Computer Science • Vol. 10 • 2009

Maciej Gierdziewicz[*]

PARALLEL ALGORITHM FOR SORTING ANIMAL PEDIGREES

In many analyses of animal genotype with the methods of quantitative genetics there is a need to account for relationships among individuals. Incorrectly calculated relationship coefficients may lead to biased estimates. The number of software packages exist which deal with that problem; however, in many of them it is assumed that pedigrees of the individuals are sorted chronologically, but in real data sets – containing information on traits and pedigrees – birth dates are often missing. In extreme cases, when (almost) no birth dates are present, the ordering must be made by comparing – at least once – each pair of individuals separately, since it is not sufficient to compare adjacent elements in order to check whether the data set is sorted. Two versions of parallel computer programs were compared, with constant or variable distance between elements of compared pairs. The results indicate that the second algorithm is more efficient.

Keywords: *animal breeding, pedigree, chronological order, parallel computing*

ALGORYTM W WERSJI WSPÓŁBIEŻNEJ DO SORTOWANIA RODOWODÓW ZWIERZĄT

Badając genotypy zwierząt metodami genetyki ilościowej, trzeba uwzględniać spokrewnienia między zwierzętami. Niepoprawnie obliczone współczynniki spokrewnienia mogą prowadzić do oszacowań obciążonych błędem. W wielu gotowych pakietach ten problem jest uwzględniony; jednak często wymagane jest chronologiczne uporządkowanie rodowodów, ale w danych doświadczalnych często brakuje daty urodzenia zwierzęcia. W przypadkach skrajnych dla ustalenia porządku należy porównać – przynajmniej raz – każdą parę osobników w celu ich posortowania. Porównano dwie wersje algorytmu – ze stałym albo zmiennym odstępem między elementami pary w obrębie iteracji. Wyniki wskazują, że druga wersja algorytmu działa szybciej.

Słowa kluczowe: *hodowla zwierząt, rodowód, porządek chronologiczny, przetwarzanie współbieżne*

[*] Department of Genetics and Animal Breeding, Faculty of Animal Sciences, `rzgierdz@cyf-kr.edu.pl`

1. Introduction

1.1. Motivation

In quantitative genetics the common task is to estimate breeding (genetic) values of individuals, and therefore there is a need to account for relationships among individuals (Quaas 1989 [7]; Kennedy *et al.* 1992 [4]). Incorrectly calculated relationship coefficients may lead to bias in estimation of genetic values. The number of software packages exist which deal with that problem, allowing to add the inverse of the animal relationship matrix to Mixed Model Equations (MME) (Henderson 1976 [3]; Boldman *et al.* 1993 [1]; Gilmour *et al.* 2000 [2]); however, in many of them it is assumed that pedigrees of the individuals are sorted chronologically. The chronological order of pedigrees is easy to achieve when they include birth dates but, especially for older ancestors, when the data set contains information on traits and pedigrees, it often happens that birth dates are missing. In extreme cases, when (almost) no birth dates are present, the inference about the order of the animals must be made by comparing – at least once – each pair of individuals separately; it is not sufficient to compare adjacent elements in order to conclude that the data set is sorted. The corresponding computer program may look quite simple. Step 1: for each pair, swap the elements if they are incorrectly ordered; Step 2: repeat Step 1 until no swapping is done. The method is easy to program but it uses a lot of computer time.

1.2. Related work

The solution of the problem of sorting pedigree files without birth dates presented recently is called “pyramid algorithm”. It has been proposed this year (Zhiwu Zhang *et al.* 2009 [9]). Unfortunately, it is rather difficult to parallelize. The program assigns – step by step – generation numbers to the animals in such a way that progeny should be given lower number than the ancestor. The problem is that if an animal is an ancestor of two or more different animals, in parallel version of the algorithm it may be assigned two different generation numbers at the same time. The solution would be to include intra-process communication which slows down sorting speed considerably. On the other hand, it would be profitable to parallelize pedigree sorting; parallel computing has already been used to calculate breeding values of animals (Lidauer *et al.* 1998 [5]; Lidauer, Stranden 1999 [6]; Stranden, Lidauer 2001 [8]). In that case, in order to parallelize the programs, the iterations had to be made independent.

