Cuda Cross-correlation Notes

Introduction

The package contains several separated steps to get the cross-correlation between array1 and array2. As you can see, there are four directories for different steps and three files including "params.h", "segy.info" and "segy.list". STEP1_preprocessing folder is for data preprocessing (mainly downsample and filtering). STEP2_xcorr is for cross-correlation (cc) between array1 and array2 with GPU, the cc result is saved as PASSCAL_SEGY format. STEP3_segy2sac is to post-process the output cc results (converting PASSCAL_SEGY formats to SAC formats). Lastly, STEP4 is optional but useful to stack multiple cc results. For example, if you have hundreds or thousands of hourly segys to be cross-correlated, you can divide these data into several independent jobs for calculating cross-correlations, say 3 jobs. The you will have three cc results. Step4 is to stack these 3 independent cc results into a final stacked cc. The usage for these steps will be explained in the following.

"params.h" is the control file for all steps. Therefore, once you change the params.h, you have to go to each STEP folders and re-compile the code. When you open the params.h, you will find these parameters below:

1) ARRAY1_ORIGIN_DIR: the directory that stores the original hourly PASSCAL_SEGY data of array1.
2) ARRAY2_ORIGIN_DIR: same as (1) but for array2
3) ARRAY1_DOWNSAMPLE_DIR: the directory that stores the downsampled and filtered hourly PASSCAL_SEGY data from STEP1. If not created, the code will do it automatically. STEP2 will read data from this directory.
4) ARRAY2_DOWNSAMPLE_DIR: same as (3) but for array2. It is fine that you prepare the data for STEP2 by yourself and do not use the STEP1 to do that. But you need to make sure that the data is hourly PASSCAL_SEGY format and write the directory path here since STEP2 will read data from this directory.
5) XCORR_OUTPUT_DIR: the directory you want to save the output stacked cc data. The output is in PASSCAL_SEGY format. Now, I fixed the output name as "stack.segy". If you want to change it, you have to go to source code and change it. In the future, it can be easier to add one parameter in params.h to control the output name.
6) SEGY_LIST: the file that lists the data to be cross-correlated.
7) SEGY_INFO: the file that contains the basic information of two arrays. If you use step1 to preprocess the data, this file will be generated automatically. Otherwise, you need to create it by yourself. STEP2 will read this file.
8) STACK_XCORR_OUTPUT_DIR: The directory you want to save the stacked cc. Only used in STEP4.
9) ARRAY1_DECIMATE: decimation factor for array1. Only used in

STEP1. [available value: 2-7]

10) ARRAY2_DECIMATE: same as (9) but for array2.
11) Freql: low end frequency for bandpass filtering. This will be used in STEP1, and also STEP2 if SPECTRAL_WHITENING is set to be 1.
12) Freqh: same as (11) but for high frequency end.
13) TEMPORAL_NORMALIZATION: 0 for no temporal normalization, 1 for one-bit temporal normalization and 2 for running-absolute-mean (RAM). RAM is suggested here.
14) MAX_LAG: the half length of stacked cc. Unit is second.
15) XCOR_SEGMENT: time window to be cross-correlated. Due to limited GPU memory size, it is optimal if the data points of the time window are close to 2^N. Otherwise, GPU will probably overflow.

"segy.list": lists the segy file names to be cross-correlated. Note that for the same hour, the data names for array1 and array2 must be the same.

"segy.info": Basic information for array1 and array2.

STEP1. Pre-processing

CASE 1. DAS-DAS cross-correlation

Under the directory STEP1_preprocessing, first compile with "**sh compile.sh**". Before compiling the code, you need to setup the parameters in "params.h". In particular, the parameters "ARRAY1_ORIGIN_DIR", "ARRAY2_ORIGIN_DIR", "ARRAY1_DOWNSAMPLE_DIR", "ARRAY2_DOWNSAMPLE_DIR", "SEGY_LIST", "SEGY_INFO", "ARRAY1_DECIMATE", "ARRAY2_DECIMATE", "freql" and "freqh" will be used in STEP1. The preprocessed data will be saved in "ARRAY1[2]_DOWNSAMPLE_DIR" and new segy information will be outputted in "SEGY_INFO". In the example, the raw hourly PASSCAL_SEGY data has a sampling rate of 250 Hz. We downsampled them to 50 Hz and bandpass filtered between 0.1-10 Hz. After compiling, execute "**./preprocess**". You will find the processed data in the directory you just setup. It is generally slow in this step.

CASE 2. Broadband-DAS cross-correlation

In this case, you have to preprocess the broadband (BB) and DAS data separately. The current code doesn't support preprocessing the BB and DAS data at the same time. To preprocess the DAS data, you need to go the preprocess.c and comment lines from L106 to L150, assuming array1 is DAS array. Similarly, you can comment array1-related lines in preprocess.c if DAS array is array2. Note that the following instruction is for SCEC STP data.

This step is to download BB sac files, examine the sac files (e.g. any missing points), filter the BB waveforms and convert to PASSCAL_SEGY files.

