frontiers in Genetics

METHODS
published: 14 August 2018
doi: 10.3389/fgene.2018.00226

PIBLUP: High-Performance Software for Large-Scale Genetic Evaluation of Animals and Plants

Huimin Kang[1,2,3], *Chao Ning*[1,2,3], *Lei Zhou*[1,2,3], *Shengli Zhang*[1,2,3], *Ning Yang*[1,2,3] *and Jian-Feng Liu*[1,2,3]*

[1] National Engineering Laboratory for Animal Breeding, China Agricultural University, Beijing, China, [2] Key Laboratory of Animal Genetics, Breeding and Reproduction, Ministry of Agriculture, China Agricultural University, Beijing, China, [3] Department of Animal Genetics, Breeding & Reproduction, College of Animal Science and Technology, China Agricultural University, Beijing, China

Today, the rapid increase in phenotypic and genotypic information is leading to larger mixed model equations (MMEs) and rendering genetic evaluation more time-consuming. It has been demonstrated that a preconditioned conjugate gradient (PCG) algorithm via an iteration on data (IOD) technique is the most efficient method of solving MME at a low computing cost. Commonly used software applications implementing PCG by IOD merely employ functions from the Intel Math Kernel Library (MKL) to accelerate numerical computations and have not taken full advantage of the multicores or multiprocessors of computer systems to reduce the execution time. Making the most of multicore/multiprocessor systems, we propose PIBLUP, a parallel, shared memory implementation of PCG by IOD to minimize the execution time of genetic evaluation. In addition to functions in MKL, PIBLUP uses Message Passing Interface (MPI) shared memory programming to parallelize code in the entire workflow where possible. Results from the analysis of the two datasets show that the execution time was reduced by more than 80% when solving MME using PIBLUP with 16 processes in parallel, compared to a serial program using a single process. PIBLUP is a high-performance tool for users to efficiently perform genetic evaluation. PIBLUP with its user manual is available at https://github.com/huiminkang/PIBLUP.

Keywords: genetic evaluation, Message Passing Interface, Intel Math Kernel Library, preconditioned conjugate gradient, iteration on data, large-scale, multicore/multiprocessor

OPEN ACCESS

Edited by:
Shrikant S. Mantri, National Agri-Food Biotechnology Institute, India

Reviewed by:
Patrik Waldmann, Swedish University of Agricultural Sciences, Sweden
Rui Alves, Universitat de Lleida, Spain

***Correspondence:**
Jian-Feng Liu
liujf@cau.edu.cn

Specialty section:
This article was submitted to Bioinformatics and Computational Biology, a section of the journal Frontiers in Genetics

Received: *14 November 2017*
Accepted: *07 June 2018*
Published: *14 August 2018*

Citation:
Kang H, Ning C, Zhou L, Zhang S, Yang N and Liu J-F (2018) PIBLUP: High-Performance Software for Large-Scale Genetic Evaluation of Animals and Plants. Front. Genet. 9:226. doi: 10.3389/fgene.2018.00226

INTRODUCTION

Genetic evaluation is widely accepted as an efficient method to improve genetically complex traits and is increasingly applied to the breeding programs of animals and plants. Solving mixed model equations (MMEs) is a major part of genetic evaluation. Statistical models used in genetic evaluations are usually complicated, which include either multiple correlated effects within the same trait or multiple traits analyzed simultaneously. An example of the former is random regressions nested within an additive genetic effect or within a permanent environmental effect in test-day models for milk production traits (Schaeffer et al., 2000). Examples of the latter are multiple-trait analyses of type traits (Tsuruta et al., 2011), fertility traits (Liu et al., 2008), and a multiple-trait multiple-lactation

test-day model for milk production traits (Schaeffer et al., 2000). Moreover, with genomic selection, the relationship matrix based on markers becomes denser than that based on pedigree. All the aspects mentioned above make solving MMEs a time-consuming task. In animal breeding, large MMEs are usually solved using a preconditioned conjugate gradient (PCG) by iteration on data (IOD). PCG is easy to implement and has a superior convergence rate compared to other commonly used iterative methods, such as the Gauss–Seidel method (Tsuruta et al., 2001) and the Gauss–Seidel second-order Jacobi method (Strandén and Lidauer, 1999). Incorporating the IOD technique into the PCG algorithm avoids the high memory consumption of storing a large coefficient matrix of MME and enables the implementation of large-scale genetic evaluation (Strandén and Lidauer, 1999).

With the accumulation of phenotypic and genomic data, current genetic evaluations are more time consuming than those that existed before the genomic era, especially in situations where the aforementioned complicated models are employed. Parallel computing is a practically workable solution to this problem in which a computational task is typically divided into several similar, independent subtasks whose results are combined afterwards.

The Intel Math Kernel Library (MKL) (https://software.intel.com/en-us/intel-mkl) provides optimized and threaded math functions to obtain immediate performance benefits. Among the currently available software packages, as far as we know, BLUPF90 (Misztal et al., 2002) and DMU (Madsen and Jensen, 2013) employed MKL to speed up computations.

When the entire MME system is stored in main memory, almost all the steps of the PCG algorithm can be parallelized using math functions in the MKL library. However, with the implementation of an IOD technique, one of the most time-consuming parts can no longer be parallelized with MKL. In this case, Message Passing Interface (MPI) (MPI Forum, 2012) provides a good way to implement parallel programming with the computational load manually partitioned across multiple processes.

