Computers and Geosciences 124 (2019) 37–45

Contents lists available at ScienceDirect

Computers and Geosciences

journal homepage: www.elsevier.com/locate/cageo

Accelerating geostatistical seismic inversion using TensorFlow: A heterogeneous distributed deep learning framework

Mingliang Liu*,[1], Dario Grana[2]

Department of Geology and Geophysics, University of Wyoming, 1000 E. University Ave, Laramie, WY, 82071, USA

ARTICLE INFO

Keywords:
Seismic inversion
Geostatistics
TensorFlow
GPU

ABSTRACT

Geostatistical seismic inversion is one of emerging technologies in reservoir characterization and reservoir uncertainty quantification. However, the challenge of intensive computation often restricts its application in practical studies. To circumvent the computational limitation, in this work, we present a distributed parallel approach using TensorFlow to accelerate the geostatistical seismic inversion. The approach provides a general parallel scheme to efficiently take advantage of all the available computing resources, i.e. CPUs and GPUs: the computational workflow is expressed and organized as a Data Flow Graph, and the graph can be divided into several sub-graphs which are then mapped to multiple computing devices to concurrently evaluate the operations in them. The high-level interface and the feature of automatic differentiation provided by TensorFlow also makes it much easy for users to implement their algorithms in an efficient parallel manner, and allows employing programs on any computing platform almost without alteration. The proposed method was validated on a 3D seismic dataset consisting of 600 × 600 × 200 grid nodes. The results indicate that it is feasible to the practical application and the computational time can be largely reduced by using multiple GPUs.

1. Introduction

The objective of seismic reservoir characterization is to estimate elastic and petrophysical properties of interest from seismic and well log data (Doyen, 2007). In the last decades, a variety of seismic inversion methods have been proposed, including deterministic and stochastic algorithms (Tarantola, 2005; Aster et al., 2011). In deterministic approaches to seismic inversion, the aim is to find a set of reservoir properties that minimize the synthetic-to-seismic misfit by using numerical optimization techniques, such as gradient-based methods. The deterministic approaches are generally easy to implement and computationally fast, but provide only a single estimate of subsurface parameters that is unrealistically smooth and might exaggerate the reservoir connectivity. However, the misfit functions of geophysical inverse problems are not necessarily monotonically convex, and thus deterministic solutions might fall into local optimums and fail to describe the subsurface reservoir accurately. In addition, due to the limited bandwidth of measured data (in particular, the absence of low frequencies) and the low signal-to-noise of seismic data, solutions of seismic inverse problems are inherently non-unique, which implies that theoretically there are an infinite number of reservoir realizations that honor the available measurements. Alternatively, stochastic inversion schemes allow generating multiple plausible highly-detailed realizations conditioning on the measurements and investigating the associated model uncertainties through the empirical covariance matrix. Stochastic inversion algorithms include simulated annealing (Sen and Stoffa, 1991), genetic algorithms (Mallick, 1995), gradual deformation (Hu, 2000), probability perturbation method (Caers and Hoffman, 2006; Grana et al., 2012), and ensemble-based methods (Gineste and Eidsvik, 2017; Thurin et al., 2017; Liu and Grana, 2018).

Geostatistical seismic inversion is an inversion method in which multiple prior reservoir models are simulated using geostatistical algorithms, and then updated to honor geophysical observations by deterministic or stochastic optimization methods (Bortoli, 1992; Haas and Dubrule, 1994; González et al., 2007; Bosch et al., 2010; Azevedo et al., 2015; Azevedo and Soares, 2017). The posterior reservoir realizations can be used not only for model uncertainty quantification, but also as starting models in history matching to be further refined by assimilating production data or/and time-lapse seismic. However, due to the size of seismic data and the large number of reservoir realizations in the geostatistical seismic inversion, this method becomes computationally prohibitive for real datasets, which largely limits its application in

* Corresponding author.
E-mail address: mliu4@uwyo.edu (M. Liu).
[1] Mingliang Liu developed the methodology, implemented the code, and applied it to a real dataset.
[2] Dario Grana supervised the research project and contributed to the preparation of the manuscript.

https://doi.org/10.1016/j.cageo.2018.12.007
Received 29 April 2018; Received in revised form 1 October 2018; Accepted 20 December 2018
Available online 21 December 2018
0098-3004/

practice. Therefore, it is necessary to accelerate the inversion method to make it computationally efficient for practical applications. With the recent advent of high-speed multi-core CPUs and GPUs, specialized parallel algorithms for geostatistical seismic inversion have been developed for the high-performance parallel computing platforms. Vargas et al. (2007) proposed to parallelize the sequential simulation algorithm by subdividing the study area into small spatial domains, and then independently simulating on different processors. Since the simulation of each node does not consider the currently simulated nodes, the result is not exactly the same as the original sequential algorithm. Alternatively, Nunes and Almeida (2010) proposed a parallelization scheme by using a functional decomposition to split the simulation algorithm into single sub-tasks, and then implement parallelly using multi-threads with multi-cores CPUs. Tahmasebi et al. (2012) exploited the graphics computational units (GPU) to reduce the execution time using the Compute Unified Device Architecture (CUDA). Huang et al. (2013) improved the computational performance of Direct Sampling method for multiple-point geostatistics simulations with the use of GPUs. Ferreirinha et al. (2015) introduced a generic strategy based on OpenCL to efficiently use multiple computational devices (CPUs and GPUs) for stochastic seismic inversion. Similar parallel algorithms have been presented by Mariethoz (2010) and Peredo et al. (2014).

In this paper, we propose a distributed parallel method to accelerate the geostatistiscal seismic inversion using TensorFlow, an open-source heterogeneous distributed deep learning framework developed by Google and first released to the public at the end of 2015. Although TensorFlow is designed with the hopes of speeding up deep learning by providing a simple-to-use and computationally efficient infrastructure, its generic architecture and extensibility make it applicable to any numerical problems that can be expressed as a Data Flow Graph. One of the advantages of TensorFlow is that it efficiently and easily exploits heterogeneous distributed computational devices, i.e. CPU, GPU and TPU (Tensor Processing Unit developed by Google to specifically speed up tensor calculations). It provides an extremely efficient and user-friendly platform for compute-intensive applications. With such platform programs can be easily deployed to a scalable computing environment with a uniform Application Programming Interface (API). In TensorFlow, underlying computing devices are transparent to users, and computational tasks are scheduled by the dedicated distributed execution engine. In contrast with MPI and CUDA, users no longer need to explicitly handle data transmission, memory management and message passing between devices: the algorithms are expressed in terms of Data Flow Graph and the underlying operations are completely implicit. Other frameworks, i.e. PyTorch and MXNet, could also be applied, alternatively to TensorFlow.

Seismic inversion, geostatistical simulations and uncertainty quantification in reservoir modeling generally require a large computational effort due to the size of seismic datasets and the dimension of the model space. The proposed reservoir characterization workflow requires the generation of a large number of geostatistical realizations of the reservoir model and the optimization of the models to match the measured seismic data. The simulations are obtained using geostatistical algorithms and the optimization is performed using gradient-based methods but could also be obtained using stochastic perturbation algorithms. The implementation using stochastic optimization methods generally allows avoiding local minima of the objective function; however, the proposed gradient-based inversion method using multiple initial models allows exploring the model space with a smaller computational effort thanks to the parallelization approach. Indeed, with the advent of TensorFlow, it becomes simple and flexible to accelerate the geostatistical seismic inversion in three parallel strategies. First, geostatistical simulation and seismic inversion typically involve many large-scale matrix operations that can be massively parallelized, and thus computed on GPUs; secondly, as multiple reservoir realizations are used in the geostatistical inversion, we can utilize multiple CPUs or GPUs to simulate and invert the models simultaneously (model parallelism); thirdly, 3D seismic measurements generally include enormous volumes of data (tens or hundreds of gigabytes), which might be beyond the capacity of a single computing device. In this case, we can subdivide the 3D seismic data into several small sub-volumes and process them across multiple CPUs or GPUs (data parallelism). The new contribution is in the parallelization of the seismic inversion problem and the computational advantage of the proposed workflow.

