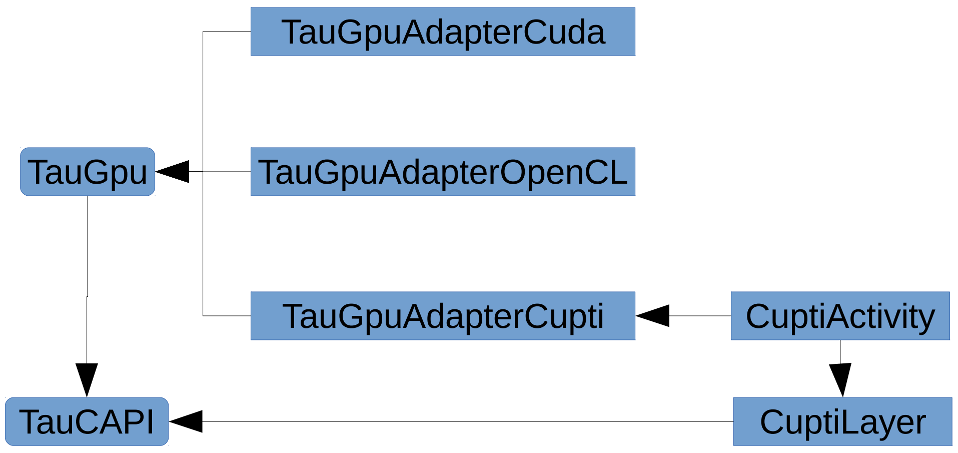
TAU GPU architecture diagram



TauGpu(.h/.cpp): Backbone of the GPU measurement system. Uses TAU 'fake' tasks and a user clock to record events taking place on a GPU after they have already occurred.

Adapter(.h/.cpp): Programing model specifics, each adapter implements a subclass of GpuEvent defined in TauGpu.h.

CuptiAcitivty(.h/.cpp): Component directly in contact with CUPTI, must be compiled with GCC and use only C interface functions to the rest of the TAU library.

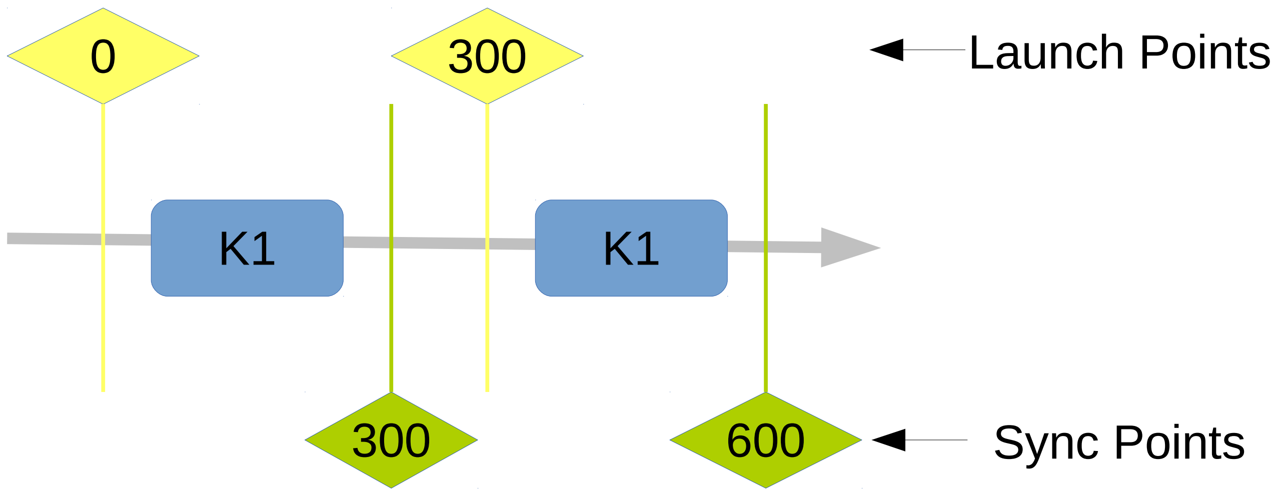
CuptiLayer(.h/.cpp): Based on the approach taking by PAPI, allows to the querying and recording of CUDA counters.