Taking full advantage of multicore/multiprocessor systems, we present PIBLUP, a C implementation of PCG by IOD using both MPI parallel programming and MKL math functions in order to reduce the execution time of genetic evaluations and facilitate large-scale analyses. PIBLUP with its user manual is available at https://github.com/huiminkang/PIBLUP.

METHODS

PIBLUP is written in C and is supported on a Linux operating system with a shared memory architecture. PIBLUP is driven by a parameter file, where the MME information is specified in sections defined by keywords (see examples in user manual at https://github.com/huiminkang/PIBLUP).

The parallel version of PIBLUP is programmed based on an optimized serial version. The workflows of these two versions (shown in **Figure 1**) are the same, except that MPI parallel programming was not used in the serial version. Details of the workflow are explained as follows.

Statistical Model Construction

Based on the specified parameter file, PIBLUP constructs the statistical model used in genetic evaluation. Then, (co)variance matrices for random effects are inverted to facilitate the following computations. To be more user-friendly, PIBLUP renumbers categorical variables in data files, such as phenotype file, pedigree, and genotype file, permitting the use of raw data including alpha-numeric characters in input files.

Subsequently, inverses of kinship matrices are constructed, including the inverse of a numerator relationship matrix (**A**) based on pedigree (Henderson, 1975, 1976), the inverse of a genomic relationship matrix (**G**) based on genomic information (VanRaden, 2008) and the inverse of a hybrid relationship matrix (**H**) of **A** and **G** (Legarra et al., 2009; Christensen and Lund, 2010). In addition to these commonly used matrices, PIBLUP can leverage user-specified kinship matrices (their inverse matrices stored in input files) alternatively.

In PIBLUP, the **G** matrix is constructed utilizing the first method of VanRaden (2008) as follows:

$$\mathbf{G} = \frac{(\mathbf{S}-\mathbf{P})(\mathbf{S}-\mathbf{P})'}{2\sum_{j=1}^{n} p_j(1-p_j)}, \quad (1)$$

where **S** is a matrix of SNP genotypes for each individual, **P** is a matrix of two times the observed allele frequency of the second allele p at locus j (p_j), and n is the number of SNPs used in the construction of **G**. In PIBLUP, the **G** matrix is explicitly inverted. PIBLUP has two options to ensure that **G** is invertible. First, a small arbitrary value (specified in the parameter file, e.g., 0.01) can be added to the diagonals of **G**. Moreover, the nearest positive definite matrix of **G** could be computed using Higham's method (Higham, 2002) (specified in the parameter file).

The inverse of the **H** matrix in PIBLUP is constructed as follows (Legarra et al., 2009; Christensen and Lund, 2010):

$$\mathbf{H}^{-1} = \mathbf{A}^{-1} + \begin{bmatrix} \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \tau[(1-w)\,\mathbf{G}^{*} + w\mathbf{A}_{22}]^{-1} - \omega\mathbf{A}_{22}^{-1} \end{bmatrix}, \quad (2)$$

where $\mathbf{G}^{*} = \beta\,\mathbf{G} + \alpha$ with α and β are calculated from the system of equations:

$$\begin{cases} \mathrm{Avg}\left(\mathrm{diag}\left(\mathbf{G}\right)\right)\beta + \alpha = \mathrm{Avg}\left(\mathrm{diag}\left(\mathbf{A}_{22}\right)\right) \\ \mathrm{Avg}\left(\mathrm{offdiag}\left(\mathbf{G}\right)\right)\beta + \alpha = \mathrm{Avg}\left(\mathrm{offdiag}\left(\mathbf{A}_{22}\right)\right) \end{cases}, \quad (3)$$

where $\mathrm{Avg}\left(\mathrm{diag}\left(\mathbf{B}\right)\right)$ and $\mathrm{Avg}\left(\mathrm{offdiag}\left(\mathbf{B}\right)\right)$ represent the average values of diagonal and non-diagonal elements of matrix **B**, respectively (Christensen et al., 2012). Matrix $\mathbf{A}_{22}$ is the partition of **A** and subscript 2 represents genotyped individuals. The adjustment of **G** is to avoid the potential incompatibility in scale between **G** and $\mathbf{A}_{22}$. Three parameters τ, w, and ω, whose values are specified by users, are set to achieve better accuracy and lower bias by fine-tuning (Tsuruta et al., 2011; Harris et al., 2012).