One of the advantages of TensorFlow is its fast auto-differentiation ability that allows computing the gradients and Jacobians of any variables without analytically deriving the necessary partial derivatives that is usually time-consuming and error prone. Therefore, any gradient-based method can benefit from this feature. Combining auto-differentiation with the set of first-rate optimizers implemented in TensorFlow, model parameters can be automatically and iteratively updated according to the selected objective function. It is also possible to apply other user-defined or optimizers from third-party libraries.

Overall, with the advantages of distributed computation and auto-differentiation, TensorFlow allows us to implement the geostatistical seismic inversion algorithm in parallel in a simple and accessible way. At present, TensorFlow provides a high-level API for building and executing a Data Flow Graph in various languages, including Python, C++, Java and Go, among which the Python API is the most complete and easiest one to use due to the advantages of multi-platform compatibility, readable syntax, and easy integration with other scientific computing libraries implemented via C, C++ or Fortran. The backend and core modules of TensorFlow are written in C/C++, therefore, independently from the API used for Data Flow Graph in the geostatistical seismic inversion, the code will be converted to C/C++ and then compiled to machine code transparently, hence boosting the performance.

2. Overview of TensorFlow

TensorFlow is currently the most widely used deep learning framework. It was initially designed to simplify the construction of deep neural networks and speed up the learning process with a heterogeneous distributed computational environment, and then became a more generic library for numerical computation, making easy large-scale numerical optimization problems, i.e. inverse problems and solutions of partial differential equations. Different with the traditional numerical libraries, TensorFlow adopts Data Flow Graph, which is a common programming model in the fields of cloud computing and machine learning, to express and organize the computation workflow, and then map the mathematical operations in the graph to different computational devices (CPUs, GPUs, and TPUs). As illustrated in Fig. 1, the design of TensorFlow is highly modular: users build the Data Flow Graph by the front end, i.e. Python, C++, Java and Go, and the core and distributed execution engine is responsible for dispatching different computing operations in the Data Flow Graph to the underlying computing devices. Corresponding to the three layers in Fig. 1, there are three important roles in a TensorFlow program, client, master and

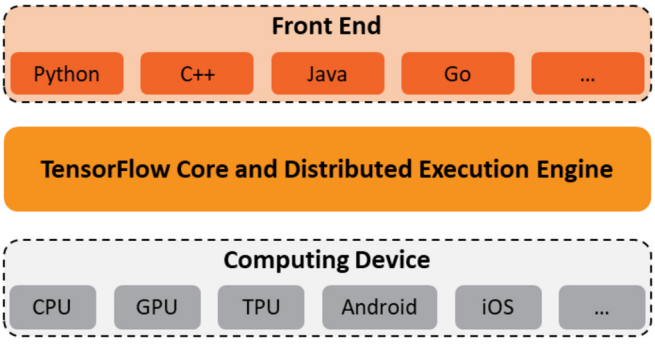

Fig. 1. Architecture of TensorFlow.

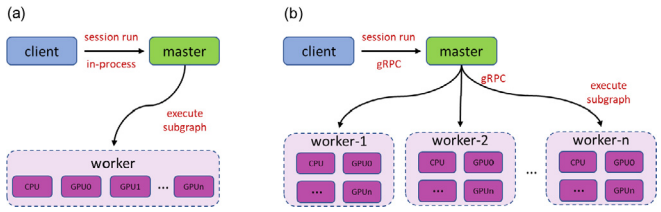

Fig. 2. Execution mechanism of TensorFlow program: (a) Standalone mode and (b) distributed mode (Abadi et al., 2015).

worker. As illustrated in Fig. 2, the client builds the computation graph and creates a session to send the defined graph to the master. Then, the master prunes and partitions the graph into multiple pieces so as to delivers them to worker services for execution. In the standalone mode, client, master and worker are located in the same machine, while in the distributed mode, the three roles can be located in different machines and the communication between them is through gPRC, a high performance, open-source universal remote procedure call framework (Abadi et al., 2015). This architecture provides a uniform API to make low level modules and devices transparent to users, which not only frees us from the cumbersome and challenging tasks of parallel programming, but also make it possible to migrate the application from one computing platform to others almost without modification. For all these strengths, both the development and maintenance costs of applications could be greatly reduced.

In TensorFlow, the Data Flow Graph models a program as a directed graph in which nodes represent mathematical operations and edges represent the multidimensional data arrays (tensors) that flow between the nodes (Martin and Estrin, 1967; Yourdon and Constantine, 1979). For example, Fig. 3 depicts the Data Flow Graph that represents a simple linear regression model

$$\mathbf{Y} = \mathbf{W} * \mathbf{X} + \mathbf{b} \tag{1}$$

where $\mathbf{X}$ and $\mathbf{Y}$ represent the input and output respectively; $\mathbf{W}$ and $\mathbf{b}$ represent the weights and bias that we want to estimate. The matrix multiplication in the Data Flow Graph corresponds to a single node that connects two inputs ($\mathbf{W}$, $\mathbf{X}$) and an output (the result of $\mathbf{W}$*$\mathbf{X}$); similarly, the add operation corresponds to a node with two inputs ($\mathbf{W}$*$\mathbf{X}$, $\mathbf{b}$) and an output ($\mathbf{Y}$). To better understand, debug, and optimize programs, TensorFlow also provides a suite of visualization tools, namely TensorBoard, to visualize the Data Flow Graph and plot quantitative metrics about the execution of the graph.

The mechanism of Data Flow Graph brings several advantages to TensorFlow. First, constructing Data Flow Graph is like building blocks, which makes the model very flexible to be modified and extended by adding or removing components in the graph. Secondly, this mechanism is inherently parallel and distributed, and can work well in large-scale numerical problems: it is easy for TensorFlow to identify operations that can execute in parallel, and we can also explicitly subdivide the Data Flow Graph into several sub-graphs to execute them on different devices simultaneously. Thirdly, the Data Flow Graph is a language-independent representation of models. One can easily build the model in Python or other languages, and then deploy it to a variety of computing platforms. This “code one and run everywhere” nature provides a high scalability across computing machines.

Once the Data Flow Graph is built and gets started in a session, the TensorFlow core and distributed execution engine will map operations in the graph that are actually needed to the underlying computing devices to fetch the variables of interest. Thanks to the built-in feature of auto-differentiation, the variables can be updated automatically and iteratively by minimizing the pre-defined objective function through the backpropagation algorithm (Rumelhart et al., 1986) which is the workhorse of learning in neural networks.

