Multilevel communication optimal LU and QR factorizations for hierarchical platforms

Grigori, Laura and Jacquelin, Mathias and Khabou, Amal

2013

MIMS EPrint: **2013.11**

Manchester Institute for Mathematical Sciences
School of Mathematics

The University of Manchester

Reports available from: http://eprints.maths.manchester.ac.uk/
And by contacting: The MIMS Secretary
School of Mathematics
The University of Manchester
Manchester, M13 9PL, UK

ISSN 1749-9097

Multilevel communication optimal LU and QR factorizations for hierarchical platforms

Laura Grigori*, Mathias Jacquelin†, and Amal Khabou‡

*INRIA Paris - Rocquencourt, Paris, France
†Lawrence Berkeley National Laboratory, Berkeley, USA
†The University of Manchester, Manchester, UK

laura.grigori@inria.fr, mjacquelin@lbl.gov, amal.khabou@manchester.ac.uk

***Abstract*—This study focuses on the performance of two classical dense linear algebra algorithms, the LU and the QR factorizations, on multilevel hierarchical platforms. We first introduce a performance model called Hierarchical Cluster Platform (HCP), encapsulating the characteristics of such platforms. The focus is set on reducing the communication requirements of studied algorithms at each level of the hierarchy. Lower bounds on communication are therefore extended with respect to the HCP model. We then introduce multilevel LU and QR algorithms tailored for those platforms, and provide a detailed performance analysis. We also provide a set of performance predictions showing the need for such algorithms on large platforms.**

Keywords: QR, LU, exascale, hierarchical platforms.

I. Introduction

Numerical algorithms and solvers play a crucial role in scientific computing. They lie at the heart of many applications and are often key to performance and scalability. Due to the ubiquity of multicore processors, solvers should be adapted to better exploit the hierarchical structure of modern architectures, where the tendency is towards multiple levels of parallelism. Thus with the increasing complexity of nodes, it is important to exploit these many levels of parallelism even within a single compute node. For that reason, classical algorithms need to be revisited so as to fit modern architectures that expose parallelism at different levels in the hierarchy. We believe that such an approach is mandatory in order to exploit upcoming hierarchical exascale computers at their full potential.

Studying the communication complexity of linear algebra operations and designing algorithms that are able to minimize communication is a topic that has received an important attention in the recent years. The most advanced approach in this context assumes one level of parallelism and takes into account the computation, the volume of communication, and the number of messages exchanged along the critical path of a parallel program. In this framework, the main previous theoretical result on communication complexity is a result derived by Hong and Kung in the 80's providing lower bounds on the volume of communication of dense matrix multiplication for sequential machines [1]. This result has been extended to parallel machines [2], to dense LU and QR factorizations (under certain assumptions) [3], and then to basically all direct methods in linear algebra [4]. Given an algorithm that performs a certain number of floating point operations, and considering the memory size, the lower bounds on communication are obtained by using the Loomis-Whitney inequality, as for example in [2, 4]. While theoretically important, these lower bounds are derived with respect to a performance model that supposes a memory hierarchy in the sequential case, and P processors without memory hierarchy in the parallel case. Such a model is not sufficient to encapsulate the features of modern hierarchical architectures.

On the practical side, several algorithms have been introduced recently [5, 6, 7, 8]. Most of them propose to use multiple reduction trees depending on the hierarchy. However, the focus is set on reducing the running time without explicitly taking communication into consideration. In [8], Dongarra et al. propose a generic algorithm implementing several optimizations regarding pipelining of computation, and allowing to select different elimination trees on platforms with two levels of parallelism. They provide insights on choosing the appropriate tree, a binary tree being for instance more suitable for a cluster with many cores, while a flat tree allows more locality and CPU efficiency. However, neither theoretical bounds nor cost analysis are provided in these studies.

Moreover, even if cache-oblivious algorithms are natural good candidates for reducing communication requirements at every level, they are not good candidates for large parallel implementations. We thus focus on cache-aware algorithms.

In the first part of this paper we introduce a performance model that we refer to as the Hierarchical Cluster Platform (HCP) model. Provided that two supercomputers might have different communication topologies and different compute nodes with different memory hierarchies, a detailed performance model tailored for one particular supercomputer is likely to not reflect the architecture of another supercomputer. Hence the goal of our performance model is to capture the main characteristics that influence the communication cost of peta- and exa- scale supercomputers which are based on multiple levels of parallelism and memory hierarchy. We use the proposed HCP model to extend the existing lower bounds on communication for direct linear algebra, to account for the hierarchical nature of present-day computers. We determine

the minimum amount of communication that is necessary at every level in the hierarchy, in terms of both number of messages and volume of communication.

Moreover, to the best of our knowledge, there is currently no algorithm targeting hierarchical platforms with more than two levels, nor any lower bound on communications for such platforms.

In the second part of the paper we introduce two multilevel algorithms for computing QR and LU factorizations (*ML-CAQR* and *ML-CALU*) that are able to minimize the communication at each level of the hierarchy, while performing a reasonable amount of extra computations. These recursive algorithms rely on their corresponding 1-level algorithms (resp. *CAQR* and *CALU*) as their base case. Indeed, *CAQR* and *CALU* are known to attain the communication lower bounds in terms of both bandwidth and latency with respect to the simpler one level performance model.

II. Background

A. The QR factorization

The QR factorization of an m-by-n matrix is a widely used algorithm, be it for orthogonalizing a set of vectors or for solving least squares problems with m equations and n unknowns. It is known to be an expensive, but very stable factorization. It is thus crucial to optimize its performance. The algorithm decomposes a m-by-n matrix A into two matrices Q and R such that $A = QR$, where Q is a m-by-m orthogonal matrix, while the m-by-n matrix R is upper triangular.

The QR factorization is obtained by applying a sequence of m-by-m unitary orthogonal transformations on the input matrix A.

An unitary transformation U_i introduces some zeros below the diagonal in the current updated matrix. The two basic transformations are Givens rotations and Householder reflections. A Givens rotation introduces a single zero while a Householder reflection zeroes out every element below the diagonal. Using Givens rotations, disjoint pairs of rows can be processed concurrently. Householder reflections, though not displaying the same parallelism, are less computationally expensive.

Tree based algorithms intent to benefit from both methods. Householder transformations are applied on local domains, or tiles, before getting eliminated two-by-two in a Givens-like approach. Communication Avoiding QR (*CAQR*) [3] belongs to this category, and organizes the computations so as to match the lower bounds on communication introduced in [4].

After $min(m, n)$ transformations, the resulting R factor is stored in place in the upper triangular part of matrix A while the matrix Q is assumed to be implicitly stored in the lower triangular part using the compact YTY^T representation for Householder reflections [9]. If needed, Q can be retrieved at the cost of extra computations by computing $Q = I - YTY^T$.

B. The LU factorization

LU factorization is another cornerstone of many scientific computations, and is the method of choice for solving most linear systems. It consists in decomposing a matrix A into a lower triangular matrix L and an upper triangular matrix U such that $A = LU$. However, in general pivoting is required to ensure numerical stability. The LU decomposition is rather applied to ΠA, where Π is a permutation matrix.

We note that many pivoting strategies could be used allowing either more efficiency or more numerical stability. The widely used Gaussian Elimination with Partial Pivoting (GEPP) introduces a bottleneck in terms of latency cost, $O(n \log P)$ synchronizations being needed overall, where P is the number of processors working on the panel and n is the number of columns of the input matrix. To reduce the amount of communication, a new parallel pivoting strategy has been introduced in the context of Communication Avoiding LU (*CALU*) [3], referred to as Tournament Pivoting. Each panel is first distributed over P processors, which perform GEPP on their local blocks to select b candidate pivot rows. These P sets of candidate rows are then reduced by using GEPP as the reduction operator. The final b rows are then moved to the first positions of the current panel. The different steps of the reduction are depicted in Figure 1, where we consider a panel W partitioned over 4 processors and a binary reduction tree.

