Parallel Partial Reduction for Large-Scale Data Analysis and Visualization

Wenbin He [1][*] Hanqi Guo [2][†] Tom Peterka [2][‡] Sheng Di [2][§] Franck Cappello [2][¶] Han-Wei Shen [1][‖]

1) Department of Computer Science and Engineering, The Ohio State University, Columbus, OH, USA
2) Mathematics and Computer Science Division, Argonne National Laboratory, Lemont, IL, USA

ABSTRACT

We present a novel partial reduction algorithm to aggregate sparsely distributed intermediate results that are generated by data-parallel analysis and visualization algorithms. Applications of partial reduction include flow trajectory analysis, big data online analytical processing, and volume rendering. Unlike traditional full parallel reduction that exchanges dense data across all processes, the purpose of partial reduction is to exchange only intermediate results that correspond to the same query, such as line segments of the same flow trajectory. To this end, we design a three-stage algorithm that minimizes the communication cost: (1) partitioning the result space into groups; (2) constructing and optimizing the reduction partners for each group; and (3) initiating collective reduction operations for all groups concurrently. Both theoretical and empirical analyses show that our algorithm outperforms the traditional methods when the intermediate results are sparsely distributed. We also demonstrate the effectiveness of our algorithm for flow visualization, big log data analysis, and volume rendering.

Index Terms: I.3.1 [COMPUTER GRAPHICS]: Hardware Architecture—Parallel processing; I.3.2 [COMPUTER GRAPHICS]: Graphics Systems—Distributed/network graphics

1 INTRODUCTION

Data parallelism is the default pattern in today's scalable data analysis and visualization applications. In data parallelism, large input data are partitioned and distributed to parallel processes, and then each process generates a set of intermediate results. The final output is combined from the intermediate analysis results that are distributed in different processes through parallel reduction. For example, in the online analytical processing (OLAP) of multidimensional data, one can distribute the data into parallel processes, compute intermediate results such as histograms in each process independently, and then combine the intermediate results. In the Lagrangian-based flow trajectory analyses for aerodynamics, climate, and weather simulation data, one can partition the data domain into blocks for parallel trajectory tracing. As a result, each trajectory is generated and distributed in different processes for further aggregation and analysis. In the application of parallel volume rendering, the data is first partitioned into blocks and distributed into parallel processes. The distributed data blocks are then rendered in parallel, and the intermediate rendering results are combined into one final image for visualization.

The focus of this paper is to scale the parallel reduction stage in data-parallel analysis and visualization algorithms. Although the parallel reduction is a well-studied problem for general computational science applications, we must redesign the parallel reduction for data analysis and visualization. The traditional and general parallel reduction algorithm, which is regarded as *full reduction* in this paper, is designed to aggregate data that are distributed in all parallel processes. Each process is involved in the parallel communication to compute the correct outputs in full reduction.

In this study, we propose *partial reduction*, as opposed to full reduction, to reduce communication cost for data analysis and visualization. The rationale of partial reduction is based on the observation that the intermediate partial analysis results are sometimes sparsely distributed in parallel processes. We can thus optimize the parallel reduction by taking the sparsity into consideration.

One of the driver problems is Lagrangian flow trajectory analysis. For example, in Figure 1 we need a parallel reduction algorithm to accumulate the intermediate analysis results—lengths of streamlines in this case—across the parallel processes. These streamline segments are generated by the data-parallel particle tracing and thus distributed in parallel processes. The length of each streamline can be derived by summing up the length of the streamline segments in every process. As illustrated by the blue line in Figure 1, however many of the processes do not have segments for this streamline. We have to assign a zero length for these processes, in order to use the traditional full reduction algorithm. Even worse, these "placeholder" zero values are exchanged over processes during the reduction, causing extra data movement and communication overhead. In the following sections, we will discuss how Lagrangian flow trajectory analysis could benefit from the partial reduction.

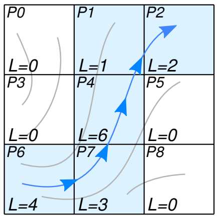


Figure 1: Computing the lengths of streamlines through parallel reduction. The streamline segments are distributed in 9 processes (i.e., P0 to P8). The length of the streamline colored in blue within each process is denoted as L.

Another driver problem is the OLAP data cube that assists interactive visualization of multidimensional data arrays such as financial data and big log data. The output of data cubes is usually a group of large but sparse histograms, and users can slice and dice the data with respect to attributes, space, and time to understand the data distribution in different dimensions. Although existing data cube implementations, such as Nanocubes [19], imMens [20], and Kylin [1], are designed for either single-node or big-data clusters, we found the data cube hard to scale in large HPC environments. The main bottleneck is the parallel reduction of large but sparse data cube outputs. We will demonstrate how partial reduction could accelerate OLAP data cubes for real-time exploration of tens of millions of system logs generated by a supercomputer.

Other driver problems include volume rendering [22] and data-parallel density estimation [30], which sparsely distribute intermediate rendered images and density fields, respectively. For these large-scale data analysis and visualization applications, we advocate and explore using partial reduction, instead of full reduction,

[*] e-mail: he.495@osu.edu
[†] e-mail: hguo@anl.gov
[‡] e-mail: tpeterka@mcs.anl.gov
[§] e-mail: sdi1@anl.gov
[¶] e-mail: cappello@mcs.anl.gov
[‖] e-mail: shen.94@osu.edu

to combine sparse intermediate results.

The main research challenge is to minimize the communication cost during the parallel reduction of sparse data. A naive approach to implement partial reduction is to first partition the sparse data into groups, to create a communicator for the set of processes that have the data, and then to execute an `MPI_Reduce()` for each communicator. However, the naive approach is not scalable because of design limitations in MPI [13]. First, creating each subcommunicator is expensive because it involves collective operations. Second, communicators are limited resources because memory bounds. Third, the only way to create subcommunicators is to call `MPI_Comm_create()` sequentially multiple times, because each process is usually associated to multiple groups in our applications. As shown in Figure 2, the parallel reduction of sparse data with `MPI_Comm_create()` and `MPI_Reduce()` takes much more time than do full reduction and partial reduction.

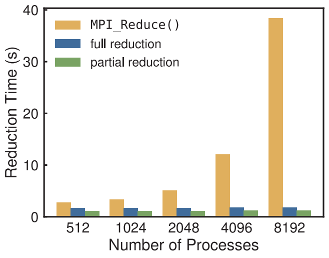


Figure 2: Reduction time of sparse data with `MPI_Reduce()`, full reduction, and partial reduction.

We instead design a scalable and lightweight partial reduction algorithm that minimizes communication and synchronization costs. Our partial reduction algorithm has three stages. First, we partition the result space into groups to conduct partial reductions per group. Second, we construct reduction partners for each group. The reduction partners are a subset of processes that contain the partial results of each group. Third, we concurrently conduct the radix-*k* [31] reduction operations for all communication partners in parallel.

We analyze the performance of partial reduction both theoretically and empirically. The theoretical analyses show that our partial reduction performs better than traditional full reduction algorithms when the partner matrix is sparse. The empirical performance benchmark also demonstrates the same conclusion. Although there is some overhead in the first stage in our algorithm, our method usually outperforms full reductions in real applications. In summary, the contributions of this study are threefold:

- a novel partial reduction algorithm that minimizes data movement in the parallel reduction of sparsely distributed data,
- a theoretical performance analysis of our partial reduction algorithm, and
- an empirical performance benchmark that validates the correctness of our theoretical analysis.

2 Driver Applications

In this section, we describe three example driver applications in large-scale data analysis and visualization.

2.1 Driver application I: Lagrangian flow analysis

Lagrangian flow analyses, which need to trace the trajectories of many massless particles in the flow by numerical integration, have two basic parallelization strategies: task parallelism [12,28,34] and data parallelism [4, 7, 17, 29, 32, 40, 42]. In this study, we focus only on the data parallelism that requires parallel reduction. The data parallelism partitions and distributes the input flow data into parallel processes. Each particle is traced by the process that owns the corresponding partition, and the particle is exchanged to other processes if the particle is out of the local bound of the partition. As a result, the trajectory of each particle will reside in multiple partitions that the particle passes through, and each corresponding process has only incomplete segment(s). Although these intermediate results can be exchanged and aggregated during the particle tracing process, the cost of such a naive and recurrent communication pattern can be prohibitively high when the results are large.