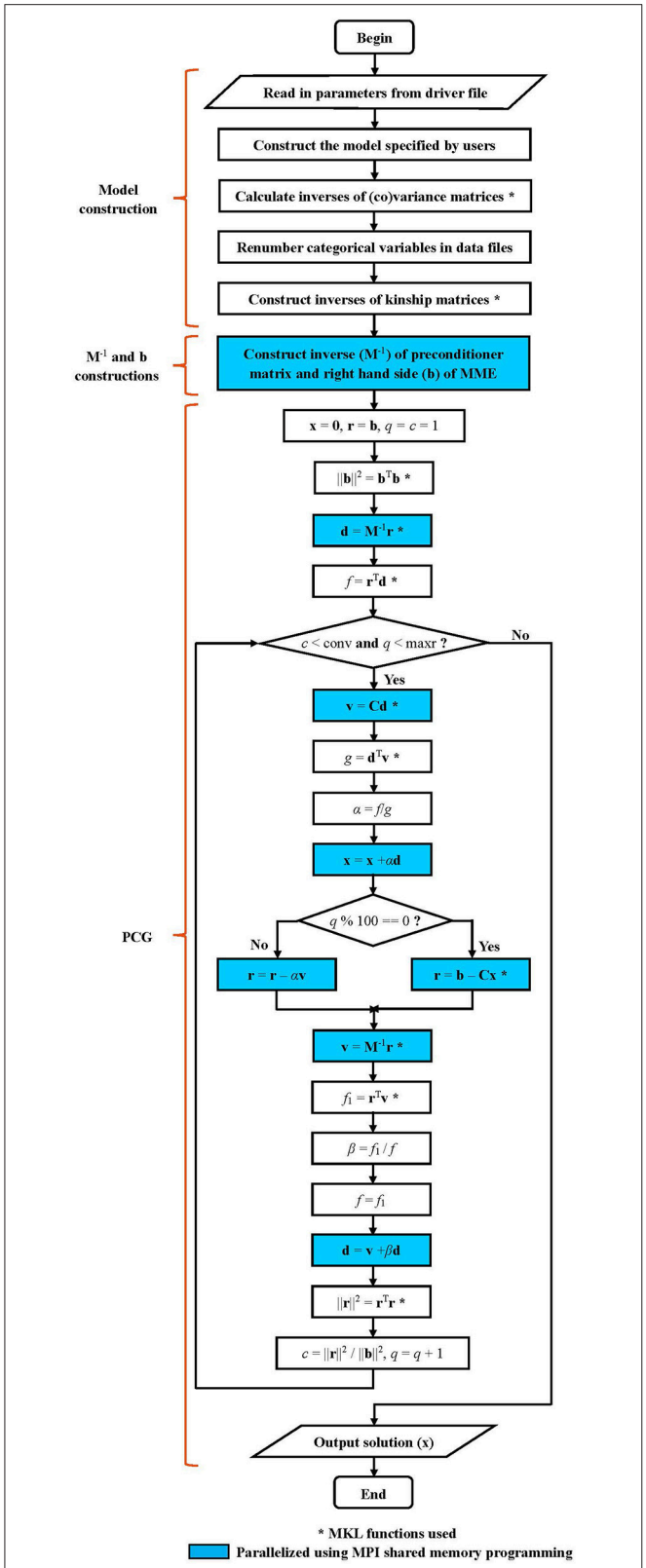

FIGURE 1 | Workflow diagram for PIBLUP. The mixed model equation to be solved is $\mathbf{Cx} = \mathbf{b}$. The preconditioner matrix is $\mathbf{M}$, a block diagonal matrix formed from the coefficient matrix $\mathbf{C}$. Implementation of steps marked by "*" employs math functions in Intel Math Kernel Library. Code in parts of $\mathbf{M}^{-1}$ and $\mathbf{b}$ constructions and PCG are parallelized using Message Passing Interface shared memory programming. Steps in cyan color are executed by all the processes to reduce runtime, and those in white are executed by the master process. All the processes execute the two conditional statements in diamonds.

Preconditioner Construction in Parallel

Let the MME be:

$$\begin{bmatrix} \mathbf{X}'\mathbf{R}^{-1}\mathbf{X} & \mathbf{X}'\mathbf{R}^{-1}\mathbf{Z} \\ \mathbf{Z}'\mathbf{R}^{-1}\mathbf{X} & \mathbf{Z}'\mathbf{R}^{-1}\mathbf{Z} + \mathbf{G}^{-1} \otimes \mathbf{G}_0^{-1} \end{bmatrix} \begin{bmatrix} \widehat{\mathbf{f}} \\ \widehat{\mathbf{u}} \end{bmatrix} = \begin{bmatrix} \mathbf{X}'\mathbf{R}^{-1}\mathbf{y} \\ \mathbf{Z}'\mathbf{R}^{-1}\mathbf{y} \end{bmatrix}. \quad (4)$$

The coefficient matrix can be partitioned into two parts:

$$\begin{bmatrix} \mathbf{X}'\mathbf{R}^{-1}\mathbf{X} & \mathbf{X}'\mathbf{R}^{-1}\mathbf{Z} \\ \mathbf{Z}'\mathbf{R}^{-1}\mathbf{X} & \mathbf{Z}'\mathbf{R}^{-1}\mathbf{Z} + \mathbf{G}^{-1} \otimes \mathbf{G}_0^{-1} \end{bmatrix} = \begin{bmatrix} \mathbf{X}'\mathbf{R}^{-1}\mathbf{X} & \mathbf{X}'\mathbf{R}^{-1}\mathbf{Z} \\ \mathbf{Z}'\mathbf{R}^{-1}\mathbf{X} & \mathbf{Z}'\mathbf{R}^{-1}\mathbf{Z} \end{bmatrix} + \begin{bmatrix} \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{G}^{-1} \otimes \mathbf{G}_0^{-1} \end{bmatrix}, \quad (5)$$

where the first part, the least square part, is constructed based on observations and the second part is based on the inverse of kinship matrix $\mathbf{G}^{-1}$ and the inverse of the (co)variance matrix for the random effect $\mathbf{G}_0^{-1}$. $\mathbf{X}$ and $\mathbf{Z}$ are the design matrices of fixed effects $\mathbf{f}$ and random effects $\mathbf{u}$ respectively, $\mathbf{R}$ is the residual (co)variance matrix, and $\otimes$ is the Kronecker product. If the random effect is not a genetic effect, $\mathbf{G}^{-1}$equals the identity matrix $\mathbf{I}$.

