MARCOS RODRIGUES LAGROTTA

COMPUTAÇÃO DE ALTO DESEMPENHO NA SELEÇÃO GENÔMICA

Tese apresentada à Universidade Federal de Viçosa, como parte das exigências do Programa de Pós-Graduação em Zootecnia, para obtenção do título de *Doctor Scientiae*.

VIÇOSA
MINAS GERAIS – BRASIL
2012

Ficha catalográfica preparada pela Seção de Catalogação e Classificação da Biblioteca Central da UFV

T
L179c
2012

Lagrotta, Marcos Rodrigues, 1981-
Computação de alto desempenho na seleção genômica / Marcos Rodrigues Lagrotta. – Viçosa, MG, 2012.
xi, 69f. : il. (algumas col.) ; 29cm.

Inclui apêndices.
Orientador: Ricardo Frederico Euclydes.
Tese (doutorado) - Universidade Federal de Viçosa.
Referências bibliográficas: f. 48-54

1. Animais - Melhoramento genético. 2. Genômica.
3. Animais - Seleção - Simulação por computador.
4. Programação paralela (Computação). I. Universidade Federal de Viçosa. II. Título.

CDD 22. ed. 636.0821

MARCOS RODRIGUES LAGROTTA

COMPUTAÇÃO DE ALTO DESEMPENHO NA SELEÇÃO GENÔMICA

Tese apresentada à Universidade Federal de Viçosa, como parte das exigências do Programa de Pós-Graduação em Zootecnia, para obtenção do título de *Doctor Scientiae*

APROVADA: 27 de julho de 2012.

Fabyano Fonseca e Silva

Gustavo Henrique de Souza

Robledo de Almeida Torres
(Coorientador)

Carlos de Castro Goulart

Ricardo Frederico Euclydes
(Orientador)

DEDICO

A Deus, pela vida, sabedoria, força e fé para seguir adiante.

Aos meus pais e irmãos, pelo eterno amor, ensinamentos e incentivo.

Aos meus avós e tias, por todo carinho e apoio.

A minha companheira, por todo amor, apoio e dedicação.

Aos meus sogros, pela paciência, compreensão e incentivo.

AGRADECIMENTOS

À Universidade Federal de Viçosa e ao Departamento de Zootecnia, pela oportunidade de realizar mais esta etapa da minha vida.

À University of Wisconsin - Madison e ao Departamenty of Animal Science, por todo suporte para realização da minha pesquisa.

À Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES), pela bolsa de estudo.

Ao Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq), pela concessão da bolsa de estudo no exterior.

Ao meu orientador Ricardo Frederico Euclydes (Bajá), pelo apoio, incentivo, confiança e pelos longos anos de rico convívio.

Aos meus conselheiros Robledo Torres e Guilherme Rosa, pelas valiosas contribuições e sugestões, pelo apoio, incentivo e pela amizade.

Aos professores Carlos Goulart, Fabyano Fonseca, Gustavo Souza e ao pesquisador Xiao Lin Wu (Nick), pelas valiosas contribuições e sugestões.

Aos amigos de Madison Afishin, Bruno Valente, Ana Angélica, José Fernando, Rogério Sassonia, Rafael Rocha, Vívian, Saleh, Sabrina, Marina e Vera, pelos momentos de alegria, conselhos e incentivo.

Aos meus amigos Gustavo, Raphael e Prata, por todo apoio, boas conversas e momentos ímpares.

A Zé Maria e Cida, pelo apoio incondicional e carinho.

Aos amigos e amigas do DZO que contribuíram com momentos de alegria, estudos e conselhos.

Aos funcionários do DZO que sempre me ajudaram.

Enfim, a todos que, de alguma forma, contribuíram na concretização deste trabalho.

BIOGRAFIA

Marcos Rodrigues Lagrotta, filho de Jorge Costa Lagrotta e Eliana Lacerda Rodrigues Lagrotta, nasceu em 09 de outubro de 1981, em Leopoldina, Minas Gerais.

Em março de 2002, ingressou no curso de Zootecnia da Faculdade Federal de Odontologia de Diamantina - MG, denominada, em 2006, Universidade Federal dos Vales do Jequitinhonha e Mucuri, graduando-se em julho deste ano.

Em outubro de 2006, iniciou o curso de Mestrado em Zootecnia, na área de Melhoramento Animal, na Universidade Federal de Viçosa - MG.

Em 28 de julho de 2008, submeteu-se aos exames finais de defesa de dissertação para obtenção do título de *Magister Scientiae* em Zootecnia.

Em agosto de 2008, iniciou o curso de Doutorado em Zootecnia, na área de Melhoramento Animal, na Universidade Federal de Viçosa - MG.

Em 27 de julho de 2012, submeteu-se aos exames finais de defesa de tese para obtenção do título de *Doctor Scientiae* em Zootecnia.

SUMÁRIO

RESUMO

LAGROTTA, Marcos Rodrigues, D.Sc., Universidade Federal de Viçosa, julho de 2012. **Computação de alto desempenho na seleção genômica.** Orientador: Ricardo Frederico Euclydes. Coorientador: Robledo de Almeida Torres.

A computação paralela vem crescendo nos últimos anos em virtude do menor custo dos computadores e do aumento exponencial dos bancos de dados. O processamento em paralelo envolve a execução de múltiplas tarefas simultaneamente em diferentes processadores. No contexto da seleção genômica, o grande número de marcadores genéticos utilizado nas análises, bem como a grande demanda computacional dos modelos bayesianos fundamentados nos métodos de Monte Carlo Via Cadeias de Markov, faz com que certas análises despendem semanas ou meses de processamento. Assim, a computação paralela representa uma solução natural a este problema. O método usado para análise foi o BayesCπ, o qual possui apenas passos do Amostrador de Gibbs. O algoritmo foi inicialmente escrito na forma sequencial usando o FORTRAN. Duas estratégias de paralelização foram então estudadas. A primeira envolveu a análise de múltiplas cadeias em paralelo, sendo recomendada na situação em que o *burn-in* não seja longo. A segunda estratégia referiu-se à paralelização da própria cadeia, sendo indicada para situações em que o *burn-in* é muito longo. Utilizou-se a biblioteca MPI e o pacote OpenMPI associado ao compilador gfortran para tal propósito. As computações foram realizadas em um computador pessoal, com seis núcleos de processamento de 3,3 GHz e 16 GB de memória RAM e em um *cluster* com 120 processadores de 2,77 GHz. Foram utilizados dados simulados para duas características produtivas de bovinos de leite, referentes a 10.000 marcadores e 4.100 indivíduos. No computador pessoal, o algoritmo sequencial foi processado em 77,29 horas e ao usar múltiplas cadeias em paralelo o processamento foi quase cinco vezes mais rápido com seis núcleos de processamento. A relação de desempenho entre o algoritmo paralelo e o sequencial foi maior no *cluster*, pois a sua arquitetura de memória escalona melhor com o número de processadores em uso do que

a arquitetura de memória compartilhada do computador pessoal. A segunda estratégia de paralelização apresentou um ganho de desempenho de apenas 19% com dois processadores. Contudo, usando mais processadores não houve melhora de desempenho. Esta estratégia só se aplica em sistemas com arquitetura de memória compartilhada, devido ao elevado *overhead* (sobrecarga) gerado pela intensa troca de informações e sincronização das tarefas. Conclui-se que a computação paralela é uma técnica de fundamental importância para a seleção genômica, e isto será mais expressivo nos próximos anos devido ao rápido crescimento dos bancos de dados. Estratégias mais eficientes de paralelização da própria cadeia devem ser desenvolvidas, visto que nas situações em que o *burn-in* é muito longo o processamento de múltiplas cadeias em paralelo não é recomendado. O ideal seria que estas novas abordagens apresentassem bom desempenho em sistemas com arquitetura de memória distribuída (*clusters*).

ABSTRACT

LAGROTTA, Marcos Rodrigues, D.Sc., Universidade Federal de Viçosa, July, 2012. **High performance computing in genomic selection.** Adviser: Ricardo Frederico Euclydes. Co-adviser: Robledo de Almeida Torres.

Parallel computing has been growing in recent years due to the lower cost of computers and the exponential growth of databases. The parallel processing involves performing multiple tasks simultaneously on different processors. In the context of genomic selection, the large number of genetic markers used in the analyzes as well as the high computational demand of Bayesian models based on methods of Markov Chain Monte Carlo makes that certain analyzes have weeks or months of runtime. Thus parallel computing is a natural solution to this problem. The method used for analysis was BayesCπ, which has only the Gibbs sampling steps. The algorithm was initially written in a sequential manner using FORTRAN. It was studied two parallelization strategies. The first involved the analysis of multiple parallel chains being recommended in the situation that the burn-in is not long. The second strategy is relative to the parallelization of the chain itself, being indicated for cases in which the burn-in time is too long. It was used the MPI library and the packet OpenMPI associated to the gfortran compiler for this purpose. The computations were performed on a personal computer, with six processing cores of 3.3 GHz and 16 GB of RAM (Random Access Memory) and a cluster with 120 processors of 2.77 GHz. Simulated data for two traits of dairy cattle, referring to 10,000 markers and 4,100 individuals, were used. In the personal computer, the sequential algorithm was processed at 77.29 hours and by using parallel multiple chains the processing was almost five times faster with six cores. The performance ratio between parallel and sequential algorithms was higher in the cluster, because its memory architecture scales better with the number of processors in use than the shared memory architecture of the personal computer. The second parallelization strategy presented a performance gain of only 19% with two processors. Using more processors the processing speed was diminishing slowly. This strategy applies only on systems with shared memory architecture, due to the high overhead generated by the

intense exchange of information and tasks synchronization. Therefore parallel computing is a technique of fundamental importance for genomic selection and it will be more significant in coming years due to rapid growth of databases. More efficient strategies for parallelization of the chain itself must be developed, because in situations where the burn-in is too long the processing of multiple chains in parallel is not recommended. The ideal would be that these new approaches have good performance in systems with distributed memory architecture (clusters).

1. INTRODUÇÃO

Uma nova tecnologia, denominada seleção genômica, vem proporcionando mudanças expressivas nos programas de melhoramento genético animal e vegetal. Antes, as avaliações genéticas eram fundamentadas no fenótipo e pedigree. Agora, com o emprego de ferramentas moleculares avançadas, como o chip de polimorfismo de sítio único (SNP chip), que permite a varredura uniforme do genoma para milhares de marcadores simultaneamente, é possível a obtenção de mapas genômicos densos extremamente informativos e, consequentemente, valores genéticos com alta acurácia para algumas características. Contudo, existem muitos desafios para aplicação da seleção genômica, tanto metodologicamente quanto computacionalmente.

O desafio estatístico está em predizer o valor genético genômico numa situação em que o número de indivíduos é muito menor do que o número de marcadores. Os modelos são, portanto, superparametrizados. Para resolver este problema, métodos estatísticos sofisticados têm sido propostos. Os mais utilizados são baseados na Inferência Bayesiana e nos métodos de Monte Carlo Via Cadeias de Markov (MCMC), que é uma técnica iterativa computacionalmente intensiva para amostragem a partir de uma distribuição de probabilidade (geralmente muito grande). A complexidade desses métodos, associado às informações de milhares de marcadores e animais genotipados, tem resultado em longo tempo de computação, chegando a perdurar por semanas ou meses, mesmo em computadores de alto desempenho.

A solução natural para esse problema encontra-se na computação paralela. Para tanto, o algoritmo deve ser dividido em *n* tarefas independentes que podem ser processadas simultaneamente (em paralelo), contribuindo para redução do tempo de processamento. A melhora no desempenho computacional pode significar mais lucratividade e/ou economia para empresas e instituições de pesquisa, pois além da solução rápida dos problemas, os computadores paralelos são construídos a partir de componentes baratos (*commodities*).

Anos atrás, o trabalho computacional pesado era realizado em um grande computador centralizado, denominado supercomputador. Estas máquinas além de muito caras, tinham disponibilidade limitada. No entanto, como os computadores pessoais estão cada vez mais rápidos e baratos, os usuários da computação paralela vêm, nos últimos anos, migrando para o uso de um grande número de computadores organizados em estruturas denominadas *clusters*. Estas estruturas se tornaram muito populares e já estão disponíveis em diversas Universidades.

Diante do exposto, objetivou-se demonstrar o quão importante a computação paralela é para a seleção genômica. Para isso, comparou-se a eficiência de processamento da metodologia BayesCπ programada em paralelo com a forma sequencial padrão. Duas estratégias de paralelismo foram estudadas com o intuito de demonstrar qual a mais recomenda dependendo do tamanho do *burn-in* e da arquitetura de computador utilizada (*cluster* ou computador pessoal).

2. REVISÃO BIBLIOGRÁFICA

2.1. Seleção Genômica

A seleção genética tem sido praticada com base em dados fenotípicos avaliados a campo. A primeira proposição realizada para aumentar a eficiência deste procedimento foi por meio da Seleção Assistida por Marcadores (SAM). Esta técnica utiliza simultaneamente dados fenotípicos e dados de marcadores moleculares em ligação gênica próxima com alguns loci controladores de características quantitativas (QTLs - *quantitative trait loci*).

A SAM apresenta algumas características básicas. Primeiro, requer o estabelecimento de associações marcadores-QTLs (análise de ligação) para cada família em avaliação, ou seja, essas associações apresentam utilidade para seleção apenas dentro de cada família mapeada. Segundo, para ser útil precisa explicar grande parte da variação genética de uma característica quantitativa, que é governada por muitos loci de pequenos efeitos. Entretanto, isto não tem sido observado na prática, exatamente em função da natureza poligênica e alta influência ambiental nos caracteres quantitativos, fato que conduz à detecção de um pequeno número de QTLs de grandes efeitos, os quais capturam apenas uma pequena proporção da variância genética total. Por fim, só apresenta superioridade considerável em relação à seleção baseada em dados fenotípicos, quando o tamanho de família avaliado e genotipado é muito grande (da ordem de 500 ou mais). Em função desses aspectos, bem como dos altos custos da genotipagem, a implementação da SAM tem sido limitada e os ganhos em eficiência muito reduzidos (Dekkers, 2004; Resende et al., 2008).

Com o surgimento de uma nova classe de marcadores, um novo método de avaliação genética, denominado Seleção Genômica (SG) ou Seleção Genômica Ampla (SGA), foi proposto por Mewussien et al., em 2001. Esta técnica é fundamentada na seleção de milhares de marcadores por todo genoma, explicando grande parte da variação genética de um caráter quantitativo por capturar todos os genes que a afeta. Os principais objetivos do uso da SG no melhoramento genético são obter alta eficiência seletiva, grande rapidez na obtenção de ganhos genéticos, pois a predição e

a seleção podem ser realizadas em fases muito juvenis de plantas e animais, e baixo custo dos programas de melhoramento.

Diferente da SAM, na SG não é necessário identificar previamente marcadores com efeitos significativos e mapear QTLs, pois virtualmente todos os QTLs estarão em desequilíbrio de ligação (DL) com algum marcador. A SG pode ser aplicada em todas as famílias em avaliação nos programas de melhoramento genético, de tal forma que o impacto de determinadas famílias específicas (com específicos padrões de DL) nas estimativas dos efeitos dos marcadores seja minimizado. Além disso, apresenta alta acurácia seletiva para a seleção baseada exclusivamente em marcadores (Muir, 2007; Resende et al., 2008; Hayes et al., 2009).

DL é uma medida da dependência ou não entre alelos de dois ou mais *loci*, não necessariamente no mesmo cromossoma. Em um grupo de indivíduos, se dois alelos são encontrados juntos com frequência maior do que aquela esperada com base no produto de suas frequências infere-se que tais alelos estão em DL. Valores de DL próximos de zero indicam equilíbrio ou independência entre os alelos de diferentes genes e valores próximos de um indicam desequilíbrio ou dependência (ligação) entre alelos de diferentes genes.

A técnica de SG proposta por Mewussien et al. (2001) ficou esquecida por alguns anos, tornando-se realidade somente depois do desenvolvimento de chips contendo milhares de marcadores genéticos denominados polimorfismo de nucleotídeo único ou SNP (em inglês *Single Nucleotide Polymorphism*). O primeiro SNP chip foi desenvolvido para bovinos de leite da raça holandesa (Matukumalli et al., 2009). Posteriormente, foram desenvolvidos chips para algumas raças de bovinos de corte, suínos, aves, peixes e algumas espécies de planta, como milho e eucalipto.

Cada SNP refere-se à alteração de um único par de bases no genoma. Os SNPs são a forma mais abundante de variação do DNA em genomas e são preferidos em relação a outros marcadores genéticos devido à sua baixa taxa de mutação, baixa ou nula taxa de recombinação observada, pelo fato de muitas vezes esse polimorfismo estar muito próximo ou ser a mutação funcional do gene, e facilidade de genotipagem. Para que

uma variação seja considerada SNP, ela deve ocorrer em pelo menos 1% da população. Além da predição do valor genético, esses marcadores podem ser utilizados para diversas outras aplicações, tais como determinação de paternidade, construção de mapas genéticos, mapeamento de características de herança quantitativa e isolamento de genes.

Atualmente, utilizam-se mais de 50.000 marcadores SNP nas avaliações genômicas, assim a probabilidade de encontrar um QTL em DL com pelo menos um marcador é muito alta. Este aspecto é importante, uma vez que somente os marcadores em DL com os QTLs serão úteis na predição dos valores genéticos genômicos (VGGs) dos indivíduos (Hayes et al., 2009).

Para predição dos VGGs, uma equação de predição deve ser derivada, tendo como base uma população de referência em que os animais são fenotipados e genotipados. Com a estimação dos efeitos dos milhares de marcadores, praticamente os efeitos de todos loci que contribuem para a variação genética são capturados, mesmo se os efeitos são muito pequenos. Como os efeitos dos marcadores são assumidos como predições válidas para toda a população e não apenas para um grupo de indivíduos, os seus valores são utilizados para predizer o VGG de qualquer indivíduo da população, desde que os mesmos sejam genotipados. A maioria dos marcadores apresenta efeitos relativamente pequenos, refletindo o que acontece no modelo infinitesimal (muitos genes de pequeno efeito associados ao caráter quantitativo). Em subsequentes gerações, os animais a serem avaliados podem ser apenas genotipados. Assim, cada indivíduo pode ter seu VGG predito desde o momento em que é gerado (Goddard & Hayes, 2007; Hayes et al., 2009). Os marcadores dos indivíduos genotipados podem ser usados para predição do VGG de qualquer característica, mas as estimativas dos efeitos dos marcadores serão diferentes para cada uma.

Espera-se que a seleção genômica tenha grande impacto sobre as características de alto custo ou de difícil mensuração, como aquelas associadas aos custos de produção (consumo e eficiência alimentar) e qualidade do produto final (qualidade de carne e carcaça), as limitadas pelo sexo, como fertilidade da fêmea e tamanho de leitegada, e as de difícil

seleção devido à baixa herdabilidade, como precocidade sexual e eficiência reprodutiva (VanRaden et al., 2009).

