The CEBUS project: History and overview

Martin Larsen[1] and Per Madsen[2]
[1]UNI C, Danish Computing Centre for Research and Education, DTU, Bld. 304, DK-2800 Lyngby, Denmark.
[2]Danish Institute of Agricultural Sciences (DIAS), Department of Animal Breeding and Genetics, P.O.Box 50, DK-8830, Tjele, Denmark.

Abstract

In 1995 a project was initiated by DIAS and UNI•C aiming at the development of a parallel solver for mixed model equations (MME). The project is based on the assumption, that data parallelism (DP) can be an affordable and realistic way of satisfying the steadily increasing demands of computing resources, which is needed for the solution of the still larger and more complex MMEs used in animal breeding programs. Since mid 1997, the project has received support from EU under the name of CEBUS[1] (Continuous Estimation of Breeding values Using Supercomputers).

The iterative linear equation solver (DMU5) from the DMU package developed at DIAS is the starting point for the parallelisation attempts. This solver can handle large systems by "iteration on data" using a combination of Gauss-Seidel and Jacobi algorithms. The parallel programs are developed for a distributed memory architecture using a master-slave concept. The IBM RISC 6000/SP at UNI•C (with 105 processors among which 64 + 12 are connected by a high performance switch) is used. Several parallel implementations have been tested with gradually increasing success. The first implementation used standard recursive doubling of the corrected right hand side (crhs) of the MME. This involved sending all vector components of crhs between all processors, and led to huge amounts of communication as well as memory problems. In large mixed models, the performance was worse than in the serial application.

The second implementation used summation of a packed crhs (containing nonzero components only) on the master processor. It led to substantial reduction in amount of communication and memory consumption. This implementation (available mid 1998) demonstrated good parallel speedup in medium sized MMEs, and some parallel speedup in large models provided all data could be in distributed memory (using sufficiently many processors).

These experiences has led to a third and even more successful implementation, capable of producing parallel speedup even in very large mixed models. This implementation is described in detail (together with some principles common to all the implementations) in the paper "Attacking the problem of scalability in parallel Gauss-Seidel and Jacobi solvers for mixed model equations" by Madsen and Larsen (1999) (This workshop).

1. Introduction

Estimation of Breeding Values (EBVs) by means of BLUP, is done by solving a linear system known as Mixed Model Equations (MME). Increasing size and complexity of the models in use, puts still higher demands on the computing resources. With the number of animals in Danish dairy cattle breeding, an animal model combined with a repeatability model will need approx. 10 million equations per trait. In multi-trait calculations, the number of equations will increase in proportion with the number of traits. Still larger systems need to be considered in case of test day models. Since both accurate and frequently updated EBVs are needed to support fast genetic progress, the execution time for calculation of a complete update of the EBVs need to be fairly short (in the range of few days).

An affordable and realistic way of addressing this computational challenge could be to use parallel computers based on standard type workstations and the distributed memory model. Computations are thus

[1] Esprit Project no. 24721

distributed on a number of processors with local memory, interconnected by a communication network. Special software for calculation of EBVs is needed on such a machine, in order to manage distribution of data and communication between the processors. In theory (assuming ideal load balancing), use of n processors can reduce the execution time by a factor of n.

In 1995 a project was initiated by DIAS and UNI•C aiming at the development of a parallel solver for mixed model equations (MME) using data parallelism (DP).The iterative linear equation solver (DMU5) from the DMU package developed at DIAS by Jensen and Madsen (1994) is the starting point for the parallelisation attempts. This solver can handle large systems by "iteration on data" using a combination of Gauss-Seidel and Jacobi algorithms. Since mid 1997, the project has received support from EU under the name of CEBUS (Continuous Estimation of Breeding values Using Supercomputers).