So, in order to obtain the solution as quickly as possible, an efficient way of performing independent calculations has to be found, which means arranging pairs of animals being compared in such a way that different iterations do not involve the same animal.

1.3. Scientific objectives

The aim of the work was to create a parallel version of the simple algorithm to compare and sort animal pedigrees without birth dates, basing on relationships between ancestors and progeny.

2. Problem solution

Two attempts to solve the title problem are presented in this paper. In both solutions each pair of animal pedigrees ($n(n+1)/2$ pairs if there are n animals) had to be compared; however, the pairs had to be arranged in such a way that allowed for parallelization of the algorithm. For both algorithms, the arrangement of the pairs within each iteration of the main programming loop is described below.

In the first algorithm the pairs were compared – and, if necessary, swapped – according to the distance of the elements (pedigrees), beginning with the most distant pairs: In the first phase of the algorithm the following comparisons were performed: first, Pedigree 1 with Pedigree n (distance $d = n-1$); then 1 with $n-1$ and 2 with n ($d = n-2$); then 1 with $n-2$, 2 with $n-1$ and 3 with n; the distance within a pair decreased and was: $n-1$, $n-2$, etc. and for each distance the iterations were executed in parallel.

Let $i = d+1$; the first phase ended when the distance within a pair was small enough to create at least three subsets of elements: $1, \ldots, d$, $d+1, \ldots, 2d$ and $2d+1, \ldots, n$. If we number the subsets consecutively, then two parallel loops could be executed: one for pairs with the first element from an "odd" subset and the second one from next "even" subset and, afterwards, the second one, for pairs with the first element from an "even" subset and the second one from next "odd" subset. The last comparisons were made between least distant (adjacent) elements: first, simultaneously, for pairs (1,2), (3,4), (5,6), etc., then also independently, for pairs (2,3), (4,5), (6,7) etc. In the second algorithm the pairs were arranged in the following way:

Two loops were designed. The first one began with the pair (element 1, element n), then $(2, n-1)$, $(3, n-2)$ etc., and ended, depending on the number of animals, with the pair $(n/2, n/2+1)$ for even n, and with $((n-1)/2, (n+1)/2+1)$ for odd n. The second loop followed the same pattern, but the starting pair was $(1, n-1)$ instead of $(1, n)$.

The calculations in the above two loops were performed in a kind of an "external" loop assuming that the starting value (the position of the first element of the first pair) varied from 1 to n; all other indices had to be shifted right (increased), with the use of modulo function i.e. element n was followed by element 1.

In both algorithms, the calculations within each iteration in the main program loop (which means comparing all possible pairs of pedigrees) were repeated until no swapping was done during the whole iteration.

3. Algorithms and methods

3.1. Algorithm with constant distance between pair elements within iteration

The outline of the first algorithm (for equally distant pairs) is presented in listing 1, using Fortran-like convention. First, the main program “loop” was designed, controlled by the variable “iswp”. The positive value of iswp indicated that at least one swapping of the elements has been done and more iteration is needed.

Then the inner “loop” was designed, controlled by the variable “i” which denotes the distance between elements of the pair and varies from $n-1$ to 1. Within this loop the following tasks are performed: if “i” is small enough, then the pairs which are to be compared are arranged in at least two subsets. The positions of first elements of these pairs are stored in an auxiliary array “ltab”. Then the comparisons can be performed in parallel. Next, if the last subset of indices is smaller than the others, k1 and k2 are assigned values of positions of the first and second elements of the current pair.

In order to compare a pair of elements the program verified if the animal in the first pedigree was a progeny of the second animal. If this was true, the program interchanged these two animals (pedigrees).

Listing 1: Outline of the first algorithm of sorting animal pedigrees (for equally distant animal pairs per iteration)

```
111   continue
       iswp = 0
       i = n - 1
222    continue
        j = 1
333     continue
         if ( j .le. n-2*i+1 ) then
            itt = 0
            do k = j, n-2*i+1, 2*i
              do l = k, k + i - 1
                itt = itt + 1
                ltab ( itt ) = l
              enddo
            enddo
            ntt=itt
!$OMP       PARALLEL DO [...]
            do itt = 1, ntt
              iswp = 0
              l = ltab ( itt )
               if ( "l is a progeny of l+i" ) then
                 swap ( l, l + i )
                 iswp = iswp + 1
               endif
            enddo
!$OMP       END PARALLEL DO
         endif
```

```
          j = j + i
         if ( j .le. i+1 ) goto 333
         if ( mod ( n, i ) .gt. 0 ) then
           k1 = n - mod ( n, i ) - i
           k2 = n - i
!$OMP      PARALLEL DO [...]
           do k = k1, k2
             if ( "k is a progeny of k+i" ) then
               swap ( k, k + i )
               iswp = iswp + 1
             endif
           enddo
!$OMP      END PARALLEL DO
         endif
         i = i - 1
        if ( i .ge. 1 ) goto 222
       if ( iswp > 0 ) goto 111
```

