Received: 20 June 2024 | Accepted: 19 August 2024

DOI: 10.1002/cpe.8274

SPECIAL ISSUE PAPER

WILEY

A distributed memory parallel randomized Kaczmarz for sparse system of equations

Ercan Selçuk Bölükbaşı[1] | Fahreddin Şükrü Torun[2] | Murat Manguoğlu[1]

[1]Computer Engineering Department, Middle East Technical University, Ankara, Türkiye

[2]Computer Engineering Department, Ankara Yıldırım Beyazıt University, Ankara, Türkiye

Correspondence
Ercan Selçuk Bölükbaşı, Computer Engineering Department, Middle East Technical University, Ankara, Türkiye.
Email: ercan@ceng.metu.edu.tr

Abstract

Kaczmarz algorithm is an iterative projection method for solving system of linear equations that arise in science and engineering problems in various application domains. In addition to classical Kaczmarz, there are randomized and parallel variants. The main challenge of the parallel implementation is the dependency of each Kaczmarz iteration on its predecessor. Because of this dependency, frequent communication is required which results in a substantial overhead. In this study, a new distributed parallel method that reduces the communication overhead is proposed. The proposed method partitions the problem so that the Kaczmarz iterations on different blocks are less dependent. A frequency parameter is introduced to see the effect of communication frequency on the performance. The communication overhead is also decreased by allowing communication between processes only if they have shared non-zero columns. The experiments are performed using problems from various domains to compare the effects of different partitioning methods on the communication overhead and performance. Finally, parallel speedups of the proposed method on larger problems are presented.

KEYWORDS

distributed memory, iterative methods, Kaczmarz, parallel computing, randomized Kaczmarz

1 | INTRODUCTION

Given a system of equations;

$$Ax = b, \text{where } A \in \mathbb{R}^{m \times n}, b \in \mathbb{R}^{m}, \tag{1}$$

the classical Kaczmarz (CK) algorithm[1] is an iterative method to solve such linear system of equations. It was later rediscovered as Algebraic Reconstruction Technique (ART)[2] to be used in image reconstruction problems. CK and several other Kaczmarz-like methods are being used in fields of engineering problems such as signal processing,[3] compressed sensing,[4] neural networks[5] and inverse problems.[6]

Algorithm 1 shows the steps of CK. It starts with an arbitrary vector $x^{(0)}$ in the row space of matrix A. At iteration k, the algorithm finds $x^{(k)}$; the closest vector to $x^{(k-1)}$ satisfying $A^T(i,:)x^{(k)} = b(i)$, where $i \equiv k(\ mod\ \ m)$. We use MATLAB notation $A^T(i,:)$ to denote the i^{th} row of A and $b(i)$ is the i^{th} element of b. At line 5 of the algorithm, the iteration is formulated as follows:

$$x^{(k+1)} \leftarrow x^{(k)} + \frac{b(i) - \langle x^{(k)}, A(i,:)\rangle}{\|A(i,:)\|_2^2} A(i,:), \tag{2}$$

Algorithm 1. Classical Kaczmarz (CK)

procedure CK(A, b, T) ▷ $A \in \mathbb{R}^{m\times n}, b \in \mathbb{R}^m$
  Set $x^{(0)}$ to any vector in the row space of matrix A
  for $k = 0, 1, 2, \dots, T-1$ **do**
    $i \leftarrow k \ (mod\ m)$
    $x^{(k+1)} \leftarrow x^{(k)} + \frac{b(i)-\langle x^{(k)}, A(i,:)\rangle}{\|A(i,:)\|_2^2} A(i,:)$
  end for
  return $x^{(T)}$
end procedure

Algorithm 2. Randomized Kaczmarz (RK)

procedure RK(A, b, T) ▷ $A \in \mathbb{R}^{m\times n}, b \in \mathbb{R}^m$
  Set $x^{(0)}$ to any vector in the row space of matrix A
  for $k = 0, 1, 2, \dots, T-1$ **do**
    Pick $i \in \{1, \dots, m\}$ with probability $p_i := \|A(i,:)\|_2^2 / \|A\|_F^2$
    $x^{(k+1)} \leftarrow x^{(k)} + \frac{b(i)-\langle x^{(k)}, A(i,:)\rangle}{\|A(i,:)\|_2^2} A(i,:)$
  end for
  return $x^{(T)}$
end procedure

where $\langle\cdot,\cdot\rangle$ denotes the inner product of two vectors. As illustrated in Figure 1, the iterations proceed by computing projections of consecutive approximations ($x^{(k+1)}$) onto the solution hyperplane of $A^T(i,:)x = b(i)$.

The algorithm converges faster if the angle between two consecutive rows are wide and slower if the angle is narrow. Therefore the convergence rate depends on how the rows of the matrix are ordered. To get rid of this dependency, randomized versions of the algorithm have been proposed.[7–9] Later, the convergence of randomized Kaczmarz (RK) was also proved.[10] Pseudocode of the algorithm is given in Algorithm 2, where T is the number of iterations. The same idea of this algorithm is generalized and applied as randomized Coordinate Descent and randomized Newton methods for symmetric positive definite matrices.[11] RK can be seen as a special case for stochastic gradient descent selecting the stepsize as inverse Lipschitz constant of the stochastic gradient.[12]

While being useful on over-determined systems, RK does not guarantee convergence to the least squares solution if the system is inconsistent. Since this is mostly the case on real-world problems there are several methods[13–15] introduced to alleviate this problem. Randomized Extended Kaczmarz (REK)[16] is an extension of the algorithm that guarantees convergence to the least squares solution. Other methods for finding the least squares solution are using strong under-relaxation[17] and adaptive stepsizes.[18]

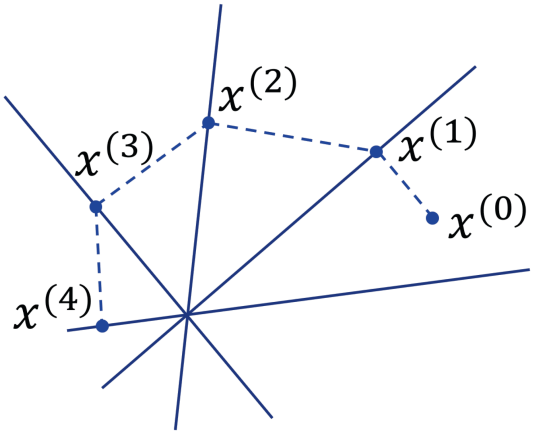

FIGURE 1 Projection of Kaczmarz iterations.

Strohmer et al.[10] emphasize the probability distribution for the selection of the rows. Note that in line 5 of Algorithm 2, the coefficient of the vector $A(i,:)$ is normalized with respect to the l_2-norm of the same vector. Therefore, it is shown that the selection method offered in Algorithm 2 is no better than selection with uniform probability.[19]

There are further studies on RK to determine the probability criterion for selecting the rows. For example, in Greedy Randomized Kaczmarz (GRK),[20,21] at each iteration a row selection that would result in a better residual has higher chance to be selected. There is also a variant using a probability distribution based on the angle between consecutive rows.[22]

Since operations executed on the main iteration of Kaczmarz (line 5 of Algorithms 1 and 2) depends on the previous computations, algorithmic changes are needed in order to execute Kaczmarz in parallel. This is also true for RK.

