International Scholarly Research Network
ISRN Computational Mathematics
Volume 2012, Article ID 264040, 9 pages
doi:10.5402/2012/264040

Research Article

High Performance Gibbs Sampling for IRT Models Using Row-Wise Decomposition

Yanyan Sheng[1] and Mona Rahimi[2]

[1] *Educational Measurement and Statistics, Department of Educational Psychology & Special Education, Southern Illinois University Carbondale, Carbondale, IL 62901-4618, USA*
[2] *Department of Computer Science, Southern Illinois University Carbondale, Carbondale, IL 62901, USA*

Correspondence should be addressed to Yanyan Sheng, ysheng@siu.edu

Received 15 October 2012; Accepted 4 November 2012

Academic Editors: L. S. Heath, R. Tuzun, and P. B. Vasconcelos

Copyright © 2012 Y. Sheng and M. Rahimi. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Item response theory (IRT) is a popular approach used for addressing statistical problems in psychometrics as well as in other fields. The fully Bayesian approach for estimating IRT models is computationally expensive. This limits the use of the procedure in real applications. In an effort to reduce the execution time, a previous study shows that high performance computing provides a solution by achieving a considerable speedup via the use of multiple processors. Given the high data dependencies in a single Markov chain for IRT models, it is not possible to avoid communication overhead among processors. This study is to reduce communication overhead via the use of a row-wise decomposition scheme. The results suggest that the proposed approach increased the speedup and the efficiency for each implementation while minimizing the cost and the total overhead. This further sheds light on developing high performance Gibbs samplers for more complicated IRT models.

1. Introduction

Item response theory (IRT) is a popular approach used for describing probabilistic relationships between correct responses on a set of test items and continuous latent traits (see [1–4]). In addition to educational and psychological measurement, IRT models have been used in other areas of applied mathematics and statistical research, including US Supreme Court decision-making processes [5], alcohol disorder analysis [6–9], nicotine dependency [10–12], multiple-recapture population estimation [13], and psychiatric epidemiology [14–16], to name a few.

IRT has the advantage of allowing the inference of what the items and persons have on the responses to be modeled by distinct sets of parameters. As a result, a primary concern associated with IRT research has been on parameter estimation, which offers the basis for the theoretical advantages of IRT. Specifically, of concern are the statistical complexities that can often arise when item and person parameters are simultaneously estimated (see [1, 17–19]). More recent attention has focused on the fully Bayesian estimation where Markov chain Monte Carlo (MCMC, [20, 21]) simulation techniques are used. In spite of the many advantages, the fully Bayesian estimation is computationally expensive, which further limits its actual applications. It is hence important to seek ways to reduce the execution time. A suitable solution is to use high performance computing via the Message Passing Interface (MPI) standard, which employs supercomputers and computer clusters to tackle problems with complex computations.

The implementation of parallel computing is, however, not straightforward due to the high data dependencies in a single Markov chain for an IRT model, such as the dependency of one state of the chain to the previous state and the dependencies among the data within the same state. Patsias et al. [22] developed a parallel algorithm to implement Gibbs sampling [23], one of the most efficient MCMC algorithms, to the two-parameter normal ogive

(2PNO, [24]) IRT model. In their study, the implementation of parallel computing was realized through decomposition of data matrices and item parameters into columns while minimizing the communication overhead among processors. In their implementation, the person parameters were communicated between the root and the processor nodes. Given that the fully Bayesian estimation of IRT models requires a minimum of 20 or 30 times more subjects than test items, which typically occurs in a test situation, it is believed that one can reduce the communication overhead if item parameters are communicated instead of person parameters. Hence, an alternative approach is to decompose data matrices and person parameters into rows. The present study is to develop such a high performance computing algorithm for the 2PNO model and further demonstrate its utility by comparing it with that developed in [22].

The remainder of the paper is organized as follows. Section 2 reviews the 2PNO IRT model, the Gibbs sampler, and the decomposition scheme used in [22]. Section 3 illustrates the approach we propose in the present study to parallelize the serial algorithm. In Section 4, the performance of the developed parallel algorithm is investigated by comparing it with the serial algorithm and further with the parallel algorithm developed in [22].

2. Preliminaries

The 2PNO IRT model provides a fundamental framework in modeling the person-item interaction by assuming one latent trait. Let $\mathbf{y} = [y_{ij}]$ denote a matrix of n responses to k items where $y_{ij} = 1$ ($y_{ij} = 0$) if the ith person answers the jth item correctly (incorrectly) for $i = 1, \ldots, n$ and $j = 1, \ldots, k$. The probability of person i obtaining correct response for item j is then defined for the 2PNO model as

$$P\left(y_{ij} = 1\right) = \Phi\left(\alpha_j\theta_i - \beta_j\right) = \int_{-\infty}^{\alpha_j\theta_i - \beta_j} \frac{1}{\sqrt{2\pi}} e^{-t^2/2} dt, \quad (1)$$

where the scalars α_j and β_j denote item parameters, θ_i denotes the continuous person trait parameter, and Φ denotes the unit normal cdf.

The Gibbs sampler involves updating three sets of parameters in each iteration, namely, an augmented continuous variable Z_{ij} (which is positive if $y_{ij} = 1$ and negative if $y_{ij} = 0$), the person parameter θ_i, and the item parameters $\boldsymbol{\xi}_j$, where $\boldsymbol{\xi}_j = (\alpha_j, \beta_j)'$ from their respective full conditional distributions, namely,

$$Z_{ij} \mid \cdot \sim \begin{cases} N_{(0,\infty)}\left(\alpha_j\theta_i - \beta_j\right), & \text{if } y_{ij} = 1 \\ N_{(-\infty,0)}\left(\alpha_j\theta_i - \beta_j\right), & \text{if } y_{ij} = 0, \end{cases} \quad (2)$$

$$\theta_i \mid \cdot \sim N\left(\frac{\sum_j \left(Z_{ij} + \gamma_j\right)\alpha_j + \mu}{1/\sigma^2 + \sum_j \alpha_j^2}, \frac{1}{1/\sigma^2 + \sum_j \alpha_j^2}\right), \quad (3)$$

$$\boldsymbol{\xi}_j \mid \cdot \sim N\left((\mathbf{x}'\mathbf{x})^{-1}\mathbf{x}'\mathbf{Z}_j, (\mathbf{x}'\mathbf{x})^{-1}\right) I\left(\alpha_j > 0\right), \quad (4)$$

where $\mathbf{x} = [\theta, -1]$, assuming $\theta_i \sim N(\mu, \sigma^2)$, $\alpha_j > 0$ and $p(\beta_j) \propto 1$ (see, e.g., [25, 26]).

