Parallel tools for solving incremental dense least squares problems. Application to space geodesy. LAPACK Working Note 179

Marc Baboulin* Luc Giraud† Serge Gratton* Julien Langou‡

September 19, 2006

Abstract

We present a parallel distributed solver that enables us to solve incremental dense least squares arising in some parameter estimation problems. This solver is based on ScaLAPACK [8] and PBLAS [9] kernel routines. In the incremental process, the observations are collected periodically and the solver updates the solution with new observations using a QR factorization algorithm. It uses a recently defined distributed packed format [3] that handles symmetric or triangular matrices in ScaLAPACK-based implementations. We provide performance analysis on IBM pSeries 690. We also present an example of application in the area of space geodesy for gravity field computations with some experimental results.

Keywords: scientific computing, dense linear algebra, parallel distributed algorithms, ScaLAPACK, QR factorization, gravity field computation.

1 Introduction

Many parameter estimation problems lead to linear least squares problems (LLSP) of the type

$$\min_{x \in \mathbb{R}^n} \|Ax - b\|_2 \tag{1}$$

where each row of $A \in \mathbb{R}^{m \times n}$ and $b \in \mathbb{R}^m$ are collected periodically. The result is that we have to solve successive LLSP. Throughout this paper, b and A will be called respectively the observation vector and the data matrix. After accumulating a sufficient number of observations and/or by using regularization techniques, A is a full column rank matrix. In that case (1) has a unique solution [13, p. 237].

In general n and m are large (at least several tens of thousands for n and several hundreds of thousands for m) and we have $m \gg n$. This problem requires the use of parallel platforms that currently offer increasing capabilities in terms of floating-point operations per second and memory storage.

The normal equations method that solves the system

$$A^T A x = A^T b$$

is a classical way of solving an LLSP. This method has the advantage of being easy to code and faster than other direct methods. For instance we deduce from [13, p. 225 and p. 238] that when

*CERFACS, 42 avenue Gaspard Coriolis, 31057 Toulouse Cedex, France. Email : baboulin@cerfacs.fr - gratton@cerfacs.fr

†ENSEEIHT, 2 rue Camichel, 31071 Toulouse cedex, France. Email : giraud@n7.fr

‡Dept. of Mathematical Sciences, The University of Colorado at Denver and Health Sciences Center, Downtown Denver Campus, Denver, Colorado 80217-3364, USA. Email : langou@math.cudenver.edu

$m \gg n$, the normal equations approach requires about half the flop count of the Householder QR factorization (mn^2 vs $2mn^2$). This explains why the normal equations method is often favored by statisticians. For large scale problems, a parallel distributed solver that performs the assembly of the normal equations and computes a solution using a Cholesky factorization is described in [2]. This solver performs the assembly of the normal equations at the sustained peak rate of a matrix-matrix product and stores the symmetric matrix $A^T A$ compactly. The Cholesky factorization gives performance results similar to the corresponding ScaLAPACK [8] routine on moderately parallel platforms (up to 32 processors) while requiring about half the memory.
When the application requires an accurate solution, orthogonal transformations and in particular the QR factorization may be more appropriate. In that case, the computational time may be an issue if we want to compute a solution that complies with the requirements of the physical application (e.g. daily computation). Also, similarly to the storage of $A^T A$ for the normal equations approach, we have to take advantage of the triangular structure of the R factor.
Out-of-core parallel QR solvers were implemented in several projects [14, 15] that are based on the PLAPACK library [20] and where only the blocks of one triangular part are stored. But for faster computation of a partial solution or of the covariance $(A^T A)^{-1}$, we may want to keep the R factor in-core.
We propose in this paper a parallel implementation based on ScaLAPACK routines that solve incremental least squares using QR factorization and stores the R factor in-core and compactly. This implementation uses a recently defined distributed packed storage [3] on top of ScaLAPACK routines. In the incremental process, there will be a trade-of between performance and memory depending on how many observations we consider at a time for updating the R factor.

This paper is organized as follows. In Section 2, we recall how to use the QR factorization for solving incremental least squares. The purpose of Section 3 is to give an overview of the distributed packed storage that can be used in ScaLAPACK-based implementation that handle symmetric or triangular matrices. In Section 4, we describe the parallel implementation of the QR updating functionality using the distributed packed format, this includes algorithms description and performance analysis on the IBM pSeries 690 machine. In Section 5, we present first results in the area of gravity field computations. Finally, some concluding comments are given in Section 6.