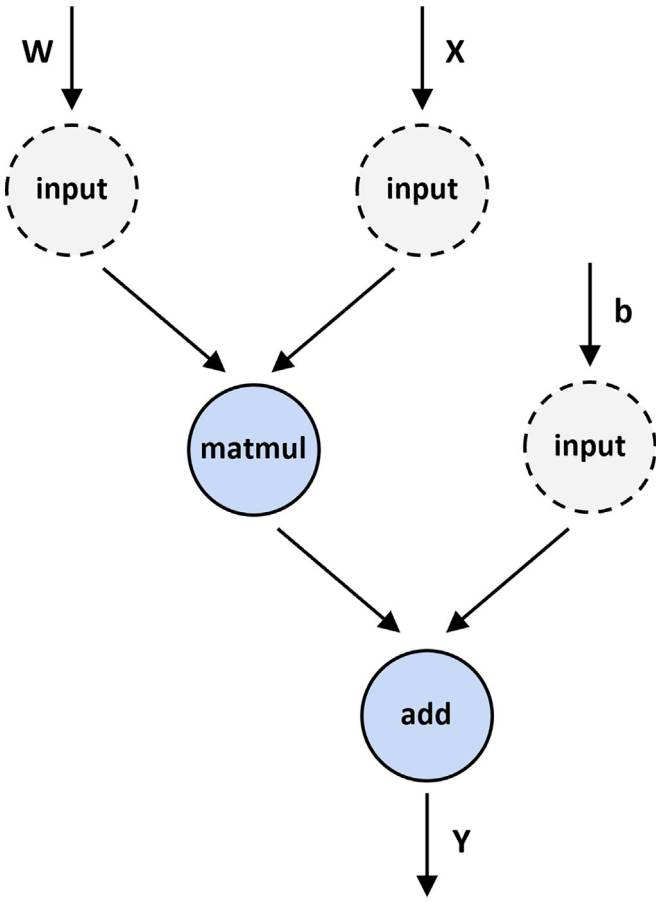

Fig. 3. Data Flow Graph of the linear regression model described as Equation (1).

3. Geostatistical seismic inversion

The estimation of elastic properties from seismic data is a common inverse problem in geophysics, which can be written in the following general form

$$\mathbf{d} = \boldsymbol{f}(\mathbf{m}) + \mathbf{e} \tag{2}$$

where $\mathbf{d}$ represents the seismic data, $\mathbf{m}$ represents the set of elastic attributes, $\boldsymbol{f}$ represents the seismic forward model mapping $\mathbf{m}$ into $\mathbf{d}$; and $\mathbf{e}$ is the measurement error. The goal of seismic inversion is to estimate the unknown model variables $\mathbf{m}$ from $\mathbf{d}$ assuming that the forward model $\boldsymbol{f}$ is a good approximation of the physical relations between model properties and data.

Machine learning and inverse theory methods are tools for parameter estimation (Kappler et al., 2005). Indeed, we can apply the framework and optimization methods of machine learning to solve inverse problems. In this work, we aim to introduce a state-of-the-art heterogeneous distributed computing framework for the acceleration of seismic inversion. To evaluate the computational performance, we focus on a geostatistical post-stack inversion application to compare the execution times on different computing environments. The inversion can be extended to pre-stack inversion, full waveform inversion, reservoir history matching and any other model-based inverse problems.

In post-stack seismic inversion, the model parameter of interest is acoustic impedance and the forward operator is generally a convolutional model of the form

$$\mathbf{d} = \mathbf{w} * \mathbf{r} + \mathbf{e} = \mathbf{w} * \mathbf{g}(\mathbf{I}_p) + \mathbf{e} \tag{3}$$

where $\mathbf{w}$ is the wavelet estimated from well logs and seismic, $\mathbf{r}$ represents the reflection coefficients calculated from the acoustic

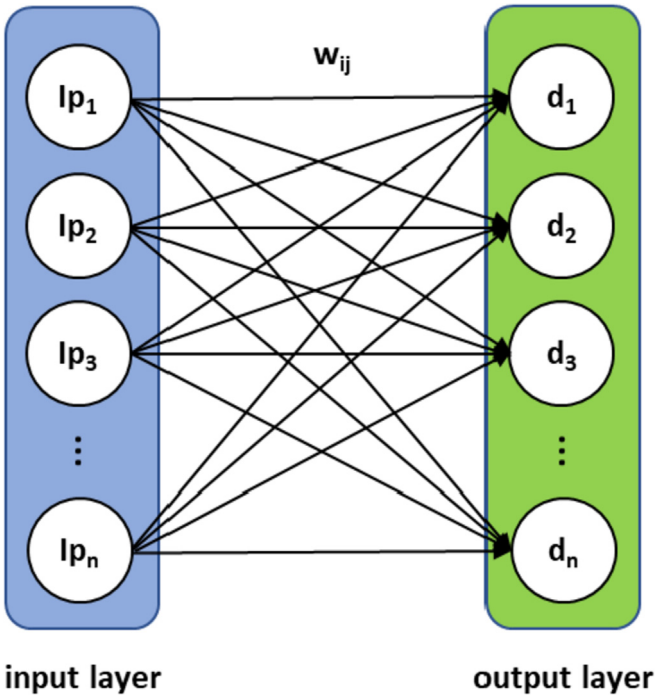

Fig. 4. Two-layer neural network corresponding to the convolution forward model in seismic inversion.

impedance $\mathbf{I}_p$ through the function $\mathbf{g}$. In the paradigm of machine learning, Equation (3) can be modeled as a two-layer neural network, including an input layer (model variable including $\mathbf{I}_p$) and output layer (seismic response $\mathbf{d}$) without hidden layers (Fig. 4). Different from typical neural networks, in this model, the input variables ($\mathbf{I}_p$) are unknow variables to be estimated rather than the weights (wavelet $\mathbf{w}$).

More specifically, the geostatistical seismic inversion is a mathematical approach consisting of three steps: simulation, inversion, and uncertainty evaluation. In geostatistical simulation, an ensemble of initial reservoir models is built from the prior geological knowledge and spatial correlation parameters through a chosen geostatistics simulation algorithm, i.e. Fast Fourier transform with moving average (FFT-MA; Le Ravalec et al., 2000) probability field simulation (PFS; Srivastava, 1992; Froidevaux, 1993), sequential Gaussian simulation (SGS; Doyen, 2007) and multiple-point statistics (MPS; Mariethoz and Caers, 2014). These prior reservoir simulations are then updated to assimilate seismic data and other geophysical measurements to provide a better description of the true reservoir behavior. Finally, we collect all inversion results of the reservoir realizations to evaluate the associated model uncertainty, which could make the decision-making in exploration and production more robust.

3.1. FFT-moving average and probability field simulation

In this paper, we use probability field simulation (Srivastava, 1992; Froidevaux, 1993), which is a fast-conditional simulation method and computationally efficient for large grids with millions of nodes, to generate the prior reservoir models. Unlike constructing the local probability density distribution (PDF) at every grid cell to incorporate previously simulated samples in sequential simulation methods, the local PDF in probability field simulation are estimated through kriging analysis of well data. Once the local PDF is obtained, we use a correlated probability field to draw samples from the PDF to simulate reservoir models that account for spatial correlation (Doyen, 2007). In particular, a reservoir simulation is the sum of the local mean and a spatially correlated Gaussian noise field scaled by the local standard deviation

$$z(\mathbf{u}) = m(\mathbf{u}) + \sigma(\mathbf{u})\varepsilon(\mathbf{u}) \tag{4}$$

where $z(\mathbf{u})$ is the simulated value at location $\mathbf{u}$; $m(\mathbf{u})$ is a deterministic function that represents the local mean; $\sigma(\mathbf{u})$ is the local standard deviation; and $\varepsilon(\mathbf{u})$ is the spatially correlated Gaussian noise field.

