Proc. Assoc. Advmt. Anim. Breed. Genet. 20:270-273

UTILITY OF GRAPHICS PROCESSING UNITS FOR DENSE MATRIX CALCULATIONS IN COMPUTING AND INVERTING GENOMIC RELATIONSHIP MATRICES

Karin Meyer and Bruce Tier

Animal Genetics and Breeding Unit*, University of New England, Armidale, NSW 2351

SUMMARY

The era of genomic evaluation has brought the need to perform computations involving large, dense matrices. Particular tasks are the computation and inversion of the genomic relationship matrix. This paper investigates the suitability of Graphics Processing Units together with highly optimised software libraries for these computations, using blocked algorithms. It is shown that calculations are readily sped up by parallel processing, using freely available library routines, and that reductions in time by factors of 4 to 5 are achievable even for 'consumer' grade graphics cards.

INTRODUCTION

Computer gaming requires computing of large numbers of pixel values at a fast rate. This computational load has stimulated development of 'co-processors' – so-called Graphics Processing Units (GPU). Modern GPU devices have thousands of cores and capabilities well suited to general purpose computing, providing very high rates of floating point operations. However, GPU cores have limited features and memory, restricting the type of computations that can be accelerated by GPUs. Typically, this requires computations to be executable in subsets and thus to be highly parallelisable.

Cole *et al.* (2012) discuss the potential of GPUs for applications in animal breeding. For a long time, efficient mixed model computations in animal breeding have relied on the sparseness of the pertaining equations. However, the advent of genomic evaluation has resulted in the need for large-scale manipulation of dense matrices. Fortunately, highly optimised software routines are available to perform many of the tasks required, especially in the BLAS (Dongarra *et al.* 1988) and LAPACK (Anderson *et al.* 1999) libraries. Their efficiency in computing the genomic relationship matrix (GRM) and its inverse has been demonstrated by Aguilar *et al.* (2011) and Meyer *et al.* (2013).

Use of a GPU requires a special programming interfaces such as CUDA (Compute Unified Device Architecture), the NVIDIA proprietary platform (NVIDIA Corporation 2013). Matrix computations on GPUs are greatly aided by corresponding software libraries: CUBLAS, part of the CUDA toolkit, provides GPU accelerated BLAS routines, and the equivalents to LAPACK routines are available from the CULA (Humphrey *et al.* 2010) or MAGMA (e.g. Dongarra *et al.* 2012) libraries. This allows for applications using such tools to be readily ported to GPUs, though challenges arise from their limited memory which requires matrices and computations to be broken into blocks accommodated on GPU devices. This paper presents a first investigation into the scope of GPUs to accelerate dense matrix computations, such as required to calculate and invert the GRM.

MATRIX MANIPULATION BY PARTS

Calculation of the GRM, $\mathbf{G}$, involves a matrix product of form $\alpha\mathbf{ZZ}'$ with dimensions of $\mathbf{Z}$ equal to the number of individuals (n) $\times$ number of alleles (s) and α a scale factor (Van Raden 2008). As $\mathbf{G}$ is symmetric, only one triangle needs to be computed. Calculations represent a rank-k update of a symmetric matrix, a task performed by BLAS routine SYRK. Partition $\mathbf{G}$ and $\mathbf{Z}$ into blocks $\mathbf{G}_{ij}$ ($i, j = 1, r$) and $\mathbf{Z}_{ik}$ ($i = 1, r$ and $k = 1, t$), as dictated by memory available on the GPU. Blocks $\mathbf{G}_{ij} = \alpha \sum_{l=1}^{t} \mathbf{Z}_{il}\mathbf{Z}'_{jl}$ can then be computed by repeated calls to SYRK for $i = j$ and BLAS routine GEMM (which evaluates a general matrix by matrix product) for $i \neq j$.

