Building a Portable, Scalable, Performant ZFP Backend using oneAPI and SYCL to Advance Exascale Computing: A Developer Perspective

ALPER SAHISTAN, NATE MORRICAL, PETER LINDSTROM, VALERIO PASCUCCI

Agenda

- Introduction
- Compression and ZFP
- Utilizing DPCT Migration Tool
    - Comparison between CUDA and generated SYCL codes
    - Some Rewrites
- Issues and Solutions with DPCT
- Debugging with oneAPI-gdb
- Kernel Timing
- Results
- Conclusions
- Future Work

Introduction

- Most applications are memory-bound
  - Device memory or transfer-time > Computation time
  - Especially relevant for **GPU**s
- **ZFP** as a compression tool plays a vital role in HPC
  - Relaxes memory constraints
  - Exploits the regularity in data
- **oneAPI** and **SYCL** as enablers
  - Portable and scalable ZFP backend
  - We aim to share our experiences using these tools.

Compression and ZFP

Problems with large data today:

- Processor and clock speeds have plateaued and emphasis is on parallel computing.
- Memory per-core ⬇
- Data set sizes and computation need ⬆
- Memory > FLOPs

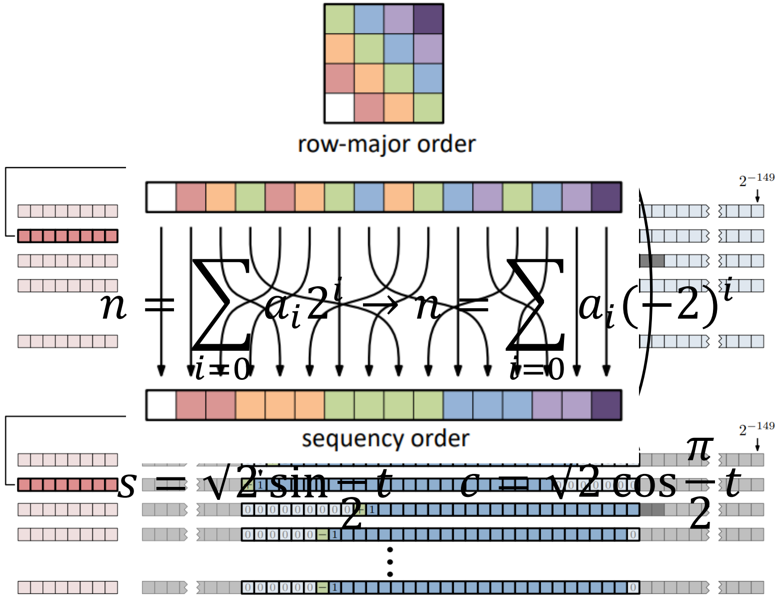
Viable Solution: **Compression**

- With floating-point data, lossy-compression offers more data reduction.

Compression and ZFP

ZFP data compression:

1. Block-Partitioning: *d*- dimentional array into 4^d blocks.
2. FP numbers are converted into block-floating-point representation and they are shifted/rounded to 4^d signed integers
3. Application of decorrelating transform
4. Reorder integer coefficients by sequency
5. Two's complement signed integers are converted into negabinary
6. List of 4^d integers are transposed: least to most significant ordering
7. Each bit plane is compressed losslessly using embedded coding
8. The embedded coder emits one bit at a time until stopping criteria are satisfied: **fixed rate, fixed precision, or fixed accuracy**



$$n = \sum_{i=0} a_i 2^i \rightarrow n = \sum_{i=0} a_i(-2)^i$$

$$s = \sqrt{2}\sin\frac{\pi}{2}t \quad c = \sqrt{2}\cos\frac{\pi}{2}t$$

Utilizing DPCT Migration Tool

- We have utilized DPCT tool to convert CUDA implementation to SYCL.

