Presentation

Developing Medical imaging application across GPU, FPGA, and CPU using oneAPI

Presenters: Wang Yong (yong4.wang@intel.com), Scott Wang (scott.wang@intel.com)
With contribution from: Zhou Yongfa, Wang Yang, Xu Qing, Wang Chen
Table of Content

• Background
• Code Migration
• Beamforming Optimization on GPU
• Beamforming implementation on FPGA
• Results and performance
Background
What is SUPRA and why we need it?

SUPRA is an open-source pipeline for fully software defined ultrasound processing.

- https://github.com/IFL-CAMP/supra
What is software beamforming?

SUPRA contains standard medical ultrasound software beamforming algorithms.

![Software beamforming illustration](source: Lars Grønvold)

Intel’s products in software beamforming:
- Core & Gen9 graphics, DG1, Arria 10 & Stratix 10, Intel oneAPI.
Code Migration
The Intel® DPC++ Compatibility Tool assists in migrating your existing CUDA code to Data Parallel C++ (DPC++) code.

DPC++ is based on ISO C++ and incorporates standard SYCL* and community extensions to simplify data parallel programming.

Inline comments help you finish writing and tuning your DPC++ code.

### Migration Command:
```
```

### SUPRA Migration Summary

<table>
<thead>
<tr>
<th>OneAPI Version</th>
<th>Total num of migration place</th>
<th>Success migrated</th>
<th>Need modify</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>beta07</td>
<td>84</td>
<td>63</td>
<td>21</td>
<td>75%</td>
</tr>
<tr>
<td>golden</td>
<td>84</td>
<td>75</td>
<td>9</td>
<td>89%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>File Type</th>
<th>*.cpp</th>
<th>*.cu</th>
<th>*.h</th>
</tr>
</thead>
<tbody>
<tr>
<td>File num</td>
<td>1</td>
<td>4</td>
<td>23</td>
</tr>
</tbody>
</table>

*File num: number of migrated file*
## Migrated code APIs

<table>
<thead>
<tr>
<th>Category</th>
<th>oneAPI APIs</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Memory Management</strong></td>
<td>sycl::malloc_device()</td>
</tr>
<tr>
<td></td>
<td>sycl::malloc_shared()</td>
</tr>
<tr>
<td></td>
<td>sycl::free()</td>
</tr>
<tr>
<td></td>
<td>sycl::malloc_host()</td>
</tr>
<tr>
<td></td>
<td>sycl::queue.memcpy()</td>
</tr>
<tr>
<td></td>
<td>Sycl::queue.memset()</td>
</tr>
<tr>
<td><strong>sycl::queue</strong></td>
<td>dpct::get_current_device().create_queue()</td>
</tr>
<tr>
<td></td>
<td>dpct::get_default_queue()</td>
</tr>
<tr>
<td></td>
<td>sycl::queue()</td>
</tr>
<tr>
<td></td>
<td>sycl::queue.submit()</td>
</tr>
<tr>
<td></td>
<td>sycl::queue.wait()</td>
</tr>
<tr>
<td><strong>Math</strong></td>
<td>sycl::sqrt(); sycl::floor(); sycl::fabs(); sycl::round()</td>
</tr>
<tr>
<td></td>
<td>sycl::max(); sycl::min(); sycl::log10(); sycl::pow()</td>
</tr>
<tr>
<td><strong>Express Parallel</strong></td>
<td>sycl::nd_item&lt;&gt;; Sycl::nd_range&lt;&gt;; Sycl::range&lt;&gt;</td>
</tr>
<tr>
<td></td>
<td>sycl::id&lt;&gt;; Sycl::nd_item().get_local_range()</td>
</tr>
<tr>
<td></td>
<td>Sycl::nd_item().get_group()</td>
</tr>
<tr>
<td></td>
<td>Sycl::nd_item().get_local_id()</td>
</tr>
</tbody>
</table>
Manually migration example

1. **migrate**
   ```
   #ifdef HAVE_CUDA
   --> m_creationEvent;
   #endif
   
   #ifdef HAVE_CUDA
   --> sycl::event m_creationEvent;
   --> std::chrono::time_point<std::chrono::steady_clock> m_creationEvent_ctl1;
   #endif
   ```

