Computers & Geosciences 93 (2016) 88–95

Contents lists available at ScienceDirect

Computers & Geosciences

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

Case study

# *pSIN*: A scalable, *P*arallel algorithm for *S*eismic *IN*terferometry of large-N ambient-noise data

Po Chen[*], Nicholas J. Taylor, Ken G. Dueker, Ian S. Keifer, Andra K. Wilson, Casey L. McGuffy, Christopher G. Novitsky, Alec J. Spears, W. Steven Holbrook

*Department of Geology and Geophysics, University of Wyoming, United States*

A R T I C L E I N F O

*Article history:*
Received 17 December 2015
Received in revised form
9 May 2016
Accepted 10 May 2016
Available online 11 May 2016

*Keywords:*
Seismic interferometry
Ambient-noise
Parallel algorithm
Message-passing interface

A B S T R A C T

Seismic interferometry is a technique for extracting deterministic signals (i.e., ambient-noise Green's functions) from recordings of ambient-noise wavefields through cross-correlation and other related signal processing techniques. The extracted ambient-noise Green's functions can be used in ambient-noise tomography for constructing seismic structure models of the Earth's interior. The amount of calculations involved in the seismic interferometry procedure can be significant, especially for ambient-noise datasets collected by large seismic sensor arrays (i.e., "large-N" data). We present an efficient parallel algorithm, named *pSIN* (*P*arallel *S*eismic *IN*terferometry), for solving seismic interferometry problems on conventional distributed-memory computer clusters. The design of the algorithm is based on a two-dimensional partition of the ambient-noise data recorded by a seismic sensor array. We pay special attention to the balance of the computational load, inter-process communication overhead and memory usage across all MPI processes and we minimize the total number of I/O operations. We have tested the algorithm using a real ambient-noise dataset and obtained a significant amount of savings in processing time. Scaling tests have shown excellent strong scalability from 80 cores to over 2000 cores.

## 1. Introduction

It has been demonstrated, both experimentally and theoretically, that "by cross correlating and stacking the ambient noise recorded at two receivers, it is possible to recover the response of the material recorded at one receiver as if there were an impulse excitation at the other receiver" (i.e. the Green's function) (Rickett and Claerbout, 1999). Theoretical backgrounds of this principle have been developed using the normal-mode theory (e.g., Lobkis and Weaver, 2001), representation theorems (e.g., Weaver and Lobkis, 2004), time-reversal invariance (e.g., Derode et al., 2003), the principle of stationary phase (e.g., Snieder, 2004) and the reciprocity theorem (e.g., Wapenaar, 2004). This capability to extract deterministic response of the Earth from random noise is playing an increasingly important role in passive-source seismic tomography, since it allows us to exploit the density of the seismic network without waiting for natural earthquakes to occur.

Rapid advances in seismic data acquisition technology, in particular the availability of cable-free, autonomous geophones (e.g., Freed, 2008), have now opened up the possibility of recording the full ambient-noise wavefields and conducting ambient-noise tomography using large, dense 2D seismic arrays (e.g., Ritzwoller et al., 2011; Lin et al., 2013). Such full-wavefield analysis using dense seismic array data is sometimes called "large-N" seismic analysis. The number of autonomous receivers used in such large-N studies can be much larger than those used in conventional passive-source seismic experiments. For the ambient-noise tomography study in Long Beach, California documented in Lin et al. (2013), more than 5200 autonomous receivers recorded the ambient-noise wavefield for three weeks. For the Blair Wallis, Wyoming and Sierra Nevada, California critical-zone ambient-noise tomography study, we deployed 6 square arrays with about 400 autonomous receivers per array and each receiver recorded the ambient-noise wavefield for 3–4 days at a sampling rate of 500 samples per second, producing a dataset of about 1.5 TB.

To estimate ambient-noise Green's functions that can be used in ambient-noise tomography, the noise data recorded by a seismic array need to be processed following a sequence of operations that are often called "seismic interferometry". The computational cost of the entire procedure depends upon the total number of receivers $N_r$ (Table 1) and the duration of the recording and can become a lengthy compute for large-N data. For the Blair-Wallis and Sierra-Nevada datasets used in this study, it took about 13 days of uninterrupted computing time on a single state-of-the-art four-core desktop computer to process the noise data recorded by one array of about 400 receivers. A natural choice for speeding up

[*] Corresponding author.
*E-mail address:* pchen@uwyo.edu (P. Chen).

http://dx.doi.org/10.1016/j.cageo.2016.05.003
0098-3004/© 2016 Elsevier Ltd. All rights reserved.

**Table 1**
List of symbols.

| Symbol | Meaning |
|---|---|
| $M$ | MPI process row number (Fig. 1) |
| $N$ | MPI process column number (Fig. 1) |
| $N_p$ | Total number of MPI processes, i.e., $N_p = M \times N$ |
| $P_{i,j}$ | The MPI process on the $i$-th row and $j$-th column (Fig. 1) |
| $N_r$ | Total number of receivers of the seismic receiver array |
| $N_r^p$ | Number of receivers per process row, i.e., $N_r^p = N_r / M$ |
| $N_{seg}$ | Total number of time segments (e.g., if the entire noise recording is 1-h long and each time segment is 2-min long, then $N_{seg} = 1$ h/2 min$=30$) |
| $N_{seg}^p$ | Number of time segments per process column, i.e., $N_{seg}^p = N_{seg}/N$ |
| $N_c$ | Total number of stacked cross-correlations, i.e., $N_c = (N_r - 1) \times N_r / 2$ |
| $N_s$ | Number of data samples per time segment |
| $N_s^p$ | Number of data samples per MPI process, i.e., $N_s^p = N_r^p \times N_{seg}^p \times N_s$ |

the entire process is through parallelization.