2 QR approach for incremental linear least squares

A reliable way of solving LLSP consists of using orthogonal transformations. The commonly used QR factorization can be performed by using orthogonal transformations called Householder reflections [13, p. 208]. The QR factorization of A is given by

$$A = Q \begin{pmatrix} R \\ 0 \end{pmatrix}$$

where Q is an m-by-m orthogonal matrix and R is an n-by-n upper triangular matrix. Since A has full column rank, then R is nonsingular. If we denote $Q = \begin{pmatrix} Q_1 & Q_2 \end{pmatrix}$ where Q_1 and Q_2 correspond respectively to the first n columns and the $m-n$ remaining columns of Q, then we have $A = Q_1 R$.
From $\|Ax - b\|_2 = \left\|Q^T b - Q^T A x\right\|_2 = \left\| \begin{matrix} {Q_1}^T b - Rx \\ {Q_2}^T b \end{matrix} \right\|_2$, it follows that x can be computed by solving the triangular system $Rx = {Q_1}^T b$.
In the QR method, the matrix R may overwrite the upper triangular part of A while the Householder vectors are stored in the lower trapezoidal part of A.
Note that Q_1 is not required explicitly and we just need to apply Q_1^T to a vector. The computation of ${Q_1}^T b$ can be performed by applying the Householder transformation used to compute R to the vector b. This is achieved by appending b to A and factorizing $\begin{pmatrix} A & b \end{pmatrix}$.

This factorization overwrites b by $\tilde{b}$ and we solve $Rx = \tilde{b}$ using the first n elements of $\tilde{b}$.
If we neglect the cost of the triangular solve, then the computational cost of the least squares solution using a Householder QR factorization is $2n^2(m - n/3)$ [13, p. 225].

We now describe an algorithm that aims to solve an incremental least squares problem where, at each step of this algorithm, we solve an LLSP.

Let $\mathcal{A}_N$ and $\mathcal{B}_N$ be respectively the cumulated parameter matrix and observation vector up to date N.
Let A_N and b_N be respectively the data matrix and observation vector that has been collected at date N.
Then we have:

$$\mathcal{A}_N = \begin{pmatrix} A_1 \\ \vdots \\ A_N \end{pmatrix} \text{ and } \mathcal{B}_\mathrm{N} = \begin{pmatrix} b_1 \\ \vdots \\ b_N \end{pmatrix}.$$

The least squares problem to solve at date N can be stated as: $\min_{x \in \mathbb{R}^n} \|\mathcal{A}_N x - \mathcal{B}_N\|_2$. We assume that we have solved the least squares problem at date N.
Then, at date $N+1$, we have:

$$\mathcal{A}_{N+1} = \begin{pmatrix} \mathcal{A}_N \\ A_{N+1} \end{pmatrix} \text{ and } \mathcal{B}_\mathrm{N+1} = \begin{pmatrix} \mathcal{B}_N \\ b_{N+1} \end{pmatrix}.$$

We consider the case where we use a QR approach that utilizes Householder transformations. If we denote by R_N the R-factor obtained at date N, then R_{N+1} corresponds to the R-factor in the QR factorization of $\mathcal{A}_{N+1}$. But we observe that the QR factorization of $\mathcal{A}_{N+1} = \begin{pmatrix} \mathcal{A}_N \\ A_{N+1} \end{pmatrix}$ produces the same upper triangular factor as does the factorization of $\begin{pmatrix} R_N \\ A_{N+1} \end{pmatrix}$ i.e R_{N+1}.
Furthermore, the storage of the Householder vectors can be avoided by appending the observation vector b_N to the matrix to be factorized and overwriting this vector with the $(n+1)$-th column of the so-obtained triangular factor.
The result is that the updating of the R-factor at date $N+1$ is done by performing the QR factorization of $\begin{pmatrix} R_N & \tilde{\mathcal{B}}_N \\ A_{N+1} & b_{N+1} \end{pmatrix}$ where $\tilde{\mathcal{B}}_N$ contains the updated values of $\mathcal{B}_K$ $(K \leq N)$ resulting from the N previous QR factorizations.
This enables us to obtain the upper triangular matrix $\begin{pmatrix} R_{N+1} & \tilde{\mathcal{B}}_{N+1} \end{pmatrix}$ and the solution x_{N+1} is computed by solving $R_{N+1} x_{N+1} = \mathcal{Z}_{N+1}$ where $\mathcal{Z}_{N+1}$ contains the first n elements of $\tilde{\mathcal{B}}_{N+1}$. The algorithm that performs the QR factorization of $\begin{pmatrix} R_N & \tilde{\mathcal{B}}_N \\ A_{N+1} & b_{N+1} \end{pmatrix}$ will be described in Section 4.
Similar algorithms for incremental Cholesky and QR factorizations are described in [7, p. 224] in the framework of block angular least squares problems. Note that alternative algorithms referred to as covariance algorithms can update $(\mathcal{A}^T \mathcal{A})^{-1}$ or a factor of it [18].

