Brazilian Journal of Geophysics (2020) 38(2): 1–14
© 2020 Sociedade Brasileira de Geofísica
ISSN 0102-261X
www.rbgf.org.br
DOI: 10.22564/rbgf.v38i2.2048

EXTENDING THE USAGE OF GRAPHICS PROCESSING UNITS ON THE CLOUD FOR COST SAVINGS ON SEISMIC DATA REGULARIZATION

Nicholas Torres Okita[1], Tiago A. Coimbra[1], José Ribeiro[1] and Martin Tygel[1]

ABSTRACT. The usage of graphics processing units is already known as an alternative to traditional multi-core CPU processing, offering faster performance in the order of dozens of times in parallel tasks. Another new computing paradigm is cloud computing usage as a replacement to traditional in-house clusters, enabling seemingly unlimited computation power, no maintenance costs, and cutting-edge technology, dynamically on user demand. Previously those two tools were used to accelerate the estimation of Common Reflection Surface (CRS) traveltime parameters, both in zero-offset and finite-offset domain, delivering very satisfactory results with large time savings from GPU devices alongside cost savings on the cloud. This work extends those results by using GPUs on the cloud to accelerate the Offset Continuation Trajectory (OCT) traveltime parameter estimation. The results have shown that the time and cost savings from GPU devices' usage are even larger than those seen in the CRS results, being up to fifty times faster and sixty times cheaper. This analysis reaffirms that it is possible to save both time and money when using GPU devices on the cloud and concludes that the larger the data sets are and the more computationally intensive the traveltime operators are, we can see larger improvements.

Keywords: cloud computing; GPU; seismic processing.

RESUMO. O uso de aceleradores gráficos para processamento já e uma alternativa conhecida ao uso de CPUs multi-cores, oferecendo um desempenho na ordem de dezenas de vezes mais rápido em tarefas paralelas. Outro novo paradigma de computação e o uso da nuvem computacional como substituta para os tradicionais clusters internos, possibilitando o uso de um poder computacional aparentemente infinito sem custo de manutenção e com tecnologia de ponta, dinamicamente sob demanda de usuário. Anteriormente essas duas ferramentas foram utilizadas para acelerar a estimação de parâmetros do tempo de trânsito de Common Reflection Surface (CRS), tanto em zero-offset quanto em offsets finitos, obtendo resultados satisfatórios com amplas economias tanto de tempo quanto de dinheiro na nuvem. Este trabalho estende os resultados obtidos anteriormente, desta vez utilizando GPUs na nuvem para acelerar a estimação de parâmetros do tempo de trânsito em Offset Continuation Trajectory (OCT). Os resultados obtidos mostraram que as economias de tempo e dinheiro foram ainda maiores do que aquelas obtidas no CRS, sendo até cinquenta vezes mais rápido e sessenta vezes mais barato. Esta análise reafirma que é possível economizar tanto tempo quanto dinheiro usando GPUs na nuvem, e conclui que quanto maior for o dado e quanto mais computacionalmente intenso for o operador, maiores serão os ganhos de desempenho observados e economias.

Palavras-chave: computação em nuvem; GPU; processamento sísmico.

Corresponding author: Nicholas Torres Okita

[1]CEPETRO/UNICAMP, R. Cora Coralina, 350 - Cidade Universitária, Campinas - SP, Brazil, 13083-896 – E-mails: nicholas.okita@gmail.com, tgo.coimbra@gmail.com, joseribeiro@ggaunicamp.com, mtygel@gmail.com

INTRODUCTION

Seismic processing programs are compute-intensive software, relying on powerful machines to process large amounts of data. These powerful machines were built as in-house clusters for a long time, recently offering both multi-core processors and graphics processing unit (GPU) accelerators. However, building a cluster demands a big upfront money investment and a long waiting time until the on-site infrastructure is done, which can take months or even years. Due to how fast technology evolves, it can lead to an outdated specification when it starts to run. Furthermore, there are high maintenance costs, such as electric energy bills and technical support. However, we have recently seen an alternative to building these large, expensive machines: cloud computing.

Companies such as Amazon and Microsoft offer part of their data-centers to users as virtual machines (VM). These resources are delivered through the internet and priced differently based on the hardware for an advertised price per hour (usually charged per second of use). This model has a huge advantage over traditional in-house clusters, not only because the user can choose the best hardware for his program instead of relying on a do-it-all solution but also since there are no upfront payments and the hardware is ready in a matter of minutes instead of months. On the flip side, a bad VM selection can lead to undesirable performance and high prices; therefore, the good VM selection is a very important step (see, Okita et al., 2018).

Graphics processing unit accelerators are highly parallel devices with hundreds to thousands of computing threads, which can be used to reduce the execution time of parallel programs while keeping the power usage lower than their CPU counterparts (relative to the same performance). Some public cloud providers have GPU accelerated instances available; hence it is possible to reduce processing time. Since the user pays for what is being used while being used on the cloud, the faster the execution, the lower the billings. This work objective involves using GPU accelerated instances to reduce the execution time and processing cost of a seismic processing algorithm in the Amazon Web Services Elastic Computing Cloud (AWS EC2).

For more details about cloud computing, Armbrust et al. (2010) explains its concept and its pricing schemes in more detail. Furthermore, it points some challenges in the model, such as bottlenecks introduced by data transferring and storage. Emeras et al. (2016) compares the operational costs of using AWS EC2 against a traditional in-house cluster. The authors first estimate the total cost of ownership of their own university's cluster. Then, they perform a linear regression on AWS EC2 prices (based on GFLOPs, memory, storage, and number of GPUs) to estimate a price per hour of their cluster nodes as if they were instances on AWS. They conclude that the price per hour of building the cluster is lower than running the same hardware on the cloud. On a similar aspect, Deelman et al. (2008) analyzes the cost to run a real-life astronomy application on the Amazon cloud infrastructure. The authors propose a series of scenarios of usage, e.g., sporadic computations on the cloud, analyzing the costs of each scenario given their software (including data transfers, storage, and actual computation costs). The authors conclude that when properly allocating the resources required, it is possible to reduce how much is spent using the service. Therefore, using the cloud is not as simple as getting equivalent hardware to what is available locally in a cluster.

