2016 IEEE 55th Conference on Decision and Control (CDC)
ARIA Resort & Casino
December 12-14, 2016, Las Vegas, USA

Distributed Asynchronous Cholesky Decomposition

Gabriele Oliva*†, Roberto Setola†, and Christoforos N. Hadjicostis‡

***Abstract*— The *Cholesky decomposition* represents a fundamental building block in order to solve several matrix-related problems, ranging from matrix inversion to determinant calculation, and it finds application in several contexts, e.g., in Unscented Kalman Filters or other least-square estimation problems. In this paper we develop a distributed algorithm for performing the Cholesky decomposition of a sparse matrix. We model a network of n agents as a connected undirected graph, and we consider a symmetric positive definite $n \times n$ matrix $\mathbf{M}$ with the same structure as the graph (i.e., except for the diagonal entries, nonzero coefficients m_{ij} are allowed only if there is a link between the i-th and the j-th agent). We develop an asynchronous and distributed algorithm to let each agent i calculate the nonzero coefficients in the i-th column of the Cholesky decomposition of $\mathbf{M}$. With respect to the state of the art, the proposed algorithm does not require orchestrators and finds application in sparse networks with limited bandwidth and memory requirements at each node.**

I. Introduction

Being able to execute the *Cholesky decomposition* [1], [2] of a square positive definite matrix $\mathbf{M}$ in a distributed way, i.e., finding a lower triangular matrix $\mathbf{R}$ such that $\mathbf{M} = \mathbf{R}\mathbf{R}^T$, is a valuable achievement as it represents a fundamental building block to perform otherwise computationally expensive matrix-related operations.

For instance, the solution of a system of linear equations in the form $\mathbf{M}\mathbf{x} = \mathbf{b}$ is greatly simplified due to the triangular nature of $\mathbf{R}$; it is, in fact, sufficient to solve $\mathbf{R}\mathbf{y} = \mathbf{b}$ for $\mathbf{y}$ by forward substitution, and then to solve $\mathbf{R}^T\mathbf{x} = \mathbf{y}$ for $\mathbf{x}$ by backward substitution [2].

The Cholesky decomposition is extensively used in several complex control/estimation applications which would benefit of a distributed way to calculate $\mathbf{R}$; these include, among others, Unscented Kalman Filters [3]–[5], matrix inversion [6], solution of linear systems [7], or weighted least-square estimation problems [8]. Last but not least, the Cholesky decomposition greatly simplifies the calculation of the determinant of matrix $\mathbf{M}$, as it holds [2]

$$\det(\mathbf{M}) = \prod_{i=1}^{n} r_{ii}^2,$$

where r_{ii}^2 is the i-th diagonal entry of $\mathbf{R}$.

Such a property can be exploited, for instance, to calculate in a distributed way the *spanning betweenness* [9] of a node or edge (i.e., the number of spanning trees in G passing through a specific node or edge). This follows from the well known property that the number of spanning trees in an undirected graph coincides with the determinant of an $(n-1) \times (n-1)$ symmetric positive definite matrix $\mathbf{L}[i]$ obtained from the *Laplacian matrix* $\mathbf{L}$ of the graph by deleting the corresponding i-th row and i-th column [10]. Hence, it is possible to obtain the spanning betweenness of a specific node v_x or edge (v_x, v_y) by first calculating the total number of spanning trees (i.e., by calculating the determinant of $\mathbf{L}[i]$ for a given i) and then calculating the number of spanning trees that do not feature the node/edge of interest (i.e., by further removing the node v_x or the edge (v_x, v_y) and calculating the resulting determinant).

A. Contribution

In this paper, we consider a network of n agents, represented by a connected undirected graph, and an $n \times n$ symmetric and positive definite sparse matrix $\mathbf{M}$ with the same structure as the graph (i.e., except for the diagonal entries, nonzero coefficients m_{ij} are allowed only if there is a link between the i-th and the j-th agent). We provide an asynchronous and distributed algorithm to let each agent i calculate the nonzero coefficients in the i-th column of the Cholesky decomposition of $\mathbf{M}$, i.e., the unique lower triangular matrix $\mathbf{R}$ such that $\mathbf{M} = \mathbf{R}\mathbf{R}^T$ [1], [2].