3 Brief review of packed distributed storage

Contrary to the serial library LAPACK [1], the parallel library ScaLAPACK does not currently support packed format for symmetric, Hermitian or triangular matrices [11]. We recently described in [3] a packed storage scheme that allows us to store compactly symmetric or triangular matrices in parallel implementation on top of ScaLAPACK routines. We quickly recall in this section the principle of this packed format.

ScaLAPACK proposes a data layout based on a two-dimensional block cyclic distribution. In this type of distribution, a matrix of size n is divided into blocks of size s (if we use square blocks) that are assigned to processors in cyclic manner according to a $p \times q$ process grid. We refer to [8] for more details about this data layout. The blocks of size s that are spread among processors are called elementary blocks and the blocks of size $p.s \times q.s$ corresponding to the $p \times q$ process grid are called grid blocks.

In order to be stored in a distributed packed format, a matrix is first partitioned into larger square blocks of size b such that b is proportional to $l.s$ where l is the least common multiple of p and q ($b \geq l.s$). We define these blocks as "distributed blocks".
In the remainder of this paper, the algorithms will be expressed in terms of distributed blocks that will be simply called "blocks". Note that the distributed block performs naturally what is defined in [20] as algorithmic blocking or tile for out-of-core implementations [15].
The following figure summarizes the hierarchy between the elementary block (hosted by one process), the grid block (corresponding to the process grid), and the distributed block (square block consisting of grid blocks). It shows the three kinds of blocks that we get when we consider a 2×3 process grid, $s = 1$, $b = 6$ and each block is labeled with the number of process that stores it.

$$\begin{array}{|c|}\hline 0 \\ \hline\end{array} \text{ : elementary block, } \begin{array}{|c|c|c|}\hline 0 & 1 & 2 \\ \hline 3 & 4 & 5 \\ \hline\end{array} \text{ : grid block, } \begin{array}{|c|c|c||c|c|c|}\hline 0 & 1 & 2 & 0 & 1 & 2 \\ \hline 3 & 4 & 5 & 3 & 4 & 5 \\ \hline\hline 0 & 1 & 2 & 0 & 1 & 2 \\ \hline 3 & 4 & 5 & 3 & 4 & 5 \\ \hline\hline 0 & 1 & 2 & 0 & 1 & 2 \\ \hline 3 & 4 & 5 & 3 & 4 & 5 \\ \hline\end{array} \text{ : distributed block.}$$

We consider here a matrix A partitioned into distributed blocks A_{ij} and A can be either symmetric or upper triangular or lower triangular. We propose to store A compactly in a distributed packed format that consists in storing the blocks belonging to the upper or the lower triangle of A in a ScaLAPACK matrix ADP (A Distributed Packed).
The blocks of A will be stored horizontally in ADP so that the entries in the elementary, grid and distributed blocks are contiguous in memory and then will map better to the highest levels of cache.

Let us consider the following symmetric matrix A described using distributed blocks, that is

$$A = \begin{pmatrix} A_{11} & A_{21}^T & A_{31}^T \\ A_{21} & A_{22} & A_{32}^T \\ A_{31} & A_{32} & A_{33} \end{pmatrix}.$$

We provide two ways of storing A using our distributed packed format. In the Lower distributed packed format, the lower triangle of A is packed **by columns** in ADP i.e:

$$ADP = \begin{bmatrix} A_{11} & A_{21} & A_{31} & A_{22} & A_{32} & A_{33} \end{bmatrix}.$$

In the Upper distributed packed format, the upper triangle of A is packed **by rows** in ADP i.e:

$$ADP = \begin{bmatrix} A_{11} & A_{21}^T & A_{31}^T & A_{22} & A_{32}^T & A_{33} \end{bmatrix}.$$

The distributed packed storage for upper and lower triangular matrices follows from that of a symmetric matrix since the upper triangular blocked matrix $A = \begin{pmatrix} A_{11} & A_{12} & A_{13} \\ 0 & A_{22} & A_{23} \\ 0 & 0 & A_{33} \end{pmatrix}$ is stored in a packed distributed format as

$$ADP = \begin{bmatrix} A_{11} & A_{12} & A_{13} & A_{22} & A_{23} & A_{33} \end{bmatrix}$$

and the lower triangular blocked matrix $A = \begin{pmatrix} A_{11} & 0 & 0 \\ A_{21} & A_{22} & 0 \\ A_{31} & A_{32} & A_{33} \end{pmatrix}$ will be stored as

$$ADP = \left[\begin{array}{cccccc} A_{11} & A_{21} & A_{31} & A_{22} & A_{32} & A_{33} \end{array} \right].$$

We point out that the matrix ADP corresponds "physically" to a ScaLAPACK array that is laid out on the $p \times q$ mesh. We also specify that, contrary to LAPACK where upper and lower triangular matrices are both packed by columns, our distributed packed format is different for upper and lower triangular matrices since they are respectively packed by rows and columns. Note also that the diagonal blocks are full blocks and thus do not exploit the triangular or symmetric structure.
In the rest of this paper we will use the following designations and notations. The distributed packed format will be simply referred to as the packed format and an implementation using this format as a packed implementation. We denote by $N = \frac{n}{b}$ the number of block rows in A. The packed structure ADP will be described using the A_{ij} as previously or will be denoted as the blocked matrix $\left[\begin{array}{cccccc} B_1 & B_2 & B_3 & B_4 & B_5 & B_6 \end{array} \right]$. A block $B_k = A_{ij}$ in ADP will be addressed through the indirect addressing INDGET that maps (i,j) to k (this mapping depends whether a lower or an upper packed storage is chosen).