On distributed memory architectures, component averaging (CAV)[23] and component averaged row projections (CARP)[24] are two related parallel implementations of CK, where each process iterates through all the indexes assigned to them and the average of the results are taken according to the weights determined by the number of non-zeros of the columns. Communication is only needed for averaging and in CARP, communication may occur after several number of Kaczmarz sweeps if needed. Another example is based on parallel block projections, and uses conjugate gradient (CG) acceleration.[25,26] More recently, a distributed memory parallel implementation of RK is proposed with two different communication variations.[27]

A number of parallel RK algorithms have also been proposed for shared memory architectures. A notable example that is based on HOGWILD![28] algorithm, AsyRK,[29] updates only one random index of the solution vector asynchronously by each process. Another implementation of RK is based on selecting as many rows as the number of threads, and updating the solution vector by the weighted average of the update vectors.[30]

Global randomized block Kaczmarz (GRBK)[31] shares a similar weighted average update average but executed on block Kaczmarz operations. Randomized sparse Kaczmarz with averaging (RSKA)[32] is another block Kaczmarz method but aims to approximate a sparse solution.

In our study, we propose a new parallel RK method to solve linear system of equations on distributed memory platforms. The main contribution of this work is threefold. First the problem is intelligently partitioned into blocks so that the blocks are less dependent on each other, therefore less communication overhead is expected. Secondly, we use a frequency parameter for deciding on a proper period between each communication phase. The last one is that in our method, the processes can only communicate with each other when they share non-zero columns and when the solution entries corresponding to the indices of those columns are updated.

2 | PROPOSED METHOD

Given the matrix A in Equation (1), we assume it is normalized by rows:

$$\|A^T(j,:)\|_2 = 1 \ \forall j = 1, 2 \ \dots \ , m. \tag{3}$$

A and b are partitioned into p block rows conformably, using a partitioner mentioned in Section 3.1;

$$\begin{bmatrix} A_1 \\ A_2 \\ \vdots \\ A_p \end{bmatrix} x = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_p \end{bmatrix}. \tag{4}$$

Each process ($l \in 1 \ \dots \ p$) works with the corresponding block matrix ($A_l \in \mathbb{R}^{m_l \times n}$) and the right hand side vector ($b_l \in \mathbb{R}^{m_l}$).

The general outline of the proposed algorithm, Parallel Randomized Kaczmarz (PARK), is given in Algorithm 3. For every iteration k until T, each process l picks a random row number $i_l \in 1 \cdots m_l$ and executes main Kaczmarz iteration in parallel;

$$x_l^{(k+1)} \leftarrow x_l^{(k)} + (b_l(i_l) - \langle x_l^{(k)}, A_l(i_l,:)\rangle)A_l(i_l,:). \tag{5}$$

Since the updated indices of $x_l^{(k+1)}$ are the ones corresponding to the non-zero entries of $A_l(i_l,:)$ in the worst case, we need to keep track of those indices. To achieve this, an index list C_l in each process is updated on each iteration as:

$$C_l \leftarrow C_l \cup \text{non-zero indices of } A_l(i_l,:). \tag{6}$$

When k is a multiple of $\frac{m}{pf}$, process l sends the indices from C_l and their corresponding values on $x_l^{(k)}$ to the relevant processes. Then, the updated index list is reset: $C_l \leftarrow \emptyset$.

The relevant processes are determined by the shared non-zero columns between the blocks. For an index i, l_x and l_y are two relevant processes if both A_{l_x} and A_{l_y} have at least one non-zero value on their i^{th} columns.

Algorithm 3. Parallel randomized Kaczmarz (PARK)

procedure PARK($A, b, x^{(0)}, f, p, T$) ▷ $A \in \mathbb{R}^{m\times n}, b \in \mathbb{R}^m$
  Given A has normalized rows:$\|A^T(j,:)\|_2 = 1\ \forall j = 1, 2 \dots, m$
  Partition A into p block rows A_p, and b into b_p.
  for each block l **do** in parallel
    for $k = 0, 1, 2, \dots, T-1$ **do**
      Pick $i_l \in \{1, \dots, m_l\}$ with equal probability
      $x_l^{(k+1)} \leftarrow x_l^{(k)} + (b_l(i_l) - \langle x_l^{(k)}, A_l(i_l,:)\rangle)A_l(i_l,:)$
      Update C_l
      If $k(\mathrm{mod}\ \frac{m}{pf}) = 0$, communicate and $C_l \leftarrow \emptyset$
    end for
  end for
  return $x^{(T)}$
end procedure

3 | REDUCING COMMUNICATION TIME

While working with an m by n matrix, each Kaczmarz iteration performs only $2n$ multiplication and $2n + 1$ addition operations at most. Therefore, "iterations" in the context of Kaczmarz are much cheaper compared to other classical iterative methods. For sparse matrices, which is mostly the case in practice, the iterations are even cheaper. However, consecutive iterations depend on each other which is an issue for parallel implementations, even for the sparse case. The communication an synchronization could dominate the total cost. Hence, for parallel implementations of RK, it is crucial to reduce the time consumed by communication and synchronization.

3.1 | Preprocessing via partitioning

At each Kaczmarz iteration, the solution vector indices to be updated are determined by the non-zero columns of the selected row. Therefore, the dependency between iterations that work on different rows are determined by the shared non-zero columns of the selected rows.

In the parallel implementation where each process works on different blocks of the matrix, communication is only required if the blocks have dependent rows. This dependency is determined by the structure of the matrix.

Graph[33] and hypergraph[34] based combinatorial methods are commonly used to partition the input matrix for different parallel applications. The main objective of these methods is to minimize the communication overhead between parallel processors as much as possible while maintaining load balance on them.

In our study, we use METIS,[35] a multilevel graph partitioning tool, and PaToH,[36] a multilevel hypergraph partitioning tool, to determine row blocks of A so that blocks have fewer dependent rows with each other. In this way, the communication overhead of PARK can be reduced and the number of nonzero in each block will be similar. METIS is often utilized for partitioning square sparse matrices since it employs the graph model[34] of a sparse matrix. On the other hand, PaToH uses the hypergraph model[34] of a sparse matrix, giving it the advantage of partitioning both rectangular and square matrices. For METIS, partitioning rectangular matrices is feasible by creating and partitioning a bipartite graph model[37] of the matrix; however, this process is not as straightforward as with hypergraphs.[34] We compare naive partitioning, METIS and PaToH on different problems to evaluate their effects on the number of shared columns between blocks and the convergence performance.

3.2 | Other algorithmic improvements

We also offer other algorithmic improvements to limit the communication. Similar to CARP,[24] which was a parallel algorithm using CK as inner iterations, the communication is limited to occur between fixed number of RK iterations. However, to achieve the best performance, the best value to be chosen for the number of iterations between each communication, depends on the properties of the problem and the number of parallel processes. The importance of the contribution of communicating some values to the solution could also differ. Therefore, we introduce a frequency parameter f, and execute numerical experiments with various communication frequencies to see its effect on the performance.

In our study, processes communicate only after every $\frac{m}{pf}$ iterations, where $\frac{m}{p}$ is the average number of rows per process. Using a smaller frequency parameter f, results in less communication.

If there are no shared columns between block rows, the operation becomes embarrassingly parallel. Furthermore, as also been done by earlier parallel implementations,[27] we communicate only the resulting vector indices corresponding to the columns that are shared by at least two blocks. In the proposed method, we limit the communicated indices further by allowing communication only between two processes having the same non-zero column. Each process has a bit-wise table keeping the non-zero column index shared by other processes. The table requires additional $(p-1)n$ bits memory per process.

Another improvement that we propose is to reduce total communication by sending only the updated indices. If an index of the solution vector for a process is not changed by the iterations executed after the last communication, this process does not send the corresponding entry to others even if they share the same non-zero column index. Each process keeps a list of updated indices until the communication operation. It sends only those indices and the corresponding solution vector entries to the relevant processes and clears this list afterwards.

