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

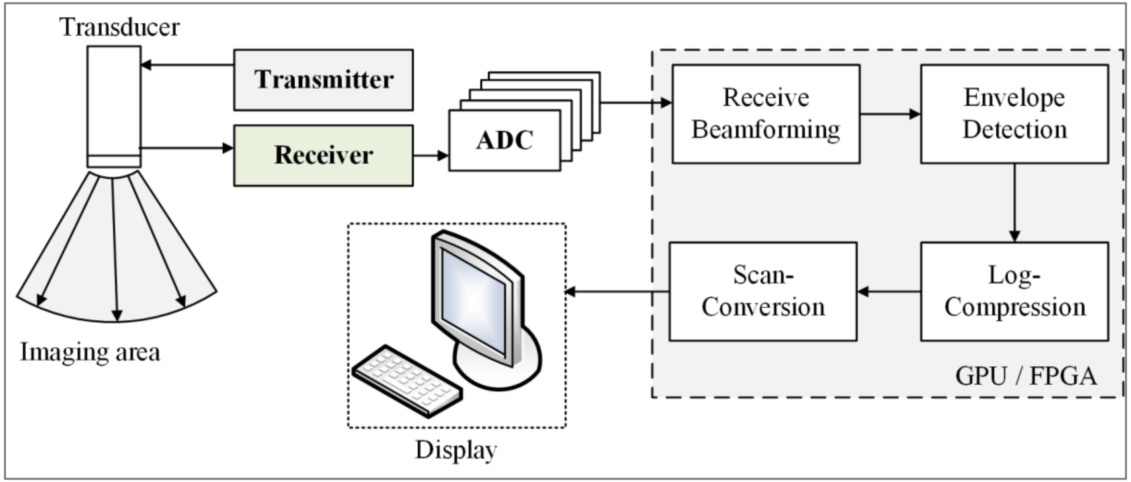
- Background
- Code Migration
- Beamforming Optimization on GPU
- Beamforming implementation on FPGA
- Results and performance

Background

What is SUPRA and why we need it?

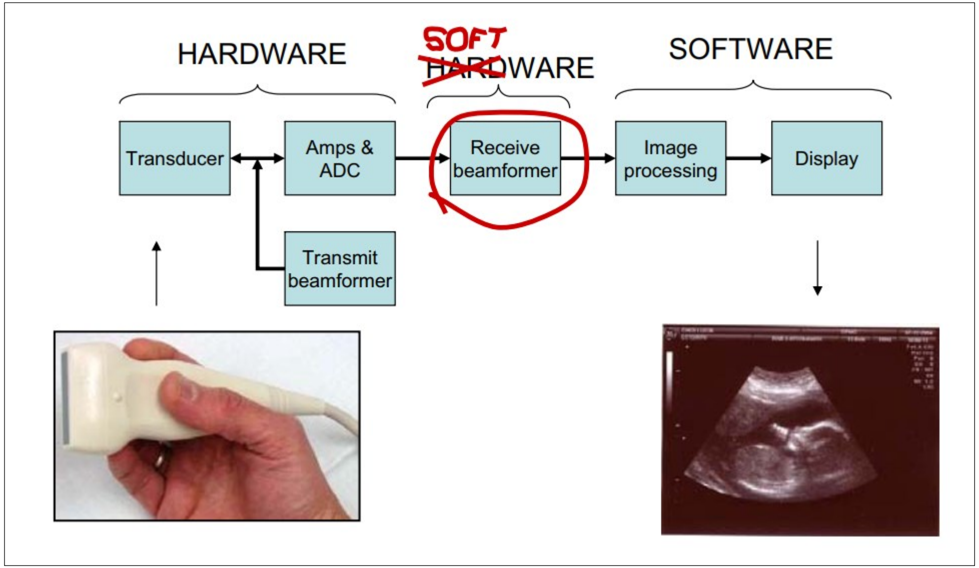
SUPRA is an open-source pipeline for fully software defined ultrasound processing.

- https://github.com/IFL-CAMP/supra



What is software beamforming?

SUPRA contains standard medical ultrasound software beamforming algorithms.



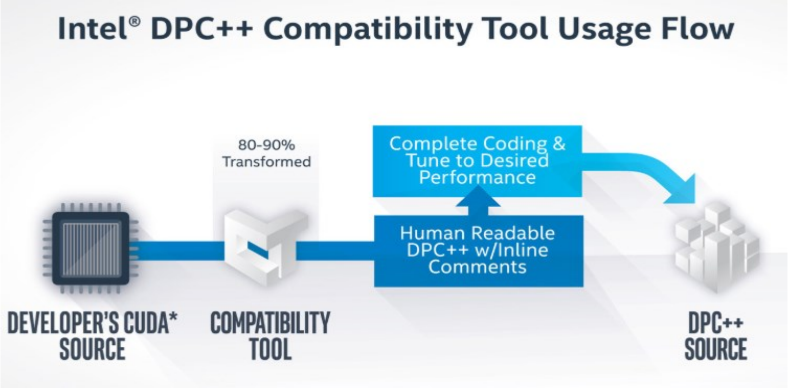
Software beamforming illustration.(Fig source: Lars Grønvold)

Intel's products in software beamforming :

- Core & Gen9 graphics, DG1, Arria 10 & Stratix 10, Intel oneAPI.

Code Migration

migration flow


Intel® DPC++ Compatibility Tool Usage Flow
80-90% Transformed
Complete Coding & Tune to Desired Performance
Human Readable DPC++ w/Inline Comments
DEVELOPER'S CUDA* SOURCE
COMPATIBILITY TOOL
DPC++ SOURCE

migration flow

- 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

OneAPI Version	Total num of migration place	Success migrated	Need modify	Accuracy
beta07	84	63	21	75%
golden	84	75	9	89%

SUPRA migration summary

File Type	*.cpp	*.cu	*.h
File num	1	4	23

num. of migrated file

Migration Command:

```
dpct --in-root=./ --out-root=./oneapi  --extra-arg=-Isrc/SupraLib --extra-arg=-Isrc/SupraLib/Beamformer --extra-arg=-Isrc/SupraLib/utilities  --extra-arg=-std=c++11 --extra-arg=-Wno-c++11-narrowing  --extra-arg=-DHAVE_CUDA ./src/SupraLib/Beamformer/ScanConverter.cu ./src/SupraLib/Beamformer/HilbertFirEnvelope.cu ./src/SupraLib/Beamformer/LogCompressor.cu ./src/SupraLib/Beamformer/RxBeamformerCuda.cu ./src/SupraLib/ContainerFactory.cpp
```

Migrated code APIs

Category	oneAPI APIs

Memory Management	sycl::malloc_device()
	sycl::malloc_shared()
	sycl::free()
	sycl::malloc_host()
	sycl::queue.memcpy()
	Sycl::queue.memset()

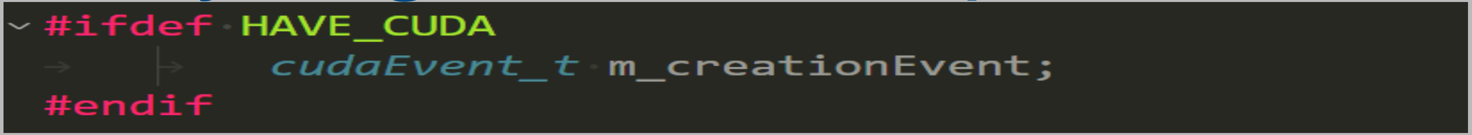
sycl::queue	dpct::get_current_device().create_queue()
	dpct::get_default_queue()
	sycl::queue()
	sycl::queue.submit()
	sycl::queue.wait()

Math	sycl::sqrt(); sycl::floor(); sycl::fabs(); sycl::round()
	sycl::max(); sycl::min(); sycl::log10(); sycl::pow()

Express Parallel	sycl::nd_item<>; Sycl::nd_range<>; Sycl::range<>
	sycl::id<>; Sycl::nd_item().get_local_range()
	sycl::nd_item().get_group()
	sycl::nd_item().get_local_id()

Manually migration example

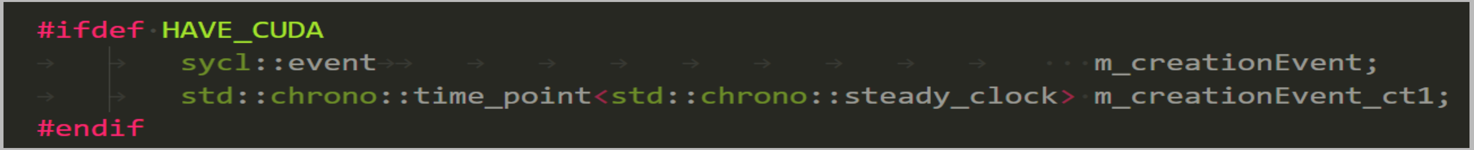
cuda



```
#ifdef HAVE_CUDA
        cudaEvent_t m_creationEvent;
#endif
```

1.

