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

(Regular Submission)

Laura Grigori
INRIA Paris - Rocquencourt
laura.grigori@inria.fr

Mathias Jacquelin
INRIA Paris - Rocquencourt
mathias.jacquelin@inria.fr

Amal Khabou
LRI - University Paris Sud
amal.khabou@lri.fr

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 new 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 communications 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 numerical experiments and performance predictions demonstrating the need for such algorithms on large platforms.

Keywords: QR, LU, exascale, hierarchical platforms.

1 Introduction

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 [14]. This result has been extended to parallel machines [15], to dense LU and QR factorizations (under certain assumptions) [7], and then to basically all direct methods in linear algebra [4]. Given an algorithm that performs a certain number of floating point operations, and given a size of the memory M, the lower bounds on communication are obtained by using the Loomis-Whitney inequality, as for example in [15, 4]. While theoretically important, these lower bounds are derived with respect to a simple 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 architectures with multilevel hierarchy.

On the practical side, several algorithms have been introduced recently [3, 17, 5, 10]. 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 [10], 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, no theoretical bounds nor cost analysis are provided in these studies regarding communication.

In the first part of this paper we introduce a new model that we refer to as the 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 be applicable to 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 and heterogeneous nature of present-day computers. We determine what is 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.

In the second part of the paper we introduce two multilevel algorithms for computing LU and QR 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 computation. These recursive algorithms rely on their corresponding 1-level algorithms (resp. CAQR and CALU) as their base case. Indeed, these algorithms are known to attain the communication lower bounds in terms of both bandwidth, and latency with respect to the simpler one level performance model.

2 Toward a realistic Hierarchical Cluster Platform model (HCP)

The focus is set on hierarchical platforms implementing deeper and deeper hierarchies. Typically, these platforms are composed of two kinds of hierarchies: (1) a network hierarchy composed of interconnected network nodes, which is stacked on top of a (2) computing nodes hierarchy [6]. This compute hierarchy can be composed for instance of a shared memory NUMA multicore platform.

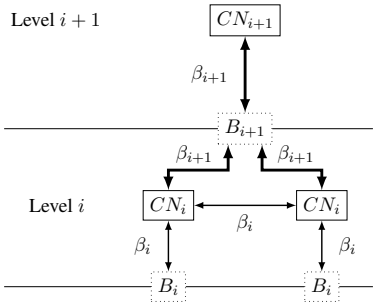


Figure 1: Components of a level i in the HCP model.

Our 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). An intermediate level $i > 1$ and its components, as depicted in Figure 1, have the following characteristics. A compute node of level $i+1$, denoted as CN_{i+1} in the figure, is formed by P_i compute nodes of level i (two nodes in our example). These P_i compute nodes are organized along a 2D grid topology, that is $P_i = P_{r_i} \times P_{c_i}$. The total number of compute nodes of the entire platform is $P = \prod_{i=1}^{l} P_i$, while the total number of compute nodes of level i is denoted $P_i^* = \prod_{j=i}^{l} P_j$. We let $M_i = M_1 \cdot \prod_{j=1}^{i-1} P_j$ be the aggregated memory size of a node of level $i > 1$, where M_1 denotes the memory size of a processing element of level 1. The network latency α_i and the inverse bandwidth β_i apply throughout an entire level i, however the higher in the hierarchy, the more important communication costs become. We also consider a network buffer B_i at each level of the hierarchy. This allows to take into account the possibility of message aggregation for network communication, and determine the number of messages required to send a given amount of data, thus the latency cost associated with each communication at every level of the hierarchy. These notations will be used throughout the rest of the paper. In addition, we refer to the number of messages sent by a node of level i as S_i, and to the exchanged volume of data as W_i. $\bar{S}_i$ is the latency cost at level i, $\bar{S}_i = S_i \cdot \alpha_i$. Similarly, $\bar{W}_i$ is the bandwidth cost, $\bar{W}_i = W_i \cdot \beta_i$.

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 performance predicted by such a model might not be extremely 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 it will also allow us to understand the communication bottlenecks of linear algebra operations.

