Attacking the problem of scalability in parallel Gauss-Seidel and Jacobi solvers for mixed model equations.

Per Madsen[1] and Martin Larsen[2]

[1]Danish Institute of Agricultural Sciences (DIAS), Department of Animal Breeding and Genetics, Research Centre Foulum, P. O. Box 50, DK-8830 Tjele, Denmark

[2]UNI C, Danish Computing Centre for Research and Education, DTU, Bld. 304, DK-2800 Lyngby, Denmark

Abstract

The history of the CEBUS project and some of its early experiences was described in: "The CEBUS project: History and overview" by Larsen and Madsen (This workshop).

Test of the second implementation showed that parallel speedup could be achieved, but communication among the processors could limit the parallel speedup substantially. An analysis of amount and pattern of communication indicates that it should be possible to reduce communication at the expense of a little more computation. This is achieved in a third implementation by splitting the MMEs in a local and a global subset, as described below.

Common to all tree implementations are the following: To obtain data parallelism (DP), records are sorted according to level codes of one of the effects in the model. Typically effects as herd-year-season or management group is used to obtain DP. Initially the master processor distributes disjoint sets of records to the slave processors. In the same pass through data, diagonal blocks for all effects except the one used to obtain DP are built, inverted and stored. In every round of iteration, each slave processor processes its part of the data, solve for the DP effect by Gauss-Seidel and accumulate contributions to the corrected right hand sides for all other effects in a local vector. When all data has been processed a complete corrected right hand side (*crhs*) can be build by summing over processors. In the first two implementations a global reduce operation was used to obtain the global *crhs*. In the third implementation only a small subset of the global *crhs* is needed. Each processor then solves for a part of the remaining effects by second order Jacobi.

The third implementation is characterised by a split of the system in equations within a processor (local equations) and equations across processors (global equations). The local equations only receive contributions from data on the processor in question, while the global equations receive contributions from data on more than one processor. This approach has been programmed, and test runs on models of dimensions between 1.2 and 30.1 mio. equations has shown close to linear scalability and in some cases super linear scalability.

1. Introduction

The use of Animal Models for prediction of breeding values, have led to the need for solving large systems of simultaneous linear equations. Although the Mixed Model Equation (MME) for an Animal Model is very sparse, it is in many practical applications too large to build and solve directly even in sparse format. Taking the Danish Dairy Cattle population as an example, the number of equations per trait is approx. 10 mio. A half-stored sparse representations of the coefficient matrix is approx. 500 GB, while the data is only 2 GB. Schaeffer and Kennedy (1986) described an iterative procedure for solving the MME without constructing the equations explicitly. This procedure is known as "Iteration on Data".

In many animal breeding programs, there is a trend to use more complex models for prediction of breeding values, i. e. going from single- to multi-trait models, changing to models with random regressions, or including more factors in the operational model. All these changes increase the dimension and complexity of the MME. At the same time, there is a requirement for more frequent evaluations. One way to overcome this computational challenge is to distribute the com-

putations on several computers, i.e. use parallel computations.

Previous Danish attempts to parallelise a solver for MME's based on "iteration on data" and a combination of Gauss-Seidel and Jacobi iteration, have shown that a parallel speedup could be achieved, but communication among the processors could limit the speedup substantially. For a detailed description of these experiences see Larsen and Madsen (1999).

This paper describes an attempt to attack the scalability problem by utilising the structure of the MME system to minimise communication.

2. Model

For reference purpose, let the general multivariate linear mixed model be:

$$\mathbf{y} = \mathbf{X_1} b_1 + \mathbf{X_2} b_2 + \sum_{\mathbf{i=1}}^{\mathbf{r}} \mathbf{Z_i u_i} + \mathbf{Z_a a} + \mathbf{e} \quad (1)$$

where *y* is a vector of observations on t traits, b_1 and b_2 are vectors of fixed effects, $\boldsymbol{u}_i$, i=1, 2,…,r are vectors of random effects, $\boldsymbol{a}$ is a vector of random additive genetic effects and *e* is a vector of random residuals. $\mathbf{X}_1$, $\mathbf{X}_2$, $\mathbf{Z}_i$, i=1, 2,…, r, and $\mathbf{Z}_a$ are known incidence matrices.

