QR Factorization of Tall and Skinny Matrices in a Grid Computing Environment
Emmanuel Agullo, Camille Coti, Jack Dongarra, Thomas Herault, Julien Langou

To cite this version:
Emmanuel Agullo, Camille Coti, Jack Dongarra, Thomas Herault, Julien Langou. QR Factorization of Tall and Skinny Matrices in a Grid Computing Environment. 24th IEEE International Parallel and Distributed Processing Symposium (IPDPS 2010), Apr 2010, Atlanta, United States. inria-00548900

HAL Id: inria-00548900
https://inria.hal.science/inria-00548900
Submitted on 21 Dec 2010
QR Factorization of Tall and Skinny Matrices in a Grid Computing Environment

Emmanuel Agullo *, Camille Coti †, Jack Dongarra *,
Thomas Herault ‡, Julien Langou §

* Dpt of Electrical Engineering and Computer Science, University of Tennessee †INRIA Saclay-Île de France,
F-91893 Orsay, France ‡Univ Paris Sud, University of Tennessee, LRI, INRIA § Dpt of Mathematical and
Statistical Sciences, University of Colorado Denver
eagullo@eecs.utk.edu, coti@lri.fr, dongarra@eecs.utk.edu,
herault@lri.fr, julien.langou@ucdenver.edu

Abstract—Previous studies have reported that common dense
linear algebra operations do not achieve speed up by using
multiple geographical sites of a computational grid. Because such
operations are the building blocks of most scientific applications,
conventional supercomputers are still strongly predominant in
high-performance computing and the use of grids for speeding up
large-scale scientific problems is limited to applications exhibiting
parallelism at a higher level. We have identified two performance
bottlenecks in the distributed memory algorithms implemented in
ScalAPACK, a state-of-the-art dense linear algebra library. First,
because ScalAPACK assumes a homogeneous communication
network, the implementations of ScalAPACK algorithms lack
locality in their communication pattern. Second, the number
of messages sent in the ScalAPACK algorithms is significantly
greater than other algorithms that trade flops for communication.
In this paper, we present a new approach for computing a QR
factorization – one of the main dense linear algebra kernels – of
tall and skinny matrices in a grid computing environment that
overcomes these two bottlenecks. Our contribution is to articulate
a recently proposed algorithm (Communication Avoiding QR)
with a topology-aware middleware (QCG-OMPI) in order to
confine intensive communications (ScalAPACK calls) within the
different geographical sites. An experimental study conducted on the Grid’5000 platform shows that the resulting performance
increases linearly with the number of geographical sites on large-
scale problems (and is in particular consistently higher than
ScalAPACK’s).

I. INTRODUCTION

Grid computing \cite{18} as a utility has reached the mainstream.
Many large-scale scientific problems have been successfully
solved thanks to the use of computational grids (or, simply, grids).
These problems cover a wide range of scientific disciplines including biology (protein folding \cite{28}), medicine
(cure muscular dystrophy \cite{9}), financial modeling, earthquake
simulation, and climate/weather modeling. Such scientific
breakthroughs have relied on the tremendous processing power
provided by grid infrastructures. For example, the Berkeley Open
Infrastructure for Network Computing (BOINC) \cite{3} gathers the
processing power of personal computers provided by people
volunteering all over the world. This processing power is then made available to researchers through different
projects such as Climateprediction.net \cite{2}, Rosetta@home \cite{14}
and World Community Grid (WCG)\cite{1}. As of September 2009,
18, BOINC had 566,000 active computers (hosts) worldwide
for an average total processing power of 2.4 Pflop/s\cite{2}. Furthermore, following the supercomputing trends, grid computing
infrastructures have successfully exploited the emerging hard-
ware technologies. The Folding@Home project \cite{7} – which
aims at understanding protein folding, misfolding, and related
diseases – achieves 7.9 Pflop/s thanks to grid exploiting spes-
cialized hardware such as graphics processing units (GPUs),
multicore chips and IBM Cell processors.