The project has developed and tested three parallel implementations with gradually increasing success. In this paper, we will give an outline of the parallelisation project, with special emphasis on the two first implementations of the parallel solver. Based on the experiences with the first two implementations, a third implementation has recently been developed. The third implementation is described in detail by Madsen and Larsen (1999) in "Attacking the problem of scalability in parallel Gauss-Seidel and Jacobi solvers for mixed model equations" (This workshop).

2. Model

For reference purposes, we write a general multivariate linear mixed model:

$$y = \mathbf{X_1} b_1 + \mathbf{X_2} b_2 + \sum_{i=1}^{r} \mathbf{Z}_i u_i + \mathbf{Z_a} a + e$$

where y is a vector of observations on t traits, b_1 and b_2 are vectors of fixed effects, u_i , i=1, 2,…,r are vectors of random effects, 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 on (co)variances are:

$$\begin{aligned}
V[u_i] &= \mathbf{G}_i = \mathbf{G}_{0i} \otimes \mathbf{I}, \; i = 1, 2,\ldots,r, \\
V[a] &= \mathbf{G}_a = \mathbf{G}_{0a} \otimes \mathbf{A}, \\
V[e] &= \mathbf{R} = \mathbf{R}_0 \otimes \mathbf{I}, \\
Cov[u_i, u_j] &= 0 \text{ , if } i \neq j, \\
Cov[u_i, a] &= 0 , \; \forall \; i, \\
Cov[u_i, e'] &= 0 , \; \forall \; i \text{ and} \\
Cov[a, e'] &= 0 , \; \forall \; i
\end{aligned}$$

where $\mathbf{G}_{0i,}$, i=1, 2,…,r are (co)variance matrices for the traits influenced by the i'th of the r random effects 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 $\mathbf{A}$ is the additive relationship matrix.

Let: $\mathbf{G} = \sum_{i=1}^{r} \mathbf{G}_i$

$\mathbf{Z} = [\mathbf{Z_1} \; ; \mathbf{Z_2} \; ; \ldots \; ; \mathbf{Z_r}]$,

$u' = [u_1 \; ; u_2 \; ; \ldots \; ; u_r]$

Use of BLUP and the assumptions on covariances leads to the MME:

$$\begin{pmatrix}
\mathbf{X}_1'\mathbf{R}^{-1}\mathbf{X}_1 & \mathbf{X}_1'\mathbf{R}^{-1}\mathbf{X}_2 & \mathbf{X}_1'\mathbf{R}^{-1}\mathbf{Z} & \mathbf{X}_1'\mathbf{R}^{-1}\mathbf{Z_a} \\
\mathbf{X}_2'\mathbf{R}^{-1}\mathbf{X}_1 & \mathbf{X}_2'\mathbf{R}^{-1}\mathbf{X}_2 & \mathbf{X}_2'\mathbf{R}^{-1}\mathbf{Z} & \mathbf{X}_2'\mathbf{R}^{-1}\mathbf{Z_a} \\
\mathbf{Z}'\mathbf{R}^{-1}\mathbf{X}_1 & \mathbf{Z}'\mathbf{R}^{-1}\mathbf{X}_2 & \mathbf{Z}'\mathbf{R}^{-1}\mathbf{Z} + \mathbf{G}^{-1} & \mathbf{Z}'\mathbf{R}^{-1}\mathbf{Z_a} \\
\mathbf{Z}_a'\mathbf{R}^{-1}\mathbf{X}_1 & \mathbf{Z}_a'\mathbf{R}^{-1}\mathbf{X}_2 & \mathbf{Z}_a'\mathbf{R}^{-1}\mathbf{Z} & \mathbf{Z}_a'\mathbf{R}^{-1}\mathbf{Z_a} + \mathbf{G_a}^{-1}
\end{pmatrix}
\begin{pmatrix} b_1 \\ b_2 \\ u \\ a \end{pmatrix}
=
\begin{pmatrix}
\mathbf{X}_1'\mathbf{R}^{-1}y \\ \mathbf{X}_2'\mathbf{R}^{-1}y \\ \mathbf{Z}'\mathbf{R}^{-1}y \\ \mathbf{Z}_a'\mathbf{R}^{-1}y
\end{pmatrix} \qquad (1)$$