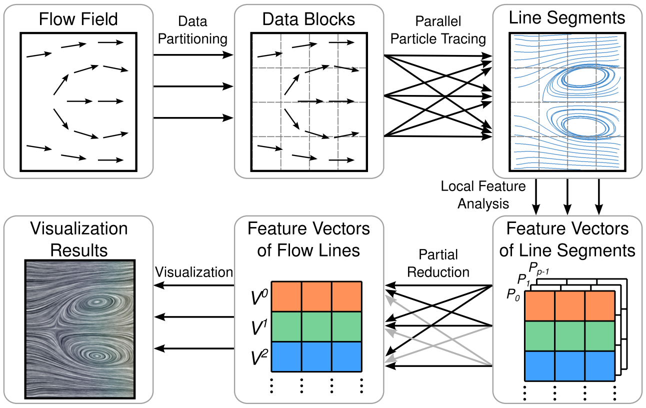


Figure 3: Workflow of data-parallel Lagrangian flow analysis paired with parallel partial reduction.

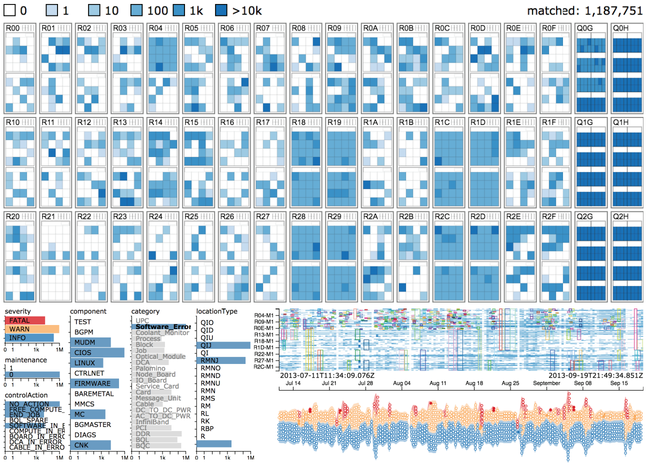


Figure 4: Data cube query results of 55 million RAS log data [11]. From the visualizations we can see that the result space of data cube results is sparsely occupied.

Table 1 lists various Lagrangian-based flow visualization applications that can benefit from our partial reduction methods. For example, the total length of flow lines can be calculated by summing the lengths of flow line segments; the operator for the reduction is “add.” Similarly, streamline statistics (histograms, mean, and variance) are also based on the add operator. The line integral convolutions (LIC) [3] result of a given streamline seed can be computed by summing the convolution values across processes. The reduction of flow line queries, such as predicates [36] and pattern matching [39], is based on the logical “or” operator.

Table 1: Examples of Lagrangian-based flow visualization and analysis methods.

Measurement	Data Type	Operator
Length	Number	Add
Statistics [21]	Histogram	Add
Geometry quantifications [15]	Number	Add/Max/Min
Pattern matching [39]	Boolean	Or
Predicates [36]	Boolean	Or
LIC [3]	Number	Add

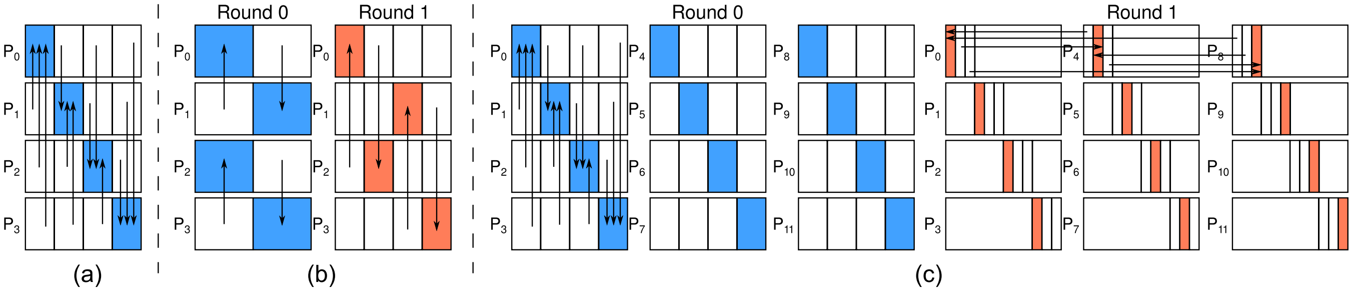


Figure 5: Examples of three parallel reduction algorithms: (a) direct-send [14], (b) binary-swap [23], and (c) radix-k [31]. Blocks represent partitions of the result space, and arrows represent movement of the partitions in reduction. Shaded blocks are destinations of the partitions; different colors represent different rounds in reduction.

Figure 3 illustrates how data-parallel Lagrangian flow analysis is paired with parallel reduction. First, we partition the input data into blocks and assign the blocks with different parallel processes. Second, we trace particles in each block and exchange particles that leave the block. The trajectories of particles—partial flow lines—are not exchanged and thus distributed in processes. Third, we analyze the partial flow lines in every process and store the partial results. Fourth, the partial results across processes are aggregated into complete analysis results by partial reduction. Fifth, the aggregated results are gathered for further processing.

2.2 Driver application II: Data cubes

OLAP data cubes are a fundamental approach to help users understand the spatiotemporal and attribute correlations in multidimensional arrays through interactive visualizations. Figure 4 demonstrates a data cube visualization [11] for 55 million reliability, availability, and serviceability (RAS) logs that are generated over five years by Mira, an IBM Blue Gene/Q supercomputer at Argonne National Laboratory. Each log message has multiple attributes including timestamp, location, severity, components, and category. Data cubes provide statistics on how selected data are projected in each dimension. In Figure 4, a user submits a request for viewing fatal messages within a time scope. The data cube query engine then returns a group of histograms that record how the filtered log messages are distributed in each dimension.

The performance of the data cubes is a critical issue, because interactivity is critical for visual analytics. In relational databases, data cubes are implemented by precomputing the results of every possible combination of multidimensional queries, but both storage and computation costs are prohibitive for large datasets. The visualization community has recently developed lightweight data cube implementations that can run on a single server, such as Nanocubes [19] and imMens [20] for interactive exploration. The performance is boosted by using screen space approximations and hierarchical data structures. Distributed and parallel implementations of data cubes also have been developed in recent years to accelerate the queries [2]. For example, Kylin [1] is a Spark-based distributed data cube engine that runs on big data clusters.

We found that data cubes are difficult to scale on HPC platforms because the bottleneck of parallel reduction. The problem can be alleviated by using partial reduction, because the data cube outputs are usually very sparse. For example, in the 14,400-bin time-varying histogram visualization in Figure 4, many of the slots are empty. Thus one can avoid the unnecessary data movement of zero elements with partial reduction. More details and experimental results are in the following sections.

2.3 Driver application III: Volume rendering

Parallel rendering has three basic approaches: sort-first, sort-middle, and sort-last. Among the three approaches, sort-last parallel volume rendering [25] has been widely used in the visualization community. In sort-last parallel volume rendering, data are first partitioned into blocks and distributed over parallel processes. Then, the parallel processes render their associated data blocks separately, and the intermediate rendering results are composited at the end through image compositing.

Image compositing involves inter processor communication, which is often the bottleneck of sort-last parallel volume rendering. For image compositing, direct-send [14] and binary-swap [22, 23] are the two most commonly used approaches. Various methods have been proposed to improve and combine direct-send and binary-swap, such as 2-3 swap [41] and radix-k [16, 31]. More details of these methods are discussed in the following section.