However, conventional supercomputers are strongly pre-
dominant in high-performance computing (HPC) because dif-
ferent limiting factors prevent the use of grids for solving
large-scale scientific problems. First of all, security require-
ments for grids are not completely met in spite of the
important efforts in that direction \cite{25}. Second, contrary to
their original purpose (the term grid itself is a metaphor
for making computer power as easy to access as an electric
power grid \cite{18}), grids have not been historically very user-
friendly. Third, not all the grid infrastructures are dimensioned
for HPC, which is only one of the aims of grid computing.
Even recent commercial offerings such as Amazon Elastic
Compute Cloud (EC2)\cite{3} are not considered mature yet for HPC
because of under-calibrated components \cite{38}. Furthermore,
other aspects are still the focus of intensive research, such
as service discovery \cite{11}, scheduling \cite{4}, etc.

But, above all, the major limiting factor to a wider usage of
grids by computational scientists to solve large-scale problems
is the fact that common dense linear algebra operations do not
achieve performance speed up by using multiple geographical
sites of a computational grid, as reported in previous
studies \cite{30}, \cite{31}. Because those operations are the building
blocks of most scientific applications, the immense processing
power delivered by grids vanishes. Unless the application
presents parallelism at a higher level (most of the applica-
tions running on BOINC are actually embarrassingly parallel,
i.e., loosely coupled), its performance becomes limited by the
processing power of a single geographical site of the

\footnote{This work was partly supported by the EC FP7 (grant # FP6-2005-IST-5
033883), and NSF-CCF (grant #881520).}
grid infrastructure, ruining the ambition to compete against conventional supercomputers. In this paper, we present a new approach for factorizing a dense matrix – one of the most important operations in dense linear algebra – in a grid computing environment that overcomes these two bottlenecks. Our approach consists of articulating a recently proposed algorithm (Communication Avoiding algorithm [15]) with a topology-aware middleware (QCG-OMPI[13]) in order to confine intensive communications (ScaLAPACK calls) within the different geographical sites.

In this study, we focus on the QR factorization [21] of a tall and skinny (TS) dense matrix into an orthogonal matrix $Q$ and an upper triangular matrix $R$ and we discuss how our approach generalizes to all one-sided factorizations (QR, LU and Cholesky) of a general dense matrix (Section IV). Furthermore, we focus on the computation of the triangular factor $R$ and do not explicitly form the orthogonal matrix $Q$. However, we show that the performance behavior would be similar if we compute $Q$ or not.

The paper is organized as follows. We present the related work and define the scope of our paper in Section II. In Section III, we present the implementation of a QR factorization of TS matrices that confines intensive communications within the different geographical sites. Section IV discusses a performance model that allows us to understand the basic trends observed in our experimental study (Section V). We conclude and present the future work in Section VI.

II. BACKGROUND

We present here the related work. We first describe previous experimental studies of the behavior of dense linear algebra operations in a grid computing environment (Section II-A). We then succinctly present the operation we focus on in this paper, the QR factorization, as it is implemented in ScaLAPACK, a state-of-the-art dense linear algebra library for distributed memory machines (Section II-B). We continue with the introduction of a recently proposed algorithm trading flops for communication (Section II-C). To take advantage in a grid computing environment of the limited amount of communication induced by such an algorithm, we need to articulate it with the topology of the grid. We present in Section II-D a middleware enabling this articulation by (i) retrieving the system topology to the application and even (ii) allowing the application to reserve suitable resources. Such an articulation of the algorithms with the topology is critical in an environment built on top of heterogeneous networks such as a grid. We conclude this review by discussing the scope of this paper (Section II-E).

A. Dense linear algebra on the grid

The idea of performing dense linear algebra operations on the grid is not new; however, success stories are rare in the related bibliography. Libraries that have an MPI [17] interface for handling the communication layer, such as ScaLAPACK or HP Linpack, can be run on a grid by linking them to a grid-enabled implementation of the MPI standard such as MPICH-G2 [26], PACX-MPI [20] or GridMPI4. MPI has become the de facto language for programming parallel applications on distributed memory architectures such as clusters. Programmers have gained experience using this programming paradigm throughout the past decade; scientific libraries have been developed and optimized using MPI. As a consequence, it is natural to consider it as a first-choice candidate for programming parallel applications on the grid in order to benefit from this experience and to be able to port existing applications for the grid. The GrADS5 project had the purpose of simplifying distributed, heterogeneous computing and making grid application development as well as performance tuning for real applications an everyday practice. Among other accomplishments, large matrices could be factorized thanks to the use of a grid whereas it was impossible to process them on a single cluster because of memory constraints [31], [37]. The resource allocation (number of clusters, etc.) was automatically chosen in order to maximize the performance. However, for matrices that could fit in the (distributed) memory of the nodes of a cluster, the experiments (conducted with ScaLAPACK) showed that the use of a single cluster was optimal [31]. In other words, using multiple geographical sites led to a slow down of the factorization. Indeed, the overhead due to the high cost of inter-cluster communications was not balanced by the benefits of a higher processing power.