4 | NUMERICAL RESULTS

In order to evaluate the performance of the proposed method, first we perform sequential experiments and then we illustrate its parallel performance. In the sequential experiments, relatively small matrices are used to show the effects of different partitioning methods on the communication length and convergence. For parallel experiments, larger matrices are used to illustrate parallel speedups. Due to the random nature of the proposed algorithm, each run initializes a random seed based on the clock and presented results are the average of 10 such runs. We report the relative residuals, $\frac{\|Ax-b\|_2}{\|b\|_2}$. Iterations start with $x^{(0)}$ as the zero vector. The right hand side vector b is obtained from a solution vector x of all ones, in order to ensure the system is consistent. Test matrices are obtained from the SuiteSparse matrix collection.[38] Tables 1, 2 and 3 show the number of rows, columns, non-zero ratio ($\frac{nnz}{mn}$, where nnz is the number of non-zeros) and the problem domain of the test matrices.

TABLE 1 Test matrices for determined problems.

Matrix name	m	n	Non-zero ratio	Problem kind
oscil_dcop_01	430	430	8.35×10^{-3}	Circuit simulation
Trefethen_300	300	300	5.20×10^{-2}	Combinatorial
Trefethen_2000	2000	2000	1.05×10^{-2}	Combinatorial
circuit_2	4510	4510	1.04×10^{-3}	Circuit simulation
crystm01	4875	4875	4.43×10^{-3}	Materials
pde2961	2961	2961	1.66×10^{-3}	2D/3D
poli	4008	4008	5.10×10^{-4}	Economic
add32	4960	4960	8.07×10^{-4}	Circuit simulation
lhr02	2954	2954	4.23×10^{-3}	Chemical process
swang1	3169	3169	2.08×10^{-3}	Semiconductor device
cage9	3534	3534	3.33×10^{-3}	Directed w. graph
adder_dcop_30	1813	1813	3.42×10^{-3}	Circuit simulation
fpga_trans_01	1220	1220	4.96×10^{-3}	Circuit simulation
rajat12	1879	1879	3.63×10^{-3}	Circuit simulation
t2dal_a	4257	4257	2.07×10^{-3}	Duplicate model reduction
laser	3002	3002	9.99×10^{-4}	Materials
c-28	4598	4598	1.45×10^{-3}	Optimization
1138_bus	1138	1138	3.13×10^{-3}	Power network
af_shell10	1.51×10^{6}	1.51×10^{6}	2.30×10^{-5}	Structural
analytics	3.04×10^{5}	3.04×10^{5}	2.17×10^{-5}	Data analytics
ASIC_680k	6.83×10^{5}	6.83×10^{5}	5.66×10^{-6}	Circuit simulation
bundle_adj	5.14×10^{5}	5.14×10^{5}	7.67×10^{-5}	Computer vision

TABLE 2 Test matrices for overdetermined problems.

Matrix name	m	n	Non-zero ratio	Problem kind
photogrammetry2	4472	936	8.85×10^{-3}	Computer graphics
n2c6-b5	4945	3003	2.00×10^{-3}	Combinatorial
n4c5-b4	2852	1350	3.70×10^{-3}	Combinatorial
Franz4	6784	5252	1.31×10^{-3}	Combinatorial
ch7-8-b4	1.41×10^{5}	5.88×10^{4}	8.50×10^{-5}	Combinatorial
D6-6	1.21×10^{5}	2.37×10^{4}	5.12×10^{-5}	Combinatorial
Franz11	4.71×10^{4}	3.01×10^{4}	2.32×10^{-4}	Combinatorial
LargeRegFile	2.11×10^{6}	8.01×10^{5}	2.92×10^{-6}	Circuit simulation

TABLE 3 Test matrices for underdetermined problems.

Matrix name	m	n	Non-zero ratio	Problem kind
n3c6-b9	2511	4935	2.03×10^{-3}	Combinatorial
model4	1337	4962	6.90×10^{-3}	Linear programming
lp_pilot	1441	4860	6.34×10^{-3}	Linear programming
cep1	1521	4769	1.14×10^{-3}	Linear programming
lp_bnl2	2324	4486	1.44×10^{-3}	Linear programming
mri2	6.32×10^{4}	1.48×10^{5}	6.10×10^{-5}	Computer vision
watson_1	2.01×10^{5}	3.87×10^{5}	1.36×10^{-5}	Linear programming

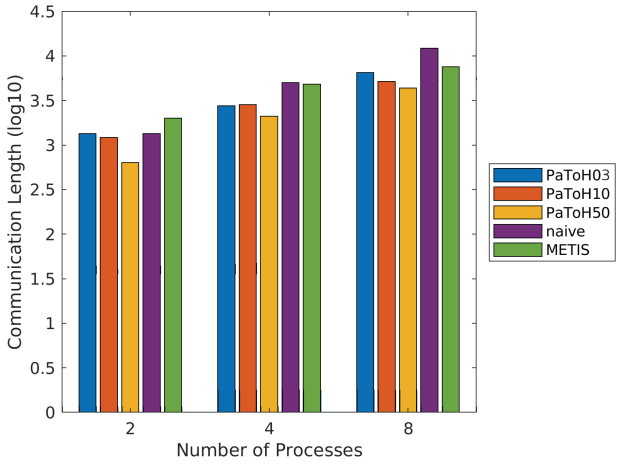

FIGURE 2 Communication length for various partitioning methods (circuit_2).

4.1 | Sequential experiments

In this section, we compare the performance improvement of PARK over sequential RK in terms of relative residual when the number of iterations executed are the same. Moreover, we evaluate the effects of different partitioning methods, compare both the residuals and shared non-zero columns between blocks. As the number of shared columns between blocks decrease, the dependency between processes and the need for communication also decrease. For each partitioning method, we present the communication length (Figure 2) which is the number of communicated entries of $x^{(k)}$ for 2, 4 and 8 processes. It is determined by the sum of the shared non-zero columns between A_i and A_j, where $i \neq j$ and $i, j \in 1 \ldots p$.

Another comparison is based on the targeted imbalance ratio between blocks. The imbalance ratio, which is calculated as the ratio of the maximum loaded partition to the average load distributed across all partitions, measures the degree of imbalance among the parts. Using a higher imbalance ratio relaxes the constraint on the loads of the parts, thus providing more opportunities for the partitioning tool to reduce the