3. Iterative solving strategy

The MME in (1) can be solved by Gauss-Seidel (GS), Jacobi or a combination of the two methods. Rewrite (1) as: $\mathbf{C}x = b$. Decompose $\mathbf{C}$ as $\mathbf{C} = \mathbf{L} + \mathbf{D} + \mathbf{U}$, where $\mathbf{L}$ is lower and $\mathbf{U}$ is upper triangular and $\mathbf{D}$ is diagonal. The Gauss-Seidel iteration is:

$$\mathbf{D}x^{k+1} = (-\mathbf{L}x^{k+1} - \mathbf{U}x^{k} + b) = crhs$$
$$\Rightarrow x^{k+1} = \mathbf{D}^{-1}\, crhs \qquad (2)$$

where x^{k+1} denotes the $(k + 1)^{th}$ iterate to the solution vector, and *crhs* is corrected right hand side vector.

Then the first order Jacobi method is:

$$\mathbf{D}x^{k+1} = -(\mathbf{L} + \mathbf{U})x^{k} + b = crhs$$
$$\Rightarrow x^{k+1} = \mathbf{D}^{-1}\, crhs \qquad (3)$$

In Jacobi iteration, only solutions from the previous round of iteration are used, while the GS method always use the most recent updates of the solutions. Jacobi iteration is known to have poor convergence rate on equation systems like the MME. Extending the method to second-order Jacobi improves rate of convergence (Misztal & Gianola, 1987).

The extension of (3) to second-order Jacobi iteration is:

$$x^{k+1} = \mathbf{D}^{-1}\, crhs + \alpha(x^{k} - x^{k-1}) \qquad (4)$$

where α is a relaxation factor.

Both GS and Jacobi iteration can be used in a blocked form. $\mathbf{D}$ then contains squared blocks, and elements of x and *crhs* are grouped in subspaces of dimensions corresponding to the blocks in $\mathbf{D}$.

4. Solving strategy in DMU5

We shortly sketch the solving strategy as it is implemented in the DMU5 module of the DMU package by Jensen and Madsen (1994). This basic strategy is common to the serial and all tree parallel implementations of the solver.

The explicit construction of the MME is avoided by "iteration on data" as proposed by Schaeffer and Kennedy (1986). The effect in b_1 is chosen as the one with the largest number of levels (typically herd-year-season or management group effect). The blocks used in the iterative solver are defined by level codes for b_1, u and a. For example, all equations for additive genetic effects for one animal are treated as a block and will be solved simultaneously.

A prepare program common to the serial and all parallel implementations of the solver, transforms level codes in the original datasets into consecutive numbers, and produce recoded data and pedigree files. The number of processors to be used for solving the model are set dynamically when the solver is invoked. The solver first executes an initialisation step during which the largest possible number of records are placed in memory, the remaining part on disk. Diagonal blocks are inverted and stored in memory if possible, otherwise on disk. Equations corresponding to b_2 are treated as a single block. This block is relatively dense. Its LU-factorisation is computed and stored.

In each round of iteration, data are processed in level code sequence of the effect in b_1. When all data belonging to one level code of b_1 has been processed, there are no additional contributions to the *crhs* elements for that group of equations, and an updated iterate can be obtained by expression (2). Contributions to *crhs* for the remaining part of the MME can then be calculated using the updated iterates for b_1 and the values for b_2, u and a from the previous round of iteration. When all data has been processed the complete b_1 vector has been updated, and all contributions from data to the elements in *crhs* corresponding to equations for b_2, u and a have been calculated. Correction due to the relationship matrix for the part of *crhs* corresponding to animal equations (a) are then calculated. An iterate for b_2 is found by

a backsolve using the LU-factorisation. Updated iterates for u and a are obtained by (4). Iterations continue until a criterion of convergence in the 2-norm of the solution is met. An outline of the serial solver is shown in the left part of Figure 1.