The next step is to perform a LU factorization without pivoting on the pivoted panel. Finally, the block row of U is computed and the trailing matrix is updated. These two last steps of CALU are performed as in a classic LU factorization.

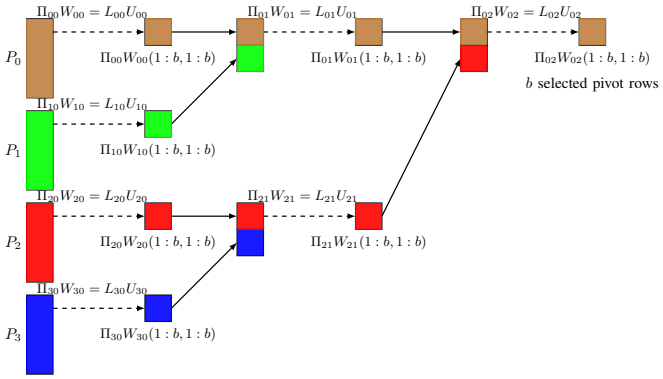


Fig. 1: Selection of b pivot rows for the panel factorization using Tournament Pivoting.

We note that Grigori et al. have shown in [10] that *CALU* is stable and Tournament Pivoting presents a good trade-off between parallelism and numerical stability. We use the same pivoting strategy in our multilevel *ML-CALU* algorithms (1D and 2D recursive variants). The main difference lies in the reduction operator used to select pivot rows.

III. Toward a realistic Hierarchical Cluster Platform model (HCP)

The focus of this study is set on hierarchical platforms running HPC applications and displaying increasingly deeper hierarchies. Such platforms are composed of two kinds of hierarchies: (1) a network hierarchy composed of interconnected network nodes, stacked on top of a (2) compute nodes

hierarchy [11]. This compute hierarchy can be composed for instance of shared memory NUMA multicore nodes.

Moreover, on most modern supercomputers, compute nodes are often grouped into *drawers* displaying higher local communication speeds. Such drawers typically belong to the network hierarchy, which is clearly not only a router hierarchy.

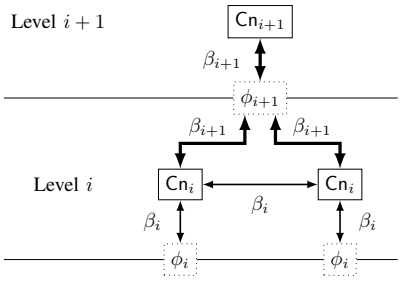


Fig. 2: Components of a level i in the HCP model.

The HCP model considers such platforms with l levels of parallelism, and uses the following assumptions. Level 1 is the deepest level in the hierarchy, where actual processing elements are located (for example cores). Each of these processing elements has its own local memory of size M_1.

A compute node of level $i+1$, denoted as Cn_{i+1} on Figure 2, is formed by P_i compute nodes of level i (two nodes in our example). The total number of processing elements of the entire platform is $P = \prod_{i=1}^{l} P_i$, while the total number of compute nodes of level i is $P_i^* = \prod_{j=i}^{l} P_j$. We let $M_i = M_1 \cdot \prod_{j=1}^{i-1} P_j$ be the aggregated memory size of a node of level $i > 1$.

The network latency α_i and the inverse bandwidth β_i apply throughout an entire level i. Moreover, we assume that generally, the higher in the hierarchy, the more expensive communication costs.

We also consider a message aggregation capacity ϕ_i at each level of the hierarchy, which determines the actual number of messages required to send a given amount of data.

We refer to the number of messages sent at level i as S_i, and to the exchanged volume of data as W_i. $\bar{S}_i = S_i \cdot \alpha_i$ is the associated latency cost, while $\bar{W}_i = W_i \cdot \beta_i$ is the bandwidth cost. These notations will be used throughout the rest of the paper.

For the sake of simplicity in both algorithm description and cost analysis, we assume the P_i compute nodes of level i to be virtually organized along a 2D grid topology, that is $P_i = P_{r_i} \times P_{c_i}$ (note that any topology could be mapped onto a 2D grid).

We note that the model makes abstraction of the detailed architecture of a compute node or the interconnection topology at a given level of the hierarchy. Hence such an approach has its own limitations, since the predicted performance might not be accurate. However, while keeping the model tractable, this model better reflects the actual nature of supercomputers than the one level model assumed so far, and helps to understand the communication bottlenecks of common linear algebra operations.

1) Communicating under the HCP *model:* We now describe how communication happens in the HCP model, and how messages flow through the hierarchy. We assume that if a compute node of level i communicates, all of its lower level nodes participate. We denote as ***counterparts*** of a compute node of level i all the nodes of level i lying in remote compute nodes of level $i+1$ and having the same local coordinates. We therefore have the relation $W_i = W_{i+1}/P_i$.

As an example, let us detail a communication of a volume of data W_i taking place between two nodes of level i. A total of P/P_i^* processing elements of level 1 are involved. Each has to send a chunk of data $W_1 = W_i P/P_i^*$. Since this amount of data has to fit in memory, we obviously have $\forall i, M_1 \geq W_1 = W_i P/P_i^*$. These blocks are transmitted to the level above in the hierarchy, i.e. to level 2. A compute node of level 2 has to send a volume of data $W_2 = P_1 W_1$. Since the aggregation capacity at level 2 is ϕ_2, this requires (W_2/ϕ_2) messages. The same holds for any level k such that $1 < k \leq i$, where data is forwarded by sending(W_k/ϕ_k) messages. We therefore have the following costs:

$$\bar{W}_k = \frac{W_i P_k^*}{P_i^*} \cdot \beta_k, \qquad \bar{S}_k = \frac{W_k}{\phi_k} \cdot \alpha_k = \frac{W_i P_k^*}{\phi_k P_i^*} \cdot \alpha_k.$$

This "regular" communication pattern is often encountered in HPC applications, the main target of the HCP model, and is simpler than a purely heterogeneous pattern (which could be encountered in grid environments for instance). Moreover, this organization allows to aggregate data at the algorithm level rather than relying on the actual network topology.

2) Network types: We assume three kinds of networks, depending on their aggregation capacity:

- *Fully-pipelined networks*, aggregating all incoming messages into a single message. This case is ensured whenever $\phi_i \geq P_{i-1} W_{i-1}$. Since M_i is the size of the largest message sent at level i, we assume $\phi_i = M_i$. We also assume that all levels below are themselves fully-pipelined. Therefore, the aggregation capacity becomes $\phi_i = M_i = P_{i-1}\phi_{i-1}$.
- *Aggregating networks*, aggregating data up to volume of $\phi_i < M_i$ before sending a message.
- *Forward networks*, where messages coming from lower levels are simply forwarded. For a given level i, it is required that $\phi_i = \phi_{i-1}$: when each sub-node from level $i-1$ sends S_{i-1} messages, the number of forwarded messages is $\bar{S}_i = P_{i-1}\bar{S}_{i-1}$.

Based on the two extreme cases, we assume the aggregation capacity ϕ_i to satisfy $\phi_{i-1} \leq \phi_i \leq P_{i-1}\phi_{i-1}$.

3) An example of hierarchical platform modeled by HCP*:* Consider a distributed memory platform composed of D drawers having N compute nodes apiece. Let each node be a NUMA shared memory machine, with P processors. Within a node, each socket is connected to a local memory bank of size M, thus leading to a total shared memory of size $M \times P$ per node.

Within a drawer, nodes are interconnected with high speed interconnect such as fiber optics, whereas drawers are connected with more classical copper links. Let inter-drawer

communication bandwidth and latency respectively be W_{inter} and S_{inter}. Let intra-drawer communications have a bandwidth W_{d} and a latency S_{d}. For intra-node communications, we let W_{mem} (resp. S_{mem}) be the bandwidth (resp. latency) to exchange data with memory.