The simulation of Gaussian noise field is obtained by the FFT-moving average (FFT-MA) method. A spatially correlated Gaussian noise field is simulated by convolving a Gaussian white noise with the spatial correlation structure of the random field, namely moving average (Equation (5)). To speed up this procedure, we apply the FFT to perform the convolution in the frequency domain.

$$\varepsilon(\mathbf{u}) = n(\mathbf{u}) * G(\mathbf{u}) \tag{5}$$

where $n(\mathbf{u})$ is the Gaussian white noise and $G(\mathbf{u})$ represents the spatial correlation structure of the random field.

We can apply the same simulation procedure many times using different noise fields as input to generate multiple plausible reservoir realizations honoring the well data and spatial correlation. In this geostatistical post-stack seismic inversion, we use probability field simulation in conjunction with FFT-MA to build a set of acoustic impedance models as prior for the following seismic inversion.

3.2. Seismic inversion

The prior AI models are only conditional to well log data, but not honor seismic data. To provide a more accurate description of the subsurface model, we should update the models according to the misfit between the synthetic and measured seismic data. To make the solution more stable, we adopt the Tikhonov regularization method to control the complexity of the models. Therefore, the objective function is

$$J(\mathbf{m}) = ||\mathbf{d}_{\text{syn}} - \mathbf{d}^2_{\text{true}}||_2 + \lambda||\mathbf{m} - \mathbf{m}^2_{\text{true}}||_2 \tag{6}$$

where $\mathbf{d}_{\text{syn}}$ and $\mathbf{d}_{\text{true}}$ represent the synthetic seismic and true seismic respectively; $\mathrm{m}_{\text{prior}}$ represents the prior model. The first term in the objective function represents the data misfit, and the second term represents the model misfit, namely regularization term. The parameter λ is a hyper-parameter that controls the balance between model misfit and data misfit. With the advantage of auto-differentiation underlying TensorFlow, it is simple to add any regularization term, such as L1 regularization and constraint of space smoothing, to the objective function.

The objective function is also called loss function and it measures the quality of model parameters based on how well the synthetic seismic consist with the real seismic data. The best estimate of model parameters $\mathbf{m}$ is defined to be the model that minimizes the loss function

$$\mathbf{m} = \arg\min_{\mathbf{m}} J(\mathbf{m}) \tag{7}$$

Many optimization algorithms have been developed to find the optimal model: gradient descent (steepest descent) is the most common one and it extensively applied in the field of inverse problems. It minimizes the loss function by updating the model parameters in the direction opposite to the gradient, as described in Equation (8). By iteratively performing the gradient descent until the desired misfit is reached, we can obtain the optimal solution.

$$\mathbf{m}^{l+1} = \mathbf{m}^l - \eta\mathbf{g}^l \tag{8}$$

where l is the iteration index, $\mathbf{m}^l$ is the model parameters at the l th iterate, $\mathbf{g}^l = \nabla_{\mathbf{m}} J(\mathbf{m}^l)$ denotes the gradient of loss function with regard to model parameters evaluated at the l th iterate, and η is the learning rate.

In practice, the learning rate is the most important hyper-parameter that controls the step size to guide us how to optimize the model parameters according to the loss function. If the learning rate is too small, gradient descent can be very slow and take more time to converge; however, gradient descent with a large learning rate might

overshoot the minimum and fail to converge or even diverge. To deal with the aforementioned challenges, we apply a more efficient algorithm, Adaptive Moment Estimation (Adam - Kingma and Ba, 2014), to perform the optimization. Adam combines the advantages of two recently popular optimizers proposed in machine learning, namely AdaGrad (Duchi et al., 2011), RMSProp (Tieleman and Hinton, 2012) and AdaDelta (Zeiler, 2012), and the numerical experiments show that Adam is computationally efficient for optimization problems with high-dimensional parameter space, because it only uses the first and second moments of the gradients without Hessian matrix to compute adaptive learning rates for each model parameter (Kingma and Ba, 2014). Like gradient descent, the model parameters are updated iteratively by the following equation

$$\mathbf{m}^{l+1} = \mathbf{m}^{l} - \frac{\eta}{\sqrt{\hat{\mathbf{v}}^{l}} + \varepsilon}\hat{\mathbf{u}}^{l} \tag{9}$$

with

$$\hat{\mathbf{u}}^{l} = \frac{\mathbf{u}^{l}}{1-\beta_1} \tag{10}$$

$$\hat{\mathbf{v}}^{l} = \frac{\mathbf{v}^{l}}{1-\beta_2} \tag{11}$$

$$\mathbf{u}^{l} = \beta_1 \mathbf{u}^{l-1} + (1-\beta_1)\mathbf{g}^{l} \tag{12}$$

$$\mathbf{v}^{l} = \beta_2 \mathbf{v}^{l-1} + (1-\beta_2)(\mathbf{g}^{l} \circ \mathbf{g}^{l}) \tag{13}$$

where $\mathbf{u}^l$ and $\mathbf{v}^l$ are referred to as the first moment (the mean) and the second moment (the uncentered variance) of the gradient respectively; $\hat{\mathbf{u}}^l$ and $\hat{\mathbf{v}}^l$ are the corresponding biased-corrected first and second moment estimates to avoid the raw moments biased towards zero during the initial updating; the symbol $\circ$ denotes the Schur product; the hyper-parameters β_1 and β_2 control the exponential decay rates for the moment estimates, defaulting to 0.9 and 0.999 respectively; ε is equal to 10^{-8}.

Overall, the workflow of geostatistical seismic inversion in this paper is as illustrated in Fig. 5. We first generate a set of prior acoustic impedance models through FFT-MA and probability field simulation algorithms, and then convolve the wavelet estimated from seismic well tie with the reflectivity coefficients computed from these acoustic impedance models to generate the synthetic seismic responses. According to the misfit between the synthetic and the observed seismic data, we update the initial ensemble of reservoir models iteratively until convergence through the backpropagation algorithm.

4. Implementation with TensorFlow

In this section, we focus on the implementation of the geostatistical seismic inversion introduced in Section 3 in a parallel manner under the framework of TensorFlow.

4.1. Implementation with a single GPU

The simplest parallel scheme is to perform the inversion using a single GPU. In this scheme, reservoir realizations are updated sequentially: the inversion of one realization starts running until the previous one is totally completed. Fig. 6 shows the corresponding Data Flow Graph of one specific realization exported from TensorBoard. In the graph, each node, i.e. Simulation, Reflectivities, Synthetic, Misfit and Optimization, includes a set of operations, but for the sake of conciseness we use the technique of name scope to hide the unnecessary details. We can dive into those nodes of interest in by clicking to expand them.

