Calculs pour les matrices denses : coût de communication et stabilité numérique

Amal Khabou

N° d'ordre : *N° attribué par la bibliothèque :*

- UNIVERSITÉ PARIS-SUD -
Laboratoire de Recherche en Informatique - UMR 427 - LRI

THÈSE

en vue d'obtenir le grade de

Docteur de l'Université Paris-Sud

Spécialité : Informatique

au titre de l'École Doctorale Informatique Paris-Sud

présentée et soutenue publiquement le 11 Février 2013 par

Amal KHABOU

Dense matrix computations : communication cost and numerical stability.

Directeur de thèse :	Docteur	Laura GRIGORI	
Après avis de :	Professeur	Nicholas J. HIGHAM	*Rapporteur*
	Professeur	Yves ROBERT	*Rapporteur*
Devant la commission d'examen formée de :			
	Professeur	Iain DUFF	*Examinateur*
	Professeur	Yannis MANOUSSAKIS	*Examinateur*
	Enseignant-chercheur	Jean-Louis ROCH	*Examinateur*

Abstract:
This dissertation focuses on a widely used linear algebra kernel to solve linear systems, that is the LU decomposition. Usually, to perform such a computation one uses the Gaussian elimination with partial pivoting (GEPP). The backward stability of GEPP depends on a quantity which is referred to as the growth factor, it is known that in general GEPP leads to modest element growth in practice. However its parallel version does not attain the communication lower bounds. Indeed the panel factorization represents a bottleneck in terms of communication. To overcome this communication bottleneck, Grigori et al [60] have developed a communication avoiding LU factorization (CALU), which is asymptotically optimal in terms of communication cost at the cost of some redundant computation. In theory, the upper bound of the growth factor is larger than that of Gaussian elimination with partial pivoting, however CALU is stable in practice. To improve the upper bound of the growth factor, we study a new pivoting strategy based on strong rank revealing QR factorization. Thus we develop a new block algorithm for the LU factorization. This algorithm has a smaller growth factor upper bound compared to Gaussian elimination with partial pivoting. The strong rank revealing pivoting is then combined with tournament pivoting strategy to produce a communication avoiding LU factorization that is more stable than CALU. For hierarchical systems, multiple levels of parallelism are available. However, none of the previously cited methods fully exploit these hierarchical systems. We propose and study two recursive algorithms based on the communication avoiding LU algorithm, which are more suitable for architectures with multiple levels of parallelism. For an accurate and realistic cost analysis of these hierarchical algorithms, we introduce a hierarchical parallel performance model that takes into account processor and network hierarchies. This analysis enables us to accurately predict the performance of the hierarchical LU factorization on an exascale platform.

Keywords:
LU factorization, Gaussian elimination with partial pivoting, growth factor, strong rank revealing QR factorization, minimizing the communication cost, parallel algorithms, hierarchical systems, performance models, pivoting strategies.

Résumé :

Cette thèse traite d'une routine d'algèbre linéaire largement utilisée pour la résolution des systèmes linéaires, il s'agit de la factorisation LU. Habituellement, pour calculer une telle décomposition, on utilise l'élimination de Gauss avec pivotage partiel (GEPP). La stabilité numérique de l'élimination de Gauss avec pivotage partiel est caractérisée par un facteur de croissance qui est reste assez petit en pratique. Toutefois, la version parallèle de cet algorithme ne permet pas d'atteindre les bornes inférieures qui caractérisent le coût de communication pour un algorithme donné. En effet, la factorisation d'un bloc de colonnes constitue un goulot d'étranglement en termes de communication. Pour remédier à ce problème, Grigori et al [60] ont développé une factorisation LU qui minimise la communication(CALU) au prix de quelques calculs redondants. En théorie la borne supérieure du facteur de croissance de CALU est plus grande que celle de l'élimination de Gauss avec pivotage partiel, cependant CALU est stable en pratique. Pour améliorer la borne supérieure du facteur de croissance, nous étudions une nouvelle stratégie de pivotage utilisant la factorisation QR avec forte révélation de rang. Ainsi nous développons un nouvel algorithme pour la factorisation LU par blocs. La borne supérieure du facteur de croissance de cet algorithme est plus petite que celle de l'élimination de Gauss avec pivotage partiel. Cette stratégie de pivotage est ensuite combinée avec le pivotage basé sur un tournoi pour produire une factorisation LU qui minimise la communication et qui est plus stable que CALU. Pour les systèmes hiérarchiques, plusieurs niveaux de parallélisme sont disponibles. Cependant, aucune des méthodes précédemment citées n'exploite pleinement ces ressources. Nous proposons et étudions alors deux algorithmes récursifs qui utilisent les mêmes principes que CALU mais qui sont plus appropriés pour des architectures à plusieurs niveaux de parallélisme. Pour analyser d'une façon précise et réaliste les coûts de ces algorithmes hiérarchiques, nous introduisons un modèle de performance hiérarchique parallèle qui prend en compte aussi bien l'organisation hiérarchique des processeurs et la hiérarchie réseau. Cette analyse nous permet de prédire d'une manière réaliste la performance de ces factorisations hiérarchiques sur une plate-forme exascale.

Mots-clés :

Factorisation LU, élimination de Gauss avec pivotage partiel, facteur de croissance, factorisation QR avec forte révélation de rang, minimisation de la communication, algorithmes parallèles, systèmes hiérarchiques, modèles de performance, stratégies de pivotage.

Contents

List of Figures

List of Tables

Introduction

In many computational applications dense linear algebra represents an essential and fundamental part. Hence for several scientific and engineering applications, the core phase of resolution consists of solving a linear system of equations $Ax = b$. For that reason the dense linear algebra algorithms have to be numerically stable, robust, and accurate. Furthermore these applications are more and more involving large linear systems. Thus the developed algorithms must be scalable and efficient on massively parallel computers with multi-core nodes. This is a part of High performance Computing (HPC), which represents a wide research area that includes the design and the analysis of parallel algorithms to be efficient and scalable on massively parallel computers. The principal reason behind the significant and increasing interest in HPC is the widespread adoption of multi-core processor systems. Thus designing high performance algorithms for dense matrix computations is a very demanding and challenging task, especially to be well adapted to the multi-core architectures. These architectures ranging from multicore processors to petascale and exascale platforms offer an increasing computing power. However it is not always easy to efficiently use and fully exploit this power, since in addition to the issues related to the development of sequential algorithms, one has to take into consideration both the opportunities and the limitations of the underlying hardware on which the parallel algorithms are executed. In this work we focus on the LU decomposition with respect to two directions. The first direction is the numerical stability, which mainly aims at developing accurate algorithms and the second direction is the communication issue, which if it is taken into account at the design level leads to efficient and scalable algorithms. Thus these two axis are important to achieve the goals presented above. Note that we study both sequential and parallel algorithms.

It was shown by Wilkinson that much insight into the behavior of numerical algorithms can be gained through studying their backward stability [96]. This observation has lead to a significant advance in numerical analysis. Considering the LU decomposition, a stability issue is that the Gaussian elimination with partial pivoting, although it is stable in practice, is not backward stable. To quantify the numerical stability of Gaussian elimination with partial pivoting, one should consider a quantity referred to as the growth factor g_W, which measures how large the entries of the input matrix are getting during the elimination steps. Wilkinson showed that for a square matrix of size $n \times n$, the growth factor is upper bounded by 2^{n-1}, and that this upper bound is attainted for a small class of matrices. However extensive experiment sets showed that it is still small in practice. Understanding such a behavior remains since decades an unsolved linear algebra problem. The previous example shows how challenging is the design of numerical stable algorithms.

The second issue we consider throughout this work is communication, which refers to the data movement between processors in the parallel case and between memory levels (typically main and fast memory) in the sequential case. This problem was addressed in prior work [69, 72, 15]. Due to the increasing number of processors in actual machines, running a classical algorithm on such a system

might significantly reduce its performance. The processors are indeed spending much more time communicating than performing actual computations. Based on this observation, it becomes necessary to design new algorithms or rearrange the classical ones in order to reduce the amount of communication and thus be more adapted for exascale machines. An attempt to meet these requirements has lead to the design of a class of algorithms referred to as communication avoiding algorithms [38, 60, 59]. These algorithms asymptotically attain lower bounds on communication. Furthermore they perform some extra floating-point operations, which remain however a lower order term. Thus this class of algorithms leads to significant performance gains comparing to classical algorithms, since it allows to overcome the communication bottlenecks. In the framework of these algorithms, a communication avoiding LU factorization referred to as CALU was developed by Grigori et al in [60], where the authors introduce a new pivoting strategy, referred to as tournament pivoting based on Gaussian elimination with partial pivoting. The tournament pivoting strategy allows to minimize the amount of communication during the panel factorization. However using such a pivoting strategy results in a less stable factorization in terms of growth factor upper bound comparing to the original method, that is Gaussian elimination with partial pivoting. Improving the numerical stability of the communication avoiding LU factorization presents the motivation behind the first part of our work, that is the second and the third chapters.

In this thesis, we study both sequential and parallel algorithms that compute a dense LU decomposition. For the sake of accuracy and efficiency, we discuss their numerical stability as well as their ability to reduce the amount of communication. The methods presented throughout this work are based on the communication avoiding algorithms techniques and principle, in particular the communication avoiding pivoting strategy, however they aim at improving the numerical stability of the existing algorithms and leading to better performance on systems with multiple levels of parallelism. Thus the thesis is mainly composed of three parts. The first part introduces a new LU factorization using a new pivoting strategy based on strong rank revealing QR factorization. This algorithm is more stable than Gaussian elimination with partial pivoting in terms of growth factor upper bound. The second part presents a communication avoiding variant of the algorithm presented in the previous part. Besides reducing communication, this algorithm is more stable than the original communication avoiding LU factorization. In the last part, we study hierarchical LU algorithms that are suitable for machines with multiple levels of parallelism.

Summary and contributions

The rest of this thesis is organized as follows.

– Chapter 1 presents a background of our work and discusses the state of the art approaches. It especially includes previous research works, we think they are relevant to our work and contributions. First we present the two key linear algebra direct methods used for solving linear systems of equations, that is the LU factorization and the QR factorization. In particular we discuss their numerical stability, which is an important axis of the work presented throughout this thesis (section 1.2). After that we move to the second axis, that is communication (section 1.3). We mainly define communication and explain why it might be an important issue to address when designing parallel algorithms. Then we present state of the art parallel pivoting strategies including those we use throughout this thesis (section 1.4). In section 1.5, we describe some principles of parallel dense matrix computations ranging from algorithmic techniques to scheduling techniques. Finally, we present several parallel performance models and discuss both the contributions and the drawbacks of each (section 1.6).

– Chapter 2 introduces a new algorithm, referred to as LU_PRRP, for the LU decomposition. This algorithm uses a new pivoting strategy based on strong rank revealing QR factorization. The first step of the algorithm produces a block LU decomposition, where the factors L and U are block matrices. The LU_PRRP algorithm mainly aims at improving the numerical stability of the LU factorization. We start by presenting the algorithm in section 2.2. Then we discuss its numerical stability (section 2.2.1). Finally we evaluate the LU_PRRP algorithm based on an extensive experiment set (section 2.2.2).

– Chapter 3 presents a communication avoiding version of LU_PRRP, referred to as CALU_PRRP. This algorithm aims at overcoming the communication bottleneck during the panel factorization if we consider a parallel version of LU_PRRP. We show that CALU_PRRP is asymptotically optimal in terms of both bandwidth and latency. Moreover, it is more stable than the communication avoiding LU factorization based on Gaussian elimination with partial pivoting in terms of growth factor upper bound. We detail the algebra of the CALU_PRRP algorithm and discuss its numerical stability in section 3.2. This chapter together with the previous one has lead to an under revision publication in SIAM Journal Matrix Analysis Applications as *LU Factorization with Panel Rank Revealing Pivoting and its Communication Avoiding version* [A. Khabou, J. Demmel, L. Grigori, and M. Gu] [A2].

– Chapter 4 introduces two hierarchical LU factorizations suitable for systems with multiple levels of parallelism. These algorithms are based on recursive calls to the CALU routine suitable for one level of parallelism. The first algorithm, referred to as multilevel CALU, is based on bi-dimensional recursive pattern, where CALU is recursively applied to blocks of the input matrix. The second algorithm, referred to as recursive CALU, is based on an uni-dimensional recursive pattern, where CALU is recursively performed on an entire block of columns. In this chapter we present the two algorithms (sections 4.2 and 4.3). However we limit our analysis to two levels of parallelism. Thus we study both 2-level multilevel CALU, referred to as 2-level CALU, and 2-level recursive CALU, referred to as 2-recursive CALU. We show that the former is asymptotically optimal in terms of both latency and bandwidth at each level of parallelism, while the latter only minimizes the bandwidth cost, because the recursive call is performed on the entire panel. Note that the part of this chapter dealing with multilevel CALU has lead to a publication in EuroPar2012 as *Avoiding Communication through a Multilevel LU Factorization* [S. Donfack, L. Grigori, and A. Khabou] [A1].

– Chapter 5 presents a more detailed study of multilevel CALU, one of the hierarchical algorithms introduced in the previous chapter. To be able to model in a realistic way the multilevel CALU algorithm, we first build a performance model, the HCP model, that takes into account both the processor hierarchy and the network hierarchy (section 5.2). Then we introduce a multilevel Cannon algorithm that we use as a tool to model matrix matrix operations performed during the multilevel CALU algorithm (section 5.3). The evaluation of our algorithm under the HCP model shows that the multilevel CALU algorithm is asymptotically optimal in terms of both bandwidth and latency at each level of the hierarchy (section 5.4). We also show that multilevel CALU is predicted to be significantly faster than CALU on exascale machines with multiple levels of parallelism (section 5.5).

– Appendix A presents all our experimental results for algorithms introduced throughout this work

including the LU_PRRP factorization, the binary tree based CALU_PRRP, the flat tree based CALU_PRRP, the binary tree based 2-level CALU, and the binary tree based 3-level CALU. The tested matrices include random matrices and a set of special matrices. For each algorithm, we show results obtained for both the LU decomposition and the linear solver. We also display the growth factor and results for Gaussian elimination with partial pivoting for comparison.

– Appendix B details the costs analysis performed along this thesis. All these costs are evaluated, with respect to the critical path, in terms of arithmetic cost and communication cost. The communication cost includes both the latency cost, due to the number of messages sent, and the bandwidth cost, due to the volume of data communicated during the execution. We first present the cost analysis of the LU_PRRP algorithm and that of the CALU_PRRP algorithm. These two evaluations are performed with respect to a simplified bandwidth-latency performance model. Then we present the costs analysis of both 2-level CALU and 2-recursive CALU, using the same simplified performance model extended by several constraint on memory. Finally we detail the cost of multilevel CALU with respect to our hierarchical performance model (HCP).

– Appendix C presents the algebra of the block parallel CALU_PRRP algorithm, a less stable communication avoiding variant of the LU_PRRP algorithm introduced in chapter 3.

– Appendix D presents the MATLAB code used to generate a generalized Wilkinson matrix.

This thesis makes the following contributions.

– The design of a new algorithm (LU_PRRP) that produces an LU decomposition. This algorithm uses strong rank revealing QR to select sets of pivot rows and to compute the L_{21} factor of each panel. Due to both the new pivoting strategy and the manner used to compute the L_{21} factors, the LU_PRRP algorithm is more stable than Gaussian elimination with partial pivoting in terms of growth factor upper bound.
– Using the previous pivoting strategy, together with the tournament pivoting strategy introduced in the context of CALU, we design a communication avoiding version of LU_PRRP (CALU_PRRP) which on one hand asymptotically attains the lower bounds on communication, and on the other hand is more stable than the CALU factorization in terms of growth factor upper bound.
– The design of a hierarchical parallel performance model which is both simple and realistic (the HCP model).
– The design of a hierarchical LU factorization (multilevel CALU) that is asymptotically optimal in terms of both latency and bandwidth at each level of the hierarchy with respect to the HCP model.

Chapter 1

High performance numerical linear algebra

1.1 Introduction

This chapter presents a background of our work. It is organized as follows. We start with presenting the linear algebra direct methods, we either work on or use throughout this work (section 1.2). This section includes mainly several versions of both LU factorization and QR factorization. We consider in particular Gaussian elimination with partial pivoting and strong rank revealing QR factorization. Hence in chapter 2 we introduce an algorithm based on strong rank revealing pivoting strategy in order to compute a block LU decomposition. Then in section 1.3, we briefly explain what communication means for both sequential and parallel algorithms and why it is important to consider in design efficient algorithms. In this context a class of algorithms referred to as communication avoiding was first introduced in [55, 80] for tall and skinny matrices (with many more rows than columns) using a binary tree reduction for the QR factorization, then generalized by Demmel et al. in [39] to any reduction tree shape and to rectangular matrices. Here we only focus on the communication avoiding LU factorization (CALU). Note however that a similar algorithm exists for the QR factorization. In section 1.4, we detail and discuss state-of the art parallel pivoting strategies and their numerical stability. After that we present some principles of parallel dense matrix computation (section 1.5), including both algorithmic methods (block algorithms, recursive algorithms, tiled algorithms) and implementation techniques, for which we restrict our discussion to task decomposition and scheduling methods. Finally in section 1.6, we discuss several performance models. The point here is to motivate the design of our performance model HCP introduced in chapter 5, and which takes into account in a simple way the different level of parallelism in a hierarchical system. Note here that in all the algorithms we use MATLAB notations.

1.2 Direct methods

Many scientific applications require the resolution of linear systems,

$$Ax = b.$$

In this section we introduce two main computational kernels widely used in direct linear algebra, the LU factorization and the QR factorization. We first present the original algorithms of these two methods and some other variants which were relevant to our work, including Gaussian elimination with partial pivoting (GEPP) and strong rank revealing QR (strong RRQR). Then we discuss the numerical stability of these algorithms and address some issues.

1.2.1 The LU factorization

Suppose we want to solve a nonsingular n by n linear system

$$Ax = b. \tag{1.1}$$

One way to do that is to use Gaussian elimination without pivoting, which decomposes the matrix A into a lower triangular matrix L, and an upper triangular matrix U. Thus the linear system above is very easy to solve by forward substitution for the lower triangular matrix and backward substitution for the upper triangular matrix.

Algorithm 1: LU factorization without pivoting (in place)

Input: $n \times n$ matrix A
for *k = 1:n-1* **do**
  for *i = k+1:n* **do**
    $A(i,k) = A(i,k)/A(k,k)$
  for *i = k+1:n* **do**
    for *j = k+1:n* **do**
      $A(i,j) = A(i,j) - A(i,k) * A(k,j)$

For dense matrices, LU factorization can be performed in-place. It means that the output LU factors overwrite the input matrix A. We recall that the floating-point operations of the LU decomposition applied to a square matrix is $(2n^3/3)$. However Gaussian elimination without pivoting is numerically unstable unless A is for example diagonal dominant or positive definite. Thus to improve the numerical stability of this method pivoting is required $(PA = LU)$, in particular we discuss Gaussian elimination with partial pivoting.

Partitioned LU factorization

The Algorithm 1 presented above is based on products of scalars. To allow more parallelism and to lead to better performance on machines with hierarchical memories, the LU factorization can be rearranged in order to allow the use of matrix-vector operations (BLAS 2) [2] and matrix-matrix operations (BLAS 3) [2]. Given a matrix A of size $n \times n$ and a block of size b, after the first step of the partitioned LU we have,

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} = \begin{bmatrix} L_{11} & 0 \\ L_{21} & I_{n-b} \end{bmatrix} \times \begin{bmatrix} U_{11} & U_{12} \\ 0 & A^s_{22} \end{bmatrix},$$

where $A_{11} = L_{11}U_{11}, U_{12} = L_{11}^{-1}A_{12}, L_{21} = A_{21}U_{11}^{-1}$, and $A^s_{22} = A_{22} - L_{21}U_{12}$. Algorithm 2 presents a partitioned LU factorization, where we assume that n is a multiplier of b.

Block LU factorization

Here we introduce a block LU factorization, which is relevant to our work. This factorization produces a block LU decomposition, where the factors L and U are composed of blocks [40]. Here is the

Algorithm 2: Partitioned LU factorization without pivoting (in place)

Input: $n \times n$ matrix A, block size b
N= n/b
for $k = 1 : N$ **do**
  Let $K = (k-1) * b + 1 : k * b$
  /* Compute the LU factorization of the current diagonal block */
  $A(K, K) = L(K, K)U(K, K)$
  /* Compute the block column of L */
  $A(k * b + 1 : n, K) = A(k * b + 1 : n, K)U(K, K)^{-1}$
  /* Compute the block row of U */
  $A(K, k * b + 1 : n) = L(K, K)^{-1}A(K, k * b + 1 : n)$
  /* Update the trailing matrix */
  $A(k*b+1 : n, k*b+1 : n) = A(k*b+1 : n, k*b+1 : n) - A(k*b+1 : n, K)A(K, k*b+1 : n)$

detail of one step of the block LU factorization, where we assume that A_{11} is a non singular matrix of size $b \times b$:

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} = \begin{bmatrix} I_b & 0 \\ L_{21} & I_{n-b} \end{bmatrix} \times \begin{bmatrix} A_{11} & A_{12} \\ 0 & A^s_{22} \end{bmatrix}$$

Note that the algorithms that we will introduce in chapters 2 and 3, without additional Gaussian eliminations on the diagonal blocks, produce a block LU factorization as defined here.

Numerical stability

The division by $A(k, k)$ in Algorithm 1 might break down the algorithm if $A(k, k)$ is equal to zero or cause a numerical problem if it is small in magnitude. If such a problem occurs, the factorization fails. Moreover, giving that the accuracy of a computation is limited by the number representation on a computer, the computed term $A(i, k)$ may grow (line 3 in 1) because of rounding errors, and thus becomes dominating during the update of the trailing matrix. For that reason it is necessary to perform pivoting, that is to move a larger element into the diagonal position by swapping rows (partial pivoting), columns, or rows and columns(complete pivoting, rook pivoting) in the matrix. We will discuss several parallel pivoting strategies in 1.4. Here we consider partial pivoting that only performs row interchanges.

Algorithm 3: LU factorization with partial pivoting (in place)

Input: $n \times n$ matrix A
for k = *1:n* **do**
  i = index of max element in $|A(k : n, k)|$
  $piv(k) = i$
  swap rows k and i of A
  $A(k + 1 : n, k) = A(k + 1 : n, k)/A(k, k)$
  $A(k + 1 : n, k + 1 : n) = A(k + 1 : n, k + 1 : n) - A(k + 1 : n, k) * A(k, k + 1 : n)$

The first step in Algorithm 3 determines the largest entry in the k-th column in the subdiagonal part of A. After row swaps, the process continues as described for the LU factorization without pivoting. Thus the decomposition that it produces is of the form $A = P^T LU$ or $PA = LU$, where P is the permutation matrix that illustrates the pivoting process.
The stability of an LU decomposition mainly depends on the growth factor g_W, which measures how large the elements of the input matrix are getting during the elimination,

$$g_W = \frac{\max_{i,j,k} |a_{ij}^{(k)}|}{\max_{i,j} |a_{ij}|} \geq 1,$$

where $a_{ij}^{(k)}$ denotes the entry in position (i, j) obtained after k steps of elimination.

In [96], Wilkinson showed that for partial pivoting, the growth factor upper bound is 2^{n-1}. This worst case growth factor is attainable. In fact Gaussian elimination with partial pivoting could fail for a small set of problems, including a set of matrices designed to make the partial pivoting strategy fail [67] and some matrices arising from practical applications [50, 99]. Note however, that Gaussian elimination with partial pivoting is stable in practice. Understanding such a behavior is still one of the biggest unsolved problems in numerical analysis.

1.2.2 The QR factorization

In this section we first recall the classic QR factorization. We mainly discuss its different versions. One interesting variant of the QR decomposition is the rank revealing QR (RRQR). QR factorization is one of the major one-sided factorizations in dense linear algebra. Based on orthogonal transformations, this method is well known to be numerically stable and is a first step toward the resolution of least square systems.

Classic QR factorization

The QR factorization decomposes a given matrix A into the product of an orthogonal matrix Q and an upper triangular matrix R. There are several methods to compute the QR factorization of a matrix, such as the Gram-Schmidt process, the Householder transformations, or the Givens rotations [54]. The numerical stability of the QR factorization depends on the approach used.

The most common approach is that based on Householder transformations. During this factorization successive orthogonal matrices of the form,

$$H = I - \frac{2}{\|u\|^2} uu^T,$$

are computed. These are the Householder matrices. During the factorization, applying the Householder matrices to the input matrix A allow to successively annihilate all elements below the diagonal element, and thus obtain the upper triangular factor R. Let's note H_k the Householder matrix that eliminates the subdiagonal elements of the k^{th} column of the input matrix. Therefore we obtain,

$$H_n H_{n-1} \cdots H_2 H_1 A = R.$$

Thus,

$$A = (H_n H_{n-1} \cdots H_2 H_1)^{-1} R = QR = H_1^T H_2^T \cdots H_{n-1}^T H_n^T R,$$

since the Householder matrices are orthogonal. Note that the Q factor is never explicitly formed. To apply the Householder matrix H to a given matrix A, we rather proceed as follows,

$$HA = (I - \frac{2}{\|u\|^2}uu^T)A = A - \frac{2}{\|u\|^2}u(u^TA).$$

Hence it consists of a rank-1 modification of the identity. Algorithm 4 illustrates a QR factorization based on Householder transformations.

Algorithm 4: Householder QR factorization

for *k = 1:n-1* **do**
- $\alpha_k = \frac{sign(A(k,k))}{\|A(k:m,k)\|_2}$
- $u_k = [0 \cdots 0 A(k,k) \cdots A(m,k)]^T - \alpha_k e_k$
- $\beta_k = u^T u$
- **if** $\beta_k = 0$ **then**
  - continue with next k
- **for** *j = k to n* **do**
  - $A(k+1:m, j) = A(k+1:m, j) - \frac{2}{\beta_k}(u^T A(k+1:m, j))u$

Rank revealing QR with column pivoting

In many numerical problems including subset selection, total least squares, regularization, and matrix approximation, it is necessary to compute the rank of a matrix [32]. The most reliable method to find the numerical rank of a matrix is the singular value decomposition (SVD). However the SVD has a high computational complexity comparing to the QR factorization. Hansen and Chan showed in [32] that rank revealing QR factorization with column pivoting is a reliable alternative to the SVD. This factorization defined in [20], is competitive for solving the problems cited above. Hence the rank revealing QR factorization produces tight bounds for the small singular values of a matrix and builds an approximated basis to its null-space.

Algorithm 5 details the different steps of rank revealing QR with column pivoting. This factorization was introduced by Golub and Businger [21]. The algorithm iteratively performs computations on columns of the matrix. At each step j, the column with the largest norm is found and moved to the first position, a Householder matrix is generated and applied to the trailing matrix.

Strong rank Revealing QR

Here we describe the strong RRQR introduced by M. Gu and S. Eisenstat in [61]. This factorization will be used in our new LU decomposition algorithm, which aims to obtain an upper bound of the growth factor smaller than that of GEPP (see Section 2.2). Consider a given threshold $\tau > 1$ and an $h \times p$ matrix B with $p > h$, which is the case of interest for our research. A Strong RRQR factorization on such a matrix B gives (with an empty $(2,2)$ block)

$$B^T\Pi = QR = Q \begin{bmatrix} R_{11} & R_{12} \end{bmatrix},$$

where $\|R_{11}^{-1}R_{12}\|_{max} \leq \tau$, with $\| \ . \ \|_{max}$ being the biggest entry of a given matrix in absolute value. This factorization can be computed by a classical rank revealing QR factorization with column pivoting

Algorithm 5: Classical Rank Revealing QR based on column pivoting

Column norm vector: $colnorms(j) = \|A(:,j)\|_2 \ (j = 1:n)$
for *j = 1 : n* **do**
  Find the column with the vector i with the maximal norm:
  $colnorms(i) = max(colnorms(j:n))$
  Interchange column i and column j: $A(:,[j\ i]) = A(:,[i\ j])$ and
  $colnorms([j\ i]) = colnorms([i\ j])$
  Reduction: Determine a Householder matrix H_j such that
  $H_j A(j:m,j) = \pm\|A(j:m,j)\|_2 e_1$
  Matrix Update: $A(j:m, j+1:n) = H_j A(j:m, j+1:n)$
  Norm Downdate: $colnorms(j+1:n) = colnorms(j+1:n) - A(j, j+1:n)^2$

introduced in section 1.2.2 followed by a limited number of additional swaps and QR updates if necessary.

Algorithm 6: Strong RRQR

Compute $B^T\Pi = QR$ using the classical RRQR with column pivoting
while *there exist i and j such that* $|(R_{11}^{-1}R_{12})_{ij}| > \tau$ **do**
  Set $\Pi = \Pi\Pi_{ij}$ and compute the QR factorization of $R\,\Pi_{ij}$ (QR updates)
Ensure $B^T\Pi = QR$ with $\|R_{11}^{-1}R_{12}\|_{max} \leq \tau$

The while loop in Algorithm 6 interchanges any pairs of columns that can increase $|det(R_{11})|$ by at least a factor τ. At most $O(\log_\tau n)$ such interchanges are necessary before Algorithm 6 finds a strong RRQR factorization. The QR factorization of $B^T\Pi$ can be computed numerically via efficient and numerically stable QR updating procedures. See [61] for details.

Numerical stability

The stability of the QR factorization highly depends on the transformation used. The orthogonal transformations including Householder transformations and Givens rotations are very stable [98], since the update of the matrix is based on rank-1 modification of the identity.

1.3 Communication "avoiding" algorithms

In this section, we first define communication, then explain why its cost is expensive comparing to the arithmetic cost. This represents the reasons behind developing communication avoiding algorithms, which communicate less than the standard methods, at the cost of some redundant computations. In section 1.3.1, we define communication. It includes both data movement between processors and data movement between levels of the memory hierarchy. Then in section 1.3.2, we describe the latency bandwidth performance model used to estimate the performance of the communication avoiding algorithms, including those developed throughout this work, before the design of our performance model HCP. In section 1.3.3 we introduce lower bounds on communication with respect to a simplified performance model. Thus an algorithm would be communication avoiding if it asymptotically attains these lower

bounds. After that, we present the ScaLAPACK LU routine, a case of interest that motivates the design of communication avoiding algorithms (section 1.3.4). Finally, in sections 1.3.5, we present an example of communication avoiding algorithms, communication avoiding LU factorization (CALU), and discuss its numerical stability in 1.3.6. For details, see for example [39, 59, 38, 83].

1.3.1 Communication

To be able to avoid communication, we should first understand its meaning. To model the performance of computers and predict the performance of algorithms running on these systems, we consider both computation and communication. Computation means floating-point operations, such as addition, multiplication, or division of two floating-point numbers. Communication means data movement. It includes both data movement between levels of a memory hierarchy (sequential case), and data movement between processors working in parallel (parallel case).

Communication between parallel processors may take the following forms, among others:
– Messages between processors considering a distributed-memory system.
– Cache coherency traffic considering a shared-memory system.

Communication between levels of a memory hierarchy can be for example between main memory and disk, this is the case of an out-of-core code.

1.3.2 A simplified performance model

The performance model introduced in the context of communication avoiding algorithms and used to estimate the run time of an algorithm is based on a latency-bandwidth model. In this model, communication is performed in messages. Each message is a sequence of n words in consecutive memory locations. The same model is used whether the implementation uses a shared-memory or distributed-memory programming model, or whether the data is moving between processors or between slow and fast memory on a single processor.

– To send a message containing w words, the required time is

$$Time_{Message}(w) = \alpha + \beta \cdot w,$$

where α is the latency and β is the inverse of the bandwidth.

This communication model implies that for any pair of nodes in the parallel machine, the time to communicate is the same. This means that we assume a completely connected network, that is each node in the network has a direct communication link to every other node. It is also important to note that this simplified communication cost is still valid as long as there is no congestion that occurs in any part of the network. Hence a given communication pattern might lead to a network congestion regarding the underlying architecture and the network topology.

– To perform n floating-point operations, the required time is

$$Time_{Flops}(n) = \gamma \cdot n,$$

where γ is the time required to perform one floating-point operation.

For parallel algorithms, the runtime is evaluated along the critical path of the algorithm. Note that throughout this work all our cost analysis is performed on the critical path.

Mark Hoemmen stated in his PhD dissertation [68] that "Arithmetic is cheap, bandwidth is money, latency is physics". Hence computation performance scales with the number of transistors on a chip and bandwidth scales with cost of hardware (number of wires, pipes, disks, ...). However, the latency is still the most difficult to improve, because it is in general limited by physical laws. These hard facts represent the main motivations behind the development of new algorithms which take into consideration the physical (and economic) laws of hardware. Thus, rearranging classical algorithms to reduce communication can often lead to better performance, even when some redundant computation is required.

1.3.3 Theoretical lower bounds

In [69] Hong and Kung proved a lower bound on the amount of data moved between fast memory and slow memory (sequential case) in the context of dense n-by-n matrix-multiplication based on the classical $O(n^3)$ algorithm. Then Irony, Toledo and Tiskin [72] presented a new proof of this result and extended it to the parallel case, that is considering the amount of data moved between processors. In [15], these lower bounds on communications have been generalized for linear algebra. The authors show that these lower bounds apply for a wide class of problems that could be written as three nested loops (matrix-multiplication-like problems). These lower bounds apply to both sequential and parallel distributed memory model. We refer to the number of moved words as W, to the number of messages as S, and the number of floating-point operations as $\#\text{flops}$.

We recall that to describe lower bounds we use the big-omega notation $f(n) = \Omega(g(n))$, saying that for some constant $c > 0$ and all large enough n, $f(n) \geq cg(n)$.

Given a system with a fast memory of size M, the communication bandwidth is lower bounded by

$$W = \Omega\left(\frac{\#\text{flops}}{\sqrt{M}}\right),$$

and the latency is lower bounded by

$$S = \Omega\left(\frac{\#\text{flops}}{M^{\frac{3}{2}}}\right).$$

Considering a system composed of p processors and a memory of size M, and a matrix-multiplication-like problem performed on a dense n-by-n matrix, the lower bounds above become,

$$W = \Omega\left(\frac{n^3/p}{\sqrt{M}}\right),$$

$$S = \Omega\left(\frac{n^3/p}{M^{\frac{3}{2}}}\right).$$

These lower bounds apply for what we call 2D-algorithms, that is when one copy of input/output matrices are distributed across a 2D grid of processors, $p = \sqrt{p} \times \sqrt{p}$. In this case $M \approx 3n^2/p$. For example Cannon's algorithm [27] simultaneously balances the load, minimizes the number of words moved (bandwidth), and minimizes the number of messages (latency). They also apply for 3D-algorithms, that is when $p^{1/3}$ copies of the input/output matrices are distributed across a 3D grid of processors,

$p = p^{1/3} \times p^{1/3} \times p^{1/3}$. Therefore the memory size of each processor satisfies $M \approx 3n^2/p^{2/3}$. Examples of such algorithms are presented in [8, 9, 35]. Thus considering the two extreme cases detailed above, the memory size must satisfy $n^2/p \leq M \leq n^2/p^{2/3}$.

In the context of 2.5D algorithms, an intermediate variant between 2D algorithms and 3D algorithms, it was shown that the lower bound on bandwidth can be attained for Gaussian-elimination without pivoting or with tournament pivoting factorizations [83].

1.3.4 Motivation case: LU factorization as in ScaLAPACK (pdgetrf)

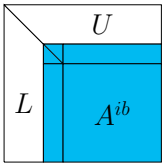
Considering a parallel implementation of Gaussian elimination with partial pivoting, as for example the $PDGETRF$ routine in ScaLAPACK, we can see that the panel factorization is a bottleneck in terms of latency cost. Thus to select the largest element in each column of a given panel of size b, $O(b \log P)$ messages need to be exchanged. In other words during the panel factorization the synchronization of the processors is required at each column, and $O(n \log P)$ synchronizations are needed overall.

Algorithm 7: LU factorization on a $P = P_r \times P_c$ grid of processors

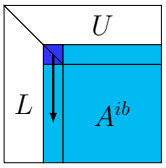
for $ib = 1$ *to* $n - 1$, *step* b **do**

$A^{ib} = A(ib : n, ib : n)$

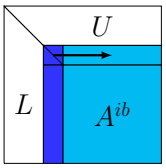
1. Compute panel factorization (pdgetf2): $O(n\ log_2(P_r))$
   - Find pivot in each column, swap rows
2. Apply all row permutations (pdlaswp): $O(\frac{n}{b}\ (log_2(P_r) + log_2(P_c)))$
   - Broadcast pivot information along the rows
   - Swap rows at left and right
3. Compute block row of U (pdtrsm): $O(\frac{n}{b}\ log_2(P_c))$
   - Broadcast right diagonal block of L of current panel
4. Update trailing matrix (pdgemm): $O(\frac{n}{b}\ (log_2(P_r) + log_2(P_c)))$
   - Broadcast right block column of L
   - Broadcast down block row of U



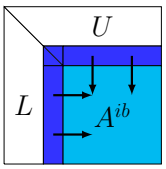
(1)



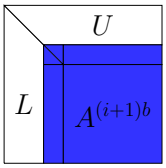
(2)



(3)



(4)



The LU factorization, as implemented is ScaLAPACK, does not attain the lower bounds previously introduced in terms of latency. This is because of the partial pivoting. In order to reduce the number of messages and thereby overcome the latency bottleneck during the panel factorization, a new algorithm was introduced, referred to as communication avoiding LU factorization.

1.3.5 Communication avoiding LU (CALU)

The communication avoiding LU factorization aims at minimizing the number of messages exchanged during the panel factorization. To show the improvement that CALU performs in terms of latency cost, we consider for example the routine, cited above, performing the LU factorization as implemented in ScaLAPACK. This routine is based on Gaussian elimination with partial pivoting. Thus for each column of the matrix processor synchronization is required to find the element of maximum magnitude and move it to the diagonal position. However for CALU synchronization of processors is needed only for each block of columns. Hence given a square matrix of size n, and a panel of size b, CALU will only exchange $O(n/b \log P)$ messages while the corresponding routine in ScaLAPACK will perform $O(n \log P)$ messages. So the larger the panel is, the lower the number of messages which need to be exchanged is.

CALU algorithm

Here we describe briefly the CALU algorithm and show that comparing to a classic LU factorization, the main difference lies on the panel factorization referred to as TSLU. CALU is a partitioned algorithm that performs iteratively on blocks of columns (panels). At each iteration a block column is distributed over the P processors working on the panel. First each processor performs Gaussian elimination with partial pivoting on its block to select b candidate pivot rows. The next step of the panel factorization is a reduction operation where the previously selected pivot rows are merged with respect to the underlying reduction tree. For example considering a binary reduction tree, at the first level of the reduction the P sets of b pivot candidates are merged two by two, and again Gaussian elimination with partial pivoting is applied to the block of size $2b \times b$ to select b pivot candidates for the next level of the reduction. After $logP$ steps of reduction, the binary tree is traversed bottom up and a final set of b pivot rows is selected. The different steps of reduction (preprocessing step) are depicted in Figure 1.1, where we consider a panel W partitioned over 4 processors and a binary reduction tree. The final b rows are then moved to the first positions of the current panel and an LU factorization without pivoting is performed on the panel. Finally the block row of U is computed and the trailing matrix is updated. These two last steps of the CALU factorization are performed as in a classic LU factorization.

The performance model of parallel CALU (as detailed in [60]), with respect to the latency-bandwidth model introduced previously, is presented in table 1.1. Considering a 2D grid of processors $P = P_r \times P_c$, to attain the communication bounds recalled in 1.3.3, an optimal layout is used. This optimal layout was first introduced in the context of the communication avoiding QR factorization [38]:

$$P_r = \sqrt{\frac{mP}{n}} \ , \ P_c = \sqrt{\frac{nP}{m}} \ \text{ and } b = \log^{-2}\left(\sqrt{\frac{mP}{n}}\right) \cdot \sqrt{\frac{mn}{P}}, \tag{1.2}$$

The optimal block size b is chosen such that on one hand the lower bound on latency is attained, modulo polylogarithmic factor factors, and on the other hand the number of extra floating point operations remains a lower order term. With this layout, the cost model of parallel CALU is given in Table 1.1. We can see that CALU attains the lower bounds on both number of words and number of messages, modulo polylogarithmic factors. For recall, these counts assume a globally load balance and are performed with respect to the critical path. More details about the later will be given in 1.5.2. Note that for all the communication avoiding algorithms we develop in this work we use the same optimal layout detailed above.

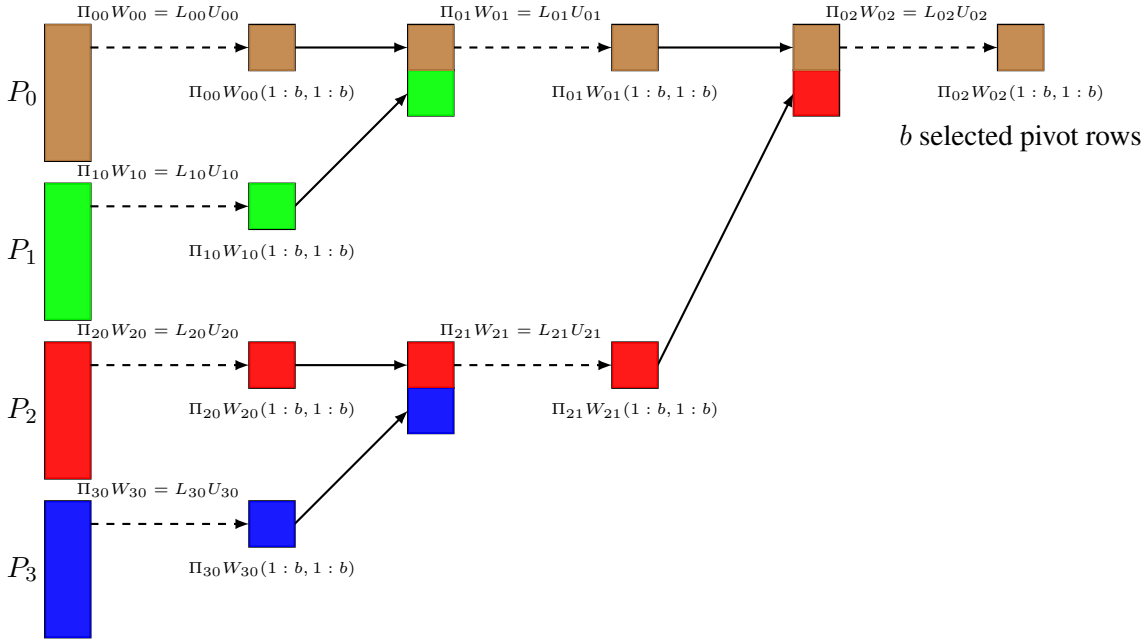


Figure 1.1: Selection of b pivot rows for the panel factorization using the CALU algorithm: preprocessing step.

CALU implementation

Several implementations of the communication avoiding LU factorization were performed recently, including a distributed implementation and a multithreaded implementation. In [44], Donfack et al. have developed a fully dynamic multithread communication avoiding LU factorization, which leads to significant speedup comparing to MKL and PLASMA routines, especially for tall and skinny matrices. Note that we will use this implementation in our hybrid MPI-pthread multilevel LU factorization in chapter 4. Hence the distributed implementation based on MPI is called at the first level of recursion and the multithreaded implementation is used at the second level of recursion as reduction operator instead of Gaussian elimination with partial pivoting. In [43], the authors present an implementation of CALU based on a hybrid static/dynamic scheduling strategy of the task dependency graph, a strategy that ensures a balance of data locality, load balance, and low dequeue overhead. This implementation results on significant performance gains. Thus further improvements could be considered regarding the multilevel LU factorization presented in this work 4.5. Note that task decomposition and scheduling techniques will be discussed more in details in section 1.5.2.

1.3.6 Numerical stability

The numerical stability of communication avoiding LU factorization is quite different from that of Gaussian elimination with partial pivoting in theory, because it is based on a new pivoting strategy referred to as tournament pivoting. It is shown in [60] that in exact arithmetic, performing communication avoiding LU on a matrix A is equivalent to applying Gaussian elimination with partial pivoting to a larger matrix formed by blocks of A, sometimes slightly perturbed to get rid of singular blocks. Thus the growth factor upper bound of CALU is $2^{n(H+1)-1}$, where H is the height of the reduction tree used during the preprocessing step and n is the number of columns of the input matrix. Thereby the growth factor upper bound of communication avoiding LU is worse than that of Gaussian elimination with par-

	Parallel CALU with optimal layout	Lower bound
	Input matrix of size $m \times n$	
# messages	$3\sqrt{\frac{nP}{m}}log^2\left(\sqrt{\frac{mP}{n}}\right)\log P$	$\Omega(\sqrt{\frac{nP}{m}})$
# words	$\sqrt{\frac{mn^3}{P}}\left(log^{-1}\left(\sqrt{\frac{mP}{n}}\right)+\log\frac{P^2m}{n}\right)$	$\Omega(\sqrt{\frac{mn^3}{P}})$
# flops	$\frac{1}{P}\left(mn^2-\frac{n^3}{3}\right)+\frac{5mn^2}{2Plog^2\left(\sqrt{\frac{mP}{n}}\right)}+\frac{5mn^2}{3Plog^3\left(\sqrt{\frac{mP}{n}}\right)}$	$\frac{1}{P}\left(mn^2-\frac{n^3}{3}\right)$
	Input matrix of size $n \times n$	
# messages	$3\sqrt{P}\log^3 P$	$\Omega(\sqrt{P})$
# words	$\frac{n^2}{\sqrt{P}}\left(2\log^{-1}P+1.25\log P\right)$	$\Omega(\frac{n^2}{\sqrt{P}})$
# flops	$\frac{1}{P}\frac{2n^3}{3}+\frac{5n^3}{2P\log^2 P}+\frac{5n^3}{3P\log^3 P}$	$\frac{1}{P}\left(mn^2-\frac{n^3}{3}\right)$

Table 1.1: Performance models binary tree based communication avoiding LU factorization.

tial pivoting (2^{n-1}). Note that for a binary tree based CALU and given P processors working on the panel, the height of the reduction tree H is equal to $\log P$. Therefore, with an increasing number of processor working on the panel, the theoretical upper bound of the growth factor increases, which might lead to a less accurate factorization. However, Grigori et al. have shown in [60] via an extensive set of experiments including tests on random matrices and a set of special matrices, that CALU is stable in practice. Thus the upper bound cited above is never attained in practice. The experiments show that in practice the binary tree based CALU growth factor increases as $C \cdot n^{2/3}$, where C is around $3/2$. Trefthen et al. have presented in [92] an average case stability of Gaussian elimination with partial pivoting, where it is shown that the growth factor of GEPP grows as $O(n^{2/3})$. Note that the core motivation of the two first chapters of this work is the improvement of the numerical stability of CALU. Thus we introduce in chapter 3, an LU factorization that also reduces the communication and which is more stable than CALU in terms of the upper bound of the growth factor.

1.4 Parallel pivoting strategies

As it can be seen from the previous section, the main difficulty regarding the development of an efficient parallel algorithm for the LU decomposition is the pivoting strategy used. The algorithm should ensure a trade-off between efficient parallelization and numerical stability. Here we present some state-of the art approaches.

Pairwise pivoting

Gaussian elimination based on pairwise pivoting is illustrated in Algorithm 8. We can see that the selection of the largest element in a column is performed pairwise, which allows more parallelism.

More details about this pivoting strategy could be found in [86], where it is shown that the growth factor upper bound using pairwise pivoting is 4^{n-1}, which is 2 times larger than the upper bound of the growth factor of Gaussian elimination with partial pivoting. However, it is shown to be stable in practice, since it is based on low rank update in practice, according to the experiments performed in [92] on random matrices. Grigori et al. developed in [60] a block pairwise pivoting strategy to compute a communication avoiding LU factorization, where they observe, for random matrices, that the elements grow faster than linearly with respect to the matrix size.

Algorithm 8: LU factorization with pairwise pivoting

```
for k = 1:n-1 do
    for i = k+1:n do
        if A(k,k) < A(i,k) then
            swap rows i and k
        A(i,k) = A(i,k)/A(k,k)
    for i = k+1:n do
        for j = k+1:n do
            A(i,j) = A(i,j) − A(i,k) ∗ A(k,j)
```

Parallel pivoting (incremental pivoting)

This pivoting strategy for the LU decomposition, was first implemented in [81]. The main idea of this methods is to divide the panel into tiles, factorize the diagonal block (tile) using partial pivoting then annihilate pairwise the subdiagonal blocks. This strategy can be considered as a block version of pairwise algorithm. Thus, if the tile size is equal to the problem size $(t = n)$, the algorithm becomes Gaussian elimination with partial pivoting, while if the size of the tile is one element then the algorithm becomes Gaussian elimination with pairwise pivoting. Despite the significant gain in terms of efficiency, this pivoting strategy still deserves further investigations. Similarly here a block parallel pivoting strategy was explored in [60] as alternative to develop a communication avoiding LU factorization, where the panel factorization is performed only once. The numerical experiments performed do not allow to make any general conclusion. Note that in chapter 3 of this work we also consider similar approaches to develop alternatives to our LU factorization.

Tournament pivoting

This refers to the pivoting strategy introduced in the context of the communication avoiding LU factorization and recalled previously when describing the CALU algorithm (section 1.3.5). Remember that this pivoting strategy is mainly based on a reduction tree scheme, where at each level of the reduction, sets of candidate pivot rows are selected using Gaussian elimination with partial pivoting as a reduction operator. As discussed before, this pivoting strategy present a good trade-off between parallelism and numerical stability. Throughout this work, we use the same pivoting strategy for our communication avoiding algorithms, either for the CALU_PRRP algorithm in chapter 3 or in chapter 4 for multilevel CALU algorithms (1D recursive and 2D recursive). The main difference lies in the reduction operator used to select pivot rows. For example, for the CALU_PRRP factorization, the reduction operator will be strong rank revealing QR.

1.5 Principles of parallel dense matrix computations

The goal of this section is to introduce some principles of parallel dense matrix computations. We consider both algorithmic and implementation techniques.

1.5.1 Parallel algorithms

Partitioned algorithms

This technique basically aims at rearranging classical algorithms in order to replace scaler operations by matrix operations (matrix-vector operations BLAS 2 or matrix-matrix operations BLAS 3). We have already presented such an algorithm (Algorithm 2) for the LU factorization in section 1.2.1. Similar algorithm exists for the QR factorization using the technique introduced in [19], where the authors proposed a new way to represent products of Householder matrices (WY representation), a representation that allows to rearrange a typical Householder matrix algorithm to obtain matrix operations.

Recursive approach

A recursive algorithm typically decomposes the input matrix in equal parts, and recursively repeats the same computations in each part. Each level of recursion includes a factorization step and an update step. The main benefit from recursion formulation consists, first, of an automatic variable blocking [62], that is the recursive algorithm will be called recursively until reaching a level of recursion, where the handled matrix is small enough to fit in the cache memory of the processor. At this level of recursion a classical algorithm will be applied without generating cache misses during the computation. Thus recursion results in what is referred to as cache oblivious algorithms [52, 73]. Moreover recursive algorithms replace BLAS 2 operations in a standard partitioned algorithm with BLAS 3 operations, which improves the performance. An example of recursive LU factorization is discussed in [91]. Note, however, that these algorithms result in significant additional computation, because of the recursive updates. We will use recursive formulation in chapter 4 in order to design hierarchical algorithms suitable for systems with multiple levels of parallelism.

Tiled algorithms

Tiled algorithms [26] consist of decomposing the computation performed by a given algorithm into small tasks. These tasks can be executed with respect to the existing dependencies. Thus independent tasks are executed in parallel. The main goal behind developing such algorithms is to get rid of the sequential nature of the algorithms implemented in LAPACK. Additionally, the data layout used for tiled algorithms is based on contiguous blocks, while LAPACK uses a column-wise data layout. Note that the tile data layout allows to exploit cache locality and reuse. Hence it enforces the computation to be applied to small blocks of data that fit into cache.

Parallel implementations

Several libraries implement several routines for solving systems of linear equations, eigenvalue problems, and singular value decomposition. They basically implement partitioned algorithms discussed above. These libraries include the Linear Algebra PACKage, **LAPACK** [4], which can be adapted for shared memories, the Scalable Linear Algebra PACKage, **ScaLAPACK** [7], which is adapted for distributed systems, and the Parallel Linear Algebra for Scalable Multi-core Architectures, **PLASMA** [6], which basically implements tiled algorithms.

1.5.2 Tasks decomposition and scheduling techniques

Considering a parallel algorithm running on a parallel system, to ensure a good performance, the algorithm should be decomposed into small computations referred to as tasks to be scheduled onto

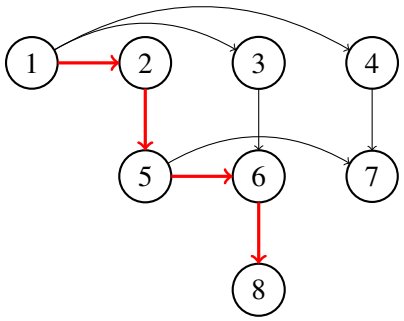


Figure 1.2: An example of task dependency graph, it represents 8 tasks and the precedence constraints between them. The red arrows corresponds to the critical path.

available processors. There are many different techniques for both the decomposition into tasks and the scheduling. Here we discuss some of these techniques, which we think are most relevant to our work. The decomposition of a computation into tasks can be performed in a static way, that is the tasks have a fixed size along the execution, or in a dynamic way, that is the size of tasks might vary during the execution.

One of the most important parameters to take into account when performing the task decomposition is the granularity of the tasks. The number of tasks to run does not only affect the scheduling overhead but also the performance of the tasks. On one hand fine grained tasks expose more parallelism but might use less efficiently the memory hierarchy, and hence it becomes difficult to perform each task efficiently. On the other hand coarse grained task decomposition exhibits less task parallelism but improves task performance since it can be executed more efficiently. Thus there is a trade-off between the potential task parallelism and the performance of each task. In other words, finding the best task decomposition can easily become an issue [11].

After the task decomposition we focus on the assignment of these tasks to the available processors, that is the scheduling. The scheduling can be static, dynamic, or hybrid, which includes both static and dynamic techniques with proportions that ensure an efficient performance. An example of hybrid scheduling used in the context of linear algebra kernels can be found in [43].

To obtain accurate results, the scheduler should respect the order of task execution, which is characterized by precedence constraints. Each of these constraints determines which tasks must finish before that a given task can start running. The dependency between tasks can be modeled by a graph such as a Directed Acyclic Graph (DAG). In such a graph the nodes represent the different tasks and the edges represent the precedence constraints, which determine the dependency between tasks. In Figure 1.2 we give an example of graph dependency. There we can see that all the tasks depend on task 1. Thus task 1 should run first. At the next step, tasks 2, 3, and 4 can be performed in parallel, and so on. The red path represents a **critical path**: $1 \rightarrow 2 \rightarrow 5 \rightarrow 6 \rightarrow 8$, which imposes a lower bound on the running time. We recall that a critical path is a path with maximal length. The path length is given by the sum of the task running times, when the communication between tasks is ignored. Assuming an uniform processing time for all tasks, the execution order presented in Figure 1.3 is optimal since it respects the linear order dictated by the critical path. We note here that in this work all the costs of our algorithms are evaluated on the critical path.

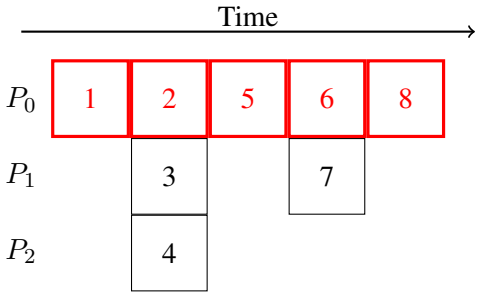


Figure 1.3: An example of execution of the task dependency graph depicted in Figure 1.2 using 3 processors, the red part represents the critical path.

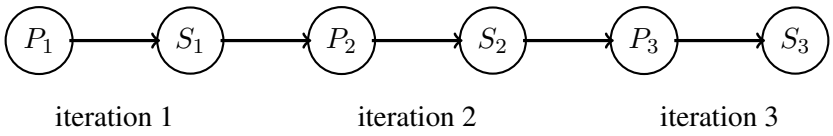


Figure 1.4: The task dependency graph of LU factorization without lookahead.

However, it turns out that it is very difficult in practice to determine an optimal execution order even considering strong assumptions such as uniform task running time. Hence there are many parameters that can affect a task running time including system noise, caches reuse, data locality, and even the scheduling technique itself. Therefore in order to find an efficient schedule for parallel algorithms including matrix computations, many heuristics are used, see for example [79].

In the following we focus on the scheduling techniques used for one-sided parallel matrix factorization algorithms. We study in particular the LU factorization but the same reasoning still applies to Cholesky and QR factorizations. We first discuss the lookahead concept, then tiled algorithms.

Here we consider a right-looking variant of the LU factorization. This algorithm performs iteratively on block of columns. Each iteration is composed of two parts: the panel factorization and the update of the trailing matrix using the previously factorized panel. If we note 'P_i' the task corresponding to the panel factorization and 'S_i' the task corresponding to the trailing matrix update for an iteration i, then we obtain the task dependency graph depicted in Figure 1.4. We note that with respect to the previous graph, there is no possible overlap between the panel factorization and the update of the trailing matrix, which can lead to significant performance decrease and scalability. Hence it is more challenging and more difficult to parallelize and to improve the panel factorization.

Since the panel factorization lies on the critical path, it is important to parallelize it. However at a given iteration, the factorization of the next panel of the matrix only depends on the update of that panel and not on the update of the entire trailing matrix. Thus the task 'S_i' in Figure 1.4 can be split into two tasks: the update of the next panel referred to as 'SP_i' and the update of the rest of the trailing matrix referred to as 'SR_i'. Figure 1.5 illustrates the new task dependency graph. We can see for example that the task 'P_2' can start once the task 'SP_1' finished. There is no need to wait anymore for the update of the entire trailing matrix. This new task decomposition improves the possibility of overlapping communication and computation and it is referred to as lookahead technique. The usage of this technique increases the scalability of the LU factorization and similar factorizations (QR, Cholesky)

on parallel architectures [31, 71, 25].

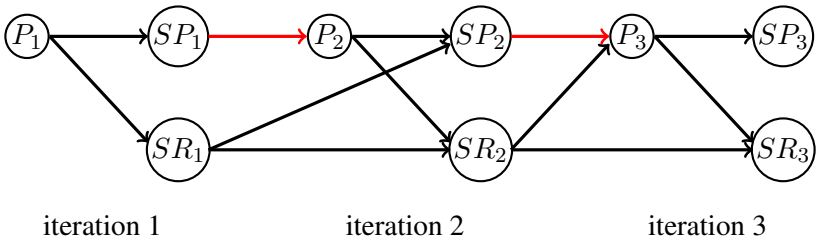


Figure 1.5: The task dependency graph of LU factorization with lookahead.

The two task decompositions detailed in figures 1.4 and 1.5 do not exhibit a lot of parallelism. We note that if we schedule the first panel factorization to the first processor, then the rest of the processors are idle during this factorization. Additionally, the first processor is idle during the update of the trailing matrix. To extract more parallelism and to improve the overlap and thus the performance of the factorization, the panel factorization and the update of the trailing matrix can be further partitioned into smaller tasks. However these tasks should not be too small to keep a good local performance and to not increase too much the scheduling overhead. We note that the utilization of the first panel can be improved by making it participate in the update of the trailing matrix whenever the panel is factored. One way to do so is to use dynamic scheduling via a queue of tasks [30]. This approach improves the overall execution time, that is the length of the critical path, when the size of the tasks is well chosen.

The technique detailed above is used in many implementations of one sided factorizations. We cite here some examples and techniques used for tiled algorithms [84, 47, 24, 63, 74]. In particular, we detail the task scheduling for multithreaded CALU. This routine developed in [44] is used as a building block in the 2-level CALU algorithm that we introduce in chapter 4.

Task scheduling for multithreaded CALU

Here we detail the different tasks executed by multithreaded CALU at each iteration. We consider a matrix of size $m \times n$, a panel of size b, and T_r threads working on the panel.

- Task **P** is a part of the preprocessing step. It corresponds to the application of Gaussian elimination with partial pivoting to a block of size $m/T_r \times b$ or $2b \times b$ depending on the reduction level.
- Task **L** computes a chunk of size $m/T_r \times b$ of the block column L.
- Task **U** apply row permutation to a block of the matrix and computes a chunk of size $b \times b$ of the block row U.
- Task **S** updates a chunk of the trailing matrix of size $m/T_r \times b$.

Figure 1.6 illustrates an example of task graph dependency of CALU performed on a matrix, partitioned into 4×4 blocks, using 2 threads. Each task is presented by circle. The yellow circles correspond to the preprocessing step, that is the selection of candidate pivot rows. In the example, there is only one level of reduction for each panel since we have 2 threads. Each red circle and each green circle corresponds respectively to the computation of a chunk of the current block column of L and a chunk of the current block row of U. Finally, each blue circle represents the update of a block of the trailing matrix. For recall, stable implementation of the tournament pivoting strategy requires that the panel factorization remains in the critical path.

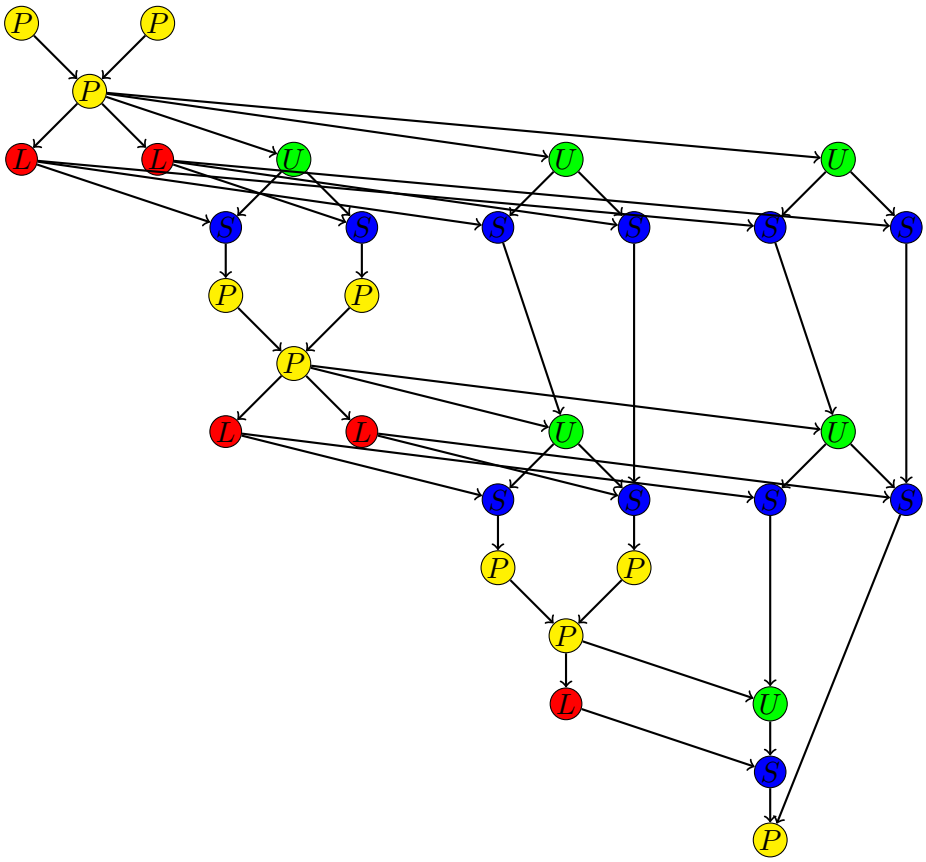


Figure 1.6: A task graph dependency of CALU applied to a matrix partitioned into 4×4 blocks, and using 2 threads.

We note that the updates are performed on blocks of size $m/T_r \times b$ with respect to the lookahead technique. We also note that the multithreaded CALU ensures considerable speedups, especially for tall and skinny matrices comparing to its corresponding routines in MKL and PLASMA. Further details and experiments can be found in [44].

1.6 Parallel performance models

Shared-memory and network models were widely used for the design of parallel algorithms, especially in the 1980's. However these models present several issues regarding the accuracy of the performance prediction they produce and their portability. We first discuss the PRAM model [64], which is an extension of the sequential Random Access Machine model (RAM) [76] to the parallel case. After that, we present network models and some bridging models, and discuss the drawbacks of each model. Note that similarly to the simplified model previously presented in section 1.3.2, the bridging models discussed here are also bandwidth-latency based.

1.6.1 PRAM model

The PRAM model is based on p processors working synchronously [77]. Each processor has a local memory and inter-processor communication is modeled by access to a shared memory. We note that the PRAM model does not make any distinction between shared and local memory accesses. This represents one of its major drawbacks since in general inter-processor communication is more expensive than local memory access. Thus the PRAM model is unrealistic and the cost estimation it produces is in general far from the real performance observed on parallel systems. Several drawbacks of the PRAM model are addressed in [95] and some references therein.

1.6.2 Network models

We now consider the network model class. For these models the communication is only performed between processors that are directly connected. Thus only nearest neighbors can communicate directly. The communication between the other processors is performed via explicit forward operated by intermediate processors. The main issue with this class of models is the lack of robustness. An algorithm designed with respect to a particular network model such as mesh, hypercube, or butterfly can be efficient on that specific interconnection structure. However significant change in the algorithm efficiency may occur if the network topology of the underlying architecture is different from the network topology of the initial model. More details on network models can be found in [75].

1.6.3 Bridging models

A bridging model "is intended neither as a hardware nor a programming model but something in between" [93]. Thus this class of models aims at reducing the gap between algorithm design and parallel system architecture. The main significant improvements of such models comparing to the previous ones consist of taking into account interprocessor communication costs and attempting to be the most independent from the underlying hardware. These models mainly address the issue of limited communication bandwidth. Here we only present the most relevant bridging models. Several other variations exist.

Some variants of Bulk-Synchronous Parallel models (BSP)

To analyze a given algorithm based on the BSP model [93], one should divide the algorithm into supersteps (Figure 1.7). Each superstep includes three phases, first independent computation performed by the processors on local data, then point-to-point communication between the processors, and finally a synchronization phase. Thus the model decouples computation and communication. The performance analysis based on a BSP model involves three parameters including the number of processors p, the gap g, which is the minimum interval between consecutive messages (the inverse of the bandwidth), and the latency l that qualifies the minimum time between subsequent synchronizations. Note that the original BSP model does not support nested supersteps. Thus it only allows to analyze algorithms running on systems with one level of parallelism.