B. Related Work

In the literature several parallel implementations of the Cholesky decomposition (or the closely related LU and QR decompositions) have been provided [11]–[13]. However, in distributed systems such as sensor networks, where the focus is more on distributing the computation load than on handling a distributed communication network or keeping memory and bandwidth low, there are few available solutions. Among others, in [14] (and references therein) the authors develop an approach based on a *distributing node*, i.e., a leader that assigns tasks to the agents; each task consists in the decomposition of a block of the $\mathbf{M}$ matrix, and the decomposition is carried out locally at the node. In this perspective, however, while the calculations are distributed, each node might need to solve a problem of possibly high dimension, and the nodes need to share a communication medium without constraints on their ability to communicate with each other; moreover, they must all be able to communicate with the distributing node.

C. Paper Outline

The paper outline is as follows: after some preliminary definitions, which conclude this introduction, Section II describes the *right-looking* approach to calculate the Cholesky

† Unit of Automatic Control, Department of Engineering, Università Campus Bio-Medico di Roma, via Álvaro del Portillo 21, 00128, Rome, Italy.
‡ Department of Electrical and Computer Engineering, University of Cyprus, 75 Kallipoleos Avenue, P.O. Box 20537, 1678 Nicosia, Cyprus.
* Corresponding author, email: g.oliva@unicampus.it

decomposition in a centralized fashion, while Section III develops the proposed asynchronous distributed algorithm; we analyze the characteristics of the proposed algorithm in Section IV and we provide some simulations in Section V; finally, we draw some conclusions and future work directions in Section VI.

D. Preliminaries

We denote vectors and matrices via boldface letters; we denote the (i,j)-th entry of a matrix $\mathbf{A}$ by a_{ij} or by $(\mathbf{A})_{ij}$. We represent a *message* containing a set of variables $\alpha_1, \dots \alpha_n$ by $< \alpha_1, \dots \alpha_n >$. Moreover, we represent by $\alpha == \beta$ the evaluation of the equality of the value of two variables α and β and by $\alpha \leftarrow \beta$ the assignment to a variable α of the value of a variable β. Let $G = \{V, E\}$ be a *graph* with n nodes $V = \{v_1, v_2, \dots, v_n\}$ and e edges $E \subseteq V \times V$, where $(v_i, v_j) \in E$ captures the existence of a communication link from node v_i to node v_j. A graph is said to be *undirected* if $(v_i, v_j) \in E$ whenever $(v_j, v_i) \in E$, and is said to be *directed* otherwise. In this paper, we focus on undirected graphs.

A *path* P over a graph $G = \{V, E\}$ from node $v_i = v_{i_1}$ to node $v_j = v_{i_t}$ is a sequence of nodes $v_{i_1}, \dots, v_{i_t}$ such that $(v_{i_k}, v_{i_{k+1}}) \in E$ for $k = 1, 2, ..., t-1$. An undirected graph is *connected* if for each pair of nodes $v_i, v_j \in V$ there is a path over G that connects them.

Let the *neighborhood* $\mathcal{N}_i$ of node $v_i \in V$ be the set of nodes $v_j \in V$ such that $(v_j, v_i) \in E$. The *degree* d_i of a node v_i is the number of its edges, i.e., $d_i = |\mathcal{N}_i|$.

II. Centralized Approach

In the literature, several (centralized) ways to calculate the Cholesky decomposition have been provided; in this paper, we consider the method used within the Cholesky–Banachiewicz and Cholesky–Crout algorithms (see, for instance [2]), which consists in the following recursive equations

$$r_{jj} = \sqrt{m_{jj} - \sum_{k=1}^{j-1} r_{jk}^2}, \quad (1)$$

$$r_{ij} = \frac{1}{r_{jj}} \Big(m_{ij} - \sum_{k=1}^{j-1} r_{ik} r_{jk} \Big), \quad \text{for } i > j. \quad (2)$$

There are several ways to calculate the Cholesky decomposition based on the above equations; in the remainder of this section we review the *right-looking algorithm* (see, for instance [15]).