This work is the expansion of a previous work published in a conference (Okita et al., 2019) that had the objective of using GPUs to reduce both the time and execution price of seismic processing algorithms in the Amazon Web Services Elastic Computing Cloud (AWS EC2). This work aims to test how GPU accelerated instances perform in a different traveltime other than the ones shown in the original paper and explore the performance increase when using multiple CPU instances and multiple GPUs.

FORMULATION OF THE PROBLEM

In 2D seismic data, we want to estimate traveltime's kinematic parameters, normal-moveout (NMO) velocities, slopes, and curvatures of reflections in both the post-stack and prestack domain. For simplicity, we assume one-component data and events of interest are non-converted primary reflections. The data samples in the data set can be expressed as $u(m,h,t)$, in which u represents the observed amplitude, m is the midpoint location, h is the half-offset distance, and t is the time sample, all these being continuous variables. With that defined, we estimate the parameters in all data samples to identify points of interest, such as reflections and diffractions without a priori identification. This estimation problem can be of great interest in processing tasks, such as data regularization. Specifically, in this work, the Offset-continuation trajectory traveltime operator proposed by Coimbra et al., 2016 is used for both zero-offset stacking and finite-offset data regularization.

Coherency

Reflections and other seismic events have the key property of presenting themselves as coherent signals along a traveltime surface within the seismic data. Since the coherence measure has a quantitative characteristic, it can be used as an objective function; in this case, the estimation problem becomes an optimization problem. We want to find the parameters that maximize the objective function. Alongside that, using coherence allows us to apply a stacking procedure, increasing the amplitude wherever there are points of interest with high coherence and low amplitude in points without coherence.

For this work we use the semblance coherence measure (see, Neidell and Taner, 1971). First, we suppose a given reference data point location (m_0, h_0, t_0) that belonging to a traveltime surface $t = t(m,h)$ defined for midpoint and half-offset pairs (m,h) in the neighborhood of (m_0, h_0). In the same neighborhood, we now consider a given traveltime surface $T = T(m,h;p_1,\cdots,p_n)$ in which p_i are given kinematic parameters. Finally, the semblance between the two traveltime functions is given by

$$S(p_1,\ldots,p_n) = \frac{\sum_{k=-W}^{k=W}\left[\sum_{i=-I}^{i=I}\sum_{j=-J}^{j=J} u_{i,j,k}\right]^2}{N\sum_{k=-W}^{k=W}\left[\sum_{i=-I}^{i=I}\sum_{j=-J}^{j=J} u_{i,j,k}^2\right]} \qquad (1)$$

where $N = (2I+1)(2J+1)$, $u_{i,j,k} = u(m_i, h_j, T_k)$ represents the amplitudes computed at the data points $(m_i, h_j, T_k(m_i, h_j; p_1, \cdots, p_n))$. Notations are as follows:

$$m_i = m_o + i\Delta m, h_j = h_o + i\Delta h,$$
$$T_k(m_i, h_j; p_1 \ldots, p_n) \tag{2}$$
$$= T(m_i, h_j; p_1 \ldots, p_n) + k\Delta t$$

where Δm, Δh, and Δt denote midpoint, half-offset, and uniform time sampling, respectively. Note that at these locations, actual data points may not exist. In this case, these are replaced by interpolated values from neighboring points.

Differential Evolution meta-heuristic

A traditional approach to estimate the parameters that maximize the objective function is brute force through the parameters. The domain is discretized, and every combination of parameters is attempted. Instead of this, a heuristic can be used to produce similar quality results with fewer iterations, such as the works of Barros et al. (2015) and Walda and Gajewski (2017); in this work, we use the Differential Evolution (DE) meta-heuristic (see, Storn and Price, 1997). This heuristic relies on the dynamics of a population. Starting with N_P individuals, each a vector of parameters, distributed randomly in the objective function domain, these individuals interact with each other through processes called mutation and cross over and generate a new individual. This new individual is then compared to the original one, then the one that is closer to the maximum value is kept in the population, while the other is discarded. This process applied to every population member is called a generation. The algorithm executes for a predefined number of generations, picking the individual’s parameters with the best objective function value.

RESULTS

The results are divided into a few subsections, with the first two corresponding to the ZO-OCT stacking and FO-OCT regularization. At the same time, a third one corresponds to a scalability analysis. The Amazon Web Services Elastic Compute Cloud instances are shown in Table 1 are used as hardware options. Their prices are from the On-Demand instances in North Virginia (us-east-1) region during June and July 2019. The qualitative comparison from the parameter estimation is between only one CPU configuration (CPU1) and one GPU configuration (GPU1). The reason being that the discrepancies between different CPU configurations or GPU configurations are negligible. That means the parameters obtained from GPU1 are the same as those obtained from GPU2 and GPU3; similarly, the parameters obtained from CPU1 are the same as those obtained from CPU2 and CPU3.