2.1.1. Implementação da Seleção Genômica

Para implementar a SG em um programa de melhoramento genético, primeiramente é necessário coletar material genético dos animais (pelo ou amostra do sangue, por exemplo) para formar um banco de dados de genótipos. Esses dados serão analisados conjuntamente com um banco de dados de fenótipos. Ou seja, é preciso fazer a avaliação do fenótipo dos reprodutores e seus filhos de forma bastante semelhante ao que se faz atualmente. Após um longo trabalho de coleta e estudo de dados o sistema está montado e deve ser alimentado constantemente com novas informações. A partir desse ponto é possível gerar informações para predição dos VGGs dos reprodutores pertencentes ao programa de melhoramento genético.

Na prática, para predizer os VGGs com acurácia, e assim serem empregados para seleção dos melhores reprodutores, três populações podem ser definidas: treinamento, validação e seleção (Figura 1). Essas podem ser (i) fisicamente distintas (três populações diferentes), (ii) exercer duas funções ao mesmo tempo (uma só população usada para treinamento e validação), (iii) exercer três funções ao mesmo tempo (uma só população usada para treinamento, validação e seleção). Em geral, as estratégias (i) e (ii) são mais usadas (Goddard & Hayes, 2007; Resende et al., 2010).

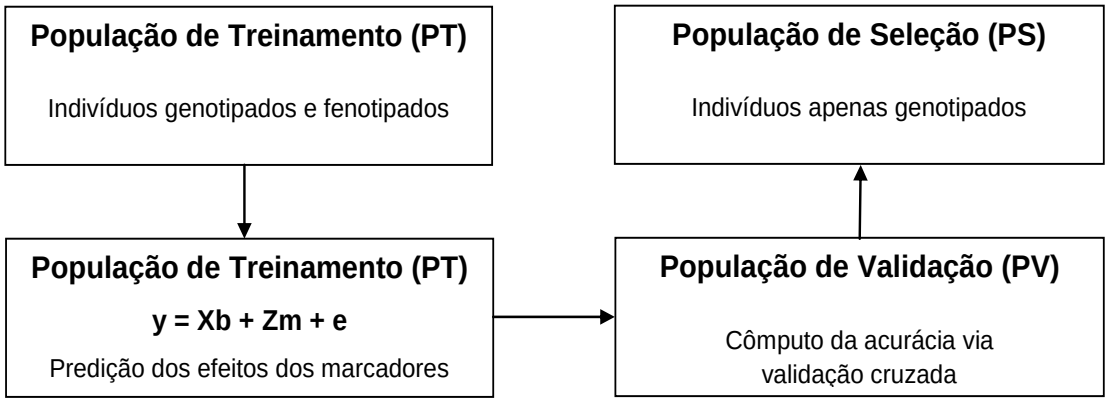

Figura 1 – Esquema de aplicação da seleção genômica em um programa de melhoramento genético
Fonte: adaptado de Resende et al. (2010)

População de Treinamento: também denominada população de estimação, descoberta, ou referência, esse conjunto de dados contempla um grande número de marcadores avaliados em um número moderado de indivíduos, os quais devem ter seus fenótipos avaliados para os vários caracteres de interesse. Equações de predição de VGGs são obtidas para cada caráter de interesse. Essas equações associam a cada marcador ou intervalo (haplótipo) o seu efeito no caráter de interesse. Nessa população são descobertos os marcadores que explicam os loci que controlam os caracteres, bem como são estimados os seus efeitos.

População de Validação: esse conjunto de dados é menor do que o da população de treinamento e contempla indivíduos avaliados para os marcadores SNPs e para os vários caracteres de interesse. As equações de predição dos VGGs são testadas para verificar a acurácia nessa amostra independente. Esta técnica é denominada validação cruzada e será descrita no próximo tópico.

População de Seleção: esse conjunto de dados contempla apenas os marcadores SNP avaliados nos candidatos à seleção. Essa população não necessita ter os seus fenótipos avaliados. As equações de predição derivadas na população de descoberta são então usadas na predição dos VGGs ou fenótipos futuros dos candidatos à seleção. Mas, a acurácia seletiva associada refere-se àquela calculada na população de validação.

2.1.2. Validação Cruzada

A validação cruzada, também denominada estimativa de rotação, é uma técnica utilizada para avaliar como os resultados de uma análise estatística vão generalizar para um conjunto de dados independente (Devijver, 1982; Geisser, 1993; Kohavi, 1995). É usada principalmente em situações onde o objetivo é a predição, visando estimar o quão acurado um modelo preditivo irá ser na prática. Uma rodada da validação cruzada envolve a separação de uma amostra de dados em subconjuntos complementares, realizando a análise em um subconjunto (conjunto de treinamento), e validando a análise no outro subconjunto (conjunto de validação ou conjunto de teste), como demonstrado na Figura 1. Para reduzir

a variabilidade, várias rodadas da validação cruzada são realizadas utilizando diferentes partições, e calcula-se a média dos resultados da validação gerados em cada rodada.

O fato de não se usar o conjunto de dados completo ao treinar um modelo ajuda a definir quão bem esse modelo será quando ele é solicitado a fazer novas predições em diferentes dados. Para tanto, alguns dos dados são removidos antes que o treinamento comece, de forma que quando o treinamento está concluído os dados que foram removidos podem ser usados para testar o desempenho do modelo com “novos” dados.

O procedimento de obtenção da acurácia das predições genômicas por meio da técnica de validação cruzada é feita sob a suposição de que os dados divididos aleatoriamente entre conjuntos de treinamento e validação sejam independentes (Goddard & Hayes, 2007; Vazquez et al., 2010). No entanto, em aplicações de melhoramento animal os indivíduos apresentam graus variados de relações genéticas, e obter conjunto de dados de treinamento e validação independentes é raramente possível (Pérez-Cabal et al., 2012; Cleveland et al., 2012). Assim, a maneira com que as partições de treinamento e validação são construídas tem um efeito importante nos resultados da validação cruzada, sendo o nível de parentesco entre os indivíduos das diferentes partições um fator relevante (VanRaden et al., 2009). Todavia, a literatura disponível neste assunto é de certa forma contraditória.

Legarra et al. (2008), por exemplo, dividiram os dados de camundongos entre e dentro de famílias, com o último resultando em maior acurácia. Neste estudo, as covariâncias familiares podem ter sido aumentadas devido a efeitos ambientais comuns (como gaiola) que contribuem com semelhança adicional para os membros de famílias de irmãos completos. Da mesma forma, Habier et al. (2007), usando dados simulados, e Habier et al. (2010), com dados reais, relataram que para características de baixa a alta herdabilidade, os indivíduos no conjunto de dados de validação com maiores relações genéticas aditivas com indivíduos no conjunto de dados de treinamento tiveram maior acurácia de predição.

Luan et al. (2009), diferentemente, obtiveram uma vasta gama de valores de acurácia que dependiam do método utilizado para separar os

conjuntos de dados de treinamento e validação, e sugeriram que o parentesco não é muito importante. Eles descobriram que um esquema denominado *cohort* produziu maior acurácia quando os efeitos dos marcadores foram preditos pelo BLUP, ao passo que um esquema aleatório produziu melhores resultados quando se utiliza o método BayesB de Meuwissen et al. (2001).

No trabalho de Habier et al. (2011), em vez de se prender ao problema das relações genéticas para determinar a acurácia das predições, eles discutem a importância do DL. Esses autores aplicaram validações cruzadas em um cenário onde as relações genéticas aditivas entre touros de treinamento e validação foram baixas de forma que as acurácias dos VGGs foram dominadas pela informação de DL. Segundo esses autores, este critério revela o potencial da seleção genômica melhor do que a acurácia obtida usando conjuntos de dados de treinamento que contêm parentes próximos com touros na validação, como pais, irmãos completos e meios-irmãos. A razão é que os futuros candidatos à seleção em programas de melhoramento podem não ter parentes próximos no treinamento quando a seleção genômica é aplicada no início da vida.

2.1.3. Métodos de Predição do Valor Genético Genômico

Quanto aos métodos estatísticos empregados na predição do VGG, uma grande preocupação é em relação ao elevado número de efeito de marcadores a serem estimados, o qual pode ser muito maior do que o número de indivíduos (Meuwissen, 2007). Tal questão é referida como “a maldição da dimensionalidade.” Do ponto de vista estatístico, isso significa que não há graus de liberdade suficiente para estimar todos os efeitos simultaneamente usando o método de Quadrados Mínimos. Além disso, um alto grau de multicolinearidade pode existir entre os marcadores, resultando em um modelo superparametrizado. Para lidar com essas dificuldades, vários métodos estatísticos sofisticados foram propostos, tais como BLUP/Genômico, BayesA, BayesB (Meuwissen et al., 2001), BayesB Acelerado (Meuwissen et al., 2009), BayesCπ, BayesDπ (Habier et al., 2011), LASSO (*Least Absolute Shrinkage and Selection Operator*) (Usai et al., 2009), LASSO Bayesiano (Park & Casella, 2008; De Los Campos et al.,

2009; Cleveland et al., 2010), Aprendizado de Máquinas (Long et al., 2007), Regressão via Componentes Principais, Regressão via Quadrados Mínimos Parciais (Solberg et al., 2009), Regressão Kernel não Paramétrica via Modelos Aditivos Generalizados (Gianola et al., 2006), regressão RKHS (*Reproducing Kernel Hilbert Spaces*) (Gianola & Van Kaam, 2008), entre outros. Dentre esses métodos, o BLUP/Genômico e os bayesianos, BayesA e BayesB, são os mais empregados atualmente.

O BLUP/Genômico assume um modelo infinitesimal, isto é, um grande número de genes distribuídos ao longo dos cromossomos, com cada um apresentando pequeno efeito. Além disso, assume que os efeitos de QTL possuem distribuição normal e variância constante para todos os segmentos cromossômicos. Esse método contorna, por meio da estimação simultânea dos efeitos de todos os marcadores, a questão da necessidade de estimação de um grande número de efeitos a partir de um tamanho amostral restrito e, adicionalmente, o fato de muitos efeitos apresentarem colinearidade advinda do desequilíbrio de ligação entre os próprios marcadores. O BLUP/Genômico resulta em muitas estimativas de efeito de marcadores próximas a zero, mas não iguais a zero. Entretanto, tendo em vista que as análises genômicas normalmente envolvem mais de 30.000 marcadores é de se esperar que a maioria deles tenha efeito nulo. Assim, na soma dessas estimativas, esse efeito acumulado pode introduzir algum erro de predição (Meuwissen et al., 2001).

Segundo estes autores os métodos bayesianos consideram mais adequadamente a distribuição *a priori* dos efeitos dos QTLs e, assim como o BLUP/Genômico, permitem ajustar todos os efeitos alélicos simultaneamente. Detalhes da estimação bayesiana são apresentados por Resende (2002) e Sorensen & Gianola (2002).

O método BayesA equivale ao método BLUP, com um modelo infinitesimal, porém as variâncias dos segmentos cromossômicos diferem para cada segmento e são estimadas considerando a informação combinada dos dados e da distribuição *a priori* para essas variâncias. Esta distribuição é tomada como uma qui-quadrado invertida escalada. Para obtenção da distribuição *a posteriori* das variâncias, Meuwissen et al. (2001) adotaram o procedimento da amostragem de Gibbs.

No método BayesB muitos efeitos de marcadores são assumidos como zero, *a priori*, o que reduz o tamanho do genoma, concentrando-se nas partes que existem QTLs. Dito de outra maneira, o método BayesB assume uma distribuição *a priori* dos efeitos dos QTLs com alta densidade em $\sigma_{gi}^2 = 0$ e distribuição qui-quadrado invertida para $\sigma_{gi}^2 > 0$. Assim, considera-se que em muitos loci não existe variação genética, ou seja, não estão segregando.

Em princípio, o algoritmo de amostragem de Gibbs do método BayesA poderia também ser usado para BayesB. No entanto, o amostrador de Gibbs não irá mover-se através do *espaço de amostragem* inteiro do método BayesB. Isto ocorre porque a amostragem de $\sigma_{gi}^2 = 0$ não é possível, se $g_i' g_i > 0$. Por outro lado, a amostragem de $g_i = 0$ tem uma probabilidade infinitesimal se $\sigma_{gi}^2 > 0$. Assim, Meuwissen et al. (2001) decidiram usar o algoritmo de Metropolis-Hastings para amostrar $(\sigma_{gi}^2|y^*)$, onde a distribuição *priori*, $p(\sigma_{gi}^2)$, é usada como a distribuição condutora para sugerir atualizações para a cadeia de Metropolis-Hastings. O vetor y^* refere-se aos dados y corrigidos para a média e todos os outros efeitos genéticos exceto g_i.

Gianola et al. (2009) apontaram inconvenientes estatísticos dos métodos BayesA e BayesB em relação à *priori* para os efeitos de SNP. *A priori*, um efeito de SNP é igual a zero com probabilidade π, e normalmente distribuído com média zero e variância locus-específica com probabilidade $1 - \pi$. Esta variância locus-específica tem uma distribuição *priori* qui-quadrado invertida escalada com poucos graus de liberdade e um parâmetro escalar, S_a^2, que é frequentemente derivado de uma variância genética aditiva assumida sob determinadas hipóteses genéticas (Fernando et al., 2008; Gianola et al., 2009). A condicional completa *a posteriori* de uma variância locus-específica tem apenas um grau de liberdade adicional comparado com a sua *priori*, independentemente do número de genótipos ou fenótipos. Tal fato entra em conflito com o conceito de aprendizagem Bayesiana, e como consequência, a redução dos efeitos de SNP depende fortemente do S_a^2 como detalhado por Gianola et al. (2009). Este problema se torna ainda mais importante com o aumento da densidade de SNPs.

Segundo Habier et al. (2011), outra desvantagem dos métodos BayesA e BayesB é que a probabilidade do efeito de SNP ser zero (probabilidade π) é tratada como conhecida. No método BayesA $\pi = 0$, de forma que todos SNPs têm efeito diferente de zero, enquanto que no BayesB, $\pi > 0$, para acomodar a suposição de que muitos SNPs têm efeito zero. A redução dos efeitos de SNP é influenciada por π, e, por isso, este parâmetro deveria ser tratado como desconhecido e inferido a partir dos dados. Assim sendo, Habier et al. (2011) trataram π como desconhecido em BayesC e BayesD, os quais passaram a ser denominados BayesCπ e BayesDπ, respectivamente. No primeiro método, uma variância de efeito único comum a todos SNPs é usada em vez de variâncias locus específicas. Então, a influência de S_a^2 é menor. No segundo, o parâmetro escalar da *priori* qui-quadrado invertida referente às variâncias locus-específicas é tratado como desconhecido com sua própria *priori*.

Por meio de simulações demonstrou-se que métodos que usam todos marcadores simultaneamente resultam em maiores acurácias dos VGGs preditos e, como consequência, maior proporção da variância genética é explicada (Meuwissen et al., 2001; Gianola et al., 2003; Xu (2003); Habier et al., 2007; Habier et al., 2009). Esses resultados foram similares aos obtidos com dados reais (Moser et al. 2009; Hayes et al., 2009; VanRaden et al., 2009; Habier et al., 2010). Moser et al. (2009), por exemplo, compararam as acurácias e possíveis viés de diferentes métodos em bovinos de leite da raça Holandesa, na Austrália. Os autores concluíram que o método dos quadrados mínimos, que fundamenta em modelos que incluem um pequeno número de marcadores, apresenta baixa acurácia e alto viés de predição, e as acurácias dos VGGs preditos por métodos que estimam os efeitos de todos os SNPs ao mesmo tempo foram similares, apesar das diferentes pressuposições de cada modelo. Em estudo com bovinos da raça holandesa na Alemanha, Habier et al. (2010) concluíram que embora diferenças em acurácia dos VGGs preditos pelos métodos BayesB e BLUP sejam pequenas, o primeiro método explora melhor a informação proveniente do DL entre SNPs e QTL e, por isso, é um método mais promissor. Hayes et al. (2009) e VanRaden et al. (2009) avaliaram a efetividade prática da seleção genômica em gado de leite nos Estados

Unidos, Austrália e Nova Zelândia, e concluíram que o método BLUP mostrou-se aproximadamente igual a outros métodos mais complexos, em termos de acurácia. Isso ocorre para caracteres em que o modelo infinitesimal (muitos genes de pequeno efeito) se aplica. Os autores relataram também a importância da inclusão do efeito poligênico no modelo de avaliação genética, como forma de capturar e selecionar QTLs de baixa frequência não capturados pelos marcadores.

2.2. Inferência Bayesiana

Maiores esclarecimentos sobre inferência Bayesiana podem ser obtidos em O'Hagan (1994). Na inferência Bayesiana o teorema de Bayes é usado para atualizar a estimativa de probabilidade de uma hipótese, isto é, deriva-se a probabilidade *posteriori* como consequência de uma probabilidade *priori* e uma "função de verossimilhança" derivada de um modelo de probabilidade assumido para os dados a serem observados. Basicamente, no caso de dados contínuos interessa-se na inferência do vetor de parâmetros ϕ de um modelo de probabilidade (densidade) $p(y|\phi)$, onde y é o vetor dos dados observados. Se tratar os parâmetros como desconhecidos, e atribuir-lhes uma densidade de probabilidade "*priori*" $\pi(\phi)$, então o teorema de Bayes produz a densidade "*posteriori*"

$$\pi(\phi|y) = \frac{\pi(\phi)p(y|\phi)}{p(y)},$$

em que $p(y)$ é a densidade marginal para y obtida pela integração sobre a *priori* para ϕ. Visto que $\pi(\phi|y)$ é considerada como uma função de ϕ com y fixado (observado), pode-se reescrever

$$\pi(\phi|y) \propto \pi(\phi)p(y|\phi),$$

de forma que a *posteriori* é proporcional a *priori* vezes a função de verossimilhança.

Para modelos mais complexos esta descrição simples esconde alguns detalhes importantes. Se y são os dados observados, então ϕ representa tudo o mais no modelo que não é observado diretamente. Isto sem dúvida incluirá parâmetros "convencionais" do modelo, mas pode incluir também outros aspectos de um modelo, tais como parâmetros de "distúrbio", efeitos aleatórios e/ou dados perdidos. No contexto de modelos mais

complexos é frequentemente útil a partição $\phi = (\sigma, \theta)$, onde σ representa parâmetros "convencionais" do modelo de interesse inferencial direto e θ os demais aspectos. A capacidade da estrutura Bayesiana de lidar facilmente com modelos com tais características, tratando todos os aspectos de um modelo em uma forma unificada, faz com que a Bayesiana seja tão atrativa para muitos pesquisadores. *A priori,* em algumas situações, pode ser fatorada como $\pi(\phi) = \pi(\sigma, \theta) = \pi(\sigma)\pi(\theta|\sigma)$, de forma que o teorema de Bayes torna-se

$$\pi(\sigma, \theta|y) \propto \pi(\sigma)\pi(\theta|\sigma)p(y|\sigma, \theta), \tag{2.1}$$

em que a constante de proporcionalidade é independente de σ e θ. Se σ é de preocupação primária, então o interesse real será na marginal *a posteriori* obtida integrando θ para fora de (2.1), isto é

$$\pi(\sigma|y) \propto \pi(\sigma) \int \pi(\theta|\sigma)p(y|\sigma, \theta)\, d\theta,$$

que pode ser escrito como

$$\pi(\sigma|y) \propto \pi(\sigma)p(y|\sigma), \tag{2.2}$$

onde

$$p(y|\sigma) = \int \pi(\theta|\sigma)p(y|\sigma, \theta)\, d\theta \tag{2.3}$$

é a verossimilhança marginal para θ, e é tipicamente indisponível ou computacionalmente cara para avaliar. Antes de avançar, no entanto, é importante notar que há uma representação alternativa de (2.3) conforme o teorema de Bayes:

$$p(y|\sigma) = \frac{\pi(\theta|\sigma)\, p(y|\sigma, \theta)}{\pi(\theta|\sigma, y)}. \tag{2.4}$$

As dificuldades computacionais na inferência Bayesiana surgem da intratabilidade de integrais de alta dimensão. Normalmente, elas não são somente intratáveis analiticamente, mas também difíceis de se obterem numericamente. Apesar da constante de proporcionalidade em (2.2) parecer ser mais fácil de avaliar, a dificuldade da integração refere-se à avaliação da verossimilhança marginal (2.3). Assim, várias abordagens de integração direta têm sido propostas, como os métodos de Monte Carlo via Cadeias de Markov (MCMC).