*AGBU is a joint venture of NSW Department of Department of Primary Industries and the University of New England

Inversion. A standard method to invert a symmetric, positive definite matrix is to carry out a Cholesky decomposition, calculate the inverse of the factor and multiply the latter with its transpose, taking advantage of the triangular nature of these matrices. This can be performed by LAPACK routines POTRF and POTRI. For block-wise inversion, Gauss-Jordan elimination type algorithms have been suggested (Quintana *et al.* 2001; Ezzatti *et al.* 2011; Benner *et al.* 2011). This can be carried out 'in place', overwriting **G** with $\mathbf{G}^{-1}$. For each step, partition **G** into current (C), previous (P) and trailing (T) blocks with n_1, n_b (chosen block size) and n_2 rows, respectively.

$$\mathbf{G} = \begin{pmatrix} \mathbf{G}_{PP} & \mathbf{G}_{PC} & \mathbf{G}_{PT} \\ \mathbf{G}_{CP} & \mathbf{G}_{CC} & \mathbf{G}_{CT} \\ \mathbf{G}_{TP} & \mathbf{G}_{TC} & \mathbf{G}_{TT} \end{pmatrix}$$

```
$\mathbf{G}_{CC} := \mathrm{chol}(\mathbf{G}_{CC})$                                   POTRF
$\mathbf{G}_{CC} := \mathbf{G}_{CC}^{-1}$                                          TRTRI
$\mathbf{G}_{PC} := \mathbf{G}_{PC}\mathbf{G}_{CC}$                                TRMM
$\mathbf{G}_{PP} := \mathbf{G}_{PP} + \mathbf{G}_{PC}\mathbf{G}'_{PC}$             SYRK
$\mathbf{G}_{CT} := \mathbf{G}'_{CC}\mathbf{G}_{CT}$                               TRMM
$\mathbf{G}_{TT} := \mathbf{G}_{TT} - \mathbf{G}'_{CT}\mathbf{G}_{CT}$             SYRK
$\mathbf{G}_{PT} := \mathbf{G}_{PT} - \mathbf{G}_{PC}\mathbf{G}_{CT}$              GEMM
$\mathbf{G}_{CT} := -(\mathbf{G}_{CC}\mathbf{G}_{CT})$                             TRMM
$\mathbf{G}_{PC} := \mathbf{G}_{PC}\mathbf{G}'_{CC}$                               TRMM
$\mathbf{G}_{CC} := \mathbf{G}_{CC}\mathbf{G}'_{CC}$                               LAUUM
```

Figure 1. Algorithm for block-wise matrix inversion

At the beginning, current and trailing blocks contain the respective parts of **G** given $\mathbf{G}_{PP}$. The algorithm then starts with the Cholesky factorisation of the current, diagonal block, $\mathbf{G}_{CC} = \mathbf{R}'\mathbf{R}$, and inversion of **R**, an upper triangular matrix. This is followed by steps adjusting previous blocks for the contribution of $\mathbf{G}_{CC}^{-1}$ to their inverses, and trailing blocks by 'absorbing' rows and columns $n_1 + 1$ to $n_1 + n_b$. Finally, $\mathbf{G}_{CC}^{-1}$ is obtained as $\mathbf{R}^{-1}(\mathbf{R}^{-1})'$. Blockwise calculations are repeated, updating n_1 to $n_1 + n_b$ and n_2 to $n2 - n_b$, until $n_1 = n$ and $n_2 = 0$. Pseudo-code adapted from Benner *et al.* (2011) (correcting errors in their description), together with the appropriate BLAS or LAPACK routines for individual calculations are given in Figure 1 (with $\mathbf{A} := \mathbf{B}$ denoting replacement of **A** by **B**).

MATERIAL AND METHODS

Time required for both types of matrix operations were compared using simulated matrices. For the matrix product, allele counts were obtained by sampling values 0, 1 or 2 from a uniform distribution for $s = 512000$ and $n = 512$ to 20,480 individuals. For matrix inversion, successive submatrices of a GRM set up as in Meyer *et al.* (2013), were used considering $n = 512$ to 16,384.

Calculations were performed in single precision, using either a single CPU (CPU1), all (4) CPU cores available (CPU4) or the GPU, performing computations in blocks as required by memory limits. For matrix multiplications on the CPU, **Z** was processed in up to 5 blocks, splitting **Z** adaptively into submatrices $\mathbf{Z}_{il}$ of size $n \times z$ with z chosen that $\mathbf{Z}_{il}$ did not exceed 10 Gb. Corresponding computations on the GPU used 250 blocks of size of $n \times 2048$ for $n \leq 12800$, and $n/2 \times 2048$ otherwise. For matrix inversion, use of LAPACK routines POTRF and POTRI for the complete matrix on both CPU and GPU was contrasted with the block algorithm described above on the GPU (GPUB). This used a block size ($\mathbf{G}_{CC}$) of $n_b = 2048$ and, as suggested by Benner *et al.* (2011), employed a hybrid algorithm with LAPACK routines (POTRF, TRTRI and LAUUM) executed on the CPU, using all 4 cores.

