SYNTHETIC APERTURE BEAMFORMING MODULE: USER GUIDE

This document is intended to provide details on how to set up and use the synthetic aperture (SA) beamforming module that uses a graphical processing unit (GPU) for processing. The next section gives a step-by-step overview on how to install the beamforming module onto the PC and import it into LabVIEW. On Page 13, a case example is given on how to use the GPU-based module in LabView. On Page 18, technical details are provided on how the SA beamforming module is being implemented and how well it performs in practice.

INSTALLING THE BEAMFORMING MODULE

1. Overview

The SA beamforming module is implemented in the form of a dynamic link library (DLL). It is flexible enough to be used in both C/C++ environment and LabVIEW environment. This section provides a step-wise walkthrough on how to import this module into LabVIEW.

2. What's Inside

Inside the package, there should be ***four folders***:

- ***binary file***: Contains binary files that are essential for the beamforming module to work properly.
- ***LabVIEW Example***: Contains a showcase example on how to use the beamforming module in LabVIEW environment. It also contains a pre-compiled .dll file for demonstration purpose. Testing data is also included in that folder.
- ***performance***: Contains a cvp file that can be read from NVIDIA Compute Visual profiler. The file contains information on the performance on the beamforming module.
- ***riversideSA(dll)***: Contains the C++ project that can be compiled to generate the dll file of the beamforming module. A detailed setup guide will be provided in section 3d.

3. System Requirement

CPU	Any
RAM	>2 GB
Hard Drive Space	1 GB space available for installation
GPU	NVIDIA GPU (recommend with CUDA 2.0 capability or above)

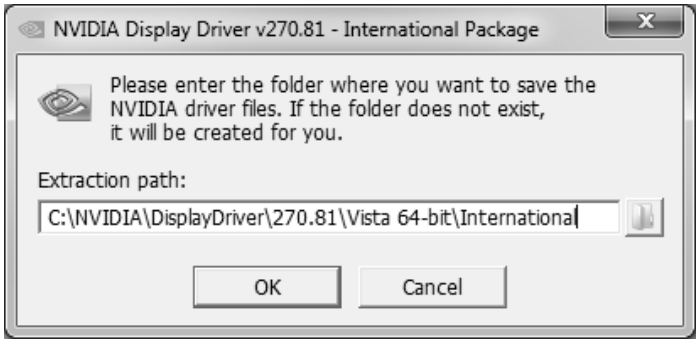
4. Environment Setup

a. Download Driver and Software

Visit the NVIDIA website to download the CUDA driver, toolkits and compute SDK. Please choose either the 32- or 64-bit version depending on the operation system. The latest version of driver can be found at: http://developer.NVIDIA.com/cuda-toolkit-40

b. Installing NVIDIA Developer Drivers

- Run the downloaded driver executable, and extract the temporary files under the default path as shown.



- After finished extracting, click *AGREE AND CONTINUE*

- Install the driver with the *Express* option and click *Next*

- Uncheck *Install NVIDIA Update* and click *Next*

- While installing, you may encounter the following security warning. It is okay to select *Install this driver software anyway*

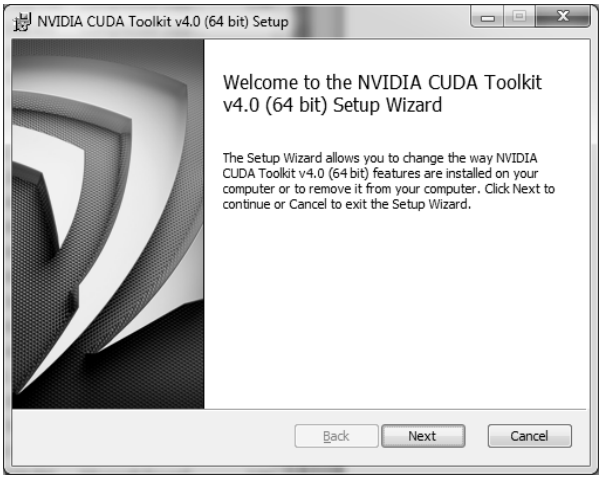
- Click *FINISH* to finish installation

c. Installing NVIDIA CUDA Toolkit

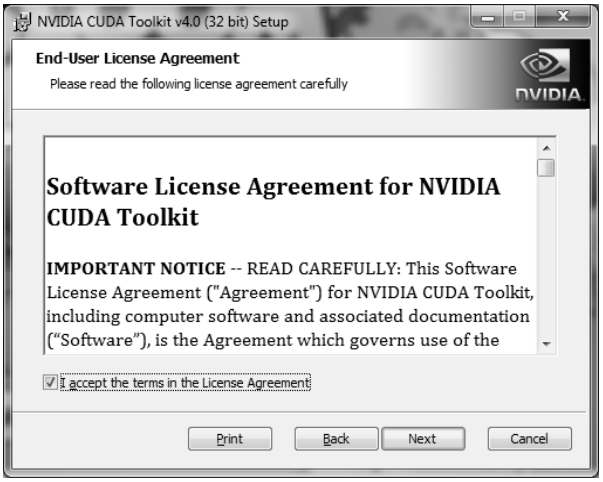
Note: It is recommended to install Visual Studio for C++ programming before installing the CUDA Toolkit and GPU Computing SDK.