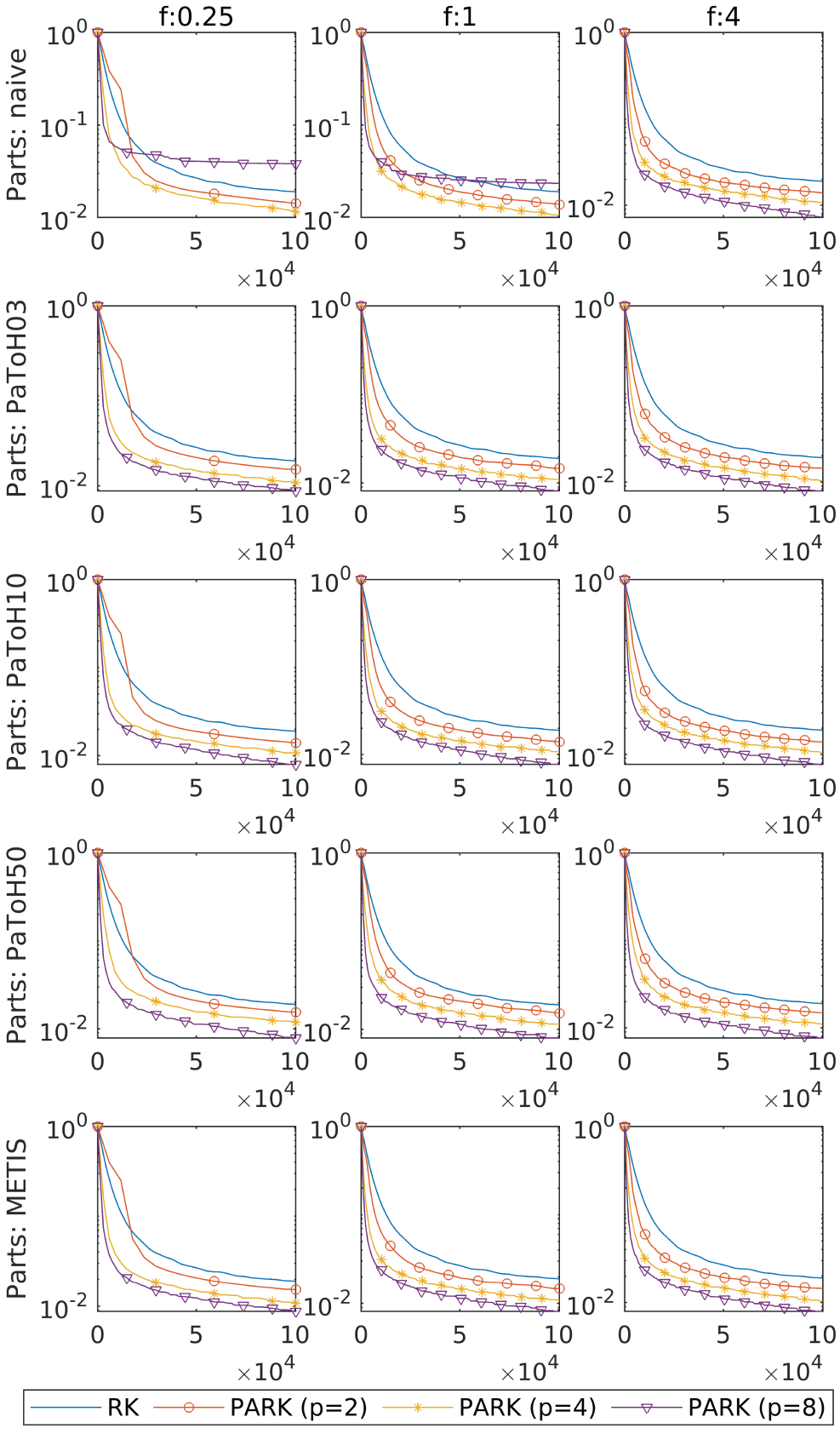

FIGURE 3 The vertical and horizontal axes represent relative residual and the number of iterations, respectively (lhr02).

interconnection between parts. Since every iteration of RK executes on a single row, the negative effect of imbalance between blocks is expected to be less than its effect on the classical iterative methods. On the other hand, reducing the number of shared non-zero columns is a more important partitioning objective for the proposed method. PaToH partitioning experiments are executed with 10% (PaToH10) and 50% (PaToH50) target imbalance ratios in addition to the default 3% (PaToH03). METIS is only evaluated with default parameters because, for almost all of the test matrices, increasing the allowed imbalance ratio does not affect the output of the partitioner.

We also measure the effects of communication frequency on the convergence of PARK. Communication frequency is inversely proportional to the period between each communication. Hence, the behavior with respect to the change of frequency is also important in order to demonstrate the effect of partitioning method used.

The tests are executed on MATLAB 2022b[39] using 27 matrices from different problem domains, with number of processes ($p \in \{2, 4, 8\}$) and frequency parameter ($f \in \{0.25, 1, 4\}$). For each square matrix in the dataset, there are 45 test results (for different partitioning methods, p and f values). In Figure 3 horizontal and vertical axes represent the iteration numbers (per block) and the relative residuals, respectively. As seen in the figure, using a partitioning tool enables the proposed method to perform significantly better than RK for different number of partitions. Furthermore, relative residuals improve as p increases (which is expected since the iteration numbers are given as per block) provided that a partitioning tool is used or a higher f is used.

Other than reducing the length of each communication (Figure 4), preprocessing via partitioning might also be beneficial for allowing a reduced communication frequency. If the significant non-zero values (i.e., non-zero values that are important for convergence) are distributed so that the shared columns between different blocks contain fewer of them, processes can also communicate less without any significant performance degradation and vice versa. For example Figure 5 for model4 matrix, illustrates that the residual decreases for a given f and p when PaToH03 partitioning is used.

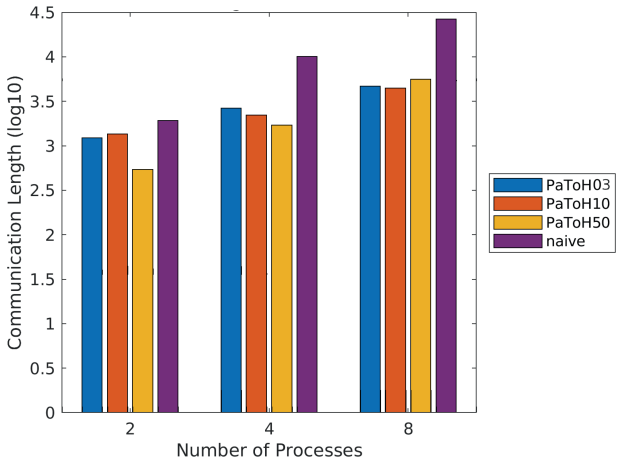

FIGURE 4 Communication length for various partitioning methods (model4).

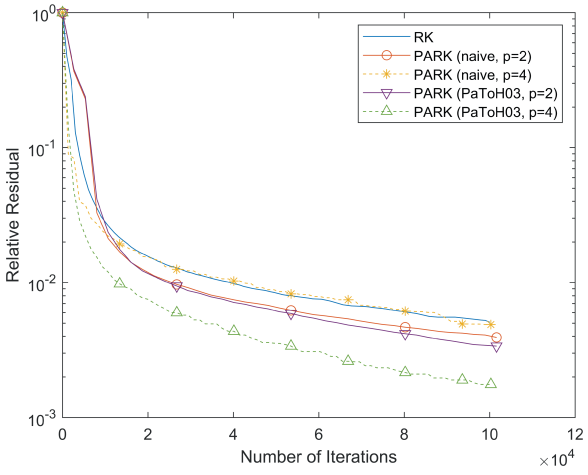

FIGURE 5 Experiments on model4 using $f = 0.25$ and $p \in \{2, 4\}$.