The multi-BSP model [94] is a hierarchical model, which takes into account hierarchies both in communication and in the cache/memory system. Hence it extends the BSP model by adding a fourth parameter, which is the memory size. This idea was first presented by Tiskin in [78, 90]. Another hierarchical variant of BSP model referred to as decomposable BSP model (D-BSP) was proposed in [17]. D-BSP models a computational platform by a suitable decomposition into subset of processors. Each

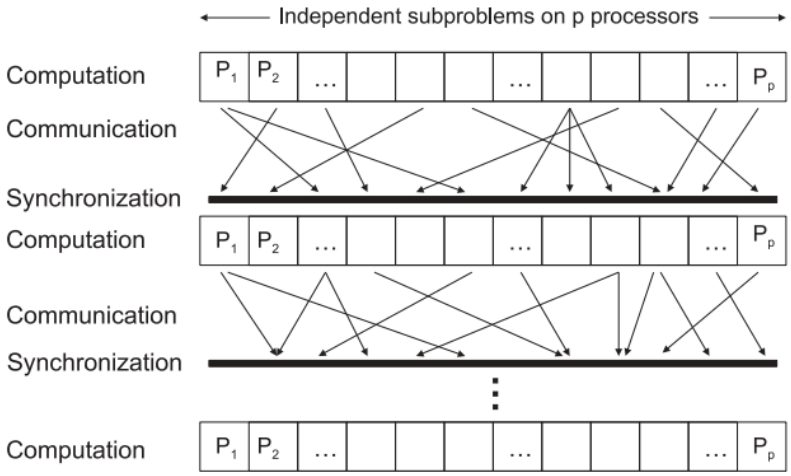


Figure 1.7: Supersteps under the BSP model [12].

subset is characterized by its own bandwidth and latency parameters. Thereby the D-BSP only captures communication hierarchy. Thus at each level of the hierarchy the communication within one subset of processors is faster than between different subsets. Both D-BSP model and multi-BSP models are more realistic than the original BSP model, since they are hierarchical models. But this comes along with some additional difficulties using these models. Note that the performance model we develop in section 5.2 is also based on communication hierarchy. However it uses a different technique to model how the messages are routed through the different levels of hierarchy.

LogP model

The LogP model was introduced in [33]. It is a bandwidth-latency model in which processors communicate by point-to-point messages. However comparing to the BSP model, LogP model takes into account the overhead o, that is the time when processor is either sending or receiving a message. It is shown in [18] that despite of being important for more accurate prediction, it is not compulsory to consider such a parameter when analyzing the algorithms asymptoticly. Furthermore, the LogP model uses a form of message passing based on pairwise synchronization, and defines the inverse of the bandwidth g for each processor, while it is defined globally in the original BSP model. To sum things up we can say that "LogP + barriers - overhead = BSP"(see e.g [5]). A more detailed comparison between the BSP model and the LogP model can be found in [18]. Note that this model does not enable the analysis of algorithms of hierarchical systems, since it is based on one level of parallelism.

Coarse-Grained Multicomputer (CGM)

The BSP model presented above might have non accurate speedup predictions, this is because of small messages. In order to overcome such a problem the coarse-grained multicomputer (CGM) model was proposed in [34]. This model performs communication based on message grouping. It does not make any assumption regarding the network topology and allows both distributed and shared memory. Similarly to the BSP model, the CGM model decomposes the algorithm into supersteps, each includes a computation phase and a communication phase. However during the communication phase, the messages that need to be sent by a given processor to the same target are aggregated up to the available

memory size, $O(n/p)$, where n is the problem size and p the number of processors.

1.6.4 Discussion

Performance is an important feature of parallel and distributed computing systems. Therefore, the performance evaluation of parallel algorithms on these computing systems is a necessary step towards both efficient algorithms and computing systems. Above, we have presented several state of the art approaches for performance modeling. Each model has its contributions and its drawbacks. Here we present a summary of the main issues we have identified and that motivate the development of our own performance model in chapter 5. We are mainly interested in analyzing hierarchical algorithms suitable for systems with multiple levels of parallelism, for that reason the only models of interest for us would be the hierarchical models, in particular the multi-BSP and the D-BSP models presented above. These two models does not explicit how parallel messages are handled throughout the network hierarchy. In our model we explicitly describe how messages are routed, we mainly use message grouping technique detailed in the context of the coarse-grained multicomputer (CGM) model. However we establish a different criterion for message aggregation. Details will be given in section 5.2.

Chapter 2

LU factorization with panel rank revealing pivoting

2.1 Introduction

The LU factorization is an important operation in numerical linear algebra since it is widely used for solving linear systems of equations, computing the determinant of a matrix, or as a building block of other operations. It consists of the decomposition of a matrix A into the product $A = LU$, where L is a lower triangular matrix and U is an upper triangular matrix. The performance of the LU decomposition is critical for many applications, and it has received significant attention over the years. Recently large efforts have been invested in optimizing this linear algebra kernel, in terms of both numerical stability and performance on emerging parallel architectures.

As detailed is the previous chapter, the LU decomposition can be computed using Gaussian elimination with partial pivoting, a very stable operation in practice, except for pathological cases such as the Wilkinson matrix [96, 66], the Foster matrix [50], or the Wright matrix [99]. Many papers [92, 82, 88] discuss the stability of the Gaussian elimination, and it is known [66, 56, 49] that the pivoting strategy used, such as complete pivoting, partial pivoting, or rook pivoting, has an important impact on the numerical stability of this method, which depends on a quantity referred to as the growth factor, which measures how large the elements of the input matrix are getting during elimination steps. However, in terms of performance, these pivoting strategies represent a limitation, since they require asympotically more communication than established lower bounds on communication indicate is necessary [38, 15].

In this chapter we study the LU_PRRP factorization, a novel LU decomposition algorithm based on what we call panel rank revealing pivoting (PRRP). The LU_PRRP factorization is based on a block algorithm that computes the LU decomposition as follows. At each step of the block factorization, a block of columns (panel) is factored by computing the strong rank revealing QR (RRQR) factorization [61] of its transpose. The permutation returned by the panel rank revealing factorization is applied to the rows of the input matrix, and the L factor of the panel is computed based on the R factor of the strong RRQR factorization. Then the trailing matrix is updated. In exact arithmetic, the LU_PRRP factorization first computes a block LU decomposition [40], here is one step of the block LU factorization, where we assume that A_{11} is a non singular matrix of size $b \times b$:

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} = \begin{bmatrix} I & 0 \\ L_{21} & I \end{bmatrix} \times \begin{bmatrix} A_{11} & A_{12} \\ 0 & A_{22}^{s} \end{bmatrix}$$

Our block LU factorization is based on a new pivoting strategy, the panel rank revealing pivoting. Then the U factors and the diagonal blocks of L are produced by GEPP. The factors obtained from this decomposition can be stored in place, and so the LU_PRRP factorization has the same memory requirements as standard LU and can easily replace it in any application.

We show that LU_PRRP is more stable than GEPP. Its growth factor is upper bounded by $(1 + \tau b)^{(n/b)-1}$, where b is the size of the panel, n is the number of columns of the input matrix, and τ is a parameter of the panel strong RRQR factorization. This bound is smaller than 2^{n-1}, the upper bound of the growth factor for GEPP. For example, if the size of the panel is $b = 64$ and $\tau = 2$, then $(1 + 2b)^{(n/b)-1} = (1.079)^{n-64} \ll 2^{n-1}$. In terms of cost, LU_PRRP performs only $O(n^2 b)$ more floating point operations than GEPP. In addition, our extensive numerical experiments on random matrices and on a set of special matrices show that the LU_PRRP factorization is very stable in practice and leads to modest growth factors, smaller than those obtained with GEPP.

We also discuss the backward stability of LU_PRRP using three metrics, the relative error $\|PA - LU\|/\|A\|$, the normwise backward error (2.10), and the componentwise backward error (2.11).

For the matrices in our set, the relative error, the normwise backward error, and the componentwise backward error are all of the order of 10^{-14} (with the exception of three matrices, *sprandn*, *compan*, and *Demmel*, for which the componentwise backward error is 1.3×10^{-13}, 6.9×10^{-12}, and 1.16×10^{-8} respectively). Later in this chapter, Figure 2.2 displays the ratios of these errors versus the errors of GEPP, obtained by dividing the maximum of the backward errors of LU_PRRP and the machine epsilon (2^{-53}) by the maximum of those of GEPP and the machine epsilon. For all the matrices in our set, the growth factor of LU_PRRP is always smaller than that of GEPP (with the exception of one matrix, the *compar* matrix). For random matrices, the relative error of the factorization of LU_PRRP is always smaller than that of GEPP. However, for the normwise and the componentwise backward errors, GEPP is slightly better, with a ratio of at most 2 between the two. For the set of special matrices, the ratio of the relative error is at most 1 in over 75% of cases, that is LU_PRRP is more stable than GEPP. For the rest of the 25% of the cases, the ratio is at most 3, except for one matrix (*hadamard*) for which the ratio is 23 and the backward error is on the order of 10^{-15}. The ratio of the normwise backward errors is at most 1 in over 75% of cases, and always 3.4 or smaller. The ratio of the componentwise backward errors is at most 2 in over 81% of cases, and always 3 or smaller (except for one matrix, the *compan* matrix, for which the componentwise backward error is 6.9×10^{-12} for LU_PRRP and 6.2×10^{-13} for GEPP).

The rest of this chapter is organized as follows. Section 2.2 presents LU_PRRP, the new algorithm designed to improve the numerical stability of the LU factorisation. Section 2.2.1 studies its numerical stability and derives new bounds on the growth factor. In section 2.2.2 we evaluate the behavior of the algorithm on various matrices. We also experimentally evaluate this algorithm, together with GEPP on a set of special matrices. Finally in Section 2.3, we provide final remarks, and directions for future work.

2.2 Matrix algebra

In this section we introduce the LU_PRRP factorization, an LU decomposition algorithm based on panel rank revealing pivoting strategy. It is based on a block algorithm, that factors at each step a block of columns (a panel), and then it updates the trailing matrix. The main difference between LU_PRRP and GEPP resides in the panel factorization. In the partitioned version of GEPP, the panel factorization is computed using LU with partial pivoting, while in LU_PRRP it is computed by performing a strong

RRQR factorization of its transpose. This leads to a different selection of pivot rows, and the obtained R factor is used to compute the block L factor of the panel. In exact arithmetic, LU_PRRP performs a block LU decomposition with a different pivoting scheme, which aims at improving the numerical stability of the factorization by bounding more efficiently the growth of the elements. We also discuss the numerical stability of LU_PRRP, and we show that both in theory and in practice, LU_PRRP is more stable than GEPP.

LU_PRRP is based on a block algorithm that factors the input matrix A of size $m \times n$ by traversing blocks of columns of size b. Consider the first step of the factorization, with the matrix A having the following partition,

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}, \tag{2.1}$$

where A_{11} is of size $b \times b$, A_{21} is of size $(m-b) \times b$, A_{12} is of size $b \times (n-b)$, and A_{22} is of size $(m-b) \times (n-b)$.

The main idea of the LU_PRRP factorization is to eliminate the elements below the $b \times b$ diagonal block such that the multipliers used during the update of the trailing matrix are bounded by a given threshold τ. For this, we perform a strong RRQR factorization on the transpose of the first panel of size $m \times b$ to identify a permutation matrix Π, that is b pivot rows,

$$\begin{bmatrix} A_{11} \\ A_{21} \end{bmatrix}^T \Pi = \begin{bmatrix} \bar{A}_{11} \\ \bar{A}_{21} \end{bmatrix}^T = Q \begin{bmatrix} R(1:b, 1:b) & R(1:b, b+1:m) \end{bmatrix} = Q \begin{bmatrix} R_{11} & R_{12} \end{bmatrix},$$

where $\bar{A}$ denotes the permuted matrix A. The strong RRQR factorization ensures that the quantity $R_{12}^T (R_{11}^{-1})^T$ is bounded by a given threshold τ in the max norm. The strong RRQR factorization, as described in Algorithm 6 in section 1.2.2, computes first the QR factorization with column pivoting, followed by additional swaps of the columns of the R factor and updates of the QR factorization, so that $\|R_{12}^T (R_{11}^{-1})^T\|_{\max} \leq \tau$.

After the panel factorization, the transpose of the computed permutation Π is applied to the input matrix A, and then the update of the trailing matrix is performed,

$$\bar{A} = \Pi^T A = \begin{bmatrix} I_b & \\ L_{21} & I_{m-b} \end{bmatrix} \begin{bmatrix} \bar{A}_{11} & \bar{A}_{12} \\ & \bar{A}_{22}^s \end{bmatrix}, \tag{2.2}$$

where

$$\bar{A}_{22}^s = \bar{A}_{22} - L_{21} \bar{A}_{12}. \tag{2.3}$$

Note that in exact arithmetic, we have $L_{21} = \bar{A}_{21} \bar{A}_{11}^{-1} = R_{12}^T (R_{11}^{-1})^T$. Hence the factorization in equation (2.2) is equivalent to the factorization

$$\bar{A} = \Pi^T A = \begin{bmatrix} I_b & \\ \bar{A}_{21} \bar{A}_{11}^{-1} & I_{m-b} \end{bmatrix} \begin{bmatrix} \bar{A}_{11} & \bar{A}_{12} \\ & \bar{A}_{22}^s \end{bmatrix},$$

where

$$\bar{A}_{22}^{s} = \bar{A}_{22} - \bar{A}_{21}\bar{A}_{11}^{-1}\bar{A}_{12}, \tag{2.4}$$

and $\bar{A}_{21}\bar{A}_{11}^{-1}$ was computed in a numerically stable way such that it is bounded in max norm by τ.

Since the block size is in general $b \geq 2$, performing LU_PRRP on a given matrix A first leads to a block LU factorization, with the diagonal blocks $\bar{A}_{ii}$ being square of size $b \times b$. The block LU_PRRP factorization is described in Algorithm 9.

Algorithm 9: Block LU_PRRP factorization of a matrix A of size $n \times n$ using a panel of size b

Let B_1 be the first panel $B_1 = A(1:n, 1:b)$.
Compute panel factorization $B_1^T \Pi_1 := Q_1 R_1$ using strong RRQR factorization,
$L_{21} := (R_1(1:b, 1:b)^{-1} R_1(1:b, b+1:n))^T$.
Apply the permutation matrix Π_1^T to the entire matrix, $A = \Pi_1^T A$.
$U_{11} := A(1:b, 1:b)$, $U_{12} := A(1:b, b+1:n)$.
Update the trailing matrix,
$A(b+1:n, b+1:n) = A(b+1:n, b+1:n) - L_{21}U_{12}$.
Compute the block LU_PRRP factorization of $A(b+1:n, b+1:n)$, recursively.

An additional Gaussian elimination with partial pivoting is performed on the $b \times b$ diagonal block $\bar{A}_{11}$ as well as the update of the corresponding trailing matrix $\bar{A}_{12}$. Then the decomposition obtained after the elimination of the first panel of the input matrix A is

$$\bar{A} = \Pi^T A = \begin{bmatrix} I_b & \\ L_{21} & I_{m-b} \end{bmatrix} \begin{bmatrix} \bar{A}_{11} & \bar{A}_{12} \\ & \bar{A}_{22}^{s} \end{bmatrix},$$

$$\bar{A} = \Pi^T A = \begin{bmatrix} I_b & \\ L_{21} & I_{m-b} \end{bmatrix} \begin{bmatrix} L_{11} & \\ & I_{m-b} \end{bmatrix} \begin{bmatrix} U_{11} & U_{12} \\ & \bar{A}_{22}^{s} \end{bmatrix}, \tag{2.5}$$

where L_{11} is a lower triangular $b \times b$ matrix with unit diagonal and U_{11} is an upper triangular $b \times b$ matrix. We show in section 2.2.1 that this step does not affect the stability of the LU_PRRP factorization. Note that the factors L and U can be stored in place, and so LU_PRRP has the same memory requirements as the standard LU decomposition and can easily replace it in any application.

Algorithm 10 presents the LU_PRRP factorization of a matrix A of size $n \times n$ partitioned into (n/b) panels. The number of floating-point operations performed by this algorithm is

$$\#flops = \frac{2}{3}n^3 + O(n^2 b),$$

which is only $O(n^2 b)$ more floating point operations than GEPP. The detailed counts are presented in Appendix B .1. When the QR factorization with column pivoting is sufficient to obtain the desired bound for each panel factorization, and no additional swaps are performed, the total cost is

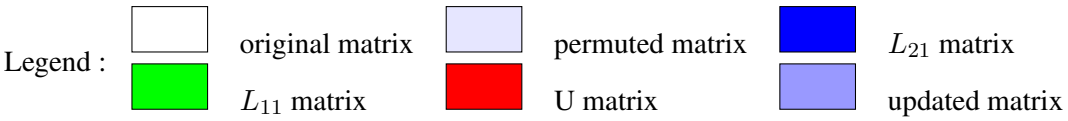
$$\#flops = \frac{2}{3}n^3 + \frac{3}{2}n^2 b.$$

Algorithm 10: LU_PRRP factorization of a matrix A of size $n \times n$

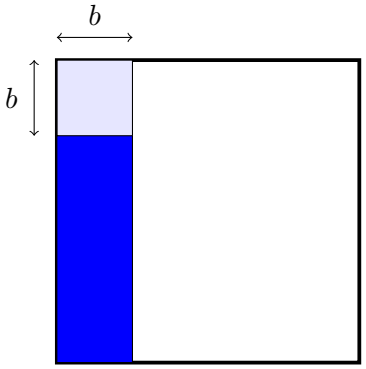
for *j from 1 to* $\frac{n}{b}$ **do**

Let $A_j = A((j-1)b+1:n, (j-1)b+1:jb)$ be the current panel.
Compute panel factorization $A_j^T \Pi_j := Q_j R_j$ using strong RRQR factorization,
$L_{2j} := (R_j(1:b, 1:b)^{-1} R_j(1:b, b+1:n-(j-1)b))^T$.
Apply the permutation matrix Π_j^T to the entire matrix, $A = \Pi_j^T A$.
Update the trailing matrix,
$A(jb+1:n, jb+1:n) -= L_{2j} A((j-1)b+1:jb, jb+1:n)$.
Let A_{jj} be the current $b \times b$ diagonal block,
$A_{jj} = A((j-1)b+1:jb, (j-1)b+1:jb)$.
Compute $A_{jj} = \Pi_{jj} L_{jj} U_{jj}$ using GEPP.
Compute $U((j-1)b+1:jb, jb+1:n) = L_{jj}^{-1} \Pi_{jj}^T A((j-1)b+1:jb, jb+1:n)$.

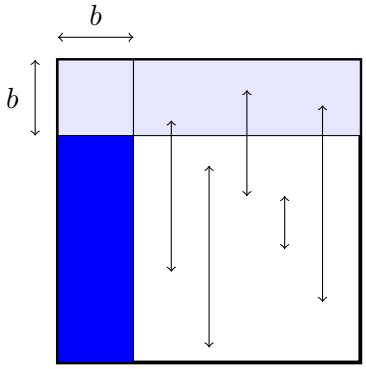
As detailed previously, the LU_PRRP factorization involves 5 steps. We illustrate this different steps in the following figures, which use the legend described below.



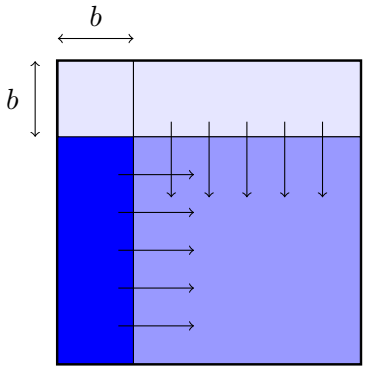
1. The L_{21} factor is computed by applying a Strong Rank Revealing QR (Strong RRQR) on the transpose of the first panel $\begin{pmatrix} A_{11} \\ A_{21} \end{pmatrix}$ (the pivot search is done on the entire panel), $L_{21} = \bar{A}_{21} \bar{A}_{11}^{-1} = R_{12}^T (R_{11}^{-1})^T$.



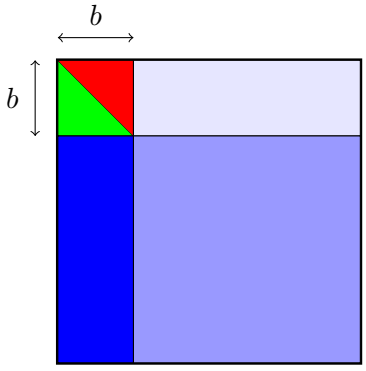
2. We apply the permutation on the rest of the matrix, $\begin{pmatrix} A_{12} \\ A_{22} \end{pmatrix}$.



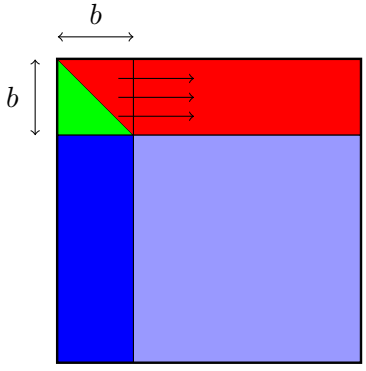
3. We update the trailing matrix as follows, $\bar{A}_{22}^s = \bar{A}_{22} - \bar{A}_{21}\bar{A}_{11}^{-1}\bar{A}_{12}$.



4. We perform Gaussian elimination with partial pivoting (GEPP) on the $b \times b$ diagonal block $\bar{A}_{11}$.



5. We compute U_{12} by solving the triangular linear system $U_{12} = L_{11}^{-1}\bar{A}_{12}$.



6. We perform the same steps 1 to 5 on the trailing matrix.

2.2.1 Numerical stability

In this section we discuss the numerical stability of the LU_PRRP factorization. The stability of an LU decomposition mainly depends on the growth factor. In his backward error analysis [96], Wilkinson proved that the computed solution $\hat{x}$ of the linear system $Ax = b$, where A is of size $n \times n$, obtained by Gaussian elimination with partial pivoting or complete pivoting satisfies

$$(A + \Delta A)\hat{x} = b, \quad \|\Delta A\|_\infty \leq p(n) g_W u \|A\|_\infty.$$

Here, $p(n)$ is a cubic polynomial, u is the machine precision, and g_W is the growth factor defined by

$$g_W = \frac{\max_{i,j,k} |a_{ij}^{(k)}|}{\max_{i,j} |a_{ij}|} \geq 1,$$

where $a_{ij}^{(k)}$ denotes the entry in position (i, j) obtained after k steps of elimination. Thus the growth factor measures the growth of the elements during the elimination. The LU factorization is backward stable if g_W is small. Lemma 9.6 of [65] (section 9.3) states a more general result, showing that the LU factorization without pivoting of A is backward stable if the growth factor is small. Wilkinson [96] showed that for partial pivoting, the growth factor $g_W \leq 2^{n-1}$, and this bound is attainable. He also showed that for complete pivoting, the upper bound satisfies $g_W \leq n^{1/2}(2.3^{1/2}.....n^{1/(n-1)})^{1/2} \sim cn^{1/2}n^{1/4\log n}$. In practice the growth factors are much smaller than the upper bounds.

In the following, we derive an upper bound for the growth factor for the LU_PRRP factorization. We use the same notation as in the previous section and we assume without loss of generality that the permutation matrix is the identity. It is easy to see that the growth factor obtained after the elimination of the first panel is bounded by $|1 + \tau b|$. At the k-th step of the block factorization, the active matrix $A^{(k)}$ is of size $(m - (k-1)b) \times (n - (k-1)b)$, and the decomposition performed at this step can be written as

$$A^{(k)} = \begin{bmatrix} A_{11}^{(k)} & A_{12}^{(k)} \\ A_{21}^{(k)} & A_{22}^{(k)} \end{bmatrix} = \begin{bmatrix} I_b & \\ L_{21}^{(k)} & I_{m-(k+1)b} \end{bmatrix} \begin{bmatrix} A_{11}^{(k)} & A_{12}^{(k)} \\ & A_{22}^{(k)s} \end{bmatrix}.$$

The active matrix at the $(k+1)$-st step is $A_{22}^{(k+1)} = A_{22}^{(k)s} = A_{22}^{(k)} - L_{21}^{(k)} A_{12}^{(k)}$. Then $\max_{i,j} |a_{i,j}^{(k+1)}| \leq \max_{i,j} |a_{i,j}^{(k)}|(1 + \tau b)$ with $\max_{i,j} |L_{21}^{(k)}(i, j)| \leq \tau$ and we have

$$g_W{}^{(k+1)} \leq g_W{}^{(k)}(1 + \tau b). \tag{2.6}$$

Induction on equation (2.6) leads to a growth factor of the block factorization based on LU_PRRP performed on the $\frac{n}{b}$ panels of the matrix A that satisfies

$$g_W \leq (1 + \tau b)^{(n/b)-1}. \tag{2.7}$$

As explained in section 2.2, the LU_PRRP factorization leads first to a block LU factorization, which is completed with additional GEPP factorizations of the diagonal $b \times b$ blocks and updates of corresponding blocks of rows of the trailing matrix. These additional factorizations lead to a growth factor bounded by 2^{b-1} on the trailing blocks of rows. We note that for the last $b \times b$ diagonal block, the element growth generated by GEPP is added to that generated by the first steps of LU_PRRP, that is the block LU decomposition. Thus the maximum element of the factor U in equation (2.5) is bounded by

$$g_W \leq (1 + \tau b)^{(n/b)-1} \times 2^{b-1}. \tag{2.8}$$

We note that when $b \ll n$, the growth factor of the entire factorization is still bounded by $(1+\tau b)^{(n/b)-1}$.

The improvement of the upper bound of the growth factor of the block factorization based on LU_PRRP with respect to GEPP is illustrated in Table 2.1, where the panel size varies from 8 to 128, and the parameter τ is set to 2. The worst case growth factor becomes arbitrarily smaller than for GEPP, for $b \geq 64$.

Table 2.1: Upper bounds of the growth factor g_W obtained from factoring a matrix of size $m \times n$ using the block factorization based on LU_PRRP with different panel sizes and $\tau = 2$.

b	8	16	32	64	128
g_W	$(1.425)^{n-8}$	$(1.244)^{n-16}$	$(1.139)^{n-32}$	$(1.078)^{n-64}$	$(1.044)^{n-128}$

Despite the complexity of our algorithm in pivot selection, we still compute an LU factorization, only with different pivots. Consequently, the rounding error analysis for LU factorization still applies (see, for example, [36]), which indicates that element growth is the only factor controlling the numerical stability of our algorithm.

2.2.2 Experimental results

We measure the stability of the LU_PRRP factorization experimentally on a large set of test matrices by using the growth factor, the normwise backward stability, and the componentwise backward stability. The tests are performed using MATLAB. In the tests, in most of the cases, the panel factorization is performed by using the QR with column pivoting factorization instead of the strong RRQR factorization. This is because in practice $R_{12}^T(R_{11}^{-1})^T$ is already well bounded after performing the RRQR factorization with column pivoting ($\|R_{12}^T(R_{11}^{-1})^T\|_{\max}$ is rarely bigger than 3 for random matrices). Hence no additional swaps are needed to ensure that the elements are well bounded. However, for the ill-conditionned special matrices (condition number $\geq 10^{14}$), to get small growth factors, we perform the panel factorization by using the strong RRQR factorization. In fact, for these cases, QR with column pivoting does not ensure a small bound for $R_{12}^T(R_{11}^{-1})^T$.

We use a collection of matrices that includes random matrices, a set of special matrices described in Table 3, and several pathological matrices on which Gaussian elimination with partial pivoting fails because of large growth factors. The set of special matrices includes ill-conditioned matrices as well as sparse matrices. The pathological matrices considered are the Wilkinson matrix and two matrices arising from practical applications, presented by Foster [50] and Wright [99], for which the growth factor of GEPP grows exponentially. The Wilkinson matrix was constructed to attain the upper bound of the growth factor of GEPP [96, 66], and a general layout of such a matrix is

$$A = diag(\pm 1) \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 & 1 \\ -1 & 1 & 0 & \ldots & 0 & 1 \\ -1 & -1 & 1 & \ddots & \vdots & \vdots \\ \vdots & \vdots & \ddots & \ddots & 0 & 1 \\ -1 & -1 & \cdots & -1 & 1 & 1 \\ -1 & -1 & \cdots & -1 & -1 & 1 \end{bmatrix} \times \begin{bmatrix} & & & 0 \\ & T & & \vdots \\ & & & 0 \\ 0 & \cdots & 0 & \theta \end{bmatrix},$$

where T is an $(n-1)\times(n-1)$ non-singular upper triangular matrix and $\theta = \max|a_{ij}|$. We also test a generalized Wilkinson matrix, the general form of such a matrix is

$$A = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 & 1 \\ 0 & 1 & 0 & \ldots & 0 & 1 \\ & & 1 & \ddots & \vdots & \vdots \\ \vdots & & \ddots & \ddots & 0 & 1 \\ & & & & 1 & 1 \\ 0 & & \cdots & & 0 & 1 \end{bmatrix} + T^T,$$

where T is a $n \times n$ strictly upper triangular matrix. The MATLAB code of the matrix A is detailed in Appendix D.

The Foster matrix represents a concrete physical example that arises from using the quadrature method to solve a certain Volterra integral equation and it is of the form

$$A = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 & -\frac{1}{c} \\ -\frac{kh}{2} & 1-\frac{kh}{2} & 0 & \ldots & 0 & -\frac{1}{c} \\ -\frac{kh}{2} & -kh & 1-\frac{kh}{2} & \ddots & \vdots & \vdots \\ \vdots & \vdots & & \ddots & 0 & -\frac{1}{c} \\ -\frac{kh}{2} & -kh & \cdots & -kh & 1-\frac{kh}{2} & -\frac{1}{c} \\ -\frac{kh}{2} & -kh & \cdots & -kh & -kh & 1-\frac{1}{c}-\frac{kh}{2} \end{bmatrix}.$$

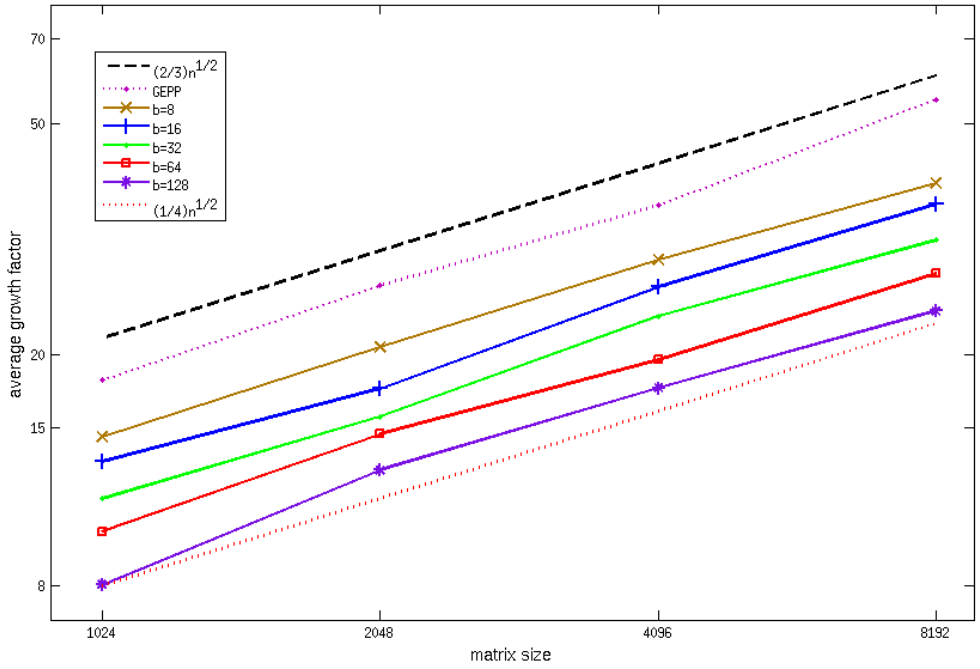
Wright [99] discusses two-point boundary value problems for which standard solution techniques give rise to matrices with exponential growth factor when Gaussian elimination with partial pivoting is used. This kind of problems arise for example from the multiple shooting algorithm [23]. A particular example of this problem is presented by the following matrix,

$$A = \begin{bmatrix} I & & & & I \\ -e^{Mh} & I & & & 0 \\ & -e^{Mh} & I & & \vdots \\ & & \ddots & \ddots & 0 \\ & & & -e^{Mh} & I \end{bmatrix},$$

where $e^{Mh} = I + Mh + O(h^2)$.

The experimental results show that the LU_PRRP factorization is very stable. Figure 2.1 displays the growth factor of LU_PRRP for random matrices of size varying from 1024 to 8192 and for sizes of the panel varying from 8 to 128. We observe that the smaller the size of the panel is, the bigger the element growth is. In fact, for a smaller size of the panel, the number of panels and the number of updates on the trailing matrix is bigger, and this leads to a larger growth factor. But for all panel sizes, the growth factor of LU_PRRP is smaller than the growth factor of GEPP. For example, for a random matrix of size 4096 and a panel of size 64, the growth factor is only about 19, which is smaller than the growth factor obtained by GEPP, and as expected, much smaller than the theoretical upper bound of $(1.078)^{4095}$.

Figure 2.1: Growth factor g_W of the LU_PRRP factorization of random matrices.



Tables 4 and 5 in Appendix A present more detailed results showing the stability of the LU_PRRP factorization for random matrices and a set of special matrices. There, we include different metrics, such as the norm of the factors, the value of their maximum element and the backward error of the LU factorization.

We evaluate the normwise backward stability by computing an accuracy test as performed in the HPL (High-Performance Linpack) benchmark [45], and denoted as HPL3.

$$\text{HPL3} \quad = \quad ||Ax - b||_\infty / (\epsilon ||A||_\infty ||x||_\infty * N). \tag{2.9}$$

In HPL, the method is considered to be accurate if the value of HPL3 is smaller than 16. More generally, the value should be of order $O(1)$. For the LU_PRRP factorization HPL3 is at most 1.60×10^{-2}. We also display the normwise backward error, using the 1-norm,

$$\eta := \frac{||r||}{||A||\ ||\hat{x}|| + ||b||}, \tag{2.10}$$

and the componentwise backward error

$$w := \max_i \frac{|r_i|}{(|A|\ |\hat{x}| + |b|)_i}, \tag{2.11}$$

where $\hat{x}$ is a computed solution to $Ax = b$, and $r = b - A\hat{x}$ is the computed residual. For our tests residuals are computed with double-working precision.

Figure 2.2 summarizes all our stability results for LU_PRRP. This figure displays the ratio of the maximum between the backward error and machine epsilon of LU_PRRP versus GEPP. The backward

error is measured using three metrics, the relative error $\|PA - LU\|/\|A\|$, the normwise backward error η, and the componentwise backward error w of LU_PRRP versus GEPP, and the machine epsilon. We take the maximum of the computed error with epsilon since smaller values are mostly roundoff error, and so taking ratios can lead to extreme values with little reliability. Results for all the matrices in our test set are presented, that is 20 random matrices for which results are presented in Table 4, and 37 special matrices for which results are presented in Tables 11 and 5. This figure shows that for random matrices, almost all ratios are between 0.5 and 2. For special matrices, there are few outliers, up to 23.71 (GEPP is more stable) for the backward error ratio of the special matrix *hadamard* and down to 2.12×10^{-2} (LU_PRRP is more stable) for the backward error ratio of the special matrix *molar*.

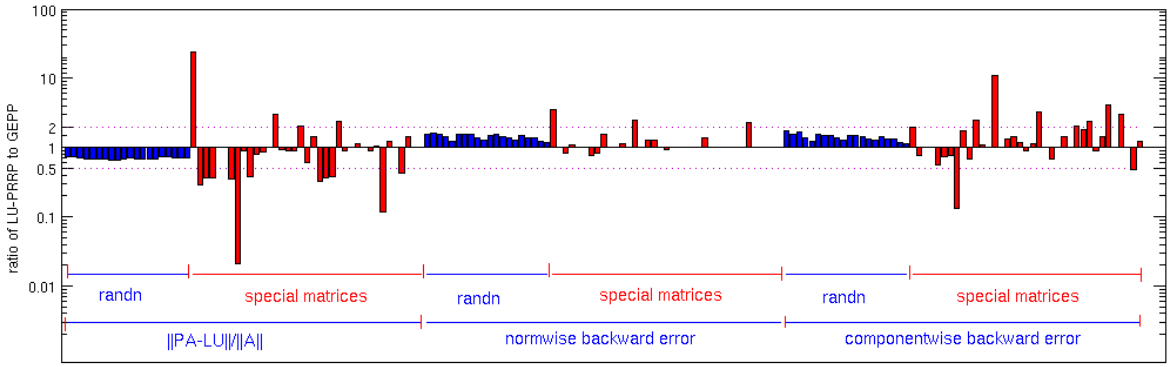


Figure 2.2: A summary of all our experimental data, showing the ratio between max(LU_PRRP's backward error, machine epsilon) and max(GEPP's backward error, machine epsilon) for all the test matrices in our set. Bars above $10^0 = 1$ mean that LU_PRRP's backward error is larger, and bars below 1 mean that GEPP's backward error is larger. The test matrices are further labeled either as "randn", which are randomly generated, or "special", listed in Table 3.

We consider now pathological matrices on which GEPP fails. Table 2.2 presents results for the linear solver using the LU_PRRP factorization for a Wilkinson matrix [97] of size 2048 with a size of the panel varying from 8 to 128. The growth factor is 1 and the relative error $\|PA - LU\|/\|A\|$ is on the order of 10^{-19}. Table 2.3 presents results for the linear solver using the LU_PRRP algorithm for a generalized Wilkinson matrix of size 2048 with a size of the panel varying from 8 to 128.

n	b	g_W	$\|\|U\|\|_1$	$\|\|U^{-1}\|\|_1$	$\|\|L\|\|_1$	$\|\|L^{-1}\|\|_1$	$\frac{\|\|PA-LU\|\|_F}{\|\|A\|\|_F}$
2048	128	1	1.02e+03	6.09e+00	1	1.95e+00	4.25e-20
	64	1	1.02e+03	6.09e+00	1	1.95e+00	5.29e-20
	32	1	1.02e+03	6.09e+00	1	1.95e+00	8.63e-20
	16	1	1.02e+03	6.09e+00	1	1.95e+00	1.13e-19
	8	1	1.02e+03	6.09e+00	1	1.95e+00	1.57e-19

Table 2.2: Stability of the LU_PRRP factorization of a Wilkinson matrix on which GEPP fails.

For the Foster matrix, it was shown that when $c = 1$ and $kh = 2/3$, the growth factor of GEPP is $(\frac{2}{3})(2^{n-1} - 1)$, which is close to the maximum theoretical growth factor of GEPP of 2^{n-1}. Table 2.4 presents results for the linear solver using the LU_PRRP factorization for a Foster matrix of size 2048 with a size of the panel varying from 8 to 128 ($c = 1$, $h = 1$ and $k = 2/3$). According to the obtained

n	b	g_W	$\|\|U\|\|_1$	$\|\|U^{-1}\|\|_1$	$\|\|L\|\|_1$	$\|\|L^{-1}\|\|_1$	$\frac{\|\|PA-LU\|\|_F}{\|\|A\|\|_F}$
2048	128	2.69	1.23e+03	1.39e+02	1.21e+03	1.17e+03	1.05e-15
	64	2.61	9.09e+02	1.12e+02	1.36e+03	1.15e+03	9.43e-16
	32	2.41	8.20e+02	1.28e+02	1.39e+03	9.77e+02	5.53e-16
	16	4.08	1.27e+03	2.79e+02	1.41e+03	1.19e+03	7.92e-16
	8	3.35	1.36e+03	2.19e+02	1.41e+03	1.73e+03	1.02e-15

Table 2.3: Stability of the LU_PRRP factorization of a generalized Wilkinson matrix on which GEPP fails.

results, LU_PRRP gives a modest growth factor of 2.66 for this practical matrix, while GEPP has a growth factor of 10^{18} for the same parameters.

n	b	g_W	$\|\|U\|\|_1$	$\|\|U^{-1}\|\|_1$	$\|\|L\|\|_1$	$\|\|L^{-1}\|\|_1$	$\frac{\|\|PA-LU\|\|_F}{\|\|A\|\|_F}$
2048	128	2.66	1.28e+03	1.87e+00	1.92e+03	1.92e+03	4.67e-16
	64	2.66	1.19e+03	1.87e+00	1.98e+03	1.79e+03	2.64e-16
	32	2.66	4.33e+01	1.87e+00	2.01e+03	3.30e+01	2.83e-16
	16	2.66	1.35e+03	1.87e+00	2.03e+03	2.03e+00	2.38e-16
	8	2.66	1.35e+03	1.87e+00	2.04e+03	2.02e+00	5.36e-17

Table 2.4: Stability of the LU_PRRP factorization of a practical matrix (Foster) on which GEPP fails.

For matrices arising from the two-point boundary value problems described by Wright, it was shown that when h is chosen small enough such that all elements of e^{Mh} are less than 1 in magnitude, the growth factor obtained using GEPP is exponential. For our experiment the matrix $M = \begin{bmatrix} -\frac{1}{6} & 1 \\ 1 & -\frac{1}{6} \end{bmatrix}$, that is $e^{Mh} \approx \begin{bmatrix} 1-\frac{h}{6} & h \\ h & 1-\frac{h}{6} \end{bmatrix}$, and $h = 0.3$. Table 2.5 presents results for the linear solver using the LU_PRRP factorization for a Wright matrix of size 2048 with a size of the panel varying from 8 to 128. According to the obtained results, again LU_PRRP gives minimum possible pivot growth 1 for this practical matrix, compared to the GEPP method which leads to a growth factor of 10^{95} using the same parameters.

n	b	g_W	$\|\|U\|\|_1$	$\|\|U^{-1}\|\|_1$	$\|\|L\|\|_1$	$\|\|L^{-1}\|\|_1$	$\frac{\|\|PA-LU\|\|_F}{\|\|A\|\|_F}$
2048	128	1	3.25e+00	8.00e+00	2.00e+00	2.00e+00	4.08e-17
	64	1	3.25e+00	8.00e+00	2.00e+00	2.00e+00	4.08e-17
	32	1	3.25e+00	8.00e+00	2.05e+00	2.07e+00	6.65e-17
	16	1	3.25e+00	8.00e+00	2.32e+00	2.44e+00	1.04e-16
	8	1	3.40e+00	8.00e+00	2.62e+00	3.65e+00	1.26e-16

Table 2.5: Stability of the LU_PRRP factorization on a practical matrix (Wright) on which GEPP fails.

All the previous tests show that the LU_PRRP factorization is very stable for random, and for more special matrices, and it also gives modest growth factor for the pathological matrices on which GEPP fails. We note that we were not able to find matrices for which LU_PRRP attains the upper bound of $(1+\tau b)^{(n/b)-1}$ for the growth factor.

2.3 Conclusion

In this chapter, we have introduced an LU factorization which is more stable than Gaussian elimination with partial pivoting in terms of the upper bound of the growth factor. It is also very stable in practice for various classes of matrices, including pathological cases on which GEPP fails. For example LU_PRRP gives modest growth factor for wilkinson matrix, while GEPP gives exponential growth. Note that we are currently working on extending the backward stability analysis of block LU factorization provided in [40] to our block LU_PRRP algorithm. As in Gaussian elimination with partial pivoting, the panel factorization in LU_PRRP is a bottleneck in terms of latency. Hence the strong rank revealing QR is performed on the transpose of the panel. To overcome this drawback in terms of communication cost, we have further developed a communication avoiding version of LU_PRRP. This algorithm is detailed in the next chapter 3.

As future work we intend to develop routines that implement LU_PRRP and that will be integrated in LAPACK and ScaLAPACK. That is a sequential and a parallel code for this algorithm. We are also considering different other aspects of the numerical stability including the design of a matrix that attains the growth factor upper bound, a deeper study of the behavior of LU_PRRP on sparse matrices and the amount of the fill in that it could generate.
An other direction is the design of an incomplete LU preconditioner based on the panel rank revealing pivoting (ILU0_PRRP). we believe that such a preconditioner would be more stable than ILU0 and hence it would be very useful for the resolution of sparse linear systems.

Chapter 3

Communication avoiding LU factorization with panel rank revealing pivoting

3.1 Introduction

Technological trends show that computing floating point operations is becoming exponentially faster than moving data from the memory where they are stored to the place where the computation occurs. Due to this, the communication becomes in many cases a dominant factor of the runtime of an algorithm, that leads to a loss of its efficiency. This is a problem for both sequential algorithms, where data needs to be moved between different levels of the memory hierarchy, and parallel algorithms, where data needs to be communicated between processors.

This challenging problem has prompted research on algorithms that reduce the communication to a minimum, while being numerically as stable as classic algorithms, and without increasing significantly the number of floating point operations performed [38, 59]. We refer to these algorithms as communication avoiding. One of the first such algorithms is the communication avoiding LU factorization (CALU) [59, 60]. This algorithm is optimal in terms of communication, that is it performs polylogarithmic factors more than the theoretical lower bounds on communication [38, 15]. Thus, it brings considerable improvements to the performance of the LU factorization compared to the classic routines that perform the LU decomposition such as the PDGETRF routine of ScaLAPACK, thanks to a novel pivoting strategy referred to as tournament pivoting. It was shown that CALU is faster in practice than the corresponding routine PDGETRF implemented in libraries as ScaLAPACK or vendor libraries, on both distributed [59] and shared memory computers [42], especially on tall and skinny matrices. While in practice CALU is as stable as GEPP, in theory the upper bound of its growth factor is worse than that obtained with GEPP. One of our goals is to design an algorithm that minimizes communication and that has a smaller upper bound of its growth factor than CALU.

In this chapter we introduce the CALU_PRRP factorization, the communication avoiding version of LU_PRRP, the algorithm discussed in chapter 2. CALU_PRRP is based on tournament pivoting, a strategy introduced in [60] in the context of CALU, a communication avoiding version of GEPP. With tournament pivoting, the panel factorization is performed in two steps. The first step selects b pivot rows from the entire panel at a minimum communication cost. For this, sets of b candidate pivot rows are selected from blocks of the panel, which are then combined together through a reduction-like procedure, until a final set of b pivot rows are chosen. CALU_PRRP uses the strong RRQR factorization to select b rows at each step of the reduction operation, while CALU is based on GEPP. In the second step of the

panel factorization, the pivot rows are permuted to the diagonal positions, and the QR factorization with no pivoting of the transpose of the panel is computed. Then the algorithm proceeds as the LU_PRRP factorization introduced previously. Note that the usage of the strong RRQR factorization ensures that bounds are respected locally at each step of the reduction operation, but it does not ensure that the growth factor is bounded globally as in LU_PRRP.

To address the numerical stability of the communication avoiding factorization, we show that performing the CALU_PRRP factorization of a matrix A is equivalent to performing the LU_PRRP factorization of a larger matrix, formed by blocks of A and zeros. This equivalence suggests that CALU_PRRP will behave as LU_PRRP in practice and it will be stable. The dimension and the sparsity structure of the larger matrix also allows us to upper bound the growth factor of CALU_PRRP by $(1+\tau b)^{\frac{n}{b}(H+1)-1}$, where in addition to the parameters n, b, and τ previously defined, H is the height of the reduction tree used during tournament pivoting.

This algorithm has two significant advantages over other classic factorization algorithms. First, it minimizes communication, and hence it will be more efficient than LU_PRRP and GEPP on architectures where communication is expensive. Here communication refers to both latency and bandwidth costs of moving data between levels of the memory hierarchy in the sequential case, and the cost of moving data between processors in the parallel case. Second, it is more stable than CALU. Theoretically, the upper bound of the growth factor of CALU_PRRP is smaller than that of CALU, for a reduction tree with a same height. More importantly, there are cases of interest for which it is smaller than that of GEPP as well. Given a reduction tree of height $H = \log P$, where P is the number of processors on which the algorithm is executed, the panel size b and the parameter τ can be chosen such that the upper bound of the growth factor is smaller than 2^{n-1}. Extensive experimental results show that CALU_PRRP is as stable as LU_PRRP, GEPP, and CALU on random matrices and a set of special matrices. Its growth factor is slightly smaller than that of CALU. In addition, it is also stable for matrices on which GEPP fails.

As for the LU_PRRP factorization, we discuss the stability of CALU_PRRP using three metrics. For the set of matrices we tested, the relative error is at most 9.14×10^{-14}, the normwise backward error is at most 1.37×10^{-14}, and the componentwise backward error is at most 1.14×10^{-8} for Demmel matrix. Figure 3.4 displays the ratios of the errors with respect to those of GEPP, obtained by dividing the maximum of the backward errors of CALU_PRRP and the machine epsilon by the maximum of those of GEPP and the machine epsilon. For random matrices, all the backward error ratios are at most 2.4. For the set of special matrices, the ratios of the relative error are at most 1 in over 62% of cases, and always smaller than 2, except for 8% of cases, where the ratios are between 2.4 and 24.2. The ratios of the normwise backward errors are at most 1 in over 75% of cases, and always 3.9 or smaller. The ratios of componentwise backward errors are at most 1 in over 47% of cases, and always 3 or smaller, except for 7 ratios which have values up to 74.

We also discuss other versions of LU_PRRP that minimize communication, but can be less stable than CALU_PRRP, our method of choice for reducing communication. In this different version, the panel factorization is performed only once, during which its off-diagonal blocks are annihilated using a reduce-like operation, with the strong RRQR factorization being the operator used at each step of the reduction. Every such factorization of a block of rows of the panel leads to the update of a block of rows of the trailing matrix. Independently of the shape of the reduction tree, the upper bound of the growth factor of this method is the same as that of LU_PRRP. This is because at every step of the algorithm,

a row of the current trailing matrix is updated only once. We refer to the version based on a binary reduction tree as block parallel LU_PRRP, and to the version based on a flat tree as block pairwise LU_PRRP. There are similarities between these two algorithms, the LU factorization based on block parallel pivoting (an unstable factorization), and the LU factorization based on block pairwise pivoting (whose stability is still under investigation) [85, 92, 16].

All these methods perform the panel factorization as a reduction operation, and the factorization performed at every step of the reduction leads to an update of the trailing matrix. However, in block parallel pivoting and block pairwise pivoting, GEPP is used at every step of the reduction, and hence U factors are combined together during the reduction phase. While in the block parallel and block pairwise LU_PRRP, the reduction operates always on original rows of the current panel.

We note that despite having better bounds, the block parallel LU_PRRP based on a binary reduction tree of height $H = \log P$ is unstable for certain values of the panel size b and the number of processors P. The block pairwise LU_PRRP based on a flat tree of height $H = \frac{n}{b}$ appears to be more stable. The growth factor is larger than that of CALU_PRRP, but it is smaller than n for the sizes of the matrices in our test set. Hence, potentially this version can be more stable than block pairwise pivoting, but requires further investigation.

The remainder of the chapter is organized as follows. Section 3.2 presents the algebra of CALU_PRRP, a communication avoiding version of LU_PRRP. It describes similarities between CALU_PRRP and LU_PRRP and it discusses its stability. The communication optimality of CALU_PRRP is shown in section 3.3, where we also compare its performance model with that of the CALU algorithm. Section 3.4 discusses two alternative algorithms that can also reduce communication, but can be less stable in practice. Finally section 3.5 presents our conclusions and give some perspectives.

3.2 Communication avoiding LU_PRRP

In this section we present a version of the LU_PRRP algorithm that minimizes communication, and so it will be more efficient than LU_PRRP and GEPP on architectures where communication is expensive. We show in this section that this algorithm is more stable than CALU, an existing communication avoiding algorithm for computing the LU factorization [60]. More importantly, its parallel version is also more stable than GEPP (under certain conditions).

3.2.1 Matrix algebra

CALU_PRRP is a block algorithm that uses tournament pivoting, a strategy introduced in [60] that allows to minimize communication. As in LU_PRRP, at each step the factorization of the current panel is computed, and then the trailing matrix is updated. However, in CALU_PRRP the panel factorization is performed in two steps. The first step, which is a preprocessing step, uses a reduction operation to identify b pivot rows with a minimum amount of communication. The strong RRQR factorization is the operator used at each node of the reduction tree to select a new set of b candidate pivot rows from the candidate pivot rows selected at previous stages of the reduction. The final b pivot rows are permuted into the first positions, and then the QR factorization with no pivoting of the transpose of the entire panel is computed.

In the following we illustrate tournament pivoting on the first panel, with the input matrix A parti-

tioned as in equation (2.1). We consider that the first panel is partitioned into $P = 4$ blocks of rows,

$$A(:, 1:b) = \begin{bmatrix} A_{00} \\ A_{10} \\ A_{20} \\ A_{30} \end{bmatrix}.$$

The preprocessing step uses a binary reduction tree in our example, and we number the levels of the binary reduction tree starting from 0. At the leaves of the reduction tree, a set of b candidate rows are selected from each block of rows A_{i0} by performing the strong RRQR factorization on the transpose of each block A_{i0}. This gives the following decomposition,

$$\begin{bmatrix} A_{00}^T\Pi_{00} \\ A_{10}^T\Pi_{10} \\ A_{20}^T\Pi_{20} \\ A_{30}^T\Pi_{30} \end{bmatrix} = \begin{bmatrix} Q_{00}R_{00} \\ Q_{10}R_{10} \\ Q_{20}R_{20} \\ Q_{30}R_{30} \end{bmatrix},$$

which can be written as

$$A(:, 1:b)^T\bar{\Pi}_0 = A(:, 1:b)^T \begin{bmatrix} \Pi_{00} & & & \\ & \Pi_{10} & & \\ & & \Pi_{20} & \\ & & & \Pi_{30} \end{bmatrix} = \begin{bmatrix} Q_{00}R_{00} \\ Q_{10}R_{10} \\ Q_{20}R_{20} \\ Q_{30}R_{30} \end{bmatrix},$$

where $\bar{\Pi}_0$ is an $m \times m$ permutation matrix with diagonal blocks of size $(m/P) \times (m/P)$, Q_{i0} is an orthogonal matrix of size $b \times b$, and each factor R_{i0} is an upper triangular matrix of size $b \times (m/P)$.

There are now 4 sets of b pivot candidate rows. At the next level of the binary reduction tree, two matrices A_{01} and A_{11} are formed by combining together two sets of candidate rows,

$$\begin{aligned} A_{01} &= \begin{bmatrix} (\Pi_{00}^T A_{00})(1:b, 1:b) \\ (\Pi_{10}^T A_{10})(1:b, 1:b) \end{bmatrix} \\ A_{11} &= \begin{bmatrix} (\Pi_{20}^T A_{20})(1:b, 1:b) \\ (\Pi_{30}^T A_{30})(1:b, 1:b) \end{bmatrix}. \end{aligned}$$

Two new sets of candidate rows are identified by performing the strong RRQR factorization of each matrix A_{01} and A_{11},

$$\begin{aligned} A_{01}^T\Pi_{01} &= Q_{01}R_{01}, \\ A_{11}^T\Pi_{11} &= Q_{11}R_{11}, \end{aligned}$$

where Π_{10}, Π_{11} are permutation matrices of size $2b \times 2b$, Q_{01}, Q_{11} are orthogonal matrices of size $b \times b$, and R_{01}, R_{11} are upper triangular factors of size $b \times 2b$.

The final b pivot rows are obtained by performing one last strong RRQR factorization on the transpose of the following $b \times 2b$ matrix :

$$A_{02} = \begin{bmatrix} (\Pi_{01}^T A_{01})(1:b, 1:b) \\ (\Pi_{11}^T A_{11})(1:b, 1:b) \end{bmatrix},$$

that is

$$A_{02}^T\Pi_{02} = Q_{02}R_{02},$$

where Π_{02} is a permutation matrix of size $2b \times 2b$, Q_{02} is an orthogonal matrix of size $b \times b$, and R_{02} is an upper triangular matrix of size $b \times 2b$. This operation is performed at the root of the binary reduction tree, and this ends the first step of the panel factorization. In the second step, the final pivot rows identified by tournament pivoting are permuted to the first positions of A,

$$\bar{A} = \bar{\Pi}^T A = \bar{\Pi}_2^T \bar{\Pi}_1^T \bar{\Pi}_0^T A,$$

where the matrices $\bar{\Pi}_i$ are obtained by extending the matrices $\bar{\Pi}$ to the dimension $m \times m$, that is

$$\bar{\Pi}_1 = \begin{bmatrix} \bar{\Pi}_{01} & \\ & \bar{\Pi}_{11} \end{bmatrix},$$

with $\bar{\Pi}_{i1}$, for $i = 0, 1$ formed as

$$\bar{\Pi}_{i1} = \begin{bmatrix} \Pi_{i1}(1:b, 1:b) & & \Pi_{i1}(1:b, b+1:2b) & \\ & I_{\frac{m}{P}-b} & & \\ \Pi_{i1}(b+1:2b, 1:b) & & \Pi_{i1}(b+1:2b, b+1:2b) & \\ & & & I_{\frac{m}{P}-b} \end{bmatrix},$$

and

$$\bar{\Pi}_2 = \begin{bmatrix} \Pi_{02}(1:b, 1:b) & & \Pi_{02}(1:b, b+1:2b) & \\ & I_{2\frac{m}{P}-b} & & \\ \Pi_{02}(b+1:2b, 1:b) & & \Pi_{02}(b+1:2b, b+1:2b) & \\ & & & I_{2\frac{m}{P}-b} \end{bmatrix}.$$

Once the pivot rows are in the diagonal positions, the QR factorization with no pivoting is performed on the transpose of the first panel,

$$\bar{A}^T(1:b, :) = QR = Q \begin{bmatrix} R_{11} & R_{12} \end{bmatrix}.$$

This factorization is used to update the trailing matrix, and the elimination of the first panel leads to the following decomposition (we use the same notation as in section 2.2),

$$\bar{A} = \begin{bmatrix} I_b & \\ \bar{A}_{21}\bar{A}_{11}^{-1} & I_{m-b} \end{bmatrix} \begin{bmatrix} \bar{A}_{11} & \bar{A}_{12} \\ & \bar{A}_{22}^s \end{bmatrix},$$

where

$$\bar{A}_{22}^s = \bar{A}_{22} - \bar{A}_{21}\bar{A}_{11}^{-1}\bar{A}_{12}.$$

Figure 3.1 illustrates an example of a panel factorization using CALU_PRRP with 4 processors working on the panel. Every processor is represented by a color. The dashed arrows represent the computations at each level of the binary reduction tree, that is the application of strong RRQR to each block A_{ij}^T. The solid arrows represent communication between processors, that is the merge of 2 $b \times b$ blocks.

As in the LU_PRRP factorization, the CALU_PRRP factorization computes a block LU factorization of the input matrix A. To obtain the full LU factorization, an additional GEPP is performed on the diagonal block $\bar{A}_{11}$, followed by the update of the block row $\bar{A}_{12}$. The CALU_PRRP factorization continues the same procedure on the trailing matrix $\bar{A}_{22}^s$.

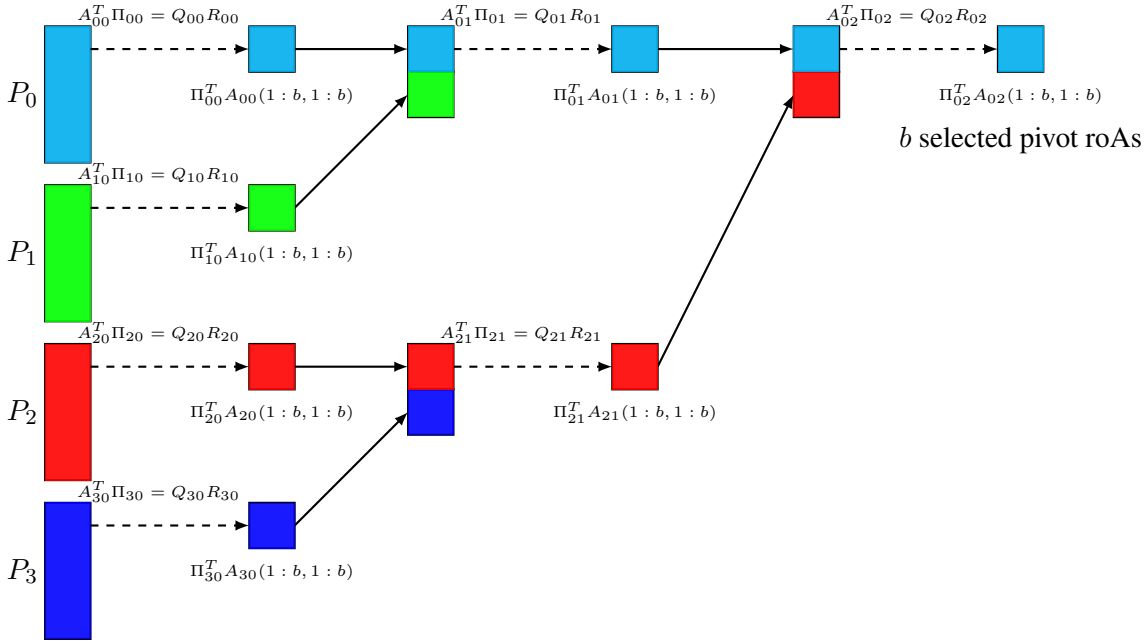
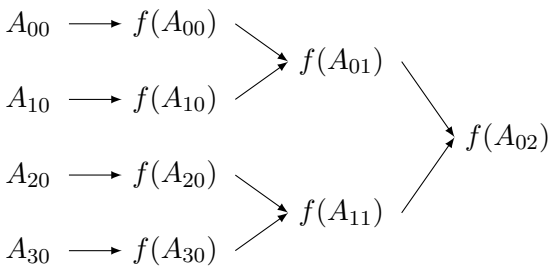


Figure 3.1: Selection of b pivot rows for the panel factorization using the CALU_PRRP factorization, preprocessing step.

Note that the factors L and U obtained by the CALU_PRRP factorization are different from the factors obtained by the LU_PRRP factorization. The two algorithms use different pivot rows, and in particular the factor L of CALU_PRRP is no longer bounded by a given threshold τ as in LU_PRRP. This leads to a different worst case growth factor for CALU_PRRP, that we will discuss in the following section.

The following figure displays the binary tree based tournament pivoting performed on the first panel using an arrow notation (as in [60]). The function $f(A_{ij})$ computes a strong RRQR of the matrix A_{ij}^T to select a set of b candidate rows. At each node of the reduction tree, two sets of b candidate rows are merged together and form a matrix A_{ij}, the function f is applied to A_{ij}, and another set of b candidate pivot rows is selected. While in this section we focused on binary trees, tournament pivoting can use any reduction tree, and this allows the algorithm to adapt on different architectures. Later in the paper we will consider also a flat reduction tree.



3.2.2 Numerical Stability of CALU_PRRP

In this section we discuss the stability of the CALU_PRRP factorization and we identify similarities with the LU_PRRP factorization. We also discuss the growth factor of the CALU_PRRP factorization, and we show that its upper bound depends on the height of the reduction tree. For the same reduction tree, this upper bound is smaller than that obtained with CALU. More importantly, for cases of interest, the upper bound of the growth factor of CALU_PRRP is also smaller than that obtained with GEPP.

To address the numerical stability of CALU_PRRP, we show that in exact arithmetic, performing CALU_PRRP on a matrix A is equivalent to performing LU_PRRP on a larger matrix A_{LU_PRRP}, which is formed by blocks of A (sometimes slightly perturbed) and blocks of zeros. This reasoning is also used in [60] to show the same equivalence between CALU and GEPP. While this similarity is explained in detail in [60], here we focus only on the first step of the CALU_PRRP factorization. We explain the construction of the larger matrix A_{LU_PRRP} to expose the equivalence between the first step of the CALU_PRRP factorization of A and the LU_PRRP factorization of A_{LU_PRRP}.

Consider a nonsingular matrix A of size $m \times n$ and the first step of its CALU_PRRP factorization using a general reduction tree of height H. Tournament pivoting selects b candidate rows at each node of the reduction tree by using the strong RRQR factorization. Each such factorization leads to an L factor which is bounded locally by a given threshold τ. However this bound is not guaranteed globally. When the factorization of the first panel is computed using the b pivot rows selected by tournament pivoting, the L factor will not be bounded by τ. This results in a larger growth factor than the one obtained with the LU_PRRP factorization. Recall that in LU_PRRP, the strong RRQR factorization is performed on the transpose of the whole panel, and so every entry of the obtained lower triangular factor L is bounded by τ.

However, we show now that in exact arithmetic, the growth factor g_W obtained after the first step of the CALU_PRRP factorization is bounded by $(1 + \tau b)^{H+1}$, where

$$g_W = \frac{\max_{i,j,k} |a_{ij}^{(k)}|}{\max_{i,j} |a_{ij}|}.$$

Consider a row j, and let $A^s(j, b + 1 : n)$ be the updated row obtained after the first step of elimination of CALU_PRRP. Suppose that row j of A is a candidate row at level $k - 1$ of the reduction tree, and so it participates in the strong RRQR factorization computed at a node s_k at level k of the reduction tree, but it is not selected as a candidate row by this factorization. We refer to the matrix formed by the candidate rows at node s_k as $\bar{A}_k$. Hence, row j is not used to form the matrix $\bar{A}_k$. Similarly, for every node i on the path from node s_k to the root of the reduction tree of height H, we refer to the matrix formed by the candidate rows selected by strong RRQR as $\bar{A}_i$. Note that in practice it can happen that one of the blocks of the panel is singular, while the entire panel is nonsingular. In this case strong RRQR will select less than b linearly independent rows that will be passed along the reduction tree. However, for simplicity, we assume in the following that the matrices $\bar{A}_i$ are nonsingular. For a more general solution, the reader can consult [60].