Let us partition $\mathbf{M}$ and $\mathbf{R}$ as follows

$$\mathbf{M} = \left[\begin{array}{c|c} m_{11} & \mathbf{M}_{21}^T \\ \hline \mathbf{M}_{21} & \mathbf{M}_{22} \end{array}\right], \quad \mathbf{R} = \left[\begin{array}{c|c} r_{11} & \mathbf{0} \\ \hline \mathbf{R}_{21} & \mathbf{R}_{22} \end{array}\right],$$

where $m_{11}, r_{11} \in \mathbb{R}$, $\mathbf{M}_{21}, \mathbf{R}_{21} \in \mathbb{R}^{n-1}$ and $\mathbf{M}_{22}, \mathbf{R}_{22} \in \mathbb{R}^{(n-1)\times(n-1)}$.

Since $\mathbf{M} = \mathbf{R}\mathbf{R}^T$ we get

$$\mathbf{M} = \left[\begin{array}{c|c} r_{11}^2 & r_{11}\mathbf{R}_{21}^T \\ \hline r_{11}\mathbf{R}_{21} & \mathbf{R}_{21}\mathbf{R}_{21}^T + \mathbf{R}_{22}\mathbf{R}_{22}^T \end{array}\right],$$

and it follows that

$$r_{11} = \sqrt{m_{11}} \quad (3)$$

$$\mathbf{R}_{21} = \frac{1}{r_{11}} \mathbf{M}_{21} \quad (4)$$

$$\mathbf{R}_{22}\mathbf{R}_{22}^T = \mathbf{M}_{22} - \mathbf{R}_{21}\mathbf{R}_{21}^T \quad (5)$$

The algorithm, therefore, starts with $\mathbf{Q} = \mathbf{M}$ and calculates the first column of $\mathbf{R}$; then it sets

$$\mathbf{Q} = \mathbf{M}_{22} - \mathbf{R}_{21}\mathbf{R}_{21}^T \quad (6)$$

and iterates the above procedure over $\mathbf{Q}$, until all entries of $\mathbf{R}$ are calculated.

Remark 1: Even if $\mathbf{M}$ is compatible with the graph G, the matrices $\mathbf{Q}$ obtained during the right-looking approach are not, in general, compatible with G. We notice, in fact, that the matrix $\mathbf{Q}$ obtained at the end of the first iteration contains a term $\mathbf{R}_{21}\mathbf{R}_{21}^T$ which, in general, is not compatible with the sub-graph of G obtained by removing node v_1 along with its incident links (and the same holds for successive iterations). We notice, moreover, that the entries of $\mathbf{R}_{21}$ are nonzero only for the neighbors of node v_1 in G; this implies that $(\mathbf{R}_{21}\mathbf{R}_{21}^T)_{jk} = 0$ if $v_{j+1} \in V \backslash \mathcal{N}_1$ or $v_{k+1} \in V \backslash \mathcal{N}_1$, that is to say, the (j,k)-th entry of $\mathbf{R}_{21}\mathbf{R}_{21}^T$ can be nonzero only if both v_{j+1}, v_{k+1} are neighbors of node v_1. This fact implies that each pair of neighbors of node v_1 in G needs to interact in order to calculate the next diagonal entry of $\mathbf{R}$.

In the next section, we provide a distributed implementation of the right-looking algorithm which takes advantage of the above remark.

III. Distributed Cholesky Decomposition

We discuss next a distributed and asynchronous algorithm to let each node v_i in a network calculate the nonzero coefficients associated to the i-th column of $\mathbf{R}$; the pseudocode of the proposed algorithm is given in Algorithm 1.