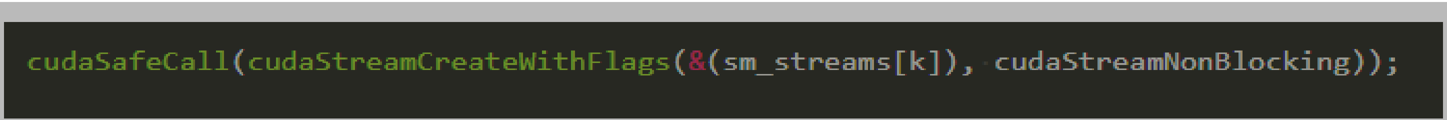
migrate



```
#ifdef HAVE_CUDA
        sycl::event                                              m_creationEvent;
        std::chrono::time_point<std::chrono::steady_clock> m_creationEvent_ct1;
#endif
```

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

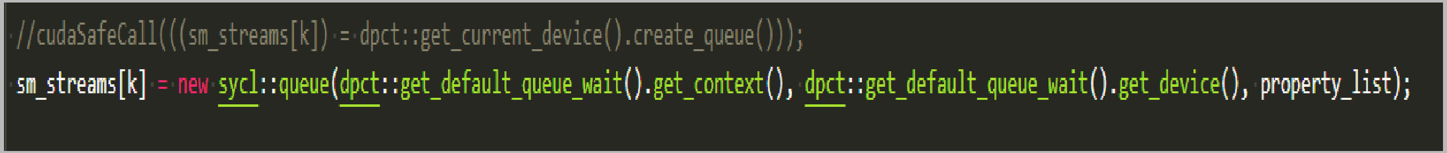
cuda



```
cudaSafeCall(cudaStreamCreateWithFlags(&(sm_streams[k]), cudaStreamNonBlocking));
```

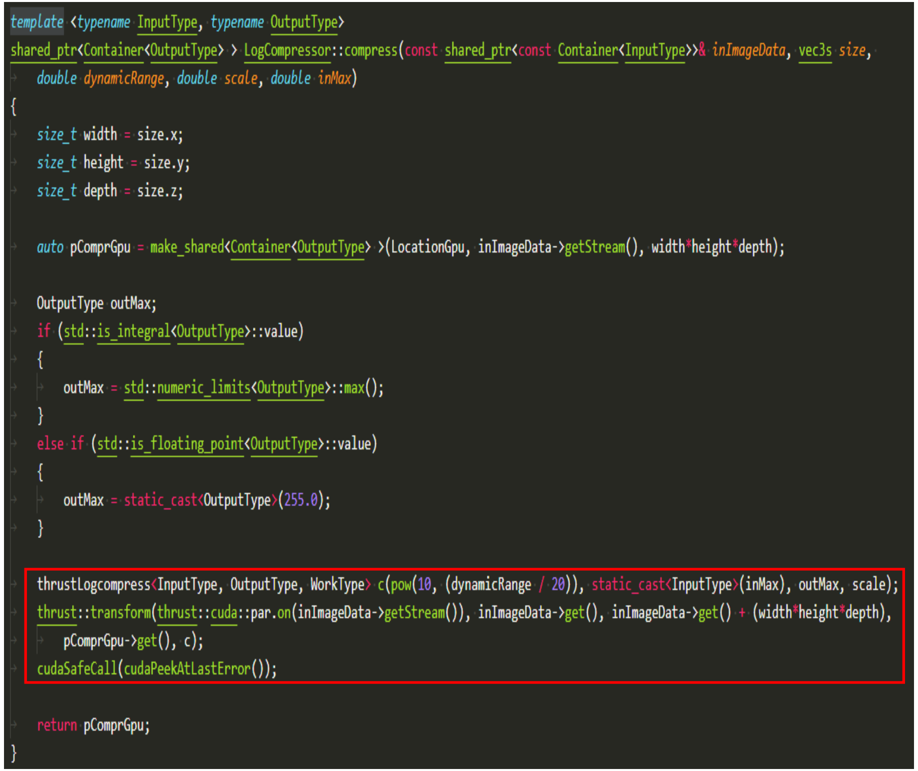
2. migrate

modify

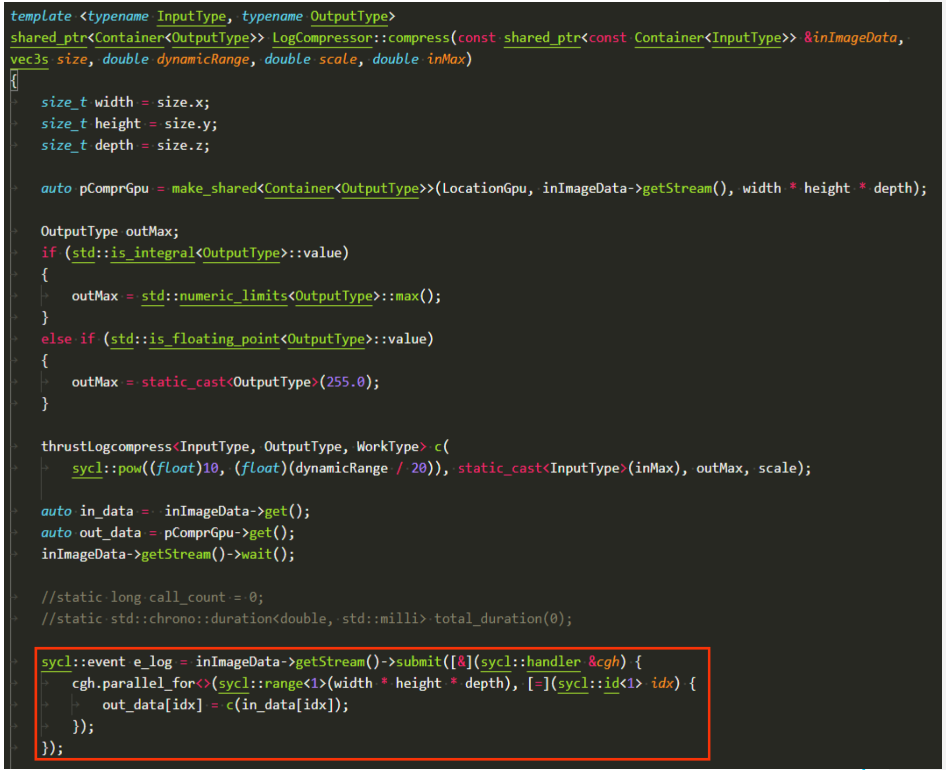


```
//cudaSafeCall(((sm_streams[k]) = dpct::get_current_device().create_queue()));
sm_streams[k] = new sycl::queue(dpct::get_default_queue_wait().get_context(), dpct::get_default_queue_wait().get_device(), property_list);
```

Manually migration example



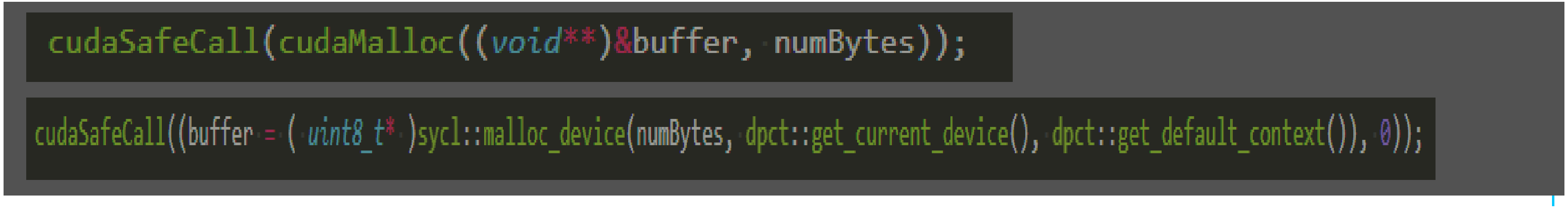
```
template <typename InputType, typename OutputType>
shared_ptr<Container<OutputType> > LogCompressor::compress(const shared_ptr<const Container<InputType>>& inImageData, vec3s size,
    double dynamicRange, double scale, double inMax)
{
    size_t width = size.x;
    size_t height = size.y;
    size_t depth = size.z;

    auto pComprGpu = make_shared<Container<OutputType> >(LocationGpu, inImageData->getStream(), width*height*depth);

    OutputType outMax;
    if (std::is_integral<OutputType>::value)
    {
        outMax = std::numeric_limits<OutputType>::max();
    }
    else if (std::is_floating_point<OutputType>::value)
    {
        outMax = static_cast<OutputType>(255.0);
    }

    thrustLogcompress<InputType, OutputType, WorkType> c(pow(10, (dynamicRange / 20)), static_cast<InputType>(inMax), outMax, scale);
    thrust::transform(thrust::cuda::par.on(inImageData->getStream()), inImageData->get(), inImageData->get() + (width*height*depth),
        pComprGpu->get(), c);
    cudaSafeCall(cudaPeekAtLastError());

    return pComprGpu;
}
```



```
template <typename InputType, typename OutputType>
shared_ptr<Container<OutputType>> LogCompressor::compress(const shared_ptr<const Container<InputType>> &inImageData,
vec3s size, double dynamicRange, double scale, double inMax)
{
    size_t width = size.x;
    size_t height = size.y;
    size_t depth = size.z;

    auto pComprGpu = make_shared<Container<OutputType>>(LocationGpu, inImageData->getStream(), width * height * depth);

    OutputType outMax;
    if (std::is_integral<OutputType>::value)
    {
        outMax = std::numeric_limits<OutputType>::max();
    }
    else if (std::is_floating_point<OutputType>::value)
    {
        outMax = static_cast<OutputType>(255.0);
    }

    thrustLogcompress<InputType, OutputType, WorkType> c(
        sycl::pow((float)10, (float)(dynamicRange / 20)), static_cast<InputType>(inMax), outMax, scale);

    auto in_data =  inImageData->get();
    auto out_data = pComprGpu->get();
    inImageData->getStream()->wait();

    //static long call_count = 0;
    //static std::chrono::duration<double, std::milli> total_duration(0);

    sycl::event e_log = inImageData->getStream()->submit([&](sycl::handler &cgh) {
        cgh.parallel_for<>(sycl::range<1>(width * height * depth), [=](sycl::id<1> idx) {
            out_data[idx] = c(in_data[idx]);
        });
    });
```

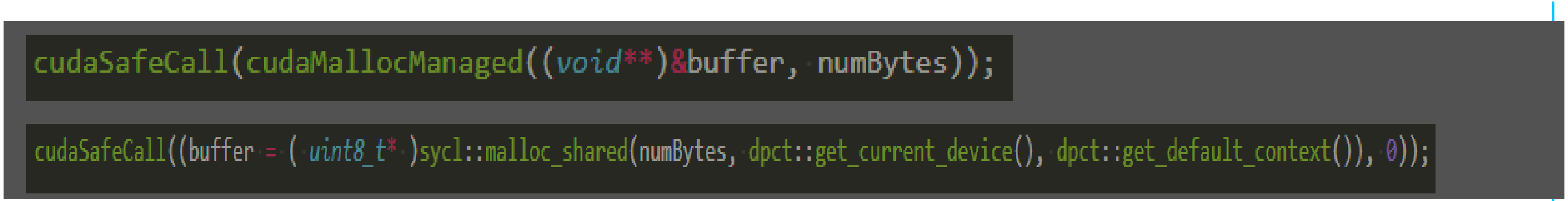
DPCT tool can’t migrate CUDA thrust library related code, so it must be rewritten using oneAPI model.

