RESEARCH   **Open Access**

Right-hand-side updating for fast computing of genomic breeding values

Mario PL Calus

Abstract

Background: Since both the number of SNPs (single nucleotide polymorphisms) used in genomic prediction and the number of individuals used in training datasets are rapidly increasing, there is an increasing need to improve the efficiency of genomic prediction models in terms of computing time and memory (RAM) required.

Methods: In this paper, two alternative algorithms for genomic prediction are presented that replace the originally suggested residual updating algorithm, without affecting the estimates. The first alternative algorithm continues to use residual updating, but takes advantage of the characteristic that the predictor variables in the model (i.e. the SNP genotypes) take only three different values, and is therefore termed "improved residual updating". The second alternative algorithm, here termed "right-hand-side updating" (RHS-updating), extends the idea of improved residual updating across multiple SNPs. The alternative algorithms can be implemented for a range of different genomic predictions models, including random regression BLUP (best linear unbiased prediction) and most Bayesian genomic prediction models. To test the required computing time and RAM, both alternative algorithms were implemented in a Bayesian stochastic search variable selection model.

Results: Compared to the original algorithm, the improved residual updating algorithm reduced CPU time by 35.3 to 43.3%, without changing memory requirements. The RHS-updating algorithm reduced CPU time by 74.5 to 93.0% and memory requirements by 13.1 to 66.4% compared to the original algorithm.

Conclusions: The presented RHS-updating algorithm provides an interesting alternative to reduce both computing time and memory requirements for a range of genomic prediction models.

Background

Many models have been suggested for genomic prediction (for a review: see [1]). The computing time required to estimate SNP (single nucleotide polymorphism) effects varies considerably between models, e.g. [2]. Computing time depends both on the number of SNPs used and the number of animals in the training dataset. The latter is rapidly increasing, exceeding 15 000 animals in some cases, e.g. [3]. The number of SNPs used is also increasing rapidly with the availability of high-density SNP panels in cattle with 648 874 and 777 962 SNPs [4] and recently, investigations on the use of whole-genome sequence data in genomic prediction have been reported [5,6]. These developments emphasize an increasing need to improve the efficiency of genomic prediction models in terms of computing time and memory requirements. To overcome

computing limitations, some fast implementations have been reported for genomic prediction models such as BayesA [7], BayesB [8,9] and Bayesian Lasso [10]. At the same time, it has been suggested that variable selection methods such as BayesB are required to make optimal use of whole-genome sequence data in genomic prediction [6]. The number of reports that compare the fast implementation of such variable selection methods to the Markov chain Monte Carlo (MCMC) based counterparts has thus far been limited, and all of the aforementioned studies were based on simulated data with a limited number of simulated QTL. To enable the comparison of these fast methods to their MCMC based counterparts in real datasets with whole-genome sequence data, efficient implementations of MCMC genomic prediction models are also required.

Genomic prediction models can be classified into those that involve implicit estimation of SNP effects (using genomic relationships), e.g. [11], and those that involve

Correspondence: mario.calus@wur.nl
Animal Breeding and Genomics Centre, Wageningen UR Livestock Research, 6700 AC Wageningen, The Netherlands

explicit estimation of SNP effects [12]. Genomic prediction models that explicitly estimate SNP effects, commonly perform regression with SNP genotypes as predictor variables [1], coded as 0,1,2 or -1,0,1, referring respectively to the homozygous, heterozygous, and the alternative homozygous genotype. The characteristic that the predictor variables can take only three possible values provides an interesting opportunity to reduce the computing time of algorithms to estimate SNP effects.

The objective of this paper was to describe two efficient algorithms to estimate SNP effects that take advantage of the characteristic that each predictor variable (SNP genotype) can take only three different values. The efficiency of the two algorithms is compared in terms of memory and computing time requirements to that of a commonly used algorithm that is based on residual updating.

Methods
Updating schemes to estimate SNP effects
In general, the efficiency of algorithms to estimate SNP effects can be improved by avoiding redundant computations. The general conditional genomic prediction model to estimate SNP effects for locus *j*, is:

$$\mathbf{y}_j^* = \mathbf{1}\mu + \mathbf{x}_j a_j + \mathbf{e},$$

where $\mathbf{y}_j^*$ is a vector with conditional phenotypes for SNP *j*, $\mathbf{1}$ is a vector of 1's, μ is the overall mean, $\mathbf{x}_j$ is a vector with SNP genotypes at locus *j*, a_j is the allele substitution effect for locus *j*, and $\mathbf{e}$ is a vector of residuals. Note that elements of $\mathbf{x}_j$ could be simply equal to 0, 1, or 2, or take any other value. I.e., elements of $\mathbf{x}_j$ could be scaled and centred, such that they take the following values: $\frac{0-2p_j}{\sqrt{2p_j(1-p_j)}}$, $\frac{1-2p_j}{\sqrt{2p_j(1-p_j)}}$ or $\frac{2-2p_j}{\sqrt{2p_j(1-p_j)}}$, where p_j is the frequency of the allele at locus *j* for which the homozygous genotype is coded as 2. Such scaling of the genotype coding is reported to have some numerical advantages when using MCMC methods [13]. Conditional phenotypes $(\mathbf{y}_j^{*,l+1})$ for SNP *j* in iteration $l+1$ are defined as phenotypes corrected for estimated effects at all other SNP loci, as [14]:

$$\mathbf{y}_j^{*,l+1} = \mathbf{y} - \mathbf{X}_{1:j-1}\hat{\mathbf{a}}_{1:j-1}^{l+1} - \mathbf{X}_{j+1:n}\hat{\mathbf{a}}_{j+1:n}^{l} - \mu, \quad (1)$$

where *n* is the number of SNPs included in the model and $\mathbf{X}$ is a matrix that stores all genotypes. The conditional mean of the allele substitution effect $(\hat{a}_j^{l+1})$ for locus *j* in iteration $l+1$ is obtained as follows:

$$\hat{a}_j^{l+1} = \frac{\mathbf{x}_j'\left(\mathbf{y} - \mathbf{X}_{1:j-1}\hat{\mathbf{a}}_{1:j-1}^{l+1} - \mathbf{X}_{j+1:n}\hat{\mathbf{a}}_{j+1:n}^{l} - \mu\right)}{\mathbf{x}_j'\mathbf{x}_j + \lambda_j}$$
$$= \frac{\mathbf{x}_j'\mathbf{y}_j^{*,l+1}}{\mathbf{x}_j'\mathbf{x}_j + \lambda_j}, \quad (2)$$

where $\lambda_j = \frac{\sigma_e^2}{\sigma_{a_j}^2}$, σ_e^2 is the residual variance, and $\sigma_{a_j}^2$ is the variance associated with locus *j*. Note that $\sigma_{a_j}^2$ in equation (2) can be estimated in several ways, as done in well-known models such as BayesA and BayesB [12], BayesC [15] or Bayesian Stochastic Search Variable Selection [16,17], or can be assumed to be known as in RR-BLUP [12,18]. Note that equation (2) gives the value required in a Gauss-Seidel algorithm to compute BLUP estimates for the allele substitution effects, while it gives the mean of the conditional posterior density if a Bayesian model is used to estimate the allele substitution effects.