Communicating under the HCP model. We now describe how communication happens in the HCP model, and how messages are routed between different levels of the network hierarchy. First, we assume that if a compute node of level i communicates, all the nodes below it participates. We denote as ***counterparts*** of a compute node of level i all the nodes of level i lying in remote compute nodes of level $i+1$ 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 taking place between two compute nodes of a given level i. A total of P/P_i^* processing elements of level 1 are involved in sending a global volume of data W_i. Each of these elements has to send a chunk of data $W_1 = W_i P/P_i^*$. Since this amount of data has to fit in the memory of a processing element of level 1, 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 network buffer size at level 2 is B_2, this requires (W_2/B_2) messages. The same holds for any level k such that $1 < k \leq i$, where data is forwarded by sending(W_k/B_k) messages. We therefore have the following costs:

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

Network types. We assume three kinds of networks, depending on their respective buffer size:

1. *fully-pipelined networks*, able to aggregate all incoming messages into a single message. This requires that $B_i \geq M_i$, since at most M_i words of data can be sent.
   The *fully-pipelined network* case is ensured whenever $B_i \geq P_{i-1} W_{i-1}$. Since M_i is the size of the largest message sent at level i, we assume $B_i = M_i$. We also assume that all levels below a fully-pipelined level are themselves fully-pipelined. Therefore, the constraint on the buffer size becomes $B_i = M_i = P_{i-1} B_{i-1}$.
2. *bufferized networks*, allowing for message aggregation up to a buffer size $B_i < M_i$.
3. *forward networks*, where messages coming from lower level are simply forwarded to higher levels.
   For a given level i, a *forward network* requires that $B_i = B_{i-1}$. Indeed, when all the sub-nodes from level $i-1$ send S_{i-1} messages each, 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 buffer size B_i to satisfy $B_{i-1} \leq B_i \leq P_{i-1} B_{i-1}$.

Lower bounds on communication. 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 buffer size B_i of the considered level i, where a volume $\bar{W}_i$ needs to be sent in messages of size B_i. Hence the lower bounds on communications at level i:

$$\bar{W}_i = \Omega\left(\frac{n^2}{\sqrt{P_i^*}} \cdot \beta_i\right) \quad (1) \qquad \bar{S}_i = \Omega\left(\frac{n^2}{B_i \sqrt{P_i^*}} \cdot \alpha_i\right) \quad (2)$$

Note that, for simplicity, we expressed the bound on latency with respect to B_i for all level i. Since we consider $B_1 = M_1$, the lower bound on latency for level 1 can also be expressed as $\bar{S}_1 = \Omega\left(\sqrt{P}\right)$.

3 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 relative 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 tailored for hierarchical platforms, thus reducing the communication needs at each level of the hierarchy.

3.1 Multilevel QR factorization

The QR factorization is a widely used algorithm. Example of use are numerous, be it for orthogonalizing a set of vectors or for solving least squares problems. It decomposes a matrix A into two matrices Q and R such that $A = QR$, where Q is orthogonal and R is upper triangular. The decomposition is obtained by using very stable transformations, such as Householder reflections. We assume in the following that the matrix Q is not stored explicitly, but rather using the compact YTY^T representation [16].

ML-CAQR, given in Algorithm 1, is a multilevel tree-based algorithm computing the QR factorization of a matrix using Householder reflections. It is tailored for hierarchical platforms. Moreover, as ML-CAQR is a tree-based algorithm, these Householder reflectors are stored in the lower triangular part of matrix A using a tree structure as in [7].

As CAQR on platforms with one level of parallelism, ML-CAQR aims at reducing the amount of communication required during the factorization, but at each level of parallelism of a hierarchical platform. At the topmost level of the hierarchy, ML-CAQR processes the entire input matrix A panel by panel. Each panel is first factored by a recursive call to ML-CAQR on the next lower level. The Householder reflectors are then sent to remote compute nodes and the trailing matrix updated using two recursive routines: ML-UPFACT and ML-UPELIM.

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

```
if r = 1 then
    Call CAQR(A, P)
else
    for kk = 1 to n/b_r do
        for Processor p = 1 to P_{r_r} in parallel do
            h_p = (m − (kk − 1)b_r)/P_{r_r}
            panel = A(kk · b_r + (p − 1)h_p : kk · b_r + ph_p, kk · b_r : (kk + 1) · b_r)
            Call ML-CAQR(panel, h_p, b_r, r − 1, p)
        for j = 1 to log P_{r_r} do
            (p_source, p_target) is the pair of processors with which this elimination is performed.
            Send local b_r-by-b_r to the remote processor 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_source)
            Call ML-CAQR(RR, 2b_r, b_r, r − 1, p_target)
        for Processor p = 1 to P_{r_r} in parallel do
            Broadcast the sets of Householder vectors to every processor belonging to the same processor row
            for Processor pp = 2 to P_{c_r} on same row than p in parallel do
                Call ML-UPFACT(r − 1, pp)
        for j = 1 to log P_{r_r} do
            (p_source, p_target) is the pair of processors with which this elimination was performed.
            for Processor pp = 2 to P_{c_r} on same row than p_source in parallel do
                pp_target is the remote processor on the same row than p_target and in the same column than pp
                pp sends its local C to pp_target
                Call ML-UPELIM(r − 1, pp)
                Call ML-UPELIM(r − 1, pp_target)
```