- Run the executable/windows installer downloaded from the NVIDIA website

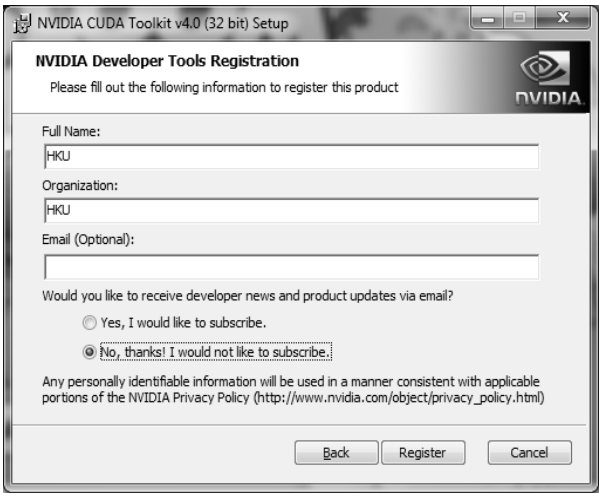
- Click *Next* to begin installation



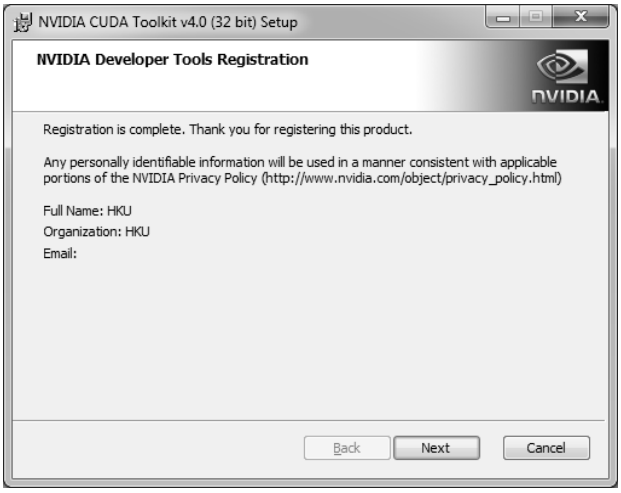
- Check the *Accept License Agreement* box and click *Next* to proceed



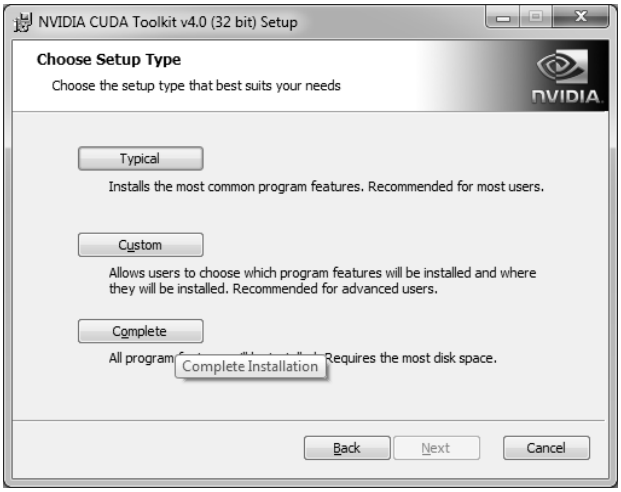
- Enter the *Name* and *Organization*. You can choose to subscribe for the new product if you wish. Then click *Register* to continue.



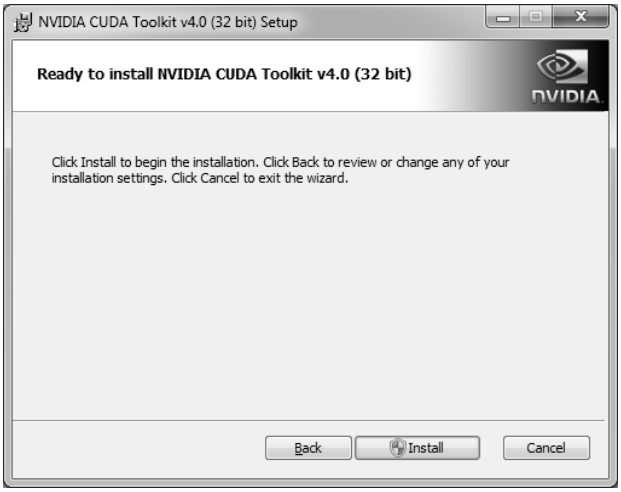
- Click *Next* to proceed



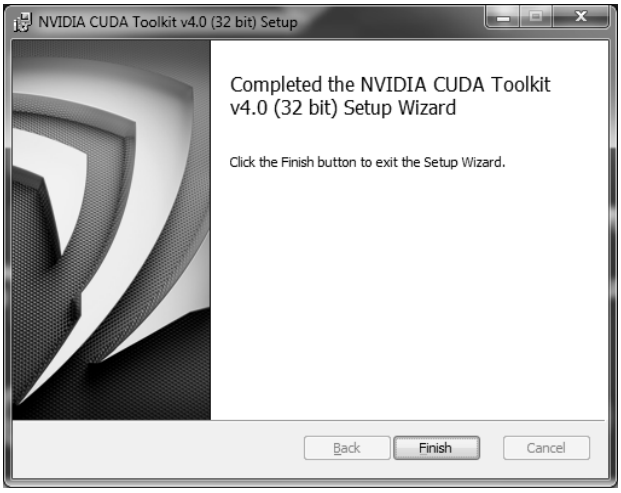
- Select *Complete* installation and then click *Next*