For the same reason, the EC2 cloud has recently been shown to be inadequate for dense linear algebra [30]. In this latter study, the authors address the question whether cloud computing can reach the Top5006, i.e., the ranked list of the fastest computing systems in the world. Based on experiments conducted with the parallel LU factorization [21] implemented in the HP Linpack Benchmark [16], not only did they observe a slow down when using multiple clusters, but they also showed that the financial cost (in dollars) of performance (number of floating-point operations per second, in Gflop/s) increases exponentially with the number of computing cores used, much in contrast to existing scalable HPC systems such as supercomputers.

The HeteroScaLAPACK project7 aims at developing a parallel dense linear algebra package for heterogeneous architectures on top of ScaLAPACK. This approach is orthogonal (and complementary) to ours since it focuses on the heterogeneity of the processors [34], whereas we currently aim at mapping the implementation of the algorithm to the heterogeneity of the network (topology) through QCG-OMPI. In our present work, we do not consider the heterogeneity of the processors. Another fundamental difference with HeteroScaLAPACK is that we are using TSQR, an algorithm that is not available in ScaLAPACK.

B. ScaLAPACK’s QR factorization

The QR factorization of an $M \times N$ real matrix $A$ has the form $A = QR$, where $Q$ is an $M \times M$ real orthogonal matrix.

4http://www.gridmpiro.org
5Software Support for High-Level Grid Application Development http://www.hipersoft.rice.edu/grads/
6http://www.top500.org
7http://hcl.ucd.ie/project/HeteroScaLAPACK
and R is an $M \times N$ real upper triangular matrix. Provided $A$ is nonsingular, this factorization is essentially unique, that is, it is unique if we impose the diagonal entries of $R$ to be positive. There is a variety of algorithms to obtain a QR factorization from a given matrix, the most well-known arguably being the Gram-Schmidt algorithm. Dense linear algebra libraries have been traditionally focusing on algorithms based on unitary transformations (Givens rotations or Householder reflections) because they are unconditionally backward stable [21]. Givens rotations are advantageous when zeroing out a few elements of a matrix whereas Householder transformations are advantageous when zeroing out a vector of a matrix. Therefore, for dense matrices, we consider the QR factorization algorithm based on Householder reflections. The algorithm consists of applying successive elementary Householder transformations of the form $H = I - \tau vv^T$ where $I$ is the identity matrix, $v$ is a column reflector and $\tau$ is a scaling factor [21].

To achieve high performance on modern computers with different levels of cache, the application of the Householder reflections is blocked [36]. In ScalAPACK [8], b elementary Householder matrices are accumulated within a panel (a block-column) $V$ consisting of $b$ reflectors. The consecutive applications of these $b$ reflectors $(H_1H_2...H_b)$ is then constructed all at once using the matrix equality $H_1H_2...H_b = I - VTV^T$ ($T$ is a $b \times b$ upper triangular matrix). However, this blocking incurs an additional computational overhead. The overhead is negligible when there is a large number of columns to be updated but is significant when there are only a few columns to be updated. Default values in the ScalAPACK PDGEQRF subroutine are NB=64 and NX=128, where NB is the block size, $b$, and NX is the cross-over point; blocking is not to be used if there is less than NX columns are to be updated. PDGEQRF uses PDGEQR2 to perform the panel factorizations. Due to the panel factorization, the algorithm in ScalAPACK requires one allreduce operation for each column. In other words, ScalAPACK uses at least $N \log_2(P)$ messages to factor an $M$-by-$N$ matrix.

C. Communication Avoiding QR (CAQR) factorization

In this paper we propose an implementation of the so-called “Communication Avoiding QR” (CAQR) algorithm originally proposed by Demmel et al. [15]. CAQR belongs to the class of the (factor panel) / (update trailing matrix) algorithms. For all algorithms in this class, the update phase is entirely dictated by the panel factorization step and is easily parallelizable. Therefore, we only discuss the panel factorization step. The panel factorization in CAQR is based on the “Tall and Skinny QR” factorization algorithm (TSQR) [15]. In contrast to the ScalAPACK panel factorization algorithm (subroutine PDGEQRF), which requires one allreduce operation per column, TSQR requires one allreduce operation per $b$ columns where $b$ is an arbitrary block size. The number of communications is therefore divided by $b$. The volume of communication stays the same. The number of operations on the critical path is increased in TSQR by an additional $O(\log_2(P)N^3)$ term. TSQR effectively trades communication for flops.

