Commands which can be scheduled in parallel should/may run faster than the same commands serialized

A command is a request to execute work that is submitted to a queue such as the invocation of a SYCL kernel function, the invocation of a host task or an asynchronous copy (SYCL 2020-Specification, Glossary)

Overlaps is and optimization and is not required by the specification.

Which Commands can run Concurrently?

1. Multiple Low occupancy kernels
   • In this paper we use a traditional "dual" kernel pair of PMA on one (C) scheduled to use one work-item
2. Host to Device data-transfer + Device to Host data-transfer
   • In this paper data-transfer occur between host memory (either allocated via malloc(), or pinned memory via sycl::malloc_host()) and Device Memory via sycl::malloc_device().
   • e.g. H2D, D2H...
3. Low occupancy kernel + Host to/from Device data-transfer

This poster aims to report on conditions different SYCL runtimes achieve concurrency. Explanation of the results are out of the scope of this poster but we are more than happy to discuss low-level details.

How to Achieve Concurrency?

The SYCL specification allow concurrency execution in two major scenarios:

• Out Of Order Queue Multiple SYCL commands submitted in a out of order queue can be executed concurrently("OpenCL way")
  - sycl::queue q(sycl::gpu_selector());
  - for (auto command : commands)
  - do_work(q,command);
  - Q wait();

• Multiple In Order Queues. Multiple SYCL commands can be submitted in a multiple in-order queues. Each queue can be executed concurrently ("CUDA way")
  - const sycl::device D(sycl::gpu_selector());
  - const sycl::context (CD);
  - const sycl::queue Q,
  - for (auto & command) :
  - Q.push_back(sycl::queue(C, D, sycl::property::queue::in_order()));
  - for (int i = 0; i < commands.size(); i++)
  - do_work(Q[i],command[i]);
  - Q wait();

Result Color Code

Concurrent Execution
Concurrent Execution but Serial Execution when using Profiling enable Queues
Serial Execution

Table 1. Color Scheme Legend

<table>
<thead>
<tr>
<th>SYCL Platform</th>
<th>Lowoccupancykernel</th>
<th>Outoforder &amp;</th>
<th>MultipleLowoccupancykernels</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intel GPU</td>
<td>C</td>
<td>D</td>
<td></td>
</tr>
<tr>
<td>NVIDIA</td>
<td>C</td>
<td>D</td>
<td></td>
</tr>
<tr>
<td>AMD</td>
<td>C</td>
<td>D</td>
<td></td>
</tr>
</tbody>
</table>

Compiler & Drivers Versions Used

<table>
<thead>
<tr>
<th>ID</th>
<th>Compiler Name</th>
<th>Commit</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>DPCOM</td>
<td>2022.1.0</td>
</tr>
<tr>
<td>2</td>
<td>Intel/LVLM</td>
<td>03F41F</td>
</tr>
<tr>
<td>3</td>
<td>Intel/LVLM</td>
<td>1F65EA</td>
</tr>
<tr>
<td>4</td>
<td>hipSYCL</td>
<td>258b97</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ID</th>
<th>Drivers Name</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>ComputeRuntime</td>
<td>21.40.21.182</td>
</tr>
<tr>
<td>2</td>
<td>CUDA</td>
<td>11.0.2</td>
</tr>
<tr>
<td>3</td>
<td>RCDM</td>
<td>4.5.2</td>
</tr>
</tbody>
</table>

Methodology

We run each step N times, and take the minimum time to reduce measurement noise

1. Run commands lists serially. Compute the maximum Theoretical Speedup
   max_command_time/total_time_serial

2. Commands parameters (number of PMA, size of the data transfer) are auto-tuned so that each command take roughly the same time
3. By default memory transfers are "as big as possible" (e.g. g.get_info<device_max_mem_alloc_size>(1))
4. Run the list of commands in a mode who allow concurrency. Compute Speedup
   total_time_concurrent/total_time_serial
5. Verify that Theoretical speedup and Empirical speedup roughly match

Tool Output Example

- $/sycl_con out_of_order C D2H
  - Performing Autotuning
  - Parameters tuned:
    - tripocount: 466765
    - globalizing_D2H: 1073741824
    - Best Total Time Serial: 412971us
    - Best Time Command 0 (C): 303298us
    - Best Time Command 1 (D): 306973us
    - Maximum Theoretical Speedup: 4.0
    - Best Total Time //: 355451us
    - Speedup Relative to Serial: 1.2789
    - SUCCESS: Close from Theoretical Speedup

Summary

1. We developed a empirical concurrency testing framework. People are encouraged to use it! See url in the bottom right of the poster
2. Using "pinned memory", sycl::malloc_host() may be required by drivers for concurrency
3. On Intel Platform, L0 need a little more love
4. On NVIDIA and AMD Platform, hipSYCL delivers most reliable overlap of operations. Intel/LVLM has a hard time with out-of-order queue
5. Discussion on-going for Intel/LVLM CUDA/HIP back-end to implement an M:N mapping between streams and sycl queues. This should help out-of-order queues performance.