We model this platform in HCP using three levels, with the following characteristics:

# Comp. nodes	Bandwidth	Latency	Memory	Agg. capacity
$P_1 = P$	$W_1 = W_{\text{mem}}$	$S_1 = S_{\text{mem}}$	$M_1 = M$	$\phi_1 = M$
$P_2 = N$	$W_2 = W_{\text{d}}$	$S_2 = S_{\text{d}}$	$M_2 = P_1 M_1$	$\phi_2 = M \cdot P_1$
$P_3 = D$	$W_3 = W_{\text{inter}}$	$S_3 = S_{\text{inter}}$	$M_3 = P_2 M_2$	$\phi_3 \leq M \cdot P_2$

The aggregation capacities are chosen as follows: (1) On such hierarchical platform, a processor is able to transfer, in one message, its entire local bank of memory to another processor within the same compute node. This is ensured by setting ϕ_1 to M. (2) A compute node can transfer its entire shared memory to a remote node in the same drawer in a single message. The aggregation capacity is therefore chosen as $\phi_2 = MP_1$. (3) Finally, at the topmost level, the interconnect generally does not allow for sending the global volume of data coming from all drawers using a single message. The aggregation capacity is thus chosen as $\phi_3 \leq MP_2$.

HCP allows to model typical HPC platforms, giving communication details at each level of the hierarchy. The switch from a shared memory to a distributed memory environment is handled through the choice of the aggregation capacities.

4) Lower bounds on communication: We now introduce lower bounds on communication at every level of the hierarchy.

Lower bounds on communication have been generalized in [4] for direct methods of linear algebra algorithms which can be expressed as three nested loops. We refine these lower bounds under our hierarchical model. For matrix product-like problems, at least one copy of the input matrix has to be stored in memory: a compute node of level i thus needs a memory of $M_i = \Omega(n^2/P_i^*)$. Furthermore, the lower bound on latency depends on the aggregation capacity ϕ_i of the considered level i, where a volume $\bar{W}_i$ needs to be sent in messages of size ϕ_i. Hence the lower bounds on communication at level i:

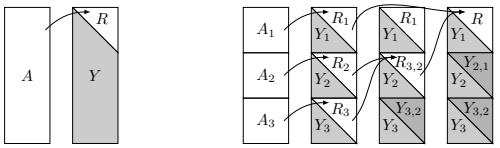
$$\bar{W}_i \geq \Omega\left(\frac{\#flops}{\sqrt{memory}}\right) = \Omega\left(\frac{n^2}{\sqrt{P_i^*}} \cdot \beta_i\right) \quad (1)$$

$$\bar{S}_i \geq \Omega\left(\frac{\bar{W}_i}{\phi_i} \cdot \alpha_i\right) = \Omega\left(\frac{n^2}{\phi_i \sqrt{P_i^*}} \cdot \alpha_i\right) \quad (2)$$

Note that, for simplicity, we expressed the bound on latency with respect to ϕ_i for all level i. Since we consider $\phi_1 = M_1$, the lower bound on latency for level 1 can also be expressed as $\bar{S}_1 = \Omega\left(\sqrt{P}\right)$.

IV. MULTILEVEL ALGORITHMS

In this section, we introduce *ML-CAQR* and *ML-CALU*, two multilevel algorithms for computing the QR and the LU factorizations of a dense matrix A. These multilevel algorithms heavily rely on their respective 1-level communication optimal algorithms, *CAQR* and *CALU*, and can be seen as a recursive version of these algorithms. *ML-CAQR* and *ML-CALU* recursive layout naturally allows for local elimination trees adapted to fit hierarchical platforms, thus reducing the communication needs at each level of the hierarchy.



(a) Single domain (b) Tree-based elimination

Fig. 3: Structure of the Householder reflectors

A. Multilevel QR factorization

ML-CAQR is a dense QR factorization algorithm targeting large scale hierarchical platforms. The focus is set on keeping the communication requirements as low as possible at every level of the hierarchy, like *CAQR* on platforms with one level of parallelism.

ML-CAQR, given in Algorithm 1, uses a recursive tree-based elimination scheme based on Householder reflections. As a tree-based algorithm, *ML-CAQR* stores the Householder reflectors in the lower triangular part of matrix A using a tree structure as in [3]. A small example is depicted on Figure 3, where a panel of matrix A is first split into three domains which are independently factored, then eliminated two by two. The resulting Householder reflectors should be applied following the same order to reflect the update or this panel.

At the topmost level of the hierarchy, *ML-CAQR* factors the entire input matrix A panel by panel. A panel is processed in multiple elimination steps following a tree-based approach. At the leaves of the tree, rectangular blocks are factored. The obtained R factors are then grouped two-by-two and eliminated in a sequence of elimination of size $2b_l$-by-b_l. Each factorization or elimination corresponds to a recursive call to *ML-CAQR* on the next lower level. After panel factorization, Householder reflectors are sent to remote compute nodes so as to update the trailing matrix using two recursive routines: *ML-Fact* and *ML-Elim*.

More formally, for each recursion level r, let b_r be the block size, $m_s^{(r)}$ be the panel row count at step s, and $n_s^{(r)}$ be the number of columns in the trailing matrix. At the topmost level l, we have $m_s^{(l)} = (m - (s-1)b_l)$ and $n_s^{(l)} = (n - sb_l)$.

For each panel of size b_r, *ML-CAQR* proceeds as follows:

1) The panel is factored by using a reduction operation, where *ML-CAQR* is the reduction operator. With a binary tree, it processes as follows:
   a) First, the panel is divided into P_{r_r} subdomains of size $m_s^{(r)}/P_{r_r}$-by-b_r, which are recursively factored with *ML-CAQR* at level $r-1$. At the deepest level, *CAQR* is called.
   b) The resulting b_r-by-b_r R factors are eliminated two-by-two by *ML-CAQR* at level $r-1$, requiring $\log P_{r_r}$ steps along the critical path.

Algorithm 1: *ML-CAQR*(A, m, n, r, P)

Input: Matrix A, m is the row-dimension of A, n is the number of columns, r is the level of recursion, P is the current compute node

Output: Factored matrix with R in the upper triangular part and the Householder reflectors Y in the lower triangular part

if $r = 1$ **then**
  Call *CAQR*(A, m, n, b_1, P)
else
  for $kk \leftarrow 1$ to n, with step of b_r **do**
    for Compute nodes $p \leftarrow 1$ to P_{r_r} **in parallel do**
      $h_p \leftarrow (m - kk - b_r)/P_{r_r}$
      $panel \leftarrow A(kk + (p-1)h_p : kk + p \cdot h_p, kk : kk + 1)$
      Call *ML-CAQR*$(panel, h_p, b_r, r-1, p)$
    for $j \leftarrow 1$ to $\log P_{r_r}$ **do**
      Nodes (p_{source},p_{target}) used to perform the elimination.
      Send local b_r-by-b_r to the remote node p_{target}
      Stack two b_r-by-b_r upper triangular matrices in RR
      Call *ML-CAQR*$(RR, 2b_r, b_r, r-1, p_{\text{source}})$
      Call *ML-CAQR*$(RR, 2b_r, b_r, r-1, p_{\text{target}})$
    for Compute nodes $p \leftarrow 1$ to P_{r_r} **in parallel do**
      Broadcast Householder vectors along processor row
      for Compute node $rp \leftarrow 2$ to P_{c_r} on same row than p **do**
        Call *ML-Fact*$(r-1, rp)$
    for $j \leftarrow 1$ to $\log P_{r_r}$ **do**
      Nodes (p_{source},p_{target}) used to perform the elimination.
      for Nodes $rp \leftarrow 2$ to P_{c_r} on same row than p_{source} **in parallel do**
        Remote node rp_{target} is on same row than p_{target} and same column than rp
        rp sends its local C to rp_{target}
        Call *ML-Elim*$(r-1, rp)$
        Call *ML-Elim*$(r-1, rp_{\text{target}})$

2) The current trailing matrix is then updated:
   a) Householder reflectors in lower trapezoidal part of the panel have to be broadcasted along processor rows.
   b) Updates corresponding to factorizations at the leaves of the tree are applied using the *ML-Fact* routine. *ML-Fact* broadcasts P_{r_r} blocks of Householder reflectors of size $m_s^{(r)}/P_{r_r}$-by-b_r from the column of nodes holding the current panel along rows of compute nodes. At the deepest level, the update corresponding to a leaf is applied as in *CAQR* (see [12]).
   c) The updates due to the eliminations of the intermediate R factors are then applied to the trailing matrix using the *ML-Elim* procedure. Blocks of size b_r-by-$n_s^{(r)}/P_{c_r}$ are exchanged within a pair of compute nodes. At the lowest level, a partial update is locally computed before being independently applied onto each processing elements (similarly to *CAQR*).