TSQR is a single complex allreduce operation [15]. The TS matrix is split in $P$ block-rows, called domains; the factorization of a domain is the operation performed on the leaves of the binary tree associated to the reduction. The binary operation then used in this allreduce operation is as follows: from two input triangular matrices $R_1$ and $R_2$, stack $R_1$ on top of $R_2$ to form $[R_1; R_2]$, perform the QR factorization of $[R_1; R_2]$, the output $R$ is given by the R-factor of $[R_1; R_2]$. One can show that this binary operation is associative. It is also commutative if one imposes each computed R-factor to have nonnegative diagonal entries. As for any reduce operation, the shape of the optimal tree depends on the dimension of the data and the underlying hardware. CAQR with a binary tree has been studied in the parallel distributed context [15] and CAQR with a flat tree has been implemented in the context of out-of-core QR factorization [23]. CAQR with a flat tree delivers wide parallelism and, for this reason, has been used in the multicore context [10], [27], [33].

Previous implementations of CAQR have used either a flat tree or a binary tree. One key originality of our present work lies in the fact that our reduction tree is neither binary nor flat. It is tuned for the targeted computational grid, as illustrated in Fig. 1(b). First we reduce with a binary tree on each cluster. Then we reduce with a second binary tree the result of each cluster at the grid level. The binary tree used by ScalAPACK PDGEQR2 (Fig. 1(a)) minimizes the sum of the inter-cluster messages and the intra-cluster messages. Our tree is designed to minimize the total number of inter-cluster messages.

We now give a brief history of related algorithmic work in contrast to the reference work of Demmel et al. [15]. The parallelization of the Givens rotations based and Householder reflections based QR factorization algorithms is a well-studied area in Numerical Linear Algebra. The development of the algorithms has followed architectural trends. In the late 1970s / early 1980s [24], [29], [35], the research was focusing on algorithms based on Givens rotations. The focus was on extracting as much parallelism as possible. We can interpret these sequences of algorithms as scalar implementations using a flat tree of the algorithm in Demmel et al. [15]. In the late 1980s, the research shifted gears and presented algorithms based on Householder reflections [32], [12]. The motivation was to use vector computer capabilities. We can interpret all these algorithms as vector implementations using a flat tree and/or a binary tree of the algorithm in Demmel et al. [15]. All these algorithms require a number of messages greater than $n$, the number of columns of the initial matrix $A$, as in ScalAPACK. The algorithm in Demmel et al. [15] is a generalization with multiple blocks of columns with a nontrivial reduction operation, which enables one to divide the number of messages of these previous algorithms by the block size, $b$. Demmel et al. proved that TSQR and CAQR algorithms induce a minimum amount of communication (under certain conditions, see Section 17 of [15] for more details) and are numerically as stable as the Householder QR factorization.

D. Topology-aware MPI middleware for the grid: QCG-OMPI

Programming efficient applications for grids built by federating clusters is challenging, mostly because of the difference of performance between the various networks the application...
has to use. As seen in the table of Figure 2(a) we can observe two orders of magnitude between inter and intra-cluster latency on a dedicated, nation-wide network, and the difference can reach three or four orders of magnitude on an international, shared network such as the Internet. As a consequence, the application must be adapted to the intrinsically hierarchical topology of the grid. In other words, the communication and computation patterns of the application must match the physical topology of the hardware resources it is executed on.

ScaLAPACK, and many of the linear algebra libraries for scientific computing, are programmed in MPI. MPI is fit for homogeneous supercomputers: processes are mostly indistinguishable one from another, and the standard does not specify anything about process / node placement.

As a consequence, to efficiently program a parallel application on top of a non-uniform network, typically on top of a hierarchical network like a grid, MPI must be extended to help programmers adapt the communications of the application to the machine. MPICH-G2 [26] introduced the concept of colors to describe the available topology to the application at runtime. Colors can be used directly by MPI routines in order to build topology-aware communicators (the abstraction in MPI that is used to group processors together). However, the application is fully responsible to adapt itself to the topology that is discovered at runtime.