More precisely, for each recursion level r, let b_r be the block size, $m_s^{(r)} = (m_s^{(r+1)}/P_{r_{r+1}} - (s-1)b_r)$ be the panel row count at step s, and $n_s^{(r)} = (b_{r+1} - sb_r)$ 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 $n_s^{(l)} = (n - sb_l)$. 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, the computation becomes:
   (a) First P_{r_r} subsets of the panel, of size $m_s^{(r)}/P_{r_r}$-by-b_r, are recursively factored with ML-CAQR (with a block size b_{r-1} corresponding to the next level deeper in the hierarchy). At the deepest level of recursion, ML-CAQR calls the CAQR algorithm.
   (b) The resulting b_r-by-b_r R factors are eliminated two-by-two using an elimination tree by multiple calls to ML-CAQR, requiring $\log P_{r_r}$ steps along the critical path.
2. The current trailing matrix is then updated to reflect the factorization of the panel:
   (a) Updates corresponding to factorizations at the leaves of the tree are applied using the ML-UPFACT routine. This routine is called in parallel on P_{r_r} blocks rows of size $m_s^{(r)}/P_{r_r}$-by-$n_s^{(r)}$.

ML-UPFACT broadcasts P_{r_r} blocks of Householder's reflectors of size $m_s^{(r)}/P_{r_r}$-by-b_r from the column of nodes holding current panel along rows of compute nodes. At the deepest level, the update corresponding to a leaf is applied as in CAQR (see [8]).

(b) Finally, the updates due to the eliminations of the intermediate R factors are applied onto the trailing matrix using the ML-UPELIM 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 then computed locally before being applied independently onto each processing elements, similarly to CAQR

3.2 Multilevel LU factorizations

LU factorization is the 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 $PA = LU$, where P is a permutation matrix required for numerical stability reasons. We present two variants of a multilevel algorithm, ML-CALU, for computing the LU factorization of a dense matrix. ML-CALU is a recursive algorithm. The first variant, 1D-ML-CALU is a uni-dimensional approach where the entire panel is processed by a single recursive call. The second variant, 2D-ML-CALU, processes a panel by multiple recursive calls on sub-panels followed by a "reduction" phase similar to that of ML-CAQR. The base case of both recursive variants is CALU [13], which uses tournament pivoting to select pivots.

In the case of 1D-ML-CALU, the algorithm proceeds as follows at each recursion level r. (1) 1D-ML-CALU is recursively applied to an entire panel of b_r columns, only the number of compute nodes along the columns 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 reader to Algorithm 2 in Appendix A. 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. Algorithm 3 in Appendix B describes in details 1D ML-CALU 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 [9] and analyzed for two-levels platforms. Here we extend the analysis of Algorithm 4, given in Appendix B, to deeper platforms. 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 the 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) Then the computed permutation is 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.

4 Performance models

In this section, we provide a cost analysis of both ML-CAQR and ML-CALU within the HCP model. We first analyse two recursive communication routines which are used by all multilevel algorithms presented in

this study, namely the point to point communication and the broadcast operations.

Point to point communication of a volume D of data between two compute nodes of level r involves compute nodes from level 1 to level r. At every level, subnodes send their local data to their counterparts in the remote node of level r. The associated communication costs are:

$$W_{\text{RCOMM}}(1\dots r, D) = \sum_{k=1}^{r} \frac{D \cdot P_r^*}{P_k^*}\beta_k \ , \qquad S_{\text{RCOMM}}(1\dots r, D) = \alpha_1 + \sum_{k=2}^{r} \frac{D \cdot P_r^*}{B_k P_k^*}\alpha_k \ .$$

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

4.1 ML-CAQR

We now review the cost of ML-CAQR in terms of computations as well as communications. At each recursion level r, parameters are adapted to a grid of P_{r_r}-by-P_{c_r} compute nodes. The current panel is first factored by doing P_{r_r} parallel calls to ML-CAQR at the leaves of the tree. Then, the resulting R factors are eliminated through $\log P_{r_r}$ successive factorizations of a $2b_r$-by-b_r matrix formed by stacking up two upper triangular R factors. Once a panel is factored, the trailing matrix is updated. However, as the Householder's reflectors are stored in a tree structure, like in [8], the updates must be done by going through each level of the tree again. These operations are recursively performed using ML-UPFACT for the leaves and ML-UPELIM for higher levels.