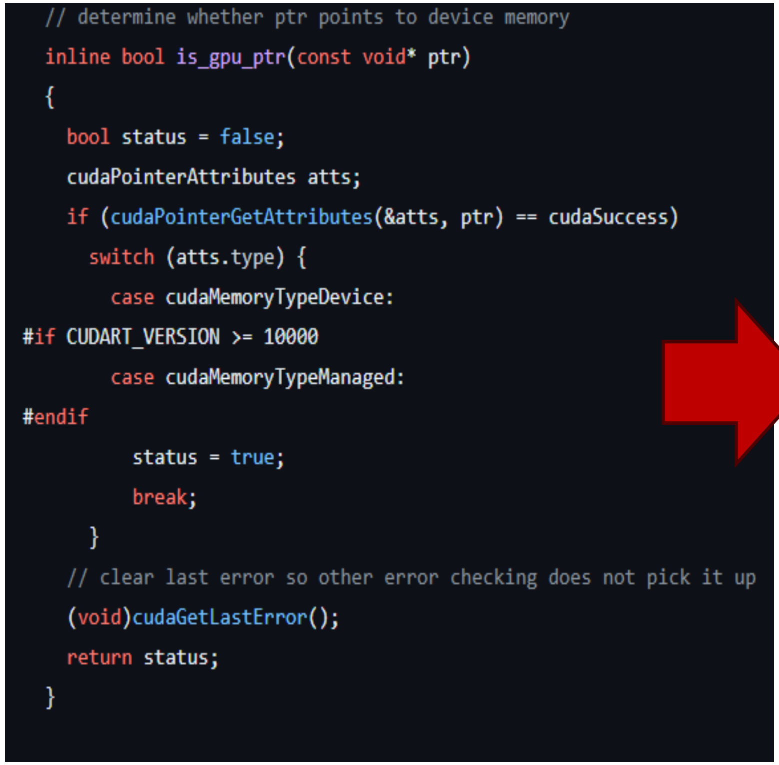
  -
    ```
    dpct --in-root=. ./src/cuda/interface.cu  --use-experimental-features="occupancy-calculation,nd_range_barrier" --extra-arg="-I ./include“
    ```
- Translations for several CUDA functions were unavailable in the 2023.1.0 so we are strictly using DPCT 2023.2.0
- Currently implementation works for fixed-rate mode on Linux systems.

```
// encode kernel
template <typename Scalar> __global__
void encode1_kernel(…){ //omitted for brevity
  const size_t blockId = blockIdx.x + (size_t)gridDim.x *
(blockIdx.y + (size_t)gridDim.y * blockIdx.z);
  // each thread gets a block; block index = global thread index
  const size_t block_idx = blockId * blockDim.x + threadIdx.x;
  // number of zfp blocks
  const size_t blocks = (size + 3) / 4;
  // return if thread has no blocks assigned
  if (block_idx >= blocks)
    return;
  // logical position in 1d array
  const size_t pos = block_idx;
  const ptrdiff_t x = pos * 4;
  // offset into field
  const ptrdiff_t offset = x * stride;
  // initialize block writer
  BlockWriter::Offset bit_offset = block_idx * maxbits;
  BlockWriter writer(d_stream, bit_offset);
  // gather data into a contiguous block
  Scalar fblock[ZFP_1D_BLOCK_SIZE];
  const uint nx = (uint)min(size_t(size - x), size_t(4));
...
}
```