4 Parallel implementation of QR updating

4.1 Description of the algorithm

A packed implementation of the QR factorization updating can be based on the ScaLAPACK routines PDGEQRF (QR factorization) and PDORMQR (multiplication by Q^T).
We suppose that the R factor is partitioned into distributed blocks $\begin{pmatrix} B_1 & B_2 & B_3 \\ 0 & B_4 & B_5 \\ 0 & 0 & B_6 \end{pmatrix}$ and stored in a distributed packed format as $\left[\begin{array}{cccccc} B_1 & B_2 & B_3 & B_4 & B_5 & B_6 \end{array} \right]$ that we also denote by $B_{1:6}$.
The new observations are stored in a block matrix $L_{1:N}$ that contains $N.b$ columns and we first assume that L contains b rows. The updating of R is obtained by successively performing the QR factorization of each block row of R with L, as described below. At the first step, we factor:

$$\begin{array}{|c|}\hline B_{1:3} \\ \hline L_{1:3} \\ \hline \end{array} \longrightarrow \begin{array}{|c|}\hline \widetilde{B}_{1:3} \\ \hline \widetilde{L}_{1:3} \\ \hline \end{array}$$

and we advance the updating of the R factor as follows:

$$\begin{array}{|c|}\hline B_{4:5} \\ \hline \widetilde{L}_{2:3} \\ \hline \end{array} \longrightarrow \begin{array}{|c|}\hline \widetilde{B}_{4:5} \\ \hline \overline{L}_{2:3} \\ \hline \end{array}$$

and so on until completion.
We now consider the work array $C = \begin{bmatrix} B_{j:j+N-i} \\ \widetilde{L}_{i:N} \end{bmatrix}$ where $j = INDGET(i,i)$, i.e C contains $2b$ rows and $(N-i+1)b$ columns.
The different ways for implementing the i-th stage in the R updating are described in Figure 1, where the shaded part refers to the part of C that is factored by the routine PDGEQRF and the dark shaded part represents the part of C to which we apply the Householder transformations

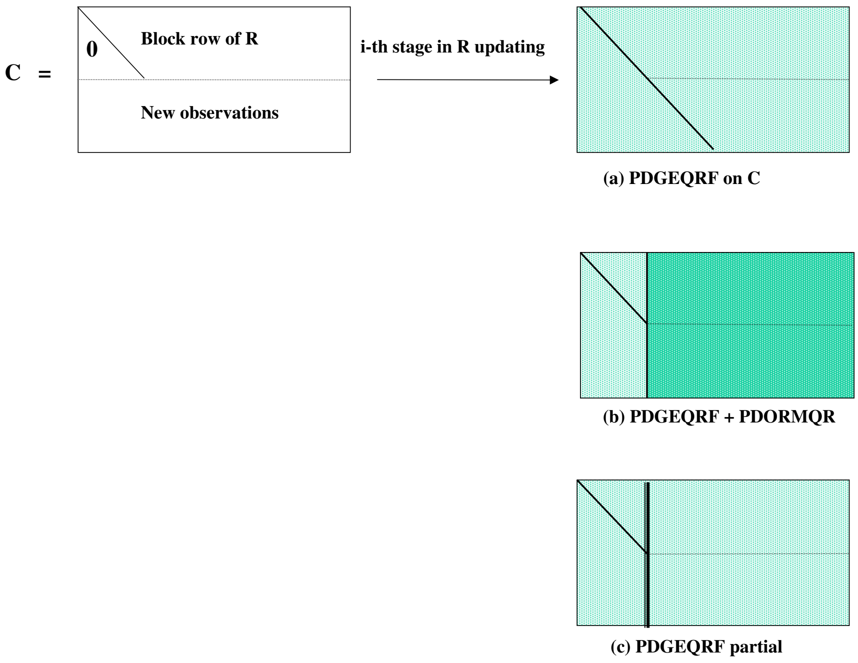

Figure 1: Different possibilities for the QR factorization of C.

using the routine PDORMQR.
In Figure 1 (a), we perform the QR factorization of the whole matrix C. In that case, using the flop counts given in [13, p. 213 and 225], the R updating algorithm involves about $4bn^2$ operations (if $n \gg b$).
This flop count can be reduced by performing a QR factorization of the first b columns of C subsequently followed by the updating of the remaining columns by the Householder transformations (Figure 1 (b)). From [13], the computational cost becomes about $3bn^2$ (still if $n \gg b$).
Figure 2 compares the gigaflops rate of a QR factorization of a $b \times 2b$ matrix performed either using PDGEQRF on the whole matrix or using PDGEQRF on the first b columns then PDORMQR on the remaining b columns. One may notice that the combination of PDGEQRF and PDORMQR is much less efficient due to the extra-cost in communication (using two routines involves one more synchronization and also PDORMQR exchanges data that was already available while executing PDGEQRF).