The parallelization of the seismic interferometry algorithm appears to be trivial, as there is no apparent interdependence among the calculations of the stacked cross-correlations of different receiver pairs. A naive implementation is to execute the sequential seismic interferometry code on multiple CPU cores with each core calculating the stacked cross-correlations for a subset of all receiver pairs and no inter-process message passing is required. This problem seems to be “embarrassingly parallel”. However, this naive implementation is heavily I/O-bound, especially on small to medium-sized computer clusters not equipped with powerful I/O subsystems. On most current-generation computer clusters, the memory size per core is limited. For our Blair-Wallis and Sierra-Nevada datasets, the average size of the binary file for one receiver is about 0.62 GB (each time sample is stored as a 4-byte single-precision float). On the Mount Moran cluster (a 284-node IBM System X cluster with each node having two 8-core Intel Xeon E5-2670 2.6 GHz processors) at the Advanced Research Computer Center (ARCC), University of Wyoming, the usable memory size per core is about 1.8 GB, which allows holding the recordings of a maximum of 3 receivers at one time, not counting the memory needed for storing other data during the calculation. To obtain all the stacked cross-correlations (approximately 80,000), each process needs to access the disk repeatedly during the entire calculation, which incurs heavy I/O overhead, especially at large core count.

One possibility for reducing the I/O overhead in the naive implementation is to adopt the Hadoop framework and the associated MapReduce computational paradigm and the Hadoop Distributed File System (HDFS). In fact, the ambient-noise seismic interferometry problem is an ideal candidate for the Hadoop implementation (Addair et al., 2014). When combined with the HDFS, the Hadoop framework leverages data locality by moving computation to the data, thereby providing extremely high I/O speed. However, on small to medium-sized computer clusters that are shared by many different types of applications, such as Mount Moran, the benefit for the ambient-noise seismic interferometry application may not justify the effort involved in setting up the Hadoop framework on the entire cluster.

In this study, we explore the possibility of reducing the I/O burden of ambient-noise seismic interferometry calculations on conventional distributed-memory computer clusters through algorithmic redesign. The *pSIN* code provided with this paper reads the ambient-noise data of the entire array from disk only once at the beginning of the execution and writes out all stacked cross-correlations at the end of the execution. Parallelization is implemented using the Message-Passing Interface (MPI). Computation, inter-process communication, and memory usage are well balanced across all CPU cores. Scaling tests using one of our Blair-Wallis and Sierra-Nevada datasets show excellent strong scalability from 80 cores to more than 2000 cores.

## 2. Algorithm

A widely adopted seismic interferometry technique (e.g., Bensen et al., 2007) involves three steps: single-receiver processing, inter-receiver cross-correlation and temporal stacking. At the single-receiver processing step, the entire time series recorded by each receiver is cut into $N_{seg}$ equal-length time segments with $N_s$ samples per segment (Table 1) and each time segment is normalized both in the time domain and in the frequency domain. At the inter-receiver cross-correlation step, each time segment at one receiver is cross-correlated with the corresponding time segment of every other receiver in the same array. For an array composed of $N_r$ receivers, the total number of unique cross-correlation calculations is therefore $(N_r - 1) \times N_r \times N_{seg}/2$. At the temporal stacking step, all the $N_{seg}$ cross-correlations for the same receiver pair are summed, producing $N_c = (N_r - 1) \times N_r/2$ stacked cross-correlations (Table 1), which are often called “ambient-noise Green's functions”.

### 2.1. Single-receiver processing

Single-receiver processing is the first step in the seismic interferometry workflow. In this step, we need to read the ambient-noise data of the entire array from disk and decide the layout of the massive amount of noise data across all MPI processes. The data layout will determine the amount of inter-process communication overhead during the inter-receiver cross-correlation and temporal stacking steps, as well as the overhead for writing the stacked cross-correlations back to disk. The smallest data unit in the entire seismic interferometry process is one time segment of the noise data recorded by one receiver. We therefore need to consider how to group these smallest data units together so that the inter-process communication overhead in later steps can be minimized.

The smallest data unit has two natural coordinates: the index of the receiver that recorded this segment of noise data (i.e., receiver number) and the time-segment index that determines the offset from the beginning of the entire time series recorded by that receiver. We therefore adopt a two-dimensional Cartesian virtual topology to organize the MPI processes. The horizontal dimension is the time dimension with $N$ MPI processes (Table 1) and the vertical dimension is the receiver dimension with $M$ MPI processes (Table 1, Fig. 1). The total number of processes (i.e., cores) used in the entire calculation is therefore $N_p = M \times N$ (Table 1). The total number of receivers of a seismic array $N_r$ is partitioned across the $M$ rows of the process array and each row stores the noise data of $N_r^p = N_r/M$ receivers (Table 1). The entire time series of each receiver is partitioned across the $N$ columns of the process array and each column stores $N_{seg}^p = N_{seg}/N$ time segments (Table 1). The total number of noise data samples per process is therefore $N_s^p = N_r^p \times N_{seg}^p \times N_s$ (Table 1), represented as black circles in Fig. 1.

On disk, the ambient-noise data are usually stored as one binary file per receiver, which contains the entire time series recorded by that receiver during the deployment, as well as some metadata (e.g., receiver ID and start time). The root rank of each process row, denoted as $P_{*,0}$ in Fig. 1, where “*” denotes any row rank, can read the binary files for different receivers simultaneously. After reading the binary file for one receiver, the root process of each row evenly distributes the data to the $N$ processes in the same row using the MPI scatter function with each process receiving $N_{seg}^p \times N_s$ data samples. This reading/scattering procedure

