

# Strategies for network parallel 3-D phase-shift migration

*Tong Chen\* and Dave Hale‡*

\* Center for Wave Phenomena, Dept. of Mathematics, Colorado School of Mines, Golden, CO.

‡ Formerly Center for Wave Phenomena, Dept. of Geophysics, Colorado School of Mines, Golden, CO. Now at Advance Geophysical Corporation, Englewood, CO.

**ABSTRACT**

Their excellent performance-to-cost ratio makes the use of network-connected workstations for parallel computing attractive. However, in 3-D seismic processing, the cost for disk input/output operations and network communication in such systems can be prohibitively high. Here, we suggest doing not only computation but also data input/output operations in parallel by distributing the dataset to several disks, each attached to its own workstation. When data flow among disks is needed, we design an optimal strategy to make the flow also run in parallel so as to reduce the communication cost.

Without specific attention paid to load balancing in the migration, some workstations would be idle for much of the time that others are computing. Balancing the computational load among the workstations can thus also significantly aid the overall performance. Load balancing generally requires increased communication effort in a network environment, so we give a method here to obtain optimal load balancing while keeping the communication cost minimal.

Based on the strategies of data distribution, data communication and load balancing, a 3-D phase-shift migration of moderate size ( $1000 \times 1000 \times 1000$ ) can be performed overnight on five network-connected workstations.

**INTRODUCTION**

The use of network-connected computers to do parallel computing is of current interest to geoscientists (Koshy et al., 1991; Black and Su, 1991). Advances in the processing power of workstations and communication speed of networks make such network parallel computing (NPC) feasible today. By running many high-performance

workstations in parallel, we may achieve speeds comparable to those of supercomputers, at higher performance-to-cost ratio. Further improvements in communication speed using optical fibers will make NPC even more attractive (Almasi et al., 1991). Meanwhile some software systems, e.g. Linda (Carriero and Gelernter, 1989) and PVM (Beguelin et al., 1991) make NPC easy to implement.

For any application, the deciding factor in NPC is the ratio of computation to communication costs. Certain geoscience applications, such as seismic modeling and imaging, seem to be well suited for NPC because they have high computation-to-communication ratios. A detailed discussion of speed-up relative to single-workstation computation for a 2-D migration application, under various circumstances, can be found in Almasi (1991).

For many 2-D seismic applications, where the entire dataset can reside in memory, NPC implementations are not difficult. Figure 1 illustrates the diagram for this kind of implementation using a message-passing system.

In 3-D seismic processing, however, the situation is different. For many 3-D applications, besides so-called *inline* processing, *crossline* processing is also required, which in turn requires that the dataset be accessed from orthogonal directions. Moreover, 3-D datasets are typically so large that they cannot reside in memory, which necessitates lots of disk input/output (I/O) operations, both sequential and random access. The cost for disk I/O, especially random-access I/O, is so high in 3-D seismic processing that high speed-up can hardly be achieved using the scheme illustrated in Figure 1. Also, a 3-D dataset may be too large to be stored on the disks attached to a single workstation, which makes NPC implementation difficult. Black and Su (1991) suggested using a large machine capable of managing the large volume of data. Here, we present a different strategy of using several workstations, each with its own disk (Figure 2). Fortunately, having several disks available is a sufficiently common

situation in network environments. Also, using several disks attached to different workstations makes it possible to perform disk I/O in parallel.

For NPC implementations of 3-D seismic processing, distribution of the data among several disks is necessary. It turns out, then, that the strategy for data distribution can dominate the efficiency of processing. Using 3-D phase-shift migration (Gazdag, 1978) as an example, we suggest a particular strategy of data distribution.

3-D phase-shift migration is an example wherein accessing of data on different disks is needed. Unfortunately, this access requires network communication, which is expensive using network hardware and software now available. We design a method to make the network communication run in parallel to reduce the communication cost.

Balancing the computational workload among the workstations is yet another way to improve performance. In 3-D phase-shift migration, because computations need not be done for the evanescent region, load balancing which generally requires more communication in a network environment, nevertheless can improve processing efficiency. Here, we design a method to obtain optimal load balancing while keeping the communication cost minimal.

By following a combination of the above strategies, we performed a 3-D turning wave, post-stack migration (Hale et al., 1991) for a dataset of size  $500 \times 500 \times 500$  on five IBM RS/6000 workstations in about 1.4 hours. Based on this performance, we predict that in the same environment (but with more disk storage than we had available), we could perform the same type of migration for a 3-D dataset of size  $1000 \times 1000 \times 1000$  in about 15 hours.

## DATA DISTRIBUTION

In 3-D seismic processing, the cost of disk I/O is very high because of the huge amount of data involved. In a network environment, the dataset is typically too large to be stored on a single disk attached to one workstation. A 3-D seismic survey of dimension 1000 samples  $\times$  1000 traces  $\times$  1000 lines, for example, requires 4 Gbytes of storage. The disks we usually have for our particular workstations each have 1.2 Gbytes capacity, so that we must distribute the data to several disks. Of course, it is possible to attach all the disks to one workstation, in which case all the disk I/O is performed by only one workstation. A better way to handle this is to use several disks, each attached to its own workstation (Figure 2). In so doing, we can perform disk input/output, as well as the computation, in parallel.