Thus step i in the R updating was implemented as a call to the PDGEQRF routine applied to the whole matrix C that stops the factorization after the first b columns (Figure 1 (c)). The global updating involves about $3bn^2$ operations that are performed efficiently. For that reason, we modified the ScaLAPACK routine PDGEQRF by stopping the QR factorization of an m-by-n matrix after the first $\frac{m}{2}$ columns. The so-modified routine is named PDGEQRF_partial. Algorithm 1 represents the updating of the R factor using PDGEQRF_partial.

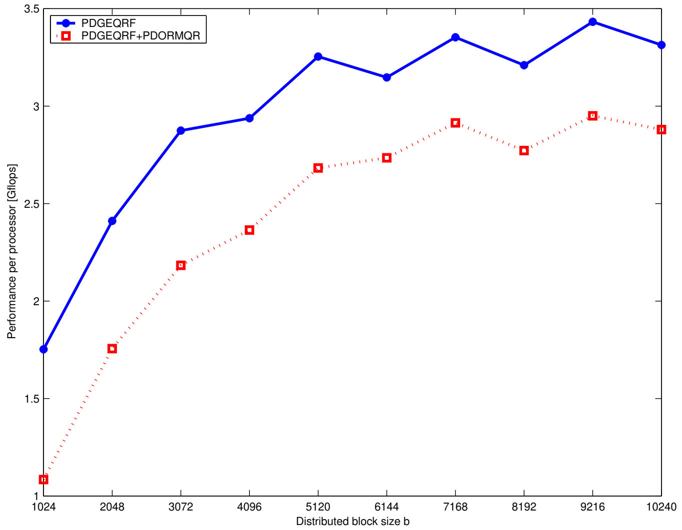

Figure 2: QR factorization of a $b \times 2b$ matrix (4 processors of IBM pSeries 690).

Algorithm 1. *:* Updating the R factor in a QR factorization

read new data in $L_{1:N}$*;* $\widetilde{L}_{1:N} \leftarrow L_{1:N}$

for $i = 1 : N$

$j = INDGET(i, i)$

$$C \leftarrow \left[\begin{array}{c} B_{j:j+N-i} \\ \widetilde{L}_{i:N} \end{array} \right]$$

$\widetilde{C} = qr(C)$ *stopped after the first* b *columns have been factored*

$$\rightarrow \widetilde{C} = \left[\begin{array}{cc} \widetilde{B}_{j:j+N-i} & \\ * & \overline{L}_{i+2:N} \end{array} \right] (\textbf{PDGEQRF\_partial})$$

$B_{j:j+N-i} \leftarrow \widetilde{B}_{j:j+N-i}$

end *(i-loop)*

One may notice that this algorithm does not take into account the upper triangular structure of $B_{j:j}$. As will be confirmed by our experiments, this can be compensated by storing more data into L and thus applying PDGEQRF_partial to a block matrix C containing more than $2b$ rows. As a result, the number of floating-point operations will also decrease.
The initialization of the R factor can be implemented by starting with $R = 0$ and then by successively updating the previous rows by a new one until we have processed the N block rows of R. This allows us to compare the performance of Algorithm 1 with that of PDGEQRF applied to an n-by-n matrix (notice that the factorization time cannot be compared since the initialization of R by successive updates involves 1.5 more operations).
We point out that the ScaLAPACK or PBLAS routines do not have to be modified to support the new packed storage format. The PDGEQRF routine has been modified only for sake of performance.

Note that in the case where A has a given block structure, it is possible to reduce the computational effort by exploiting the structure of A and R [12].

Remark 1. Triangular solve and condition number estimate:

We consider a right-hand side $X = \begin{pmatrix} X_1 \\ \vdots \\ X_N \end{pmatrix}$ that is partitioned into distributed row-blocks of size b and mapped onto the same process grid as the R factor. Then a simple ScaLAPACK-based implementation of the triangular solve in packed format can be described in Algorithm 2.

