Computing Least Squares Condition Numbers on Hybrid Multicore/GPU Systems

M. Baboulin, J. Dongarra and R. Lacroix

Abstract This chapter presents an efficient computation for least squares conditioning or estimates of it. We propose performance results using new routines on top of the multicore-GPU library MAGMA. This set of routines is based on an efficient computation of the variance–covariance matrix for which, to our knowledge, there is no implementation in current public domain libraries LAPACK and ScaLAPACK.

1 Introduction

Linear least squares (LLS) is a classical linear algebra problem in scientific computing, arising for instance in many parameter estimation problems [5]. We consider the overdetermined full rank linear least squares problem $\min_{x \in \mathbb{R}^n} \|Ax - b\|_2$, with $A \in \mathbb{R}^{m \times n}, m \geq n$ and $b \in \mathbb{R}^m$.

In addition to computing LLS solutions efficiently, an important issue is to assess the numerical quality of the computed solution. The notion of conditioning provides a theoretical framework that can be used to measure the numerical sensitivity of a problem solution to perturbations. Similarly to [2, 3], we suppose that the perturbations on data are measured using the Frobenius norms for matrices and the Euclidean norm for vectors. Then we can derive simple formulas for the condition number of the LLS solution x or its components using the R factor (from the QR decomposition of A), the residual and x. We can also use the variance–covariance matrix.

M. Baboulin (✉)
Inria and University of Paris-Sud, Orsay, France
e-mail: marc.baboulin@inria.fr

J. Dongarra
University of Tennessee, Knoxville, TN, USA
e-mail: dongarra@eecs.utk.edu

R. Lacroix
Inria and University Pierre et Marie Curie, Paris, France
e-mail: remi.lacroix@inria.fr

M. G. Cojocaru et al. (eds.), *Interdisciplinary Topics in Applied Mathematics, Modeling and Computational Science,* Springer Proceedings in Mathematics & Statistics 117,
DOI 10.1007/978-3-319-12307-3_6

In this chapter, we propose algorithms to compute LLS condition numbers in a computational time that is affordable for large-scale simulations, in particular using the variance–covariance matrix. We also compute statistical condition estimates that can be obtained cheaply ($\mathcal{O}(n^2)$ operations) and with a satisfying accuracy using an approach similar to [6, 8]. For these algorithms, we describe an implementation for LLS conditioning using the MAGMA library [4, 10], which is a dense linear algebra library for heterogeneous multicore-GPU architectures with interface similar to LAPACK. Our implementation takes advantage of current hybrid multicore-GPU systems by splitting the computational work between the GPU and the multicore host. We present performance results, and these results are compared with the computational cost for computing the LLS solution itself.

2 Closed Formulas and Statistical Estimates

In this section, we recall some existing formulas to compute or estimate the condition number of an LLS solution x or of its components. We suppose that the LLS problem has already been solved using a QR factorization (the normal equations method is also possible but the condition number is then proportional to $cond(A)^2$). Then the solution x, the residual $r = b - Ax$, and the factor $R \in \mathbb{R}^{n \times n}$ of the QR factorization of A are readily available.

From [3], we obtain a closed formula for the absolute condition number of the LLS solution as

$$\kappa_{LS} = \|R^{-1}\|_2 \left(\|R^{-1}\|_2^2 \|r\|_2^2 + \|x\|_2^2 + 1\right)^{\frac{1}{2}}, \tag{1}$$

where x, r and R are exact quantities.

We can also compute $\bar{\kappa}_{LS}$, statistical estimate of κ_{LS} that is obtained using the condition numbers of $z_i^T x$ where $z_1, z_2, ..., z_q$ are q random orthogonal vectors of $\mathbb{R}^n$, obtained for instance via a QR factorization of a random matrix $Z \in \mathbb{R}^{n \times q}$. The condition number of $z_i^T x$ can be computed using the expression given in [3] as

$$\kappa_i = \left(\|R^{-1}R^{-T}z_i\|_2^2 \|r\|_2^2 + \|R^{-T}z_i\|_2^2 (\|x\|_2^2 + 1)\right)^{\frac{1}{2}}. \tag{2}$$

Then $\bar{\kappa}_{LS}$ is computed using the expression $\bar{\kappa}_{LS} = \frac{\omega_q}{\omega_n}\sqrt{\sum_{j=1}^{q} \kappa_j^2}$ with $\omega_q = \sqrt{\frac{2}{\pi(q-\frac{1}{2})}}$. As explained in [6], choosing $q = 2$ random vectors enables us to obtain a satisfying accuracy.