3 Background: Parallel full reduction

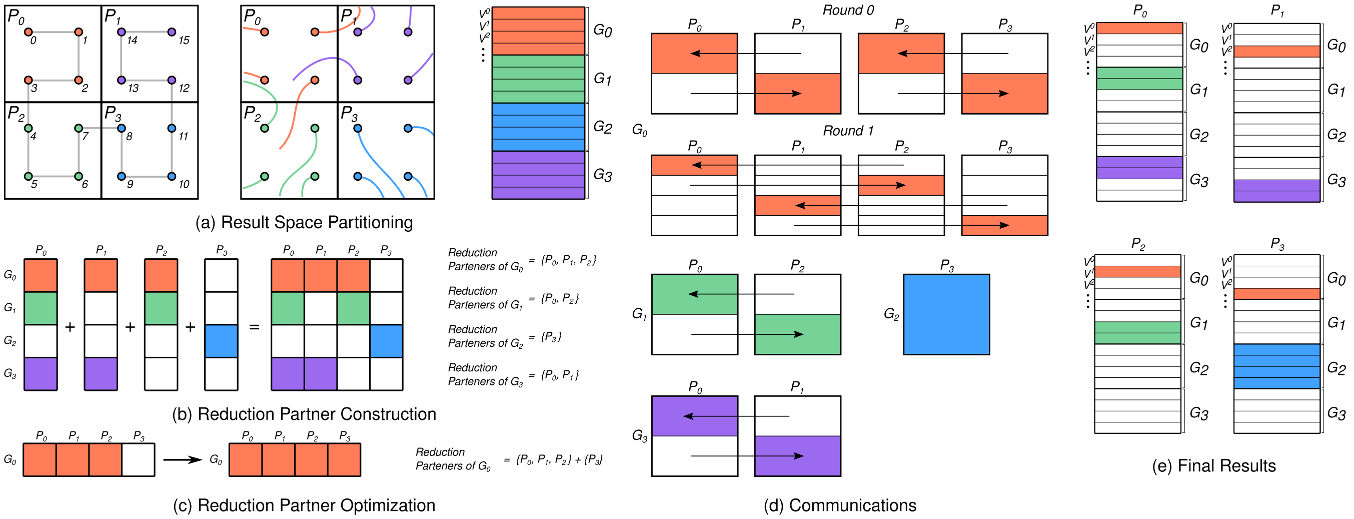
This section formalizes the definitions and reviews related work on parallel full reduction algorithms. The concept of reduction is to reduce a set of values into a smaller set, for example, the summation of an array or the histogram of a scalar field. Parallel reduction, the collective operation that reduces values in distributed memory, is studied as a general topic in the HPC community [18, 35]. Parallel reduction can be formalized as follows:

$$V = V_0 \oplus V_1 \oplus \ldots \oplus V_{n-1}, \tag{1}$$

where n is the number of parallel processes, V is the *feature vector* denoting the output results, $\{V_i\}$ are the feature vectors of intermediate results in the ith process, and $\oplus$ is the operator that composes the intermediate results $\{V_i\}$ into final results V. Each component of V_i is denoted as V_i^j, where j is the index of the query. For example, in the analysis of flow trajectory, V_i^j is the length of streamline segment in the ith process for the jth streamline.

For the purpose of image compositing, two early foundational approaches to reduce dense $\{V_i\}$ into V are direct-send [14] and binary-swap [22, 23], illustrated in Figure 5(a) and (b), respectively. Both methods subdivide the result space (output image) into n parts. The direct-send method sends the partitions to their designated processes directly in one single round. The binary-swap method reduces the data in multiple rounds. A process swaps the partition with the designated process within each round, until all rounds are finished. As documented by various studies [8, 33], direct-send performs better with small numbers of processes but suffers network congestion with larger scales; binary-swap scales better, but the number of processes is restricted to powers of two.

The visualization community has contributed various methods to improve and combine direct-send and binary-swap algorithms. For example, the 2-3 swap method [41] decomposes an arbitrary process number p into pairs and triplets, which supports binary-swap and direct-send, respectively. A multi round reduction similar to binary-swap is then conducted. Radix-k [16, 31] generalizes binary-swap by factoring p into k processes, which perform direct-send per round, as illustrated in Figure 5(c). Both 2-3 swap and radix-k


P_0
P_1
P_2
P_3
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
V^0
V^1
V^2
G0
G1
G2
G3
Round 0
Round 1
G_0
G_1
G_2
G_3
+
=
Reduction Parteners of G_0 = $\{P_0, P_1, P_2\}$
Reduction Parteners of G_1 = $\{P_0, P_2\}$
Reduction Parteners of G_2 = $\{P_3\}$
Reduction Parteners of G_3 = $\{P_0, P_1\}$
Reduction Parteners of G_0 = $\{P_0, P_1, P_2\} + \{P_3\}$

(a) Result Space Partitioning

(b) Reduction Partner Construction

(c) Reduction Partner Optimization

(d) Communications

(e) Final Results

Figure 6: Example of partial reduction for Lagrangian flow analysis. (a) Streamlines starting from different seed locations are partitioned into 4 groups, which are illustrated using different colors. (b) Reduction partners of each group (i.e., processes that contain line segments of the group) are constructed. (c) Reduction partners are optimized to improve the performance of radix-k parallel reduction. (d) Data reduction of each group is performed using radix-k. (e) Final results of partial reduction are outputted.

do not require the power of two processes. Further improvements include optimizing domain partitioning [9, 37], reducing communication by compression [38], using multi- [10] and many-core [24] architectures, and creating production libraries [26].

4 Our Method: Parallel partial reduction

Our partial reduction algorithm has three steps: result space partitioning, reduction partner construction and optimization, and communication. First, the feature vector is partitioned into a number of groups. Second, the reduction partners (a subset of processes) for each group are determined. Third, a full reduction algorithm is applied for each group. Depending on the full reduction algorithm in use, we name our partial reduction algorithms partial direct-send and partial radix-k. For partial radix-k, we also adjust the number of reduction partners by introducing other processes into the group. In the following, we focus on partial radix-k; the partial direct-send is treated as a special case of partial radix-k when k of each group equals the number of processes in the group.

Without loss of generality, we use Lagrangian flow analysis to describe the three steps, as shown in Figure 6. First, the result space is partitioned into groups. In this stage, flow lines that traverse through similar blocks are grouped together. Second, the reduction partners for each group of flow lines are determined and optimized based on the data blocks traversed by each group of flow lines. Third, the radix-k reduction algorithm is applied for each group to reduce flow analysis results.

We compare our approach with two baseline approaches: full direct-send and full radix-k. The full direct-send and full radix-k are the full reduction algorithms that exchange data between all processes. Notice that we do not explicitly compare our method with full binary-swap, which is the special case of full radix-k when k equals to 2.

4.1 Result space partitioning

We partition the result space—feature vector V—into g groups to conduct partial reductions per group. The partition can be arbitrary, but ideally the distribution of values V_i^j in each group should be similar, where j is the index of components in the feature vector V_i.

For example, in Lagrangian flow analysis, we choose to partition the result space based on the Hilbert space-filling curves of seed locations. Each seed can be transformed into a z-index for partitioning. This is based on the fact that flow trajectories starting from vicinity seed locations are close to each other and thus have a better chance to intersect with similar data blocks. The z-index also preserves spatial coherence of the seeds. In the partitioning process, we first label the flow line seeds with the z-index. We then uniformly partition the result space by the indices. For example, in Figure 6(a), 16 flow lines are uniformly partitioned into four groups based on the indices of their seeds. Result space partitioning for other applications is discussed in the following section.

4.2 Reduction partner construction

The *reduction partners* $\mathscr{P}_i$ are defined as a set of processes that exchange data for the ith group. We construct reduction partners for each group, in order to schedule communications within the group.

The reduction partners can be derived from the *partner matrix* $\mathbf{M}$, which is a $g \times p$ matrix of Boolean values. Each value m_{ij} in $\mathbf{M}$ indicates whether the intermediate results in the ith group exist in the jth process. The reduction partner $\mathscr{P}_i$ is thus a set of processes that satisfy $m_{ij} = true$.

The reduction partner construction has two steps. First, we calculate the partner matrix $\mathbf{M}$. For example, in Lagrangian flow analysis, this process is done by checking whether the group of flow lines intersect a block across the process, as shown in Figure 6(b). The partner matrix in different processes can then be gathered and distributed to all processes with the `MPI_Allgather()`. Second, we construct the reduction partners $\mathscr{P}_i$ based on $\mathbf{M}$ for each group.

4.3 Reduction partner optimization

If radix-k is used for the reduction of each group G_i, we also optimize the reduction partners $\mathscr{P}_i$ to improve the performance. As documented by Kendall et al. [16] and Moreland et al. [26], the performance of radix-k highly depends on the number of processes, that is, p_i in our study. However, p_i, which eventually depends on the partner matrix $\mathbf{M}$, could be a prime number or an integer that cannot be factorized into the product of multiple numbers. Such a case will lead to poor performance of radix-k in the group and become the bottleneck of our partial reduction algorithm.