   **modify** Remove the migrated sycl::event and std::chrono object

2. **migrate**
   ```
   cudaMemcpy((sm_streams[k]), dpct::get_current_device().create_queue());
   
   sm_streams[k] = new sycl::queue(dpct::get_default_queue_wait().get_context(), dpct::get_default_queue_wait().get_device(), property_list);
   ```
Manually migration example

DPCT tool can't migrate CUDA thrust library related code, so it must be rewritten using oneAPI model.
Migration success example

Memory allocate related function were successfully migrated.
Beamforming Optimization on GPU
Beamforming (Delay and Sum) introduction

Geometrical illustration of the pulse-echo process

\[ \Delta t = \frac{(T + R)}{c_0} \]

c₀: the speed of ultrasound travel in the body.
Beamforming introduction

Suppose pre-beamformed data: 128 scanlines, 64 channels, 2337 samples. It is arranged in a 3-D data structure: $\text{rf}\_\text{data}[\text{scanline}][\text{channel}][\text{sample}]$.

Memory required to store pre-beamformed data per image frame is:

$$(128 \times 64 \times 2334 \times 2) \text{ bytes} = 36.5 \text{ MB}$$
Beamforming algorithm implementation in single thread

Input: 3 dimensional pre-beamformed rf_data[numScanline][numSample][numChannel].
Output: A single frame image.

```plaintext
for k = 0 -> (numScanline - 1) do
    Read rf_data of k-th scanline;
    for i = 0 -> (numSample - 1) do
        for j = 0 -> (numChannel - 1) do
            calculate delay of j-th channel.
            load data and perform weighting.
            Sum all channels.
        write pixel value.
    Save and display image.
```

Iteration in output image column direction. Can be calculate Independently.
Iteration in output image row direction. Can be calculate Independently.
Beamforming implementation in parallel on GPU
Optimization #1

The optimization is in RxBeamformerCuda.dp.cpp and RxSampleBeamformerDelayAndSum.h file.

Function rxBeamformingDTSPACEKernel and sampleBeamfor2D are optimized.

Optimization idea:

**CUDA**: Each thread calculates a point; every point iterates 64 times.

**oneAPI**: each thread load 2 points in vertical direction, Iterates 8 times.
Optimization #1  oneAPI code

In `RxBeamformerCuda.cu` the function been called; the return value is a `float`.

In `sampleBeamform2D` function, calculate single point each call. The for loop at least iterates 64 times.

Source code: supra/src/SupraLib/Beamformer/ RxSampleBeamformerDelayAndSum.h
Optimization #2
Another optimization in BeamformingNode is directly move into kernel function rather than using function call.

Before optimization, fetch data from windowFuction->m_data.
Optimization #2

Another optimization in BeamformingNode is directly move into kernel function rather than using function call.

```cpp
auto mdataGpu = std::make_shared<
    Container<float, ContainerLocation::LocationGpu, gRawData->getStream(), m_windowFunction->m_data>;
```

After optimization, fetch data from mdataGpu, mdataGpu was directly pass to kernel function. For data copy, change m_data in WindowFunctionGpu to public.

Source code: supra/src/SupraLib/Beamformer/RxSampleBeamformerDelayAndSum.h

```cpp
for (int i = 0; i < VEC_SIZE; i += 2) {
    weight[i] = mdataGpu[absoluteIndex_int[i]]; 
    weight[i + 1] = mdataGpu[absoluteIndex_int[i + 1]]; 
}
```

After optimization, fetch data from mdataGpu, mdataGpu was directly pass to kernel function.
Optimization #2

Before optimization

After optimization
Using ESIMD to optimize beamforming

Original Beamforming implementation on GPU

Optimized Beamforming implementation on GPU
Beamforming implementation on FPGA
## Supra on Intel FPGA Arria 10