Using residual updating, the conditional phenotypes in equation (1) in iteration $l+1$, $\mathbf{y}_j^{*,l+1}$, can be more efficiently computed as [14]:

$$\mathbf{y}_j^{*,l+1} = \mathbf{e}_j^{l+1} + \mathbf{x}_j\hat{\mathbf{a}}_j^l, \quad (3)$$

where $\mathbf{e}_j^{l+1}$ contains the current residuals, i.e.:

$$\mathbf{e}_j^{l+1} = \mathbf{y} - \mathbf{X}_{1:j}\hat{\mathbf{a}}_{1:j}^{l+1} - \mathbf{X}_{j+1:n}\hat{\mathbf{a}}_{j+1:n}^{l} - \mu.$$

Using residual updating, the conditional mean of the allele substitution effect $(\hat{a}_j^{l+1})$ in iteration $l+1$ can then be obtained per locus as follows [14]:

$$\hat{a}_j^{l+1} = \frac{\mathbf{x}_j'\mathbf{e}_j^{l+1} + \mathbf{x}_j'\mathbf{x}_j\hat{a}_j^l}{\mathbf{x}_j'\mathbf{x}_j + \lambda_j}. \quad (4)$$

Considering that, in each iteration, allele substitution effects must be estimated for *n* loci, using phenotypes of *m* individuals, computing all conditional phenotypes using equation (1) requires $mn(n-1)$ multiplications and $mn(n-1)$ subtractions, whereas equation (3) requires only mn multiplications and mn summations. After calculating $\hat{a}_j^{l+1}$, the residual updating step is finalized by updating all residuals such that they can be used to compute conditional phenotypes for SNP $j+1$ [14]:

$$\mathbf{e}_{j+1}^{l+1} = \mathbf{e}_j^{l+1} - \mathbf{x}_j\left(\hat{a}_j^{l+1} - \hat{a}_j^l\right). \quad (5)$$

Hereafter, the algorithm that uses equations (3), (4) and (5) will be referred to as "original residual updating". In the original residual updating algorithm, updating of the residuals and obtaining the sum of cross-products of the residuals and genotypes of each individual are the most time-consuming steps [15]. As indicated by Legarra and Misztal [14], $\mathbf{x}_j'\mathbf{x}_j$ can be calculated once and stored in

memory. As a result, to compute $\hat{a}_j^{l+1}$ using equation (4), $m+1$ multiplications are required. The number of multiplications can be reduced by first summing residuals ($\mathbf{e}_j^{l+1}$) across animals with the same genotypes, and then multiplying each of those three sums by the appropriate genotype. Considering this, equation (4) can be rewritten as:

$$\hat{a}_j^{l+1} = \frac{\boldsymbol{\gamma}_j' \mathbf{f}_j^{l+1} + \mathbf{n}_j' \boldsymbol{\gamma}_j^2 \hat{a}_j^l}{\mathbf{x}_j' \mathbf{x}_j + \lambda_j}, \qquad (6)$$

where the $\boldsymbol{\gamma}_j \left(\boldsymbol{\gamma}_j^2 \right)$ is a vector that contains the (squared) centred and scaled values of the three genotypes that are present at locus j, $\mathbf{f}_j^{l+1}$ is a 3×1 vector that contains the sum of the residuals for each genotype i at locus j, i.e. $\mathbf{f}_j^{l+1} = \sum_i e_{i,j}^{l+1}$, and vector $\mathbf{n}_j$ contains the number of animals for each genotype at locus j. It should be noted here that $\mathbf{n}_j' \boldsymbol{\gamma}_j^2 = \mathbf{x}_j' \mathbf{x}_j$, but the notation $\mathbf{n}_j' \boldsymbol{\gamma}_j^2$ is introduced here to clarify implementation in the newly proposed algorithm, as will be shown later. Equation (6) involves only four multiplications and, thus, requires m-3 fewer multiplications than equation (4) (noting that all values for $\mathbf{n}_j' \boldsymbol{\gamma}_j^2$ can be computed once and stored). Those m-3 multiplications are replaced by m-3 summations that are computationally less demanding than multiplications using standard Fortran functions. Hereafter, the algorithm that uses equations (5) and (6) will be referred to as "improved residual updating".

Further reduction of the number of required computations is possible in the update for locus $j+1$ by using the residual information that was already calculated in the update of locus j:

$$\mathbf{f}_{k,j+1}^{l+1} = \mathbf{h}_{k,j+1}^{l+1} - \mathbf{N}_{j+1,j}' \boldsymbol{\gamma}_j \left(\hat{a}_j^{l+1} - \hat{a}_j^l \right), \qquad (7)$$

where $\mathbf{f}_{k,j+1}^{l+1} = \sum_k e_{k,j+1}^{l+1}$, i.e. a 3×1 vector that contains the sums of the residuals before updating locus $j+1$ for each genotype k at locus $j+1$ in iteration $l+1$, and $\mathbf{N}_{j+1,j}'$ is a 3×3 matrix that contains the number of animals that have any of the nine combinations of genotypes at loci j and $j+1$. Note that $\mathbf{N}_{j+1,j}' \boldsymbol{\gamma}_j$ can be computed once and stored. The term $\mathbf{h}_{k,j+1}^{l+1}$ is a vector that contains the sum of the residuals for each of the three genotypes k at locus $j+1$. Each of these sums is computed from the sums of residuals (before updating) for each of the three genotypes i at locus j that were computed within groups of animals having genotype k at locus $j+1$, i.e. $\mathbf{h}_{k,j+1}^{l+1} = \sum_{k,j+1} \sum_i e_{i,j|k,j+1}^{l+1}$. Thus, for locus j, first $3 \times 3 = 9$ sums of residuals are calculated, one for each unique combination of genotypes at loci j and $j+1$. These sums include the residuals before $\hat{a}_j^{l+1}$ is used to update

them. The update at locus j is accounted for by the term $\mathbf{N}_{j+1,j}' \boldsymbol{\gamma}_j \left(\hat{a}_j^{l+1} - \hat{a}_j^l \right)$.

Implementing equation (7) in (6) yields:

$$\hat{a}_{j+1}^{l+1} = \frac{\boldsymbol{\gamma}_{j+1}' \left(\mathbf{h}_{k,j+1}^{l+1} - \left(\mathbf{N}_{j+1,j}' \boldsymbol{\gamma}_j \left(\hat{a}_j^{l+1} - \hat{a}_j^l \right) \right) \right) + \mathbf{n}_{j+1}' \boldsymbol{\gamma}_{j+1}^2 \hat{a}_{j+1}^l}{\mathbf{x}_{j+1}' \mathbf{x}_{j+1} + \lambda_{j+1}}. \qquad (8)$$

The alternative proposed updating scheme to compute $\hat{a}_{j+1}^{l+1}$ involves applying equation (8) instead of (4) or (6) for locus $j+1$. This updating scheme is hereafter referred to as "right-hand-side updating" (RHS-updating), since it essentially involves direct updating of the right-hand-sides of the model to estimate the SNP effects, without having to explicitly update the residuals every time a SNP effect is estimated.