Once the Data Flow Graph is built, we can start the program by feeding the seismic data, wavelet and geostatistical parameters into the graph. Then, the necessary operation nodes to compute the error

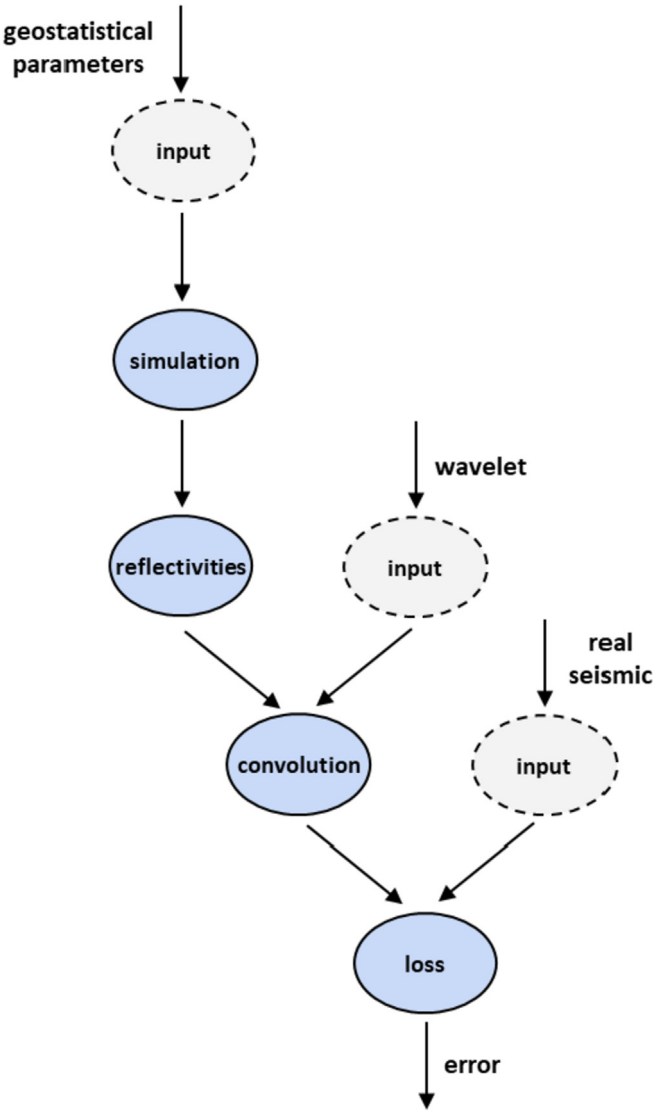

Fig. 5. Workflow of the geostatistical seismic inversion.

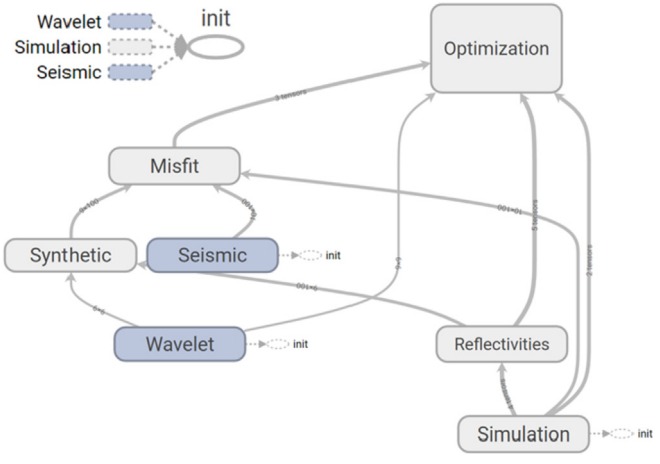

Fig. 6. Data Flow Graph of the geostatistical seismic inversion with single GPU.

defined in Equation (6) would be evaluated automatically. In this forward stage, the partial derivatives of any operations would also be automatically computed using the chain rule and implicitly added to the graph as new operation nodes. In the backward stage, the

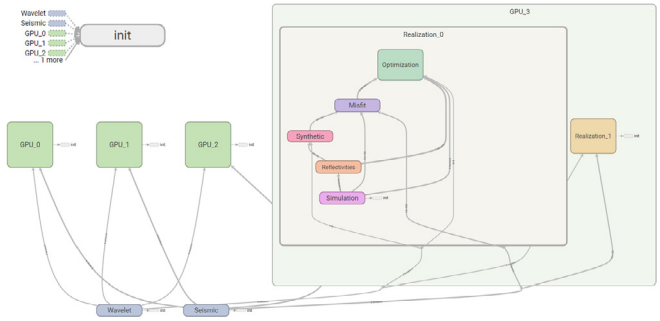

Fig. 7. Data Flow Graph of the geostatistical seismic inversion with multiple GPUs in model parallelism.

predefined optimizer would update the acoustic impedance to minimize the loss function using the back-propagation algorithm. By running the forward and backward step in an alternation manner, we can update the reservoir models iteratively until their seismic response convergence to the measured seismic data.

4.2. Implementation with multiple GPUs

To further reduce the inversion time, we can deploy the TensorFlow program on a GPU cluster in which each node is equipped with one or more GPUs. In this scheme, there are two parallel strategies, namely model and data parallelism, to perform the inversion of multiple reservoir realizations concurrently using multiple GPUs.

In the sense of model parallelism, each reservoir realization would be divided into a sub-graph and mapped to a specific GPU for inversion. For example, as illustrated in Fig. 7, eight reservoir realizations are evenly allocated to 4 GPUs. In this case, 4 reservoir realizations would be updated simultaneously at a time, and it would take two-round to accomplish the inversion of all reservoir realizations. In fact, operations in each GPU is identical, and we thus can scale up the inversion to tens or even hundreds of reservoir realizations across more machines.

When the size of seismic data is beyond the capacity of a single machine, we can also adopt the scheme of data parallelism. In this scheme, we split the seismic data into several small chunks that are then concurrently inversed on multiple GPUs. As illustrated in Fig. 8, a seismic data is divided into 4 sub-datasets and processed in 4 different GPUs concurrently. It is easily extensible and scalable to more computing devices and larger seismic data size. However, one limitation of data parallelism is that transition artefacts might exist between the sub-datasets, such as discontinuity at the boundaries of the sub-volumes.

5. Results

In this study, we use a 3D seismic dataset consisting of 600 × 600 × 200 grid nodes as the benchmark to evaluate the feasibility and the computational performance of the proposed schemes. The trace spacing of the seismic survey is 25 m in both inline and crossline directions, and the time sample interval is 1 ms. For the geostatistical inversion, we generate 1000 realizations of acoustic impedance models using the probability field simulation method. We assume that the spatial correlation is stationary and characterized by an exponential autocorrelation function (Equation (14)) with correlation range of 1250 m in both inline and crossline directions (a = 1250m, b = 1250m), and 5 ms in the vertical direction (c = 5ms).

$$r(x, y, z) = \exp[-(\frac{x^2}{a^2} + \frac{y^2}{b^2} + \frac{z^2}{c^2})^{1/2}] \tag{14}$$

To analyze the computational performance of the proposed parallel schemes in Section 4, we adopt Google Cloud Platform (GCP) to build the computing environments. The CPU used in the following experiments is Intel's Xeon Scalable Processor Skylake and the GPU is NVIDIA Tesla K80. The detailed specifications can be queried from the official websites of Intel (https://ark.intel.com/products/codename/37572/Skylake) and Nvidia (https://www.nvidia.com/en-us/data-center/tesla-k80). For optimal CPU performance, we compiled TensorFlow with the Intel-optimized Math Kernel Library (MKL) and Math Kernel Library for Deep Neural Networks (MKL-DNN). However, when building a cluster on GCP, it is generally not very clear about the specific hardware configuration, for example, whether the nodes are on the same rack. More importantly, we cannot configure the network bandwidth, which is the typical bottleneck in the distributed computing. For these reasons, we only evaluate the performance of multi-core CPU and multi- GPU on a single machine in this paper.

5.1. Quality of the inversion results

Fig. 9 compares the performance of different optimizers, including Adam, AdaDelta, AdaGrad, RMSProp and stochastic gradient descent (SGD). We can figure out that the Adam Optimizer has a great advantage in the convergence rate and accuracy over other optimizers: the Adam Optimizer takes much less iterations to converge asymptotically toward a smaller misfit between the synthetic and real seismic data, which means that the proposed optimization method can obtain more accurate reservoir models that are consistent with the seismic data in less computational time.

