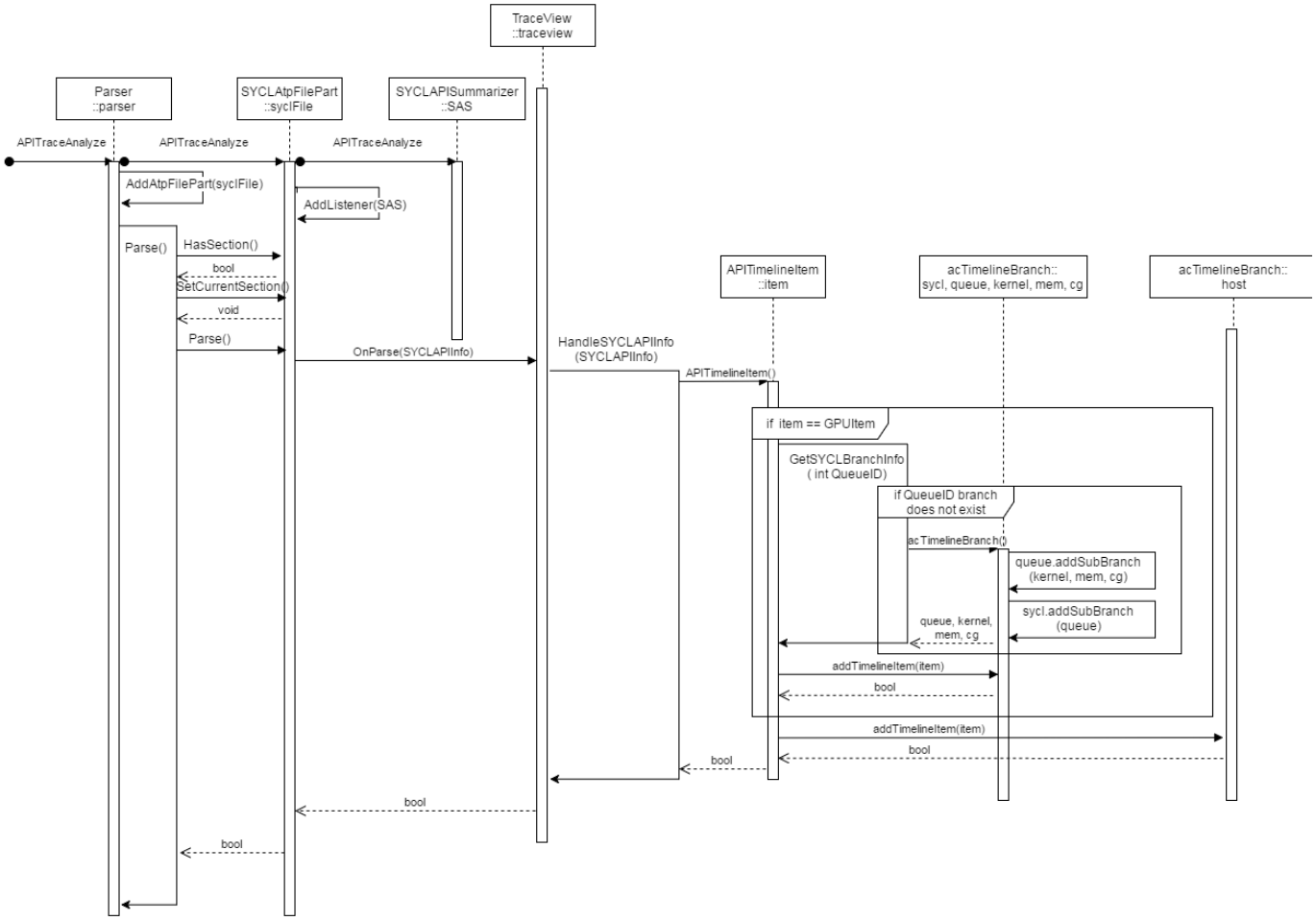
CodeXL - extending traceview for the SYCL backend

Intoduction

This page should give an explanation of how we modified CodeXL to parse the SYCL profiling information that our SYCL profiling backend to CodeXL creates. The following UML sequence diagram give a brief overview of how the program goes from creating a parser to adding an already created SYCLAPIInfo to the trace view in the GUI. The entry point is the APITraceAnalyze function, which is called by the CodeXLGpuProfiler.exe main.cpp (in the project CodeXLGpuProfiler).