In some applications, such as deconvolution, which is a one-dimensional process, the choice of data distribution is simple. We can store a portion of data (perhaps many seismic lines) on each disk. Each workstation then processes only its own assigned piece of data. After all the workstations have finished their own processing, the results are composited. Little data communication is necessary.

In many other cases, however, since we must do both inline and crossline processing, a great deal of data communication is needed. One example is the 2-D fast Fourier transform (FFT) along inline and crossline dimensions in 3-D phase-shift migration or modeling programs. A schematic instruction flow for 3-D phase-shift migration might be as follows:

FFT  $f(t, x, y)$  to  $F(t, k_x, y)$

FFT  $F(t, k_x, y)$  to  $F(t, k_x, k_y)$

For all  $k_y$  {

For all  $k_x$  {

$$\begin{aligned}
& \text{Migrate } F(t, k_x, k_y) \text{ to } G(\tau, k_x, k_y) \\
& \quad \} \\
& \quad \} \\
& \text{IFFT } G(\tau, k_x, k_y) \text{ to } G(\tau, k_x, y) \\
& \text{IFFT } G(\tau, k_x, y) \text{ to } g(\tau, x, y),
\end{aligned}$$

where  $t, x, y$  correspond to the time, inline and crossline directions respectively;  $k_x, k_y$  are the inline and crossline wavenumbers;  $\tau$  is the migrated time; and IFFT denotes inverse FFT. Let's consider several strategies for this example.

### **T-slice**

Since 2-D FFT is a two-dimensional process, the seemingly ideal method of data distribution would be to let each workstation store a set of time slices (Figure 3a). For every time slice, each workstation reads the X-Y data plane and performs a 2-D FFT. In this way, each workstation can process its own piece of data. No network communication seems necessary, but this strategy requires that time T be the slowest computational dimension, while in real applications, it is typically the fastest dimension. Transposition of the entire dataset is thus needed, which is difficult due to its large size.

### **Y-slice**

Depending on the format of the data recorded, we can make Y-slices (i.e., data in which the crossline coordinate is held fixed), as shown in Figure 3b. Every workstation stores a set of Y-slices (X-T sections), or seismic lines. Obviously, in this way, an FFT along the X-direction is easy. For every Y slice, each workstation reads the X-T data plane and performs an FFT along the X-direction. But, the FFT along the Y-direction is difficult, since it requires data that are on different disks, and thus

indicates much network communication. Furthermore, it is difficult for the FFT along the Y-direction to run in parallel, since at one moment, several workstations may need to access the data on one disk, resulting in I/O conflict.

### **XY-slice**

From the above, we have seen that the Y-slice makes the FFT along the X-direction easy to implement. To deal with the problem of computing the FFT along the Y-direction, we make additional X-slices (Figure 3c). We do this by chopping the dataset into an  $n \times n$  matrix of “block” files, where each of the  $n$  workstations has  $n$  block files on its local disk. Each block file contains seismic traces for a rectangular subset of  $(x, y)$  locations. For the FFT along the X-direction, each workstation simply reads data from the block files on its local disk and forms an X-T data plane, as is done for the X-slice distribution. For the FFT along the Y-direction, we would also like for the workstations to read data from their local disks. One way to satisfy this need is to swap the block files and put the data for a given Y-T data plane onto a single disk. This swapping is achieved with the use of the additional X-slices. After the data for the FFT along the Y-direction are on one disk, network communication can be avoided.

## **DATA COMMUNICATION**

Using the 2-D FFT in 3-D phase-shift migration as an example, we examined different strategies of data distribution. Even though the XY-slice method seems to be the best for this application, in this strategy swapping of the block files is needed, after the FFT along the X-direction. This swapping requires network communication, so we seek an efficient method for doing it.

When we use  $n$  workstations, we have  $n \times n$  block files. Swapping the block files is like transposing an  $n$  by  $n$  matrix. Clearly, the block files on the diagonal of the matrix need no swapping. The other  $n(n - 1)$  files are the ones to be swapped. Swapping the block files sequentially requires the time necessary for  $n(n - 1)$  read and the same number of write operations. Notice that when swapping sequentially, only one workstation and one disk are busy at any given time. The other workstations and disks are idle waiting for the swap to finish. This suggests the possibility of swapping in parallel. Since swapping the block files requires network communication as well as disk I/O, the relative performances of all of these operations will determine our parallel strategy. We did some experiments to estimate the costs of network communication and disk I/O, since, unfortunately, theoretical analysis of these costs is difficult.

Figure 4a shows the costs of local disk read and write operations, as functions of the amount of data involved in some given operation, i.e. the package size. From the figure, we can see that reading from a local disk is less expensive than writing to a local disk. Except for packages of size larger than 800K bytes, the costs do not change much as the package size varies. Figure 4b shows the costs of remote disk read and write operations, through a token ring network. When operating with a remote disk, the cost of writing is much more expensive than that of reading from it.

A swap of block files requires the same number of read and write operations. Also since the dataset is distributed, remote disk access is thus unavoidable. From Figures 4a and 4b, we see that the best combination of read and write operations is to read from a remote disk and write to a local one, as shown in Figure 5.

Also in Figure 5, we see that to swap the block files in parallel, two workstations may have to compete for access to the same disk. Our experimental results indicate that the competition between one read and one write operation on a given disk is not

severe; that is, performing the read and write operations simultaneously, in competition, is still faster than first performing all the read operations and then all the write operations.