2.3. Método de Monte Carlo

O método de Monte Carlo, originalmente desenvolvido por físicos, é uma ampla classe de algoritmos computacionais que geram amostras aleatórias, de forma que uma fração dessas amostras obedece a uma propriedade ou algumas propriedades (Robert & Casella, 2004). Devido ao elemento aleatório, os algoritmos de Monte Carlo são tipicamente empregados para resolverem problemas que são muito complicados de serem resolvidos analiticamente. Um bom exemplo é a Integração de Monte Carlo, que usa a geração de números aleatórios para calcular integrais. Quando se utiliza qualquer método de Monte Carlo é importante obter um grande número de amostras, visto que quanto mais amostras são usadas, mais acurado o resultado final será.

Suponha que se queira calcular uma integral complexa

$$\int_a^b h(x)dx. \tag{2.5a}$$

Se $h(x)$ puder ser decomposto em uma função $f(x)$ e uma função densidade de probabilidade $p(x)$ definida no intervalo (a, b), então observar-se que

$$\int_a^b h(x)dx = \int_a^b f(x)p(x)dx = E_{p(x)}[f(x)], \tag{2.5b}$$

de modo que a integral pode ser expressa como uma esperança de $f(x)$ sobre a densidade $p(x)$. Assim, se for amostrado um grande número de variáveis aleatórias $(x_1,\ldots, x_n)$ da densidade $p(x)$, tem-se

$$\int_a^b h(x)dx = E_{p(x)}[f(x)] \simeq \frac{1}{n}\sum_{i=1}^{n} f(x_i). \tag{2.5c}$$

Portanto, a Integração de Monte Carlo pode ser usada para aproximar as distribuições *posterioris* (ou marginais *posterioris*) necessárias para uma análise Bayesiana. Considere, por exemplo, a integral $I(y) = \int f(y|x)p(x)dx$, que pode ser aproximada por

$$\hat{I}(y) = \frac{1}{n}\sum_{i=1}^{n} f(y|x_i) \tag{2.6}$$

onde x_i são amostras da densidade $p(x)$.

2.4. Cadeias de Markov

Uma Cadeia de Markov é uma sequência de estados descrevendo uma caminhada aleatória através de um *espaço de estados*, com a propriedade de que o próximo estado na sequência é dependente apenas do estado atual da sequência, sendo os estados anteriores irrelevantes. Se $X_1, X_2, \dots, X_n$ é uma sequência aleatória com a condição de que

$$P(X_{n+1} = x | X_1 = x_1, X_2 = x_2, \dots, X_n = x_n) = P(X_{n+1} = x | X_n = x_n) \quad (2.7)$$

então essa sequência é uma Cadeia de Markov (Grimmett & Stirzaker, 2001), em que o conjunto de todos os valores possíveis de X é denomidado o *espaço de estados* da cadeia.

Visto que a determinação do próximo estado de uma cadeia de Markov é aleatório, não é possível predizer o estado exato da cadeia no futuro. Entretanto, sob determinadas condições, é possível determinar algumas propriedades estatísticas de longo prazo sobre a cadeia. Especificamente, se as probabilidades governando as transições entre os vários estados não mudam com o tempo (são homogêneas com o tempo), então uma matriz P única pode ser usada para representar a probabilidade de cada transição de estado possível. A matriz P é definida como probabilidade de transição. Se um vetor π existe tal que suas entradas somam para 1 e

$$\pi = \pi P, \quad (2.8)$$

então π é denominada “distribuição estacionária” da cadeia, isto é, π é uma distribuição de probabilidade que representa o comportamento do “estado estacionário” da cadeia. Intuitivamente a equação (2.8) declara que se uma cadeia de Markov está no passo t da distribuição estacionária, então ela permanecerá na distribuição estacionária no passo $t + 1$. Assim, desde que seja possível chegar a qualquer estado no *espaço de estados* a partir de qualquer outro estado (a Cadeia de Markov é irredutível) e que a cadeia seja aperiódica (não contém estados que reaparecem em intervalos regulares), então a frequência relativa dos estados ocorrendo na Cadeia de Markov tenderá para a distribuição estacionária independentemente do seu estado inicial, embora possa ter um grande número de transições de estado antes da amostragem da cadeia tornar-se representativa da distribuição

estacionária (Motwani & Raghavan, 1995). Quando existem estados suficientes para que isso ocorra a cadeia é dita ter alcançado o equilíbrio, ou convergido para sua distribuição estacionária ou de equilíbrio. A amostragem aleatória dos estados de uma Cadeia de Markov convergida é equivalente à amostragem direta de sua distribuição estacionária.

2.5. Métodos de Monte Carlo Via Cadeias de Markov

Basicamente, os métodos MCMC, tais como Metropolis-Hastings e Amostrador de Gibbs, funcionam a partir da simulação de amostras de alguma distribuição complexa de interesse. O valor da amostragem anterior é usado para gerar aleatoriamente o valor da amostra seguinte, gerando uma cadeia de Markov, em que as probabilidades de transição entre os valores da amostra são apenas uma função do valor da amostra mais recente. O estado da cadeia depois de um grande número de passos é então usado como uma amostra da distribuição desejada. A qualidade da amostra melhora com o número de passos. Detalhada explicação dos métodos MCMC encontra-se em Gilks et al. (1996), Tanner (1996), Robert & Casella (2004).

Um programa ou algoritmo para amostragem da cadeia de Markov é um processo iterativo de simulação. A cada iteração da simulação uma transição é proposta para mover a Cadeia de Markov do estado x para algum estado y, fazendo algumas pequenas alterações em x (Green, 1999). Ao longo das iterações a cadeia irá conduzir a um caminho aleatório através do *espaço de estados*, eventualmente convergindo na distribuição de equilíbrio/estacionária da cadeia (que foi organizada para ser o mesmo da distribuição *posteriori*). Embora a cadeia convirja na distribuição estacionária independentemente de seu estado inicial, isto pode levar muitas iterações. Uma vez que a cadeia de Markov esteja em andamento deve ser reservado um adequado período de "*burn-in*" (descarte da porção inicial da cadeia de Markov) antes que as amostras da distribuição de interesse sejam recolhidas.

2.5.1. Metropolis-Hastings

O núcleo de transição padrão (o algoritmo para decidir a probabilidade a qual uma alteração de estado proposto é aceito e aplicado) utilizado no MCMC é denominado Metropolis-Hastings e foi proposto em (Hastings, 1970) como um desenvolvimento de uma técnica anterior a partir de (Metropolis et al., 1953).

Suponha que o objetivo é obter amostras de alguma distribuição $p(\theta)$, onde $p(\theta) = f(\theta)/K$, em que a constante de normalização *K* pode não ser conhecida e muito difícil de calcular. O algoritmo de Metropolis (Metropolis & Ulam, 1949; Metropolis et al., 1953) gera uma sequência de amostras a partir dessa distribuição da seguinte forma:

1. Começa-se com qualquer valor inicial θ_0 que satisfaça f (θ_0) > 0.

2. Usando o valor atual θ, amostrar um **ponto candidato** θ^* a partir de uma **distribuição de salto** $q(\theta_1, \theta_2)$, que é a probabilidade de retornar um valor de θ_2 dado um valor anterior de θ_1. Esta distribuição é também referida como **distribuição proposta** ou **candidata-geradora**. A única restrição à densidade de salto no algoritmo de Metropolis é que seja simétrica, isto é, $q(\theta_1, \theta_2) = q(\theta_2, \theta_1)$.

3. Dado o ponto candidato θ^*, calcular a razão da densidade entre os pontos candidato (θ^*) e atual (θ_{t-1}),

$$\alpha = \frac{p(\theta^*)}{p(\theta_{t-1})} = \frac{f(\theta^*)}{f(\theta_{t-1})} \ . \tag{2.9}$$

Como está considerando a razão de $p(\theta)$ com dois valores diferentes, a constante de normalização K é cancelada.

4. Se o salto aumenta a densidade ($\alpha > 1$), aceitar o ponto candidato (usar $\theta_t = \theta^*$) e retornar para o passo 2. Se o salto diminui a densidade ($\alpha < 1$), então com probabilidade α aceita-se o ponto candidato, senão rejeita-o e retorna ao passo 2.

Pode-se resumir o amostrador de Metropolis computando

$$\alpha = min\left(\frac{f(\theta^*)}{f(\theta_{t-1})}, 1\right) \tag{2.10}$$

e depois aceitando um ponto candidato com probabilidade α (a probabilidade de um movimento). Isso gera uma cadeia de Markov $(\theta_0, \theta_1, \ldots, \theta_k, \ldots)$, visto que as probabilidades de transição de θ_t para θ_{t+1} dependem apenas de θ_t e

não de $(\theta_0, \dots, \theta_{t-1})$. Após um suficiente período de *burn-in* (de k passos), a cadeia aproxima da sua distribuição estacionária, em que amostras do vetor $(\theta_{k+1}, \dots, \theta_{k-n})$ são amostras de $p(x)$.

Uma revisão detalhada do método de Metropolis-Hastings pode ser obtida em Chib & Greenberg (1995).

2.5.2. Amostrador de Gibbs

O amostrador de Gibbs é um caso especial da amostragem de Metropolis-Hastings, em que o valor aleatório é sempre aceito, isto é, $\alpha = 1$ (Geman & Geman, 1984). A tarefa para especificar como construir uma Cadeia de Markov cujos valores convergem para a distribuição alvo continua. A chave para o amostrador de Gibbs é apenas considerar distribuições condicionais univariada - distribuição em que para todas as variáveis aleatórias, exceto uma, são atribuídos valores fixos. Tais distribuições condicionais são muito mais fáceis de simular do que distribuições conjuntas complexas e, normalmente, têm formas simples (sendo, geralmente, normais, inversa χ^2, ou outras distribuições *prioris* comuns). Assim, simula-se n variáveis aleatórias sequencialmente a partir das n condicionais univariadas em vez de gerar um único vetor *n*-dimensional num único caminho utilizando a distribuição conjunta completa.

Para introduzir o amostrador de Gibbs, considere uma variável aleatória bivariada (x, y), e suponha que se queira calcular uma ou ambas as marginais, $p(x)$ e $p(y)$. Por exemplo, $p(x) = \int p(y|x)dy$. A ideia por trás do amostrador é que é muito mais fácil considerar uma sequência de distribuições condicionais, $p(x|y)$ e $p(y|x)$, do que obter a marginal por integração da densidade conjunta $p(x, y)$. O amostrador começa com algum valor inicial y_0 para y e obtém x_0 através da geração de uma variável aleatória a partir da distribuição condicional $p(x|y = y_0)$. O amostrador usa então x_0 para gerar um novo valor de y_1, amostrando a partir da distribuição condicional com base no valor x_0. O amostrador procede como se segue

$$x_i \sim p(x|y = y_{i-1}) \quad \text{(2.12a)},$$

$$y_i \sim p(y|x = x_i) \quad \text{(2.12b)}.$$

Repetindo este processo k vezes, é gerada uma **sequência de Gibbs** de comprimento k, em que um subconjunto de pontos (x_i, y_j) são tomados como amostras simuladas da distribuição conjunta completa. Uma iteração de todas as distribuições univariadas é freqüentemente chamada de uma varredura do amostrador. Para obter o total desejado de m pontos da amostra (cada "ponto" no amostrador é um vetor de dois parâmetros), amostra-se a cadeia depois de um *burn-in* longo o suficiente para remoção dos efeitos de valores iniciais da amostragem, e faz-se o *thinning* (intervalo de utilização amostral) após o *burn-in*. A sequência de Gibbs converge para uma distribuição estacionária (de equilíbrio) que é independente dos valores iniciais (Tierney, 1994).

Quando mais de duas variáveis estão envolvidas, o amostrador é estendido na mesma forma. Por exemplo, se há quatro variáveis, w, x, y, z, o amostrador torna-se

$$w_i \sim p(w|x = x_{i-1}, y = y_{i-1}, z = z_{i-1})$$
$$x_i \sim p(x|w = w_i, y = y_{i-1}, z = z_{i-1})$$
$$y_i \sim p(y|w = w_i, x = x_i, z = z_{i-1})$$
$$z_i \sim p(z|w = w_i, x = x_i, y = y_i)$$

Gelfand & Smith (1990) ilustraram o poder do amostrador de Gibbs para tratar uma grande variedade de problemas estatísticos, enquanto Smith & Roberts (1993) mostraram a união natural do amostrador de Gibbs com estatísticas Bayesianas (na obtenção de distribuições *posterioris*). Além dessas referências, pode-se obter uma boa introdução para o amostrador de Gibbs em Casella & George (1992), e uma visão mais aprofunda pode ser obtida em Besag et al. (1995), Tanner (1996) e Lee (1997).

2.6. Computação Paralela

A computação paralela permite resolver problemas grandes e/ou complexos que são impraticáveis ou impossíveis de serem resolvidos em um único computador serial, especialmente com uma memória limitada. Enquanto na computação serial o programa é dividido em uma série discreta de instruções executadas uma após a outra, no processamento paralelo o programa é divido em partes (tarefas) independentes que são executadas

em diferentes processadores simultaneamente. Mais detalhes sobre este assunto podem ser obtidos em Wilkinson & Allen (1999) e Pacheco (2011).

O primeiro passo para desenvolver uma solução visando programação paralela deve ser a análise do problema para identificar o seu grau de paralelismo. Quanto mais independente for o processamento dos dados, maior será a possibilidade de definição de tarefas que poderão ser executadas simultaneamente em diferentes processadores. Contudo, programar em paralelo consiste não somente da capacidade de escrever programas que explorem múltiplos processadores, mas também, se necessário, da troca de mensagens (informações e dados) entre eles.

Normalmente, a paralelização do código introduz várias classes de potenciais *bugs* (erros de programação). A comunicação e a sincronização entre tarefas, por exemplo, estão entre os maiores obstáculos.

A sincronização pode ser definida como a coordenação de execução das tarefas nos diferentes processadores. Uma tarefa só irá iniciar depois que receber um sinal de OK de outra tarefa, ou conjunto de tarefas. Normalmente, a sincronização das tarefas está associada às comunicações entre os processadores.

Ao invés de desenvolver protocolos de comunicação usando bibliotecas de rede de baixo nível, é muito mais eficiente usar bibliotecas de transmissão de mensagens projetadas, especificamente, para o desenvolvimento de aplicações paralelas. As bibliotecas *Parallel Virtual Machines* (PVM) e *Message Passing Interface* (MPI) são as escolhas mais populares. MPI é uma biblioteca mais nova e sofisticada, e é provavelmente a melhor escolha para novos usuários, por ser mais fácil de usar e possuir uma interface de comunicação mais confiável, com mais opções e parâmetros por chamada do que a PVM. A biblioteca MPI é padronizada e portátil, projetada para funcionar em uma ampla variedade de computadores paralelos e disponível para as principais linguagens de programação científica.

A troca de mensagens entre tarefas paralelas pode ser feita através do barramento (*bus*) de uma estrutura de memória compartilhada ou através de uma rede de interconexão para acesso a uma estrutura de memória distribuída. A primeira estrutura refere a um computador pessoal e

a segunda a um *cluster* de computadores. Nos sistemas de memória distribuída, cada processador possui sua própria memória local, e nos de memória compartilhada todos os processadores têm acesso direto a uma memória física comum, que pode ser dividida em bancos de acesso independentes para aumentar a eficiência de acesso. Alguns sistemas podem também ser heterogêneos ou mistos, possuindo memória distribuída e compartilhada simultaneamente.

Nos computadores com arquitetura de memória compartilhada, as trocas de informações entre os processadores são ultrarrápidas, devido ao barramento que os conecta diretamente a uma memória comum. Contudo, a principal desvantagem dessa arquitetura é a falta de dimensionalidade entre memória e número de processadores, ou seja, adicionando mais processadores, o acesso a informações da memória RAM (*Random Access Memory* - Memória de Acesso Aleatório) e o tráfico de dados pela memória *cache*, devido à constante troca de mensagens entre processadores, chegam a ser tão intensas que podem comprometer o desempenho de cada processador. Outras desvantagens seriam a responsabilidade do programador para construção de sincronização que assegure “correto” acesso à memória global e o aumento do custo e a dificuldade de projetar e produzir máquinas com um número cada vez maior de processadores.

O desempenho de cada processador tende a diminuir à medida que mais processadores são utilizados para execução paralela do código devido a um fator denominado *overhead* ou sobrecarga do sistema. O *overhead* ocorre devido à tentativa do sistema de garantir a transferência sem erros de informações (dados) através da rede de interconexão de um *cluster* ou barramento de um computador pessoal. Portanto, quanto maior a troca de mensagens maior é o *overhead* do sistema. Outro fator que proporciona *overhead* ao usar trocas de mensagem é a sincronização de tarefas.

Cluster é um conjunto de computadores (chamados nós) contendo vários processadores conectados por uma rede de alta velocidade, designado para o processamento paralelo de alto desempenho (Manika, 1999). Os *clusters* têm como vantagens o escalonamento da memória com o número de processadores em uso e a possibilidade de uso irrestrito do

número de processadores, com cada um acessando, sem interferência e rapidamente, sua própria memória. Além disso, possui boa relação custo-benefício, uma vez que podem ser usados processadores padrões e rede de comunicação. As desvantagens são: necessidade de evitar intensa comunicação entre processadores devido ao elevado *overhead;* o usuário é responsável por muitos detalhes associados à comunicação de dados entre processadores, dificuldade em mapear estruturas de dados existentes baseado na memória global; e acesso não uniforme à memória. Portanto, para problemas computacionais que possam ser resolvidos com diversos processadores e com pouca comunicação entre eles, o uso dessas arquiteturas é de fundamental importância, uma vez que o desempenho computacional pode ser muito maior.

