Assumptions & Needs for Interoperability of Tools, Middleware, and Vendor libraries

CUDA contexts.

- A profiling library needs the CUDA context that the application layer has created (if any).
- This can be done in different ways:
  - Use the subscriber callback mechanism. But only one tool/library can use this, so we can't have both PAPI and a tool using it subscribe a callback.
  - The tool could initialize the middleware library NOT as early as possible, but as late as possible. Why create eventsets before the application creates their contexts?
  - Tools could offer their own callback subscriber so we can create "chains of listeners".
    - This could also allow exchange of information about other things, say utility thread creation.
  - Vendor libraries could allow multiple subscribers, not just one, but this requires the vendors to make changes in their libraries.
    - Performance overhead of doing this is not really critical since cupti operations are heavy weight as it is.
- Regarding threads, John MC had a proposal for this last year that involved elf-notes to inform the tool about the nature of a newly created thread.
  - Not yet implemented by vendor libraries.
  - AMD is considering it.

CUDA contexts.

- A profiling library needs the CUDA context that the application layer has created (if any).
- This can be done in different ways:
  - Use the subscriber callback mechanism. But only one tool/library can use this, so we can't have both PAPI and a tool using it subscribe a callback.
  - The tool could initialize the middleware library NOT as early as possible, but as late as possible. Why create eventsets before the application creates their contexts?
  - Tools could offer their own callback subscriber so we can create "chains of listeners".
    - This could also allow exchange of information about other things, say utility thread creation.
  - Vendor libraries could allow multiple subscribers, not just one, but this requires the vendors to make changes in their libraries.
    - Performance overhead of doing this is not really critical since cupti operations are heavy weight as it is.
- Regarding threads, John MC had a proposal for this last year that involved elf-notes to inform the tool about the nature of a newly created thread.
  - Not yet implemented by vendor libraries.
  - AMD is considering it.

Semantics of software counters

- Hardware counters are always long long int, but now PAPI allows SDE with arbitrary data.
- How do we communicate the type to a tool?
  - For each counter sde:::XYZ, PAPI can provide a sde:::XYZ:type
- Which thread is counting? If it's in the thread private data of a library code then it can't be read by arbitrary tool threads.
  - Open problem.

Hardware Counter Semantics

- More than type might be needed:
  - Socket-based counter vs thread-based needed to understand the meaning.
- Multiple GPUs with multiple threads: Tools (Score-P) want to know which device it came from
  - PAPI ZYX:device=1
  - Liwkid reads all devices
- Likwid has compatibility problems with perf because they bypass perf and use the registers directly, but perf ignores them and overwrites the registers anyway.
- Intel has a PMU sharing guide, but perf ignores it.
  - Creating noise through a tools-wide white paper could help.

Multi-pass counters/metrics

- PAPI, Likwid, Caliper, Score-P, and MAQAO don’t count such counters.
- HPCToolkit does the replaying themselves.
- If PAPI did it under them, they would break
- An assumption could be that if a middleware layer offers reading multipass counters/metrics then that layer does the replaying.
  - This has as a side-effect that timing of regions that include this reading becomes inaccurate, but this could be a known limitation.

GPU counter granularity

- Can we get fine-grain counter information when running multiple kernels?
  - We either get aggregate information, or kernels get serialized.
- On AMD, this would require one counter per wavefront per CU. This would make the hardware too complex.
- With PC samples, AMD try to correlate back to the dispatch. NVIDIA doesn't seem to do this.
- **Idea for vendors**: we would like an additional mode, similar to kernel mode, but allowing multiple kernels to execute in the GPU, but no two kernels sharing a CU. This way, if we can read per CU counters and we are given the CUs in which a kernel ran on, we can figure out kernel accurate counters, without limiting the parallelism.
- Hardware vendors move their development toward larger industries like gaming, so needs of tools are not as influential. Yet, the game devs clearly need these kinds of capabilities. We should interact with game developers to see how (if?) they are doing performance analysis.

Miscellaneous

- If a tool calls roctx before initializing rocm, it leads to a crash AND there is no way to tell if rocm has been initialized.
  - The tool could call HSAInit internally, but roctx shouldn’t crash anyway.
- As soon as you use CUPTI, energy aware and frequency optimizations are disabled. NVIDIA switches to a default frequency.
- NVIDIA has been changing the API and dropping functions between minor versions of CUPTI (11.*)
  - A white paper could include “best practices”, since they don’t seem to be obvious.