Hence, with starting values $\boldsymbol{\theta}^{(0)}$ and $\boldsymbol{\xi}^{(0)}$, observations$(\mathbf{Z}^{(l)}, \boldsymbol{\theta}^{(l)}, \boldsymbol{\xi}^{(l)})$ can be simulated from the Gibbs sampler by iteratively drawing from their respective full conditional distributions as specified in (2), (3), and (4). To go from $(\mathbf{Z}^{(l-1)}, \boldsymbol{\theta}^{(l-1)}, \boldsymbol{\xi}^{(l-1)})$ to $(\mathbf{Z}^{(l)}, \boldsymbol{\theta}^{(l)}, \boldsymbol{\xi}^{(l)})$, it takes three transition steps:

(1) draw $\mathbf{Z}^{(l)} \sim p(\mathbf{Z} \mid \boldsymbol{\theta}^{(l-1)}, \boldsymbol{\xi}^{(l-1)})$;

(2) draw $\boldsymbol{\theta}^{(l)} \sim p(\boldsymbol{\theta} \mid \mathbf{Z}^{(l)}, \boldsymbol{\xi}^{(l-1)})$;

(3) draw $\boldsymbol{\xi}^{(l)} \sim p(\boldsymbol{\xi} \mid \mathbf{Z}^{(l)}, \boldsymbol{\theta}^{(l)})$.

This iterative procedure produces a sequence of $(\boldsymbol{\theta}^{(l)}, \boldsymbol{\xi}^{(l)})$, $l = 0, \ldots, L$. To reduce the effect of the starting values, early iterations in the Markov chain are set as burn-ins to be discarded. Samples from the remaining iterations are then used to summarize the posterior density of item parameters $\boldsymbol{\xi}$ and ability parameters $\boldsymbol{\theta}$. For more detailed examples of the Gibbs sampler, interested readers should refer to Cassella and George [27].

Since a large number of iterations are needed for the Markov chain to reach convergence, the algorithm is computationally intensive and requires a considerable amount of execution time, especially with large datasets [26]. In an effort to achieve a speedup, Patsias et al. [22] developed a parallel algorithm where they used domain decomposition to divide the updating tasks into blocks. In particular, their approach was to assign each processor a column block of the data matrix ($\mathbf{y}$) so that they could update a block of $\mathbf{Z}$ and item parameters ($\boldsymbol{\xi}$). This scheme is depicted in Figure 1 and is now called column-wise decomposition. Since $\boldsymbol{\theta}$ is of size $n \times 1$ (a column vector), it could not be decomposed. Therefore, in each iteration, all processor nodes need to send their portions of $\mathbf{Z}$ and $\boldsymbol{\xi}$ to the root for it to update $\boldsymbol{\theta}$ and consequently send back to the rest of the nodes to proceed. The approach of [22] is certainly limited in situations where n is large because the number of communications is increased. Given that test data typically involve less k than n, it is reasonable to believe that if the domain decomposition is done differently, such as the one depicted in Figure 2, one can achieve a greater speedup. The present study focuses on the development of the parallel algorithm using such a row-wise decomposition.

3. Methodology

The study was performed using the Maxwell Linux cluster, a cluster with 106 processing nodes. Maxwell uses the message-passing model via the MPICH MPI framework implementation. One of the 106 nodes acted as the root node, while the rest of the nodes acted as slave nodes. The root node was responsible for generating and partitioning the matrix $\mathbf{y}$, transmitting the submatrices, updating and broadcasting item parameters, and recording execution time, in addition to assuming the same duties as the slave nodes.

Each node on the cluster has an Intel Xeon dual CPU quad-core processor clocked at 2.3 GHz, 8 GB of RAM, 90 TB storage, and Linux 64 bit operating system. MPICH allows the user to choose how many nodes to use before

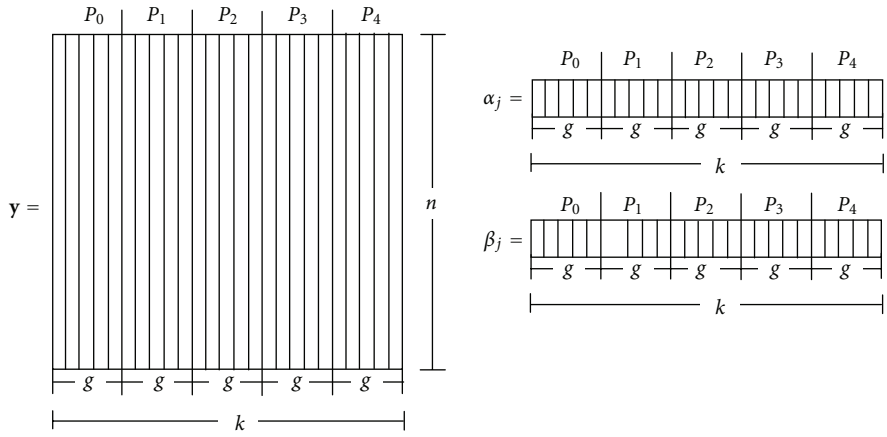

Figure 1: Domain decomposition of the input $\mathbf{y}$ matrix and item parameter vectors by column using five processing nodes.