Assumptions for the random vectors are:

$$\begin{array}{ll} \mathbf{E[u_i] = 0}, \forall \mathbf{i}, & \mathbf{V[u_i] = G_i = G_{0_i} \otimes I}, \forall \mathbf{i} \\ \mathbf{E[a] = 0}, & \mathbf{V[a] = G_a = G_{0_a} \otimes A}, \\ \mathbf{E[e] = 0}, & \mathbf{V[e] = R = R_0 \otimes I}, \\ \mathbf{Cov[u_i, u_j'] = 0}, \mathbf{i \neq j}, & \mathbf{Cov[u_i, e'] = 0}, \forall \mathbf{i} \\ \mathbf{Cov[u_i, a'] = 0}, \forall \mathbf{i}, & \mathbf{Cov[a, e'] = 0} \end{array}$$

where $\mathbf{G}_{0i,}$, i=1, 2,…,r are (co)variance matrices for the traits influenced by the i'th of the r random effect other than animal, $\mathbf{G}_{0a}$ is the additive genetic (co)variance matrix for the t traits, $\mathbf{R}_0$ is the residual (co) variance matrix for the t traits and **A** is the additive genetic relationship matrix.

Let:

$$\mathbf{G} = \bigoplus_{\mathbf{i=1}}^{\mathbf{r}} \mathbf{G_i}, \mathbf{Z} = [\mathbf{Z_1 M Z_2 M \cdots M Z_r}],$$
$$\mathbf{u} = [\mathbf{u_1' M u_2' M \cdots M u_r'}]$$

Then the MME for (1) is:

$$\begin{bmatrix} \mathbf{X_1'R^{-1}X_1} & \mathbf{X_1'R^{-1}X_2} & \mathbf{X_1'R^{-1}Z} & \mathbf{X_1'R^{-1}Z_a} \\ \mathbf{X_2'R^{-1}X_1} & \mathbf{X_2'R^{-1}X_2} & \mathbf{X_2'R^{-1}Z} & \mathbf{X_2'R^{-1}Z_a} \\ \mathbf{Z'R^{-1}X_1} & \mathbf{Z'R^{-1}X_2} & \mathbf{Z'R^{-1}Z + G^{-1}} & \mathbf{Z'R^{-1}Z_a} \\ \mathbf{Z_a'R^{-1}X_1} & \mathbf{Z_a'R^{-1}X_2} & \mathbf{Z_aR^{-1}Z} & \mathbf{Z_a'R^{-1}Z_a + G_a^{-1}} \end{bmatrix} \times \begin{bmatrix} \hat{b}_1 \\ \hat{b}_2 \\ \hat{\mathbf{u}} \\ \hat{\mathbf{a}} \end{bmatrix} = \begin{bmatrix} \mathbf{X_1'R^{-1}y} \\ \mathbf{X_2'R^{-1}y} \\ \mathbf{Z'R^{-1}y} \\ \mathbf{Z_a'R^{-1}y} \end{bmatrix} \qquad \mathbf{(2)}$$

The blocks $\mathbf{X_1'R^{-1}X_1}$ and $\mathbf{Z'R^{-1}Z + G^{-1}}$ consists of even smaller diagonal blocks, with blocks of dimensions equal to the number of traits affected by the respective fixed or random effects. The diagonal block for the remaining fixed effects ($\mathbf{X_2'R^{-1}X_2}$) is relatively dense, but in most practical applications it is of limited size. The diagonal block for the animal equations ($\mathbf{Z_a'R^{-1}Z_a + G_a^{-1}}$) is sparse, but have a complex structure due to $\mathbf{G_a^{-1}}$**.** The off diagonal blocks shown in (2) are sparse, but can in principle contain nonzero elements anywhere.

