SYCL Academy

# DEBUGGING AND PROFILING



codeplay

# LEARNING OBJECTIVES

- Learn some tips to help debug in SYCL.
- Learn how to profile SYCL code for CUDA backend.
- Learn about coalesced global memory access.
- Learn some optimization tips.

# SYCL EXCEPTIONS

- In SYCL errors are handled by throwing exceptions.
- It is crucial that these errors are handled, otherwise your application could fail in unpredictable ways.
- In SYCL there are two kinds of error:
  - Synchronous errors (thrown in user thread) .
  - Asynchronous errors (thrown by the SYCL scheduler).

# SYCL EXCEPTIONS

- Asynchronous SYCL exceptions will only appear when `wait()` is called on a queue or event.

# DEBUGGING STRATEGIES

- If using default constructed queues, use `SYCL_DEVICE_FILTER` to run code on the host.
- `SYCL_QUEUE_THREAD_POOL_SIZE=1` also ensures that kernel code is executing completely serially.
- Standard tools like GDB (compile with `-gdwarf-4`), valgrind can be used to debug SYCL code.
- In-kernel `printf`s are a great way to debug code from device.

# DEBUGGING STRATEGIES

SYCL_DEVICE_FILTER=host SYCL_QUEUE_THREAD_POOL_SIZE=1 gdb ./a.out
SYCL_DEVICE_FILTER=host SYCL_QUEUE_THREAD_POOL_SIZE=1 valgrind ./a.out

# TEMPORARY FILES

- Temporary files can be outputted by using the `--save-temps` compiler flag.
- The compilation command must be invoked from inside an empty directory.

```
clang++ -fsycl -fsycl-targets=nvptx64-nvidia-cuda ../somefile.cpp --save-temps
```

# OUTPUT COMPILATION PROCESSES

- The flag `-###` can be used to output all commands used for compilation.

```
clang++ -fsycl -fsycl-targets=nvptx64-nvidia-cuda somefile.cpp -###
```

# PROFILING AND OPTIMIZATION

# COALESCED GLOBAL MEMORY

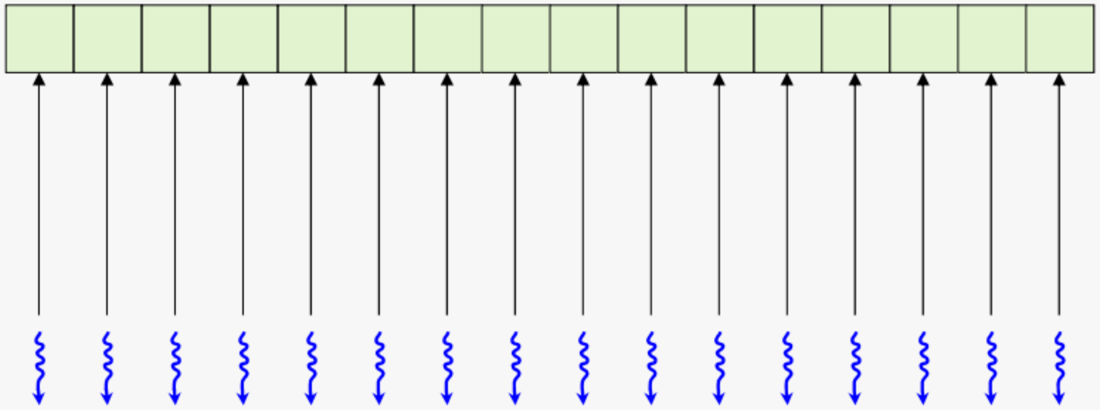
- Memory access patterns can significantly affect performance.
- Especially important when reading or writing to global memory.