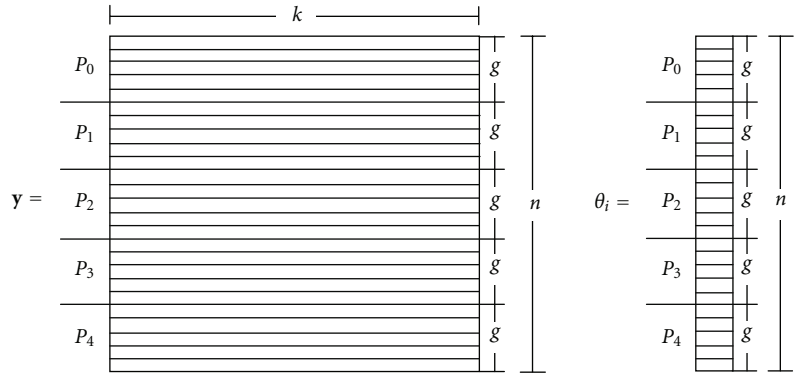

Figure 2: Domain decomposition of the input $\mathbf{y}$ matrix and the person parameter vector by row using five processing nodes.

the execution of a program so that various numbers of processing nodes may be used in every execution.

3.1. Parallel Gibbs Sampler. Given the decomposition of $\mathbf{y}$ and $\boldsymbol{\theta}$ that is illustrated in Figure 2, we see that each processor is using a submatrix of $\mathbf{y}$ to update a block of $\mathbf{Z}$ (which is of the same size of $\mathbf{y}$) and $\boldsymbol{\theta}$. For instance, P_0 is updating a block of $\mathbf{Z}$, $\mathbf{Z}_{p_0}$, from $\mathbf{Z}_{0,0}$ to $\mathbf{Z}_{n-1,g-1}$, and a block of $\boldsymbol{\theta}$, $\boldsymbol{\theta}_{p_0}$, from $\boldsymbol{\theta}_0$ to $\boldsymbol{\theta}_{g-1}$, where $g = k/P$ and P is the number of processing nodes.

Since $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ are each of size $1 \times k$ (a row vector), they are not decomposed and hence have to be communicated among processors. In order to minimize the communication cost, after finishing updating $\boldsymbol{\theta}$ in each iteration, each node needs to calculate $\mathbf{x}'\mathbf{x}$ and $\mathbf{x}'\mathbf{Z}$ and send them to the root for it to update $\boldsymbol{\xi}$ from

$$\boldsymbol{\xi}_j \mid \cdot \sim N\left(\left(\sum_p \mathbf{x}'\mathbf{x}\right)^{-1}\left(\sum_p \mathbf{x}'\mathbf{Z}_j\right), \left(\sum_p \mathbf{x}'\mathbf{x}\right)^{-1}\right) I\left(\alpha_j > 0\right). \tag{5}$$

This way, each processing node is sending a vector of size $2k+2$ to the root, which in return is broadcasting one message of size $2k$ to each processing node. The total data transferred between all the nodes for a single Markov chain involving L iterations is

$$L((2k+2) \times P) + L(2k \times P) = LP(4k+2). \tag{6}$$

The total data transferred between all the nodes using column-wise decomposition is $LP(2n+1)$ [22, page 67]. In practice, $n \gg 2k+1$. In addition, the sequential Gibbs sampler described in Section 2 calls for larger n than k to ensure $(\mathbf{x}'\mathbf{x})^{-1}$ in (4) exits so that item parameters can be updated. Hence, as an example, if a multiple-choice test consisting of 10 items is given to 500 examinees, the sample item response data can be denoted using a binary data matrix $\mathbf{y}$ of size 500×10. We can implement the high performance Gibbs sampling algorithm to fit the 2PNO model, which would reach convergence within 10,000 iterations. With five processing nodes used, the total data transferred is 2.10×10^6 for row-wise decomposition and 5.005×10^7 for column-wise decomposition. The difference would be even larger with a larger sample size (which is very common in large-scale testing situations), more processing nodes, and/or a longer Markov chain. Hence, the total data transferred using the column-wise decomposition are more than that of the proposed approach.

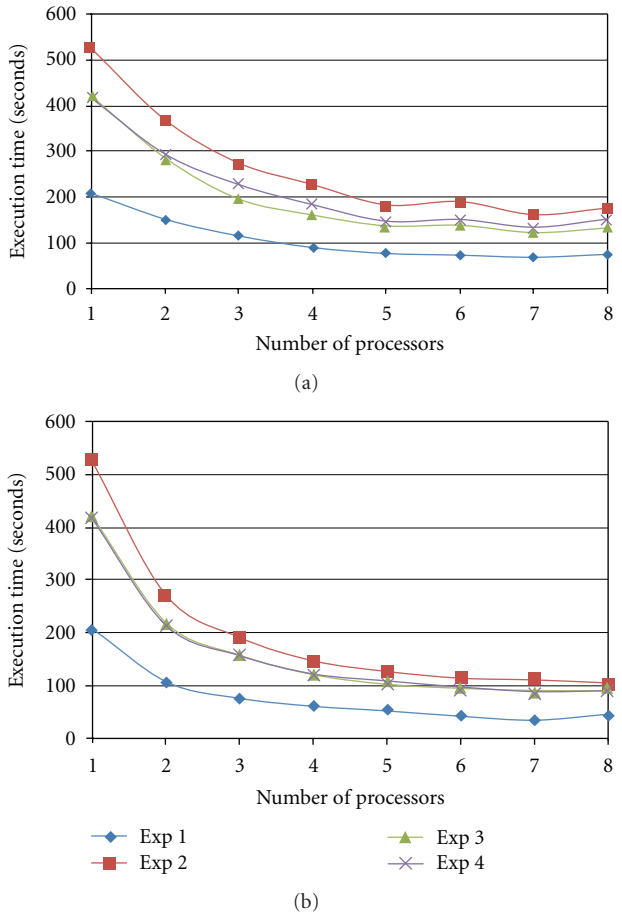

Figure 3: Execution time of the algorithm using (a) column-wise and (b) row-wise decompositions for all the experiments with one to eight processors.