Migration success example



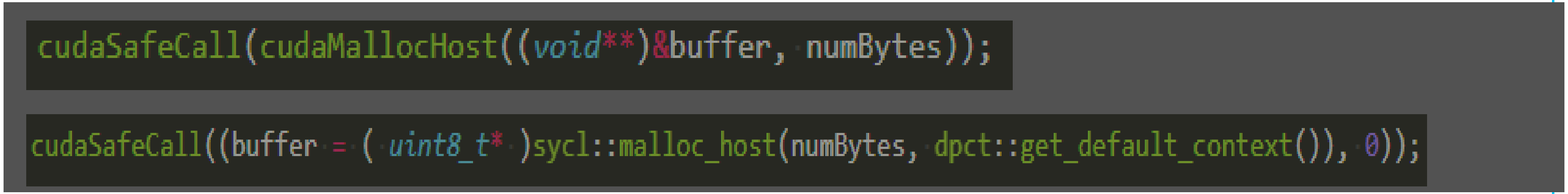
```
cudaSafeCall(cudaMalloc((void**)&buffer, numBytes));
```

```
cudaSafeCall((buffer = ( uint8_t* )sycl::malloc_device(numBytes, dpct::get_current_device(), dpct::get_default_context()), 0));
```



```
cudaSafeCall(cudaMallocManaged((void**)&buffer, numBytes));
```

```
cudaSafeCall((buffer = ( uint8_t* )sycl::malloc_shared(numBytes, dpct::get_current_device(), dpct::get_default_context()), 0));
```



```
cudaSafeCall(cudaMallocHost((void**)&buffer, numBytes));
```

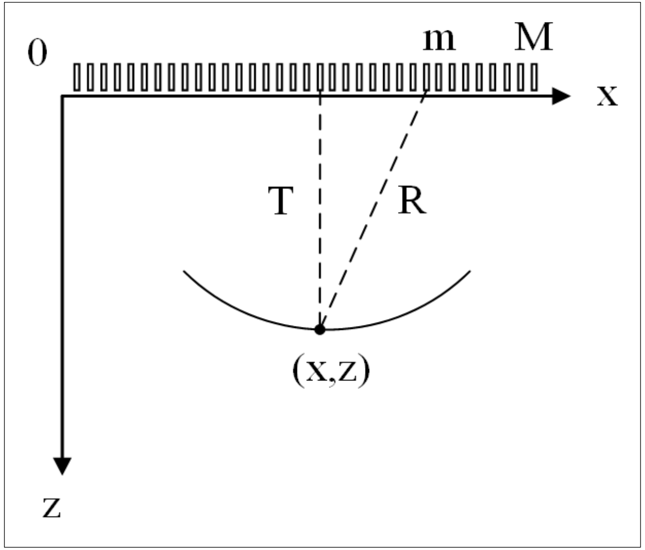
```
cudaSafeCall((buffer = ( uint8_t* )sycl::malloc_host(numBytes, dpct::get_default_context()), 0));
```

Memory allocate related function were successfully migrated.

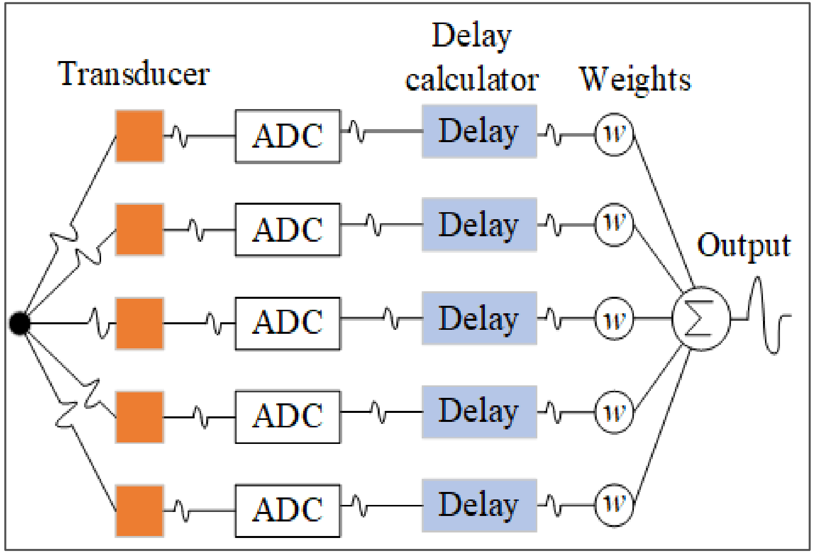
Table 3 migrated CUDA API summary

Beamforming Optimization on GPU

Beamforming(Delay and Sum) introduction



Geometrical illustration of the pulse-echo process



Delay and sum algorithm illustration

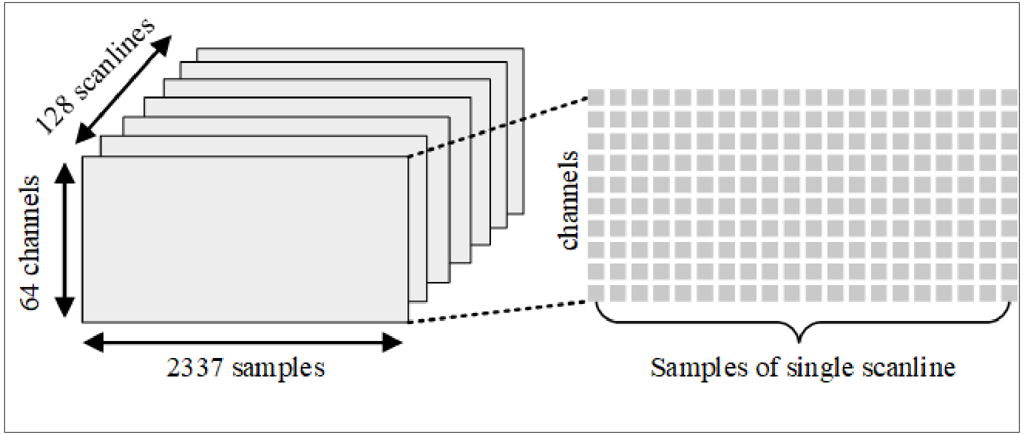
$$\Delta t \ = (T + R)/c_0$$

c_0: the speed of ultrasound travel in the body.

Beamforming introduction

Suppose pre-beamformed data: 128 scanlines. 64 channels. 2337 samples. It arranged in a 3-D data structure: rf_data[scanline][channel][sample].

Memory required to store pre-beamformed data per image frame is:
(128 * 64 * 2334 * 2) bytes = 36.5 MB



Pre-beamforming data store pattern

Beamforming algorithm implementation in single thread

Input: 3 dimensional pre-beamformed rf_data[numScanline][numSample][numChannel].