Global recursive cost of ML-CAQR. We define the following contributions to the global cost of ML-CAQR: $T_{\text{CAQR}}\,(m, n, b, P)$ is the cost of factoring a matrix of size m-by-n with CAQR using P processors and a block size b. $T_{\text{ML-CAQR}}\,(m, n, b, P)$ is the cost of ML-CAQR on a m-by-n matrix using P processors and a block size b. $T_{\text{ML-UPFACT}}\,(m, n, b, P)$ is the cost of updating the trailing matrix to reflect factorizations at the leaves of the elimination trees. $T_{\text{ML-UPELIM}}\,(m, n, b, P)$ is the cost of applying updates corresponding to higher levels in the trees. In terms of communication, ML-UPFACT consists in broadcasting Householder reflectors along process rows, while ML-UPELIM corresponds to $\log P_{r_r}$ point to point communications of trailing matrix blocks between pairs of nodes. Using these notations, the cost of ML-CAQR can be recursively expressed as:

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

Bounding cost of ML-CAQR. ML-CAQR uses successive elimination trees at each recursion depth r, each completed in $\log P_{r_r}$ steps. As successive trees from level l down to level r come from different recursive calls, they are sequentialized. Thus, the total number of calls at a given recursion depth r can be upper-bounded by $N_r = 2^{l-r}\prod_{j=r}^{l} \log P_{r_j}$. The global cost of ML-CAQR can therefore 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. Details on this cost is given in Appendix C. Assuming that for each level k, we have $P_{r_k} = P_{c_k} = \sqrt{P_k}$, and block sizes 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 is:

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

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

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

Altogether, 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 [11], it allows to reach the lower bound at all levels of the hierarchy up to polylogarithmic factors. Indeed, chosing appropriate block sizes makes most of the extra computational costs lower order while maintaining the optimality in terms of communications.

4.2 2D multilevel 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. We note that we use the same reasoning as ML-CAQR to derive the recursive cost of ML-CALU, and the same approach and approximations to estimate its total cost. Thus for a square n-by-n matrix and using l levels of recursion ($l \geq 2$) the cost of ML-CALU is:

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

$$\bar{W}_{\text{ML-CALU}}(n,n) \leq \left[\frac{n^2}{2\sqrt{P}}\log P_1\prod_{j=2}^{l}(1+\frac{1}{2}\log P_j)\right]\beta_1 \tag{8}$$

$$+\sum_{k=1}^{l}\Big[\frac{n^2}{\sqrt{P_k^*}}\Big(\frac{8}{3}\log^2 P_l(1+\frac{l-k}{\sqrt{P_k}}) + \frac{(l-2)}{8}(1+\frac{l}{4})\prod_{j=3}^{l}(1+\frac{1}{2}\log P_j)\Big)\Big]\beta_k.$$

$$\bar{S}_{\text{ML-CALU}}(n,n) \leq \left[\frac{n^2}{2\sqrt{P}}\log P_1\prod_{j=2}^{l}(1+\frac{1}{2}\log P_j)\right]\alpha_1 \tag{9}$$

$$+\sum_{k=1}^{l}\Big[\frac{n^2}{B_k\sqrt{P_k^*}}\Big(\frac{8}{3}\log^2 P_l(1+\frac{l-k}{\sqrt{P_k}}) + \frac{(l-2)}{8}(1+\frac{l}{4})\prod_{j=3}^{l}(1+\frac{1}{2}\log P_j)\Big)\Big]\alpha_k$$

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.

In terms of communications, we can conclude that ML-CALU attains the lower bounds derived in section 2 modulo a factor that depends on $l^2\prod_{j=2}^{l}\log P_j$ at each level k of hierarchy. Thus it reduces the communication cost at each level of a hierarchical system. Note that in practice the number of levels is going to remain small, while the number of processors will be large. We note that we do not give the detailed cost of 1D-ML-CALU here. However we would like to point that it attains the lower bounds derived under the HCP model in terms of bandwidth at each level of parallelism. However in terms of latency the lower bound is only met at the deepest level of parallelism.

5 Experimental results

5.1 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 $\|PA - LU\|/\|A\|$.

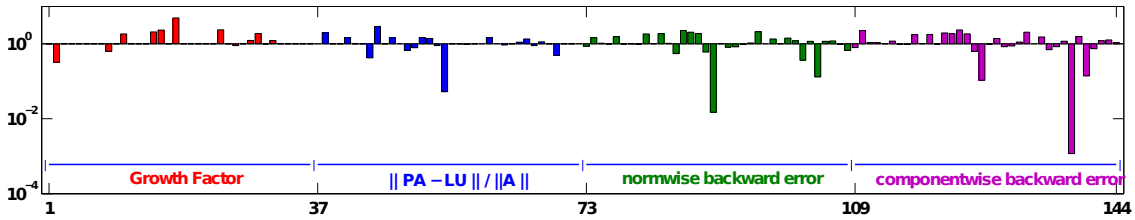


Figure 2: Ratios of 3-level CALU's growth factor and backward errors to GEPP's.