Instead of explicitly updating residuals after computing the allele substitution effect, the change of the residuals is stored for each possible combination of genotypes at loci j and $j+1$ as:

$$\Delta \mathbf{e}_{j+1,j}^{l+1} = \boldsymbol{\gamma}_{j+1} \mathbf{1}' \left(\hat{a}_{j+1}^{l+1} - \hat{a}_{j+1}^l \right) + \mathbf{1} \boldsymbol{\gamma}_j' \left(\hat{a}_j^{l+1} - \hat{a}_j^l \right), \qquad (9)$$

where $\Delta \mathbf{e}_{j+1,j}^{l+1}$ is a 3×3 matrix that contains updates to the residuals for each combination of genotypes at loci j and $j+1$ after computing the allele substitution effects for those loci, and $\mathbf{1}$ is a vector of 1's, such that both $\boldsymbol{\gamma}_{j+1} \mathbf{1}'$ and $\mathbf{1} \boldsymbol{\gamma}_j'$ are 3×3 matrices. After computing $\hat{a}_j^{l+1}$ and $\hat{a}_{j+1}^{l+1}$, residuals for each combination of genotypes at loci j and $j+1$ can be updated as:

$$\mathbf{e}_{k,j+1;\, i,j}^{l+1,j+2} = \mathbf{e}_{k,j+1;\, i,j}^{l+1,j} - \Delta e_{k,j+1;\, i,j}^{l+1}, \qquad (10)$$

where $\Delta e_{k,j+1;\, i,j}^{l+1}$ is the element in $\Delta \mathbf{e}_{j+1,j}^{l+1}$ that corresponds to genotype k at locus $j+1$ and genotype i at locus j. Applying equations (9) and (10) for locus j and $j+1$, finalizes the RHS-updating step, just like equation (5) finalizes the residual updating step.

A set of SNPs that is consecutively analysed using RHS-updating, is hereafter referred to as a "RHS-block". It should be noted that for the first locus within an RHS-block, here referred to as locus j, there is no dependency on the previously evaluated locus and therefore equation (8) reduces to:

$$\hat{a}_j^{l+1} = \frac{\boldsymbol{\gamma}_j \mathbf{f}_{i,j}^{l+1} + \mathbf{n}_j' \boldsymbol{\gamma}_j^2 \hat{a}_j^l}{\mathbf{x}_j' \mathbf{x}_j + \lambda_j}. \qquad (11)$$

This can be interpreted as the initialization step where, first, the sums of residuals for all nine RHS-group are computed as $\sum_i e_{i,j|k,j+1}^{l+1}$ and, second, the sum of residuals for each genotype at locus j is computed as:
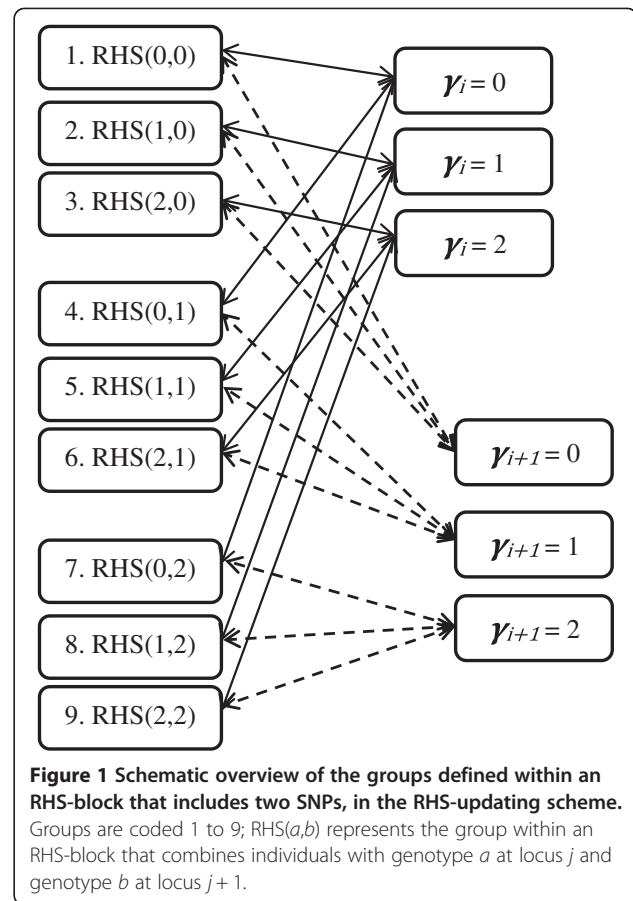
$\mathbf{f}_{i,j}^{l+1} = \sum_i \sum_i e_{i,j|k,j+1}^{l+1}$. Thus, the RHS-updating scheme for "RHS-blocks" of two loci involves the following steps:

1. Apply equation (11) for locus j,
2. Apply equation (8) for locus $j+1$,
3. Apply equation (9) for locus j and $j+1$, and
4. Apply equation (10) to update the residuals.

This implies that the number of operations for locus j is similar for the residual updating and RHS-updating algorithms. However, for locus $j+1$, applying equation (8) requires only 20 summations and subtractions and 11 multiplications, compared to m-3 summations and 4 multiplications that are required when applying (6). This indicates that the total number of operations is drastically reduced by the RHS-updating algorithm.

Consider that for each pair of loci, groups of animals can be identified that have the same combination of genotypes at those two loci. With regard to the RHS-updating scheme, two important points should be noted. First, the groups within RHS-blocks can be coded such that each group code always contains the same genotypes on the first and second SNP. E.g., considering that there are 3^2 groups. At locus $j+1$, groups 1-3 contain genotype 0, groups 4-6 contain genotype 1, and groups 7-9 contain genotype 2. At locus j, groups 1, 4 and 7 contain genotype 0, groups 2, 5 and 8 contain genotype 1, and groups 3, 6 and 9 contain genotype 2. A schematic representation of this group coding within RHS-blocks is in Figure 1. Using such unique coding for the groups within RHS-blocks implies that genotypes do not need to be stored explicitly in memory, since they are stored implicitly through the group numbers. In the RHS-updating algorithm, the array that stores the group codes was stored as integer(2). In the residual updating algorithms, similarly, per locus and individual, the "codes" of the genotypes were stored in an integer(1) array while for each locus the actual values of the three genotypes were stored separately. In fact, the array storing the genotypes for the residual updating algorithm or the group codes for the RHS-updating algorithm are the largest arrays used in those algorithms, and therefore largely determine the total amount of RAM used. For residual updating, this array is of size $m \times n$ and was stored as integer(1) and the amount of RAM used is therefore expected to be proportional to mn. For RHS-updating, this array is of size $m \times n/s$, where s is the number of SNPs per RHS-block and was stored as integer(2). The amount of RAM used with RHS-updating is therefore expected to be proportional to $2mn/s$. These simple formulas, adjusted to predict RAM use in Gb (Table 1), will be compared to empirically measured RAM use.