Computing environment. Calculations were carried out on a desktop computer running Linux, with CUDA 5.0. This was equipped with a quad-core Intel I7-960 processor rated at 3.2 Ghz with 8 Mb cache and 12 GB of RAM, and GPU capable NVIDIA GeForce GT240 graphics card with 96 cores, a clock speed of 1.46GHz and 1 Gb of memory. Programs were written in Fortran and compiled using gfortran (gcc 4.4.3), loading BLAS and LAPACK routines from the CUBLAS and CULA libraries and the Intel MKL 11.0 library for computations on the GPU and CPU, respectively.

RESULTS

Computing times required to form the product $\mathbf{ZZ}'$, shown on a logarithmic scale, are constrasted in Figure 2. With $sn(n + 1)$ floating point operations per product, half multiplications and half additions, these increase quadratically with the number of individuals. As previously shown by Aguilar *et al.* (2011) (though they utilised BLAS routine GEMM which does not exploit the symmetry of **G** and thus requires $2sn^2$ operations), results demonstrate that calculations involved are highly suited to parallel processing. Using all 4 CPU processors available decreased the computing time on average by a factor of 3.69. Employing the GPU reduced times further for all cases, even if n was too large to carry out computations for all n through one call to routine SYRK with the memory available on the GPU device, yielding an average speed-up of 5.16 times. Additional investigations using other values for s (not shown) yielded comparable patterns, suggesting that results are scalable and that similar improvements can be achieved for larger problems.

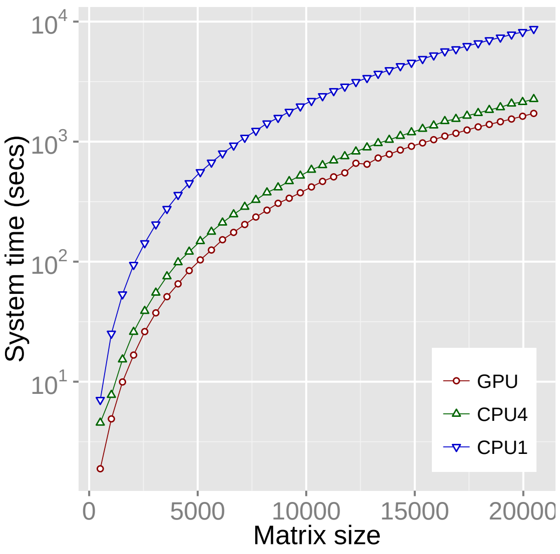

Figure 2. Times for matrix product

Corresponding results for matrix inversion are presented in Figure 3. For this case, parallelisation was slightly less successful with computations using a single CPU for $n > 4000$ requiring on average 3.33 times as long as those utilizing all 4 cores available. For small matrices, processing on the GPU required longer than CPU1. For $n > 4000$, single block computations on the GPU performed best, reducing computing times by factors of 4.14 and 1.23 compared to CPU1 and CPU4, respectively. However, memory available on the GPU restricted these to $n < 15000$. Block-wise inversion on the GPU required similar times than using all CPU cores available in parallel. Other sizes of n_b were tried (not shown), but offered little advantage – indeed for small block sizes, times exceeded those for CPU1. Benner *et al.* (2011) reported greatly increased speeds of computation for their algorithm compared to LAPACK routines, both for parallel computations on the CPU and a hybrid approach, while calculations on the GPU only required matrix sizes of more than 7,000 to be advantageous. Nevertheless, none of their findings could be repeated with our hardware set-up.