Figure 2 displays the values of the ratios of 3-level CALU's growth factor and backward errors to those of GEPP for 36 special matrices [13]. 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.

We note that in most cases, ML-CALU uses tournament pivoting to select pivots at each level of the recursion, which does not ensure that the element of maximum magnitude in the column is used as pivot, neither at each 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 both binary tree based 2-level CALU and binary tree based 3-level CALU. For example, for binary tree based 3-level CALU, 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.

5.2 Performance predictions

In this section, 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.

We model the platform with respect to the HCP model, and use it to estimate the running times of our algorithms. As exascale platforms are not available yet, we base our sample exascale platform on the characteristics of NERSC Hopper [2, 1], a petascale platform planned to reach the Exascale around year 2018. 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* [18, 12]. Detailed parameters of the Hopper platform are presented in Table 1.

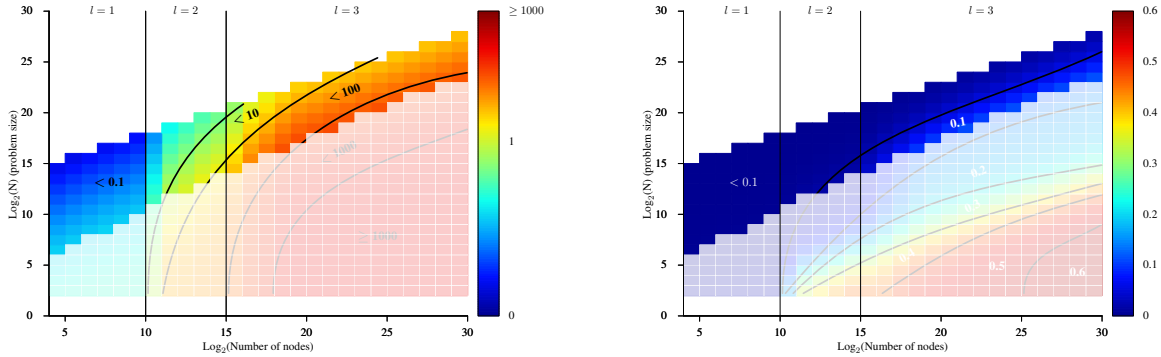
The target exascale platform is obtained by increasing the number of nodes at all 3 levels, leading to a total of $1M$ nodes. The amount of memory per processing element is kept constant at 1.3 GB, while latencies and bandwidths are derived using an average 15% decrease per year for the latency and a 26% increase for the bandwidth [12, 18]. These parameters are detailed in Table 1.

	NERSC Hopper				Exascale platform			
Level	Type	#	Bandwidth	Latency	Type	#	Bandwidth	Latency
1	2x 6-cores Opterons	12	19.8 GB/s	1×10^{-9}s	Multi-cores	1024	300 GB/s	1×10^{-10}s
2	Hopper nodes	2	10.4 GB/s	1×10^{-6}s	Nodes	32	150 GB/s	1×10^{-7}s
3	Gemini ASICS	9350	3.5 GB/s	1.5×10^{-6}s	Interconnects	32768	50 GB/s	1.5×10^{-7}s

Table 1: Characteristics of NERSC Hopper and sample exascale platform.

Moreover, in order to assess the performance of multilevel algorithms, costs of state-of-the-art 1-level communication avoiding algorithms need to be expressed in the 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 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 bounded.



(a) 1-level CAQR

(b) ML-CAQR

Figure 3: Prediction of communication to computation ratio on an exascale platform.

Performance predictions of ML-CAQR 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 Figure 3, 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. As the number of processing elements increases, cost of CAQR (on Figure 3a) is dominated by communication. Our multilevel approach alleviates this trend, and ML-CAQR (on Figure 3b) 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 perform more computations than CAQR, we compare the expected running times of both algorithms. Here by running time, we denote the sum between computational and communication costs. We thus assume no overlap between computations and communications. The ratio of the ML-CAQR running time over CAQR is depicted on Figure 4. ML-CAQR clearly outperforms CAQR when using the entire platform, despite its higher computational costs. As a matter of a fact in this domain,