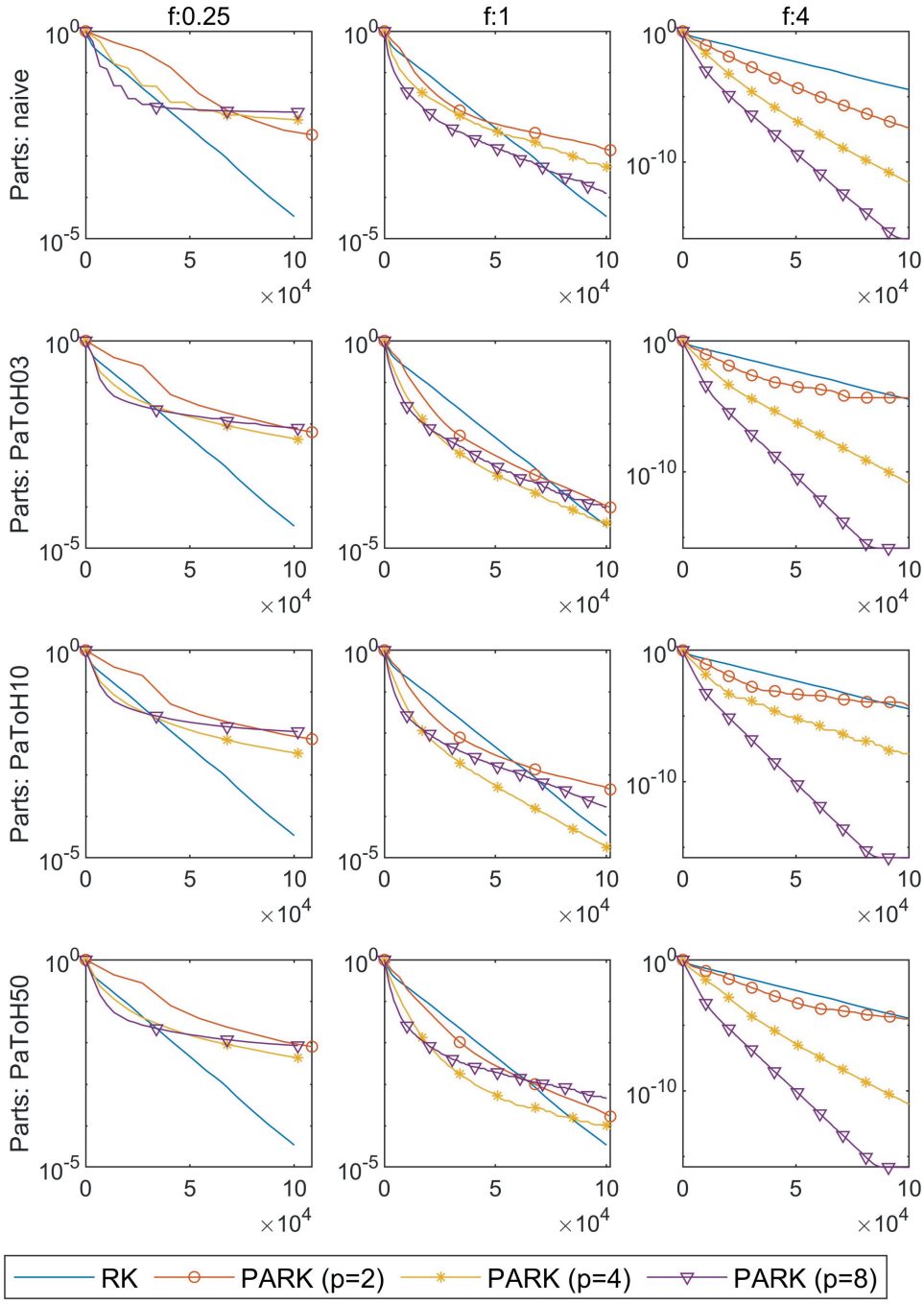

FIGURE 6 The vertical and horizontal axes represent relative residual and the number of iterations, respectively (Franz4).

Almost in all cases, as expected, partitioning methods do not eliminate dependencies completely. Therefore, communication is still required. Furthermore, for some problems, communicating frequently is essential for acceptable convergence rates. In Figure 6 (including 36 test results, since METIS is not used for overdetermined and underdetermined problems), it is shown that PARK only converges better than RK if f is 4, regardless of the partitioning method chosen. Figure 7 shows that partitioning reduces the communication length. This might be caused by the structure of the matrix that leads to a partitioning where the significant non-zero values are still shared after reordering. Nonetheless, partitioning is still effective for these problems, since it reduces the total communication length.

Finally, Table 4 shows percentage of the cases that the given methods works within best 20% with respect to the communication length. In other words, 0% and 100% means none and all of the test instances are within the best, respectively. As seen in the Table, for more than 90% of the test problems, matrices partitioned with PaToH50 has communication length within the best. Therefore, PaToH50 is chosen for the following parallel experiments.

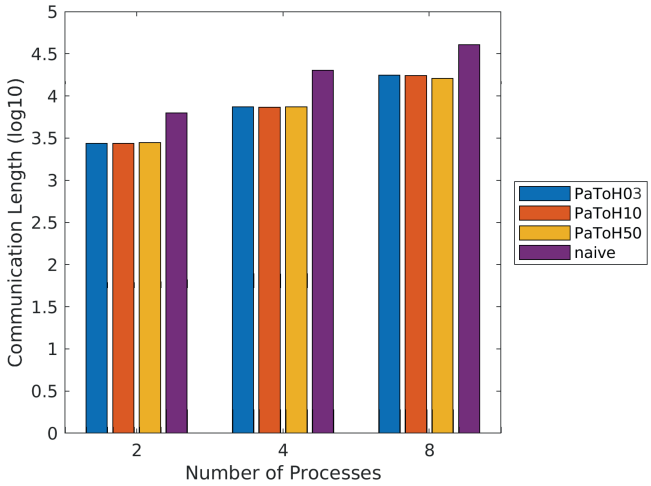

FIGURE 7 Communication length for various partitioning methods (Franz4).

TABLE 4 Percentage of test instances that are within the best 20%.

	Partitioner				
System	**Naive**	**PaToH03**	**PaToH10**	**PaToH50**	**METIS**
Determined	12.96%	75.93%	81.48%	90.74%	59.26%
Overdetermined	00.00%	58.33%	58.33%	91.67%	—
Underdetermined	06.67%	73.33%	93.33%	86.67%	—
Overall	09.88%	72.84%	80.25%	90.12%	59.26%

TABLE 5 Relative residuals and speedups (using the best frequency).

	Relative	Naive partitioning						Partitioning with PaToH50					
Matrix name	**Residual**	**2**	**4**	**8**	**16**	**32**	**64**	**2**	**4**	**8**	**16**	**32**	**64**
af_shell10	1×10^{-3}	1.61	3.03	4.60	6.25	5.26	4.54	1.27	3.19	5.37	9.49	12.78	21.88
analytics	4×10^{-4}	1.00	—	—	—	—	—	3.07	4.22	4.40	2.30	—	—
ASIC_680k	5×10^{-4}	3.94	4.66	3.47	2.78	2.04	—	5.16	4.73	2.41	—	—	—
bundle_adj	1×10^{-2}	1.60	2.01	1.82	1.67	1.06	—	1.37	4.06	10.09	16.27	29.42	25.01
ch7-8-b4	1×10^{-15}	1.47	1.82	2.09	1.90	—	—	1.36	1.69	2.02	1.62	—	—
D6-6	1×10^{-8}	1.86	1.90	2.24	2.97	2.18	—	1.31	1.73	2.10	2.42	2.07	—
Franz11	1×10^{-15}	1.21	1.25	1.61	—	—	—	1.70	2.45	3.29	3.82	—	—
LargeRegFile	7×10^{-3}	2.58	5.38	13.59	25.59	40.29	144.05	1.39	3.31	9.77	18.03	27.47	102.47
mri2	6×10^{-4}	2.10	4.46	9.51	18.46	16.61	16.33	1.49	4.17	8.95	18.08	23.23	32.40
watson_1	3×10^{-3}	2.11	4.13	6.86	12.30	17.67	23.96	2.16	5.41	11.79	24.09	48.70	71.63

4.2 | Parallel experiments

For parallel experiments, we execute our experiments for 10 matrices from 7 application domains. Matrices are chosen with the number of rows and columns between 20,000 and 5,000,000. The lower limit is set to work with data large enough to observe the benefits of parallelism and the upper limit is determined based on the limitations of the computing platform. The computing platform is a cluster with 2×28 Core Intel(R) Xeon(R) Gold 6258R CPU @2.70GHz processors each core having 3.4 GB memory using Centos-7.9 and OpenMPI 4.1.1 to compile and run the codes. In our implementation, processes call *MPI_Isend*() and *MPI_Irecv*() routines for interprocess communication of updated indices and the corresponding entries on the solution vector.

For each matrix, PARK is executed with naive and PaToH50 partitioning methods with $p \in \{2, 4, 8, 16, 32, 64\}$ and $f \in \{0.25, 0.5, 1, 2, 4\}$. The running time of PARK to converge to the same residual is then compared with the running time of sequential RK. In many applications, the coefficient matrix does not change and linear systems with different right hand side vectors are solved. Therefore, preprocessing time can be easily amortized and thus we do not consider the preprocessing times in the following experiments. Table 5 presents the speedup ($\frac{sequential\ execution\ time}{parallel\ execution\ time}$) results of all the test matrices where the results for the best experimented f is selected for each number of process. The results for the settings that do not grant any speedup, or the settings that can not be evaluated due to the communication size limitations of the computing platform are not included. To choose the target relative residual, we ran the baseline algorithm(sequential RK) until the relative residual levels out and those target relative residual values are used for each problem throughout the experiments, given in Table 5.