Let Π be the permutation returned by the tournament pivoting strategy performed on the first panel, that is the permutation that puts the matrix $\bar{A}_H$ on diagonal. The following equation is satisfied,

$$\begin{pmatrix} \bar{A}_H & \bar{\bar{A}}_H \\ A(j, 1:b) & A(j, b+1:n) \end{pmatrix} = \begin{pmatrix} I_b & \\ A(j, 1:b)\bar{A}_H^{-1} & 1 \end{pmatrix} \cdot \begin{pmatrix} \bar{A}_H & \bar{\bar{A}}_H \\ & A^s(j, b+1:n) \end{pmatrix}, \tag{3.1}$$

where

$$\begin{aligned}\bar{A}_H &= (\Pi A)(1:b, 1:b),\\ \bar{\bar{A}}_H &= (\Pi A)(1:b, b+1:n).\end{aligned}$$

The updated row $A^s(j, b+1:n)$ can be also obtained by performing LU_PRRP on a larger matrix A_{LU_PRRP} of dimension $((H-k+1)b+1) \times ((H-k+1)b+1)$,

$$\begin{aligned}A_{LU\_PRRP} &= \begin{pmatrix} \bar{A}_H & & & & & & \bar{\bar{A}}_H \\ \bar{A}_{H-1} & \bar{A}_{H-1} & & & & & \\ & \bar{A}_{H-2} & \bar{A}_{H-2} & & & & \\ & & \ddots & \ddots & & & \\ & & & \bar{A}_k & \bar{A}_k & & \\ & & & & (-1)^{H-k}A(j,1:b) & A(j,b+1:n) \end{pmatrix} \\ &= \begin{pmatrix} I_b & & & & & & \\ \bar{A}_{H-1}\bar{A}_H^{-1} & I_b & & & & & \\ & \bar{A}_{H-2}\bar{A}_{H-1}^{-1} & I_b & & & & \\ & & \ddots & \ddots & & & \\ & & & \bar{A}_k\bar{A}_{k+1}^{-1} & I_b & & \\ & & & & (-1)^{H-k}A(j,1:b)\bar{A}_k^{-1} & 1 \end{pmatrix} \\ &\cdot \begin{pmatrix} \bar{A}_H & & & & & \bar{\bar{A}}_H \\ & \bar{A}_{H-1} & & & & \bar{\bar{A}}_{H-1} \\ & & \bar{A}_{H-2} & & & \bar{\bar{A}}_{H-2} \\ & & & \ddots & & \vdots \\ & & & & \bar{A}_k & \bar{\bar{A}}_k \\ & & & & & A^s(j,b+1:n) \end{pmatrix}, \end{aligned} \tag{3.2}$$

where

$$\bar{\bar{A}}_{H-i} = \begin{cases} \bar{A}_H & \text{if } i = 0, \\ -\bar{A}_{H-i}\bar{A}_{H-i+1}^{-1}\bar{\bar{A}}_{H-i+1} & \text{if } 0 < i \leq H-k. \end{cases} \tag{3.3}$$

Equation (3.2) can be easily verified, since

$$\begin{aligned}A^s(j, b+1:n) &= A(j, b+1:n) - (-1)^{H-k}A(j,1:b)\bar{A}_k^{-1}(-1)^{H-k}\bar{\bar{A}}_k \\ &= A(j, b+1:n) - A(j,1:b)\bar{A}_k^{-1}\bar{A}_k\bar{A}_{k+1}^{-1}\ldots\bar{A}_{H-2}\bar{A}_{H-1}^{-1}\bar{A}_{H-1}\bar{A}_H^{-1}\bar{\bar{A}}_H \\ &= A(j, b+1:n) - A(j,1:b)\bar{A}_H^{-1}\bar{\bar{A}}_H.\end{aligned}$$

Equations (3.1) and (3.2) show that in exact arithmetic, the Schur complement obtained after each step of performing the CALU_PRRP factorization of a matrix A is equivalent to the Schur complement obtained after performing the LU_PRRP factorization of a larger matrix A_{LU_PRRP}, formed by blocks of A (sometimes slightly perturbed) and blocks of zeros. More generally, this implies that the entire CALU_PRRP factorization of A is equivalent to the LU_PRRP factorization of a larger and very sparse matrix, formed by blocks of A and blocks of zeros (we omit the proofs here, since they are similar to the proofs presented in [60]).

Equation (3.2) is used to derive the upper bound of the growth factor of CALU_PRRP from the upper bound of the growth factor of LU_PRRP. The elimination of each row of the first panel using

CALU_PRRP can be obtained by performing LU_PRRP on a matrix of maximum dimension $m \times b(H+1)$. Hence the upper bound of the growth factor obtained after one step of CALU_PRRP is $(1+\tau b)^{H+1}$. In exact arithmetic, this leads to an upper bound of $(1+\tau b)^{\frac{n}{b}(H+1)-1}$ for a matrix of size $m \times n$.

Table 3.1 summarizes the bounds of the growth factor of CALU_PRRP derived in this section, and also recalls the bounds of LU_PRRP, GEPP, and CALU, the communication avoiding version of GEPP. It considers the growth factor obtained after the elimination of b columns of a matrix of size $m \times (b+1)$, and also the general case of a matrix of size $m \times n$. As discussed in section 2.2.1 already, LU_PRRP is more stable than GEPP in terms of worst case growth factor. From Table 3.1, it can be seen that for a reduction tree of a same height, CALU_PRRP is more stable than CALU. We note here that we limit our analysis to the block factorizations. Hence we consider that LU_PRRP and CALU_PRRP are block factorizations, and the growth factor does not include the additional Gaussian elimination with partial pivoting applied to the diagonal blocks.

In the following we show that CALU_PRRP can have a smaller worst case growth factor than GEPP. Consider a parallel version of CALU_PRRP based on a binary reduction tree of height $H = \log(P)$, where P is the number of processors. The upper bound of the growth factor becomes $(1+\tau b)^{(n/b)(logP+1)-1}$, which is smaller than $2^{n(logP+1)-1}$, the upper bound of the growth factor of CALU. For example, if the threshold is $\tau = 2$, the panel size is $b = 64$, and the number of processors is $P = 128 = 2^7$, then $g_{WCALU\_PRRP} \approx (1.8)^n$. This quantity is much smaller than 2^{7n} the upper bound of CALU, and even smaller than the worst case growth factor of GEPP of 2^{n-1}. In general, the upper bound of CALU_PRRP can be smaller than the one of GEPP, if the different parameters τ, H, and b are chosen such that the condition

$$H \leq \frac{b}{(\log b + \log \tau)} \tag{3.4}$$

is satisfied. For a binary tree of height $H = \log P$, it becomes $\log P \leq b/(\log b + \log \tau)$. This is a condition which can be satisfied in practice, by choosing b and τ appropriately for a given number of processors P. For example, when $P \leq 512$, $b = 64$, and $\tau = 2$, the condition (3.4) is satisfied, and the worst case growth factor of CALU_PRRP is smaller than the one of GEPP.

However, for a sequential version of CALU_PRRP using a flat tree of height $H = n/b$, the condition to be satisfied becomes $n \leq b^2/(\log b + \log \tau)$, which is more restrictive. In practice, the size of b is chosen depending on the size of the memory, and it might be the case that it will not satisfy the condition in equation (3.4).

3.2.3 Experimental results

In this section we present experimental results and show that CALU_PRRP is stable in practice and compare them with those obtained from CALU and GEPP in [60]. We present results for both the binary tree scheme and the flat tree scheme.

As in section 2.2.2, we perform our tests on matrices whose elements follow a normal distribution. In MATLAB notation, the test matrix is $A = \mathsf{randn}(n, n)$, and the right hand side is $b = \mathsf{randn}(n, 1)$. The size of the matrix is chosen such that n is a power of 2, that is $n = 2^k$, and the sample size is 10 if $k < 13$ and 3 if $k \geq 13$. To measure the stability of CALU_PRRP, we discuss several metrics,

Table 3.1: Bounds for the growth factor g_W obtained from factoring a matrix of size $m \times (b+1)$ and a matrix of size $m \times n$ using CALU_PRRP, LU_PRRP, CALU, and GEPP. CALU_PRRP and CALU use a reduction tree of height H. The strong RRQR used in LU_PRRP and CALU_PRRP depends on a threshold τ. For the matrix of size $m \times (b+1)$, the result corresponds to the growth factor obtained after eliminating b columns.

	matrix of size $m \times (b+1)$			
	TSLU(b,H)	LU_PRRP(b,H)	GEPP	LU_PRRP
g_W upper bound	$2^{b(H+1)}$	$(1+\tau b)^{H+1}$	2^b	$1+\tau b$
	matrix of size $m \times n$			
	CALU	CALU_PRRP	GEPP	LU_PRRP
g_W upper bound	$2^{n(H+1)-1}$	$(1+\tau b)^{\frac{n(H+1)}{b}-1}$	2^{n-1}	$(1+\tau b)^{\frac{n}{b}}$

that concern the LU decomposition and the linear solver using it, such as the growth factor, normwise and componentwise backward errors. We also perform tests on several special matrices including sparse matrices, they are described in Appendix A.

Figure 3.2 displays the values of the growth factor g_W of the binary tree based CALU_PRRP, for different block sizes b and different number of processors P. Here the growth factor is determined by using the computed values of A,

$$g_W = \frac{\max_{i,j,k} |\hat{a}_{ij}^{(k)}|}{\max_{i,j} |\hat{a}_{ij}|}.$$

As explained in section 3.2.1, the block size determines the size of the panel, while the number of processors determines the number of block rows in which the panel is partitioned. This corresponds to the number of leaves of the binary tree. We observe that the growth factor of binary tree based CALU_PRRP is in the most of the cases better than GEPP. The curves of the growth factor lie between $\frac{1}{2}n^{1/2}$ and $\frac{3}{4}n^{1/2}$ in our tests on random matrices. These results show that binary tree based CALU_PRRP is stable and the growth factor values obtained for the different layouts are better than those obtained with binary tree based CALU. The Figure 3.2 includes also the growth factor of the LU_PRRP method with a panel of size $b = 64$. We note that that results are better than those of binary tree based CALU_PRRP.

Figure 3.3 displays the values of the growth factor g_W for flat tree based CALU_PRRP with a block size b varying from 8 to 128. The growth factor g_W is decreasing with increasing the panel size b. We note that the curves of the growth factor lie between $\frac{1}{4}n^{1/2}$ and $\frac{3}{4}n^{1/2}$ in our tests on random matrices. We also note that the results obtained with the LU_PRRP method with a panel of size $b = 64$ are better than those of flat tree based CALU_PRRP. The growth factors of both binary tree based and flat tree based CALU_PRRP have similar (sometimes better) behavior than the growth factors of GEPP.

Table 3.2 presents results for the linear solver using binary tree based CALU_PRRP, together with binary tree based CALU and GEPP for comparaison and Table 3.3 presents results for the linear solver using flat tree based CALU_PRRP, together with flat tree based CALU and GEPP for comparaison. We note that for the binary tree based CALU_PRRP, when $m/P = b$, for the algorithm we only use $P1 = m/(b+1)$ processors, since to perform a Strong RRQR on a given block, the number of its rows should be at least equal to the number of its columns $+1$.

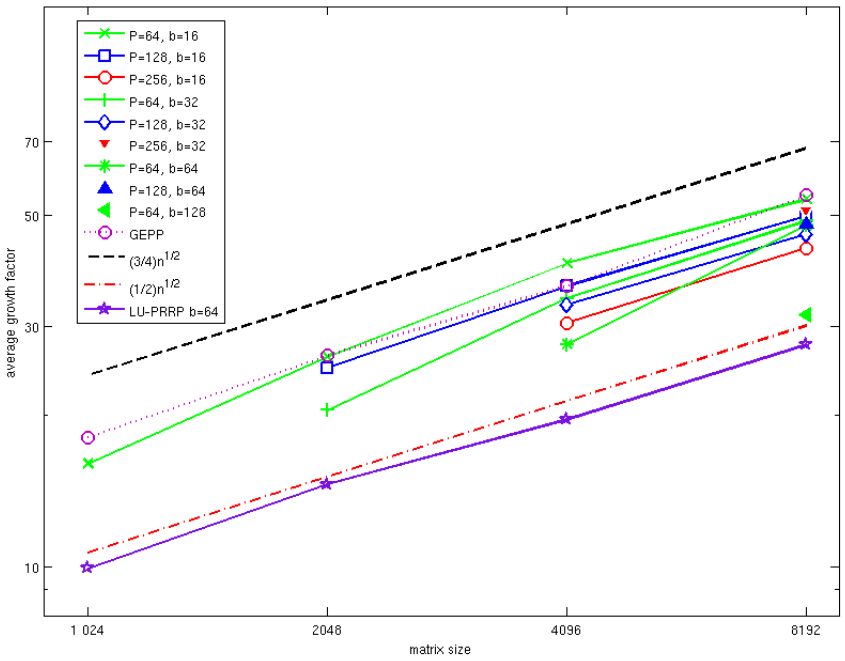


Figure 3.2: Growth factor g_W of binary tree based CALU_PRRP for random matrices.

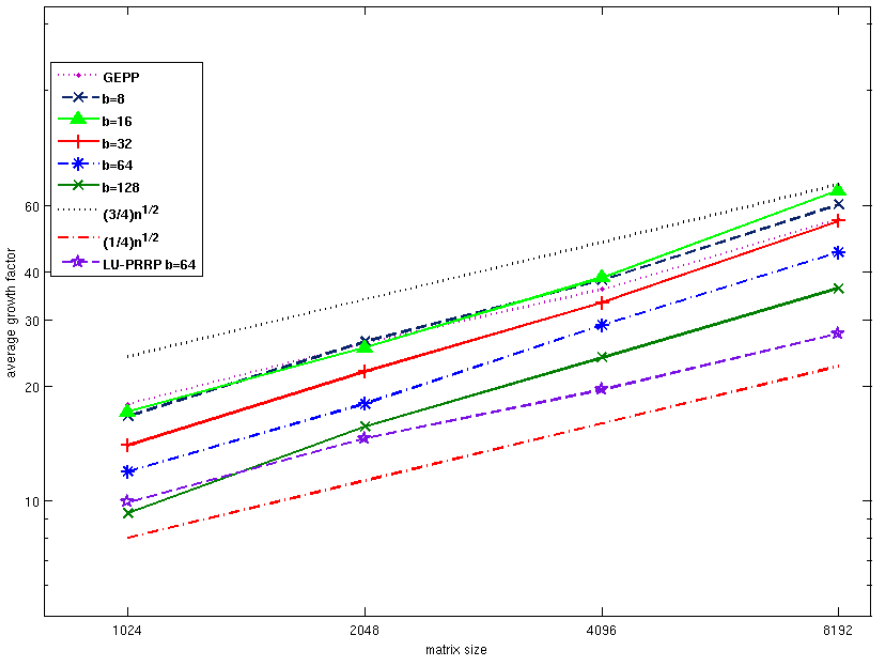


Figure 3.3: Growth factor g_W of flat tree based CALU_PRRP for random matrices.

Tables 3.2 and 3.3 also include results obtained by iterative refinement used to improve the accuracy of the solution. For this, the componentwise backward error in equation (2.11) is used. In the previous tables, w_b denotes the componentwise backward error before iterative refinement and N_{IR} denotes the number of steps of iterative refinement. N_{IR} is not always an integer since it represents an average. For all the matrices tested CALU_PRRP leads to results as accurate as the results obtained with CALU and GEPP.

In Appendix A we present more detailed results. There we include some other metrics, such as the norm of the factors, the norm of the inverse of the factors, their conditioning, the value of their maximum element, and the backward error of the LU factorization. Through the results detailed in this section and in Appendix A we show that binary tree based and flat tree based CALU_PRRP are stable, have the same behavior as GEPP for random matrices, and are more stable than binary tree based and flat tree based CALU in terms of growth factor.

Figure 3.4 summarizes all our stability results for the CALU_PRRP factorization based on both binary tree and flat tree schemes, which is analogous to Figure 2.2. Results for all the matrices in our test set are presented, that is 25 random matrices for binary tree base CALU_PRRP from Table 8, 20 random matrices for flat tree based CALU_PRRP from Table 6, and 37 special matrices from Tables 7 and 9. As can be seen, nearly all ratios are between 0.5 and 2.5 for random matrices. However there are few outliers, for example the relative error ratio has values between 24.2 for the special matrix *hadamard*, and 5.8×10^{-3} for the special matrix *moler*.

We consider now the same set of pathological matrices as in section 2.2.2 on which GEPP fails. For the Wilkinson matrix, both CALU and CALU_PRRP based on flat and binary tree give modest element growth. For the generalized Wilkinson matrix, the Foster matrix, and Wright matrix, CALU fails with both flat tree and binary tree reduction schemes.

Tables 3.4 and 3.5 present the results obtained for the linear solver using the CALU_PRRP factorization based on flat and binary tree schemes for a generalized Wilkinson matrix of size 2048 with a size of the panel varying from 8 to 128 for the flat tree scheme and a number of processors varying from 32 to 128 for the binary tree scheme. The growth factor is of order 1 and the quantity $||PA - LU||/||A||$ is on the order of 10^{-16}. Both flat tree based CALU and binary tree based CALU fails on this pathological matrix. For example for a generalized Wilkinson matrix of size 1024 and a panel of size $b = 128$, the growth factor obtained with flat tree based CALU is of size 10^{234}.

For the Foster matrix, we have seen in section 2.2.2 that LU_PRRP gives modest pivot growth, whereas GEPP fails. Both flat tree based CALU and binary tree based CALU fails on this matrix. However flat tree based CALU_PRRP and binary tree based CALU_PRRP solve easily this pathological matrix.

Tables 3.6 and 3.7 present results for the linear solver using the CALU_PRRP factorization based on the flat tree scheme and the binary tree scheme, respectively. We test a Foster matrix of size 2048 with a panel size varying from 8 to 128 for the flat tree based CALU_PRRP and a number of processors varying from 32 to 128 for the binary tree based CALU_PRRP. We use the same parameters as in section 2.2.2, that is $c = 1$, $h = 1$ and $k = \frac{2}{3}$. According to the obtained results, CALU_PRRP gives a modest growth factor of 1.33 for this practical matrix.

As GEPP, both the flat tree based and the binary tree based CALU fail on the Wright matrix. In fact

Table 3.2: Stability of the linear solver using binary tree based CALU_PRRP, flat tree based CALU, and GEPP.

n	P	b	η	w_b	N_{IR}	HPL3
Binary tree based CALU_PRRP						
8192	256	32	7.5E-15	4.4E-14	2	4.4E-03
		16	6.7E-15	4.1E-14	2	4.6E-03
	128	64	7.6E-15	4.7E-14	2	4.9E-03
		32	7.5E-15	4.9E-14	2	4.9E-03
		16	7.3E-15	5.1E-14	2	5.7E-03
	64	128	7.6E-15	5.0E-14	2	5.2E-03
		64	7.9E-15	5.3E-14	2	5.9E-03
		32	7.8E-15	5.0E-14	2	5.0E-03
		16	6.7E-15	5.0E-14	2	5.7E-03
4096	256	16	3.5E-15	2.2E-14	2	5.1E-03
	128	32	3.8E-15	2.3E-14	2	5.1E-03
		16	3.6E-15	2.2E-14	1.6	5.1E-03
	64	64	4.0E-15	2.3E-14	2	4.9E-03
		32	3.9E-15	2.4E-14	2	5.7E-03
		16	3.8E-15	2.4E-14	1.6	5.2E-03
2048	128	16	1.8E-15	1.0E-14	2	4.8E-03
	64	32	1.8E-15	1.2E-14	2	5.6E-03
		16	1.9E-15	1.2E-14	1.8	5.4E-03
1024	64	16	1.0E-15	6.3E-15	1.3	6.1E-03
Binary tree based CALU						
8192	256	32	6.2E-15	4.1E-14	2	4.5E-03
		16	5.8E-15	3.9E-14	2	4.1E-03
	128	64	6.1E-15	4.2E-14	2	4.6E-03
		32	6.3E-15	4.0E-14	2	4.4E-03
		16	5.8E-15	4.0E-14	2	4.3E-03
	64	128	5.8E-15	3.6E-14	2	3.9E-03
		64	6.2E-15	4.3E-14	2	4.4E-03
		32	6.3E-15	4.1E-14	2	4.5E-03
		16	6.0E-15	4.1E-14	2	4.2E-03
4096	256	16	3.1E-15	2.1E-14	1.7	4.4E-03
	128	32	3.2E-15	2.3E-14	2	5.1E-03
		16	3.1E-15	1.8E-14	2	4.0E-03
	64	64	3.2E-15	2.1E-14	1.7	4.6E-03
		32	3.2E-15	2.2E-14	1.3	4.7E-03
		16	3.1E-15	2.0E-14	2	4.3E-03
2048	128	16	1.7E-15	1.1E-14	1.8	5.1E-03
	64	32	1.7E-15	1.0E-14	1.6	4.6E-03
		16	1.6E-15	1.1E-14	1.8	4.9E-03
1024	64	16	8.7E-16	5.2E-15	1.6	4.7E-03
GEPP						
8192	-		3.9E-15	2.6E-14	1.6	2.8E-03
4096	-		2.1E-15	1.4E-14	1.6	2.9E-03
2048	-		1.1E-15	7.4E-15	2	3.4E-03
1024	-		6.6E-16	4.0E-15	2	3.7E-03

Table 3.3: Stability of the linear solver using flat tree based CALU_PRRP, flat tree based CALU, and GEPP.

n	P	b	η	w_b	N_{IR}	HPL3
Flat tree based CALU_PRRP						
8096	-	8	6.2E-15	3.8E-14	1	4.0E-03
	-	16	6.6E-15	4.3E-14	1	4.3E-03
	-	32	7.2E-15	4.8E-14	1	5.3E-03
	-	64	7.3E-15	5.1E-14	1	5.4E-03
4096	-	8	3.2E-15	2.0E-14	1	4.7E-03
	-	16	3.6E-15	2.3E-14	1	5.1E-03
	-	32	3.8E-15	2.4E-14	1	5.5E-03
	-	64	3.7E-15	2.5E-14	1	5.6E-03
2048	-	8	1.4E-15	1.1E-14	1	5.1E-03
	-	16	1.9E-15	1.1E-14	1	5.3E-03
	-	32	2.1E-15	1.3E-14	1	5.8E-03
	-	64	1.9E-15	1.25E-14	1	5.9E-03
1024	-	8	9.4E-16	5.5E-15	1	5.3E-03
	-	16	1.0E-15	6.0E-15	1	5.4E-03
	-	32	1.0E-15	5.6E-15	1	5.4E-03
	-	64	1.0E-15	6.8E-15	1	6.7E-03
Flat tree based CALU						
8096	-	8	4.5E-15	3.1E-14	1.7	3.4E-03
	-	16	5.6E-15	3.7E-14	2	3.3E-03
	-	32	6.7E-15	4.4E-14	2	4.7E-03
	-	64	6.5E-15	4.2E-14	2	4.6E-03
4096	-	8	2.6E-15	1.7E-14	1.3	4.0E-03
	-	16	3.0E-15	1.9E-14	1.7	3.9E-03
	-	32	3.8E-15	2.4E-14	2	5.1E-03
	-	64	3.4E-15	2.0E-14	2	4.1E-03
2048	-	8	1.5E-15	8.7E-15	1.6	4.2E-03
	-	16	1.6E-15	1.0E-14	2	4.5E-03
	-	32	1.8E-15	1.1E-14	1.8	5.1E-03
	-	64	1.7E-15	1.0E-14	1.2	4.5E-03
1024	-	8	7.8E-16	4.9E-15	1.6	4.9E-03
	-	16	9.2E-16	5.2E-15	1.2	4.8E-03
	-	32	9.6E-16	5.8E-15	1.1	5.6E-03
	-	64	8.7E-16	4.9E-15	1.3	4.5E-03
GEPP						
8192	-		3.9E-15	2.6E-14	1.6	2.8E-03
4096	-		2.1E-15	1.4E-14	1.6	2.9E-03
2048	-		1.1E-15	7.4E-15	2	3.4E-03
1024	-		6.6E-16	4.0E-15	2	3.7E-03

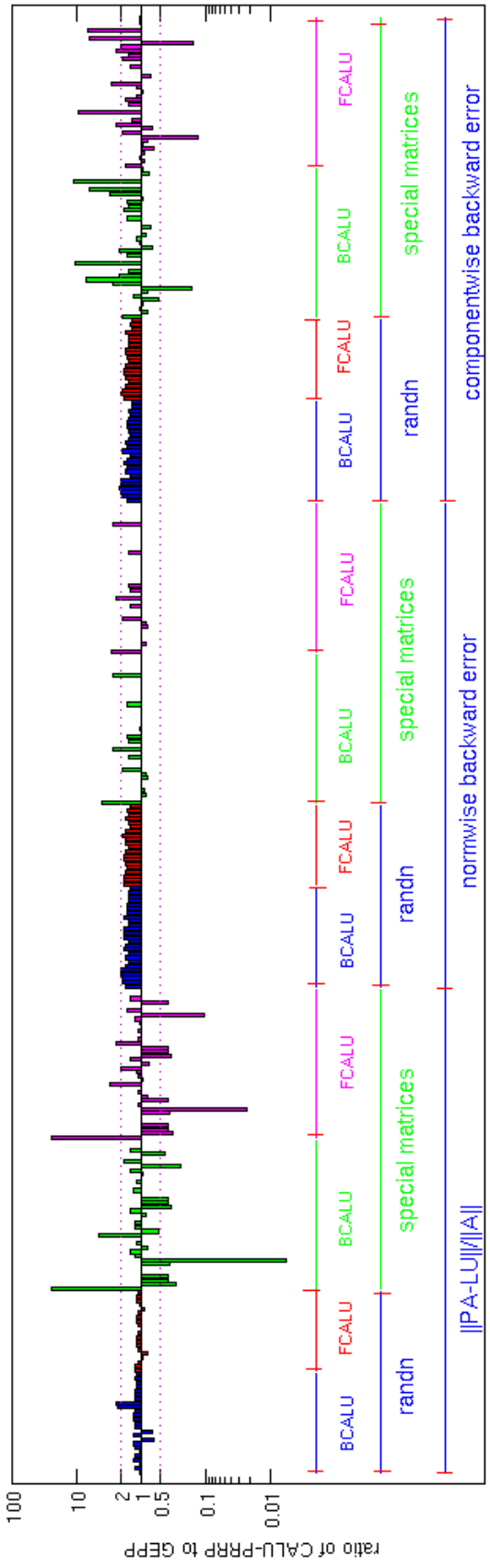


Figure 3.4: A summary of all our experimental data, showing the ratio of max(CALU_PRRP's backward error, machine epsilon) to max(GEPP's backward error, machine epsilon) for all the test matrices in our set. Each test matrix is factored using CALU_PRRP based on a binary reduction tree (labeled BCALU for BCALU_PRRP) and based on a flat reduction tree (labeled FCALU for FCALU_PRRP).

n	b	g_W	$\|\|U\|\|_1$	$\|\|U^{-1}\|\|_1$	$\|\|L\|\|_1$	$\|\|L^{-1}\|\|_1$	$\frac{\|\|PA-LU\|\|_F}{\|\|A\|\|_F}$
2048	128	2.01	1.01e+03	1.40e+02	1.31e+03	9.76e+02	9.56e-16
	64	2.02	1.18e+03	1.64e+02	1.27e+03	1.16e+03	1.01e-15
	32	2.04	8.34e+02	1.60e+02	1.30e+03	7.44e+02	7.91e-16
	16	2.15	9.10e+02	1.45e+02	1.31e+03	8.22e+02	8.07e-16
	8	2.15	8.71e+02	1.57e+02	1.371e+03	5.46e+02	6.09e-16

Table 3.4: Stability of the flat tree based CALU_PRRP factorization of a generalized Wilkinson matrix on which GEPP fails.

n	P	b	g_W	$\|\|U\|\|_1$	$\|\|U^{-1}\|\|_1$	$\|\|L\|\|_1$	$\|\|L^{-1}\|\|_1$	$\frac{\|\|PA-LU\|\|_F}{\|\|A\|\|_F}$
2048	128	8	2.10e+00	1.33e+03	1.29e+02	1.34e+03	1.33e+03	1.08e-15
	64	16	2.04e+00	6.85e+02	1.30e+02	1.30e+03	6.85e+02	7.85e-16
		8	8.78e+01	1.21e+03	1.60e+02	1.33e+03	1.01e+03	9.54e-16
	32	32	2.08e+00	9.47e+02	1.58e+02	1.41e+03	9.36e+02	5.95e-16
		16	2.08e+00	1.24e+03	1.32e+02	1.35e+03	1.24e+03	1.01e-15
		8	1.45e+02	1.03e+03	1.54e+02	1.37e+03	6.61e+02	6.91e-16

Table 3.5: Stability of the binary tree based CALU_PRRP factorization of a generalized Wilkinson matrix on which GEPP fails.

n	b	g_W	$\|\|U\|\|_1$	$\|\|U^{-1}\|\|_1$	$\|\|L\|\|_1$	$\|\|L^{-1}\|\|_1$	$\frac{\|\|PA-LU\|\|_F}{\|\|A\|\|_F}$
2048	128	1.33	1.71e+02	1.87e+00	1.92e+03	1.29e+02	6.51e-17
	64	1.33	8.60e+01	1.87e+00	1.98e+03	6.50e+01	4.87e-17
	32	1.33	4.33e+01	1.87e+00	2.01e+03	3.30e+01	2.91e-17
	16	1.33	2.20e+01	1.87e+00	2.03e+03	1.70e+01	4.80e-17
	8	1.33	1.13e+01	1.87e+00	2.04e+03	9.00e+00	6.07e-17

Table 3.6: Stability of the flat tree based CALU_PRRP factorization of a practical matrix (Foster) on which GEPP fails.

n	P	b	g_W	$\|\|U\|\|_1$	$\|\|U^{-1}\|\|_1$	$\|\|L\|\|_1$	$\|\|L^{-1}\|\|_1$	$\frac{\|\|PA-LU\|\|_F}{\|\|A\|\|_F}$
2048	128	8	1.33	1.13e+01	1.87e+00	2.04e+03	9.00e+00	6.07e-17
	64	16	1.33	2.20e+01	1.87e+00	2.03e+03	1.70e+01	4.80e-17
		8	1.33	1.13e+01	1.87e+00	2.04e+03	9.00e+00	6.07e-17
	32	32	1.33	4.33e+01	1.87e+00	2.01e+03	3.300e+01	2.91e-17
		16	1.33	2.20e+01	1.87e+00	2.03e+03	1.70e+01	4.80e-17
		8	1.33	1.13e+01	1.87e+00	2.04e+03	9.00e+00	6.07e-17

Table 3.7: Stability of the binary tree based CALU_PRRP factorization of a practical matrix (Foster) on which GEPP fails.

for a matrix of size 2048, a parameter $h = 0.3$, with a panel of size $b = 128$, the flat tree based CALU gives a growth factor of 10^{98}. With a number of processors $P = 64$ and a panel of size $b = 16$, the binary tree based CALU also gives a growth factor of 10^{98}. Tables 3.8 and 3.9 present results for the linear solver using the CALU_PRRP factorization for a Wright matrix of size 2048. For the flat tree based CALU_PRRP, the size of the panel is varying from 8 to 128. For the binary tree based CALU_PRRP, the number of processors is varying from 32 to 128 and the size of the panel from 16 to 64 such that the number of rows in the leaf nodes is equal or bigger than two times the size of the panel. The obtained results, show that CALU_PRRP gives a modest growth factor equal to 1 for this practical matrix.

n	b	g_W	$\|\|U\|\|_1$	$\|\|U^{-1}\|\|_1$	$\|\|L\|\|_1$	$\|\|L^{-1}\|\|_1$	$\frac{\|\|PA-LU\|\|_F}{\|\|A\|\|_F}$
2048	128	1	3.25e+00	8.00e+00	2.00e+00	2.00e+00	4.08e-17
	64	1	3.25e+00	8.00e+00	2.00e+00	2.00e+00	4.08e-17
	32	1	3.25e+00	8.00e+00	2.05e+00	2.02e+00	6.65e-17
	16	1	3.25e+00	8.00e+00	2.32e+00	2.18e+00	1.04e-16
	8	1	3.40e+00	8.00e+00	2.62e+00	2.47e+00	1.26e-16

Table 3.8: Stability of the flat tree based CALU_PRRP factorization of a practical matrix (Wright) on which GEPP fails.

n	P	b	g_W	$\|\|U\|\|_1$	$\|\|U^{-1}\|\|_1$	$\|\|L\|\|_1$	$\|\|L^{-1}\|\|_1$	$\frac{\|\|PA-LU\|\|_F}{\|\|A\|\|_F}$
2048	128	8	1	3.40e+00	8.00e+00	2.62e+00	2.47e+00	1.26e-16
	64	16	1	3.25e+00	8.00e+00	2.32e+00	2.18e+00	1.04e-16
		8	1	3.40e+00	8.00e+00	2.62e+00	2.47e+00	1.26e-16
	32	32	1	3.25e+00	8.00e+00	2.05e+00	2.02e+00	6.65e-17
		16	1	3.25e+00	8.00e+00	2.32e+00	2.18e+00	1.04e-16
		8	1	3.40e+00	8.00e+00	2.62e+00	2.47e+00	1.26e-16

Table 3.9: Stability of the binary tree based CALU_PRRP factorization of a practical matrix (Wright) on which GEPP fails.

All the previous tests show that the CALU_PRRP factorization is very stable for random and more special matrices, and it also gives modest growth factor for the pathological matrices on which CALU fails, this is for both binary tree and flat tree based CALU_PRRP.

3.3 Cost analysis of CALU_PRRP

In this section we focus on the parallel CALU_PRRP algorithm based on a binary reduction tree, and we show that it minimizes the communication between different processors of a parallel computer, modulo polylogarithmic factors. For this, we use known lower bounds on the communication performed during the LU factorization of a dense matrix of size $n \times n$, which are

$$\text{\# words\_moved} = \Omega\left(\frac{n^3}{\sqrt{M}}\right), \tag{3.5}$$

$$\text{\# messages} = \Omega\left(\frac{n^3}{M^{\frac{3}{2}}}\right), \tag{3.6}$$

where *# words_moved* refers to the volume of communication, *# messages* refers to the number of messages exchanged, and M refers to the size of the memory (the fast memory in the sequential case, or the memory per processor in the parallel case). These lower bounds were first introduced for dense matrix multiplication [69], [72], generalized later to LU factorization [38], and then to almost all direct linear algebra [15].

Note that these lower bounds apply to algorithms based on orthogonal transformations under certain conditions [15]. However, this is not relevant to our case, since CALU_PRRP uses orthogonal transformations that need communication only to select pivot rows, while the update of the trailing matrix is still performed as in the classic LU factorization algorithm. We note that at the end of the preprocessing step, each processor working on the panel has the final set of b pivot rows. Thus all these processors perform in parallel a QR factorization without pivoting on the transpose of the final $b \times b$ block. After this phase each processor below the diagonal has the R_{11} factor, computes his chunk of R_{12}, and finally computes his block of L_{21} factor as detailed previously. Therefore the QR factorization applied to the transpose of the panel does not imply any communication. Hence the lower bounds from equations (3.5) and, (3.6) are valid for CALU_PRRP.

More details about the parallel block factorization of a panel using CALU_PRRP are presented in Algorithm 11. We consider a panel is of size $m \times b$. It is partitioned over a set of P processors using a 1-D block row layout. Based on strong rank revealing QR factorization, the tournament pivoting identifies b pivot rows using a reduction tree of height H. Assuming that the tournament pivoting is performed in an all-reduce manner, we ensure that at the end of the preprocessing step all the processors have the final result. Hence all the processors participate at all the levels of the reduction. We note that Algorithm 11 does not include the Gaussian elimination step that is applied to the $b \times b$ diagonal block to complete the LU factorization.

We estimate now the cost of computing in parallel the CALU_PRRP factorization of a matrix A of size $m \times n$. The matrix is distributed on a grid of $P = P_r \times P_c$ processors using a two-dimensional (2D) block cyclic layout. We use the following performance model. Let γ be the cost of performing a floating point operation, and let $\alpha + \beta w$ be the cost of sending a message of size w words, where α is the latency cost and β is the inverse of the bandwidth. Then, the total running time of an algorithm is estimated to be

$$\alpha \cdot (\text{\# messages}) + \beta \cdot (\text{\# words\_moved}) + \gamma \cdot (\text{\# flops}),$$

where #messages, #words_moved, and #flops are counted along the critical path of the algorithm.

Table 3.10 displays the performance estimation of parallel CALU_PRRP (a detailed estimation of the counts is presented in Appendix B .2). It also recalls the performance of two existing algorithms, the PDGETRF routine from ScaLAPACK which implements GEPP, and the CALU factorization. All three algorithms have the same volume of communication, since it is known that PDGETRF already minimizes the volume of communication. However, the number of messages of both CALU_PRRP and CALU is smaller by a factor of the order of b than the number of messages of PDGETRF. This improvement is achieved thanks to tournament pivoting. In fact, partial pivoting, as used in the routine PDGETRF, leads to an $O(n \log P)$ number of messages, and because of this, GEPP cannot minimize the number of messages.

Algorithm 11: Parallel TSLU_PRRP: block phase

Require S is the set of P processors, $i \in S$ is my processor's index.
Require All-reduction tree with height H.
Require The $m \times b$ input matrix $A(:, 1:b)$ is distributed using a 1-D block row layout; $A_{i,0}$ is the block of rows belonging to my processor i.
Compute $A_{i,0}^T \Pi_{i,0} = Q_{i,0} R_{i,0}$ using Strong Rank Revealing QR.
for *k from 1 to H* **do**
  if *I have any neighbors in the all-reduction tree at this level* **then**
    Let q be the number of neighbors.
    Send $(\Pi_{i,k-1}^T A_{i,k-1})(1:b, 1:b)$ to each neighbor j
    Receive $(\Pi_{j,k-1}^T A_{j,k-1})(1:b, 1:b)$ from each neighbor j
    Form the matrix $A_{i,k}$ of size $qb \times b$ by stacking the matrices $(\Pi_{j,k-1}^T A_{j,k-1})(1:b, 1:b)$ from all neighbors.
    Compute $A_{i,k}^T \Pi_{i,k} = Q_{i,k} R_{i,k}$ using Strong Rank Revealing QR.
  else
    $A_{i,k}^T := A_{i,k-1}^T \Pi_{i,k-1}$
    $\Pi_{i,k} := I_{b \times b}$
Compute the final permutation $\bar{\Pi} = \bar{\Pi}_H \ldots \bar{\Pi}_1 \bar{\Pi}_0$, where $\bar{\Pi}_i$ represents the permutation matrix corresponding to each level in the reduction tree, formed by the permutation matrices of the nodes at this level extended by appropriate identity matrices to the dimension $m \times m$.
Compute the QR factorization with no pivoting of $(A^T \bar{\Pi})(1:b, :) = QR = Q\,[R_{11} R_{12}]$
Compute the L_{21} factor based on the previous QR factorization $L_{21} = R_{12}^T (R_{11}^{-1})^T$
Ensure $R_{i,H}$ is the R_{11} factor obtained at the end of the preprocessing step, for all processors $i \in S$.

Compared to CALU, CALU_PRRP sends a small factor of less messages (depending on P_r and P_c) and performs $\frac{1}{P_r}\left(2mn - n^2\right) b + \frac{nb^2}{3}(5 \log_2 P_r + 1)$ more flops (which represents a lower order term). This is because CALU_PRRP uses the strong RRQR factorization at every node of the reduction tree of every panel factorization, while CALU uses GEPP.

We show now that CALU_PRRP is optimal in terms of communication. We choose optimal values of the parameters P_r, P_c, and b, as used in CAQR [38] and CALU [60], that is,

$$P_r = \sqrt{\frac{mP}{n}} \ , \ P_c = \sqrt{\frac{nP}{m}} \ \text{ and } b = \frac{1}{4} \log^{-2}\left(\sqrt{\frac{mP}{n}}\right) \cdot \sqrt{\frac{mn}{P}} = \log^{-2}\left(\frac{mP}{n}\right) \cdot \sqrt{\frac{mn}{P}}.$$

For a square matrix of size $n \times n$, the optimal parameters are,

$$P_r = \sqrt{P} \ , \ P_c = \sqrt{P} \ \text{ and } b = \frac{1}{4} \log^{-2}\left(\sqrt{P}\right) \cdot \frac{n}{\sqrt{P}} = \log^{-2}(P) \cdot \frac{n}{\sqrt{P}}.$$

Table 3.11 presents the performance estimation of parallel CALU_PRRP and parallel CALU when using the optimal layout. It also recalls the lower bounds on communication from equations (3.5) and (3.6) when the size of the memory per processor is on the order of (n^2/P). Both CALU_PRRP and CALU attain the lower bounds on the number of words and on the number of messages, modulo poly-logarithmic factors. Note that the optimal layout allows to reduce communication, while keeping the

	Parallel CALU_PRRP
# messages	$\frac{3n}{b}\log_2 P_r + \frac{2n}{b}\log_2 P_c$
# words	$\left(nb + \frac{3}{2}\frac{n^2}{P_c}\right)\log_2 P_r + \frac{1}{P_r}\left(mn - \frac{n^2}{2}\right)\log_2 P_c$
# flops	$\frac{1}{P}\left(mn^2 - \frac{n^3}{3}\right) + \frac{2}{P_r}\left(2mn - n^2\right)b + \frac{n^2 b}{2P_c} + \frac{10nb^2}{3}\log_2 P_r$
	Parallel CALU
# messages	$\frac{3n}{b}\log_2 P_r + \frac{3n}{b}\log_2 P_c$
# words	$\left(nb + \frac{3n^2}{2P_c}\right)\log_2 P_r + \frac{1}{P_r}\left(mn - \frac{n^2}{2}\right)\log_2 P_c$
# flops	$\frac{1}{P}\left(mn^2 - \frac{n^3}{3}\right) + \frac{1}{P_r}\left(2mn - n^2\right)b + \frac{n^2 b}{2P_c} + \frac{nb^2}{3}(5\log_2 P_r - 1)$
	PDGETRF
# messages	$2n\left(1 + \frac{2}{b}\right)\log_2 P_r + \frac{3n}{b}\log_2 P_c$
# words	$\left(\frac{nb}{2} + \frac{3n^2}{2P_c}\right)\log_2 P_r + \log_2 P_c \frac{1}{P_r}\left(mn - \frac{n^2}{2}\right)$
# flops	$\frac{1}{P}\left(mn^2 - \frac{n^3}{3}\right) + \frac{1}{P_r}\left(mn - \frac{n^2}{2}\right)b + \frac{n^2 b}{2P_c}$

Table 3.10: Performance estimation of parallel (binary tree based) CALU_PRRP, parallel CALU, and PDGETRF routine when factoring an $m \times n$ matrix, $m \geq n$. The input matrix is distributed using a 2D block cyclic layout on a $P_r \times P_c$ grid of processors. Some lower order terms are omitted.

number of extra floating point operations performed due to tournament pivoting as a lower order term. While in this section we focused on minimizing communication between the processors of a parallel computer, it is straightforward to show that the usage of a flat tree during tournament pivoting allows CALU_PRRP to minimize communication between different levels of the memory hierarchy in the sequential case.

	Parallel CALU_PRRP with optimal layout	Lower bound
# messages	$\frac{5}{2}\sqrt{P}\log^3 P$	$\Omega(\sqrt{P})$
# words	$\frac{n^2}{\sqrt{P}}\left(\frac{1}{2}\log^{-1} P + \log P\right)$	$\Omega(\frac{n^2}{\sqrt{P}})$
# flops	$\frac{1}{P}\frac{2n^3}{3} + \frac{5n^3}{2P\log^2 P} + \frac{5n^3}{3P\log^3 P}$	$\frac{1}{P}\frac{2n^3}{3}$
	Parallel CALU with optimal layout	Lower bound
# messages	$3\sqrt{P}\log^3 P$	$\Omega(\sqrt{P})$
# words	$\frac{n^2}{\sqrt{P}}\left(\frac{1}{2}\log^{-1} P + \log P\right)$	$\Omega(\frac{n^2}{\sqrt{P}})$
# flops	$\frac{1}{P}\frac{2n^3}{3} + \frac{3n^3}{2P\log^2 P} + \frac{5n^3}{6P\log^3 P}$	$\frac{1}{P}\frac{2n^3}{3}$

Table 3.11: Performance estimation of parallel (binary tree based) CALU_PRRP and CALU with an optimal layout. The matrix factored is of size $n \times n$. Some lower-order terms are omitted.

3.4 Less stable factorizations that can also minimize communication

In this section, we present briefly two alternative algorithms that are based on panel strong RRQR pivoting and that are conceived such that they can minimize communication. But we will see that they can be unstable in practice. These algorithms are also based on block algorithms, that factor the input

matrix by traversing panels of size b. The main difference between them and CALU_PRRP is the panel factorization, which is performed only once in the alternative algorithms.

We present first a parallel alternative algorithm, which we refer to as block parallel LU_PRRP. At each step of the block factorization, the panel is partitioned into P block-rows $[A_0; A_1; \ldots; A_{P-1}]$. The blocks below the diagonal $b \times b$ block of the current panel are eliminated by performing a binary tree of strong RRQR factorizations. At the leaves of the tree, the elements below the diagonal block of each block A_i are eliminated using strong RRQR. The elimination of each such block row is followed by the update of the corresponding block row of the trailing matrix.

The algorithm continues by performing the strong RRQR factorization of pairs of $b \times b$ blocks stacked atop one another, until all the blocks below the diagonal block are eliminated and the corresponding trailing matrices are updated. The algebra of the block parallel LU_PRRP algorithm is detailed in Appendix C, while in Figure 3.5 we illustrate one step of the factorization by using arrow notation, where the function $g(A_{ij})$ computes a strong RRQR on the matrix A_{ij}^T and updates the trailing matrix in the same step.

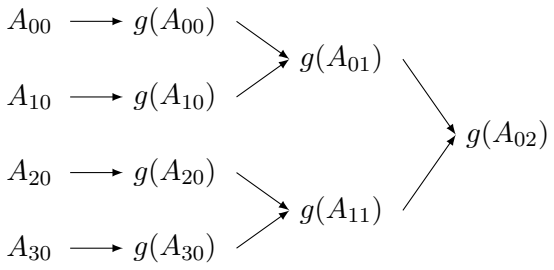


Figure 3.5: Block parallel LU_PRRP

A sequential version of the algorithm is based on the usage of a flat tree, and we refer to this algorithm as block pairwise LU_PRRP. Using the arrow notation, the Figure 3.6 illustrates the elimination of one panel.

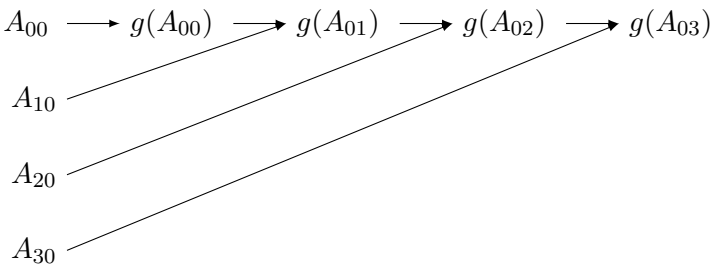


Figure 3.6: Block pairwise LU_PRRP

The block parallel LU_PRRP and the block pairwise LU_PRRP algorithms have similarities with the block parallel pivoting and the block pairwise pivoting algorithms. These two latter algorithms

were shown to be potentially unstable in [60]. There is a main difference between all these alternative algorithms and algorithms that compute a classic LU factorization as GEPP, LU_PRRP, and their communication avoiding variants. The alternative algorithms compute a factorization in the form of a product of lower triangular factors and an upper triangular factor. And the elimination of each column leads to a rank update of the trailing matrix larger than one. It is thought in [92] that the rank-1 update property of algorithms that compute an LU factorization inhibits potential element growth during the factorization, while a large rank update might lead to an unstable factorization.

Note however that at each step of the factorization, block parallel and block pairwise LU_PRRP use at each level of the reduction tree original rows of the active matrix. Block parallel pivoting and block pairwise pivoting algorithms use U factors previously computed to achieve the factorization, and this could potentially lead to a faster propagation of ill-conditioning.

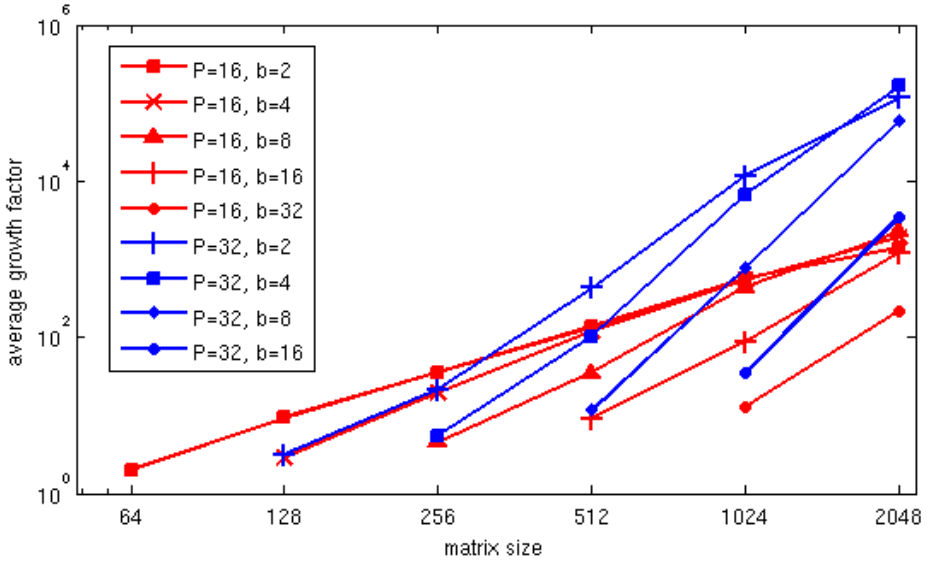


Figure 3.7: Growth factor of block parallel LU_PRRP for varying block size b and number of processors P.

The upper bound of the growth factor of both block parallel and block pairwise LU_PRRP is $(1 + \tau b)^{(n/b)-1}$, since for every panel factorization, a row is updated only once. Hence they have the same bounds as the LU_PRRP factorization, and smaller than that of the CALU_PRRP factorization. Despite this, they are less stable than the CALU_PRRP factorization. Figures 3.7 and 3.8 display the growth factor of block parallel LU_PRRP and block pairwise LU_PRRP for matrices following a normal distribution. In figure 3.7, the number of processors P working on the panel is partitioned is varying from 16 to 32, and the block size b is varying from 2 to 16. The matrix size varies from 64 to 2048, but we have observed the same behavior for matrices of size up to 8192. When the number of processors P is equal to 1, the block parallel LU_PRRP corresponds to the LU_PRRP factorization. The results show that there are values of P and b for which this method can be very unstable. For the sizes of matrices tested, when b is chosen such that the blocks at the leaves of the reduction tree have more than $2b$ rows, the number of processors P has an important impact, the growth factor increases with increasing P, and the method is unstable.

In Figure 3.8, the matrix size varies from 1024 to 8192. For a given matrix size, the growth factor increases with decreasing the size of the panel b, as one could expect. We note that the growth factor of block pairwise LU_PRRP is larger than that obtained with the CALU_PRRP factorization based on a flat tree scheme presented in Table 6. But it stays smaller than the size of the matrix n for different panel sizes. Hence this method is more stable than block parallel LU_PRRP. Further investigations are required to conclude on the stability of these methods.

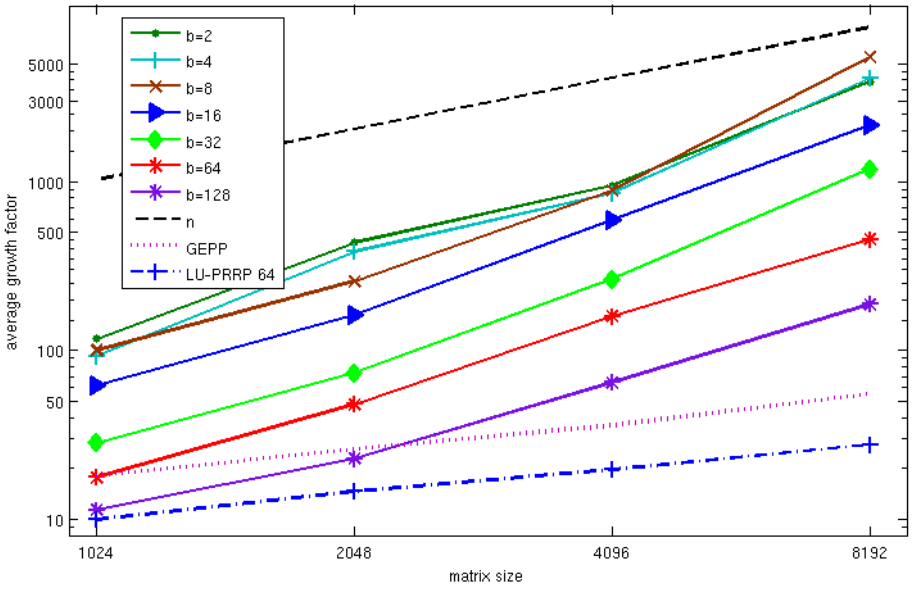


Figure 3.8: Growth factor of block pairwise LU_PRRP for varying matrix size and varying block size b.

3.5 Conclusion

In this chapter we have introduced CALU_PRRP, a communication avoiding LU algorithm. The main motivation behind the design of such an algorithm is the improvement of the numerical stability of CALU, the communication avoiding version of GEPP, especially in terms of worst case growth factor. The upper bound of the growth factor of CALU is worst than that of Gaussian elimination with partial pivoting.

The CALU_PRRP is the communication avoiding version of LU_PRRP, the algorithm introduced in chapter 2. This algorithm is performed in two main phases: the first phase produces a block LU factorization, which is completed by Gaussian eliminations with partial pivoting on the diagonal blocks to compute block rows of U and diagonal blocks of L. Thus the factorization is organized as a product of L factors and an U factor. In this chapter we focus on the first phase, which is also performed in two steps, the preprocessing step that aims at finding a "good" set of pivot rows for the panel factorization based on strong rank revealing pivoting, then the update of the trailing matrix.

To discuss the numerical stability of the CALU_PRRP algorithm, we have first studied the similarities between CALU_PRRP and LU_PRRP. Thus in exact arithmetic, performing CALU_PRRP on a given matrix A is equivalent to performing LU_PRRP on a larger and more sparse matrix formed by blocks of A. This proof was done in a similar way to that in [60]. From the previous equivalence we derive an upper bound of the growth factor of CALU_PRRP. We note that CALU_PRRP is more stable in terms of worst case growth factor than CALU. More importantly, there are cases of interest for which the upper bound of the growth factor of CALU_PRRP is smaller than that of GEPP. Extensive experiments show that CALU_PRRP is very stable in practice and leads to results of the same order of magnitude as GEPP, sometimes even better.

We note that recently Toledo et al have stated in [100] that TSLU can produce large growth factor and becomes unstable when the block of columns is singular and large (e.g., RIS and Fiedler). We note that the authors used communication avoiding LU as a building block in their block Aasen's algorithm. We tested our method on these matrices and we got modest growth comparing to that obtained with CALU. Nevertheless we think that further investigations should be done in this direction. Hence while LU_PRRP easily overcomes the rank deficiency since it is based on strong rank revealing QR, the same conclusion is not straightforward in the case of CALU_PRRP because of the use of tournament pivoting.

Our future work focuses on two main directions. The first direction investigates the design of a communication avoiding algorithm that has smaller bounds on the growth factor than that of GEPP in general. We note that this implies a certain bound on the additional swaps performed during the strong rank revealing pivoting. The second direction consists of estimating the performance of CALU_PRRP on parallel machines based on multicore processors, and comparing it with the performance of CALU. We mention here that the implementation of CALU_PRRP is an ongoing work but it is still unachieved. Thus we were unable to give performance results of our algorithm in this work. We expect that CALU_PRRP would be better than the corresponding routine in ScaLAPACK, but we do not think that it will outperform CALU since it performs more floating-point operations because of the QR factorizations.

Chapter 4

Avoiding communication through a multilevel LU factorization

4.1 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 multiple levels of parallelism even within a single compute node. In this chapter we present a hierarchical algorithm for the LU factorization of a dense matrix. We aim at ensuring a good performance. For that reason we need to adjust the classical algorithms to be more adapted to these modern architectures that expose parallelism at different levels in the hierarchy. The performance of our algorithm relies on its ability to minimize the amount of communication and synchronization at each level of the hierarchy of parallelism. In essence we adapt the algorithm to the machine architecture. We believe that such an approach is appropriate for petascale and exascale computers that are hierarchical in structure.

A good example of machine with two levels of parallelism is a computer formed by nodes of multicore processors. With respect to these two levels of parallelism, this kind of machine exhibits two levels of communication: inter-node communication, that is communication between two or more nodes, and intra-node communication, that is communication performed inside one node. To take advantage of such an architecture, a given algorithm should be tailored considering the specific features of the architecture at the design level. One way to adapt an existing algorithm to such an architecture is the use of hybrid programming model. For example,in algorithms based on MPI programming model, a possible adaptation consists of identifying sequential routines running on a node, and replacing them by parallel routines based on multithreaded programming model. Such a hybrid approach, which combines MPI and threads, has been utilized in several applications. However, it can have several drawbacks such as increasing the amount of communication and synchronization between MPI processes or between threads, load imbalance inside each node, and the difficulty to manually tune the different parameters of the algorithm. Therefore a simple adaptation of an existing algorithm can lead to an important decrease of the overall performance on these modern architectures.

Motivated by the increasing gap between the cost of communication and the cost of computation [57], a new class of algorithms has been introduced in the recent years for dense linear algebra, referred to as communication avoiding algorithms. These algorithms were first presented in the context of dense LU factorization (CALU) [60] and QR factorization (CAQR) [37]. The comminication

avoiding algorithms aim at minimizing the cost of communication between the different processors of a compute system in the parallel case (the memory hierarchy is not taken into account in that case), and between fast and slow memory of one processor in the sequential case. These algorithms were shown to be as stable as the corresponding classic algorithms as for example those implemented in LAPACK [13] and ScaLAPACK [22]. They also exhibited a good performance on distributed memory machines [37, 60], on multicore processors [41], and on grids [10]. The distributed version of CALU and CAQR are implemented using MPI processes. For this version, the input matrix is partitioned over processors. And during the different steps of the factorization, the processors communicate via MPI messages. The multithreaded version of CALU [41] is based on threads. It is adapted for multicore processors. For this version, each operation applied to a block is identified as a task, which is scheduled statically or dynamically to the available cores. However, none of these algorithms has addressed the more realistic model of today's hierarchical parallel computers, which combine both memory hierarchy and parallelism hierarchy.

In this chapter we introduce two hierarchical algorithms computing the LU factorization of a dense matrix. These algorithms are adapted for computer systems with multiple levels of parallelism, and strongly rely on the CALU routine [59, 60]. Hence the first algorithm is based on bi-dimensional recursive calls to CALU, since CALU is recursively performed on blocks of a given panel. Therefore it can be called 2D-multilevel CALU. While the second algorithm is based on recursive calls to CALU on the entire panel. Thus it illustrates a 1D-multilevel CALU algorithm, where the recursion is unidimensional. For simplicity, throughout this work, we refer to the first algorithm as multilevel CALU (ML-CALU) and to the second algorithm as recursive CALU (Rec-CALU). We recall that the initial 1-level CALU algorithm is described in details in section 1.3.5.

Both multilevel CALU (ML-CALU) and recursive CALU (Rec-CALU) use the same approach as CALU, and they are based on tournament pivoting and an optimal distribution of the input matrix among compute units. Hence the main goal is to reduce the communication at each level of the hierarchy. For multilevel CALU, each building block is itself a recursive function that allows to be optimal at the next level of the hierarchy of parallelism. For the panel factorization, at each node of the reduction tree of tournament pivoting, ML-CALU is used instead of GEPP to select pivot rows, using an optimal layout adapted to the current level of parallelism. Hence it minimizes the communication in terms of both bandwidth and latency at each level of the hierarchy, and it is based on a multilevel tournament pivoting strategy. While recursive CALU only minimizes the volume of communication but the pivoting strategy used is tournament pivoting. In fact recursive CALU recursively calls CALU to perform the factorization of the entire current panel, which means that at the deepest level of the recursion, the communication is performed along all the hierarchy. We also model the performance of our two approaches by computing the number of floating-point operations, the volume of communication, and the number of messages exchanged on a computer system with two levels of parallelism. We show that multilevel CALU is optimal at every level of the hierarchy and attains the lower bounds on communication of the LU factorization (modulo polylogarithmic factors). The lower bounds on communication for the multiplication of two dense matrices were introduced in [70, 72] and were shown to apply to LU factorization in [37]. We discuss how these bounds can be used in the case of two levels of parallelism under certain conditions on the memory constraints. Due to the multiple calls to CALU, both multilevel CALU and recursive CALU perform additional flops compared to 1-level CALU. It is known in the literature that in some cases, these extra flops can degrade the performance of a recursive algorithm (see for example [48]). However, for two levels of parallelism, the choice of an optimal layout at each level of the hierarchy allows to keep the extra flops as a lower order term. Furthermore, while recursive CALU has

the same stability as 1-level CALU, multilevel CALU may change the stability of the 1-level CALU. We argue in section 4.2.2 through numerical experiments that 2-level CALU and 3-level CALU specifically studied here are stable in practice. We also show that multilevel CALU is up to 4.5 times faster than the corresponding routine PDGETRF from ScaLAPACK tested in multithreaded mode on a cluster of multicore processors.

Note that, in the following, multilevel CALU refers to bi-dimensional (2D) multilevel CALU and recursive CALU refers to uni-dimensional (1D) multilevel CALU, regarding the recursion pattern used. Note also that for simplicity, i-level CALU refers to i-level multilevel CALU and i-recursive CALU refers to i-level recursive CALU. Thus for example, by 2-level CALU we mean 2-level multilevel CALU.

The remainder of the chapter is organized as follows. We start by introducing the multilevel CALU algorithm in section 4.2 and discussing its stability in section 4.2.2, where we provide experiments showing that multilevel CALU is stable in practice (at least up to three levels of parallelism). Then we present the recursive CALU algorithm in section 4.3. In section 4.4, we analyze the cost of respectively 2-recursive CALU and 2-level CALU. Then we show through a set of experiments that multilevel CALU (2 levels) outperforms the corresponding routine PDGETRF from ScaLAPACK (section 4.5). Finally we present our conclusions ans final remarks in section 4.6.

Note that the multilevel CALU algorithm and the evaluation of the performance of 2-level CALU on a multi-core cluster system was performed as a collaboration with Simplice Donfack and the text presented in section 4.2 and section 4.5 is in part from [A1].

4.2 LU factorization suitable for hierarchical computer systems

In this section we consider a hierarchical computer system with l levels of parallelism. Each compute unit at a given level i is formed by P_{i-1} compute units of level $i-1$. Correspondingly, the memory associated with a compute unit at level i is formed by the sum of the memories associated with the P_{i-1} compute units of level $i-1$. Level l is the topmost level and level 1 is the deepest level of the hierarchy of parallelism. Here we describe our multilevel CALU algorithm and show how it is suitable for such an architecture. Note that we use such a hierarchical system in the entire chapter.

In this section we introduce a multilevel communication avoiding LU factorization, presented in Algorithm 12, that is suitable for a hierarchical computer system with l levels of parallelism. Multilevel CALU aims at minimizing the communication amount at every level of a hierarchical system. Multilevel CALU is based on a bi-dimensional recursive approach, where at every level of the recursion optimal parameters are chosen, such as optimal layout and distribution of the matrix over compute units, optimal reduction tree for tournament pivoting.

Multilevel CALU is applied to a matrix A of size $m \times n$. For simplicity we consider that the input matrix A is partitioned into blocks of size $b_l \times b_l$,

$$A = \begin{pmatrix} A_{11} & A_{12} & \dots & A_{1N} \\ A_{21} & A_{22} & \dots & A_{2N} \\ \vdots & \vdots & & \vdots \\ A_{M1} & A_{M2} & \dots & A_{MN} \end{pmatrix},$$

where $M = m/b_l$ and $N = n/b_l$. The block size b_l and the dimension of the grid $P_{r_l} \times P_{c_l}$ are chosen such that the communication at the top level of the hierarchy is minimized, by following the same approach as for the 1-level CALU algorithm [60]. That is, the blocks of the matrix are distributed among the P_l compute units using a two-dimensional block cyclic distribution over the two-dimensional grid of compute units $P_l = P_{r_l} \times P_{c_l}$ (we will discuss later the values of P_{r_l} and P_{c_l}) and tournament pivoting is based on a binary reduction tree. At each step of the factorization, a block of b_l columns (panel) of L is factored, a block of b_l rows of U is computed, and then the trailing submatrix is updated. Each of these steps is performed by calling recursively functions that will be able to minimize communication at the next levels of the hierarchy of parallelism.

4.2.1 Multilevel CALU algorithm (2D multilevel CALU)

Algorithm 12 receives as input the matrix A of size $m \times n$, the number of levels of parallelism in the hierarchy l, and the number of compute units P_l that will be used at the topmost level of the hierarchy and that are organized as a two-dimensional grid of compute units of size $P_l = P_{r_l} \times P_{c_l}$.

Note that Algorithms 12 and 13 do not detail the communication performed during the factorization, which is triggered by the distribution of the data. A multilevel CALU algorithm (Algorithm 16), where we give the details of the communication between the processors is presented in section 5.4. By abuse of notation, the permutation matrices need to be considered as extended by identity matrices to the correct dimensions. For simplicity, we also consider that the number of processors are powers of 2.

We describe in more detail now the panel factorization (line 8 of Algorithm 12) computed by using multilevel TSLU (ML-TSLU), described in Algorithm 13. ML-TSLU is a multilevel version of TSLU, the panel factorization used in the 1-level CALU algorithm [59]. Let B denote the first panel of size $m \times b_l$, which is partitioned into P_{r_l} blocks. As in TSLU, ML-TSLU is performed in two steps. In the first step, a set of b_l pivot rows are selected by using tournament pivoting. In the second step, these rows are permuted into the first positions of the panel, and then the LU factorization with no pivoting of the panel is computed. Tournament pivoting uses a reduction operation, where at the leaves of the tree b_l candidate pivot rows are selected from each block B_I of B. Then a tournament is performed among the P_{r_l} sets of candidate pivot rows to select the final pivot rows that will be used for the factorization of the panel. Thus the reduction tree is traversed bottom-up. At each node of the reduction tree, a new set of candidate pivot rows is selected from the sets of candidate pivot rows of the nodes of the previous level in the reduction tree. 1-level TSLU uses GEPP as a reduction operator to select a set of candidate pivot rows. However this means that at the next level of parallelism, the compute units involved in one GEPP factorization will need to exchange $O(b_l)$ messages for each call to GEPP because of the use of partial pivoting, and hence the number of messages will not be minimized at that level of the hierarchy of parallelism. While multilevel TSLU (ML-TSLU) selects a set of pivot rows by calling multilevel CALU at each node of the reaction tree, hence being able to minimize communication cost (modulo polylogarithmic factors) at the next levels of parallelism. That is, at each level of the reduction tree every compute unit from the topmost level calls multilevel CALU on its blocks with adapted parameters and data layout. At the deepest level of recursion (level 1), 1-level CALU is called. Note that in the algorithms we refer to 1-level CALU as CALU.

After the panel factorization, the trailing submatrix is updated using a multilevel solve for a triangular system of equations (referred to as dtrsm) and a multilevel algorithm for multiplying two matrices (referred to as dgemm). We do not detail here these algorithms, but one should use recursive versions of

Cannon [27], or SUMMA [53], or a cache oblivious approach [51] if the transfer of data across different levels of the memory hierarchy is to be minimized as well (with appropriate data storages).

In this chapter we will estimate the performance of the multilevel CALU algorithm on a machine with two levels of parallelism. In the next chapter we will estimate the running time of multilevel CALU in a hierarchical platform, and there we will use multilevel Cannon to model the matrix-matrix computations, this algorithm is introduced later in section 5.3.