Based on the above reasoning, an optimal strategy for swapping the block files should satisfy the following constraints:

1. make every write operation local;
2. for a given disk, have only one read operation by a remote workstation and one write operation by the local workstation occur at a time;
3. for efficient use of disk space, generate only one temporary file at each step when swapping.

Figure 6 illustrates the strategy for the swap when using five workstations. The block files on the NE-SW diagonal need no swapping. Our strategy is to start from the diagonal. First, swap the block files on the sub-diagonal and super-diagonal. Notice that for  $n$  workstations, only  $n - 1$  elements are on the sub-diagonal and  $n - 1$  on the super-diagonal. In a periodic sense (modulo  $n$ ), however, two more block files are also on the sub- and super-diagonals. These are the block files shown in the upper left and lower right of Figure 6a. We can swap these  $2n$  block files in parallel. Second, swap the block files one step further from the diagonal, as illustrated in Figure 6b. For  $n > 5$ , repeat the process for blocks successively further from the diagonal. In this way, we can satisfy all three above constraints. Instead of swapping two elements of the matrix at each step, we actually swap two diagonal columns. For  $n = 5$ , the processes are shown as follows:

$$n = 5$$

$$\begin{array}{ccc}
& \text{Swap 1} & \text{Swap 2} \\
\left( \begin{array}{c} (0, 1) \\ (1, 2) \\ (2, 3) \\ (3, 4) \\ (4, 0) \end{array} \right) & \Rightarrow & \left( \begin{array}{c} (1, 0) \\ (2, 1) \\ (3, 2) \\ (4, 3) \\ (0, 4) \end{array} \right), & \left( \begin{array}{c} (0, 2) \\ (1, 3) \\ (2, 4) \\ (3, 0) \\ (4, 1) \end{array} \right) & \Rightarrow & \left( \begin{array}{c} (2, 0) \\ (3, 1) \\ (4, 2) \\ (0, 3) \\ (1, 4) \end{array} \right).
\end{array}$$

Here  $(i, j)$  refers to the  $j$ th block file on workstation  $i$ . When using  $n$  workstations, we need  $n(n - 1)$  read and the same number of write operations, to swap the block files. Performing this I/O in parallel, we need only the time for  $n - 1$  read and the same number of write operations. In this way, we can expect a speed-up factor of  $n$  for the data communication. In one test, with  $n = 5$ , we achieved a speed-up factor of about six, through the combination of swapping scheme and care to have all write operations be local.

## LOAD BALANCING

For 3-D phase-shift migration, the main effort is in the Fourier transformations along the X- and Y-directions, because it involves intensive data communication. The migration part is relatively easy since each wavenumber can be migrated independently. However, another factor in obtaining good performance is load balancing.

To achieve stability and reduce costs in migration, we usually choose to attenuate the wavenumbers in the evanescent region (i.e., the higher horizontal wavenumbers). These wavenumbers do not need to be migrated. As a result, as seen in Figure 7, the computational workload in 3-D phase-shift migration varies for different wavenumbers. If, to avoid network communication, each workstation migrates only the data on its local disk, then the computational workload is severely unbalanced. For example, in Figure 7, workstation 1 has much more work to do than does workstation 5; workstation 5 is idle most of the time. It is best to minimize the idle time on the

various workstations. A common approach in parallel computing is the *pool-of-tasks* method (see e.g. Golub and Van Loan, 1989). In the pool-of-tasks approach, a list of remaining tasks is maintained. When a workstation completes a task, it checks the list. If the list is nonempty, then the workstation selects a task from the “pool” of remaining tasks, and the chosen task is then removed from the pool. In this way, workload is balanced dynamically. A shortcoming of this approach, however, is that one has no control over the potentially intensive network communication, because the tasks are assigned to workstations randomly. To obtain optimal load balancing while keeping network communication to a minimum, we suggest a modified pool-of-tasks method.

We define each task to be the migration of one block file of data. Each workstation migrates the block files on its local disk first, and after all the local data for a given workstation are migrated, that workstation looks for the unmigrated block files in the list of tasks, selects one (the one that will require the most work), and migrates it. This procedure of processing the local data first and then selecting from the list gives good load balancing, as well as minimal network communication. For the example in Figure 7, after workstation 5 has migrated all its local files, it may access the data on workstation 1 and share some of its workload. In our experiment, with  $5 \times 5$  block files and no load balancing, workstation 1 required six times as long as workstation 5 to do the migration. After load balancing, the variation in time spent in migration was much reduced among the workstations, and the ratio between the maximum and minimum became about 5 : 4. Also, since only two to three accesses of remote block files were required for the load balancing, this achieved good load balancing with minimal network communication. As a result, we obtained a speed-up factor of about two for the migration part, relative to the case without load balancing.

## CONCLUSION

While one does not generally think of having a cluster of workstations for the sole purpose of doing large-scale processing in parallel, installations with such clusters provide opportunities for such processing (overnight, for example). We have suggested some methodologies for doing 3-D phase-shift migration in parallel on a cluster of workstations. Efficient parallel processing requires much more than just doing the computation in parallel. By distributing the dataset, we do data input/output in parallel as well. To reduce data communication cost, we have devised a strategy for controlling the data flow, and by the modified pool-of-tasks method, we minimize idle time on the workstations that otherwise would have relatively little migration to do.