1. Create the directory for storing downloaded BB data.
   "**mkdir [YourWorkDir]**"
2. Enter the directory, prepare a file "segy_list" that describes hours needed to be downloaded/cross-correlated (Format: "YYYYMMDDHH"). Once generated, run "**sh getstp.sh**" to download from STP. Note that we define channel code "0000" as "HHE" (velocity); "0001" as "HHN"; "0002" as "HHZ"; "0003" as "HNE" (acceleration); "0004" as "HNN"; "0005" as "HNZ". You can find the correspondence in file "chan_code".
3. Once downloaded the sac files, you need to scan the sachead information to check if all the continuous data are complete or miss any data points. First, check if all the hours have data. Second, check if any sacfile has anomaly amplitude or incomplete time series before filtering. Comment the Line 8 in "check.sh" and run "**sh check.sh**".
   Note: If any anomaly sacfiles jump out, you need to delete all six channel sacfiles of that hour manually, e.g. "**rm XXXXXXXXXX.*.sac**". Also, remember to delete this hour in "segy_list" file!
4. Proceed to filter, create sac_bp directory: "**mkdir sac_bp**"
   Move all sacfiles to sac_bp directory: "**mv *.sac sac_bp**"
   Copy run.sh into sac_bp directory: "**cp run.sh sac_bp**"
   Run run.sh in sac_bp directory: "**cd sac_bp**" & "**sh run.sh**"
5. Check these filtered sacfiles again. Copy check.sh to sac_bp folder: "**cp ../check.sh .**", comment first two steps and uncomment the third step (8th line), then run this script "**sh check.sh**"
   Note: It should be no errors coming out from this step. But if does, remove that hour data completely and also delete that hour in "segy_list" file.
6. Convert the sacfiles into PASSCAL_SEGY format. GO to "Codes/STEP1_preprocessing/sac2segy" and modify the code "writesegy_stp.c":
   a. Replacing the previous directory with your new downloaded sac directory in Line 44, 45 & 64
   b. Changing the directory in Line59 to the directory you want to save the newly generated segy files.

   Once setup, compile the code "**gcc writesegy_stp.c -lm**"
   And then execute "**./a.out**". Finally, you can find the new PASSCAL_SEGY in the directory you just setup.

STEP2. XCORR

In directory STEP2_XCORR, there is a small code that checks if the preprocessed PASSCAL_SEGY files are normal. But this code is not flexible yet. For example, you need to change "NSTA" in the code to make sure that the number of channels is correct. To use it, you need to prepare a file that lists the segy files.

In the example case, we set NSTA to 6 for BB and compile the code with "**gcc check.c -lm -o check**". Then run it with "**./check segy_lst_check**". If you are confident with the preprocessed data, it is okay to skip this check step.

Next is to run the cross-correlation code. Before using it, you need to check the "params.h" files, in particular the parameters: "ARRAY1_DOWNSAMPLE_DIR", "ARRAY2_DOWNSAMPLE_DIR", "XCORR_OUTPUT_DIR", "SEGY_LIST", "SEGY_INFO", "freql", "freqh" ("freql[h]" are only used if "TEMPORAL_NORMALIZATION" is 2), "TEMPORAL_NORMALIZATION", "SPECTRAL_WHITENING", "MAX_LAG" and "XCOR_SEGMENT". Carefully check the "segy.list" and "segy.info". Note that segy.info is automatically generated from STEP1 for DAS-DAS cc. For BB-DAS cc, you need to setup "segy.info" manually. Once ready, compile the code "**sh compile.sh**". Then run the code with

./xcorr 0[/1/2/3]

The argument 0[/1/2/3] denotes the GPU ID. Currently, the code is only designed for one GPU per case. To check the GPU IDs and current working status, you can use the command "**nvidia-smi**". Once the program starts running, you can run "nvidia-smi" to monitor the status of GPUs. For big dataset, currently the only way to use multiple GPUs is to separate segy files in "SEGY_LIST", compile multiple executable commands (e.g. xcorr1, xcorr2…) for each SEGY_LIST file, and then run them separately on different GPUs. For example, "./xcorr1 0", "./xcorr2 1"…. In this way, STEP4 will be used to stack all outputs into a final cc result.

● STEP3. segy2sac

This is a post-processing step once cc is generated. This step converts PASSCAL_SEGY result to sac files. Before compiling the code, make sure the parameters in "params.h": "XCORR_OUTPUT_DIR" and "MAX_LAG" are correct. The sac files will be saved under "XCORR_OUTPUT_DIR" as well. If these parameters are correct, compile the code: "**sh compile.sh**" then run it "**./segy2sac**".

● STEP4. stackcc_optional

This is an optional step, but useful if you have multiple cc to be stacked. Say you have multiple PASSCAL_SEGY cc files stored in different directories. You need to list all the cc files in "add_lst". And make sure the parameter in "params.h": "STACK_XCORR_OUTPUT_DIR" and "MAX_LAG" are correct. If so, compile the code: "**sh compile.sh**" and then run it "**./stack add_lst**". Unfortunately, there is no example for using this step.

Note: In file add_lst, the first column is the file name and the second column is the hours used for cross-correlation.

Free feel to email me if you have any questions or problems on using the code.

Zhichao Shen
szchao40@gmail.com