B. Multilevel LU factorizations

Here we present two variants of a multilevel algorithm for computing the LU factorization of a dense matrix, *ML-CALU*. Both algorithms are recursive. The first variant, 1D-*ML-CALU*, follows a uni-dimensional approach where the recursion is applied to the entire panel at each recursive call. The second variant, 2D-*ML-CALU*, processes a panel by multiple recursive calls on sub-blocks of the panel followed by a "reduction" phase similar to that of *ML-CAQR*. The base case of both recursive variants is *CALU* [10], which uses tournament pivoting to select pivot rows.

Algorithm 2: 1D-*ML-CALU* (A, m, n, r, P)

Input: $m \times n$ matrix A, the recursion level r, block size b_r, the total number of compute nodes $P = \prod_{i=1}^{l} P_i = P_r \times P_c$

Output: Factored matrix such that $A = LU$

if $r = 1$ **then**
  $[\Pi_1, L_1, U_1] \leftarrow$ *CALU*$(A, b_1, P_r \times P_{c_1})$
else
  $M \leftarrow m/b_r, N = n/b_r$
  for $K \leftarrow 1$ to N **do**
    $[\Pi_{KK}, L_{K:M,K}, U_{KK}] \leftarrow$
    1D-*ML-CALU*$(A_{K:M,K}, r-1, b_{r-1}, P_r \times \prod_{i=1}^{r-1} P_{c_i})$
    Apply permutation and compute block row of U
    $A_{K:M,:} \leftarrow \Pi_{KK} A_{K:M,:}$
    for each compute node owning a block $A_{K,J}$, $J \leftarrow K+1$ to N **in parallel do**
      Call multilevel dtrsm using $P_{c_r} \times \prod_{i=1}^{r-1} P_i$ compute nodes of level 1
      $U_{K,J} \leftarrow L_{KK}^{-1} A_{K,J}$
    Update the trailing submatrix
    for each compute node owning a trailing block $A_{I,J}$, $I, J \leftarrow K+1$ to M, N **in parallel do**
      Call multilevel dgemm using $P_r \times \prod_{i=1}^{r} P_{c_i}$ compute node of level 1
      $A_{I,J} \leftarrow A_{I,J} - L_{I,K} U_{K,J}$

Algorithm 2 describes 1D-*ML-CALU* variant. At each recursion level r the algorithm proceeds as follows: (1) 1D-*ML-CALU* is recursively applied to a smaller panel of b_r columns. We note that only the number of compute nodes along the columns is varying. (2) Once a panel is factored, a block of rows of U is computed by $P_{c_r} \times P_{r_r}^*$ compute nodes of level r. (3) The trailing matrix is finally updated after a broadcast of the block column of L along rows of the process grid and the block row of U along columns of the process grid. The matrix-matrix operations are performed using a multilevel matrix product algorithm, *ML-Cannon*, which is based on the optimal Cannon algorithm. For more details, we refer the interested readerto the related research report [13].

At the deepest level of recursion, panels of size $m \times b_1$ are factored by *CALU* using P_r processing elements of level 1. Gaussian elimination with partial pivoting is first applied to blocks of size (m/P_r)-by-b_1, located at the leaves of the reduction tree. These candidate pivot rows are then combined using tournament pivoting, which involves communications at every level in the hierarchy.

In terms of numerical stability, 1D-*ML-CALU* is equivalent to performing *CALU* on a matrix of size $m \times n$, using a block size b_1 and a grid of processors $P = P_r \times P_c$.

The 2D-*ML-CALU* algorithm was first introduced in [14]

and analyzed for two levels of parallelism. Here we extend the analysis to more levels of parallelism. Algorithm 3 describes in details the different steps of 2D-*ML-CALU*. It proceeds as follows: (1) the panel is recursively factored with 2D-*ML-CALU* with a block size corresponding to the next level in the hierarchy. Note that at the deepest level of recursion, 2D-*ML-CALU* calls *CALU*. (2) The selected sets of pivot candidates are merged two-by-two along the reduction tree, where the reduction operator is 2D-*ML-CALU*. At the end of this preprocessing step, the final set of pivot rows is selected and each node working on the panel has the pivot information and the diagonal block of U. (3) The computed permutation is then applied to the input matrix, the block column of L and the block row of U are computed. (4) Finally, after the broadcast of L and U to appropriate nodes, as in 1D-*ML-CALU*, the trailing matrix is updated with *ML-Cannon*.

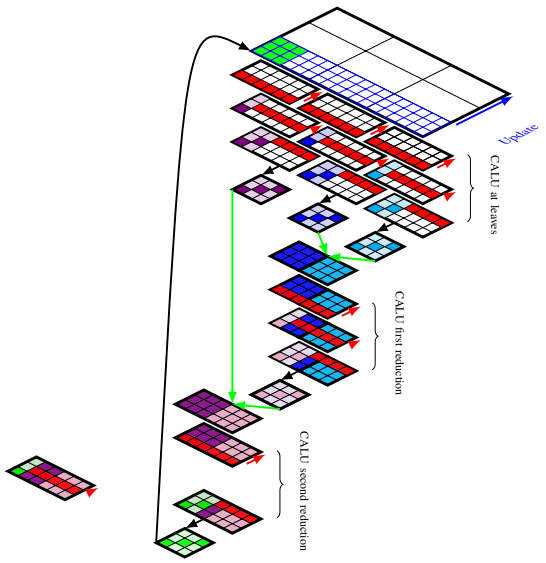


Fig. 4: Factorization of a panel with 2D-*ML-CALU* a machine with two levels of parallelism.

Figure 4 illustrates the panel factorization for 2D-*ML-CALU* running on a machine with two levels of parallelism, that is each compute node has several cores. We consider P_2 nodes (3 are presented in the figure), each having P_1 cores ($P_1 = 6$ in the figure). There are two kinds of communication: the communication between the different nodes, which corresponds to the topmost reduction tree and the synchronization between the different cores inside a node, which corresponds to the internal reduction trees. Hence the figure presents two levels of reduction. The topmost reduction tree allows compute nodes to synchronize and perform inter-node communication. The deepest reduction trees represent the communication inside each compute node.