To evaluate the feasibility of the proposed workflow, we first performed the inversion at Well A and B. Fig. 10 shows 1000 inverted acoustic impedance realizations as well as their mean and synthetic seismic response. As we can see, the mean inverted results and the synthetic seismic data have a good agreement with the actual well logs and the true seismic traces respectively. Moreover, the model uncertainty can be evaluated by the 1000 posterior realizations. It indicates that the proposed workflow is feasible, and therefore, we then applied it to the entire seismic volume.

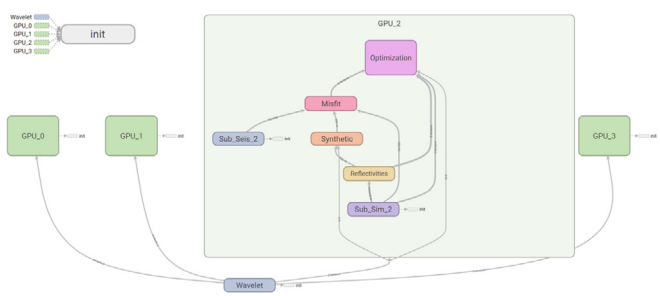

Fig. 8. Data Flow Graph of the geostatistical seismic inversion with multiple GPUs in data parallelism.

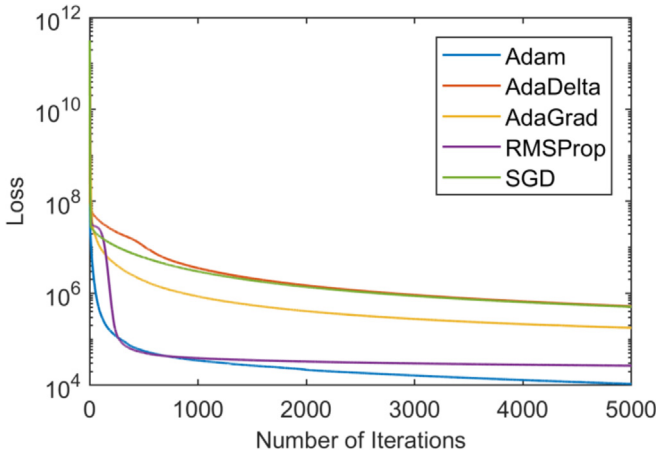

Fig. 9. Performance of different optimization methods.

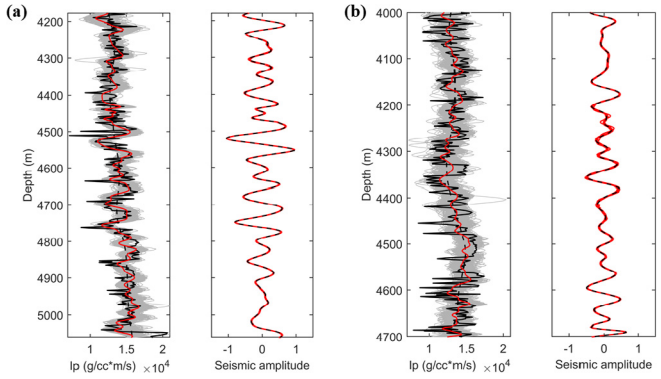

Fig. 10. Inversion results at Well A (a) and Well B (b). Left: inverted $\mathbf{I}_p$ models (the black curve represents the actual $\mathbf{I}_p$, the dash black cure represents the prior mean, the light gray curves represent 1000 posterior $\mathbf{I}_p$ realizations and the red curve represents the posterior mean); Right: synthetic seismic response (the dash black curves represent the real seismic trace and the red curves represent the synthetic seismic response of the posterior realizations). (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

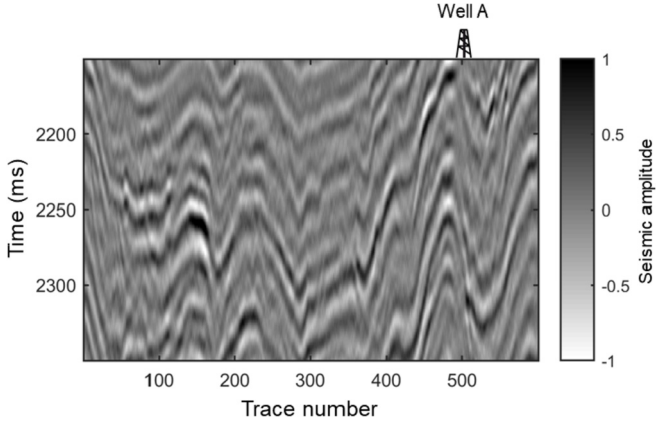

Fig. 11. Inline section of seismic data cross Well A.

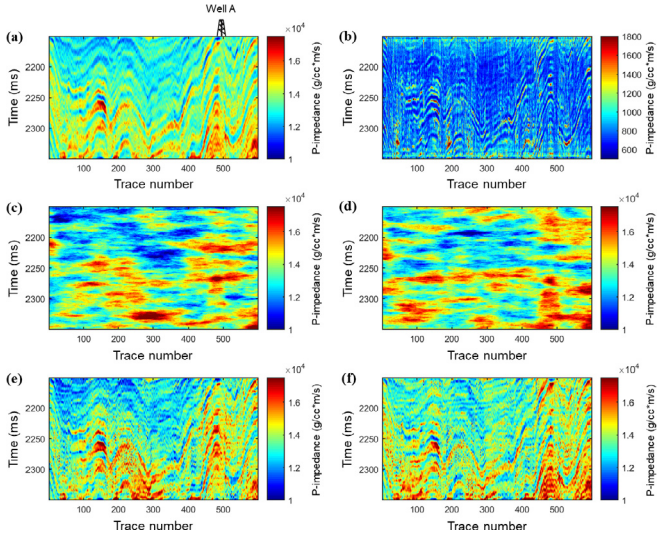

Fig. 12. Inline section of the inverted $\mathbf{I}_p$ models cross Well A in the scheme of model parallelism. (a) the mean of 1000 inverted realizations; (b) the standard deviation of 1000 inverted realizations; (c) and (d) two randomly selected prior realizations; (e) and (f) the updated realizations corresponding to the two prior realizations above.

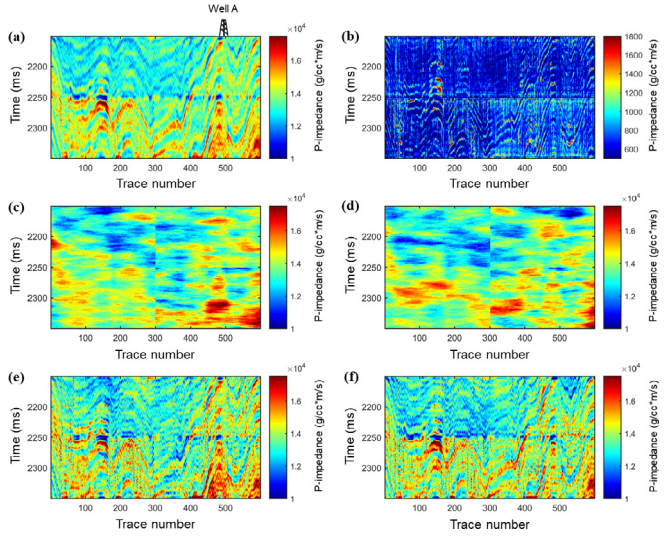

Fig. 13. Inline section of the inverted $\mathbf{I}_p$ models cross Well A in the scheme of data parallelism. (a) the mean of 1000 inverted realizations; (b) the standard deviation of 1000 inverted realizations; (c) and (d) two randomly selected prior realizations; (e) and (f) the updated realizations corresponding to the two prior realizations above.