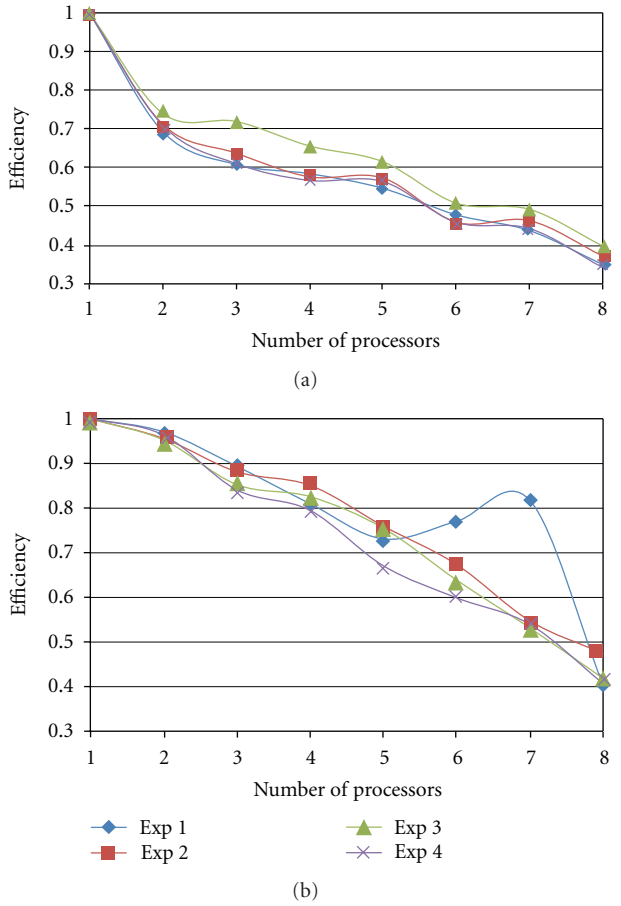

Figure 4: Relative efficiency of using parallel algorithm over the serial algorithm using (a) column-wise and (b) row-wise decompositions in all the experiments.

3.2. Implementation. The proposed algorithm was implemented in ANSI C and MPI with utilization of the GNU Scientific Library (GSL) [28]. To achieve the parallel computation as illustrated in the previous section, the MPI_Gather and MPI_Bcast routines were used for collective communications. For more detailed implementation, see the appendix for part of the source code of the parallel algorithm in updating the model parameters.

3.3. Performance Analyses. In order to investigate the advantages of the proposed parallel solution over its serial counterpart, and further over the parallel algorithm based on column-wise decomposition, four experiments were carried out in which the sample size (n), test length (k), and number of iterations (L) were manipulated to vary:

Experiment 1: $n = 2000,\ k = 50,\ L = 10{,}000,$

Experiment 2: $n = 5000,\ k = 50,\ L = 10{,}000,$

Experiment 3: $n = 2000,\ k = 100,\ L = 10{,}000,$

Experiment 4: $n = 2000,\ k = 50,\ L = 20{,}000.$

In all these experiments, one (representing the serial algorithm) to eight processing nodes were used to implement the Gibbs sampler. They were evaluated using four performance metrics in addition to the execution time. These metrics are the total overhead, relative speedup, relative efficiency, and cost.

(i) The total overhead is defined as

$$T = PT_P - T_S, \tag{7}$$

where P is the number of available processing nodes, T_S is the fastest sequential algorithm execution time and T_P is the parallel algorithm execution time.

(ii) Relative speedup is the factor by which execution time is reduced on P processors and it is defined as:

$$S = \frac{T_S}{T_P}. \tag{8}$$

(iii) Efficiency describes how well the algorithm manages the computational resources. More specifically, it tells us how much time the processors spend executing important computations [29]. Relative efficiency is defined as

$$E = \frac{T_S}{PT_P}. \tag{9}$$

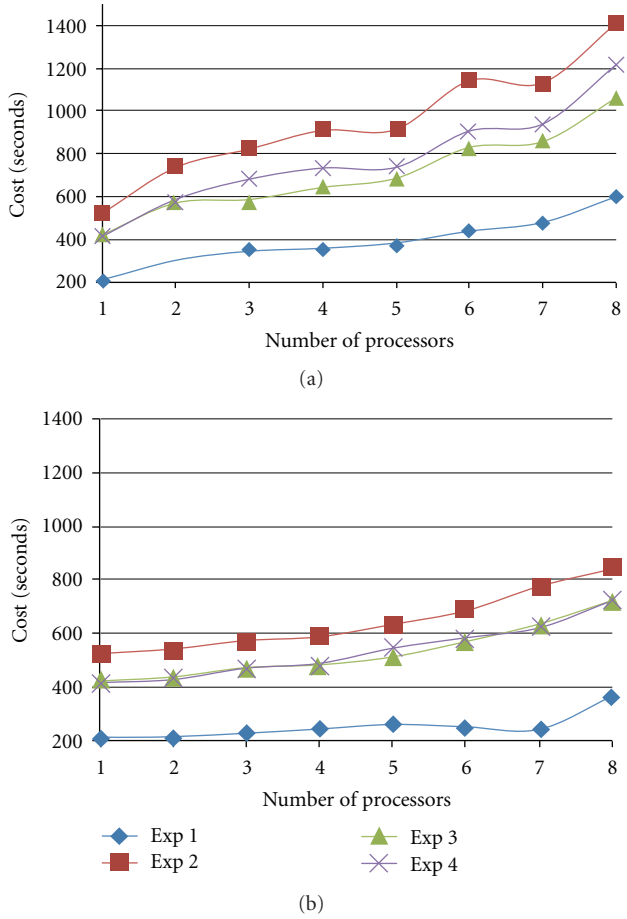

FIGURE 5: Cost of the algorithm using (a) column-wise and (b) row-wise decompositions for all the experiments with one to eight processors.