CUPTI counter scheme:

There is a lot of particulars/limitations to recording counters and assigning their value to a particular kernels. 1. counter values are only recorded at kernel launch and kernel synchronization. 2. counter values are device-wide and could the result of any particular kernel running. Here are three potential scenarios and how the counters are assigned to kernels

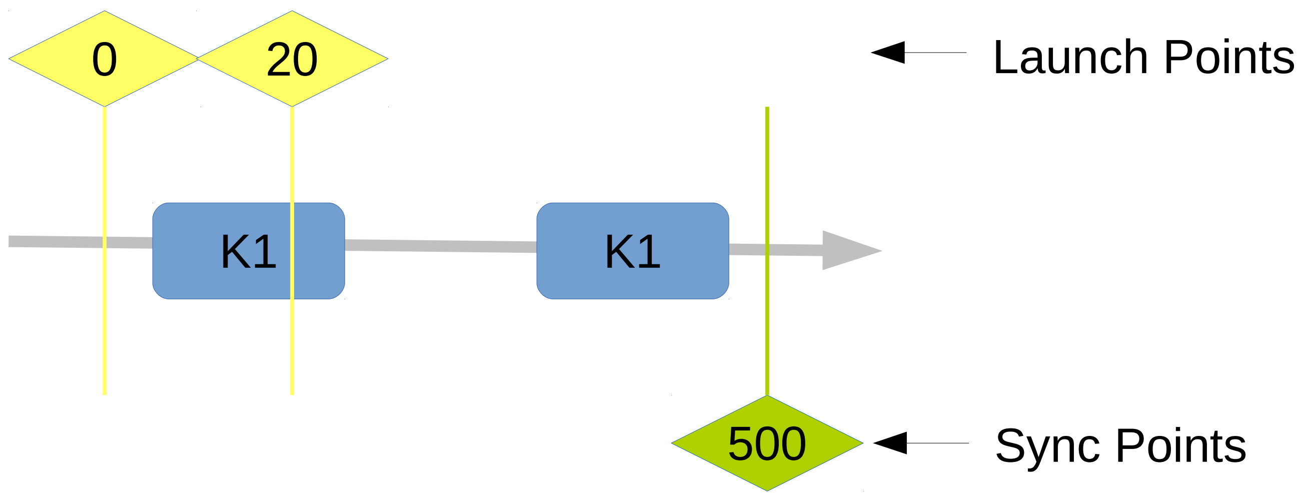
One important point to keep in mind is the performance results are provided on a per kernel basis as the best guess for what the counter value would be for a single run of that kernel. For example if we record for given interval (which contains 10 kernels of the same name) 200,000 instructions executed we will report the inclusive/exclusive instructions as 20,000 (averaged).

The first scenario has two kernels of the same type with run synchronously and the values recorded are 0 (first launch), 300 (first sync), 300 (second launch), and 600 (second sync) the kernel will get an value of 300 (2 calls).



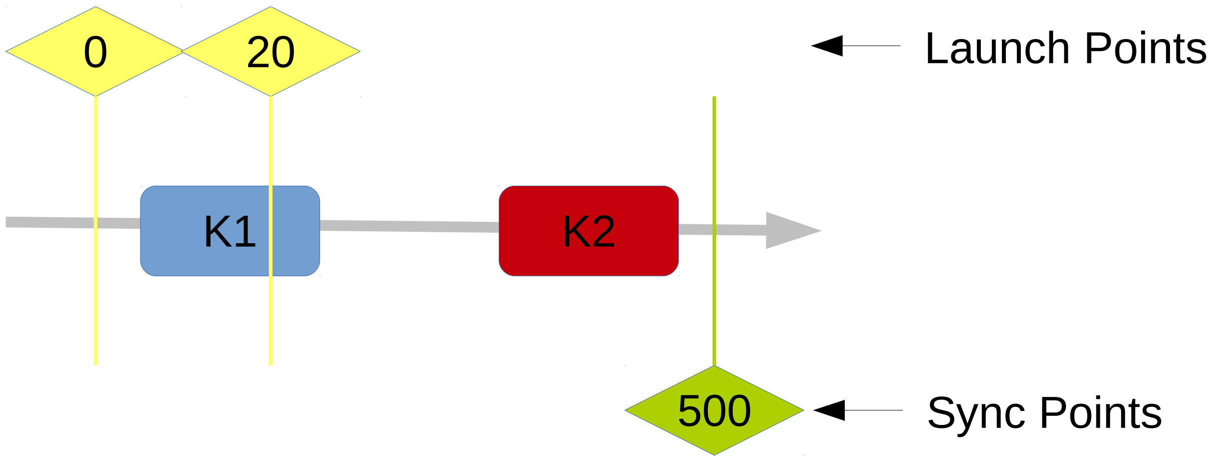
Calls	Count	
2	300	K1

The second scenario has two kernel launched one after the other and a synchronization point for both of them later. We have no way to determine which kernel accounts for how much of the recorded GPU counter so we just provided the averaged result. To obtain the total one must divide the inclusive/exclusive time from the reported averaged because in theory either kernel could account for the reported counts.



Calls	Count	
2	500	K1 (averaged)

Thirdly, we could have the same situation with multiple kernels and once again any of the kernels could account for the recorded counter values.



Calls	Count	
1	500	K1 (averaged) (upper bound)
1	500	K2 (averaged) (upper bound)