2.7. Computação Bayesiana em Paralelo

O uso da inferência Bayesiana para a análise de modelos estatísticos complexos aumentou em parte devido à crescente melhora da capacidade computacional. Há uma grande variedade de técnicas disponíveis para execução da inferência Bayesiana, mas a falta de tratabilidade analítica para a grande maioria dos modelos de interesse significa que a maioria das técnicas é numérica e, muitas vezes, computacionalmente exigente. De fato, para modelos não lineares de alta dimensão, os únicos métodos práticos para análise baseiam-se nas técnicas de Monte Carlo Via Cadeias de Markov (MCMC), que são notoriamente de computação intensiva, com algumas análises demandando semanas de processamento mesmo em computadores de alto desempenho. Portanto, é evidente que a utilização da tecnologia de computação paralela para análise de modelos complexos é de grande importância (Wilkinson, 2005). Gamerman (1997) fornece uma boa introdução sobre técnicas computacionais baseadas em MCMC.

Como na simulação de Monte Carlo as amostras são tomadas de forma totalmente independente uma da outra, este método pode ser denominado como “embaraçosamente” paralelo (Rosenthal, 2000; Doornik et al., 2006), ou seja, é perfeitamente possível ter vários processadores que trabalhem de forma independente para produzir amostras. O número de

processadores que pode executar as amostras é quase ilimitado. Os resultados da amostragem de cada processador podem ser combinados para formar o resultado final. Cada processador adicional aumenta a acurácia e reduz o tempo de execução do algoritmo

Embora o processamento paralelo possa ser aplicado de forma óbvia no contexto da simulação de Monte Carlo, técnicas para paralelizar algoritmos MCMC não são tão óbvios, exceto a abordagem natural de gerar múltiplas cadeias em paralelo. Enquanto a geração de cadeias paralelas é geralmente a abordagem mais fácil, nos casos em que o *burn-in* é um problema sério, é frequentemente desejável utilizar a paralelização com o intuito de acelerar a geração de uma única cadeia. Este é o grande desafio, pois a simulação da cadeia de Markov é um processo iterativo, em que a simulação do próximo valor da cadeia não pode começar até que o valor atual tenha sido simulado. Em outras palavras, o estado atual dos parâmetros da cadeia de Markov depende do estado imediatamente anterior (Wilkinson, 2005; Ren & Orkoulas, 2007).

É importante ter em mente que, para qualquer distribuição *posteriori*, $\pi(\phi|y)$, existem diversos algoritmos MCMC que podem ser implementados. Ao projetar um algoritmo MCMC para um problema especial, há uma série de acertos que são feitos. Muitas vezes, o esquema mais fácil de implementar é baseado em um amostrador de Gibbs ou no esquema de Metropolis dentro do Gibbs que cicla através de grande número de componentes de baixa dimensão. Embora de fácil implementação, esses esquemas muitas vezes possuem propriedades ruins de convergência se a dimensão de ϕ é alta. Os resultados desses esquemas têm geralmente uma proporção significativa da execução descartada no *burn-in*, e as iterações restantes são muitas vezes “refinadas” (mantendo-se, por exemplo, apenas 1 em 100 iterações) a fim de reduzir a dependência nas amostras utilizadas para análise. A questão do *burn-in* é de especial preocupação em um ambiente de computação paralela. O fato de que cada processador deve despender parte significativa de seu tempo produzindo amostras que serão descartadas estabelece sérias limitações no desempenho do algoritmo à medida que se aumenta o número de processadores (Rosenthal, 2000). Dado que há uma grande variedade de formas de melhorar a convergência

de algoritmos MCMC, incluindo agrupamento em blocos e reparametrização (Roberts & Sahu 1997), pode ser particularmente importante despender algum tempo melhorando a convergência do amostrador se a estratégia de cadeias paralelas será adotada. Se isto não é prático, então paralelização de uma única cadeia MCMC é possível, mas isto também apresenta dificuldades, e é improvável escalonar bem para um grande número de processadores (Wilkinson, 2005).

Uma consideração importante no contexto de múltiplas cadeias em paralelo é o ponto de partida de cada cadeia. Em princípio, desde que um adequado período de *burn-in* é usado, o ponto de partida não importa, e, portanto, qualquer ponto de partida poderia ser usado (de fato, todas as cadeias poderiam começar no mesmo ponto). Por outro lado, se cada cadeia pode ser iniciada com uma realização independente da distribuição alvo, então o *burn-in* não seria necessário, e a paralelização seria "perfeita". Rosenthal (2000) sugere que, quando possível, uma técnica de amostragem exata deve ser usada em cada processador, a fim de gerar a primeira amostra, e então a cadeia executa a partir desses pontos de partida mantendo todas iterações. Infelizmente, é difícil implementar uma amostragem exata para muitos modelos complexos, e assim uma solução pragmática é inicializar cada cadeia em um ponto de partida aleatório que sejam, de alguma forma, "muito dispersos" em relação ao alvo, e então faz-se o *burn-in* até que todas as cadeias convirjam (Wilkinson, 2005).

A teoria estatística sugere que muitas cadeias de Markov independentes podem ser executadas e suas amostras serem combinadas para inferência da *posteriori* (Gelman & Rubin, 1992). Contudo, há uma discussão se é ou não melhor executar uma cadeia longa ou várias cadeias menores. Geyer (1992) e Robert & Casella (2004) relatam que ao usar várias cadeias independentes algumas destas podem não convergir, sendo preferível usar uma cadeia longa. Segundo estes autores, muitas cadeias paralelas independentes podem artificialmente exibir um comportamento mais robusto que não corresponde a uma convergência real do algoritmo. Contrariamente, outros autores argumentam que a análise com várias cadeias independentes de fato melhora a convergência para a distribuição de equilíbrio. Inclusive, uma estratégia para melhorar as propriedades de

convergência dos amostradores MCMC é por meio da troca informações entre as cadeias (Chauveau & Vandekerkhove, 2002; Laskey & Myers, 2003; Chao, 2004; Drugan et al., 2004).

2.8. Aplicação da Computação Paralela na Seleção Genômica

Como demonstrado acima, vários métodos estatísticos sofisticados têm sido propostos para seleção genômica, tais como modelos bayesianos paramétricos e modelos de regressão não paramétricos. Computar esses modelos não é trivial, e alguns podem levar semanas ou meses para terminar (Wu et al., 2011). Assim, o longo tempo de computação pode limitar a aplicação desses métodos na seleção genômica.

Wu et al. (2011) discutiram a importância da computação paralela na seleção genômica. Eles utilizaram 147 bovinos Angus genotipados com o BeadChip Illumina Bovina SNP50 para a característica escore de marmoreio. O método estudado foi o LASSO Bayesiano, em que o pacote BRL do R (www.R-project.org) desenvolvido por de los Campos et al. (2009) foi utilizado. A amostragem do MCMC, que consistiu de 100.000 iterações, levou 23,1 horas. Para acelerar esse cálculo foi executado o MCMC em paralelo. A cadeia maior foi dividida em 10 cadeias menores, com cada implementação tendo 10.000 iterações, sendo 1.000 de *burn-in*. O processamento paralelo foi realizado no *cluster* da University of Wisconsin-Madison, o qual foi 7,7 vezes mais rápido do que o processamento serial. Assim, levou-se menos de 3 horas para todas as 10 tarefas paralelas terminarem. Essa diferença seria mais evidente se várias características fossem avaliadas. Suponha que se tenha 10 características, cada uma analisada por simulação de 100.000 iterações. Se executar essas tarefas sequencialmente, dada as mesmas especificações de computação, o processamento poderia levar até 10 dias. No entanto, em um *cluster* pode-se paralelizar 100 tarefas facilmente com amostras da *posteriori* para cada característica coletada a partir de um conjunto de 10 tarefas (10 cadeias), mantendo ainda um tempo de computação de cerca de 3 horas.

Wu et al., 2011 esperam que a computação de alto desempenho, por meio do uso da computação paralela e da infraestrutura com vários processadores usando uma rede de interconexão (*clusters*), tenha o

potencial de proporcionar mudanças revolucionárias para os programas de seleção genômica, tais como soluções rápidas, decisões mais informativas e produtos mais competitivos.

3. MATERIAL E MÉTODOS

Primeiramente, o algoritmo BayesCπ, proposto por Habier et al. (2011), foi escrito na linguagem FORTRAN (Apêndice A). Depois, foram realizados testes para determinar o *burn-in*, o *thin* e a convergência. Para tanto, utilizou-se a biblioteca BOA (*Bayesian Output Analysis* - Análise da Saída Bayesiana) do programa R (Smith, 2007). Duas estratégias de paralelismo foram utilizadas: múltiplas cadeias em paralelo e paralelização da própria cadeia (Apêndices B e C, respectivamente). Para paralelização do algoritmo foi utilizada a biblioteca MPI, sendo os códigos compilados pelo gfortran (gcc.gnu.org/wiki/GFortran) associado ao pacote OpenMPI (www.open-mpi.org).

Para comparar o desempenho dos algoritmos serial e paralelos foram utilizados dados simulados. O tempo médio de processamento para cada situação foi determinado a partir de três mensurações. Determinou-se também o quanto os algoritmos paralelos foram mais rápidos que o algoritmo sequencial correspondente por uma medida de aceleração denominada *speedup*. Essas computações foram realizadas em um computador pessoal com um processador de seis núcleos de processamento de 3.3 GHz, 16 GB de memória RAM e 14 MB de *cache*; e no cluster da Universidade Federal de Viçosa, constituído por 120 processadores de 2.77 GHz cada.

3.1. Dados Simulados

Foi utilizado um banco de dados simulado de bovinos de leite disponibilizado no XVI Workshop QTLMAS (qtl-mas-2012.kassiopeagroup.com/en/program.php). Uma população base (G0) de 1020 indivíduos não aparentados (20 machos e 1000 fêmeas) foi gerada com um genoma de 499,75 Mb de comprimento, consistindo de 5 cromossomos. Cada cromossomo tinha um tamanho de 99,95 Mb e 2.000 SNPs igualmente distribuídos (1 SNP a cada 0,05 Mb ou cM). Os haplótipos da população base foram gerados a fim de obter um decaimento estável do DL. Cada uma das quatro gerações seguintes (G1-G4) consistiu de 20 machos e 1.000 fêmeas, sendo geradas a partir do acasalamento aleatório de cada macho com 51 fêmeas. O conjunto de dados contém o pedigree de

4.100 indivíduos (apenas os 20 machos da G0 e demais animais nas gerações de G1 a G4). As gerações não se sobrepõem. Os animais das três primeiras gerações (G1-G3) têm informações de pedigree e fenótipo, e os 1020 animais jovens da G4 não têm informações de fenótipo, mas possuem informações completa de marcador. Portanto, os valores genéticos dos indivíduos da G4 podem ser preditos por meio da seleção genômica.

Duas características quantitativas de produção foram simuladas. As características foram correlacionadas e geradas de forma a imitar as características de produção leite (PL) e gordura (PG). Uma vez que tais características produtivas apenas se expressam no sexo feminino, os fenótipos referem-se a 3.000 fêmeas da G1 a G3. Os fenótipos são dados como desvios de produção do indivíduo.

Foram calculadas a acurácia de predição do método BayesCπ e a herdabilidade para cada característica. A acurácia foi obtida correlacionando o valor genético verdadeiro com o valor genético genômico predito. Como o tempo computacional de cada análise foi longo, para demonstrar o desempenho computacional com a computação paralela apenas a característica PL foi utilizada.

3.2. Modelo Estatístico

O método BayesCπ proposto por Habier et al. (2011) tem o objetivo de resolver as desvantagens dos métodos BayesA e BayesB desenvolvidos por Meuwissen et al. (2001) no que diz respeito ao impacto dos hiperparâmetros na redução dos efeitos dos SNPs e trata o parâmetro π como desconhecido.

O modelo estatístico geral pode ser escrito como

$$y = X\beta + \sum_{k=1}^{K} z_k\, a_k + e,$$

onde $\boldsymbol{y}$ é um vetor N × 1 de fenótipos da característica, $\boldsymbol{X}$ é uma matriz de incidência dos efeitos fixos em $\boldsymbol{\beta}$, $\boldsymbol{K}$ é o número de SNPs, $\boldsymbol{z_k}$ é um vetor N × 1 de genótipos do SNP k, $\boldsymbol{a_k}$ é o efeito genético aditivo do SNP, e $\boldsymbol{e}$ é um vetor de efeitos residuais. O único efeito fixo em $\boldsymbol{\beta}$ é a média total μ, e os

genótipos SNP foram codificados como o número de cópias de um dos alelos do SNP, isto é, 0, 1 ou 2.

3.2.1. Especificações da Priori

A *priori* para μ foi uma constante. A *priori* para a_k depende da variância σ_a^2 e da probabilidade *priori* π que SNP *k* tenha efeito zero:

$$a_k|\pi,\sigma_a^2 = \begin{cases} 0 & \text{com probabilidade } \pi, \\ \sim N(0,\sigma_a^2) & \text{com probabilidade } (1-\pi). \end{cases} \quad (3.1)$$

O π é tratado como desconhecido com *priori* uniforme (0,1). Os efeitos de SNP têm variância comum, σ_a^2. A *priori* desta variância segue uma distribuição qui-quadrada invertida escalada com graus de liberdade $\nu_a = 4{,}2$ e escalar S_a^2. O parâmetro S_a^2 foi derivado por Habier et al. (2011) a partir do valor esperado da variável aleatória distribuída de uma qui-quadrada invertida escalada, $E(\sigma_a^2) = \frac{\nu_a S_a^2}{\nu_a - 2} = \tilde{\sigma}_a^2$; portanto,

$$S_a^2 = \frac{\tilde{\sigma}_a^2(\nu_a - 2)}{\nu_a}, \quad (3.2)$$

onde $\tilde{\sigma}_a^2$ é a variância do efeito aditivo para um locus amostrado aleatoriamente, o qual pode estar relacionado com a variância genética aditiva explicada pelos SNPs, $\tilde{\sigma}_s^2$, como

$$\tilde{\sigma}_a^2 = \frac{\tilde{\sigma}_s^2}{(1-\pi)\sum_{k=1}^{K} 2p_k(1-p_k)}, \quad (3.3)$$

onde p_k é a frequência alélica do SNP *k* (Fernado et al., 2008; VanRaden, 2008; Gianola et al., 2009).

A *priori* para os efeitos residuais é normal com média zero e variância σ_e^2. A *priori* para σ_e^2 segue uma distribuição qui-quadrada invertida escalada com valor arbitrariamente menor que 4,2 para os graus de liberdade, e parâmetro escalar S_e^2. Este parâmetro escalar é produzido pela fórmula $\frac{\tilde{\sigma}^2(4{,}2-2)}{4{,}2}$, onde $\tilde{\sigma}^2$ é o valor *a priori* de σ_e^2.

3.2.2. Inferência dos Parâmetros do Modelo

O algoritmo MCMC para BayesCπ consiste apenas dos passos de Gibbs. As variáveis μ, a_k, σ_e^2, σ_a^2, S_a^2 e π são amostradas por meio de suas *posteriori*s condicionais completas. A decisão de incluir SNP *k* no modelo

depende da *posteriori* condicional completa para a variável indicadora δ_k, a qual é introduzida para este propósito. Esta variável indicadora é igual a 1 se SNP *k* é ajustado no modelo e é zero caso contrário. Seguindo as regras Bayesianas gerais, a probabilidade da *posteriori* condicional completa para que $\delta_k = 1$ é

$$p(\delta_k|y, ELSE) = \frac{p(y|\delta_k = 1, \sigma_a^2, ELSE)p(\delta_k = 1|\pi)}{p(y|ELSE)}.$$

Em que:

$p(y|ELSE) = p(y|\delta_k = 0, ELSE)p(\delta_k = 0|\pi) + p(y|\delta_k = 1, \sigma_a^2, ELSE)p(\delta_k = 1|\pi)$; $p(y|\delta_k = 1, \sigma_a^2, ELSE)$ denota a densidade do modelo de dados dado que SNP *k* está ajustado com variância de efeito comum σ_a^2 e os valores atualmente aceitos de todos os outros parâmetros; $p(y|\delta_k = 0, ELSE)$ é a densidade do modelo de dados sem SNP *k*, $p(\delta_k = 0|\,\pi) = \pi$ é a probabilidade *priori* que SNP *k* tem efeito zero; e, correspondentemente, $p(\delta_k = 1|\pi) = 1 - \pi$. $ELSE$ representa todos os outros parâmetros do modelo (Sorensen & Gianola, 2002).

A *posteriori* para a_k é dada por

$$a_k|\sigma_a^2 = \begin{cases} \sim N\left(\frac{x_k' r_k}{c_k}, \frac{\sigma_e^2}{c_k}\right) & \sigma_a^2 > 0, \\ 0 & \sigma_a^2 = 0. \end{cases} \qquad (3.4)$$

onde $r_k = y - \sum_{k' \neq k}^{K} x_{k'} a_{k'}$ e $c_k = x_k' x_k + \frac{\sigma_e^2}{\sigma_a^2}$.

A variância de efeito comum, σ_a^2, é amostrada a partir de uma *posteriori* condicional completa, que é uma qui-quadrado invertida escalada com graus de liberdade $\tilde{v}_a = v_a + m^{(t)}$ e escalar $\tilde{S}_a^2 = (v_a S_a^2 + \sum_{k=1}^{K} a_k^2)/\tilde{v}_a$, em que $m^{(t)}$ é o número de SNPs ajustados com efeito diferente de zero na iteração *t*. A variância residual, σ_e^2, é amostrada a partir de uma qui-quadrado invertida escalada com $\tilde{v}_e = v_e + n$ e $\tilde{S}_e^2 = (v_e S_e^2 + e'e)/\,\tilde{v}_e$ onde *n* é o número de indivíduos no treinamento. , π é amostrado de uma distribuição Beta $(K - m^{(t)} + 1, m^{(t)} + 1)$.

O valor de partida para π, π_0, determina S_a^2 como pode ser visto a partir das equações (3.2) e (3.3). Por outro lado, S_a^2 pode afetar até que ponto π é usado para diminuir os efeitos do SNP, por isso faz-se a estimativa de π. Como S_a^2 aumenta com π_0, menor redução é esperada através de S_a^2,

mas a redução pode ser aumentada com maiores valores de π, o que pode ser considerado como uma compensação para a menor redução através de S_a^2 (Habier et al, 2011). Os graus de liberdade da *priori* qui-quadrado invertida escalada, ν_a, também determina S_a^2 através da fórmula (3.2) e, portanto, pode afetar as estimativas de π.

3.3. Análise de Convergência

Como acontece com qualquer abordagem MCMC baseada em computação Bayesiana, o processamento da saída crua é uma parte vital da análise. Os sistemas mais utilizados para a análise de saída são CODA e BOA. O último é um aperfeiçoamento do primeiro e, por isso, o utilizado. BOA tem um excelente suporte para a análise de cadeias paralelas.

Os primeiros valores gerados do algoritmo MCMC antes de atingir a cadeia estacionária devem ser descartados. Este período inicial da amostragem é referido como aquecimento da cadeia ou *burn-in*. Os valores posteriores gerados após o *burn-in* são considerados como uma amostra da distribuição de interesse. A cadeia gerada após o *burn-in* é proveniente de uma distribuição estacionária ou de equilíbrio.