To summarize, let us compare what we have achieved here with these schemes. It would be good to compare the speed of the network-parallel approach with that obtained with a single workstation. Unfortunately, we could not even do the test on a single workstation because of limited disk space. Likewise, we might envision using either the T-slice or Y-slice arrangement for distribution of the data, but that would clearly be extremely slow unless something clever is done. Our choice was to develop the block file scheme. On five workstations, for the swapping effort we obtained a speed-up factor of six using the parallel swapping scheme, with care taken to have the write operations local. We also obtained a speed-up factor of two in the migration stage by load balancing. For a test data set of size  $500 \times 500 \times 500$ , on five IBM RS/6000 workstations (three Model 530 and two Model 520), the entire process took about 1.4 hours, twice as fast as without parallel swapping and load balancing. For a data set of size  $1000 \times 1000 \times 1000$ , still on five workstations, we predict a total processing time of about 15 hours, based on the operation counts for the different stages:  $N^3 \log(N)$  for the FFT,  $N^3$  for the swapping and  $N^4$  for the migration.

**ACKNOWLEDGMENTS**

Thanks to Dr. Ken Larner for his review of this paper. Financial support for this work was provided by the members of the Consortium Project on Seismic Inverse Methods for Complex Structures at the Center for Wave Phenomena, Colorado School of Mines.

**REFERENCES**

- Almasi, G., McLuckie, T., Bell, J., Gordon, A., and Hale, D., 1991, Parallel distributed seismic migration: IBM Research Report, RC17323.
- Beguelin, A., Dongarra, J., Geist, A., Manchek, R., and Sunderam, V., 1991, A Users' Guide to PVM—Parallel Virtual Machine, Oak Ridge National Laboratory.
- Black, J., and Su, C., 1991, Networked parallel 3-D depth migration: 61st Ann. Internat. Mtg., Soc. Expl. Geophys., Expanded Abstracts, 349-356.
- Carriero, N., and Gelernter, D., 1989, Linda in context: Communications of the ACM, 32, 4(April), 444-458.
- Gazdag, J., 1978, Wave equation migration with the phase-shift method: Geophysics, 43, 1342-1351.
- Golub, G. H., and Van Loan, C. F., 1989, Matrix Computations, 260-300.
- Hale, D., Hill, N. R., and Stefani, J. P., 1991, Imaging salt with turning seismic waves: 61st Ann. Internat. Mtg., Soc. Expl. Geophys., Expanded Abstracts, 1171-1174.
- Koshy, M., Pereyra, V., and Meza, J., 1991, Distributed computing application in forward and inverse geophysical modeling: 61st Ann. Internat. Mtg., Soc. Expl. Geophys., Expanded Abstracts, 349-352.

### Figure Captions

**FIG. 1.** Working scheme for a message-passing system in NPC when data can reside in memory. The black arrows denote commands, and the clear ones, data. The master reads data from the disk and sends them out to the workers. After being processed, the data are sent back to the master, which writes them to the disk. Notice that all the disk I/O is carried out by the master.

**FIG. 2.** Working scheme for a message-passing system in NPC when data are distributed among several disks. The black arrows denote commands, and the clear ones, data. The master functions only to coordinate the workers. Each worker reads data from its own disk and writes data to its own disk. In this case, disk I/O is performed in parallel by different workers.

**FIG. 3.** Different strategies of data distribution. (a) is the T-slice. (b) is the X-slice. (c) is the XY-slice.

**FIG. 4.** Costs for network communication and disk read/write operations. (a) is the cost for local disk read and write. (b) is the cost for remote disk read and write.

**FIG. 5.** Reading from a remote disk and writing to the local one can reduce the communication cost.

**FIG. 6.** Strategy of parallel swapping of block files. In this example of five workstations, each with five block files on local disk, the swap is done in two steps. (a) is the first step, where the block files along the sub- and super-diagonals of the matrix get swapped. Those along the diagonal need no swapping. Also, the files at the corners of the matrix, which can be considered as along the sub- and super-diagonals, are swapped. These swaps are done in parallel. (b) is the second step, which finishes all the swaps in this case. (c) is the detail of

a representative swap. First read from a file and write it to a temporary file. Then write the file to be swapped to the file just read. Finally, rename the temporary file to the file swapped.

**FIG. 7.** Contours of equal computational workload in 3-D phase-shift migration. Each block here represents a block file containing seismic traces of certain wavenumbers. The data in the darker region require more computation. If each of the five workstations migrates only its local data (a horizontal strip of five blocks), the workload is not balanced.