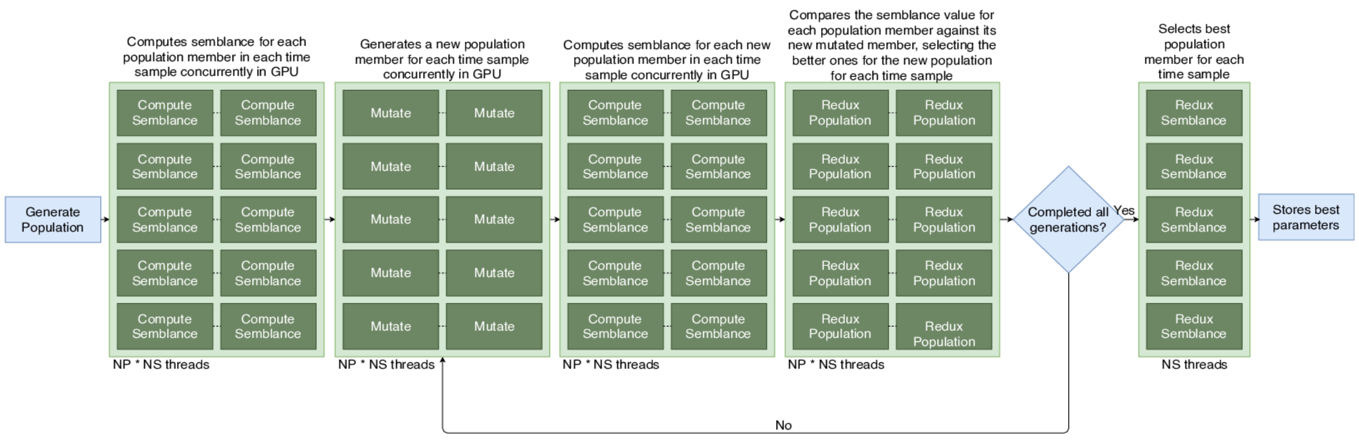

Generate Population
Computes semblance for each population member in each time sample concurrently in GPU
Compute Semblance
NP * NS threads
Generates a new population member for each time sample concurrently in GPU
Mutate
NP * NS threads
Computes semblance for each new population member in each time sample concurrently in GPU
Compute Semblance
NP * NS threads
Compares the semblance value for each population member against its new mutated member, selecting the better ones for the new population for each time sample
Redux Population
NP * NS threads
Completed all generations?
Yes
No
Selects best population member for each time sample
Redux Semblance
NS threads
Stores best parameters

Figure 1 – Flowchart of the Differential Evolution implementation in GPU, illustrating the parallelism in the steps of the Differential Evolution algorithm.

It is worth mentioning that the CPU implementation is done with C++ and compiled using **gcc** with the **-O3** flag, enabling most compiler optimizations. On the other hand, we use CUDA compiled with **nvcc** for the GPU implementation, which provides the best performance for the GPUs we use in our experiments. Furthermore, for execution, we use our inhouse framework called SPITS (see, Benedicto et al., 2017) to parallelize the algorithm, allowing for maximum usage of all cores and GPU accelerators, even on different machines. Finally, the results are obtained from a single execution.

Table 1 – Machine configurations.

Configuration	Processing Unit	RAM (GB)	Price (USD/h)
CPU1 (*c5.18xlarge*)	72 Xeon Platinum 8124 3GHz vCores	144	3.060
CPU2 (*m5.24xlarge*)	96 Xeon Platinum 8175 2.5GHz vCores	384	4.608
CPU3 (*c5.24xlarge*)	96 Xeon Platinum 8124 3GHz vCores	192	4.080
GPU1 (*p3.2xlarge*)	1 Nvidia Tesla V100 16GB	61	3.060
GPU2 (*g3s.xlarge*)	1 Nvidia Tesla M60 8GB	30.5	0.750
GPU3 (*p2.xlarge*)	1 Nvidia Tesla K80 12GB	61	0.900

Zero-Offset OCT

The execution times for the ZO-OCT traveltime in both data sets and their respective prices are shown in Tables 2 and 3. For qualitative analysis, Figure 3 depicts the stacked section for the synthetic (St) data set that was output by CPU1 and GPU1 configurations. The parameters are shown in Figure 4. Similarly, Figure 5 depicts the stacked section for the real (Re) data set, while its parameters are shown in Figure 6.

Table 2 – Execution time and prices for ZO-OCT in the St Data set.

Configuration	Execution Time (minutes)	Price (USD)
CPU1	2.69	0.137
CPU2	2.35	0.181
CPU3	2.02	0.138
GPU1	0.20	0.010
GPU2	0.34	0.004
GPU3	0.41	0.006

Comparing the CPU configurations in the synthetic workload (St) in Table 2, they all have similar execution times with similar prices. On the other hand, we see differences between the GPU configurations regarding the execution time (with

GPU3 taking over twice as long as GPU1 to process the task) and prices (with GPU2 completing the task spending less money). It is noticeable that the biggest difference in performance appears when we compare CPU and GPU configurations, with GPU1 being over ten times faster than any CPU configuration and GPU2 being over thirty times cheaper than the CPU powered instances. However, since this is a small synthetic workload, the time difference appears as only some seconds while the savings are in the order of cents of a dollar.

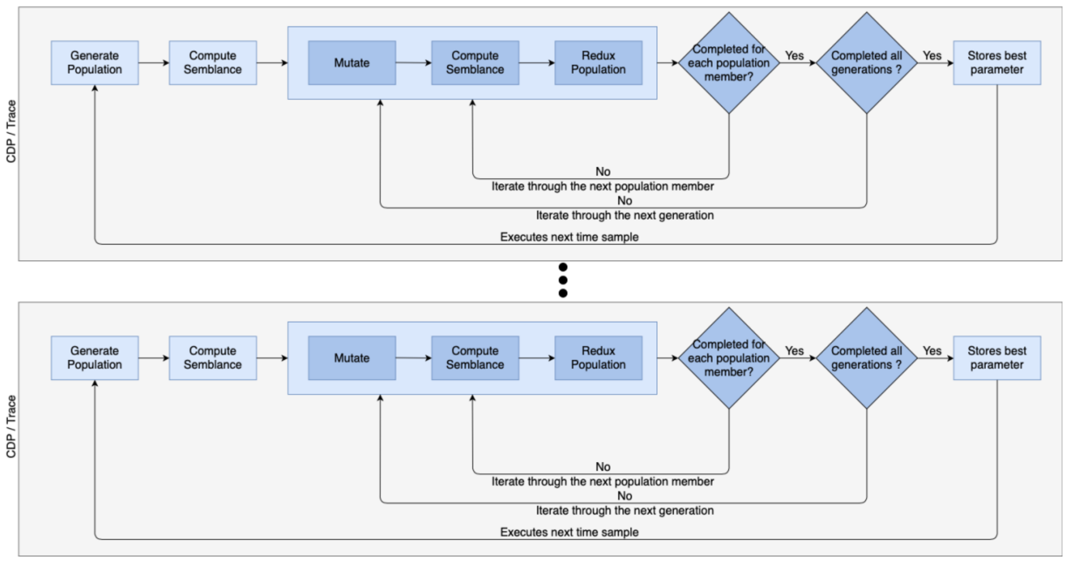

CDP / Trace
Generate Population
Compute Semblance
Mutate
Compute Semblance
Redux Population
Completed for each population member?
Yes
Completed all generations ?
Yes
Stores best parameter
No
Iterate through the next population member
No
Iterate through the next generation
Executes next time sample