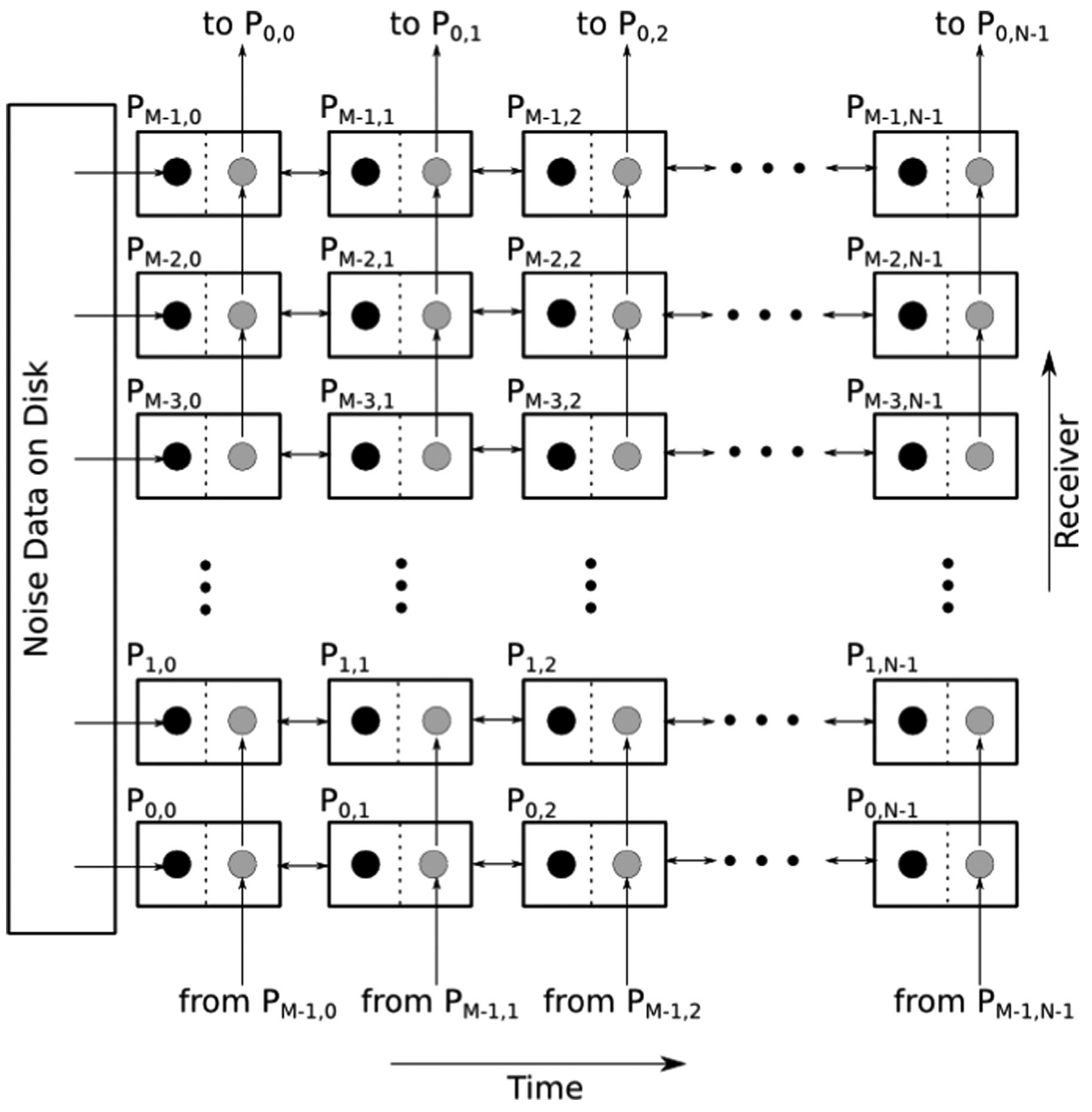

**Fig. 1.** A schematic diagram of the 2D process array. Each process $P_{i,j}$ ($i=0, 1, 2, \ldots M-1$; $j=0, 1, 2, \ldots, N-1$) is represented as a rectangle and black arrows indicate directions of data movement. Black circles: noise data belongs to the process; gray circles: data buffers for hosting noise data from neighboring processes during data shuffle.

is iterated $N_r^p$ times on each process row to move the noise data of the entire seismic array from disk to the memory of the 2D process array. The memory usage is balanced across the entire process array. The binary file of each receiver is read only once.

The calculation in the single-receiver processing step involves normalization in the time domain and in the frequency domain. Each data unit (i.e., one time segment of the noise data of one receiver) needs to be transformed to the frequency domain through FFT. To reduce memory usage, we only retain the positive-frequency half of the entire spectrum after FFT. The computational cost is proportional to the total number of data units on each process $N_r^p \times N_{seg}^p$ and is therefore balanced across the 2D process array. There is no inter-process communication involved in the normalization and FFT calculations and we obtain ideal scaling for these calculations.

## 2.2. Inter-receiver cross-correlation

The inter-receiver cross-correlation between two receivers is computed only between the same time segments. Hence, computation of the cross-correlation between different time segments at different receivers is not required. This observation means that no inter-column communication is needed in our 2D process array in the inter-receiver cross-correlation step. We therefore separate this step into two stages: the local stage, in which no inter-process communication is required, and the global stage, in which inter-row communication (i.e., along the vertical dimension across process rows) (Fig. 1) is needed. All cross-correlations are computed in the frequency-domain using the frequency-domain noise data obtained in the single-receiver processing step. The frequency-domain cross-correlation is transformed back to the time-domain through IFFT. We apply a time-domain normalization using the maximum amplitude of the cross-correlation before temporal stacking, as suggested in Bensen et al. (2007).

### 2.2.1. Local stage

In the local stage, each process calculates the time-domain normalized cross-correlation between the same time segments of different receivers stored in its own memory. The number of cross-correlation calculations per process, as well as the number of IFFT and time-domain normalization calculations per process, is therefore $(N_r^p-1)\times N_r^p\times N_{seg}^p/2$. The computational load is balanced.

### 2.2.2. Global stage

In the global stage, the frequency-domain noise data on process $P_{i,j}$ are duplicated on process $P_{i+1,j}$ so that the time-domain normalized cross-correlations between noise data on receivers stored on two neighboring process rows can be calculated. We call this procedure "data shuffle" (upward arrows in Fig. 1). To store the transferred noise data, a separate data buffer on each process is created (gray circles in Fig. 1). For each shuffle, we transfer the noise data of $N_r^q$ receivers and $N_{seg}^p$ time segments from processes $P_{i,j}$ to the data buffer on process $P_{i+1,j}$ and if the data buffer is sufficiently large, we can choose $N_r^q=N_r^p$ and the entire noise data is shuffled by one process row. After each shuffle, the number of time-domain normalized cross-correlations that can be computed on each process (i.e., cross-correlations between the data in the black circle and those in the gray circle in Fig. 1) is $N_r^p\times N_r^q\times N_{seg}^p$. The computational load and memory usage are balanced across the entire process array.