V. Performance models

In this section, we provide cost analysis of both *ML-CAQR* and *ML-CALU* algorithms with respect to the the HCP model. In each multilevel algorithm, two types of communication

Algorithm 3: 2D-*ML-CALU* (A, m, n, r, P)

Input: $m \times n$ matrix A, level of parallelism r in the hierarchy, block size b_r, number of nodes $P_r = P_{r_r} \times P_{c_r}$
Output: Factored matrix such that $A = LU$
if $r = 1$ **then**
  Call *CALU*(A, m, n, b_1, P)
else
  for $k \leftarrow 1$ to n/b_r **do**
    $m_p \leftarrow (m - (k-1)b_r)/P_{r_r}$
    $n_p \leftarrow (n - (k-1)b_r)/P_{c_r}$
    Factor leaves of the panel
    for Processor $p \leftarrow 1$ to P_{r_r} **in parallel do**
      leaf $\leftarrow A((k-1)\cdot b_r + (p-1)m_p + 1 : (k-1)\cdot b_r + p\cdot m_p, (k-1)\cdot b_r + 1 : k\cdot b_r)$
      Call 2D-*ML-CALU*(leaf, $m_p, b_r, r-1, P_{r-1}$)
    Reduction steps
    for $j \leftarrow 1$ to $\log P_{r_r}$ **do**
      Stack two b_r-by-b_r sets of candidate pivot rows in B
      Call 2D-*ML-CALU*$(B, 2b_r, b_r, r-1, P_{r-1})$
    Compute block column of L
    for Processor $p \leftarrow 1$ to P_{r_r} **in parallel do**
      Compute $L_{p,k} \leftarrow L(k\cdot b_r + (p-1)m_p + 1 : k\cdot b_r + p\cdot m_p, (k-1)\cdot b_r + 1 : k\cdot b_r)$
    Apply all row permutations
    for Processor $p \leftarrow 1$ to P_{r_r} **in parallel do**
      Broadcast pivot information along the rows of the process grid
      All to all reduce operation using P_r processors of level r
      Swap rows at left and right
    Broadcast right diagonal block of $L_{k,k}$ along rows of the process grid
    Compute block row of U
    for Processor $p \leftarrow 1$ to P_{c_r} **in parallel do**
      Compute $U_{k,p} \leftarrow U((k-1)\cdot b_r + 1 : k\cdot b_r, k\cdot b_r + (p-1)n_p + 1 : k\cdot b_r + p\cdot n_p)$
    Update trailing matrix
    for Processor $p \leftarrow 1$ to P_{c_r} **in parallel do**
      Broadcast $U_{k,p}$ along the columns of the process grid
    for Processor $p \leftarrow 1$ to P_{r_r} **in parallel do**
      Broadcast $L_{p,k}$ along the rows of the process grid
    for Processor $p \leftarrow 1$ to P_r **in parallel do**
      $A(k\cdot b_r + (p-1)m_p + 1 : k\cdot b_r + p\cdot m_p, k\cdot b_r + (p-1)n_p + 1 : k\cdot b_r + p\cdot n_p) \leftarrow A(k\cdot b_r + (p-1)m_p + 1 : k\cdot b_r + p\cdot m_p, k\cdot b_r + (p-1)n_p + 1 : k\cdot b_r + p\cdot n_p) - L_{p,k}\cdot U_{k,p}$
      Using *ML-Cannon* with P_r nodes at level r

primitives are used, namely point-to-point and broadcast operations. To simplify the analysis, we define two recursive costs corresponding to these communication patterns.

In a *point-to-point communication*, a volume D is transfered between two compute nodes of level r. All compute nodes from level 1 to level $r-1$ below those two nodes of level r are involved, sending their local data to their respective counterparts in the remote node of level r. The associated communication costs are therefore:

$$W_{\text{P2P}}(1 \ldots r, D) = \sum_{k=1}^{r} \frac{D \cdot P_r^*}{P_k^*} \beta_k,$$

$$S_{\text{P2P}}(1\dots r, D) = \alpha_1 + \sum_{k=2}^{r} \frac{D \cdot P_r^*}{\phi_k P_k^*} \alpha_k.$$

A *broadcast operation* between P_{c_r} compute nodes of level r is very similar to point to point communication. However at every level, a participating node broadcasts its data to P_{c_r} counterparts. A broadcast can thus be seen as $\log P_{c_r}$ point-to-point communications.

A. ML-CAQR

We now review the global computation and communication costs of *ML-CAQR*. At each recursion level r, the current panel is factored by doing P_{r_r} parallel calls to *ML-CAQR*. Then, the resulting R factors are eliminated through $\log P_{r_r}$ successive factorizations of $2b_r$-by-b_r matrices formed by stacking up two upper triangular R factors. Once a panel is factored, the trailing matrix is updated. However, as the Householder reflectors are stored in a tree structure, the updates must be done in the same order as during panel factorizations. These operations are recursively performed using *ML-Fact* for the leaves and *ML-Elim* for higher levels in the tree.

The global recursive cost of *ML-CAQR* is composed of several contributions. We let:

- $T_{\textit{CAQR}}(m, n, b, P)$ be the cost of factoring a matrix of size m-by-n with *CAQR* using P processors and a block size b.
- $T_{\textit{ML-CAQR}}(m, n, b, P)$ be the cost of *ML-CAQR* on a m-by-n matrix using P processors and a block size b.
- $T_{\textit{ML-Fact}}(m, n, b, P)$ be the cost of updating the trailing matrix to reflect factorizations at the leaves of the elimination trees.
- Finally, $T_{\textit{ML-Elim}}(m, n, b, P)$ be the cost of applying updates corresponding to higher levels in the trees.

In terms of communication, *ML-Fact* consists in broadcasting Householder reflectors along process rows, while *ML-Elim* corresponds to $\log P_{r_r}$ point to point communications of trailing matrix blocks between pairs of nodes within a process column. Using these notations, the cost $T_{\textit{ML-CAQR}}(m, n, b_r, P_r)$ of *ML-CAQR* can be expressed as,

$$\begin{cases} \sum_{s=1}^{n/b_r} \Big[T_{\textit{ML-CAQR}}\left(\frac{m-(s-1)b_r}{P_{r_r}}, b_r, b_{r-1}, P_{r-1}\right) \\ \quad + \log P_{r_r} \cdot T_{\text{P2P}}(1\dots r, \frac{b_r^2}{2}) \\ \quad + \log P_{r_r} \cdot T_{\textit{ML-CAQR}}(2b_r, b_r, b_{r-1}, P_{r-1}) & \text{if } r > 1 \\ \quad + T_{\textit{ML-Fact}}\left(\frac{m-(s-1)b_r}{P_{r_r}}, \frac{n-sb_r}{P_{c_r}}, b_{r-1}, P_{r-1}\right) & \quad (3) \\ \quad + \log P_{r_r} \cdot T_{\textit{ML-Elim}}\left(2b_r, \frac{n-sb_r}{P_{c_r}}, b_{r-1}, P_{r-1}\right) \Big] \\ \\ T_{\textit{CAQR}}(m, n, b_1, P_1) & \text{if } r = 1 \end{cases}$$

ML-CAQR uses successive elimination trees at each recursion level r, each of which are traversed in $\log P_{r_r}$ steps. Moreover, successive trees from level l down to level r come from different recursive calls: they are inherently sequentialized. Thus, the total number of calls at a given recursion level r can be upper-bounded by $N_r = 2^{l-r} \prod_{j=r}^{l} \log P_{r_j}$.

An upper bound on the global cost of *ML-CAQR* can be expressed in terms of number of calls at each level of recursion, broken down between calls performed on leaves or higher level in the trees.