```
// encode kernel
template <typename Scalar>
void encode1_kernel(…){ //omitted for brevity
  const size_t blockId = item_ct1.get_group(2) +
    (size_t)item_ct1.get_group_range(2) *
    (item_ct1.get_group(1) +
    (size_t)item_ct1.get_group_range(1) *
     item_ct1.get_group(0));
  // each thread gets a block; block index = global thread
index
  const size_t block_idx = blockId *
     item_ct1.get_local_range(2) + item_ct1.get_local_id(2);
  // number of zfp blocks
  const size_t blocks = (size + 3) / 4;
  // return if thread has no blocks assigned
  if (block_idx >= blocks)
    return;
  // logical position in 1d array
  const size_t pos = block_idx;
  const ptrdiff_t x = pos * 4;
  // offset into field
  const ptrdiff_t offset = x * stride;
  // initialize block writer
  BlockWriter::Offset bit_offset = block_idx * maxbits;
  BlockWriter writer(d_stream, bit_offset);
  // gather data into a contiguous block
  Scalar fblock[ZFP_1D_BLOCK_SIZE];
  const uint nx = (uint)::sycl::min(size_t(size - x),
size_t(4));
...
}
```

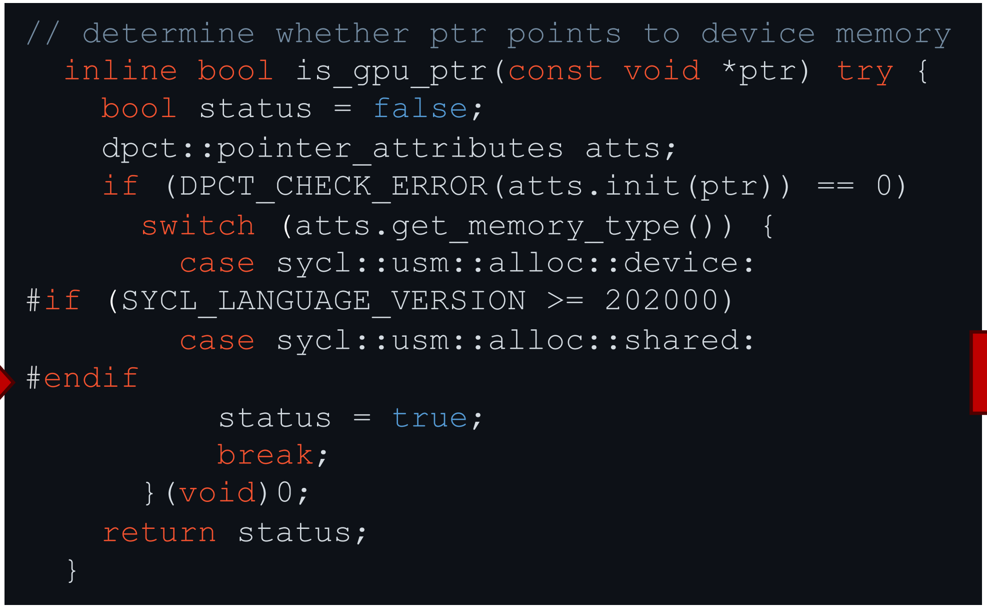
Some Rewrites

CUDA



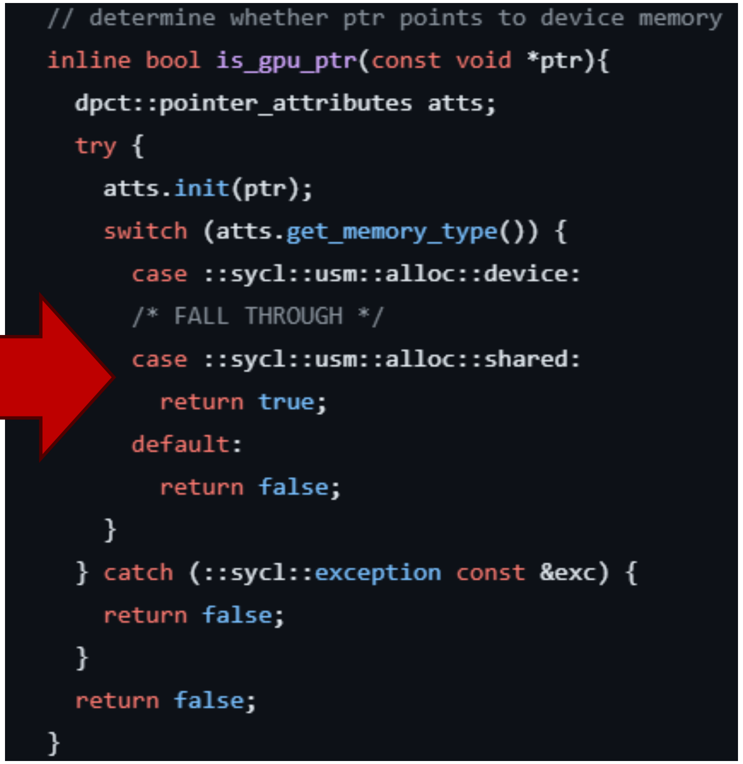
```
  // determine whether ptr points to device memory
  inline bool is_gpu_ptr(const void* ptr)
  {
    bool status = false;
    cudaPointerAttributes atts;
    if (cudaPointerGetAttributes(&atts, ptr) == cudaSuccess)
      switch (atts.type) {
        case cudaMemoryTypeDevice:
#if CUDART_VERSION >= 10000
        case cudaMemoryTypeManaged:
#endif
          status = true;
          break;
      }
    // clear last error so other error checking does not pick it up
    (void)cudaGetLastError();
    return status;
  }
```

SYCL from DPCT