We impose the periodic boundary condition on the vertical dimension of the 2D process array (Fig. 1). The noise data stored in the gray circle on process $P_{M-1,*}$ are shuffled to process $P_{0,*}$ (Fig. 1), where "*" denotes any column rank in the 2D process array. The total number of data shuffle operations needed to cover all receiver pairs is $C_M^2/2=(M-1)/2$, where $C_M^2$ is the total number of combinations of $M$ process rows taken 2 at a time.

The data shuffle operation can be implemented conveniently and efficiently using the MPI send-receive routine, which combines the sending of a message to a destination with the receiving of another message from a source. On a Cartesian process virtual topology, the ranks of the source and destination processes needed in the send-receive call can be obtained by calling the MPI Cartesian coordinate shift routine. This technique for shifting data across a chain of processes in a certain direction is also widely used in parallelizing finite-difference solutions of partial differential equations using a domain decomposition approach, in which state variables on the ghost points of a sub-mesh need to be shifted in order to compute the derivatives on a neighboring sub-mesh.

### 2.2.3. An example

Consider the example shown in Fig. 2. We have a seismic array composed of $N_r=10$ receivers, indexed as 0, 1, 2, …, 9, and we distribute the noise data evenly on a process array with $M=5$ rows and $N=1$ column, therefore each process row holds the entire noise recordings of $N_r^p=2$ receivers. The total number of stacked cross-correlations for the entire array is $N_c=(N_r-1)\times N_r/2=45$. In the local stage, each process row calculates $(N_r^p-1)\times N_r^p/2=1$ cross-correlations, providing 5 cross-correlations in total (left column, Fig. 2). The 5 process rows have $C_5^2=10$ different combinations taken 2 at a time. After the first shuffle (center column, Fig. 2), we obtain $M=5$ combinations out of the 10 combinations. Each process row can now calculate $N_r^p\times N_r^p=4$ cross-correlations, providing 20 cross-correlations in total. After the second shuffle (right column, Fig. 2), we obtain 5 more combinations, covering all 10 combinations, and the entire process array can provide 20 more cross-correlations, covering all 45 cross-correlations. The total number of data shuffle operations is $C_5^2/2=10/5=2$ for this example. Each process row calculates 9 time-domain normalized cross-correlations in total and the computational load is balanced.

## 2.3. Temporal stacking

The stacking process involves summing the time-domain normalized cross-correlations for the same receiver pair over all the $N_{seg}$ time segments. Hence for the first time in our algorithm, we

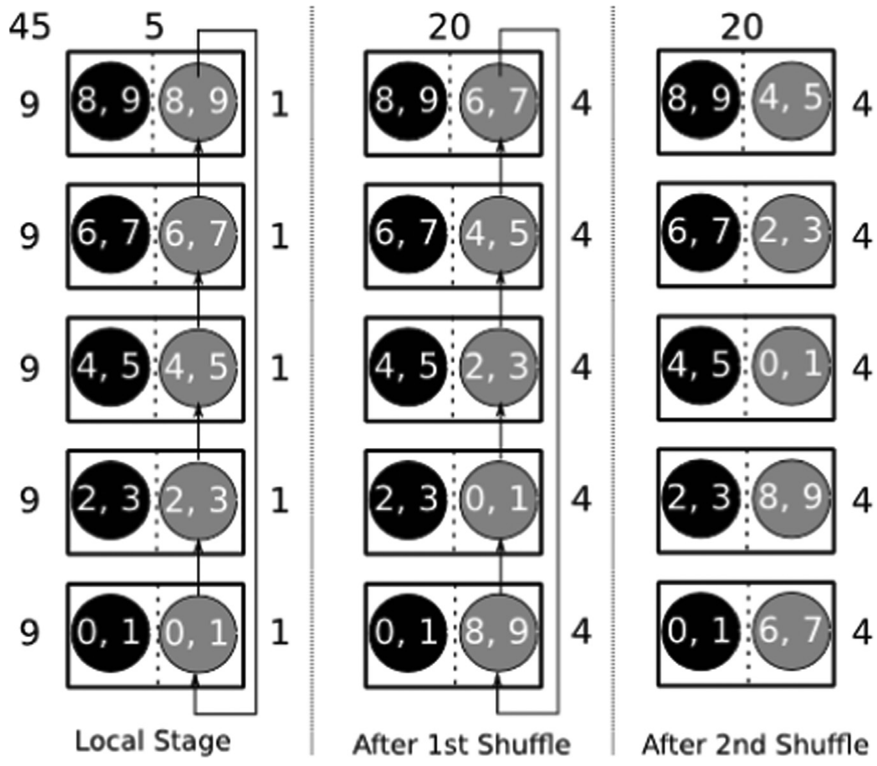

**Fig. 2.** An example of the data shuffle operation for a process array with 5 rows and 1 column. Rectangles, black and gray circles have the same meaning as in Fig. 1. The numbers on the left side of each column indicate the number of cross-correlations computed by each process during the current stage. The numbers listed on the right side of the first column show the total number of cross-correlation computed by each process in all stages. The numbers listed on top of the columns show the total number of cross-correlations computed by all processes during the current stage. The number "45" is the total number of cross-correlations computed by all processes in all stages.

need to perform inter-column, not inter-row, communication across the 2D process array. We also separate this step into the local stage, in which no inter-process communication is needed, and the global stage, in which communication along the horizontal dimension across the different columns of the same row is needed. We note that the stacking process does not need to wait until the cross-correlations for all receiver pairs and all time segments are completed. As soon as the cross-correlations for all time segments of a subset of receiver pairs are calculated, a stacking of that subset can be performed. Note that by stacking the $N_{seg}$ cross-correlations, large quantities of memory are freed. Hence, by weaving together the stacking and the cross-correlation operations, we can effectively reduce the total memory usage per process.

### 2.3.1. *Local stage*

During the local stage, we sum the time-domain normalized cross-correlations for the same receiver pair at different time segments stored locally on each process. As soon as the summation is completed for a subset of receiver pairs, the memory used for storing the time-segment cross-correlations of those receiver pairs can be reused for other receiver pairs. We use a separate data buffer to store the locally stacked cross-correlations.