Algorithm 2. *:* Packed triangular solve

for $i = 1 : N$

$j = INDGET(i, i)$

for $k = i + 1 : N$

$X_i \leftarrow X_i - B_{j+k-i} X_k$ *(***PDGEMV***)*

end *(k-loop)*

$X_i \leftarrow B_j^{-1} X_i$ *(***PDTRSV***)*

end *(i-loop)*

This routine performs the product of R^{-1} by a vector X. Using the same kind of implementation, we obtain routines that perform the products RX, $R^T X$ and $R^{-T} X$ using the packed format. Then it becomes easy to get a packed implementation of the condition number of $R^T R$ based for instance on the Power method or on the Lanczos method [13]. Since $A^T A = R^T R$ and $K(A^T A) = K(A)^2$, we obtain $K(A)$ from $\sqrt{K(R^T R)}$.

4.2 Performance results

Our algorithm was implemented on an IBM pSeries 690 (2 nodes of 32 processors Power-4/1.7 GHz and 64 Gbytes memory per node) installed at CINES and linked with the PBLAS and ScaLAPACK libraries provided by the vendor (in particular the Pessl library).
There is only one ScaLAPACK routine that must be considered when tuning our program parameters s, b and the $p \times q$ process grid. Here s will be chosen equal to 128 since it provides good performance of the PDGEQRF routine on the chosen platform. The $p \times q$ process grid is determined in accordance with [10] i.e such that $\frac{1}{4} \leq \frac{p}{q} \leq \frac{1}{2}$. The value of b cannot be too large since it also influences the size of the matrix C and then the required storage for the calculation. We take $b = l.s$ in our experiments.

Table 1 compares the performance of the initialization of R by successive updates with the performance of a QR factorization of an n-by-n matrix using PDGEQRF. To see the effect of the communication on the performance, we choose values of n such that each processor uses roughly the same amount of memory. We notice that the gigaflops rates of our implementation are similar to the ones of ScaLAPACK for all processor counts considered in this study.

Let n_L be the number of rows in the matrix L that contains the new observations for updating the QR factorization. In Table 2, we update a 25600×25600 matrix R by 51200 new observations and n_L varies from 512 to 25600. As expected at the end of Section 4.1, the number of operations decreases as n_L increases. This gain in operations is evaluated by computing the ratio between the

Table 1: Initialization of R by updates vs ScaLAPACK QR (Gflops).

problem size n $p \times q$ process grid	10240 1	14336 1×2	20480 1×4	28672 2×4	40960 2×8	61440 4×8	81920 4×16
Initialization of R	2.47	3.02	3.30	2.87	2.89	2.80	2.37
ScaLAPACK PDGEQRF	3.50	3.36	3.20	3.25	2.93	2.83	2.63

operations involved in the updating of R and the operations required in a QR factorization of the 76800×25600 matrix containing the original data and the new observations. Then if n_L increases, the factorization time decreases but the performance is stable (close to the peak performance of the ScaLAPACK routine PDGEQRF). Here again, choosing the best size for L corresponds to finding a compromise between performance (in time) and storage since large L demands more storage.

Table 2: Updating of a 25600×25600 R factor by 51200 new observations (1×4 procs).

Number of rows in L	512	1024	2048	5120	10240	12800	25600
Storage (Gbytes)	0.72	0.75	0.80	0.96	1.22	1.35	2.00
Flops overhead (vs ScaLAPACK)	1.50	1.31	1.22	1.16	1.14	1.14	1.13
Factorization time (sec)	7577	5824	5255	5077	5001	4894	4981
Performance (Gflops)	3.33	3.61	3.59	3.47	3.44	3.50	3.40

5 Application to space geodesy

An important task in space geodesy is the computation of an accurate model of the Earth's gravity field and of the geoid. It will have applications in many scientific areas such as solid-Earth physics, geodesy, oceanography, glaciology and climate change. The geoid corresponds to a particular surface of equal potential of a hypothetical ocean at rest. It is used to defined physical altitudes and to forecast water circulation that enables us to study ocean circulation, ice motion, sea-level change. The computational task is quite challenging because of the huge quantity of daily accumulated data and because of the coupling of the parameters resulting in completely dense matrices. An imminent GOCE* satellite mission [4, 19] will estimate $90,000$ parameters of the Earth's gravitational potential via an incremental least squares problem involving millions of observations.