# COALESCED GLOBAL MEMORY

```
float data[size];

...

f(a[globalId]);
```

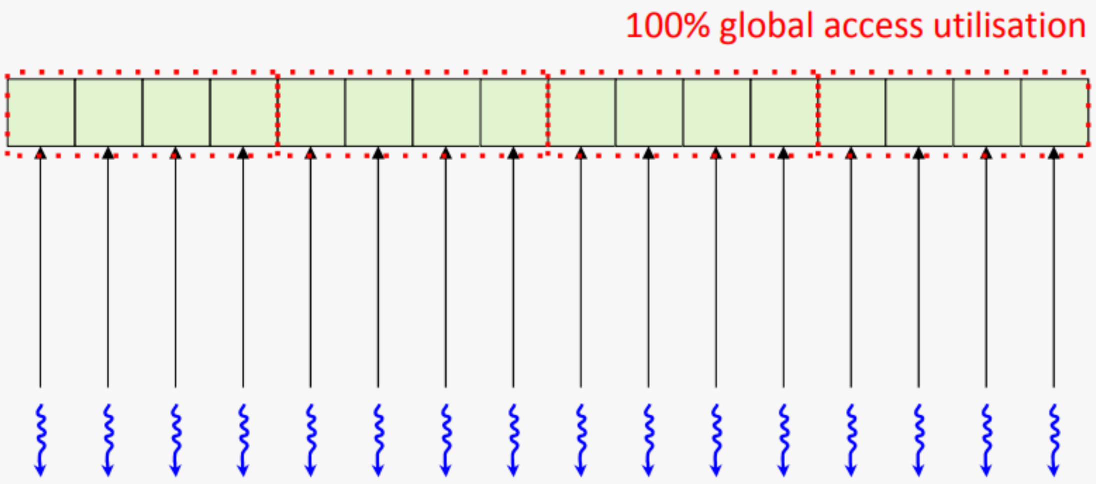


# COALESCED GLOBAL MEMORY

```
float data[size];

...

f(a[globalId]);
```

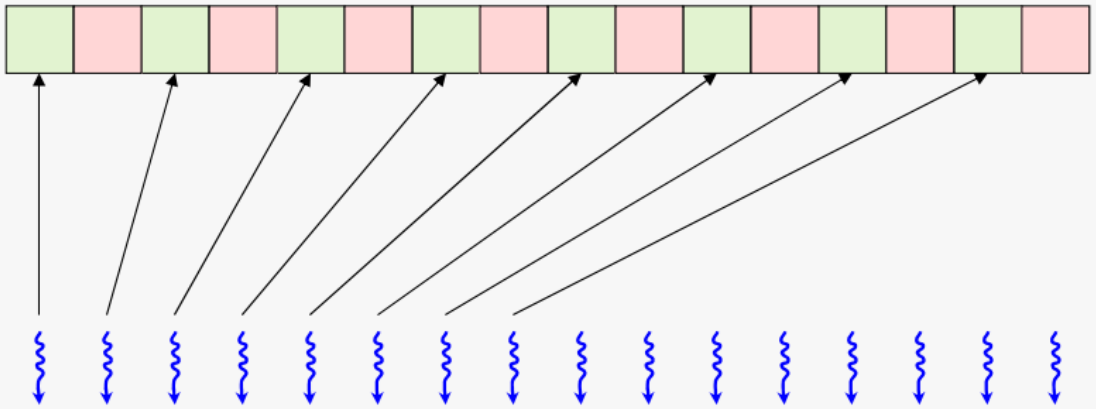


# COALESCED GLOBAL MEMORY

```
float data[size];

...

f(a[globalId * 2]);
```

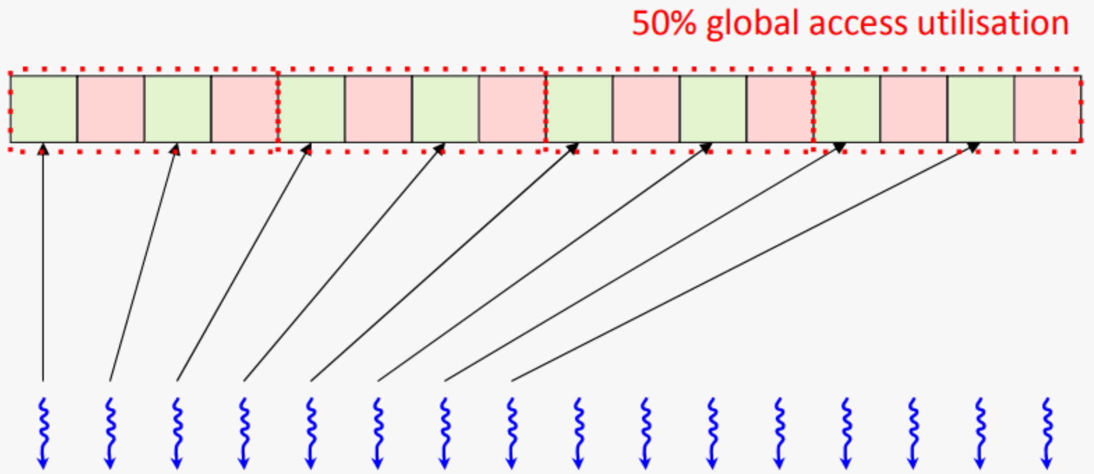


# COALESCED GLOBAL MEMORY

```
float data[size];

...

f(a[globalId * 2]);
```



# OPTIMIZATION STRATEGIES

- Test different work group sizes
- Minimize memory transfers.
- Prefer `malloc_device` over `malloc_shared`.
- Inline functions called from kernels.

# OPTIMIZATION STRATEGIES

- Use local memory where possible.
- Keep work groups converged where possible.
- Use the `sycl::native` namespace (e.g. `sycl::native::sin`), if the native accuracy is tolerable.

# OPTIMIZATION STRATEGIES

- Do not allocate more resources than you need.
- Too much pressure on local memory and registers can reduce occupancy.
- Occupancy tells us the proportion of time that compute units are kept busy.