<table>
<thead>
<tr>
<th>SUPRA Node</th>
<th>oneAPI (ms) UHD630</th>
<th>oneAPI (ms) Arria 10</th>
</tr>
</thead>
<tbody>
<tr>
<td>RxBeamforming</td>
<td>9.24</td>
<td>5.94↓ (Max)</td>
</tr>
<tr>
<td>HilbertFirEnvelope</td>
<td>1.50</td>
<td>2.61</td>
</tr>
<tr>
<td>LogCompressor</td>
<td>0.27</td>
<td>0.34</td>
</tr>
<tr>
<td>ScanConverter</td>
<td>2.65</td>
<td>5.66</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>13.66</strong></td>
<td></td>
</tr>
</tbody>
</table>

SUPRA on FPGA has been tested on DevCloud, a Jupyter notebook provided to quick build and run.

Link: https://gitlab.devtools.intel.com/qwang12/ultrasound-emu/-/tree/intelfpga_beta10
Beamforming on the FPGA

Original pre-beamformed data store pattern

Shuffled pre-beamformed data store pattern
Beamforming on the FPGA

Beamforming implementation on FPGA

FPGA code: intelfpga-devcloud-golden/SupraLib/Beamformer/ RxBeamformerCuda.dp.cpp
oneAPI provides high level language (DPC++) to programming FPGA, which is more flexible, easy to learn, easy to develop, easy to debug.

To use oneAPI programming for FPGA, Professional knowledge of FPGA is required.
Results and Performance
SUPRA GUI and DevCloud usage
Intel DevCloud usage – FPGA

Intel DevCloud: https://devcloud.intel.com/oneapi/

Welcome to Jupyter Notebooks on the Intel DevCloud for SUPRA-on-oneAPI-FPGA Project

This document contains the process of using Intel® oneAPI Base Toolkit build and run SUPRA on Intel FPGA.

Table of Contents

1. SUPRA introduction
2. Build steps
   2.1 Download Ultrasound data and unzip files

1. SUPRA introduction

The SUPRA is an open source project. You can find detailed information at SUPRA.

2. Build steps

2.1 Download Ultrasound data and unzip files

```python
import wget
ultrasound_data_url = 'http://cavear.in.tum.de/Files/gnmrlr/nekDataLinearProbe.zip'
config_file_url = 'https://github.com/IVL-CMM/supra/raw/master/config/ConfigHomo.xml'
wget -O data
wget -O config
```
## SUPRA performance on Intel hardware

<table>
<thead>
<tr>
<th>SUPRA Node</th>
<th>oneAPI (ms) – UHD630</th>
<th>Tiger lake Iris Xe</th>
<th>oneAPI - DG1(WA)</th>
<th>oneAPI – Arria 10</th>
</tr>
</thead>
<tbody>
<tr>
<td>RxBeamforming</td>
<td>9.24</td>
<td>4.36</td>
<td>3.81</td>
<td>5.94</td>
</tr>
<tr>
<td>HilbertFirEnvelope</td>
<td>1.50</td>
<td>0.73</td>
<td>0.65</td>
<td>2.61</td>
</tr>
<tr>
<td>LogCompressor</td>
<td>0.27</td>
<td>0.1</td>
<td>0.08</td>
<td>0.34</td>
</tr>
<tr>
<td>ScanConverter</td>
<td>2.65</td>
<td>2.22</td>
<td>1.14</td>
<td>5.66</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>13.66</strong></td>
<td><strong>7.41</strong></td>
<td><strong>5.68</strong></td>
<td><strong>5.94(max)</strong></td>
</tr>
</tbody>
</table>

For the source code, please refer to: https://github.com/intel/supra-on-oneapi

For other vendors hardware performance, please refer to: https://doi.org/10.1007/s11548-018-1750-6
Summary

- Unified programing framework/language to implement medical algorithm accelerations on Intel HW
- Samples to implement Ultrasound beamforming on Intel xGPU
- Samples to implement Ultrasound beamforming on Intel FPGA
- Possibility to integrate algorithm acceleration and AI inference on a heterogenous compute system (Intel oneAPI and OpenVINO)
- Future Intel acceleration hardware (xPU) support
Thanks for your time!