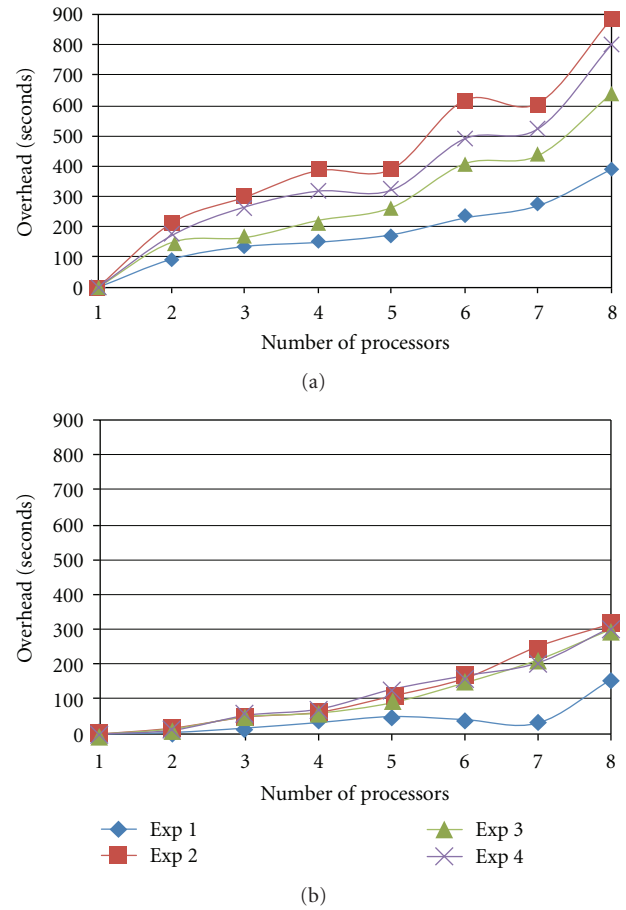

FIGURE 6: Total overhead of using parallel algorithm over the serial algorithm using (a) column-wise and (b) row-wise decompositions in all the experiments.

(iv) The definition of the cost of solving a problem on a parallel system is the product of parallel runtime and P. Consequently, cost is a quantity that reveals the sum of individual processing node runtime.

4. Numerical Results

Results from the four experiments are summarized in Figures 3 to 7, where those based on the column-wise decomposition are in (a) and those based on the row-wise decomposition are in (b). Note that the values plotted represent the average of ten replications. As expected, the execution time decreased as the number of processing nodes increased, and row-wise decomposition resulted in a much shorter execution time than column-wise decomposition in all the experimented conditions (see Figure 3).

In the case of column-wise decomposition, the communication overhead (see Figure 6(a)) incurred using six nodes overshadowed the increase in computational speedup compared to five nodes and hence at this point, some decrease in the speedup (see Figure 7(a)) and increase in the execution time (see Figure 3(a)) are observed. On the other hand, when seven nodes were used, the computational speedup overpowered the increase in communication overhead and speedup was higher than that with five nodes. A similar scenario is observed when we compare the performance using eight nodes with that using seven or five nodes. In addition, because the communication size in column-wise decomposition depends only on n, not k, Experiment 3 (where $k = 100$) maintained a higher speedup (Figure 7(a)) and was more efficient (Figure 4(a)) compared to the other experiments where $k = 50$. This agrees with the findings by [22].

With respect to row-wise decomposition, the communication overhead using eight nodes dominated the increase of the computational speedup by adding one more executing processor in experiments where $k = 50$ and $n = 2000$. Hence, there was less speedup by increasing the number of processors from seven to eight in these experiments (see Figure 7(b)). This can be further illustrated by profiling the time necessary for carrying out individual sections of the codes. In particular, as is shown in the appendix, the parallel algorithm using row-wise decomposition involves six tasks in each iteration of the Markov chain: (1) update $\mathbf{Z}$, (2) update $\boldsymbol{\theta}$, (3) update $\mathbf{x}'\mathbf{x}$ and $\mathbf{x}'\mathbf{Z}$, (4) send $\mathbf{x}'\mathbf{x}$ and $\mathbf{x}'\mathbf{Z}$

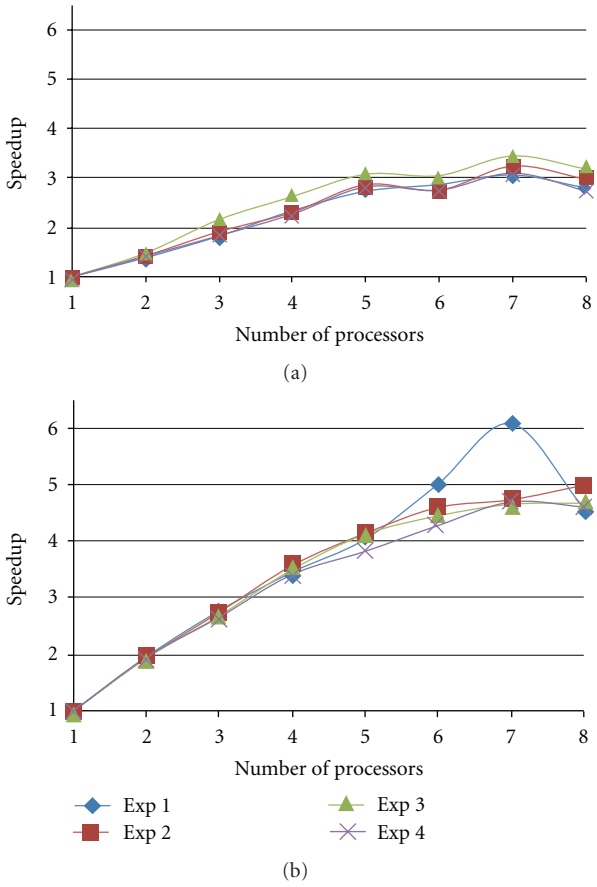

Figure 7: Relative speedup of using parallel algorithm over the serial algorithm using (a) column-wise and (b) row-wise decompositions in all the experiments.