3.2. Algorithm with constant number of pairs per iteration

In listing 2 the second solution of the title problem is presented.

Listing 2: Outline of the second algorithm of sorting animal pedigrees (for equal number of animal pairs per iteration)

```
111  continue
      iswp = 0
222    continue
        ishift = 0
        if ( mod ( nmax, 2 ) = 0 ) then
!$OMP    PARALLEL DO [ ]
          do i = 1, n / 2
           i1 = min ( mod(ishift+i-1,n) + 1, mod(ishift+n-i,n) + 1 )
           i2 = max ( mod(ishift+i-1,n) + 1, mod(ishift+n-i,n) + 1 )
             if ( i1 is progeny of i2 ) then
               swap ( i1, i2 )
               iswp = iswp + 1
             endif
           enddo
!$OMP    END PARALLEL DO
!$OMP    PARALLEL DO [ ]
          do i = 1, n / 2 - 1
           i1 = min ( mod(ishift+i-1,n)+1, mod(ishift+n-i-1,n) + 1 )
           i2 = max ( mod(ishift+i-1,n)+1, mod(ishift+n-i-1,n) + 1 )
           if ( i1 is progeny of i2 ) then
             swap ( i1, i2 )
             iswp = iswp + 1
           endif
          enddo
!$OMP    END PARALLEL DO
        else
!$OMP    PARALLEL DO [ ]
          do i = 1, n / 2
           i1 = min ( mod(ishift+i-1,n) + 1, mod(ishift+n-i,n) + 1 )
           i2 = max ( mod(ishift+i-1,n) + 1, mod(ishift+n-i,n) + 1 )
           if ( i1 is progeny of i2 ) then
             swap ( i1, i2 )
             iswp = iswp + 1
           endif
          enddo
!$OMP   END PARALLEL DO
!$OMP   PARALLEL DO [ ]
          do i = 1, n / 2 - 1
           i1 = min ( mod(ishift+i-2,n)+1, mod(ishift+n-i-2,n)+1 )
           i2 = max ( mod(ishift+i-2,n)+1, mod(ishift+n-i-2,n)+1 )
           if ( i1 is progeny of i2 ) then
             swap ( i1, i2 )
             iswp = iswp + 1
           endif
          enddo
!$OMP   END PARALLEL DO
        endif
        ishift = ishift + 1
        if ( ishift < n ) go to 222
        if (iswp > 0) goto 111
      enddo
```

This time the distance between pair elements in each iteration was variable, but the number of animal pairs per iteration was constant.

Like in the first method, the main program loop was repeated until there were no swapped elements in the last iteration. Within the main loop, the second loop was defined.

That second (inner) loop was controlled by the variable “ishift” which denoted the distance of shifting (modulo n i.e. modulo number of pedigrees) of all the indices of the elements to the right.

Within the second loop four parallel loops were designed; first and second were executed when n was even, the last two – when n was odd. In each loop comparison began with the most distant elements and then both indices were moved closer together, giving, as the result, pairs: $(1, n)$, $(2, n-1)$, $(3, n-2)$ etc. When the number of elements was even, no elements remained after such a loop; otherwise the middle element remained. As mentioned above, the algorithm stopped when there was no swapping done during the last iteration of the main program loop.

3.3. Data, software and hardware

The data were 63264 one-generation Polish Black-and-White cattle pedigrees. The FORTRAN program was parallelized via “OMP PARALLEL DO” directive with “DYNAMIC” scheduling: the calculations that were designed for parallelization were distributed among processes in “slices” 1000 iterations each. The Fortran ifort compiler was used in SGI Altix 3700 computer (“Baribal”) with 256 Intel Itanium 2 processors, 1.5 GHz clock and SUSE Linux Enterprise Server 10 operating system, and in SGI Altix 4700 computer (“Panda”) with the same operating system, 32 Intel

Itanium 2 processors and 1,66 GHz clock. Real (wall clock) time and CPU time were used as measures of efficiency of the algorithms.