```
// determine whether ptr points to device memory
  inline bool is_gpu_ptr(const void *ptr) try {
    bool status = false;
    dpct::pointer_attributes atts;
    if (DPCT_CHECK_ERROR(atts.init(ptr)) == 0)
      switch (atts.get_memory_type()) {
        case sycl::usm::alloc::device:
#if (SYCL_LANGUAGE_VERSION >= 202000)
        case sycl::usm::alloc::shared:
#endif
          status = true;
          break;
      }(void)0;
    return status;
  }
```

SYCL after corrections

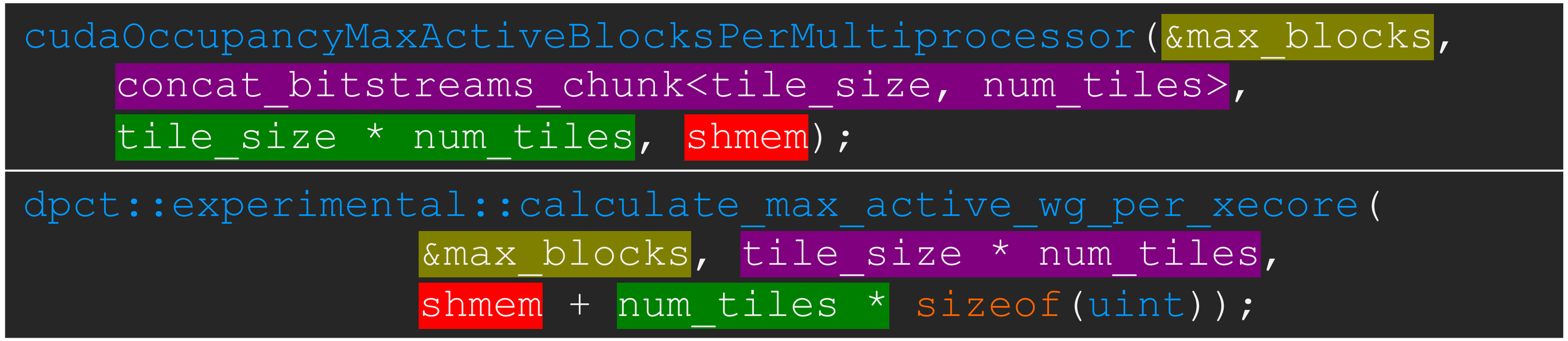


```
// determine whether ptr points to device memory
inline bool is_gpu_ptr(const void *ptr){
  dpct::pointer_attributes atts;
  try {
    atts.init(ptr);
    switch (atts.get_memory_type()) {
      case ::sycl::usm::alloc::device:
      /* FALL THROUGH */
      case ::sycl::usm::alloc::shared:
        return true;
      default:
        return false;
    }
  } catch (::sycl::exception const &exc) {
    return false;
  }
  return false;
}
```

Issues and Solutions with DPCT

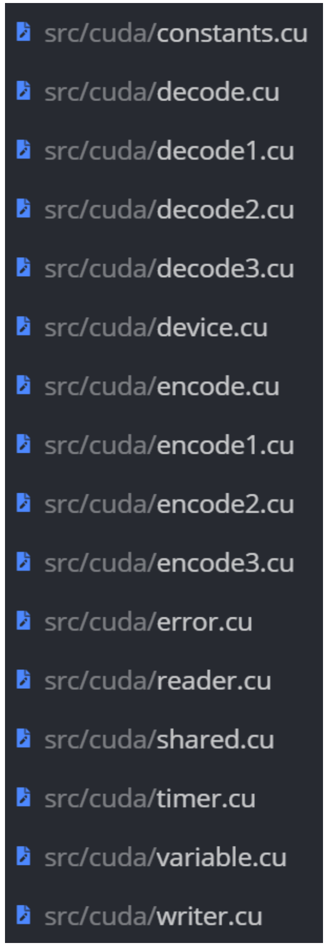
- **Problem:** DPCT does not translate .cuh files.
  Solution: Renaming all .cuh files to .cu and adjusting the inclusions and Cmake.
  - We have reported this problem

- **Problem:** `cudaOccupancyMaxActiveBlocksPerMultiprocessor` is not supported.
  Solution: Updating DPCT tool to 2023.2.0 and turning on `--use-experimental-features=occupancy-calculation`