We hence optimize the reduction partners by involving additional processes that are not in $\mathscr{P}_i$, in order to obtain a better value of p_i and thus to boost the performance of our algorithm. The pseudocode of the reduction partner optimization is in Algorithm 1. For

Algorithm 1 Optimization of the reduction partners $\mathscr{P}_0, \dots, \mathscr{P}_{g-1}$, where g is the number of groups. p is the total number of processes; p_i is the number of processes within $\mathscr{P}_i$; t is the maximum number used to factorize p_i.

```
for group id $i$ from 0 to $g-1$ do
    $p_i \leftarrow |\mathscr{P}_i|$
    if $Factor(p_i,t)$ then
        continue
    else
        for $l$ from $p_i$ to $p$ do
            if $Factor(p_i,t)$ then    ▷ $Factor(p_i,t)$ returns true if $p_i$ can be factored into numbers smaller than $t$
                $d \leftarrow l - p_i$
                break
            end if
        end for
        while $d > 0$ do
            $q \leftarrow p/g * i$
            $P = FindProcess(q, \mathscr{P}_i)$    ▷ $FindProcess(q, \mathscr{P}_i)$ returns the first process $P$ which is not inside $\mathbb{P}_i$ starting from $P_q$ to $P_{p-1}$ then from $P_0$ to $P_{q-1}$
            Push $P$ into $\mathscr{P}_i$
            $d \leftarrow d-1$, $p_i \leftarrow p_i + 1$
        end while
    end if
end for
```

example, in Figure 6(c), G_0 has 3 processes in the reduction partners. Assume k is 2, which cannot divide 3 evenly. We then introduce another process into this group; thus p_i equals 4, which can be factored into 2×2 for efficient radix-k for this group.

4.4 Concurrent reduction for all groups

We concurrently use the radix-k algorithm to reduce results for each group G_i based on the reduction partners $\mathscr{P}_i$. The reduction process is illustrated in Figure 6(d). First, we factorize the number of processes p_i into $\prod_{j=0}^{r_i} k_i^j$, where i is the index of group, r_i is the number of factors, and k_i^j is the jth factor. Second, we exchange data between processes in r_i rounds based on the radix-k algorithm.

Notice that the per group reductions are conducted simultaneously over all processes. We synchronize after every round of radix-k in each group, so we have $\max r_i, 0 \le i \le g-1$, rounds in total. In each round, messages that have the same destination are aggregated to reduce the number of messages.

4.5 Theoretical costs

Table 2 compares the theoretical costs of our method with three baseline algorithms in latency, bandwidth, and computation. Notice that the theoretical model in this section does not include the reduction partner construction.

We follow the model used by Chan et al. [6], Cavin and Demengeon [5], and Peterka et al. [31] to model the theoretical costs of our algorithm. First, there are p processes in a distributed memory parallel architecture with a fully connected network. Processes send and receive messages simultaneously with non-overlapping communication and computation and without network contention. Second, there are n data items that need to be reduced in each process. Third, the communication cost of sending one message between two nodes is $\alpha + n\beta$, where α is the latency of one message, and β is the transmission time per data item (bandwidth). Fourth, the computation time γ is the cost of reducing one single data item.

The three baseline approaches—direct-send, binary-swap, and radix-k—have fixed cost of latency, bandwidth, and computation. For the latency, each process in the direct-send algorithm sends and receives $p-1$ messages; hence the latency is $\alpha(p-1)$. The binary-swap algorithm has $\log_2(p)$ rounds, and each process sends and receives only one message; thus the latency is $\alpha \log_2(p)$. The latency of radix-k is between the binary-swap and direct-send methods, depending on the k value. The bandwidth of the three full reduction methods is bounded by $n\beta\frac{p-1}{p}$, and the computation cost of the three full reduction methods is bounded by $n\gamma\frac{p-1}{p}$.

Table 2: Comparison of our partial reduction algorithm with three full reduction algorithms in theoretical latency, badwidth, and computation costs.

Algorithm	Latency	Bandwidth	Computation
Direct-send	$\alpha(p-1)$	$n\beta\frac{p-1}{p}$	$n\gamma\frac{p-1}{p}$
Binary-swap	$\alpha\log_2(p)$	$n\beta\frac{p-1}{p}$	$n\gamma\frac{p-1}{p}$
Radix-k	$\alpha\sum_{i=0}^{r-1}(k_i-1)$	$n\beta\frac{p-1}{p}$	$n\gamma\frac{p-1}{p}$
Our method	$0 \sim r\alpha(p-1)$	$n\beta\sum_{i=0}^{g-1}\frac{p_i-1}{p_i g}$	$n\gamma\sum_{i=0}^{g-1}\frac{p_i-1}{p_i g}$

We derive the theoretical costs of our algorithm, which depend on the reduction partners. We assume that each group has n/g components of the feature vector V and has r rounds in total.

The latency of our algorithm method is bounded by 0 and $r\alpha(p-1)$. In the best case that the number of processes in a group is 0 or 1, the latency is 0 because there is no communication. In the worst case, if a process needs to exchange messages with all other processes, the latency is $r\alpha(p-1)$.

The bandwidth and computation costs of our algorithm are less than the costs of the full reduction algorithms. For each group, the bandwidth and computation costs of the partial reduction method are $n\beta\frac{p_i-1}{p_i g}$ and $n\gamma\frac{p_i-1}{p_i g}$, respectively. By summing up the costs for all groups, the total bandwidth is

$$n\beta\sum_{i=0}^{g-1}\frac{p_i-1}{p_i g}, \tag{2}$$

and the computation cost is

$$n\gamma\sum_{i=0}^{g-1}\frac{p_i-1}{p_i g}. \tag{3}$$

We can see that the coefficients in bandwidth and computation costs are $\sum_{i=0}^{g-1}\frac{p_i-1}{p_i g}$ and $\frac{p-1}{p}$ in our method and the full reduction algorithms, respectively. Because p_i is always less than or equal to p, we have $\frac{p_i-1}{p_i} \le \frac{p-1}{p}$ and obviously $\sum_{i=0}^{g-1}\frac{p_i-1}{p_i g} \le \frac{p-1}{p}$.

Aside from communication and computation costs, our algorithm has a fixed cost in reduction partner construction and optimization. In the following section, we will study the overhead by experiments.

5 Performance Benchmark

We benchmark our method on Mira, an IBM Blue Gene/Q supercomputer at Argonne National Laboratory. The supercomputer has 49,152 nodes, each of which has 16 PowerPC A2 cores working at 1600 MHz and sharing 16 GB of RAM. The nodes are interconnected with a 5D torus network. We use up to 8,192 processes (4 processes per node) in our experiments.

We present four groups of experiments results: synthetic data, Lagrangian flow analyses, data cubes, and volume rendering. In the experiments, we compare the partial direct-send and partial radix-k with two baseline approaches: the full direct-send and full radix-k methods. The k value is set to 4 for the full radix-k method, because it performs the best among different k values in the experiments. The implementations of both the partial reduction and the applications are based on C++, MPI, and the DIY2 library [27].

5.1 Synthetic experiments

We study the performance of the partial reduction methods and evaluate under what conditions they perform better than the baseline methods by sweeping four parameters: number of processes p,

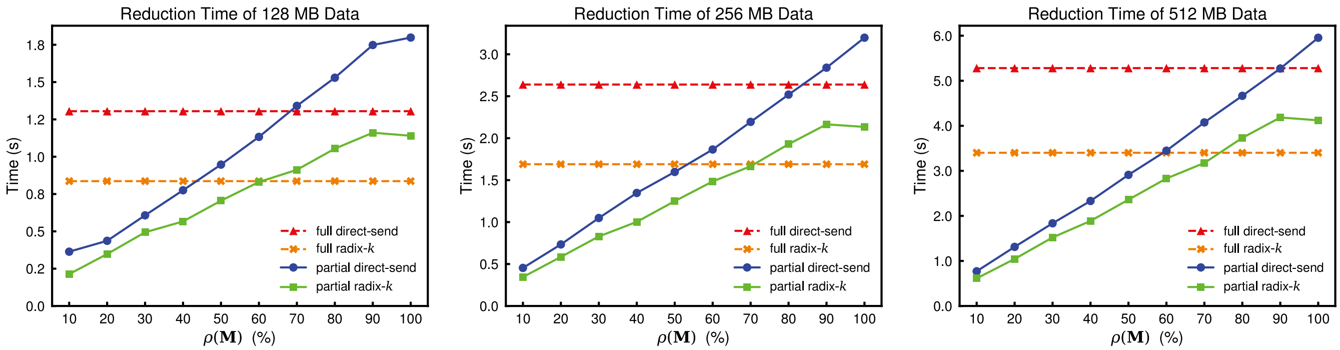


Figure 7: Reduction time for different partner densities ρ and different sizes of feature vectors. Partner densities ρ are from 10% to 100%, and the step size is 10%. Size of the feature vectors are 128 MB, 256 MB, and 512 MB. The number of processes p and the number of groups g are 2,048 and 512, respectively.