A possible implementation of "Iteration on data" goes as follows:

1. Chose the fixed factor with the largest number of levels (typically a management

group effect) as fixed factor 1 (b_1). In a pre-processing step, data are sorted on level codes of that factor, and equation numbers are assigned to all factors affecting the records. The relative dense diagonal block $\mathbf{X}_2'\mathbf{R}^{-1}\mathbf{X}_2$ are formed and a LU-factorisation of this block is computed and stored. Also the small diagonal blocks in $\mathbf{Z}'\mathbf{R}^{-1}\mathbf{Z}+\mathbf{G}^{-1}$ and $\mathbf{Z}_a'\mathbf{R}^{-1}\mathbf{Z}_a+\mathbf{G}_a^{-1}$ are formed, inverted and stored.

2. The sorted data are processed in blocks corresponding to one level code of fixed factor 1 (a GS-group). These records generate one equation in single trait and t equations in multi-trait applications for the first fixed factor. Effects of all other factors affecting the records in this GS-group are absorbed into a corrected right hand side (*crhs*) based on the k^{th} iterates of b_2, $\boldsymbol{u}$ and $\boldsymbol{a}$. New iterate(s) (k+1) for the unknown (t unknowns) in b_1 is calculated by multiplying the element(s) in *crhs* with the inverse of the corresponding diagonal element (block) of $\mathbf{X}_2'\mathbf{R}^{-1}\mathbf{X}_2$. The block of records also contributes to elements for the other factors. These contributions times the appropriate components of b_1 from the current $(k+1)^{th}$ round of iteration and b_2, $\boldsymbol{u}$ and $\boldsymbol{a}$ of iterates from the previous $(k)^{th}$ round of iteration are accumulated in *crhs*. When all data are processed, a new (k+1) iterate for the complete b_1 vector has been calculated.

3. The pedigree file is processed for accumulation of contributions corresponding to off-diagonal elements in $\mathbf{Z}_a'\mathbf{R}^{-1}\mathbf{Z}_a+\mathbf{G}_a^{-1}$.

4. After this the equations for b_2, $\boldsymbol{u}$ and $\boldsymbol{a}$ are on a (block) diagonal form. Iterate k+1for b_2 is obtained by back solving based on the stored LU-factorisation. New iterates for $\boldsymbol{u}$ and $\boldsymbol{a}$ are calculated by multiplying elements in *crhs* with inverse diagonal elements (in multi trait cases t elements of *crhs* are multiplied by the inverse of the corresponding diagonal block).

Step 2 to 4 are repeated until a stopping criteria is met.

In the CEBUS[1] project there has been two earlier approaches of implementing a parallel solver for MME's based on the principles described above. The parallelisation used data parallelism obtained by distributing the data on several processors. Larsen and Madsen (1999) describe the two implementations in detail.

3. Limitations in previous implementations

In implementation #1, a standard text book recursive doubling technique was used for summing and distributing the vector of corrected right hand sides (*crhs*). This method lead to an enormous amount of communication in each round of iteration. In a test example with 9.5 mio. equations, the amount of communication were approximate 1.3, 3.7 and 9.5 GB when executed on 4, 8 or 16 processors respectively.

In implementation #2, each slave processor had a compressed *crhs* containing only elements that receive contributions from data on that slave. During the initial data distribution, mapping vectors were generated for each slave processor. Each slave processor had a mapping vector from the global to the local addressing space. The master processor had mapping vectors for each slave processor, to map from the slaves local to the global addressing space. The global reduce of *crhs* was implemented as follows: Each slave sends its compressed *crhs* to the master. The master unpack and sum the contributions from each slave. Then disjoint parts of the global *crhs* needed for the Jacobi iteration was sent to the slaves.

[1] Continuous Estimation of Breeding values Using Supercomputers. Esprit Project no. 24721

This approach lead to a considerable reduction in the amount of communication. For the example mentioned above the amount was 0.5, 0.7 and 1.3 GB. This corresponds to a reduction in communication of 62, 81 and 86 % respectively.

Extensive timing of the second approach on systems of different sizes showed, that parallel speedup could be achieved in some cases, but that communication still limited the speedup.