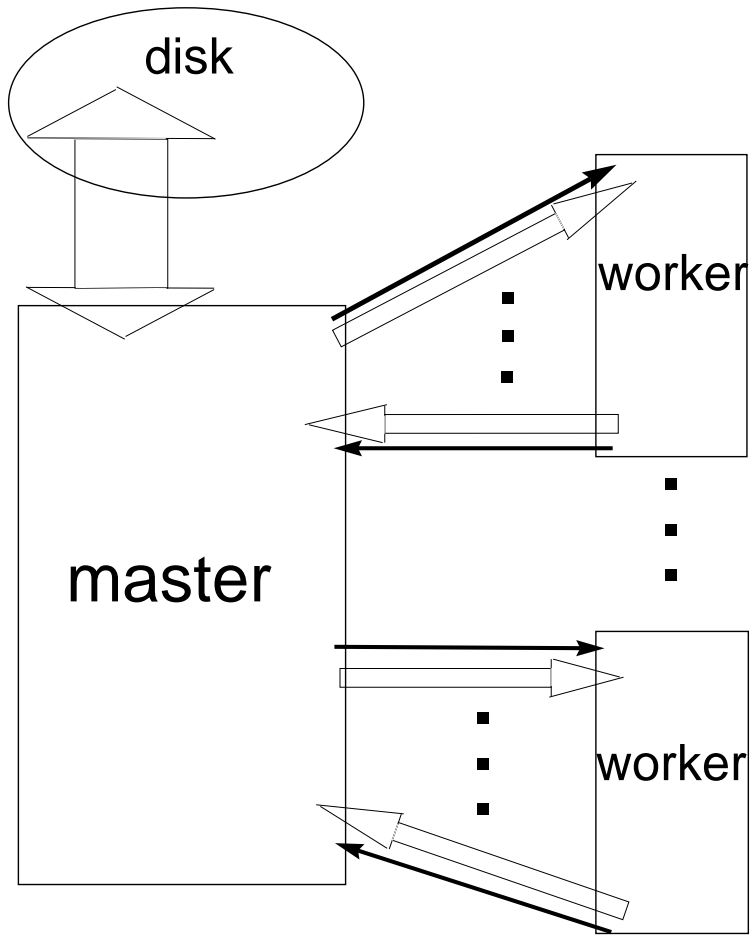


FIG. 1.

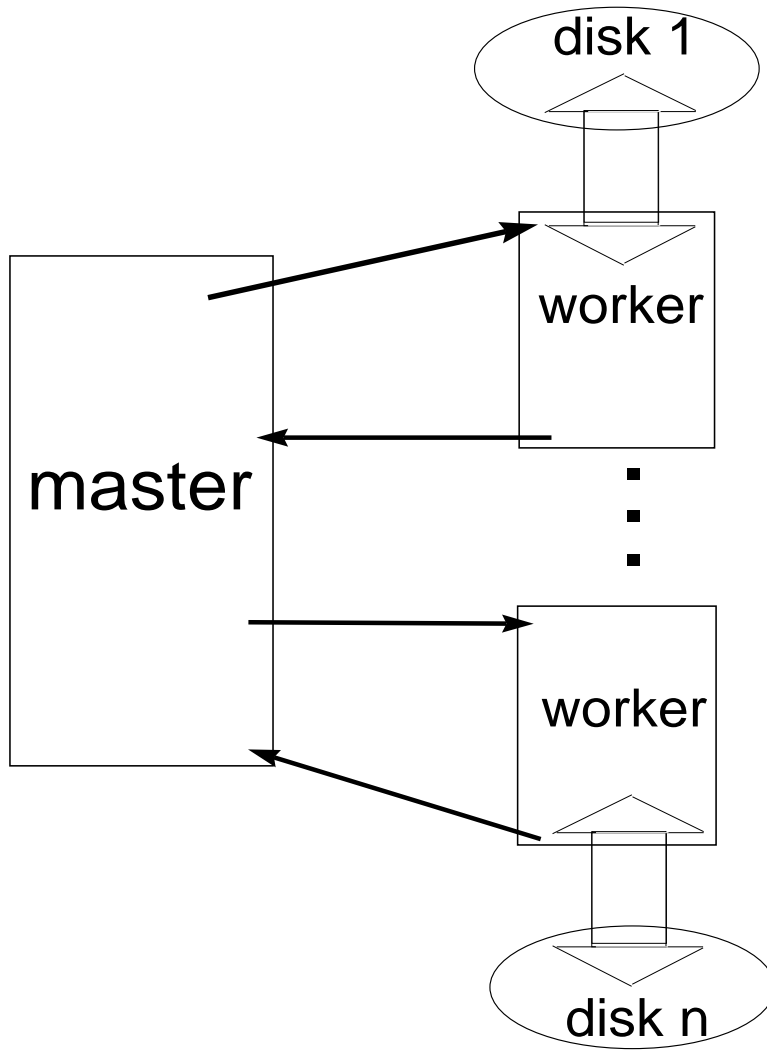


FIG. 2.