Parsing the Atp File

First a AtpFileParser, SYCLAtpFilePart and SYCLAPISummarizer object are created.

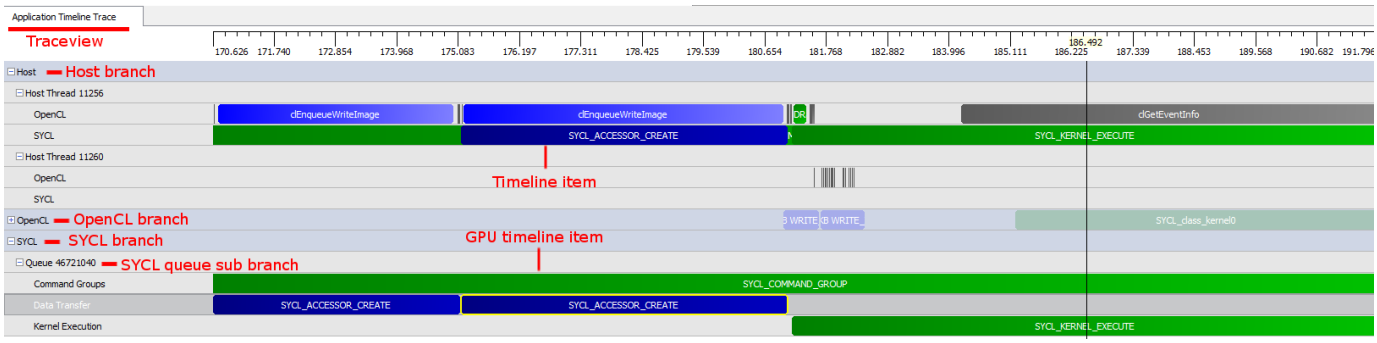
The SYCLAtpFilePart class contains functions for parsing the SYCL part of the Atp file and for creating the SYCL temporary Atp file. CodeXL has all backends create a temporary Atp File before merging them altogether into a single Atp file and parsing that using all backend parsers. It is therefore important to set a header line either when you create or merge the Atp file such as "=====SYCL timestamp output=====" so the parser knows which backend parser to use for each section of the Atp file.

The SYCLAPISummarizer is just a typedef for class APISummarizer<SYCLAPIInfo> defined in APISummarizer.h.

To parse the Atp file we must:

1. instantiate and add the SYCLAtpFilePart to the AtpFileParser using AddAtpFilePart(syclFile)
2. add SYCLAPISummarizer as a listener to the SYCLAtpFilePart using AddListener(SAS)
3. invoke SYCLAtpFilePart::Parse() to parse the Atp file through AtpFileParser::Parse()

    a. this checks if the SYCLAtpFilePart has a section name the same as the one AtpFileParser::Parse() has just parsed
    b. if so, this is set as the current section and the file is parsed
4. SYCLAtpFilePart::Parse() creates a SYCLAPIInfo from each line of the relevant section of the Atp file and feeds it to the TraceView via OnParse()



Creating TraceView items

Once a SYCLAPIInfo object has been created by the parser, TraceView must create a APITimelineItem for it and if it has not already been done, the appropriate acTimelineBranch for this item. We have decided to structure our timeline branches such that each Queue have its own branch, which in turn has 3 child branches, the kernel, memory and command group branches (kernel, meme & cg in the diagram above). Only relevant timeline items will be on each branch, with all items appearing on their host thread branches.

A non-GPU item is simple: HandleSYCLAPIInfo is called by OnParse and creates a new APITimeline item and this item is added to the timeline via addTimelineItem(item).

For a GPU item:

1. HandleSYCLAPIInfo is called by OnParse and creates a new APITimeline item
2. GetSYCLBranchInfo is called to retrieve the relevant queue branches to add the item to
    a. if the branches doesn't exist, we create new acTimelineBranchs for kernel, mem, cg and queue
    b. add mem, cg and kernel as subbranches of queue
    c. add queue as a subbranch of the main SYCL traceview branch
3. add the item to its relevant queue branch and subbranch via addTimelineItem(item)
4. add the item to its host branch