- Proceed by clicking *Install*



- After that, click *Finish* to end installation

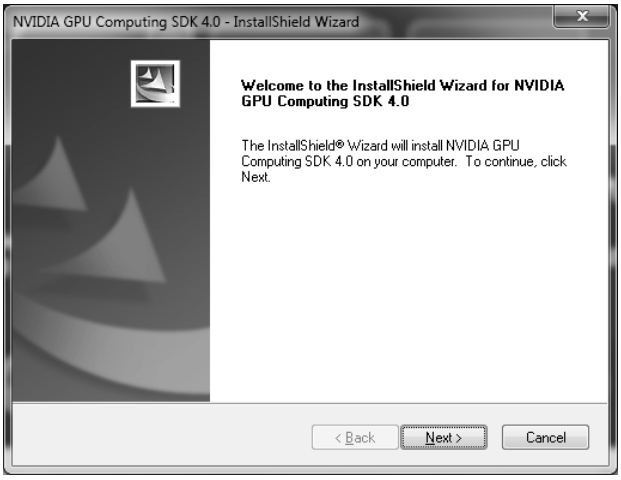


d. Installing NVIDIA CUDA Toolkit 4.0 Build Customization BUG FIX Update

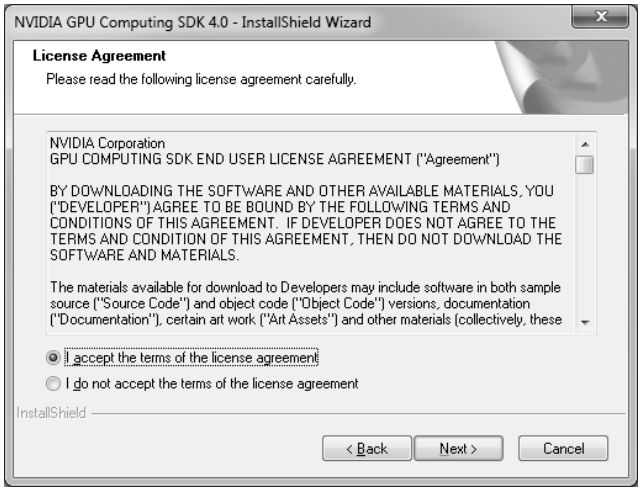
- Extract the zip file downloaded from the NVIDIA website
- Follow the instructions inside the README to fix build customization bug

e. Installing GPU Computing SDK

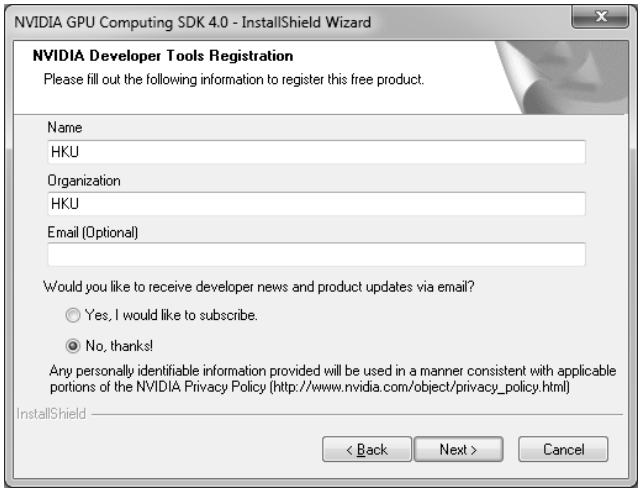
- Run the executable downloaded from the NVIDIA website
- Click *Next* to proceed with installation



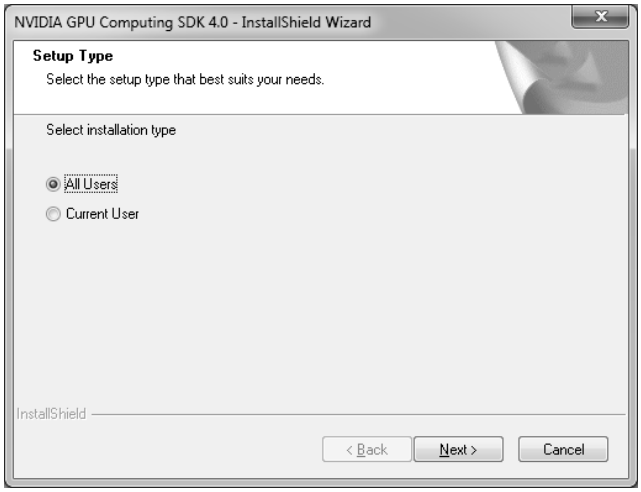
- Accept the license term and click *Next* to continue



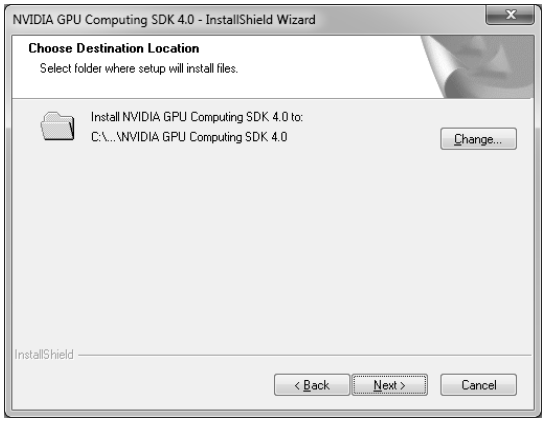
- Enter name and organization and click *Next* to proceed



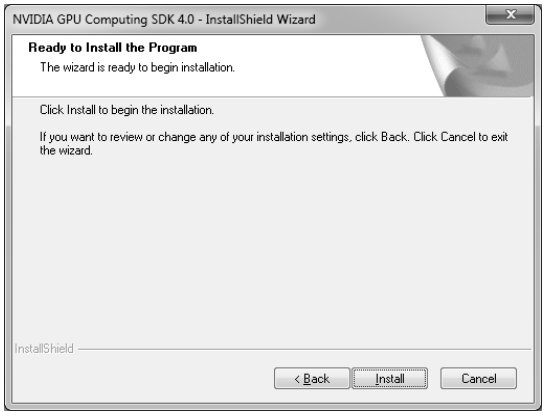
- Check *All Users* and click *Next* to continue



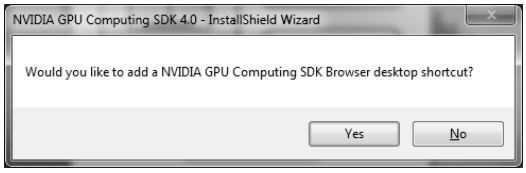
- Use the default installation path and *Next* to continue