Algorithm 12: ML-CALU: multilevel communication avoiding LU factorization

Input: $m \times n$ matrix A, level of parallelism l in the hierarchy, block size b_l, number of compute units $P_l = P_{r_l} \times P_{c_l}$

if $l == 1$ **then**

  $[\Pi_l, L_l, U_l] = \text{CALU}(A, b_l, P_l)$

else

  $M = m/b_l, N = n/b_l$

  for $K = 1$ *to* N **do**

    $[\Pi_{KK}, L_{K:M,K}, U_{KK}] = ML - TSLU(A_{K:M,K}, l, b_l, P_{r_l})$

    /* 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 on P_{c_l} compute units */

    /* 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 on P_l compute units */

Figure 4.1 illustrates the first step of multilevel TSLU on a machine with two levels of parallelism that is each compute unit 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 units to synchronize and perform inter-node communication. The deepest reduction trees represent the communication inside each compute unit.

As we can see in Figure 4.1, at the deepest level of the parallelism, each block of size $(m/P_2) \times b_2$ assigned to one compute unit is again partitioned into block columns of size b_1 ($b_2 = 3b_1$ in the figure) and 1-level CALU is performed on these blocks by applying some steps of 1-level TSLU to panels of size b_1. These blocks correspond to the matrices located on the leaves of the topmost reduction tree. At the end of this step, b_2 candidate pivot rows are selected from each block. Then they are merged tow by two,and 1-level CALU is performed on blocks of size $2b_2 \times b_2$ using a panel of size b_1. Here again b_1 candidate pivot rows are selected from each panel and at the end $b_2 = 3b_1$ pivot rows are selected from the entire block. Then the final set of b_2 pivot rows is moved to the first position and the

Algorithm 13: ML-TSLU: multilevel panel factorization

Input: panel B, level of parallelism l in the hierarchy, block size b_l, number of compute units P_{r_l}
Partition B on P_{r_l} blocks /* Here $B = (B_1^T, B_2^T, \cdots, B_{P_{r_l}}^T)^T$ */
/*Each compute unit owns a block B_I*/
for ***each*** *block* B_I ***in parallel*** **do**
  $[\Pi_I, L_I, U_I]$ = ML-CALU$(B_I, l-1, b_{l-1}, P_{l-1})$
  Let B_I be formed by the pivot rows, $B_I = (\Pi_I B_I)(1 : b_l, :)$
for $level$ = 1 *to* $log_2(P_{r_l})$ **do**
  for ***each*** *block* B_I ***in parallel*** **do**
    if $((I-1) \mod 2^{level-1} == 0)$ **then**
      $[\Pi_I, L_I, U_I]$ = ML-CALU$([B_I; B_{I+2^{level-1}}], l-1, b_{l-1}, P_{l-1})$
      Let B_I be formed by the pivot rows, $B_I = (\Pi_I [B_I; B_{I+2^{level-1}}])(1 : b_l, :)$
Let Π_{KK} be the permutation performed for this panel
/* Compute block column of L */
for ***each*** *block* B_I ***in parallel*** **do**
  $L_I = B_I U_1(1 : b_l, :)^{-1}$ /* using multilevel dtrsm on P_{r_l} compute units */

LU factorization with no pivoting of the panel is performed. We note that at the second level we select blocks of size $b_1 \times b_1$ that we extend to blocks of size $b_1 \times b_2$, which are used as pivots for the entire panel. We show in the section 4.2.2 that this could affect the numerical stability of multilevel CALU especially beyond 2 or 3 levels of parallelism.

4.2.2 Numerical stability of multilevel CALU

We discuss here the numerical stability of multilevel CALU factorization. In particular we focus on the 2-level CALU case. Note that we were unable to derive a theoretical upper bound for the growth factor of multilevel CALU. However the entire experiment set we performed suggests that multilevel CALU is stable in practice, at least up to three levels of parallelism. We also present experimental results for both 2-level CALU and 3-level CALU on random matrices and a set of special matrices presented in Appendix A.

On the numerical stability of the 2-level CALU case

In this section we show that there is a relation between 2-level CALU and Gaussian elimination with partial pivoting at each node of the reduction tree. However, we are not able to find a direct relation between these blocks along the reduction tree. Thus multilevel CALU does not allow to obtain a bounded growth factor. Here we present our approach to make conclusion about the numerical stability of multilevel CALU.

To address the numerical stability of the 2-level CALU factorization, we consider the matrix A

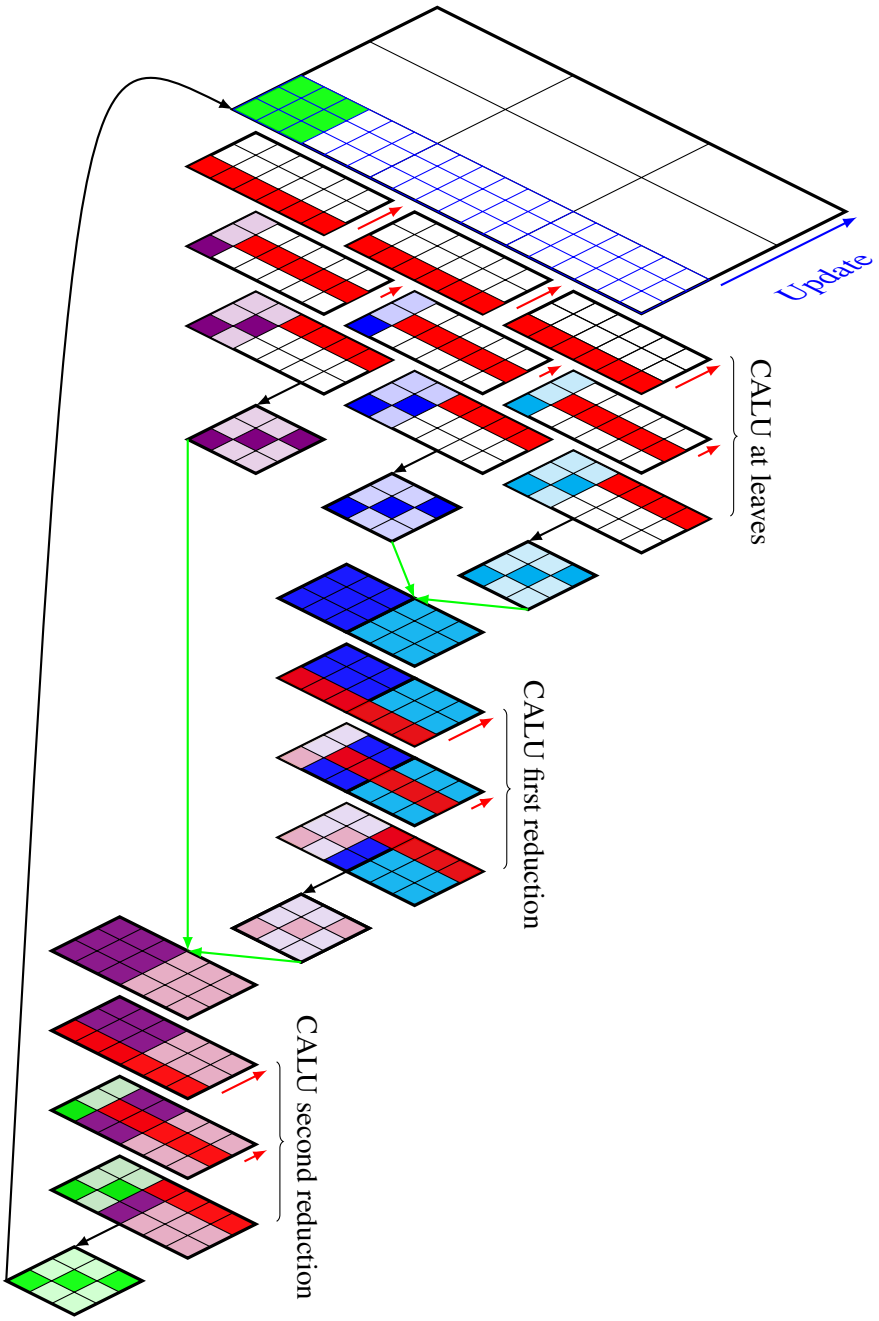


Figure 4.1: First step of multilevel TSLU on a machine with two levels of parallelism

partitioned as follows,

$$A = \begin{bmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \\ A_{41} & A_{42} & A_{43} \\ A_{51} & A_{52} & A_{53} \\ A_{61} & A_{62} & A_{63} \end{bmatrix},$$

where A_{11}, A_{12}, A_{21}, A_{22}, A_{41}, A_{42}, A_{51} and A_{52} are of size $b_1 \times b_1$, A_{31}, A_{32}, A_{61}, and A_{62} are of size $(\frac{m}{2} - 2b_1) \times b_1$, A_{13}, A_{23}, A_{43}, and A_{53} are of size $b_1 \times (n - 2b_1)$, and A_{33} and A_{63} are of size $(\frac{m}{2} - 2b_1) \times (n - 2b_1)$.

We first focus on 2-level TSLU, that is the factorization of the panel. We show that the reduction operation performed by 2-level TSLU at each block located at a node of the global reduction tree is equivalent to the reduction operation performed by 1-level TSLU on a larger block. Hence as stated before, there is an equivalence between 1-level CALU and Gaussian elimination with partial pivoting in exact arithmetic. However, the entire preprocessing phases are not equivalent. Thus the different levels of the global reduction tree are decoupled. Thereby we were unable to derive a bound on the blocks which are not involved in the final reduction operation.

We suppose that $b_2 = 2b_1$, and we perform 2-level CALU on the matrix A. We first consider the panel $A(:, 1 : b_2)$, and we apply 2-level TSLU on that panel. That is, we perform 1-level CALU on the blocks located at the leaves of the global reduction tree,

$$\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \\ A_{31} & A_{32} \end{bmatrix} and \begin{bmatrix} A_{41} & A_{42} \\ A_{51} & A_{52} \\ A_{61} & A_{62} \end{bmatrix},$$

using a panel of size b_1. For that we perform two 1-level TSLUs on the blocks $A(1 : \frac{m}{2} - 2b_1, 1 : b_1)$ and $A(\frac{m}{2} - 2b_1 + 1 : m, 1 : b_1)$. Here, without loss of generality, we consider the simple case where the reduction tree used to apply 1-level TSLU is of height one.

We first perform Gaussian elimination with partial pivoting (GEPP) on the blocks $[A_{21}; A_{31}]$ and $[A_{51}; A_{61}]$. At this step we suppose that the pivots are the diagonal elements of the blocks A_{21} and A_{51}. Then GEPP is applied to the blocks $[A_{11}; A_{21}]$ and $[A_{41}; A_{51}]$. After this step, we refer to the pivot rows selected for the factorization of the first panel of each block as $\bar{A}_{11}$ and $\bar{A}_{41}$. In order to use the schur complement notation for the global trailing matrix, here we note $\tilde{A}_{ij}$ an updated block and $\tilde{\bar{A}}_{ij}$ a permuted and updated block. We note Π_0 the computed permutation. After applying Π_0 to the original matrix, we obtain:

$$(\Pi_0 A)(1 : (\frac{m}{2} - 2b_1), 1 : b_2) = \begin{bmatrix} \bar{A}_{11} & \bar{A}_{12} \\ \bar{A}_{21} & \bar{A}_{22} \\ A_{31} & A_{32} \end{bmatrix} = \begin{bmatrix} \bar{L}_{11} & & \\ \bar{L}_{21} & I_{b_1} & \\ \bar{L}_{31} & & I_{\frac{m}{2}-2b_1} \end{bmatrix} \cdot \begin{bmatrix} \bar{U}_{11} & \bar{U}_{12} \\ & \tilde{\bar{A}}_{22} \\ & \tilde{A}_{32} \end{bmatrix}$$

$$(\Pi_0 A)((\frac{m}{2} - 2b_1) + 1 : m, 1 : b_2) = \begin{bmatrix} \bar{A}_{41} & \bar{A}_{42} \\ \bar{A}_{51} & \bar{A}_{52} \\ A_{61} & A_{62} \end{bmatrix} = \begin{bmatrix} \bar{L}_{41} & & \\ \bar{L}_{51} & I_{b_1} & \\ \bar{L}_{61} & & I_{\frac{m}{2}-2b_1} \end{bmatrix} \cdot \begin{bmatrix} \bar{U}_{41} & \bar{U}_{42} \\ & \tilde{\bar{A}}_{52} \\ & \tilde{A}_{62} \end{bmatrix}.$$

Then a second 1-level TSLU is applied to the panels $[\tilde{\bar{A}}_{22}; \tilde{A}_{32}]$ and $[\tilde{\bar{A}}_{52}; \tilde{A}_{62}]$ with respect to the

following reduction trees, where we suppose that the pivot rows are already on the first positions,

$$\begin{matrix}\tilde{\bar{A}}_{22}\\ \tilde{A}_{32}\end{matrix} \searrow \tilde{\bar{A}}_{22} \qquad\qquad \begin{matrix}\tilde{\bar{A}}_{52}\\ \tilde{A}_{62}\end{matrix} \searrow \tilde{\bar{A}}_{52}$$

Thus for the second panels, the b_1 selected pivot rows from each block are $\tilde{\bar{A}}_{22}$ and $\tilde{\bar{A}}_{52}$. For each block, these 2 steps of 1-level TSLU are equivalent to some b_1 steps of GEPP which involves independent factorizations of submatrices of dimension $b_1(H_1+1)$, where H_1 is the height of the reduction tree at the deepest level of parallelism. We note that each independent factorization updates only one block of the trailing matrix. Moreover, the update is only performed on the trailing matrix corresponding to the block where 1-level CALU is applied, and not the entire trailing matrix.
Let's consider the block,

$$A_1 = (\Pi_0 A)(1:(\frac{m}{2}-2b_1), 1:n) = \begin{bmatrix} \bar{A}_{11} & \bar{A}_{12} & \bar{A}_{13} \\ \bar{A}_{21} & \bar{A}_{22} & \bar{A}_{23} \\ A_{31} & A_{32} & A_{33} \end{bmatrix},$$

and the following matrices:

$$(1): \begin{bmatrix} \bar{A}_{11} & & \bar{A}_{12} & \bar{A}_{13} \\ A_{21} & A_{21} & & \\ & -A_{31} & A_{32} & A_{33} \end{bmatrix}, (2): \begin{bmatrix} \bar{A}_{11} & \bar{A}_{12} & \bar{A}_{13} \\ \bar{A}_{21} & \bar{A}_{22} & \bar{A}_{23} \end{bmatrix}, and\,(3): \begin{bmatrix} \tilde{\bar{A}}_{22} & \tilde{\bar{A}}_{23} \\ \tilde{\bar{A}}_{32} & \tilde{\bar{A}}_{33} \end{bmatrix}.$$

The blocks $\tilde{\bar{A}}_{22}$ and $\tilde{\bar{A}}_{23}$ are obtained after performing b_1 steps of GEPP on the matrix (2) (or directly on the matrix A_1, but we define here matrix (2) to detail the different steps). The blocks $\tilde{\bar{A}}_{32}$ and $\tilde{\bar{A}}_{33}$ are obtained after performing 2 b_1 steps of GEPP on the matrix (1). These independent GEPP factorizations correspond to the first 1-level TSLU of the 1-level CALU factorization applied to A_1. Then b_1 steps of GEPP are applied to the matrix (3), which correspond to the second 1-level TSLU factorization performed on A_1. At this level, as described before, we suppose that the pivot rows are already on the first positions $\tilde{\bar{A}}_{22}$. The different GEPP steps detailed previously could be applied to the following matrix,

$$G_1 = \begin{bmatrix} \bar{A}_{11} & & & \bar{A}_{12} & \bar{A}_{13} \\ A_{21} & A_{21} & & & \\ & & \bar{A}_{11} & \bar{A}_{12} & \bar{A}_{13} \\ & & \bar{A}_{21} & \bar{A}_{22} & \bar{A}_{23} \\ & -A_{31} & & A_{32} & A_{33} \end{bmatrix},$$

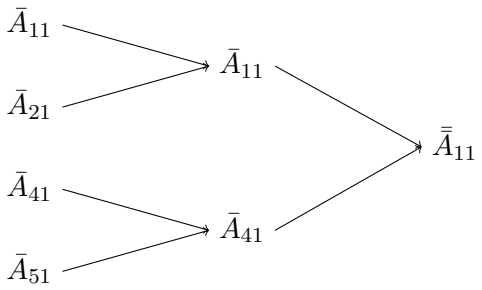
$$G_1 = \begin{bmatrix} \bar{L}_{11} & & & & \\ A_{21}\bar{U}_{11}^{-1} & L_{21} & & & \\ & & \bar{L}_{11} & & \\ & & \bar{L}_{21} & \tilde{\bar{L}}_{22} & \\ & -L_{31} & & \tilde{L}_{32} & I_{\frac{m}{2}-2b_2} \end{bmatrix} . \begin{bmatrix} \bar{U}_{11} & & & \bar{U}_{12} & \bar{U}_{13} \\ & U_{21} & & -L_{21}^{-1}A_{21}\bar{U}_{11}^{-1}\bar{U}_{12} & -L_{21}^{-1}A_{21}\bar{U}_{11}^{-1}\bar{U}_{13} \\ & & \bar{U}_{11} & \bar{U}_{12} & \bar{U}_{13} \\ & & & \tilde{\bar{U}}_{22} & \tilde{\bar{U}}_{23} \\ & & & & \tilde{A}_{33}^{s} \end{bmatrix}.$$

Then applying 2 steps of 1-level CALU with a panel of size b_1 to the block A_1 is equivalent to performing $4 \times b_1$ steps of GEPP on the matrix G_1. This reasoning applies to each block located at a node of the topmost reduction tree. Thus a matrix G_2 similar to G_1 can be derived for the second block.

At the second level of the topmost reduction tree, the two $b_2 \times b_2$ blocks formed by the selected candidate pivot rows are merged, and again 1-level CALU is performed on the resulting $2b_2 \times b_2$ block. We note Π_1 the permutation computed during this step. It is extended by the appropriate identity matrices to the correct dimension.

$$\begin{bmatrix} (\Pi_0\Pi_1 A)(1:b_2, 1:b_2) \\ (\Pi_0\Pi_1 A)(\frac{m}{2} - 2b_1 + 1 : \frac{m}{2} - 2b_1 + 1 + b_2, 1:b_2) \end{bmatrix} = \begin{bmatrix} \bar{A}_{11} & \bar{A}_{12} \\ \bar{A}_{21} & \bar{A}_{22} \\ \bar{A}_{41} & \bar{A}_{42} \\ \bar{A}_{51} & \bar{A}_{52} \end{bmatrix},$$

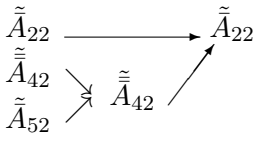
We first perform 1-level TSLU on the first panel $[\bar{A}_{11}; \bar{A}_{21}; \bar{A}_{41}; \bar{A}_{51}]$ of size $2b_2 \times b_1$ using the following binary reduction tree.



At the first level of the reduction tree, we suppose that the pivot rows are on the blocks $\bar{A}_{11}$ and $\bar{A}_{41}$. Then we merge the two blocks $\bar{A}_{11}$ and $\bar{A}_{41}$, and apply GEPP to the panel $[\bar{A}_{11}; \bar{A}_{41}]$ to select b_1 candidate pivot rows. We note Π_2 the permutation computed during this step. Then we obtain:

$$\Pi_2 \times \begin{bmatrix} \bar{A}_{11} & \bar{A}_{12} \\ \bar{A}_{21} & \bar{A}_{22} \\ \bar{A}_{41} & \bar{A}_{42} \\ \bar{A}_{51} & \bar{A}_{52} \end{bmatrix} = \begin{bmatrix} \bar{\bar{A}}_{11} & \bar{\bar{A}}_{12} \\ \bar{A}_{21} & \bar{A}_{22} \\ \bar{\bar{A}}_{41} & \bar{\bar{A}}_{42} \\ \bar{A}_{51} & \bar{A}_{52} \end{bmatrix} = \begin{bmatrix} \bar{\bar{L}}_{11} & & & \\ \bar{L}_{21} & I_{b_1} & & \\ \bar{\bar{L}}_{41} & & I_{b_1} & \\ \bar{L}_{51} & & & I_{b_1} \end{bmatrix} \times \begin{bmatrix} \bar{\bar{U}}_{11} & \bar{\bar{U}}_{12} \\ & \bar{A}^s_{22} \\ & \bar{\bar{A}}^s_{42} \\ & \bar{A}^s_{52} \end{bmatrix}.$$

Here we use the same notations as in the previous step and we perform 1-level TSLU on the panel $[\tilde{\bar{A}}_{22}; \tilde{\bar{\bar{A}}}_{42}; \tilde{\bar{A}}_{52}]$ with respect to the following local reduction tree,



Again the entire 1-level CALU process, that is the two steps of 1-level TSLU, is equivalent, in exact arithmetic, to performing GEPP on the following matrix, formed by permuted blocks of the input matrix A,

$$G = \begin{bmatrix} \bar{\bar{A}}_{11} & & & & \bar{\bar{A}}_{12} & & \bar{\bar{A}}_{13} \\ & \bar{\bar{A}}_{11} & & & & \bar{\bar{A}}_{12} & \bar{\bar{A}}_{13} \\ & & \bar{\bar{A}}_{11} & & \bar{\bar{A}}_{12} & & \bar{\bar{A}}_{13} \\ & & & \bar{\bar{A}}_{11} & & \bar{\bar{A}}_{12} & \bar{\bar{A}}_{13} \\ & & \bar{A}_{21} & & \bar{A}_{22} & & \bar{A}_{23} \\ & & & \bar{A}_{21} & & \bar{A}_{22} & \bar{A}_{23} \\ \bar{\bar{A}}_{41} & & & & \bar{\bar{A}}_{42} & & \bar{\bar{A}}_{43} \\ & \bar{A}_{51} & & & & -\bar{A}_{52} & \bar{A}_{53} \end{bmatrix}.$$

Note that applying a reduction across the pivot row sets selected from matrices G_1 and G_2, we are not able to construct the matrix G. Thus we were unable to make conclusion about the theoretical numerical stability of 2-level CALU and as a consequence about multilevel CALU. Hence for multilevel CALU we are not able to derive a bound on the growth factor.

Experimental results

It was shown in [60] that communication avoiding LU is very stable in practice. Since multilevel CALU is based on a bi-dimensional recursive call to CALU, its stability can be different from that of CALU. We present here a set of experiments performed on random matrices and a set of special matrices that were also used in [60] for discussing the stability CALU. We present numerical results both 2-level CALU and 3-level CALU. These results show that up to three levels of parallelism multilevel CALU exhibits a good stability, however further investigation is required if more than three levels of parallelism are used. The size of the test matrices is varying from 1024 to 8192. 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 $\|PA - LU\|/\|A\|$.

2-level CALU

For 2-level CALU we use different combinations of the number of processors P_1 and P_2 and of the panel sizes b_1 and b_2. For all the test matrices, the worst growth factor obtained is of order 10^2.

Figure 4.2 shows the values of the growth factor g_W for binary tree based 2-level CALU for different block sizes b_2 and b_1 and different number of processors P_2 and P_1. As explained previously, the block size b_2 determines the size of the panel at the top level of parallelism and the block size b_1 determines the size of the panel at the deep level of parallelism, while the number of processors P_2 and P_1 determine the number of block rows in which the panels are partitioned at each level of parallelism. This corresponds respectively to the number of leaves of the top level binary reduction tree, and the deep level binary reduction tree. In our tests on random matrices, we observe that the curves of the growth factor lie between $\frac{1}{5}n^{2/3}$ and $\frac{2}{5}n^{2/3}$. We can also note that the growth factor of binary tree based 2-level CALU has similar behavior to the growth factor of both GEPP and binary tree based 1-level CALU.

Table 4.1: Stability of the linear solver using binary tree based 2-level CALU, binary tree based CALU, and GEPP.

n	P_2	b_2	P_1	b_1	η	w_b	N_{IR}	HPL3
Binary tree based 2-level CALU								
8192	64	32	4	8	6.1E-14	4.7E-14	2	4.7E-03
		16	4	4	5.8E-14	4.5E-14	2	4.9E-03
	32	64	4	16	6.4E-14	4.7E-14	2	5.2E-03
		32	4	8	6.2E-14	4.6E-14	2	4.4E-03
		16	4	4	5.8E-14	4.3E-14	2	4.5E-03
	16	128	4	32	6.3E-14	4.4E-14	2	5.2E-03
		64	4	16	6.3E-14	4.6E-14	2	4.7E-03
		32	4	8	6.2E-14	4.5E-14	2	5.0E-03
		16	4	4	5.7E-14	4.3E-14	2	4.9E-03
4096	64	16	4	4	2.9E-14	2.2E-14	2	5.0E-03
	32	32	4	8	3.1E-14	2.1E-14	2	5.1E-03
		16	4	4	3.0E-14	2.0E-14	2	4.5E-02
	16	64	4	16	3.2E-14	2.3E-14	2	5.4E-03
		32	4	8	3.1E-14	2.0E-14	2	4.6E-03
		16	4	4	2.9E-14	2.1E-14	2	5.1E-03
2048	32	16	4	4	1.5E-14	1.0E-14	1.8	4.5E-03
	16	32	4	8	1.5E-14	1.0E-14	2	4.8E-03
		16	4	4	1.5E-14	9.9E-15	1.8	4.5E-03
1024	16	16	4	4	7.4E-15	5.2E-15	1.5	4.9E-03
Binary tree based CALU								
8192	256	32	-	-	6.2E-15	4.1E-14	2	4.5E-03
		16	-	-	5.8E-15	3.9E-14	2	4.1E-03
	128	64	-	-	6.1E-15	4.2E-14	2	4.6E-03
		32	-	-	6.3E-15	4.0E-14	2	4.4E-03
		16	-	-	5.8E-15	4.0E-14	2	4.3E-03
	64	128	-	-	5.8E-15	3.6E-14	2	3.9E-03
		64	-	-	6.2E-15	4.3E-14	2	4.4E-03
		32	-	-	6.3E-15	4.1E-14	2	4.5E-03
		16	-	-	6.0E-15	4.1E-14	2	4.2E-03
4096	256	16	-	-	3.1E-15	2.1E-14	1.7	4.4E-03
	128	32	-	-	3.2E-15	2.3E-14	2	5.1E-03
		16	-	-	3.1E-15	1.8E-14	2	4.0E-03
	64	64	-	-	3.2E-15	2.1E-14	1.7	4.6E-03
		32	-	-	3.2E-15	2.2E-14	1.3	4.7E-03
		16	-	-	3.1E-15	2.0E-14	2	4.3E-03
2048	128	16	-	-	1.7E-15	1.1E-14	1.8	5.1E-03
	64	32	-	-	1.7E-15	1.0E-14	1.6	4.6E-03
		16	-	-	1.6E-15	1.1E-14	1.8	4.9E-03
1024	64	16	-	-	8.7E-16	5.2E-15	1.6	4.7E-03
GEPP								
8192	-				3.9E-15	2.6E-14	1.6	2.8E-03
4096	-				2.1E-15	1.4E-14	1.6	2.9E-03
2048	-				1.1E-15	7.4E-15	2	3.4E-03
1024	-				6.6E-16	4.0E-15	2	3.7E-03

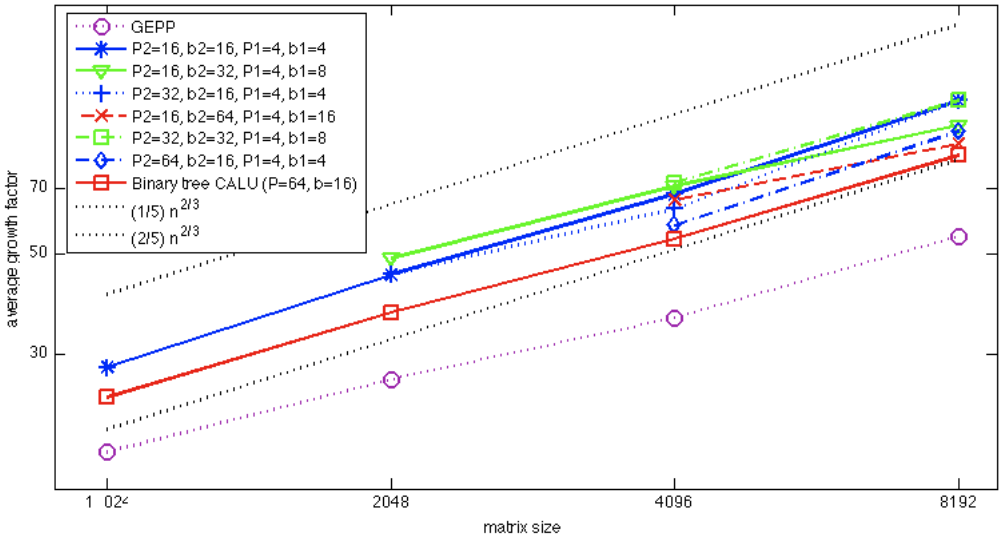


Figure 4.2: Growth factor g_W of binary tree based 2-level CALU for random matrices.

Table 4.1 presents results for the linear solver using binary tree based 2-level CALU, together with binary tree based 1-level CALU and GEPP for the comparison. The normwise backward stability is evaluated by computing an accuracy test denoted as HPL3 (2.9). We also display the normwise backward error η (2.10), the componentwise backward error w_b (2.11) before iterative refinement, and the number of steps of iterative refinement N_{IR}. We note that for the different numerical experiments detailed in Table 4.1 , 2-level CALU leads to results within a factor of 10 of the results of both 1-level CALU and GEPP.

More detailed results on random and special matrices are presented in Appendix A in Tables 10, 13, and 16, where we consider different metrics including the norm of the factors L and U, their conditioning, and the value of their maximum element.

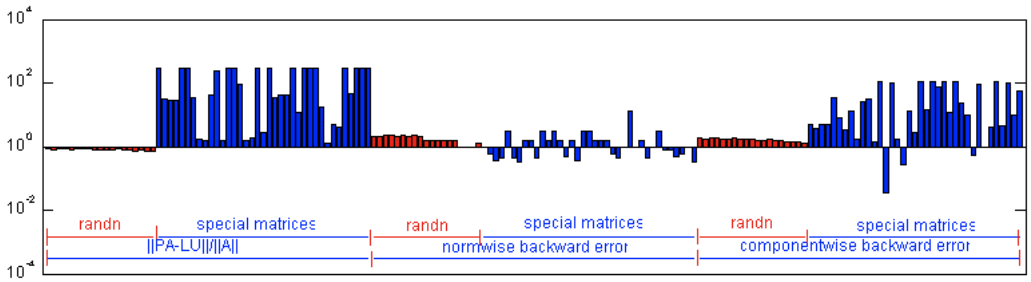


Figure 4.3: The ratios of 2-level CALU's backward errors to GEPP's backward errors.

Figure 4.3 shows the ratios of 2-level CALU's backward errors to those of GEPP for random matrices of varying sizes from 1024 to 8192 and for special matrices of size 4096. For almost all test matrices, 2-level CALU's normwise backward error is at most $3\times$ larger than GEPP's normwise backward error.

However, for special matrices the other two backward errors of 2-level CALU can be larger by a factor of the order of 10^2 than the corresponding backward errors of GEPP. We note that for several test matrices, at most two steps of iterative refinement are necessary to attain the machine epsilon.

3-level CALU

Figure 4.4 displays the values of the ratios of 3-level CALU's growth factor and backward errors to those of GEPP for 36 special matrices (those detailed in Appendix A except poisson matrix). The tested matrices are of size 8192. The parameters used for these numerical experiments are as follows: at the topmost level of parallelism, the number of processors working on the panel factorization is $P_3 = 16$ and the panel size is $b_3 = 64$, that is the blocks located at the leaves of the topmost reduction tree are of size 512×64. At the second level of recursion, the number of processors working on the panel factorization is $P_2 = 4$ and the panel size is $b_2 = 32$, and finally at the deepest level of parallelism, the number of processors working on the panel factorization is $P_1 = 4$ and the panel size is $b_1 = 8$, that is the blocks located at the leaves of the deepest reduction tree are of size 32×8.

Figure 4.4 shows that with these parameters, 3-level CALU exhibits a good numerical stability, even better than that of 2-level CALU. In fact as it can be seen, 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.

The previous results show how crucial the choice of the parameters is for the numerical stability of ML-CALU in general. For that reason, further investigations are needed to be able to conclude on the effect of the recursion on the numerical behavior of the multilevel CALU algorithm.

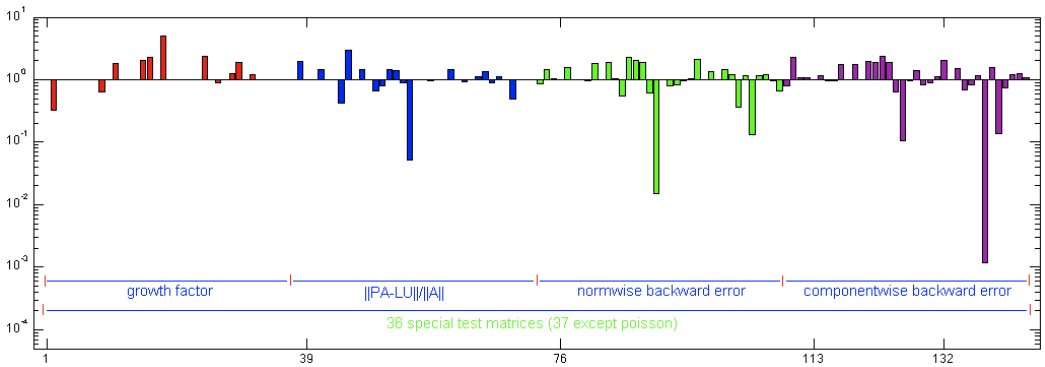


Figure 4.4: The ratios of 3-level CALU's growth factor and backward errors to GEPP's growth factor and backward errors.

Note that in general, ML-CALU uses tournament pivoting to select candidate pivot rows 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, for our two cases (binary tree based 2-level CALU and 3-level CALU), 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. In our numerical experiments, we compute the minimum value and the

average value of τ_k, $\tau_{\min} = \min_k \tau_k$ and $\tau_{ave} = (\sum_{k=1}^{n-1} \tau_k)/(n-1)$, where n is the number of columns of the input matrix. 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 2-level CALU the minimum threshold $\tau_{\min}$ is larger than 0.29 on all our test matrices. For binary tree based 3-level CALU, the selected pivots 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. These values are detailed in Table 17 in Appendix A.

4.3 Recursive CALU (1D multilevel CALU)

Given that multilevel CALU does not allow to obtain a bounded growth factor, we present here a different algorithm which has the same numerical stability as CALU and minimizes the volume of data to send at each level of the hierarchy, but not the number of messages. We refer to this algorithm as Rec-CALU. Here we present the recursive CALU algorithm, then discuss its numerical stability.

4.3.1 Recursive CALU algorithm

Recursive CALU is described in Algorithm 14. 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 units of level 1 at the topmost level of the recursion. These compute units are organized as a two-dimensional grid at each level, $P_i = P_{r_i} \times P_{c_i}$.

As multilevel CALU, recursive CALU is based on the communication avoiding LU factorization (CALU). The main difference between these two algorithms consists of the panel factorization. Thus considering l levels of recursion, Recursive CALU calls recursively CALU at each step of the recursion to perform the entire panel factorization. At each level l of the recursion, the total number of compute units of level 1 working on the panel factorization is $P_r \times \prod_{i=1}^{l-1} P_{c_i}$. Hence only the number of the compute units on the horizontal dimension is varying. For example at the deepest level of recursion, the panel of size $m \times b_1$ is factored using $P_r \times P_{c_1}$ compute units of level 1. This factorization is performed using TSLU, where Gaussian elimination with partial pivoting is applied to blocks of size $(m/P_r) \times b_1$ at the first step of the factorization and at the first level of the reduction tree. Note here that the efficiency of the reduction tree highly depends on the underlying structure of the hierarchical architecture. For example if we assume that the number of processors at each level of the hierarchy and at each dimension is a power of 2, then a binary reduction tree will be optimal. Hence during the reduction process, communication between compute units of level $i > 1$ will only occur after $\log(\prod_{k=1}^{i-1} P_{r_k})$ reduction steps.

Figure 4.5 illustrates the reduction tree used at the deepest level of recursive CALU running on a hierarchy with 3 levels of parallelism. The dashed black arrows correspond to the reduction operation, that is the application of GEPP to the blocks of the current panel. The black arrows represent the intra-node communication, which correspond to the communication between the compute units of level 1. The red arrows represent a communication that involves two compute units belonging to two different nodes of level 2. Finally the green arrows correspond to a communication between two compute units belonging to two different nodes of level 3.

Thus considering a parallel platform combining parallel distributed multi-core nodes, the preprocessing step uses a hierarchical approach. Hence for each panel, first local binary trees are performed

Algorithm 14: Recursive-CALU: recursive 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 */

in parallel within each node, then several global reduction trees are applied across the different levels of the hierarchy. Note that the global reduction operations involve inter-node communications, which are slower than intra-node shared memory accesses. Additionally, the local reduction trees depend on the degree of parallelism of the cluster. A binary tree is more suitable for a cluster with many cores, while a flat tree allows more locality and CPU efficiency. The choice of the reduction trees at each hierarchical level was addressed in the context of tiled QR factorization, where at least two levels of reduction trees are combined [46].

Once the panel is factored, a block of rows of U is computed (line 12 in Algorithm 14). At level l of recursion, this computation is performed using $P_{c_l} \times \prod_{i=1}^{l-1} P_i$ compute units of level 1. Then the trailing matrix is updated (line 17 in Algorithm 14), using $P_r \times \prod_{i=1}^{l} P_{c_i}$ compute units of level 1. Note that in Algorithm 14, we neither detail the communication between the processing node at each level of the hierarchy nor explicit how multilevel dtrsm and multilevel dgemm are performed. Later to estimate the running time of recursive CALU, we assume that these computations are performed using multilevel Cannon (Algorithm 15), a hierarchical algorithm for matrix multiplication.

In terms of stability, recursive CALU is equivalent to calling CALU on a matrix of size $m \times n$, using a block size b_1 and a grid of processors $P = P_r \times P_c$. In other words, 1D multilevel CALU uses tournament pivoting performed on a binary tree of height $\log P_r$.

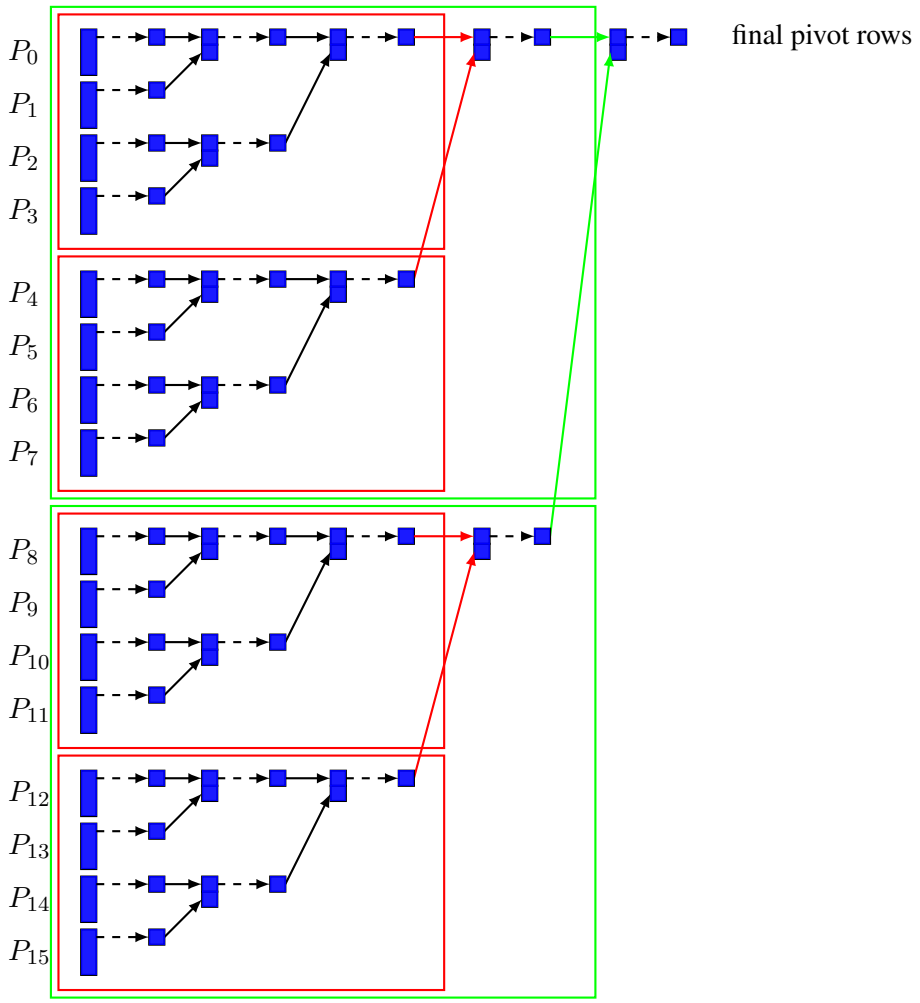


Figure 4.5: TSLU on a hierarchical system with 3 levels of hierarchy.

4.4 Performance model of hierarchical LU factorizations

In this section we present a performance model of both 2-recursive CALU and 2-level CALU factorizations of a matrix of size $n \times n$ in terms of the number of floating-point operations (#flops), the volume of communication (#words moved), and the number of messages exchanged (#messages) during the factorization. Note that our counts are computed on the critical path. We show that 2-level CALU attains the lower bounds modulo polylogarithmic factors at each level of the hierarchy. While 2-recursive CALU attains the lower bounds modulo polylogarithmic factors only at the deepest level of the hierarchy. At the top level of the hierarchy 2-recursive CALU exceeds the lower bound by a factor $\sqrt{P_1}$. Thus 2-recursive CALU is latency bounded at the top level of the hierarchy.

Note that the performance model used for this analysis is quite simplified, since it does not take into account the memory constraints. To be able to give a more general and more realistic performance model for both recursive CALU and multilevel CALU, we introduce in chapter 5 a hierarchical cluster platform model (HCP). There we discuss more in details the drawbacks of the current model and we estimate in a more realistic way the cost of multilevel CALU running on a hierarchical system with l

levels of parallelism.

We condider a hiearchical system with 2 levels of parallelism. Let $P_2 = P_{r2} \times P_{c2}$ be the number of processors and b_2 be the block size at the top level of parallelism. Each compute unit P_2 is formed by $P_1 = P_{r1} \times P_{c1}$ compute units at the bottom level of parallelism. Hence the total number of compute units at the deepest level of parallelism is $P = P_2 \cdot P_1$. Let b_1 the block size at the deepest level of parallelism.

4.4.1 Cost analysis of 2-level CALU

In this section we estimate the flops, the messages, and the words counts of 2-level based multilevel CALU. We note CALU (m, n, P, b) the routine that performs 1-level CALU on a matrix of size $m \times n$ with P processors and a panel of size b.

We first consider the arithmetic cost of 2-level CALU. It includes the factorization of the panel, the computation of a block row of U, and the update of the trailing matrix, at each step k of the algorithm. To factorize the panel k of size b_2, we perform 1-level CALU on each block located at a node of the reduction tree, using a 2D-grid of P_1 smaller compute units and a panel of size b_1. The number of flops performed to factor the k-th panel is,

$$\#\text{flops}(\text{CALU}(\frac{n_k}{P_{r2}}, b_2, P_1, b_1)) + \log P_{r2} \cdot \#\text{flops}(\text{CALU}(2b_2, b_2, P_1, b_1)),$$

where n_k denotes the number of rows of the k-th panel.

To perform the rank-b_2 update, first the input matrix of size $n \times n$ is divided into P_2 blocks of size $(n/P_{r2}) \times (n/P_{c2})$. Then each block is further divided among P_1 compute units. Hence each processor from the deepest level computes a rank-b_2 update on a block of size $n/(P_{r2}.P_{r1}) \times n/(P_{c2}.P_{c1})$. It is then possible to estimate the flops count of this step as a rank-b_2 update of a matrix of size $n \times n$ distributed into $P_{r2}.P_{r1} \times P_{c2}.P_{c1}$ processors. The same reasoning holds for the arithmetic cost of the computation of a block row of U.

We estimate now the communication cost at each level of parallelism. At the topmost level we consider the communication between the P_2 compute units. This corresponds to the communication cost of the initial 1-level CALU algorithm, which is presented in detail in [60]. The size of the memory of one compute unit at the top level is formed by the sum of the sizes of the memories of the compute units at the next level in the hierarchy. We consider here that this size is of $O(n^2/P_2)$, that is each node stores a part of the input and output matrices, and this is sufficient for determining a lower bound on the volume of communication that needs to be performed by our algorithm. However, the number of messages that are transferred at this level depends on the maximum size of data that can be transferred from one compute unit to another compute unit in one single message. We consider here the case when the size of one single message is of the order of (n^2/P_2), which is realistic if shared memory is used at the deepest level of parallelism. However, if the size of one single message is smaller, and it can be as small as (n^2/P) when distributed memory is used at the deepest level of parallelism, the number of messages and the lower bounds presented in this section need to be adjusted for the given memory size. For that reason we introduce a more general performance model in section 5.2.

At the deepest level we consider in addition the communication between the P_1 smaller compute units inside each compute unit of the top level. We note that we consider Cannon's matrix-matrix

multiplication algorithm [27] in our model. Here we detail the communication cost of the factorization of a panel k at the deepest level of parallelism. Inside each node we first distribute the data on a grid of P_1 processors, then we apply 1-level CALU using P_1 processors and a panel of size b_1. Thus,

$$\begin{aligned}\#\text{messages}_k &= \#\text{messages}(\text{CALU}(\tfrac{n_k}{P_{r2}}, b_2, P_1, b_1)) \\ &+ \log P_{r2} \cdot \#\text{messages}(\text{CALU}(2b_2, b_2, P_1, b_1)) + \log P_1 \times (1 + \log P_{r2}),\end{aligned}$$

$$\begin{aligned}\#\text{words}_k &= \#\text{words}(\text{CALU}(\tfrac{n_k}{P_{r2}}, b_2, P_1, b_1)) \\ &+ \log P_{r2} \cdot \#\text{words}(\text{CALU}(2b_2, b_2, P_1, b_1)) + b_2^2 \log P_1 \times (1 + \log P_{r2}).\end{aligned}$$

Then at the deepest level of 2-level CALU, the number of messages and the volume of data that need to be sent during the panel factorizations with respect to the critical path are,

$$\#\text{ messages} = \tfrac{3n}{b_1}(1 + \log P_{r2})(\log P_{r1} + \log P_{c1}),$$

$$\#\text{ words} = n(b_1 + \tfrac{3b_2}{2P_{c1}})(1 + \log P_{r2}) + \tfrac{3}{2}\tfrac{nb_2}{P_{r1}} \log P_{r2} \log P_{c1} + \tfrac{n}{P_{r1}}(\tfrac{m}{P_{r2}} - \tfrac{b_2}{2}) \log P_{c1}.$$

The detailed counts of the entire factorization are presented in Appendix B .3.

We estimate now the performance model for a square matrix using an optimal layout, that is we choose values of P_{ri}, P_{ci}, and b_i at each level of the hierarchy that allow to attain the lower bounds on communication. By following the same approach as in [60], for two levels of parallelism these parameters can be written as $P_{r2} = P_{c2} = \sqrt{P_2}$, $P_{r1} = P_{c1} = \sqrt{P_1}$, $b_2 = \frac{n}{\sqrt{P_2}} \log^{-2}(\sqrt{P_2})$, and $b_1 = \frac{b_2}{\sqrt{P_1}} log^{-2}(\sqrt{P_1}) = \frac{n}{\sqrt{P_1 P_2}} log^{-2}(\sqrt{P_1}) log^{-2}(\sqrt{P_2})$. We note that $P = P_2 \times P_1$.

Table 4.2 presents the performance model of 2-level CALU. It shows that 2-level CALU attains the lower bounds on communication of dense LU factorization, modulo polylogarithmic factors, at each level of parallelism.

4.4.2 Cost analysis of 2-recursive CALU

In this section we focus on 2-recursive CALU. The main difference between this factorization and 2-level CALU lies on the panel factorization. Hence to factor a panel of size b_2, 2-level CALU applies 1-level CALU to the blocks located at the nodes of the global reduction tree using a panel of size b_1 and P_1 compute units of level 1 for each reduction operation, while 2-recursive CALU performs 1-level CALU on the entire panel using a panel of size b_1 and $P_r \times P_{c1}$ compute units of level 1. Thus for each step of TSLU applied to a panel of size b_1, 2-recursive CALU exchanges $O(H_2)$ messages between nodes of level 2, where H_2 is the height of the segment corresponding to the communication between nodes of level 2. However 2-level CALU performs such a communication only once.

The remainder of the factorization, that is the computation of the factor U and the update of the global trailing matrix is performed in the same way as for 2-level CALU. Note that here again we use Cannon's algorithm for matrix-matrix multiplication. In the following, we only detail the panel factorization. At each step of the panel factorization, TSLU is applied to a panel of size b_1 using a reduction tree of height $(H_1 + H_2)$. It performs $O(H_1)$ intra-node messages, that is messages between compute units of level 1, and $O(H_2)$ inter-node messages, that is messages between nodes of level 2. Here we consider a binary reduction tree at both the intra-node and the inter-node levels, $H_2 = \log P_{r2}$, and

Table 4.2: Performance estimation of parallel (binary tree based) 2-level CALU with optimal layout. The matrix factored is of size $n \times n$. Some lower-order terms are omitted.

	Communication cost at the top level of parallelism	Lower bound	Memory size
# messages	$O(\sqrt{P_2}\log^3 P_2)$	$\Omega(\sqrt{P_2})$	$O(\frac{n^2}{P_2})$
# words	$O(\frac{n^2}{\sqrt{P_2}}\log P_2)$	$\Omega(\frac{n^2}{\sqrt{P_2}})$	$O(\frac{n^2}{P_2})$
	Communication cost at the bottom level of parallelism		
# messages	$O(\sqrt{P}\log^6 P_2 + \sqrt{P}\log^3 P_2 \log^3 P_1)$	$\Omega(\sqrt{P})$	$O(\frac{n^2}{P})$
# words	$O(\frac{n^2}{\sqrt{P}}\log^2 P_2)$	$\Omega(\frac{n^2}{\sqrt{P}})$	$O(\frac{n^2}{P})$
	Arithmetic cost of 2-level CALU		
# flops	$\frac{1}{P}\frac{2n^3}{3} + \frac{3n^3}{2P\log^2 P} + \frac{5n^3}{6P\log^3 P}$	$\frac{1}{P}\frac{2n^3}{3}$	

$H_1 = \log P_{r1}$. For the rest of the panel factorization, a communication performed along rows of processors is local, that is inside one node of level 2, while a communication along columns of processors involves both local and global communications, that is inside and between nodes of level 2.

The number of messages and the volume of data at the top level of parallelism are,

$$\text{\# messages} = \frac{3n}{b_1}\log P_{r2} + \frac{2n}{b_2}\log P_{r2} + \frac{3n}{b_2}\log P_{c2},$$

$$\text{\# words} = (\frac{3n^2}{2P_{c2}} + \frac{3nb_2}{2} + nb_1)\log P_{r2} + (\frac{1}{P_{r2}}(mn - \frac{n^2}{2}) + \frac{nb_2}{2} + n)\log P_{c2}.$$

The detailed counts of the entire factorization are presented in Appendix B .4.

Table 4.3 presents the performance model of 2-recursive CALU using the same optimal layout detailed in section 4.4.1. It shows that 2-recursive CALU attains the lower bounds of communication, modulo polylogarithmic factor, at the deepest level of the hierarchy. However it is not optimal at the top level of the hierarchy in terms of latency, unless $P_1 << P_2$, which is not the case in general.

In the following we focus on the performance of multilevel CALU on a multi-core cluster system since it is optimal up to some polylogarithmic factors at each level of the hierarchy, and it is also stable in practice.

4.5 Multilevel CALU performance

4.5.1 Implementation on a cluster of multicore processors

In the following we describe the specific implementation of ML-CALU on a cluster of nodes of multicore processors. At the top level, we have used a static distribution of the data on the nodes,

Table 4.3: Performance estimation of parallel (binary tree based) 2-recursive CALU with optimal layout. The matrix factored is of size $n \times n$. Some lower-order terms are omitted.

	Communication cost at the top level of parallelism	Lower bound	Memory size
# messages	$O(\sqrt{P_2}\sqrt{P_1}\log^2 P_2 \log^3 P_2)$	$\Omega(\sqrt{P_2})$	$O(\frac{n^2}{P_2})$
# words	$O(\frac{n^2}{\sqrt{P_2}}\log P_2)$	$\Omega(\frac{n^2}{\sqrt{P_2}})$	$O(\frac{n^2}{P_2})$
	Communication cost at the bottom level of parallelism		
# messages	$O(\sqrt{P}\log^6 P_2)$	$\Omega(\sqrt{P})$	$O(\frac{n^2}{P})$
# words	$O(\frac{n^2}{\sqrt{P}}\log^2 P_2)$	$\Omega(\frac{n^2}{\sqrt{P}})$	$O(\frac{n^2}{P})$
	Arithmetic cost of 2-recursive CALU		
# flops	$\frac{1}{P}\frac{2n^3}{3} + \frac{n^3}{2P\log^2 P_1 \log^4 P_2} + \frac{5n^3}{6P\log^3 P_1 \log^4 P_2}$	$\frac{1}{P}\frac{2n^3}{3}$	

more specifically the matrix is distributed using a two-dimensional block cyclic partitioning on a two-dimensional grid of processors. This is similar to the distribution used in 1-level CALU [59], and hence the communication between nodes is performed as in the 1-level CALU factorization. For each node, the blocks are again decomposed using a two-dimensional layout, and the computation of each block is associated with a task. A dynamic scheduler is used to schedule the tasks to the available threads as described in [41], and recalled in section 1.5.2.

4.5.2 Performance of 2-level CALU

In this section we evaluate the performance of 2-level CALU on a cluster of multicore processors. The cluster is composed of 32 nodes based on Intel Xeon EMT64 processor. Each node has 8 cores formed by two-socket, quad-core and each core has a frequency of 2.50GHz. The cluster is part of grid 5000 [28].

Both ScaLAPACK and multilevel CALU are linked with the Intel BLAS from MKL 10.1. We compare the performance of our algorithm with the corresponding routines of ScaLAPACK used in a multithreaded mode. The algorithm is implemented using MPI and Pthreads. P is the number of processors, T is the number of threads per node, and b_2 and b_1 are the block sizes used respectively at the topmost and the deepest level of the hierarchy. The choice of the optimal block size at each level depends on the architecture, the number of levels in the hierarchy, and the input matrix size. Thus a small block size increases the parallelism and also the scheduling overhead which can slow down the algorithm. However, a large block size can help to exploit more BLAS-3 possibility but may reduce the parallelism and cause inactivity. In our experiments, we empirically tune block size b_2 and b_1. However the optimal parameters should be determined based on an automatic approach such as an autotuning algorithm for a machine with multiple levels of parallelism.

At the deepest level of parallelism, 2-level CALU uses a grid of T threads $T = T_r \times T_c$, where T_r is the number of threads on the vertical dimension, that is working on the panel and T_c is is the number

of threads on the horizontal dimension. Here we evaluate the performance of 3 variants: $T = 8 \times 1$, $T = 4 \times 2$, and $T = 2 \times 4$.

The three variants of our algorithm behave asymptotically in the same way and are all faster than ScaLAPACK routine. For $m \leq 10^4$, the variant $T = 2 \times 4$ is better than the others but for $m \geq 10^6$, the variant $T = 8 \times 1$ becomes better. The reason is that communication avoiding algorithms perform well on tall and skinny matrices. Hence when we increase the number of rows, the panel becomes very tall and skinny, and too large for the number of processors working on it. Then the algorithm spends more time factorizing the panel than updating the trailing matrix. In that case, increasing the number of processors working on the panel increases the efficiency of the panel factorization, but it also augments the number of tasks and the synchronization time. To ensure a certain trade-off, T_r can be chosen close to the optimal layout $\sqrt{\frac{mT}{n}}$ with respect to the memory constraints.

Figure 4.6 shows the performance of 2-level CALU relative to ScaLAPACK routine on tall and skinny matrices with varying row number. We observe that 2-level CALU is scalable and faster than ScaLAPACK. For a matrix of size $10^6 \times 1200$, 2-level CALU is 2 times faster than ScaLAPACK. Furthermore, an important loss of performance is observed for ScaLAPACK around $m = 10^4$, while all the variants of 2-level CALU lead to good performance. Hence ScaLAPACK routine does not optimize the latency at any level of the memory, while multilevel TSLU reduces significantly the amount of communication and memory transfers during the panel factorization.

We recall that the number of tasks and the synchronization time increase with the number of processors working on the panel. For a matrix that is relatively square ($m \approx n$), T_r can be chosen around $\sqrt{T}$, which is close to the optimal layout $\sqrt{\frac{mT}{n}}$. This change requires several adaptation to respect memory constraints. Thus b should be chosen such as the task, of size $\frac{m}{T_r} \times b$, fits in the thread memory.

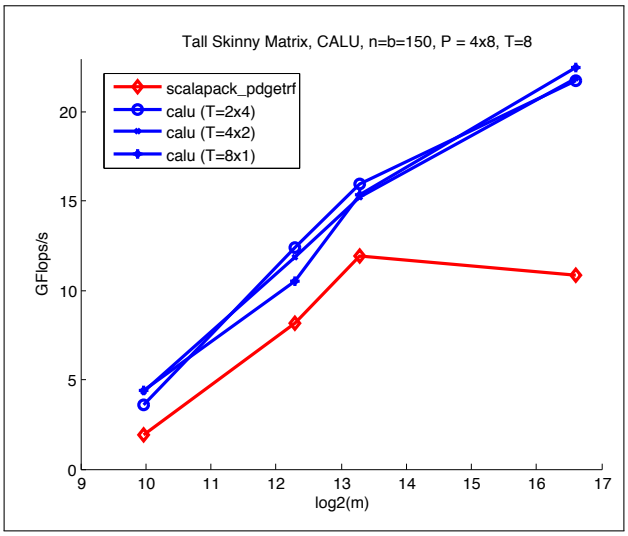


Figure 4.6: Performance of 2-level CALU and ScaLAPACK on a grid 4x8 nodes, for matrices with n=1200 , $b_2 = 150$, and m varying from 10^3 to 10^6. $b_1 = MIN(b_1, 100)$.

Figure 4.7 shows the performance of 2-level CALU on tall and skinny matrices with varying number

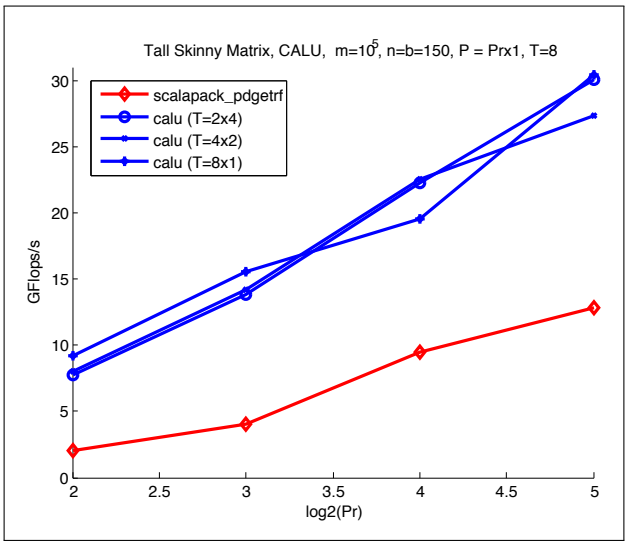


Figure 4.7: Performance of 2-level CALU and ScaLAPACK on a grid $P_r \times 1$ nodes, for matrices with $n = b_2 = 150$, $m = 10^5$, and $b_1 = MIN(b_1, 100)$.

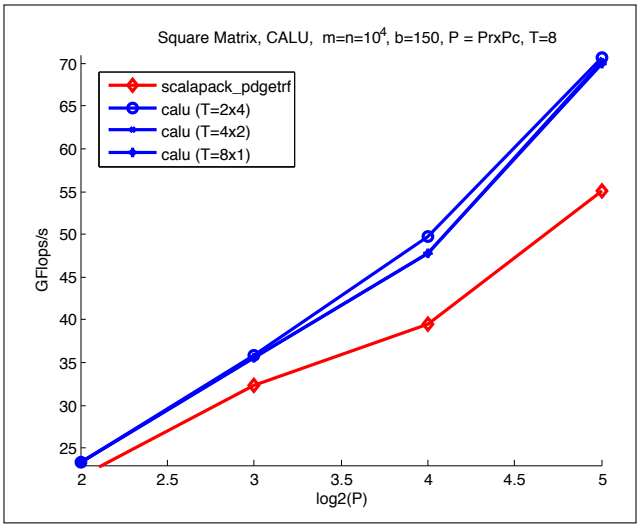


Figure 4.8: Performance of 2level-CALU and ScaLAPACK on a grid $P = P_r \times P_c$ nodes, for matrices with $m = n = 10^4$, $b_2 = 150$, and $b_1 = MIN(b_1, 100)$.

of processors working on the panel factorization. Since the matrix has only one panel, 2-level TSLU is called at the top level of the hierarchy and the adapted multhithreaded CALU is called at the deepest level of the hierarchy. We observe that for $P_r = 4$, 2-level CALU is 4.5 times faster than ScaLAPACK. Note that for $P_r > 4$, ScaLAPACK performance is bounded by 10 GFlops/s, while 2-level CALU shows an increasing performance up to 30 GFlops/s, that is 200% faster. For $P_r = 8$, the variant $T = 8 \times 1$ is slightly better than the others. This shows the importance of determining a good layout at runtime. This figure also shows the scalability of 2-level CALU when the number of processors increases.

Figure 4.8 shows the performance of 2-level CALU on a square matrix using varying processor number. For each value of P, we choose the optimal layout for ScaLAPACK and we use the same layout at the top level of 2-level CALU. The layout at the deepest level is determined by one of the three variants detailed above. We observe that all the variants of 2-level CALU are faster than ScaLAPACK. The variant $T = 2 \times 4$ is usually better than the others. This behavior has also been observed in multithreaded 1-level CALU. We recall that the matrix is partitioned into $T_r \times N/b$ blocks, increasing T_r increases the number of tasks and then the scheduling overhead which impact the performance of the entire algorithm. In that case, the panel will be factorized efficiently, but the entire algorithm will be slowed down by the large number of updates in the trailing matrix. Hence the scheduling overhead has a considerable impact on the performance of the algorithm on large square matrices. Therefore the 2-level CALU algorithm needs to choose the parameters that ensure a good trade-off between the fast panel factorization and the scheduling overhead.

In summary, when the matrix is tall and skinny, the variant $T = 8 \times 1$ achieves the best performance, but the variant $T = 2 \times 4$ becomes better when the number of columns increases. We note that increasing the number of threads T_r at the second level of the parallelism leads to better performance on tall and skinny matrices. Our experiments show robustness of 2-level CALU especially for tall and skinny matrices.

4.6 Conclusion

In this chapter we have introduced two hierarchical algorithms for the LU factorization, recursive CALU (1D multilevel CALU) and multilevel CALU (2D multilevel CALU). These two methods are based on communication avoiding LU algorithm and are adapted for a computer system with multiple levels of parallelism. Recursive CALU is equivalent to CALU with tournament pivoting in terms of numerical stability. However it does not reduce the latency cost at each level of the hierarchical machine. Hence for the panel factorization at the deepest level of recursion, Rec-CALU uses a hierarchical reduction tree, that combines several levels of reduction: one for each level of parallelism. Multilevel CALU minimizes communication at each level of the hierarchy of parallelism in terms of both volume of data and number of messages exchanged during the decomposition. We have presented experimental results for both 2-level and 3-level CALU, showing that multilevel CALU is stable in practice up to three levels of parallelism.

On a multi-core cluster system, that is a system with two levels of parallelism (distributed memory at the top level and shared memory at the deepest level), our experiment set shows that multilevel CALU outperforms ScaLAPACK, especially for tall and skinny matrices.

The performance model we have used to analyze the cost of 2-level CALU and 2-recursive CALU is a simple model which is not adapted for a hierarchical structure of architecture including both memory hierarchy and parallelism hierarchy. In order to have a more realistic estimation of the running time of multilevel algorithms over the different levels of parallelism. we introduce in section 5.2 the HCP model, a performance model which takes into account the complex structure of the architecture.

Future work includes several promising directions. From a theoretical perspective, we could further study the numerical stability of multilevel CALU or design a hybrid algorithm that takes advantages of both the numerical stability of recursive CALU and the efficiency of multilevel CALU. From a more practical perspective, we could implement a more flexible and modular multilevel CALU algorithm for clusters of multi-core processors, where it would be possible to choose the reduction tree at each level of parallelism in order to be cache efficient at the core level and communication avoiding at the distributed level. Note that such an algorithm was implemented for the QR factorization [46].

Chapter 5

On the performance of multilevel CALU

5.1 Introduction

There has been a considerable research work aiming at developing programming models, which are efficient across machine architectures and are easy to use. Similarly, designing architectures that achieve optimal performance for a class of applications is still a challenging task. Thus the design of accurate performance models is of primary importance for both algorithm and system design. We have discussed several performance models in section 1.6. We have mainly enumerated the advantages and the drawbacks of each model, in particular the state-of-the-art hierarchical performance models. The design of these models aims at understanding costs, bottlenecks, and unpredictable communication performance of parallel algorithms running on systems with multiple levels of parallelism. Motivated by the different issues detailed above and for the sake of a realistic performance model that is able to capture the most important features of a hierarchical architecture, we develop our hierarchical performance model, the HCP model, that we believe is suitable for hierarchical platforms. We note that our model takes into consideration both the network hierarchy and the compute node hierarchy. It is also based on a quite simple routing technique, which facilitates the analysis of the communication cost of parallel algorithms. Thus, despite its few drawbacks that will be discussed in this chapter, the HCP model presents a trade-off between accuracy and simplicity regarding the existing hierarchical performance models.

