2022 EAGE ANNUAL
83RD CONFERENCE & EXHIBITION
MADRID I SPAIN

EFFICIENT ESTIMATION OF LOCAL KINEMATIC OPERATORS IN NONLINEAR BEAMFORMING USING GPU GRAPHICS CARDS

Y. Sun[1], I. Silvestrov[2], A. Bakulin[2]

[1] Aramco Research Center; [2] Saudi Aramco EXPEC Advanced Research Center

Summary

Nonlinear beamforming is an effective method to enhance the quality of noisy seismic data. It uses local kinematic operators to describe local wavefronts, and then stack neighboring traces guided by these operators to improve the signal-to-noise ratio of data. Although the 2+2+1 method is a pragmatic solver to estimate local kinematic operators from input data, its computation efficiency is still challenging when the solution space is big. We propose to speed up the 2+2+1 method using graphics processing unit (GPU) computing with the Compute Unified Device Architecture (CUDA) programming language. We introduce our GPU-based 2+2+1 algorithm, and demonstrate its efficiency improvement using a field data example. A speed-up factor of ~10 is obtained compared to the CPU version of the 2+2+1 method.

Efficient estimation of local kinematic operators in nonlinear beamforming using GPU graphics cards

Introduction

Modern 3D land seismic data acquisition prefers high-channel count surveys or even single-sensor surveys over the traditional scheme of acquiring sparse data with large field arrays (Bakulin et al., 2020), as the former can provide better wavefield sampling that benefits subsequent data processing, imaging, and inversion. However, denser grids of smaller receiver arrays in reality lead to low-quality data. Figure 1 in Bakulin et al. (2018) shows a representative comparison on the raw data quality between a traditional data acquisition scheme and a modern scheme. Nowadays, it is a general technical challenge for the seismic industry to properly deal with data acquired by the modern scheme.

Nonlinear beamforming (NLBF) aims at improving the quality of modern 3D seismic data. NLBF is an operator-based technology, and it is somewhat similar to some established technologies in the field of seismics, such as the Common Focus Point technology (Sun et al., 2014), the Common Reflection Surface technology (Jäger et al., 2001) and the multifocusing technology (Berkovitch et al., 2011). Technically, NLBF is a two-step process (Sun et al., 2021): it first estimates local kinematic operators from input data by maximizing a semblance-type cost function; next, every trace in the original dataset is stacked with its neighboring traces via the guidance of the estimated local kinematic operators. The biggest challenge in NLBF is its first step, which has to solve a lot of nonlinear optimization problems. The 2+2+1 method is a pragmatic solver for estimating local kinematic operators (Bakulin et al., 2018; Sun et al., 2022). Even though the 2+2+1 method is a local-search solver, once the solution space becomes big, its computation efficiency is still a challenge.

Thanks to the Compute Unified Device Architecture (CUDA) programming language (NVIDIA, 2020), graphics processing unit (GPU) computing has been playing a more and more important role in the field of high-performance computing in recent years. Today, more and more supercomputers are powered by GPUs. In terms of software development, it is now a common practice to implement the same software for both the CPU platform and the GPU platform. In this paper, we introduce our GPU implementation of the 2+2+1 method, which obtains an efficiency gain of ~10 compared to the CPU version of the 2+2+1 method.

Local Kinematic Operators & The 2+2+1 Method

The local kinematic operators used in NLBF are the same as those introduced in Hoecht et al. (2009) and Buzlukov and Landa (2013). Mathematically, they are described by a second-order equation:

$$\begin{aligned}\Delta t(x,y;x_0,y_0) &= t(x,y) - t(x_0,y_0)\\ &= A\cdot(x-x_0) + B\cdot(y-y_0) + C\cdot(x-x_0)\cdot(y-y_0) + D\cdot(x-x_0)^2 + E\cdot(y-y_0)^2\,, \end{aligned} \quad (1)$$

where $t(x,y)$ is the travel time of the trace located at (x,y) in a seismic gather, $t(x_0,y_0)$ is the travel time of the NLBF parameter trace located at (x_0,y_0), and the coefficients $\{A,B,C,D,E\}$ are the unknown coefficients for a local kinematic operator centered at $t(x_0,y_0)$. In a rigorous mathematical language, these coefficients $\{A,B,C,D,E\}$ should be written as $\{A(x_0,y_0,t), B(x_0,y_0,t), C(x_0,y_0,t), D(x_0,y_0,t), E(x_0,y_0,t)\}$, but for the convenience of writing, in the rest of this paper, we will just refer to them by $\{A,B,C,D,E\}$.