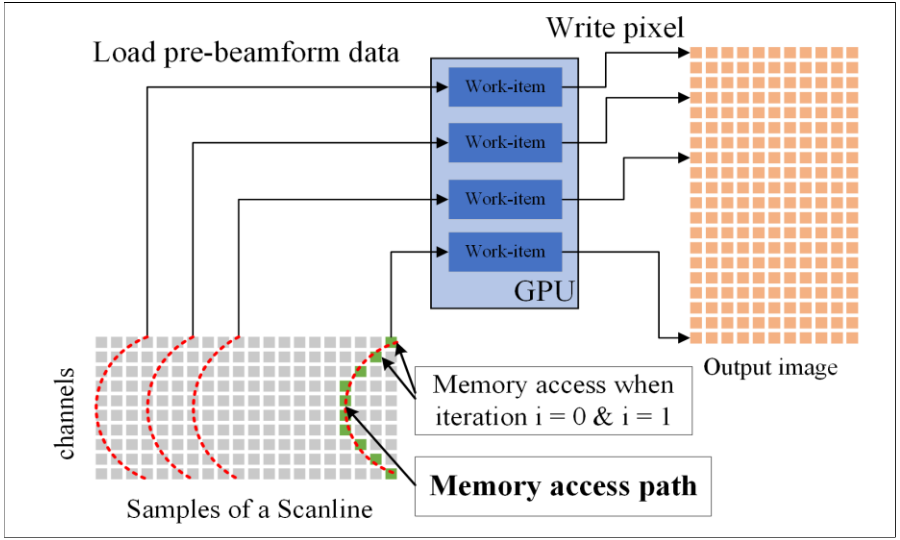
Output: A single frame image.

```
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

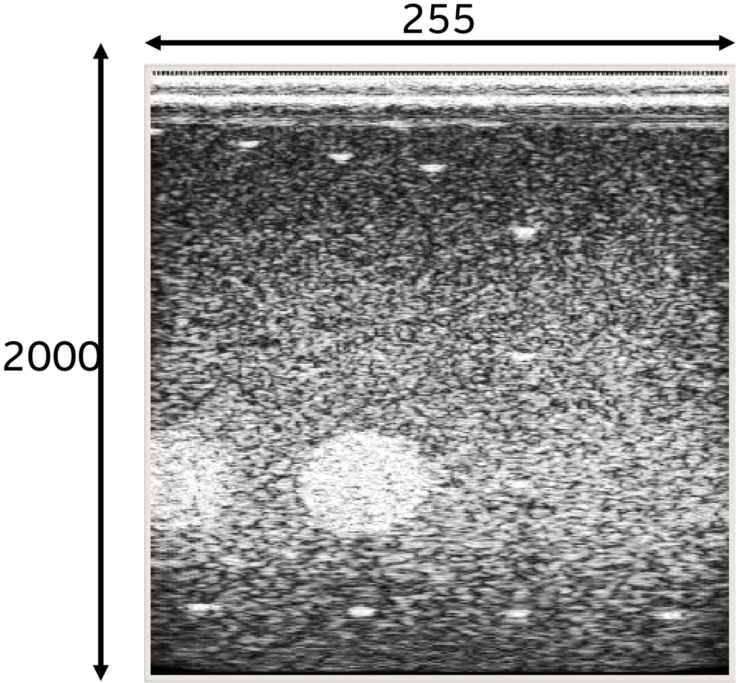


Original Beamforming implementation on GPU

Optimization #1

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

Function rxBeamformingDTSPACEKernel and sampleBeamfor2D are optimized.



Optimization idea:

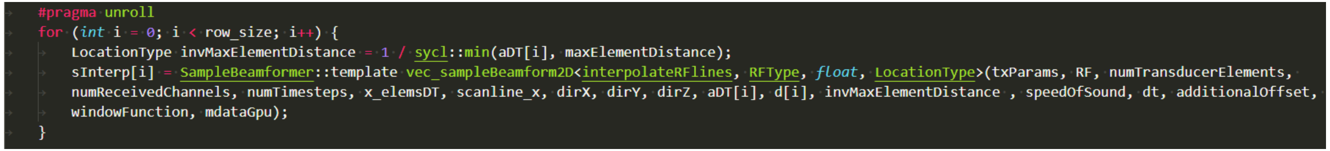
CUDA： Each thread calculates a point; every point iterates 64 times.

oneAPI: each thread load 2 points in vertical direction, Iterates 8 times.

Optimization #1

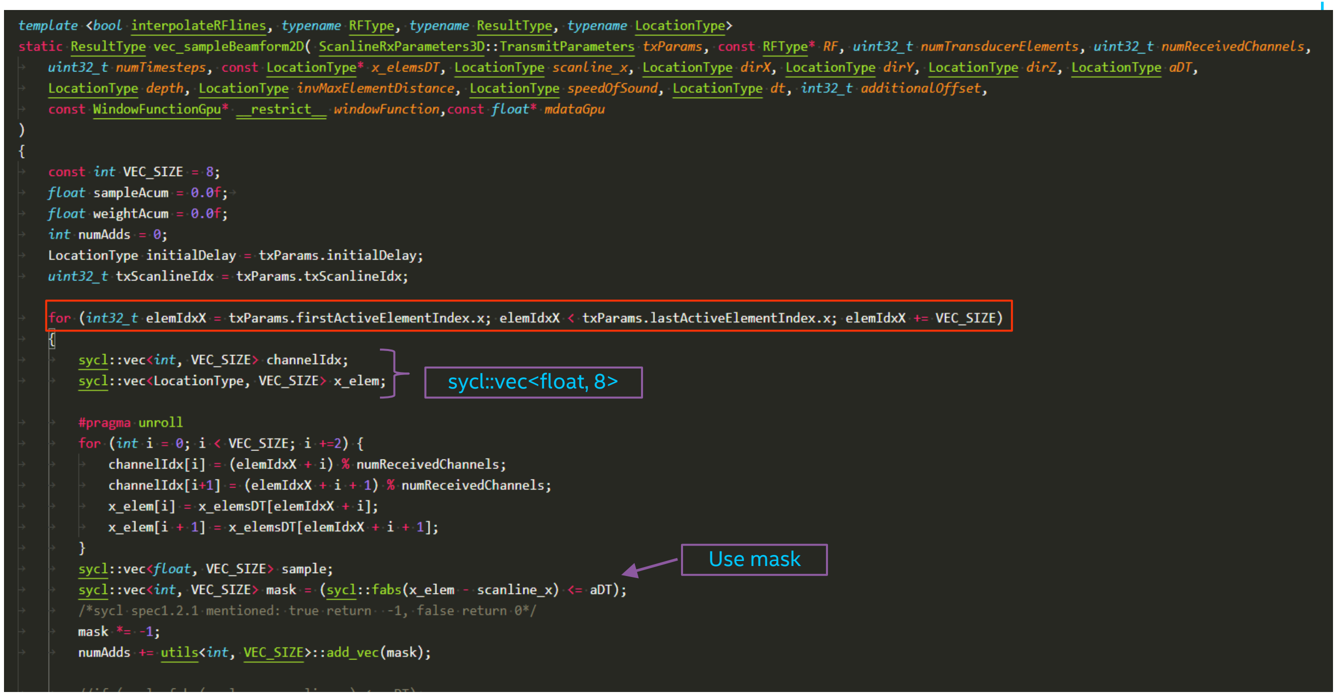
oneAPI code

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



```
#pragma unroll
for (int i = 0; i < row_size; i++) {
    LocationType invMaxElementDistance = 1 / sycl::min(aDT[i], maxElementDistance);
    sInterp[i] = SampleBeamformer::template vec_sampleBeamform2D<interpolateRFlines, RFType, float, LocationType>(txParams, RF, numTransducerElements,
    numReceivedChannels, numTimesteps, x_elemsDT, scanline_x, dirX, dirY, dirZ, aDT[i], d[i], invMaxElementDistance , speedOfSound, dt, additionalOffset,
    windowFunction, mdataGpu);
}
```

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

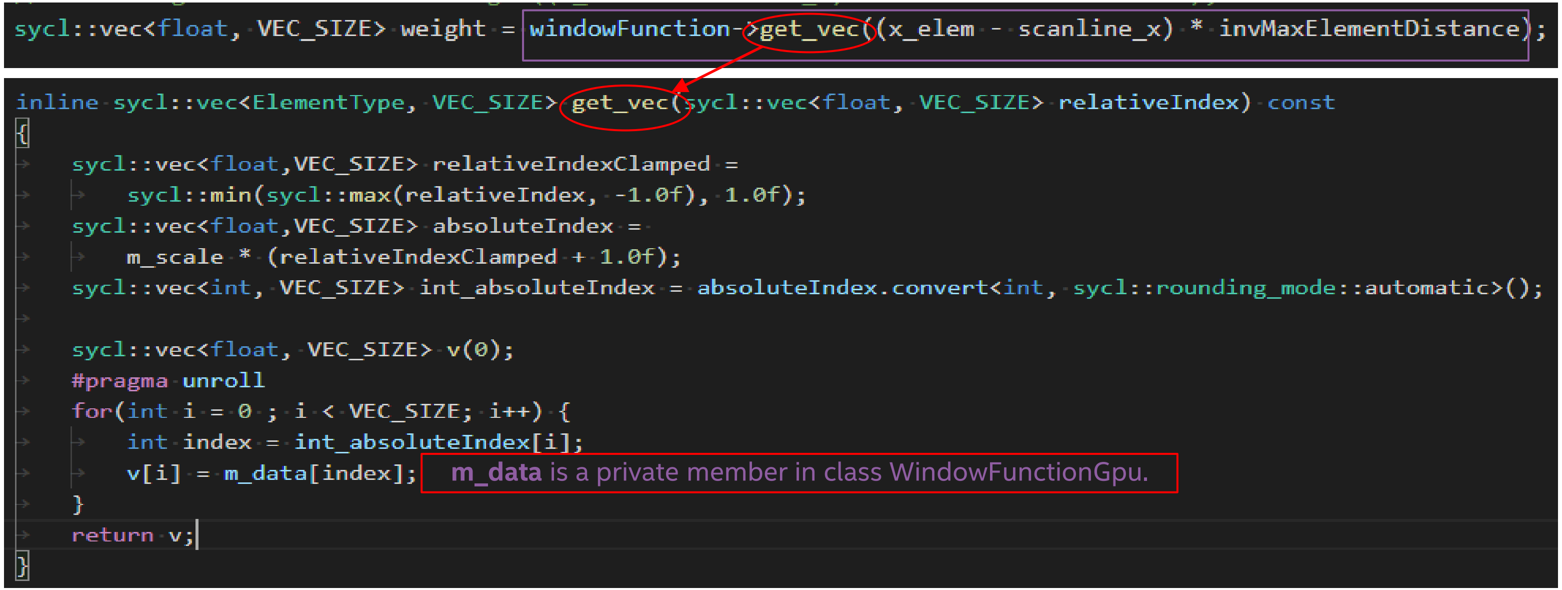


```
template <bool interpolateRFlines, typename RFType, typename ResultType, typename LocationType>
static ResultType vec_sampleBeamform2D( ScanlineRxParameters3D::TransmitParameters txParams, const RFType* RF, uint32_t numTransducerElements, uint32_t numReceivedChannels,
    uint32_t numTimesteps, const LocationType* x_elemsDT, LocationType scanline_x, LocationType dirX, LocationType dirY, LocationType dirZ, LocationType aDT,
    LocationType depth, LocationType invMaxElementDistance, LocationType speedOfSound, LocationType dt, int32_t additionalOffset,
    const WindowFunctionGpu* __restrict__ windowFunction,const float* mdataGpu
)
{
    const int VEC_SIZE = 8;
    float sampleAcum = 0.0f;
    float weightAcum = 0.0f;
    int numAdds = 0;
    LocationType initialDelay = txParams.initialDelay;
    uint32_t txScanlineIdx = txParams.txScanlineIdx;

    for (int32_t elemIdxX = txParams.firstActiveElementIndex.x; elemIdxX < txParams.lastActiveElementIndex.x; elemIdxX += VEC_SIZE)
    {
        sycl::vec<int, VEC_SIZE> channelIdx;
        sycl::vec<LocationType, VEC_SIZE> x_elem;

        #pragma unroll
        for (int i = 0; i < VEC_SIZE; i +=2) {
            channelIdx[i] = (elemIdxX + i) % numReceivedChannels;
            channelIdx[i+1] = (elemIdxX + i + 1) % numReceivedChannels;
            x_elem[i] = x_elemsDT[elemIdxX + i];
            x_elem[i + 1] = x_elemsDT[elemIdxX + i + 1];
        }
        sycl::vec<float, VEC_SIZE> sample;
        sycl::vec<int, VEC_SIZE> mask = (sycl::fabs(x_elem - scanline_x) <= aDT);
        /*sycl spec1.2.1 mentioned: true return  -1, false return 0*/
        mask *= -1;
        numAdds += utils<int, VEC_SIZE>::add_vec(mask);
```

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.