Além do *burn-in*, outro fator importante analisado foi a correlação entre amostras consecutivas de certo parâmetro. Elevada correlação significa que a cadeia tem má convergência. Para diminuir esta dependência foram usadas amostras intervaladas por meio de um processo denominado *thin*, ou intervalo de utilização amostral. As análises de convergência foram fundamentadas nos diagnósticos de Raftery & Lewis (1992) e Heidleberger & Welch (1983) para as variáveis σ_e^2, π, σ_a^2 e μ.

O diagnóstico de convergência de Raftery & Lewis é adequado para a análise de cadeias individuais. Se suficientes iterações MCMC estão disponíveis, o BOA lista o “limite inferior”, o número total de iterações necessário para cada parâmetro, o número de iterações iniciais para descartar (*burn-in*), e o intervalo de amostras (*thin*) para ser usado. Assim, o teste de convergência de Raftery & Lewis para a distribuição estacionária sugere o número de iterações necessárias para estimar o quantil especificado para a acurácia desejada utilizando amostras independentes.

O diagnóstico de convergência de Heidelberger & Welch também é apropriado para análise de cadeias individuais. Ele baseia-se na teoria de ponte Browniana e utiliza a estatística Cramer-von-Mises. Se há evidência de não estacionariedade, o teste é repetido após o descarte dos primeiros 10% das iterações. Este processo continua até que a cadeia resultante passe no teste ou mais do que 50% das iterações sejam descartadas. A falha para este teste indica que uma cadeia mais longa do amostrador MCMC é necessária a fim de alcançar a convergência. O BOA informa o número de iterações descartadas. Um teste denominado *halfwidth* é realizado sobre a porção da cadeia que passou do teste de estacionariedade anterior para cada variável. Falha do teste de *halfwidth* implica que uma cadeia mais longa do amostrador MCMC é necessária para aumentar a acurácia média da *posteriori* estimada.

Para cada característica, várias análises do algoritmo MCMC com diferentes valores iniciais e 100.000 iterações foram realizadas. O diagnóstico de Heidleberger & Welch para PL indicou que um *burn-in* de até 10.000 iterações é suficiente para a cadeia alcançar a estacionariedade e que é necessário ter uma cadeia com mais de 100.000 iterações para aumentar a acurácia média da *posteriori* estimada para o parâmetro μ, que teve maior dificuldade de convergência devido à alta correlação entre amostras sucessivas. Já o teste de Raftery & Lewis recomendou um *burn-in* de 1.274 iterações, *thin* de 151 e, pelo menos, 1.289.029 de iterações necessárias (Tabela 1). Baseado nas informações dos dois testes decidiu-se realizar 1.300.000 iterações, com um *burn-in* de 10.000. A grande cadeia de amostragem pós *burn-in* foi devido ao elevado *thin*. Assim, com uma quantidade adequada de amostras (8.600), tem-se uma melhor acurácia da média das amostras da *posteriori* para PL.

Tabela 1 - Diagnóstico de Raftery & Lewis para determinar o *burn-in* e o *thin* adequados para PL

Parâmetros	Iterações necessárias	Iterações descartadas	Intervalo de utilização amostral
$\boldsymbol{\mu}$	1.289.029	1.274	151
$\boldsymbol{\pi}$	800.510	533	93
$\boldsymbol{\sigma_e^2}$	27.618	18	8
$\boldsymbol{\sigma_a^2}$	745.107	745	86

Para a característica PG, fundamentando-se nos dois diagnósticos realizados, decidiu-se executar 2.050.000 iterações, com 40.000 de *burn-in*, e um *thin* de 120 (Tabela 2).

Tabela 2 - Diagnóstico de Raftery & Lewis para determinar o *burn-in* e o *thin* adequados para PG

Parâmetros	Iterações necessárias	Iterações descartadas	Intervalo de utilização amostral
$\boldsymbol{\mu}$	2.045.086	1.952	122
$\boldsymbol{\pi}$	312.320	305	61
$\boldsymbol{\sigma_e^2}$	42.360	30	10
$\boldsymbol{\sigma_a^2}$	256.581	172	43

Depois do *burn-in* e o *thin* terem sido ajustados, ambos diagnósticos foram realizados novamente para confirmar se a cadeia de fato convergiu. A autocorrelação entre as amostras e o erro de Monte-Carlo (Geyer, 1992) também foram determinados.

3.4. Programação Paralela

Duas estratégias foram utilizadas para paralelizar o algoritmo MCMC. A primeira é a mais simples e considerada a mais eficiente, baseando-se na execução de múltiplas cadeias em paralelo. A segunda concentrou-se na paralelização da própria cadeia de Markov, isto é, objetivou-se acelerar a execução de uma única cadeia. Cada estratégia se adéqua a situações específicas, como será discutido abaixo.

3.4.1. Múltiplas Cadeias Paralelas

Um método óbvio para paralelizar algoritmos MCMC é a construção de múltiplas cadeias de Markov executadas em paralelo (Apêndice B). Depois que cada processador passar do estágio de *burn-in*, cada uma irá fornecer sua própria sequência de amostras independentes (Wilkinson 2005).

As execuções preliminares determinaram que um *burn-in* (*b*) de 10.000 iterações foi necessário, seguido por uma cadeia principal em equilíbrio e 1.290.000 (*p*) iterações. Assim, todas as cadeias executadas em paralelo tiveram o mesmo *burn-in* e as *p* iterações restantes foram divididas

para os processadores em uso. Cada cadeia paralela foi inicializada com valores iniciais bastante dispersos.

A biblioteca MPI foi utilizada para iniciar certo número de tarefas ou processos através de diferentes processadores executando o mesmo código. Para assegurar que cada processador tivesse sua própria sequência de amostras independentes, a semente de cada um foi determinada como um número aleatório diferente. Assim, cada processador teve sua própria semente inicial de forma que as sequências de número aleatórios geradas fossem diferentes para cada processador.

As rotinas MPI necessárias para execução deste código são:

include 'mpif.h'

CALL MPI_INIT(erro)

CALL MPI_COMM_RANK (MPI_COMM_WORLD, meu_proc, erro)

CALL MPI_COMM_SIZE (MPI_COMM_WORLD, n_proc, erro)

CALL MPI_REDUCE (dados de envio, dados recebido, número de elementos, tipo de dados, operação, destino, MPI_COMM_WORLD)

CALL MPI_FINALIZE (erro)

Todos programas devem conter *include 'mpif.h',* que é usado para chamar a biblioteca MPI. *MPI_INIT (erro)* deve ser chamado antes de qualquer outra rotina MPI para inicializar o sistema de execução MPI. A rotina *MPI_COMM_RANK (MPI_COMM_WORLD, meu_proc, erro)* retorna o posto (*rank*) dos processadores chamados no comunicador especificado, o qual foi MPI_COMM_WORLD. O posto é retornado na variável *meu_proc.* A rotina *MPI_COMM_SIZE (MPI_COMM_WORLD, n_proc, erro)* retorna o número total de processadores no comunicador especificado pela variável *n_proc*. O comando *MPI_REDUCE () condensa o resultado de todos processadores em um processador.* A última operação MPI em qualquer programa paralelizado é *MPI_FINALIZE (erro),* a qual é utilizada para finalizar o ambiente MPI.

Geralmente, é razoável assumir que cada iteração leva aproximadamente a mesma quantidade de tempo do processador para computar, de modo que as iterações podem ser utilizadas como uma

unidade de tempo. Se *N* processadores estão disponíveis para executar cadeias paralelas, então, as iterações para a execução da cadeia principal podem ser divididas entre eles, de modo que $b + p/N$ iterações são necessárias em cada processador. Relembrando que *b* representa os primeiros valores gerados do algoritmo MCMC que devem ser descartados (*burn-in*) antes de atingir a cadeia estacionária e *p* os valores posteriores gerados após o *burn-in*, os quais são considerados amostras da distribuição de interesse.

O incremento de velocidade é obtido pela equação

$$SpeedUp(N) = \frac{b+p}{b+p/N} \xrightarrow[\infty]{N} \frac{b+p}{b}.$$

Observa-se que o *speedup* potencial é limitado para *b*, ou seja, se *b* for muito grande o ganho em velocidade de processamento não será satisfatório, sendo aconselhado utilizar outra estratégia de paralelização. Quando *N* processadores estão disponíveis o aumento de velocidade potencial oferecido, em princípio, para uma solução "perfeitamente paralela" é *N* (obtida no caso especial de $b = 0$). Como o período de *burn-in* está relacionado com a taxa de convergência da cadeia, e a taxa de convergência está relacionada com o comprimento da cadeia principal, *b* e *p* são muitas vezes relacionados (Wilkinson, 2005).

3.4.2. Cadeia Paralelizada

Para um algoritmo MCMC sofisticado com boas propriedades de convergência, uma abordagem de múltiplas cadeias paralelas seria o mais adequado. No entanto, muitos algoritmos com propriedades de convergência ruim podem ser acelerados paralelizando uma única cadeia (Apêndice C). Por conseguinte, os problemas de projeção e estratégias de paralelização do algoritmo devem ser considerados em conjunto (Rosenthal, 2000).

A presente estratégia de paralelização da própria cadeia ocorreu de forma que em cada iteração do algoritmo algumas equações (tarefas) eram realizadas concomitantemente. Os resultados gerados pelos diferentes processadores foram utilizados pelo processador mestre. Assim, a cada iteração da cadeia as troca de informações entre processadores foi muito intensa.

Além das rotinas MPI usadas para execução do código anterior, com exceção da *REDUCE (),* foram necessárias também:

CALL MPI_SEND (dados, número de elementos, tipo dos dados, destino, etiqueta, MPI_COMM_WORLD, erro)

CALL MPI_RECV (dados, número de elementos, tipo de dados, origem, etiqueta, MPI_COMM_WORLD, status, erro)

A transferência de dados entre os processadores requereram que operações de cooperação fossem executadas por cada processador, isto é, para cada operação de envio (*MPI_SEND)* teve uma operação de recebimento correspondente (*MPI_ RECV)*.

4. RESULTADOS E DISCUSSÃO

Nas Tabelas 3 e 4 estão demonstradas as médias e desvios padrão das densidades *a posteriori* das condicionais completas, erros padrão de Monte Carlo e testes de convergência do amostrador de Gibbs para as características produção de leite (PL) e produção de gordura (PG). As medidas de autocorrelação foram baixas, o que era esperado depois dos ajustes para *burn-in* e *thin*, e os erros padrão de Monte Carlo foram pequenos para os parâmetros das duas características analisadas, indicando que a convergência foi alcançada pelas cadeias de Gibbs. O fator de dependência foi menor que cinco para os parâmetros das duas características, o que confirma também que as cadeias de fato convergiram. O fator de dependência mede o aumento multiplicativo no número de iterações necessárias para atingir a convergência devido à correlação dentro da cadeia. Fatores de dependência maiores que 5,0 geralmente indicam falha de convergência e uma necessidade de reparametrização do modelo.

Tabela 3 - Médias e desvios padrão (DP) das densidades *a posteriori* para PL

Parâmetros	Média	DP	r	EpMC	Fator de dependência
μ	19,1147	171,977	0,09	11,6186	2,7471
π	0,9818	0,0045	0,02	0,0001	1,0405
σ_e^2	21670,61	680,461	0,03	12,5979	1,1046
σ_a^2	151,1186	39,939	0,02	0,9515	1,0107

r: correlação entre duas amostras sucessivas; EpMC: erro padrão de Monte Carlo.

Tabela 4 - Médias e desvios padrão (DP) das densidades *a posteriori* para PG

Parâmetros	Média	DP	r	EpMC	Fator de dependência
μ	-10,0211	7,7287	-0,05	0,3488	0,9552
π	0,9917	0,0021	0,01	0,00006	1,0384
σ_e^2	63,6554	1,9596	0,02	0,0407	1,0384
σ_a^2	1,1728	0,3741	0,02	0,0155	0,9819

r: correlação entre duas amostras sucessivas; EpMC: erro padrão de Monte Carlo.

A Tabela 5 traz as estimativas de herdabilidade e acurácia das características. Ressalta-se que os parâmetros estimados nos três

algoritmos foram iguais. As herdabilidades foram estimadas considerando a frequência alélica igual a 0,5 para todos os alelos ajustados no modelo.

Utilizando as três primeiras gerações (G1-G3), referentes aos 3.000 indivíduos que foram genotipados e fenotipados, como conjuntos de treinamento e validação ao mesmo tempo, a acurácia para PL e PG foram 0,84 e 0,85, respectivamente, significando que o método se ajustou bem aos dados.

Tabela 5 - Herdabilidade e acurácia para as características de PL e PG

Características	σ_s^2	σ_e^2	h^2	Acurácia
PL	13.751,7926	21.670,6100	0,39	0,84
PG	48,4661	63,6554	0,43	0,85

Na Figura 2 observa-se a redução do tempo de execução do algoritmo usando diferente número de processadores em um computador pessoal. Para esse banco de dados, com apenas 10.000 marcadores e 3.000 indivíduos, o algoritmo MCMC sequencial demorou 77,29 horas para finalizar. Usando múltiplas cadeias paralelas (em azul), observa-se que à medida que o número de processadores aumenta, o tempo de execução diminui consideravelmente até atingir um limite.

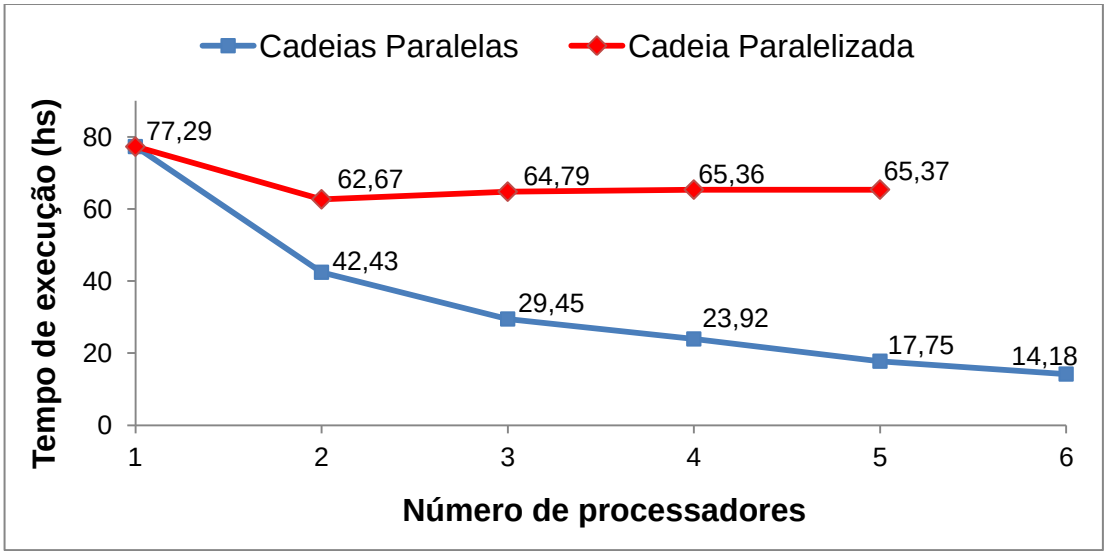

Figura 2 - Gráfico da curva de redução do tempo de computação para as duas estratégias de paralelização em um computador pessoal

Na situação do presente trabalho $p = 129b$, de tal forma que

$$SpeedUp(N) = \frac{b+p}{b+p/N} = \frac{1.300.000}{10.000 + 1.290.000/N} \xrightarrow[\infty]{N} 130.$$

Portanto, em teoria, utilizando múltiplas cadeias paralelas seria possível obter um algoritmo 130 vezes mais rápido que o sequencial. Contudo, para atingir este limite é necessário um grande número de processadores. No caso em particular, com seis processadores o *speedup* teórico foi, aproximadamente, 5,78. Já em uma situação onde se usa 20 processadores de um cluster, o *speedup* teórico seria 17,45. Na prática, entretanto, é impossível obter tais ganhos.

A Figura 3 mostra a taxa de aumento de velocidade (*speedup*) usando-se múltiplas cadeias paralelas em um computador pessoal. Ao se passar de um para dois processadores o desempenho quase dobrou, mantendo-se bom até seis processadores. No entanto, observa-se que a partir de quatro processadores o desempenho começa a diminuir de forma expressiva. O *speedup* obtido utilizando seis processadores foi de 4,83, menor que o limite teórico de 5,78. Embora a execução de múltiplas cadeias simultaneamente utilize pouca comunicação entre processadores, por se tratar de um computador com arquitetura de memória compartilhada, quanto mais processadores forem utilizados, mais dificuldade em acessar a memória RAM o processador terá. Consequentemente, o desempenho do algoritmo paralelo diminuirá expressivamente com o número de processadores em uso. Em outras palavras, isto equivale a dizer que computadores com esta arquitetura de memória não escalonam bem com o número de processadores utilizado.

Por outro lado, em estações de trabalho ou clusters fundamentados em uma estrutura de memória distribuída ou mista, espera-se que o *speedup* seja maior (Wilkinson, 2005), desde que a rede para troca de informações entre processadores seja de boa qualidade. Tal fato observa-se na Figura 4, onde o *speedup* para seis processadores, por exemplo, foi igual ao teórico, demonstrando que o rendimento computacional é melhor nestas estruturas, pois a redução de desempenho com mais processadores é menor do que nos computadores com arquiteturas de memória compartilhada.