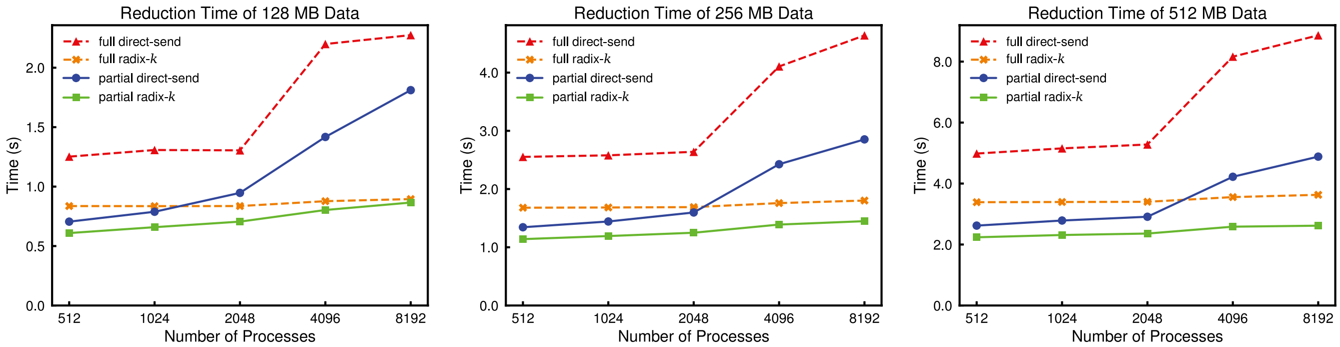


Figure 8: Reduction time for different numbers of processes and different size of feature vectors. The number of processes is from 512 to 8,192. Size of the feature vectors are 128 MB, 256 MB, and 512 MB. Number of groups is fixed to 512. Partner density is fixed to 50%.

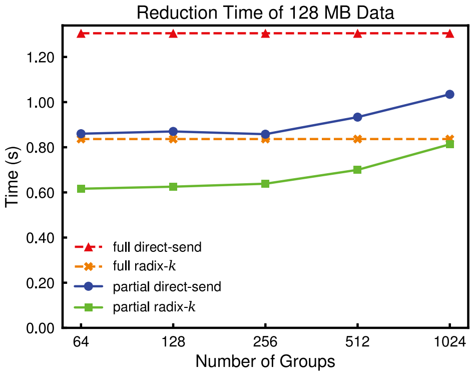


Figure 9: Reduction time for different numbers of groups, from 64 to 1,024. Number of processes is fixed to 2,048. Partner density is 50%. Size of the feature vector is 128 MB.

number of groups g, the storage size of the feature vector s, and the *partner density* $\rho(\mathbf{M})$. The partner density is the average number of processes per group over the total number of processes, or the density of the partner matrix $\mathbf{M}$:

$$\rho(\mathbf{M}) = \frac{\sum_{i,j} m_{ij}}{gp}. \tag{4}$$

Basically, $\rho(\mathbf{M})$ indicates the average number of reduction partners. In extreme cases when ρ equals 100%, the complexity of our algorithm is identical to the cost of the full reduction.

In the synthetic experiments, we randomly generate partner matrices $\mathbf{M}$, in order to sweep arbitrary ρ for performance analysis. Specifically, we increase ρ from 10% to 100% in steps of 10%. For other parameters, p is changing from 512 to 8,192, g is from 128 to 1,024, and s is either 128 MB, 256 MB, or 512 MB. We use the add operator to reduce the arbitrarily generated feature vector V.

Figure 7 shows timings of the full direct-send, full radix-*k*, partial direct-send, and partial radix-*k* with different partner density ρ and feature size s. The number of processes p and the number of groups g are 2,048 and 512, respectively. We can see that when ρ increases, the reduction time of the partial direct-send and partial radix-*k* increases for all three data sizes s, which is because more data need to be exchanged when ρ increases. The timings of the full direct-send and full radix-*k* in each sub figure remain constant because they are independent of ρ, and the full radix-*k* always performs better than the full direct-send. The reduction time for the partial direct-send performs better than the full radix-*k* when ρ is less than 40%, 50%, and 50% for s equal to 128 MB, 256 MB, or 512 MB, respectively. The partial radix-*k* always performs better than the full and partial direct-send for all three data sizes and performs better than the full radix-*k* when ρ is less than 60%, 70%, and 70% for s equals to 128 MB, 256 MB, or 512 MB, respectively. When ρ approaches 100%, partial reduction methods exchange a similar amount of data compared with the amount of the full reduction methods and take additional time to construct reduction partners. Hence, partial reduction methods take more time than do full reduction methods when ρ approaches 100%. The reduction time for the partial radix-*k* decreases when ρ changes from 90% to 100%. The reason that when ρ equals 100%, each group has all processes involved for reduction. As a result, each process will communicate with the same set of processes for all the group in each round, which reduces the number of messages in the partial radix-*k*.

Figure 8 compares the reduction time for the four reduction meth-

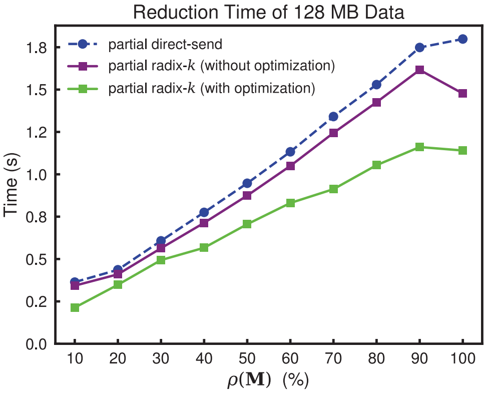


Figure 10: Reduction time for the partial direct-send, partial radix-k without optimization, and partial radix-k with optimization. Number of processes and number of groups is fixed to 2,048 and 512, respectively. Partner density increase from 10% to 100% and the step size is 10%. Size of the feature vector is 128 MB.

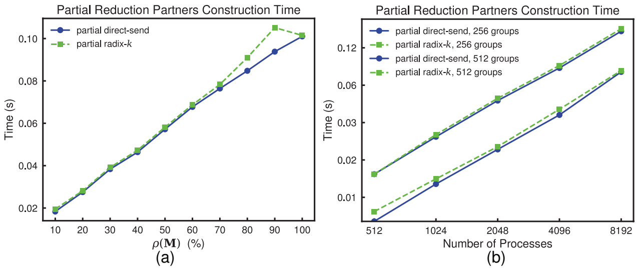


Figure 11: (a) Time to construct reduction partners for different partner density ρ from 10% to 100% for every 10%. The number of processes, the number of groups, and the feature size are fixed to 2,048, 512, and 256 MB, respectively. (b) Time to construct reduction partners for different numbers of processes and different numbers of groups. Number of processes are from 512 to 8,192 and number of groups are 256 and 512. Partner density is fixed to 50%.

ods with different numbers of processes from 512 to 8,192 and three feature sizes: 128 MB, 256 MB, and 512 MB. The number of groups is fixed to 512. Partner density ρ is fixed to 50%. We can see that the partial direct-send always performs better than the full direct-send and performs better than the full radix-k when p is less than 1,024, 2,048, and 2,048 for s equal to 128 MB, 256 MB, or 512 MB, respectively. The partial radix-k performs the best among all four methods for all three feature sizes. The partial radix-k scales as well as the full radix-k when the feature size is 256 MB and 512 MB. When the feature size is 128 MB, the partial radix-k increases faster than the full radix-k as the number of processes increases, because the proportion of the reduction partners construction time is high when the feature size is small and the reduction partners construction time increases when the number of processes increases.

Figure 9 compares the reduction time for the four reduction methods with different numbers of groups from 64 to 1,024. The number of processes, the feature size, and the partner density ρ is fixed to 2,048, 128 MB, and 50%, respectively. We can see that the partial radix-k performs the best among all four methods for all three feature sizes. The reduction time for the partial direct-send and the partial radix-k increases when the number of groups increases, because the reduction partner construction time increases when the number of groups increases.

Figure 10 compares the reduction time for the partial radix-k with and without reduction partner optimization. The number of processes, the number of groups, and the feature size are fixed to 2,048, 512, and 128 MB, respectively. Partner density ρ ranges

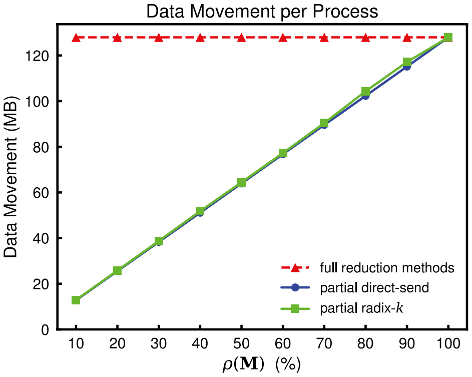


Figure 12: Data movement per process for different partner density ρ from 10% to 100% for every 10%. The number of processes, the number of groups, and the feature size are fixed to 2,048, 512, and 128 MB, respectively.