```
sycl::vec<float, VEC_SIZE> weight = windowFunction->get_vec((x_elem - scanline_x) * invMaxElementDistance);
```

```
inline sycl::vec<ElementType, VEC_SIZE> get_vec(sycl::vec<float, VEC_SIZE> relativeIndex) const
{
    sycl::vec<float,VEC_SIZE> relativeIndexClamped =
        sycl::min(sycl::max(relativeIndex, -1.0f), 1.0f);
    sycl::vec<float,VEC_SIZE> absoluteIndex =
        m_scale * (relativeIndexClamped + 1.0f);
    sycl::vec<int, VEC_SIZE> int_absoluteIndex = absoluteIndex.convert<int, sycl::rounding_mode::automatic>();

    sycl::vec<float, VEC_SIZE> v(0);
    #pragma unroll
    for(int i = 0 ; i < VEC_SIZE; i++) {
        int index = int_absoluteIndex[i];
        v[i] = m_data[index];
    }
    return v;
}
```

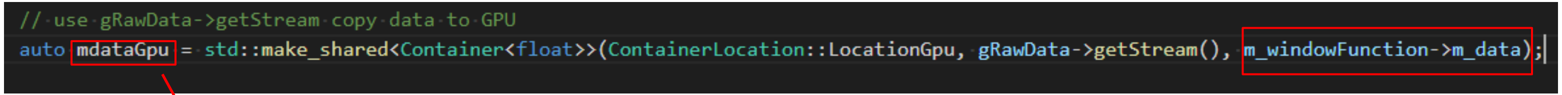
m_data is a private member in class WindowFunctionGpu.

Source code: supra/src/SupraLib/Beamformer/WindowFunction.cpp

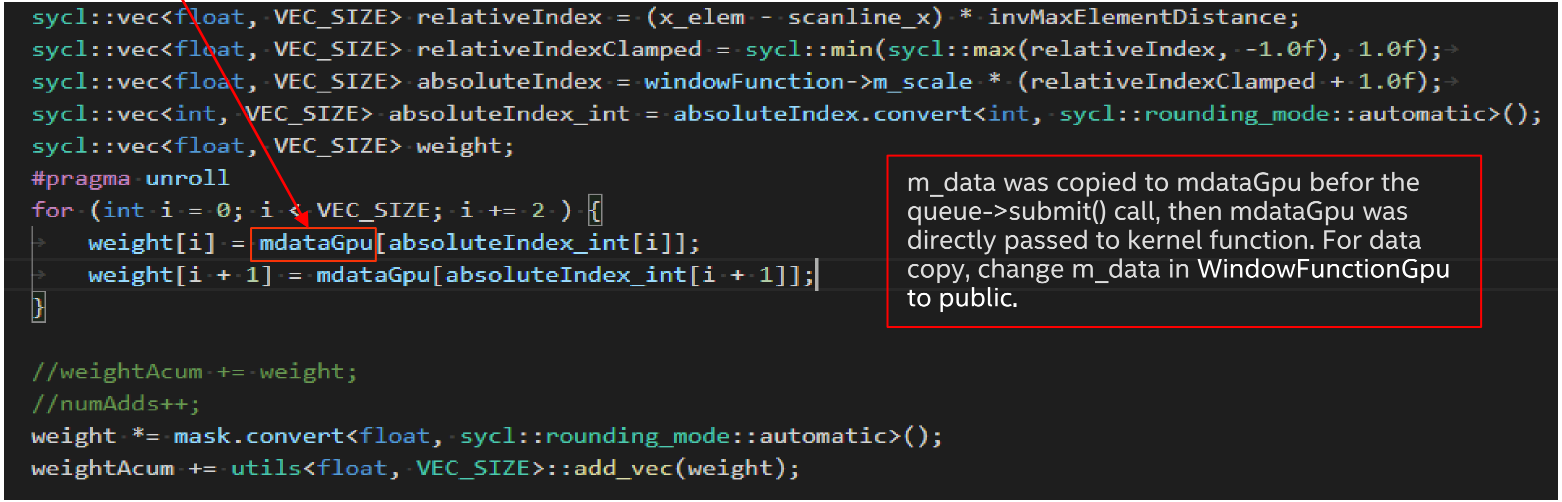
Before optimization, fetch data from windowFcuntion->m_data.

Optimization #2

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



```
// use gRawData->getStream copy data to GPU
auto mdataGpu = std::make_shared<Container<float>>(ContainerLocation::LocationGpu, gRawData->getStream(), m_windowFunction->m_data);
```



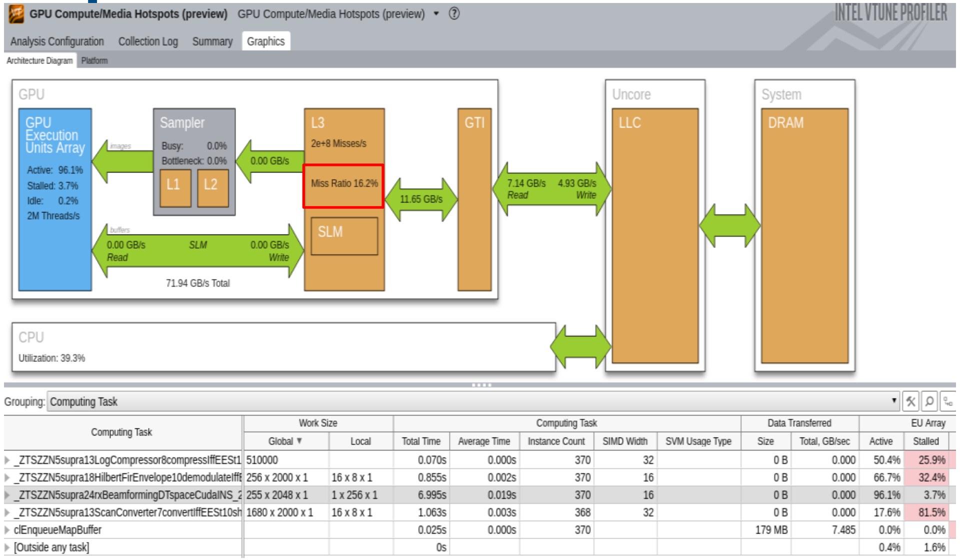
```
sycl::vec<float, VEC_SIZE> relativeIndex = (x_elem - scanline_x) * invMaxElementDistance;
sycl::vec<float, VEC_SIZE> relativeIndexClamped = sycl::min(sycl::max(relativeIndex, -1.0f), 1.0f);
sycl::vec<float, VEC_SIZE> absoluteIndex = windowFunction->m_scale * (relativeIndexClamped + 1.0f);
sycl::vec<int, VEC_SIZE> absoluteIndex_int = absoluteIndex.convert<int, sycl::rounding_mode::automatic>();
sycl::vec<float, VEC_SIZE> weight;
#pragma unroll
for (int i = 0; i < VEC_SIZE; i += 2 ) {
    weight[i] = mdataGpu[absoluteIndex_int[i]];
    weight[i + 1] = mdataGpu[absoluteIndex_int[i + 1]];
}