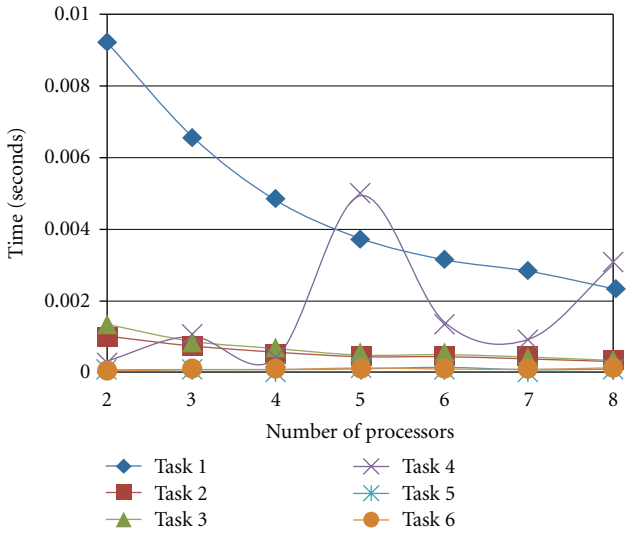

Figure 8: Execution time for completing each task within one iteration of the Markov chain using two to eight processing nodes using row-wise decomposition for $n = 2000$ and $k = 50$ (Experiments 1 and 4).

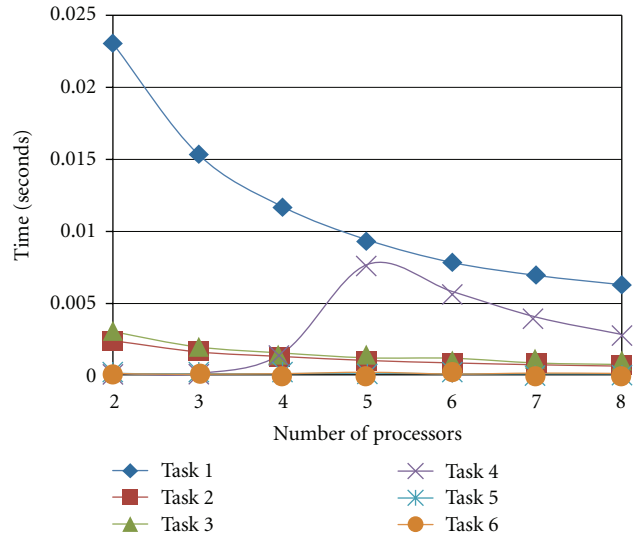

Figure 9: Execution time for completing each task within one iteration of the Markov chain using two to eight processing nodes using row-wise decomposition for $n = 5000$ and $k = 50$ (Experiment 2).

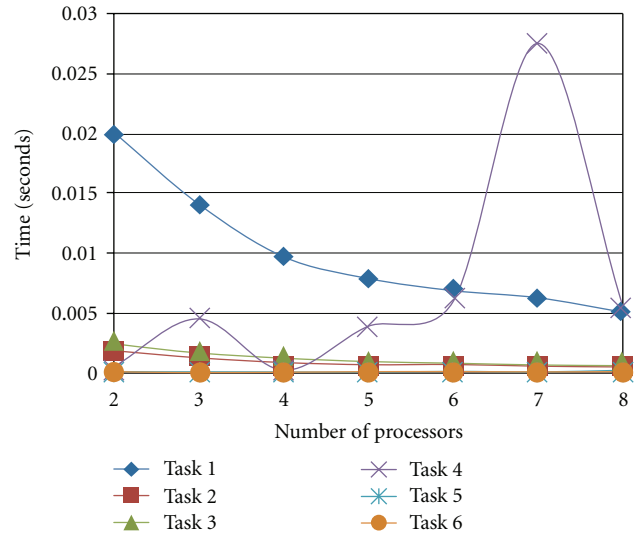

Figure 10: Execution time for completing each task within one iteration of the Markov chain using two to eight processing nodes using row-wise decomposition for $n = 2000$ and $k = 100$ (Experiment 3).

to the root, (5) update $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$, and (6) broadcast $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$. The time for completing each task in a single iteration was hence recorded and plotted in Figure 8 using two to eight processing nodes for the experiments where $k = 50$ and $n = 2000$. One can see that when more processing nodes were used, the time for updating $\mathbf{Z}$, $\boldsymbol{\theta}$, $\mathbf{x}'\mathbf{x}$, and $\mathbf{x}'\mathbf{z}$ (tasks 1 to 3) decreased, and the time for updating $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ (task 5) remained the same, whereas the time for sending $\mathbf{x}'\mathbf{x}$ and $\mathbf{x}'\mathbf{Z}$ to the root (task 4) varied depending on the number of processing nodes. This communication overhead was the largest with 5 nodes and the smallest with 4 or 7 nodes.

```
// Start iteration:
for (m = 0; m < l; m++){
  count++;
  update_Z(Z, y, TH, A, G, r);
  update_TH(TH, THV, Z, A, G, r);
  update_KSI_AR(KSI_array, Z, TH);
  MPI_Gather (KSI_array, (2*k)+2, MPI_DOUBLE, KSI_rec, (2*k)+2, MPI_DOUBLE, ROOT, MPI_COMM_WORLD);
  if(rank == ROOT){
    double XX00=0.0; // XX00 = XX[0][0]
    double XX01=0.0; // XX01 = XX[0][1]
    double XZ00=0.0; // XZ0 = XZ0[0][0]
    double XZ10=0.0; // XZ1 = XZ1[1][0]
// Retrieve x'x and x'Z from KSI_rec::
    for(i=0; i<size; i++) {
        XX00 += KSI_rec[2*k+(2*k+2)*i];
        XX01 += KSI_rec[(2*k+1)+(2*k+2)*i];
    }
    gsl_matrix_set(XX, 0, 0, XX00);
    gsl_matrix_set(XX, 0, 1, XX01);
    gsl_matrix_set(XX, 1, 0, XX01);
    gsl_matrix_set(XX, 1, 1, n);
    for(j=0; j < k; j++) {
      XZ00=0.0;
      XZ10=0.0;
      for(i=0; i<size; i++) {
        XZ00 += KSI_rec[2 * j + (2 * k + 2) * i];
        XZ10 += KSI_rec[(2 * j + 1) + (2 * k + 2) * i];
      }
      gsl_matrix_set(XZ, 0, 0, XZ00);
      gsl_matrix_set(XZ, 1, 0, XZ10);
  }
update_A_G(A, G, AV, GV, XX, XZ, unif, count, r);
// Transfer A and G data into a buffer so that they can be broadcasted:
for(i=0; i < k; i++){
      A_G_array[i] = gsl_vector_get(A, i);
      A_G_array[i+k] = gsl_vector_get(G, i);}
}
  MPI_Bcast(A_G_array, 2*k, MPI_DOUBLE, ROOT, MPI_COMM_WORLD);
// Transfer A and G received to a vector structure:
  for(i=0; i < k; i++){
  gsl_vector_set(A,i,A_G_array[i]);
  gsl_vector_set(G,i,A_G_array[i+k]);
  }
}// End iteration
```