The QosCosGrid system offers a resource-aware grid meta-scheduler that gives the possibility to allocate resources that match requirements expressed in a companion file called the application’s JobProfile that describe the future communications of the application [6]. The JobProfile defines process groups and requirements on the hardware specifications of the resources that have to be allocated for these processes such as amount of memory, CPU speed, and network properties between groups of processes, such as latency and bandwidth. As a consequence, the application will always be executed on an appropriate resource topology.

The QosCosGrid system features QCG-OMPI, an MPI implementation based on OpenMPI [19] and targeted to computational grids. Besides grid-specific communication features that enable communicating throughout the grid described in [13], QCG-OMPI has the possibility to retrieve topology information provided to the scheduler in the JobProfile at runtime. We explain in Section III how we have implemented and articulated TSQR with QCG-OMPI in order to take advantage of the topology.

E. Scope

The QR factorization of TS matrices is directly used as a kernel in several important applications of linear algebra. For instance, block-iterative methods need to regularly perform this operation in order to obtain an orthogonal basis for a set of vectors. TSQR can also be used to perform the panel factorization of an algorithm handling general matrices (CAQR). This present study can be viewed as a first step towards the factorization of general matrices on the grid.

Grids aggregate computing power from any kind of resource. However, in some typical grid projects, such as Superlink@Technion, the Lattice project, EdGES, and the Condor pool at Univ. of Wisconsin-Madison, a significant part of the power comes from a few institutions featuring environments with a cluster-like setup. In this first work, we focus our study on clusters of clusters, to enable evaluation in a stable and reproducible environment. Porting the work to a general desktop grid remains a future work.

Finally, we emphasize that the objective of this paper is to show that we can achieve a performance speed up over the
grid with common dense linear algebra operations. To illustrate our claim, we compare our approach against a state-of-the-art library for distributed memory architectures, ScaLAPACK. In order to highlight the differences, we chose to base our approach on ScaLAPACK (see Section III).

III. QCG-TSQR: ARTICULATION OF TSQR WITH QCG-OMPI

We explain in this section how we articulate the TSQR algorithm with QCG-OMPI in order to confine intensive communications within the different geographical sites of the computational grid. The first difference from the TSQR algorithm as presented in Section II-C is that a domain is processed by a call to ScaLAPACK (but not LAPACK as in [15]). By doing so, we may attribute a domain to a group of processes (instead of a single process) jointly performing its factorization. The particular case of one domain per process corresponds to the original TSQR (calls to LAPACK). At the other end of the spectrum, we may associate one domain per geographical site of the computational grid. The choice of the number of domains impacts performance, as we will illustrate in Section V-C. In all cases, we call our algorithm TSQR (or QCG-TSQR), since it is a single reduce operation based on a binary tree, similarly to the algorithm presented in Section II-C.

As explained in Section II-D, the first task of developing a QCG-OMPI application consists of defining the kind of topologies expected by the application in a JobProfile. To get enough flexibility, we request that processes are split into groups of equivalent computing power, with good network connectivity inside each group (low latency, high bandwidth) and we accept a lower network connectivity between the groups. This corresponds to the classical cluster of clusters approach, with a constraint on the relative size of the clusters to facilitate load balancing.

The meta-scheduler will allocate resources in the physical grid that matches these requirements. To enable us to complete an exhaustive study on the different kind of topologies we can get, we also introduced more constraints in the reservation mechanism, depending on the experiment we ran. For each experiment, the set of machines that are allocated to the job are passed to the MPI middleware, which exposes those groups using two-dimensional arrays of group identifiers (the group identifiers are defined in the JobProfile by the developer). After the initialization, the application retrieves these group identifiers from the system (using a specific MPI attribute) and then creates one MPI communicator per group, using the MPI_Comm_split routine. Once this is done, the TSQR algorithm has knowledge of the topology that allows it to adapt to the physical setup of the grid.

The choice to introduce a requirement of similar computing power between the groups however introduces constraints on the reservation mechanism. For example, in some experiments discussed later (Section V), only half the cores of some of the machines were allocated in order to fit this requirement. Another possibility would have been to handle load balancing issues at the algorithmic level (and not at the middleware level) in order to relieve this constraint on the JobProfile and thus increase the number of physical setups that would match our needs. In the particular case of TSQR, this is a natural extension; we would only have to adapt the number of rows attributed to each domain as a function of the processing power dedicated to a domain. This alternative approach is future work.