4. How to improve scalability

Based on experience from implementation #2, and inspired by the methods used for incorporating external information's into MME's (Henderson, 1975, Bonaiti and Boichard, 1995), a new approach has been programmed. In this implementation (#3), the MME's are rearranged into local and global equations. Local equations have direct links to data on one processor only. Global equations have direct links to data on more than one processor. The global equations would typically be equations for fixed effects other then the one solved by Gauss-Siedel, animal equations for sires and bull dams and equations for cow's in herds, that are distributed on more than one processor. In order to keep the number of global equations as low as possible, all data from a herd should always be on the same processor. The structure of the MME in (2) distributed on a master and 3 slave processors and after rearrangement is illustrated in Figure 1.

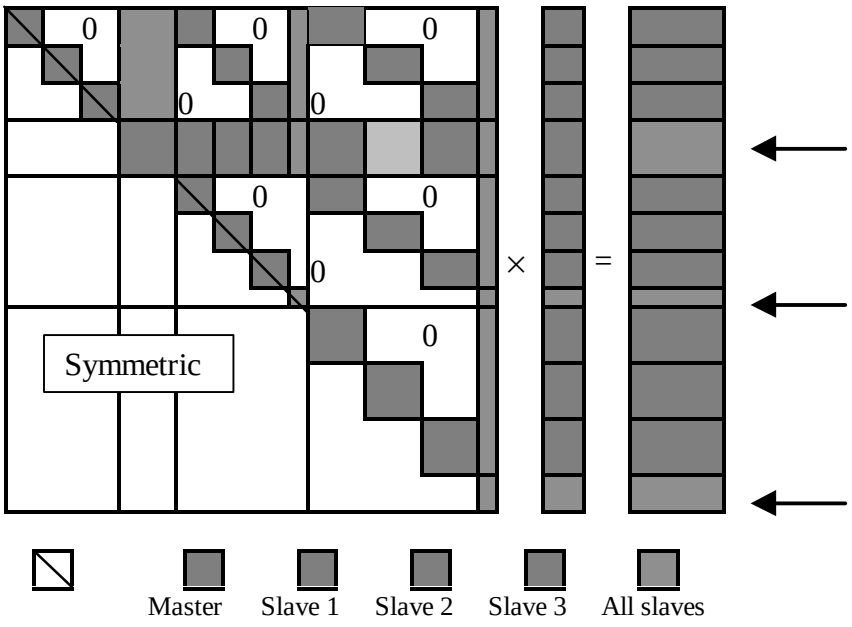

Figure 1. Structure of rearranged MME distributed on a master and 3 slave processors. The arrows ← ( )indicates where communication is needed.

Implementation #3 has the following steps:

1. Data distribution, where the master sends disjoint sets of data to a number of slave processors and collects information on which equations that are linked to data on each of the slave processors. When all data are distributed, equations linked to data on more than one slave are classified as a global equation, while the remaining equations are local to the slave having the data they are linked to. Each slave processor then pass through its part of data and convert class codes into a local addressing space, and at the same time build a mapping vector between the local and the global addressing space. A subset of the pedigree file containing all animals with data on that slave and all their ancestors is extracted on each slave processor.

2. Each slave perform the Gauss-Seidel step on its subset of data as outlined in step 2 on page 3.

3. Each slave processor process its subset of the pedigree file

4. Each slave sends the parts of its *crhs* vector that corresponds to global equations to the master. The master sums over processors, and broadcast the summed *crhs* for global equations to all slave processors.

5. On the master, a new iterate for b_2 are obtained by back solving based on the stored LU-factorisation and it is broadcasted to the slaves. On each of the slave processors, new iterates for the local and global part of $\boldsymbol{u}$ and $\boldsymbol{a}$ are calculated and the new iterate for b_2 are received.

6. Step 2 to 5 are repeated until a stopping criteria is met.
7. Finally the complete solution vector is build based on solution vectors and mapping vectors from each of the processors.

It is important to notice that in this implementation it is only the global equations that generate communication in each round of iteration. The amount of communication for the examples used above are 17.6, 35.5 and 70.2 MB respectively. Compared to implementation #2 the amounts of communication is reduced by 96, 95 and 94 % respectively

5. Test runs

Implementation #3 has been tested on the following four data sets:

1. Protein yield for all Red Danish dairy cows (RD)

2. Protein, fat and milk yield for all Red Danish dairy cows (RD3)

3. Protein yield for all dairy cows in Denmark (AC)

4. Protein, fat and milk yield for all dairy cows in Denmark (AC3)

The model used for the RD was the same as used in the routine evaluation as described by Pedersen et al. (1999). For AC the multibreed model proposed by Madsen et al. (1997) was used. For the multi-trait models, the model from the corresponding single-trait analysis was used for each of the traits, and genetic and residual correlation's among the traits was included. The dimensions of the MME to solve were 1.2., 4.3, 9.5 and 30.1 mio. for RD, RD3, AC and AC3 respectively.