A block diagonal preconditioner $\mathbf{M}$ based on the coefficient matrix is used in PIBLUP to improve the convergence rate of PCG. The blocks of $\mathbf{M}$ are due to traits, resulting in dense blocks of the $t \times t$ matrix, where t is the number of traits. Both $\mathbf{M}$ and $\mathbf{M}^{-1}$ are constructed using MPI shared memory programming.

During the construction of $\mathbf{M}$, PIBLUP stores the upper triangles of blocks in memory. A region of shared memory is allocated for $\mathbf{M}$ by a master process. All processes can access it without communication via a network. According to the partition of the coefficient matrix, the construction of $\mathbf{M}$ involves two tasks. All processes participate in both tasks. In the first task, the translated values are added to their corresponding positions based on each phenotypic record. All the processes read the phenotype file simultaneously, but each process calculates the addend and performs addition operations for specific positions. PIBLUP assigns at least one position to each process and tries to ensure an even distribution of the number of addition operations. The second task is adding values to $\mathbf{M}$ for each level of random effects. PIBLUP distributes work among processes according to the total number of levels and the number of processes.

After $\mathbf{M}$ has been constructed, its inverse is computed by inverting each diagonal block it contains. This is also implemented by all of the processes in parallel, with each process computing inverses for nearly an equal number of blocks. The inverse is calculated using the MKL math functions

LAPACKE_dgetrf and LAPACKE_dgetri. Finally, $\mathbf{M}^{-1}$ is stored in disk.

While phenotypic records are processed in the construction of **M**, all processes construct the right-hand side (**b**) of MME in a similar manner in parallel. Specifically, **b** is stored in the region of shared memory allocated by the master process. All processes participate in adding translated values to their corresponding positions in **b** based on phenotypic records. Processes handle the same positions that were assigned in the construction of the least square part of **M**.

Implementation of PCG by IOD in Parallel

As shown in **Figure 1**, PIBLUP begins to solve MME using PCG by IOD after $\mathbf{M}^{-1}$ and **b** have been constructed. The parallelization of this process is realized using MPI shared memory programming and MKL functions (indicated by "*" in **Figure 1**). All the processes execute the steps in blue in **Figure 1** in parallel to accelerate the computational speed.

With an IOD technique (Strandén and Lidauer, 1999), the coefficient matrix of MME (**C**) does not require to be stored in memory. Therefore, the memory consumption is dominated by five vectors, i.e., right-hand side vector **b**, residual vector **r**, search direction vector **d**, solution vector **x**, and vector **v**. With MPI shared memory model, these five vectors are stored in the shared memory of the master process and can be accessed by all processes. Each process only need to allocate an area of shared memory to store the intermediate vector **i**, which is used in the calculations of **Cd** and **Cx**.

The master process computes the inner products, i.e., $\mathbf{b}^T\mathbf{b}$, $\mathbf{r}^T\mathbf{d}$, $\mathbf{d}^T\mathbf{v}$, $\mathbf{r}^T\mathbf{v}$, and $\mathbf{r}^T\mathbf{r}$, using the MKL function cblas_ddot. All the processes participate in the computation of $\mathbf{M}^{-1}\mathbf{r}$, using the MKL function cblas_dsymv. As $\mathbf{M}^{-1}$ is a block diagonal matrix, each process performs an equal number of multiplications of blocks and vectors. $\mathbf{M}^{-1}$is read from disk each time.

The heaviest computation step in PCG is the multiplication of the coefficient matrix (**C**) and vector **d** or **x**. As shown in Equation [5], the **C** matrix can be partitioned into two parts. After being read from disk, the *j*th phenotypic record is translated into $\mathbf{x}_j$ and $\mathbf{z}_j$. Therefore,

$$\begin{aligned}\mathbf{Cd} &= \begin{bmatrix}\mathbf{X}'\mathbf{R}^{-1}\mathbf{X} & \mathbf{X}'\mathbf{R}^{-1}\mathbf{Z}\\ \mathbf{Z}'\mathbf{R}^{-1}\mathbf{X} & \mathbf{Z}'\mathbf{R}^{-1}\mathbf{Z}\end{bmatrix}\mathbf{d} + \begin{bmatrix}\mathbf{0} & \mathbf{0}\\ \mathbf{0} & \mathbf{G}^{-1}\otimes\mathbf{G}_0^{-1}\end{bmatrix}\mathbf{d}\\ &= \sum_{j=1}^{q}\mathbf{w}_j\mathbf{R}_j^{-1}\mathbf{w}'_j\mathbf{d} + \begin{bmatrix}\mathbf{0} & \mathbf{0}\\ \mathbf{0} & \left(\mathbf{G}^{-1}\otimes\mathbf{I}_{\mathbf{G}_0^{-1}}\right)\left(\mathbf{I}_{\mathbf{G}^{-1}}\otimes\mathbf{G}_0^{-1}\right)\end{bmatrix}\mathbf{d},\end{aligned} \tag{6}$$

where $\mathbf{w}'_j = \left[\mathbf{x}'_j\ \mathbf{z}'_j\right]$, q is the number of records, and $\mathbf{I}_{\mathbf{G}_0^{-1}}$ and $\mathbf{I}_{\mathbf{G}^{-1}}$ are identity matrices of orders equal the orders of $\mathbf{G}_0^{-1}$ and $\mathbf{G}^{-1}$, respectively. The products $\mathbf{t}_j = \mathbf{w}_j\mathbf{R}_j^{-1}\mathbf{w}'_j\mathbf{d}$ and $\mathbf{s} = \left(\mathbf{G}^{-1}\otimes\mathbf{I}_{\mathbf{G}_0^{-1}}\right)\left(\mathbf{I}_{\mathbf{G}^{-1}}\otimes\mathbf{G}_0^{-1}\right)\mathbf{d}$ are both calculated from the right to the left (Strandén and Lidauer, 1999), i.e.,