Assuming that for each level k, we have $P_{r_k} = P_{c_k} = \sqrt{P_k}$, and that block sizes are chosen to make the additional costs lower order terms, that is $b_k = O(n/(\sqrt{P_k^*} \cdot \prod_{j=k}^{l} \log^2 P_j))$, the cost of *ML-CAQR* becomes:

$$\bar{F}_{\textit{ML-CAQR}}(n, n) \leq \frac{4n^3}{P}\gamma + O\left(\frac{l \cdot n^3}{P \prod_{j=1}^{l} \log P_j}\right)\gamma \quad (4)$$

$$\begin{aligned} \bar{W}_{\textit{ML-CAQR}}(n, n) \leq{}& \frac{n^2}{\sqrt{P}}\left(l \cdot \log P_1 + 4l \cdot \prod_{j=1}^{l} \log P_j + \log P_l\right)\beta_1 \quad (5) \\ &+ \sum_{k=2}^{l-1} \frac{(l-k) \cdot n^2}{\sqrt{(P_k^*)}}\left(1 + \frac{2\prod_{j=k}^{l} \log P_j}{\sqrt{P_l}}\right)\beta_k + \frac{n^2 \cdot \log P_l}{\sqrt{P_l^*}}\beta_l \\ &+ O\left(\frac{l \cdot n^2}{\sqrt{P} \log P_l} \cdot \beta_1 + \sum_{k=2}^{l-1} \frac{(l-k) \cdot n^2}{\sqrt{P_k^*} \log P_l} \cdot \beta_k + \frac{n^2}{\sqrt{P_l^*} \log P_l} \cdot \beta_l\right) \end{aligned}$$

$$\begin{aligned} \bar{S}_{\textit{ML-CAQR}}(n, n) \leq{}& l \cdot \sqrt{P} \cdot \prod_{j=1}^{l} \log^3 P_j \alpha_1 + \sum_{k=2}^{l-1} \frac{n^2 \cdot (l-k) \log P_k}{\phi_k \sqrt{P_k^*}}\alpha_k \quad (6) \\ &+ \frac{n^2 \cdot \log P_l}{\phi_l \sqrt{P_l}}\left(1 + \frac{1}{\prod_2^{l-1} \sqrt{P_j}}\right)\alpha_l \\ &+ O\left(\sqrt{P} \cdot \prod_{j=1}^{l} \log^2 P_j \alpha_1 + \sum_{k=2}^{l-1} \frac{(l-k) \cdot n^2}{\phi_k \sqrt{P_k^*} \log P_l}\alpha_k + \frac{n^2}{\phi_l \sqrt{P_l \log P_l}}\alpha_l\right) \end{aligned}$$

Though the recursive nature of *ML-CAQR* leads to at least three times more computations than the optimal algorithm, which is similar to other recursive approaches [15], it allows to reach the lower bounds on communications at all levels of the hierarchy up to polylogarithmic factors. Indeed, chosing appropriate block sizes makes most of the extra computational costs lower order terms while maintaining the optimality in terms of communication. We refer the interested reader to the related research report [13] for more details on these costs.

B. 2D-ML-CALU

In this section we only detail the cost of 2D-*ML-CALU* with respect to the HCP model. Thus, for simplicity, we refer to it as *ML-CALU* throughout the rest of the paper. Using the same reasoning than for *ML-CAQR*, for a square n-by-n matrix and using l levels of recursion ($l \geq 2$), the cost of *ML-CALU* is:

$$\begin{aligned} \bar{F}_{\textit{ML-CALU}}(n, n) \leq{}& \left[\frac{2n^3}{3P} + \frac{n^3}{P \log^2 P_l} + \frac{n^3}{P}\left(\frac{3}{8}\right)^{l-2}\left(\frac{5}{16}l - \frac{53}{128}\right)\right]\gamma \quad (7) \\ &+ O\left(\frac{n^2}{\sqrt{P}}\right)\gamma \end{aligned}$$

$$\begin{aligned} \bar{W}_{\textit{ML-CALU}}(n, n) \leq{}& \left[\frac{n^2}{2\sqrt{P}} \log P_1 \prod_{j=2}^{l}(1 + \frac{1}{2}\log P_j)\right]\beta_1 \quad (8) \\ &+ \sum_{k=1}^{l}\Big[\frac{n^2}{\sqrt{P_k^*}}\Big(\frac{8}{3}\log^2 P_l(1 + \frac{l-k}{\sqrt{P_k}}))\prod_{j=3}^{l}(1 + \frac{1}{2}\log P_j)\Big)\Big]\beta_k. \\ &+ \sum_{k=1}^{l}\Big[\frac{n^2}{\sqrt{P_k^*}}\Big(\frac{(l-2)}{8}(1 + \frac{l}{4})\prod_{j=3}^{l}(1 + \frac{1}{2}\log P_j)\Big)\Big]\beta_k. \end{aligned}$$

$$\begin{aligned} \bar{S}_{\textit{ML-CALU}}(n, n) \leq{}& \left[\frac{n^2}{2\sqrt{P}} \log P_1 \prod_{j=2}^{l}(1 + \frac{1}{2}\log P_j)\right]\alpha_1 \quad (9) \\ &+ \sum_{k=1}^{l}\Big[\frac{n^2}{\phi_k\sqrt{P_k^*}}\Big(\frac{8}{3}\log^2 P_l(1 + \frac{l-k}{\sqrt{P_k}}))\prod_{j=3}^{l}(1 + \frac{1}{2}\log P_j)\Big)\Big]\alpha_k \\ &+ \sum_{k=1}^{l}\Big[\frac{n^2}{\phi_k\sqrt{P_k^*}}\Big(\frac{(l-2)}{8}(1 + \frac{l}{4})\prod_{j=3}^{l}(1 + \frac{1}{2}\log P_j)\Big)\Big]\alpha_k \end{aligned}$$

Equation 7 shows that *ML-CALU* performs more floating-point operations than *CALU*. This is because of the recursive calls during the panel factorization. Note that certain assumptions should be done regarding the hierarchical structure of

the computational system in order to keep the extra flops as a low order term, and therefore asymptotically reach the lower bounds on computation. More details on these costs could be found in [16, section 5.4].

In terms of communication *ML-CALU* attains the lower bounds derived in section III modulo a factor that depends on $l^2 \prod_{j=2}^{l} \log P_j$ at each level of the hierarchy. We do not give the detailed cost of 1D-*ML-CALU* here. However we would like to point out that it attains the lower bounds derived under the HCP model in terms of bandwidth at each level of parallelism. In terms of latency the lower bound is only met at the deepest level of parallelism.

VI. Experimental results

A. Numerical stability of ML-CALU

Since *ML-CALU* is based on recursive calls, its stability can be different from that of *CALU*. Our experiments show that up to three levels of parallelism *ML-CALU* exhibits a good stability, however further investigation is required if more than three levels of parallelism are used. We study both the stability of the LU decomposition and of the linear solver, in terms of growth factor and three different backward errors: the normwise backward error, the componentwise backward error, and the relative error $\|PA - LU\|/\|A\|$.

Figure 5 displays the values of the ratios of 3-level *ML-CALU*'s growth factor and backward errors to those of GEPP for 36 special matrices [10]. The tested matrices are of size 8192, using the following parameters: $P_{r_3} = 16$, $b_3 = 64$, $P_{r_2} = 4$, $b_2 = 32$, $P_{r_1} = 4$, and $b_1 = 8$. We can see that nearly all ratios are between 0.002 and 2.4 for all tested matrices. For the growth factors, the ratio is of order 1 in 69% of the cases. For the relative errors, the ratio is of order 1 in 47% of the cases. Using different number of nodes and size of blocks we obtain similar results.

We recall that *ML-CALU* uses tournament pivoting to select pivots at each level of recursion, which does not ensure that the element of maximum magnitude in the column is used as pivot, neither at a single level of the hierarchy nor at each step of the LU factorization, that is globally for the panel. For that reason we consider a threshold τ_k, defined as the quotient of the pivot used at step k divided by the maximum value in column k. We observe that in practice the pivots used by recursive tournament pivoting are close to the elements of maximum magnitude in the respective columns. For example, for a binary tree based *ML-CALU* on 3 levels, the selected pivot rows are equal to the elements of maximum magnitude in 63% of the cases, and for the rest of the cases the minimum threshold $\tau_{\min}$ is larger than 0.30. We should note that regarding the matrix size and the number of compute nodes used in these experiments we can not derive general conclusions with respect to exascale platforms. However the previous experiments give us an insight on the numerical stability of the algorithm.

B. Performance predictions

Multilevel communication avoiding algorithms are tailored for large scale platforms displaying a significant gap between processing power and communication speed. The upcoming Exascale is a natural target for these algorithms.

We present performance predictions on a sample exascale platform. Current petascale platforms already display a hierarchical nature which strongly impacts the performance of parallel applications. Exascale will dramatically amplify this trend. We plan here to provide an insight on what could be observed on such platforms.