The computer environment was an IBM RS/6000 SP at UNI•C. This computer has 80 processors (48 with 512 MB and 32 with 256 MB memory) for parallel computation. The timing of the new approach was made on processors with 512 MB. The elapse time per round of iteration for the four test examples is shown in Fig. 2.

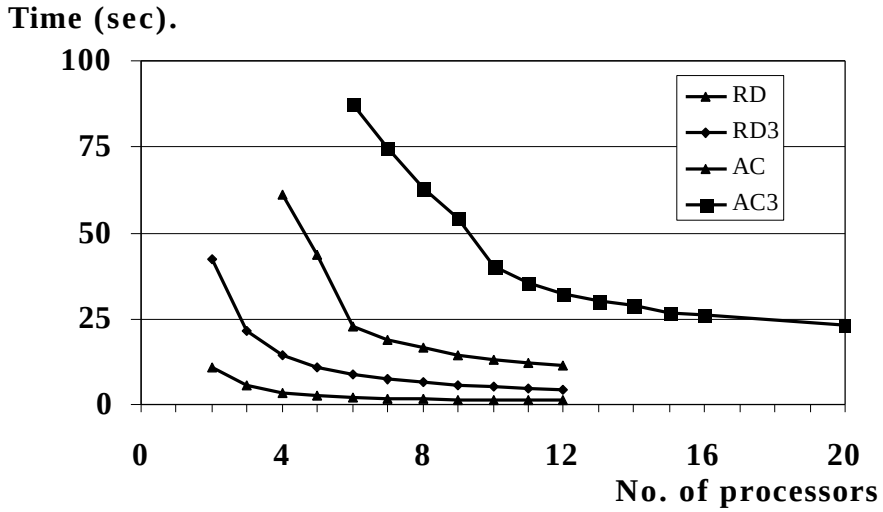

Figure 2. Elapse time per round of iteration for the four test example on varying number of proce ssors.

A measure of parallel speedup is relative speed, which is defined as the reciprocal of elapse time. If speedup is 100% then relative speed is doubled when the number of proce s-sors are doubled. The relative speed for the four test examples is in Fig 3. In order to make the curves comparable relative speed has been normalised so that the speed is set to 6 when executed on 6 processors, which is the lowest number of processors all exa m-ples can be executed on due to memory r e-quirement.

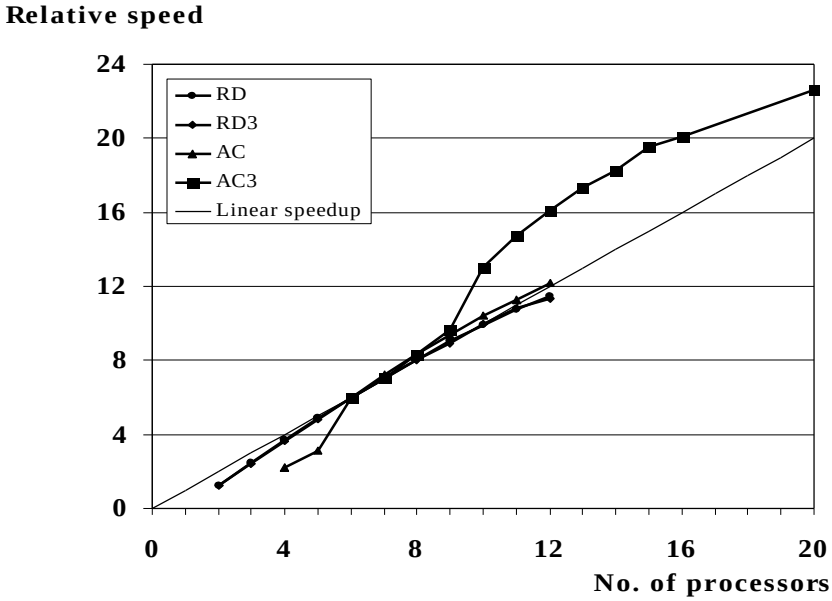

Figure 3. Relative speed for the four test examples. Speed is normalised so speed is 6 when executed on 6 processors.