Figure 2 – Flowchart of the Differential Evolution implementation in CPU, showing the algorithm parallelism with each CPU thread processing a CDP (in the post-stack domain) or a set of traces (in the prestack domain).

When we compare the results obtained in Figures 3 and 4, it is noticeable that the only difference comes from rounding errors. Therefore, the time and price improvements come at no quality cost.

Table 3 – Execution time and prices for ZO-OCT in the Re Data set.

Configuration	Execution Time (minutes)	Price (USD)
CPU1	269.07	13.722
CPU2	223.11	17.135
CPU3	192.32	13.078
GPU1	6.00	0.306
GPU2	21.63	0.270
GPU3	20.34	0.305

The real workload results, execution times, and prices are shown in Table 3. This time around, it is possible to see improvements when using larger CPU instances, such as CPU3, compared to smaller ones, such as CPU1, with the execution finishing in less than an hour for roughly the same price. On the other hand, even though CPU2 concludes the task faster than CPU1, it still is slower than CPU3 and more expensive than both. When comparing GPU instances, it is noticeable that GPU1 is significantly faster than the other options, albeit at a higher price.

This comparison becomes more interesting as we compare CPU and GPU instances. The fastest GPU instance, GPU1, is over thirty times faster than any CPU option, offering over forty times cost savings, reflecting in time differences in the magnitude of hours and price comparison of over ten dollars compared to dozens of cents. On the other hand, even the cheapest and slowest GPU instance, GPU2, is still around ten times faster than any CPU option and around fifty times cheaper than any CPU option.

Yet again, this performance improvement comes at no quality deterioration. It is notable in Figure 5 that both implementations produced the same results, except for rounding errors. Furthermore, there are no discernible differences when comparing the GPU output from the CPU output shown in Figure 6; therefore, the parameter output also suffered no loss of quality with the massive performance improvement. This comes as no surprise, because the implementation does not change the heuristic itself, but its parallelism; the small differences that appear are expected to happen since the heuristic used to optimize the parameters relies on random mechanisms.

Finite-Offset OCT

Both performance and price improvements from the GPU usage become much more evident when applying regularization using the OCT traveltime. Similarly to the previous zero-offset stacking case, we see massive time and price reductions along with near-identical regularized data set and parameters; these improvements can be seen in Tables 4 and 5, which shows both the execution times and prices obtained from the regularization in data set St and Re, respectively. To analyze the quality impact we show both regularized data sets in Figures 7 and 9, for St and Re, respectively; while their parameters are shown in Figures 8 and 10. To simplify the analysis we only show the result in the offset positioned at 200m for the St data set and 150m for the Re data set.

Table 4 – Execution time and prices for FO-OCT in the St data set.

Configuration	Execution Time (minutes)	Price (USD)
CPU1	45.10	2.300
CPU2	37.90	2.911
CPU3	33.91	2.306
GPU1	2.95	0.150
GPU2	7.86	0.098
GPU3	9.03	0.135

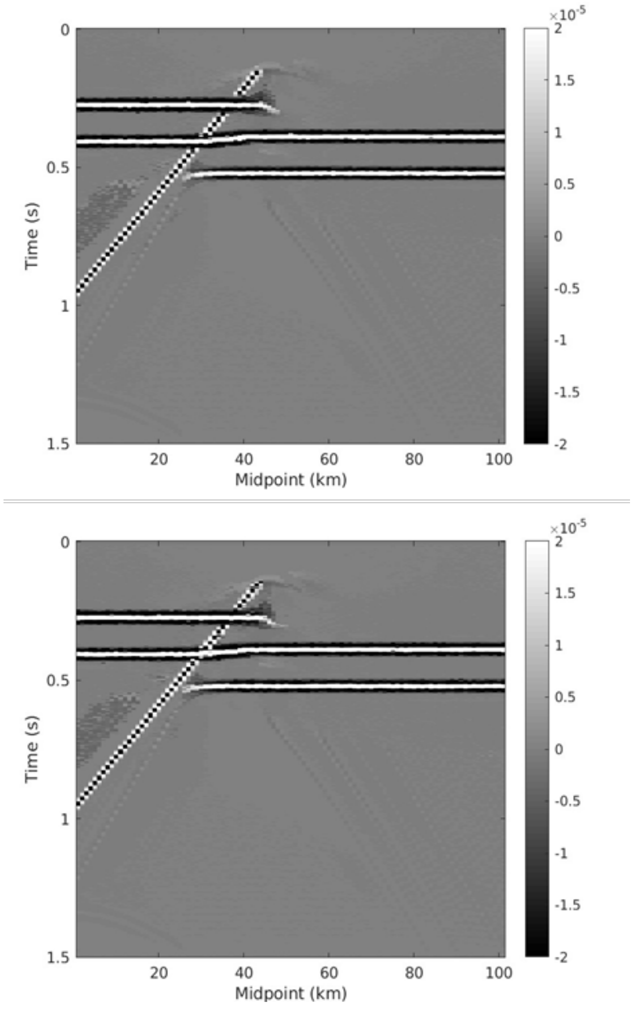
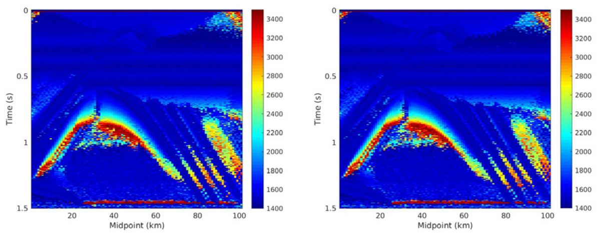
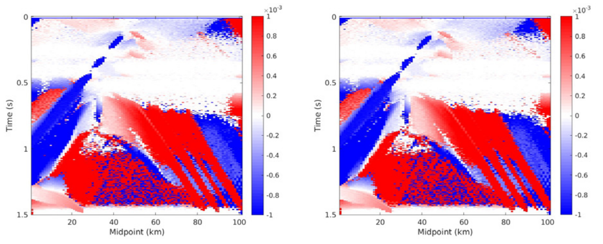