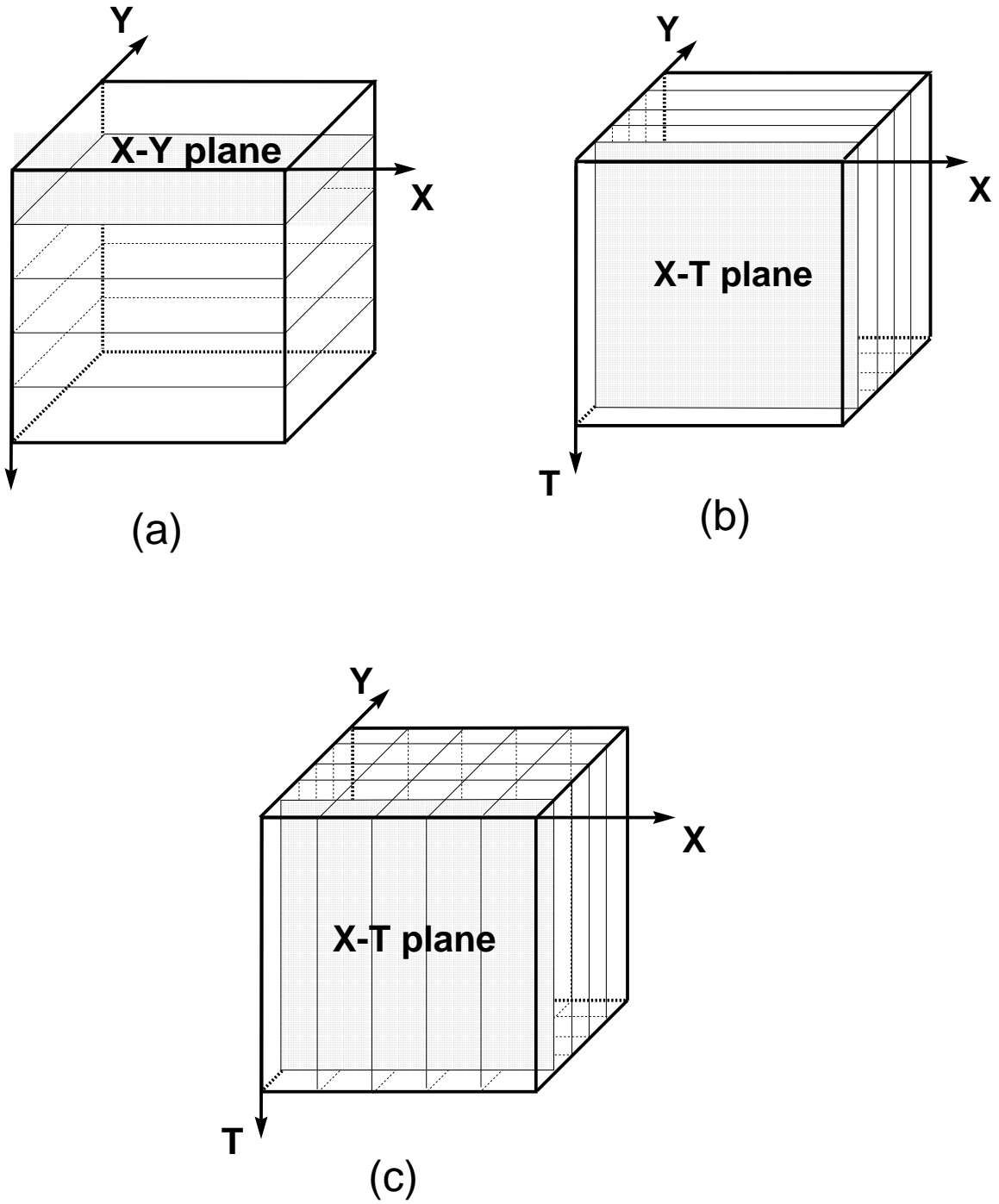


FIG. 3.

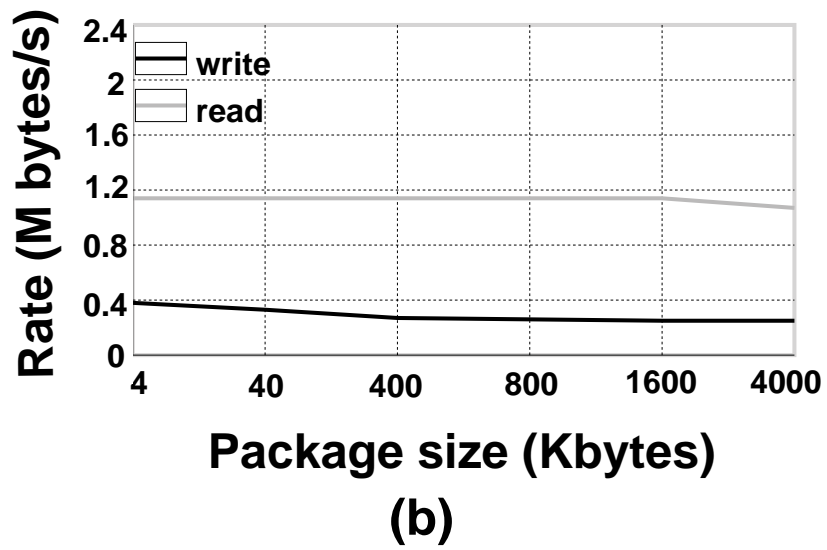
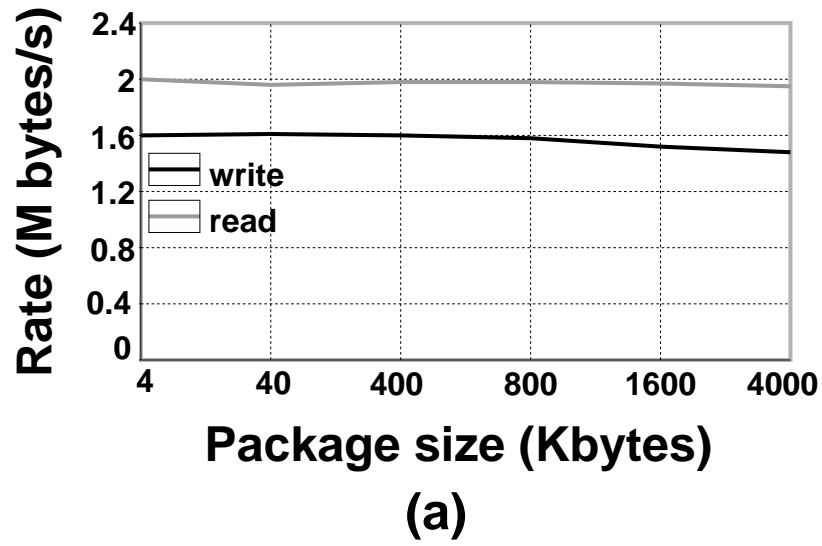


FIG. 4.