$$\mathbf{t}_{j1} = \mathbf{w}'_j\mathbf{d};\ \mathbf{t}_{j2} = \mathbf{R}_j^{-1}\mathbf{t}_{j1};\ \mathbf{t}_j = \mathbf{w}_j\mathbf{t}_{j2}, \tag{7}$$

and

$$\mathbf{s}_1 = \left(\mathbf{I}_{\mathbf{G}^{-1}}\otimes\mathbf{G}_0^{-1}\right)\mathbf{d};\ \mathbf{s} = \left(\mathbf{G}^{-1}\otimes\mathbf{I}_{\mathbf{G}_0^{-1}}\right)\mathbf{s}_1. \tag{8}$$

As phenotypic records are independent of each other, all processes read nearly equal numbers of records and calculate the corresponding $\mathbf{t}_j$. Similarly, the computations of $\mathbf{s}_1$ and **s** are assigned to processes according to the number of levels of random effects and the number of non-zero elements in $\mathbf{G}^{-1}$, respectively.

In each process, the values of vectors $\mathbf{t}_j$ and $\mathbf{s}_1$ are stored in the intermediate vector **i**. Then, vector **i** for $\mathbf{t}_j$ is summed up across processes and stored in vector **v** in the master process. The value of **i** for **s** from each process is directly added to **v** in the master process, as different processes cannot have concurrent addition operations on the same position in **v**.

Convergence Criterion

The convergence criterion used in PIBLUP is the relative average difference between the right-hand side and left-hand side as in the study of Tsuruta et al. (2001):

$$c = \frac{\|\mathbf{b} - \mathbf{Cx}\|^2}{\|\mathbf{b}\|^2}, \tag{9}$$

where $\|\mathbf{y}\|$ is the length of vector **y** and $\|\mathbf{y}\|^2 = \sum_i y_i^2$. The **b**–**Cx** in the numerator is approximated by $\mathbf{r}^k = \mathbf{r}^{k-1} - \alpha\mathbf{v}$, where k means the kth round. The residual vector **r** is updated using the exact formula **r** = **b** – **Cx** every 100 rounds to eliminate accumulated rounding errors.

Application

We evaluated the performance of PIBLUP using two datasets of different sizes, and two commonly used statistical models in genomic selection were employed.

In the first dataset, there were 7,556 individuals with 72,507 markers genotyped. A total of 5,334 individuals among them have de-regressed proofs (DRPs) and corresponding weights (Garrick et al., 2009) for at most three traits. We used multi-trait genomic best linear unbiased prediction (GBLUP) method (VanRaden, 2008) to predict genomic estimated breeding values (GEBV):

$$\mathbf{y} = \mu\mathbf{1}_n + \mathbf{Zg} + \mathbf{e}, \tag{10}$$

where **y** is a vector of DRP of three traits, μ is the overall mean, $\mathbf{1}_n$ is a vector of n ones, **g** is the vector of additive genomic effects with a distribution of $N(\mathbf{0}, \mathbf{G}\otimes\mathbf{G}_0)$, **Z** is the corresponding incidence matrix, and **e** is the vector of random residuals with a distribution of $N(\mathbf{0}, \mathbf{D}\otimes\mathbf{R})$. **G** is a genomic relationship matrix. **D** is a diagonal matrix with diagonal elements $\frac{1}{weight}$. $\mathbf{G}_0$ and **R** were the (co)variance matrices for additive genomic effects and residuals, respectively.

The second dataset contained test-day records of three milk production traits in the first three lactations of dairy

cattle. There were 3,571,661 individuals in pedigree and 7,847,612 test-day records in the phenotype file. Genotypic data from the first dataset were included in this dataset. The statistical model used was the multiple-trait multiple-lactation single-step random regression model (Kang et al., 2016):

$$\mathbf{y} = \mathbf{X}_1\mathbf{b}_1 + \mathbf{X}_2\mathbf{b}_2 + \mathbf{Qa} + \mathbf{Zp} + \mathbf{e}, \tag{11}$$

where $\mathbf{y}$ is the vector of observations, $\mathbf{b}_1$ is the vector of the herd-test date-parity effect, $\mathbf{b}_2$ is the vector of fixed regressions for calving age-season-parity effect, $\mathbf{a}$ and $\mathbf{p}$ are vectors of random regressions for additive genetic effect and permanent environmental effect, $\mathbf{X}_1$, $\mathbf{X}_2$, $\mathbf{Q}$, and $\mathbf{Z}$ are design matrices of $\mathbf{b}_1$, $\mathbf{b}_2$, $\mathbf{a}$, and $\mathbf{p}$, respectively; and $\mathbf{e}$ is the vector of residuals. We adopted Legendre polynomials of the third order for random regressions for permanent environmental effect and Legendre polynomials of the fourth order for fixed and additive genetic regressions. It was assumed that