Figure 3 – Comparison between CPU1 (top) and GPU1 (bottom) stacked result for ZO-OCT in data set St.



(a) Velocity parameter in m/s



(b) Angle parameter in s/m

Figure 4 – Comparison between CPU1 (left) and GPU1 (right) estimated parameter result for ZO-OCT in data set St.

Since the data set regularization involves processing the whole data set even in the synthetic workload, we can see a significant difference when comparing CPU instances to GPU instances. The CPU instance with the best performance is CPU3, being only 0.006 USD more expensive than its slower counterpart CPU1, while CPU2 shows middle ground performance between those two, however at a higher price. Among the GPU devices, we see the same behavior shown in the synthetic zero-offset section; GPU1 being the fastest while being more expensive than GPU2, both better choices than GPU3 that is both slower and more expensive than GPU2. Comparing CPU and GPU results we see that we have improvements in dozens of minutes with over tenfold cost savings.

The offset produced by both implementations only differs in rounding errors. The same applies to the parameters. Therefore, we can conclude that, as expected, we can get that massive performance boost without sacrificing the parameters' quality.

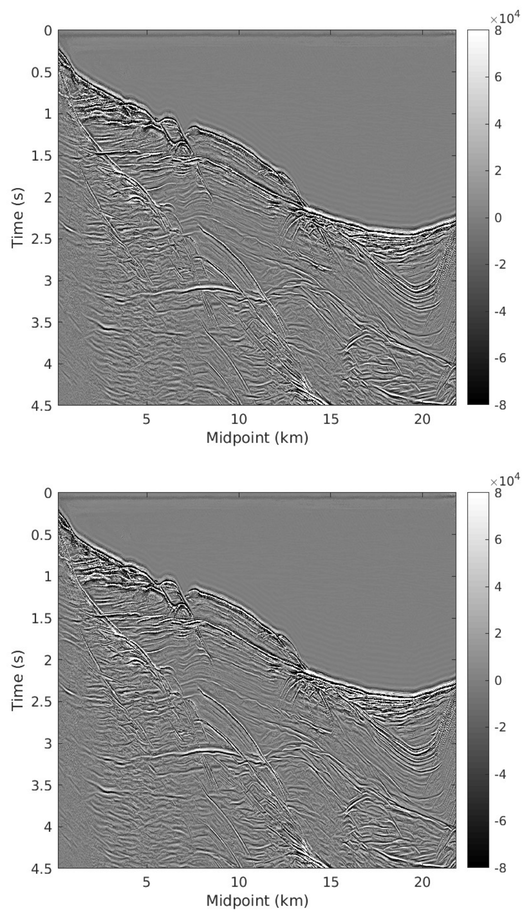
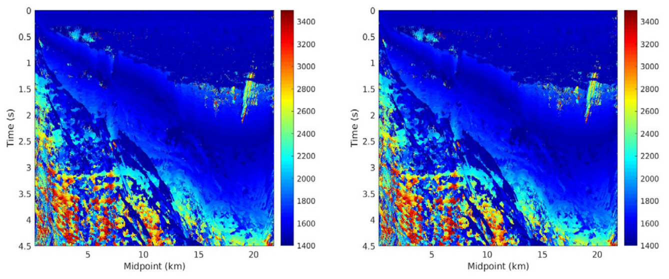
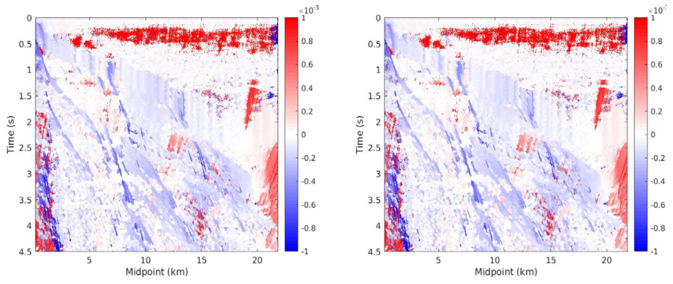


Figure 5 – Comparison between CPU1 (top) and GPU1 (bottom) stacked result for ZO-OCT in data set Re.

Similar to what was seen in the zero-offset stacking section, when we execute the more extensive real data set, the performance difference becomes even more massive. Since we are dealing with a more extensive data set and a more computationally intensive operator, we now see that while every GPU can produce the output in less than one hour, every CPU instance took at least eight hours to process, CPU1 taking about eleven. The price takes a significant cut alongside the execution time reduction, as it becomes noticeable that GPU1 is now over fifty times cheaper than any CPU option and is also the most affordable option to regularize this data set.



(a) Velocity parameter in m/s



(b) Angle parameter in s/m

Figure 6 – Comparison between CPU1 (left) and GPU1 (right) estimated parameter result for ZO-OCT in data set Re.

Finally, as expected, there is no quality degradation neither in the parameters shown in Figure 10, nor in the regularized offset section shown in Figure 9. We imply that the GPU implementation did not affect the metaheuristic quality and massive performance improvement alongside cost savings.