The second point that should be noted, is that within the RHS-updating scheme, the initialization step described in

Figure 1 Schematic overview of the groups defined within an RHS-block that includes two SNPs, in the RHS-updating scheme. Groups are coded 1 to 9; RHS(a,b) represents the group within an RHS-block that combines individuals with genotype a at locus j and genotype b at locus $j+1$.

equation (10) is the most time-consuming and is recurrent every two loci. By applying the same principles, the RHS-updating scheme can be also applied to more than two loci consecutively. Increasing the number of loci per RHS-block may decrease the relative cost of the initialization step (10), but this will be eventually off-set by the exponential increase in the number of groups that is defined per RHS-block, which is equal to 3^s, where s is the number of SNPs per RHS-block. When the number of loci per RHS-block increases, then at some stage the equivalent expression of equation (7) becomes computationally more demanding. The optimal value to be used for s likely depends on the number of individuals in the training data (m) and will be empirically derived in this study.

Table 1 Formulas to predict RAM requirement for original and improved residual updating, and for RHS-updating

Algorithm	Predicted RAM requirements (Gb)[1]
Original residual updating	$nm \times 10^{-9}$
Improved residual updating	$nm \times 10^{-9}$
RHS-updating	$2\,nm \times 10^{-9}/s$

[1] n = number of loci; m = number of animals; s = number of loci included per RHS-block.

The three algorithms described above, are mathematically equivalent, in the sense that they estimate SNP effects using the same information. Thus, all three algorithms are expected to give the same results.

Implementation of RHS-updating in Bayesian stochastic search variable selection

Model
The above updating schemes to estimate SNP effects were implemented for a model that is commonly referred to as Bayes SSVS (Stochastic Search Variable Selection) [16,17] that is solved using Gibbs sampling and implemented in a computer program written in Fortran 95. The genomic model is generally described as:

$$\mathbf{y} = \mathbf{1}\mu + \mathbf{X}\boldsymbol{\alpha} + \mathbf{e},$$

where $\mathbf{y}$ contains phenotypic records, μ is the overall mean, $\mathbf{1}$ is a vector of 1's, $\mathbf{X}$ is an $m \times n$ matrix that contains the scaled and centered genotypes of all individuals, $\boldsymbol{\alpha}$ contains the (random) allele substitution effects for all loci, and $\mathbf{e}$ contains the random residuals. The specific parameterization of the genomic model that results in the Bayes SSVS model is described below.

Prior densities
The likelihood of the Bayes SSVS model conditional on all unknowns is assumed to be normal:

$$p(y_i|\mu, \boldsymbol{\alpha}, \sigma_e^2) = N\left(y_i - \mu - \mathbf{x}_i'\boldsymbol{\alpha}, \sigma_e^2\right),$$

where $\mathbf{x}_i$ denotes the SNP genotypes of animal i. Definitions of the unknowns and their prior distributions are described hereafter.

The prior for μ was a constant. The residual variance σ_e^2 has a prior distribution of $p(\sigma_e^2) = \chi^{-2}(-2, 0)$, which yields a flat prior.

The prior for α_j depends on the variance σ_α^2 and the QTL indicator $I_j = 1$:

$$\alpha_j|\pi, \sigma_\alpha^2 = \begin{cases} \sim N\left(0, \dfrac{\sigma_\alpha^2}{100}\right) & \text{when } I_j = 0 \\ \sim N\left(0, \sigma_\alpha^2\right) & \text{when } I_j = 1 \end{cases}.$$

The prior distribution for the indicator variable I_j is:

$$p(I_j) = \text{Bernoulli}(1-\pi),$$

where π was assigned a value of 0.999, and σ_α^2 has the following prior distribution:

$$p(\sigma_\alpha^2) = \chi^{-2}(v_\alpha, S_\alpha^2),$$

where v_α is the degrees of freedom, set to 4.2 according to [12,15], and the scale parameter S_α^2 is calculated as $S_\alpha^2 = \dfrac{\tilde{\sigma}_\alpha^2(v_\alpha - 2)}{v_\alpha}$ where $\tilde{\sigma}_\alpha^2$ is computed as [1]:

$$\tilde{\sigma}_\alpha^2 = \left(\frac{100}{100 + \pi(1-100)}\right)\frac{\sigma_a^2}{n},$$

where n is the number of loci.

Conditional posterior densities
The conditional posterior density of α_j is:

$$N\left(\hat{\alpha}_j; \frac{\omega_j\hat{\sigma}_e^2}{\mathbf{x}_j'\mathbf{x}_j + \lambda_j}\right),$$

where $\hat{\alpha}_j$ is the conditional mean of the allele substitution effect at locus j, whose computation was explained previously, $\lambda_j = \frac{\omega_j\hat{\sigma}_e^2}{\sigma_\alpha^2}$, and

$$\omega_j = 1 \quad \text{if} \quad I_j = 1$$
$$\omega_j = 100 \quad \text{if} \quad I_j = 0.$$

The conditional posterior density of σ_α^2 is an inverse-χ^2 distribution:

$$\sigma_\alpha^2\Big|\alpha \tilde{\,} \chi^{-2}\left(v_\alpha + n, S_\alpha^2 + \boldsymbol{\omega}'\hat{\boldsymbol{\alpha}}^2\right),$$

where $\hat{\boldsymbol{\alpha}}^2$ is a vector with squares of the current estimates of the allele substitution effects of all loci, weighted by vector $\boldsymbol{\omega}$, which contains values of 1 or 100 for each locus.

Finally, the conditional posterior distribution of the QTL-indicator I_j was (following the notation in [19]):

$$\Pr(I_j = 1) = \frac{f(r_j|I_j = 1)(1 - \pi)}{f(r_j|I_j = 0)\pi + f(r_j|I_j = 1)(1 - \pi)},$$

where $1 - \pi$ (π) is the prior probability that $I_j = 1$ ($I_j = 0$), $r_j = \mathbf{x}_j'\mathbf{y}^* + \mathbf{x}_j'\mathbf{x}_j\hat{\alpha}_j$, where $\mathbf{y}^*$ contains the conditional phenotypes as defined previously, and $f(r_j|I_j = \delta)$, where δ is either 0 or 1, and is proportional to $\frac{1}{\sqrt{v}}e^{-\frac{r_j^2}{2v}}$, where $v = \left(\mathbf{x}_j'\mathbf{x}_j\right)^2\frac{\sigma_{\alpha_j}^2}{\omega_j} + \mathbf{x}_j'\mathbf{x}_j\sigma_e^2$. It should be noted that v depends on I_j through its dependence on ω_j, i.e. if $I_j = 0$ ($I_j = 1$) then $\omega_j = 100$ ($\omega_j = 1$).