In the following, we consider not only the neighbors $\mathcal{N}_i$ of a node v_i, but also a set of *virtual neighbors* $\mathcal{M}_i$ (it holds $\mathcal{N}_i \subseteq \mathcal{M}_i$). The virtual neighbors include, besides the regular neighbors, also a set of nodes v_j that are not directly connected to v_i via an edge of the graph and need to exchange messages with v_i in order to execute the proposed algorithm. We consider, therefore, a routing scheme where each node v_i maintains a set of *viapoints* for each of its virtual neighbors v_j, i.e., the identifier of the neighbor that node v_i has to send a message to in order for that message to reach v_j. An example is given in Figure 1.

A. Communication Model

We assume the nodes are able to communicate with their neighbors on a point to point and bidirectional basis, according to the structure of the graph G, which we assume to be undirected and connected. In the proposed network model, the nodes are able to transmit just one message at a time to each neighbor. The messages are sent in an asynchronous and event-based manner. Specifically, the reception of a message by the i-th agent triggers specific functions, depending on the

Algorithm 1 Distributed Cholesky Decomposition (from i-th agent's point of view)

procedure INITIALIZATION(is-pivot)
  active$_i \leftarrow$ true; ▷ the node is active
  pivot$_i \leftarrow$ false; ▷ the node is not the pivot
  visited$_i \leftarrow$ false; ▷ the node is not visited
  $\mathcal{M}_i \leftarrow \mathcal{N}_i$; ▷ virtual neighbors
  viapoint$_j \leftarrow j$, for all $j \in \mathcal{N}_i$; ▷ neighbors are addressed directly
  $q_{ji} \leftarrow m_{ji}$, for all $j \in \mathcal{N}_i \cup \{i\}$; ▷ store i-th column of $\mathbf{M}$
  $r_{ji} \leftarrow 0$, for all $j \in \mathcal{N}_i \cup \{i\}$; ▷ store i-th column of $\mathbf{R}$
  buffer$_i \leftarrow \emptyset$; ▷ buffer for received off-diagonal entries of $\mathbf{R}$
  buffer-full$_i \leftarrow$ false; ▷ buffer is not full
  if is-pivot **then**
    pivot$_i \leftarrow$ true; ▷ the node becomes the pivot
procedure ONPIVOT( )
  $r_{ii} \leftarrow \sqrt{q_{ii}}$; ▷ Calculate diagonal entry of $\mathbf{R}$
  isdiagonal$\leftarrow$ true;
  send $< i, j, r_{ii}, \text{isdiagonal} >$ to viapoint$_j$ for each $j \in \mathcal{M}_i$;
procedure ONRECEIVEDIAGONAL($< j, k, r_{jj}, \text{isdiagonal} >$)
  if active$_i$ **then** ▷ if i is the intended recipient k
    $r_{ij} \leftarrow q_{ji}/r_{jj}$; ▷ calculate off-diagonal entry of $\mathbf{R}$
    $q_{ii} \leftarrow q_{ii} - r_{ji}^2$ ▷ update q_{ji}
    isdiagonal$\leftarrow$ false;
    send $< i, j, r_{ij}, \text{isdiagonal} >$ to node viapoint$_j$;
  else ▷ forward message
    send $< j, k, r_{jj}, \text{isdiagonal} >$ to node viapoint$_j$;
procedure ONRECEIVEOFFDIAGONAL($< j, k, r_{kj}, \text{isdiagonal} >$)
  if active$_i$ **then** ▷ if i is the intended recipient j
    put r_{kj} in buffer$_i$;
    if |buffer$_i$| == $|\mathcal{M}_i| - 1$ **then** ▷ buffer is full
      buffer-full$_i \leftarrow$ true;
  else ▷ forward message
    send $< j, k, r_{kj}, \text{isdiagonal} >$ to node viapoint$_j$;
procedure ONBUFFERFULL( )
  send $< i, j, k, r_{ji} r_{ki} >$ to viapoint$_j$ for all $j, k \in \mathcal{M}_i$;
  pivot$_i \leftarrow$ false;
  active$_i \leftarrow$ false;
  send $< i, \text{active}_i >$ to all $j \in \mathcal{M}_i$ ▷ notify inactivity
  if $|\mathcal{M}_i| > 1$ **then** ▷ if a virtual neighbor can become pivot
    select $j^* \in \mathcal{M}_i$ at random ▷ choose new pivot
    send $< i, j^* >$ to j^* ▷ notify new pivot
  else ▷ Every agent j knows r_{jj}, terminate
    send termination message to all $j \in \mathcal{N}_i$
procedure ONRECEIVECROSSPRODUCT($< m, j, k, r_{jh} r_{kh} >$)
  if active$_i$ **then** ▷ if i is the intended recipient j
    if $k \in \mathcal{M}_i$ **then** ▷ q_{ki} is already stored
      $q_{ki} \leftarrow q_{ki} - r_{jh} r_{kh}$ ▷ update q_{ki}
    else
      $\mathcal{M}_i \leftarrow \mathcal{M}_i \cup \{k\}$; ▷ add virtual neighbor k
      $q_{ki} \leftarrow -r_{jh} r_{kh}$ ▷ create q_{ki}
  else ▷ forward message
    send $< i, j, k, r_{jh} r_{kh} >$ to node viapoint$_j$;
  viapoint$_k \leftarrow m$ ▷ create viapoint for k
procedure ONRECEIVEINACTIVE($< j, \text{active}_j >$)
  $\mathcal{M}_i \leftarrow \mathcal{M}_i \backslash \{j\}$;
procedure ONRECEIVEPIVOT($< j, i >$)
  pivot$_i \leftarrow$ true; ▷ the node becomes the pivot
procedure ONRECEIVETERMINATION($< j, i >$)
  send termination message to all $j \in \mathcal{N}_i$
  return r_{ii};