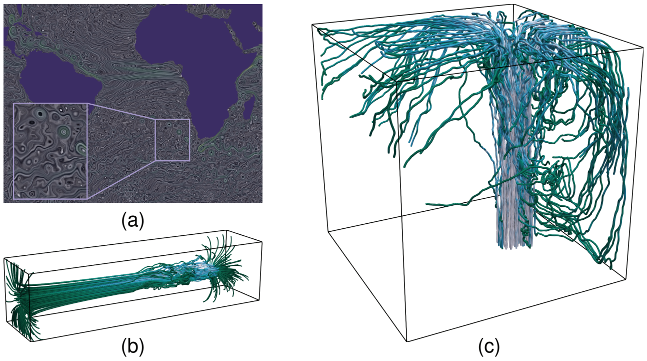


Figure 13: Flow analyses results of the Ocean, Plume, and Nek datasets. (a) LIC visualization of the Ocean dataset. (b) Streamlines with similar curvature histograms for the Plume dataset. (c) Streamlines with maximum curvature greater than 0.02 for the Nek dataset.

from 10% to 100%, in steps of 10%. In comparison, the partial radix-k performs better with reduction partner optimization.

Besides the overall reduction time, we further analyzed the overhead of the partial reduction methods, which is the time for constructing reduction partners, as shown in Figure 11. Figure 11(a) shows the time for constructing reduction partners for the partial reduction methods with different ρ. The number of processes, the number of groups, and the feature size are fixed to 2,048, 512, and 256 MB, respectively. By comparison, the time used for constructing reduction partners in the partial radix-k is slightly higher than for the partial direct-send. This is because, in addition to constructing the reduction partners, the partial radix-k optimizes the reduction partners as well. For both methods, reduction partner construction time increases when ρ increases, because when ρ increases, each process needs to construct reduction partners for more groups. Figure 11(b) shows the reduction partner construction time with different numbers of processes p and groups g. The data size is fixed to 128 MB. Partner density is fixed to 50%. We can see that as the number of processes and the number groups increase, the time for constructing reduction partners increases. As the size of data increases, however the reduction partner construction time is still negligible even with larger numbers of processes.

We also compared the average data movement per process for the four reduction methods with different ρ, as shown in Figure 12. The number of processes, the number of groups, and the feature size are fixed to 2,048, 512, and 128 MB, respectively. We can see that the data movement of the partial reduction methods is smaller

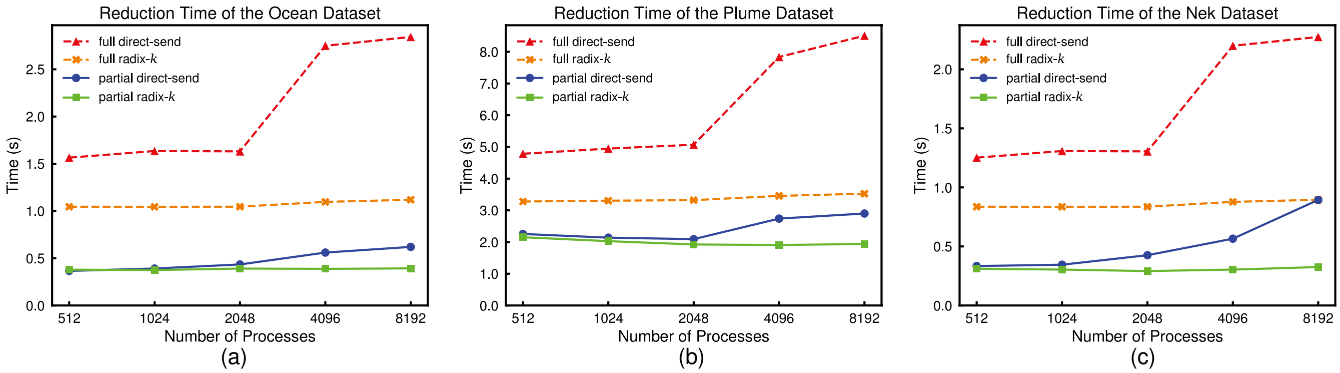


Figure 14: Reduction time of the three flow datasets with respect to different numbers of processes from 512 to 8,192. Number of groups is fixed to 512.

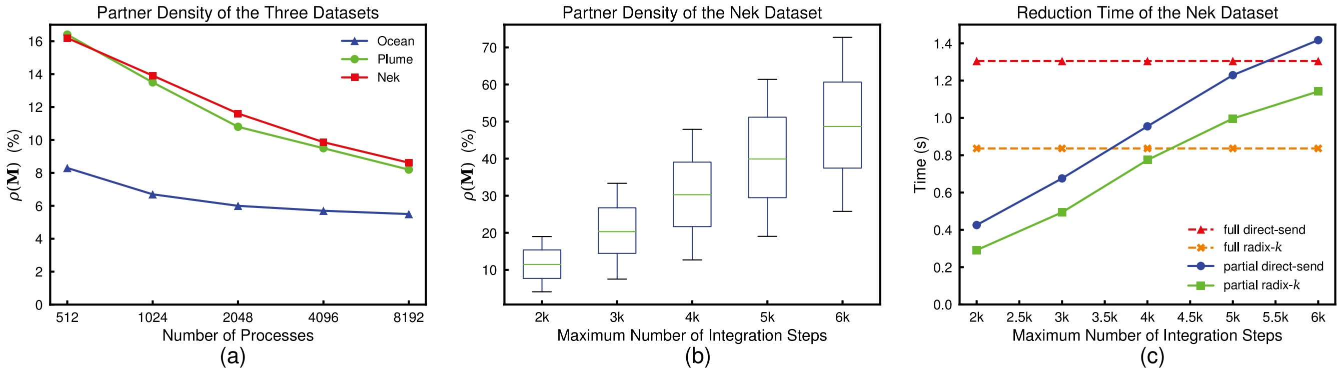


Figure 15: Performance benchmarks of Lagrangian flow analyses. (a) Partner density ρ of the three datasets. (b) Number of processes in each group for the Nek dataset with different numbers of maximum integration steps. Number of processes and number of groups are fixed to 2,048 and 512, respectively. (c) Reduction time of the Nek dataset with different numbers of maximum integration steps. Number of processes and number of groups are fixed to 2,048 and 512, respectively.

than that of the full reduction methods by taking the advantage of the data sparsity. When ρ increases, the data movement for the partial reduction methods. The data movement of the partial radix-*k* is slightly higher than for the partial direct-send because unrelated processes are introduced to certain groups when the partial radix-*k* performs reduction partner optimization.

5.2 Lagrangian flow analyses

Table 3: Data specifications and Lagrangian flow analysis method applied on each dataset.

Data	Feature Size (per process)	Measurements	Operator
Ocean	164.8 MB	LIC	Add
Plume	496 MB	Statistics	Add
Nek	128 MB	Predicates	Or

We demonstrate the efficiency of the partial reduction methods through experiments on three flow datasets with different applications: the Ocean dataset, the Plume dataset, and the Nek dataset. The datasets are listed in Table 3. The Ocean dataset is the output from an eddy simulation. $1800 \times 1200 \times 20$ seeds are sampled regularly for every 2 grid points on each dimension, and streamlines are generated with maximum 2,000 integration steps and a step size of 0.2. LIC is generated based on the resulting streamlines. The total size of the LIC result is 164.8 MB. The Plume dataset is generated from a simulation of solar plume on the surface of the sun. For this application $63 \times 63 \times 256$ seeds are sampled regularly for every 8 grid points on each dimension, and streamlines are generated with maximum 2,000 integration steps and a step size 2.0. A histogram of 128 bins is computed for the curvatures of each streamline. The total size of the histograms is 496 MB. The Nek dataset is generated by a thermal hydraulics simulation. For this application, $256 \times 256 \times 256$ seeds are sampled regularly for every 8 grid points on each dimension, and streamlines are generated with a step size 0.04. We increase the maximum number of integration steps from 2,000 to 6,000. Two predicates are selected to test whether the maximum curvature of each streamline is greater than two different thresholds. To be consistent with other flow line analysis results, we use 4-byte floating-point values to represent Boolean results, where 1.0 means true and 0.0 means false. The total size of the analysis results is 128 MB. The Runge-Kutta 4th-order method is used to trace particles for all three datasets. Figure 13 shows the flow analysis results of these three datasets.