The speedup curves for RD, and RD3 show close to linear speedup. From 2 to 8 processors super scalability is achieved. Doubling the number of processors from 2 to 4 results in 3-fold increase in speed from 1.24 to 3.70. The AC example shows a clear super scalability in the interval from 4 to 6 processors. This is caused by the fact, that only a part of the data can be in memory when executed on less than 6 processors. From 8 to 12 processors the scalability is linear. Similar patterns for speedup is seen for AC3, but here the super scalability continues until 12 processors. From 12 to 16 processors the speedup is linear. A further increase in the number of processors results in less than linear scalability.

The less than linear scalability achieved when the number of processors exceed a certain number is caused by the following: As the number of processors increase, the data is split in an increasing number of subsets. This leads to an increase in the number of global equations. In implementation #3, the global equations are solved on all slave processors, so the total number of equations solved increase with increasing number of processors. In the AC3 example the number of global equations increase from 1.2 to 1.7 mio., and the total number of equations solved increase from 36.1. to 60.7 mio. going from 6 to 20 processors. As long as the increase in number of global equations is less then the reduction caused by distribution on more processors, the number of equations solved on each slave processor decrease. However, the proportion of global equations solved on each slave increases.

The number of equations solved on each slave processor, and the proportion that are global equations for the AC3 example is in Fig.4. The total number of equations solved can be obtained by multiplying the number solved on each slave with number of processors minus one.

Scalability of a parallel program can also be expressed as execution time with increasing problem size when executed on a fixed number of processors. Execution times for the four test examples when executed on 6 and 12 processors are shown in Fig 5. The observed times are shown as solid lines, while linear scalability based on times for the RD example (1.2*mio. equations) are showed as dotted

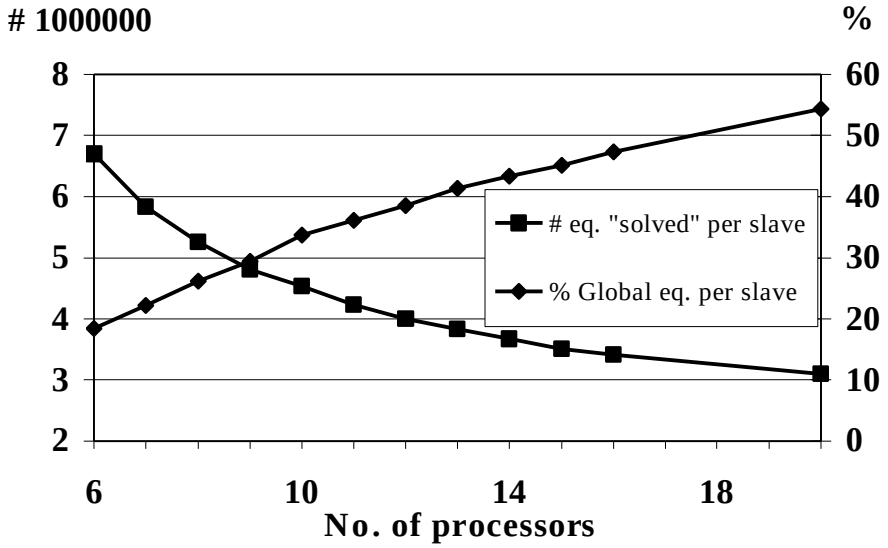

Figure 4. Number of equations solved per slave processor and the proportion (in percent) of global equation depending on number of processors for the AC3 example (30.1 mio. equations).