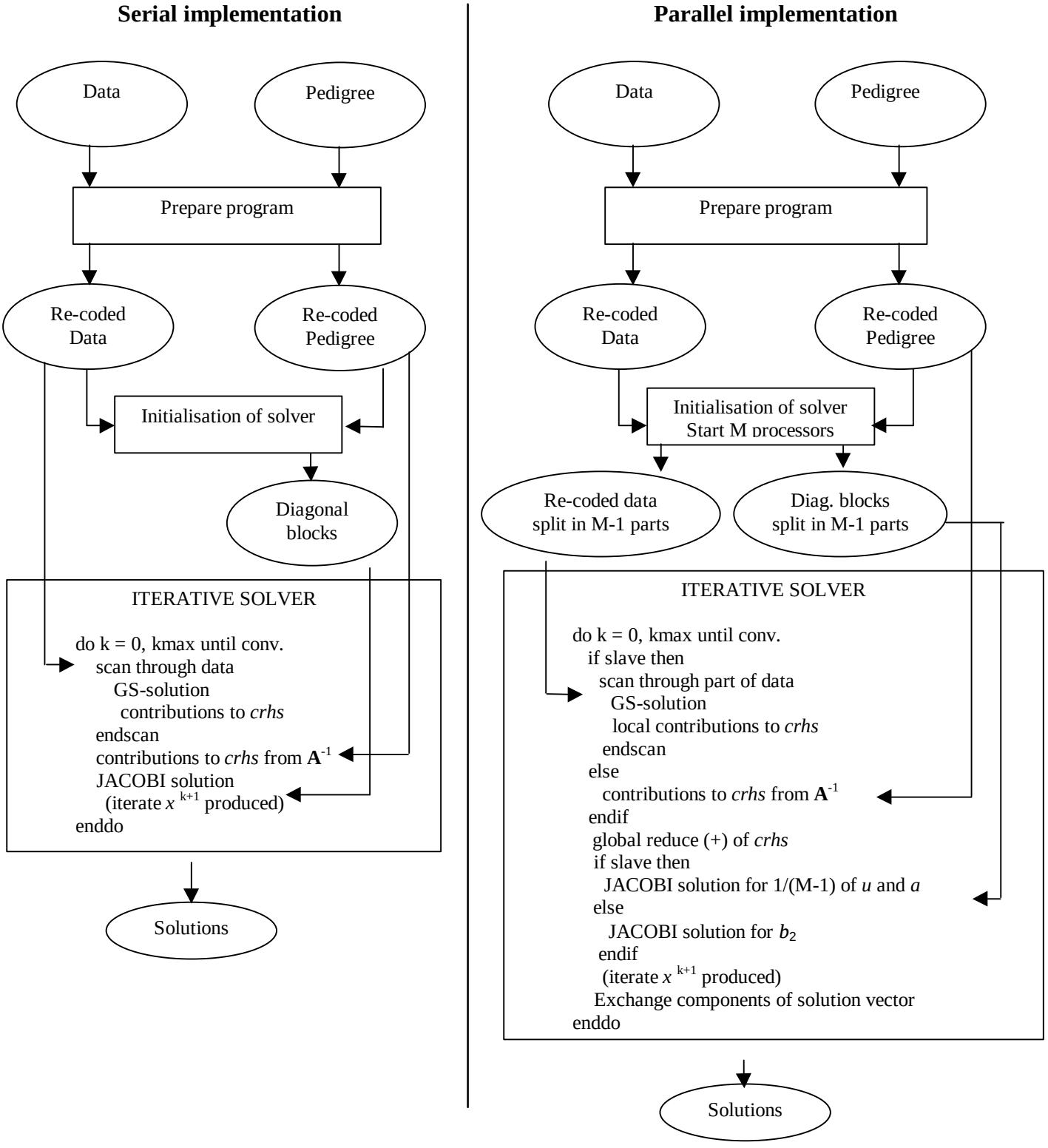

Figure 1. Program layout for a serial (left) and parallel implementation #2 (right) of iteration on data