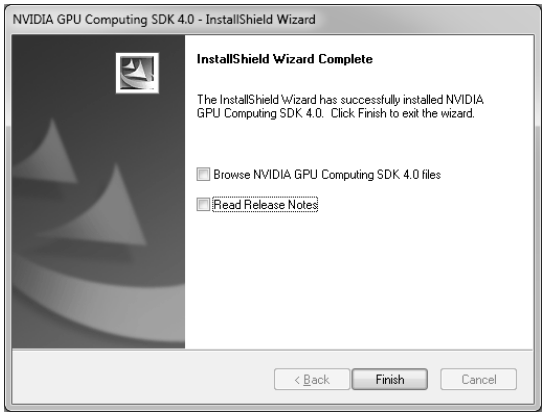
- Click *Install* to begin



- Click *Yes* to create shortcut if you wish to



- After that, click *Finish* to end installation



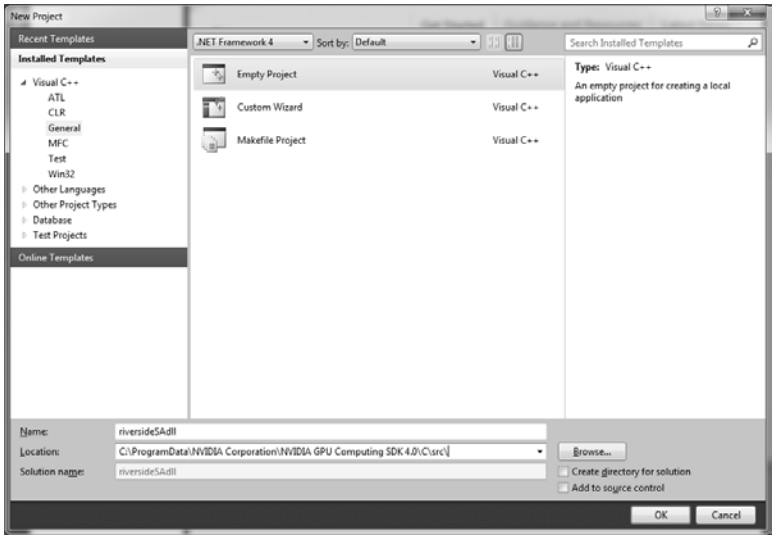
f. Building Common Libraries

- Browse to *C:\ProgramData\NVIDIA Corporation\NVIDIA GPU Computing SDK 4.0\C\common* (assuming default install path is used)
- Open the solution file *Release_vs2008*
- Change the configuration to *Release* and the solution platform according to your operation system (Win32/x64)
- Build the solution under *Build>Build Solution*

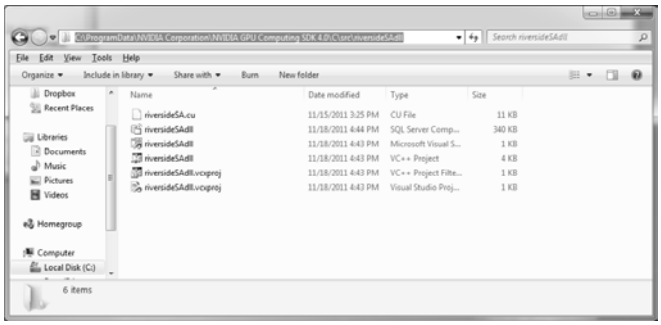
g. Compiling the Dynamic Link Library

Here we use Visual Studio 2010 to compile the module. Most of the steps in setting up the compilation are the same, except for the ones for setting up new coding projects.

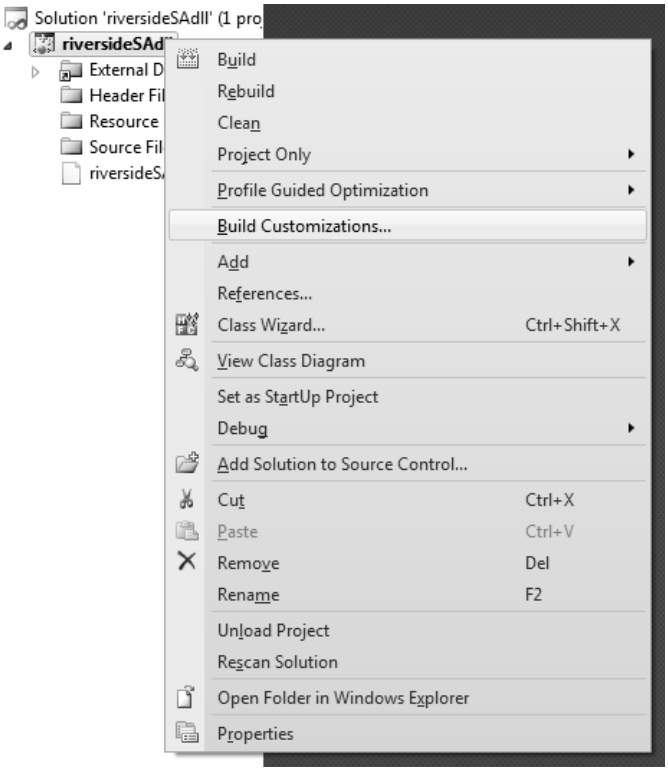
- Create a new project under *File>New>Project*. The new project window will pop up.



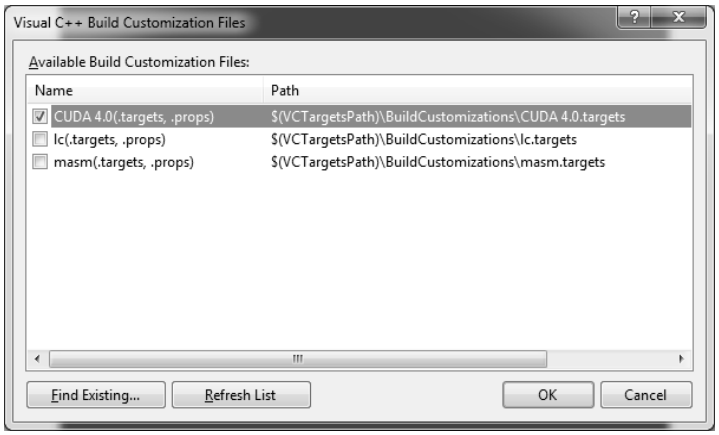
- Select *Empty Project*, and enter riversideSAdll as the name. Then navigate to *C:\ProgramData\NVIDIA Corporation\NVIDIA GPU Computing SDK 4.0\C\src* (assuming default install path is used) for location, and uncheck both *Create Directory for Solution* and *Add to Source Control*
- Copy *riversideSA.cu* inside the *riversideSA(dll)* folder to the folder that was just created for the new project (*C:\ProgramData\NVIDIA Corporation\NVIDIA GPU Computing SDK 4.0\C\src\riversideSAdll*, assuming default install path is used)