### 2.3.2. *Global stage*

Because each process column stores only a subset of all time segments, the locally stacked cross-correlations need to be summed across process columns to generate the globally stacked cross-correlations. This global summation can be implemented conveniently using the MPI reduction routine. To ensure memory usage balance for the global summation, the destinations of different reduction operations are different. We modify the example shown in Fig. 2 by adding two more process columns. The entire noise recording of each receiver is evenly partitioned into 9 time segments and each process column stores 3 time segments. Fig. 3 shows the layout of the cross-correlations on the bottom row of the 2D process array. Fig. 2. shows that the bottom row calculates the cross-correlations for 9 receiver pairs: 0–1, 0–8, 0–9, 1–8, 1–9, 0–6, 0–7, 1–6, 1–7. After the local stage, each column of the bottom row stores the locally stacked cross-correlations (solid-line ellipses in left column, dash-line ellipses in center column, dash-dot-line ellipses in right column, Fig. 3) for these 9 receiver pairs. We group these 9 locally stacked cross-correlations into 3 different data buffers with each buffer containing the cross-correlations for 3 receiver pairs (ellipses in Fig. 3). The first reduction operation reduces the 3 cross-correlations in the first group on all process columns onto column 1 (top row, Fig. 3). The data buffers for storing the first group on columns 2 and 3 can be released. The second reduction operation reduces the 3 cross-correlations in the second group on all process columns to column 2 (second row, Fig. 3) and the data buffers for the second group on columns 1 and 3 can be released immediately. The last reduction operation reduces the 3 cross-correlations in the third group on all process columns onto column 3 (third row, Fig. 3) and the data buffers for the third group on columns 1 and 2 can be released.

In practice, the global reduction does not need to wait until locally stacked cross-correlations for all 9 receiver pairs are completed. For instance, the first reduction (top row, Fig. 3) can be carried out immediately after the locally stacked cross-correlations

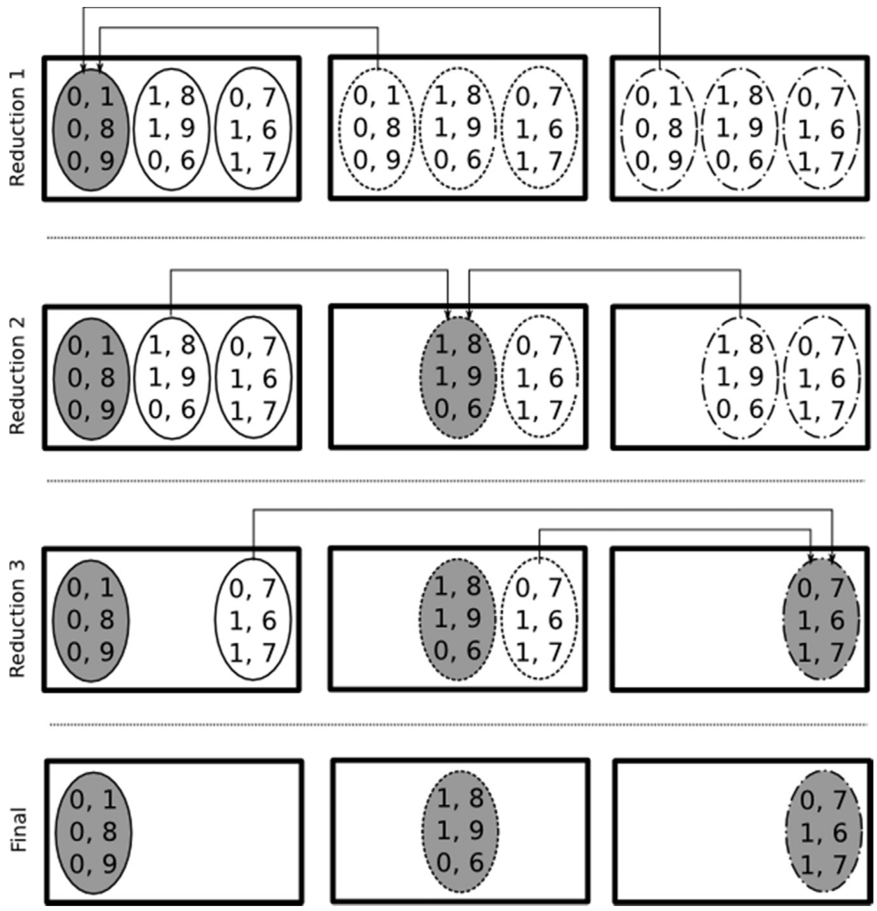

**Fig. 3.** An example of the global stage in the temporal stacking process on the bottom row of a 2D process array with 5 rows and 3 columns. Black arrows indicate data movement. Ellipses indicate data buffers for storing the locally stacked cross-correlations; different line styles of the ellipses indicate different time intervals of the noise data; gray ellipses indicate the destinations of the reduction operation. Numbers inside each ellipse show the receiver indices for the cross-correlations.

for the 3 receiver pairs in the first group (i.e., receiver pairs 0–1, 0–8, 0–9 in Fig. 3) are completed. The memory used for storing the globally stacked cross-correlations is balanced across the process array (bottom row, Fig. 3). Those globally stacked cross-correlations are the ambient-noise Green's functions and can be written to disk for ambient-noise tomography analysis.

## 3. Results

During the summer of 2015, seismologists of the Wyoming Center for Environmental Hydrology and Geophysics (WYCEHG) deployed six seismic arrays at the WYCEHG Blair-Wallis study area and the Southern Sierra-Nevada Critical Zone Observatory with the goal of generating three dimensional shear wave velocity models to characterize critical zone development in varying regions and provide constraints for hydrologic flow modeling. In this paper we use the ambient-noise data from one of the Sierra-Nevada critical zone seismic arrays for evaluating the performance of the *pSIN* code. This seismic array with 396 active autonomous receivers was deployed at the Bald Mountain outcrop (approximately N37.12923, W119.19139) in the Southern Sierra-Nevada, California from August 17th to 20th in 2015 (Fig. 4). The 396 receivers were distributed in a roughly 200-m by 200-m square area. Each autonomous receiver is equipped with a one-channel 24-bit analog data logger, a GPS, rechargeable lithium-ion batteries that can last 12 days and a 2GB flash memory for storing the data. Each receiver weights about 4.8 lb, is about 6-in. high with about 5-in. diameter and has a detachable 5-in. spike. For the example dataset used in this study, each receiver recorded the vertical-component ambient-noise wavefield continuously for about 4 days at a sampling rate of 500 samples per second. Examples of the stacked cross-correlations generated using *pSIN* are shown in Fig. 5.