Figure 14 shows the timings of the four reduction methods with different numbers of processes from 512 to 8,192 on the three different datasets. The maximum number of integration steps and the number of groups are fixed to 2,000 and 512, respectively. The partial reduction methods outperform the full reduction methods for all three datasets, and the partial radix-*k* method performs the best among all four methods. Figure 15(a) shows the partner density ρ of the three datasets with different numbers of processes. We can see that the partner density ρ is less than 17% for all three datasets and decreases when the total number of processes increases. Figure 15(b) shows the median and variance for the number of processes in each group for the Nek dataset with streamlines of different maximum integration steps. The number of processes and the number of groups are fixed to 2,048 and 512, respectively. We can see that when the number of integration steps increases, the median and variance of the number of processes in each group increase as

Table 4: Parallel reduction benchmark in a data cube query of 55 million RAS logs with respect to different numbers of processes p. d_f and d_p is the amount of data movement in the full and partial radix-k methods, respectively; t_f and t_p is the execution time of the full and partial radix-k methods, respectively, in milliseconds. $(d_f - d_p)/d_f$ denotes the percentage of reduced amount of data movement with the partial radix-k method.

p	$\rho(\mathbf{M})$	$(d_f - d_p)/d_f$	t_f	t_p
64	4.64%	96.88%	18.11	3.46
128	3.89%	96.85%	16.50	3.22
256	3.50%	96.85%	16.70	4.60
512	3.31%	96.80%	17.31	5.52
1,024	3.22%	96.82%	19.38	7.72

well. As a result, the timings of the partial reduction methods increase when the maximum number of integration steps increase, as shown in Figure 15(c). Based on this figure, when the maximum number of integration steps is greater than 3,500, the partial reduction methods take more time than the full radix-k for reduction does.

5.3 Data cubes

In the data cube application, we study the performance of the full and partial radix-k methods for a data cube query of the large RAS log dataset with 55 million RAS logs. The query result contains 755,424 histogram bins, and the size equals 2.882 MB. For the partial radix-k method, the resulting space is partitioned into 32 equal sized groups based on the date and time of the RAS logs. For each group, its corresponding reduction partners are the processes whose data blocks contain RAS logs within the time interval of the group.

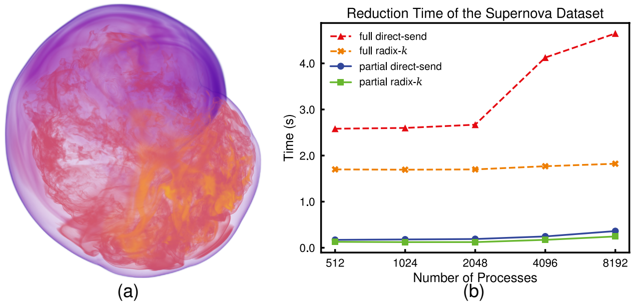
Table 4 shows the benchmark of the full and partial radix-k methods of the data cube application with respect to different numbers of processes from 64 to 1,024. From the table we can see that the partner matrices are sparse in all configurations; thus partial reduction is very efficient for this problem. We can also see that the partial radix-k method outperforms the full radix-k method with all the different numbers of processes. The partial radix-k method also avoids more than 96% of the data movement, compared with the data movement of the full radix-k method.

5.4 Volume rendering

We studied the performance of the partial reduction methods for parallel volume rendering on a scalar dataset that is from a supernova core collapse simulation. The resolution of the output image is set to 4096 × 4096, which has a total size of 256 MB. The rendering result of the supernova dataset is shown in Figure 16(a).

Figure 16(b) compares the timings of the four reduction methods with different numbers of processes from 512 to 8,192. For the partial reduction methods, the result space is partitioned into 16 × 16 equal sized groups. The reduction partners of each group are defined as the processes whose associated blocks are intersected with the group when projected on the image space. As shown in Figure 16(b), the partial reduction methods outperform the full reduction methods with all the different numbers of processes, and the partial radix-k performs slightly better than the partial direct-send.

Table 5 shows the partner density ρ and the percentage of data movement that the partial radix-k reduced. We can see that the partner density ρ equals 3.11% with 512 processes and keeps decreasing as the number of processes increases. We can also see that the partial radix-k avoids more than 97% of the data movement, compared with the data movement of the full reduction methods.



(a)
(b)

Figure 16: Results of the volume rendering application. (a) Rendering result of the supernova dataset. (b) Reduction time of the four reduction methods with respect to different numbers of processes p.

Table 5: Analysis results of partner density ρ and data movement of the volume rendering application with respect to different numbers of processes p; d_f and d_p are the amount of data movement in full and partial reduction, respectively. Here, $(d_f - d_p)/d_f$ denotes the percentage of reduced amount of data movement with the partial reduction.

p	512	1,024	2,048	4,096	8,192
$\rho(\mathbf{M})$	3.11%	2.21%	1.58%	1.44%	1.05%
$(d_f - d_p)/d_f$	97.27%	98.17%	98.80%	98.94%	99.33%

6 Discussion

We discuss the scenarios where our partial reduction algorithm could benefit data-parallel analysis and visualization algorithms. We notice that in two situations in our benchmarks, our partial reduction algorithm cannot outperform full reduction algorithms: larger ρ and smaller s. First, our method could run slower than full reductions if the partner density ρ is large. In this case, we cannot significantly reduce the data movement cost. Second, our partial reduction does not pay off if the feature size s is too small. The main reason is due to the overhead time to determine reduction partners in our algorithm.

However, our partial reduction method outperforms full reduction methods in many real-world data analysis and visualization applications. We can adaptively alternate full/partial reduction according to ρ. Our end-to-end performance benchmark provides guidelines for choosing optimal algorithms for better performance. Moreover, we foresee that the feature size s will grow in the future, as the complexities of data analysis and visualization applications increase.

7 Conclusions and Future Work

In this work, we present a new partial reduction algorithm to aggregate sparse intermediate analysis results distributed in parallel processes. The purpose of partial reduction is to exchange only necessary data in the collective communication. Our algorithm schedules and optimizes communication patterns for efficient communication. Both theoretical analysis and experiments show that our method outperforms the traditional parallel reduction algorithms when the partner matrix is sparse enough.

We would like to explore several directions in the future. First, we would like to apply our method to even more data analysis and visualization algorithms. Second, we are going to study the load balancing issues of the partial reduction algorithm. Third, we would also like to reduce the synchronizations between each reduction round to further improve the performance.

Acknowledgments

This work is supported by the U.S. Department of Energy, Office of Advanced Scientific Computing Research, Scientific Discovery through Advanced Computing (SciDAC) program. This material is also based upon work supported by the U.S. Department of Energy, Office of Science, under contract number DE-AC02-06CH11357. This research used resources of the Argonne Leadership Computing Facility, which is a DOE Office of Science User Facility supported under Contract DE-AC02-06CH11357.

References

[1] *Apache Kylin: Extreme OLAP Engine for Big Data*, 2015 (accessed March 19, 2018). http://kylin.apache.org/.

[2] M. Budiu, R. Isaacs, D. Murray, G. Plotkin, P. Barham, S. Al-Kiswany, Y. Boshmaf, Q. Luo, and A. Andoni. Interacting with large distributed datasets using sketch. In *EGPGV'16: Proc. Eurographics Symposium on Parallel Graphics and Visualization*, pp. 31–43.

[3] B. Cabral and L. C. Leedom. Imaging vector fields using line integral convolution. In *SIGGRAPH'93: Proc. ACM Annual Conference on Computer Graphics*, pp. 263–270, 1993.

[4] D. Camp, C. Garth, H. Childs, D. Pugmire, and K. I. Joy. Streamline integration using MPI-hybrid parallelism on a large multicore architecture. *IEEE Trans. Vis. Comput. Graph.*, 17(11):1702–1713, 2011.

[5] X. Cavin and O. Demengeon. Shift-based parallel image compositing on infiniband fat-trees. In *EGPGV'12: Proc. Eurographics Symposium on Parallel Graphics and Visualization*, pp. 129–138, 2012.

[6] E. Chan, M. Heimlich, A. Purkayastha, and R. A. van de Geijn. Collective communication: theory, practice, and experience. *Concurrency and Computation: Practice and Experience*, 19(13):1749–1783, 2007.

[7] L. Chen and I. Fujishiro. Optimizing parallel performance of streamline visualization for large distributed flow datasets. In *Proceedings of IEEE Pacific Visualization Symposium 2008*, pp. 87–94, 2008.

[8] S. Eilemann and R. Pajarola. Direct send compositing for parallel sort-last rendering. In *EGPGV'07: Proc. Eurographics Conference on Parallel Graphics and Visualization*, pp. 29–36, 2007.