Finally, the conditional posterior density of σ_e^2 is an inverse-χ^2 distribution:

$$\sigma_e^2|e \sim \chi^{-2}\left(m-2, \mathbf{e}'\mathbf{e}\right),$$

where m is the number of animals with records and $\mathbf{e}$ is a vector with the current residuals.

Derivation of the optimal number of SNPs included per RHS-block

Simulated data - CPU time
To investigate to what extent the CPU time of the newly developed algorithms depended on the number of individuals included in the analysis, a dataset of 420 SNPs

was simulated. The number of SNPs was limited to 420, to reduce total computation time, and this limited number of SNPs was sufficient to compare the relative computation time of the different algorithms because total CPU time scales linearly with the number of SNPs. Further details of the simulation are not included here because the aim of the subsequent analysis was only to compare CPU time. The size of the dataset was increased with steps of 500 individuals from 500 to 100 000 individuals and each dataset was analysed 11 times. The newly developed RHS-updating algorithm was used in nine of those analyses, using one to nine SNPs per RHS-block. The tenth analysis was an implementation using original residual updating. The eleventh analysis implemented improved residual updating. Each analysis was run for 900 iterations, and the CPU time for these 900 iterations was recorded.

Simulated data - RAM use

To investigate to what extent the required amount of RAM of the newly developed algorithm depended on the number of individuals in the analysis, a dataset of 50 000 SNPs was simulated. The number of individuals was increased in steps of 5000 individuals from 500 to 95 500 individuals. This number of SNPs and animals yielded a range of datasets with dimensions that corresponded to the size of currently used practical datasets and each dataset was analysed 11 times. These analyses involved the same models and settings as used to evaluate CPU time. Each analysis was run until the iterations started, i.e. when the maximum RAM requirement was reached, at which point the used RAM was recorded and the process was aborted. The maximum RAM requirement was measured by retrieving the process ID and then storing the RAM use for that process. Details of this procedure are provided in the Appendix 1.

All comparisons were run on a Windows XP-64 desktop pc with an Intel(R) Xeon(R) 64-bit CPU E5420 with a clock speed of 2.50 GHz. Comparisons on CPU time were also run on a Linux platform with an AMD Opteron 8431 64-bit CPU with a clock speed of 2.39 GHz running Ubuntu 12.04.3. The programs were compiled with the Intel® Fortran Compiler 11.0.075 for Windows and the Intel® Fortran Compiler 13.0.079 for Linux.

Results
CPU time

The required CPU time on the Windows workstation for the RHS-updating algorithm depended strongly on the number of SNPs included per RHS-block (Figure 2). In all cases, including more than six SNPs per RHS-block resulted in a longer CPU time. Conversely, including less than four or five SNPs per RHS-block also resulted in a longer CPU time, especially when the number of individuals was large. The solid line in Figure 2 shows that the

optimal number of SNPs included per RHS-block that results in the lowest CPU time, changed with the number of individuals in the analysis. This is further illustrated in Figure 3, where the actual number of SNPs that gave the minimum CPU time is plotted against the number of individuals in the analysis. Although there was a clear trend, across the number of individuals, especially with a larger number of individuals, there was no exact threshold when either five or six SNPs were included per RHS-block, which indicates that there was very little difference in CPU time when five or six SNPs were used. Similar trends in CPU time were observed when running the analysis on the Linux server, for which slightly longer CPU times were generally observed (Figure 4). Nevertheless, with our implementation of RHS-updating, it appears to be appropriate to include two SNPs when the number of individuals is less than 1000, three SNPs when it is between 1000 and 2500, four SNPs when it is between 2500 and 11 000, five SNPs when it is between 11 000 and 50 000, and six SNPs when it is between 50 000 and 100 000.

The CPU time for the Gibbs chain using the optimal number of SNPs per RHS-block was compared to the CPU time of the original and improved residual updating schemes across the different numbers of animals included in the analysis (Figure 5). Compared to original residual updating, improved residual updating reduced CPU time on the Windows workstation by 35.3 to 43.3%, and RHS-updating reduced CPU time by 74.5 to 93.0%. Improvements in terms of CPU time were similar for the Linux server (results not shown). The required CPU time for pre-processing the data was slightly larger for the RHS-updating versus the residual updating algorithm. On the Windows workstation, for the dataset with 95 500 animals and 50 000 SNPs, the time to transform the SNP data into the (integer) coding, required 954 s and 1217 s, respectively, for the residual updating and the RHS-updating algorithm. These results show that, although most of the reduction in CPU time achieved by RHS-updating originated from evaluating SNPs within RHS-blocks rather than individually, at the same time, the first step to implement improved residual updating already makes an important contribution to the reduction in CPU time.

RAM use

The required amount of RAM for the RHS-updating algorithm also depended on the number of SNPs included per RHS-block (Figure 6). In nearly all cases, including more than six SNPs per RHS-block resulted in more RAM used. Conversely, including less than three to six SNPs per RHS-block also resulted in more RAM used, especially when the number of individuals was large. The pattern in RAM use (Figure 6), changed with the number of individuals in the dataset, in a pattern that
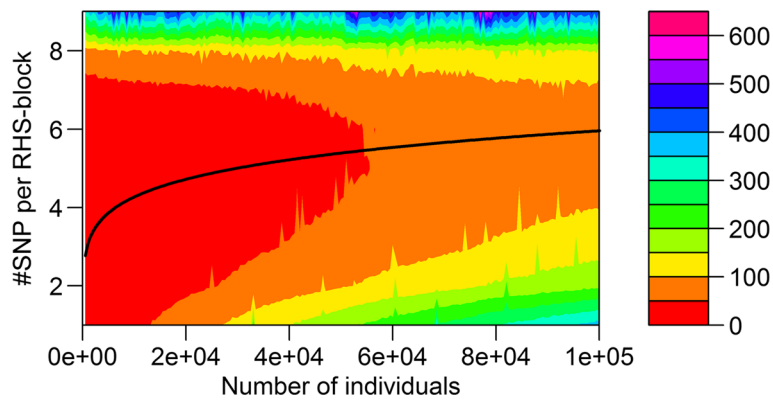
Figure 2 CPU time on a Windows workstation for RHS-updating using different numbers of SNPs per RHS-block. The reported time is for Gibbs chains of 900 iterations; the algorithm used one to nine SNPs per RHS-block, and the data contained 420 SNPs and an increasing number of individuals (500 to 100 000); the black line is the fitted curve through the number of SNPs that gave the minimum CPU time.

was quite similar to that of the CPU time. This implies that choosing an optimal number of SNPs per RHS-block based on required CPU time, yields an algorithm that is also close to optimal in terms of RAM use.