By considering in Eq. (2) the special case where $z_i = e_i$ where e_i is a canonical vector of $\mathbb{R}^n$, we can express the condition number of the component $x_i = e_i^T x$ in Eq. (3). Then we can calculate a vector $\kappa_{CW} \in \mathbb{R}^n$ with components κ_i being the exact condition number of x_i and expressed by

$$\kappa_i = \left(\|R^{-1}R^{-T}e_i\|_2^2 \|r\|_2^2 + \|R^{-T}e_i\|_2^2 (\|x\|_2^2 + 1)\right)^{\frac{1}{2}}. \tag{3}$$

We can also find in [6, 8] a statistical estimate for each κ_i.

3 Variance–Covariance Matrix

In many physical applications, LLS problems are expressed using a statistical model often referred to as *linear statistical model* where we have to solve

$$b = Ax + \epsilon,\ A \in \mathbb{R}^{m\times n},\ b \in \mathbb{R}^m,$$

with ϵ being a vector of random errors having expected value 0 and variance-covariance $\sigma_b^2 I$. The matrix A is called the regression matrix and the unknown vector x is called the vector of regression coefficients. Following the Gauss–Markov theorem [11], the least squares estimate $\hat{x}$ is the linear unbiased estimator of x satisfying $\hat{x} = arg\min_{x\in\mathbb{R}^n} \|Ax - b\|_2$,

with minimum variance–covariance equal to

$C = \sigma_b^2 (A^T A)^{-1}$.

The diagonal elements c_{ii} of C give the variance of each component $\hat{x}_i$. The off-diagonal elements $c_{ij},\ i \neq j$ give the covariance between $\hat{x}_i$ and $\hat{x}_j$. Then instead of computing condition numbers (which are notions more commonly handled by numerical linear algebra practitioners) physicists often compute the variance-covariance matrix whose entries are intimately correlated with condition numbers κ_i and κ_{LS} mentioned previously.

When the variance–covariance matrix has been computed, the condition numbers can be easily obtained. Indeed, we can use the fact that $\left\|R^{-1}\right\|_2^2 = \frac{\|C\|_2}{\sigma_b^2}$, $\|R^{-T}e_i\|_2^2 = \frac{c_{ii}}{\sigma_b^2}$, and $\|R^{-1}R^{-T}e_i\|_2 = \frac{\|C_i\|_2}{\sigma_b^2}$ where C_i and c_{ii} are respectively the ith column and the ith diagonal element of the matrix C. Then by replacing respectively in Eqs. (1) and (3), we get the formulas

$$\kappa_{LS} = \frac{\|C\|_2^{1/2}}{\boldsymbol{\sigma}_b}((m-n)\|C\|_2 + \|x\|_2^2 + 1)^{1/2}, \tag{4}$$

and

$$\kappa_i = \frac{1}{\boldsymbol{\sigma}_b}((m-n)\|C_i\|_2^2 + c_{ii}(\|x\|_2^2 + 1))^{1/2}. \tag{5}$$

Note that, when $m > n$, $\frac{1}{m-n}\ \|r\|_2^2$ is an unbiased estimate of $\boldsymbol{\sigma}_b^2$ [7, p. 4].

4 Implementation Details

We developed a set of routines that compute the following quantities using the MAGMA library (release 1.2.1):

- Variance–covariance matrix C
- κ_{LS}, condition number of x
- κ_{CW}, vector of the κ_i, condition numbers of the solution components

- $\bar{\kappa}_{LS}$, statistical estimate of κ_{LS}
- $\bar{\kappa}_{CW}$, vector of the statistical estimates κ_i

The variance–covariance computation requires inverting a triangular matrix and multiplying this triangular matrix by its transpose (similarly to the LAPACK routine DPOTRI [1, p. 26] that computes the inverse of a matrix from its Cholesky factorization). These operations use a block algorithm, which, for the diagonal blocks, is performed recursively. The recursive part is performed by the CPU for sake of performance while the rest of the algorithm is executed on the GPU.

The computation of the exact condition number κ_{LS} from the variance–covariance using Eq. (4) involves the computation of the spectral norm of C which is generally computed via an SVD. However, since A is a full-rank matrix, C is symmetric positive definite and its singular values coincide with its eigenvalues. Then we use an eigenvalue decomposition of C which is faster than an SVD because it takes into account the symmetry of C. The tridiagonalization phase is performed on the GPU while the subsequent eigenvalue computation is performed on the CPU host.

The statistical estimates require the generation and orthonormalization of random vectors followed by two triangular solves. The random generation and the triangular solves are performed on the GPU. The orthonormalization is performed on the CPU because it is applied to small matrices (small number of samples).