Level	Type	#	Bandwidth	Latency
1	2x 6-cores Opterons	12	19.8 GB/s	1×10^{-9}s
2	Hopper nodes	2	10.4 GB/s	1×10^{-6}s
3	Gemini ASICS	9350	3.5 GB/s	1.5×10^{-6}s

TABLE I: Characteristics of NERSC Hopper.

As exascale platforms are not available yet, we base our sample exascale platform on the characteristics of the NERSC Hopper [17, 18] petascale platform. It is composed of *Compute Nodes*, each with two hexacore AMD Opteron Magny-cours 2.1GHz processors offering a peak performance of 8.4 GFlop/s, with 32 GB of memory. Nodes are connected in pairs to *Gemini ASICs*, which are interconnected through the *Gemini network* [19, 20]. Detailed parameters of the Hopper platform are presented in Table I.

Level	Type	#	Bandwidth	Latency (formula)	Latency (adjusted)
1	Multi-cores	1024	300 GB/s	1×10^{-10}s	1×10^{-9}s
2	Nodes	32	150 GB/s	1×10^{-7}s	1.2×10^{-7}s
3	Interconnects	32768	50 GB/s	1.5×10^{-7}s	1×10^{-6}s

TABLE II: Characteristics of a sample exascale platform.

Our target platform is obtained by increasing the number of nodes at all 3 levels, leading to a total of $1M$ nodes. The amount of memory per processing element is kept constant at 1.3 GB. Moreover, exascale platforms are likely to be available around year 2018. Therefore, latencies and bandwidths are derived using an average 15% decrease per year for the latency and a 26% increase for the bandwidth [20, 19].

However, doing so might conduct to latencies so low that electrical signals would have to travel faster than the speed of light in vacuum. This is of course impossible. Therefore, to alleviate this problem, we assume that electrical signal travels at 10% of the speed of light in copper, against 90% in fiber optics. We consider the links within a multicore processor to be made out of copper (at level 1) and the die to be at most 3cm-by-3cm. The links between a group of nodes (i.e. at level 2) are assumed to be based on fiber optics while the interconnect at the last level are assumed to be copper links. Finally, we assume the global supercomputer footprint to be 30m-by-30m. These parameters are detailed in Table II.

We model the platform with respect to the HCP model, and use it to estimate the running times of our algorithms.

We note that in order to assess the performance of multilevel algorithms, costs of state-of-the-art 1-level communication avoiding algorithms need to be expressed with respect to the

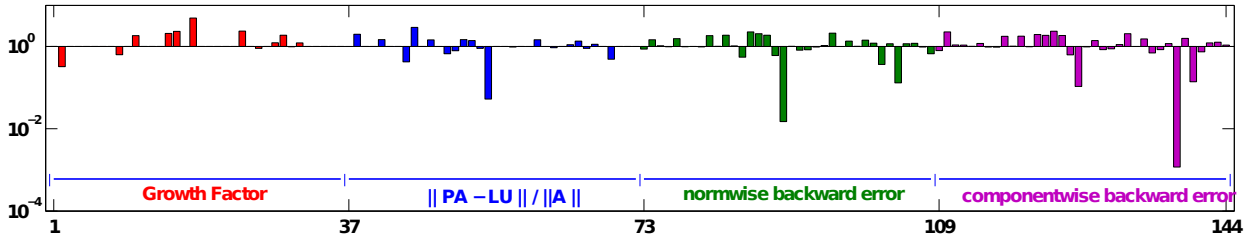


Fig. 5: Ratios of 3-level *CALU*'s growth factor and backward errors to GEPP's.

HCP model. To this end, we assume (1) each communication to go through the entire hierarchy: two communicating nodes thus belong to two distant nodes of level l, hence a bandwidth β_l. (2) Bandwidth is shared among parallel communications.

We evaluate the performance of the *ML-CAQR* and *ML-CALU* algorithms as well as their corresponding 1-level routines on a square matrix of size $n \times n$, distributed over a square 2D grid of P_k processors at each level k of the hierarchy, $P_k = \sqrt{P_k} \times \sqrt{P_k}$. In the following, we assume all levels to be *fully-pipelined*. Similar results are obtained regarding *forward* hierarchies, which is explained by the fact that realistic test cases are not latency bound, but are mostly impacted by their bandwidth cost.

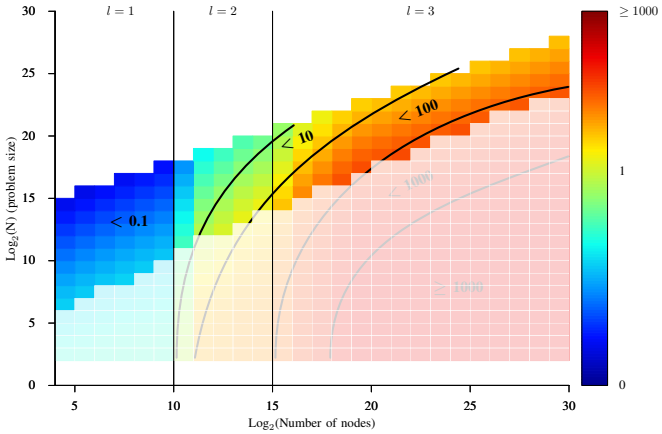


Fig. 6: Prediction of communication for 1-level *CAQR* to computation ratio on an exascale platform.

The larger the platform is, the more expensive the communications become. This trend can be illustrated by observing the communication to computation ratio, or *CCR* of an algorithm.

On Figures 6 and 7, we plot the *CCR* of both *CAQR* and *ML-CAQR* on the exascale platform. The shaded areas correspond to unrealistic cases where there are more processing elements than matrix elements and should not be considered. As the number of processing elements increases, cost of *CAQR* (on Figure 6) is dominated by communications. Our multilevel approach alleviates this trend, and *ML-CAQR* (on Figure 7) allows to maintain a good computational density, especially when the number of levels involved is large. Note that for $l = 1$, *ML-CAQR* and *CAQR* are equivalent.

However, as *ML-CAQR* performs more computations than

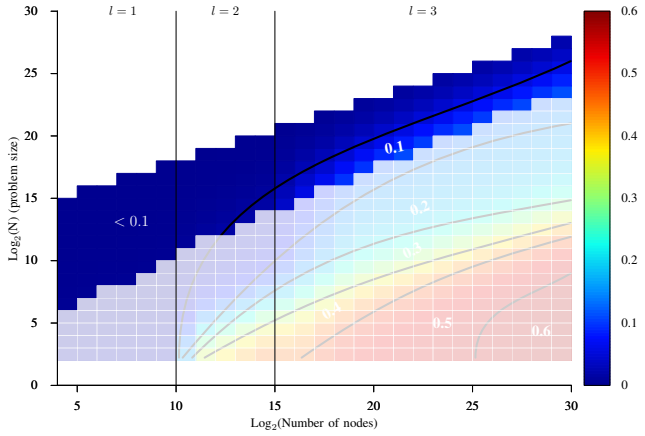


Fig. 7: Prediction of communication for *ML-CAQR* to computation ratio on an exascale platform.

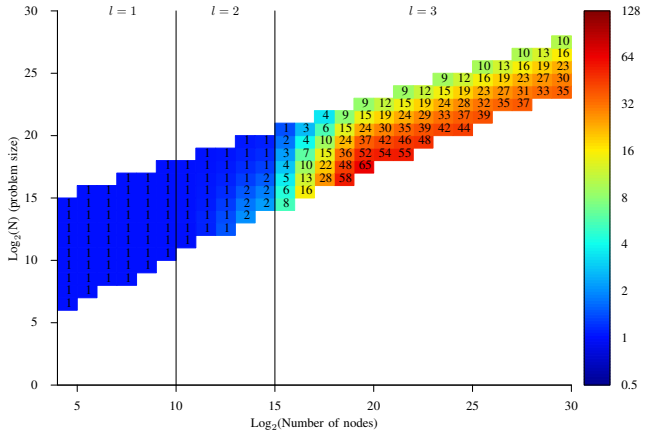


Fig. 8: Speedup of *ML-CAQR* vs. 1-level *CAQR*

CAQR, we compare the expected running times of both algorithms. Here, we denote by running time the sum of computational and communication costs. We thus assume no overlap between computations and communications, which is generally hard to achieve at such an extreme scale. The ratio of the *ML-CAQR* running time over *CAQR* is depicted on Figure 8. *ML-CAQR* clearly outperforms *CAQR* when using the entire platform, despite its higher computational costs. As a matter of a fact in this regime, the running time is dominated

by the bandwidth cost, and *ML-CAQR* significantly reduces it at all levels.