The effect of f and the selected partitioning method for the speedup can be observed in Figure 8 for matrix watson_1. As expected, for smaller p, f does not seem to have a significant effect on the performance. On the other hand, smaller frequencies tend to be better as the number of processes increases. This is caused because of the increasing communication overhead. The effect of partitioning for this problem is also clear. PaToH50 provides up to three times better performance than naive partitioning. This improvement might be caused by blocks both having less shared non-zero columns and having less significant non-zero entries on shared columns after partitioning. In addition to improving an already acceptable speedup, partitioning might also enable the proposed algorithm to achieve significantly better speedup than naive partitioning (Figure 9).

Note that some of the results on Table 5 have better speedup than the ideal. Figure 10 is a clear example of this super-linear speedup. It is probably caused by two factors combined. The first one is the independence between blocks which reduces the time cost of communication. The other one is not caused by parallelism but our approach. Recall from Section 1 that the convergence rate of RK depends on the properties of rows chosen between consecutive iterations. Since each process works in different blocks, the consecutive rows chosen are always within a block. This behavior becomes an advantage depending on the structure of the matrix. It also explains the reason for poorer performance observed after partitioning of such matrices. Another reason for partitioning to have no effect for some problems is that the matrices have dense rows.

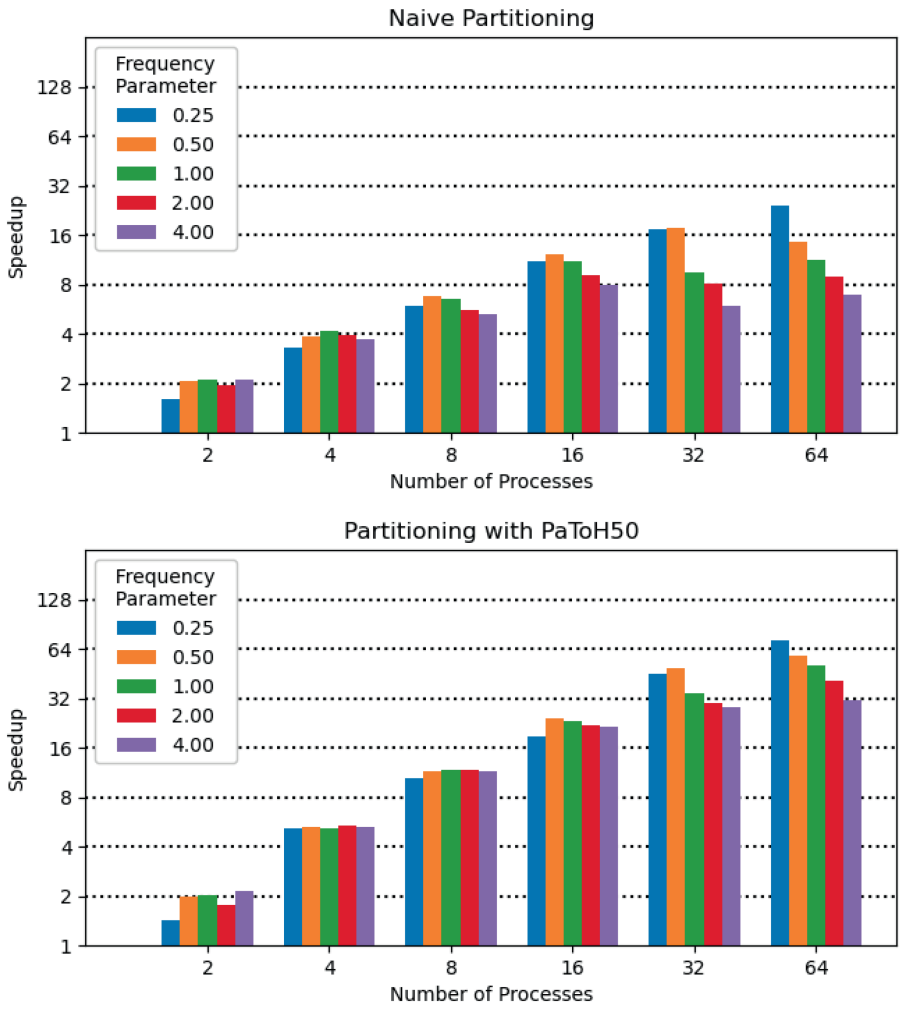

FIGURE 8 Speedup comparison for matrix watson_1.

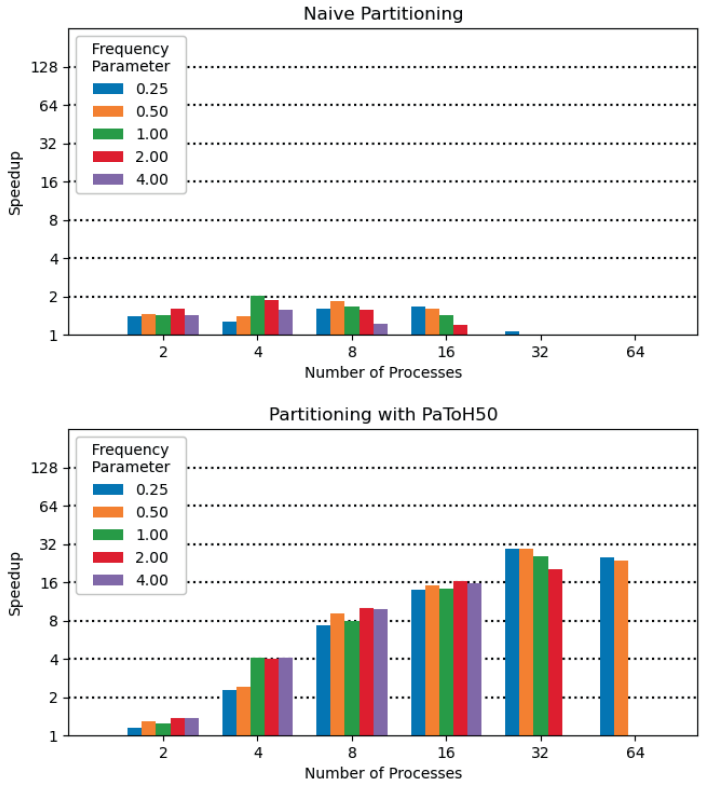

FIGURE 9 Speedup comparison for matrix bundle_adj.

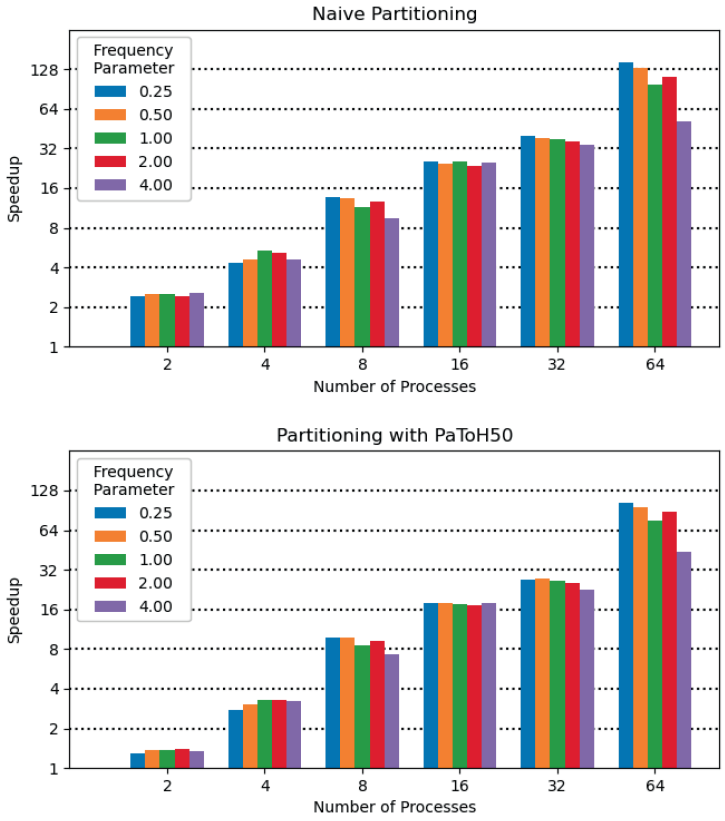

FIGURE 10 Speedup comparison for matrix LargeRegFile.