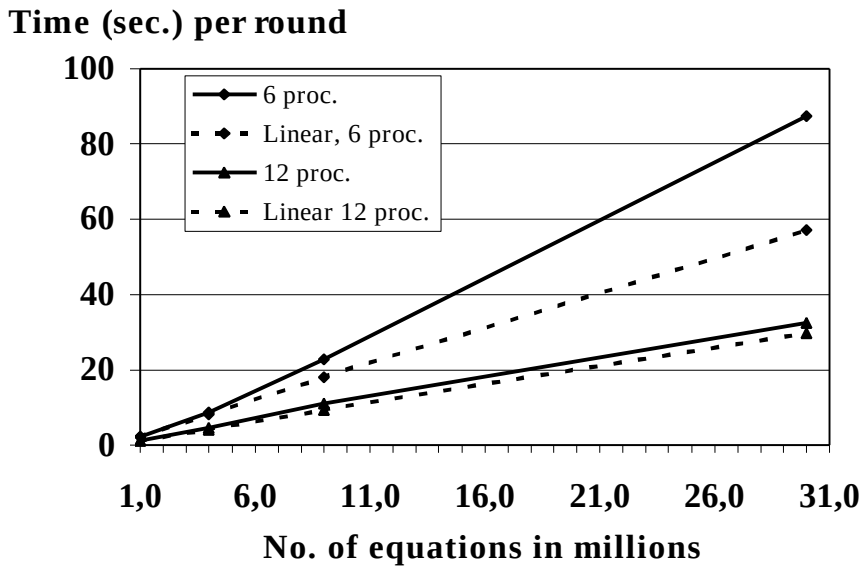

Figure 5. Size scalability on 6 and 12 processors. Solid line are observed execution time. Dotted lines are linear scalability based on execution time for the RD example (1.2 mio. eq.).

lines. On 6 processors execution time increase more than corresponding to linear scalability. This is also caused by the fact, that for the larger systems, only a small proportion of the data can be in the combined memory of the 6 processors. The 12 processor test show close to linear scalability over the size interval tested.

6. Further improvement

Further improvements of implementation #3 could be a change in the data distribution over slave processors so that all records from a herd always are on the same processor. In repeatability models as used for testing, this will eliminate all global equations for the permanent environmental effect. At the same time it will reduce the number of global an imal equations, because only relationship across herds will generate global equations.

An other possible improvement of implementation #3 could be to restrict the number of global equations solved on the individual slave processors to the subset of the global equations that are linked to the local data. This would limit the increase in the total number of equations solved with increasing number of processors.

Implementation #3 will be included in the DMU package (Jensen and Madsen, 1994) shortly. The DUM package can be downloaded by anonymous ftp from: genetics.argsci.dk in the directory pub/dmu.

7. Conclusions

Implementation #3 show good scalability both over number of processors and over dimension of the equation system to solve in the interval tested.

A parallel solver with good scalability opens for solving larger problems than can be done on a single processors or problems can be solved in shorter time.

The evolution in the GEBUS project clearly demonstrates, that it is crucial for obtaining parallel speedup that the interprocessor communication is minimised.

References

Bonaiti, B. and D. Boichard, 1995. Accounting for Foreign Information in Genetic

Evaluation.. INTERBULL Bulletin no, **11**, 4pp.

Henderson, C.R., 1975. Use of All Relatives in Intraherd Prediction of Breeding Values and Producing Abilities. J. Dairy Sci. **58,** 1910-1916

Jensen, J. and P. Madsen, 1994. DMU: A package for the analysis of multivariate mixed models. Proc. of 5th World Congress on Genetics Applied to Livestock Production., **22**, 45-46.

Larsen, M. and P. Madsen, 1999. The CEBUS project: History and overview. Computational Cattle Breeding '99 (this workshop). 9pp.

Madsen, P., J. Jensen and U. S. Nielsen, 1997. Effect of Heterosis and Imported Germ Plasm on ProductionTraits Estimated in the Danish Multi-breed Animal Model. INTERBULL Bulletin no. **16**, 85-88.

Pedersen, J., G.A. Pedersen, U.S. Nielsen, J. Jensen, P. Mansen and J.R. Thomassen. Animal Model for ydelse. (in preparation)

Schaeffer, L. R. and B. W. Kennedy, 1986. Computing Solutions to Mixed Model Equations. Proc. of 3rd World Congress on Genetics Applied to Livestock Production., **12**, 382-393.