$$\text{var}\begin{bmatrix}\mathbf{a}\\ \mathbf{p}\\ \mathbf{e}\end{bmatrix} = \begin{bmatrix}\mathbf{H}\otimes\mathbf{G}_0 & \mathbf{0} & \mathbf{0}\\ \mathbf{0} & \mathbf{I}\otimes\mathbf{P} & \mathbf{0}\\ \mathbf{0} & \mathbf{0} & \mathbf{R}\end{bmatrix}, \tag{12}$$

where $\mathbf{G}_0$ (45×45) and $\mathbf{P}$ (36×36) are (co)variance matrices of additive genetic and permanent environmental regression coefficients, and $\mathbf{H}$ is the aforementioned hybrid relationship matrix of $\mathbf{A}$ and $\mathbf{G}$, with $\tau = 1.6$, $w = 0.1$, and $\omega = 0.5$. The lactation was divided into four periods, and residual (co)variance matrices $\mathbf{R}$ were assumed to be the same within each period.

Tests were performed on a Linux server, with a shared memory architecture and a total memory size of about 529 GB. It had 64-bit Intel Xeon E7-4820 processors, each with a base frequency of 2.00 GHz.

RESULTS AND DISCUSSION

The primary motivation for developing PIBLUP lies in reducing the runtime of genetic evaluation by using several concurrent processes. We have compared the EBV or GEBV predicted using PIBLUP and DMU in analyses of all the datasets we obtained. The correlation between results of the two software packages was 1.00 in all the cases. **Table 1** shows the runtime of the serial and parallel versions of PIBLUP in the analyses of the two datasets, for three different parts in **Figure 1**. For each part of PIBLUP, the program using four processes was faster than either program that run with one process, except during the construction of $\mathbf{M}^{-1}$ and $\mathbf{b}$ for dataset 1. This exception occurred owing to the different speeds of file management functions used in the serial version and the parallel version. The effect was significant when the total runtime was short in constructions of $\mathbf{M}^{-1}$ and $\mathbf{b}$ for dataset 1 (0.09 min). Although the part of the model construction in PIBLUP merely employed MKL math functions to accelerate the speed of the matrix inversion (**Figure 1**), the time saved was about 60 and 40% for datasets 1 and 2, respectively. This can be explained by the fact that the computational load in this part was mainly due to the inversion of $\mathbf{G}$ matrix, which is reflected by the time spent in the kinship matrix inversion step accounting for about 90 and 66% of the total runtime for datasets 1 and 2, respectively. Solving MME by PCG is the most time-consuming part, as many rounds of iterations are required to reach convergence in practical applications. With the convergence criterion of 10^{-13}, the rounds of iterations required were 2,252 and 628 for datasets 1 and 2, respectively. For this part, the parallel version with four processes used 31 and 49% of the runtime of the serial program for datasets 1 and 2, respectively. One reason why the ideal percentage of 25% was not achieved was due to some existing overhead, such as synchronization. Moreover, for dataset 2, the number of equations in MME during the analysis was 185 million, which is significantly greater than that in dataset 1. Therefore, the effect of the aforementioned different file input/output speeds of the serial and parallel versions of PIBLUP was more significant in the analysis of dataset 2, as more data were read from disk.

In genetic and genomic evaluations, BLUPF90 (Misztal et al., 2002) and DMU (Madsen and Jensen, 2013) are the two most commonly used software packages. As the blupf90 program from BLUPF90 package and DMU4 program from DMU package are well developed and freely available, we compared the total runtime of PIBLUP (parallel version) with these two programs in the analysis of dataset 1. **Table 2** shows the total runtime of different software packages run with a single process and four processes. With the maximum number of processes set to one, the runtime of PIBLUP, blupf90 and DMU4 was about 51,

TABLE 1 | Runtime (min) of different parts (model construction, $\mathbf{M}^{-1}$ and $\mathbf{b}$ constructions, and PCG) in PIBLUP in analyses of datasets 1 and 2.

Program[a]	Dataset 1[b]			Dataset 2[c]		
	Model construction	$\mathbf{M}^{-1}$ and b constructions	PCG[d]	Model construction	$\mathbf{M}^{-1}$ and b constructions	PCG[d]
Serial	21.97	0.09	0.61	36.56	14.45	87.21
Parallel (1)	21.97	1.56	0.62	36.56	17.36	98.76
Parallel (4)	8.63	0.41	0.19	22.57	5.47	42.61

[a] *Serial version (Serial), parallel version of PIBLUP with a single process (Parallel (1)) and four processes (Parallel (4)) were tested.*
[b] *There were 22,671 equations in mixed model equations for analysis of dataset 1.*
[c] *There were 185,048,637 equations in mixed model equations for analysis of dataset 2.*
[d] *PCG was iterated 50 rounds for easy comparison.*

TABLE 2 | Runtime (min) of PIBLUP, BLUPF90 and DMU in the analysis of dataset 1.