In particular, we use the HCP model to derive the performance model of multilevel Cannon algorithm, a hierarchical algorithm for matrix multiplication based on recursive calls to Cannon's algorithm. We introduce such an algorithm to be able to model the matrix-matrix operations performed during the multilevel CALU algorithm presented in the previous chapter 4. We show through the cost analysis of multilevel Cannon with respect to the HCP model that, this algorithm is asymptotically optimal in terms of both latency cost and bandwidth cost, modulo a factor that depends on the number of levels in the hierarchy and the number of processors at each level. Moreover, multilevel Cannon performs the same amount of computation as the classical Cannon's algorithm. We then estimate the running time of multilevel CALU with respect to the HCP model. We show that this algorithm attains asymptotically the lower bound on communication at each level of parallelism. However, since it is based on recursive calls, multilevel CALU performs extra computation. Thus we discuss the panel size aiming at minimizing the communication cost, while the number of floating-point operations increases as a function of the number of levels in the hierrachy. Furthermore, our performance prediction on an exascale platform show that with respect to appropriate panel sizes, multilevel CALU leads to a significant improvement over CALU, especially for three levels of parallelism.

The remainder of this chapter is organized as follows. We start with detailing the key principles underlying the design of the HCP model in section 5.2. The design of the HCP model is one of the core contributions of this chapter. Then we introduce in section 5.3 the multilevel Cannon algorithm and we evaluate its cost in terms of both computation and communication under the HCP model. Section 5.4 describes the communication details of multilevel CALU and estimates its running time under the HCP model. We show that both multilevel Cannon and multilevel CALU are asymptotically optimal regarding the lower bounds derived with respect to the HCP model. After that, section 5.5 presents performance predictions comparing multilevel CALU to CALU on an exascale platform based on Hopper supercomputer as a building block. In section 5.6 we present some related work. Section 5.7 presents concluding remarks and some perspectives.

Note that the design of the HCP model was performed as a collaboration with Mathias Jacquelin, who focuses on a multilevel QR factorization.

5.2 Toward a realistic Hierarchical Cluster Platform model (HCP)

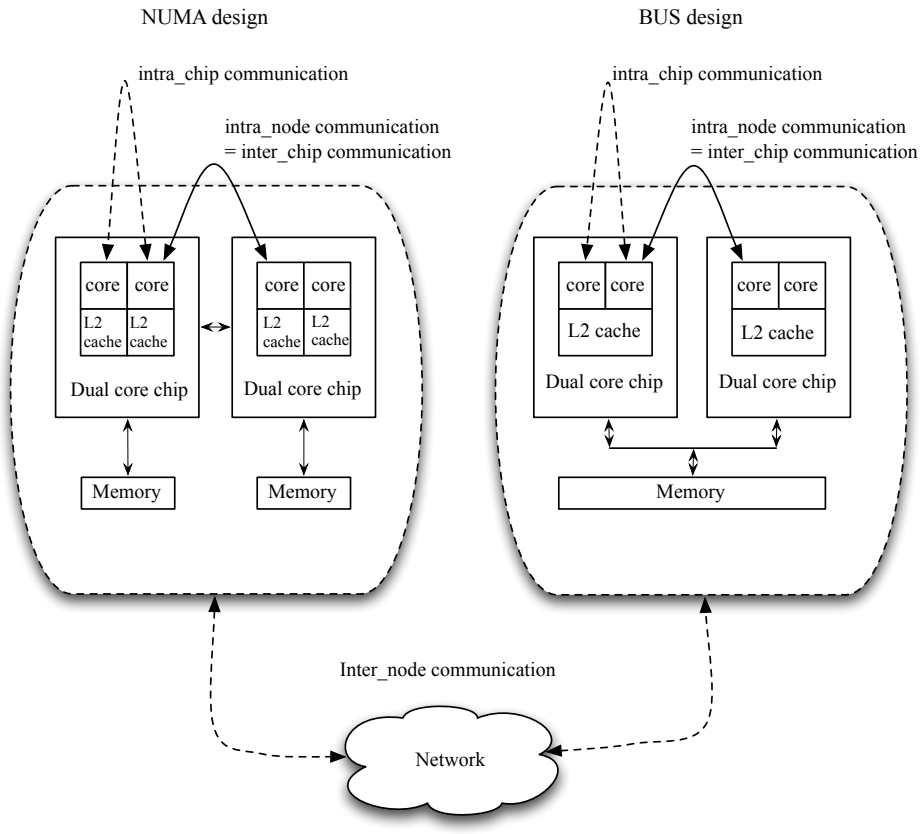


Figure 5.1: Cluster of multicore processors.

We target here 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 [29]. This compute

hierarchy can be composed for instance of a shared memory NUMA multicore platform. Figure 5.1 illustrates a simple example of such a platform. It also presents the different possible communication, including intra-node communication and inter-node communication.

We model such platforms with l levels of parallelism, and use the following notations:

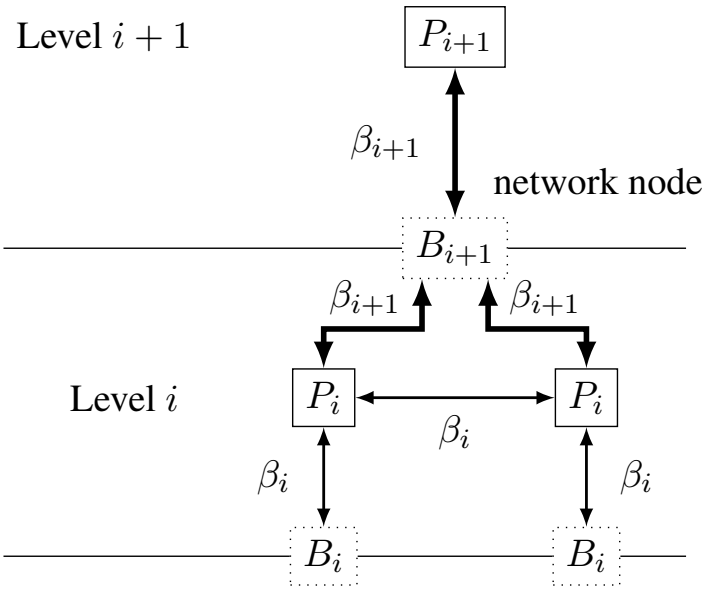


Figure 5.2: The components of a level i of the HCP model.

– l is the total number of levels in the hierarchy, and also denotes the index of the highest level in the hierarchy. Thus there are P_l nodes of level l.

– Level 1 is the deepest level in the hierarchy, actual processing elements are on this level. Compute units at this level present either a distributed memory, a shared memory, or a NUMA architecture. At each node of level 2, there are P_1 compute units.

– Let $P_i = P_{r_i} \times P_{c_i}$ be the number of computational nodes of level i inside a node of level $i+1$. The total number of compute units of the entire platform is given by the relation $P = \prod_{i=1}^{l} P_i$. These P compute units are organized as a two-dimensional grid, $P = P_r \times P_c$. Such a 2D grid topology is used at each level in the hierarchy.

– We let B_i be the network buffer size at a given level i. This buffer size possibly allows message aggregation, as we will discuss later. Hence we assume network nodes that provide support for internal buffering.

– We let M_i be the memory size of a computational node of level $i > 1$. We assume that $M_i = M_1 \prod_{j=1}^{i-1} P_j$, where M_1 is the memory size of a compute unit of level 1. A detailed relation between M_i and B_i will be explained thereafter.

– α_i is the network latency at level i while β_i is the inverse of the bandwidth.

– S_i is the number of messages sent by a node of level i at a given step of an algorithm.

– W_i is the volume of data (the number of words) that is sent by a node of level i at a given step of an algorithm.

– $\bar{S}_i$ is the cost due to the communication of S_i messages at level i, $\bar{S}_i = S_i \times \alpha_i$. We refer to this cost as the latency cost.

– Similarly, $\bar{W}_i$ is the cost due to the communication of a volume of data W_i at level i, $\bar{W}_i = W_i \times \beta_i$. We refer to this cost as the bandwidth cost.

Note that we will use these notations throughout the rest of the chapter.

An intermediate level $i > 1$ is depicted on Figure 5.2, where we present the bandwidth of the links connecting computational nodes and network nodes.

Communicating under the HCP model.

We now describe how communication happens in the HCP model, and how messages are routed between the different levels of the network hierarchy. In our model, we assume that if a computational node of level i communicates, all of the nodes below it are involved, leading to a total of $\prod_{j=1}^{i-1} P_j$ compute units of level 1 participating in that communication. Remember that W_i is the amount of data that is sent by a node of level i. We therefore have the following relation:

$$W_i = \frac{W_{i+1}}{P_i}.$$

As an example, we now detail a communication taking place between two nodes of a given level i. For this specific communication, $\prod_{j=1}^{i-1} P_j$ compute units of level 1 are involved in sending a global volume of data W_i. Each of these compute units has to send a chunk of data $W_1 = W_i/(\prod_{j=1}^{i-1} P_j)$. Since this amount of data has to fit in the memory of a compute unit of level 1, we obviously have $\forall i, M_1 \geq W_1 = W_i/(\prod_{j=1}^{i-1} P_j)$. Here for the sake of simplicity, and to be able to derive general lower bounds valid for each level of the hierarchy, we assume the network buffer size at level 1 to be equal to the memory size of a compute unit of level 1, $B_1 = M_1$, which means that the first level of the network is fully-pipelined. Therefore, data is always sent using only one message at level 1, by a compute unit at this level. The corresponding bandwidth cost is $W_1 \times \beta_1$.

These blocks are then transmitted to the level above in the hierarchy, i.e. to level 2. A computational 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 can be performed in (W_2/B_2) messages.

The same holds for any level k with $1 < k \leq i$, where data is forwarded by sending(W_k/B_k) messages, therefore we have the following counts for the number of words and the number of messages,

$$W_k = \frac{W_i}{\prod_{j=k}^{i-1} P_j},$$

$$S_k = \frac{W_k}{B_k} = \frac{W_i}{B_k \prod_{j=k}^{i-1} P_j}.$$

The corresponding costs are,

$$\bar{W}_k = \frac{W_i}{\prod_{j=k}^{i-1} P_j}\beta_k,$$
$$\bar{S}_k = \frac{W_k}{B_k} = \frac{W_i}{B_k \prod_{j=k}^{i-1} P_j}\alpha_k.$$

Network types.

We assume three kinds of networks, depending on their buffer size.

1. A fully-pipelined network, which is able to aggregate all the messages coming from the deeper network levels in the hierarchy into a single message. This means that $B_i \geq M_i$, since the largest volume of data a node can send is of size M_i.
2. A bufferized network, which is able to aggregate messages up to its buffer size. For such a network $B_i < M_i$. This is the most common case.
3. A forward network, which just forwards the messages coming from below to the level above it. This means that starting from a certain network level, the buffer size remains a constant.

The two particular kinds of network, fully-pipelined and forward, enforce constraint on network buffer sizes:

– For a given level i, the forward network requires that $B_i = B_{i-1}$. In the case when all the P_{i-1} compute units from the level $i-1$ send S_{i-1} messages each, the number of messages send by one compute unit of level i is $S_i = P_{i-1}S_{i-1}$.

– The fully-pipelined network case is ensured whenever $B_i \geq P_{i-1}W_{i-1}$, where W_{i-1} denotes the volume of data sent by each single node of level $i-1$ below. We thus consider that $B_i = M_i$, since M_i is the size of the largest message which could be sent by a node at level i using full pipelining. We also assume that all levels below a fully-pipelined level are themselves fully pipelined. Therefore, the constraint on the buffer size for a network of a given level i becomes $B_i = M_i = P_{i-1}M_{i-1} = P_{i-1}B_{i-1}$.

Based on these two extreme cases, we can safely assume that the buffer size B_i of a given level i satsfies,

$$B_{i-1} \leq B_i \leq P_{i-1}B_{i-1}. \tag{5.1}$$

Moreover, on matrix product-like problems, assuming that one copy of the input matrix is stored in memory, the largest block exchanged at the topmost level l generally has a size of (n^2/P_l). Therefore, the largest block exchanged at a given level i has a size of $W_i \leq (n^2/\prod_{j=i}^{l} P_j)$.

Note that for our performance predictions we consider either a fully pipelined network at each level of the hierarchy, that is at each level $B_i = M_i$, or a fully forward network at each level of the hierarchy, that is at each level the buffer size satisfies $B_i = M_1$.

Lower bounds on communication

Lower bounds on communication have been generalized in [14] for direct methods of linear algebra algorithms which can be expressed as three nested loops. We refine these lower bounds under our hierarchical model. We assume that for a given node of level i, the memory size $M_i = \Omega\left(\frac{n^2}{\prod_{j=i}^{l} P_j}\right)$. Therefore, the lower bound on bandwidth at each level i becomes,

$$W_i \geq \Omega\left(\frac{n^3}{\prod_{j=i}^{l} P_j\sqrt{M_i}}\right),$$
$$\geq \Omega\left(\frac{n^2}{\prod_{j=i}^{l} \sqrt{P_j}}\right).$$

The lower bound on latency depends on the network capacity (buffer size) of the current level i. We have W_i words to send in messages of size B_i. Therefore the lower bound on latency at each level i becomes,

$$S_i \geq \Omega\left(\frac{n^3}{B_i \prod_{j=i}^{l} P_j\sqrt{M_i}}\right),$$
$$\geq \Omega\left(\frac{n^2}{B_i \prod_{j=i}^{l} \sqrt{P_j}}\right).$$

Note that although our performance model (HCP) is more realistic than simpler performance models, since it takes into account both the network hierarchy and the compute node hierarchy, it still suffers from a few drawbacks. We enumerate some of them here. First, the HCP model does not include the routing overhead at each network node. This could be important especially when modeling the store-and-forward case. Moreover, this particular routing technique ensures a high level of error-free network traffic. However, the associated checking process could also result in an important delay. Additionally, in the HCP model we assume that a communication between two compute units of level 1, belonging to two different nodes of level k, should be routed up to the network level k, then down to the network level 1. Thus we consider the hierarchical aspect of the network. However, our model does not take into consideration the network topology at each level of the network hierarchy, and thus its connectivity. Therefore it does not minimize the routing paths with respect to the underlying network topology at each level of the hierarchy. Furthermore, for the sake of simplicity, and because it is difficult to estimate the congestion degree of a given link in the network, our model does not handle the congestions that may occur in parts of the network hierarchy. Hence we not consider adaptive routing schemes.

To sum things up, despite the few drawbacks enumerated above and which might slightly impact the accuracy of our performance predictions, we believe that the HCP model presents a reasonable trade-off between simplicity and reality.

5.3 Cost analysis of multilevel Cannon algorithm under the HCP model

To perform matrix multiplication $C = C + A \cdot B$, we can use Cannon's algorithm. It is a parallel algorithm that uses a square processor grid $\sqrt{P} \times \sqrt{P}$. $\sqrt{P}$ shifts of both A and B are achieved along the two dimensions of the grid of processors. For a matrix of size $n \times n$, each processor sends a volume of data of order $O(n^2/\sqrt{P})$ using $O(\sqrt{P})$ messages along the two dimensions of the grid. It is known that (see for example [14]) 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 processor are asymptotically optimal. This analysis is based on a simple performance model, which assumes in the parallel case P processors without memory hierarchy. In the following we evaluate the cost of recursive Cannon algorithm over a hierarchy of nodes. We refer to this algorithm as ML-CANNON, and we consider l levels of parallelism. We restrict our discussion to square matrices and to a hierarchy of parallelism where P_k can be written as $P_k = \sqrt{P_k} \times \sqrt{P_k}$ at each level k. Algorithm 15 presents the communication details of multilevel Cannon.

We design such an algorithm to estimate the cost of multilevel CALU (ML-CALU) introduced in the previous chapter. Hence the multilevel dgemm and the multilevel dtrsm presented in ML-CALU algorithm 12 and in ML-TSLU algorithm 13 will be evaluated based on the cost analysis of multilevel Cannon below.

Algorithm 15: $ML - CANNON(C, A, B, P_k, k)$

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

We analyze both the communication cost, and the computation cost of ML-CANNON algorithm described in Algorithm 15 with respect to the HCP model. All the costs presented below are computed on the critical path.

We first define the following costs:

- for Cannon's algorithm applied to two square matrices of size $n \times n$ using a square grid of P_1 processors, we note:
  - $\bar{W}_{\text{CANNON}}(n, n, P_1)$ the cost due to the communication of a volume of data $W_{\text{CANNON}}(n, n, P_1)$ on the critical path,
  - $\bar{S}_{\text{CANNON}}(n, n, P_1)$ the cost of the communication of $S_{\text{CANNON}}(n, n, P_1)$ messages on the critical path,
  - $F_{\text{CANNON}}(n, n, P_1)$ the number of floating-point operations performed on the critical path.

- for multilevel Cannon's algorithm applied to two square matrices of size $n \times n$ using a square grid of P_l processors at level l of the recursion, we note:
  - $\bar{W}_{\text{ML-CANNON}}(n, n, P_l, l)$ the cost of the communication of a volume of data $W_{\text{ML-CANNON}}(n, n, P_l, l)$ on the critical path,
  - $\bar{S}_{\text{ML-CANNON}}(n, n, P_l, l)$ the cost of the communication of $S_{\text{ML-CANNON}}(n, n, P_l, l)$ messages on the critical path,
  - $F_{\text{ML-CANNON}}(n, n, P_l, l)$ the number of floating-point operations performed on the critical path.

Communication cost

The volume of data which needs to be sent by each node at the top level l is $\frac{4n^2}{\sqrt{P_l}}$, since the number of shifts at this level is $4\sqrt{P_l}$. With respect to the HCP model, this volume of data is sent by processors of level 1. Therefore,

$$\bar{W}_{\text{ML-CANNON}}(n, n, P_l, l) = \begin{cases} \sum_{k=1}^{l} \frac{4n^2\sqrt{P_l}}{\prod_{j=k}^{l} P_j}\beta_k + \sqrt{P_l}\bar{W}_{\text{ML-CANNON}}\left(\frac{n}{\sqrt{P_l}}, \frac{n}{\sqrt{P_l}}, P_{l-1}, l-1\right) & \text{if } l > 1 \\ \bar{W}_{\text{CANNON}}(n, n, P_1) & \text{if } l = 1 \end{cases}$$

Then going down in the recursion, we obtain:

$$\bar{W}_{\text{ML-CANNON}}(n, n, P_l, l) = \sum_{k=1}^{l} \left(\frac{4n^2}{\prod_{j=k}^{l} P_j} \sum_{i=k}^{l} \left(\prod_{j=i}^{l} \sqrt{P_j} \right) \beta_k \right).$$

We can upper bound this cost as in the following, where we assume that

$$\begin{aligned} \sum_{i=k}^{l} \left(\prod_{j=i}^{l} \sqrt{P_j} \right) &= \left(1 + \frac{1}{\sqrt{P_k}} + \frac{1}{\sqrt{P_k}\sqrt{P_{k+1}}} + \cdots + \frac{1}{\prod_{j=i}^{l-1}\sqrt{P_j}} \right) \prod_{j=k}^{l} \sqrt{P_j} \\ &\leq \left(1 + \frac{l-k}{\sqrt{P_k}} \right) \prod_{j=k}^{l} \sqrt{P_j}. \end{aligned}$$

Therefore,

$$\bar{W}_{\text{ML-CANNON}}(n, n, P_l, l) \leq \begin{cases} \sum_{k=1}^{l} \left[4\left(1 + \frac{l-k}{\sqrt{P_k}} \right) \frac{n^2}{\prod_{j=k}^{l}\sqrt{P_j}} \beta_k \right] & \text{if } l > 1 \\ \frac{4n^2}{\sqrt{P_1}}\beta_1 & \text{if } l = 1 \end{cases}$$

At a given level k of the hierarchy, the number of messages to send depends on the network capacity. Thus,

$$\bar{S}_{\text{ML-CANNON}}(n,n,P_l,l) \leq \begin{cases} \sum_{k=1}^{l}\left[4\left(1+\frac{l-k}{\sqrt{P_k}}\right)\frac{n^2}{B_k\prod_{j=k}^{l}\sqrt{P_j}}\alpha_k\right] & \text{if } l>1 \\ 4\sqrt{P_1}\alpha_1 & \text{if } l=1 \end{cases}$$

Computation cost

To evaluate the number of floating point operations performed by multilevel Cannon, we consider the following recursion:

$$\begin{aligned} F_{\text{ML-CANNON}}(n,n,P_l,l) &= \sqrt{P_l}F_{\text{ML-CANNON}}\left(\tfrac{n}{\sqrt{P_l}},\tfrac{n}{\sqrt{P_l}},P_{l-1},l-1\right), \\ &= \prod_{j=1}^{l}\sqrt{P_j}F_{\text{CANNON}}\left(\tfrac{n}{\prod_{j=1}^{l}\sqrt{P_j}},\tfrac{n}{\prod_{j=1}^{l}\sqrt{P_j}},P_1\right), \\ &= \tfrac{2n^3}{\prod_{j=1}^{l}P_j} = \tfrac{2n^3}{P}. \end{aligned}$$

Finally the total cost of multilevel cannon's algorithm is bounded as follows,

$$T_{\text{ML-CANNON}}(n,n,P_l,l) \leq \begin{cases} \frac{2n^3}{P}\gamma+\sum_{k=1}^{l}\left[4\left(1+\frac{l-k}{\sqrt{P_k}}\right)\frac{n^2}{\prod_{j=k}^{l}\sqrt{P_j}}\right](\beta_k+\frac{\alpha_k}{B_k}) & \text{if } l>1 \\ \frac{2n^3}{P_1}\gamma+\frac{4n^2}{\sqrt{P_1}}\beta_1+4\sqrt{P_1}\alpha_1 & \text{if } l=1 \end{cases}$$

We can then conclude that multilevel Cannon algorithm is asymptotically optimal at each level of parallelism, in terms of both latency and bandwidth, regarding the lower bounds of the HCP model.

5.4 Cost analysis of multilevel CALU under the HCP model

Here we estimate the running time of the multilevel CALU algorithm, presented in Algorithm 12 in chapter 4, with respect to the HCP model. We show that this algorithm asymptotically attains the communication lower bounds detailed in section 5.2 at each level of the recursion.

The ML-CALU algorithm, given in Algorithm 16, details the communication performed during the factorization. We recall here that multilevel CALU proceeds as follows: (1) it first recursively factors the panel with ML-CALU with a block size corresponding to the next level in the hierarchy; (2) the selected sets of candidate pivot rows are merged two-by-two along the reduction tree, where the reduction operator is ML-CALU; (3) 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; (4) then the computed permutation is applied to the input matrix, the block column of L and the block row of U are computed; (5) finally, after the 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 trailing matrix is updated. Note that at the deepest level of recursion, hence in the hierarchy, ML-CALU calls the communication optimal CALU algorithm.

Algorithm 16: ML-CALU(A, m, n, l, P_l)

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 */

Let P be the total number of processors in the hierarchy. Let $P_l = P_{r_l} \times P_{c_l}$ be the number of processors at the l-th level of the processor hierarchy and b_l the block size at this level. The total number of processors across the different levels of the hierarchy is $\prod_{k=1}^{l} P_k = P$. Note that forsimplicity Algorithm 16 assumes that m and n are respectively multiples of P_{r_l} and P_{c_l}.

To estimate the cost of ML-CALU, we first define the following costs, where $\bar{W}$ refers to the cost of communicating a volume of data W, $\bar{S}$ to the cost of sending S messages, and F to the number of floating-point operations. We note that all of these costs are evaluated on the critical path.

- $T_{\text{CALU}}(m, n, b, P)$ is the cost of factoring a matrix of size $m \times n$ with 1-level CALU using P processors and a block size b.

$$\begin{aligned} T_{\text{CALU}}(m,n,b,P) &= \bar{W}_{\text{CALU}}(m,n,b,P) + \bar{S}_{\text{CALU}}(m,n,b,P) + F_{\text{CALU}}(m,n,b,P)\,\gamma, \\ &= W_{\text{CALU}}(m,n,b,P)\,\beta_1 + S_{\text{CALU}}(m,n,b,P)\,\alpha_1 + F_{\text{CALU}}(m,n,b,P)\,\gamma. \end{aligned}$$

- $T_{\text{ML-CALU}}(m, n, b_l, P_l, l)$ is the cost of factoring a matrix of size $m \times n$ with multilevel CALU using P_l processors and a block size b_l at the top level of recursion. We note k a given level of the hierarchy.

$$\begin{aligned} T_{\text{ML-CALU}}(m,n,b_l,P_l,l) &= \bar{W}_{\text{ML-CALU}}(m,n,b_l,P_l,l) + \bar{S}_{\text{ML-CALU}}(m,n,b_l,P_l,l) \\ &+ F_{\text{ML-CALU}}(m,n,b_l,P_l,l)\,\gamma, \\ &= \sum_{k=1}^{l} \Big(W_{\text{ML-CALU}}(m,n,b_l,P_l,k)\,\beta_k + S_{\text{ML-CALU}}(m,n,b_l,P_l,k)\,\alpha_k \Big) \\ &+ F_{\text{ML-CALU}}(m,n,b_l,P_l,l)\,\gamma. \end{aligned}$$

- $T_{\text{ML-LU-Fact}}(m, n, b, P, l)$ includes the broadcast of the pivot information along the rows of the process grid, the application of the pivot information to the remainder of the rows, the broadcast of the $b_l \times b_l$ upper part of the current panel of L along row of processes owning the corresponding block of U, and finally the computation of the block column of L and the block row of U. We refer to this step as the panel factorization. It is performed once the pivot rows are selected.

$$\begin{aligned} T_{\text{ML-LU-Fact}}(m,n,b_l,P_l,l) &= \bar{W}_{\text{ML-LU-Fact}}(m,n,b_l,P_l,l) + \bar{S}_{\text{ML-LU-Fact}}(m,n,b_l,P_l,l) \\ &+ F_{\text{ML-LU-Fact}}(m,n,b_l,P_l,l)\,\gamma, \\ &= \sum_{k=1}^{l} \Big(W_{\text{ML-LU-Fact}}(m,n,b_l,P_l,k)\,\beta_k + S_{\text{ML-LU-Fact}}(m,n,b_l,P_l,k)\,\alpha_k \Big) \\ &+ F_{\text{ML-LU-Fact}}(m,n,b_l,P_l,l)\,\gamma. \end{aligned}$$

- $T_{\text{ML-LU-Up}}(m, n, b_l, P_l, l)$ is the cost of the update of the trailing matrix. It includes the broadcast of the block column of L along rows of processors of the grid, the broadcast of the block row of U along columns of processors of the grid, and the rank-b update of the trailing matrix.

$$\begin{aligned} T_{\text{ML-LU-Up}}(m,n,b_l,P_l,l) &= \bar{W}_{\text{ML-LU-Up}}(m,n,b_l,P_l,l) + \bar{S}_{\text{ML-LU-Up}}(m,n,b_l,P_l,l) \\ &+ F_{\text{ML-LU-Up}}(m,n,b_l,P_l,l)\,\gamma, \\ &= \sum_{k=1}^{l} \Big(W_{\text{ML-LU-Up}}(m,n,b_l,P_l,l)\,k\beta_k + S_{\text{ML-LU-Up}}(m,n,b_l,P_l,l)\,k\alpha_k \Big) \\ &+ F_{\text{ML-LU-Up}}(m,n,b_l,P_l,l)\,\gamma. \end{aligned}$$

- In the formulas, we note $T_{\text{FactUp-ML-CALU}}(m, n, b_l, P_l, l)$ the cost of both the factorization of the current panel without pivoting and the update of the corresponding trailing matrix. That is,

$$\begin{aligned} T_{\text{FactUp-ML-CALU}}(m,n,b_l,P_l,l) &= \sum_{s=1}^{n/b_l} T_{\text{ML-LU-Fact}}\left(\tfrac{m-sb_l}{P_{r_l}}, \tfrac{n-sb_l}{P_{c_l}}, b_l, P_l, l\right) \\ &+ \sum_{s=1}^{n/b_l} T_{\text{ML-LU-Up}}\left(\tfrac{m-sb_l}{P_{r_l}}, \tfrac{n-sb_l}{P_{c_l}}, b_l, P_l, l\right). \end{aligned}$$

The recursive cost of multilevel CALU is given by the contributions detailed below. We consider l levels of parallelism, and we denote s the current step of ML-CALU. We first focus on the recursive preprocessing step, which includes the application of ML-CALU to the blocks located at the leaves of the top level reduction tree to select b_l candidate pivot rows from each block (Equation 5.2),

$$\sum_{s=1}^{n/b_l} T_{\text{ML-CALU}}\left(\frac{m-(s-1)b_l}{P_{r_l}}, b_l, b_{l-1}, P_{l-1}, l-1\right). \tag{5.2}$$

Additionaly, along the reduction tree, blocks of size $b_l \times b_l$ are merged two by two and again we perform ML-CALU on these blocks, each of size $2b_l \times b_l$ at level l, to select b_l candidate pivot rows (Equation 5.3),

$$\sum_{s=1}^{n/b_l} log(P_{r_l}) T_{\text{ML-CALU}}\left(2b_l, b_l, b_{l-1}, P_{l-1}, l-1\right). \tag{5.3}$$

To form $2b_l \times b_l$ blocks, a volume of data of size b_l^2 needs to be sent, $\log P_{r_l}$ times along the critical path. This communication is performed by processors of level 1 as presented in Figure 5.3, where we consider a hierarchy of 3 levels. The cost of this operation is (Equation 5.4),

$$logP_{r_l}\left(\sum_{k=1}^{l} \frac{b_l^2 P_l}{\prod_{j=k}^{l} P_j}(\beta_k + \frac{1}{B_k}\alpha_k)\right). \tag{5.4}$$

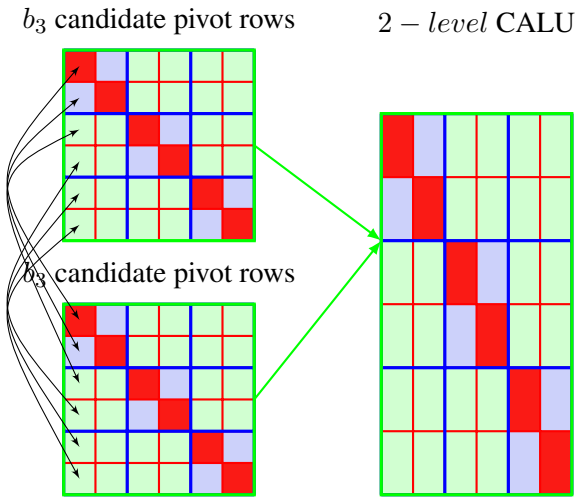


Figure 5.3: Communication between two nodes of level 3: merge of two blocks of b_3 candidate pivot rows

Therefore the cost of the preprocessing step is given by the following terms,

$$\sum_{s=1}^{n/b_l} T_{\text{ML-CALU}}\left(\tfrac{m-(s-1)b_l}{P_{r_l}}, b_l, b_{l-1}, P_{l-1}, l-1\right) \quad + \quad \sum_{s=1}^{n/b_l} log(P_{r_l}) T_{\text{ML-CALU}}\left(2b_l, b_l, b_{l-1}, P_{l-1}, l-1\right)$$
$$+\tfrac{n}{b_l} logP_{r_l}\left(\sum_{k=1}^{l} \tfrac{b_l^2 P_l}{\prod_{j=k}^{l} P_j}(\beta_k + \tfrac{1}{B_k}\alpha_k)\right).$$

Thus the following formula presents the general recursive cost of multilevel CALU, where s is the

current step of the algorithm and k is the level of parallelism in the hierarchy.

$$T_{\text{ML-CALU}}(m,n,b_l,P_l,l) = \begin{cases} \sum_{s=1}^{n/b_l} \Big[T_{\text{ML-CALU}}\left(\frac{m-(s-1)b_l}{P_{r_l}}, b_l, b_{l-1}, P_{l-1}, l-1\right) \\ \quad + \; logP_{r_l} T_{\text{ML-CALU}}(2b_l, b_l, b_{l-1}, P_{l-1}, l-1) \\ \quad + \; logP_{r_l} \sum_{k=1}^{l} \frac{b_l^2 P_l}{\prod_{j=k}^{l} P_j}(\beta_k + \frac{1}{B_k}\alpha_k) \Big] \\ \quad + \; T_{\text{FactUp-ML-CALU}}(m,n,b_l,P_l,l) & \text{if } l > 1 \\ T_{\text{CALU}}(m,n,b_1,P_1) & \text{if } l = 1 \end{cases}$$

If $l \geq 3$, going down in the recursion and knowing that $\frac{m-sb_l}{P_{r_l}} \leq \frac{m}{P_{r_l}}$, we can upper bound this cost as in the following,

$$T_{\text{ML-CALU}}(m,n,b_l,P_l,l) \leq \begin{cases} \frac{n}{b_{l-1}} T_{\text{ML-CALU}}\left(\frac{m}{P_{r_l}P_{r_{l-1}}}, b_{l-1}, b_{l-2}, P_{l-2}, l-2\right) \\ \quad + \; \frac{n}{b_{l-1}} logP_{r_l} T_{\text{ML-CALU}}\left(\frac{2b_l}{P_{r_{l-1}}}, b_{l-1}, b_{l-2}, P_{l-2}, l-2\right) \\ \quad + \; \frac{n}{b_{l-1}}(logP_{r_{l-1}} logP_{r_l} + logP_{r_{l-1}}) T_{\text{ML-CALU}}(2b_{l-1}, b_{l-1}, b_{l-2}, P_{l-2}, l-2) \\ \quad + \; \frac{n}{b_{l-1}}(logP_{r_{l-1}} logP_{r_l} + logP_{r_{l-1}}) \sum_{k=1}^{l-1} \frac{b_{l-1}^2 P_{l-1}}{\prod_{j=k}^{l-1} P_j}(\beta_k + \frac{\alpha_k}{B_k}) \\ \quad + \; \frac{n}{b_l} logP_{r_l} \sum_{k=1}^{l} \frac{b_l^2 P_l}{\prod_{j=k}^{l} P_j}(\beta_k + \frac{1}{B_k}\alpha_k) & \text{if } l > 1 \\ \quad + \; \frac{n}{b_l} T_{\text{FactUp-ML-CALU}}\left(\frac{m}{P_{r_l}}, b_l, b_{l-1}, P_{l-1}, l-1\right) \\ \quad + \; \frac{n}{b_l} logP_{r_l} T_{\text{FactUp-ML-CALU}}(2b_l, b_l, b_{l-1}, P_{l-1}, l-1) \\ \quad + \; T_{\text{FactUp-ML-CALU}}(m,n,b_l,P_l,l) \\ T_{\text{CALU}}(m,n,b_1,P_1) & \text{if } l = 1 \end{cases}$$

Here we assume that $(logP_{r_{l-1}} logP_{r_l} + logP_{r_{l-1}}) \leq 2logP_{r_{l-1}} logP_{r_l}$. We make the same assumption at each level of recursion. Thus, going down in the recursion we get the cost of multilevel CALU for l levels of recursion ($l \geq 2$), where r denotes the level of recursion and k the level of parallelism in the hierarchy. Hence each recursion level r invokes communication over all levels of parallelism k, $1 \leq k \leq r$. Note that the recursive call r is performed at level r of parallelism.

$$\begin{aligned} T_{\text{ML-CALU}}(m,n,b_l,P_l,l) \;\lesssim\; & \frac{n}{b_2} T_{\text{CALU}}\left(\frac{m}{\prod_{j=2}^{l} P_{r_j}}, b_2, b_1, P_1\right) \quad (1) \\ & + \; \frac{n}{b_2} \sum_{r=3}^{l} 2^{l-r} \prod_{j=r}^{l} logP_{r_j} T_{\text{CALU}}\left(\frac{2b_r}{\prod_{j=2}^{r-1} P_{r_j}}, b_2, b_1, P_1\right) \quad (1) \\ & + \; \frac{n}{b_2} 2^{l-2} \prod_{j=2}^{l} logP_{r_j} T_{\text{CALU}}(2b_2, b_2, b_1, P_1) \quad (1) \\ & + \; \sum_{r=3}^{l} \frac{n}{b_r} T_{\text{FactUp-ML-CALU}}\left(\frac{m}{\prod_{j=r}^{l} P_{r_j}}, b_r, b_{r-1}, P_{r-1}, r-1\right) \quad (2) \\ & + \; \sum_{k=4}^{l} \sum_{r=3}^{k-1} \frac{n}{b_r} 2^{l-k} \prod_{j=k}^{l} logP_{r_j} T_{\text{FactUp-ML-CALU}}\left(\frac{2b_k}{\prod_{j=r}^{k-1} P_{r_j}}, b_r, b_{r-1}, P_{r-1}, r-1\right) \quad (2) \\ & + \; \sum_{r=3}^{l} \frac{n}{b_r} 2^{l-r} \prod_{j=r}^{l} logP_{r_j} T_{\text{FactUp-ML-CALU}}(2b_r, b_r, b_{r-1}, P_r, r-1) \quad (2) \\ & + \; \sum_{r=2}^{l} \frac{n}{b_r} \prod_{j=r}^{l} logP_{r_j} \sum_{k=1}^{r} \frac{b_r^2 P_r}{\prod_{j=k}^{r} P_j}(\beta_k + \frac{\alpha_k}{B_k}) \quad (3) \\ & + \; T_{\text{FactUp-ML-CALU}}(m,n,b_l,P_l,l) \quad (4) \end{aligned}$$

In the previous equation, the three first terms (1) correspond to the cost of applying 1-level CALU at the deepest level of the recursion. This includes leaves of size $(m/\prod_{j=2}^{l} P_{r_j}) \times b_2$, intermediate blocks of size $(2b_r/\prod_{j=2}^{r-1} P_{r_j}) \times b_2$, and blocks of size $2b_2 \times b_2$. The three following terms (2) present the cost of the panel factorizations and the updates performed through all the recursion levels $l > 2$, during the preprocessing step. The term (3) corresponds to the cost of merging blocks of pivot candidate rows along all the reduction trees and through all the recursion levels $r \geq 2$. Thus the first 7 terms of the previous equation correspond to the preprocessing step of ML-CALU. Finally the last term, (4), corresponds to the cost of the panel factorization and the update of the principal trailing matrix throughout all the steps of the algorithm.

Now we estimate the cost of $T_{\text{FactUp-ML-CALU}}(m, n, b_l, P_l, l)$. The detailed counts and approximations are presented in Appendix B .5. We first consider the panel factorization then the update of the trailing matrix. Then we give an estimation of the total running time of multilevel CALU. The counts here are given for a square matrix of size $n \times n$ using a square grid of processors at each level r of recursion, that for each level r, $P_{r_r} = P_{c_r} = \sqrt{P_r}$. We discuss later the panel size b_r.

The panel factorization

The arithmetic cost of the panel factorization corresponds to the computation of a block column of L of size $(n - sb_l) \times b_l$ and a block row of U of size $b_l \times (n - sb_l)$. To evaluate this cost we use the multilevel Cannon algorithm detailed in section 5.3. The communication cost includes the broadcast of the b_l pivots selected during the preprocessing step, the permutation of the remainder of the rows (this communication is performed using an all to all reduce operation), the broadcast of the $b_l \times b_l$ upper part of L, and finally the communication cost due to the computation of the block column of L and the block row of U using ML-CANNON. We obtain the following costs on the critical path (low order terms are omitted),

– Arithmetic cost

$$\sum_{s=1}^{n/b_l} F_{\text{ML-LU-Fact}}\left(\frac{n-sb_l}{\sqrt{P_l}}, \frac{n-sb_l}{\sqrt{P_l}}, b_l, P_l, l\right)\gamma = \frac{2n^2 b_l}{\sqrt{P_l}\prod_{j=1}^{l-1} P_j}\gamma.$$

– Communication cost

$$\sum_{s=1}^{n/b_l} \bar{W}_{\text{ML-LU-Fact}}\left(\frac{n-sb_l}{\sqrt{P_l}}, \frac{n-sb_l}{\sqrt{P_l}}, b_l, P_l, l\right) + \bar{S}_{\text{ML-LU-Fact}}\left(\frac{n-sb_l}{\sqrt{P_l}}, \frac{n-sb_l}{\sqrt{P_l}}, b_l, P_l, l\right) =$$

$$\sum_{s=1}^{n/b_l}\sum_{k=1}^{l} W_{\text{ML-LU-Fact}}\left(\frac{n-sb_l}{\sqrt{P_l}}, \frac{n-sb_l}{\sqrt{P_l}}, b_l, P_l, k\right)\beta_k + S_{\text{ML-LU-Fact}}\left(\frac{n-sb_l}{\sqrt{P_l}}, \frac{n-sb_l}{\sqrt{P_l}}, b_l, P_l, k\right)\alpha_k \leq$$

$$\sum_{k=1}^{l}\left(\frac{nb_lP_l + n^2\sqrt{P_l}}{\prod_{j=k}^{l} P_j}\log\sqrt{P_l} + \frac{4n^2\left(1+\frac{l-k}{\sqrt{P_k}}\right)}{b_l\sqrt{P_l}\prod_{j=k}^{l}\sqrt{P_j}}\right)(\beta_k + \frac{\alpha_k}{B_k}).$$

The update of the trailing matrix

Here again to evaluate the arithmetic cost corresponding to the trailing matrix update we utilize ML-CANNON. The communication cost includes the broadcast of a block column of L, the broadcast of a block row of U, and finally the communication cost due to the matrix multiplication using multilevel Cannon. Regarding the critical path, we obtain the following costs,

– Arithmetic cost

$$\sum_{s=1}^{n/b_l} F_{\text{ML-LU-Up}}\left(\frac{n-sb_l}{\sqrt{P_l}}, \frac{n-sb_l}{\sqrt{P_l}}, b_l, P_l, l\right)\gamma \approx \frac{2}{3}\frac{n^3}{P}\gamma.$$

– Communication cost

$$\sum_{s=1}^{n/b_l} \bar{W}_{\text{ML-LU-Up}}\left(\frac{n-sb_l}{\sqrt{P_l}}, \frac{n-sb_l}{\sqrt{P_l}}, b_l, P_l, l\right) + \bar{S}_{\text{ML-LU-Up}}\left(\frac{n-sb_l}{\sqrt{P_l}}, \frac{n-sb_l}{\sqrt{P_l}}, b_l, P_l, l\right) =$$

$$\sum_{s=1}^{n/b_l}\sum_{k=1}^{l} W_{\text{ML-LU-Up}}\left(\frac{n-sb_l}{\sqrt{P_l}}, \frac{n-sb_l}{\sqrt{P_l}}, b_l, P_l, k\right)\beta_k + S_{\text{ML-LU-Up}}\left(\frac{n-sb_l}{\sqrt{P_l}}, \frac{n-sb_l}{\sqrt{P_l}}, b_l, P_l, k\right)\alpha_k \leq$$

$$\sum_{k=1}^{l}\left(\frac{n^2\sqrt{P_l}}{\prod_{j=k}^{l} P_j}\log\sqrt{P_l} + \frac{4n^3}{3b_l P_l \prod_{j=k}^{l-1}\sqrt{P_j}}(1+\frac{l-k}{\sqrt{P_k}})\right)(\beta_k + \frac{\alpha_k}{B_k}).$$

Total cost of multilevel CALU

Here we estimate the total cost of multilevel CALU on a square $n \times n$ matrix, using l levels of recursion ($l \geq 2$), with respect to the HCP model. We present an approximation of the computation time, the bandwidth cost, and the latency cost, all evaluated on the critical path. At each level r of recursion we consider the following parameters, $P_{r_r} = P_{c_r} = \sqrt{P_r}$ and $b_r = n/(8 \times 2^{l-r}\prod_{j=r}^{l}(\sqrt{P_j}\log^2(\sqrt{P_j})))$ Note that to derive these approximations, we assume that at each level k of hierarchy, the number of nodes $P_k \geq 16$. This assumption results in a non dominating panel factorization in terms of computation. The details of the counts are presented in Appendix B .5.

The computation time of multilevel CALU on the critical path is

$$F_{\text{ML-CALU}}(m,n,b_l,P_l,l)\,\gamma \quad \approx \quad \left(\frac{2n^3}{3P} + \frac{n^3}{Plog^2P_l} + \frac{n^3}{P}\left(\frac{3}{8}\right)^{l-2}\left(\frac{5}{16}l - \frac{53}{128}\right)\right)\gamma.$$

The previous equation shows that multilevel CALU performs more floating-point operations than 1-level CALU, and thus Gaussian elimination with partial pivoting. 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 attain the lower bounds on computation. For example, if we consider a hierarchy with two levels and two nodes at level 2, then the estimated computation time becomes of order $O(\frac{5}{6}\frac{n^3}{P})$. In this case multilevel CALU exceeds the lower bounds on computation by a constant factor.

The bandwidth cost of multilevel CALU, evaluated on the critical path, is bounded as follows

$$\begin{aligned}
\bar{W}_{\text{ML-CALU}}\left(m,n,b_l,P_l,l\right) &= \textstyle\sum_{k=1}^{l} W_{\text{ML-CALU}}\left(m,n,b_l,P_l,k\right)\beta_k \\
&\leq \frac{n^2}{2\prod_{j=1}^{l}\sqrt{P_j}} logP_1 \prod_{j=2}^{l}(1+\tfrac{1}{2}logP_j)\beta_1 \\
&+ \textstyle\sum_{k=1}^{l}\left[\frac{n^2}{\prod_{j=k}^{l}\sqrt{P_j}}\left(\frac{8}{3}log^2P_l(1+\frac{l-k}{\sqrt{P_k}})\right.\right. \\
&+ \left.\left.\frac{(l-2)}{8}(1+\frac{l}{4})\prod_{j=3}^{l}(1+\tfrac{1}{2}logP_j)\right)\right]\beta_k.
\end{aligned}$$

Finally the latency cost of multilevel CALU with respect to network capacity of each level of hierarchy is bounded as follows

$$\begin{aligned}
\bar{S}_{\text{ML-CALU}}\left(m,n,b_l,P_l,l\right) &= \textstyle\sum_{k=1}^{l} S_{\text{ML-CALU}}\left(m,n,b_l,P_l,k\right)\alpha_k \\
&\leq \frac{n^2}{2\prod_{j=1}^{l}\sqrt{P_j}} logP_1 \prod_{j=2}^{l}(1+\tfrac{1}{2}logP_j)\frac{\alpha_1}{B_1} \\
&+ \textstyle\sum_{k=1}^{l}\left[\frac{n^2}{\prod_{j=k}^{l}\sqrt{P_j}}\left(\frac{8}{3}log^2P_l(1+\frac{l-k}{\sqrt{P_k}})\right.\right. \\
&+ \left.\left.\frac{(l-2)}{8}(1+\frac{l}{4})\prod_{j=3}^{l}(1+\tfrac{1}{2}logP_j)\right)\right]\frac{\alpha_k}{B_k}.
\end{aligned}$$

In summary,

$$\begin{aligned}
\bar{W}_{\text{ML-CALU}}\left(m,n,b_l,P_l,l\right) &\leq O(\textstyle\sum_{k=1}^{l}\left[\frac{n^2}{\prod_{j=k}^{l}\sqrt{P_j}}l^2\prod_{j=2}^{l}logP_j\right]\beta_k), \\
\bar{S}_{\text{ML-CALU}}\left(m,n,b_l,P_l,l\right) &\leq O(\textstyle\sum_{k=1}^{l}\left[\frac{n^2}{\prod_{j=k}^{l}\sqrt{P_j}}l^2\prod_{j=2}^{l}logP_j\right]\frac{\alpha_k}{B_k}).
\end{aligned}$$

We recall here the lower bounds on bandwidth and latency under the HCP model at a level k of parallelism.

$$W_k \geq \Omega\left(\frac{n^2}{\prod_{j=k}^{l}\sqrt{P_j}}\right), \qquad S_k \geq \Omega\left(\frac{n^2}{B_k\prod_{j=k}^{l}\sqrt{P_j}}\right).$$

We can therefore conclude that multilevel CALU attains these lower bounds modulo a factor that depends on $l^2\prod_{j=2}^{l}logP_j$ 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.

5.5 Performance predictions

In this section, we present performance predictions of the multilevel CALU algorithm on a large scale platform. Indeed, current petascale platforms already display a hierarchical nature, with heterogeneous network bandwidths and latencies strongly impacting performance of parallel applications. This trend will even be amplified when exascale platforms will become available. We plan here to provide an

insight on what could be observed on such platforms.

We start by modeling an exascale platform with respect to the HCP model, then estimate the running time of our algorithm on this platform. The platform we build here is based on the NERSC Hopper supercomputer [3]. We evaluate the performance of both multilevel CALU and its corresponding 1-level routine 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, $P_k = \sqrt{P_k} \times \sqrt{P_k}$. The matrix size n is varied in the range of $2^4, 2^5, \ldots 2^{30}$. The number of processors of the platform is varied in the same range. For CALU, we set the panel size b to the optimal value used in [60], that is $b = n/\prod_{j=1}^{l}(\sqrt{P_j}\log^2\sqrt{P_j})$. Using this panel size for multilevel CALU, when $l \geq 2$, minimizes both the bandwidth cost and the latency cost at each level of parallelism, with respect to the lower bounds on communication in the HCP model. However it does not allow to keep the extra computation performed during the preprocessing step as a low order term.

To be able to minimize both the communication cost and the additional arithmetic cost, due to the recursive calls, we set the panel size of multilevel CALU to its "best" experimental value, that is,

$$b_i = \frac{n}{8 \times 2^{l-i} \prod_{j=i}^{l}(\sqrt{P_j}\log^2(\sqrt{P_j}))}.$$

Note that for all the figures we present in this section, the white regions in the plots signify either that the problem needs too much memory with respect to the memory available on the platform (the upper left corner), or that it is too small to involve all the compute units of the hierarchy (the bottom right corner). Note also that the two vertical lines in all the figures separate the three levels of parallelism. Thus the first part corresponds to one level of parallelism, the second part to two levels of parallelism, and the third part to three levels of parallelism. Moreover, we would like to mention that, in our figures, the displayed labels correspond to an average of four test cases.

As displayed in Figure 5.4, the use of the "best" experimental panel size results on few test cases, because we only present cases where $b_i \geq 1$. Indeed, we assume that each processor participating in the execution holds at least one element of the input matrix. These cases correspond to the blue region in Figure 5.4. In order to display some additional cases, we the set the panel size b_i to $\frac{n}{8\times 2^{l-i}\prod_{j=i}^{l}(\sqrt{P_j}\log^2(\sqrt{P_j}))}$. The test cases that use this panel size correspond to the green region. When the previous panel size become smaller than 1, we set the panel size b_i to the optimal panel size of CALU at each level of recursion, that is, $b_i = n/\prod_{j=i}^{l}(\sqrt{P_j}\log^2\sqrt{P_j})$. The test cases using this panel size are presented in the orange region. For all our figures which present result for multilevel CALU, we display all the previous test cases except for figures 5.6, 5.16, and 5.17. There, we delimit the three region as follows, a black path between test cases displayed in the blue region and those displayed in the green region and a red path between the latter and test cases presented in the orange region.

For both 1-level CALU and multilevel CALU, we detail the fraction of time in computation, in bandwidth, and in latency on our model of an exascale platform. We also discuss the speedup predictions, comparing 1-level CALU to multilevel CALU on the exascale platform. Then we present the computation to communication ratio for each algorithm. In all our experiments, we assume that for a given level of parallelism, at least 16 compute nodes are used at each level.

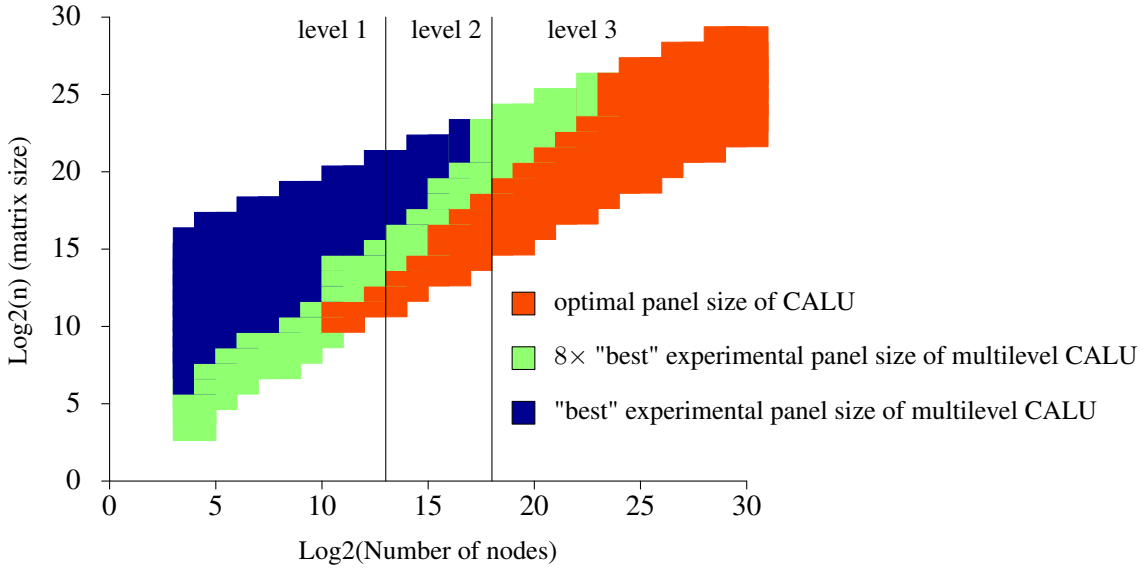


Figure 5.4: All test cases using three different panel sizes.

5.5.1 Modeling an Exascale platform with HCP

There are many unknowns regarding the architecture of exascale platforms. Here we use the characteristics of the NERSC Hopper supercomputer as a building block of our exascale platform [1]. Hopper is a petascale platform which is planned to be upgraded to reach the exascale around year 2018. It is composed of **Compute Nodes** made out of four hexacore AMD Opteron processors Magny-cours running at 2.1 GHz. Each of these 12 cores offers a peak performance of 8.4 GFlop/s. Each node has a total of 32 GB of memory and a HyperTransport interconnect allowing the processors to communicate with an average bandwidth of 19.8 GB/s. Two of these Compute Nodes are connected to a **Gemini ASIC** through an HyperTransport link providing a bandwidth of 10.4 GB/s to each node. Finally, these Gemini ASICs are interconnected through the **Gemini network**, offering an average bandwidth of 3.5GB/s on each link [89, 58]. The parameters of the Hopper platform are presented in Table 5.1.

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

Table 5.1: The Hopper platform (2009).

In order to model an exascale platform based on this architecture, we increase the number of nodes at each level. For the first level we consider 1024 cores instead of 12 cores. We also increase the number of nodes to 32 and the number of interconnects so as to reach a total of $1M$ nodes. This leads to a total of 32768 interconnects. We keep the same amount of memory per core as in the Hopper platform, that is a total of, approximately, 1.3 GB of memory. To estimate the latency and the bandwidth at each level

of the exascale platform, we consider a decrease of order 15% each year for the latency and an increase of order 26% for the bandwidth. The parameters of such a platform are detailed in Table 5.2. We can see that inter-core bandwidth is higher than the intra-socket bandwidth, which is higher than the network chip bandwidth. Note, however, that this is only an attempt to build such an exascale platform. Thus the parameters used here and the number of nodes at each level might be reconsidered with respect to actual exascale system.

Level	#	Bandwidth	Latency
Multi-core processors (level 1)	1024	300 GB/s	1×10^{-10}s
Nodes (level 2)	32	150 GB/s	1×10^{-7}s
Interconnects (level 3)	32768	50 GB/s	1.5×10^{-7}s

Table 5.2: Exascale parameters based on the Hopper platform. (2018)

In order to compare the predicted performance of multilevel algorithms to state-of-the-art communication avoiding algorithms on exascale platforms, we need to estimate the cost of the latter under the HCP model. We therefore make the following assumptions,

- Each communication in a 1-level algorithm corresponds to a communications launched at the deepest level of the hierarchy in the HCP model.
- Each communication goes through the entire hierarchy of the HCP model. In other words, each communication is performed between two compute units belonging to two distant nodes located at the highest level l of the hierarchy. Thus its bandwidth cost is β_l.
- Communication running in parallel share the bandwidth β_l. Thus the effective bandwidth is $\beta_l / \prod_{i=1}^{l-1} P_i$.

In the following, we estimate the running time of our algorithm using an extreme case of the HCP model, where all the network levels are full pipelining. It means that at each level of the network, we set the buffer size B_i of a switch node to the aggregated memory size of the compute units at this level of hierarchy. Thus, for our estimations, we set B_1 to M_1, that is 1.3 GB, B_2 to 1331.2 GB, and B_3 to 42.6 TB. Similar predictions are obtained regarding a full forward network hierarchy, where the buffer size of a switch node of a given level is set to the memory size of an actual compute unit of level 1. The absence of differences between these two extreme cases of the HCP model is mainly due to the fact that we are not analyzing latency bounded cases, which correspond to the bottom right corner. Note, however, that a more realistic model would be in between, that is, it includes both forward and pipelining network levels.

5.5.2 Performance predictions of multilevel CALU

In this section we estimate the running time of multilevel CALU and 1-level CALU on our model of an exascale platform. We base our predictions on the cost model developed in section 5.4 with respect to the HCP model. Figures 5.9, 5.5, 5.11, 5.10, 5.7, and 5.12 show our performance estimations. We consider the division of the running time between computation time, latency cost, and bandwidth cost. Figures 5.9, 5.5, and 5.11 present this division for 1-level CALU. Note that we estimate the running time of 1-level CALU on the exascale platform with respect to the assumptions detailed in section 5.5.1. Figures 5.10, 5.7, and 5.12 present the running time division into floating-point operations cost, latency

cost, and bandwidth cost for multilevel CALU.

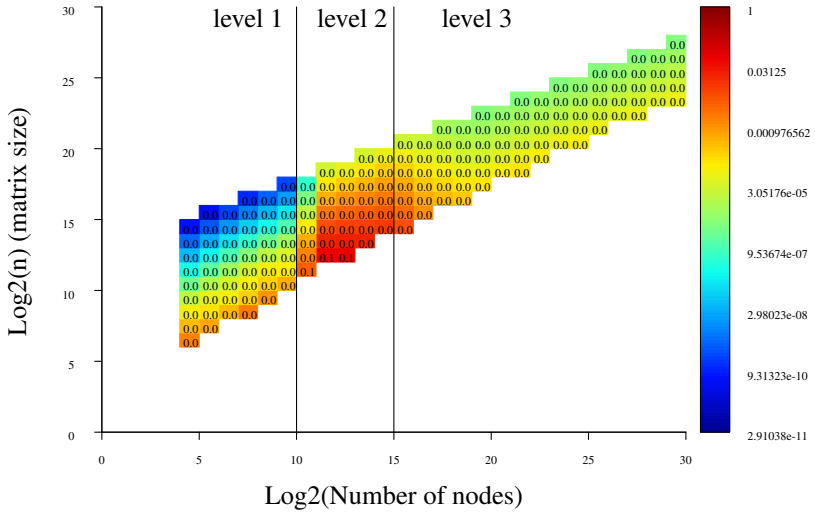


Figure 5.5: Prediction of fraction of time in latency of 1-level CALU on an exascale platform.

These figures show that for both 1-level CALU and multilevel CALU, the estimated computation time dominates the estimated total running time in the upper left corner, that is when the algorithms are applied to large problem using a small number of processors. However when these algorithms are applied to relatively small problems using a large number of processors, the estimated latency is dominating. This corresponds for example, in the case of 1-level CALU, to the bottom part of Figure 5.6. Note that this figure displays non realistic cases, where we allow small block sizes (smaller than 1), just to be able to see cases where the latency matter. We only present it to show the latency cost. While the bandwidth dominates the estimated total running time in the middle region between the two previous ones. Regarding the realistic cases displayed, the bandwidth region corresponds to the bottom part in figures 5.9 and 5.10.

Considering the figures 5.5 and 5.7, which present the latency cost of 1-level CALU and multilevel CALU, we can see that in both cases the latency cost is not dominant, since it is minimized. However we can see in Figure 5.8, that beyond two levels of parallelism the fraction of time in latency is smaller for the multilevel CALU comparing to 1-level CALU. Indeed the ratio of 1-level CALU latency cost to multilevel CALU latency cost is between 6.2 and 10.8. Thus multilevel CALU is expected to be more effective in decreasing the latency cost. Note, however, that the comparison between the two algorithms in terms of latency cost could be improved by estimating the 1-level CALU algorithm with respect to the HCP model.

Figures 5.9 and 5.10 display the fraction of time in bandwidth for respectively 1-level CALU and multilevel CALU. We can see that for 1-level CALU, there are more test cases for which the bandwidth cost is dominant. It is even a bottleneck in the bottom right corner. Moreover, in the region where both multilevel CALU and 1-level CALU are bandwidth-bounded, for the same test case, that is the same

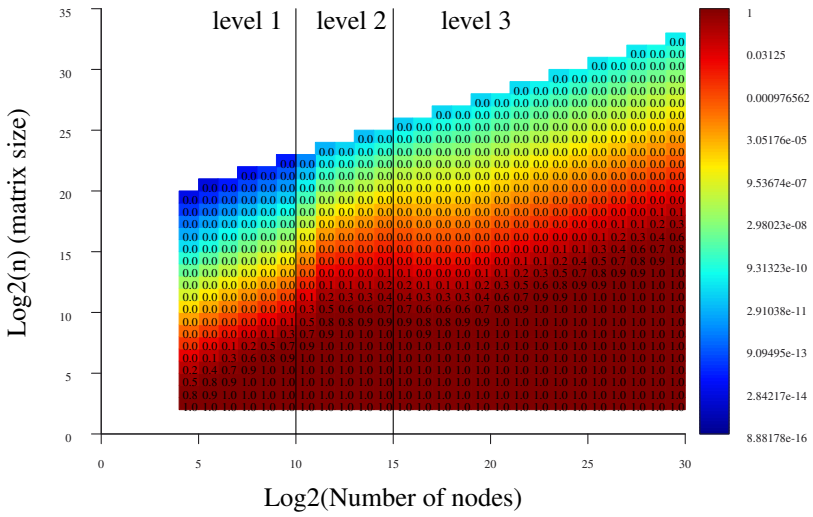


Figure 5.6: Prediction of fraction of time in latency of 1-level CALU on an exascale platform (including non realistic cases, where not all processors have data to compute).

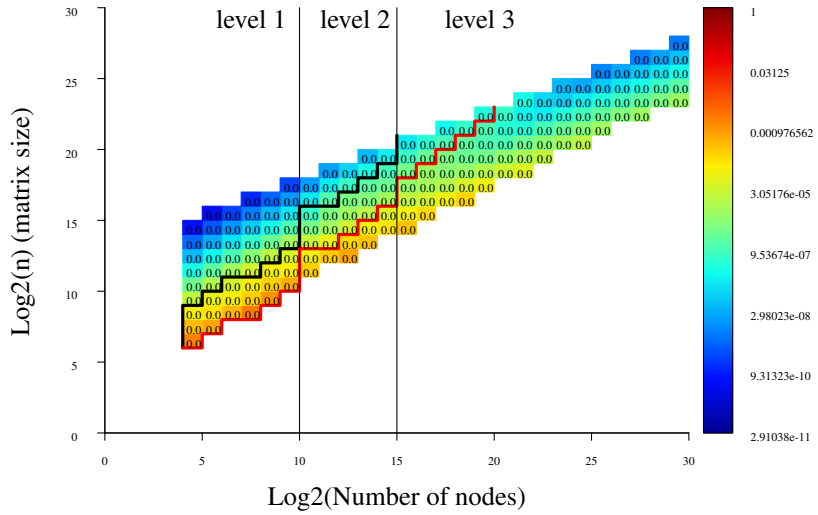


Figure 5.7: Prediction of fraction of time in latency of multilevel CALU on an exascale platform.

matrix size and the same hierarchical system, the fraction of time in bandwidth of multilevel CALU is smaller than that of 1-level CALU. This shows that multilevel CALU is expected to significantly reduce the volume of data which needs to be sent during the hierarchical factorization.

As we can see, in figures 5.11 and 5.12, for multilevel CALU, there are more test cases for which

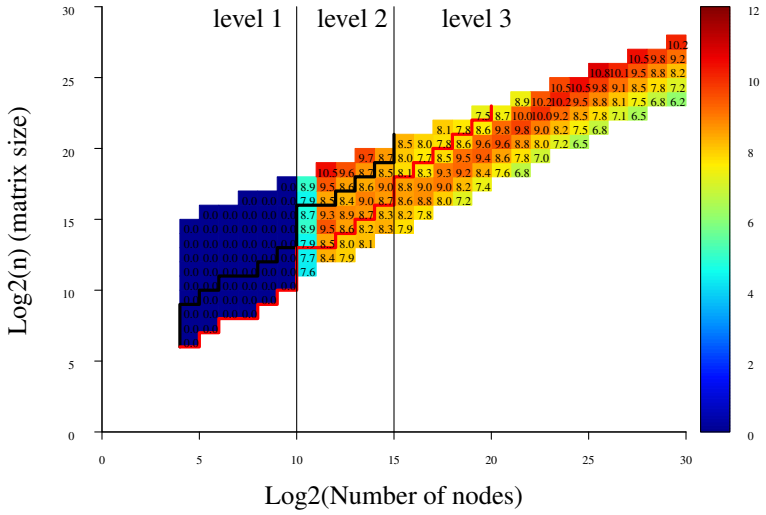


Figure 5.8: Prediction of latency ratio of multilevel CALU comparing to CALU on an exascale platform.