5 Performance Results

Our experiments have been achieved on a multicore processor Intel Xeon E5645 (2 sockets $\times$ 6 cores) running at 2.4 GHz (the cache size per core is 12 MB and the size of the main memory is 48 GB). This system hosts two GPU NVIDIA Tesla C2075 running at 1.15 GHz with 6 GB memory each. MAGMA was linked with the libraries MKL 10.3.8 and CUDA 4.1, respectively, for multicore and GPU. We consider random LLS problems obtained using the method given in [9] for generating LLS test problems with known solution x and residual norm.

We plot in Fig. 1, the CPU time to compute LLS solution and condition numbers using 12 threads and 1 GPU. We observe that the computation of the variance–covariance matrix and of the components conditioning κ_i are significantly faster than the cost for solving the problem with respectively a time factor larger than 3 and 2, this factor increasing with the problem size. The κ_i are computed using the variance-covariance matrix via Eq. (5). The time overhead between the computation of the κ_i and the variance–covariance computation comes from the computation of the norms of the columns (routine cublasDnrm2) which has a nonoptimal implementation.

As expected, the routines SCE_LLS and SCE_LLS_CW that compute statistical condition estimates for the solution and all solution components, respectively, outperform the other routines. Note that we did not mention on this graph the performance for computing κ_{LS} using Eq. (4). Indeed this involves an eigenvalue decomposition of the variance–covariance matrix (MAGMA routine magma_dsyevd_gpu), which turns out to be much slower than the LLS solution (MAGMA routine

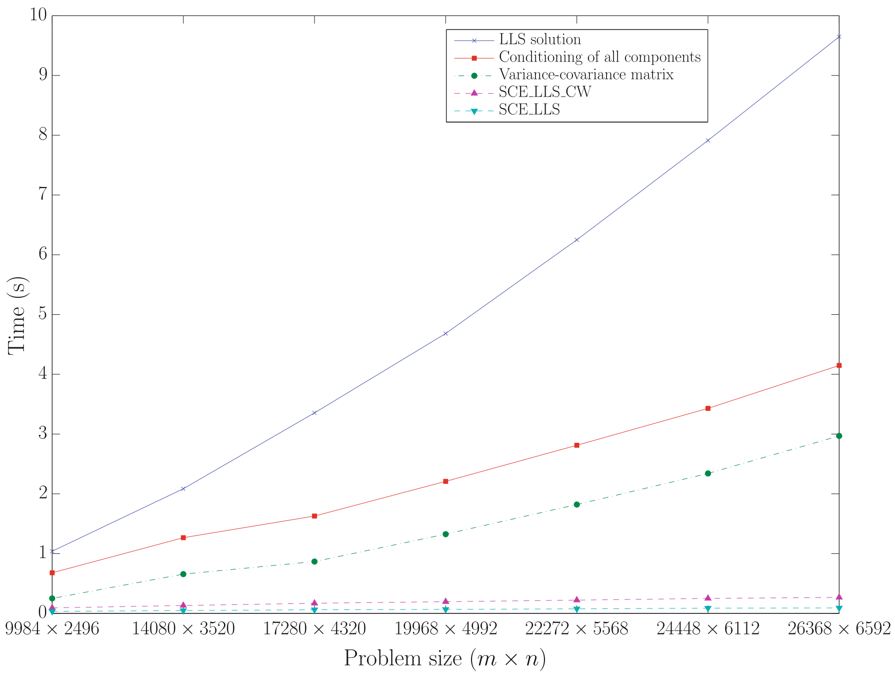

Fig. 1 Performance for computing LLS condition numbers with MAGMA

magma_dgels3_gpu) in spite of a smaller number of flops ($\mathcal{O}(n^3)$ vs $\mathcal{O}(mn^2)$) which shows that having an efficient implementation on the targeted architecture is essential to take advantage of the gain in flops.

We can illustrate this by comparing in Fig. 2 the time for computing an LLS solution and its conditioning using LAPACK and MAGMA. We observe that MAGMA provides faster solution and condition number but, contrary to LAPACK, the computation of the condition number is slower than the time for the solution, in spite of a smaller flop count. This shows the need for improving the Gflop/s performance of eigensolvers or SVD solvers for GPUs, but it also confirms the interest of considering statistical estimates on multicore-GPU architectures to get fast computations.