Table 5 – Execution time and prices for FO-OCT in the Re data set.

Configuration	Execution Time (minutes)	Price (USD)
CPU1	669.84	34.162
CPU2	552.16	42.406
CPU3	484.15	32.922
GPU1	13.26	0.676
GPU2	56.75	0.709
GPU3	54.85	0.823

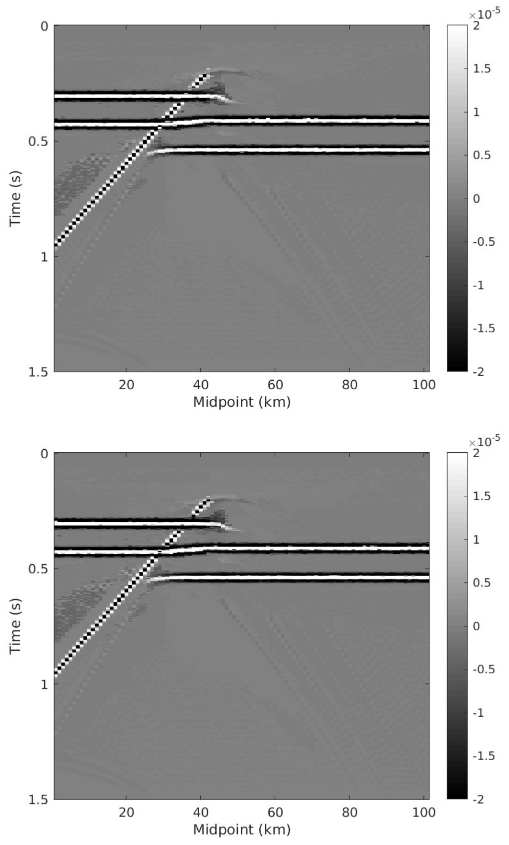


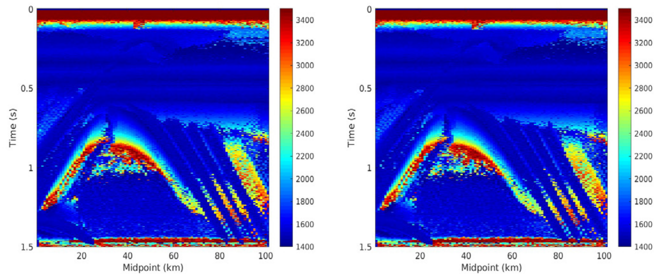
Figure 7 – Comparison between CPU1 (top) and GPU1 (bottom) stacked result for FO-OCT in data set St.

Scalability

This subsection brings a final discussion about scalability, in which we experiment on what kind of performance we can achieve when using multiple CPU instances against numerous GPUs. Table 6 shows the hardware configurations used for these experiments with their respective prices per hour, again disregarding storage prices.

The first experiment again estimates the ZO-OCT stacking operator parameters, with both data sets St and Re. Also, the execution times and prices for each configuration when running data set St are shown in Table 7, while those measurements when running data set Re are shown in Table 8; for comparison reasons, the

CPU3 and GPU1 results from the previous section are also in the table. The parameters and stacked sections are not displayed due to no differences in the previous section.


(a) Velocity parameter in m/s

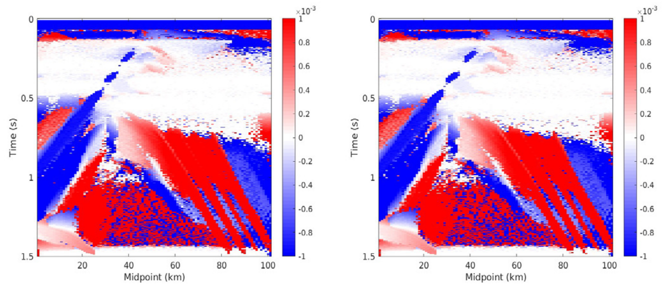

(b) Angle parameter in s/m

Figure 8 – Comparison between CPU1 (left) and GPU1 (right) estimated parameter result for FO-OCT in data set St.

Table 6 – Hardware configuration for scalability tests.

Configuration	Processing Unit	RAM (GB)	Price (USD/h)
MULCPU1 (2×*c5.24xlarge*)	192 Xeon Platinum 8124 3GHz vCores	384	9.060
MULGPU1 (*p3.8xlarge*)	4 Nvidia Tesla V100 16GB	244	12.240
MULGPU2 (*p3.16xlarge*)	8 Nvidia Tesla V100 16GB	488	24.480

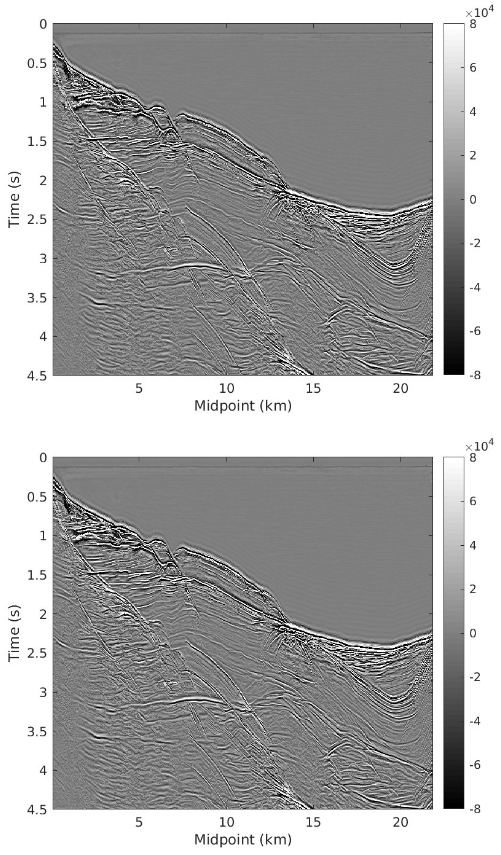
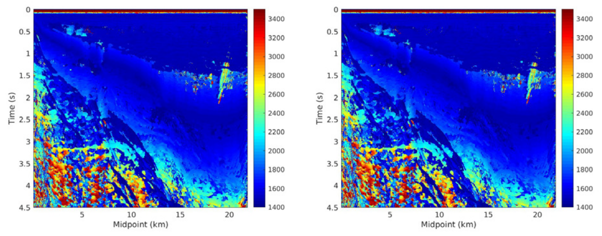


Figure 9 – Comparison between CPU1 (top) and GPU1 (bottom) stacked result for FO-OCT in data set Re.