In order to estimate local kinematic operators from input data, we need to optimize a semblance-based cost function (Sun et al., 2021; Sun et al., 2022):

$$S(x_0,y_0) = \frac{\sum_{j=1}^{N}\{\sum_{i=1}^{M} u[x_i,y_i;t_j(x_0,y_0)+\Delta t(x_i,y_i;x_0,y_0)]\}^2}{M\sum_{j=1}^{N}\sum_{i=1}^{M}\{u[x_i,y_i;t_j(x_0,y_0)+\Delta t(x_i,y_i;x_0,y_0)]\}^2}\,, \quad (2)$$

where $u(x_i,y_i;t)$ represents a time sample of the trace located at (x_i,y_i) in the seismic data gather, M is the total number of traces inside the spatial aperture of the local kinematic operator, and N is the total number of time samples within the temporal aperture of the local kinematic operator.

The 2+2+1 method is a solver based upon a local-search method, and it has been used as a pragmatic solution (Bakulin et al., 2020) towards equation (2), which is a highly nonlinear function of parameters $\{A, B, C, D, E\}$. We use Figure 1 to explain this method in detail. NLBF parameter traces, denoted by red dots in Figure 1, parameterize the data space regularly. Green dots in Figure 1 represent seismic traces in the input data, and the red-circled black dot is the NLBF parameter trace that the 2+2+1 method is working on. We refer to the coordinates of this NLBF parameter trace of interest by $\{x_0, y_0\}$. Yellow areas in Figure 1 stand for spatial apertures required by the 2+2+1 method for estimating different parameters of local kinematic operators. We first set $\{B, C, E\}$ at $\{x_0, y_0\}$ to 0s and search for the optimal values of $\{A, D\}$ in a brute-force manner using all the traces existed in its estimation aperture. Next and likewise, parameters $\{B, E\}$ are estimated in the same fashion. As the last step, we fix the parameters $\{A, B, D, E\}$ at their already estimated values, and brute-force search for the optimal value of $\{C\}$ using the traces in its estimation aperture. Although the 2+2+1 method is a local search method, its calculation cost is still significant as it requires three brute-force searches.

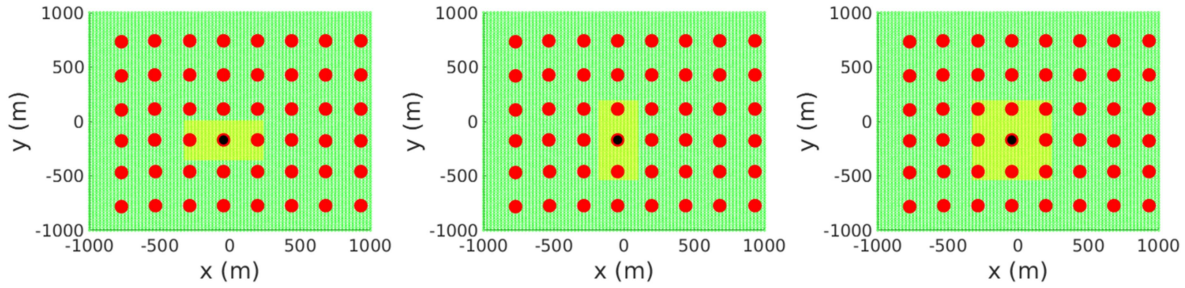

Figure 1: Spatial apertures for estimating parameters (left) {A, D}, (middle) {B, E}, and (right) {C}.

The 2+2+1 CPU / GPU Algorithm

The left picture of Figure 2 presents the 2+2+1 CPU algorithm. Generally speaking, it is a straightforward implementation of the 2+2+1 method introduced before. The "foreach" loop highlighted in green is parallelized via OpenMP for efficiency gains. In other words, the loop over the time direction in an NLBF parameter trace is carried out in parallel by different CPU threads.