6 Conclusion

We proposed new implementations for computing LLS condition numbers using the software libraries LAPACK and MAGMA. The performance results that we obtained on a current multicore-GPU system confirmed the interest of using statistical

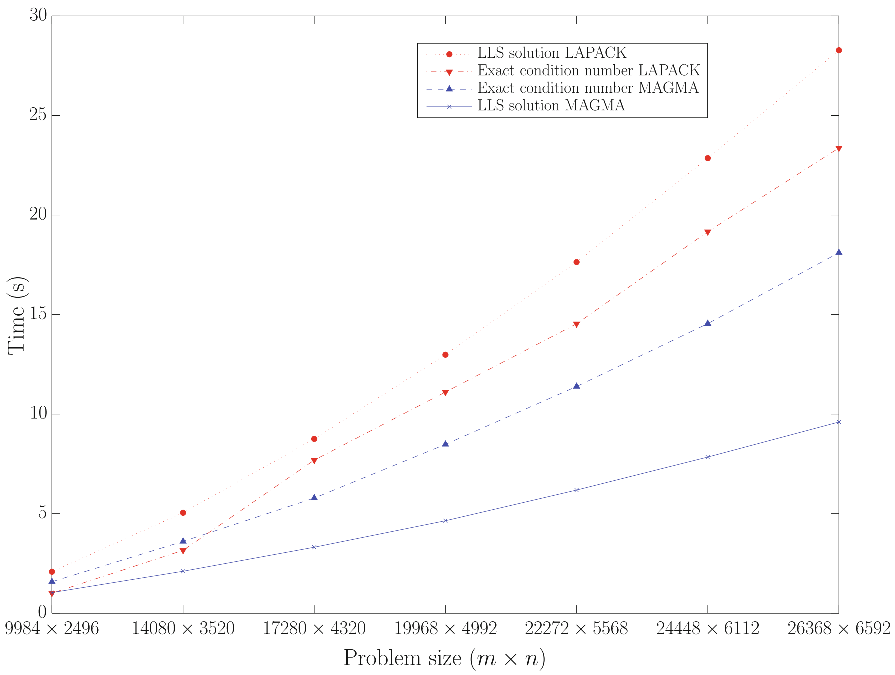

Fig. 2 Time for LLS solution and condition number

condition estimates. New routines will be integrated in the next releases of LAPACK and MAGMA to compute the variance–covariance matrix after a linear regression.

References

1. Anderson, E., Bai, Z., Bischof, C., Blackford, S., Demmel, J., Dongarra, J., Croz, J.D., Greenbaum, A., Hammarling, S., McKenney, A., Sorensen, D.: LAPACK Users' Guide, 3rd edn. SIAM, Philadelphia (1999)
2. Arioli, M., Baboulin, M., Gratton, S.: A partial condition number for linear least-squares problems. SIAM J. Matrix Anal. Appl. **29**(2), 413–433 (2007)
3. Baboulin, M., Dongarra, J., Gratton, S., Langou, J.: Computing the conditioning of the components of a linear least squares solution. Numer. Linear Algebra Appl. **16**(7), 517–533 (2009)
4. Baboulin, M., Dongarra, J., Tomov, S.: Some issues in dense linear algebra for multicore and special purpose architectures. In: 9th International Workshop on State-of-the-Art in Scientific and Parallel Computing (PARA'08), *Lecture Notes in Computer Science*, vol. 6126–6127. Springer-Verlag (2008)
5. Baboulin, M., Giraud, L., Gratton, S., Langou, J.: Parallel tools for solving incremental dense least squares problems. Application to space geodesy. J Algorithms Comput. Technol. **3**(1), 117–133 (2009)

6. Baboulin, M., Gratton, S., Lacroix, R., Laub, A.J.: Statistical estimates for the conditioning of linear least squares problems. In: Proceedings of 10th International Conference on Parallel Processing and Applied Mathematics (PPAM 2013) (2013)
7. Björck, A.: Numerical Methods for Least Squares Problems. SIAM, Philadelphia (1996)
8. Kenney, C.S., Laub, A.J., Reese, M.S.: Statistical condition estimation for linear least squares. SIAM J. Matrix Anal. Appl. **19**(4), 906–923 (1998)
9. Paige, C.C., Saunders, M.A.: LSQR: An algorithm for sparse linear equations and sparse least squares. ACM Trans. Math. Softw. **8**(1), 43–71 (1982)
10. Tomov, S., Dongarra, J., Baboulin, M.: Towards dense linear algebra for hybrid GPU accelerated manycore systems. Parallel Comput. **36**(5&6), 232–240 (2010)
11. Zelen, M.: Linear estimation and related topics. In: Todd, J. (ed.) Survey of Numerical Analysis, pp. 558–584. McGraw-Hill, New York (1962)