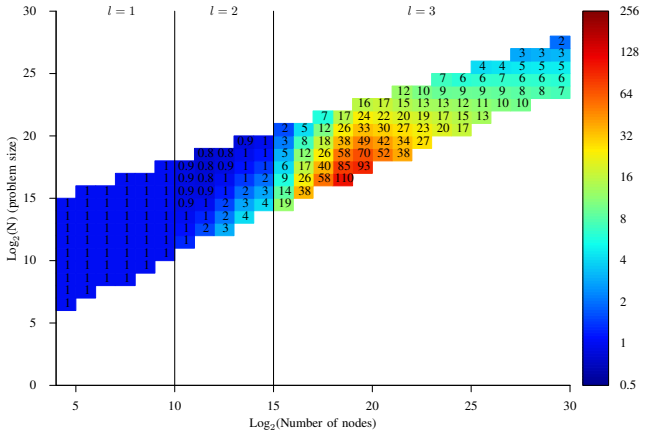


Fig. 9: Speedup of *ML-CALU* vs. 1-level *CALU*

The same observations can be made on the *CCR* of *CALU* and *ML-CALU*, we will therefore not present the details here.

Regarding the running times ratio, depicted on Figure 9, we can also conclude that *ML-CALU* is able to keep communication costs significantly lower than *CALU*, leading to significant performance improvements.

Altogether, our performance predictions validate our multilevel approach for large scale hierarchical platforms that will arise with the Exascale. Indeed, by taking communication into consideration at all levels, *ML-CAQR* and *ML-CALU* deliver a high level of performance at scales where performance is hindered by communication costs even with 1-level communication avoiding algorithms.

VII. Conclusion

In this paper we have introduced two algorithms, *ML-CAQR* and *ML-CALU*, that minimize communication over multiple levels of parallelism at the cost of performing redundant computation. The complexity analysis is performed within HCP, a model that takes into account the communication cost at each level of a hierarchical platform. The multilevel QR algorithm has similar stability properties to classic algorithms. Two variants of the multilevel LU factorization are discussed. A first variant, based on a uni-dimensional recursive approach, has the same stability as *CALU*. However, while it minimizes bandwidth over multiple levels of parallelism, it allows to minimize latency only over one level of parallelism. The second variant which uses a two-dimensional recursive approach, is shown to be stable in practice, and reduces both bandwidth and latency over multiple levels of parallelism.

Our performance predictions on a model exascale platform show that for strong scaling, the multilevel algorithms lead to important speedups compared to algorithms minimizing communication over only one level of parallelism.

Moreover, in most of the cases, minimizing bandwidth is the key factor for improving scalability, and hence the 1D-*ML-CALU* is also an appropriate choice for an efficient LU factorization while ensuring a good numerical stability in practice.

References

[1] J.-W. Hong and H. T. Kung, "I/O complexity: The Red-Blue Pebble Game," in STOC '81: Proceedings of the Thirteenth Annual ACM Symposium on Theory of Computing. New York, NY, USA: ACM, 1981, pp. 326–333.

[2] D. Irony, S. Toledo, and A. Tiskin, "Communication lower bounds for distributed-memory matrix multiplication," J. Parallel Distrib. Comput., vol. 64, no. 9, pp. 1017–1026, 2004.

[3] J. W. Demmel, L. Grigori, M. Hoemmen, and J. Langou, "Communication-optimal parallel and sequential QR and LU factorizations," SIAM Journal on Scientific Computing, 2012, short version of technical report UCB/EECS-2008-89 from 2008.

[4] G. Ballard, J. Demmel, O. Holtz, and O. Schwartz, "Minimizing communication in numerical linear algebra," SIAM Journal on Matrix Analysis and Applications, vol. 32, pp. 866–901, 2011.

[5] E. Agullo, C. Coti, J. Dongarra, T. Herault, and J. Langou, "QR factorization of tall and skinny matrices in a grid computing environment," in IPDPS'10, the 24st IEEE Int. Parallel and Distributed Processing Symposium, 2010.

[6] F. Song, H. Ltaief, B. Hadri, and J. Dongarra, "Scalable tile communication-avoiding QR factorization on multicore cluster systems," in SC'10, the 2010 ACM/IEEE conference on Supercomputing. IEEE Computer Society Press, 2010.

[7] G. Bosilca, A. Bouteiller, A. Danalis, M. Faverge, A. Haidar, T. Herault, J. Kurzak, J. Langou, P. Lemarinier, H. Ltaief, P. Luszczek, A. YarKhan, and J. Dongarra, "Flexible development of dense linear algebra algorithms on massively parallel architectures with DPLASMA," in 12th IEEE International Workshop on Parallel and Distributed Scientific and Engineering Computing (PDSEC'11), 2011.

[8] J. Dongarra, M. Faverge, T. Herault, M. Jacquelin, J. Langou, and Y. Robert, "Hierarchical qr factorization algorithms for multi-core clusters," Parallel Computing, no. 0, pp. –, 2013.

[9] R. Schreiber and C. Van Loan, "A storage efficient WY representation for products of Householder transformat ions," SIAM J. Sci. Stat. Comput., vol. 10, no. 1, pp. 53–57, 1989.

[10] L. Grigori, J. Demmel, and H. Xiang, "CALU: A communication optimal LU factorization algorithm," SIAM Journal on Matrix Analysis and Applications, vol. 32, pp. 1317–1350, 2011.

[11] F. Cappello, P. Fraigniaud, B. Mans, and A. Rosenberg, "An algorithmic model for heterogeneous hyper-clusters: rationale and experience," International Journal of Foundations of Computer Science, vol. 16, no. 02, pp. 195–215, 2005.

[12] J. W. Demmel, L. Grigori, M. F. Hoemmen, and J. Langou, "Communication-optimal parallel and sequential QR and LU factorizations," LAPACK Working Note 204, Aug. 2008.

[13] L. Grigori, M. Jacquelin, and A. Khabou, "Multilevel communication optimal LU and QR factorizations for hierarchical platforms," CoRR, vol. abs/1303.5837, 2013.

[14] S. Donfack, L. Grigori, and A. Khabou, "Avoiding Communication through a Multilevel LU Factorization," in Euro-Par, 2012, pp. 551–562.

[15] J. Frens and D. Wise, "Qr factorization with morton-ordered quadtree matrices for memory re-use and parallelism," in ACM SIGPLAN Notices, vol. 38, no. 10. ACM, 2003, pp. 144–154.

[16] A. Khabou, "Dense matrix computations: communication cost and numerical stability." Ph.D. dissertation, Université Paris Sud-Paris XI, 2013.

[17] NERSC, "Hopper configuration page," http://www.nersc.gov/users/computational-systems/hopper/configuration.

[18] J. Shalf, "Cray xe6 architecture," http://www.nersc.gov/assets/Uploads/ShalfXE6ArchitectureSM.pdf, 2011.

[19] P. K. Editor & lead study, "Exascale computing study: Technology challenges in achieving exascale systems," 2008.

[20] S. Graham, M. Snir, C. Patterson, and National Research Council (U.S.). Committee on the Future of Supercomputing, Getting up to speed: the future of supercomputing. National Academies Press, 2005.