In Figure 7, the amount of RAM used is plotted for all three algorithms, for datasets with 50 000 SNPs and 500 to 95 500 individuals. Note that both residual updating algorithms required about the same amount of RAM, therefore only one curve was plotted for residual updating. In the case of RHS-updating, for each number of individuals, either the number of SNPs per RHS-block that gave the minimum CPU time or the number that gave the minimum amount of required RAM was used. Both sets of numbers of SNPs per RHS-block gave very similar answers. Compared to the residual updating schemes, the reduction in required RAM for RHS-updating ranged from 13.1 to 66.4%.

As shown in Figure 7, for a fixed number of SNPs, the RAM use of the residual updating and the RHS-updating algorithms were linearly related to the number of animals included. This agrees with the derived formulas for expected RAM use in Table 1. The ability of those equations to predict measured RAM use, was investigated by regressing measured RAM use on predicted RAM use for each method, based on datasets containing 50 000 SNPs and 500 to 95 500 animals. The results of those regressions are presented in Table 2 and show that the equations predicted RAM use with an R^2 value of 1.0 in all cases. The intercepts of the regression generally had a positive value, indicating that the prediction equation missed only a small proportion of the used RAM. This value became substantial for the RHS-updating algorithm when the number of SNPs per RHS-block was equal to 7 or more because, e.g., the relative size of the array that
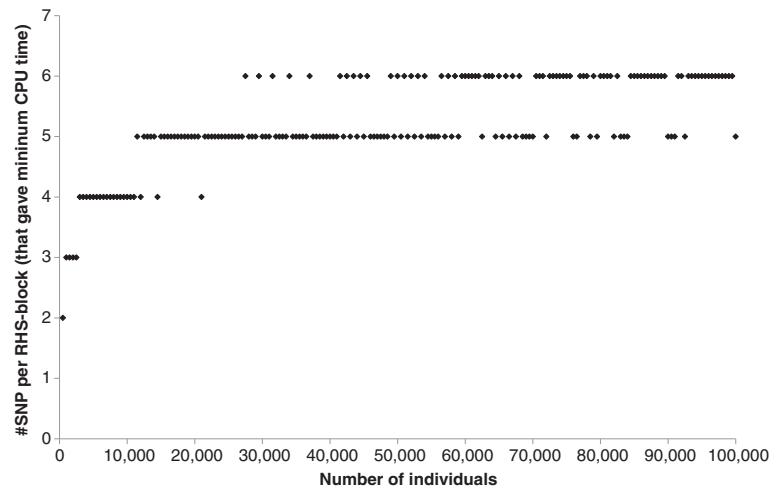

Figure 3 The number of SNPs included per RHS-block in RHS-updating, that yielded the minimum computing time. Minimum computing time was evaluated for an increasing number of individuals (500 to 100 000).
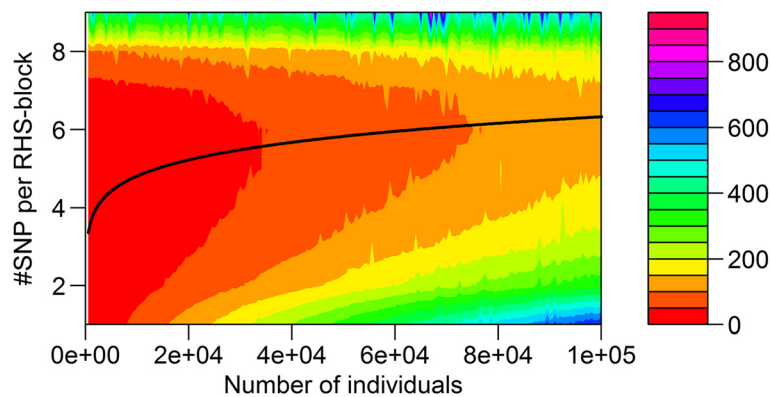
Figure 4 CPU time on a Linux server for RHS-updating using different numbers of SNPs per RHS-block. The reported time is for Gibbs chains of 900 iterations; the algorithm used one to nine SNPs per RHS-block, and the data contained 420 SNPs and an increasing number of individuals (500 to 100 000); the black line is the fitted curve through the number of SNP that gave the minimum CPU time.

stores the number of individuals for each RHS-group increases considerably when the number of RHS-blocks increases.

Discussion

Two alternative algorithms were presented that can be implemented in various genomic prediction models for fast computing of SNP effects. The algorithms replace the originally suggested residual updating algorithm [14], without affecting the results obtained. Differences in results between algorithms were similar to those within algorithms when using different random seeds and correlations between different sets of results were greater than 0.99 (results not shown). Both algorithms use the characteristic that only three different genotypes are observed for each SNP. Both algorithms can accommodate loci with more than three genotypes, but this may reduce

or eliminate their benefit in terms of computing time over original residual updating schemes. The limitation on the number of genotypes per locus implies that imputed genotypes defined as gene contents cannot be used for individuals in the training data in the algorithm. Nevertheless, a simple transformation of gene contents to the most likely genotype overcomes this problem. This transformation could for instance be (on a 0-2 scale): genotypes ≤ 0.5 are set to 0, genotypes ≥ 1.5 are set to 2, and all other genotypes are set to 1. Such transformations for imputed genotypes in the training data are expected to have a minor impact on the estimated SNP effects, provided that the genotypes are imputed with reasonable accuracy. Using gene contents for selection candidates, i.e. individuals whose genetic merit is predicted using SNP effects estimated from the training data, is not inhibited by the proposed algorithms, because their predicted genetic
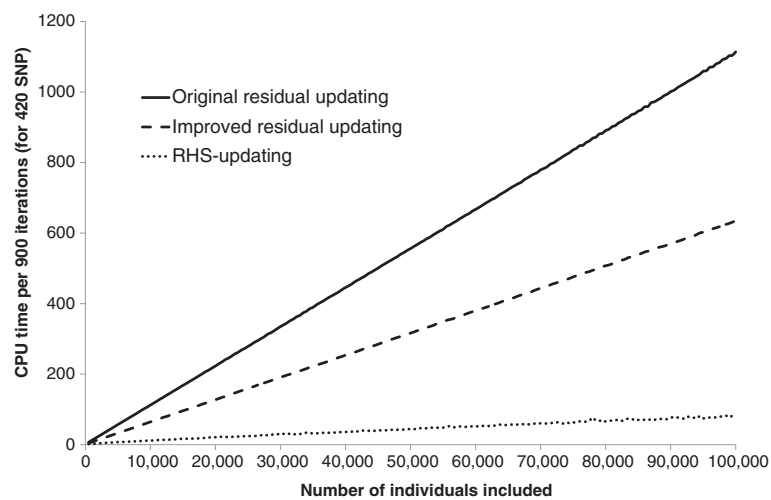