(a) Velocity parameter in m/s

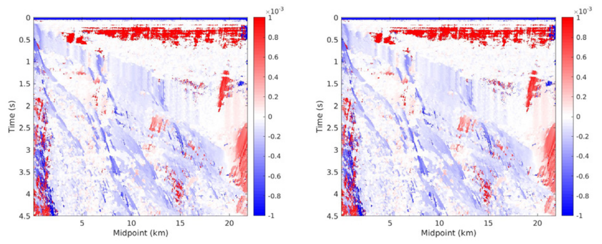

(b) Angle parameter in s/m

Figure 10 – Comparison between CPU1 (left) and GPU1 (right) estimated parameter result for FO-OCT in data set Re.

Table 7 – Execution time and prices for ZO-OCT in data set St.

Configuration	Execution Time (minutes)	Price (USD)
CPU3	2.02	0.138
MULCPU1	1.75	0.238
GPU1	0.20	0.010
MULGPU1	0.17	0.035
MULGPU2	0.30	0.124

Comparing the scalability of CPU instances in Table 7, it is noticeable that there is almost no performance impact when using multiple CPUs instances compared to a single one; the same behavior can be seen in GPU instances. On the other hand, the price increases to the same proportion as we add more GPUs or CPU instances. This problem is that the data set is too small; therefore, adding more instances to work on the estimation would not improve performance, as the bottleneck is not in the estimation process anymore.

Table 8 – Execution time and prices for ZO-OCT in data set Re.

Configuration	Execution Time (minutes)	Price (USD)
CPU3	192.32	13.078
MULCPU1	102.16	13.894
GPU1	6.00	0.306
MULGPU1	1.05	0.215
MULGPU2	0.80	0.325

Differently from the St data set, the significantly larger Re data set produces noticeable performance improvements, as seen in Table 8. As expected, when using double the number of CPU instances, we see that the execution time drops in about fifty percent. Following the same pattern, we see nearly eight performance improvement improvements when using eight GPUs and, surprisingly, about five performance improvement times using four GPUs.

The explanation for the superlinear performance scalability when using configuration MULGPU1 compared to GPU1 relies on more details from the Amazon Web Services cloud infrastructure. Briefly, it is explained by using different availability zones when creating the instances, with GPU1 and MULGPU2 running on zone us-east-1b, while MULGPU1 ran on zone us-east-1f. Even though there should be no differences when using instances from different availability zones within the same region, this does not happen, with some instances in some zones outperforming the same hardware configuration in other zones. The performance variation between different availability zones and in different moments in time is an already known challenge when using cloud computing (see, Schad et al., 2010). However, such variations do not impact the overall conclusion, even though they were obtained from a single execution, as we observe orders of magnitude of differences.

With these linear performance increases, the MULCPU and MULGPU configurations' execution price also remains about the same as the CPU and GPU ones; since the price per hour is also directly proportional to the number of instances being deployed (or the number of GPUs in an instance). Notably, there is also a price increase when using multiple instances caused by waiting for the new instance to boot and load the data before the estimation process. The only exception to that small increase in price is MULGPU2. Due to its superlinear performance improvement, it ends up costing even less than GPU1.

The second experiment performed is the OCT regularization in both data sets St and Re, with their results shown in Tables 9 and 10, respectively. Again the parameters and regularized sections are not displayed since there is no difference to those obtained in the previous section.

Table 9 – Execution time and prices for FO-OCT in data set St.

Configuration	Execution Time (minutes)	Price (USD)
CPU3	33.91	2.306
MULCPU1	17.15	2.332
GPU1	2.95	0.150
MULGPU1	0.80	0.164
MULGPU2	0.67	0.271

The regularization performed in data set St has its execution times and prices shown in Table 9. As expected, MULCPU1 cuts the execution time to around 50% of the original CPU3 time, with a very similar price; this happens due to the price per hour being twice as expensive as the CPU3 configurations while the execution time is only half the original. Similar behavior happens in the GPU configurations, in which we can see MULGPU1 performing four times faster than GPU1, without drastically increasing the price. In MULGPU2, on the other hand, we see that the performance increase was below the expected; along with it comes a larger increase in price, since it is eight times more expensive per hour than GPU1, while its performance is only around 4.5 times faster.

Table 10 – Execution time and prices for FO-OCT.

Configuration	Execution Time (minutes)	Price (USD)
CPU3	484.15	32.922
MULCPU1	248.29	33.768
GPU1	13.26	0.676
MULGPU1	2.45	0.500
MULGPU2	1.75	0.715

Finally, with a larger data set such as the Re data set, we can see the performance improvements as we increase the number of machines. Notably, the MULCPU1 configuration is still around twice as fast as the CPU3 configuration, both at very similar prices. Regarding the GPU configurations, MULGPU1 presents a superlinear performance (being over four times faster than the GPU1 configuration with four times the number of GPUs; the reason was briefly aforementioned) MULGPU2 is about eight times faster than the GPU1 configuration. Regarding pricing, MULGPU2 has a very similar price to GPU1, while MULGPU1, with its superlinear performance increase, offers the cheapest option among all configurations.