Pseudocode 1

Hence, given that the algorithm with seven nodes involved reduced communication overhead and computation time, it resulted in a much improved speedup and efficiency for Experiments 1 and 4.

In addition, as we noted earlier, the communication size for this approach depends on k, not n. Hence, although Experiment 2 (where $n = 5000$) had the largest input size ($5000 \times 50 = 250{,}000$), its overhead was close to, or even smaller than, Experiments 3 and 4 (where $n = 2000$) with the use of the parallel algorithm (see Figure 6). But due to the reason of the large matrix size, the time used for each slave node to calculate their blocks of $\mathbf{Z}$ and $\boldsymbol{\theta}$ increased, and consequently Experiment 2 had only a slight advantage in speedup and efficiency than the other experiments. To further understand this, we can also look at the time for completing each task within one iteration of the Markov chain for Experiments 2 and 3 (see Figures 9 and 10). A comparison of Figures 8 to 10 indicates that Experiment 2 involved more calculation due to the larger input size, but in spite of this, it had less communication overhead (e.g., by sending $\mathbf{x}'\mathbf{x}$ and $\mathbf{x}'\mathbf{Z}$ to the root) than Experiment 3.

With both decomposition schemes, the communication overhead increased more than the computational speedup when a certain number of processors are used. As a result, the speedup does not increase with increasing number of processors and, consequently, the cost increases (see Figure 5).

However, a comparison of the two parallel algorithms clearly indicates that row-wise decomposition involves much less computational overhead than column-wise decomposition in all the experimented conditions (see Figure 6) and hence is a more efficient approach in terms of enhancing speedup (Figure 7) and efficiency (Figure 4) while reducing cost (Figure 5).

5. Conclusion

This study developed a high performance Gibbs sampling algorithm for the 2PNO IRT model with the purpose of achieving a shorter execution time possible using a row-wise decomposition scheme. The algorithm was implemented using the ANSI C programming language and the message-passing interface. Experiments were performed to empirically evaluate its performance with varying sample sizes, test lengths, and chain lengths.

Results indicated that the proposed parallel algorithm using row-wise decomposition (for the given problem size) performed much better compared with that using column-wise decomposition. Regarding the number of processing nodes, the algorithm worked relatively better, in terms of efficiency and cost, using two to seven processing nodes. On the other hand, in the experiments with relatively large input matrix sizes (e.g., Experiments 2 and 3), it had the smallest execution time when eight processing nodes were used, which was the largest number of processing nodes used in the experiments. Therefore, given the high data dependencies in the Gibbs sampler for IRT models, such as the dependency of one state of the chain to the previous states, and the dependencies between the data within the same state, it is not possible to completely avoid communications among processors in each iteration of the chain. By changing the domain decomposition scheme from column-wise to row-wise, we managed to reduce the size of the data communicated so that a speedup was achieved. This further sheds light on developing high performance Gibbs sampler for more complicated IRT models. In the IRT literature, the model can be more complex by assuming multiple latent traits, where row-wise decomposition is theoretically more appealing than column-wise decomposition.

This study achieved parallelization through a row-wise decomposition and the use of all-to-one and one-to-all broadcast schemes. Further studies can be undertaken to increase the speedup and the efficiency and minimize the cost and the total overhead. For example, an all-to-all broadcast scheme may be adopted in order to achieve a smaller communication overhead.

Appendix

The pseudocode for updating the values of $\mathbf{Z}$, $\boldsymbol{\theta}$, $\mathbf{x}'\mathbf{x}$, $\mathbf{x}'\mathbf{Z}$, $\boldsymbol{\alpha}$, and $\boldsymbol{\beta}$ is shown in Pseudocode 1. First of all, $\mathbf{Z}$ and $\boldsymbol{\theta}$ are updated through the functions update_Z and update_TH, respectively. Then, update_KSI_AR is called to update $\mathbf{x}'\mathbf{x}$ and $\mathbf{x}'\mathbf{Z}$, and MPI_Gather is called to send $\mathbf{x}'\mathbf{x}$ and $\mathbf{x}'\mathbf{Z}$ to the root. The root receives $\mathbf{x}'\mathbf{x}$ and $\mathbf{x}'\mathbf{Z}$ and calls update_A_G to update $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$. It then broadcasts $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ by calling MPI_Bcast. In order to reduce communication overhead, $\mathbf{x}'\mathbf{x}$ and $\mathbf{x}'\mathbf{Z}$ are sent in the same message. To achieve that, an array of size $2k + 2$ is set up, where the first $2k$ entries consist of the elements of $\mathbf{x}'\mathbf{Z}$ and entries $2k + 1$ and $2k + 2$ consist of elements for $\mathbf{x}'\mathbf{x}$ (the name of this array in the code is KSI_array) (See Pseudocode 1).

References

[1] R. D. Bock and M. Aitkin, "Marginal maximum likelihood estimation of item parameters: application of an EM algorithm," *Psychometrika*, vol. 46, no. 4, pp. 443–459, 1981.

[2] R. J. Mislevy, "Estimation of latent group effects," *Journal of the American Statistical Association*, vol. 80, no. 392, pp. 993–997, 1985.