Figure 5 CPU time on a Windows workstation using two residual updating schemes and RHS-updating. CPU time for a Gibbs chain of 900 iterations was evaluated for 420 SNPs and an increasing number of individuals (500 to 100 000); the reported CPU time for the RHS-updating scheme is the minimum computing time of nine analyses that include one to nine SNPs per RHS-block, for each number of individuals.
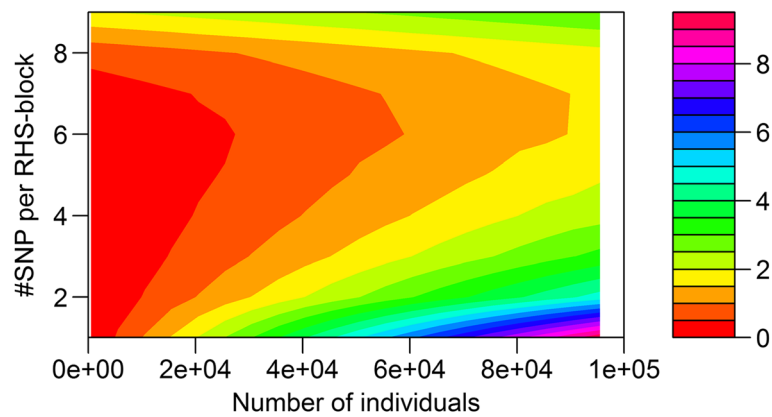
Figure 6 RAM use (Gb) for RHS-updating with one to nine SNPs per RHS-block. RAM use was evaluated for 50 000 SNPs and an increasing number of individuals (500 to 95 500).

merit can simply be obtained outside the algorithm by multiplying their gene contents with allele substitution effects that are estimated in the algorithm. The literature shows that for selection candidates, predictions differ when gene contents or the most likely genotypes are used [20].

The residual updating algorithms were implemented using standard (e.g. dot_product) Fortran functions. Computer-specific optimized libraries are available [21,22] that can considerably reduce the CPU time required for, e.g., vector and matrix multiplications [23]. Using such libraries may have a larger impact on CPU time for the residual updating algorithms than for the RHS-updating algorithm, since the former involves many more multiplications. However, even when using such optimized libraries, the RHS-updating algorithm is still expected

to be considerably more efficient, because it drastically reduces the total number of required operations.

It should be noted that the RHS-updating scheme requires slightly more overhead in terms of computing time than the residual updating scheme, for instance to define the RHS-blocks and the group coding within those blocks. Once RHS-blocks and group coding are defined, they can be used in each iteration of the Gibbs chain. This may limit flexibility in the algorithm. For instance, one way to improve mixing of the Gibbs chains, may be to permute the order of evaluation of SNP effects between iterations. With the RHS-updating scheme, the order of evaluation of SNP effects within RHS-blocks must be the same throughout the Gibbs chain, such that group coding within RHS-blocks needs to be defined only once.
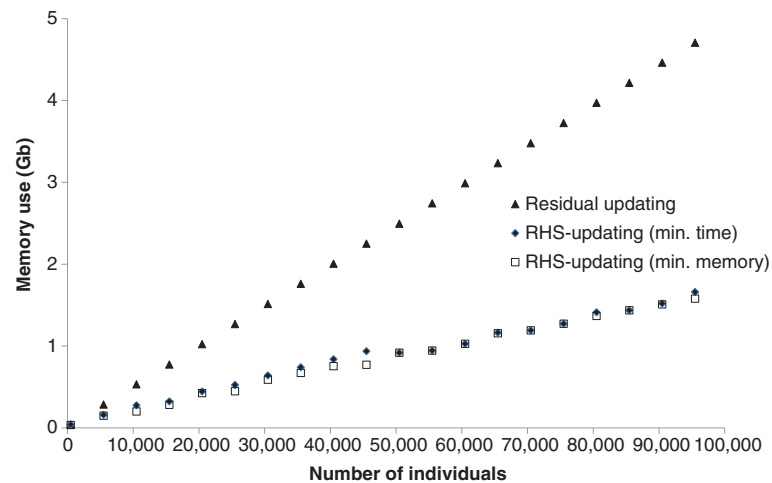

Figure 7 RAM use (Gb) for residual updating versus RHS-updating. For RHS-updating, the number of SNPs per RHS-block was set to the number that gave either the minimum computing time (min. time) or the minimum RAM requirement (min. RAM); RAM use was evaluated on a Windows workstation for a dataset with 50 000 SNPs and for an increasing number of individuals (500 to 95 500); note that only one curve is plotted for residual updating, because both residual updating algorithms require practically the same amount of RAM.

Table 2 Coefficients of the regression of measured on predicted RAM requirements for original and improved residual updating, and for RHS-updating

Algorithm	Intercept	Slope	R^2
Original residual updating	0.014	0.983	1.000
Improved residual updating	0.014	0.983	1.000
RHS-updating (1)[1]	0.015	0.981	1.000
RHS-updating (2)	0.015	0.983	1.000
RHS-updating (3)	0.016	0.985	1.000
RHS-updating (4)	0.024	0.981	1.000
RHS-updating (5)	0.035	0.983	0.999
RHS-updating (6)	0.065	0.977	0.996
RHS-updating (7)	0.229	0.989	1.000
RHS-updating (8)	0.659	0.990	1.000
RHS-updating (9)	1.948	0.992	1.000

[1]The number between brackets indicates the number of SNPs per RHS-block.

Nevertheless, the order of evaluation of the RHS-blocks can still be permuted. Furthermore, to avoid that neighbouring SNPs are always evaluated in the same order, SNPs can be allocated to RHS-blocks at random.

The RHS-updating scheme not only considerably reduced computing time, by up to 93%, but also resulted in a reduction of the amount of RAM used of up to 66%. Due to the nature of the RHS-updating algorithm, computing time and RAM use are linearly related with the number of SNPs considered, similar to the residual updating algorithm. This implies that the relative benefit of using the RHS-updating algorithm compared to the residual updating algorithm is not affected by the number of SNPs included. In our implementation, which is written in Fortran 95, group codes within RHS-blocks were stored as an integer(2) variable, while the genotypes in the original implementation with residual updating were stored as an integer(1) variable. It should be noted that storing group codes as integer(1) would lead to a further reduction in RAM requirements of almost 50%, because the array that stores the group codes uses close to 100% of the RAM used by the algorithm. Storing group codes as integer(1), implies that the number of SNPs included per RHS-block should be equal to four or less, i.e. the maximum value an integer(1) variable can take is equal to 127, and including 4 (5) SNPs per RHS-block yields $3^4 = 81$ ($3^5 = 243$) groups; the maximum value an integer (2) variable can take, is equal to 32 767. This means that a maximum of nine SNPs can be included per RHS-block, otherwise the group code must be stored as integer(4), i.e. including 9 (10) SNPs per RHS-block yields $3^9 = 19\ 683$ ($3^{10} = 59\ 049$) groups. Our results show that with the largest number of individuals considered (100 000), the optimal number of SNPs included per RHS-block was equal to six. This suggests that it is unlikely that a number

of individuals in the data that justifies including more than nine SNPs per RHS-block, and therefore requires storing group codes as integer(4), is reached in the near future.