Pseudocode for the 2+2+1 CPU Algorithm

Input: a seismic dataset, user-specified parameters to define the solution space for {A, B, C, D, E}
Output: {A, B, C, D, E, S} for each gather in the input dataset, where {S} is the semblance value corresponding to the parameters {A, B, C, D, E}

```
Use user-specified parameters to set search schemes for {A, D}, {B, E} and {C}
Set different apertures (shown in Figure 1) for {A, D}, {B, E} and {C}
foreach gather_i ∈ the input seismic dataset do
    Set locations of NLBF parameter traces for this gather;
    foreach y_j ∈ y direction of NLBF parameter traces do
        foreach x_k ∈ x direction of NLBF parameter traces do
            Collect traces falling into the aperture of {C};
            Collect traces falling into the aperture of {A, D};
            Collect traces falling into the aperture of {B, E};
            foreach t_l ∈ time direction of NLBF parameter traces do
                Set {B, C, E} to 0s and scan {A, D} in their solution space using traces in
                its aperture (Figure 1(a)) to pick out the combination that
                maximizes equation (2);
                Set {A, C, D} to 0s and scan {B, E} in their solution space using traces in
                its aperture (Figure 1(b)) to pick out the combination that
                maximizes equation (2);
                Fix {A, B, D, E} to the estimated values, and scan {C} in its solution space
                using traces in its aperture (Figure 1(c)) to pick out the value that
                maximizes the semblance value S defined by equation (2);
                Save these estimated {A, B, C, D, E, S} at (x_k, y_j, t_l);
            end
        end
    end
end
Output {A, B, C, D, E, S} of all NLBF parameter traces.
```

Pseudocode for the 2+2+1 GPU Algorithm

Input: a seismic dataset, user-specified parameters to define the solution space for {A, B, C, D, E}, heap size and stream amount for the GPU
Output: {A, B, C, D, E, S} for each gather in the input dataset, where {S} is the semblance value corresponding to the parameters {A, B, C, D, E}

```
Use user-specified parameters to set search schemes for {A, D}, {B, E} and {C};
Set different apertures (shown in Figure 1) for {A, D}, {B, E} and {C};
Set the user-specified heap size on the GPU;
Adaptively build the block and grid for the CUDA calculation;
Create a pool of CUDA streams;
foreach gather_i ∈ the input seismic dataset do
    Set locations of NLBF parameter traces for this gather;
    foreach y_j ∈ y direction of NLBF parameter traces do
        foreach x_k ∈ x direction of NLBF parameter traces do
            Fetch an available CUDA stream from the stream pool;
            Collect traces falling into the aperture of {A, D};
            Asynchronously copy these traces in the aperture of {A, D} to the GPU, with the
            fetched stream as the stream identifier;
            Collect traces falling into the aperture of {B, E};
            Asynchronously copy these traces in the aperture of {B, E} to the GPU, with the
            fetched stream as the stream identifier;
            Collect traces falling into the aperture of {C};
            Asynchronously copy these traces in the aperture of {C} to the GPU, with the
            fetched stream as the stream identifier;
            Run the 2+2+1 CUDA kernel to calculate {A, B, C, D, E, S} at (x_k, y_j),with the
            fetched stream as the stream identifier;
            Asynchronously copy {A, B, C, D, E, S} at (x_k, y_j) from the GPU to the memory;
        end
    end
end
Output {A, B, C, D, E, S} of all NLBF parameter traces.
```

Figure 2: Pseudo code for the 2+2+1 CPU algorithm (left) and the 2+2+1 GPU algorithm (right).

In terms of hardware design, a GPU is for a completely different purpose than a CPU (NVIDIA, 2020). A CPU is designed to be a low-latency device to execute a sequence of operations as fast as possible. A GPU, on the other hand, is designed for a high instruction throughput at the cost of slower single-thread performance. In addition, a GPU also comes with a much larger memory bandwidth than a CPU to support its higher instruction throughput. Accordingly, we design our 2+2+1 GPU algorithm

in such a way that allows us to make the best usage of a GPU's high instruction throughput, and it is thus more complex than our CPU algorithm. The right picture in Figure 2 shows the skeleton of our 2+2+1 GPU algorithm. Due to the space limitation, those parts in green will be shown in another paper later. In a nutshell, we exploit the concept of CUDA streams to parallelize the computation burdens of different NLBF parameter traces; we adaptively build thread blocks and block grids for the computation kernel considering the hardware limitations (NVIDIA, 2020), i.e., the x dimension limit of a block grid is 2^{31}-1 and the maximum number of threads in a thread block is 1024; in the computation kernel, every thread block parallelizes the evaluations of equation (2) over concrete solution space of {A, D}, {B, E} and {C}, and finally the 0^{th} thread picks out the best result as the final solution.