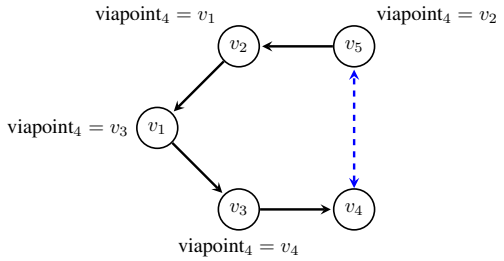

Fig. 1. Example of routing of a message from node v_5 to a virtual neighbor v_4 (the blue dotted edge is not an edge in the graph, but is instead a virtual relation created between v_5 and v_4).

structure of the message, which result in the transmission of one or more messages by the i-th node. When, as a consequence of a received message, a node has to transmit multiple messages to the same neighbor, it stores them in a queue and simply sends them one after another, as soon as possible.

As for the available bandwidth, we assume at most $O(\log n + \mathcal{W})$ bits are transmitted at each transmission along each link, where n is the number of nodes and $\mathcal{W}$ is the *word size*, i.e., the number of bits required to represent a real value with the desired precision.

B. Stored Variables

We assume each node stores:

- i: a unique identifier, which for simplicity we take to be a natural number from 1 to n;
- active$_i$: a Boolean variable that indicates the activation state of the node (initialized to true for all nodes);
- pivot$_i$: a Boolean variable that indicates whether node i is the *pivot*, i.e., a node that plays a pivotal role during the execution of the algorithm, as explained later (initialized to false for all but one node);
- visited$_i$: a Boolean variable that indicates whether node i has already received a termination message (initialized to false);
- q_{ii}: current i-th diagonal entry of the matrix $\mathbf{Q}$ used in the right-looking algorithm (initialized as $q_{ii} = m_{ii}$);
- r_{ii}: current i-th diagonal entry of the Cholesky matrix $\mathbf{R}$ (initialized as $r_{ii} = 0$);
- $\mathcal{M}_i$: set of virtual neighbors of the i-th node, initialized to $\mathcal{N}_i$ for all nodes. For each virtual neighbor, the agents store[1]:
  - viapoint$_j$: identifier of the node the agent i needs to send a message to in order for that message to reach the j-th virtual neighbor (initialized to j for all initial virtual neighbors v_j, since they coincide with the actual neighbors).
  - q_{ji}: current value of the (j, i)-th entry of matrix $\mathbf{Q}$ used in the right-looking algorithm (initialized as