[3] R. J. Patz and B. W. Junker, "A straightforward approach to markov Chain Monte Carlo Methods for item response models," *Journal of Educational and Behavioral Statistics*, vol. 24, no. 2, pp. 146–178, 1999.

[4] R. K. Tsutakawa and H. Y. Lin, "Bayesian estimation of item response curves," *Psychometrika*, vol. 51, no. 2, pp. 251–267, 1986.

[5] J. Bafumi, A. Gelman, D. K. Park, and N. Kaplan, "Practical issues in implementing and understanding Bayesian ideal point estimation," *Political Analysis*, vol. 13, no. 2, pp. 171–187, 2005.

[6] C. S. Martin, T. Chung, L. Kirisci, and J. W. Langenbucher, "Item response theory analysis of diagnostic criteria for alcohol and cannabis use disorders in adolescents: implications for DSM-V," *Journal of Abnormal Psychology*, vol. 115, no. 4, pp. 807–814, 2006.

[7] U. Feske, L. Kirisci, R. E. Tarter, and P. A. Pilkonis, "An application of item response theory to the DSM-III-R criteria for borderline personality disorder," *Journal of Personality Disorders*, vol. 21, no. 4, pp. 418–433, 2007.

[8] C. L. Beseler, L. A. Taylor, and R. F. Leeman, "An item-response theory analysis of DSM-IV Alcohol-Use disorder criteria and "binge" drinking in undergraduates," *Journal of Studies on Alcohol and Drugs*, vol. 71, no. 3, pp. 418–423, 2010.

[9] D. A. Gilder, I. R. Gizer, and C. L. Ehlers, "Item response theory analysis of binge drinking and its relationship to lifetime alcohol use disorder symptom severity in an American Indian Community sample," *Alcoholism: Clinical and Experimental Research*, vol. 35, no. 5, pp. 984–995, 2011.

[10] A. T. Panter and B. B. Reeve, "Assessing tobacco beliefs among youth using item response theory models," *Drug and Alcohol Dependence*, vol. 68, no. 1, pp. S21–S39, 2002.

[11] D. Courvoisier and J. F. Etter, "Using item response theory to study the convergent and discriminant validity of three questionnaires measuring cigarette dependence," *Psychology of Addictive Behaviors*, vol. 22, no. 3, pp. 391–401, 2008.

[12] J. S. Rose and L. C. Dierker, "An item response theory analysis of nicotine dependence symptoms in recent onset adolescent smokers," *Drug and Alcohol Dependence*, vol. 110, no. 1-2, pp. 70–79, 2010.

[13] S. E. Fienberg, M. S. Johnson, and B. W. Junker, "Classical multilevel and Bayesian approaches to population size estimation using multiple lists," *Journal of the Royal Statistical Society A*, vol. 162, no. 3, pp. 383–405, 1999.

[14] M. Reiser, "An application of the item-response model to psychiatric epidemiology," *Sociological Methods and Research*, vol. 18, no. 1, pp. 66–103, 1989.

[15] M. Orlando, C. D. Sherbourne, and D. Thissen, "Summed-score linking using item response theory: application to depression measurement," *Psychological Assessment*, vol. 12, no. 3, pp. 354–359, 2000.

[16] A. Tsutsumi, N. Iwata, N. Watanabe et al., "Application of item response theory to achieve cross-cultural comparability of occupational stress measurement," *International Journal of Methods in Psychiatric Research*, vol. 18, no. 1, pp. 58–67, 2009.

[17] A. Birnbaum, "Statistical theory for logistic mental test models with a prior distribution of ability," *Journal of Mathematical Psychology*, vol. 6, no. 2, pp. 258–276, 1969.

[18] F. B. Baker and S. H. Kim, *Item Response Theory: Parameter Estimation Techniques*, Dekker, New York, NY, USA, 2nd edition, 2004.

[19] I. W. Molenaar, "Estimation of item parameters," in *Rasch Models: Foundations, Recent Developments, and Applications*, G. H. Fischer and I. W. Molenaar, Eds., pp. 39–51, Springer, New York, NY, USA, 1995.

[20] A. F. M. Smith and G. O. Roberts, "Bayesian computation via the Gibbs sampler and related Markov chain Monte Carlo methods (with discussion)," *Journal of the Royal Statistical Society B*, vol. 55, no. 1, pp. 3–23, 1993.

[21] L. Tierney, "Markov chains for exploring posterior distributions," *The Annals of Statistics*, vol. 22, no. 4, pp. 1701–1728, 1994.

[22] K. Patsias, M. Rahimi, Y. Sheng, and S. Rahimi, "Parallel computing with a Bayesian item response model," *American Journal of Computational Mathematics*, vol. 2, no. 2, pp. 65–71, 2012.

[23] S. Geman and D. Geman, "Stochastic relaxation, Gibbs distributions, and the Bayeisan restoration of images," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 6, no. 6, pp. 721–741, 1984.

[24] F. M. Lord and M. R. Novick, *Statistical Theories of Mental Test Scores*, Addison-Wesley, Boston, Mass, USA, 1968.

[25] J. H. Albert, "Bayesian estimation of normal ogive item response curves using Gibbs sampling," *Journal of Educational Statistics*, vol. 17, no. 3, pp. 251–269, 1992.

[26] Y. Sheng and T. C. Headrick, "An algorithm for implementing Gibbs sampling for 2PNO IRT models," *Journal of Modern Applied Statistical Methods*, vol. 6, no. 1, pp. 341–349, 2007.

[27] G. Casella and E. I. George, "Explaining the Gibbs sampler," *The American Statistician*, vol. 46, no. 3, pp. 167–174, 1992.

[28] M. Galassi, J. Davies, J. Theiler et al., *GNU Scientific Library Reference Manual*, Network Theory, Bristol, UK, 3rd edition, 2009.

[29] I. Foster, *Designing and Building Parallel Programs: Concepts and Tools for Parallel Software Engineering*, Addison-Wesley, Boston, Mass, USA, 1995.