5. Distributed memory architecture and DP.

Common to the parallelisation attempts is use of Data Parallelism (DP) on a distributed memory architecture. DP means that data records are grouped in disjoint sets. The groups are mapped to specific processors. In parallel DMU5 data distribution takes place in the solver initialisation phase. For good performance (minimisation of interprocessor communication) it is important that records belonging to the same level code of b_1 never map to different processors.

In a distributed memory architecture, each processor has its own local memory. The cpu in one processor cannot read directly in the memory of another processor. Instead the processors exchange data through a communication network.

A master-slave concept is used. The master processor can perform tasks different from the slave processors. Slave processors execute identical codes. In all three implementations of parallel DMU5, the master processor control the data distribution step, read recoded data and transmit data records to the slaves. The tasks of the master processor during iterations differ between the implementations. The LU-decomposition and backsolve in the block corresponding to b_2 is done exclusively on the master processor.

In all three parallel implementations there is a need for some kind of "global reduction" of the *crhs* vector i.e local components of *crhs* need to be summed across processors to get a global *crhs* for use in formula (4).The global reduction is implemented very differently in the implementations. Since the global reduction implies interprocessor communication, execution time in the iteration round is minimised by minimising the number of components of *crhs* which need to take part in the global reduction.

6. Development platform

The parallel platform used for development and test is the IBM RS/6000 SP parallel computer with distributed memory architecture at UNI-C. This machine has (in the moment of writing) 105 processors. The architecture in each processor is that of a POWER2 RS/6000 workstation. There are 64 + 12 + 4 processors for parallel batch use, 20 processors for serial batch, 4 interactive nodes and a control workstation. The parallel batch nodes are interconnected by a High Performance Switch.

One parallel batch job is allowed to use not more than 32 processors. 12 parallel batch nodes are for test jobs with short execution time. The remaining 4 parallel batch processors are reserved for special use. 32 + 12 parallel batch processors have 0.5 GB memory, the remaining 32 have 256 MB memory. Each processor has 3.5 GB scratch disk. All processors have access to a large home filesystem and a parallel I/O filesystem as well.

Message passing library used is PVM. A MPI version of implementation #3 has recently been prepared.

Parallel implementation #1

In this implementation, the parallelisation of the DMU 5 solver is made as simple as possible. Data records are distributed on both master and slaves in approximately equal proportions. When all data are processed in any iteration round, each processor which solves for the iterate of b_1 gets an update of a local copy of *crhs* in local memory. A global reduce is done on all components of *crhs* corresponding to the full dimension of the MME. All processors participate symmetrically in the global reduce. When the global reduce is finished, the *crhs* on each processor is the global update of *crhs*. Each processor will use some of the components in *crhs* for the determination of the iterates to

b_2, u and a. Pedigree processing is done exclusively on the master processor.

The memory consumption in this approach is substantial. Each node need three vectors in the same dimension as the MME: x^k, $x^{k-1}$and *crhs*. The vectors are in double precision. This memory consumption is not dependent on the number of processors attached to the parallel job. Therefore a severe limit on the scalability of this implementation exists. When the three vectors take up most memory in each processor, use of virtual memory is enforced, and system paging results. In case of a $9.5*10^6$ equation model, the memory need for vectors is $(9.5*10^6*8/1024^2)*3$ MB = 217.4 MB which together with other memory use by program and system will more than exhaust a processor with 256 MB memory.