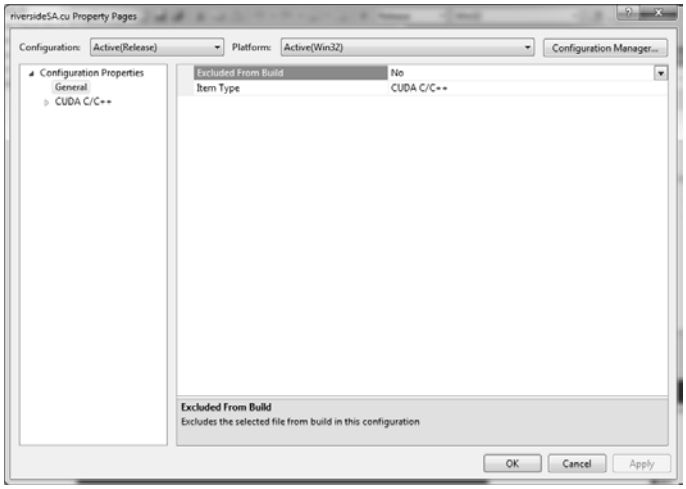
- Add *riversideSA.cu* to the project through *Project>Add Existing Item*
- Right click on the project name and select *Build Customizations*



- Check *CUDA 4.0* and click *OK*

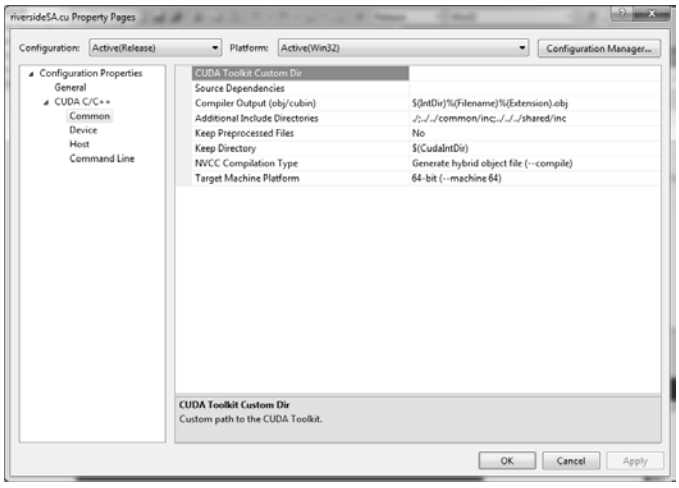


- Change the configuration from *Debug* to *Release*
- Access property tab of *riversideSA.cu* through right click on *riversideSA.cu>Properties*
- Choose *CUDA/C++* from the drop down menu of *Item Type*, and click *Apply* to apply the settings



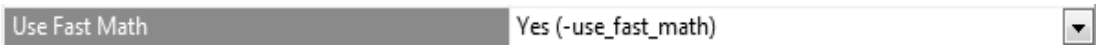
- Expand the *CUDA C/C++* tab, and add the following links to *Additional Include Directories*:

```
./
../../common/inc
../../../shared/inc
```

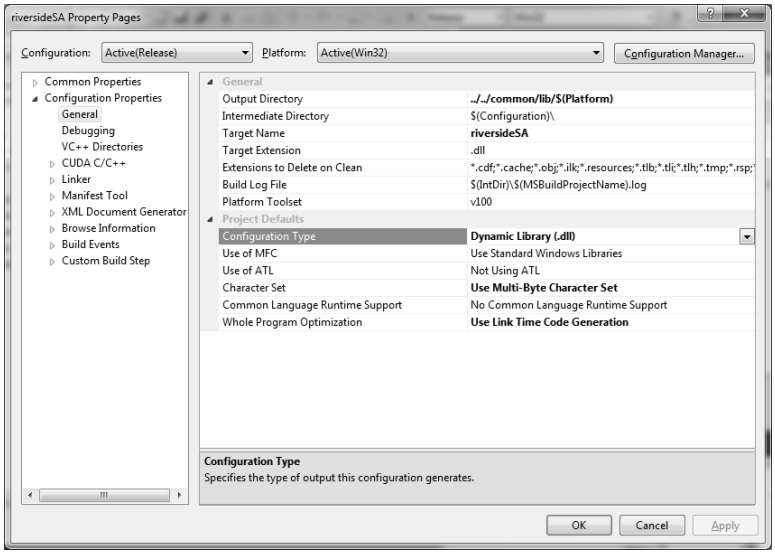


- Select the correct *Target Machine Platform* depending on the operation system (32/64-bit windows)
- Under the *Device* tab, change *Code Generation* to compute_13,sm_13 (you may change to compute_20, sm_20 for Fermi architecture)

- Under *Host* tab, enable *Use Fast Math*



- Click *Apply* and close the dialog window
- Go to *Properties* page of the project, under *Configuration Properties>General* tab, change the configuration type to *Dynamic Library (.dll)*



- For the *CUDA C/C++* tab, repeat what has been done to *riversideSA.cu>Properties* (in particular, make similar changes to changes to *Additional Include Directories*, *Target Machine Platform*, *Code Generation*, and *Use Fast Math*)
- Expand the *Linker* tab, go to *General* and add the following to *Additional Library Directories*

```
../../common/lib/$(PlatformName)
../../../shared/lib/$(PlatformName)
```

- Go to *Input* under the same tab, and add the following *Additional Dependencies*

```
cudart.lib
cutil32.lib
```

- The compiled dynamic link file will be at

  C:\ProgramData\NVIDIA Corporation\NVIDIA GPU Computing SDK 4.0\C\src\riversideSAdll\Release (assuming default install path is used)

For users with Visual Studio 2008, go to http://hpcwatch.com/integrating-cuda-4-0-with-microsoft-visual-studio-2008/ and follow steps 7 and 8. The rest of the steps would be the same. The *CUDA C/C++* tab in VS2010 would be renamed as the *Custom Build Step* tab.

h. Preparing the Work Folder

There are three binary files that come with the beamforming module:

- ***hilbert.bin***: Essential for the beamforming module to work properly. It must stay with the dynamic link library file under the same folder.
- ***riversideTdrGeometry.bin***: Contains information describing the transducer geometry.
- ***eyeData.bin***: Contains the test data for the LabVIEW example.