No. of processes	PIBLUP[a]	BLUPF90[b]	DMU[c]
1	51.49	84.97	58.18
4	17.61	49.87	38.79

[a] *Parallel version of PIBLUP.*
[b] *The blupf90 program from BLUPF90 package was employed.*
[c] *The DMU4 program from DMU package was employed.*

85, and 58 min, respectively. According to the user manual of BLUPF90, PCG is the default solver in blupf90 and the fastest one. We chose PCG as the solver when blupf90 was employed. For dense MME, as those in the analysis of dataset 1, DMU4 uses LAPACK subroutines to reach a solution. Therefore, the different runtime of the three software packages may result from the different solving methods they employ. Moreover, as previously mentioned, it is time consuming to construct the inverse of **G** matrix in the analysis of dataset 1. DMU4 saved a larger amount of time because it read in a file containing the inverse of **G** matrix instead of constructing it on its own. In contrast, PIBLUP and blupf90 computed the inverse of **G** matrix based on the genotypes of SNPs. In addition, blupf90 consumed some time to calculate certain statistics of the G matrix and performed some quality control. Considering the factors mentioned above, the differences between the runtime of PIBLUP, blupf90 and DMU4 are acceptable. In contrast to PIBLUP, which uses the IOD technique, blupf90 and DMU4 set up equations in memory. Therefore, blupf90 and DMU4 are not suitable to solve very large MMEs, such as the MME in the analysis of dataset 2.

As presented in **Table 2**, runtime was reduced for all the three programs when the maximum number of processes increased from one to four. The execution time was reduced by approximately 66%, 41%, and 33% for PIBLUP, blupf90 and DMU4, respectively. To use multiple processes in parallel, PIBLUP employs both MPI parallel programming and MKL math functions (**Figure 1**), but blupf90 and DMU4 merely employ MKL math functions. With MPI, PIBLUP can further manually partition computational load across multiple processes (**Figure 1**). Therefore, PIBLUP saved a larger percentage of time compared to blupf90 and DMU4 when four processes were available. As aforementioned, not all steps in the workflow can be parallelized, and computing the inverse of **G** matrix is time consuming. Intel MKL provides parallelized math functions to compute the inverses of matrices. Program using these functions can reduce the runtime efficiently. As DMU4 doesn't construct the inverse of **G** matrix, the percentage of time saved for DMU4 with four processes was less than that for PIBLUP and blupf90.

Figures 2, **3** show that the runtime of each part of PIBLUP decreased as the number of processes increased. For the last two parts of PIBLUP, the execution time was reduced by more than 70% when 16 processes were used in parallel. As shown in **Figure 1**, not all the steps were parallelized, particularly those in the first step. As stated in Amdahl's law (Amdahl, 1967), the sequential steps limiting the runtime can be reduced. Therefore, the percent of runtime saved in the first part was smaller than that of the other two parts. When considering all the parts of

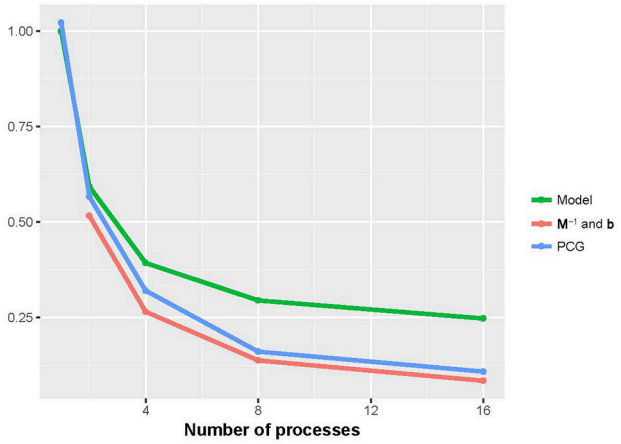

FIGURE 2 | Relative runtime of each part (model construction, $\mathbf{M}^{-1}$ and **b** constructions, and PCG) in parallel version of PIBLUP against runtime of their serial counterparts by the number of processes in analyses of dataset 1. In $\mathbf{M}^{-1}$ and **b** constructions, relative time was calculated against the parallel version using a single process. Model, model construction; $\mathbf{M}^{-1}$ and **b**: $\mathbf{M}^{-1}$ and **b** constructions.

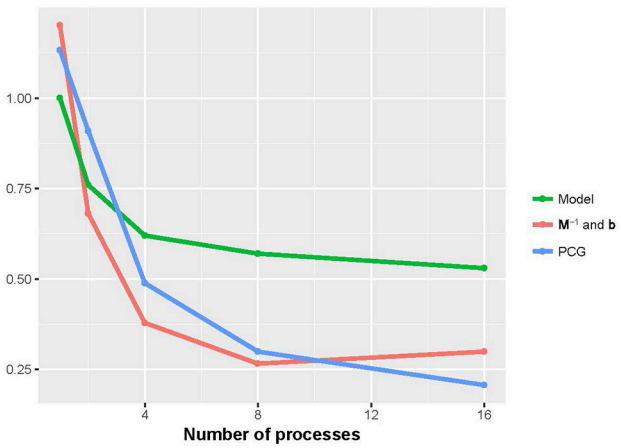

FIGURE 3 | Relative runtime of each part (model construction, $\mathbf{M}^{-1}$ and **b** constructions, and PCG) in parallel version of PIBLUP against runtime of their serial counterparts by the number of processes in analyses of dataset 2. Model, model construction; $\mathbf{M}^{-1}$ and **b**: $\mathbf{M}^{-1}$ and **b** constructions.

PIBLUP, thread managements hindered the execution time from decreasing linearly with the number of processes.

As computer systems with a shared memory architecture are common and the number of cores currently continue to increase, PIBLUP chose to use MPI-3 shared memory programming to parallelize code. Implementations using MPI shared memory programming depend on a shared memory architecture and enable regions of shared memory allocated by one process to be accessed by other processes without communication across a network. Compared to point-to-point communication used in Stranden and Lidauer's study (Strandén and Lidauer, 2001), shared memory programming has the following benefits: (1) time for communication over a network among processes is saved; (2) processes can store less variables in local memory so that

memory consumption is reduced; and (3) programming becomes relatively easy. For programs running in parallel and using shared memory programming, there is no clear method for calculating the actual total memory used by all processes. Therefore, we did not compare the memory usage herein. However, as processes other than the master process only need to store an intermediate vector **i** and other variables of small sizes, the expected total memory usage is relatively low.