An inline section cross Well A of the seismic dataset is shown in Fig. 11. The inversion results in the scheme of model parallelism and data parallelism are shown in Fig. 12 and Fig. 13, respectively, including the mean and standard deviation of the inverted 1000 realizations, and 2 specific realizations randomly selected from the ensemble. In the scheme of data parallelism, the $600 \times 600 \times 200$ reservoir model is evenly divided into 8 sub-volumes with size of $300 \times 300 \times 100$. From the results we can observe that all of the set of posterior realizations are conditioning on the seismic data but with small-scale heterogeneities, while the average model is relatively smooth and theoretically corresponding to the deterministic inversion result. It is also worth noting that both the inverted models from model parallelism and data parallelism can recover the major characteristic of the subsurface, but as mentioned in the preceding section, there exists obvious transition artefacts at the boundaries of the sub-volumes.

5.2. Performance of multicore CPU

We first test the performance using different number of CPU cores in terms of speed-up ratios. In TensorFlow, a multicore CPU is regarded as a whole device, but multithreading will be turned on by default to make full use of the computing resource. In this experiment, the corresponding Data Flow Graph is as illustrated in Fig. 6, and the speed-up ratio is defined as the ratio of the execution time using multi-core CPU to the execution time taken by the single-core CPU. Table 1 lists the absolute computational time with 1, 2, 4, 8 and 16 CPU cores, and their speed-up ratios are shown in Fig. 14. As we can see, the execution time is reduced with the increasing number of CPU cores. However, the improvement of computational efficiency is not linear to the number of CPU cores. By analyzing the execution time of each operation using Timeline, a tool built in TensorFlow, the major cause for this problem is that all the CPU cores share the same host memory with limited bandwidth, therefore, much time is spent on data reading and writing.

Table 1
The absolute computational time with different number of CPU cores.

Number of CPU cores	1	2	4	8	16
Computational time (hours)	647.78	483.33	263.33	144.31	100.90

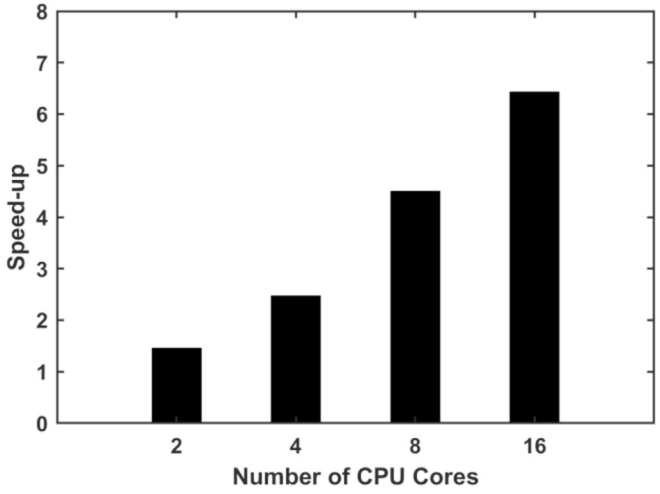

Fig. 14. Speed-ups with multicore CPU.

5.3. Performance of multiple GPUs

We then exploit multiple GPUs to further accelerate the inversion algorithm and analysis their performance. The video memory of NVIDIA Tesla K80 GPU used in this experiment is 12 GB, which is enough to handle the $600 \times 600 \times 200$ test model, so we are able to adopt the scheme of model parallelism to execute the same computational graph on different GPUs. The corresponding Data Flow Graph is as Fig. 7 illustrated. However, this is often not case in practical studies of reservoir characterization, and thus have to be split into several small chunks to be processed sequentially.

Fig. 15 shows the computational performance with different number of GPUs: a single GPU is over 20 times faster than a single-core CPU and the speed-up of 4 GPUs is about 70. It can conclude from the absolute computational time in Table 2 that the geostatistical inversion of 1000 realizations can be reduced from a couple of weeks to one day using a single GPU or even several hours using 4 GPUs, which would make the geostatistical seismic inversion method more available to practical applications. In addition, because each GPU card has its own device memory with a larger bandwidth than the host memory, the speed-up is approximately linear to the number of GPUs, which is not the case in the scheme of multiple-core CPU.

5.4. GPU performance with different grid sizes

In high performance computation, GPU is typically used to numerical problems involving large matrix operations. In theory, the algorithm based on GPU has an increasing speed-up compared to that based on CPU with the size of matrix increasing. Fig. 16 shows the speed-ups of a single GPU for different grid size compared to a single-core CPU. As expected, GPU typically has a better improvement on the larger size of seismic data.

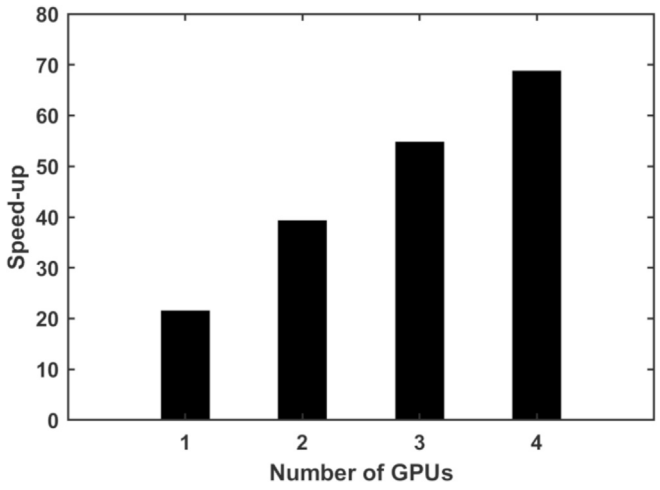

Fig. 15. Speed-ups with multiple GPUs.

Table 2
The absolute computational time with different number of GPUs.

Number of GPUs	1	2	3	4
Computational time (hours)	23.61	12.92	9.17	7.36

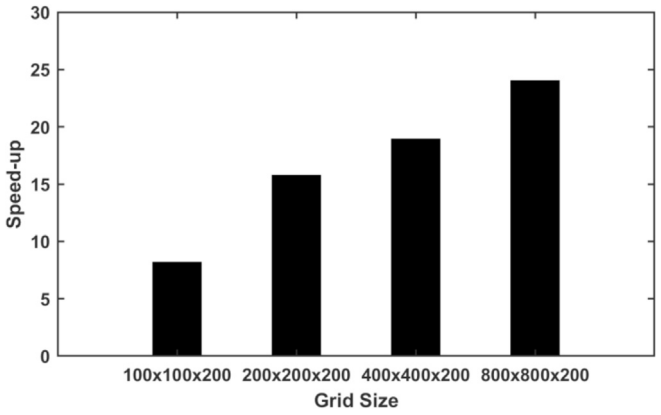

Fig. 16. Speed-ups with different grid size.

6. Summary and future work

This work investigates the acceleration of geostatistical seismic inversion using TensorFlow in a heterogeneous distributed manner, which allows taking advantage of GPUs and the cloud computing platform. To efficiently exploit multiple computing devices, i.e. CPUs and GPUs and process seismic data with large size, we proposed two multi-device parallelization schemes, namely model and data parallelism, to perform the inversion on multiple devices simultaneously. As shown in the numerical experiments, the GPU-based distributed system can offer a large computational speed-up with nearly two orders of magnitude over the CPU in the test case.