After analyzing how each configuration performs, we can conclude that adding more CPU instances or GPU instances to perform the processing task does not affect their pricing, given that the task can scale linearly. Therefore, GPU powered instances still are far superior to their CPU counterparts both in execution time and in price, with over one hundred times faster execution times with dozens of times lower prices. Notably, as we increase the number of instances being used, we also see small increases in prices. There is a constant price regarding booting the instance and loading the data before the actual processing. Furthermore, one must be careful with scaling limits since smaller data sets (e.g., the St data set) or simpler traveltimes operators (e.g., zero-offset stacking) can benefit little or even lose performance when adding more instances, leading to higher charges than necessary.

Memory usage

A final important point to highlight is memory usage, as large datasets are expected to require more memory. Also, it is noteworthy that GPU devices have considerably less memory than CPU systems, which can be seen in Tables 1 and 6, where the GPU memory is capped at 16GB, while the system memory can be in the order of hundreds of gigabytes. Therefore, the challenge of high memory requirements become more significant with such devices. It is possible to use a unified memory approach to allow a single device to access the whole memory system plus all the other GPUs' memory, dealing with the aforementioned challenge. The advantage of using such a scheme is a robust page faulting and migration mechanism, implemented on recent GPU architectures (starting with Nvidia Pascal), where data is migrated on-demand to where it is accessed. Hence, enabling the usage of more memory than what is usually available to on devices, improving code transparency, and freeing the developer from manual memory management (see, Harris, 2013). However, those benefits come at a possible cost of transfer time overhead due to the data movement from device to system memory, which can negatively affect execution time. However, if the memory required fits in the GPU memory, there is no impact on performance.

CONCLUSION

In this work, we accelerated both the ZO-OCT stacking and FO-OCT regularization with GPU devices, exploring how the performance improves. Furthermore, by using the cloud, we were able to analyze how the price can be affected as well, since, in the pay-as-you-go model, the user only pays for how long the instance is being used, then lower execution times can lead to lower prices.

The GPU implementation provided the same quality results with times up to ten times lower than the CPU implementation, along with cost savings in the order of thirty times. Furthermore, even the smallest GPU configurations were able to beat all high-end CPU configurations, resulting in less time and a fraction of the price. A final point of analysis was the scalability potential, which was performed by running the same program and data sets with multiple CPU instances and multiple GPU devices. While we saw a performance increase in more massive data sets, the price remained very similar to executing the problem with only a single CPU instance or a single GPU instance. In smaller data sets, we did not see any performance increase or even saw performance degradation; this leads to higher charges proportional to the number of new instances created (or GPU devices used). Finally, we observe that there are limits to how much the program can scale depending on the input; therefore, an analysis of the optimal number of instances may be required.

In conclusion, the OCT traveltime benefits mainly from GPU devices, drastically decreasing the time spent processing and delivering low prices on the cloud. Not only that but in large workloads, the programs can scale well (with either multiple CPUs or GPUs), delivering significantly faster execution prices at a very similar price.

ACKNOWLEDGMENTS

This work was possible thanks to the support of Petrobras and Fapesp (2013/08293-7).

REFERENCES

Armbrust, M., Fox, A., Griffith, R., Joseph, A. D., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., Stoica, I., and Zaharia, M., 2010, A view of cloud computing: Communications of the

ACM, 53, no. 4, 50–58.

Barros T., Ferrari, R., Krummenauer, R., and Lopes, R., 2015, Differential evolution-based optimization procedure for automatic estimation of the common-reflection surface traveltime parameters: Geophysics, 80, no. 6, WD189–WD200.

Benedicto, C., Rodrigues, I.L., Tygel, M., Breternitz, M., and Borin, E., 2017, Harvesting the computational power of heterogeneous clusters to accelerate seismic processing: 15th International Congress of the Brazilian Geophysical Society, Rio de Janeiro, Brazil.

Coimbra, T. A., Novais, A., and Schleicher, J., 2016, Offset-continuation stacking: Theory and proof of concept: Geophysics, 81, no. 5, V387–V401.

Deelman, E., Singh, G., Livny, M., Berriman, B., and Good, J., 2008, The cost of doing science on the cloud: The montage example: 2008 SC - International Conference for High Performance Computing, Networking, Storage and Analysis, Austin, Texas.

Emeras, J., Varrette, S., Plugaru, V., and Bouvry, P., 2016, Amazon elastic compute cloud (EC2) versus in-house HPC platform: A cost analysis: IEEE Transactions on Cloud Computing, 7, no. 2, 456– 468.

Harris, M., 2013, Unified memory in CUDA 6: NVIDIA Developer Blog. Access on September 29th 2020. Available on: https://developer.nvidia.com/blog/unified-memoryin-cuda-6/.

Neidell, N. S., and Taner, M. T., 1971, Semblance and other coherency measures for multichannel data: Geophysics, 36, no. 3, 482–497.

Okita, N., Coimbra, T., Rodamilans, C., Tygel, M., and Borin, E., 2018, Using SPITS to optimize the cost of high-performance geophysics processing on the cloud: First EAGE Workshop on High Performance Computing for Upstream in Latin America, Santander, Colombia.

Okita, N., Coimbra, T. A., Ribeiro, J., and Tygel, M., 2019, Using graphics processing units on the cloud to accelerate and reduce processing cost of parameters estimation of seismic processing algorithm: 16th International Congress of the Brazilian Geophysical Society, Rio de Janeiro, Brazil.

Schad, J., Dittrich, J., and Quiane-Ruiz, J.-A., 2010,´ Runtime measurements in the cloud: Observing, analyzing, and reducing variance: Proc. VLDB Endow., **3**, no. 1-2, 460–471.

Storn, R., and Price, K., 1997, Differential evolution a simple and efficient heuristic for global optimization over continuous spaces: Journal of Global Optimization, 11, no. 4, 341–359.

Walda, J., and Gajewski, D., 2017, Determination of wavefront attributes by differential evolution in the presence of conflicting dips: Geophysics, 82, no. 4, V229–V239.

Recebido em 16 de outubro de 2019 / Aceito em 18 de novembro de 2020

Received on October 16, 2019 / Accepted on November 18, 2020