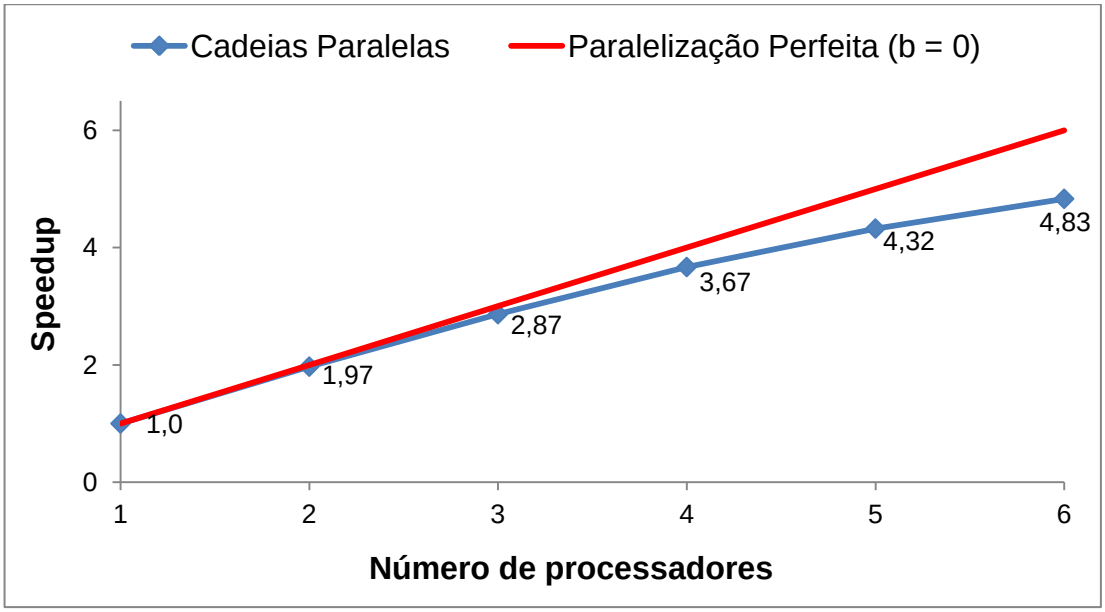

Figura 3 - Gráfico da curva de aumento de velocidade para uma abordagem de múltiplas cadeias paralelas em um computador pessoal

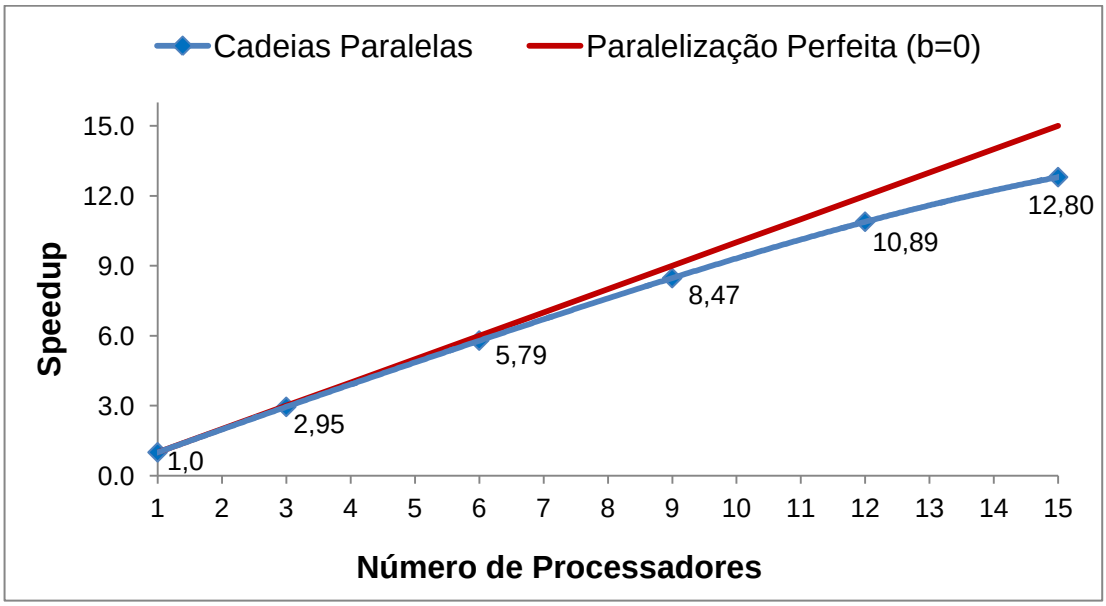

Figura 4 - Gráfico da curva de aumento de velocidade para uma abordagem de múltiplas cadeias paralelas em um cluster

Wu et al. (2011) demonstraram a importância da computação com múltiplas cadeias paralelas para seleção genômica usando *cluster*. Eles analisaram o método Bayessian Lasso separando a cadeia principal, com cem mil iterações, em 10 menores, com dez mil iterações cada e um *burn-in* de mil iterações. Utilizando apenas 147 bovinos Angus genotipados com o BeadChip Illumina Bovina SNP50 para a característica escore de marmoreio

o processamento da análise foi 7,7 vezes mais rápido do que o algoritmo serial.

É importante ressaltar que à medida que o número de marcadores e animais aumenta, provavelmente, mais dificuldade de convergência o algoritmo terá, e se o *burn-in* for longo o ganho de computação para execução de múltiplas cadeias pode vir a ser decepcionante. Uma saída seria desenvolver um algoritmo MCMC mais sofisticado com melhores propriedades de convergência e utilizar múltiplas cadeias paralelas (Wilkinson, 2005). Contudo, o interesse do programador pode ser de acelerar a convergência da própria cadeia por meio de sua paralelização, permitindo comparar o seu desempenho com as implementações em serial e de múltiplas cadeias paralelas.

O desempenho computacional referente à paralelização da própria cadeia encontra-se na Figura 2 (em vermelho). Usando dois processadores simultaneamente, o processamento foi 19% mais rápido que o algoritmo sequencial. Todavia, a partir de três processadores o desempenho computacional foi decaindo. O desempenho ficou comprometido devido à sincronização das tarefas e da intensa troca de mensagens entre os processadores a cada iteração do algoritmo. Assim, esta estratégia de paralelização funciona melhor em arquiteturas de memória compartilhada (computadores pessoais), devido à necessidade de acesso a uma memória comum e o barramento permitir o tráfico mais rápido das informações.

Embora esta estratégia de paralelização seja simples de programar, o desempenho computacional foi baixo. Assim, novas estratégias devem ser desenvolvidas, que sejam, inclusive, aplicadas com eficiência em *clusters*. Porém, o fato da simulação da cadeia de Markov ser um processo iterativo, no sentido de que a simulação do próximo valor da cadeia não pode começar até que o valor atual tenha sido simulado, dificulta a paralelização da cadeia de um modelo complexo.

Considerando θ como sendo o conjunto de parâmetros relativo aos efeitos dos marcadores $(a_1, a_2, \ldots, a_n)$ e σ representando todos os outros parâmetros do modelo (μ, π, σ_e^2, σ_a^2, etc), observou-se que as atualizações dos componentes de σ foram muito mais rápidos do que as atualizações dos componentes θ. Consequentemente, o algoritmo MCMC serial despendeu

quase todo o seu tempo realizando a atualização de θ. Portanto, é natural tentar acelerar o algoritmo paralelizando a etapa de atualização θ.

Normalmente, a forte dependência *posteriori* entre alguns componentes de σ e θ faz com que a cadeia de Markov demore mais para convergir. De fato, modelos fundamentados na amostragem de Gibbs podem ser lentos para convergir (Kuss & Rasmussen, 2005, Barbu & Zhu, 2005), principalmente se são grandes e apresentam dependências complexas. Na configuração de único processador, um método comum para acelerar um amostrador de Gibbs com lenta convergência é introduzir atualizações em blocos (Jensen & Kong, 1996; Roberts & Sahu, 1997; Hamze & de Freitas, 2004; Barbu & Zhu, 2005). Enquanto as atualizações usando condicionais individuais faz com que a cadeia convirja muito lentamente, o amostrador de Gibbs em blocos melhora a convergência por permitir que variáveis fortemente acopladas atualizem conjuntamente. Assim, como existem correlações parciais entre os componentes de θ (efeito de marcadores), agrupar os elementos de θ em blocos para serem atualizados simultaneamente além de diminuir o tempo de computação pode melhorar a convergência da cadeia.

Sucintamente, no amostrador de Gibbs em blocos as amostras das *posteriori*s são geradas iterativamente a partir dos seguintes passos:

$$\sigma_e^2|y,\theta \sim [(y-W\theta)'(y-W\theta)+\upsilon_e S_e^2]\chi_{n+\upsilon_e}^{-2} \quad (4.1)$$

$$\theta|y,\sigma_a^2,\sigma_e^2 \sim N\left(\hat{\theta}, C^{-1}\sigma_e^2\right) \quad (4.2)$$

onde $\hat{\theta} = C^{-1}W'y$, $C = \begin{bmatrix} 1'1 & 1'Z \\ Z'1 & Z'Z + I\frac{\sigma_e^2}{\sigma_a^2} \end{bmatrix}$, e *n* é o número de indivíduos.

Vários autores (Wilkinson et al., 2005; Newman et al., 2007; Asuncion et al., 2008; Doshi-Velez et al., 2009; Yan et al., 2009) propuseram métodos paralelos para acelerar a amostragem de Gibbs com base na amostragem em blocos. Contudo, embora o amostrador de Gibbs em blocos seja rápido, ele pode não ser operacional quando um grande número (50K ou mais) de marcadores $(a_j's)$ estão envolvidos, visto que a inversa de uma matriz ***C*** muito grande é difícil para a maioria dos computadores pessoais e *clusters*. De qualquer forma, testes de desempenho computacional para

algoritmos paralelizados com amostragem de Gibbs em blocos devem ser realizados.

Combinar a estratégia de paralelização de amostragem em blocos com a de múltiplas cadeias paralelas pode trazer ganhos expressivos no tempo de processamento dos algoritmos MCMC, embora a elaboração deste código seja um pouco mais complexa.

Apenas duas características foram citadas neste trabalho, mas nos programas de avaliação genética um número muito maior de características é avaliado, reforçando ainda mais a importância do uso dos *clusters* e do desenvolvimento de algoritmos paralelos aplicáveis a esta arquitetura de computadores.

Além disso, outro momento da avaliação genômica de grande demanda computacional é o processo de validação cruzada ou estimativa de rotação (Devijver, 1982; Geisser, 1993; Kohavi, 1995). Uma rodada da validação cruzada envolve a separação de uma amostra de dados em subconjuntos complementares, realizando a análise em um subconjunto (treinamento), e validando a análise no outro subconjunto (validação ou teste). Para reduzir a variabilidade das estimativas de acurácia, várias rodadas da validação cruzada são realizadas utilizando diferentes partições, e calcula-se a média dos resultados da validação gerados em cada rodada. Portanto, esta é uma fase da avaliação genômica de longa duração.

A escolha do tamanho dos conjuntos de dados de treinamento e validação é um aspecto importante, uma vez que existe uma dependência entre precisão do modelo no conjunto de treinamento e sobreajustamento no conjunto de validação. A recomendação usual é o conjunto de validação ser um quinto ou um décimo do conjunto de dados completo (Legarra et al., 2008). Assim, podem ser utilizados esboços denominados *5-fold cross validation* ou *10-fold cross validation*. Tomando a primeira situação como exemplo, no banco de dados em estudo, 600 indivíduos dos três mil genotipados e fenotipados podem ser utilizados em cada rodada da validação cruzada como conjunto de validação e os 2.400 restantes como conjunto de treinamento. A validação cruzada repete até que todos os indivíduos sejam utilizados uma vez no conjunto de teste. As iterações da validação cruzada podem ser processadas simultaneamente no cluster, e

cada rodada pode ser analisada utilizando 10 cadeias paralelas. Assim, 50 processadores seriam utilizados para avaliar a acurácia do modelo para cada característica.

A divisão do banco de dados em seguimentos (*folds*) é mais comum em aves e suínos. Em bovinos a estratégia mais empregada é correlacionar os animais mais velhos, genotipados e que já foram testados para progênie, com animais mais novos, apenas genotipados. Assim, utilizando os indivíduos da próxima geração (G4) como conjunto de teste e os indivíduos das três gerações anteriores como conjunto de treinamento a acurácia foi de 0,51 para PL. Isto pode ser explicado, em partes, pela possível queda do DL de uma geração para outra. Para comparação, o valor genético predito com base no pedigree iria resultar em uma acurácia de seleção de, aproximadamente, 0,4 para touros sem informações de progênie.

5. CONCLUSÕES

Como o método BayesCπ fundamenta-se na amostragem de Gibbs, que é um processo iterativo, se a predição genômica fosse conduzida em milhares de animais genotipados para 50.000 marcadores, ou mais, a análise poderia levar algumas semanas para concluir. Assim, com o crescente aumento dos bancos de dados utilizados na seleção genômica, aliado à crescente disponibilidade de *clusters* de computadores com memória distribuída, o uso da computação paralela torna-se inevitável.

Em um primeiro momento, a melhor solução de paralelização para os modelos complexos baseados nos métodos MCMC, desde que o *burn-in* não seja muito longo, é simplesmente gerar várias cadeias em processadores diferentes e depois combinar os resultados de forma adequada. Além da simplicidade, a grande vantagem desta estratégia é dimensionar bem com o número de processadores disponíveis. Nas situações que a cadeia de Markov apresenta má convergência, requerendo um longo período de *burn-in*, a melhor saída é ter algoritmos que foquem na paralelização da própria cadeia. A estratégia utilizada neste trabalho, apesar de apresentar simples aplicação, com poucas mudanças do código, não foi eficiente e é aplicável somente a computadores com arquitetura de memória compartilhada devido ao elevado overhead. Portanto, é necessário desenvolver estratégias de paralelização da cadeia que apresentem bom desempenho de processamento em ambas estruturas de memórias.

Para a determinação da eficiência de uma solução paralela o ambiente ideal seria a utilização de um *cluster* com um número grande de processadores e com uma rede de interconexão que permita a troca de mensagens entre os processos e o acesso à memória de maneira eficiente. A vantagem em usar códigos que possam ser processados em paralelo no *cluster* é devido a sua excelente relação custo-benefício.

Quando a prototipagem da solução paralela é feita em um computador pessoal com processador *multicore* (mais de um núcleo de processamento), o gargalo poderá ser determinado pelo barramento através do qual serão trocadas as mensagens e feitos os acessos à memória. Portanto, utilizar um grande número de processadores nesta arquitetura,

além de ser extremamente caro, pode proporcionar menores ganhos de desempenho quando comparado ao *cluster*.

Em aplicações paralelas nos computadores distribuídos (*clusters*), a comunicação entre processadores é um processo mais lento devido ao limite de velocidade da rede e tráfego de dados, em comparação com o *cache* local ou operação de memória ultrarrápido dos computadores com memória compartilhada. À medida que o número de processadores cresce, a velocidade de processamento pode cair drasticamente devido à sobrecarga de comunicação extra (*overhead*). Assim, é crucial sincronizar a informação entre os processadores a uma frequência relativamente baixa.

6. REFERÊNCIAS

ASUNCION, A.; SMYTH, P.; WELLING, M. Asynchronous distributed learning of topic models. In: NIPS, 2008.

BARBU, A.; ZHU, S. Generalizing swendsen-wang to sampling arbitrary posterior probabilities. **IEEE Transactions on Pattern Analysis and Machine Intelligence**, v.27, n.8, agosto, 2005.

BESAG, J.; GREEN, P. J.; HIGDON, D.; MENGERSEN, K. L. M. Bayesian computation and stochastic systems. **Statistical Science**, v.10, n.1, p.3-66, 1995.

CASELLA, G., GEORGE, E. I. Explaining the Gibbs sampler. **The American Statistical,** v.46, n.3, p.167-174, agosto, 1992.

CHAUVEAU, D.; VANDEKERKHOVE, P. Improving convergence of Hastings-Metropolis algorithm with an adaptive proposal. **Scandinavian Journal of Statistics,** v.29, p.13-29, 2002.

CHIB, S. Marginal likelihood from the Gibbs output. **Journal of the American Statistical Association**, v.90, n.432, p.1313-1321, 1995.

CHIB, S.; GREENBERG, E. Understanding the Metropolis-Hastings algorithm. **American Statistician.** v.49, n.4, p.327-335, 1995.

CHAO, C.T., Markov Chain Monte Carlo on optimal adaptive sampling selection. **Environmental and Ecological Statistics**, v.10, p.129-151, 2004.

CLEVELAND, M. A.; FORNI, S.; DEEB, N.; MALTECCA, C. Genomic breeding value prediction using three Bayesian methods and application to reduced density marker panels. **BMC Proceedings**, v.4, 2010.

CLEVELAND, M. A.; HICKEY; J. M.; FORNI, S. A Common Dataset for Genomic Analysis of Livestock Populations. G3 - **Genes, Genomes, Genetics.** v.2, n.4, p.429-435, 2012

DEKKERS, J. C. M. Commercial application of marker and gene assisted selection in livestock: strategies and lessons. **Journal of Animal Science**, v. 82, n.13, p.313-328, 2004.

DE LOS CAMPOS G.; NAYA, H.; GIANOLA, D.; CROSSA, J.; LEGARRA, A.; MANFREDI, E.; WEIGEL, K., COTES, J.M. Predicting quantitative traits with regression models for dense molecular markers and pedigree. **Genetics.** v. 182, n.1, p. 375-385, maio, 2009.

DEVIJVER, P. A.; KITTLER, J. **Pattern Recognition: A Statistical Approach**. London: Prentice-Hall, GB, 1982, 448 p.

DOORNIK, J. A.; SHEPHARD, N.; HENDRY, D. F. **Parallel computation in econometrics: A simplified approach.** University of Oxford, 2006.

DOSHI-VELEZ, F.; KNOWLES, D.; MOHAMED, S; GHAHRAMANI, Z. Large scale nonparametric bayesian inference: Data parallelization in the indian buffet process. In: NIPS, v.22, 2009.

DRUGAN, M. M.; THIERENS, D., Evolutionary Markov Chain Monte Carlo. **Lecture Notes in Computer Science**, v.2936, p. 63-76, 2004.

FERNANDO, R.L.; HABIER, D.; STRICKER, C.; DEKKERS, J.C.M.; TOTIR, L.R. **Acta Agric Scand A Anim Sci**, v.57, n.4, p.192-195, 2008.

GAMERMAN, D. **Markov Chain Monte Carlo: Stochastic Simulation for Bayesian Inference.** Texts in Statistical Science. Chapman and Hall, 1997, 245 p.

GELFAND, A. E.; SMITH. A.F.M. Sampling-based approaches to calculating marginal densities. **Journal of the American Statistical Association**, v.85, n.410, p.398-409, junho, 1990.

GELMAN, A.; RUBIN, D. B. Inference from iterative simulation using multiple sequences. **Statistical Science**, v.7, p.457-511, 1992.

GEMAN, S.; D. GEMAN. Stochastic relaxation, Gibbs distribution and Bayesian restoration of images. **IEE Transactions on Pattern Analysis and Machine Intelligenc.** v.6, p.721-741, 1984.

GEISSER, SEYMOUR. **Predictive Inference: An Introduction.** New York: Chapman and Hall. ISBN 0-412-03471-9, 1993, 264 p.

GEYER, C.J. Practical Markov Chain Monte Carlo. **Statistical Science**, v.7, n.4, p.473-511, novembro, 1992.

GIANOLA, D.; PEREZ-ENCISO, M.; TORO, M.A. On marker-assisted prediction of genetic value: Beyond the ridge. **Genetics**. v. 163, n.1, p.374-365, janeiro, 2003.

GIANOLA, D.; FERNANDO, R. L; STELLA, A. Genomic-assisted prediction of genetic value with semiparametric procedures. **Genetics**, v. 173, n.3, p.1761-1776, julho, 2006.

GIANOLA, D.; VAN KAAM, J. B. C. H. M. Reproducing kernel hilbert spaces regression methods for genomic assisted prediction of quantitative traits. **Genetics**, v. 178, n. 4, p.2289-2303, april, 2008.

GIANOLA, D.; DE LOS CAMPOS, G.; HILL, W.G.; MANFREDI, E.; FERNANDO, R. Additive Genetic Variability and the Bayesian Alphabet. **Genetics**. v.183, n.1, p.347-363, setembro, 2009.

GILKS, W. R.; RICHARDSON, S.; SPIEGELHALTER; D. J. **Markov chain Monte Carlo in practice**. 1 ed. Chapman and Hall, 1996, 512 p.

GRIMMETT, G.; STIRZAKER, D. **Probability and random processes**. 3 ed, Oxford University Press, 2001, 596 p.

GODDARD, M. E.; HAYES, B. J. Genomic selection. **Journal of Animal Breeding and Genetics**, v. 124, p. 323-330, 2007.

GREEN, P. J. MCMC in action: a tutorial. Given at ISI, Helsinki, August 1999.

HABIER, D.; FERNANDO, R.L.; DEKKERS, J.C. The impact of genetic relationship information on genome-assisted breeding values. **Genetics**. v. 177, p. 2389-2397, 2007.