Although this study focuses on the geostatistical seismic inversion problem, the generic architecture and extensibility of TensorFlow make it applicable to many gradient-based numeric optimization problems in geophysics, such as full waveform inversion and seismic history matching. Our future work will focus on the development of a general parallel library based on TensorFlow for geophysical inverse problems.

Computer code availability

Source code of the proposed parallel geostatistical seismic inversion approach is available from the first author and can be downloaded from https://github.com/theanswer003/SeisFlow.

Name of Code: SeisFlow
Developer: Mingliang Liu
Contact address: 1000 E. University Ave. Laramie, WY 82071, USA
Telephone number: 307-343-6599
E-mail: mliu4@uwyo.edu
Year first available: 2018
Hardware: GPU recommended
Software: Python 3.6; TensorFlow, version r1.0 or later
Program language: Python
Program size: about 30 kb

Acknowledgements

Authors acknowledge the School of Energy Resources, and the Department of Geology and Geophysics of University of Wyoming for the support. We also thank the editor and anonymous reviewers for their critical review of the paper and constructive comments.

References

Abadi, M., Agarwal, A., Barham, P., et al., 2015. TensorFlow: large-scale machine learning on heterogeneous systems. arXiv preprint arXiv. 1605.08695.
Aster, R.C., Borchers, B., Thurber, C.H., 2011. Parameter Estimation and Inverse Problems. Academic Press.
Azevedo, L., Nunes, R., Soares, A., Mundin, E.C., Neto, G.S., 2015. Integration of well data into geostatistical seismic amplitude variation with angle inversion for facies estimation. Geophysics 80 (6), M113–M128.
Azevedo, L., Soares, A., 2017. Geostatistical Methods for Reservoir Geophysics. Springer.
Bortoli, L.J., 1992. Constraining Reservoir Models with Seismic Information. Master's thesis. Stanford University.
Bosch, M., Mukerji, T., Gonzalez, E.F., 2010. Seismic inversion for reservoir properties combining statistical rock physics and geostatistics: a review. Geophysics 75 (5), 75A165–75A176.
Caers, J., Hoffman, T., 2006. The probability perturbation method: a new look at Bayesian inverse modeling. Math. Geol. 38 (1), 81–100.
Doyen, P.M., 2007. Seismic Reservoir Characterization: an Earth Modelling Perspective. EAGE.
Duchi, J., Hazan, E., Singer, Y., 2011. Adaptive subgradient methods for online learning and stochastic optimization. J. Mach. Learn. Res. 12 (Jul), 2121–2159.
Ferreirinha, T., Nunes, R., Azevedo, L., Soares, A., Pratas, F., Tomás, P., Roma, N., 2015. Acceleration of stochastic seismic inversion in OpenCL-based heterogeneous platforms. Comput. Geosci. 78, 26–36.
Froidevaux, R., 1993. Probability field simulation. Geostat. Troia 92, 73–83.
Gineste, M., Eidsvik, J., 2017. June. Seismic waveform inversion using the ensemble Kalman smoother. In: 79th EAGE Conference and Exhibition 2017.
González, E.F., Mukerji, T., Mavko, G., 2007. Seismic inversion combining rock physics and multiple-point geostatistics. Geophysics 73 (1), R11–R21.
Grana, D., Mukerji, T., Dvorkin, J., Mavko, G., 2012. Stochastic inversion of facies from seismic data based on sequential simulations and probability perturbation method. Geophysics 77 (4), M53–M72.
Haas, A., Dubrule, O., 1994. Geostatistical inversion-a sequential method of stochastic reservoir modelling constrained by seismic data. First Break 12 (11), 561–569.
Hu, L.Y., 2000. Gradual deformation and iterative calibration of Gaussian-related stochastic models. Math. Geol. 32 (1), 87–108.
Huang, T., Li, X., Zhang, T., Lu, D.T., 2013. GPU-accelerated Direct Sampling method for multiple-point statistical simulation. Comput. Geosci. 57, 13–23.
Kappler, K., Kuzma, H.A., Rector, J.W., 2005. A comparison of standard inversion, neural networks and support vector machines. In: 2005 SEG Annual Meeting. Society of Exploration Geophysicists.
Kingma, D.P., Ba, J., 2014. Adam: a method for stochastic optimization. arXiv preprint arXiv. 1412.6980.
Le Ravalec, M., 2005. Inverse Stochastic Modelling of Flow in Porous Media, Application to Reservoir Characterization. Editions Technip.
Liu, M.L., Grana, D., 2018. Stochastic nonlinear inversion of seismic data for the estimation of petroelastic properties using the Ensemble Smoother and data re-parameterization. Geophysics 83 (3), 1–60.
Mallick, S., 1995. Model-based inversion of amplitude-variations-with-offset data using a genetic algorithm. Geophysics 60 (4), 939–954.
Mariethoz, G., Caers, J., 2014. Multiple-point Geostatistics: Stochastic Modeling with Training Images. John Wiley & Sons.
Mariethoz, G., 2010. A general parallelization strategy for random path based geostatistical simulation methods. Comput. Geosci. 36 (7), 953–958.
Martin, D., Estrin, G., 1967. Models of computations and systems—evaluation of vertex probabilities in graph models of computations. J. ACM 14 (2), 281–299.
Nunes, R., Almeida, J., 2010. Parallelization of sequential Gaussian, indicator and direct simulation algorithms. Comput. Geosci. 36, 1042–1052.
Peredo, O., Ortiz, J.M., Herrero, J.R., Samaniego, C., 2014. Tuning and hybrid parallelization of a genetic-based multi-point statistics simulation code. Parallel Comput. 40 (5–6), 144–158.
Rumelhart, D.E., Hinton, G.E., Williams, R.J., 1986. Learning representations by back-propagating errors. Nature 323 (6088), 533.
Sen, M.K., Stoffa, P.L., 1991. Nonlinear one-dimensional seismic waveform inversion using simulated annealing. Geophysics 56 (10), 1624–1638.
Srivastava, R.M., 1992. Reservoir characterization with probability field simulation. In: SPE Annual Technical Conference and Exhibition, pp. 4–7 (October, Washington, D.C).
Tahmasebi, P., Sahimi, M., Mariethoz, G., Hezarkhani, A., 2012. Accelerating geostatistical simulations using graphics processing units (GPU). Comput. Geosci. 46, 51–59.
Tarantola, A., 2005. Inverse Problem Theory. SIAM.
Thurin, J., Brossier, R., Métivier, L., 2017. June. Ensemble-based uncertainty estimation in full waveform inversion. In: 79th EAGE Conference and Exhibition 2017.
Tieleman, T., Hinton, G., 2012. Lecture 6.5-rmsprop: divide the gradient by a running average of its recent magnitude. COURSERA: Neural networks for machine learning 4 (2), 26–31.
Vargas, H., Caetano, H., Filipe, M., 2007. Parallelization of sequential simulation procedures. In: Proceedings of Petroleum Geostatistics. European Association of Geoscientists & Engineers, Cascais, Portugal.
Yourdon, E., Constantine, L.L., 1979. Structured Design: Fundamentals of a Discipline of Computer Program and Systems Design. Prentice-Hall, Inc.
Zeiler, M.D., 2012. ADADELTA: an adaptive learning rate method. arXiv preprint arXiv. 1212.5701.