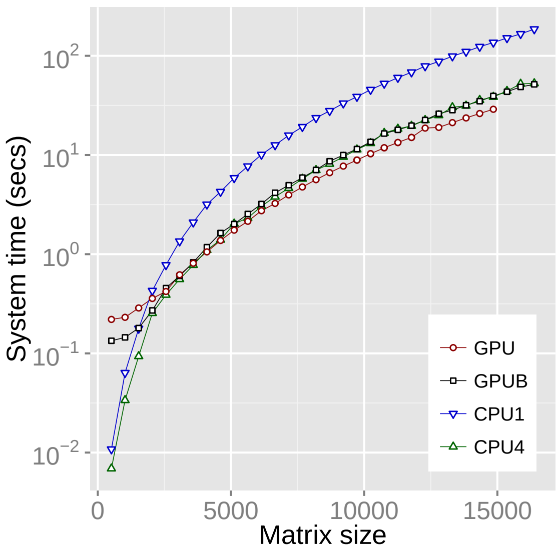

Figure 3. Times for matrix inversion

DISCUSSION

Dense matrix calculations are computationally demanding and the efficiency of computations is greatly influenced by the organisation of loops and memory access. Highly optimised linear algebra routines are available which perform common types of operations and, together with modern compilers and libraries tuned for specific hardware, can yield very fast computations. These routines are freely available and easy to use and, where possible, should be used when programming such applications.

Moreover, corresponding libraries are available to readily utilise multiple (CPU) or very many (GPU) threads. As shown, performing computations in parallel can markedly speed up calculation of the GRM and its inversion. While the advantages of using the GPU over all CPU cores available shown here might appear modest, it should be born in mind that the graphics card utilised only had very basic GPU capabilities. Hence, results should be regarded more as a ‘proof of principle’ rather than being indicative. Modern GPUs targeting general purpose computing have up to 6 Gb memory and thousands of cores, and are capable of performing double precision calculations with huge numbers of floating point operations per second, effectively turning a standard desktop computer into a personal supercomputer. Future work will repeat the calculations shown with more powerful hardware, and is likely to achieve substantially higher reductions in computing time for calculations that can be accelerated using the GPU.

CONCLUSIONS

Utilisation of multiple threads can dramatically reduce computing times for dense matrix calculations, such as required in the context of genomic evaluation. Graphic Processing Units provide powerful hardware for parallelisation of computations, and are likely to see increasing use in animal breeding applications in the future.

ACKNOWLEDGEMENTS

This work was supported by Meat and Livestock Australia under grant B.BFG.0050.

REFERENCES

Aguilar I., Misztal I., Legarra A. and Tsuruta S. (2011) *J. Anim. Breed. Genet.* **128**:422.

Anderson E., Bai Z., Bischof C., Blackford S., Demmel J., Dongarra J., Du Croz J., Greenbaum A., Hammarling S., McKenney A. and Sorensen D. (1999) *LAPACK Users’ Guide*. Society for Industrial and Applied Mathematics, Philadelphia, PA, Third edition.

Benner P., Ezzatti P., Quintana-Ortí E.S. and Remón A. (2011) In *International Conference on High Performance Computing and Simulation (HPCS)*. IEEE, pp. 640–646.

Cole J.B., Newman S., Foertter F., Aguilar I. and Coffey M. (2012) *J. Anim. Sci.* **90**:723.

Dongarra J., Dong T., Gates M., Haidar A., Tomov S. and Yamazaki I. (2012) In *SC12*. Salt Lake City, Utah, November 14, 2012.

Dongarra J.J., Croz J.D., Hammarling S. and Hanson R.J. (1988) *ACM Trans. Math. Softw.* **14**:1.

Ezzatti P., Quintana-Ortí E.S. and Remón A. (2011) *J. Supercomput.* **58**:429.

Humphrey J.R., Price D.K., Spagnoli K.E., Paolini A.L. and Kelmelis E.J. (2010) In *Modeling and Simulation for Defense Systems and Applications V*, E.J. Kelmelis, ed., Proc. SPIE 770502.

Meyer K., Tier B. and Graser H.U. (2013) *J. Anim. Sci.* **00**:000.

NVIDIA Corporation (2013) CUDA parallel programming platform. `http://www.nvidia.com/object/cuda_home_new.html`. Accessed: February 1, 2013.

Quintana E.S., Quintana G., Sun X. and van de Geijn R. (2001) *SIAM J. Sci. Comput.* **22**:1762.

Van Raden P.M. (2008) *J. Dairy Sci.* **91**:4414.