HABIER, D.; FERNANDO, RL; DEKKERS, J.C.M.. Genomic Selection Using Low - Density Marker Panels. **Genetics.** v. 182, p. 343-353, 2009.

HABIER, D.; TETENS, J.; SEEFRIED, F.R.; LICHTNER, P., THALLER, G. The impact of genetic relationship information on genomic breeding values in German Holstein cattle. **Genetics Selection Evolution.** v. 42, 2010.

HABIER, D.; FERNANDO, R.L.; KIZILKAYA, K.; GARRICK, D.J. Extension of the bayesian alphabet for genomic selection. **BMC Bioinformatics.** v. 12, p.186, 2011.

HAMZE, F.; DE FREITAS, N. From fields to trees. **In: UAI**, 2004.

HASTINGS, W. K. Monte Carlo sampling methods using Markov chains and their applications. **Biometrika**, v.57, n.1, p.97-109, 1970.

HAYES, B.J., BOWMAN PJ, CHAMBERLAIN AJ, GODDARD ME. Invited review: Genomic selection in dairy cattle: progress and challenges**. Journal of Dairy Science.** v. 92, p. 433-443, 2009.

HEIDELBERGER, P. AND WELCH, P. Simulation run length control in the presence of an initial transient. **Operations Research**, v.31, p.1109-1144, 1983.

JENSEN, S.; KONG, A. Blocking gibbs sampling for linkage analysis in large pedigrees with many loops. **American Journal of Human Genetics**, v.65, n.3, p.885-901, setembro, 1999.

KOHAVI R. A study of cross-validation and bootstrap for accuracy estimation and model selection. In: Proceedings of International Joint Conference on Artificial Intelligence. v.2, p.1137-1143, 1995.

KUSS, M.; RASMUSSEN, C. E. Assessing approximate inference for binary gaussian process classification. **Journal of Machine Learning Research**, v.6, 2005.

LASKEY, K.B.; MYERS, J.W. Population Markov Chain Monte Carlo. **Machine Learning,** v. 50, p. 175-196, 2003.

LEE, P. **Bayesian Statistics: An introduction**. 2 ed, John Wiley, New York, 1997.

LEGARRA, A.; ROBERT-GRANIÉ, C.; MAN-FREDI, E.; ELSEN, J. M. Performance of genomic selection in mice. **Genetics**, v.180, p.611-618, 2008.

LONG, N.; GIANOLA, D.; ROSA, G. J. M.; WEIGEL, K. A; AVENDAÑO, S. Machine learning classification procedure for selecting SNPs in genomic selection: application to early mortality in broilers. **Journal of Animal Breeding and Genetics**, v. 124, p. 377-389, 2007.

LUAN, T.; WOOLLIAMS, J. A.; LIEN, S.; KENT, M.; SVENDSEN, M.; MEUWISSEN, T. H. E. The accuracy of genomic selection in Norwegian Red cattle assessed by cross-validation. **Genetics**, v.183, p.1119-1126, 2009.

MANIKA, G. W. Super-Computador a Preço de Banana, volume 2. Revista do Linux. 1999.

MATUKUMALLI, L.K.; LAWLEY, C.T.; SCHNABEL, R.D.; TAYLOR, J.F.; ALLAN, M.F; HEATON, M.P.; O'CONNEL, J.; MOORE, S.S.; SMITH, T.P.L.; SONSTEGARD, T.S.; TASSELL, C.P.V. Development and Characterization of a High Density SNP Genotyping Assay for Cattle. **PLoS ONE**, v.4, n.4, abril, 2009.

METROPOLIS, N.; ULAM, S. The Monte Carlo method. **Journal of the American Statistical Association**, v.44, n.247, p.335-341, setembro, 1949.

METROPOLIS, N.; ROSENBLUTH, A. W.; ROSENBLUTH, M. N.; TELLER, A. H.; TELLER, E. Equation of state calculations by fast computing machines. **The Journal of Chemical Physics**, v.21, n.6, p.1087-1092, 1953.

MEUWISSEN, T. H. E.; GODDARD, M. E.; HAYES, B. J. Prediction of total genetic value using genome-wide dense marker maps. **Genetics**, v. 157, p. 1819-1829, 2001.

MEUWISSEN, T. H. E. Genomic selection: marker assisted selection on genome-wide scale. **Journal of Animal Breeding and Genetics**, v. 124, p. 321-322, 2007.

MEUWISSEN, T. H. E.; Solberg, T. R.; Shepherd, R.; Woolliams, J. A. A fast algorithm for BayesB type of prediction of genome-wide estimates of genetic value, **Genetics Selection Evolution**, v.41, n.2, 2009.

MOSER, G.; TIER, B.; CRUMP, R. E.; KHATKAR, M. S.; RAADSMA, H. W. A comparison of five methods to predict genomic breeding values of dairy

bulls from genome-wide SNP markers. **Genetics Selection Evolution**, v.41, n.56, 2009.

MOTWANI, R.; RAGHAVAN, P. **Randomized algorithms**. Cambridge University Press, 1995, 492p.

MUIR, W. M. Comparison of genomic and traditional BLUP estimated breeding value accuracy and selection response under alternative trait and genomic parameters. **Journal of Animal Breeding and Genetics**, v. 124, p. 342-355, 2007.

NEWMAN, A.; ASUNCION, D.; SMYTH, P.; WELLING, M.. Distributed inference for latent dirichlet allocation. In: NIPS, 2007.

O'HAGAN, A. **Bayesian Inference.** Volume 2B of Kendall's advanced theory of statistics. London: Arnold, 1994.

PACHECO, P. **An Introduction to Parallel Programming**. 1 ed., Morgan Kaufmann Publishers, 2011, p.392.

PARK, T.; CASELLA, G. The Bayesian LASSO. **Journal of the American Statistical Association**, v.103, n.482, p.681-686, junho, 2008.

PÉREZ-CABAL, M. A.; VAZQUEZ, A. I.; GIANOLA, D.; GUILHERME. J. M. R; WEIGEL, K. A. Accuracy of genome-enabled prediction in a dairy cattle population using different cross-validation layouts. **Frontiers in Genetics**. v.3, 2012.

RAFTERY, A.E.; LEWIS, S. **How many iterations in the Gibbs sampler?** In: Bayesian statistics, v.4, Oxford University Press, Oxford, p.763-773, 1992.

R Development Core Team (2009). R: A language and environment for statistical computing. R Foundation for Statistical Computing, Vienna, Austria. ISBN 3-900051-07-0, URL http://www.R-project.org.

REN, R.; ORKOULAS, G. Parallel Markov chain Monte Carlo simulations. **Journal of Chemical Physics**, v.126, 2007.

RESENDE, M. D. V. **Genética biométrica e estatística no melhoramento de plantas perenes.** Brasília: Embrapa Informação Tecnológica, 2002. 975p.

RESENDE, M. D. V.; LOPES, P. L.; SILVA, R. L.; PIRES, I. E. Seleção genômica ampla (GWS) e maximização da eficiência do melhoramento genético. **Pesquisa Florestal Brasileira**, Colombo, n.56, p.63-67, 2008.

RESENDE, M. D. V.; RESENDE JÚNIOR, M. F. R.; AGUIAR, A. M.; ABAD, J. I. M.; MISSIAGGIA, A. A.; SANSALONI, S.; PETROLI, C.; GRATTAPAGLIA, D. **Computação da Seleção Genômica Ampla (GWS).** Embrapa Florestas, Colombo, PR. Documentos 210, 2010.

ROBERT, C.P.; CASELLA G. **Monte Carlo Statistical Methods**, Springer-Verlag, New York, 2004.

ROBERTS, G. O.; SAHU, S. K. Updating schemes, correlation structure, blocking and parameterisation for the Gibbs sampler. **Journal of the Royal Statistical Society**, v.59, n.2, p.291-317, 1997.

ROSENTHAL, J.S. Parallel computing and Monte Carlo algorithms. **Far East Journal of Theoretical Statistics.** v.4, p.207-236, 2000.

SMITH, B. J. An R Package for MCMC Output Convergence Assessment and Posterior Inference. **Journal of Statistical Software**, v.21, n.11, 2007.

SMITH, A. F. M.; ROBERTS, G. O. Bayesian computation via the Gibbs sampler and related Markov chain Monte-Carlo methods. **Journal of the Royal Statistical Society**, v. 55, p.3-23, 1993.

SOLBERG, T.R.; SONESSON, A.K.; WOOLLIAMS, J.A.; MEUWISSEN, T.H. Reducing dimensionality for prediction of genome-wide breeding values. **Genetics Selection Evolution**, v.41, n.29, 2009.

SORENSEN, D; GIANOLA, D. **Likelihood, Bayesian and MCMC Methods in Quantitative Genetics**. New York: Springer Verlag, 2002.

TANNER, M.A. **Tools for statistical inference.** 3 ed. Springer-Verlag, New York, 1996.

TIERNEY, L. Markov chains for exploring posterior distributions. **Annals of Statistics.** v.22, p.1701-1762, 1994.

USAI, M. G.; GODDARD M. E.; HAYES, B. J. LASSO with cross-validation for genomic selection. **Genetics Research**, v. 91, n. 6, p. 427-36, 2009.

VANRADEN, P. M., C. P. VAN TASSELL, G. R. WIGGANS, T. S. SONSTEGARD, R. D. SCHNABEL, J. F. TAYLOR, AND F. SCHENKEL. Invited review: Reliability of genomic predictions for North American Holstein bulls. **Journal of Dairy Science**, v.92, p.16-24, 2009.

VAZQUEZ, A.I.; ROSA, G.J.M., WEIGEL; K.A., DE LOS CAMPOS, G.; GIANOLA, D.; ALLISON, D.B. Predictive ability of subsets of SNP with and of parent average for several traits in US Holstein. **Journal of Dairy Science.** v.93, p.5942-5949, 2010.

XU, S. Estimating polygenic effects using markers of the entire genome. **Genetics**, v.163, p.789-801, 2003.

WILKINSON, B. ALLEN, C. M. **Parallel Programming: Techniques and Applications Using Workstation and Parallel Computers.** Prentice Hall. 1999.

WILKINSON, D.J. **Parallel Bayesian Computation**. **Handbook of Parallel Computing and Statistics**, Marcel Dekker/CRC Press, 2005, p.481-512.

WU, X.L.; BEISSINGER T.M.; BAUCK S.; WOODWARD B.; ROSA G.J.M.; WEIGEL K.A.; GATTI N.L.; GIANOLA, D. A primer on high-throughput computing for genomic selection. **Frontiers in Genetics,** v.2, fevereiro, 2011.

XVI QTLMAS WORKSHOP. Maio, 2012. Disponível em: <qtl-mas-2012.kassiopeagroup.com/en/program.php>. Acesso em: 10 de junho de 2012.

YAN, F.; XU, N.; QI, Y. Parallel inference for latent dirichlet allocation on graphics processing units. In: NIPS, 2009.

APÊNDICE A – Algoritmo Serial

```
program prog
use ManagerMod
use RandomGenaratorMod
implicit none

!Local variables.
real :: rhs, invLhs, mean, xpx, v0, v1, logPi, logPiComp, logDelta0, logDelta1
real :: probDelta1, u, lhs, add, aa, bb, start, finish, df, vara, scalec, vare, nua
real :: mean2pq, piMean, meanVar, varEffects, meanVare, pi
real, dimension(:), allocatable :: mean_b
integer :: j, loop, countLoci, thin
integer :: timeArray1(3), timeArray2(3) !Holds the hour, minute, and second.
integer :: numiter(2) !Number of iterations.

!Variables to reset the random number generator.
integer, dimension (100) :: seeds
integer :: k
seeds(1) = 12345

!>>>>>>>>>>>>>>> Input data <<<<<<<<<<<<<
numiter(1) = 10000
numiter(2) = 1290000
thin = 150

!Reading datafile.
call readTrainData(nrecords)
allocate(x(nrecords,nmarkers+1))
allocate(ycorr(nrecords))
allocate(b(nmarkers + 1))
allocate(meanb(nmarkers + 1))
allocate(mean_b(nmarkers + 1))
allocate(var(nmarkers))
allocate(ppa(nmarkers))
allocate(newMeanb(nmarkers + 1))

!Manipulating data.
call dataTrainManip(x)

!Initial values.
call initialVal(b, meanb, var, ppa, ycorr)
piMean = 0.0
meanVar = 0.0
meanVare = 0.0
mean_b = 0.0
vara = 200
pi = 0.5 !Probability that SNP k has zero effect.
mean2pq = 0.5
```

```
nua = 4.2
logPi = log(pi)
logPiComp = log(1-pi)
varEffects = vara/(nmarkers*(1-pi)*mean2pq)
scalec = varEffects*(nua-2)/nua

do loop = 1,2 !First loop is the burning

        call random_seed()
        call random_seed(size=k)
        call random_seed(put=seeds(1:k))

        call itime(timeArray1)
        call cpu_time(start)

        !>>>>>>>>>>>>>>>> MCMC Sampling <<<<<<<<<<<<<<<
        df = nrecords + 3 !df = degrees of freedom.
        open(1,file = 'result.out')

        do i = 1,numiter(loop)

                !Sample vare.
                vare = dot_product(ycorr,ycorr)/random_chisq(df)

                !sample intercept.
                ycorr = ycorr + x(:,1)*b(1)
                rhs = sum(ycorr)/vare
                invLhs = 1.0/(nrecords/vare)
                mean = rhs*invLhs
                b(1) = random_normal(mean,sqrt(invLhs))
                ycorr = ycorr - x(:,1)*b(1)
                meanb(1) = meanb(1) + b(1)

                !sample delta and effect for each locus
                do j = 1,nmarkers !j represents the locus.
                        ycorr = ycorr + x(:,j + 1)*b(j + 1)
                        rhs = dot_product(x(:,j + 1),ycorr)
                        xpx = dot_product(x(:,j + 1),x(:,j + 1))
                        v0 = xpx*vare
                        v1 = (xpx**2*varEffects + xpx*vare)
                        logDelta0 = -0.5*(log(v0) + rhs**2/v0) + logPi
                        logDelta1 = -0.5*(log(v1) + rhs**2/v1) + logPiComp
                        probDelta1 = 1.0/(1.0 + exp(logDelta0-logDelta1))
                        call random_number(u)
                        if (u < probDelta1) then
                                lhs = xpx/vare + 1.0/varEffects
                                invLhs = 1.0/lhs
                                mean = invLhs*rhs/vare
                                b(j + 1) = random_normal(mean,sqrt(invLhs))
                                ycorr = ycorr - x(:,j + 1)*b(j + 1)
```

```
                meanb(j + 1) = meanb(j + 1) + b(j + 1)
                ppa(j) = ppa(j) + 1
                var(j) = varEffects
            else
                b(j + 1) = 0.0
                var(j) = 0.0
            end if
        end do

        !Sample common variance.
        countLoci = 0
        add = 0.0
        do j = 1,nmarkers !j represents the locus.
            if (var(j) > 0.0) then
                countLoci = countLoci + 1
                add = add + b(j + 1)**2
            end if
        end do
        varEffects = (scalec*nua + add)/random_chisq(nua + countLoci)

        !Sample Pi
        aa = nmarkers - countLoci + 1
        bb = countLoci + 1
        pi = random_beta(aa,bb)
        scalec = ((nua - 2)/nua)*(vara/((1-pi)*nmarkers*mean2pq))
        logPi = log(pi)
        logPiComp = log(1-pi)

        !Thinning.
        if (mod(i,thin) == 0.0) then
            meanVar = meanVar + varEffects
            meanVare = meanVare + vare
            mean_b = mean_b + meanb
            piMean = piMean + pi
        end if

    end do

    write (1,'("piMean: ",f20.10)')piMean/numiter(loop)/thin
    write (1,'("meanVar: ",f20.10)')meanVar/numiter(loop)/thin
    write (1,'("meanVare: ",f20.10)')meanVare/numiter(loop)/thin

    !inital values to be used after burning
    newMeanb = mean_b !It will be used in cross validation
    meanb = 0.0
    mean_b = 0.0
    piMean = 0.0
    meanVar = 0.0
    ppa = 0.0
    var = 0.0
```

```
        !Getting the current time.
        call itime(timeArray2)
        call cpu_time(finish)
        write(1, '("Start: ", (i2,1x), "hours, ", (i2,1x), "minutes and ", (i2,1x),
        "seconds.")'), timeArray1
        write(1, '("End : ", (i2,1x), "hours, ", (i2,1x), "minutes and ", (i2,1x),
        "seconds.")'), timeArray2
        write(1, '("Time to run the MCMC: ",f10.3," seconds.")'), finish – start

end do
close(1)

!>>>>>>>>>>>>>>>>>>>>>>> Cross Validation <<<<<<<<<<<<<<<<<<<<<<
!Reading datafile.
call readTestData(n)

!Determining GEBV using testData.
allocate(z(n,nmarkers+1))
call dataTestManip(z)
allocate (ahat_test(n))

ahat_test = matmul(z, newMeanb)
call pearsonCorr(r_test, ahat_test)
print*,'r = ',r_test
deallocate(meanb, ahat_test, z, newMeanb)

stop
end program prog
```