The entire calculation from reading the raw noise data from disk to writing out the 78,210 stacked cross-correlations to disk took 52.45 min of wall-time on 2112 CPU cores (Intel Xeon E5-2670 2.6 GHz). We used a 2D process array with $M=66$ rows and $N=32$ columns with each process row storing the noise data of $N_r^p=6$ receivers. The entire time series recorded by each receiver was evenly partitioned into $N_{seg}=2592$ time segments with each process column storing $N_{seg}^p=81$ time segments. Each time segment had 65,536 data samples ($\sim$2.2 min of noise recording). The total number of time-segment cross-correlations involved in the entire calculation was about 202.72 million and each CPU core computed 95,985 time-segment cross-correlations.

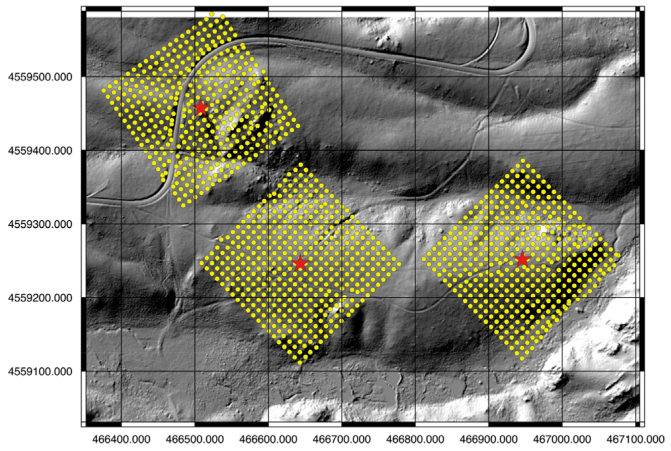

**Fig. 4.** Array geometries for the seismic deployments at the Bald Mountain outcrop in the Southern Sierra-Nevada Critical Zone Observatory (SSCZO) in California. Each node (yellow circle) recorded the local ambient field for roughly 4 continuous days. The array size is roughly 200 m by 200 m. The noise data from the lower-right array (BW3) are used as the example data in this study and examples of ambient noise Green's functions are shown in Fig. 5. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

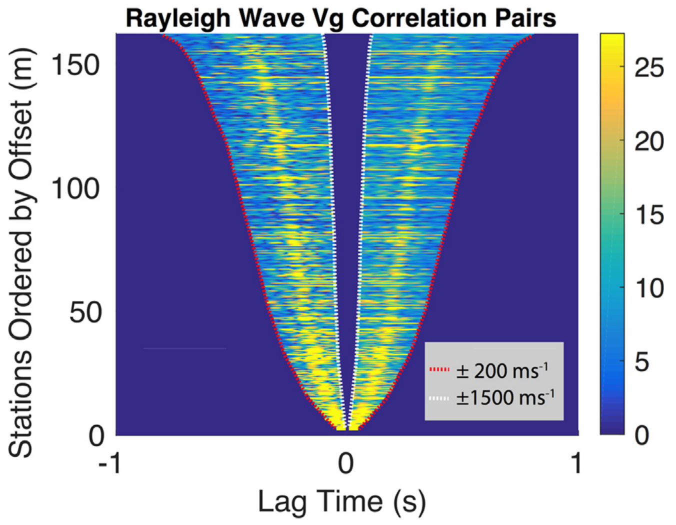

**Fig. 5.** Examples of stacked cross-correlations (i.e., ambient-noise Green's functions) between one receiver with the remaining receivers in the BW3 array (Fig. 4). All ambient-noise Green's functions have been bandpass filtered to between 65 Hz and 85 Hz. Color indicates amplitudes. Dash lines indicate velocities of 200 m/s (red) and 1500 m/s (white). (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

We have evaluated the scalability of *pSIN* by using smaller subsets of the entire Bald Mountain ambient-noise dataset. Since the process array is two dimensional (Fig. 1) and the overhead is different for inter-row communications and inter-column communications, we have carried out two strong scaling tests by: 1) increasing the number of process columns while fixing the number of process rows, and by 2) increasing the number of process rows while fixing the number of process columns. In our tests, we have grouped all operations in *pSIN* into 5 categories: (1) “Calc”, which includes all the local calculations that does not involve any inter-process communications (i.e., single-receiver processing, the local stages of the inter-receiver cross-correlation and temporal stacking steps); (2) “Sendrecv”, which includes inter-row communications during data shuffle in the global stage of the inter-receiver cross-correlation step; (3) “Reduce”, which includes inter-column communications during the global summation stage of the temporal stacking step; (4) “Scatter”, which includes inter-row communications when the root process of each row scatters the noise data of each receiver to all other processes in the same row; (5) “Read”, which includes the disk I/O operations.

### 3.1. Strong scaling with process column number N

Fig. 6 shows strong-scaling results obtained by doubling the number of process columns 5 times (i.e., $N=16$, 32, 64, 128, 256, 512) while holding the number of process rows constant at $M=5$. For strong-scaling tests, the size of the problem must be fixed and we used a subset of the Bald Mountain ambient-noise data that consists of the complete 4-day noise recordings of 30 receivers. Each process row stored the noise data of 6 receivers, while the number of time segments stored on each process column decreased from 160 to 5 as the number of process columns increased from 16 to 512.