Following [6], the Earth's gravitational potential V is expressed in spherical coordinates (r, θ, λ) by:

$$V(r,\theta,\lambda) = \frac{GM}{R} \sum_{\ell=0}^{\ell_{max}} \left(\frac{R}{r}\right)^{\ell+1} \sum_{m=0}^{\ell} \overline{P}_{\ell m}(\cos\theta) \left[\overline{C}_{\ell m} \cos m\lambda + \overline{S}_{\ell m} \sin m\lambda\right] \qquad (2)$$

where G is the gravitational constant, M is the Earth's mass, R is the Earth's reference radius, the $\overline{P}_{\ell m}$ represent the fully normalized Legendre functions of degree ℓ and order m and $\overline{C}_{\ell m}$,$\overline{S}_{\ell m}$ are the corresponding normalized harmonic coefficients. In the above expression we have $|m| \leq \ell \leq \ell_{max}$. For GOCE calculations we will have $\ell_{max} \simeq 300$ (about $90,000$ unknowns). For the previous missions CHAMP† and GRACE‡, we have respectively $\ell_{max} \simeq 120$ (about $15,000$ unknowns) and $\ell_{max} \simeq 150$ (about $23,000$ unknowns). We point out that the number of unknown parameters is

*Gravity field and steady-state Ocean Circulation Explorer - European Space Agency
†CHAllenging Minisatellite Payload for Geophysical Research an Application, GFZ, launched July 2000
‡Gravity Recovery and Climate Experiment, NASA, launched March 2002

expressed by

$$n = (\ell_{max} + 1)^2.$$

We have to compute the harmonic coefficients $\overline{C}_{\ell m}$ and $\overline{S}_{\ell m}$ the most accurately as possible.
The gravity field parameters are computed at CNES[§] using the orbit determination software GINS [5]. Measurements which are a function of a satellite position and/or velocity are taken into account. These observations are obtained via ground stations (Laser, Doppler) or other satellites (GPS). Then we aim to minimize the difference between the measurements and the corresponding quantities evaluated from the computed orbit by adjusting given parameters (here the gravity field coefficients). The measured quantities correspond to gravity gradients, i.e the second order spatial derivatives of the gravitational potential V expressed in (2). This yields to a nonlinear least squares problem that can be solved via a Gauss-Newton algorithm [17] by solving successive LLSP $\min_{x \in \mathbb{R}^n} \|Ax - b\|_2$ where

- the unknown x represents the gravity field deviation (difference between the gravity field parameters and reference values),
- the observation vector b contains the difference between the observed gravity gradients and reference values,
- the data matrix A contains the first order spatial derivatives of the gravity gradient with respect to the gravity field coefficients (Jacobian of the nonlinear least squares problem).

In our experiments we consider 10 days of observations using GRACE measurements. The total number of observations $m = 166,451$. The number of computed spherical harmonic coefficients is $n = 22,801$.
Table 3 gives performance results in gigaflops per second for computing the gravity field parameters on 4 processors of an IBM eServer p5 (1 node of 8 processors Power5/1.9GHz) dedicated for gravity field calculations at CNES. The computation can be performed in about 4 hours and this satisfies the operational constraints of the users.

Table 3: Performance for gravity field computation on 4 processors (IBM Power5).

	Performance (Gflops)
peak DGEMM	6
Init. R	4.4
Update R	4.3
Total time	4 h 10 min

To validate the accuracy of the solution, the physicists usually plot a spectrum that represents the geoid height error between the computed model and a reference model. They also depict the resulting geoid map. The solution computed in our example complies with the requirements of the physicists.
We now propose a short error analysis of our numerical results. Following Remark 1, the condition number of A was estimated using the power and inverse power methods that respectively evaluate $\|R^T R\|$ and $\|(R^T R)^{-1}\|$ and then we obtained

$$K(A) = \left(\|R^T R\| \cdot \|(R^T R)^{-1}\|\right)^{1/2} = 5 \cdot 10^6.$$

To confirm the well-known accuracy of the QR approach, we may compute the relative forward error bound resulting from the backward stability of the Householder QR factorization method [16, p. 385]. This error bound is roughly proportional to $K(A)\left(1 + K(A)\frac{\|r\|_2}{\|A\|_2\|x\|_2}\right)u$, where u is the

[§]Centre National d'Etudes Spatiales, Toulouse, France unit roundoff and r the residual $b - Ax$ [7, p. 31]. Then, if x denotes the exact solution of the LLSP and $\tilde{x}$ is the solution computed with our QR solver, we get

$$\frac{\|x - \tilde{x}\|}{\|x\|} \leq 6 \cdot 10^{-10}.$$

Let now evaluate the influence of errors of measurement. The measurement noise given by the physicists is of order $10^{-9} m/s^2$, so we perturbed the right-hand side b by adding to each component b_i a constant equal to $mod(i, 10) \cdot \|b\| \cdot 10^{-8}$. The relative error with the previously computed solution is equal to $1.2 \cdot 10^{-6}$, showing then that, for our given data, the error of measurement may be significant compared with rounding errors. The next mission GOCE will supply much more accurate measures (the measurement noise will be of order $10^{-12} m/s^2$). In that case, the errors due to finite-precision calculations may become more significant and then justify if needed the use of a QR approach for gravity field calculations.