To access these files, create a folder and copy all three binary files into it, as well as the dynamic link library that was generated in previous subsection.

LABVIEW EXAMPLE

1. Overview

Inside the folder ***LabVIEW Example***, there is an example demonstrating how to use the beamforming module in LabVIEW. The following is a stepwise walkthrough of the example:

- Open the file *SAlabView.vi*
- Click *Run*
- Browse to the folder that contains the binary files, and select *eyeData.bin*
- Browse to the folder that contains the binary files, and select *riversideTdrGeometry.bin*
- The B-mode image will be shown on the waveform graph

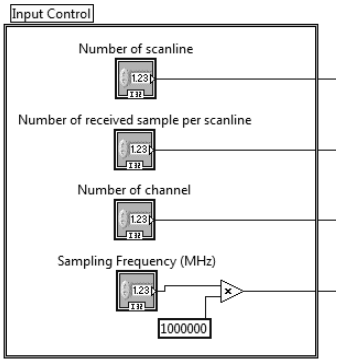
2. Description of Processing Procedure

The SAlabView example has three main processing stages: 1) read input data, 2) call the *Beamforming* module, and 3) display the result. They will be discussed one by one here.

a. Reading Inputs

i. Imaging Parameters

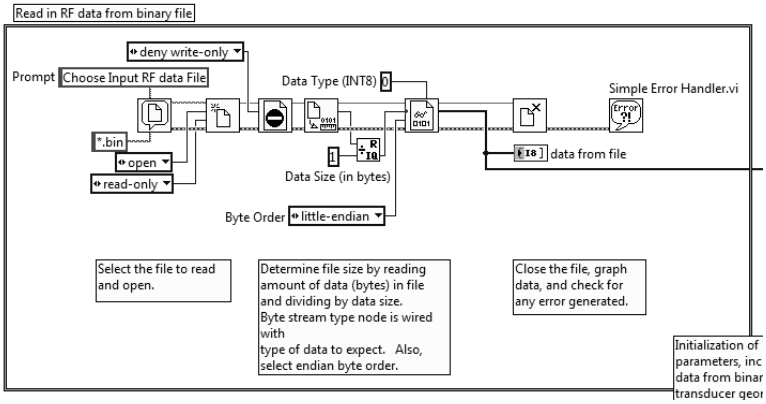
The following figure shows the block diagram for reading imaging parameters



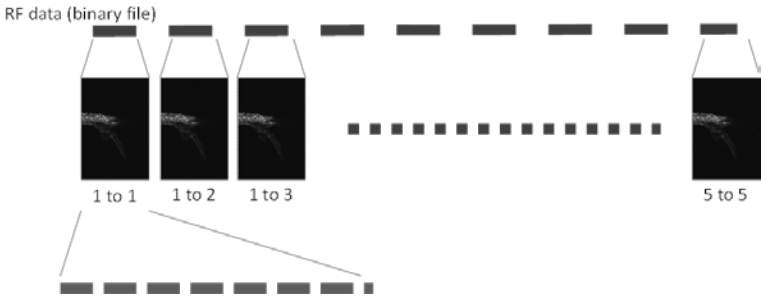
The imaging parameters include: *number of scanlines*, *number of received sample per scanline*, *number of channel*, and *sampling frequency*. These are control parameters that can be adjusted from the front panel.

ii. RF data

The following block diagram shows how the RF data is being read from binary file.



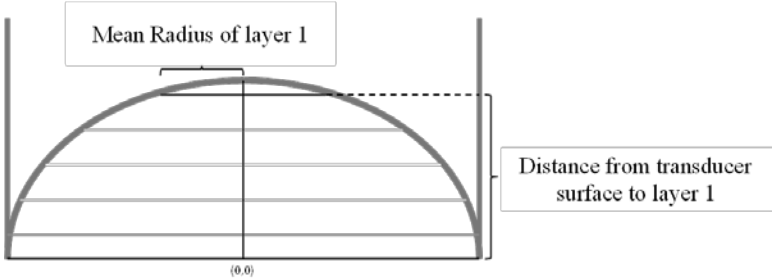
The binary file/RF data is packed according to the T/R pair of the RF data as illustrated below:



Each RF data file is organized as a 1D data array, and each of its segments corresponds to the RF data from a different T/R pair. The data format is int8.

iii. Transducer geometry

The transducer geometry is stored in a 1D data array format with 2*N* samples, where *N* is the number of array elements. The first *N* samples correspond to the mean radius of each array element, while the last *N* samples correspond to the distance of each array element from the transducer surface (see the figure below).