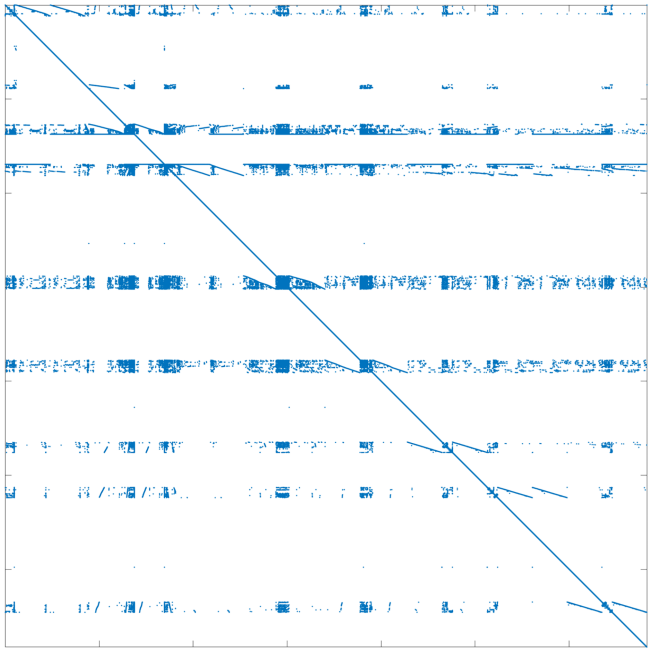

FIGURE 11 Sparsity structure of matrix ASIC_680k.

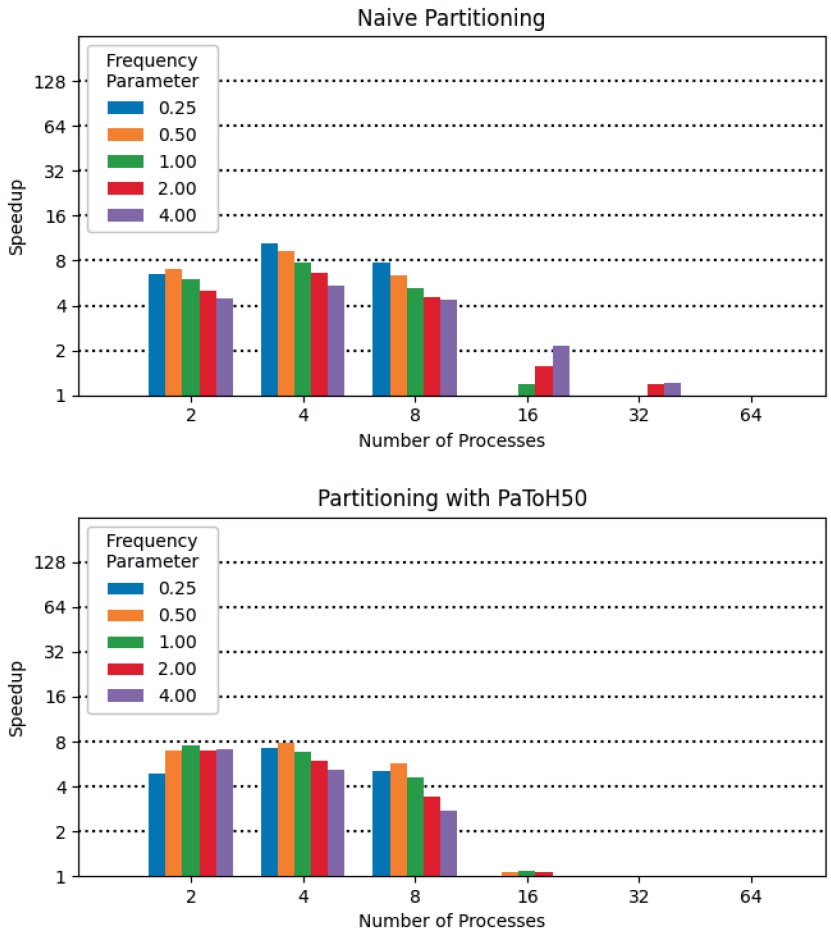

FIGURE 12 Speedup comparison for matrix ASIC_680k.

TABLE 6 Partitioning with PaToH50 speed improvement with respect to naive partitioning.

	Number of partitions					
Matrix name	**2**	**4**	**8**	**16**	**32**	**64**
af_shell10	0.79	1.05	1.17	1.52	2.43	4.82
analytics	3.07	4.22	4.40	2.30	1.00	1.00
ASIC_680k	1.31	1.02	0.70	0.36	0.49	1.00
bundle_adj	0.86	2.02	5.54	9.74	27.76	25.01
ch7-8-b4	0.93	0.93	0.97	0.85	1.00	1.00
D6-6	0.70	0.91	0.94	0.82	0.95	1.00
Franz11	1.41	1.96	2.04	3.82	1.00	1.00
LargeRegFile	0.54	0.62	0.72	0.71	0.68	0.71
mri2	0.71	0.93	0.94	0.98	1.40	1.98
watson_1	1.02	1.31	1.72	1.96	2.76	2.99
Best	3.07	4.22	5.54	9.74	27.76	25.01
Worst	0.54	0.62	0.70	0.36	0.49	0.71
Average	1.13	1.50	1.91	2.31	3.95	4.05

For all the permutations of such matrices, at least one block shares most of the non-zero columns with others. Therefore communication length can not be improved with partitioning methods that utilize row permutations only. ASIC_680k is an example for such problems. In Figure 11, the spy plot of ASIC_680k is given where dense rows are visible. Thus, as expected, partitioning does not result in a significant improvement compared to the naive partitioning (Figure 12).

In general, PARK converges faster if partitioning with PaToH50 is applied. Table 6 shows the speed improvement of partitioning with PaToH50 with respect to naive partitioning. It is also seen that as the number of processes increase, the positive effect of partitioning on parallel performance improves as well. The reason for this behavior is that communication overhead is expected to increase as the number of processes increase. PaToH50 minimizes the communication length, and improves the communication overhead.

5 | CONCLUSION AND FUTURE WORK

In this study, a distributed memory parallel randomized Kaczmarz method is proposed. The proposed method partitions the matrix so that the blocks need to communicate less. Moreover, we limit the communication frequency and the communication occurred between fixed number of iterations by keeping update lists and shared index tables. Both sequential and parallel experiments show that the proposed method converges in a fewer number of iterations with a better speedup compared to the baseline algorithms. For future work, we plan to construct a larger dataset using the results with different frequency parameters. Implementation with another partitioning algorithm (GRIP) which aims to increase the orthogonality between blocks[40,41] will be also investigated. For matrices having dense rows/columns, to overcome the difficulty that partitioning methods encounter, Schur complement approach[42] can be also considered. Lastly, we plan to use other more efficient communication subroutines provided in the MPI library.

ACKNOWLEDGMENTS

The numerical calculations reported in this paper were partially performed at TUBITAK ULAKBIM, High Performance and Grid Computing Center (TRUBA resources).

