QUDA-DPC++ migration

Alexei Strelchenko

Scientific Computing Division @ Fermilab

February 26, 2020

The QUDA library for Lattice Quantum Field Theory

- http://lattice.github.com/quda (open source, BSD license)
- GPU backend for BQCD, Chroma, CPS, MILC, TIFR, etc.
- Provides:
  - Various solvers for all major fermionic discretizations
  - Additional performance-critical routines needed for gauge field generation
- Maximize performance:
  - Exploit physical symmetries to minimize memory traffic
  - Mixed-precision methods
  - Autotuning for high performance on all CUDA-capable architectures
  - Domain-decomposed (Schwarz) and multigrid preconditioners for strong scaling
- Objectives:
  - Prepare QUDA for Intel exascale architectures with "reasonable" efforts

The QUDA library: autotuning

- QUDA includes an autotuner for ensuring optimal kernel performance
  - virtual C++ class "Tunable" that is derived for each kernel one wants to autotune
- By default Tunable classes will autotune 1-d CTA size, shared memory size, grid size
  - Derived specializations do 2-d and 3-d CTA tuning
- Tuned parameters are stored in a std::map and dumped to disk for later reuse
- Built in performance metrics and profiling

QUDA porting efforts

- QUDA major challenges
  - Low-level optimized compute kernels
  - (though most of them have x86 counterparts)
  - CUDA API calls for memory management etc.
- HW/SW specifics:
  - Some kernels exploit TC (in future releases)
  - All communication critical kernels will exploit NVSHMEM features

QUDA porting strategy

- Employ DPC++ CT for source code migration
  - Option 1: convert CUDA-based versions
  - Option 2: port x86 versions
- Identify minimal but representative subset of QUDA
- Migration procedure:
  - Create a compilation Database file
  - Use Compilation Database File to convert source to DPC++
  - Verify the Source for Correctness

Minimal source subset example

```
set (QUDA_OBJS
  # cmake-format: sortable
  timer.cpp malloc.cpp
  interface_quda.cpp util_quda.cpp
  color_spinor_field.cpp color_spinor_util.cu
  cpu_color_spinor_field.cpp cuda_color_spinor_field.cpp
  lattice_field.cpp
  tune.cpp
  blas_quda.cu reduce_quda.cu
  random.cu
  comm_common.cpp ${COMM_OBJS} ${NUMA_AFFINITY_OBJS} ${QIO_UTIL}
  copy_color_spinor.cu spinor_noise.cu
  copy_color_spinor_dd.cu copy_color_spinor_ds.cu
  copy_color_spinor_dh.cu copy_color_spinor_dq.cu
  copy_color_spinor_ss.cu copy_color_spinor_sd.cu
  copy_color_spinor_sh.cu copy_color_spinor_sq.cu
  copy_color_spinor_hd.cu copy_color_spinor_hs.cu
  copy_color_spinor_hh.cu copy_color_spinor_hq.cu
  copy_color_spinor_qd.cu copy_color_spinor_qs.cu
  copy_color_spinor_qh.cu copy_color_spinor_qq.cu
  quda_cuda_api.cpp
  version.cpp )
# cmake-format: on
```

Figure: 1: Source subset

QUDA porting efforts

- Recent DPCT (beta4)
  - crashed with the latest version of QUDA
  - as the first case used with older version

Convertion script example

```
source /opt/oneapi/inteloneapi/dpcpp-ct/latest/env/vars.sh

QUDA_HOME=/home/astrel/Work/OneAPI/quda-oneapi-v2
PROJ_DIR=$QUDA_HOME/lib

OUTDIR=dpct_output_$(date +%s)

# copy_color_spinor_d*.cu

dpct -p -in-root=$PROJ_DIR -out-root=$OUTDIR \
     --keep-original-code \
     --extra-arg="-I$QUDA_HOME/include" \
     --extra-arg="-I./include" \
     --extra-arg="-std=c++14" \
     --extra-arg="-I$QUDA_HOME/lib/copy_color_spinor.cuh" \
     --extra-arg="-I$QUDA_HOME/include/externals" \
     $QUDA_HOME/lib/copy_color_spinor_dd.cu
```

Figure: 2: Source subset

DPCT issues

- DPCT ignores header files
  - some *.h files contain CUDA specific stuff!
- Solution:
  - Change file extensions, i.e., *.h $\rightarrow$ *.cu
  - Warning: changing with *.cuh does not help!
  - apply dpct to *.cu files
  - inspect for correctness and change *.cu $\rightarrow$ *.h

DPCT issues

- DPCT may not correctly process macros or even fails to generate a dpcpp file

```
/ DPCT_ORIG       printfQuda("%e Failures: %d / %d  = %e\n",
 * pow(10.0,-(f+1)/(double)tol), /
 /* Complete FAIL! */
      printfQuda(
          "%e Failures: %d / %d  = %e\n",
          pow(printfQuda("%e Failures: %d / %d  = %e\n",
                         pow(10.0, -(f + 1) / (double)tol), fail[f],
                         u.Volume() * N, fail[f] / (double)(u.Volume() * N)),
              -(f + 1) / (double)tol),
          fail[f], u.Volume() * N, fail[f] / (double)(u.Volume() * N));

    }
```

Figure: 2: Macro pre-processing

DPCT issues

- DPCT may fail to convert standard CUDA API calls

```
463  / DPCT_ORIG     cudaMemsetAsync(v, 0, bytes, streams[Nstream-1]);/
464      streams[Nstream - 1]->memset(v, 0, bytes);
```

Figure: 2: CUDA API call

- DPCT first experience
  - Easy to use out of box
  - Requires an extra inspections of generated codes and still fails on some source files
  - Requires (manual) preparations for some files
- Deployment:
  https://github.com/lattice/quda/tree/experimental/oneapi-quda