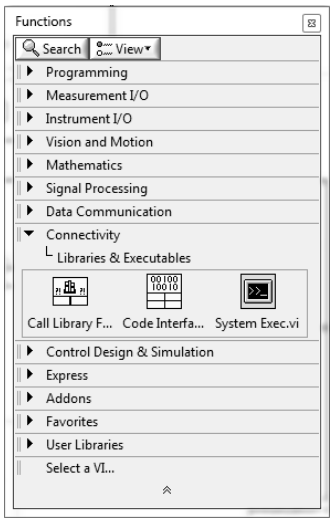


b. Beamforming

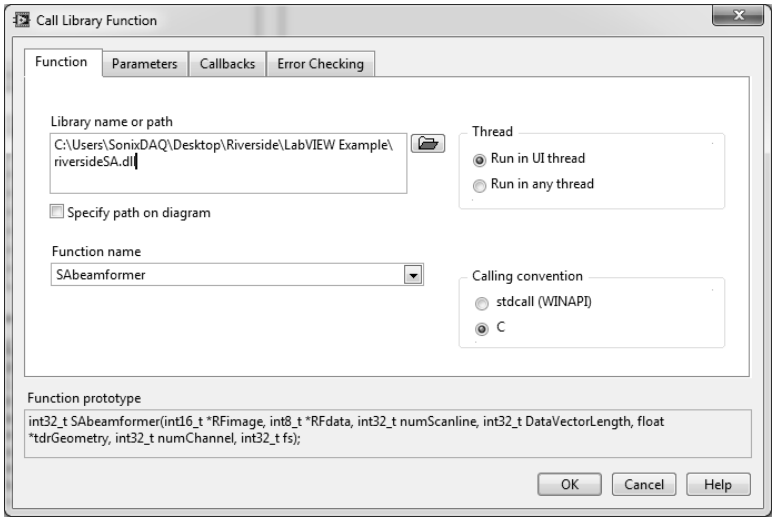
Note: The maximum imaging depth that can be processed by the beamforming module is respectively 12 cm and 4.8 cm for 100 MHz and 250 MHz sampling rate.

The beamforming module can be used by LabVIEW through the following steps.

- Drag the *Call Library Function Node* block (under *Connectivity>Libraries and Executables*) to the block diagram



- Double click the block to edit its properties



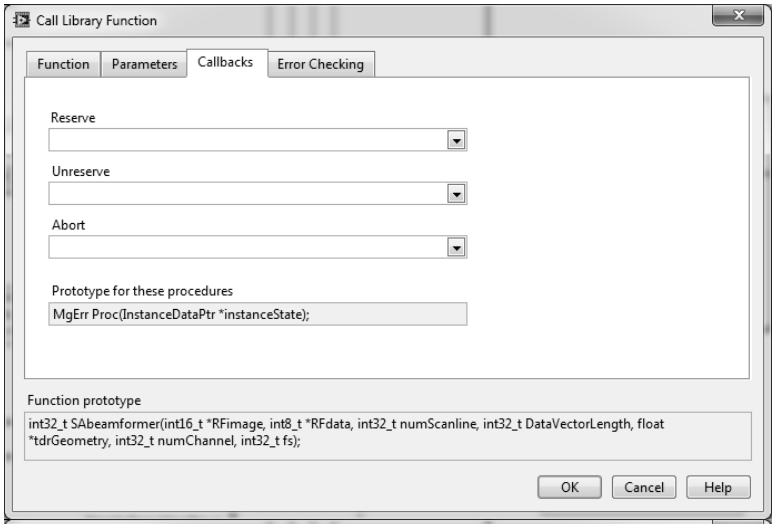
- Enter the file path to the dynamic link library compiled earlier (should be inside the folder you created containing the binary file as well)

- Change the function name to *SAbeamformer*

- In the *Parameters* tab, add the following parameters

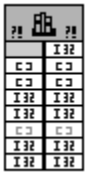
Name	Type	Data Type	Dimension	Array format
nError	Numeric	Signed 32-bit Integer	NA	NA
RFimage	Array	Signed 16-bit Integer	1	Array Data Pointer
RFdata	Array	Signed 8-bit Integer	1	Array Data Pointer
numScanline	Numeric	Signed 32-bit Integer	NA	NA
DataVectorLength	Numeric	Signed 32-bit Integer	NA	NA
tdrGeometry	Array	4-byte Single	1	Array Data Pointer
numChannel	Numeric	Signed 32-bit Integer	NA	NA

- Set no callbacks and default error checking under the *Callback* tab and *Error Checking* tab respectively

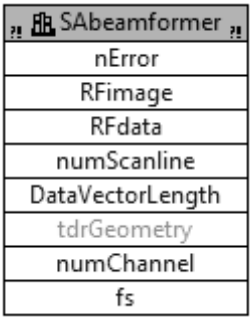


- Click *OK* to close the window

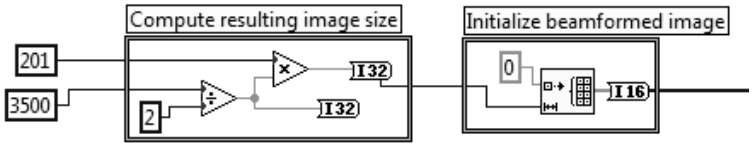
- Now the beamforming module is imported to LabVIEW and the block should look like this



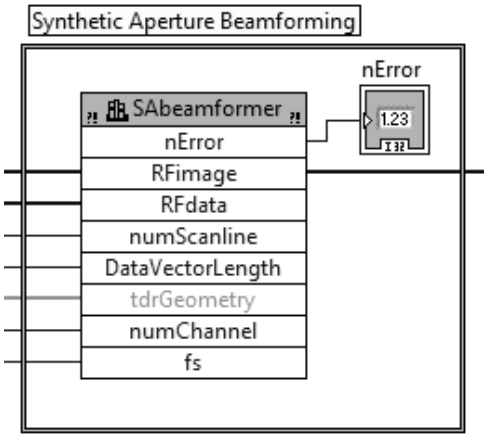
- Right click the block and change the *Name Format* to *Name* and the block will change its appearance



- Initialize the array buffer for storing the B-mode image, in which the beamforming module assumes that the image resolution is *number of sample per scanline/2* (referred to as *image length* from here on) by *number of scanlines*

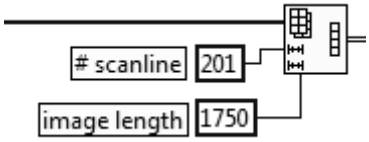


- Create an indicator for *nError*
- Connect the inputs to the beamforming module

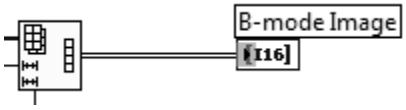


c. Display the B-mode image

- Reshape the *RFimage* array from 1D to 2D, where *number of scanlines* is the first dimension while *image length* is the second dimension



- Connect the reshaped array to the B-mode display



- If there is error feedback from the beamforming module, it is likely to be referring to one of the following issues:

nError	Problem	Solution
-2	Cannot find hilbert.bin	Copy the hilbert.bin to the folder containing the dynamic link library file
-3	Input RF data size exceed maximum size that can be handled	Reduce the Input RF data size

IMPLEMENTATION DETAILS: BEAMFORMING ALGORITHM

1. Description of Algorithm

This section provides technical details on how the SA beamforming module is being implemented in the module. The following block diagram shows the module's overall workflow:

RF data, imaging parameters → Analytic Signal Conversion → Delay Calculation → Delay and Sum → B-mode image

As can be seen, the algorithm is divided into three steps: 1) perform analytic signal conversion on the input RF data; 2) compute the time delay for each pixel; 3) beamforming the B-mode image using delay-and-sum approach. In the following, we present how each step is being carried out.