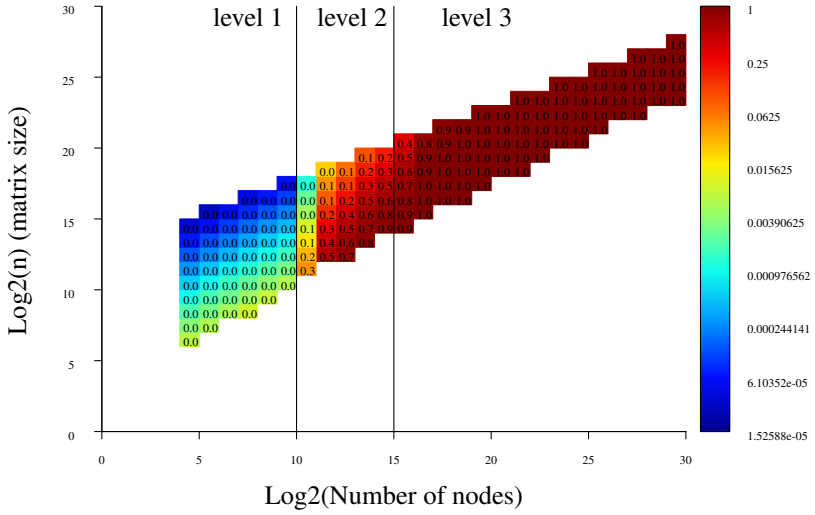


Figure 5.9: Prediction of fraction of time in bandwidth of 1-level CALU on an exascale platform.

the fraction of time spent in computation is dominating, that is where the algorithm is compute-bounded. This is because of, on one hand, the efficiency of multilevel CALU at minimizing the communication cost (hence it is expected to spend more time doing effective computation than communicating), and on the other hand because of the extra computation due to the recursive calls. Thus 1-level CALU is likely to be inefficient for hierarchical systems with multiple levels of parallelism, while multilevel CALU is

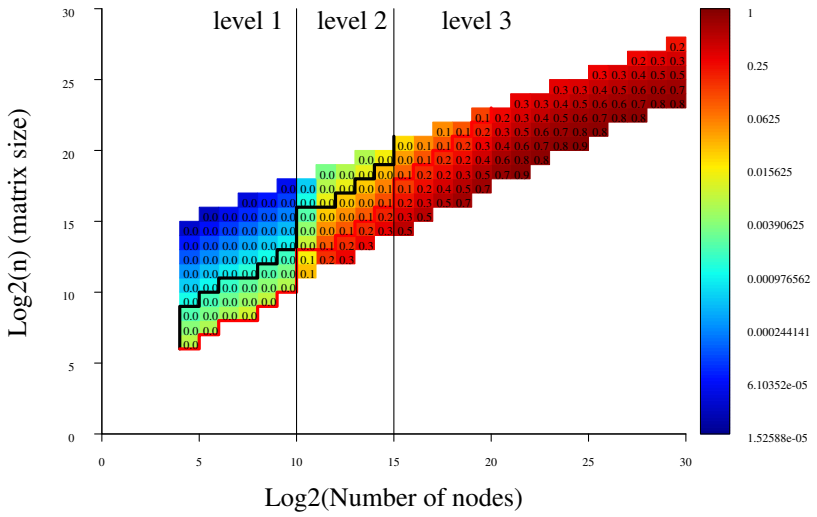


Figure 5.10: Prediction of fraction of time in bandwidth of multilevel CALU on an exascale platform.

predicted to be more efficient on exascale platforms. We can observe in Figure 5.12 that for two levels of parallelism, multilevel CALU is almost performing computation all the time, and this is due to the use of the "best" experimental panel size. While when the CALU panel size is used, the communication cost is reduced compared to CALU, but it is still dominant.

Figure 5.13 presents the predicted ratio of computation time of multilevel CALU with respect to the computation time of 1-level CALU. We can see that this ratio is varying from 1.1 to 13.2 for $l \geq 2$. In this figure, we can easily distinguish the top region, corresponding to the "best" experimental panel size, from the bottom region, where the panel size is is respectively set to the sizes previously detailed. Indeed, for the first region, the predicted computation ratio is between 1.1 and 1.3, since the panel size chosen here allows to keep the extra floating-point operations performed during the preprocessing step as a lower order term comparing to the dominant arithmetic cost. Having a small computation ratio in this region of the plot is important, because it displays cases where both multilevel CALU and 1-level CALU are compute-bounded. However, in the bottom region, the computation ratio varies between 1.8 and 13.2. Thus, the use of the two alternative panel sizes does not allow to reduce the extra floating-point operations.

Figures 5.14 and 5.15 illustrate the division of the computation time between the preprocessing step, that is the selection of the final set of pivot rows, and the rest of the factorization, which includes both the panel factorization and the update of the trailing matrix once the computed permutation is applied to the input matrix. For one level of parallelism, Gaussian elimination with partial pivoting is applied to blocks of the current panel in order to select the final pivot rows. As shown in the first region of Figure 5.14, the computation cost of the preprocessing step is negligible with resoect to the total computation time. While multilevel CALU recursively applies CALU to blocks of the panel during the preprocessing step. For $l \geq 2$, we can see in Figure 5.14 that in the region which corresponds to the "best" experimental panel size, the overall predicted preprocessing step time is still much smaller than the predicted time

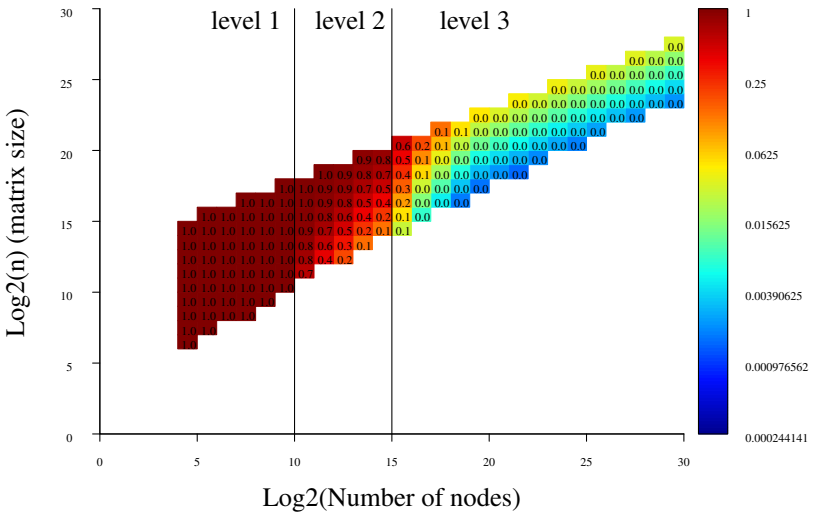


Figure 5.11: Prediction of fraction of time in computations of 1-level CALU on an exascale platform.

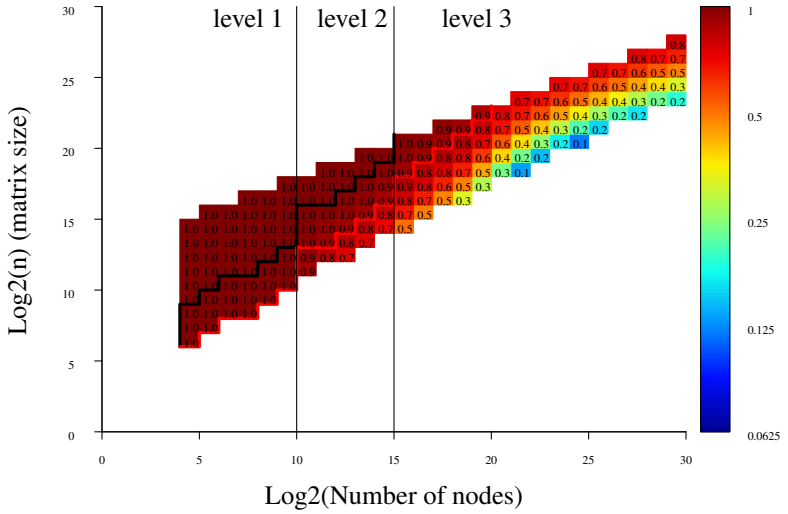


Figure 5.12: Prediction of fraction of time in computations of multilevel CALU on an exascale platform.

spent performing the panel factorizations and the updates. However, when the alternative panel sizes are used, the predicted fraction of time in preprocessing is becoming larger. Thus it attains 90% of the estimated total computation time, when the CALU optimal panel size is used at each level of the recursion. This shows again how important the choice of the panel size is.

Figures 5.16 and 5.17 present the ratio of computation time to communication time for CALU and

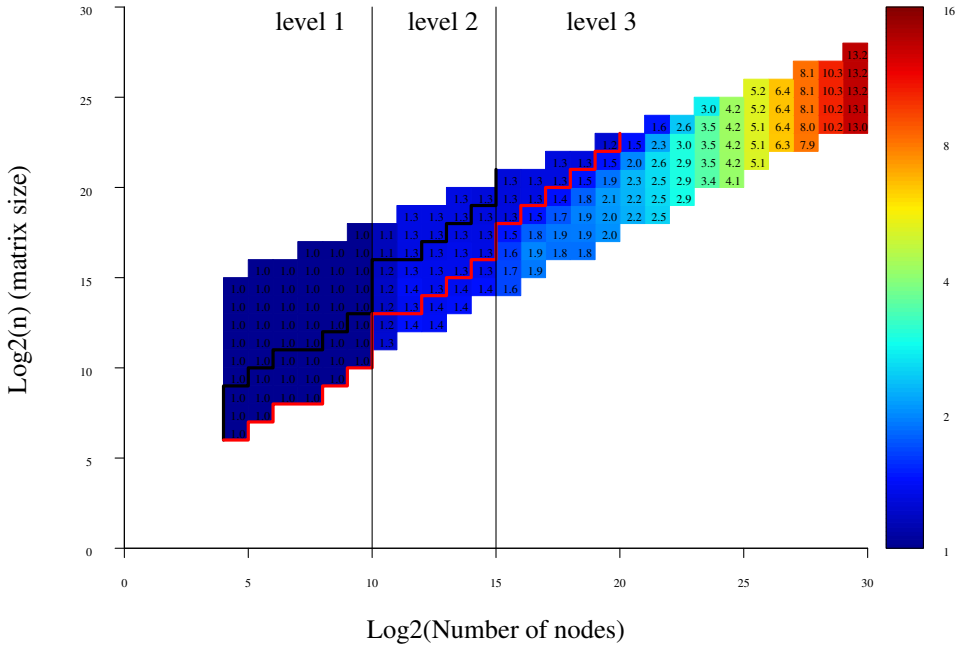


Figure 5.13: Prediction of computation ratio of multilevel CALU comparing to CALU on an exascale platform.

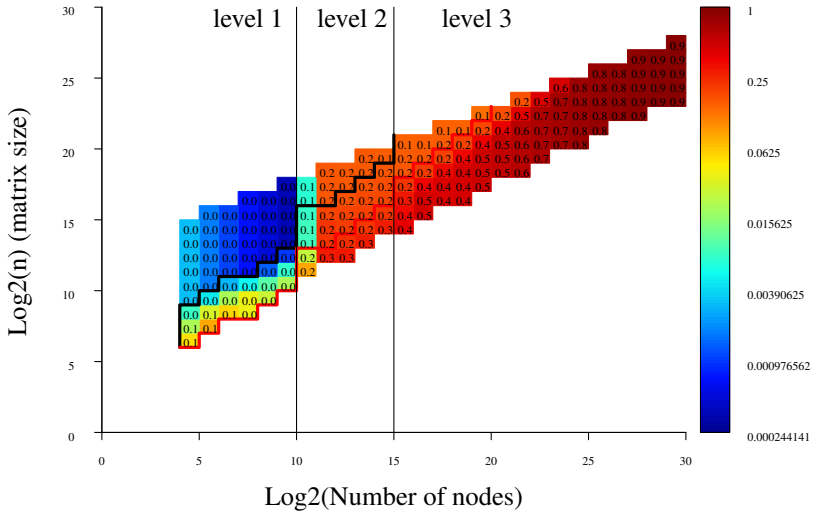


Figure 5.14: Prediction of fraction of time in preprocessing computation comparing to total computation time on an exascale platform.

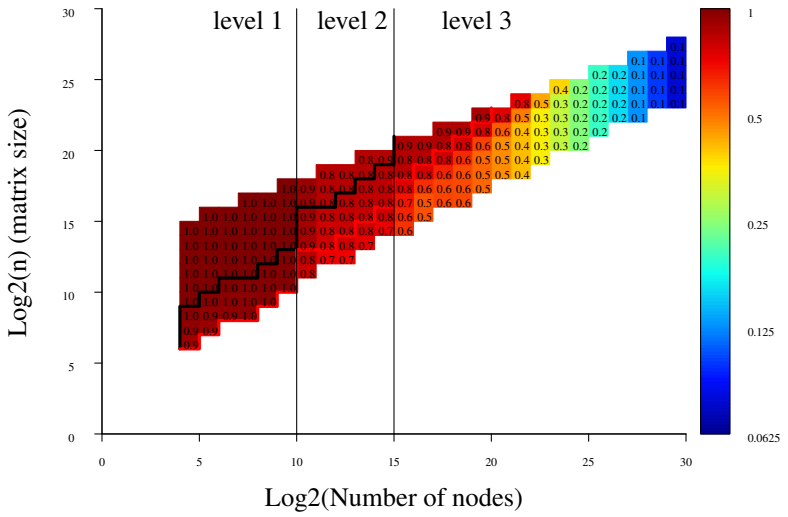


Figure 5.15: Prediction of fraction of time in panel and update computation comparing to total computation time on an exascale platform.

multilevel CALU respectively. Note that in these two figures we only display experiments with the two first panel sizes, which correspond to the blue and the green regions in Figure 5.4. We can see that in all test cases, multilevel CALU performs more computation than communication. The estimated ratio is varying from 363.6 to 9.9. CALU performs more communication than effective computation in the case of three levels of parallelism. For example, for a matrix of size 2^{20} running on a total of 2^{17} processors, the predicted communication time of CALU is up to $100\times$ the predicted computation time, while the predicted communication time of multilevel CALU is approximately $0.9\times$ the computation time for the same test case. This is because of both the reduction of the communication and the additional computation due to recursive calls. Thus 1-level CALU is predicted to be inefficient for hierarchical systems with multiple level of parallelism, while multilevel CALU is estimated to be more efficient on exascale platforms.

Multilevel CALU leads to significant improvements with respect to 1-level CALU when the communication represents an important fraction of the total time. As it can be seen in Figure 5.18, multilevel CALU is expected to show good scalability for large matrices. For example, for a matrix of size $n = 2^{15}$, we measure a predicted speedup of almost $100\times$ compared to 1-level CALU using a total of 524288 cores and three levels of paralleism. Note that multilevel CALU leads to more significant improvements when the bandwidth represents an important fraction of the total time. This mainly corresponds to the central lower part of Figure 5.18. We can see that the most important predicted speedup is observed in the region where three levels of parallelism are used. In the region which corresponds to two levels of parallelism, the expected speedup of multilevel CALU with respect to 1-level CALU is up to $4.1\times$. This means that, because of the extra computation during recursive calls, the reduction of communication amount does not ensure significant speedup. Furthermore, CALU outperforms multilevel CALU in a very small set of cases (0.85 speed down) in the 2 level region. For these cases, the gain in terms of communication cost does not cover the cost of the additional computation performed during

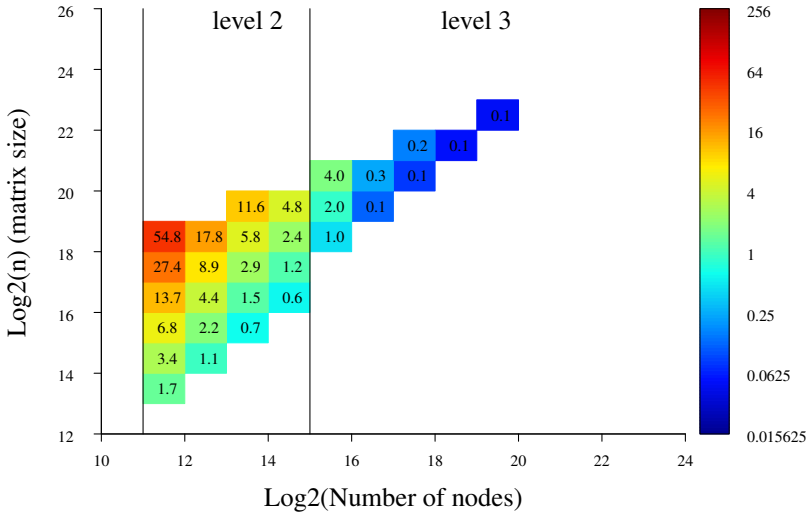


Figure 5.16: Prediction of the ratio of computation time to communication time for 1-level CALU

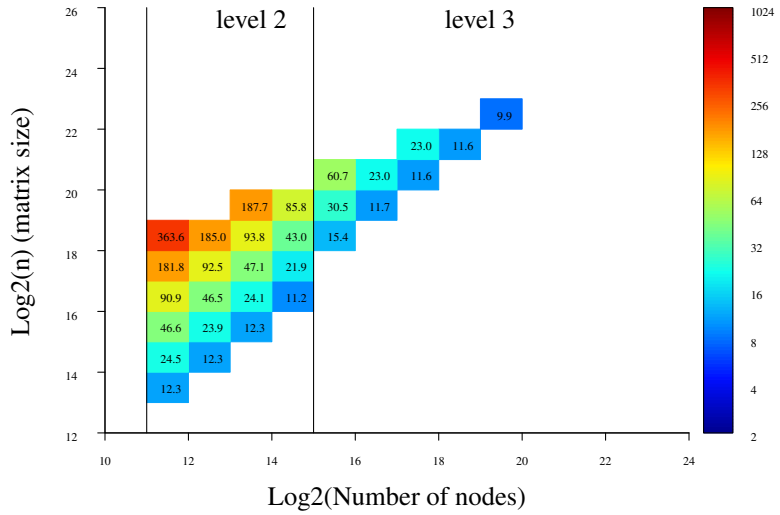


Figure 5.17: Prediction of the ratio of computation time to communication time for ML-CALU

the preprocessing step.

5.6 Related work

The $\alpha DBSP$ model is a recent hierarchical performance model that also takes into accounts varying bandwidths at each level of parallelism. This model is introduced in [87] and it is mainly based on the

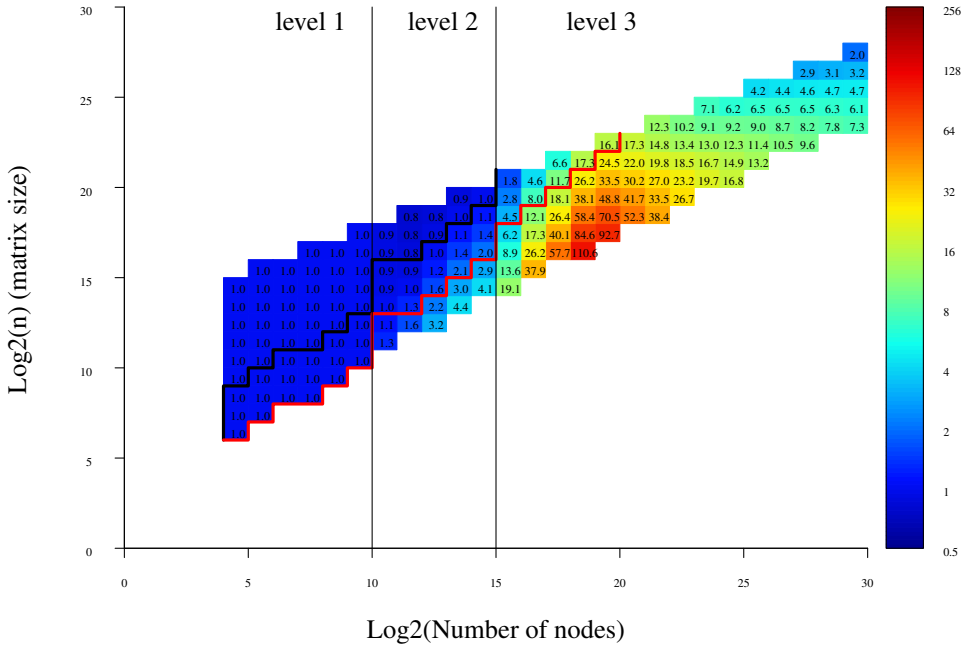


Figure 5.18: Speedup prediction comparing CALU and ML-CALU on an exascale platform

decomposable BSP (DBSP) model. For recall, the DBSP model is an extension of the BSP model which allows to execute supersteps on different sets of processors. Each set is characterized by its latency and its bandwidth. Thus the DBSP model captures different levels of parallelism. The $\alpha DBSP$ model aims at estimating hierarchical bandwidth requirements in order to avoid bandwidth bottleneck. Thus it introduces an α bandwidth growth factor that allows for the computation of a lower bound on the required hierarchical bandwidth and link capacity. At each level of parallelism, the bandwidth and the latency are defined as a function of the bandwidth growth factor and the number of compute units at this level. The bandwidth growth factor, and as a consequence the $\alpha DBSP$ model, highly depends on the network topology. Note that the main difference between the $\alpha DBSP$ model and our model consists of the way the bandwidth are evaluated. In fact, our model is based on discrete bandwidths.

5.7 Conclusion

In this chapter we have introduced a performance model (HCP) that captures the features of a system with multiple levels of parallelism. This model describes how messages and data are routed through the different levels of the network hierarchy. We believe that our model is both simple and realistic. We have also designed a multilevel Cannon algorithm that we use as a tool to model matrix matrix operations performed during the multilevel CALU factorization. We have used the HCP model to analyze the cost of both multilevel Cannon and multilevel CALU. These model costs show that these multilevel algorithms asymptotically attain the lower bounds on communication at each level of parallelism. To have an insight on how might multilevel CALU perform on an exascale platform, we have first designed

a model of such a platform, then we have evaluated our algorithm on it. Our predictions show that multilevel CALU leads to significant speedups with respect to 1-level CALU, especially beyond two levels of parallelism.

One possible perspective might be the improvement of the HCP performance model in order to take into account the network topology at each level of the network hierarchy and the overhead of the switching process, without increasing analysis effort. This is a challenging task, since improving a cost model in general means including further details, which results into a less practical model. Furthermore, we intend to derive the optimal panel size for multilevel CALU. We conjecture that this corresponds to the "best" experimental panel size presented in this chapter. However we still need to prove that it is optimal regarding the minimization of the extra computation performed during the preprocessing step. Note that a possible improvement of our performance predictions might be the analysis of the cost of CALU under the HCP model. Additionally, as mentioned in the previous chapter, using recursive CALU together with multilevel CALU to build an hybrid hierarchical CALU algorithm is likely to further improve the performance of our hierarchical LU factorization, mainly when many levels of parallelism are used. Hence performing recursive CALU at levels where communication is not expensive would reduce the amount of additional computation, since the recursive calls are applied to entire panels.

Conclusion

Throughout this thesis we have studied several methods that compute dense LU decomposition. We have mainly focused on numerical stability and communication cost of these methods. In the first part of our work, we have introduced a sequential LU factorization, LU_PRRP, which is more stable than Gaussian elimination with partial pivoting in terms of growth factor upper bound. This novel algorithm is based on a new pivoting strategy that uses the strong rank revealing QR factorization to select pivot rows and to compute a block LU factorization. We have also shown through extensive experiments on a large set of matrices that the block LU_PRRP factorization is very stable in practice.

In the second part of this thesis we have developed a communication avoiding version of LU_PRRP. This algorithm, referred to as CALU_PRRP, aims on one hand at overcoming the communication bottleneck due to the panel factorization during the LU_PRRP decomposition, and on the other hand at improving the numerical stability of CALU, the communication avoiding version of GEPP. Thus the CALU_PRRP algorithm asymptotically attains the lower bounds on communication in terms of both the volume of data communicated, that is the bandwidth, and the number of messages sent, that is the latency. This improvement in terms of communication cost, in particular latency cost, is due to the use of the tournament pivoting strategy. Moreover the CALU_PRRP factorization is more stable than CALU in terms of worst case growth factor, since the tournament pivoting is based on the same pivoting strategy as the LU_PRRP factorization, that is the strong rank revealing QR pivoting. To be able to make such a conclusion about the numerical stability of the CALU_PRRP algorithm, we have proved that in exact arithmetic, performing CALU_PRRP on a given matrix A is equivalent to performing LU_PRRP on a larger and more sparse matrix formed by blocks of A and blocks of zeros. We note that there are cases of interest for which the upper bound of the growth factor of CALU_PRRP is smaller than that of Gaussian elimination with partial pivoting. Our experiment set shows that CALU_PRRP is very stable in practice and leads to results of the same order of magnitude as GEPP, sometimes even better.

In the third part of this thesis we have presented two hierarchical algorithms for the LU factorization, uni-dimensional recursive CALU (recursive CALU) and bi-dimensional recursive CALU (multilevel CALU). These two methods are based on communication avoiding LU factorization and are designed to better exploit computer systems with multiple levels of parallelism. Regarding the numerical stability, recursive CALU is equivalent to CALU with tournament pivoting based on Gaussian elimination with partial pivoting. However it only reduces the bandwidth cost at each level of the hierarchical system. Thus the latency is still a bottleneck, since at the deepest level of recursion, and during the panel factorization, recursive CALU uses a hierarchical reduction tree, which is based on several levels of reduction, one for each level of parallelism. However multilevel CALU minimizes communication at each level of parallelism in terms of both volume of data and number of messages exchanged during the decomposition. On a multi-core cluster system, that is a system with two levels of parallelism (distributed memory at the top level and shared memory at the deepest level), our experiment set shows that multilevel CALU outperforms ScaLAPACK, especially for tall and skinny matrices. To have an insight on how might multilevel CALU perform on an exascale platform with more than two levels of

parallelism, we have introduced a hierarchical performance model (HCP) that captures the features of a system with multiple levels of parallelism. Then we have evaluated the cost of multilevel CALU with respect to this model. We have shown that it asymptotically attain the lower bounds on communication at each level of parallelism, and that it leads to significant speedups with respect to 1-level CALU.

Perspectives

Future work includes several promising directions. From a theoretical perspective, we could further study the behavior of LU_PRRP on sparse matrices and the amount of the fill in that it could generate. Sophie Moufawad considers in her PhD thesis work the design of an incomplete LU preconditioner (ILU0_PRRP) based on our new pivoting strategy, that is the panel rank revealing pivoting. We believe that such a preconditioner would be more stable than ILU0 and hence it could be very useful for the resolution of large and sparse linear systems. In the context of the communication avoiding algorithms, it is possible to investigate the design of an LU algorithm that minimizes the communication and that has a smaller bound on the growth factor than Gaussian elimination with partial pivoting in general, since, as stated before, the CALU_PRRP algorithm designed in this work could be more stable than GEPP in terms of growth factor upper bound under certain conditions on the reduction tree and the threshold τ. Another direction to explore would be the use of the panel rank revealing pivoting technique in the context of the LDL^T decomposition, for both sequential and parallel cases. However this is a challenging task because of the symmetry. We also intend to further study the numerical stability of multilevel CALU or design a hybrid algorithm that takes advantages of both the numerical stability of recursive CALU and the efficiency of multilevel CALU. This means that until a certain level in the recursion, we use recursive CALU, and beyond this level we rather use multilevel CALU. We note that a possible improvement in the numerical stability of multilevel CALU would be the design of a multilevel CALU_PRRP algorithm, since the CALU_PRRP factorization is more stable than CALU.

From a more practical perspective, we intend to develop routines that implement the LU_PRRP algorithm and that will be integrated in LAPACK and ScaLAPACK libraries. We also intend to finish our implementation of the CALU_PRRP algorithm in order to estimate the performance of this algorithm on parallel machines based on multicore processors, and comparing it with the performance of CALU. Furthermore, we could implement a more flexible and modular hierarchical CALU algorithm for clusters of multi-core processors, where it would be possible to choose the reduction tree at each level of parallelism in order to be cache efficient at the core level and communication avoiding at the distributed level.

Appendix A

Here we present experimental results for the LU_PRRP factorization, the binary tree based CALU_PRRP, and the flat tree based CALU_PRRP. We show results obtained for the LU decomposition and the linear solver. Tables 4, 6, and 8 display the results obtained for random matrices. They show the growth factor, the norm of the factor L and U and their inverses, and the relative error of the decomposition.
Tables 5, 7, and 9 display the results obtained for the special matrices presented in Table 3. The size of the tested matrices is n = 4096. For LU_PRRP and flat tree based CALU_PRRP, the size of the panel is $b = 8$. For binary tree based CALU_PRRP, we use 64 processors ($P = 64$) and a panel of size 8 ($b = 8$), this means that the size of the matrices used at the leaves of the reduction tree is 64×8.

We also present experimental results for binary tree based CALU, binary tree based 2-level CALU, and binary tree based 3-level CALU. Here again we show results obtained for both the LU decomposition and the linear solver. Tables 12 and 13 display the results obtained for the special matrices presented in Table 3. The size of the tested matrices is n = 4096. For binary tree based CALU we use 64 processors and a panel of size 8, this means that the size of the matrices used at the leaves of the reduction tree is 64×8. For binary tree based 2-level CALU, we use $P1 = P2 = 8$, $b2 = 32$, and $b1 = 8$, this means that the size of the matrices used at the leaves of the first level reduction tree is 512×32, while the size of the matrices used at the leaves of the second level reduction tree is 64×8. Note that for the set of special matrices, we could not test the matrix poisson with our code for $n = 8192$.

The tables are presented in the following order.
- Table 4: Stability of the LU decomposition for LU_PRRP and GEPP on random matrices.
- Table 5: Stability of the LU decomposition for LU_PRRP on special matrices.
- Table 6: Stability of the LU decomposition for flat tree based CALU_PRRP and GEPP on random matrices.
- Table 7: Stability of the LU decomposition for flat tree based CALU_PRRP on special matrices.
- Table 8: Stability of the LU decomposition for binary tree based CALU_PRRP and GEPP on random matrices.
- Table 9: Stability of the LU decomposition for binary tree based CALU_PRRP on special matrices.
- Table 10: Stability of the LU decomposition for 2-level CALU and GEPP on random matrices.
- Table 11: Stability of the LU decomposition for GEPP on special matrices of size 4096.
- Table 12: Stability of the LU decomposition for 1-level CALU on special matrices of size 4096.
- Table 13: Stability of the LU decomposition for 2-level CALU on special matrices of size 4096.
- Table 14: Stability of the LU decomposition for GEPP on special matrices of size 8192.
- Table 15: Stability of the LU decomposition for 1-level CALU on special matrices of size 8192.
- Table 16: Stability of the LU decomposition for 2-level CALU on special matrices of size 8192.
- Table 17: Stability of the LU decomposition for 3-level CALU on special matrices of size 8192.

Table 3: Special matrices in our test set.

No.	Matrix	Remarks
1	hadamard	Hadamard matrix, hadamard(n), where n, $n/12$, or $n/20$ is power of 2.
2	house	Householder matrix, $A = \text{eye}(n) - \beta * v * v'$, where $[v, \beta, s]$ = gallery('house', randn(n, 1)).
3	parter	Parter matrix, a Toeplitz matrix with most of singular values near π. gallery('parter', n), or $A(i,j) = 1/(i - j + 0.5)$.
4	ris	Ris matrix, matrix with elements $A(i,j) = 0.5/(n - i - j + 1.5)$. The eigenvalues cluster around $-\pi/2$ and $\pi/2$. gallery('ris', n).
5	kms	Kac-Murdock-Szego Toeplitz matrix. Its inverse is tridiagonal. gallery('kms', n) or gallery('kms', n, rand).
6	toeppen	Pentadiagonal Toeplitz matrix (sparse).
7	condex	Counter-example matrix to condition estimators. gallery('condex', n).
8	moler	Moler matrix, a symmetric positive definite (spd) matrix. gallery('moler', n).
9	circul	Circulant matrix, gallery('circul', randn(n, 1)).
10	randcorr	Random $n \times n$ correlation matrix with random eigenvalues from a uniform distribution, a symmetric positive semi-definite matrix. gallery('randcorr', n).
11	poisson	Block tridiagonal matrix from Poisson's equation (sparse), A = gallery('poisson',sqrt(n)).
12	hankel	Hankel matrix, A = hankel(c, r), where c=randn(n, 1), r=randn(n, 1), and $c(n) = r(1)$.
13	jordbloc	Jordan block matrix (sparse).
14	compan	Companion matrix (sparse), A = compan(randn(n+1,1)).
15	pei	Pei matrix, a symmetric matrix. gallery('pei', n) or gallery('pei', n, randn).
16	randcolu	Random matrix with normalized cols and specified singular values. gallery('randcolu', n).
17	sprandn	Sparse normally distributed random matrix, A = sprandn(n, n,0.02).
18	riemann	Matrix associated with the Riemann hypothesis. gallery('riemann', n).
19	compar	Comparison matrix, gallery('compar', randn(n), unidrnd(2)-1).
20	tridiag	Tridiagonal matrix (sparse).
21	chebspec	Chebyshev spectral differentiation matrix, gallery('chebspec', n, 1).
22	lehmer	Lehmer matrix, a symmetric positive definite matrix such that $A(i,j) = i/j$ for $j \geq i$. Its inverse is tridiagonal. gallery('lehmer', n).
23	toeppd	Symmetric positive semi-definite Toeplitz matrix. gallery('toeppd', n).
24	minij	Symmetric positive definite matrix with $A(i,j) = \min(i,j)$. gallery('minij', n).
25	randsvd	Random matrix with preassigned singular values and specified band-width. gallery('randsvd', n).
26	forsythe	Forsythe matrix, a perturbed Jordan block matrix (sparse).
27	fiedler	Fiedler matrix, gallery('fiedler', n), or gallery('fiedler', randn(n, 1)).
28	dorr	Dorr matrix, a diagonally dominant, ill-conditioned, tridiagonal matrix (sparse).
29	demmel	$A = D*$(eye(n) + $10^{-7}*$rand(n)), where D = diag($10^{14*(0:n-1)/n}$) [36].
30	chebvand	Chebyshev Vandermonde matrix based on n equally spaced points on the interval $[0, 1]$. gallery('chebvand', n).

31	invhess	A=gallery('invhess', n, rand($n-1, 1$)). Its inverse is an upper Hessenberg matrix.
32	prolate	Prolate matrix, a spd ill-conditioned Toeplitz matrix. gallery('prolate', n).
33	frank	Frank matrix, an upper Hessenberg matrix with ill-conditioned eigenvalues.
34	cauchy	Cauchy matrix, gallery('cauchy', randn(n, 1), randn(n, 1)).
35	hilb	Hilbert matrix with elements $1/(i+j-1)$. A =hilb(n).
36	lotkin	Lotkin matrix, the Hilbert matrix with its first row altered to all ones. gallery('lotkin', n).
37	kahan	Kahan matrix, an upper trapezoidal matrix.

Table 4: Stability of the LU decomposition for LU_PRRP and GEPP on random matrices.

LU_PRRP						
n	b	g_W	$\|\|U\|\|_1$	$\|\|U^{-1}\|\|_1$	$\frac{\|\|PA-LU\|\|_F}{\|\|A\|\|_F}$	HPL3
8192	128	2.38E+01	9.94E+04	3.34E+02	5.26E-14	4.68E-03
	64	2.76E+01	1.07E+05	1.06E+03	5.24E-14	4.36E-03
	32	3.15E+01	1.20E+05	3.38E+02	5.14E-14	4.54E-03
	16	3.63E+01	1.32E+05	3.20E+02	4.91E-14	3.79E-03
	8	3.94E+01	1.41E+05	2.65E+02	4.84E-14	3.37E-03
4096	128	1.75E+01	3.74E+04	4.34E+02	2.69E-14	4.58E-03
	64	1.96E+01	4.09E+04	1.02E+03	2.62E-14	4.64E-03
	32	2.33E+01	4.51E+04	2.22E+03	2.56E-14	4.51E-03
	16	2.61E+01	4.89E+04	1.15E+03	2.57E-14	4.08E-03
	8	2.91E+01	5.36E+04	1.93E+03	2.60E-14	3.90E-03
2048	128	1.26E+01	1.39E+04	5.61E+02	1.46E-14	5.16E-03
	64	1.46E+01	1.52E+04	1.86E+02	1.42E-14	5.05E-03
	32	1.56E+01	1.67E+04	3.48E+03	1.38E-14	4.65E-03
	16	1.75E+01	1.80E+04	2.59E+02	1.38E-14	4.59E-03
	8	2.06E+01	2.00E+04	5.00E+02	1.40E-14	4.33E-03
1024	128	8.04E+00	5.19E+03	5.28E+02	7.71E-15	5.45E-03
	64	9.93E+00	5.68E+03	5.28E+02	7.73E-15	4.88E-03
	32	1.13E+01	6.29E+03	3.75E+02	7.51E-15	4.74E-03
	16	1.31E+01	6.91E+03	1.51E+02	7.55E-15	4.08E-03
	8	1.44E+01	7.54E+03	4.16E+03	7.55E-15	4.25E-03
GEPP						
8192	-	5.47E+01	8.71E+03	6.03E+02	7.23E-14	2.84E-03
4096	-	3.61E+01	1.01E+03	1.87E+02	3.88E-14	2.90E-03
2048	-	2.63E+01	5.46E+02	1.81E+02	2.05E-14	3.37E-03
1024	-	1.81E+01	2.81E+02	4.31E+02	1.06E-14	3.74E-03

Table 5: Stability of the LU decomposition for LU_PRRP on special matrices.

	matrix	cond(A,2)	g_W	$\|\|L\|\|_1$	$\|\|L^{-1}\|\|_1$	$\|\|U\|\|_1$	$\|\|U^{-1}\|\|_1$	$\frac{\|\|PA-LU\|\|_F}{\|\|A\|\|_F}$	η	w_b	N_{IR}
well-conditioned	hadamard	1.0E+0	5.1E+02	5.1E+02	2.5E+16	5.6E+0	6.0E+00	5.2E-15	1.1E-15	9.0E-15	2
	house	1.0E+0	7.4E+00	8.3E+02	7.9E+02	3.7E+02	5.0E+01	5.8E-16	4.2E-17	4.8E-15	(2)
	parter	4.8E+0	1.5E+00	4.7E+01	3.7E+00	1.4E+01	3.6E+01	8.5E-16	6.8E-16	4.4E-15	3
	ris	4.8E+0	1.5E+00	4.7E+01	1.1E+01	7.3E+00	7.2E+01	8.5E-16	7.6E-16	4.6E-15	2
	kms	9.1E+0	1.0E+00	6.7E+00	4.0E+00	4.1E+00	1.2E+01	1.4E-16	6.8E-17	3.7E-16	1
	toeppen	1.0E+1	1.3E+00	2.3E+00	3.4E+00	3.6E+01	1.0E+00	1.3E-16	8.3E-17	2.2E-15	1
	condex	1.0E+2	1.0E+00	1.9E+00	5.5E+00	5.9E+02	1.2E+00	6.5E-16	7.4E-16	5.1E-15	(2)
	moler	1.9E+2	1.0E+00	2.7E+03	3.9E+02	1.3E+05	3.0E+15	8.1E-16	3.9E-19	3.4E-17	0
	circul	3.7E+2	1.4E+02	1.0E+03	6.7E+17	7.4E+04	1.3E+01	3.7E-14	3.2E-15	2.1E-14	2
	randcorr	1.4E+3	1.0E+00	3.0E+01	3.7E+01	3.5E+01	2.4E+04	5.8E-16	5.9E-17	5.5E-16	1
	poisson	1.7E+3	1.0E+00	1.9E+00	2.0E+01	7.32E+00	2.0E+01	1.6E-16	9.1E-17	1.8E-15	1
	hankel	2.9E+3	5.0E+01	1.0E+03	1.8E+17	5.7E+04	3.3E+01	3.5E-14	2.8E-15	1.7E-14	2
	jordbloc	5.2E+3	1.0E+00	1.0E+00	1.0E+00	2.0E+00	4.0E+03	0.0E+00	1.9E-17	6.1E-17	0
	compan	7.5E+3	1.0E+00	2.9E+00	3.9E+01	6.8E+02	4.1E+00	6.7E-16	5.5E-16	6.9E-12	1
	pei	1.0E+4	4.9E+00	2.8E+03	1.6E+01	1.7E+01	1.7E+01	6.5E-16	1.4E-17	3.2E-17	0
	randcolu	1.5E+4	3.3E+01	1.1E+03	1.5E+17	8.8E+02	1.3E+04	3.5E-14	2.9E-15	1.8E-14	2
	sprandn	1.6E+4	4.9E+00	7.8E+02	1.4E+17	8.8E+03	5.4E+03	3.1E-14	1.0E-14	1.3E-13	2
	riemann	1.9E+4	1.0E+00	4.0E+03	1.0E+03	3.1E+06	1.4E+02	4.5E-16	1.5E-16	2.0E-15	1
	compar	1.8E+6	9.0E+02	1.7E+03	9.8E+16	9.0E+05	8.0E+02	1.4E-14	1.1E-15	7.7E-15	1
	tridiag	6.8E+6	1.0E+00	1.9E+00	2.0E+02	4.0E+00	1.6E+04	3.2E-16	2.8E-17	2.4E-16	0
	chebspec	1.3E+7	1.0E+00	5.2E+01	1.6E+24	9.7E+06	1.7E+00	5.9E-16	1.0E-18	5.0E-15	1
	lehmer	1.8E+7	1.0E+00	1.5E+03	9.5E+00	8.90E+00	8.1E+03	5.6E-16	1.00E-17	6.1E-17	0
	toeppd	2.1E+7	1.0E+00	4.2E+01	1.2E+02	6.69E+04	7.2E+01	5.8E-16	2.6E-17	1.6E-16	0
	minij	2.7E+7	1.0E+00	4.0E+03	1.1E+03	1.7E+06	4.0E+00	5.3E-16	1.1E-18	1.3E-16	0
	randsvd	6.7E+7	3.9E+00	1.1E+03	3.2E+17	5.4E+00	2.5E+09	5.0E-15	4.6E-16	2.9E-15	2
	forsythe	6.7E+7	1.0E+00	1.0E+00	1.0E+00	1.0E+00	6.7E+07	0.0E+00	0.0E+00	0.0E+00	0
ill-conditioned	fiedler	2.5E+10	1.0E+00	9.3E+02	3.06E+03	6.5E+01	1.5E+07	2.4E-16	9.9E-17	2.0E-15	1
	dorr	7.4E+10	1.0E+00	2.0E+00	5.4E+01	6.7E+05	2.5E+05	1.6E-16	3.4E-17	4.0E-15	1
	demmel	1.0E+14	1.3E+00	1.1E+02	1.7E+16	5.0E+15	2.5E+01	3.2E-15	9.1E-21	1.1E-08	2
	chebvand	3.8E+19	1.8E+02	1.4E+03	1.2E+17	7.3E+04	4.3E+19	5.3E-14	4.0E-17	2.3E-16	0
	invhess	4.1E+19	2.1E+00	4.0E+03	7.4E+03	5.8E+03	1.0E+22	1.4E-15	2.5E-17	3.3E-14	1
	prolate	1.4E+20	8.9E+00	1.3E+03	1.4E+18	1.4E+03	9.7E+20	1.9E-14	1.0E-15	2.6E-14	1
	frank	1.7E+20	1.0E+00	1.9E+00	1.9E+00	4.1E+06	2.3E+16	1.6E-17	1.8E-20	6.2E-17	0
	cauchy	5.5E+21	1.0E+00	3.3E+02	2.0E+06	6.9E+07	1.8E+20	6.0E-16	3.9E-19	1.5E-14	1
	hilb	8.0E+21	1.0E+00	3.0E+03	1.1E+18	2.4E+00	3.1E+22	3.2E-16	6.6E-19	2.3E-17	0
	lotkin	5.4E+22	1.0E+00	2.6E+03	2.0E+18	2.4E+00	8.1E+22	6.6E-17	2.3E-18	1.0E-15	0
	kahan	1.1E+28	1.0E+00	1.0E+00	1.0E+00	5.3E+00	7.6E+52	0E+00	1.1E-17	5.2E-16	1

Table 6: Stability of the LU decomposition for flat tree based CALU_PRRP and GEPP on random matrices

Flat Tree based CALU_PRRP								
n	b	g_W	$\|\|L\|\|_1$	$\|\|L^{-1}\|\|_1$	$\|\|U\|\|_1$	$\|\|U^{-1}\|\|_1$	$\frac{\|\|PA-LU\|\|_F}{\|\|A\|\|_F}$	HPL3
8192	128	3.44E+01	2.96E+03	2.03E+03	1.29E+05	2.86E+03	8.67E-14	5.00E-03
	64	4.42E+01	3.30E+03	2.28E+03	1.43E+05	6.89E+02	8.56E-14	6.40E-03
	32	5.88E+01	4.34E+03	2.50E+03	1.54E+05	3.79E+02	8.15E-14	6.40E-03
	16	6.05E+01	4.54E+03	2.50E+03	1.53E+05	4.57E+02	6.59E-14	4.60E-03
	8	5.25E+01	3.24E+03	2.63E+03	1.60E+05	1.78E+02	5.75E-14	4.10E-03
4096	128	2.38E+01	1.60E+03	1.05E+03	4.75E+04	5.11E+03	4.15E-14	5.16E-03
	64	2.90E+01	1.85E+03	1.19E+03	5.18E+04	2.38E+03	4.55E-14	5.65E-03
	32	3.32E+01	1.89E+03	1.29E+03	5.57E+04	3.04E+02	4.63E-14	5.51E-03
	16	3.87E+01	1.94E+03	1.34E+03	5.90E+04	2.62E+02	4.54E-14	5.18E-03
	8	3.80E+01	1.63E+03	1.36E+03	5.90E+04	5.63E+02	4.24E-14	4.73E-03
2048	128	1.57E+01	6.90E+02	5.26E+02	1.66E+04	3.79E+02	2.01E-14	5.39E-03
	64	1.80E+01	7.94E+02	6.06E+02	1.86E+04	4.22E+02	2.25E-14	5.96E-03
	32	2.19E+01	9.30E+02	6.86E+02	2.05E+04	1.08E+02	2.41E-14	5.80E-03
	16	2.53E+01	8.62E+02	6.99E+02	2.17E+04	3.50E+02	2.36E-14	5.36E-03
	8	2.62E+01	8.32E+02	7.19E+02	2.21E+04	4.58E+02	2.27E-14	5.13E-03
1024	128	9.36E+00	3.22E+02	2.59E+02	5.79E+03	9.91E+02	9.42E-15	5.77E-03
	64	1.19E+01	3.59E+02	3.00E+02	6.67E+03	1.79E+02	1.09E-14	6.73E-03
	32	1.40E+01	4.27E+02	3.51E+02	5.79E+03	2.99E+02	1.21E-14	5.42E-03
	16	1.72E+01	4.42E+02	3.74E+02	8.29E+03	2.10E+02	1.24E-14	5.48E-03
	8	1.67E+01	4.16E+02	3.83E+02	8.68E+03	1.85E+02	1.19E-14	5.37E-03
GEPP								
8192	-	5.5E+01	1.9E+03	2.6E+03	8.7E+03	6.0E+02	7.2E-14	2.8E-03
4096	-	3.61E+01	1.01E+03	1.38E+03	2.31E+04	1.87E+02	3.88E-14	2.90E-03
2048	-	2.63E+01	5.46E+02	7.44E+02	6.10E+04	1.81E+02	2.05E-14	3.37E-03
1024	-	1.81E+01	2.81E+02	4.07E+02	1.60E+05	4.31E+02	1.06E-14	3.74E-03

Table 7: Stability of the LU decomposition for flat tree based CALU_PRRP on special matrices.

	matrix	cond(A,2)	g_W	$\|\|L\|\|_1$	$\|\|L^{-1}\|\|_1$	$\|\|U\|\|_1$	$\|\|U^{-1}\|\|_1$	$\frac{\|\|PA-LU\|\|_F}{\|\|A\|\|_F}$	η	w_b	N_{IR}
well-conditioned	hadamard	1.0E+0	5.1E+02	5.1E+02	1.4E+02	5.8E+04	5.3E+00	5.2E-15	9.5E-16	7.9E-15	1
	house	1.0E+0	6.5E+00	9.0E+02	3.3E+02	3.3E+02	5.2E+01	6.6E-16	4.8E-17	5.3E-15	1
	parter	4.8E+0	1.5E+00	4.7E+01	3.7E+00	1.4E+01	3.6E+01	8.6E-16	6.8E-16	4.4E-15	1
	ris	4.8E+0	1.5E+00	4.7E+01	3.7E+00	7.3E+00	7.2E+01	8.5E-16	6.8E-16	4.1-15	1
	kms	9.1E+0	1.0E+00	5.4E+00	3.2E+00	3.9E+00	9.8E+00	1.13E-16	7.0E-17	4.2E-16	1
	toeppen	1.0E+1	1.3E+00	2.3E+00	3.4E+00	3.6E+01	1.0E+00	1.0E-16	8.3E-17	2.7E-15	1
	condex	1.0E+2	1.0E+00	1.9E+00	5.5E+00	5.9E+02	1.2E+00	6.5E-16	7.4E-16	5.2E-15	1
	moler	1.9E+2	1.4E+00	2.7E+03	1.5E+03	2.0E+06	3.0E+15	8.7E-16	6.0E-19	4.4E-17	0
	circul	3.7E+2	1.5E+02	1.5E+03	1.4E+03	8.5E+04	2.5E+01	4.7E-14	4.0E-15	2.4E-14	1
	randcorr	1.4E+3	1.0E+00	3.0E+01	5.9E+01	3.5E+01	2.4E+04	5.8E-16	5.9E-17	5.3E-16	1
	poisson	1.7E+3	1.0E+00	1.9E+00	2.2E+01	7.3E+00	2.0E+01	1.6E-16	9.0E-17	1.8E-15	1
	hankel	2.9E+3	7.9E+01	1.6E+03	1.6E+03	6.5E+04	8.2E+01	4.5E-14	3.6E-15	2.2E-14	1
	jordbloc	5.2E+3	1.0E+00	1.0E+00	1.0E+00	2.0E+00	4.0E+03	0.0E+00	1.9E-17	6.1E-17	0
	compan	7.5E+3	1.0E+00	2.9E+00	5.9E+02	6.8E+02	4.1E+00	6.7E-16	5.4E-16	5.5E-12	1
	pei	1.0E+4	4.9E+00	2.8E+03	1.6E+01	1.7E+01	1.7E+01	6.5E-16	2.8E-17	4.7E-17	0
	randcolu	1.5E+4	3.7E+01	1.6E+03	1.3E+03	1.0E+03	1.3E+04	4.4E-14	3.3E-15	2.1E-14	1
	sprandn	1.6E+4	6.3E+00	1.2E+03	1.6E+03	9.7E+03	3.5E+03	4.0E-14	1.2E-14	1.5E-13	1
	riemann	1.9E+4	1.0E+00	4.0E+03	1.0E+03	3.1E+06	1.4E+02	4.5E-16	1.5E-16	2.0E-15	1
	compar	1.8E+6	9.0E+02	2.0E+03	1.3E+05	9.0E+05	1.0E+03	1.7E-14	1.2E-15	8.3E-15	1
	tridiag	6.8E+6	1.0E+00	1.9E+00	1.8E+02	4.0E+00	1.6E+04	3.2E-16	2.8E-17	2.6E-16	0
	chebspec	1.3E+7	1.0E+00	5.2E+01	5.6E+01	9.7E+06	1.7E+00	5.9E-16	1.0E-18	4.4E-15	1
	lehmer	1.8E+7	1.0E+00	1.5E+03	8.9E+00	8.9E+00	8.1E+03	5.6E-16	1.0E-17	6.0E-17	0
	toeppd	2.1E+7	1.0E+00	4.2E+01	1.3E+03	6.6E+04	7.2E+01	5.8E-16	2.6E-17	1.7E-16	0
	minij	2.7E+7	1.0E+00	4.0E+03	1.1E+03	1.7E+06	4.0E+00	5.3E-16	7.7E-19	1.0E-16	0
	randsvd	6.7E+7	5.0E+00	1.5E+03	1.3E+03	5.8E+00	4.1E+09	6.1E-15	5.1E-16	2.9E-15	1
	forsythe	6.7E+7	1.0E+00	1.0E+00	1.0E+00	1.0E+00	6.7E+07	0.0E+00	0.0E+00	0.0E+00	0
ill-conditioned	fiedler	2.5E+10	1.0E+00	9.3E+02	1.4E+01	6.5E+01	1.5E+07	2.4E-16	8.9E-17	1.9E-15	1
	dorr	7.4E+10	1.0E+00	2.0E+00	4.0E+01	6.7E+05	2.5E+05	1.6E-16	3.7E-17	3.4E-15	1
	demmel	1.0E+14	1.6E+00	1.6E+02	2.5E+02	5.7E+15	3.5E+01	3.7E-15	9.2E-21	1.1E-08	1
	chebvand	3.8E+19	2.1E+02	1.1E+04	2.6E+03	8.5E+04	3.5E+19	6.2E-14	6.9E-17	5.1E-16	1
	invhess	4.1E+19	2.0E+00	4.0E+03	1.5E+03	8.2E+03	2.9E+28	1.2E-15	8.0E-18	3.7E-15	1
	prolate	1.4E+20	1.1E+01	1.4E+03	4.1E+03	1.6E+03	3.6E+21	2.6E-14	1.2E-15	4.0E-14	1
	frank	1.7E+20	1.0E+00	1.9E+00	1.9E+00	4.1E+06	2.3E+16	1.6E-17	2.6E-20	6.8E-17	0
	cauchy	5.5E+21	1.0E+00	3.2E+02	1.8E+02	4.3E+07	5.6E+18	5.6E-16	6.8E-19	3.3E-14	1
	hilb	8.0E+21	1.0E+00	3.0E+03	1.3E+03	2.4E+00	9.5E+22	3.2E-16	5.2E-19	1.8E-17	0
	lotkin	5.4E+22	1.0E+00	2.8E+03	1.2E+03	2.4E+00	2.5E+21	6.5E-17	5.6E-18	2.3E-15	(1)
	kahan	1.1E+28	1.0E+00	1.0E+00	1.0E+00	5.3E+00	7.6E+52	0.0E+00	8.1E-18	4.3E-16	1

Table 8: Stability of the LU decomposition for binary tree based CALU_PRRP and GEPP on random matrices.

Binary Tree based CALU_PRRP									
n	P	b	g_W	$\|\|L\|\|_1$	$\|\|L^{-1}\|\|_1$	$\|\|U\|\|_1$	$\|\|U^{-1}\|\|_1$	$\frac{\|\|PA-LU\|\|_F}{\|\|A\|\|_F}$	HPL3
8192	256	16	4.29E+01	4.41E+03	2.68E+03	1.62E+05	2.15E+02	7.39E-14	4.69E-03
	128	32	4.58E+01	3.47E+03	2.58E+03	1.56E+05	2.96E+02	8.67E-14	4.98E-03
		16	4.97E+01	4.05E+03	2.73E+03	1.63E+05	2.16E+02	7.59E-14	5.76E-03
	64	64	4.77E+01	3.23E+03	2.35E+03	1.45E+05	3.22E+02	9.14E-14	5.91E-03
		32	4.86E+01	4.16E+03	2.64E+03	1.58E+05	5.15E+02	9.04E-14	5.09E-03
		16	5.37E+01	3.83E+03	2.67E+03	1.64E+05	2.56E+02	7.30E-14	5.75E-03
4096	256	8	3.49E+01	1.69E+03	1.4E+03	6.12E+04	3.07E+02	4.71E-14	4.69E-03
	128	16	4.09E+01	1.91E+03	1.39E+03	6.02E+04	5.13E+02	4.99E-14	5.25E-03
		8	3.85E+01	1.78E+03	1.40E+03	6.09E+04	1.52E+04	4.75E-14	4.98E-03
	64	32	3.06E+01	1.95E+03	1.34E+03	5.80E+04	2.99E+02	4.94E-14	5.71E-03
		16	3.88E+01	1.95E+03	1.37E+03	6.02E+04	2.39E+02	5.06E-14	5.44E-03
		8	3.64E+01	1.73E+03	1.40E+03	5.97E+04	1.87E+02	4.73E-14	4.29E-03
	32	64	2.80E+01	1.69E+03	1.18E+03	5.15E+04	2.12E+03	4.67E-14	6.06E-03
		32	3.71E+01	1.83E+03	1.32E+03	5.82E+04	3.74E+02	4.92E-14	5.28E-03
		16	3.75E+01	1.97E+03	1.41E+03	6.24E+04	2.23E+02	5.05E-14	5.27E-03
		8	3.62E+01	1.95E+03	1.43E+03	6.11E+04	7.21E+02	4.70E-14	4.91E-03
2048	128	8	2.91E+01	8.48E+02	7.59E+02	2.32E+04	3.37E+02	2.49E-14	4.72E-03
	64	16	2.63E+01	8.42E+02	7.34E+02	2.22E+04	2.76E+02	2.57E-14	5.29E-03
		8	3.01E+01	8.48E+02	7.36E+02	2.28E+04	3.79E+02	2.45E-14	5.10E-03
	32	32	2.18E+01	8.70E+02	6.66E+02	2.04E+04	5.79E+03	2.46E-14	5.44E-03
		16	2.66E+01	1.05E+03	7.24E+02	2.25E+04	8.48E+02	2.57E-14	5.29E-03
		8	2.81E+01	8.86E+02	7.42E+02	2.30E+04	1.45E+02	2.43E-14	5.04E-03
1024	64	8	1.73E+01	4.22E+02	3.87E+02	8.44E+03	2.33E+02	1.34E-14	5.70E-03
	32	16	1.55E+01	4.39E+02	3.72E+02	7.98E+03	1.67E+02	1.27E-14	4.89E-03
		8	1.84E+01	4.00E+02	3.92E+02	8.46E+03	1.29E+02	1.25E-14	5.40E-03
GEPP									
8192	-	-	5.5E+01	1.9E+03	2.6E+03	8.7E+03	6.0E+02	7.2E-14	2.8E-03
4096	-	-	3.61E+01	1.01E+03	1.38E+03	2.31E+04	1.87E+02	3.88E-14	2.90E-03
2048	-	-	2.63E+01	5.46E+02	7.44E+02	6.10E+04	1.81E+02	2.05E-14	3.37E-03
1024	-	-	1.81E+01	2.81E+02	4.07E+02	1.60E+05	4.31E+02	1.06E-14	3.74E-03

Table 9: Stability of the LU decomposition for binary tree based CALU_PRRP on special matrices.

	matrix	cond(A,2)	g_W	$\|\|L\|\|_1$	$\|\|L^{-1}\|\|_1$	$\|\|U\|\|_1$	$\|\|U^{-1}\|\|_1$	$\frac{\|\|PA-LU\|\|_F}{\|\|A\|\|_F}$	η	w_b	N_{IR}
well-conditioned	hadamard	1.0E+0	5.1E+02	5.1E+02	1.3E+02	4.9E+04	6.5E+00	5.3E-15	1.2E-15	8.7E-15	2
	house	1.0E+0	7.4E+00	8.3E+02	3.8E+02	3.7E+02	5.0E+01	5.8E-16	4.2E-17	4.8E-15	3
	parter	4.8E+0	1.5E+00	4.7E+01	3.7E+00	1.4E+01	3.6E+01	8.6E-16	6.9E-16	4.4E-15	3
	ris	4.8E+0	1.5E+00	4.7E+01	3.7E+00	7.3E+00	7.2E+01	8.5E-16	6.2E-16	4.4E-15	2
	kms	9.1E+0	1.0E+00	5.1E+00	3.4E+00	4.3E+00	9.3E+00	1.1E-16	6.3E-17	3.6E-16	1
	toeppen	1.0E+1	1.3E+00	2.3E+00	3.4E+00	3.6E+01	1.0E+00	1.0E-16	8.2E-17	4.0E-15	1
	condex	1.0E+2	1.0E+00	1.9E+00	5.5E+00	5.9E+02	1.2E+00	6.5E-16	7.4E-16	5.2E-15	2
	moler	1.9E+2	1.0E+00	4.0E+03	9.0E+00	2.2E+04	3.8E+15	1.0E-18	2.1E-20	2.7E-16	1
	circul	3.7E+2	1.5E+02	1.6E+03	1.4E+03	8.4E+04	2.1E+01	5.2E-14	4.1E-15	3.2E-14	2
	randcorr	1.4E+3	1.4E+02	5.4E+02	4.4E+03	2.9E+03	6.1E+03	2.2E-15	1.3E-16	5.7E-15	2
	poisson	1.7E+3	1.0E+00	1.9E+00	2.2E+01	7.3E+00	2.0E+01	1.7E-16	9.3E-17	1.5E-15	1
	hankel	2.9E+3	6.1E+01	1.84E+03	1.5E+03	6.5E+04	8.0E+01	4.9E-14	3.9E-15	2.4E-14	2
	jordbloc	5.2E+3	1.0E+00	1.0E+00	1.0E+00	2.0E+00	4.0E+03	0.0E+00	1.9E-17	7.67E-17	0
	compan	7.5E+3	2.0E+00	2.9E+00	6.8E+02	6.7E+02	9.4E+00	9.8E-16	6.1E-16	6.3E-12	1
	pei	1.0E+4	1.3E+00	5.5E+02	7.9E+00	1.6E+01	2.9E+00	3.6E-16	2.2E-17	5.0E-17	0
	randcolu	1.5E+4	4.6E+01	1.7E+03	1.4E+03	1.0E+03	1.1E+04	4.8E-14	3.5E-15	2.2E-14	2
	sprandn	1.6E+4	6.6E+00	1.1E+03	1.6E+03	9.6E+03	1.4E+03	4.1E-14	1.3E-14	2.0E-13	(1)
	riemann	1.9E+4	1.0E+00	4.0E+03	1.8E+03	7.6E+06	1.4E+02	1.7E-16	1.2E-16	1.1E-15	1
	compar	1.8E+6	9.0E+02	2.0E+03	1.3E+05	9.0E+05	6.5E+02	1.9E-14	1.2E-15	8.9E-15	1
	tridiag	6.8E+6	1.0E+00	1.9E+00	1.8E+02	4.0E+00	1.6E+04	3.2E-16	2.8E-17	2.6E-16	0
	chebspec	1.3E+7	1.0E+00	5.2E+01	5.6E+01	9.7E+06	1.7E+00	6.0E-16	7.8E-19	1.3E-15	1
	lehmer	1.8E+7	1.0E+00	1.5E+03	8.9E+00	8.9E+00	8.1E+03	5.7E-16	9.92E-18	5.7E-17	0
	toeppd	2.1E+7	1.0E+00	4.3E+01	9.1E+02	6.8E+04	1.8E+01	5.8E-16	2.7E-17	1.7E-16	0
	minij	2.7E+7	1.0E+00	4.0E+03	9.0E+00	1.8E+04	4.0E+00	6.4E-19	6.9E-19	6.1E-18	0
	randsvd	6.7E+7	5.4E+00	1.7E+03	1.4E+03	6.4E+00	3.2E+09	7.2E-15	5.5E-16	3.3E-15	2
	forsythe	6.7E+7	1.0E+00	1.0E+00	1.0E+00	1.0E+00	6.7E+07	0.0E+00	0.0E+00	0.0E+00	0
ill-conditioned	fiedler	2.5E+10	1.0E+00	9.0E+02	1.2E+01	5.5E+01	1.4E+07	2.5E-16	7.9E-17	1.7E-15	1
	dorr	7.4E+10	1.0E+00	2.0E+00	4.0E+01	6.7E+05	2.5E+05	1.6E-16	3.3E-17	3.4E-15	1
	demmel	1.0E+14	1.5E+00	1.2E+02	5.3E+02	5.5E+15	2.9E+01	3.3E-15	9.0E-21	7.9E-09	2
	chebvand	3.8E+19	3.2E+02	2.2E+03	3.3E+03	8.9E+04	1.6E+20	7.5E-14	3.7E-17	2.3E-16	0
	invhess	4.1E+19	1.7E+00	4.0E+03	9.0E+00	4.5E+03	1.4E+100	2.9E-15	1.5E-16	6.9E-14	1
	prolate	1.4E+20	1.2E+01	2.1E+03	4.7E+03	2.1E+03	1.6E+21	2.8E-14	1.2E-15	3.9E-14	1
	frank	1.7E+20	1.0E+00	1.9E+00	1.9E+00	4.1E+06	2.3E+16	1.6E-17	2.6E-20	6.8E-17	0
	cauchy	5.5E+21	1.0E+00	3.2E+02	4.1E+02	5.4E+07	4.7E+18	6.1E-16	6.7E-19	5.7E-14	2
	hilb	8.0E+21	1.0E+00	2.9E+03	1.4E+03	2.4E+00	2.3E+22	3.2E-16	6.3E-19	2.2E-17	0
	lotkin	5.4E+22	1.0E+00	3.0E+03	1.4E+03	2.4E+00	3.7E+22	6.6E-17	5.2E-18	1.75E-15	1
	kahan	1.1E+28	1.0E+00	1.0E+00	1.0E+00	5.3E+00	7.6E+52	0.0E+00	1.2E-17	3.9E-16	1

Table 10: Stability of the LU decomposition for binary tree based 2-level CALU and GEPP on random matrices.