[9] A. Garcia and H.-W. Shen. An interleaved parallel volume renderer with pc-clusters. In *EGPGV'02: Proc. Eurographics Parallel Graphics and Visualization Symposium*, pp. 51–59, 2002.

[10] A. V. P. Grosset, M. Prasad, C. Christensen, A. Knoll, and C. D. Hansen. TOD-tree: Task-overlapped direct send tree image compositing for hybrid MPI parallelism. In *EGPGV'15: Proc. Eurographics Symposium on Parallel Graphics and Visualization*, pp. 67–76, 2015.

[11] H. Guo, S. Di, R. Gupta, T. Peterka, and F. Cappello. La VALSE: Scalable log visualization for fault characterization in supercomputers. In *EGPGV'18: Proc. EuroGraphics Symposium on Parallel Graphics and Visualization*, pp. 91–100, 2018.

[12] H. Guo, J. Zhang, R. Liu, L. Liu, X. Yuan, J. Huang, X. Meng, and J. Pan. Advection-based sparse data management for visualizing unsteady flow. *IEEE Trans. Vis. Comput. Graph.*, 20(12):2555–2564, 2014.

[13] Y. Guo, C. J. Archer, M. Blocksome, S. Parker, W. Bland, K. Raffenetti, and P. Balaji. Memory compression techniques for network address management in MPI. In *IPDPS'17: Proc. IEEE International Parallel and Distributed Processing Symposium*, pp. 1008–1017, 2017.

[14] W. M. Hsu. Segmented ray casting for data parallel volume rendering. In *PRS '93: Proc. ACM Symposium on Parallel Rendering*, 1993.

[15] W. Kendall, J. Huang, and T. Peterka. Geometric quantification of features in large flow fields. *IEEE Computer Graphics and Applications*, 32(4):46–54, 2012.

[16] W. Kendall, T. Peterka, J. Huang, H.-W. Shen, and R. B. Ross. Accelerating and benchmarking radix-k image compositing at large scale. In *EGPGV'10: Proc. Eurographics Symposium on Parallel Graphics and Visualization*, pp. 101–110, 2010.

[17] W. Kendall, J. Wang, M. Allen, T. Peterka, J. Huang, and D. Erickson. Simplified parallel domain traversal. In *SC'11: Proc. ACM/IEEE Conference on Supercomputing*, pp. 1–10, article 10, 2011.

[18] S. Kumar, G. Dózsa, J. Berg, B. Cernohous, D. Miller, J. Ratterman, B. E. Smith, and P. Heidelberger. Architecture of the component collective messaging interface. In *Proc. European PVM/MPI Users' Group Meeting*, pp. 23–32, 2008.

[19] L. D. Lins, J. T. Klosowski, and C. E. Scheidegger. Nanocubes for real-time exploration of spatiotemporal datasets. *IEEE Trans. Vis. Comput. Graph.*, 19(12):2456–2465, 2013.

[20] Z. Liu, B. Jiang, and J. Heer. imMens: Real-time visual querying of big data. *Comput. Graph. Forum*, 32(3):421–430, 2013.

[21] K. Lu, A. Chaudhuri, T.-Y. Lee, H.-W. Shen, and P. C. Wong. Exploring vector fields with distribution-based streamline analysis. In *PacificVis'13: Proc. IEEE Pacific Visualization Symposium*, pp. 257–264, 2013.

[22] K. L. Ma, J. S. Painter, C. D. Hansen, and M. F. Krogh. A data distributed, parallel algorithm for ray-traced volume rendering. In *PRS'93: Proc. ACM Symposium on Parallel Rendering*, pp. 15–22, 1993.

[23] K.-L. Ma, J. S. Painter, C. D. Hansen, and M. F. Krogh. Parallel volume rendering using binary-swap compositing. *IEEE Comput. Graph. Appl.*, 14(4):59–68, 1994.

[24] S. Marchesin, C. Mongenet, and J.-M. Dischler. Multi-GPU sort-last volume visualization. In *EGPGV'08: Proc. Eurographics Symposium on Parallel Graphics and Visualization*, pp. 1–8, 2008.

[25] S. Molnar, M. Cox, D. Ellsworth, and H. Fuchs. A sorting classification of parallel rendering. *IEEE Comput. Graph. Appl.*, 14(4):23–32, 1994.

[26] K. Moreland, W. Kendall, T. Peterka, and J. Huang. An image compositing solution at scale. In *SC'11: Proc. ACM/IEEE Conference on Supercomputing*, pp. 1–11, article 25, 2011.

[27] D. Morozov and T. Peterka. Block-parallel data analysis with DIY2. In *LDAV'16: Proc. IEEE Symposium on Large Data Analysis and Visualization*, pp. 29–36, 2016.

[28] C. Mueller, D. Camp, B. Hentschel, and C. Garth. Distributed parallel particle advection using work requesting. In *LDAV'13: Proc. IEEE Symposium on Large Data Analysis and Visualization*, pp. 109–112, 2013.

[29] B. Nouanesengsy, T.-Y. Lee, and H.-W. Shen. Load-balanced parallel streamline generation on large scale vector fields. *IEEE Trans. Vis. Comput. Graph.*, 17(12):1785–1794, 2011.

[30] T. Peterka, H. Croubois, N. Li, E. Rangel, and F. Cappello. Self-adaptive density estimation of particle data. *SIAM J. Scientific Computing*, 38(5), 2016.

[31] T. Peterka, D. Goodell, R. B. Ross, H.-W. Shen, and R. Thakur. A configurable algorithm for parallel image-compositing applications. In *SC'09: Proc. ACM/IEEE Conference on Supercomputing*, pp. 1–10, article 4, 2009.

[32] T. Peterka, R. B. Ross, B. Nouanesengsy, T.-Y. Lee, H.-W. Shen, W. Kendall, and J. Huang. A study of parallel particle tracing for steady-state and time-varying flow fields. In *IPDPS'11: Proceedings of IEEE International Symposium on Parallel and Distributed Processing*, pp. 580–591, 2011.

[33] T. Peterka, H. Yu, R. B. Ross, K.-L. Ma, and R. Latham. End-to-end study of parallel volume rendering on the IBM Blue Gene/P. In *ICPP'09: Proc. International Conference on Parallel Processing*, pp. 566–573, 2009.

[34] D. Pugmire, H. Childs, C. Garth, S. Ahern, and G. H. Weber. Scalable computation of streamlines on very large datasets. In *SC'09: Proceedings of the ACM/IEEE Conference on Supercomputing*, pp. 16:1–16:12, 2009.

[35] P. Sack and W. Gropp. Faster topology-aware collective algorithms through non-minimal communication. In *SIGPLAN'12: Proc. ACM Symposium on Principles and Practice of Parallel Programming*, pp. 45–54, 2012.

[36] T. Salzbrunn and G. Scheuermann. Streamline predicates. *IEEE Trans. Vis. Comput. Graph.*, 12(6):1601–1612, 2006.

[37] A. Stompel, K.-L. Ma, E. B. Lum, J. P. Ahrens, and J. Patchett. SLIC: Scheduled linear image compositing for parallel volume rendering. In *Proc. IEEE Symposium on Parallel and Large-Data Visualization and Graphics*, pp. 33–40, 2003.

[38] M. Strengert, M. Magallón, D. Weiskopf, S. Guthe, and T. Ertl. Hierarchical visualization and compression of large volume datasets using GPU clusters. In *EGPGV'04: Proc. Eurographics Conference on Parallel Graphics and Visualization*, pp. 41–48, 2004.

[39] J. Tao, C. Wang, and C. Shene. FlowString: Partial streamline matching using shape invariant similarity measure for exploratory flow visualization. In *Proc. IEEE Pacific Visualization Symposium 2014*, pp. 9–16, 2014.

[40] H. Yu, C. Wang, and K.-L. Ma. Parallel hierarchical visualization of large time-varying 3D vector fields. In *SC'07: Proceedings of the ACM/IEEE Conference on Supercomputing*, pp. 24:1–24:12, 2007.

[41] H. Yu, C. Wang, and K.-L. Ma. Massively parallel volume rendering

using 2-3 swap image compositing. In *SC'08: Proc. ACM/IEEE Conference on Supercomputing*, pp. 1–11, article 48. IEEE/ACM, 2008.

[42] J. Zhang, H. Guo, F. Hong, X. Yuan, and T. Peterka. Dynamic load balancing based on constrained K-D tree decomposition for parallel particle tracing. *IEEE Trans. Vis. Comput. Graph.*, 24(1):954–963, 2018.