IV. PERFORMANCE MODEL

In Tables I and II, we give the amount of communication and computation required for ScaLAPACK QR2 and TSQR in two different scenarios: first, when only the R-factor is requested (Table I) and, second, when both the R-factor and the Q-factor are requested (Table II). In this model, we assume that a binary tree is used for the reductions and a homogeneous network. We recall that the input matrix $A$ is $M$–by–$N$ and that $P$ is the number of domains. The number of FLOPS is the number of FLOPS on the critical path per domain.

Assuming a homogeneous network, (the modelization of heterogeneous network in the same manner is straightforward,) the total time of the factorization is then approximated by the formula: $time = \beta \ast (# msg) + (vol. data exchanged) + \gamma \ast (# FLOPs)$, where $\alpha$ is the inverse of the bandwidth, $\beta$ the latency, and $\gamma$ the inverse of the floating point rate of a domain. Although this model is simplistic, it enables us to forecast the basic trends. Note that in the case of TS matrices, we have $M \gg N$. First we observe that the time to compute both the $Q$ and the $R$ factors is exactly twice the time for computing $R$ only. Moreover, further theoretical and experimental analysis of the algorithm (see [15]) reveal that the structure of the computation is the same in both cases and the time to obtain $Q$ is twice the time.
to obtain $R$. This leads to Trend 1. For brevity, we mainly focus our study on the computation of $R$ only.

Trend 1: The time to compute both $Q$ and $R$ is about twice the time for computing $R$ only.

One of the building blocks of the ScaLAPACK PDGEQR2 implementation and of our TSQR algorithm is the domanial QR factorization of a TS matrix. The domain can be processed by a core, a node or a group of nodes. We can not expect performance from our parallel distributed algorithms to be better than the one of its domanial kernels. This leads to Trend 2. In practice, the performance of the QR factorization of TS matrices obtained from LAPACK/ScaLAPACK on a domain (core, node, small number of nodes) is a small fraction of the peak. ($\gamma$ is likely to be small.)

Trend 2: The performance of the factorization of TS matrices is limited by the domanial performance of the QR factorization of TS matrices.

We see that the number of operations is proportional to $M$ while all the communication terms (latency and bandwidth) are independent of $M$. Therefore when $M$ increases, the communication time stays constant whereas the domanial computation time increases.

Trend 3: The performance of the factorization of TS matrices increases with $M$ until the parallel system reaches its computational peak, and the performance climbs to a plateau where it does not progress anymore.

The number of operations is proportional to $N^2$ while the number of messages is proportional to $N$. Therefore when $N$ increases, the latency term is hidden by the computation term. This leads to better performance. We also note that increasing $N$ enables better performance of the domanial kernel since it can use Level 3 BLAS when the number of columns is greater than, perhaps, 100. This is Trend 4.

Trend 4: The performance of the factorization of TS matrices increases with $N$.

Finally, we see that the latency term is $2 \log_2(P)$ for TSQR while it is $2N \log_2(P)$ for ScaLAPACK QR2. On the other hand, the FLOPs term has a non parallelizable additional $2/3 \log_2(P)N^3$ term for the TSQR algorithm. We see that TSQR effectively trades messages for flops. We expect TSQR to be faster than ScaLAPACK QR2 for $N$ in the mid-range (perhaps between five and a few hundreds). For larger $N$, TSQR will become slower because of the additional flops. This is Trend 5. (We note that for large $N$, one should stop using TSQR and switch to CAQR.)

Trend 5: The performance of TSQR is better than ScaLAPACK for $N$ in the mid range. When $N$ gets too large, the performance of TSQR deteriorates and ScaLAPACK becomes better.
4.0.3 (flag -O3) and run in dedicated mode (no other user can access the machines). ScaLAPACK 1.8.0 and GotoBLAS 1.26 libraries were used. Finally we recall that we focus on the factorization of TS dense large-scale matrices in real double precision, corresponding to up to 16 GB of memory (e.g. a 33,554,432 × 64 matrix in double precision).

We use two processes per node together with the serial version of GotoBLAS’s DGEMM in all the experiments reported in this study. With DGEMM being the fastest kernel (on top of which other BLAS operations are usually built), we obtain a rough practical performance upper bound for our computational grid of about 940 Gflop/s (the ideal case where 256 processors would achieve the performance of DGEMM, i.e., about 3.67 Gflop/s each) out of the 2,048 Gflop/s theoretical peak.