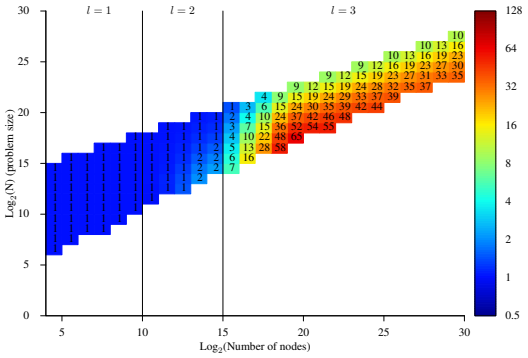


Figure 4: Speedup of ML-CAQR vs. CAQR

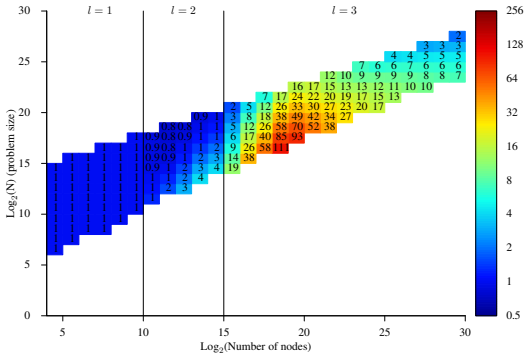


Figure 5: Speedup of ML-CALU vs. CALU

the running time is dominated by the bandwidth cost, and ML-CAQR significantly reduces it at all levels.

Performance predictions of ML-CALU The same observations can be made on the *CCR* of CALU and ML-CALU, we will therefore not present the detail here. Regarding the running times ratio, depicted on Figure 5, 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 communications into consideration at all levels, ML-CAQR and ML-CALU deliver a high level of performance, even at scales where performance is hindered by communication costs with 1-level communication avoiding algorithms.

6 Conclusion

In this paper we have introduced two algorithms, ML-CALU and ML-CAQR, that minimize communication over multiple levels of parallelism at the cost of performing redundant computation. The complexity analysis is performed by using HCP, a model that takes into account the cost of communication 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 of an exascale platform show that for strong scaling, the multilevel algorithms lead to important speedups compared to the algorithms which minimize communication over only one level of parallelism. In most of the cases, minimizing bandwidth is the key factor for improving scalability, and hence the 1D ML-CALU is an appropriate choice for both ensuring numerical stability and being efficient in practice.

References

[1] http://www.nersc.gov/assets/Uploads/ShalfXE6ArchitectureSM.pdf.

[2] Hopper. http://www.nersc.gov/users/computational-systems/hopper/.

[3] Emmanuel Agullo, Camille Coti, Jack Dongarra, Thomas Herault, and Julien 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.

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

[5] George Bosilca, Aurelien Bouteiller, Anthony Danalis, Mathieu Faverge, Azzam Haidar, Thomas Herault, Jakub Kurzak, Julien Langou, Pierre Lemarinier, Hatem Ltaief, Piotr Luszczek, Asim YarKhan, and Jack 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.

[6] F. Cappello, P. Fraigniaud, B. Mans, and A.L. Rosenberg. Hihcohp-toward a realistic communication model for hierarchical hyperclusters of heterogeneous processors. In Parallel and Distributed Processing Symposium., Proceedings 15th International, apr 2001.

[7] 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.

[8] James W. Demmel, Laura Grigori, Mark F. Hoemmen, and Julien Langou. Communication-optimal parallel and sequential QR and LU factorizations. LAPACK Working Note 204, August 2008.

[9] Simplice Donfack, Laura Grigori, and Amal Khabou. Avoiding Communication through a Multilevel LU Factorization. In Euro-Par, pages 551–562, 2012.

[10] Jack Dongarra, Mathieu Faverge, Thomas Herault, Mathias Jacquelin, Julien Langou, and Yves Robert. Hierarchical qr factorization algorithms for multi-core clusters. Parallel Computing, (0):–, 2013.

[11] J.D. Frens and D.S. Wise. Qr factorization with morton-ordered quadtree matrices for memory re-use and parallelism. In ACM SIGPLAN Notices, volume 38, pages 144–154. ACM, 2003.

[12] S.L. Graham, M. Snir, C.A. 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.

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

[14] 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, pages 326–333, New York, NY, USA, 1981. ACM.

[15] D. Irony, S. Toledo, and A. Tiskin. Communication lower bounds for distributed-memory matrix multiplication. J. Parallel Distrib. Comput., 64(9):1017–1026, 2004.

[16] R. Schreiber and C. Van Loan. A storage efficient WY representation for products of Householder transformat ions. SIAM J. Sci. Stat. Comput., 10(1):53–57, 1989.

[17] Fengguang Song, Hatem Ltaief, Bibel Hadri, and Jack 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.

[18] Peter Kogge (Editor & study lead). Exascale computing study: Technology challenges in achieving exascale systems, 2008. DARPA IPTO Technical Report.

A Appendix: ML-CANNON

To perform matrix multiplication $C = C + A \cdot B$, we can use Cannon's algorithm. It is known that (see for example [4]) assuming minimal memory usage, Cannon's algorithm is asymptotically optimal in terms of both bandwidth, and latency, that is the volume of data and the number of messages send by each node are asymptotically optimal. In the following we present a recursive Cannon algorithm over a hierarchy of nodes. We refer to this algorithm as ML-CANNON, and we consider l levels of parallelism. Algorithm 2 presents the communication details of ML-CANNON.