ORCID

Ercan Selçuk Bölükbaşı https://orcid.org/0000-0002-7369-8492

REFERENCES

1. Kaczmarz S. Angenäherte auflösung von systemen linearer gleichungen (english translation by Jason Stockmann): Bulletin international de l'académie polonaise des sciences et des lettres. 1937.
2. Herman GT. Image reconstruction from projections. *Fund Comput Tomography*. 1980;316;193-216.
3. Hanke M, Niethammer W. On the acceleration of Kaczmarz's method for inconsistent linear systems. *Linear Algebra Appl*. 1990;130:83-98.

4. Lorenz DA, Wenger S, Schöpfer F, Magnor M. A sparse Kaczmarz solver and a linearized Bregman method for online compressed sensing. 2014 IEEE International Conference on Image Processing (ICIP), IEEE. 2014;1347-1351.
5. Ma A, Molitor D. Randomized Kaczmarz for tensor linear systems. *BIT Numer Math*. 2022;62(1):171-194.
6. Jiao Y, Jin B, Lu X. Preasymptotic convergence of randomized Kaczmarz method. *Inverse Problems*. 2017;33(12):125012.
7. Feichtinger HG, Cenker C, Mayer M, Steier H, Strohmer T. New variants of the POCS method using affine subspaces of finite codimension with applications to irregular sampling. *Visual Communications and Image Processing'92*. Vol 1818. SPIE; 1992:299-310.
8. Herman GT, Meyer LB. Algebraic reconstruction techniques can be made computationally efficient (positron emission tomography application). *IEEE Trans Med Imaging*. 1993;12(3):600-609.
9. Natterer F. *The Mathematics of Computerized Tomography*. SIAM; 2001.
10. Strohmer T, Vershynin R. A randomized Kaczmarz algorithm with exponential convergence. *J Fourier Anal Appl*. 2009;15(2):262.
11. Gower RM, Richtárik P. Randomized iterative methods for linear systems. *SIAM J Matrix Anal Appl*. 2015;36(4):1660-1690.
12. Ramdas A. Rows vs columns for linear systems of equations-randomized Kaczmarz or coordinate descent? arXiv preprint arXiv:1406.5295. 2014.
13. Needell D. Randomized Kaczmarz solver for noisy linear systems. *BIT Numer Math*. 2010;50:395-403.
14. Needell D, Tropp JA. Paved with good intentions: analysis of a randomized block Kaczmarz method. *Linear Algebra Appl*. 2014;441:199-221.
15. Needell D, Zhao R, Zouzias A. Randomized block Kaczmarz method with projection for solving least squares. *Linear Algebra Appl*. 2015;484:322-343.
16. Zouzias A, Freris NM. Randomized extended Kaczmarz for solving least squares. *SIAM J Matrix Anal Appl*. 2013;34(2):773-793.
17. Censor Y, Eggermont PP, Gordon D. Strong underrelaxation in Kaczmarz's method for inconsistent systems. *Numer Math*. 1983;41:83-92.
18. Zeng Y, Han D, Su Y, Xie J. Randomized Kaczmarz method with adaptive stepsizes for inconsistent linear systems. *Numer Algorithms*. 2023;94(3):1403-1420.
19. Censor Y, Herman GT, Jiang M. A note on the behavior of the randomized Kaczmarz algorithm of Strohmer and Vershynin. *J Fourier Anal Appl*. 2009;15:431-436.
20. Bai ZZ, Wu WT. On greedy randomized Kaczmarz method for solving large sparse linear systems. *SIAM J Sci Comput*. 2018;40(1):A592-A606.
21. Bai ZZ, Wu WT. On relaxed greedy randomized Kaczmarz methods for solving large sparse linear systems. *Appl Math Lett*. 2018;83:21-26.
22. He S, Dong QL, Li X. The randomized Kaczmarz algorithm with the probability distribution depending on the angle. *Numer Algorithms*. 2023;93(1):415-440.
23. Censor Y, Gordon D, Gordon R. Component averaging: an efficient iterative parallel algorithm for large and sparse unstructured problems. *Parallel Comput*. 2001;27(6):777-808. doi:10.1016/S0167-8191(00)00100-9
24. Gordon D, Gordon R. Component-averaged row projections: a robust, block-parallel scheme for sparse linear systems. *SIAM J Sci Comput*. 2005;27(3):1092-1117.
25. Kamath C, Sameh A. A projection method for solving nonsymmetric linear systems on multiprocessors. *Parallel Comput*. 1989;9(3):291-312.
26. Bramley R, Sameh A. Row projection methods for large nonsymmetric linear systems. *SIAM J Sci Stat Comput*. 1992;13(1):168-193.
27. Kamath G, Ramanan P, Song WZ. Distributed randomized Kaczmarz and applications to seismic imaging in sensor network. 2015 International Conference on Distributed Computing in Sensor Systems, IEEE. 2015;169-178.
28. Recht B, Re C, Wright S, Niu F. Hogwild!: a lock-free approach to parallelizing stochastic gradient descent. In: Shawe-Taylor J, Zemel R, Bartlett P, Pereira F, Weinberger KQ, eds. *Advances in Neural Information Processing Systems*. Vol 24. Curran Associates, Inc.; 2011.
29. Liu J, Wright SJ, Sridhar S. An asynchronous parallel randomized kaczmarz algorithm. arXiv preprint arXiv:1401.4780. 2014.
30. Moorman JD, Tu TK, Molitor D, Needell D. Randomized Kaczmarz with averaging. *BIT Numer Math*. 2021;61(1):337-359.
31. Niu YQ, Zheng B. On global randomized block Kaczmarz algorithm for solving large-scale matrix equations. arXiv preprint arXiv:2204.13920. 2022.
32. Tondji L, Lorenz DA. Faster randomized block sparse Kaczmarz by averaging. *Numer Algorithms*. 2023;93(4):1417-1451.
33. Karypis G, Kumar V. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM J Sci Comput*. 1998;20(1):359-392.
34. Catalyurek UV, Aykanat C. Hypergraph-partitioning-based decomposition for parallel sparse-matrix vector multiplication. *IEEE Trans Parallel Distrib Syst*. 1999;10(7):673-693.
35. Karypis G, Kumar V. METIS: A software package for partitioning unstructured graphs, partitioning meshes, and computing fill-reducing orderings of sparse matrices. 1997.
36. Çatalyürek ÜV, Aykanat C. Patoh (partitioning tool for hypergraphs). *Encyclopedia of Parallel Computing*. Springer; 2011:1479-1487.
37. Hendrickson B, Kolda TG. Partitioning rectangular and structurally unsymmetric sparse matrices for parallel processing. *SIAM J Sci Comput*. 2000;21(6):2048-2072.
38. Davis TA, Hu Y. The University of Florida sparse matrix collection. *ACM Trans Math Softw*. 2011;38(1):1-25.
39. Inc. TM. MATLAB version: 9.13.0 (R2022b). 2022.
40. Torun FS, Manguoglu M, Aykanat C. A novel partitioning method for accelerating the block cimmino algorithm. *SIAM J Sci Comput*. 2018;40(6):C827-C850.
41. Duff I, Leleux P, Ruiz D, Torun FS. Row replicated block Cimmino. *SIAM J Sci Comput*. 2023;45(4):C207-C232. doi:10.1137/22M1487710
42. Torun FS, Manguoglu M, Aykanat C. Enhancing block Cimmino for sparse linear systems with dense columns via Schur complement. *SIAM J Sci Comput*. 2023;45(2):C49-C72. doi:10.1137/21M1453475

How to cite this article: Bölükbaşı ES, Torun FŞ, Manguoğlu M. A distributed memory parallel randomized Kaczmarz for sparse system of equations. *Concurrency Computat Pract Exper*. 2024;e8274. doi: 10.1002/cpe.8274