4. Results

In table 1 the time needed to perform the calculations for the two algorithms used in the two computers is presented. Since calculations were often repeated for specific combinations of computer, algorithm and number of threads (processes), only the best results are shown.

Real time of sorting in single processor version was from ca. 303s (first method, "Panda") to ca. 411s (first method, "Baribal"). The best results of parallelizing were achieved for 8 processors (first method, "Panda") and for 16 processors (second method, "Baribal"). The amount of time used for sorting in these two cases was 222s and 213s i.e. about 57.6% and 73.3% of single processor version time, respectively. The corresponding speedup factors were ca. 1.36 and ca. 1.46 i.e. about 17% and about 9% of the number of processors. It should be noted that the results are very variable, especially when multiprocessor versions o the algorithm are concerned. This may suggest that the algorithm is sensible to interference of the programming environment eg. other users' programs.

5. Conclusions and future work

Parallelization of sorting algorithm for incomplete information on relation among elements of the data set, which takes place in the case of animal pedigrees, is rather difficult. The speedup factor in this paper did not exceed 1/5 of the number of processors used for calculations. This does not mean that better results can not be obtained. However, it is unlikely to get much better results.

Table 1
Time used for calculations for two algorithms of parallel sorting of pedigrees

Algorithm	Number of processors	Computer			
		SGI Altix 3700 ("Baribal")		SGI Altix 3700 ("Panda")	
		real time [s]	CPU time [s]	real time [s]	CPU time [s]
I – constant distance within pair per iteration	1	411	408	303	295
	2	588	1091	343	669
	4	over 900	over 2700	531	1652
	8	over 900	over 2700	222	1668
	16	303	2248	314	4508
II – constant number of pairs per iteration	1	370	369	333	322
	2	995	1986	881	1783
	4	526	2103	467	1863
	8	342	2732	292	2321
	16	213	3375	320	3751

In future more processors may be used to perform sorting faster, but it would be more instructive to test the algorithms in single-user mode and/or for larger data sets.

References

[1] Boldman K. G., Kriese L. A., Van Vleck L. D., Van Tassel L. A., Kachman S. D.: *A manual for use of MTDFREML, a set of programs to obtain estimates of variances and covariances.* Clay Center, Nebraska, USA, USDA–ARS 1993

[2] Gilmour A. R., Cullis B. R., Welham S. J., Thompson R.: *ASREML reference manual.* Harpedden, UK, IACR-Rothamsted Experimental Station 2000

[3] Henderson C. R.: *A simple method of computing the inverse of a numerator relationship matrix used for prediction of breeding values.* Biometrics, vol. 32, 1976, 69–79

[4] Kennedy B. W., Quinton M., Van Arendonk J. A. M.: *Estimation of effects of single genes on quantitative traits.* Journal of Animal Science, vol. 70, 1992, 2000–2012

[5] Lidauer M., Mäntysaari E. A., Stranden I., Kettunen A., Poso J.: *DMUIOD: A multitrait BLUP program suitable for random regression testday models.* [in:] 6th World Congress „Genetics Applied to Livestock Production", Armidale, NSW, Australia 1988

[6] Lidauer M., Stranden I.: *Fast and flexible program for genetic evaluation in dairy cattle.* [in:] „International Workshop On Computational Cattle Breeding", Tuusula, Finland 1999

[7] Quaas R. L.: *Transformed mixed model equations. A recursive algorithm to eliminate A^{-1}.* Journal of Dairy Science, vol. 72, 1989, 1937–1941

[8] Stranden I., Lidauer M.: *Parallel Computing Applied to Breeding Value Estimation in Dairy Cattle.* Journal of Dairy Science, vol. 84, 2001, 276–285

[9] Zhang Z., Li C., Todhunter R. J., Lust G., Goonewardene L., Wang Z.: *An Algorithm to Sort Complex Pedigrees Chronologically without Birthdates.* Journal of Animal and Veterinary Advances, vol. 8, 2009, 177–182