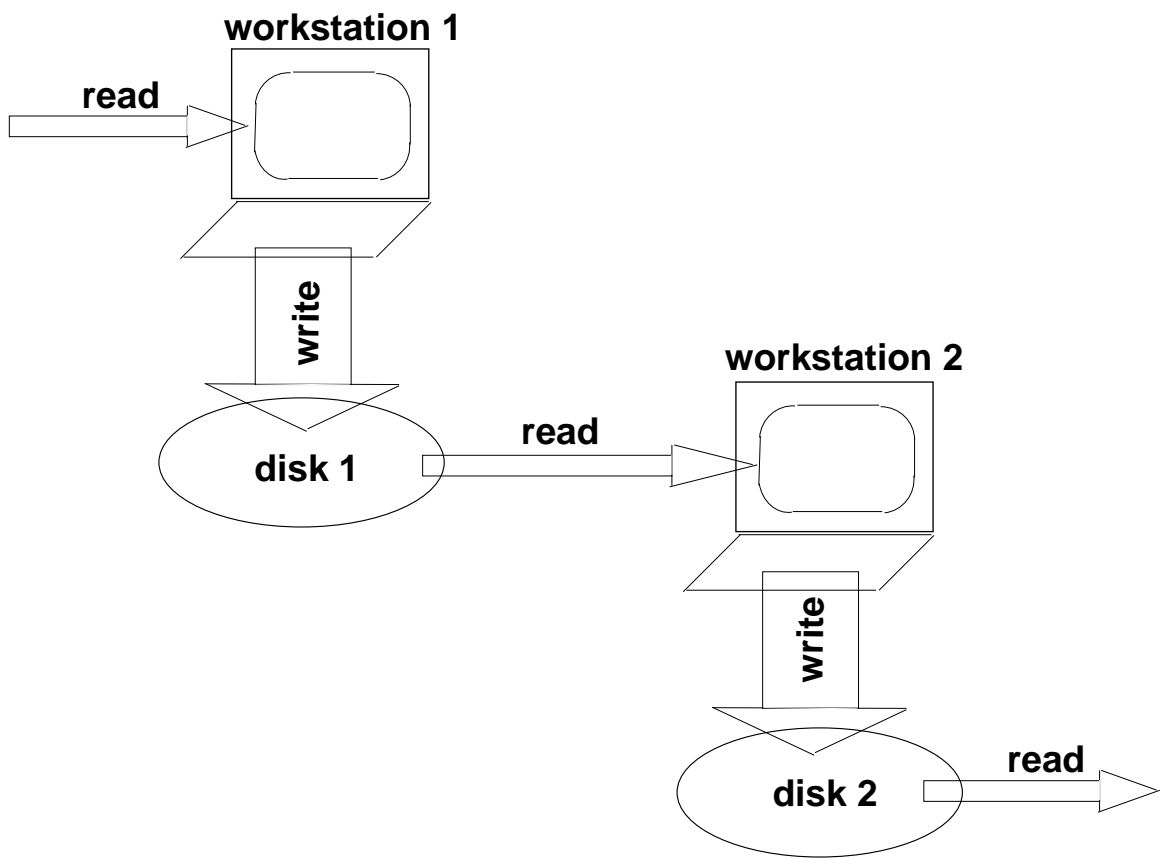


FIG. 5.