In our implementation of RHS-updating, each RHS-block containing *s* SNPs is assumed to contain all 3^s possible groups, which is most likely not always the case. Moreover, clever grouping of SNPs within RHS-block can reduce the observed number of groups within each RHS-block. Such redundancy could be used to further reduce computing time, but would also likely result in a more complicated algorithm.

Conclusions

Two algorithms are presented to estimate SNP effects that can be implemented in a range of different genomic prediction models, as an alternative to the original residual updating scheme. The first alternative algorithm uses residual updating, here termed improved residual updating, and takes advantage of the characteristic that the predictor variables in the model (i.e. SNP genotypes) have only three possible values. The second alternative algorithm, here termed "RHS-updating", extends the idea of improved residual updating across multiple SNPs. The improved residual updating algorithm achieved a reduction in computing time of 35.3 to 43.3%, but did not change the amount of RAM used, compared to the original residual updating scheme. The RHS-updating algorithm achieved a reduction in computing time of 74.5 to 93.0% and a reduction in RAM use of 13.1 to 66.4%, compared to the original residual updating scheme. Thus, the RHS-updating algorithm provides an interesting alternative to reduce both computing time and memory requirements.

Appendix 1. Pseudo-code to measure maximum RAM requirement

The maximum RAM requirement was measured by retrieving the process ID and then storing the RAM use for this particular process. On the Windows OS, this was done using the following Fortran code:

```
PROGRAM GIBBS
USE DFPORT !Module that contains function "GETPID"
IMPLICIT NONE
INTEGER :: PROC_ID
CHARACTER(LEN = 47) :: SYS_CALL
…
PROC_ID = GET_PID() !Retrieve process ID of the
current process
!Use DOS command "tasklist" to write RAM use to file
"tasklist.txt"
SYS_CALL = "tasklist /fi ""PID eq "" > tasklist.txt"
WRITE(SYS_CALL(23:29),'(i7)')PROC_ID
CALL SYSTEM(SYS_CALL)
…
END PROGRAM GIBBS
```

References
1. de los Campos G, Hickey JM, Pong-Wong R, Daetwyler HD, Calus MPL: **Whole-genome regression and prediction methods applied to plant and animal breeding.** *Genetics* 2013, **193**:327–345.
2. Daetwyler HD, Calus MPL, Pong-Wong R, de los Campos G, Hickey JM: **Genomic prediction in animals and plants: simulation of data, validation, reporting, and benchmarking.** *Genetics* 2013, **193**:347–365.
3. Lund M, de Roos S, de Vries A, Druet T, Ducrocq V, Fritz S, Guillaume F, Guldbrandtsen B, Liu Z, Reents R, Schrooten C, Seefried F, Su G: **A common reference population from four European Holstein populations increases reliability of genomic predictions.** *Genet Sel Evol* 2011, **43**:43.
4. Rincon G, Weber KL, Van Eenennaam AL, Golden BL, Medrano JF: **Hot topic: performance of bovine high-density genotyping platforms in Holsteins and Jerseys.** *J Dairy Sci* 2011, **94**:6116–6121.
5. Hayes B, Anderson C, Daetwyler HD, Fries R, Guldbrandtsen B, Lund M, Boichard D, Stothard P, Veerkamp RF, Hulsegge I, Rocha D, Van Tassel CP, Coote D, Goddard M: **Towards genomic prediction from genome sequence data and the 1000 bull genomes project.** In *Book of Abstracts of the 4th International Conference on Quantitative Genetics: 17-22 June 2012; Edinburgh.*; 2012:55.
6. Meuwissen THE, Goddard ME: **Accurate prediction of genetic value for complex traits by whole-genome resequencing.** *Genetics* 2010, **185**:623–631.
7. Hayashi T, Iwata H: **EM algorithm for Bayesian estimation of genomic breeding values.** *BMC Genet* 2010, **11**:3.
8. Shepherd RK, Meuwissen THE, Woolliams JA: **Genomic selection and complex trait prediction using a fast EM algorithm applied to genome-wide markers.** *BMC Bioinformatics* 2010, **11**:529.
9. Meuwissen THE, Solberg TR, Shepherd R, Woolliams JA: **A fast algorithm for BayesB type of prediction of genome-wide estimates of genetic value.** *Genet Sel Evol* 2009, **41**:2.
10. Cai X, Huang A, Xu S: **Fast empirical Bayesian LASSO for multiple quantitative trait locus mapping.** *BMC Bioinformatics* 2011, **12**:211.
11. VanRaden PM: **Efficient methods to compute genomic predictions.** *J Dairy Sci* 2008, **91**:4414–4423.
12. Meuwissen THE, Hayes BJ, Goddard ME: **Prediction of total genetic value using genome-wide dense marker maps.** *Genetics* 2001, **157**:1819–1829.
13. Stranden I, Christensen OF: **Allele coding in genomic evaluation.** *Genet Sel Evol* 2011, **43**:25.
14. Legarra A, Misztal I: **Computing strategies in genome-wide selection.** *J Dairy Sci* 2008, **91**:360–366.
15. Habier D, Fernando R, Kizilkaya K, Garrick D: **Extension of the Bayesian alphabet for genomic selection.** *BMC Bioinformatics* 2011, **12**:186.
16. Calus MPL, Meuwissen THE, De Roos APW, Veerkamp RF: **Accuracy of genomic selection using different methods to define haplotypes.** *Genetics* 2008, **178**:553–561.
17. Verbyla KL, Hayes BJ, Bowman PJ, Goddard ME: **Accuracy of genomic selection using stochastic search variable selection in Australian Holstein Friesian dairy cattle.** *Genet Res* 2009, **91**:307–311.
18. Habier D, Fernando R, Dekkers J: **The impact of genetic relationship information on genome-assisted breeding values.** *Genetics* 2007, **177**:2389–2397.
19. Jia Y, Jannink J-L: **Multiple-trait genomic selection methods increase genetic value prediction accuracy.** *Genetics* 2012, **192**:1513–1522.
20. Mulder HA, Calus MPL, Druet T, Schrooten C: **Imputation of genotypes with low-density chips and its effect on reliability of direct genomic values in Dutch Holstein cattle.** *J Dairy Sci* 2012, **95**:876–889.
21. Whaley RC, Dongarra JJ: **Automatically tuned linear algebra software.** In *Proceedings of the 1998 ACM/IEEE conference on Supercomputing: 7-13 November 1998: Orlando. (CDROM)*. Washington, DC, USA: IEEE Computer Society; 1998:1–27.
22. Dongarra JJ, Du Croz J, Hammarling S, Duff IS: **A set of level 3 basic linear algebra subprograms.** *ACM TOMS* 1990, **16**:1–17.
23. Aguilar I, Misztal I, Legarra A, Tsuruta S: **Efficient computation of the genomic relationship matrix and other matrices used in single-step evaluation.** *J Anim Breed Genet* 2011, **128**:422–428.