The proportion of the amount of wall-time spent on “Calc” operations, which shows perfect scaling in Fig. 6, decreased from about 96.39% to about 59.6% of the total wall-time as the number of process columns increased 32-fold. The wall-time spent on inter-row communication in “Sendrecv” operations also shows good

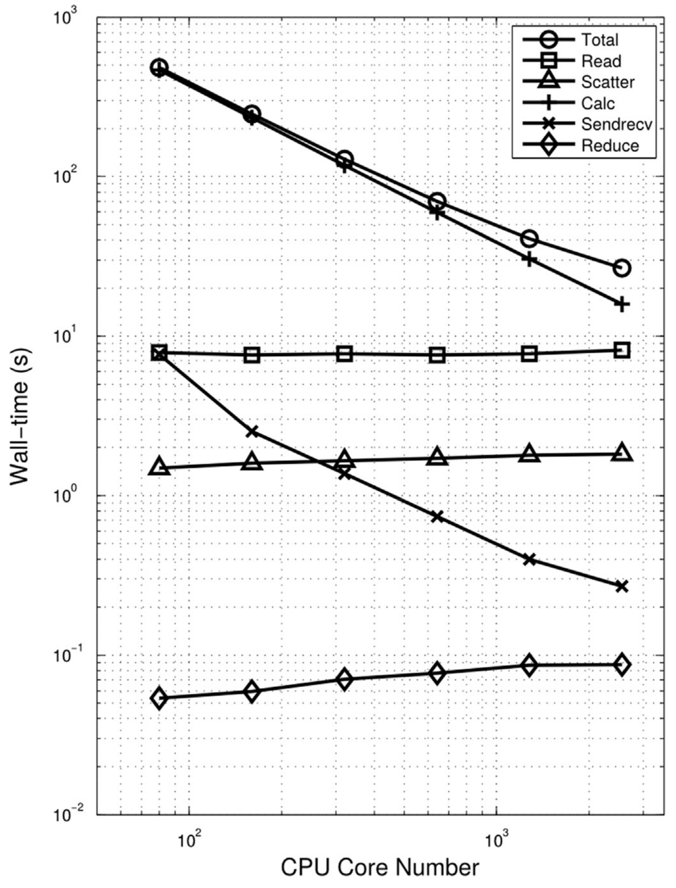

**Fig. 6.** Strong scalability with respect to the number of columns of the 2D process array. Vertical axis: wall-time in seconds; horizontal axis: number of CPU cores. Different symbols show the scaling of different types of operations as discussed in the text.

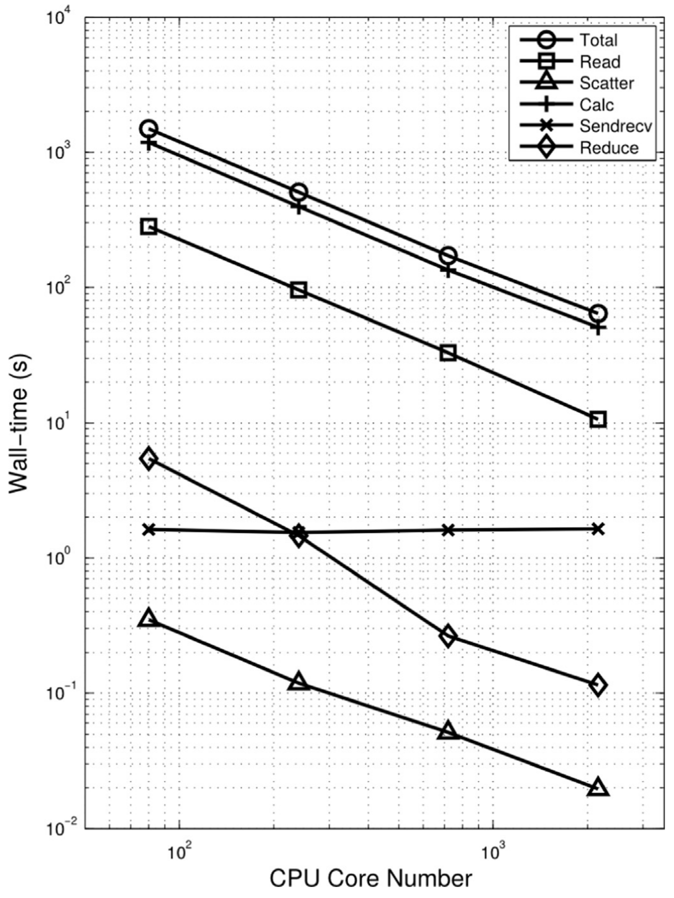

**Fig. 7.** Strong scalability with respect to the number of rows of the 2D process array. Vertical axis: wall-time in seconds; horizontal axis: number of CPU cores. Different symbols show the scaling of different types of operations as discussed in the text.

scaling. As the number of process column increases, the communication volume per process (i.e., the total amount of data sent from process $P_{i,j}$ to process $P_{i+1,j}$ during data shuffle) decreases proportionally and the total number of data shuffle operations is fixed at 2, because the number of process rows is fixed at $M=5$ for all tests. The amount of wall-time spent on “Reduce” and “Scatter” operations increased slightly with the number of process columns. The communication volume per process in “Reduce” and “Scatter” operations does decrease linearly with the number of columns. But the total number of processes involved in these two types of operations also increases linearly with the number of columns, thereby limiting the scalability of these two types of operations. However, the proportion of these two types of operations in the total amount of wall-time is relatively small (i.e., about 6.8% for “Scatter” and 0.3% for “Reduce” for the largest column number). The amount of wall-time spent on “Read” operation was about constant for all the column numbers, since the amount of ambient-noise data on disk and also the number of processes used for reading the noise data were constant.

### 3.2. Strong scaling with process row number M

Fig. 7 shows the wall-time spent on the different types of operations when increasing the number of process rows from 5 to 135 (i.e., $M=5$, 15, 45, 135) and holding the number of columns constant at $N=16$. The total number of receivers used in these tests was fixed at $N_r=2430$, which is much larger than the 400 receivers in our Bald Mountain dataset. We generated the “fake” dataset by duplicating the Bald Mountain dataset about 6 times. The number of receivers stored in each process row therefore decreased from 486 to 18 as the number of process rows increased 27-fold. The number of time segments stored on each process column was fixed at 80 and each time segment had 512 samples.