Algorithm 2: ML-CANNON (C, A, B, P_k, k)

Input: three square matrices C, A, and B, k the number of recursion level, P_k the number of processors at level k
Output: $C = C + AB$
if $k == 1$ **then**
  $CANNON(C, A, B, P_1)$
else
  for $i = 1$ to $\sqrt{P_k}$ **in parallel do**
    left-circular-shift row i of A by i
  for $i = 1$ to $\sqrt{P_k}$ **in parallel do**
    up-circular-shift column i of B by i
  for $h = 1$ to $\sqrt{P_k}$ (sequential) **do**
    for $i = 1$ to $\sqrt{P_k}$ and $j = 1$ to $\sqrt{P_k}$ **in parallel do**
      $ML - CANNON(C(i,j), A(i,j), B(i,j), P_{k-1}, k-1)$
      left-circular-shift row i of A by 1
      up-circular-shift column i of B by 1

B Appendix: 1D-ML-CALU and 2D-ML-CALU

1D multilevel CALU. 1D ML-CALU is described in Algorithm 3. It is applied to a matrix A of size $m \times n$, using l levels of recursion and a total number of $P = \prod_{i=1}^{l} P_i$ compute nodes of level 1 at the topmost level of the recursion. These compute nodes are organized as a two-dimensional grid at each level, $P_i = P_{r_i} \times P_{c_i}$.

Algorithm 3: 1D multilevel communication avoiding LU factorization

Input: $m \times n$ matrix A, the recursion level l, block size b_l, the total number of compute units $P = \prod_{i=1}^{l} P_i = P_r \times P_c$
if $l == 1$ **then**
  $[\Pi_1, L_1, U_1] = \text{CALU}(A, b_1, P_r \times P_{c_1})$
else
  $M = m/b_l, N = n/b_l$
  for $K = 1$ to N **do**
    $[\Pi_{KK}, L_{K:M,K}, U_{KK}] = Recursive - \text{CALU}(A_{K:M,K}, l-1, b_{l-1}, P_r \times \prod_{i=1}^{l-1} P_{c_i})$
    /* Apply permutation and compute block row of U */
    $A_{K:M,:} = \Pi_{KK} A_{K:M,:}$
    for each compute unit at level l owning a block $A_{K,J}$, $J = K+1$ to N **in parallel do**
      $U_{K,J} = L_{KK}^{-1} A_{K,J}$
      /* call multilevel dtrsm using $P_{c_l} \times \prod_{i=1}^{l-1} P_i$ processing node of level 1 */
    /* Update the trailing submatrix */
    for each compute unit at level l owning a block $A_{I,J}$ of the trailing submatrix, $I, J = K+1$ to M, N **in parallel do**
      $A_{I,J} = A_{I,J} - L_{I,K} U_{K,J}$
      /* call multilevel dgemm using $P_r \times \prod_{i=1}^{l} P_{c_i}$ processing node of level 1 */

2D multilevel CALU. The 2D ML-CALU algorithm, given in Algorithm 4, details the communication performed during the factorization.