Another problem in this implementation is the amount of communication generated by the global reduce operation. A method called "recursive doubling" can be used to minimise the number of point to point communications. In general n processors need

$$p = \mathbf{ceil}(\mathbf{log}_2(n))$$

levels of communication where ceil produce the smallest whole number larger than its argument. Use of 16 processors on a system with dimension $9.5*10^6$, send an amount of communication on the network in 4 levels producing $16*(9.5*10^6*8/1024^2)*4$ MB = 4.64 GB in each round of iteration. Since many of the contributions in this reduction are 0, and most components of *crhs* are not needed on the processor for its use of formula (4), this communication represents a large waist of resources. Communication buffers are allocated on both sending and receiving processors. Therefore, true communication load due to recursive doubling is twice the value mentioned above. In addition to this, iterates of u and a components need to be broadcasted from the slaves on which they are computed, to all other processors. The iterate of b_2 is broadcasted from the master to all slaves. This is necessary in order to have the x^k properly updated on all processors before the start of the next round of iteration. Taking this additional communication buffer allocation into account, the total amount of communication becomes 9.5 GB at 16 processors.

A speedup curve for implementation #1 was never produced. On large models, it seemed slower than its serial counterpart. Its main virtue was to demonstrate that parallel execution could be obtained, and to probe the possibilities and limitations of the DP approach on the machine in question.

Parallel implementation #2

Monitoring the processors during execution in implementation #1 indicated, that the master acted as a bottleneck. Therefore the data records (except pedigree records) was removed from the master, and it did not any longer solve for the iterate of b_1. Besides this, the experiences with implementation #1 demonstrated a need for radical reduction in memory usage and amount of communication. These closely related topics are addressed in implementation#2 (Madsen and Larsen 1998).

The basic idea is to compress *crhs* so that only components related to data on the slave are allocated. The solver initialisation step is modified. It predicts the components of *crhs* needed on any slave when data are distributed. This is done by the same addressing algorithm which is used later on in the iterations when data are absorbed into *crhs*. We now use two types of addressing spaces for *crhs*: The one with same dimension as MME (global), and local spaces on each slave with decreasing dimension when number of slaves increase. The initialisation generate mapping vectors between the local spaces and the global space. Each slave has a mapping vector of full length of type integer*4. This vector (stored in memory) gives the mapping from the global to the local space on that slave. The master has mapping vectors for each slave stored on disk. They are

used by the master for mapping from a slaves local to the global space.

The global reduce is implemented as follows: Each slave send its compressed *crhs* to the master. The master reads the corresponding mapping vector from disk, and use it for unpacking the received local *crhs* into the global space. The master sums the *crhs* contributions. Finally, disjoint parts of the global *crhs* are sent to those slaves which need them for the Jacobi iteration by (4). An outline of this solver is shown in the right part of Figure 1.

The amount of communication is reduced substantially by this method. Continuing with the example mentioned above, the communication buffer allocation per iteration becomes 1.3 GB at 16 processors, which means a reduction of 86%.

Even though the mapping vectors have full length, they only fill 4 byte per element compared to the 8 bytes of *crhs*. In addition to this, x^{k-1} is compressed by a simple linear transformation of the address space, so that only components of x^{k-1} needed on the slave for use by (4) are allocated on that slave. Thus, some reduction in memory consumption is obtained, although memory consumption is not fully scalable by model size since the slaves still need a full length x^k, and the master need the global *crhs*, b_2, and *a*.

This implementation has been tested on the following three data sets:

- Protein yield for all Red Danish dairy cows (RD), $1.2*10^6$ eq.
- Protein, fat and milk yield for all Red Danish dairy cows (RD3), $4.3*10^6$ eq.
- Protein yield for all dairy cows in Denmark (AC), $9.5*10^6$ eq.

Relative speedup curves for these models as a function of number of processors on the IBM SP at UNI•C is shown in Figure 2.