Example

We use a challenging 3D single-sensor field dataset from a desert environment (Sun et al., 2022) to demonstrate the efficiency gains achieved by our 2+2+1 GPU version over the CPU version. The hardware environment running the 2+2+1 CPU version is 2 Xeon Gold 6136 CPUs (3.00 GHz, 12 cores), and all 24 cores are actively engaged via OpenMP in our tests. The 2+2+1 GPU version runs on 1 NVIDIA Tesla V100 GPU with 32 GB onboard memory, and only 1 core of a Xeon Gold 6142 CPU (2.60 GHz, 16 cores) is involved. The operation system running on our CPU (GPU) platform is Red Hat 7.x (CentOS 7.x). Both versions are coded in C++, and the compilation environments are: for the 2+2+1 CPU version, we use Intel Parallel Studio XE 2017 Update 2; for the 2+2+1 GPU version, we use CUDA 9.0 Update 4 along with Intel Parallel Studio XE 2017 Update 4.

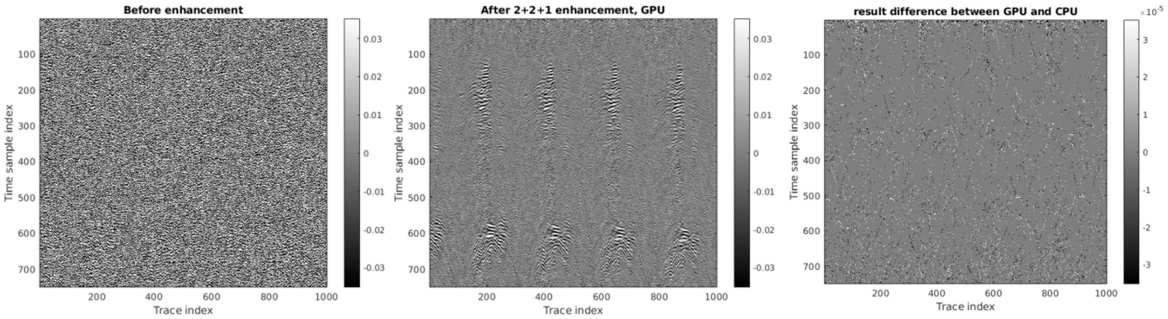

Figure 3: (Left) Several receiver lines of the raw data. (Middle) The same data after enhancement by NLBF with the 2+2+1 GPU version. (Right) Difference between the CPU result and the GPU result, plotted at a different scale (boosted by 1000 than the other pictures).

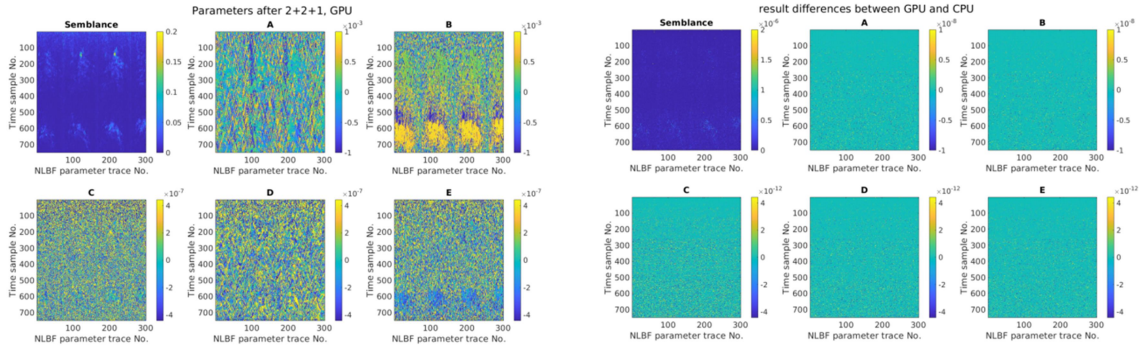

Figure 4: (Left) Semblance values and parameters {A, B, C, D, E} for some selected NLBF parameter traces in the field data example estimated by the 2+2+1 GPU version. (Right) Differences between the CPU results and the GPU results, plotted at different scales (boosted by 10^5 than the left picture).