```
cudaOccupancyMaxActiveBlocksPerMultiprocessor(&max_blocks,
    concat_bitstreams_chunk<tile_size, num_tiles>,
    tile_size * num_tiles, shmem);
```

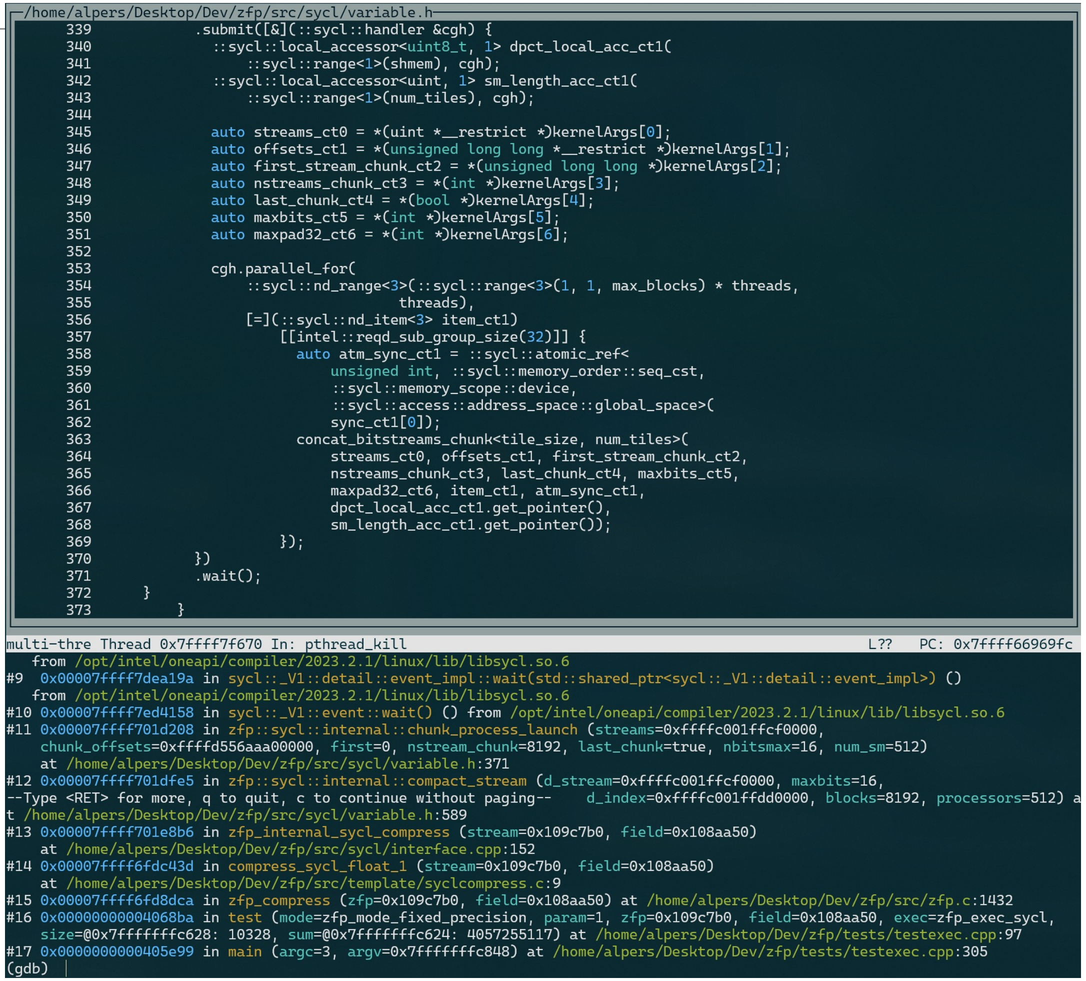
```
dpct::experimental::calculate_max_active_wg_per_xecore(
                &max_blocks, tile_size * num_tiles,
                shmem + num_tiles * sizeof(uint));
```