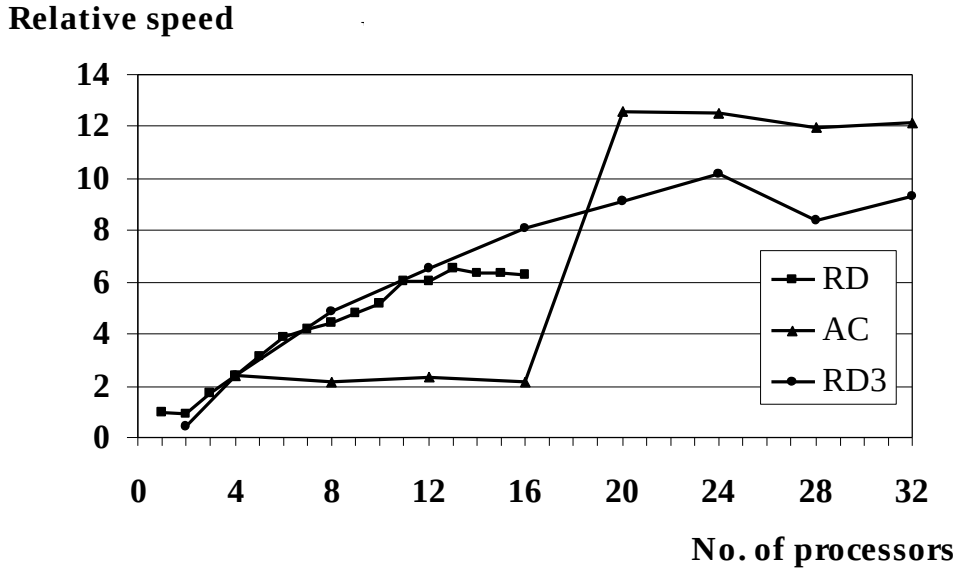

Figure 2. Relative speedup in implementation #2 for single trait Red Danish cattle (RD, $1.2*10^6$ eq.), three trait Red Danish cattle (RD3, $4.3*10^6$ eq.) and all Danish dairy cattle single trait (AC, $9.5*10^6$ eq.)

Speed is calculated as reciprocal execution time per round of iteration. Relative speed is normalised to the same value on 4 processors, because this is the smallest number of processors where AC can run due to memory requirements on 256 MB processors. Because the master only solve very few equations, a small drop in performance is observed from one to two processors in RD. In RD and RD3, parallel speedup close to linear is obtained from 2 to 8 processors. At RD it flattens around 12 processors because the master becomes a bottleneck due to pedigree processing and increased communication. For RD3 some speedup continues until 24 processors, because the larger model size means that data distribution on more slaves is needed before the pedigree processing on master becomes a bottleneck.

AC behaves very different. This model is so large, that part of data need to be stored on disk until between 16 and 20 processors. When all data can be in memory, a 6-fold increase in speed is observed. Below 16 and again above 20 processors, the communication on the master becomes a bottleneck, and no speedup is observed.

7. Conclusions

Implementation #2 demonstrate that parallel speedup can be achieved in moderately sized models. It does not show full scalability on number of processors and model size. For the very large MMEs this implementation is insufficient.

Compared to implementation #1 the result is however encouraging. The amount of communication is still relative high. If a new reduction in amount of communication as radical as the one already seen can be achieved, a fast parallel solver should be obtainable even for large models. And if all three types of working vectors can be compressed in proportion with the number of processors, the solver should be scalable as well. Precisely these objectives are addressed in implementation #3, described in detail in the paper "Attacking the problem of scalability in parallel Gauss-Seidel and Jacobi solvers for mixed model equations" by Madsen and Larsen (1999) (This workshop).

References

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.

Madsen, P. and M. Larsen, 1998. A Parallel Solver for Animal Genetics. In: Kagström, B., Dongarra, J., Elmroth, E. & Wasniewski, J. (Eds.). Applied Parallel Computing. Large Scale Scientific and Industrial Problems. Proc. 4th International Workshop PARA'98, Umeå, Sweden. LNCS, **1541**. 304-308.

Misztal, I. and D. Gianola, 1987. Indirect Solution of Mixed Model Equations. J. Dairy Sci., **70**, 716-723.

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.