# PROFILING SYCL CODE

- Standard Nvidia CLI tools are still available.
- For complex code, it is beneficial to name individual kernels.
  - `q.parallel_for<class my_reduce_kernel>(...)`

# NSYS

- Can be used for tracing.
- For timings, use:
  - `nsys-profile --stats=true ./a.out`

```
CUDA Kernel Statistics:

 Time(%)  Total Time (ns)  Instances  Average (ns)  Minimum (ns)  Maximum (ns)  StdDev (ns)                                                Name
 -------  ---------------  ---------  ------------  ------------  ------------  -----------  ----------------------------------------------------------------------------------------------------
    87.4           47,488          1      47,488.0        47,488        47,488          0.0  Typeinfo name for main::[lambda(cl::sycl::handler &) (instance 1)]::operator ()(cl::sycl::handler &…
    12.6            6,816          1       6,816.0         6,816         6,816          0.0  Typeinfo name for main::[lambda() (instance 1)]


CUDA Memory Operation Statistics (by time):

 Time(%)  Total Time (ns)  Count  Average (ns)  Minimum (ns)  Maximum (ns)  StdDev (ns)      Operation
 -------  ---------------  -----  ------------  ------------  ------------  -----------  ------------------
    76.2           11,360      1      11,360.0        11,360        11,360          0.0  [CUDA memcpy HtoD]
    23.8            3,552      1       3,552.0         3,552         3,552          0.0  [CUDA memcpy DtoH]


CUDA Memory Operation Statistics (by size):

 Total (MB)  Count  Average (MB)  Minimum (MB)  Maximum (MB)  StdDev (MB)      Operation
 ----------  -----  ------------  ------------  ------------  -----------  ------------------
      0.131      1         0.131         0.131         0.131        0.000  [CUDA memcpy HtoD]
      0.000      1         0.000         0.000         0.000        0.000  [CUDA memcpy DtoH]
```

# NSYS

- Notice that making a context associated with a queue takes considerable time for CUDA backend (0.2s).
- This can be mitigated by using CUDA primary context.

```
auto c = sycl::context{
    sycl::ext::oneapi::cuda::property::context::use_primary_context{}};
auto q = sycl::queue{c, sycl::default_selector{}};
```

# NCU

- Can be used for detailed kernel analysis.
- Occupancy metrics are usually most important.
- Example command:
  - `ncu --log-file my_ncu_output.csv --print-kernel-base mangled --details-all --csv ./a.out`

```
GPU Speed Of Light Throughput","DRAM Frequency","cycle/second","968,720,379.15",
GPU Speed Of Light Throughput","SM Frequency","cycle/second","610,083,784.70",
GPU Speed Of Light Throughput","Elapsed Cycles","cycle","4,121",
GPU Speed Of Light Throughput","Memory [%]","%","2.10",
GPU Speed Of Light Throughput","DRAM Throughput","%","0.78",
GPU Speed Of Light Throughput","Duration","nsecond","6,752",
GPU Speed Of Light Throughput","L1/TEX Cache Throughput","%","1.80",
GPU Speed Of Light Throughput","Waves Per SM","","0.15",
GPU Speed Of Light Throughput","L2 Cache Throughput","%","2.73",
GPU Speed Of Light Throughput","SM Active Cycles","cycle","1,513.48",
GPU Speed Of Light Throughput","Compute (SM) [%]","%","0.64",
SpeedOfLight","","","","SOLBottleneck","WRN","This kernel grid is too small to fill the available resour
ces on this device, resulting in only 0.1 full waves across all SMs. Look at Launch Statistics for more
details."
Launch Statistics","Block Size","","32",
Launch Statistics","Function Cache Configuration","","cudaFuncCachePreferNone",
Launch Statistics","Grid Size","","512",
Launch Statistics","Registers Per Thread","register/thread","16",
Launch Statistics","Shared Memory Configuration Size","byte","32,768",
Launch Statistics","Driver Shared Memory Per Block","byte/block","1,024",
Launch Statistics","Dynamic Shared Memory Per Block","byte/block","0",
Launch Statistics","Static Shared Memory Per Block","byte/block","0",
Launch Statistics","Threads","thread","16,384",
Launch Statistics","Waves Per SM","","0.15",
Launch Statistics","sm__maximum_warps_per_active_cycle_pct","%","50",
Launch Statistics","sm__warps_active.avg.pct_of_peak_sustained_active","%","6.11",
Occupancy","Block Size","","32",
Occupancy","Block Limit SM","block","32",
Occupancy","Block Limit Registers","block","128",
Occupancy","Block Limit Shared Mem","block","164",
Occupancy","Block Limit Warps","block","64",
Occupancy","Warp Occupancy","","3,670",
Occupancy","Warp Occupancy","","4,512",
Occupancy","Warp Occupancy","","1,036",
Occupancy","Registers Per Thread","register/thread","16",
Occupancy","Shared Memory Per Block","byte/block","1,024",
Occupancy","Theoretical Active Warps per SM","warp","32",
Occupancy","Theoretical Occupancy","%","50",
Occupancy","Achieved Occupancy","%","6.11",
Occupancy","Achieved Active Warps Per SM","warp","3.91",
Occupancy","","","","Occupancy","WRN","This kernel's theoretical occupancy (50.0%) is limited by the num
ber of blocks that can fit on the SM The difference between calculated theoretical (50.0%) and measured
achieved occupancy (6.1%) can be the result of warp scheduling overheads or workload imbalances during t
he kernel execution. Load imbalances can occur between warps within a block as well as across blocks of
the same kernel."
```

# QUESTIONS