The proportion of the amount of wall-time spent on “Calc” operations was about 79% of the total amount of wall-time in all tests. The total amount of wall-time shows excellent scaling with a maximum deviation from a linear prediction of about 16.52% (i.e., 100*(measurement − prediction)/prediction). The amount of wall-time spent on “Sendrecv” operations was about constant. As the number of process rows increases, the communication volume per process in the data shuffle operation decreases proportionally, while the total number of data shuffle operations increases proportionally, which resulted in roughly constant wall-time for all “Sendrecv” operations. The proportion of the amount of wall-time spent on “Sendrecv” operations was less than 2.54% of the total amount of wall-time. The amount of wall-time spent on “Reduce” operations decreased faster than a linear prediction. As the number of process rows increased, the number of receivers per process row $N_r^p$ decreased proportionally, while the number of stacked cross-correlations per row decreased roughly as $\left(N_r^p\right)^2$, which resulted in a decrease not only in the communication volume per process but also in the number of global stacking operations per process row. The “Scatter” operations had the worst scaling with the maximum deviation from a linear prediction of about 50.77%. However the proportion of the wall-time spent on “Scatter” operations was less than 0.03% of the total wall-time.

## 4. Summary and discussion

We have presented a complete distributed-memory parallel algorithm for performing seismic interferometry of large-N ambient-noise data and tested the implementation *pSIN* using a real large-N dataset collected by WYCEHG seismologists. The parallel *pSIN* code has allowed us to reduce the total amount of processing time from 13 days on a single state-of-the-art four-core desktop computer to about 52 min on 2112 CPU cores. The parallel algorithm is based on a two-dimensional partition of the entire noise dataset collected by the seismic array. The computational load, inter-process communication overhead and memory usage are well balanced across the entire 2D MPI process array. The *pSIN* code reads the entire noise dataset only once at the beginning of the execution and writes out all stacked cross-correlations at the end of the execution. The entire parallel algorithm has shown excellent strong scalability from 80 CPU cores to over 2000 CPU cores.

Compared with the Hadoop implementation presented in Addair et al. (2014), the *pSIN* code is relatively lightweight and easier to use on conventional distributed-memory clusters. It does not need any additional preparation steps that may require non-trivial changes to the cluster. It does not eliminate the I/O overhead, but it has minimized the total number of I/O operations involved in the entire calculation. Hence, as we have demonstrated herein, this algorithm is useful for speeding up the seismic interferometry calculations of large-*N* data using small to medium-sized conventional distributed-memory computer clusters.

In addition to seismic interferometry based on cross-correlation (e.g., Bensen et al., 2007), our parallelization algorithm can also be easily adapted to seismic interferometry based on deconvolution, coherency and transfer functions (e.g., Vasconcelos and Snieder, 2008a; 2008b; Prieto et al., 2011) and also acoustic/seismic noise data collected by different types sensors. We expect *pSIN* to be a convenient and useful tool for significantly speeding up ambient-noise seismic interferometry processing and ambient-noise tomography.

## Acknowledgment

This research performed by the Wyoming Center for Environmental Hydrology and Geophysics (WYCEHG) EPSCoR RII Track-1 team at the Southern Sierra Critical Zone Observatory (SSCZO) was funded by the National Science Foundation, USA under Grant Number EAR-1331939. Computational resources were provided by the Advanced Research Computing Center (ARCC) at University of Wyoming. Software development was supported by Crust LLC.

## Appendix A. Supplementary material

Supplementary data associated with this article can be found in the online version at http://dx.doi.org/10.1016/j.cageo.2016.05.003.

## References

Addair, T.G., Dodge, D.A., Walter, W.R., Ruppert, S.D., 2014. Large-scale seismic signal analysis with Hadoop. Comput. Geosci. 66, 145–154.
Bensen, G.D., Ritzwoller, M.H., Barmin, M.P., Levshin, A.L., Lin, F., Moschetti, M.P., Shapiro, N.M., Yang, Y., 2007. Processing seismic ambient noise data to obtain reliable broad-band surface wave dispersion measurements. Geophys. Journal. Int. 169 (3), 1239–1260.
Derode, A., Larose, E., Tanter, M., de Rosny, J., Tourin, A., Campillo, M., Fink, M., 2003. Recovering the Green's function from field-field correlations in an open scattering medium (L). J. Acoust. Soc. Am. 113, 2973–2976.
Freed, D., 2008. Cable-free nodes: the next generation land seismic system. Lead. Edge 27 (7), 878–881.
Lin, F.C., Li, D., Clayton, R.W., Hollis, D., 2013. High-resolution 3D shallow crustal structure in Long Beach, California: application of ambient noise tomography on a dense seismic array. Geophysics 78 (4), Q45–Q56.
Lobkis, O.I., Weaver, R.L., 2001. On the emergence of the Green's function in the correlations of a diffuse field. J. Acoust. Soc. Am. 110, 3011–3017.
Prieto, G.A., Denolle, M., Lawrence, J.F., Beroza, G.C., 2011. On amplitude information carried by the ambient seismic field. Comptes Rendus Geosci. 343 (8), 600–614.
Rickett, J., Claerbout, J., 1999. Acoustic daylight imaging via spectral factorization: helioseismology and reservoir monitoring. Lead. Edge 18, 957–960.
Ritzwoller, M.H., Lin, F.C., Shen, W., 2011. Ambient noise tomography with a large seismic array. Comptes Rendus Geosci. 343 (8), 558–570.
Snieder, R., 2004. Extracting the Green's function from the correlation of coda waves: a derivation based on stationary phase. Phys. Rev. E 69, 046610.
Vasconcelos, I., Snieder, R., 2008a. Interferometry by deconvolution: part 1—theory for acoustic waves and numerical examples. Geophysics 73 (3), S115–S128.
Vasconcelos, I., Snieder, R., 2008b. Interferometry by deconvolution: part 2—theory for elastic waves and application to drill-bit seismic imaging. Geophysics 73 (3), S129–S141.
Wapenaar, K., 2004. Retrieving the elastodynamic Green's function of an arbitrary inhomogeneous medium by cross correlation. Phys. Rev. Lett. 93, 254301-1–254301-4.
Weaver, R.L., Lobkis, O.I., 2004. Diffuse fields in open systems and the emergence of the Green's function. J. Acoust. Soc. Am. 116, 2731–2734.