Binary tree based 2-level CALU														
n	P_2	b_2	P_1	b_1	g_W	g_D	g_T	τ_{ave}	τ_{min}	$\|\|L\|\|_1$	$\|\|L^{-1}\|\|_1$	$\|\|U\|\|_1$	$\|\|U^{-1}\|\|_1$	$\frac{\|\|PA-LU\|\|_F}{\|\|A\|\|_F}$
8192	64	32	4	8	9.2E+1	1.2E+2	5.4E+2	0.81	0.31	4.2E+3	3.4E+3	2.1E+5	5.7E+3	6.1E-14
		16	4	4	9.4E+1	1.2E+2	5.3E+2	0.84	0.30	4.6E+3	3.3E+3	2.0E+3	4.3E+5	5.8E-14
	32	64	4	16	1.0E+2	1.1E+2	6.0E+2	0.79	0.33	4.2E+3	3.5E+3	2.1E+5	3.5E+2	6.4E-14
		32	4	8	1.1E+2	1.4E+2	6.6E+2	0.81	0.30	4.1E+3	3.4E+3	2.1E+5	1.4E+3	6.2E-14
		16	4	4	1.1E+2	1.2E+2	6.4E+2	0.84	0.31	4.5E+3	3.4E+3	2.0E+5	1.7E+3	5.8E-14
	16	128	4	32	8.3E+1	9.7E+1	4.7E+2	0.79	0.37	3.8E+3	3.3E+3	2.0E+5	2.3E+2	6.3E-14
		64	4	16	8.8E+1	1.1E+2	5.3E+2	0.79	0.30	4.4E+3	3.4E+3	2.1E+5	3.8E+3	6.3E-14
		32	4	8	9.7E+1	1.0E+2	5.5E+2	0.81	0.29	4.5E+3	3.5E+3	2.1E+5	2.2E+3	6.2E-14
		16	4	4	1.1E+2	1.1E+2	6.3E+2	0.85	0.30	4.1E+3	3.3E+3	2.0E+5	3.5E+3	5.7E-14
4096	64	16	4	4	5.8E+1	7.8E+1	3.2E+2	0.86	0.32	1.8E+3	1.7E+3	7.5E+4	2.8E+3	2.9E-14
	32	32	4	8	7.2E+1	8.2E+1	3.8E+2	0.82	0.35	1.9E+3	1.8E+3	7.7E+4	3.7E+3	3.1E-14
		16	4	4	6.3E+1	8.5E+1	3.5E+2	0.85	0.33	2.0E+3	1.7E+3	7.5E+4	4.2E+3	3.0E-14
	16	64	4	16	6.6E+1	7.7E+1	3.7E+2	0.81	0.34	2.9E+3	1.7E+3	7.6E+4	9.0E+3	3.2E-14
		32	4	8	7.1E+1	7.7E+1	3.9E+2	0.83	0.35	2.0E+3	1.8E+3	7.6E+4	2.7E+3	3.1E-14
		16	4	4	6.8E+1	9.6E+1	3.6E+2	0.85	0.31	2.2E+3	1.7E+3	7.5E+4	1.7E+3	2.9E-14
2048	32	16	4	4	4.5E+1	5.8E+1	2.3E+1	0.87	0.36	1.0E+3	9.2E+2	2.7E+4	8.0E+1	1.5E-14
	16	32	4	8	4.9E+1	4.9E+1	2.6E+2	0.84	0.35	9.3E+2	9.3E+2	2.7E+4	1.2E+3	1.5E-14
		16	4	4	4.5E+1	5.6E+1	2.3E+2	0.87	0.36	1.0E+3	9.1E+2	2.8E+4		1.5E-14
1024	16	16	4	4	2.8E+1	3.8E+1	1.4E+2	0.88	0.36	4.9E+2	4.8E+2	1.0E+4	3.3E+3	7.4E-15
GEPP														
8192	-				5.5E+1	7.6E+1	3.0E+2	1	1	1.9E+3	2.6E+3	8.7E+3	6.0E+2	7.2E-14
4096	-				3.6E+1	5.1E+1	2.0E+2	1	1	1.0E+3	1.4E+3	2.3E+4	1.9E+2	3.9E-14
2048	-				2.6E+1	3.6E+1	1.4E+2	1	1	5.5E+2	7.4E+2	6.1E+4	1.8E+2	2.0E-14
1024	-				1.8E+1	2.5E+1	9.3E+1	1	1	2.8E+2	4.1E+2	1.6E+5	4.3E+2	1.1E-14

Table 11: Stability of the LU decomposition for GEPP on special matrices of size 4096.

	matrix	cond(A,2)	g_W	$\|\|L\|\|_1$	$\|\|L^{-1}\|\|_1$	$\max_{ij} \|U_{ij}\|$	$\min_{kk} \|U_{kk}\|$	cond(U,1)	$\frac{\|\|PA-LU\|\|_F}{\|\|A\|\|_F}$	η	w_b	N_{IR}
well-conditioned	hadamard	1.0E+0	4.1E+3	4.1E+3	4.1E+3	4.1E+3	1.0E+0	5.3E+5	0.0E+0	3.3E-16	4.6E-15	2
	house	1.0E+0	5.1E+0	8.9E+2	2.6E+2	5.1E+0	5.7E-2	1.4E+4	2.0E-15	5.6E-17	6.3E-15	3
	parter	4.8E+0	1.6E+0	4.8E+1	2.0E+0	3.1E+0	2.0E+0	2.3E+2	2.3E-15	8.3E-16	4.4E-15	3
	ris	4.8E+0	1.6E+0	4.8E+1	2.0E+0	1.6E+0	1.0E+0	2.3E+2	2.3E-15	7.1E-16	4.7E-15	2
	kms	9.1E+0	1.0E+0	2.0E+0	1.5E+0	1.0E+0	7.5E-1	3.0E+0	2.0E-16	1.1E-16	6.7E-16	1
	toeppen	1.0E+1	1.1E+0	2.1E+0	9.0E+0	1.1E+1	1.0E+1	3.3E+1	1.1E-17	7.2E-17	3.0E-15	1
	condex	1.0E+2	1.0E+0	2.0E+0	5.6E+0	1.0E+2	1.0E+0	7.8E+2	1.8E-15	9.7E-16	6.8E-15	3
	moler	1.9E+2	1.0E+0	2.2E+1	2.0E+0	1.0E+0	1.0E+0	4.4E+1	3.8E-14	2.6E-16	1.7E-15	2
	circul	3.7E+2	1.8E+2	1.0E+3	1.4E+3	6.4E+2	3.4E+0	1.2E+6	4.3E-14	2.1E-15	1.2E-14	1
	randcorr	1.4E+3	1.0E+0	3.1E+1	5.7E+1	1.0E+0	2.3E-1	5.0E+4	1.6E-15	7.8E-17	8.0E-16	1
	poisson	1.7E+3	1.0E+0	2.0E+0	3.4E+1	4.0E+0	3.2E+0	7.8E+1	2.8E-16	1.4E-16	7.5E-16	1
	hankel	2.9E+3	6.2E+1	9.8E+2	1.5E+3	2.4E+2	4.5E+0	2.0E+6	4.2E-14	2.5E-15	1.6E-14	2
	jordbloc	5.2E+3	1.0E+0	1.0E+0	1.0E+0	1.0E+0	1.0E+0	8.2E+3	0.0E+0	2.0E-17	8.3E-17	0
	compan	7.5E+3	1.0E+0	2.0E+0	4.0E+0	7.9E+0	2.6E-1	7.8E+1	0.0E+0	2.0E-17	6.2E-13	1
	pei	1.0E+4	1.0E+0	4.1E+3	9.8E+0	1.0E+0	3.9E-1	2.5E+1	7.0E-16	6.6E-18	2.3E-17	0
	randcolu	1.5E+4	4.6E+1	9.9E+2	1.4E+3	3.2E+0	5.6E-2	1.1E+7	4.0E-14	2.3E-15	1.4E-14	1
	sprandn	1.6E+4	7.4E+0	7.4E+2	1.5E+3	2.9E+1	1.7E+0	1.3E+7	3.4E-14	8.5E-15	9.3E-14	2
	riemann	1.9E+4	1.0E+0	4.1E+3	3.5E+0	4.1E+3	1.0E+0	2.6E+6	5.7E-19	2.0E-16	1.7E-15	2
	compar	1.8E+6	2.3E+1	9.8E+2	1.4E+3	1.1E+2	3.1E+0	2.7E+7	2.3E-14	1.2E-15	8.8E-15	1
	tridiag	6.8E+6	1.0E+0	2.0E+0	1.5E+3	2.0E+0	1.0E+0	5.1E+3	1.4E-18	2.6E-17	1.2E-16	0
	chebspec	1.3E+7	1.0E+0	5.4E+1	9.2E+0	7.1E+6	1.5E+3	4.2E+7	1.8E-15	2.9E-18	1.6E-15	1
	lehmer	1.8E+7	1.0E+0	1.5E+3	2.0E+0	1.0E+0	4.9E-4	8.2E+3	1.5E-15	2.8E-17	1.7E-16	0
	toeppd	2.1E+7	1.0E+0	4.2E+1	9.8E+2	2.0E+3	2.9E+2	1.3E+6	1.5E-15	5.0E-17	3.3E-16	1
	minij	2.7E+7	1.0E+0	4.1E+3	2.0E+0	1.0E+0	1.0E+0	8.2E+3	0.0E+0	7.8E-19	4.2E-18	0
	randsvd	6.7E+7	4.7E+0	9.9E+2	1.4E+3	6.4E-2	3.6E-7	1.4E+10	5.6E-15	3.4E-16	2.0E-15	2
	forsythe	6.7E+7	1.0E+0	1.0E+0	1.0E+0	1.0E+0	1.5E-8	6.7E+7	0.0E+0	0.0E+0	0.0E+0	0
ill-conditioned	fiedler	2.5E+10	1.0E+0	1.7E+3	1.5E+1	7.9E+0	4.1E-7	2.9E+8	1.6E-16	3.3E-17	1.0E-15	1
	dorr	7.4E+10	1.0E+0	2.0E+0	3.1E+2	3.4E+5	1.3E+0	1.7E+11	6.0E-18	2.3E-17	2.2E-15	1
	demmel	1.0E+14	2.5E+0	1.2E+2	1.4E+2	1.6E+14	7.8E+3	1.7E+17	3.7E-15	7.1E-21	4.8E-9	2
	chebvand	3.8E+19	2.0E+2	2.2E+3	3.1E+3	1.8E+2	9.0E-10	4.8E+22	5.1E-14	3.3E-17	2.6E-16	1
	invhess	4.1E+19	2.0E+0	4.1E+3	2.0E+0	5.4E+0	4.9E-4	3.0E+48	1.2E-14	1.7E-17	2.4E-14	(1)
	prolate	1.4E+20	1.2E+1	1.4E+3	4.6E+3	5.3E+0	5.9E-13	4.7E+23	1.6E-14	4.7E-16	6.3E-15	(1)
	frank	1.7E+20	1.0E+0	2.0E+0	2.0E+0	4.1E+3	5.9E-24	1.9E+30	2.2E-18	4.9E-27	1.2E-23	0
	cauchy	5.5E+21	1.0E+0	3.1E+2	1.9E+2	1.0E+7	2.3E-15	2.1E+24	1.4E-15	6.1E-19	5.2E-15	(1)
	hilb	8.0E+21	1.0E+0	3.1E+3	1.3E+3	1.0E+0	4.2E-20	2.2E+22	2.2E-16	6.0E-19	2.0E-17	0
	lotkin	5.4E+22	1.0E+0	2.6E+3	1.3E+3	1.0E+0	3.6E-19	2.3E+22	8.0E-17	3.0E-18	2.3E-15	(1)
	kahan	1.1E+28	1.0E+0	1.0E+0	1.0E+0	1.0E+0	2.2E-13	4.1E+53	0.0E+0	9.7E-18	4.3E-16	1

Table 12: Stability of the LU decomposition for binary tree based 1-level CALU on special matrices of size 4096 with P=64 and b= 8.

matrix	g_W	τ_{ave}	$\tau_{\min}$	$\|\|L\|\|_1$	$\|\|L^{-1}\|\|_1$	$\max_{ij}\|U_{ij}\|$	$\min_{kk}\|U_{kk}\|$	cond(U,1)	$\frac{\|\|PA-LU\|\|_F}{\|\|A\|\|_F}$	η	w_b	N_{IR}
hadamard	4.1E+3	1.00	1.00	4.1E+3	3.8E+3	4.1E+3	1.0E+0	1.2E+6	0.0E+0	2.9E-16	3.7E-15	2
house	5.1E+0	1.00	1.00	8.9E+2	2.6E+2	5.1E+0	5.7E-2	1.4E+4	2.0E-15	5.6E-17	6.8E-15	3
parter	1.6E+0	1.00	1.00	4.8E+1	2.0E+0	3.1E+0	2.0E+0	2.3E+2	2.3E-15	7.5E-16	4.1E-15	3
ris	1.6E+0	1.00	1.00	4.8E+1	2.0E+0	1.6E+0	1.0E+0	2.3E+2	2.3E-15	8.0E-16	4.2E-15	3
kms	1.0E+0	1.00	1.00	2.0E+0	1.5E+0	1.0E+0	7.5E-1	3.0E+0	2.0E-16	1.1E-16	5.9E-16	1
toeppen	1.1E+0	1.00	1.00	2.1E+0	9.0E+0	1.1E+1	1.0E+1	3.3E+1	1.1E-17	7.1E-17	1.3E-15	1
condex	1.0E+0	1.00	1.00	2.0E+0	5.6E+0	1.0E+2	1.0E+0	7.8E+2	1.8E-15	9.4E-16	4.8E-15	3
moler	1.0E+0	1.00	1.00	2.2E+1	2.0E+0	1.0E+0	1.0E+0	4.4E+1	3.8E-14	2.7E-16	1.8E-15	3
circul	2.3E+2	0.91	0.41	1.8E+3	1.7E+3	7.6E+2	3.1E+0	2.0E+6	5.7E-14	2.8E-15	1.6E-14	1
randcorr	1.0E+0	1.00	1.00	3.1E+1	5.7E+1	1.0E+0	2.3E-1	5.0E+4	1.6E-15	7.7E-17	7.7E-16	1
poisson	1.0E+0	1.00	1.00	2.0E+0	3.4E+1	4.0E+0	3.2E+0	7.8E+1	2.8E-16	1.4E-16	9.8E-16	1
hankel	9.3E+1	0.92	0.42	1.8E+3	1.7E+3	4.3E+2	2.5E+0	2.3E+6	5.3E-14	3.7E-15	2.2E-14	2
jordbloc	1.0E+0	1.00	1.00	1.0E+0	1.0E+0	1.0E+0	1.0E+0	8.2E+3	0.0E+0	2.0E-17	8.8E-17	0
compan	1.0E+0	1.00	1.00	2.0E+0	4.0E+0	7.9E+0	2.6E-1	7.8E+1	0.0E+0	9.9E-18	4.0E-14	1
pei	1.0E+0	1.00	1.00	4.1E+3	9.8E+0	1.0E+0	3.9E-1	2.5E+1	7.0E-16	3.6E-17	4.7E-17	0
randcolu	4.7E+1	0.91	0.40	2.1E+3	1.6E+3	3.8E+0	4.8E-2	1.4E+7	5.2E-14	2.9E-15	1.8E-14	2
sprandn	8.0E+0	0.93	0.41	1.2E+3	1.8E+3	3.6E+1	1.4E+0	2.4E+7	4.5E-14	9.6E-15	1.4E-13	2
riemann	1.0E+0	1.00	1.00	4.1E+3	5.1E+2	4.1E+3	1.0E+0	1.7E+8	2.5E-18	1.1E-16	1.4E-15	2
compar	3.5E+1	0.91	0.42	1.7E+3	1.6E+3	1.8E+2	2.8E+0	3.3E+7	3.0E-14	1.7E-15	1.1E-14	1
tridiag	1.0E+0	1.00	1.00	2.0E+0	1.5E+3	2.0E+0	1.0E+0	5.1E+3	1.4E-18	2.5E-17	1.1E-16	0
chebspec	1.0E+0	1.00	1.00	5.4E+1	9.2E+0	7.1E+6	1.5E+3	4.2E+7	1.8E-15	3.2E-18	1.6E-15	1
lehmer	1.0E+0	1.00	0.78	1.9E+3	5.0E+2	1.0E+0	4.9E-4	1.7E+6	1.5E-15	1.8E-17	9.3E-17	0
toeppd	1.0E+0	1.00	1.00	4.2E+1	9.8E+2	2.0E+3	2.9E+2	1.3E+6	1.5E-15	5.1E-17	4.3E-16	1
minij	1.0E+0	1.00	1.00	4.1E+3	2.0E+0	1.0E+0	1.0E+0	8.2E+3	0.0E+0	5.1E-19	3.5E-18	0
randsvd	8.3E+0	0.91	0.33	1.8E+3	1.6E+3	8.5E-2	3.0E-7	2.4E+10	7.4E-15	4.5E-16	2.5E-15	2
forsythe	1.0E+0	1.00	1.00	1.0E+0	1.0E+0	1.0E+0	1.5E-8	6.7E+7	0.0E+0	0.0E+0	0.0E+0	0
fiedler	1.0E+0	1.00	0.90	1.7E+3	1.5E+1	7.9E+0	4.1E-7	2.9E+8	1.6E-16	3.5E-17	6.4E-16	1
dorr	1.0E+0	1.00	1.00	2.0E+0	3.1E+2	3.4E+5	1.3E+0	1.7E+11	6.0E-18	2.6E-17	1.4E-15	1
demmel	2.8E+0	0.98	0.38	1.3E+2	1.3E+2	2.2E+14	6.2E+3	2.0E+17	3.8E-15	1.1E-20	9.8E-9	3
chebvand	3.1E+2	0.91	0.42	2.2E+3	3.4E+3	2.3E+2	8.4E-10	3.4E+23	6.6E-14	3.7E-17	3.2E-16	1
invhess	2.0E+0	1.00	1.00	4.1E+3	2.0E+0	5.4E+0	4.9E-4	3.0E+48	1.2E-14	1.2E-16	7.1E-14	(2)
prolate	1.7E+1	0.95	0.39	1.6E+3	5.8E+3	7.5E+0	6.6E-12	1.4E+23	2.0E-14	5.1E-16	9.1E-15	(1)
frank	1.0E+0	1.00	1.00	2.0E+0	2.0E+0	4.1E+3	5.9E-24	1.9E+30	2.2E-18	7.4E-28	1.8E-24	0
cauchy	1.0E+0	1.00	0.34	3.1E+2	2.0E+2	1.0E+7	2.3E-15	6.0E+24	1.4E-15	7.2E-19	7.4E-15	(1)
hilb	1.0E+0	0.92	0.37	3.2E+3	1.6E+3	1.0E+0	1.3E-19	1.8E+22	2.2E-16	5.5E-19	2.2E-17	0
lotkin	1.0E+0	0.93	0.48	2.7E+3	1.4E+3	1.0E+0	4.6E-19	7.5E+22	8.0E-17	2.2E-18	2.1E-16	0
kahan	1.0E+0	1.00	1.00	1.0E+0	1.0E+0	1.0E+0	2.2E-13	4.1E+53	0.0E+0	7.7E-18	2.1E-16	0

Table 13: Stability of the LU decomposition for binary tree based 2-level CALU on special matrices of size 4096 with P2=8, b2=32, P1=8, and b1=8.

matrix	g_W	τ_{ave}	$\tau_{\min}$	$\|\|L\|\|_1$	$\|\|L^{-1}\|\|_1$	$\max_{ij}\|U_{ij}\|$	$\min_{kk}\|U_{kk}\|$	cond(U,1)	$\frac{\|\|PA-LU\|\|_F}{\|\|A\|\|_F}$	η	w_b	N_{IR}
hadamard	6.67E+01	0.82	0.34	1.97E+03	1.76E+03	2.83E+02	2.98E+00	1.14E+07	6.37E-14	3.52E-15	2.20E-14	2
house	6.40E+01	0.82	0.33	2.06E+03	1.85E+03	3.08E+02	3.10E+00	1.52E+07	6.41E-14	3.77E-15	2.39E-14	2
parter	7.10E+01	0.82	0.34	2.12E+03	1.80E+03	2.83E+02	2.81E+00	9.71E+06	6.38E-14	3.52E-15	2.08E-14	2
ris	7.14E+01	0.82	0.30	2.41E+03	1.80E+03	3.55E+02	2.33E+00	8.34E+07	6.39E-14	3.52E-15	2.28E-14	2
kms	7.23E+01	0.82	0.33	2.02E+03	1.79E+03	3.06E+02	2.94E+00	1.78E+07	6.45E-14	3.68E-15	2.31E-14	2
toeppen	6.15E+01	0.82	0.31	2.13E+03	1.79E+03	3.23E+02	2.93E+00	1.78E+07	6.40E-14	3.71E-15	2.37E-14	2
condex	6.38E+01	0.83	0.35	2.22E+03	1.78E+03	3.10+02	3.03E+00	2.07E+07	6.39E-14	3.73E-15	2.26E-14	2
moler	6.22E+01	0.82	0.32	2.55E+03	1.79E+03	2.85E+02	2.97E+00	2.90E+07	6.43E-14	3.50E-15	2.27E-14	2
circul	8.03E+01	0.82	0.31	2.12E+03	1.79E+03	2.77E+02	2.44E+00	1.64E+08	6.35E-14	3.52E-15	2.09E-14	2
randcorr	6.51E+01	0.82	0.35	2.00E+03	1.80E+03	3.09E+02	2.82E+00	2.85E+07	6.54E-14	3.59E-15	2.12E-14	2
poisson	8.32E+01	0.82	0.31	2.18E+03	1.81E+03	3.06E+02	2.58E+00	1.03E+08	6.40E-14	3.68E-15	2.38E-14	2
hankel	7.11E+01	0.82	0.33	2.00E+03	1.79E+03	3.30E+02	2.99E+00	4.20E+07	6.38E-14	3.68E-15	2.23E-14	2
jordbloc	6.79E+01	0.82	0.32	2.08E+03	1.81E+03	2.89E+02	2.93E+00	9.34E+06	6.48E-14	3.48E-15	2.439E-14	2
compan	6.33E+01	0.82	0.34	2.06E+03	1.79E+03	2.82E+02	2.22E+00	1.01E+08	6.463E-14	3.58E-15	2.23E-14	2
pei	6.07E+01	0.82	0.34	2.18E+03	1.81E+03	3.11E+02	2.84E+00	3.29E+07	6.37E-14	3.45E-15	2.32E-14	2
randcolu	6.68E+01	0.82	0.33	2.246E+03	1.78E+03	2.87E+02	2.75E+00	9.05E+06	6.41E-14	3.70E-15	2.44E-14	2
sprandn	6.59E+01	0.82	0.30	2.19E+03	1.83E+03	2.97E+02	2.59E+00	1.49E+07	6.35E-14	3.56E-15	2.39E-14	2
riemann	6.63E+01	0.82	0.34	1.89E+03	1.84E+03	2.67E+02	3.25E+00	2.11E+07	6.43E-14	3.58E-15	2.24E-14	2
compar	7.32E+01	0.82	0.34	1.99E+03	1.78E+03	3.43E+02	2.92E+00	2.44E+07	6.40E-14	3.67E-15	2.37E-14	2
tridiag	7.24E+01	0.82	0.33	2.10E+03	1.79E+03	3.34E+02	2.84E+00	1.10E+08	6.49E-14	3.54E-15	2.41E-14	2
chebspec	7.41E+01	0.82	0.34	1.93E+03	1.78E+03	3.30E+02	2.21E+00	1.10E+08	6.43E-14	3.73E-15	2.37E-14	2
lehmer	6.70E+01	0.83	0.31	2.07E+03	1.80E+03	3.36E+02	3.00E+00	1.47E+07	6.41E-14	3.41E-15	2.33E-14	2
toeppd	6.37E+01	0.82	0.33	1.97E+03	1.79E+03	2.77E+02	3.10E+00	2.58E+07	6.41E-14	3.60E-15	2.39E-14	2
minij	6.41E+01	0.82	0.32	2.08E+03	1.83E+03	2.88E+02	3.09E+00	5.08E+07	6.36E-14	3.68E-15	2.33E-14	2
randsvd	6.44E+01	0.82	0.32	2.07E+03	1.83E+03	2.75E+02	2.98E+00	3.98E+07	6.38E-14	3.49E-15	2.30E-14	2
forsythe	6.92E+01	0.82	0.32	2.08E+03	1.80E+03	2.68E+02	2.43E+00	7.93E+07	6.47E-14	3.84E-15	2.56E-14	2
fiedler	6.57E+01	0.82	0.34	1.98E+03	1.77E+03	2.85E+02	2.87E+00	3.35E+07	6.51E-14	3.63E-15	2.43E-14	2
dorr	7.07E+01	0.82	0.31	2.08E+03	1.86E+03	3.11E+02	2.93E+00	9.91E+06	6.38E-14	3.60E-15	2.18E-14	2
demmel	6.36E+01	0.82	0.31	2.06E+03	1.79E+03	2.78E+02	3.02E+00	9.21E+06	6.38E-14	3.60E-15	2.46E-14	2
chebvand	6.54E+01	0.82	0.31	2.32E+03	1.81E+03	3.48E+02	2.70E+00	5.12E+07	6.41E-14	3.66E-15	2.21E-14	2
invhess	7.57E+01	0.82	0.35	2.02E+03	1.82E+03	3.57E+02	3.20E+00	2.38E+07	6.38E-14	3.51E-15	2.35E-14	2
prolate	6.82E+01	0.82	0.33	2.06E+03	1.80E+03	3.35E+02	2.89E+00	1.30E+07	6.47E-14	3.82E-15	2.49E-14	2
frank	6.03E+01	0.82	0.32	2.16E+03	1.77E+03	2.70E+02	3.29E+00	6.6E+07	6.42E-14	3.68E-15	2.37E-14	2
cauchy	6.29E+01	0.82	0.31	1.94E+03	1.79E+03	2.85E+02	3.10E+00	9.31E+06	6.43E-14	3.64E-15	2.33E-14	2
hilb	6.45E+01	0.82	0.30	2.15E+03	1.84E+03	2.86E+02	2.73E+00	4.30E+07	6.40E-14	3.50E-15	2.30E-14	2
lotkin	6.07E+01	0.82	0.33	2.07E+03	1.80E+03	3.12E+02	2.58E+00	8.03E+07	6.34E-14	3.61E-15	2.23E-14	2
kahan	6.40E+01	0.82	0.33	2.06E+03	1.85E+03	3.08E+02	3.10E+00	1.52E+07	6.41E-14	3.77E-15	2.39E-14	2

Table 14: Stability of the LU decomposition for GEPP on special matrices of size 8192.

matrix	g_W	$\|\|L\|\|_1$	$\|\|L^{-1}\|\|_1$	$\max_{ij}\|U_{ij}\|$	$\min_{kk}\|U_{kk}\|$	cond(U,1)	$\frac{\|\|PA-LU\|\|_F}{\|\|A\|\|_F}$	η	w_b	N_{IR}
hadamard	8.19E+03	8.19E+03	8.19E+03	8.19E+03	1.00E+00	1.59E+06	0.00E+00	3.53E-16	5.13E-15	2
house	1.45E+01	1.67E+03	1.07E+03	1.45E+01	4.31E-02	7.67E+04	1.72E-15	4.41E-17	9.38E-15	3
parter	1.57E+00	6.76E+01	1.98E+00	3.14E+00	2.00E+00	3.20E+02	3.24E-15	9.73E-16	6.19E-15	2.66
ris	1.57E+00	6.76E+01	1.98E+00	1.57E+00	1.00E+00	3.20E+02	3.24E-15	1.01E-15	5.74E-15	3
kms	1.00E+00	3.84E+00	1.50E+00	1.00E+00	6.31E-01	6.69E+00	2.74E-16	9.10E-17	5.54E-16	1
toeppen	1.10E+00	2.10E+00	9.00E+00	1.10E+01	1.00E+01	3.33E+01	6.95E-18	7.10E-17	1.37E-15	1.66
condex	1.00E+00	1.99E+00	5.60E+00	1.00E+02	1.00E+00	7.90E+02	2.59E-15	1.35E-15	8.61E-15	3
moler	1.00E+00	5.87E+03	2.56E+00	5.76E+00	1.54E+00	7.46E+21	4.46E-14	4.67E-19	2.56E-17	0
circul	5.53E+02	1.91E+03	2.66E+03	1.77E+03	3.73E+00	5.90E+06	5.11E-14	4.48E-15	3.09E-14	2
randcorr	1.00E+00	4.33E+01	8.29E+01	1.00E+00	1.06E-01	2.59E+07	2.20E-15	7.69E-17	7.20E-16	1
hankel	9.70E+01	1.93E+03	2.97E+03	3.83E+02	4.02E+00	1.62E+07	4.78E-14	4.12E-15	2.72E-14	2
jordbloc	1.00E+00	1.00E+00	1.00E+00	1.00E+00	1.00E+00	1.63E+04	0	1.94E-17	8.40E-17	0
compan	1.00E+00	1.98E+00	4.62E+00	1.13E+01	9.90E-02	3.52E+02	3.38E-19	1.95E-17	1.59E-12	1
pei	1.00E+00	5.62E+03	9.93E+00	1.59E+00	7.35E-01	2.32E+01	8.69E-16	1.19E-17	2.39E-17	0
randcolu	6.29E+01	1.92E+03	2.60E+03	3.49E+00	3.21E-02	1.05E+08	4.78E-14	4.06E-15	2.59E-14	2
sprandn	8.52E+00	1.49E+03	2.91E+03	3.62E+01	1.53E+00	2.50E+08	4.26E-14	1.79E-14	1.91E-13	2
riemann	1.00E+00	8.19E+03	3.50E+00	8.19E+03	1.00E+00	7.05E+06	2.62E-19	2.61E-16	2.51E-15	1.66
compar	6.20E+02	3.10E+03	4.02E+03	3.62E+03	2.35E+00	4.71E+08	1.84E-14	1.53E-15	1.11E-14	1.66
tridiag	1.00E+00	1.99E+00	3.01E+03	2.00E+00	1.00E+00	1.02E+04	1.00E-18	2.51E-17	1.16E-16	0
chebspec	1.04E+00	8.18E+01	9.50E+00	2.83E+07	3.01E+03	1.81E+08	6.03E-15	1.97E-18	2.31E-15	1.66
lehmer	1.00E+00	3.01E+03	1.99E+00	1.00E+00	2.44E-04	1.63E+04	2.15E-15	2.74E-17	1.70E-16	0
toeppd	1.00E+00	5.96E+01	3.41E+02	4.11E+03	6.47E+02	6.97E+05	2.16E-15	5.40E-17	4.50E-16	1
minij	1.00E+00	8.19E+03	2.00E+00	1.00E+00	1.00E+00	1.63E+04	0	4.16E-19	2.60E-18	0
randsvd	6.52E+00	1.91E+03	2.60E+03	6.19E-02	3.64E-07	4.57E+10	6.63E-15	6.52E-16	3.91E-15	2
forsythe	1.00E+00	1.00E+00	1.00E+00	1.00E+00	1.49E-08	6.71E+07	0	0	0	0
fiedler	1.22E+00	3.65E+03	1.12E+01	9.24E+00	1.02E-07	4.67E+09	2.06E-16	1.17E-16	1.92E-15	1
dorr	1.00E+00	2.00E+00	6.27E+02	1.34E+06	5.14E+00	7.06E+11	3.83E-18	2.36E-17	8.20E-16	1
demmel	2.20E+00	1.90E+02	2.80E+02	2.03E+14	9.33E+03	5.46E+17	2.72E-15	2.36E-20	1.57E-08	2
chebvand	3.53E+02	4.46E+03	7.04E+03	3.01E+02	9.04E-10	2.88E+22	6.17E-14	3.71E-17	3.05E-16	1
invhess	1.91E+00	8.19E+03	2.00E+00	5.87E+00	1.89E-04	3.56E+145	4.44E-16	7.99E-17	9.33E-12	1
prolate	1.48E+01	2.28E+03	1.21E+04	7.43E+00	1.48E-11	1.63E+22	1.64E-14	6.50E-16	1.19E-14	1
frank	1.00E+00	1.99E+00	1.99E+00	8.19E+03	1.46E-24	3.11E+31	1.99E-18	1.73E-27	9.07E-24	0
cauchy	1.00E+00	6.66E+02	3.82E+02	3.79E+07	2.11E-14	1.01E+25	3.68E-15	7.43E-19	8.24E-15	1.33
hilb	1.00E+00	5.66E+03	2.634E+03	1.00E+00	3.06E-19	1.16E+22	4.22E-16	5.75E-19	2.20E-17	0
lotkin	1.00E+00	5.10E+03	2.34E+03	1.00E+00	2.50E-19	8.74E+22	4.00E-17	4.38E-18	3.09E-15	1.66
kahan	1.00E+00	1.00E+00	1.00E+00	1.00E+00	2.22E-13	1.62E+52	0	1.37E-17	2.85E-16	0.66

Table 15: Stability the LU decomposition for binary tree based 1-level CALU on special matrices of size 8192 with P= 256 and b=32.

matrix	g_W	τ_{ave}	$\tau_{\min}$	$\|\|L\|\|_1$	$\|\|L^{-1}\|\|_1$	$\max_{ij}\|U_{ij}\|$	$\min_{kk}\|U_{kk}\|$	cond(U,1)	$\frac{\|\|PA-LU\|\|_F}{\|\|A\|\|_F}$	η	w_b	N_{IR}
hadamard	8.19E+03	1	1	8.19E+03	2.20E+03	8.19E+03	1.00E+00	2.59E+06	0	7.98E-16	9.25E-15	2
house	1.45E+01	1	1	1.67E+03	1.07E+03	1.45E+01	4.31E-02	7.67E+04	1.70E-15	4.42E-17	9.43E-15	3
parter	1.57E+00	1	1	6.76E+01	1.98E+00	3.14E+00	2.00E+00	3.20E+02	3.24E-15	9.73E-16	6.19E-15	2.66
ris	1.57E+00	1	1	6.76E+01	1.98E+00	1.57E+00	1.00E+00	3.20E+02	3.24E-15	1.01E-15	5.74E-15	3
kms	1.00E+00	1	1	3.84E+00	1.50E+00	1.00E+00	6.31E-01	6.69E+00	2.17E-16	9.10E-17	5.57E-16	1
toeppen	1.10E+00	1	1	2.10E+00	9.00E+00	1.10E+01	1.00E+01	3.33E+01	7.48E-18	7.10E-17	1.37E-15	1.66
condex	1.00E+00	1	1	1.99E+00	5.60E+00	1.00E+02	1.00E+00	7.90E+02	2.59E-15	1.35E-15	7.78E-15	3
moler	1.00E+00	1	1	7.09E+03	4.20E+08	9.25E+03	1.27E+00	1.92E+24	1.71E-14	2.87E-20	9.54E-17	0
circul	4.64E+02	0.84	0.38	3.43E+03	3.361E+03	1.75E+03	3.31E+00	8.58E+06	1.26E-13	7.62E-15	4.70E-14	2
randcorr	1.00E+00	1	1	4.34E+01	8.26E+01	1.00E+00	1.30E-01	7.06E+05	2.19E-15	7.82E-17	7.17E-16	1
hline hankel	1.47E+02	0.84	0.39	3.46E+03	3.56E+03	5.03E+02	3.68E+00	9.36E+06	6.16E-14	7.09E-15	4.74E-14	2
jordbloc	1.00E+00	1	1	1.00E+00	1.00E+00	1.00E+00	1.00E+00	1.63E+04	0	1.94E-17	7.27E-17	0
compan	1.00E+00	1	1	1.98E+00	5.37E+00	3.07E+00	2.68E-01	8.08E+01	3.12E-19	4.31E-17	3.48E-13	1
pei	1.62E+00	1	1	8.08E+03	1.09E+01	1.97E+00	5.20E-01	4.34E+01	9.08E-16	8.44E-18	2.85E-17	0
randcolu	9.02E+01	0.84	0.37	3.95E+03	3.27E+03	5.82E+00	3.39E-02	8.98E+07	6.18E-14	6.45E-15	3.93E-14	2
sprandn	1.37E+01	0.86	0.36	2.58E+03	3.84E+03	5.31E+01	1.53E+00	4.05E+07	5.38E-14	3.04E-14	2.98E-13	2
riemann	1.00E+00	1	1	8.19E+03	4.06E+03	8.19E+03	1.00E+00	2.03E+09	2.37E-19	1.13E-16	1.38E-15	1.66
compar	9.92E+02	0.94	0.78	4.80E+03	4.27E+03	5.78E+03	3.03E+00	2.83E+07	1.18E-14	1.25E-15	7.94E-15	1.33
tridiag	1.22E+03	0.89	0.58	4.21E+03	4.96E+03	6.92E+03	2.91E+00	6.89E+07	4.58E-14	2.52E-15	1.69E-14	1.33
chebspec	1.00E+00	1	1	1.99E+00	3.01E+03	2.00E+00	1.00E+00	1.02E+04	1.00E-18	2.52E-17	1.19E-16	0
lehmer	1.00E+00	0.99	0.91	3.22E+03	3.34E+02	1.00E+00	2.44E-04	1.65E+06	2.12E-15	2.07E-17	1.19E-16	0
toeppd	1.00E+00	1	1	6.00E+01	1.99E+03	4.09E+03	6.02E+02	4.24E+06	2.14E-15	5.28E-17	4.31E-16	1
minij	1.00E+00	1	1	8.19E+03	2.00E+00	1.00E+00	1.00E+00	1.63E+04	0.00E+00	4.05E-19	2.62E-18	0
randsvd	1.41E+01	0.84	0.36	3.56E+03	3.29E+03	1.13E-01	2.91E-07	8.63E+10	8.65E-15	1.13E-15	6.50E-15	2
forsythe	1.00E+00	1	1	1.00E+00	1.00E+00	1.00E+00	1.49E-08	6.71E+07	0.00E+00	0.00E+00	0.00E+00	0
fiedler	1.00E+00	0.99	0.75	3.23E+03	1.52E+01	7.42E+00	1.07E-07	2.11E+09	1.91E-16	6.65E-17	1.33E-15	1
dorr	1.00E+00	1	1	2.00E+00	6.27E+02	1.346E+06	5.14E+00	7.06E+11	4.84E-18	2.35E-17	8.74E-16	1
demmel	2.19E+00	0.97	0.44	2.19E+02	2.68E+02	1.9E+14	8.07E+03	6.24E+17	7.68E-15	3.08E-20	1.76E-08	2
chebvand	5.56E+02	0.85	0.39	4.64E+04	8.05E+03	4.06E+02	4.47E-10	2.34E+24	1.47E-13	4.38E-17	3.58E-16	1
invhess	1.88E+00	1	1	8.19E+03	2.00E+00	5.80E+00	4.14E-04	5.52E+121	2.00E-14	7.46E-17	4.04E-14	1
prolate	1.48E+01	0.95	0.46	2.87E+03	1.50E+04	6.34E+00	8.09E-12	4.74E+23	3.39E-14	6.84E-16	9.33E-15	1
frank	1.00E+00	1	1	1.99E+00	1.99E+00	8.19E+03	1.46E-24	3.11E+31	1.68E-18	3.71E-28	1.76E-24	0
cauchy	1.00E+00	0.99	0.47	6.48E+02	4.22E+02	1.82E+07	1.66E-14	3.46E+25	3.27E-15	7.50E-19	8.70E-15	1.33
hilb	1.00E+00	0.87	0.42	8.01E+03	3.04E+03	1.00E+00	3.12E-19	2.80E+22	4.22E-16	5.52E-19	2.15E-17	0
lotkin	1.00E+00	0.89	0.35	5.67E+03	2.58E+03	1.00E+00	5.38E-19	3.57E+22	4.00E-17	4.36E-18	2.51E-15	1
kahan	1.00E+00	1	1	1.00E+00	1.00E+00	1.00E+00	2.22E-13	1.62E+52	0	1.20E-17	3.43E-16	0.66

Table 16: Stability of the LU decomposition for binary tree based 2-level CALU on special matrices of size 8192 with P2=16, b2 =32, P1=16, b1 =8 .

matrix	g_W	τ_{ave}	$\tau_{\min}$	$\|\|L\|\|_1$	$\|\|L^{-1}\|\|_1$	$\max_{ij}\|U_{ij}\|$	$\min_{kk}\|U_{kk}\|$	cond(U,1)	$\frac{\|\|PA-LU\|\|_F}{\|\|A\|\|_F}$	η	w_b	N_{IR}
hadamard	1.17E+02	0.81	0.32	4.23E+03	3.48E+03	4.68E+02	3.04E+00	4.93E+08	6.20E-14	6.65E-15	4.87E-14	2
house	8.85E+01	0.81	0.3	4.37E+03	3.49E+03	4.55E+02	3.02E+00	8.15E+07	6.34E-14	7.03E-15	4.78E-14	2
parter	1.16E+02	0.81	0.28	4.21E+03	3.46E+03	5.12E+02	3.10E+00	6.16E+07	6.30E-14	6.98E-15	4.48E-14	2
ris	1.19E+02	0.81	0.30	4.54E+03	3.47E+03	4.74E+02	2.94E+00	7.89E+07	6.30E-14	6.69E-15	4.45E-14	2
kms	1.04E+02	0.81	0.31	4.42E+03	3.48E+03	5.04E+02	1.64E+00	7.05E+08	6.28E-14	6.95E-15	4.39E-14	2
toeppen	1.01E+02	0.81	0.30	4.26E+03	3.53E+03	4.49E+02	3.20E+00	1.83E+08	6.25E-14	6.84E-15	4.57E-14	2
condex	9.75E+01	0.81	0.32	4.38E+03	3.49E+03	4.76E+02	2.82E+00	5.08E+07	6.18E-14	6.76E-15	4.28E-14	2
moler	1.09E+02	0.81	0.28	4.09E+03	3.45E+03	4.69E+02	3.34E+00	1.32E+09	6.25E-14	6.89E-15	4.39E-14	2
circul	1.07E+02	0.81	0.30	4.16E+03	3.51E+03	4.48E+02	3.12E+00	4.16E+07	6.25E-14	6.99E-15	4.54E-14	2
randcorr	1.21E+02	0.81	0.30	4.49E+03	3.45E+03	5.05E+02	2.84E+00	1.75E+08	6.33E-14	7.17E-15	4.86E-14	2
hankel	1.16E+02	0.81	0.28	4.45E+03	3.46E+03	5.46E+02	3.16E+00	5.06E+07	6.32E-14	6.93E-15	4.67E-14	2
jordbloc	9.75E+01	0.81	0.32	4.38E+03	3.49E+03	4.76E+02	2.82E+00	5.08E+07	6.18E-14	6.99E-15	4.66E-14	2
compan	1.13E+02	0.81	0.25	5.02E+03	3.49E+03	5.44E+02	3.08E+00	1.05E+08	6.34E-14	7.12E-15	4.84E-14	2
pei	9.80E+01	0.81	0.31	4.45E+03	3.49E+03	4.65E+02	2.94E+00	1.19E+08	6.34E-14	7.41E-15	4.72E-14	2
randcolu	1.00E+02	0.81	0.30	4.35E+03	3.45E+03	5.35E+02	3.17E+00	7.39E+07	6.28E-14	6.98E-15	4.46E-14	2
sprandn	9.48E+01	0.81	0.29	4.45E+03	3.48E+03	4.60E+02	2.81E+00	8.80E+07	6.38E-14	7.16E-15	4.75E-14	2
riemann	1.21E+02	0.81	0.31	4.06E+03	3.48E+03	5.19E+02	2.93E+00	7.67E+07	6.37E-14	7.02E-15	4.41E-14	2
compar	9.03E+01	0.81	0.29	4.45E+03	3.48E+03	4.68E+02	2.48E+00	1.18E+09	6.23E-14	6.79E-15	4.51E-14	2
tridiag	9.93E+01	0.81	0.29	4.67E+03	3.48E+03	4.53E+02	3.043E+00	6.20E+07	6.27E-14	6.71E-15	4.46E-14	2
chebspec	1.17E+02	0.81	0.33	4.29E+03	3.49E+03	4.72E+02	2.33E+00	1.13E+09	6.31E-14	6.69E-15	4.11E-14	2
lehmer	9.66E+01	0.81	0.32	4.04E+03	3.48E+03	4.27E+02	2.91E+00	9.33E+07	6.20E-14	6.93E-15	4.52E-14	2
toeppd	1.13E+02	0.81	0.29	4.73E+03	3.49E+03	5.18E+02	3.21E+00	4.18E+07	6.27E-14	7.31E-15	4.71E-14	2
minij	1.12E+02	0.81	0.33	3.99E+03	3.57E+03	5.42E+02	2.56E+00	1.07E+09	6.25E-14	6.81E-15	4.44E-14	2
randsvd	1.19E+02	0.81	0.30	4.64E+03	3.454E+03	5.56E+02	3.04E+00	3.54E+08	6.24E-14	6.99E-15	4.77E-14	2
forsythe	1.09E+02	0.81	0.28	4.66E+03	3.48E+03	4.96E+02	2.50E+00	2.03E+08	6.31E-14	6.89E-15	4.40E-14	1.66
fiedler	1.04E+02	0.81	0.31	4.25E+03	3.46E+03	4.49E+02	3.20E+00	1.20E+08	6.27E-14	6.92E-15	4.56E-14	2
dorr	9.75E+01	0.81	0.29	4.38E+03	3.49E+03	4.76E+02	2.82E+00	5.08E+07	6.18E-14	6.76E-15	4.28E-14	2
demmel	1.01E+02	0.81	0.31	4.37E+03	3.49E+03	4.24E+02	3.27E+00	6.26E+07	6.29E-14	7.00E-15	4.76E-14	2
chebvand	1.23E+02	0.81	0.29	3.92E+03	3.49E+03	5.14E+02	3.02E+00	5.40E+07	6.29E-14	6.91E-15	4.67E-14	2
invhess	1.20E+02	0.81	0.31	4.52E+03	3.49E+03	5.87E+02	3.01E+00	4.03E+07	6.35E-14	7.06E-15	5.02E-14	2
prolate	9.57E+01	0.81	0.32	4.26E+03	3.50E+03	4.59E+02	2.51E+00	2.32E+10	6.37E-14	6.69E-15	4.52E-14	2
frank	9.75E+01	0.81	0.32	4.38E+03	3.49E+03	4.76E+02	2.82E+00	5.08E+07	6.18E-14	6.76E-15	4.28E-14	2
cauchy	1.07E+02	0.81	0.30	4.49E+03	3.51E+03	4.81E+02	2.86E+00	1.76E+08	6.33E-14	7.21E-15	4.65E-14	1.66
hilb	9.38E+01	0.81	0.30	4.51E+03	3.53E+03	4.37E+02	2.87E+00	1.58E+08	6.25E-14	6.64E-15	4.42E-14	2
lotkin	1.16E+02	0.81	0.28	4.44E+03	3.50E+03	4.95E+02	3.02E+00	1.13E+08	6.29E-14	6.81E-15	4.30E-14	2
kahan	9.95E+01	0.81	0.31	4.46E+03	3.49E+03	4.47E+02	1.68E+00	4.52E+08	6.28E-14	7.06E-15	4.70E-14	2

Table 17: Stability of the LU decomposition for binary tree based 3-level CALU on special matrices of size 8192 with P3=16, b3 = 64, P2=4, b2 = 32, P1=4 ,and b1=8 .

matrix	g_W	τ_{ave}	$\tau_{\min}$	$\|\|L\|\|_1$	$\|\|L^{-1}\|\|_1$	$\max_{ij} \|U_{ij}\|$	$\min_{kk} \|U_{kk}\|$	$\frac{\|\|PA-LU\|\|_F}{\|\|A\|\|_F}$	η	w_b	N_{IR}
hadamard	8.19E+03	1	1	8.19E+03	7.87E+03	8.19E+03	1.00E+00	0.00E+00	3.06E-16	4.06E-15	2
house	4.70E+00	1	1	1.64E+03	3.25E+02	4.70E+00	4.38E-02	3.40E-15	6.46E-17	2.12E-14	3
parter	1.57E+00	1	1	6.76E+01	1.98E+00	3.14E+00	2.00E+00	3.24E-15	1.01E-15	6.73E-15	3
ris	1.57E+00	1	1	6.76E+01	1.98E+00	1.57E+00	1.00E+00	3.24E-15	9.97E-16	6.19E-15	2.66
kms	1.00E+00	1	1	5.05E+00	1.64E+00	1.00E+00	5.12E-01	4.04E-16	1.40E-16	5.59E-16	1
toeppen	1.10E+00	1	1	2.10E+00	9.00E+00	1.10E+01	1.00E+01	6.95E-18	7.01E-17	1.62E-15	2
condex	1.00E+00	1	1	1.99E+00	5.60E+00	1.00E+02	1.00E+00	2.59E-15	1.35E-15	8.39E-15	3
moler	1.00E+00	1	1	7.33E+03	3.48E+08	1.73E+00	1.00E+00	1.88E-14	4.55E-19	2.46E-17	0
circul	3.49E+02	0.78	0.32	4.24E+03	3.59E+03	1.35E+03	3.57E+00	1.48E-13	8.24E-15	5.48E-14	1.66
randcorr	1.00E+00	1	1	4.34E+01	7.99E+01	1.00E+00	1.92E-01	2.20E-15	7.76E-17	7.32E-16	1
hankel	1.78E+02	0.78	0.30	4.49+03	3.83E+03	6.01E+02	3.13E+00	6.92E-14	7.73E-15	4.86E-14	2
jordbloc	1.00E+00	1	1	1.00E+00	1.00E+00	1.00E+00	1.00E+00	0	2.00E-17	8.29E-17	0
compan	1.00E+00	1	1	1.98E+00	5.19E+00	2.38E+01	2.74E-01	2.24E-19	1.07E-17	3.11E-12	1
pei	1.00E+00	1	1	4.80E+03	9.74E+00	1.90E+00	9.02E-01	6.88E-16	2.69E-17	4.48E-17	0
randcolu	1.30E+02	0.78	0.31	4.68E+03	3.58E+03	7.01E+00	3.68E-02	7.03E-14	8.33E-15	6.07E-14	2
sprandn	1.97E+01	0.81	0.31	3.25E+03	4.13E+03	6.88E+01	1.56E+00	5.93E-14	3.36E-14	3.55E-13	2
riemann	1.00E+00	1	1	8.19E+03	2.04E+03	8.19E+03	1.00E+00	2.36E-19	1.58E-16	1.57E-15	2
compar	3.05E+03	1	1	5.58E+03	8.25E+03	1.79E+04	3.41E+00	9.74E-16	2.28E-17	1.18E-15	1
tridiag	1.00E+00	1	1	1.99E+00	3.01E+03	2.00E+00	1.00E+00	1.00E-18	2.53E-17	1.14E-16	0
chebspec	1.04E+00	1	1	8.18E+01	9.50E+00	2.83E+07	3.01E+03	6.03E-15	1.60E-18	3.22E-15	1.66
lehmer	1.00E+00	0.99	0.84	3.48E+03	7.51E+02	1.00E+00	2.44E-04	2.09E-15	2.29E-17	1.43E-16	0
toeppd	1.00E+00	1	1	6.04E+01	9.29E+02	4.09E+03	6.09E+02	2.15E-15	5.25E-17	3.95E-16	1
minij	1.00E+00	1	1	8.19E+03	2.00E+00	1.00E+00	1.00E+00	0	4.39E-19	2.89E-18	0
randsvd	1.54E+01	0.78	0.28	4.81E+03	3.59E+03	1.10E-01	2.57E-07	9.67E-15	1.37E-15	7.98E-15	2
forsythe	1.00E+00	1	1	1.00E+00	1.00E+00	1.00E+00	1.49E-08	0	0	0	0
fiedler	1.10E+00	0.99	0.61	3.90E+03	1.43E+01	8.40E+00	2.22E-08	1.93E-16	1.59E-16	2.94E-15	1
dorr	1.00E+00	1	1	2.00E+00	6.27E+02	1.34E+06	5.14E+00	3.83E-18	2.39E-17	5.74E-16	0.33
demmel	2.71E+00	0.95	0.32	2.92E+02	2.81E+02	2.21E+14	6.89E+03	3.00E-15	3.37E-20	1.32E-16	2
chebvand	6.65E+02	0.80	0.32	4.73E+03	9.47E+03	4.45E+02	8.29E-10	8.30E-14	4.48E-17	3.58E-16	1
invhess	1.88E+00	1	1	8.19E+03	2.00E+00	5.81E+00	3.65E-04	3.98E-16	2.91E-17	1.09E-14	0.66
prolate	1.82E+01	0.91	0.36	3.13E+03	1.93E+04	8.84E+00	2.47E-11	1.85E-14	7.58E-16	1.88E-14	1
frank	1.00E+00	1	1	1.99E+00	1.99E+00	8.19E+03	1.46E-24	1.99E-18	2.26E-28	1.25E-24	0
cauchy	1.00E+00	0.99	0.35	4.95E+02	4.42E+02	3.45E+08	2.13E-14	1.80E-15	8.66E-19	6.12E-15	1
hilb	1.00E+00	0.82	0.34	6.05E+03	3.08E+03	1.00E+00	1.69E-19	4.21E-16	6.86E-19	2.70E-17	0
lotkin	1.00E+00	0.85	0.31	6.74E+03	2.91E+03	1.00E+00	2.89E-19	4.00E-17	4.25E-18	3.92E-15	1.66
kahan	1.00E+00	1	1	1.00E+00	1.00E+00	1.00E+00	2.22E-13	0	9.18E-18	3.08E-16	1

Appendix B

.1 Cost analysis of LU_PRRP

Here we summarize the floating-point operation counts for the LU_PRRP algorithm performed on an input matrix A of size $m \times n$, where $m \geq n$. We first focus on the step k of the algorithm, that is we consider the k^{th} panel of size $(m-(k-1)b) \times b$. We first perform a Strong RRQR on the transpose of the considered panel, then update the trailing matrix of size $(m-kb) \times (n-kb)$, finally we perform GEPP on the diagonal block of size $b \times b$. We assume that $m-(k-1)b \geq b+1$. The Strong RRQR performs nearly as many floating-point operations as the QR with column pivoting. Here we consider that is the same, since in practice performing RRQR with column pivoting is enough to obtain the bound τ, and thus it is

$$Flops_{SRRQR,1block,stepk} = 2(m-(k-1)b)b^2 - \frac{2}{3}b^3.$$

If we consider the update step (2.3), then the flops count is

$$Flops_{update,stepk} = 2b(m-kb)(n-kb).$$

For the additional GEPP on the diagonal block, the flops count is

$$Flops_{gepp,stepk} = \frac{2}{3}b^3 + (n-kb)b^2.$$

Then the flops count for the step k is

$$Flops_{LU\_PRRP,stepk} = b^2(2m+n-3(k-1)b) - b^3 + 2b(m-kb)(n-kb).$$

This gives us an arithmetic operation count of

$$\begin{aligned} Flops_{LU\_PRRP}(m,n,b) &= \Sigma_{k=1}^{\frac{n}{b}} \left[b^2(2m+n-2(k-1)b-kb) + 2b(m-kb)(n-kb) \right], \\ Flops_{LU\_PRRP}(m,n,b) &= mn^2 + 2mnb + 2nb^2 - \frac{1}{2}n^2b - \frac{1}{3}n^3 \\ &\sim mn^2 - \frac{1}{3}n^3 + 2mnb - \frac{1}{2}n^2b. \end{aligned}$$

Then for a square matrix of size $n \times n$, the flops count is

$$Flops_{LU\_PRRP}(n,n,b) = \frac{2}{3}n^3 + \frac{3}{2}n^2b + 2nb^2 \sim \frac{2}{3}n^3 + \frac{3}{2}n^2b.$$

.2 Cost analysis of CALU_PRRP

Here we detail the performance model of the parallel version of the CALU_PRRP factorization performed on an input matrix A of size $m \times n$ where, $m \geq n$. We consider a 2D layout $P = P_r \times P_c$. We first focus on the panel factorization for the block LU factorization, that is the selection of the b pivot rows with the tournament pivoting

strategy. This step is similar to CALU except that the reduction operator is Strong RRQR instead of GEPP, then for each panel the amount of communication is the same as for TSLU:

$$\begin{aligned} \text{\# messages} &= \log P_r \\ \text{\# words} &= b^2 \log P_r \end{aligned}$$

However the floating-point operations count is different. We consider as in Appendix C that Strong RRQR performs as many flops as QR with columns pivoting, then the panel factorization performs the QR factorization with columns pivoting on the transpose of the blocks of the panel and $\log P_r$ reduction steps:

$$\begin{aligned} \text{\# flops} &= 2\frac{m-(k-1)b}{P_r}b^2 - \frac{2}{3}b^3 + \log P_r(2(2b)b^2 - \frac{2}{3}b^3) \\ &= 2\frac{m-(k-1)b}{P_r}b^2 + \frac{10}{3}b^3 \log P_r - \frac{2}{3}b^3 \end{aligned}$$

To perform the QR factorization without pivoting on the transpose of the panel, and the update of the trailing matrix:

– broadcast the pivot information along the rows of the process grid.

$$\begin{aligned} \text{\# messages} &= \log P_c \\ \text{\# words} &= b \log P_c \end{aligned}$$

– apply the pivot information to the original rows.

$$\begin{aligned} \text{\# messages} &= \log P_r \\ \text{\# words} &= \frac{nb}{P_c} \log P_r \end{aligned}$$

– Compute the block column L and broadcast it through blocks of columns

$$\begin{aligned} \text{\# messages} &= \log P_c \\ \text{\# words} &= \frac{m-kb}{P_r} b \log P_c \\ \text{\# flops} &= 2\frac{m-kb}{P_r}b^2 \end{aligned}$$

– broadcast the upper block of the permuted matrix A through blocks of rows

$$\begin{aligned} \text{\# messages} &= \log P_r \\ \text{\# words} &= \frac{n-kb}{P_c} b \log P_r \end{aligned}$$

– perform a rank-b update of the trailing matrix

$$\text{\# flops} = 2b\frac{m-kb}{P_r}\frac{n-kb}{P_c}$$

Thus to get the block LU factorization :

$$\begin{aligned} \text{\# messages} &= \frac{3n}{b}\log P_r + \frac{2n}{b}\log P_c \\ \text{\# words} &= (\frac{mn}{P_r} - \frac{1}{2}\frac{n^2}{P_r} + n)\log P_c + (nb + \frac{3}{2}\frac{n^2}{P_c})\log P_r \\ \text{\# flops} &= \frac{1}{P}(mn^2 - \frac{1}{3}n^3) + \frac{2}{3}nb^2(5\log P_r - 1) + \frac{b}{P_r}(4mn + 2nb - 2n^2) \\ &\sim \frac{1}{P}(mn^2 - \frac{1}{3}n^3) + \frac{2}{3}nb^2(5\log P_r - 1) + \frac{4}{P_r}(mn - \frac{n^2}{2})b \end{aligned}$$

Then to obtain the full LU factorization, for each $b \times b$ block, we perform the Gaussian elimination with partial pivoting and we update the corresponding trailing matrix of size $b \times n - kb$. During this additional step, we first perform GEPP on the diagonal block, broadcast pivot rows through column blocks (this broadcast can be done together with the previous broadcast of L, thus there is no additional message to send), apply pivots, and finally compute U.

$$\text{\# words} = n(1 + \tfrac{b}{2}) \log P_c$$

$$\text{\# flops} = \sum_{k=1}^{\frac{n}{b}} \left[\frac{2}{3}b^3 + \frac{n-kb}{P_c}b^2 \right] = \frac{2}{3}nb^2 + \frac{1}{2}\frac{n^2 b}{P_c}$$

Finally the total count is :

$$\begin{aligned}
\text{\# messages} &= \frac{3n}{b}\log P_r + \frac{2n}{b}\log P_c \\
\text{\# words} &= (\frac{mn}{P_r} - \frac{1}{2}\frac{n^2}{P_r} + \frac{nb}{2} + 2n)\log P_c + (nb + \frac{3}{2}\frac{n^2}{P_c})\log P_r \\
&\approx (\frac{mn}{P_r} - \frac{1}{2}\frac{n^2}{P_r})\log P_c + (nb + \frac{3}{2}\frac{n^2}{P_c})\log P_r \\
\text{\# flops} &= \frac{1}{P}(mn^2 - \frac{1}{3}n^3) + \frac{4}{P_r}(mn - \frac{n^2}{2})b + \frac{n^2 b}{2P_c} + \frac{10}{3}nb^2 \log P_r
\end{aligned}$$

.3 Cost analysis of 2-level CALU

In the following we estimate the performance of 2-level CALU. The input matrix is of size $m \times n$. It is distributed over a grid of $P_2 = P_{r_2} \times P_{c_2}$ nodes. Each node has $P_1 = P_{r_1} \times P_{c_1}$ processors. The block size is b_2 at the top level of parallelism, and b_1 at the deepest level of parallelism. We consider the runtime estimation of each iteration k of 2-level CALU. We note $m_k \times n_k$ the size of the active matrix at the top level of parallelism, where $m_k = m-(k-1)b_2$ and $n_k = n-(k-1)b_2$. We note CALU(rows, columns, processors, b) the routine that performs 1-level CALU on a matrix of size $rows \times columns$ with P processors and a panel of size b. We suppose that the communication between two nodes of the top level of parallelism is performed between two nodes of the deepest level of parallelism, which then broadcast or distribute the recieved chunks of data over the rest of the processors. For that reason we assume that the chunks of data transferred between nodes of the top level fit in the memory of the processors of the deepest level of parallelism. We note that this is a strong assumption considering a distributed memory model.

1. Compute 2-level TSLU of current panel. k
   - Communication
     - At the top level of parallelism:

$$\text{\# messages}_k = \log P_{r_2}$$

$$\text{\# words}_k = b_2^2 \log P_{r_2}$$

     - At the deepest level of parallelism:

$$\begin{aligned}
\text{\# messages}_k &= \text{\# messages}(CALU(\frac{m_k}{P_{r_2}}, b_2, P_1, b_1)) + \log P_{r_2} \cdot \text{\# messages}(CALU(2b_2, b_2, P_1, b_1)) \\
&\quad + \log P_1 \cdot (1 + \log P_{r_2}) \\
\text{\# words}_k &= \text{\# words}(CALU(\frac{m_k}{P_{r_2}}, b_2, P_1, b_1)) + \log P_{r_2} \cdot \text{\# words}(CALU(2b_2, b_2, P_1, b_1)) \\
&\quad + \frac{b_2^2}{P_1}\log P_1 \cdot (1 + \log P_{r_2})
\end{aligned}$$

– Floating point operations, this includes only reduction operations. The computation of the block column k of L will be detailed later.

$$\#\text{ flops}_k = \#\text{ flops}(CALU(\frac{m_k}{P_{r_2}}, b_2, P_1, b_1)) + \log P_{r_2} \cdot \#\text{ flops}(CALU(2b_2, b_2, P_1, b_1))$$

2. Broadcast the pivot information along the columns of the process grid.
   – Communication
     – At the top level of parallelism:

$$\#\text{ messages}_k = \log P_{c_2}$$

$$\#\text{ words}_k = b_2 \log P_{c_2}$$

     – At the deepest level of parallelism: collect the different chunks on one processor of the deepest level, then distribute the received chunk over the different processors of the deepest level.

$$\#\text{ messages}_k = \log P_1$$

$$\#\text{ words}_k = b_2 \log P_1$$

3. Apply the pivot information to remainder of the rows. We consider that this is performed using an all to all reduce operation.
   – Communication
     – At the top level of parallelism:

$$\#\text{ messages}_k = \log P_{r_2}$$

$$\#\text{ words}_k = \frac{n_k b_2}{P_{c_2}} \log P_{r_2}$$

     – At the deepest level of parallelism: distribute the received block among P_1 processors.

$$\#\text{ messages}_k = \log P_1$$

$$\#\text{ words}_k = \frac{n_k b_2}{P_1 P_{c_2}} \log P_1$$

4. Boradcast of the $b_2 \times b_2$ upper part of panel k of L along columns of the process grid owning block row k of U.
   – Communication
     – At the top level of parallelism:

$$\#\text{ messages}_k = \log P_{c_2}$$

$$\#\text{ words}_k = \frac{b_2^2}{2} \log P_{c_2}$$

     – At the deepest level of parallelism: distribute the received block among P_1 processors

$$\#\text{ messages}_k = \log P_1$$

$$\#\text{ words}_k = \frac{b_2^2}{2P_1} \log P_1$$

5. Compute block row k of U and block column k of L: matrix matrix multiplication using a grid of P_1 processors in each node.
   For the U factor, the multiplied blocks are of sizes $b_2 \times b_2$ and $b_2 \times \frac{n_k}{P_{c_2}}$. For the L factor, the multiplied blocks are of sizes $\frac{m_k}{P_{r_2}} \times b_2$ and $b_2 \times b_2$. To model these steps we consider recursive Cannon's algorithm.
   – Communication

$$\text{\# messages}_k = (\left\lceil \frac{n_k}{P_{c_2} b_2} \right\rceil + \left\lceil \frac{m_k}{P_{r_2} b_2} \right\rceil) 4\sqrt{P_1}$$

$$\text{\# words}_k = (\left\lceil \frac{n_k}{P_{c_2} b_2} \right\rceil + \left\lceil \frac{m_k}{P_{r_2} b_2} \right\rceil) \frac{4b_2^2}{\sqrt{P_1}}$$

   – Floating-point operations

$$\text{\# flops}_k = (\left\lceil \frac{n_k}{P_{c_2} b_2} \right\rceil + \left\lceil \frac{m_k}{P_{r_2} b_2} \right\rceil) \frac{2b_2^3}{P_1}$$

6. Broadcast the block column k of L along columns of the process grid.
   – Communication
   – At the top level of parallelism

$$\text{\# messages}_k = \log P_{c_2}$$

$$\text{\# words}_k = \frac{(m_k - b_2)b_2}{P_{r_2}} \log P_{c_2}$$

   – At the deepest level of parallelism

$$\text{\# messages}_k = \log P_1$$

$$\text{\# words}_k = \frac{(m_k - b_2)b_2}{P_{r_2} P_1} \log P_1$$

7. Broadcast the block row k of U along rows of the process grid.
   – Communication
   – At the top level of parallelism

$$\text{\# messages}_k = \log P_{r_2}$$

$$\text{\# words}_k = \frac{(n_k - b_2)b_2}{P_{c_2}} \log P_{r_2}$$

   – At the deepest level of parallelism

$$\text{\# messages}_k = \log P_1$$

$$\text{\# words}_k = \frac{(n_k - b_2)b_2}{P_{c_2} P_1} \log P_1$$

8. Perform the update of the trailing matrix: matrix matrix multiplication using a grid of P_1 processors in each node.
   The multiplied blocks are of sizes $\frac{(m_k-b_2)}{P_{r_2}} \times b_2$ and $b_2 \times \frac{(n_k-b_2)}{P_{c_2}}$. Here again we use recursive Cannon's algorithm to model the matrix matrix multiplication.
   – Communication

$$\text{\# messages}_k = \left\lceil \frac{m_k - b_2}{P_{r_2} b_2} \right\rceil \left\lceil \frac{n_k - b_2}{P_{c_2} b_2} \right\rceil 4\sqrt{P_1}$$

$$\text{\# words}_k = \left\lceil \frac{m_k - b_2}{P_{r_2} b_2} \right\rceil \left\lceil \frac{n_k - b_2}{P_{c_2} b_2} \right\rceil \frac{4b_2^2}{\sqrt{P_1}}$$

   – Floating-point operations

$$\text{\# flops}_k = \frac{m_k - b_2}{P_{r_2} b_2} \frac{n_k - b_2}{P_{c_2} b_2} \frac{2b_2^3}{P_1}$$

.4 Cost analysis of 2-recursive CALU

In the following we estimate the performance of 2-recursive CALU. The input matrix is of $m \times n$. It is distributed over a grid of $P_2 = P_{r_2} \times P_{c_2}$ nodes. Each node has $P_1 = P_{r_1} \times P_{c_1}$ processors. The block size is b_2 at the top level of recursion, and b_1 at the deepest level of recursion. We estimate the running time of each iteration k of 2-recursive CALU. We note $m_k \times n_k$ the size of the active matrix at the top level of recursion, where $m_k = m - (k-1)b_2$ and $n_k = n - (k-1)b_2$, and $m_s \times b_{2s}$ the size of the active matrix at the deepest level of recursion, where $m_s = m_k - (s-1)b_1$ and $b_{2s} = b_2 - (s-1)b_1$. Note that the computation of the factor U and the update of the global trailing matrix is performed in the same way as for 2-level CALU. Thus in the following, we only focus on the panel factorization.

1. Compute TSLU of current panel s of size $m_s \times b_1$.
   – Communication
     – At the top level of parallelism:

$$\text{\# messages}_s = \log P_{r_2}$$

$$\text{\# words}_s = b_1^2 \log P_{r_2}$$

     – At the deepest level of parallelism:

$$\text{\# messages}_s = \log P_{r_1}$$

$$\text{\# words}_s = b_1^2 \log P_{r_1}$$

   – Floating point operations for (b_2/b_1) panels s

$$\text{\# flops} = \frac{1}{P_r}(2m_k b_2 - b_2^2)b_1 + \frac{b_2 b_1^2}{3}(5\log P_r - 1)$$

   Note here that for the rest of the panel factorization, a communication performed along columns of the process grid is local, that is inside one node of level 2, while a communication along rows of the process grid involves both local and global communications, that is inside and between nodes of level 2.