//weightAcum += weight;
//numAdds++;
weight *= mask.convert<float, sycl::rounding_mode::automatic>();
weightAcum += utils<float, VEC_SIZE>::add_vec(weight);
```

m_data was copied to mdataGpu befor the queue->submit() call, then mdataGpu was directly passed to kernel function. For data copy, change m_data in WindowFunctionGpu to public.

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

After optimization, fetch data from mdataGpu, mdataGpu was directly pass to kernel function

Optimization #2

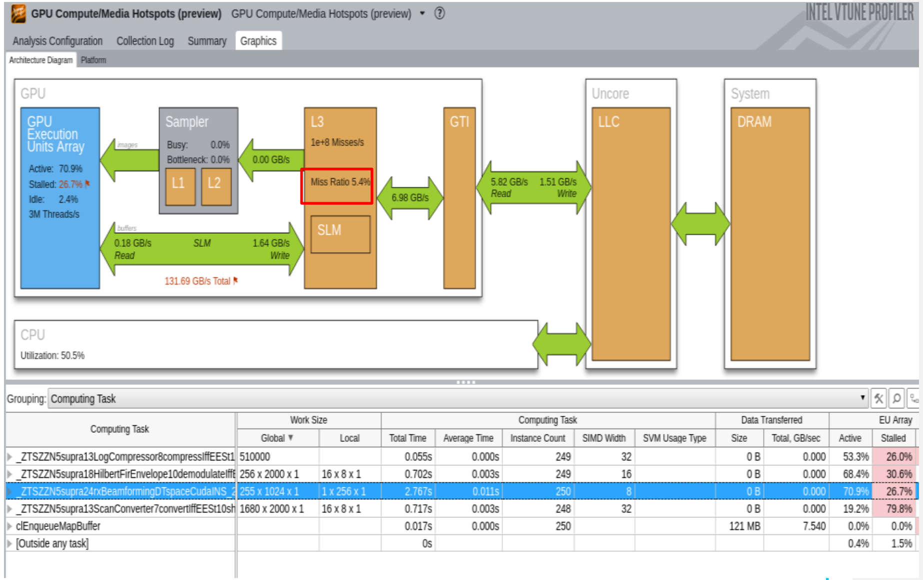


Grouping: Computing Task

Computing Task	Work Size		Computing Task					Data Transferred		EU Array	
	Global ▼	Local	Total Time	Average Time	Instance Count	SIMD Width	SVM Usage Type	Size	Total, GB/sec	Active	Stalled
_ZTSZZN5supra13LogCompressor8compressIffEESt1	510000		0.070s	0.000s	370	32		0 B	0.000	50.4%	25.9%
_ZTSZZN5supra18HilbertFirEnvelope10demodulateIffE	256 x 2000 x 1	16 x 8 x 1	0.855s	0.002s	370	16		0 B	0.000	66.7%	32.4%
_ZTSZZN5supra24rxBeamformingDTspaceCudaINS_2	255 x 2048 x 1	1 x 256 x 1	6.995s	0.019s	370	16		0 B	0.000	96.1%	3.7%
_ZTSZZN5supra13ScanConverter7convertIffEESt10sh	1680 x 2000 x 1	16 x 8 x 1	1.063s	0.003s	368	32		0 B	0.000	17.6%	81.5%
clEnqueueMapBuffer			0.025s	0.000s	370			179 MB	7.485	0.0%	0.0%
[Outside any task]			0s							0.4%	1.6%

Before optimization

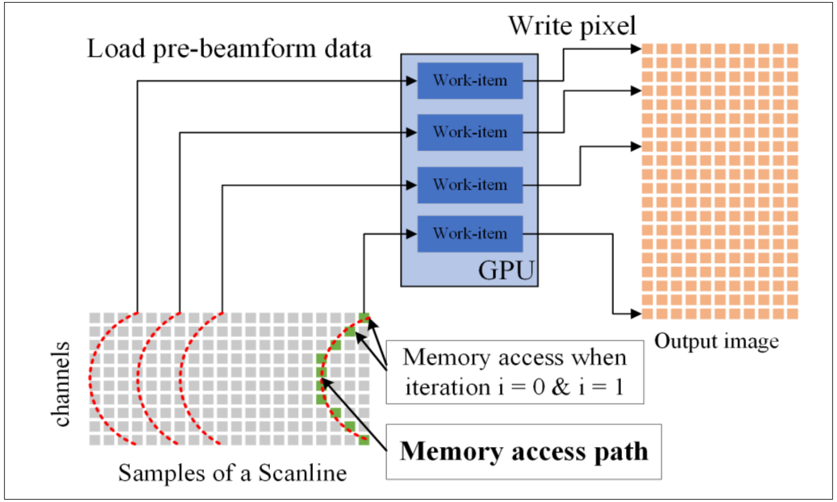
After optimization



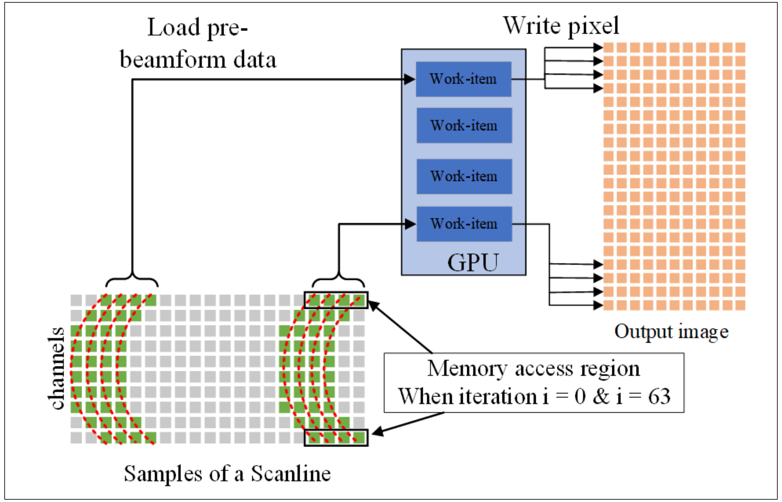
Grouping: Computing Task

Computing Task	Work Size		Computing Task					Data Transferred		EU Array	
	Global ▼	Local	Total Time	Average Time	Instance Count	SIMD Width	SVM Usage Type	Size	Total, GB/sec	Active	Stalled
_ZTSZZN5supra13LogCompressor8compressIffEESt1	510000		0.055s	0.000s	249	32		0 B	0.000	53.3%	26.0%
_ZTSZZN5supra18HilbertFirEnvelope10demodulateIffE	256 x 2000 x 1	16 x 8 x 1	0.702s	0.003s	249	16		0 B	0.000	68.4%	30.6%
_ZTSZZN5supra24rxBeamformingDTspaceCudaINS_2	255 x 1024 x 1	1 x 256 x 1	2.767s	0.011s	250	8		0 B	0.000	70.9%	26.7%
_ZTSZZN5supra13ScanConverter7convertIffEESt10sh	1680 x 2000 x 1	16 x 8 x 1	0.717s	0.003s	248	32		0 B	0.000	19.2%	79.8%
clEnqueueMapBuffer			0.017s	0.000s	250			121 MB	7.540	0.0%	0.0%
[Outside any task]			0s							0.4%	1.5%

Using ESIMD to optimize beamforming



Original Beamforming implementation on GPU



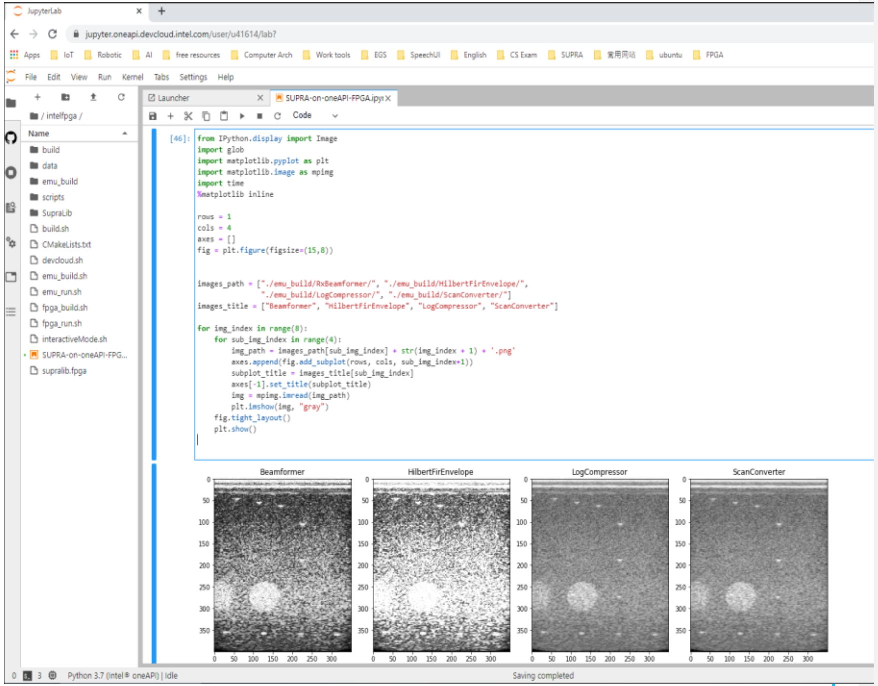
Optimized Beamforming implementation on GPU

Beamforming implementation on FPGA

Supra on Intel FPGA Arria 10

SUPRA Node	oneAPI (ms) UHD630	oneAPI (ms) Arria 10
RxBeamforming	9.24	5.94↓ (Max)
HilbertFirEnvelope	1.50	2.61
LogCompressor	0.27	0.34
ScanConverter	2.65	5.66
Total	13.66	

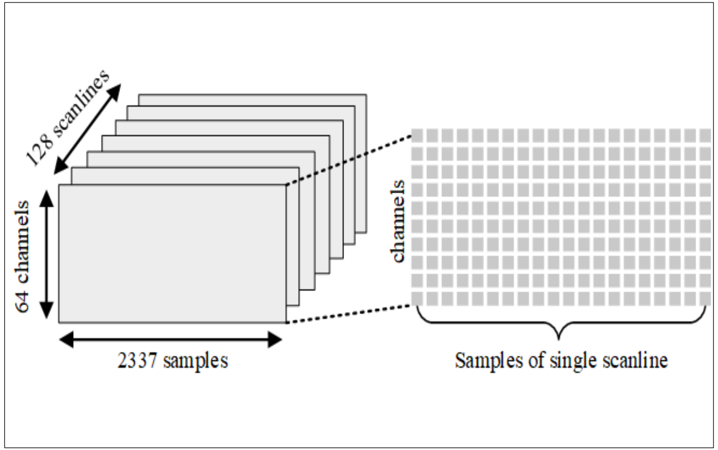
SUPRA on FPGA has been tested on DevCloud, a Jupyter notebook provided to quick build and run.



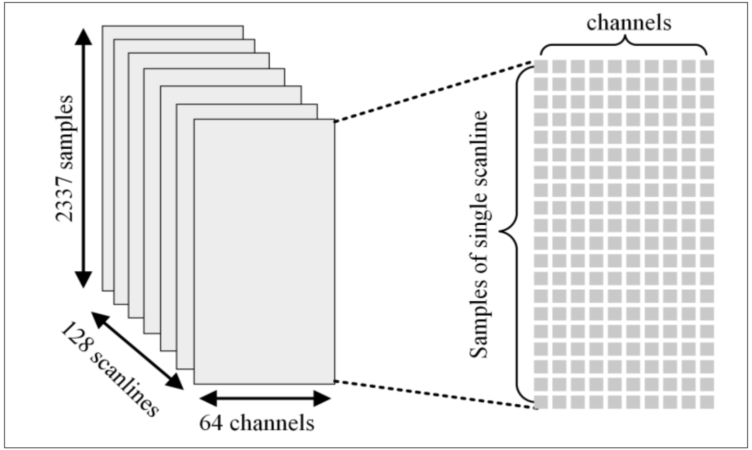
```
from IPython.display import Image
import glob
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import time
%matplotlib inline

rows = 1
cols = 4
axes = []
fig = plt.figure(figsize=(15,8))


images_path = ["./emu_build/RxBeamformer/", "./emu_build/HilbertFirEnvelope/",
               "./emu_build/LogCompressor/", "./emu_build/ScanConverter/"]
images_title = ["Beamformer", "HilbertFirEnvelope", "LogCompressor", "ScanConverter"]