APÊNDICE B – Algoritmo Para Múltiplas Cadeias Paralelas

```
program prog
use ManagerMod
use RandomGenaratorMod
implicit none
include 'mpif.h'

!Parallel variables.
integer, parameter :: master = 0
integer :: np, myproc, ierr, stat(MPI_STATUS_SIZE), thin
real :: piMeanGlobal, meanVarGlobal, meanVareGlobal, finishGlobal
real, dimension(:), allocatable :: meanbGlobal
integer :: timeArray1Global(3), timeArray2Global(3)

!Local variables.
real :: rhs, invLhs, mean, xpx, v0, v1, logPi, logPiComp, logDelta0, logDelta1
real :: probDelta1, u, lhs, add, aa, bb, start, finish, df, vara, scalec
real :: mean2pq, piMean, meanVar, varEffects, meanVare, vare, nua, pi
real, dimension(:), allocatable :: mean_b
integer :: j, loop, countLoci, numiter(2)
integer :: timeArray1(3), timeArray2(3) !Holds the hour, minute, and second.

!Variables to reset the random number generator.
integer :: clock, k
integer, dimension(:), allocatable :: seed

!>>>>>>>>>>>>>>>>> Parallel initiations <<<<<<<<<<<<<<<
call mpi_init(ierr)
call MPI_COMM_RANK(MPI_COMM_WORLD, myproc, ierr)
call MPI_COMM_SIZE(MPI_COMM_WORLD, np, ierr)
numiter(1) = 10000
numiter(2) = 129000
thin = 150

do j=0,np-1
        if (myproc==j) then
                call random_seed(size=k) !How big is the intrisic seed?
                allocate(seed(k))
                call system_clock(count=clock)
                seed = clock + 37 * (/ (i - 1, i = 1, k) /) + myproc
                call random_seed(put=seed) !Assign the seed
                deallocate(seed)
        end if
end do

!>>>>>>>>>>>>>>>>> Input data <<<<<<<<<<<<<<<<
!Reading datafile.
call readTrainData(nrecords)
```

```
allocate(x(nrecords,nmarkers+1))
allocate(ycorr(nrecords))
allocate(b(nmarkers + 1))
allocate(meanb(nmarkers + 1))
allocate(mean_b(nmarkers + 1))
allocate(var(nmarkers))
allocate(ppa(nmarkers))
allocate(newMeanb(nmarkers + 1))

!Manipulating data.
call dataTrainManip(x)

!Initial values.
call dataTrainManip(x)
call initialVal(b, meanb, var, ppa, ycorr)
piMean = 0.0
meanVar = 0.0
meanVare = 0.0
mean_b = 0.0
vara = 200
pi = 0.5 !Probability that SNP k has zero effect.
mean2pq = 0.5
nua = 4.2
logPiComp = log(1-pi)
varEffects = vara/(nmarkers*(1-pi)*mean2pq)
scalec = varEffects*(nua-2)/nua

!>>>>>>>>>>>>>>>>>>>> MCMC Sampling <<<<<<<<<<<<<<<<<<<<
df = nrecords + 3 !df is degrees of freedom.
open(1,file = 'output.out')

do loop = 1,2 !First loop is the burning

        call itime(timeArray1)
        call cpu_time(start)

        do i = 1,numiter(loop)
                !Sample vare.
                vare = dot_product(ycorr,ycorr)/random_chisq(df)

                !Sample intercept.
                ycorr = ycorr + x(:,1)*b(1)
                rhs = sum(ycorr)/vare
                invLhs = 1.0/(nrecords/vare)
                mean = rhs*invLhs
                b(1) = random_normal(mean,sqrt(invLhs))
                ycorr = ycorr - x(:,1)*b(1)
                meanb(1) = meanb(1) + b(1)

                !Sample delta and effect for each locus
```

```
do j = 1,nmarkers
        ycorr = ycorr + x(:,j + 1)*b(j + 1)
        rhs = dot_product(x(:,j + 1),ycorr)
        xpx = dot_product(x(:,j + 1),x(:,j + 1))
        v0 = xpx*vare
        v1 = (xpx**2*varEffects + xpx*vare)
        logDelta0 = -0.5*(log(v0) + rhs**2/v0) + logPi
        logDelta1 = -0.5*(log(v1) + rhs**2/v1) + logPiComp
        probDelta1 = 1.0/(1.0 + exp(logDelta0-logDelta1))

        call random_number(u)
        if (u < probDelta1) then
                lhs = xpx/vare + 1.0/varEffects
                invLhs = 1.0/lhs
                mean = invLhs*rhs/vare
                b(j + 1) = random_normal(mean,sqrt(invLhs))
                ycorr = ycorr - x(:,j + 1)*b(j + 1)
                meanb(j + 1) = meanb(j + 1) + b(j + 1)
                ppa(j) = ppa(j) + 1
                var(j) = varEffects
        else
                b(j + 1) = 0.0
                var(j) = 0.0
        end if
end do

!Sample common variance.
countLoci = 0
add = 0.0
do j = 1,nmarkers !k represents the locus.
        if (var(j) > 0.0) then
                countLoci = countLoci + 1
                add = add + b(j + 1)**2
        end if
end do
varEffects = (scalec*nua + add)/random_chisq(nua + countLoci)

!Sample pi.
aa = nmarkers - countLoci + 1
bb = countLoci + 1
pi = random_beta(aa,bb)
scalec = ((nua - 2)/nua)*(vara/((1-pi)*nmarkers*mean2pq))
logPi = log(pi)
logPiComp = log(1-pi)

!Thinning.
if (mod(i,thin) == 0.0) then
        meanVar = meanVar + varEffects
        meanVare = meanVare + vare
        mean_b = mean_b + meanb
```

```
                piMean = piMean + pi
            end if
        end do

        !Means.
        ppa = ppa/numiter(loop)
        piMean = piMean/(numiter(loop)/thin)
        mean_b = mean_b/(numiter(loop)/thin)
        meanVar = meanVar/(numiter(loop)/thin)
        meanVare = meanVare/(numiter(loop)/thin)

        call MPI_Reduce (mean_b, meanbGlobal, 10001, MPI_REAL, &
        MPI_SUM, master, MPI_COMM_WORLD, ierr)
        call MPI_Reduce (piMean, piMeanGlobal, 1, MPI_REAL, MPI_SUM, &
        master, MPI_COMM_WORLD, ierr)
        call MPI_Reduce (meanVar, meanVarGlobal, 1, MPI_REAL, &
        MPI_SUM, master, MPI_COMM_WORLD, ierr)
        call MPI_Reduce (meanVare, meanVareGlobal, 1, MPI_REAL, &
        MPI_SUM, master, MPI_COMM_WORLD, ierr)

        !Getting the current time for each processor.
        call itime(timeArray2)
        call cpu_time(finish)
        call MPI_Reduce(timeArray1, timeArray1Global, 3, MPI_INTEGER, &
        MPI_SUM, master, MPI_COMM_WORLD)
        call MPI_Reduce(timeArray2, timeArray2Global, 3, MPI_INTEGER, &
        MPI_SUM, master, MPI_COMM_WORLD)
        call MPI_Reduce(finish, finishGlobal, 1, MPI_REAL, MPI_SUM, &
        master, MPI_COMM_WORLD)

        if (myproc == master) then
            write (1,'("piMean: ",f20.10)')piMeanGlobal/np
            write (1,'("meanVar: ",f20.10)')meanVarGlobal/np
            write (1,'("meanVare: ",f20.10)')meanVareGlobal/np

            !>>>>>>>>>>>>>>> Cross Validation <<<<<<<<<<<<<
            !Reading datafile.
            call readTestData(n)

            !Determining GEBV and accuracy using test data.
            allocate(z(n,nmarkers+1))
            call dataTestManip(z)
            allocate (ahat_test(n))
            ahat_test = matmul(z,meanbGlobal/np)
            call pearsonCorr(r_test, ahat_test)
            write (1,'("r_test: ",f20.10)')r_test
            deallocate(ahat_test, z)

            !Final global time.
```

```
                write (1,'("Start: ",(i2,1x),"hours, ",(i2,1x),"minutes and ", &
                (i2,1x),"seconds.")') timeArray1Global/np
                write (1,'("End : ",(i2,1x),"hours, ",(i2,1x),"minutes and ", &
                (i2,1x),"seconds.")') timeArray2Global/np
                write (1,'("Time to run the MCMC: ",f10.3," seconds.")') &
                finishGlobal/np - start
                write (1,'("")')
                write (1,'("")')
            end if

            !inital values to be used after burning.
            meanb = 0.0
            piMean = 0.0
            meanVar = 0.0
            meanVare = 0.0
            mean_b = 0.0
            ppa = 0.0
            var = 0.0

        end do
        close(1)
        deallocate(x, meanb, ycorr, b, var, ppa, meanbGlobal, mean_b)

        call mpi_finalize (ierr)
        stop
        end program prog
```

APÊNDICE C – Algoritmo Para Cadeias Paralelizadas

```
program prog
use ManagerMod
use RandomGenaratorMod
implicit none
include 'mpif.h'

!Parallel variables.
integer, parameter :: master = 0
integer :: np, myproc, ierr, stat(MPI_STATUS_SIZE), w(nmarkers)
real, dimension (nmarkers) :: XPX, V0, V1

!Local variables.
real :: rhs, invLhs, mean, logPi, logPiComp, logDelta0, logDelta1, probDelta1
real :: lhs, add, aa, bb, start, finish, df, meanVare, vara, scalec, vare, nua
real :: mean2pq, piMean, meanVar, varEffects, pi
real, allocatable :: newMeanb(:), mean_b(:)
integer :: nloci, j, loop, countLoci,thin,timeArray1(3), timeArray2(3),numiter(2)

!Variables to reset the random number generator.
integer, dimension (12) :: old, seeds
integer :: k
seeds(1) = 12345

call mpi_init(ierr)
call MPI_COMM_RANK(MPI_COMM_WORLD, myproc, ierr)
call MPI_COMM_SIZE(MPI_COMM_WORLD, np, ierr)
numiter(1) = 10000
numiter(2) = 1290000
thin = 150

!Reading datafile.
call readTrainData(nrecords)
allocate(x(nrecords,nmarkers+1))
allocate(ycorr(nrecords))
allocate(b(nmarkers + 1))
allocate(meanb(nmarkers + 1))
allocate(var(nmarkers))
allocate(ppa(nmarkers))
allocate(newMeanb(nmarkers + 1))

!Initial values.
call dataTrainManip(x)
call initialVal(b, meanb, var, ppa, ycorr)
piMean = 0.0
meanVar = 0.0
meanVare = 0.0
mean_b = 0.0
```

```
vara = 200
pi = 0.5 !Probability that SNP k has zero effect.
mean2pq = 0.5
nua = 4.2
logPiComp = log(1-pi)
varEffects = vara/(nmarkers*(1-pi)*mean2pq)
scalec = varEffects*(nua-2)/nua

!************************* Master task only *************************
if (myproc == master) then
        call RANDOM_SEED()
        call RANDOM_SEED(SIZE=K)
        call RANDOM_SEED(PUT=SEEDS(1:K))
        do j = 1, nmarkers
                XPX(j) = dot_product(x(:,j + 1),x(:,j +1))
        end do
        do loop = 1,2 !First loop is the burning
                call itime(timeArray1)
                call cpu_time(start)
                do i = 1,numiter(loop)
                        call MPI_Recv(vare, 1, MPI_REAL, 2, 200, &
                        MPI_COMM_WORLD, stat, ierr)
                        call MPI_Recv(ycorr, nrecords, MPI_REAL, 2, 201, &
                        MPI_COMM_WORLD,stat, ierr)
                        call MPI_Recv(V0, nmarkers, MPI_REAL, 2, 202, &
                        MPI_COMM_WORLD, stat, ierr)
                        call MPI_Recv(V1, nmarkers, MPI_REAL, 2, 203, &
                        MPI_COMM_WORLD, stat, ierr)

                        !Thinning to vare.
                        if (mod(i,thin) == 0.0) then
                                meanVare = meanVare + vare
                        end if

                        !Sample delta and effect for each locus
                        call MPI_Recv(w, nmarkers, MPI_REAL, 1, 100, &
                        MPI_COMM_WORLD, stat, ierr)
                        do j = 1,nmarkers !j represents the locus.
                                ycorr = ycorr + x(:,j + 1)*b(j + 1)
                                rhs = dot_product(x(:,j + 1),ycorr)
                                logDelta0 = -0.5*(log(V0(j)) + rhs**2/V0(j)) + logPi
                                logDelta1 = -0.5*(log(V1(j)) + rhs**2/V1(j)) + &
                                logPiComp
                                probDelta1 = 1.0/(1.0 + exp(logDelta0-logDelta1))
                                if (w(j) < probDelta1) then
                                        lhs = XPX(j)/vare + 1.0/varEffects
                                        invLhs = 1.0/lhs
                                        mean = invLhs*rhs/vare
                                        b(j + 1) = random_normal (mean,sqrt(invLhs))
                                        ycorr = ycorr - x(:,j + 1)*b(j + 1)
```

```
                    var(j) = varEffects
                else
                    b(j + 1) = 0.0
                    var(j) = 0.0
                end if
            end do
            call MPI_SEND(ycorr, nrecords, MPI_REAL, 2, 204, &
            MPI_COMM_WORLD, ierr)
            call MPI_SEND(b(2:nmarkers + 1), nmarkers, &
            MPI_REAL, 3, 300, MPI_COMM_WORLD, ierr)

            !Sample common variance.
            countLoci = 0
            add = 0.0
            do j = 1,nmarkers !j represents the locus.
                if (var(j) > 0.0) then
                    countLoci = countLoci + 1
                    add = add + b(j + 1)**2
                end if
            end do
            varEffects = (scalec*nua + add)/random_chisq(nua + &
            countLoci)
            call MPI_SEND(varEffects, 1, MPI_REAL, 2, 205, &
            MPI_COMM_WORLD, ierr)
            call MPI_SEND(varEffects, 1, MPI_REAL, 4, 400, &
            MPI_COMM_WORLD, ierr)

            !Sample Pi
            aa = nmarkers - countLoci + 1
            bb = countLoci + 1
            pi = random_beta(aa,bb)
            scalec=((nua - 2)/nua)*(vara/((1-pi)*nmarkers*mean2pq))
            logPi = log(pi)
            logPiComp = log(1-pi)
            call MPI_SEND(pi, 1, MPI_REAL, 4, 402, &
            MPI_COMM_WORLD, ierr)
        end do
        call MPI_Recv(mean_b(1), 1, MPI_REAL, 2, 206, &
        MPI_COMM_WORLD, stat, ierr)
        call MPI_Recv(mean_b(2:nmarkers + 1), nmarkers, &
        MPI_REAL, 3, 301, MPI_COMM_WORLD, stat, ierr)
        call MPI_Recv(ppa, nmarkers, MPI_REAL, 3, 302, &
        MPI_COMM_WORLD, stat, ierr)
        call MPI_Recv(meanVar, 1, MPI_REAL, 4, 401, &
        MPI_COMM_WORLD, stat, ierr)
        call MPI_Recv(piMean, 1, MPI_REAL, 4, 403, &
        MPI_COMM_WORLD, stat, ierr)

        !Getting the current time.
        call itime(timeArray2)
```

```
                call cpu_time(finish)
                print '("Start: ",i2,1x,"hours, ",i2,1x,"minutes and ", &
                (i2,1x),"seconds.")',timeArray1
                print '("End : ",i2,1x,"hours, ",i2,1x,"minutes and ", &
                (i2,1x),"seconds.")',timeArray2
                print '("Time to run the MCMC: ",f10.3," seconds.")',finish - start
                write (1,'("piMean: ",f20.10)')piMean/numiter(loop)/thin
                write (1,'("meanVar: ",f20.10)')meanVar/numiter(loop)/thin
                write (1,'("meanVare: ",f20.10)')meanVare/numiter(loop)/thin

                !inital values to be used after burning
                newMeanb = mean_b !It will be used in cross validation
                mean_b = 0.0
                meanb = 0.0
                piMean = 0.0
                meanVar = 0.0
                ppa = 0.0
                var = 0.0
                call MPI_SEND(varEffects, 1, MPI_REAL, 2, 207, &
                MPI_COMM_WORLD, ierr)
        end do
        deallocate(x, ycorr, b, var, ppa)

        !>>>>>>>>>>>>>>> Cross Validation <<<<<<<<<<<<<
        !Reading datafile.
        call readTestData(n)

        !Determining GEBV using testData.
        allocate(z(n,nmarkers+1))
        call dataTestManip(z)
        allocate (ahat_test(n))
        ahat_test = matmul(z, newMeanb)
        call pearsonCorr(r_test, ahat_test)
        print*,'r = ',r_test
        deallocate(meanb, ahat_test, z, newMeanb)
end if

!*******************************Non-master tasks only ******************************
if (myproc == 1) then
        do loop = 1, 2
                do i = 1, numiter(loop)
                        call random_number(w)
                        call MPI_SEND(w, nmarkers, MPI_REAL, master, &
                        100, MPI_COMM_WORLD, ierr)
                end do
        end do
end if

if (myproc == 2) then
        do j = 1, nmarkers
```

```
            XPX(j) = dot_product(x(:,j + 1),x(:,j +1))
        end do
        do loop = 1, 2
            do i = 1, numiter(loop)
                !Sample vare.
                vare = dot_product(ycorr, ycorr)/random_chisq(df)

                !sample intercept.
                ycorr = ycorr + x(:,1)*b(1)
                rhs = sum(ycorr)/vare
                invLhs = 1.0/(nrecords/vare)
                mean = rhs*invLhs
                b(1) = random_normal(mean,sqrt(invLhs))
                ycorr = ycorr - x(:,1)*b(1)
                meanb(1) = meanb(1) + b(1)

                !Thinning to mean_b.
                if (mod(i,thin) == 0.0) then
                    mean_b(1) = mean_b(1) + meanb(1)
                end if

                do j = 1, nmarkers
                    V0(j) = XPX(j)*vare
                    V1(j) = (XPX(j)**2*varEffects + XPX(j)*vare)
                end do

                call MPI_SEND(vare, 1, MPI_REAL, master, 200, &
                MPI_COMM_WORLD, ierr)
                call MPI_SEND(ycorr, nrecords, MPI_REAL, master, &
                201, MPI_COMM_WORLD, ierr)
                call MPI_SEND(V0, nmarkers, MPI_REAL, master, &
                202, MPI_COMM_WORLD, ierr)
                call MPI_SEND(V1, nmarkers, MPI_REAL, master, &
                203, MPI_COMM_WORLD, ierr)
                call MPI_Recv(ycorr, nrecords, MPI_REAL, master, &
                204, MPI_COMM_WORLD, stat, ierr)
                call MPI_Recv(varEffects, 1, MPI_REAL, master, 205, &
                MPI_COMM_WORLD, stat, ierr)
            end do
            mean_b(1) = mean_b(1)/numiter(loop)
            call MPI_SEND(mean_b(1), 1, MPI_REAL, master, 206, &
            MPI_COMM_WORLD, ierr)
            call MPI_Recv(varEffects, 1, MPI_REAL, master, 207, &
            MPI_COMM_WORLD, stat, ierr)
        end do
end if

if (myproc == 3) then
do loop = 1,2
        do i = 1, numiter(loop)
```

```
                call MPI_Recv(b(2:nmarkers + 1), nmarkers, MPI_REAL, &
                master, 300, MPI_COMM_WORLD, stat, ierr)
                do j = 1, nmarkers !k represents the locus.
                        if (b(j + 1) /= 0.0) then
                                meanb(j + 1) = meanb(j + 1) + b(j + 1)
                                ppa(j) = ppa(j) + 1
                        end if
                end do

                !Thinning to mean_b.
                if (mod(i,thin) == 0.0) then
                        mean_b(2:nmarkers + 1) = mean_b(2:nmarkers + 1) + &
                        meanb(2:nmarkers + 1)
                end if
        end do
        meanb(2:nmarkers + 1) = meanb(2:nmarkers + 1)/numiter(loop)
        call MPI_SEND(mean_b(2:nmarkers + 1), nmarkers, MPI_REAL, &
        master, 301, MPI_COMM_WORLD, ierr)
        call MPI_SEND(ppa, nmarkers, MPI_REAL, master, 302, &
        MPI_COMM_WORLD, ierr)
end do
end if

if (myproc == 4) then
do loop = 1,2
        do i = 1, numiter(loop)
                call MPI_Recv(varEffects, 1, MPI_REAL, master, 400, &
                MPI_COMM_WORLD, stat, ierr)
                call MPI_Recv(pi, 1, MPI_REAL, master, 402, &
                MPI_COMM_WORLD, stat, ierr)

                !Thinning to meanVar and piMean.
                if (mod(i,thin) == 0.0) then
                        meanVar = meanVar + varEffects
                        piMean = piMean + pi
                end if
        end do
        call MPI_SEND(meanVar, 1, MPI_REAL, master, 401, &
        MPI_COMM_WORLD, ierr)
        call MPI_SEND(piMean, 1, MPI_REAL, master, 403, &
        MPI_COMM_WORLD, ierr)
        meanVar = 0.0
        piMean = 0.0
end do
end if
call mpi_finalize (ierr)
stop
end program prog
```