B. ScaLAPACK performance

SCALAPACK implements block-partitioned algorithms. Its performance depends on the partitioning of the matrix into blocks. Preliminary experiments (not reported here) showed that a column-wise 1D-cyclic partition is optimum for processing TS matrices in our environment. We furthermore tuned empirically the block size and settled on $b = 64$.

Figure 3 reports ScaLAPACK performance. In accordance with Property 2, the overall performance of the QR factorization of TS matrices is low (consistently lower than 90 Gflop/s) compared to the practical upper bound of our grid (940 Gflop/s). Even on a single cluster, this ratio is low since the performance at one site is consistently lower than 70 Gflop/s out of a practical upper bound of 235 Gflop/s. As expected too (properties 3 and 4), the performance increases with the dimensions of the matrix. For matrices of small to moderate height ($M \leq 5,000,000$), the fastest execution is consistently the one conducted on a single site. In other words, for those matrices, the use of a grid (two or four sites) induces a drop in performance, confirming previous studies [31], [30], [38]. For very tall matrices ($M > 5,000,000$), the proportion of computation relative to the amount of communication becomes high enough so that the use of multiple sites eventually speeds up the performance (right-most part of the graphs and Property 3). This speed up however hardly surpasses a value of 2.0 while using four sites (Figure 3(b)).

C. QCG-TSQR performance

The performance of TSQR (articulated with QCG-OMPI as described in Section III) depends on the number of domains used. In Figure 4, we report the TSQR performance for the optimum number of domains and we will return later to the effect of the number of domains. In accordance with Property 2, the overall performance is again only a fraction of the practical upper bound of our grid (940 Gflop/s). But, compared to ScaLAPACK, this ratio is significantly higher since the factorization of a 8,388,608 × 512 matrix achieves 256 Gflop/s (Figure 4(d)). Again, in accordance with properties 3 and 4, the overall performance increases with the dimensions of the matrix. Thanks to its better performance (Property 5), TSQR also achieves a speed up on the grid on matrices of moderate size. Indeed, for almost all matrices of moderate to great height ($M \geq 500,000$), the fastest execution is the one conducted on all four sites. Furthermore, for very tall matrices ($M \geq 5,000,000$), TSQR performance scales almost linearly with the number of sites (a speed up of almost 4.0 is obtained.

<table>
<thead>
<tr>
<th>ScalAPACK QR2</th>
<th>TSQR</th>
<th># msg</th>
<th>vol. data exchanged</th>
<th># FLOPs</th>
</tr>
</thead>
<tbody>
<tr>
<td>dN log₂(P)</td>
<td>$2 log₂(P)</td>
<td>(4MN² - 4/3N³)/P</td>
<td>$+ 4/3 log₂(P)N^3$</td>
<td></td>
</tr>
<tr>
<td>$2 log₂(P)(N^2/2)$</td>
<td>$2 log₂(P)(N^2/2)$</td>
<td>$(4MN² - 4/3N³)/P$</td>
<td>$+ 4/3 log₂(P)N^3$</td>
<td></td>
</tr>
</tbody>
</table>

TABLE II

COMMUNICATION AND COMPUTATION BREAKDOWN WHEN BOTH THE Q-FACTOR AND THE R-FACTOR ARE NEEDED.
on four sites). This result is the central statement of this paper and validates the thesis that computational grids are a valid infrastructure for solving large-scale problems relying on the QR factorization of TS matrices.

Figure 5 now illustrates the effect of the number of domains per cluster on TSQR performance. Globally, the performance increases with the number of domains. For very tall matrices ($M = 33,554,432$), the impact is limited (but not negligible) since there is enough computation to almost mask the effect of communications (Property 3). For very skinny matrices ($N = 64$), the optimum number of domains for executing TSQR on a single cluster is 64 (Figure 5(c)), corresponding to a configuration with one domain per processor. This optimum selection of the number of domains is translated to executions on multiple clusters where 64 domains per cluster is optimum too (Figure 5(a)). For the widest matrices studied ($N = 512$), the optimum number of domains for executing TSQR on a single cluster is 32 (Figure 5(d)), corresponding to a configuration with one domain per node. For those matrices, trading flops for intra-node communications is not worthwhile. This behavior is again transposable to executions on multiple sites (Figure 5(b)) where the optimum configuration also corresponds to 32 domains per cluster. This observation illustrates the fact that one should use CAQR and not TSQR for large $N$.