AUTHOR CONTRIBUTIONS

HK and J-FL developed software, performed statistical analysis, and wrote the manuscript. CN, LZ, SZ, and NY interpreted the results of analyses and edited the manuscript. J-FL conceived the study, supervised the work, and edited the manuscript.

FUNDING

This work was supported by the National Major Development Program of Transgenic Breeding (2014ZX0800953B), the National High Technology Research and Development Program of China (863 Program 2013AA102503), the National Natural Science Foundations of China (31661143013, 31272419), the Program for Changjiang Scholar and Innovation Research Team in University (IRT_15R62), and Jinxinnong animal science development foundation.

REFERENCES

Amdahl, G. M. (1967). Validity of the single processor approach to achieving large scale computing capabilities. in *Proceedings of the April 18-20, 1967, Spring Joint Computer Conference* (Atlantic, NJ: ACM), 483–485.

Christensen, O. F., and Lund, M. S. (2010). Genomic prediction when some animals are not genotyped. *Genet. Sel. Evol.* 42:2. doi: 10.1186/1297-9686-42-2

Christensen, O. F., Madsen, P., Nielsen, B., Ostersen, T., and Su, G. (2012). Single-step methods for genomic evaluation in pigs. *Animal* 6, 1565–1571. doi: 10.1017/S1751731112000742

Garrick, D. J., Taylor, J. F., and Fernando, R. L. (2009). Deregressing estimated breeding values and weighting information for genomic regression analyses. *Genet. Sel. Evol.* 41:55. doi: 10.1186/1297-9686-41-55

Harris, B. L., Winkelman, A. M., and Johnson, D. L. (2012). Large-scale single-step genomic evaluation for milk production traits. *Interbull. Bull.* 46, 20–24.

Henderson, C. R. (1975). Rapid method for computing the inverse of a relationship matrix. *J. Dairy Sci.* 58, 1727–1730.

Henderson, C. R. (1976). A simple method for computing the inverse of a numerator relationship matrix used in prediction of breeding values. *Biometrics* 32, 69–83.

Higham, N. J. (2002). Computing the nearest correlation matrix - a problem from finance. *Ima J. Numer. Anal.* 22, 329–343. doi: 10.1093/imanum/22.3.329

Kang, H., Zhou, L., Mrode, R., Zhang, Q., and Liu, J. F. (2016). Incorporating single-step strategy into random regression model to enhance genomic prediction of longitudinal trait. *Heredity* 119, 459–467. doi: 10.1038/hdy.2016.91

Legarra, A., Aguilar, I., and Misztal, I. (2009). A relationship matrix including full pedigree and genomic information. *J. Dairy Sci.* 92, 4656–4663. doi: 10.3168/jds.2009-2061

Liu, Z., Jaitner, J., Reinhardt, F., Pasman, E., Rensing, S., and Reents, R. (2008). Genetic evaluation of fertility traits of dairy cattle using a multiple-trait animal model. *J. Dairy Sci.* 91, 4333–4343. doi: 10.3168/jds.2008-1029

Madsen, P., and Jensen, J. (2013). *A User's Guide to DMU, Version 6, Release 5.2*. Tjele: Center for Quantitative Genetics and Genomics; Department of Molecular Biology and Genetics; Aarhus University.

Misztal, I., Tsuruta, S., Strabel, T., Auvray, B., Druet, T., and Lee, D. H. (2002). "BLUPF90 and related programs (BGF90)," in *Proceedings of the 7th World Congress on Genetics Applied to Livestock* Production (Montpellier), 743.

MPI Forum (2012). *MPI: A Message-Passing Interface Standard*. Version 3.0.

Schaeffer, L. R., Jamrozik, J., Kistemaker, G. J., and Van Doormaal, B. J. (2000). Experience with a test-day model. *J. Dairy Sci.* 83, 1135–1144. doi: 10.3168/jds.S0022-0302(00)74979-4

Strandén, I., and Lidauer, M. (1999). Solving large mixed linear models using preconditioned conjugate gradient iteration. *J. Dairy Sci.* 82, 2779–2787.

Strandén, I., and Lidauer, M. (2001). Parallel computing applied to breeding value estimation in dairy cattle. *J. Dairy Sci.* 84, 276–285. doi: 10.3168/jds.S0022-0302(01)74477-3

Tsuruta, S., Misztal, I., Aguilar, I., and Lawlor, T. J. (2011). Multiple-trait genomic evaluation of linear type traits using genomic and phenotypic data in US Holsteins. *J. Dairy Sci.* 94, 4198–4204. doi: 10.3168/jds.2011-4256

Tsuruta, S., Misztal, I., and Strandén, I. (2001). Use of the preconditioned conjugate gradient algorithm as a generic solver for mixed-model equations in animal breeding applications. *J. Anim. Sci.* 79, 1166–1172. doi: 10.2527/2001.7951166x

VanRaden, P. M. (2008). Efficient methods to compute genomic predictions. *J. Dairy Sci.* 91, 4414–4423. doi: 10.3168/jds.2007-0980

Conflict of Interest Statement: The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.