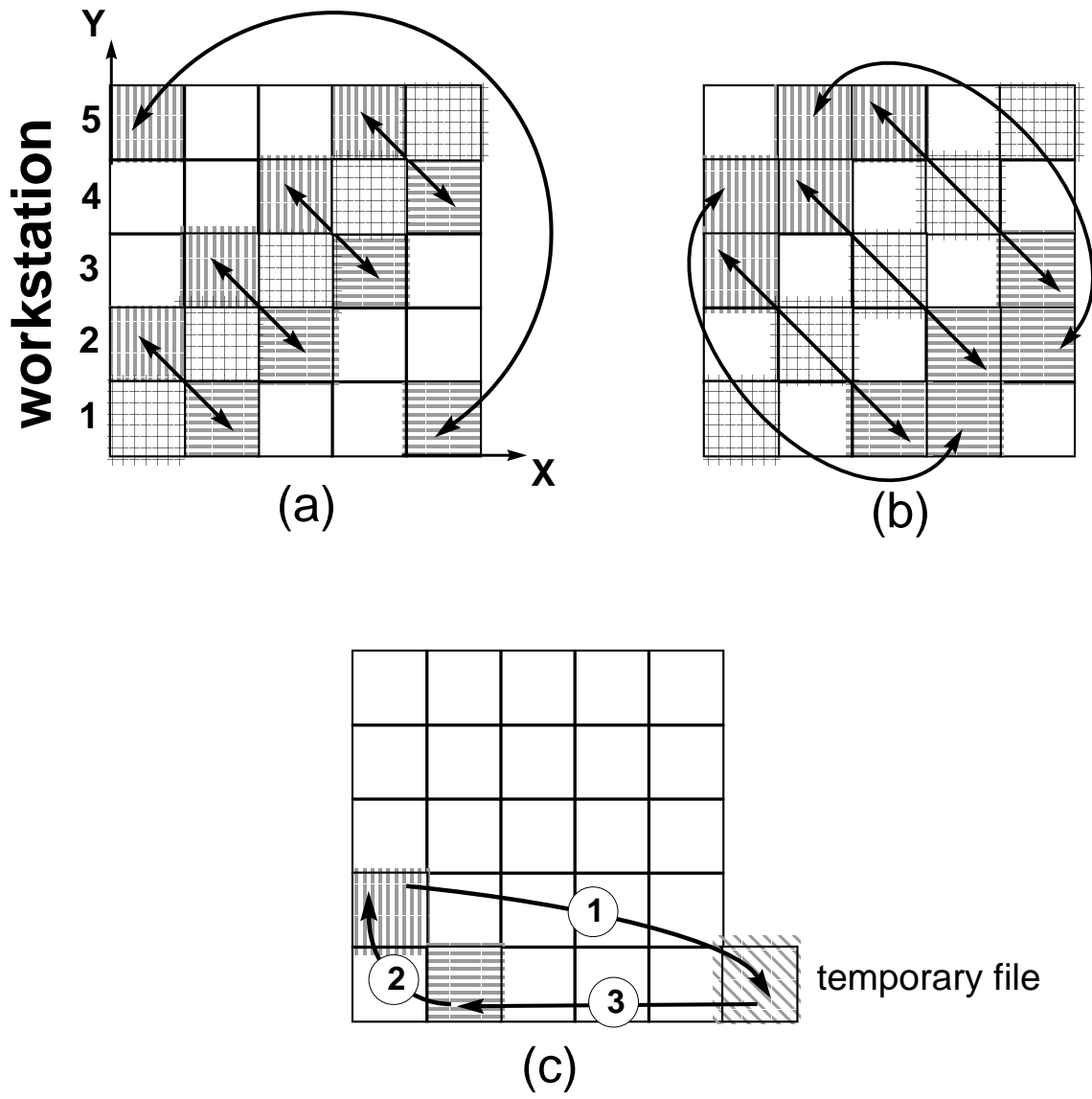


FIG. 6.

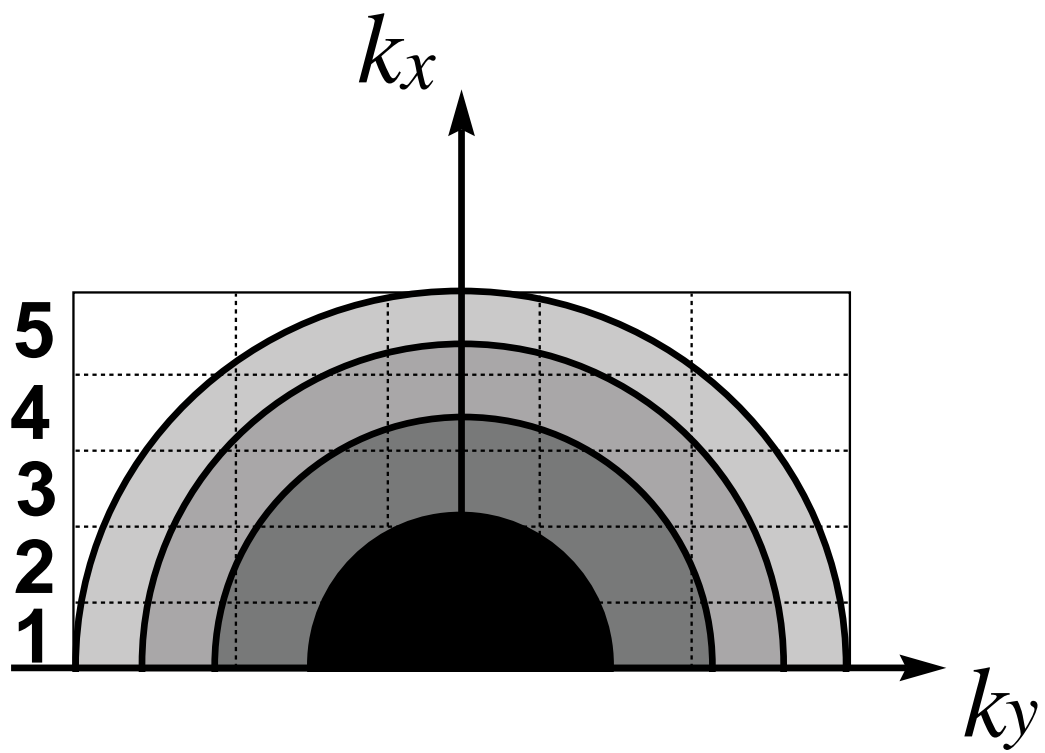


FIG. 7.