6 Conclusion

The parallel distributed solver proposed in this paper allows us to compute an accurate solution for incremental least squares. The performance analysis showed that there is a trade-off between performance and memory in the choice of some program parameters like the block size or the number of rows considered in the updating process. This QR updating functionality benefits from the packed distributed storage since the memory required for storing the R factor is near minimal (depending on the size of the block size). The good performance of the parallel code is due to the efficient implementation of the ScaLAPACK kernel routine PDGEQRF. The fact that our implementation is based on the top of ScaLAPACK and Level-3 PBLAS routines ensures the portability of our code on different parallel machines. Finally, this solver enables us to tackle the huge least squares problems encountered in the area of space geodesy. The good numerical and performance results that we obtained encourage us to perform similar experiments on GOCE simulation data.

Acknowledgments

We would like to thank the CINES (Centre Informatique National de l'Enseignement Supérieur) for allowing us to perform experimentation on its parallel computers. We also thank Georges Balmino and Jean-Charles Marty from CNES for their precious help in the geodesy application. We are grateful to Jack Dongarra (Innovative Computing Laboratory - University of Tennessee) for fruitful discussions during a visit of the first author to his group.

References

[1] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen, *LAPACK user's guide*, SIAM, 1999, Third edition.

[2] M. Baboulin, L. Giraud, and S. Gratton, *A parallel distributed solver for large dense symmetric systems: applications to geodesy and electromagnetism problems*, Int. J. of High Performance Computing Applications **19** (2005), no. 4, 353–363.

[3] M. Baboulin, L. Giraud, S. Gratton, and J. Langou, *A distributed packed storage for large dense parallel in-core calculations*, Technical Report TR/PA/05/30, CERFACS, Toulouse, France, 2005, To appear in Concurrency and Computation: Practice and Experience.

[4] G. Balmino, *The European GOCE Gravity Consortium (EGG-C)*, (April 2001), 7–12, Proceedings of the International GOCE User Workshop.

[5] G. Balmino, S. Bruinsma, and J-C. Marty, *Numerical simulation of the gravity field recovery from GOCE mission data*, (March 8-10, 2004), Proceedings of the Second International GOCE User Workshop “GOCE, The Geoid and Oceanography”, ESA-ESRIN, Frascati, Italy.

[6] G. Balmino, A. Cazenave, A. Comolet-Tirman, J. C. Husson, and M. Lefebvre, *Cours de géodésie dynamique et spatiale*, ENSTA, 1982.

[7] Å. Björck, *Numerical methods for least squares problems*, SIAM, 1996.

[8] L. Blackford, J. Choi, A. Cleary, E. D'Azevedo, J. Demmel, I. Dhillon, J. Dongarra, S. Hammarling, G. Henry, A. Petitet, K. Stanley, D. Walker, and R. Whaley, *ScaLAPACK user's guide*, SIAM, 1997.

[9] J. Choi, J. Dongarra, L. Ostrouchov, A. Petitet, D. Walker, and R. Whaley, *A proposal for a set of parallel basic linear algebra subprograms*, Tech. report, 1995, LAPACK Working Note 100.

[10] ______, *The design and implementation of the ScaLAPACK LU, QR, and Cholesky factorization routines*, Scientific Programming **5** (1996), 173–184.

[11] J. Demmel and J. Dongarra, *LAPACK 2005 prospectus: Reliable and scalable software for linear algebra computations on high end computers*, (February 2005), LAPACK Working Note 164.

[12] G. H. Golub, P. Manneback, and Ph. L. Toint, *A comparison between some direct and iterative methods for certain large scale geodetic least squares problems*, SIAM J. Scientific Computing **7** (1986), no. 3, 799–816.

[13] G. H. Golub and C. F. van Loan, *Matrix computations*, The Johns Hopkins University Press, 1996, Third edition.

[14] B. Gunter, W. Reiley, and R. van de Geijn, *Parallel out-of-core Cholesky and QR factorizations with POOCLAPACK*, IEEE Computer Society (2001), Proceedings of the 15th International Parallel and Distributed Processing Symposium (IPDPS).

[15] B. Gunter and R. van de Geijn, *Parallel out-of-core computation and updating of the QR factorization*, ACM Trans. Math. Softw. **31** (2005), no. 1, 60–78.

[16] N. J. Higham, *Accuracy and stability of numerical algorithms*, SIAM, 2002, Second edition.

[17] J. E. Dennis Jr. and R. B. Schnabel, *Numerical methods for unconstrained optimization and nonlinear equations*, SIAM, 1996.

[18] T. Kailath, A .H. Sayed, and B. Hassibi, *Linear estimation*, Prentice-Hall, 2000.

[19] H. Sünkel, *From Eötvös to milligal+, Final Report*, ESA/ESTEC Contract No. 13392/98/NL/GD, Graz University of Technology, 2000.

[20] R. van de Geijn, *Using PLAPACK*, The MIT Press, 1997.