2. Broadcast the pivot information along columns of the process grid.
   - Communication
     - At the deepest level of parallelism:

$$\#\text{ messages}_s = \log P_{c_1}$$

$$\#\text{ words}_s = b_1 \log P_{c_1}$$

3. Apply the pivot information to remainder of the rows. We consider that this is performed using an all to all reduce operation.
   - Communication
     - At the top level of parallelism:

$$\#\text{ messages}_s = \log P_{r_2}$$

$$\#\text{ words}_s = b_{2s} b_1 \log P_{r_2}$$

     - At the deepest level of parallelism: distribute the received block among P_1 processors

$$\#\text{ messages}_k = \log P_1$$

$$\#\text{ words}_k = \frac{b_{2s} b_1}{P_1} \log P_1$$

4. Boradcast of the $b_1 \times b_1$ upper part of panel s of L along columns of the process grid owning block row s of U
   - Communication
     - At the deepest level of parallelism:

$$\#\text{ messages}_s = \log P_{c_1}$$

$$\#\text{ words}_s = \frac{b_1^2}{2} \log P_{c_1}$$

5. Compute block row s of U
   - Floating-point operations

$$\#\text{ flops} = \frac{b_2^2 b_1}{2P_{c_1}}$$

6. Broadcast the block column s of L along columns of the process grid.
   - Communication
     - At the deepest level of parallelism

$$\#\text{ messages}_s = \log P_{c_1}$$

$$\#\text{ words}_s = \frac{(m_s - b_1) b_1}{P_r} \log P_{c_1}$$

7. Broadcast the block row s of U along rows of the process grid.

– Communication
  – At the top level of parallelism

$$\#\text{ messages}_s = \log P_{r_2}$$

$$\#\text{ words}_s = (b_{2s} - b_1)b_1 \log P_{r_2}$$

  – At the deepest level of parallelism

$$\#\text{ messages}_s = \log P_1$$

$$\#\text{ words}_s = \frac{(b_{2s} - b_1)b_1}{P_1} \log P_1$$

8. Perform the update of the trailing matrix
   – Floating-point operations

$$\#\text{ flops} = \frac{1}{P_r P_{c_1}}(m_k b_2^2 - \frac{b_2^3}{3})$$

Thus for the factorization of the current panel k we have,
– Communication
  – At the top level of parallelism

$$\#\text{ messages}_k = \frac{3b_2}{b_1} \log P_{r_2}$$

$$\#\text{ words}_k = (\frac{3b_2^2}{2} + b_2 b_1) \log P_{r_2}$$

  – At the deepest level of parallelism

$$\#\text{ messages}_k = \frac{3b_2}{b_1} \log P_{c_1} + \frac{b_2}{b_1} \log P_{r_1} + \frac{2b_2}{b_1} \log P_1$$

$$\#\text{ words}_k = (m_k b_2 - \frac{b_2^2}{2} + \frac{b_2 b_1}{2} + b_2) \log P_{c_1} + b_2 b_1 \log P_{r_1} + \frac{b_2^2}{P_1} \log P_1$$

– Floating-point operations

$$\#\text{ flops}_k = \frac{1}{P_r}(2m_k b_2 - b_2^2)b_1 + \frac{b_2 b_1^2}{3}(5 \log P_r - 1) + \frac{b_2^2 b_1}{2P_{c_1}} + \frac{1}{P_r P_{c_1}}(m_k b_2^2 - \frac{b_2^3}{3})$$

.5 Cost analysis of multilevel CALU

Here we estimate the running time of multilevel CALU using the same notations as in section 5.4. We especially give the detailed counts.

The panel factorization

We first evaluate the cost of the panel factorization once the pivot rows are selected, that is after the preprocessing step. This corresponds to the following term

$$\sum_{s=1}^{n/b_l} T_{\text{ML-LU-Fact}}\left(\frac{m-sb_l}{P_{r_l}}, \frac{n-sb_l}{P_{c_l}}, b_l, P_l, l\right).$$

– Arithmetic cost

The arithmetic cost corresponds to the computation of the block columns of L of size $(m-sb_l)\times b_l$ and the block rows of U of size $b_l \times (n-sb_l)$. For that we use multilevel Cannon algorithm detailed in section 5.3. Since ML-CANNON uses a square 2D grid, we assume a square processor grid topology at each level k of the hierarchy, that is $P_{r_k} = P_{c_k} = \sqrt{P_k}$. Each node P_{c_l} has to multiply two matrices of sizes $b_l \times b_l$ and $b_l \times (n - sb_l/P_{c_l})$. And each node P_{r_l} working on the panel has to multiply two matrices of sizes $(m - sb_l) \times b_l$ and $b_l \times b_l$. Since Cannon's algorithm applies to square matrices, we consider matrices of size $b_l \times b_l$, then the cost of this computation is,

$$\begin{aligned} & \textstyle\sum_{s=1}^{n/b_l} F_{\text{ML-LU-Fact}}\left(\frac{m-sb_l}{P_{r_l}}, \frac{n-sb_l}{P_{c_l}}, b_l, P_l, l\right) \\ \approx\ & \textstyle\sum_{s=1}^{n/b_l} \left(\left\lceil \frac{m-sb_l}{b_l\sqrt{P_l}} \right\rceil + \left\lceil \frac{n-sb_l}{b_l\sqrt{P_l}} \right\rceil\right) F_{\text{ML-CANNON}}\left(b_l, b_l, P_{l-1}, l-1\right) \\ =\ & \frac{(2mn-n^2)b_l}{\sqrt{P_l}\prod_{j=1}^{l-1} P_j} + \frac{n^2 b_l}{\sqrt{P_l}\prod_{j=1}^{l-1} P_j} = \frac{2mnb_l}{\sqrt{P_l}\prod_{j=1}^{l-1} P_j}. \end{aligned}$$

– Communication cost

We first consider the broadcast of the b_l pivots selected during the preprocessing step. These pivots are distributed over the diagonal nodes. For each level k, there are (b_l/b_k) processors of level k taking part in the broadcast of the pivot information, each sends a chunk of size b_k. In Figure 19, where we consider 3 levels of hierarchy, processors of level 1 participating in the broadcast are presented by the red blocks. We note that in our performance model, we prefer intra-node communication to inter-node communication. For example, the pivot information broadcast will be performed with respect to the communication scheme presented in Figure 19. We discard the extreme case presented in Figure 20, where all the communication are performed between processors belonging to different nodes of the topmost level.
Therefore the volume of data communicated during the broadcast of the pivot information along the rows at a given level k of the hierarchy is

$$\sum_{s=1}^{n/b_l} b_k \log\left(\prod_{j=k}^{l} P_{c_j}\right) = \frac{nb_k}{b_l}\log\left(\prod_{j=k}^{l} P_{c_j}\right) = \frac{nb_k}{b_l}\log\left(\prod_{j=k}^{l} \sqrt{P_j}\right).$$

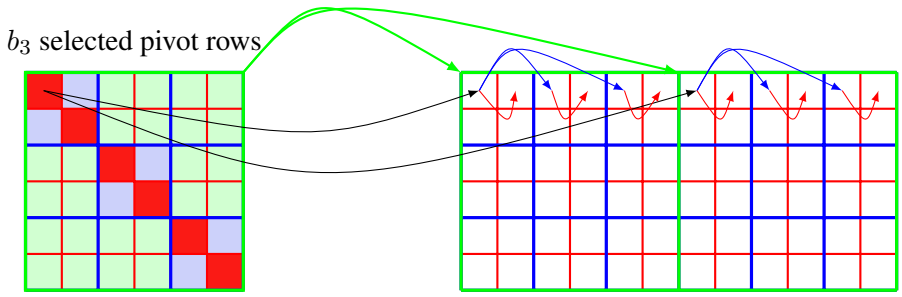


Figure 19: Broadcast of the pivot information along one row of processors (first scheme)

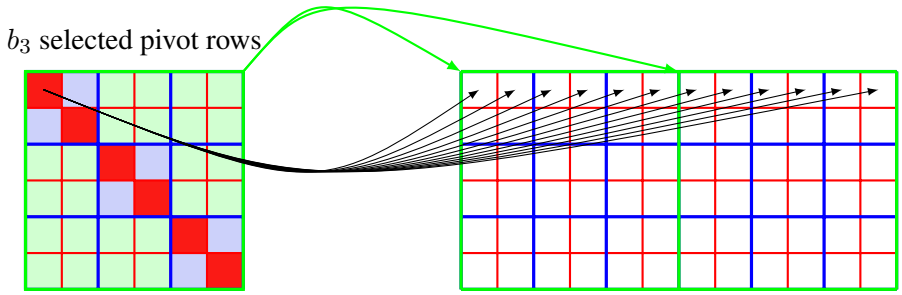


Figure 20: Broadcast of the pivot information along one row of processors (second scheme)

To apply the pivot information to the remainder of the rows, we consider an all to all reduce operation. Thus the amount of data to send by a node of a given level k of the hierarchy is $(nb_l)/(P_{c_l}\prod_{j=k}^{l-1}P_j)$. Then the volume of data to send at given level k of the hierarchy is

$$\sum_{s=1}^{n/b_l}\frac{nb_l}{P_{c_l}\prod_{j=k}^{l-1}P_j}\log P_{r_l}=\frac{n^2P_{r_l}}{\prod_{j=k}^{l}P_j}\log P_{r_l}=\frac{n^2\sqrt{P_l}}{\prod_{j=k}^{l}P_j}\log\sqrt{P_l}.$$

Now we consider the broadcast of the $b_l\times b_l$ upper part of L. There are $(P_{l-1})/2$ nodes of level $(l-1)$ participating in the broadcast. Then at a given level k, $1\leq k<l$, the volume of data to be sent is,

$$\sum_{s=1}^{n/b_l}\frac{b_l^2}{\prod_{j=k}^{l-1}P_j}\log P_{c_l}=\frac{nb_l}{\prod_{j=k}^{l-1}P_j}\log P_{c_l}=\frac{nb_l}{\prod_{j=k}^{l-1}P_j}\log\sqrt{P_l}.$$

If $k=l$, the volume of data communicated is,

$$\sum_{s=1}^{n/b_l}\frac{b_l^2}{2}\log P_{c_l}=\frac{nb_l}{2}\log P_{c_l}=\frac{nb_l}{2}\log\sqrt{P_l}.$$

For simplicity, we consider that the volume of data to send is the same for each level k of the hierarchy, $1\leq k\leq l$, and it is,

$$\frac{nb_lP_l}{\prod_{j=k}^{l}P_j}\log P_{c_l}=\frac{nb_lP_l}{\prod_{j=k}^{l}P_j}\log\sqrt{P_l}.$$

Here we evaluate the communication cost due to the computation of blocks L and U using multilevel Cannon. The volume of data to be sent at a given level k of the hierarchy is,

$$\sum_{s=1}^{n/b_l}\left(\left\lceil\frac{m-sb_l}{b_lP_{r_l}}\right\rceil+\left\lceil\frac{n-sb_l}{b_lP_{c_l}}\right\rceil\right)W_{\text{ML-CANNON}}\,(b_l,b_l,P_{l-1},l-1)\leq$$

$$\frac{2}{b_l\prod_{j=k}^{l}\sqrt{P_j}}\left(1+\frac{l-k}{\sqrt{P_k}}\right)\left(\frac{1}{P_{r_l}}(2mn-n^2)+\frac{n^2}{P_{c_l}}\right)=\frac{2}{b_l\prod_{j=k}^{l}\sqrt{P_j}}\left(1+\frac{l-k}{\sqrt{P_k}}\right)\frac{2mn}{\sqrt{P_l}}.$$

Finally the cost due to the volume of data communicated during the panel factorization once the pivot rows are selected is bounded by,

$$\sum_{k=1}^{l}\Big(\frac{nb_lP_l+n^2\sqrt{P_l}}{\prod_{j=k}^{l}P_j}\log\sqrt{P_l}+\frac{nb_k}{b_l}\log\left(\prod_{j=k}^{l}\sqrt{P_j}\right)+\frac{2\left(1+\frac{l-k}{\sqrt{P_k}}\right)}{b_l\prod_{j=k}^{l}\sqrt{P_j}}\frac{2mn}{\sqrt{P_l}}\Big)\beta_k.$$

The corresponding cost in terms of latency with respect to the network capacity of each level k of the hierarchy is bounded by,

$$\sum_{k=1}^{l}\Big(\frac{nb_lP_l+n^2\sqrt{P_l}}{\prod_{j=k}^{l}P_j}\log\sqrt{P_l}+\frac{nb_k}{b_l}\log\left(\prod_{j=k}^{l}\sqrt{P_j}\right)+\frac{2\left(1+\frac{l-k}{\sqrt{P_k}}\right)}{b_l\prod_{j=k}^{l}\sqrt{P_j}}\frac{2mn}{\sqrt{P_l}}\Big)\frac{\alpha_k}{B_k}.$$

The update of the trailing matrix

In the following we evaluate the cost of the update of the trailing matrix over the different levels of the hierarchy. This corresponds to the term $\sum_{s=1}^{n/b_l} T_{\text{ML-LU-Up}}\left(\frac{m-sb_l}{P_{r_l}},\frac{n-sb_l}{P_{c_l}},b_l,P_l,l\right)$ in the general recursive cost presented in section 5.4.

– Arithmetic cost

Here again we perform the update using multilevel Cannon algorithm and a square 2D grid of processors. Each node of level l multiplies two matrices of size $(m-sb_l)/P_{r_l}\times b_l$ and $(n-sb_l)/P_{c_l}\times b_l$ over $(l-1)$ levels of hierarchy at each step s of the factorization. Then the cost of the update on the critical path is,

$$\sum_{s=1}^{n/b_l}F_{\text{ML-LU-Up}}\left(\frac{m-sb_l}{P_{r_l}},\frac{n-sb_l}{P_{c_l}},b_l,P_l,l\right)\approx$$

$$\sum_{s=1}^{n/b_l}\left\lceil\frac{m-sb_l}{b_l\sqrt{P_l}}\right\rceil\left\lceil\frac{n-sb_l}{b_l\sqrt{P_l}}\right\rceil F_{\text{ML-CANNON}}\left(b_l,b_l,P_{l-1},l-1\right)\approx\frac{1}{P}\left(mn^2-\frac{n^3}{3}\right).$$

– Communication cost

The communication cost includes the broadcast of the block columns of L along columns of the process grid and the broadcast of the block rows of U along rows of the process grid. At the topmost level of parallelism the volume of data which needs to be sent for a given panel at iteration s is,

$$\frac{(m-sb_l)b_l}{P_{r_l}}\log P_{c_l}+\frac{(n-sb_l)b_l}{P_{c_l}}\log P_{r_l}.$$

Each node of level k broadcasts a chunk of data of size $((n-sb_l)b_l)/(P_{c_l}\prod_{j=k}^{l-1}P_j)$ along the homologous row processors, and a chunk of data of size $((m-sb_l)b_l)/(P_{r_l}\prod_{j=k}^{l-1}P_j)$ along the homologous column processors. Then the volume of data to send at a given level k of the hierarchy is

$$\sum_{s=1}^{n/b_l}\frac{(m-sb_l)b_lP_{c_l}}{\prod_{j=k}^{l}P_j}\log P_{c_l}+\frac{(n-sb_l)b_lP_{r_l}}{\prod_{j=k}^{l}P_j}\log P_{r_l}$$

$$\approx\frac{1}{\prod_{j=k}^{l}P_j}\left((mn-\frac{n^2}{2})P_{c_l}\log P_{c_l}+\frac{n^2}{2}P_{r_l}\log P_{r_l}\right)$$

$$\approx\frac{mn\sqrt{P_l}}{\prod_{j=k}^{l}P_j}\log\sqrt{P_l}.$$

Regarding the communication due to the matrix multiplication performed using multilevel Cannon, the additional volume of data communicated at a given level k of the hierarchy is,

$$\sum_{s=1}^{n/b_l}\left\lceil\frac{m-sb_l}{b_lP_{r_l}}\right\rceil\left\lceil\frac{n-sb_l}{b_lP_{c_l}}\right\rceil W_{\text{ML-CANNON}}\left(b_l,b_l,P_{l-1},l-1\right)$$

$$\leq \frac{2}{b_l P_l \prod_{j=1}^{l-1} \sqrt{P_j}}(mn^2 - \frac{n^3}{3})(1 + \frac{l-k}{\sqrt{P_k}}).$$

Finally the volume of data corresponding to the update of the trailing matrix along the critical path is bounded by,

$$\sum_{k=1}^{l} \left(\frac{mn\sqrt{P_l}}{\prod_{j=k}^{l} P_j} \log \sqrt{P_l} + \frac{2}{b_l P_l \prod_{j=k}^{l-1} \sqrt{P_j}}(mn^2 - \frac{n^3}{3})(1 + \frac{l-k}{\sqrt{P_k}}) \right) \beta_k.$$

The corresponding latency cost with respect to the network capacity of each level k of the hierarchy is bounded by

$$\sum_{k=1}^{l} \left(\frac{mn\sqrt{P_l}}{\prod_{j=k}^{l} P_j} \log \sqrt{P_l} + \frac{2}{b_l P_l \prod_{j=k}^{l-1} \sqrt{P_j}}(mn^2 - \frac{n^3}{3})(1 + \frac{l-k}{\sqrt{P_k}}) \right) \frac{\alpha_k}{B_k}.$$

The preprocessing step: selection of pivot rows

The cost of the preprocessing step corresponds to the following terms detailed in section 5.4,

$$\begin{aligned}
& \tfrac{n}{b_2} T_{\text{CALU}} \left(\tfrac{m}{\prod_{j=2}^{l} P_{r_j}}, b_2, b_1, P_1 \right) \\
+ \; & \tfrac{n}{b_2} \textstyle\sum_{r=3}^{l} 2^{l-r} \prod_{j=r}^{l} logP_{r_j} T_{\text{CALU}} \left(\tfrac{2b_r}{\prod_{j=2}^{r-1} P_{r_j}}, b_2, b_1, P_1 \right) \\
+ \; & \tfrac{n}{b_2} 2^{l-2} \textstyle\prod_{j=2}^{l} logP_{r_j} T_{\text{CALU}} \left(2b_2, b_2, b_1, P_1 \right) \\
+ \; & \textstyle\sum_{r=3}^{l} \tfrac{n}{b_r} T_{\text{FactUp-ML-CALU}} \left(\tfrac{m}{\prod_{j=r}^{l} P_{r_j}}, b_r, b_{r-1}, P_{r-1}, r-1 \right) \\
+ \; & \textstyle\sum_{k=4}^{l} \sum_{r=3}^{k-1} \tfrac{n}{b_r} 2^{l-k} \prod_{j=k}^{l} logP_{r_j} T_{\text{FactUp-ML-CALU}} \left(\tfrac{2b_k}{\prod_{j=r}^{k-1} P_{r_j}}, b_r, b_{r-1}, P_{r-1}, r-1 \right) \\
+ \; & \textstyle\sum_{r=3}^{l} \tfrac{n}{b_r} 2^{l-r} \prod_{j=r}^{l} logP_{r_j} T_{\text{FactUp-ML-CALU}} \left(2b_r, b_r, b_{r-1}, P_r, r-1 \right) \\
+ \; & \textstyle\sum_{r=2}^{l} \tfrac{n}{b_r} \prod_{j=r}^{l} logP_{r_j} \sum_{k=1}^{r} \tfrac{b_r^2 P_r}{\prod_{j=k}^{r} P_j} (\beta_k + \tfrac{\alpha_k}{B_k})
\end{aligned}$$

– Arithmetic cost

At a given level r of the recursive preprocessing step, $(r-1)$-level CALU is used as a redaction operator to select a set of b_r candidate pivot rows. At the deepest level of the recursion, the routine used to select pivot candidates from each block is 1-level CALU. Thus along all the recursion levels, from l down to 1, the number of computations performed to select the final set of pivot rows is given by the following contributions, where the three first terms correspond to the application of 1-level CALU and the three last terms present the panel factorizations and the updates performed during the preprocessing step,

$$\begin{aligned}
& \tfrac{n}{b_2} F_{\text{CALU}} \left(\tfrac{m}{\prod_{j=2}^{l} P_{r_j}}, b_2, b_1, P_1 \right) \\
+ \; & \tfrac{n}{b_2} \textstyle\sum_{r=3}^{l} 2^{l-r} \prod_{j=r}^{l} logP_{r_j} F_{\text{CALU}} \left(\tfrac{2b_r}{\prod_{j=2}^{r-1} P_{r_j}}, b_2, b_1, P_1 \right) \\
+ \; & \tfrac{n}{b_2} 2^{l-2} \textstyle\prod_{j=2}^{l} logP_{r_j} F_{\text{CALU}} \left(2b_2, b_2, b_1, P_1 \right) \\
+ \; & \textstyle\sum_{r=3}^{l} \tfrac{n}{b_r} F_{\text{FactUp-ML-CALU}} \left(\tfrac{m}{\prod_{j=r}^{l} P_{r_j}}, b_r, b_{r-1}, P_{r-1}, r-1 \right) \\
+ \; & \textstyle\sum_{k=4}^{l} \sum_{r=3}^{k-1} \tfrac{n}{b_r} 2^{l-k} \prod_{j=k}^{l} logP_{r_j} F_{\text{FactUp-ML-CALU}} \left(\tfrac{2b_k}{\prod_{j=r}^{k-1} P_{r_j}}, b_r, b_{r-1}, P_{r-1}, r-1 \right) \\
+ \; & \textstyle\sum_{r=3}^{l} \tfrac{n}{b_r} 2^{l-r} \prod_{j=r}^{l} logP_{r_j} F_{\text{FactUp-ML-CALU}} \left(2b_r, b_r, b_{r-1}, P_r, r-1 \right),
\end{aligned}$$

which can be bounded as follows,

$$\frac{n}{b_2}\prod_{j=2}^{l}(1+logP_{r_j})F_{\text{CALU}}\left(\frac{m}{\prod_{j=2}^{l}P_{r_j}},b_2,b_1,P_1\right)$$
$$+\sum_{r=3}^{l}\frac{n}{b_r}\left[\prod_{j=r}^{l}(1+logP_{r_j})F_{\text{FactUp-ML-CALU}}\left(\frac{m}{\prod_{j=r}^{l}P_{r_j}},b_r,b_{r-1},P_{r-1},r-1\right)\right].$$

To estimate the previous cost, we consider a square matrix and the "best" experimental panel size, that is for each level r of recursion these parameters can be written as $P_{r_r}=P_{c_r}=\sqrt{P_r}$ and $b_r=n/(8\times 2^{l-r}\prod_{j=r}^{l}(\sqrt{P_j}\log^2(\sqrt{P_j})))$. With respect to the previous parameters, we get the following upper bounds, where we assume that $P_r\geq 16$ at each level of parallelism,

$$\begin{aligned}\frac{n}{b_2}\prod_{j=2}^{l}(1+logP_{r_j})F_{\text{CALU}}\left(\frac{n}{\prod_{j=2}^{l}P_{r_j}},b_2,b_1,P_1\right) &\approx \frac{n^3}{P}(\frac{1}{2}+\frac{1}{log^2P_1})2^{l-1}\prod_{j=2}^{l}\frac{(1+\frac{1}{2}logP_j)}{log^2(P_j)}\\ &\leq \frac{9n^3}{16P}2^{l-1}\prod_{j=2}^{l}\frac{(1+\frac{1}{2}logP_j)}{log^2(P_j)}\\ &\leq \frac{9n^3}{16P}(\frac{3}{8})^{l-1}.\end{aligned}$$

$$\begin{aligned}&\sum_{r=3}^{l}\frac{n}{b_r}\prod_{j=r}^{l}(1+logP_{r_j})F_{\text{FactUp-ML-CALU}}\left(\frac{n}{\prod_{j=r}^{l}P_{r_j}},b_r,b_{r-1},P_{r-1},r-1\right)\\ &\approx\sum_{r=3}^{l}\frac{n^3}{P}(\frac{1}{2}+\frac{2}{log^2P_{r-1}})2^{l-r}\prod_{j=r}^{l}\frac{(1+\frac{1}{2}logP_j)}{log^2(P_j)}\\ &\leq\sum_{r=3}^{l}\frac{5n^3}{16P}\frac{3}{8}^{l-r+1}\\ &\leq\frac{5n^3}{16P}(l-2)(\frac{3}{8})^{l-2}.\end{aligned}$$

Therefore, the multilevel preprocessing step, that is the recursive selection of the pivot rows for the panel factorization is bounded as follows,

$$F_{\text{PreprocessingML-CALU}}(m,n,b_l,P_l)\leq\frac{n^3}{P}\Big(\frac{3}{8}\Big)^{l-2}\Big(\frac{5}{16}l-\frac{53}{128}\Big)<\frac{2n^3}{3P},(l\geq 2).$$

Thus, we can see that the preprocessing step is not dominating in terms of floating-point operations.

– Communication cost

The bandwidth cost of multilevel CALU preprocessing step is given by the following terms,

$$\begin{aligned}&\frac{n}{b_2}W_{\text{CALU}}\left(\frac{m}{\prod_{j=2}^{l}P_{r_j}},b_2,b_1,P_1\right)\\ +\;&\frac{n}{b_2}\sum_{r=3}^{l}2^{l-r}\prod_{j=r}^{l}logP_{r_j}W_{\text{CALU}}\left(\frac{2b_r}{\prod_{j=2}^{r-1}P_{r_j}},b_2,b_1,P_1\right)\\ +\;&\frac{n}{b_2}2^{l-2}\prod_{j=2}^{l}logP_{r_j}W_{\text{CALU}}(2b_2,b_2,b_1,P_1)\\ +\;&\sum_{r=3}^{l}\frac{n}{b_r}W_{\text{FactUP-ML-CALU}}\left(\frac{m}{\prod_{j=r}^{l}P_{r_j}},b_r,b_{r-1},P_{r-1},r-1\right)\\ +\;&\sum_{k=4}^{l}\sum_{r=3}^{k-1}\frac{n}{b_r}2^{l-k}\prod_{j=k}^{l}logP_{r_j}W_{\text{FactUP-ML-CALU}}\left(\frac{2b_k}{\prod_{j=r}^{k-1}P_{r_j}},b_r,b_{r-1},P_{r-1},r-1\right)\\ +\;&\sum_{r=3}^{l}\frac{n}{b_r}2^{l-r}\prod_{j=r}^{l}logP_{r_j}W_{\text{FactUP-ML-CALU}}(2b_r,b_r,b_{r-1},P_r,r-1)\\ +\;&\sum_{r=2}^{l}\frac{n}{b_r}\prod_{j=r}^{l}logP_{r_j}\sum_{k=1}^{r}\frac{b_r^2P_r}{\prod_{j=k}^{r}P_j}\beta_k,\end{aligned}$$

which can be bounded as follows,

$$\begin{aligned}&\frac{n}{b_2}\prod_{j=2}^{l}(1+logP_{r_j})W_{\text{CALU}}\left(\frac{n}{\prod_{j=2}^{l}P_{r_j}},b_2,b_1,P_1\right)\beta_1\\ +\;&\sum_{r=3}^{l}\frac{n}{b_r}\left[\prod_{j=r}^{l}(1+logP_{r_j})\sum_{k=1}^{r}W_{\text{FactUP-ML-CALU}}\left(\frac{n}{\prod_{j=r}^{l}P_{r_j}},b_r,b_{r-1},P_{r-1},k\right)\beta_k\right]\\ +\;&\sum_{r=2}^{l}\frac{n}{b_r}\prod_{j=r}^{l}logP_{r_j}\sum_{k=1}^{r}\frac{b_r^2P_r}{\prod_{j=k}^{r}P_j}\beta_k.\end{aligned}$$

The first term corresponds to bandwidth cost at the deepest level of the recursion, where we call 1-level CALU to select b_2 pivot rows from each block. With the parameters detailed above we have (lower order terms are omitted),

$$\frac{n}{b_2}\prod_{j=2}^{l}(1+logP_{r_j})W_{\text{CALU}}\left(\frac{n}{\prod_{j=2}^{l}P_{r_j}},b_2,b_1,P_1\right)\beta_1 \quad\approx\quad \frac{n^2}{2\prod_{j=1}^{l}\sqrt{P_j}}logP_1\prod_{j=2}^{l}(1+\tfrac{1}{2}logP_j)\beta_1$$

The second term presents the bandwidth cost of panel factorizations and updates performed during the preprocessing steps by calling r-level CALU, $3 \leq r \leq (l-1)$. The dominant cost in this term is that due to the updates, and it is bounded as follows,

$$\begin{aligned}
&\textstyle\sum_{r=3}^{l}\frac{n}{b_r}\left[\prod_{j=r}^{l}(1+logP_{r_j})\sum_{k=1}^{r}W_{\text{FactUP-ML-CALU}}\left(\frac{n}{\prod_{j=r}^{l}P_{r_j}},b_r,b_{r-1},P_{r-1},k\right)\beta_k\right]\\
&\approx\textstyle\sum_{r=3}^{l}\left[\prod_{j=r}^{l}(1+logP_{r_j})\sum_{k=1}^{r}\frac{2n^2}{P_r\prod_{j=k}^{l}\sqrt{P_j}}(1+\frac{r-k}{\sqrt{P_k}})\beta_k\right]\\
&\leq\textstyle\sum_{k=1}^{l}\frac{n^2}{8\prod_{j=k}^{l}\sqrt{P_j}}(l-2)(1+\frac{l}{4})\prod_{j=3}^{l}(1+\frac{1}{2}logP_j)\beta_k.
\end{aligned}$$

The last term present the cost of merging blocks two by two along the different recursion levels of the preprocessing steps. Using the parameters detailed above we have,

$$\textstyle\sum_{r=2}^{l}\frac{n}{b_r}\prod_{j=r}^{l}logP_{r_j}\sum_{k=1}^{r}\frac{b_r^2P_r}{\prod_{j=k}^{r}P_j}\beta_k \leq \sum_{k=1}^{l}\frac{n^2}{\prod_{j=k}^{l}\sqrt{P_j}}\frac{l-1}{4P_1\prod_{j=k}^{l}logP_j}\beta_k,$$

which is negligible comparing to the previous term. Thus, the bandwidth cost of multilevel CALU preprocessing step can be bounded by,

$$\begin{aligned}
&\frac{n^2}{2\prod_{j=1}^{l}\sqrt{P_j}}logP_1\prod_{j=2}^{l}(1+\tfrac{1}{2}logP_j)\beta_1\\
+\quad&\textstyle\sum_{k=1}^{l}\frac{n^2}{8\prod_{j=k}^{l}\sqrt{P_j}}(l-2)(1+\frac{l}{4})\prod_{j=3}^{l}(1+\frac{1}{2}logP_j)\beta_k.
\end{aligned}$$

Total cost of multilevel CALU

Here we give the total cost of multilevel CALU with l levels of recursion ($l \geq 2$), using a panel size $b_r = n/(8\times 2^{l-r}\prod_{j=r}^{l}(\sqrt{P_j}\log^2(\sqrt{P_j})))$ at each level r of recursion.

The computation time of multilevel CALU on the critical path is,

$$F_{\text{ML-CALU}}(m,n,b_l,P_l,l)\,\gamma \quad\approx\quad \left(\frac{2n^3}{3P}+\frac{n^3}{Plog^2P_l}+\frac{n^3}{P}\left(\frac{3}{8}\right)^{l-2}\left(\frac{5}{16}l-\frac{53}{128}\right)\right)\gamma.$$

The bandwidth cost of multilevel CALU, evaluated on the critical path, is bounded as follows (some lower order terms are omitted),

$$\begin{aligned}
\textstyle\sum_{k=1}^{l}W_{\text{ML-CALU}}(m,n,b_l,P_l,k)\,\beta_k \quad\leq\quad&\frac{n^2}{2\prod_{j=1}^{l}\sqrt{P_j}}logP_1\prod_{j=2}^{l}(1+\tfrac{1}{2}logP_j)\beta_1\\
+\quad&\textstyle\sum_{k=1}^{l}\Big[\frac{n^2}{\prod_{j=k}^{l}\sqrt{P_j}}\Big(\frac{8}{3}log^2P_l(1+\frac{l-k}{\sqrt{P_k}})\\
+\quad&\frac{(l-2)}{8}(1+\frac{l}{4})\prod_{j=3}^{l}(1+\tfrac{1}{2}logP_j)\Big)\Big]\beta_k.
\end{aligned}$$

Finally the latency cost of multilevel CALU with respect to network capacity of each level of hierarchy is bounded as follows,

$$\begin{aligned}\sum_{k=1}^{l} S_{\text{ML-CALU}}\,(m,n,b_l,P_l,k)\,\alpha_k &\leq \frac{n^2}{2\prod_{j=1}^{l}\sqrt{P_j}} logP_1 \prod_{j=2}^{l}(1+\tfrac{1}{2}logP_j)\,\tfrac{\alpha_1}{B_1} \\ &+ \sum_{k=1}^{l}\left[\frac{n^2}{\prod_{j=k}^{l}\sqrt{P_j}}\left(\tfrac{8}{3}log^2P_l(1+\tfrac{l-k}{\sqrt{P_k}})\right.\right. \\ &+ \left.\left.\tfrac{(l-2)}{8}(1+\tfrac{l}{4})\prod_{j=3}^{l}(1+\tfrac{1}{2}logP_j)\right)\right]\tfrac{\alpha_k}{B_k}.\end{aligned}$$

Appendix C

Here we detail the algebra of the block parallel LU-PRRP introduced in section 3.4. At the first iteration, the matrix A has the following partition,

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}.$$

The block A_{11} is of size $m/P \times b$, where P is the number of processors used assuming that $m/P \geq b+1$. The block A_{12} is of size $m/P \times n-b$, the block A_{21} is of size $(m-m/P) \times b$, and the block A_{22} is of size $(m-m/P) \times (n-b)$.

To describe the algebra, we consider 4 processors and a block of size b. We first focus on the first panel,

$$A(:, 1:b) = \begin{bmatrix} A_0 \\ A_1 \\ A_2 \\ A_3 \end{bmatrix}.$$

Each block A_i is of size $m/P \times b$. In the following we describe the different steps of the panel factorization. First, we perform Strong RRQR factorization on the transpose of each block A_i so we obtain,

$$\begin{aligned} A_0^T \Pi_{00} &= Q_{00} R_{00}, \\ A_1^T \Pi_{10} &= Q_{10} R_{10}, \\ A_2^T \Pi_{20} &= Q_{20} R_{20}, \\ A_3^T \Pi_{30} &= Q_{30} R_{30}. \end{aligned}$$

Each matrix R_i is of size $b \times m/P$. Using MATLAB notations, we can write R_i as follows,

$$\bar{R}_i = R_i(1:b, 1:b) \quad \bar{\bar{R}}_i = R_i(1:b, b+1:m/P)$$

This step aims at eliminating the last $m/P - b$ rows of each block A_i. We define the matrix $D_0 \Pi_0$ as follows,

$$D_0\Pi_0 = \begin{bmatrix} \begin{bmatrix} I_b \\ -D_{00} \quad I_{m/P-b} \end{bmatrix} & & & \\ & \begin{bmatrix} I_b \\ -D_{10} \quad I_{m/P-b} \end{bmatrix} & & \\ & & \begin{bmatrix} I_b \\ -D_{20} \quad I_{m/P-b} \end{bmatrix} & \\ & & & \begin{bmatrix} I_b \\ -D_{30} \quad I_{m/P-b} \end{bmatrix} \end{bmatrix} \times \begin{bmatrix} \Pi_{00}^T & & & \\ & \Pi_{10}^T & & \\ & & \Pi_{20}^T & \\ & & & \Pi_{30}^T \end{bmatrix},$$

where

$$\begin{aligned} D_{00} &= \bar{\bar{R}}_{00}^T (\bar{R}_{00}^{-1})^T, \\ D_{10} &= \bar{\bar{R}}_{10}^T (\bar{R}_{10}^{-1})^T, \\ D_{20} &= \bar{\bar{R}}_{20}^T (\bar{R}_{20}^{-1})^T, \\ D_{30} &= \bar{\bar{R}}_{30}^T (\bar{R}_{30}^{-1})^T. \end{aligned}$$

Multiplying $A(:, 1:b)$ by $D_0\Pi_0$ we obtain,

$$D_0\Pi_0 \times A(:,1:b) = \begin{bmatrix} (\Pi_{00}^T \times A_0)(1:b,1:b) \\ 0_{m/P-b} \\ (\Pi_{10}^T \times A_1)(1:b,1:b) \\ 0_{m/P-b} \\ (\Pi_{20}^T \times A_2)(1:b,1:b) \\ 0_{m/P-b} \\ (\Pi_{30}^T \times A_3)(1:b,1:b) \\ 0_{m/P-b} \end{bmatrix} = \begin{bmatrix} A_{01} \\ 0_{m/P-b} \\ A_{11} \\ 0_{m/P-b} \\ A_{21} \\ 0_{m/p-b} \\ A_{31} \\ 0_{m/P-b} \end{bmatrix}.$$

The second step corresponds to the second level of the reduction tree. We merge pairs of $b \times b$ blocks and as in the previous step we perform Strong RRQR factorization on the transpose of the $2b \times b$ blocks,

$$\begin{bmatrix} A_{01} \\ A_{11} \end{bmatrix}$$

and

$$\begin{bmatrix} A_{21} \\ A_{31} \end{bmatrix}.$$

We obtain,

$$\begin{bmatrix} A_{01} \\ A_{11} \end{bmatrix}^T \bar{\Pi}_{01} = Q_{01}R_{01},$$
$$\begin{bmatrix} A_{21} \\ A_{31} \end{bmatrix}^T \bar{\Pi}_{11} = Q_{11}R_{11}.$$

We note,

$$\bar{R}_{01} = R_{01}(1:b,1:b) \quad \bar{\bar{R}}_{01} = R_{01}(1:b,b+1:2b),$$
$$\bar{R}_{11} = R_{11}(1:b,1:b) \quad \bar{\bar{R}}_{11} = R_{11}(1:b,b+1:2b).$$

As in the previous level, we aim at eliminating b rows in each block. For that reason we consider the following matrix,

$$D_1\Pi_1 = \begin{bmatrix} \begin{bmatrix} I_b & & & \\ & I_{m/P-b} & & \\ -D_{01} & & I_b & \\ & & & I_{m/P-b} \end{bmatrix} & \begin{bmatrix} I_b & & & \\ & I_{m/P-b} & & \\ -D_{11} & & I_b & \\ & & & I_{m/P-b} \end{bmatrix} \end{bmatrix} \times \begin{bmatrix} \Pi_{01}^T & \\ & \Pi_{11}^T \end{bmatrix},$$

where

$$D_{01} = \bar{\bar{R}}_{01}^T(\bar{R}_{01}^{-1})^T,$$
$$D_{11} = \bar{\bar{R}}_{11}^T(\bar{R}_{01}^{-1})^T.$$

The matrices $\bar{\Pi}_{01}$ and $\bar{\Pi}_{11}$ are the permutations are obtained when performing the Strong RRQR factorizations on the two $2b \times b$ blocks. The matrices Π_{01} and Π_{11} can easily be deduced from the matrices $\bar{\Pi}_{01}$ and $\bar{\Pi}_{11}$ extended by the appropriate identity matrices to get matrices of size $2m/P \times 2m/P$. The multiplication of the block $D_0\Pi_0 A(:,1:b)$ with the matrix $D_1\Pi_1$ leeds to,

$$D_1\Pi_1 D_0\Pi_0 A(:,1:b) = \begin{bmatrix} \left[\bar{\Pi}_{01}^T \times \begin{bmatrix} A_{01} \\ A_{11} \end{bmatrix}\right](1:b,1:b) \\ 0_{2m/P-b} \\ \left[\bar{\Pi}_{11}^T \times \begin{bmatrix} A_{21} \\ A_{31} \end{bmatrix}\right](1:b,1:b) \\ 0_{2m/P-b} \end{bmatrix}.$$

This matrix can be written as,

$$D_1\Pi_1 D_0\Pi_0 A(:,1:b) = \begin{bmatrix} \left[\bar{\Pi}_{01}^T \times \begin{bmatrix} (\Pi_{00}^T \times A_0)(1:b,1:b) \\ (\Pi_{10}^T \times A_1)(1:b,1:b) \end{bmatrix}\right](1:b,1:b) \\ 0_{2m/P-b} \\ \left[\bar{\Pi}_{11}^T \times \begin{bmatrix} (\Pi_{20}^T \times A_2(1:b,1:b)) \\ (\Pi_{30}^T \times A_3(1:b,1:b)) \end{bmatrix}\right](1:b,1:b) \\ 0_{2m/P-b} \end{bmatrix} = \begin{bmatrix} A_{02} \\ 0_{2m/P-b} \\ A_{12} \\ 0_{2m/P-b} \end{bmatrix}.$$

For the final step we consider the last $2b \times b$ block

$$\begin{bmatrix} A_{02} \\ A_{12} \end{bmatrix}.$$

We perform a Strong RRQR factorization on the transpose of this block. Thus we obtain,

$$\begin{bmatrix} A_{02} \\ A_{12} \end{bmatrix}^T \bar{\Pi}_{02} = Q_{02}R_{02}.$$

We note

$$\bar{R}_{02} = R_{02}(1:b,1:b) \quad and \quad \bar{\bar{R}}_{02} = R_{02}(1:b,b+1:2b).$$

We define the matrix $D_2\Pi_2$ as,

$$D_2\Pi_2 = \begin{bmatrix} I_b & & & & & & & & \\ & I_{m/P-b} & & & & & & & \\ & & I_b & & & & & & \\ & & & I_{m/P-b} & & & & & \\ -D_{02} & & & & I_b & & & & \\ & & & & & I_{m/P-b} & & & \\ & & & & & & I_b & & \\ & & & & & & & I_{m/P-b} & \end{bmatrix} \times \Pi_{02}^T,$$

where

$$D_{02} = \bar{\bar{R}}_{02}^T(\bar{R}_{02}^{-1})^T,$$

and the permutation matrix Π_{02} can easily be deduced from the permutation matrix $\bar{\Pi}_{02}$.

The multiplication of the block $D_1\Pi_1 D_0\Pi_0 A(:,1:b)$ by the matrix $D_2\Pi_2$ leeds to,

$$D_2\Pi_2 D_1\Pi_1 D_0\Pi_0 A(:,1:b) = \begin{bmatrix} R_{021}^T Q_{02}^T \\ 0_{4m/P-b} \end{bmatrix} = \Pi_{02}^T \times \begin{bmatrix} A_{02} \\ 0_{4m/P-b} \end{bmatrix} = \begin{bmatrix} A_{03} \\ 0_{4m/P-b} \end{bmatrix}.$$

If we consider the block $A(:,1:b)$ of the beginning and all the steps performed, we get,

$$D_2\Pi_2 D_1\Pi_1 D_0\Pi_0 A(:,1:b) = \begin{bmatrix} \left[\bar{\Pi}_{02}^T \times \begin{bmatrix} A_{02} \\ A_{12} \end{bmatrix}(1:b,1:b)\right] \\ 0_{4m/P-b} \end{bmatrix},$$

which can can be written as,

$$D_2\Pi_2 D_1\Pi_1 D_0\Pi_0 A(:,1:b) = \begin{bmatrix} \begin{bmatrix} \bar{\Pi}_{02}^T \times \begin{bmatrix} \left[\bar{\Pi}_{01}^T \times \begin{bmatrix} (\Pi_{00}^T \times A_0)(1:b,1:b) \\ (\Pi_{10}^T \times A_1)(1:b,1:b) \end{bmatrix}\right](1:b,1:b) \\ \left[\bar{\Pi}_{11}^T \times \begin{bmatrix} (\Pi_{20}^T \times A_2)(1:b,1:b) \\ (\Pi_{30}^T \times A_3)(1:b,1:b) \end{bmatrix}\right](1:b,1:b) \end{bmatrix}(1:b,1:b) \\ 0_{4m/P-b} \end{bmatrix}.$$

Thus,

$$A(:,1:b) = (D_2\Pi_2 D_1\Pi_1 D_0\Pi_0)^{-1} \begin{bmatrix} \begin{bmatrix} \bar{\Pi}_{02}^T \times \begin{bmatrix} \left[\bar{\Pi}_{01}^T \times \begin{bmatrix} (\Pi_{00}^T \times A_0)(1:b,1:b) \\ (\Pi_{10}^T \times A_1)(1:b,1:b) \end{bmatrix}\right](1:b,1:b) \\ \left[\bar{\Pi}_{11}^T \times \begin{bmatrix} (\Pi_{20}^T \times A_2)(1:b,1:b) \\ (\Pi_{30}^T \times A_3)(1:b,1:b) \end{bmatrix}\right](1:b,1:b) \end{bmatrix}(1:b,1:b) \\ 0_{4m/P-b} \end{bmatrix},$$

$$A(:,1:b) = \Pi_0^T D_0^{-1} \Pi_1^T D_1^{-1} \Pi_2^T D_2^{-1} \begin{bmatrix} \begin{bmatrix} \bar{\Pi}_{02}^T \times \begin{bmatrix} \left[\bar{\Pi}_{01}^T \times \begin{bmatrix} (\Pi_{00}^T \times A_0)(1:b,1:b) \\ (\Pi_{10}^T \times A_1)(1:b,1:b) \end{bmatrix}\right](1:b,1:b) \\ \left[\bar{\Pi}_{11}^T \times \begin{bmatrix} (\Pi_{20}^T \times A_2)(1:b,1:b) \\ (\Pi_{30}^T \times A_3)(1:b,1:b) \end{bmatrix}\right](1:b,1:b) \end{bmatrix}(1:b,1:b) \\ 0_{4m/P-b} \end{bmatrix},$$

where

$$\Pi_0^T = \begin{bmatrix} \Pi_{00} & & & \\ & \Pi_{10} & & \\ & & \Pi_{20} & \\ & & & \Pi_{30} \end{bmatrix},$$

and

$$D_0^{-1} = \begin{bmatrix} \begin{bmatrix} I_b & \\ D_{00} & I_{m/P-b} \end{bmatrix} & & & \\ & \begin{bmatrix} I_b & \\ D_{10} & I_{m/P-b} \end{bmatrix} & & \\ & & \begin{bmatrix} I_b & \\ D_{20} & I_{m/P-b} \end{bmatrix} & \\ & & & \begin{bmatrix} I_b & \\ D_{30} & I_{m/P-b} \end{bmatrix} \end{bmatrix}.$$

For each Strong RRQR performed previously, the corresponding trailing matrix is updated at the same time. Thus, at the end of the process, we have

$$A = \Pi_0^T D_0^{-1} \Pi_1^T D_1^{-1} \Pi_2^T D_2^{-1} \times \begin{bmatrix} A_{03} & \hat{A}_{12} \\ 0_{4m/P-b} & \hat{A}_{22} \end{bmatrix},$$

where A_{03} is the $b \times b$ diagonal block containing the b selected pivot rows from the current panel. An additional GEPP should be performed on this block to get the full LU factorization. $\hat{A}_{22}$ is the trailing matrix already updated, and on which the block parallel LU-PRRP should be continued.

Appendix D

Here is the MATLAB code for generating the matrix T used in section 2.2.2 to define a generalized Wilkinson matrix on which GEPP fails. For any given integer $r > 0$, this generalized Wilkinson matrix is an upper triangular semi-separable matrix with rank at most r in all of its submatrices above the main diagonal. The entries are all negative and are chosen randomly. The Wilkinson matrix corresponds to the special case, where $r = 1$, and every entry above the main diagonal is 1.

```
function [A] = counterexample_GEPP(n,r,u,v);
%       Function counterexample_GEPP generates a matrix which fails
% GEPP in terms of large element growth.
%       This is a generalization of the Wilkinson matrix.
%
if (nargin == 2)
      u = rand(n,r);
      v = rand(n,r);
      A = - triu(u * v');
      for k = 2:n
          umax = max(abs(A(k-1,k:n))) * (1 + 1/n);
          A(k-1,k:n)  = A(k-1,k:n) / umax;
      end
      A = A - diag(diag(A));
      A = A' + eye(n);
      A(1:n-1,n) = ones(n-1,1);
else
      A = triu(u * v');
      A = A - diag(diag(A));
      A = A' + eye(n);
      A(1:n-1,n) = ones(n-1,1);
end
```

Bibliography

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

[2] BLAS. http://www.netlib.org/blas/.

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

[4] LAPACK. http://www.netlib.org/lapack/.

[5] Parallel computation models. http://www.cs.rice.edu/~vs3/comp422/lecture-notes/comp422-lec20-s08-v1.pdf.

[6] PLASMA. http://icl.cs.utk.edu/plasma/.

[7] ScaLAPACK. http://www.netlib.org/scalapack/.

[8] R.C. Agarwal, S. M. Balle, F. G. Gustavson, M. Joshi, and P. Palkar. A three-dimensional approach to parallel matrix multiplication. *IBM Journal of Research and Development*, 39:39–5, 1995.

[9] Alok Aggarwal, Ashok K. Chandra, and Marc Snir. Communication complexity of PRAMs. *Theor. Comput. Sci.*, 71(1):3–28, March 1990.

[10] E. Agullo, C. Coti, J. Dongarra, T. Herault, and J. Langem. QR factorization of tall and skinny matrices in a grid computing environment. In *Parallel Distributed Processing Symposium (IPDPS)*, pages 1–11. IEEE, 2010.

[11] Emmanuel Agullo, Jack Dongarra, Rajib Nath, and Stanimire Tomov. A Fully Empirical Autotuned Dense QR Factorization for Multicore Architectures. In Emmanuel Jeannot, Raymond Namyst, and Jean Roman, editors, *Euro-Par 2011 Parallel Processing*, volume 6853 of *Lecture Notes in Computer Science*, pages 194–205, Bordeaux, France, August 2011. Springer Berlin / Heidelberg.

[12] Deepak Ajwani and Henning Meyerhenke. Realistic computer models. In Matthias Müller-Hannemann and Stefan Schirra, editors, *Algorithm Engineering. Bridging the Gap between Algorithm Theory and Practice*, volume 5971 of *Lecture Notes in Computer Science*, pages 194–236. Springer-Verlag, 2010.

[13] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen. *LAPACK Users' Guide*. SIAM, Philadelphia, PA, USA, 1999.

[14] 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.

[15] Grey Ballard, James Demmel, Olga Holtz, and Oded Schwartz. Minimizing communication in numerical linear algebra. *SIAM J. Matrix Analysis Applications*, 32(3):866–901, 2011.

[16] D. W. Barron and H. P. F. Swinnerton-Dyer. Solution of Simultaneous Linear Equations using a Magnetic-Tape Store. *Computer Journal*, 3(1):28–33, 1960.

[17] Gianfranco Bilardi, Carlo Fantozzi, and Andrea Pietracaprina. On the effectiveness of d-bsp as a bridging model of parallel computation. In *In Proc. of the Int. Conference on Computational Science, LNCS 2074*, pages 579–588. Springer-Verlag, 2001.

[18] Gianfranco Bilardi, Kieran T. Herley, Andrea Pietracaprina, Geppino Pucci, and Paul Spirakis. Bsp vs logp. In *Proceedings of the eighth annual ACM symposium on Parallel algorithms and architectures*, SPAA '96, pages 25–32, New York, NY, USA, 1996. ACM.

[19] Christian H. Bischof and Charles Van Loan. The wy representation for products of householder matrices. In *PPSC*, pages 2–13, 1985.

[20] Christian H. Bischof and G. Quintana-Ortí. Computing rank-revealing QR factorizations of dense matrices. *ACM Trans. Math. Softw.*, 24(2):226–253, June 1998.

[21] Å. Björck and G. H. Golub. Iterative refinement of linear least squares solution by Householder transformation. *BIT*, 7:322–337, 1967.

[22] L.S. Blackford, J. Choi, A. Cleary, E. D'Azevedo, J. Demmel, I. Dhillon, J. Dongarra, S. Hammarling, G. Henry, A. Petitet, K. Stanley, D. Walker, and R.C. Whaley. Scalapack: A linear algebra library for message-passing computers. In *In SIAM Conference on Parallel Processing*, 1997.

[23] H.G. Bock and P. Krämer-Eis. A multiple shooting method for numerical computation of open and closed loop controls in nonlinear systems. In *Proceedings 9th IFAC World Congress Budapest*, 1984.

[24] G. Bosilca, A. Bouteiller, A Danalis, M. Faverge, H. Haidar, T. Herault, J. Kurzak, J. Langou, P. Lemarinier, H. Ltaief, P. Luszczek, A. YarKhan, and J Dongarra. Distributed-memory task execution and dependence tracking within dague and the dplasma project. Technical Report 232, LAPACK Working Note, September 2010.

[25] Alfredo Buttari, Julien Langou, Jakub Kurzak, and Jack Dongarra. Parallel tiled QR factorization for multicore architectures. *Concurrency and Computation: Practice and Experience*, 20(13):1573–1590, 2008.

[26] Alfredo Buttari, Julien Langou, Jakub Kurzak, and Jack Dongarra. A class of parallel tiled linear algebra algorithms for multicore architectures. *Parallel Comput.*, 35(1):38–53, January 2009.

[27] L. E. Cannon. *A cellular computer to implement the Kalman filter algorithm*. PhD thesis, Montana State University, 1969.

[28] F. Cappello, F. Desprez, M. Dayde, E. Jeannot, Y. Jegou, S. Lanteri, N. Melab, R. Namyst, P.V.B. Primet, O. Richard, et al. Grid5000: a nation wide experimental grid testbed. *International Journal on High Performance Computing Applications*, 20(4):481–494, 2006.

[29] 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.

[30] Ernie Chan. Runtime data flow scheduling of matrix computations. FLAME Working Note #39. Technical Report TR-09-22, The University of Texas at Austin, Department of Computer Sciences, August 2009.

[31] Ernie Chan, Robert van de Geijn, and Andrew Chapman. Managing the complexity of lookahead for lu factorization with pivoting. In *Proceedings of the 22nd ACM symposium on Parallelism in algorithms and architectures*, SPAA '10, pages 200–208, New York, NY, USA, 2010. ACM.

[32] Tony F. Chan and Per Christian Hansen. Some applications of the rank revealing QR factorization. *SIAM J. on Scientific and Statistical Computing*, 13(3):727–741, May 1992.

[33] David Culler, Richard Karp, David Patterson, Abhijit Sahay, Klaus Erik Schauser, Eunice Santos, Ramesh Subramonian, and Thorsten von Eicken. Logp: towards a realistic model of parallel computation. In *Proceedings of the fourth ACM SIGPLAN symposium on Principles and practice of parallel programming*, PPOPP '93, pages 1–12, New York, NY, USA, 1993. ACM.

[34] Frank Dehne, Andreas Fabri, and Andrew Rau-chaplin. Scalable parallel computational geometry for coarse grained multicomputers. *International Journal on Computational Geometry*, 6:298–307, 1994.

[35] Eliezer Dekel, David Nassimi, and Sartaj Sahni. Parallel matrix and graph algorithms. *SIAM J. Comput.*, 10(4):657–675, 1981.

[36] J. Demmel. *Applied Numerical Linear Algebra*. SIAM, Philadelphia, 1997.

[37] J. Demmel, L. Grigori, M. Hoemmen, and J. Langou. Communication-optimal parallel and sequential QR and LU factorizations. *Technical Report UCB/EECS-2008-89, University of California Berkeley, EECS Department, LAWN #204.*, 2008.

[38] James Demmel, Laura Grigori, Mark Hoemmen, and Julien Langou. Communication-optimal parallel and sequential QR and LU factorizations. *SIAM J. Scientific Computing*, 34(1), 2012.

[39] James Demmel, Laura Grigori, Mark Frederick Hoemmen, and Julien Langou. Communication-optimal parallel and sequential QR and LU factorizations. Technical Report UCB/EECS-2008-89, EECS Department, University of California, Berkeley, Aug 2008.

[40] James W. Demmel, Nicholas J. Higham, and Rob Schreiber. Block LU factorization. Technical Report 40, LAPACK Working Note, February 1992.

[41] S. Donfack, L. Grigori, and A. K. Gupta. Adapting communication-avoiding LU and QR factorizations to multicore architectures. In *IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, 2010.

[42] S. Donfack, L. Grigori, and A.K. Gupta. Adapting Communication-avoinding LU and QR factorizations to multicore architectures. *Proceedings of the IPDPS Conference*, 2010.

[43] Simplice Donfack, Laura Grigori, William D. Gropp, and Vivek Kale. Hybrid static/dynamic scheduling for already optimized dense matrix factorization. In *IPDPS*, pages 496–507, 2012.

[44] Simplice Donfack, Laura Grigori, and Alok Kumar Gupta. Adapting communication-avoiding LU and QR factorizations to multicore architectures. In *IPDPS*, pages 1–10, 2010.

[45] J. Dongarra, P. Luszczek, and A. Petitet. The LINPACK Benchmark: Past, Present and Future. *Concurrency: Practice and Experience*, 15:803–820, 2003.

[46] Jack Dongarra, Mathieu Faverge, Thomas Hérault, Julien Langou, and Yves Robert. Hierarchical QR factorization algorithms for multi-core cluster systems. In *26th IEEE International Parallel and Distributed Processing Symposium, IPDPS 2012, Shanghai, China, May 21-25, 2012*, pages 607–618. IEEE Computer Society, 2012.

[47] Agullo E., Demmel J., Dongarra J.and Hadri B.and Kurzak J., and Langou J.and Ltaief H.and Luszczek P.and Tomov S. Numerical linear algebra on emerging architectures: The plasma and magma projects. In *Journal of Physics: Conference Series*, volume Vol. 180, 2009.

[48] E. Elmroth and F. Gustavson. New serial and parallel recursive QR factorization algorithms for SMP systems. *Applied Parallel Computing Large Scale Scientific and Industrial Problems*, pages 120–128, 1998.

[49] L. V. Foster. The growth factor and efficiency of Gaussian elimination with rook pivoting. *J. Comput. Appl. Math.*, 86:177–194, 1997.

[50] Leslie V. Foster. Gaussian Elimination with Partial Pivoting Can Fail in Practice. *SIAM J. Matrix Anal. Appl.*, 15:1354–1362, 1994.

[51] M. Frigo, C. E. Leiserson, H. Prokop, and S. Ramachandran. Cache-oblivious algorithms. In *40th Annual Symposium on Foundations of Computer Science*, pages 285–297, 1999.

[52] Matteo Frigo, Charles E. Leiserson, Harald Prokop, and Sridhar Ramachandran. Cache-oblivious algorithms. In *Proceedings of the 40th Annual Symposium on Foundations of Computer Science*, FOCS '99, pages 285–, Washington, DC, USA, 1999. IEEE Computer Society.

[53] R. A. Van De Geijn and J. Watts. SUMMA: Scalable Universal Matrix Multiplication Algorithm. *Concurrency Practice and Experience*, 9(4):255–274, 1997.

[54] Gene H. Golub and Charles F. Van Loan. *Matrix Computations*. The Johns Hopkins University Press, 3rd edition, 1996.

[55] Gene H. Golub, Robert J. Plemmons, and Ahmed Sameh. High-speed computing: scientific applications and algorithm design. chapter Parallel block schemes for large-scale least-squares computations, pages 171–179. University of Illinois Press, Champaign, IL, USA, 1988.

[56] N. I. M. Gould. On growth in Gaussian elimination with complete pivoting. *SIAM J. Matrix Anal. Appl.*, 12:354–361, 1991.

[57] S. L. Graham, M. Snir, and C. A. Patterson. *Getting up to speed: The future of supercomputing*. National Academies Press, 2005.

[58] 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.

[59] Laura Grigori, James Demmel, and Hua Xiang. Communication avoiding Gaussian elimination. In *SC*, page 29, 2008.

[60] Laura Grigori, James Demmel, and Hua Xiang. CALU: A communication optimal LU factorization algorithm. *SIAM J. Matrix Analysis Applications*, 32(4):1317–1350, 2011.

[61] M. Gu and S. C. Eisenstat. Efficient Algorithms For Computing A Strong Rank Reaviling QR Factorization. *SIAM J. on Scientific Computing*, 17:848–896, 1996.

[62] F. G. Gustavson. Recursion leads to automatic variable blocking for dense linear-algebra algorithms. *IBM J. Res. Dev.*, 41(6):737–756, November 1997.

[63] Bilel Hadri, Hatem Ltaief, Emmanuel Agullo, and Jack Dongarra. Tile QR factorization with parallel panel processing for multicore architectures. In *IPDPS*, pages 1–10, 2010.

[64] Susanne E. Hambrusch. Models for parallel computation. In *ICPP Workshop*, pages 92–95, 1996.

[65] Nicholas J. Higham. *Accuracy and stability of numerical algorithms (2. ed.)*. SIAM, 2002.

[66] Nicholas J. Higham and D. J. Higham. Large Growth Factors in Gaussian Elimination with Pivoting. *SIMAX*, 10(2):155–164, 1989.

[67] Nicholas J. Higham and Desmond J. Higham. Large growth factors in Gaussian elimination with pivoting. *SIAM J. Matrix Analysis Applications*, 10(2):155–164, April 1989.

[68] Mark Hoemmen. *Communication-avoiding Krylov subspace methods*. PhD thesis, University of California at Berkeley, Berkeley, CA, USA, 2010. AAI3413388.

[69] 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.

[70] J.-W. Hong and H. T. Kung. I/O complexity: The red-blue pebble game. In *Proceedings of the thirteenth annual ACM symposium on Theory of computing*. ACM, 1981.

[71] Parry Husbands and Katherine A. Yelick. Multi-threading and one-sided communication in parallel lu factorization. In *SC*, page 31, 2007.

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

[73] P. Kumar. Cache oblivious algorithms. In U. Meyer, P. Sanders, and J. Sibeyn, editors, *Algorithms for Memory Hierarchies, LNCS 2625*, pages 193–212. Springer-Verlag, 2003.

[74] Jakub Kurzak, Hatem Ltaief, Jack Dongarra, and Rosa M. Badia. Scheduling dense linear algebra operations on multicore processors. *Concurrency and Computation: Practice and Experience*, 22(1):15–44, 2010.

[75] F. Thomson Leighton. *Introduction to parallel algorithms and architectures: array, trees, hypercubes*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1992.

[76] Richard J. Lipton, Walter A. Burkhard, Walter J. Savitch, Emily P. Friedman, and Alfred V. Aho, editors. *Proceedings of the 10th Annual ACM Symposium on Theory of Computing, May 1-3, 1978, San Diego, California, USA*. ACM, 1978.

[77] B.M. Maggs, L.R. Matheson, and R.E. Tarjan. Models of parallel computation: a survey and synthesis. In *System Sciences, 1995. Proceedings of the Twenty-Eighth Hawaii International Conference on System Sciences (HICSS)*, volume 2, pages 61–70 vol.2, 1995.

[78] William F. McColl and Alexandre Tiskin. Memory-efficient matrix multiplication in the bsp model. *Algorithmica*, 24(3-4):287–297, 1999.

[79] C. L. McCreary, J. J. Thompson, D. H. Gill, T. J. Smith, Y. Zhu, A. A. Khan, J. Thompson, and M. E. McArdle. A comparison of heuristics for scheduling dags on multiprocessors. Technical report, Auburn University, Auburn, AL, USA, 1993.

[80] A. Pothen and P. Raghavan. Distributed orthogonal factorization. In *Proceedings of the third conference on Hypercube concurrent computers and applications - Volume 2*, C3P, pages 1610–1620, New York, NY, USA, 1988. ACM.

[81] Gregorio Quintana-Ortí, Enrique S. Quintana-Ortí, Ernie Chan, Robert A. van de Geijn, and Field G. Van Zee. Design of scalable dense linear algebra libraries for multithreaded architectures: the LU factorization. In *IPDPS*, pages 1–8, 2008.

[82] R. D. Skeel. Iterative refinement implies numerical stability for Gaussian elimination. *Math. Comp.*, 35:817–831, 1980.

[83] Edgar Solomonik and James Demmel. Communication-optimal parallel 2.5d matrix multiplication and lu factorization algorithms. In *Euro-Par 2011 Parallel Processing - 17th International Conference*, pages 90–109, 2011.

[84] Fengguang Song, Hatem Ltaief, Bilel Hadri, and Jack Dongarra. Scalable tile communication-avoiding QR factorization on multicore cluster systems. In *Conference on High Performance Computing Networking, Storage and Analysis, SC 2010, New Orleans, LA, USA, November 13-19, 2010*, pages 1–11. IEEE, 2010.

[85] D. C. Sorensen. Analysis of pairwise pivoting in Gaussian elimination. *IEEE Transactions on Computers*, 3:274–278, 1985.

[86] D. C. Sorensen. Analysis of pairwise pivoting in gaussian elimination. *IEEE Trans. Comput.*, 34(3):274–278, March 1985.

[87] Adrian Soviani and Jaswinder Pal Singh. Estimating application hierarchical bandwidth requirements using bsp family models. In *IPDPS Workshops*, pages 914–923, 2012.

[88] G. W. Stewart. Gauss, statistics, and Gaussian elimination. *J. Comput. Graph. Statist.*, 4, 1995.

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

[90] Alexandre Tiskin. The bulk-synchronous parallel random access machine. *Theor. Comput. Sci.*, 196(1-2):109–130, 1998.

[91] S. Toledo. Locality of reference in LU decomposition with partial pivoting. *SIAM J. on Matrix Analysis and Applications*, 18(4):1065–1081, 1997.

[92] L. N. Trefethen and R. S. Schreiber. Average-case stability of Gaussian elimination. *SIAM J. Matrix Anal. Appl.*, 11(3):335–360, 1990.

[93] Leslie G. Valiant. A bridging model for parallel computation. *Commun. ACM*, 33(8):103–111, 1990.

[94] Leslie G. Valiant. A bridging model for multi-core computing. *J. Comput. Syst. Sci.*, 77(1):154–166, 2011.

[95] Uzi Vishkin, George C. Caragea, and Bryant Lee. Models for advancing PRAM and other algorithms into parallel programs for a PRAM-On-Chip platform. In *Technical Reports from UMIACS, UMIACS-UMIACS-TR-2006-21*, 2006/05/18/ 2006.

[96] J. H. Wilkinson. Error analysis of direct methods of matrix inversion. *J. Assoc. Comput. Mach.*, 8:281–330, 1961.

[97] J. H. Wilkinson. The algebric eigenvalue problem. *Oxford University Press*, 1985.

[98] James Hardy Wilkinson. Rounding errors in algebraic processes. In *IFIP Congress*, pages 44–53, 1959.

[99] S. J. Wright. A collection of problems for which Gaussian elimination with partial pivoting is unstable. *SIAM J. on Scientific and Statistical Computing*, 14:231–238, 1993.

[100] Ichitaro Yamazaki, Dulceneia Becker, Jack Dongarra, Alex Druinsky, Inon Peled, Sivan Toledo, Grey Ballard, James Demmel, and Oded Schwartz. Implementing a blocked aasen's algorithm with a dynamic scheduler on multicore architectures. Technical report, 2012. submitted to IPDPS 2013.

Publications

[A1] Simplice Donfack, Laura Grigori, and Amal Khabou. Avoiding Communication through a Multilevel LU Factorization. In *Euro-Par*, pages 551–562, 2012.

[A2] Amal Khabou, James Demmel, Laura Grigori, and Ming Gu. LU factorization with Panel Rank Revealing Pivoting and its Communication Avoiding version. Technical Report UCB/EECS-2012-15, EECS Department, University of California, Berkeley, Jan 2012. submitted to SIAM J. Matrix Analysis Applications (under revision).