Algorithm 4: 2D multilevel communication avoiding LU factorization

Input: $m \times n$ matrix A, level of parallelism l in the hierarchy, block size b_l, number of nodes $P_l = P_{r_l} \times P_{c_l}$
if $l = 1$ **then**
  Call CALU$(A, m, n, 1, P_1)$
else
  for $k = 1$ to n/b_l **do**
    $m_p = (m - (k-1)b_l)/P_{r_l}$
    $n_p = (n - (k-1)b_l)/P_{c_l}$
    /* factor leaves of the panel */
    for Processor $p = 1$ to P_{r_l} **in parallel do**
      leaf $= A((k-1) \cdot b_l + (p-1)m_p + 1 : (k-1) \cdot b_l + p \cdot m_p, (k-1) \cdot b_l + 1 : k \cdot b_l)$
      Call ML-CALU(leaf, $m_p, b_l, l-1, P_{l-1}$)
    /* Reduction steps */
    for $j = 1$ to $\log P_{r_l}$ **do**
      Stack two b_l-by-b_l sets of candidate pivot rows in B
      Call ML-CALU$(B, 2b_l, b_l, l-1, P_{l-1})$
    /* Compute block column of L */
    for Processor $p = 1$ to P_{r_l} **in parallel do**
      Compute $L_{p,k} = L(k \cdot b_l + (p-1)m_p + 1 : k \cdot b_l + p \cdot m_p, (k-1) \cdot b_l + 1 : k \cdot b_l)$
      /* $(L_{p,k} = L_{p,k} \cdot U_{k,k}^{-1})$ using multilevel algorithm with P_{l-1} nodes at level $(l-1)$ */
    /* Apply all row permutations */
    for Processor $p = 1$ to P_{r_l} **in parallel do**
      Broadcast pivot information along the rows of the process grid
      /* all to all reduce operation using P_l processors of level l */
      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 $p = 1$ to P_{c_l} **in parallel do**
      Compute $U_{k,p} = U((k-1) \cdot b_l + 1 : k \cdot b_l, k \cdot b_l + (p-1)n_p + 1 : k \cdot b_l + p \cdot n_p)$
      /* $(U_{k,p} = L_{k,k}^{-1} \cdot A_{k,p}^{-1})$ using multilevel algorithm with P_{l-1} nodes at level $(l-1)$ */
    /* Update trailing matrix */
    for Processor $p = 1$ to P_{c_l} **in parallel do**
      Broadcast $U_{k,p}$ along the columns of the process grid
    for Processor $p = 1$ to P_{r_l} **in parallel do**
      Broadcast $L_{p,k}$ along the rows of the process grid
    for Processor $p = 1$ to P_l **in parallel do**
      $A(k \cdot b_l + (p-1)m_p + 1 : k \cdot b_l + p \cdot m_p, k \cdot b_l + (p-1)n_p + 1 : k \cdot b_l + p \cdot n_p) =$
      $A(k \cdot b_l + (p-1)m_p + 1 : k \cdot b_l + p \cdot m_p, k \cdot b_l + (p-1)n_p + 1 : k \cdot b_l + p \cdot n_p) - L_{p,k} \cdot U_{k,p}$
      /* using multilevel Cannon with P_l nodes at level l */

C Appendix: detailed upper bound on the cost of ML-CAQR

The global recursive cost of ML-CAQR can be upper bounded by:

$$
\begin{aligned}
&T_{\text{ML-CAQR}}\left(m,n,b_r,P_r\right) \leq \\
&\left[\begin{array}{l}
\frac{n}{b_2}\cdot T_{\text{ML-CAQR}}\left(\frac{m}{\sqrt{P_2^*}},b_2,b_1,P_1\right) \\
\quad +\sum_{r=2}^{l}\frac{n}{b_r}\cdot T_{\text{RBCAST}}(1\ldots r,\frac{mb_r}{\sqrt{P_r^*}})+\frac{n}{b_2}\cdot T_{\text{ML-UPFACT}}\left(\frac{m}{\sqrt{P_2^*}},\frac{n}{\sqrt{P_2^*}},b_1,P_1\right)
\end{array}\right] \\
&+\sum_{r=2}^{l-1}\frac{n\cdot N_r}{b_r}\left[\begin{array}{l}
T_{\text{RCOMM}}(1\ldots r,\frac{b_r^2}{2})+\frac{b_r}{b_2}\cdot T_{\text{ML-CAQR}}\left(\frac{2b_r}{\prod_{j=2}^{r-1}\sqrt{P_j}},b_2,b_1,P_1\right) \\
\quad +T_{\text{RBCAST}}(1\ldots r,b_r^2)+\frac{b_r}{b_2}\cdot T_{\text{ML-UPFACT}}\left(\frac{b_r}{\prod_{j=2}^{r-1}\sqrt{P_j}},\frac{n}{\sqrt{P_2^*}},b_1,P_1\right) \\
\quad\quad +T_{\text{RCOMM}}(1\ldots r,\frac{b_r n}{\sqrt{P_r^*}})+\frac{b_r}{b_2}\cdot T_{\text{ML-UPELIM}}\left(\frac{2b_r}{\prod_{j=2}^{r-1}\sqrt{P_j}},\frac{n}{\sqrt{P_2^*}},b_1,P_1\right)
\end{array}\right] \\
&+\frac{n\cdot N_l}{b_l}\left[\begin{array}{l}
T_{\text{RCOMM}}(1\ldots r,\frac{b_l^2}{2})+\frac{b_l}{b_2}\cdot T_{\text{ML-CAQR}}\left(\frac{2b_l}{\prod_{j=2}^{l-1}\sqrt{P_j}},b_2,b_1,P_1\right) \\
\quad +T_{\text{RCOMM}}(1\ldots r,\frac{b_l n}{\sqrt{P_l}})+\frac{b_l}{b_2}\cdot T_{\text{ML-UPELIM}}\left(\frac{2b_l}{\prod_{j=2}^{l-1}\sqrt{P_j}},\frac{n}{\sqrt{P_2^*}},b_1,P_1\right)
\end{array}\right]
\end{aligned}
$$