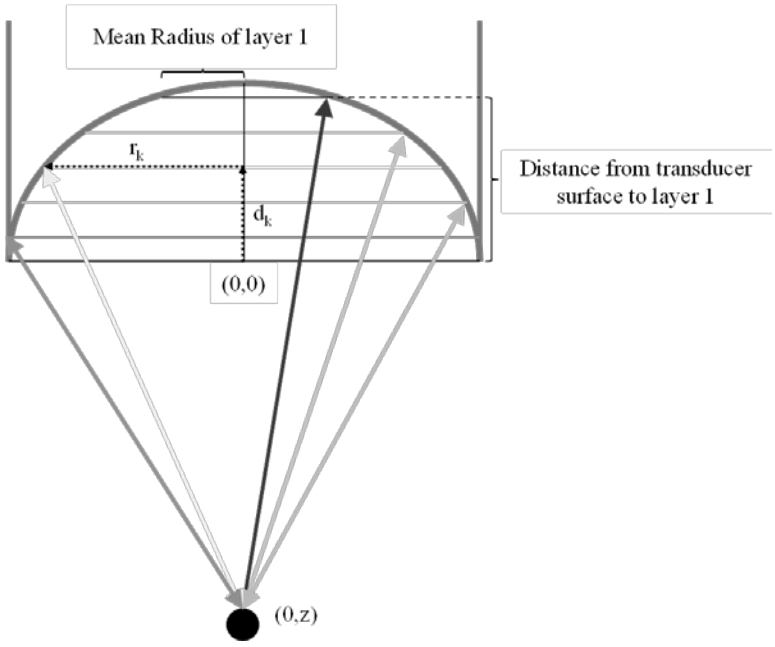
a. Demodulation

The first step of beamforming module is to convert the input RF data into its analytic form. In this module, an FIR filter with frequency response equivalents to Hilbert transform is used. Therefore, by convolving the FIR filter with the input RF data, the data is converted into its analytic form.

b. Delay Calculation

Before performing delay-and-sum (DAS) beamforming, the delay of each channel for each pixel is first computed. These values can be reused for every scanline to speed up the beamforming process in the next step, and in turn increase the processing throughput of the beamforming module.

The following figure shows the geometric configuration that is used to calculate the delays for pixel *P* at (0, *z*) for different T/R channels:



Note: In this work, the center point of the outermost transducer layer is defined as (0,0). Also, r_k and d_k are respectively denoted as the mean radius and distance between the k^{th} transducer layer and the outermost layer.

For pixel *P* with transmit channel *i* and receive channel *j*, the time delay $t_d(z, i, j)$ can be calculated as follows (for speed of sound c_o):

$$t_d(z,i,j) = \frac{\sqrt{r_i{}^2 + (z+d_i)^2} + \sqrt{r_j{}^2 + (z+d_j)^2}}{c_0}$$

c. Beamforming

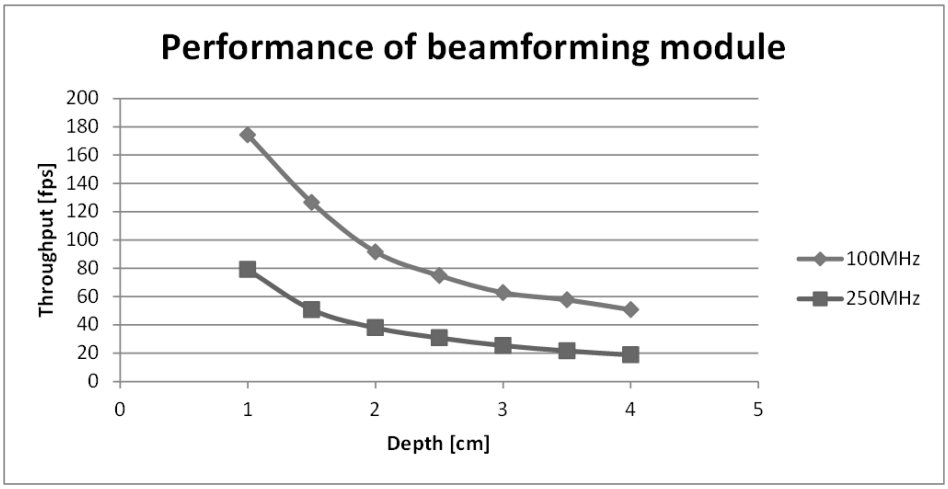
With the time delay computed using the above formula, the B-mode image can be obtained by through a delay and sum operation. For pixel *P*, its pixel value *I*(*z*) can be found from the channel-domain RF data $s_{i,j}(.)$ as follows:

$$I(z) = \sum_{i=1}^{K}\sum_{j=1}^{K} s_{i,j}(t_d(z,i,j))$$

where *K* is the number of channels in the annular array.

2. Processing Performance

An initial performance analysis of the beamforming module has been carried out using NVIDIA's Compute Visual Profiler. The time taken for each task to be completed has been measured, and the results used to calculate the processing throughput (assuming continuous availability of input data streams). The following figure shows the performance of the beamforming module at different depths and sampling frequency.



With 100 MHz sampling rate, the beamforming module has throughput of at least 50 fps (for 4 cm sampling depth). The processing throughput is higher for shorter depths because there are fewer RF samples per scanline to process.

For 250 MHz sampling rate, the minimum throughput of the beamforming module is around 20 fps (at 4 cm imaging depth).