Several receiver lines of this dataset are shown in the left picture of Figure 3. As this field dataset is heavily contaminated by near-surface noise, no coherent events can be recognized easily. The spacing between NLBF parameter traces is 50 m in both x and y direction. The spatial apertures for different

parameters are: 300 m by 35 m for {A, D}, 35 m by 300 m for {B, E}, and 300 m by 300 m for {C}. We use the format [min : step : max] to denote the search space and the search step for each parameter, and the concrete search schemes are [-10^{-3} s/m : $1.33*10^{-5}$ m/s : 10^{-3} s/m] for {A, B} and [$-4.44*10^{-7}$ s/m^2 : $4.44*10^{-8}$ s/m^2: $4.44*10^{-7}$ s/m^2] for {C, D, E}. The left picture of Figure 4 shows the semblance values and parameters {A, B, C, D, E} estimated by the 2+2+1 GPU version. We further apply NLBF with these estimated local kinematic operators to enhance this challenging dataset, and the corresponding enhanced receiver lines are shown in the middle picture of Figure 3. The quality improvement is clearly visible. The results from the 2+2+1 CPU version are visually identical to these of the GPU version, but due to the space limitation they are not shown here. We further show the result differences between the 2+2+1 GPU version and the 2+2+1 CPU version in the right pictures of Figures 3 and 4. Please note that these differences are plotted at different scales, boosted by 1000 and 10^5, respectively. These result differences are very small at the floating-number level, and they are also understandable as these results are calculated using different hardware with different software environments. In terms of the run time, the 2+2+1 GPU version takes ~147.39 s while the 2+2+1 CPU version takes ~1639.94 s on this field dataset.

Conclusions

In this paper, we introduce our 2+2+1 method for estimating local kinematic operators in the nonlinear beamforming technology and its algorithmic implementations on both the CPU platform and the GPU platform. We take advantages of the CUDA programming language and features of GPUs, including streams, block grids, thread blocks and so on, to effectively accelerate the 2+2+1 method. We demonstrate the efficacy of our 2+2+1 GPU algorithm using a challenging 3D single-sensor field dataset from a desert environment, and an efficiency gain of ~10 is obtained over the 2+2+1 CPU version. This work will benefit data processors to use the nonlinear beamforming technology more widely for enhancing low-quality field data.

References

Bakulin, A., Silvestrov, I., Dmitriev, M., Neklyudov, D., Protasov, M., Gadylshin, K., and Dolgov, V. [2018] Nonlinear beamforming for enhancing prestack seismic data with a challenging near surface or overburden. First Break, **36**, 121-126.

Bakulin, A., Silvestrov, I., Dmitriev, M., Neklyudov, D., Protasov, M., Gadylshin, K., and Dolgov, V. [2020] Nonlinear beamforming for enhancement of 3D prestack land seismic data. Geophysics, **85**, V283-V296.

Berkovitch, A., Deev, K. and Landa, E. [2011] How nonhyperbolic multifocusing improves depth imaging. First Break, **29**, 95-103.

Buzlukov, V. and Landa, E. [2013] Imaging improvement by prestack signal enhancement. Geophysical Prospecting, **61**, 1150-1158.

Hoecht, G., Ricarte, P., Bergler, S. and Landa, E. [2009] Operator-oriented CRS interpolation. Geophysical Prospecting, **57**, 957-979.

Jäger, R., Maan, J., Höcht, G. and Hubral, P. [2001] Common-reflection-surface stack: Images and attributes. Geophysics, **66**, 97-109.

NVIDIA [2020] CUDA C++ Programming Guide. https://docs.nvidia.com/cuda/archive/11.1.0/cuda-c-programming-guide/index.html

Sun, Y., Verschuur, E. and Vrolijk, J. W. [2014] Solving the complex near-surface problem using 3D data-driven near-surface layer replacement. Geophysical Prospecting, **62**, 491-506.

Sun, Y., Silvestrov, I., and Bakulin, A. [2021] An Efficiency-Improved Genetic Algorithm for Enhancing Challenging 3D Prestack Data Using Nonlinear Beamforming. 82nd EAGE Annual Conference & Exhibition, Extended Abstract, https://doi.org/10.3997/2214-4609.202112422.

Sun, Y., Silvestrov, I., and Bakulin, A. [2022] Enhancing 3D land seismic data using nonlinear beamforming based on the efficiency-improved Genetic Algorithm. IEEE Transactions on Evolutionary Computation, accepted.