for img_index in range(8):
    for sub_img_index in range(4):
        img_path = images_path[sub_img_index] + str(img_index + 1) + '.png'
        axes.append(fig.add_subplot(rows, cols, sub_img_index+1))
        subplot_title = images_title[sub_img_index]
        axes[-1].set_title(subplot_title)
        img = mpimg.imread(img_path)
        plt.imshow(img, "gray")
    fig.tight_layout()
    plt.show()
```

Link: https://gitlab.devtools.intel.com/qwang12/ultrasound-emu/-/tree/intelfpga_beta10

Beamforming on the FPGA

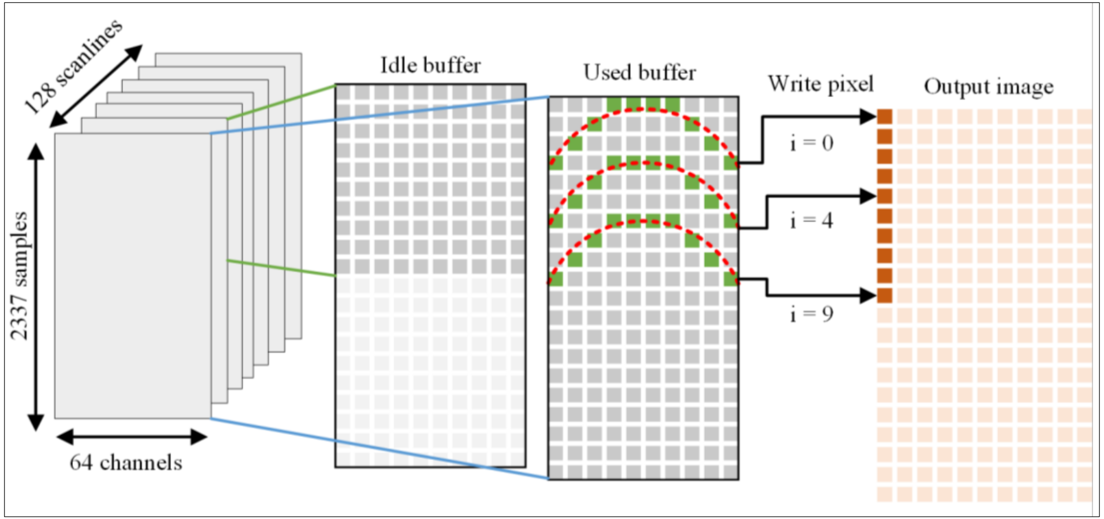


Original pre-beamformed data store pattern

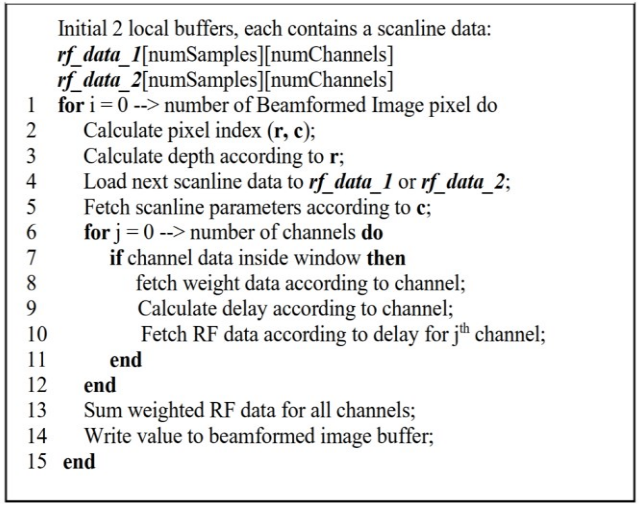


Shuffled pre-beamformed data store pattern

Beamforming on the FPGA



Beamforming implementation on FPGA

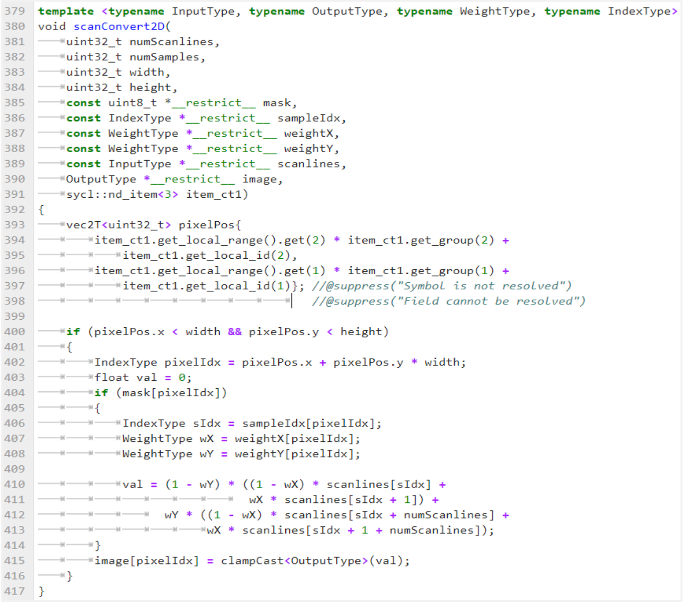


```
Initial 2 local buffers, each contains a scanline data:
rf_data_1[numSamples][numChannels]
rf_data_2[numSamples][numChannels]
for i = 0 --> number of Beamformed Image pixel do
    Calculate pixel index (r, c);
    Calculate depth according to r;
    Load next scanline data to rf_data_1 or rf_data_2;
    Fetch scanline parameters according to c;
    for j = 0 --> number of channels do
        if channel data inside window then
            fetch weight data according to channel;
            Calculate delay according to channel;
            Fetch RF data according to delay for jth channel;
        end
    end
    Sum weighted RF data for all channels;
    Write value to beamformed image buffer;
end
```

Beamforming algorithm on FPGA

FPGA code: intelfpga-devcloud-golden/SupraLib/Beamformer/ RxBeamformerCuda.dp.cpp

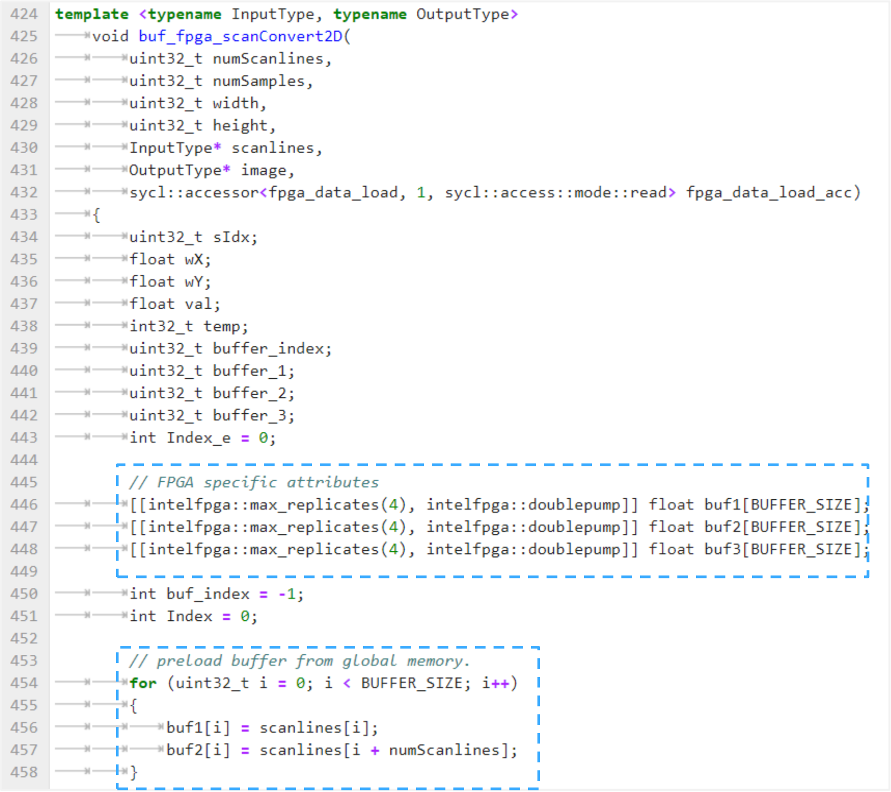
Code Sample for FPGA



```
template <typename InputType, typename OutputType, typename WeightType, typename IndexType>
void scanConvert2D(
    uint32_t numScanlines,
    uint32_t numSamples,
    uint32_t width,
    uint32_t height,
    const uint8_t *__restrict__ mask,
    const IndexType *__restrict__ sampleIdx,
    const WeightType *__restrict__ weightX,
    const WeightType *__restrict__ weightY,
    const InputType *__restrict__ scanlines,
    OutputType *__restrict__ image,
    sycl::nd_item<3> item_ct1)
{
    vec2T<uint32_t> pixelPos{
        item_ct1.get_local_range().get(2) * item_ct1.get_group(2) +
            item_ct1.get_local_id(2),
        item_ct1.get_local_range().get(1) * item_ct1.get_group(1) +
            item_ct1.get_local_id(1)}; //@suppress("Symbol is not resolved")
                                       //@suppress("Field cannot be resolved")

    if (pixelPos.x < width && pixelPos.y < height)
    {
        IndexType pixelIdx = pixelPos.x + pixelPos.y * width;
        float val = 0;
        if (mask[pixelIdx])
        {
            IndexType sIdx = sampleIdx[pixelIdx];
            WeightType wX = weightX[pixelIdx];
            WeightType wY = weightY[pixelIdx];

            val = (1 - wY) * ((1 - wX) * scanlines[sIdx] +
                                   wX * scanlines[sIdx + 1]) +
                      wY * ((1 - wX) * scanlines[sIdx + numScanlines] +
                                 wX * scanlines[sIdx + 1 + numScanlines]);
        }
        image[pixelIdx] = clampCast<OutputType>(val);
    }
}
```

- oneAPI provides high level language(DPC++) to programming FPGA, which is more flexible, easy to learn, easy to develop, easy to debug.

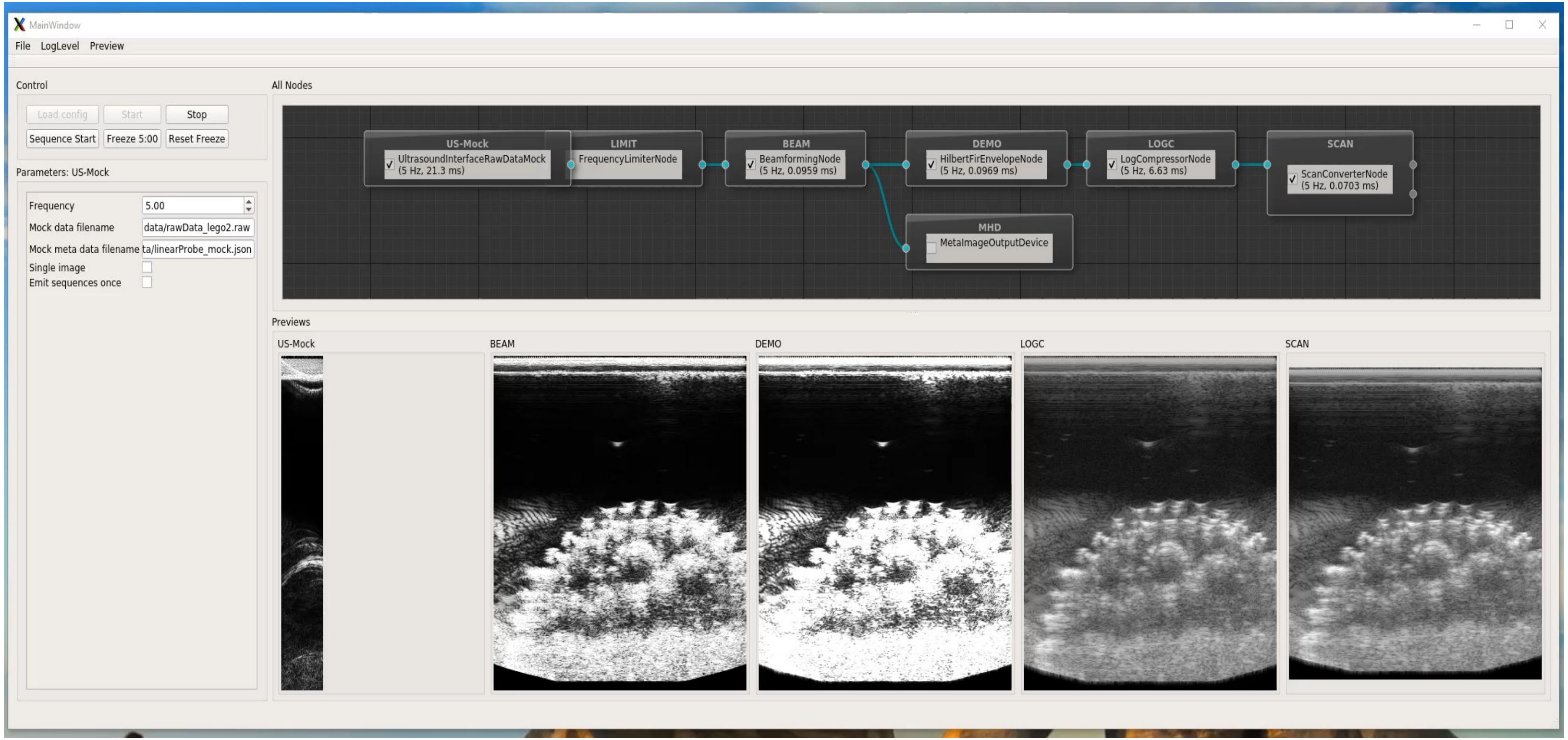
- To use oneAPI programing for FPGA, Professional knowledge of FPGA is required.