D. QCG-TSQR vs ScaLAPACK

Figure 6 compares TSQR performance (still articulated with QCG-OMPI) against ScaLAPACK’s. We report the maximum performance out of executions on one, two or four sites. For instance, the graph of TSQR in Figure 6(a) is thus the convex hull of the three graphs from Figure 4(a). In accordance with Property 5, TSQR consistently achieves a higher performance than ScaLAPACK. For matrices of limited height ($M = 131,072$), TSQR is optimum when executed on one site (Figure 4(a)). In this case, its superiority over ScaLAPACK comes from better performance within a cluster (Figure 5(c)). For matrices with a larger number of rows ($M = 4,194,304$), the impact of the number of domains per cluster is less sensitive (Figure 5(c) and Property 3). On the other hand, the matrix is large enough to allow a speedup of TSQR over the grid (Figure 4(a) and Property 3 (again)) but not of ScaLAPACK (Figure 3(a) and Property 5), hence the superiority of TSQR over ScaLAPACK for that type of matrix. For very tall matrices ($M = 33,554,432$), the impact of the number of domains per cluster becomes negligible (Figure 5(c) and Property 3). But (i) TSQR achieves a speedup of almost 4.0 on four sites (Figure 4(a)) whereas (ii) ScaLAPACK does not achieve yet such an ideal speedup (Figure 3(a)). Finally, on all the range of matrix shapes considered, and for different reasons, we have seen that TSQR consistently achieves a significantly higher performance than ScaLAPACK. For not so tall and not so skinny matrices (left-most part of Figure 6(d)), the gap between the performance of TSQR and ScaLAPACK reduces (Property 5).

One may have observed that the time spent in intra-node, then intra-cluster and finally inter-cluster communications becomes negligible while the dimensions of the matrices increase. For larger matrices (which would not hold in the memory of our machines), we may thus even expect that communications over the grid for ScaLAPACK would become negligible and thus that TSQR and ScaLAPACK would eventually achieve a similar (scalable) performance (Property 5).

VI. CONCLUSION AND PERSPECTIVES

This paper has revisited the performance behavior of common dense linear algebra operations in a grid computing environment. Contrary to past studies, we have shown that they can achieve a performance speedup by using multiple geographical sites of a computational grid. To do so, we have articulated a recently proposed algorithm (CAQR) with a topology-aware middleware (QCG-OMPI) in order to confine intensive communications (ScaLAPACK calls) within the different geographical sites. Our experimental study, conducted on the experimental Grid’5000 platform, focused on a particular operation, the QR factorization of TS matrices. We showed that its performance increases linearly with the
number of geographical sites on large-scale problems (and is in particular consistently higher than ScALAPACK’s).

We have proved theoretically through our models and experimentally that TSQR is a scalable algorithm on the grid. TSQR is an important algorithm in itself since, given a set of vectors, TSQR is a stable way to generate an orthogonal basis for it. TSQR will come handy as an orthogonalization scheme for sparse iterative methods (eigensolvers or linear solves). TSQR is also the panel factorization of CAQR. A natural question is whether CAQR scales as well on the grid. From models, there is no doubt that CAQR scales. However we will need to perform the experiment to confirm this claim. We note that the work and conclusion we have reached here for TSQR/CAQR can be (trivially) extended to TSLU/CALU ([22]) and Cholesky factorization [5].

Our approach is based on ScALAPACK. However, recent algorithms that better fit emerging architectures would have certainly improved the performance obtained on each cluster and in fine the global performance. For instance, recursive factorizations have been shown to achieve a higher performance on distributed memory machines [15]. Other codes benefit from multicore architectures [1].

If, as discussed in the introduction, the barriers for computational grids to compete against supercomputers are multiple, this study shows that the performance of large-scale dense linear algebra applications can scale with the number of geographical sites. We plan to extend this work to the QR factorization of general matrices and then to other one-sided factorizations (Cholesky, LU). Load balancing to take into account heterogeneity of clusters is another direction to investigate. The use of recursive algorithms to achieve higher performance is to be studied too.

THANKS. The authors thank Laura Grigori for her constructive suggestions.

REFERENCES