```
src/cuda/constants.cu
src/cuda/decode.cu
src/cuda/decode1.cu
src/cuda/decode2.cu
src/cuda/decode3.cu
src/cuda/device.cu
src/cuda/encode.cu
src/cuda/encode1.cu
src/cuda/encode2.cu
src/cuda/encode3.cu
src/cuda/error.cu
src/cuda/reader.cu
src/cuda/shared.cu
src/cuda/timer.cu
src/cuda/variable.cu
src/cuda/writer.cu
```

Debugging with oneAPI-gdb

- The variable-rate modes of ZFP’s SYCL port has a bug at the moment.
  - The bug causes it to crash with `terminate called after throwing an instance of 'sycl::_V1::runtime_error’ what(): Native API failed. Native API returns: -14 (PI_ERROR_EXEC_STATUS_ERROR_FOR_EVENTS_IN_WAIT_LIST) -14 (PI_ERROR_EXEC_STATUS_ERROR_FOR_EVENTS_IN_WAIT_LIST)`
- We employed oneAPI-gdb to insert breakpoints watch variables and stack-trace
  - We traced the bug to the kernel `concat_bitstreams_chunk`
  - However we failed to get the tracing to go inside the kernel.



```
/home/alpers/Desktop/Dev/zfp/src/sycl/variable.h
      .submit([&](::sycl::handler &cgh) {
        ::sycl::local_accessor<uint8_t, 1> dpct_local_acc_ct1(
            ::sycl::range<1>(shmem), cgh);
        ::sycl::local_accessor<uint, 1> sm_length_acc_ct1(
            ::sycl::range<1>(num_tiles), cgh);

        auto streams_ct0 = *(uint *__restrict *)kernelArgs[0];
        auto offsets_ct1 = *(unsigned long long *__restrict *)kernelArgs[1];
        auto first_stream_chunk_ct2 = *(unsigned long long *)kernelArgs[2];
        auto nstreams_chunk_ct3 = *(int *)kernelArgs[3];
        auto last_chunk_ct4 = *(bool *)kernelArgs[4];
        auto maxbits_ct5 = *(int *)kernelArgs[5];
        auto maxpad32_ct6 = *(int *)kernelArgs[6];

        cgh.parallel_for(
            ::sycl::nd_range<3>(::sycl::range<3>(1, 1, max_blocks) * threads,
                              threads),
            [=](::sycl::nd_item<3> item_ct1)
                [[intel::reqd_sub_group_size(32)]] {
                  auto atm_sync_ct1 = ::sycl::atomic_ref<
                      unsigned int, ::sycl::memory_order::seq_cst,
                      ::sycl::memory_scope::device,
                      ::sycl::access::address_space::global_space>(
                      sync_ct1[0]);
                  concat_bitstreams_chunk<tile_size, num_tiles>(
                      streams_ct0, offsets_ct1, first_stream_chunk_ct2,
                      nstreams_chunk_ct3, last_chunk_ct4, maxbits_ct5,
                      maxpad32_ct6, item_ct1, atm_sync_ct1,
                      dpct_local_acc_ct1.get_pointer(),
                      sm_length_acc_ct1.get_pointer());
                });
      })
      .wait();
}
    }
multi-thre Thread 0x7ffff7f670 In: pthread_kill                                                  L??   PC: 0x7ffff66969fc
   from /opt/intel/oneapi/compiler/2023.2.1/linux/lib/libsycl.so.6
#9  0x00007ffff7dea19a in sycl::_V1::detail::event_impl::wait(std::shared_ptr<sycl::_V1::detail::event_impl>) ()
   from /opt/intel/oneapi/compiler/2023.2.1/linux/lib/libsycl.so.6
#10 0x00007ffff7ed4158 in sycl::_V1::event::wait() () from /opt/intel/oneapi/compiler/2023.2.1/linux/lib/libsycl.so.6
#11 0x00007ffff701d208 in zfp::sycl::internal::chunk_process_launch (streams=0xffffc001ffcf0000,
    chunk_offsets=0xffffd556aaa00000, first=0, nstream_chunk=8192, last_chunk=true, nbitsmax=16, num_sm=512)
    at /home/alpers/Desktop/Dev/zfp/src/sycl/variable.h:371
#12 0x00007ffff701dfe5 in zfp::sycl::internal::compact_stream (d_stream=0xffffc001ffcf0000, maxbits=16,
--Type <RET> for more, q to quit, c to continue without paging--    d_index=0xffffc001ffdd0000, blocks=8192, processors=512) a
t /home/alpers/Desktop/Dev/zfp/src/sycl/variable.h:589
#13 0x00007ffff701e8b6 in zfp_internal_sycl_compress (stream=0x109c7b0, field=0x108aa50)
    at /home/alpers/Desktop/Dev/zfp/src/sycl/interface.cpp:152
#14 0x00007ffff6fdc43d in compress_sycl_float_1 (stream=0x109c7b0, field=0x108aa50)
    at /home/alpers/Desktop/Dev/zfp/src/template/syclcompress.c:9
#15 0x00007ffff6fd8dca in zfp_compress (zfp=0x109c7b0, field=0x108aa50) at /home/alpers/Desktop/Dev/zfp/src/zfp.c:1432
#16 0x00000000004068ba in test (mode=zfp_mode_fixed_precision, param=1, zfp=0x109c7b0, field=0x108aa50, exec=zfp_exec_sycl,
    size=@0x7fffffffc628: 10328, sum=@0x7fffffffc624: 4057255117) at /home/alpers/Desktop/Dev/zfp/tests/testexec.cpp:97
#17 0x0000000000405e99 in main (argc=3, argv=0x7fffffffc848) at /home/alpers/Desktop/Dev/zfp/tests/testexec.cpp:305
(gdb)
```

Kernel Timing

- We used `sycl::event` objects to time the compression/decompression kernels.
    - Surprisingly difficult.

We attempted multiple solutions:

```
// Fixes over DPCT translation for CUDA event timers in ZFP
dpct::device_ext& dev_ct1 = dpct::get_current_device();
sycl::queue q_b((::sycl::device)dev_ct1,
::sycl::property::queue::enable_profiling{});
sycl::event start =  q_b.ext_oneapi_submit_barrier();
q_ct1.submit(…{…}) .wait();//kernel
sycl::queue &q_b2((::sycl::device)dev_ct1,
::sycl::property::queue::enable_profiling{});
stop = q_b2.ext_oneapi_submit_barrier();
stop.wait_and_throw();
q_b2.wait();

float time = stop.get_profiling_info<info::event_profiling::command_start>()
- start. get_profiling_info<info::event_profiling::command_end>();
```

```
ed
yc
fi
 t
 d
ing' property.
LID_ARG_VALUE)
```

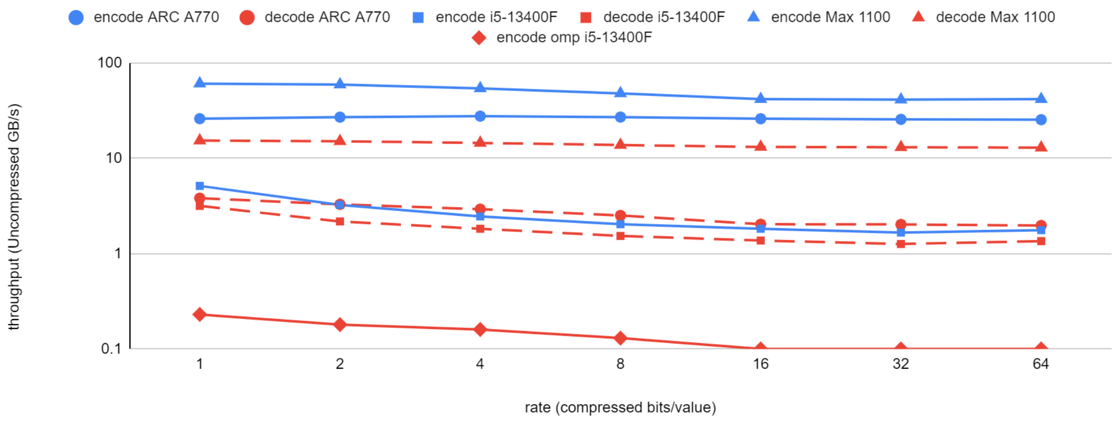
Produces inconsistent timings for different

Solution ✅

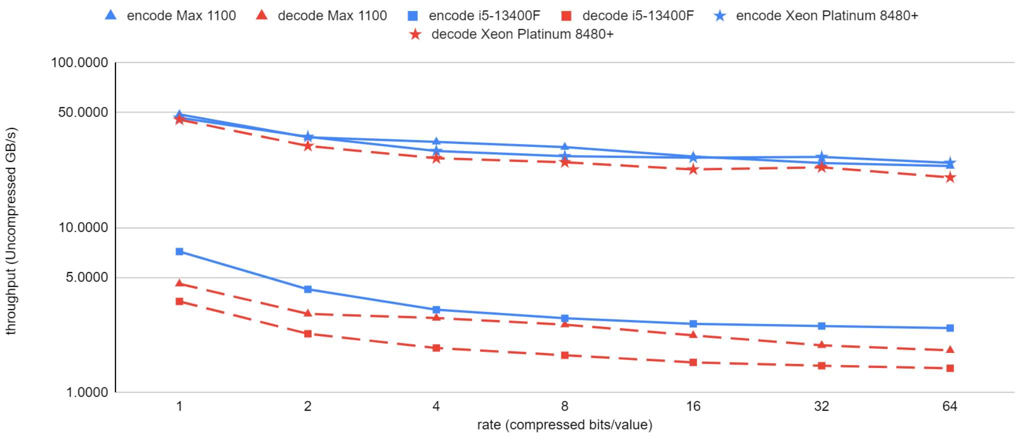
combinations (possible bug in oneAPI)

Results(higher is better)

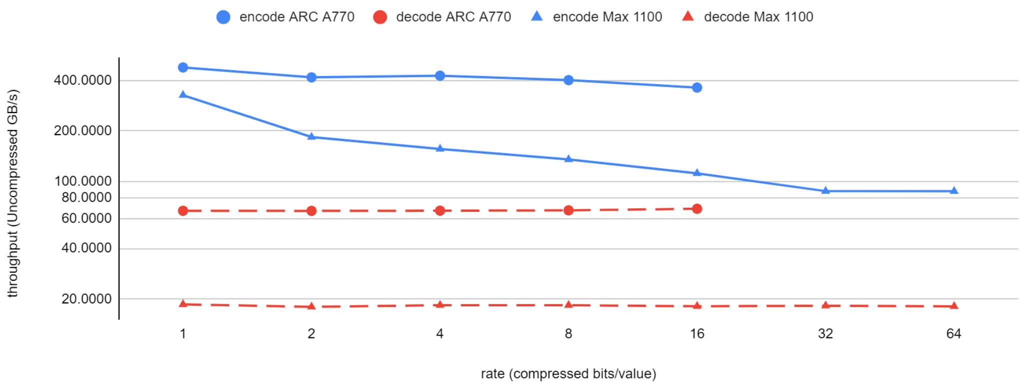
SCALE-T(0.52GB)



Miranda-density(0.28GB)



EXAFEL-LCLS(8.5GB)



[1] K. Zhao et al., "SDRBench: Scientific Data Reduction Benchmark for lossy compressors," 2020 IEEE International Conference on Big Data (Big Data), 2020. doi:10.1109/bigdata50022.2020.9378449

[2] "Scientific Data Reduction Benchmarks," sdrbench.github.io, https://sdrbench.github.io/ (accessed Nov. 26,

Conclusion

- We successfully transformed the ZFP library's fixed-rate mode from a CUDA implementation to Intel's SYCL using the DPCT tool.
- We shared our insights about using Intel's oneAPI development tools.
  - As an improvement, we recommend the implementation of stack tracing within kernel functions for oneAPI-gdb
- We recommend the support of more convenient driver maintenance for Linux systems.
  - Certain functionalities are restricted to specific kernel versions.
- As double precision floating point numbers are extensively utilized in exascale computations, we strongly advocate their emulated implementation for Intel ARC GPUs.

Future Work

- Our work on ZFP's variable-rate mode is still on going
- Several algorithmic and implementation optimizations for the SYCL port
- Possible visualization tools for rendering the compressed data