```
template <typename InputType, typename OutputType>
    void buf_fpga_scanConvert2D(
        uint32_t numScanlines,
        uint32_t numSamples,
        uint32_t width,
        uint32_t height,
        InputType* scanlines,
        OutputType* image,
        sycl::accessor<fpga_data_load, 1, sycl::access::mode::read> fpga_data_load_acc)
    {
        uint32_t sIdx;
        float wX;
        float wY;
        float val;
        int32_t temp;
        uint32_t buffer_index;
        uint32_t buffer_1;
        uint32_t buffer_2;
        uint32_t buffer_3;
        int Index_e = 0;

        // FPGA specific attributes
        [[intelfpga::max_replicates(4), intelfpga::doublepump]] float buf1[BUFFER_SIZE];
        [[intelfpga::max_replicates(4), intelfpga::doublepump]] float buf2[BUFFER_SIZE];
        [[intelfpga::max_replicates(4), intelfpga::doublepump]] float buf3[BUFFER_SIZE];

        int buf_index = -1;
        int Index = 0;

        // preload buffer from global memory.
        for (uint32_t i = 0; i < BUFFER_SIZE; i++)
        {
            buf1[i] = scanlines[i];
            buf2[i] = scanlines[i + numScanlines];
        }
```

Results and Performance

SUPRA GUI and DevCloud usage



Frequency 5.00

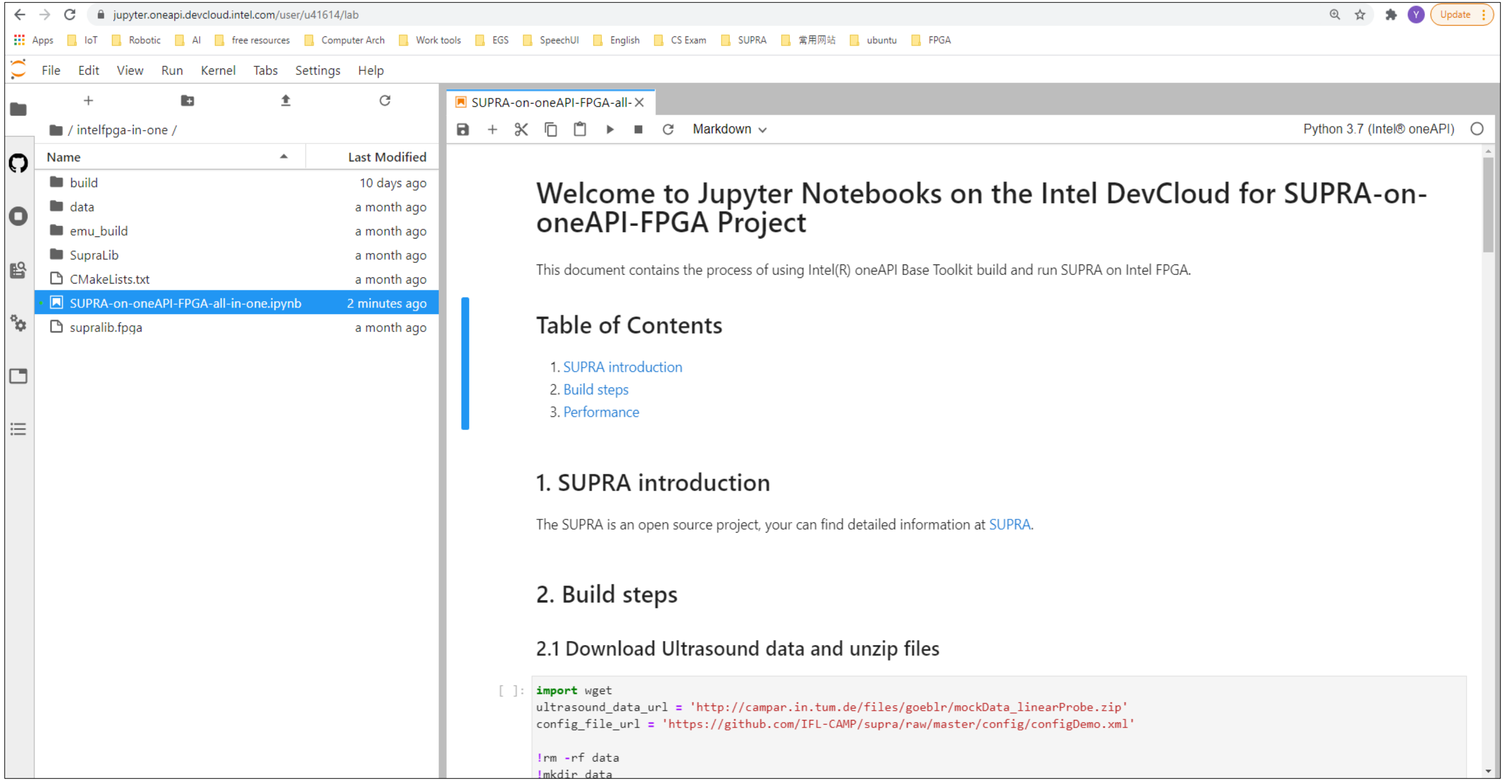
Mock data filename data/rawData_lego2.raw

Mock meta data filename ta/linearProbe_mock.json

Single image □

Emit sequences once □

Intel DevCloud usage – FPGA



Name	Last Modified
build	10 days ago
data	a month ago
emu_build	a month ago
SupraLib	a month ago
CMakeLists.txt	a month ago
SUPRA-on-oneAPI-FPGA-all-in-one.ipynb	2 minutes ago
supralib.fpga	a month ago

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

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

Table of Contents

1. SUPRA introduction

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

2. Build steps

2.1 Download Ultrasound data and unzip files

```
[ ]: import wget
ultrasound_data_url = 'http://campar.in.tum.de/files/goeblr/mockData_linearProbe.zip'
config_file_url = 'https://github.com/IFL-CAMP/supra/raw/master/config/configDemo.xml'

!rm -rf data
!mkdir data
```

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

SUPRA performance on Intel hardware

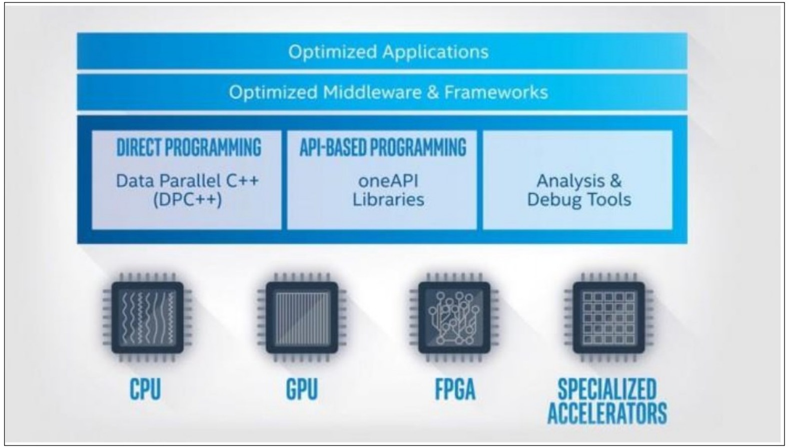
SUPRA Node	oneAPI (ms) – UHD630	Tiger lake Iris Xe	oneAPI - DG1(WA)	oneAPI – Arria 10
RxBeamforming	9.24	4.36	3.81	5.94
HilbertFirEnvelope	1.50	0.73	0.65	2.61
LogCompressor	0.27	0.1	0.08	0.34
ScanConverter	2.65	2.22	1.14	5.66
Total	13.66	7.41	5.68	5.94(max)

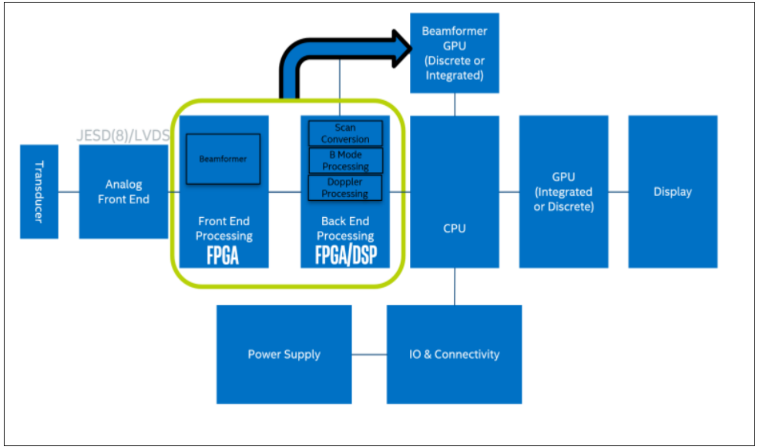
For the source code, please refer to: https://github.com/intel/supra-on-oneapi

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!

intel®