[1]As discussed at the end of the section, we assume each agent stores information on all its virtual neighbors. In particular, the i-th agent stores the i-th column of $\mathbf{R}$. As discussed at the end of this section, such requirement corresponds to the need to store $O(n)$ values in the worst case.

$q_{ji} = m_{ji}$);
  - r_{ji}: current value of the (j,i)-th entry of the Cholesky matrix $\mathbf{R}$ (initialized as $r_{ji} = 0$);
- buffer$_i$: buffer for received off-diagonal entries of $\mathbf{R}$ (initialized to $\emptyset$);
- buffer-full$_i$: Boolean variable indicating whether the buffer is full, i.e., that all intended off-diagonal entries have been received, as clarified later (initialized to false).

Let us now describe the behavior of the proposed algorithm.

C. Event-Driven Behavior

Let us suppose a *pivot* node v_i has been elected (for space reasons, the pivot election falls outside the scope of this paper, the interested reader can refer to leader election techniques like those in [16] and references therein).

At the beginning of the algorithm, the OnPivot() procedure is triggered for the pivot node, which calculates r_{ii} according to Eq. (3) and transmits it to its virtual neighbors v_j (which coincide with its neighbors, see Phase I within Figure 2 for an example). Upon reception of a diagonal entry by a node v_i from a node v_j, the OnReceiveDiagonal() procedure is triggered, and node v_i calculates the corresponding off-diagonal entry r_{ij}, according to Eq. (4), and sends it back to the pivot v_j. Moreover v_i updates $q_{ii} \leftarrow q_{ii} - r_{ij}^2$ (e.g., see Phase II within Figure 2).

Every time the pivot receives an off-diagonal entry the OnReceiveOffDiagonal() procedure is triggered, and the pivot stores the received message in its buffer. When the number of stored messages equals the number of virtual neighbors, the OnBufferFull() procedure is activated, and (in order to calculate the next diagonal entry) the matrix $\mathbf{Q}$ has to be modified according to Eq. (5). This is done, according to Remark 1, by letting the pivot v_i calculate, for each pair of (virtual) neighbors v_j, v_k, the *cross-product* $r_{ji}r_{ki}$ and then sending it to both v_j, v_k, along with the identifier of the nodes involved in the cross-product (e.g., see Phase III within Figure 2). Then, the pivot notifies its virtual neighbors about the fact that the current iteration of the right-looking algorithm is over and sets inactive$_i \leftarrow$ true; the procedure terminates when the pivot node selects at random a new pivot among its virtual neighbors and it notifies the selected new pivot.

When a node v_f receives from a node v_m a cross-product message

$$< m, j, k, r_{jh}r_{jk} >$$

related to the nodes v_j and v_k, generated by a pivot v_h, and destined to node v_j, the OnReceiveCrossProduct() routine is triggered. If node v_f is active, then it must hold $f = j$, and v_f sets $q_{kf} \leftarrow -r_{jh}r_{kj}$ if v_k is a new virtual neighbor and it adds v_k to the set of its virtual neighbors, or it updates $q_{kf} \leftarrow q_{kj} - r_{jh}r_{kh}$ if v_k is already a virtual neighbor. If, conversely, v_f is not active, then it simply forwards the message to the node whose identifier is stored in viapoint$_j$. In both cases, v_f sets viapoint$_k \leftarrow m$, so that when v_f receives a message intended for v_k it forwards it to v_m. An example of the result of the OnReceiveCrossProduct() is shown in Phase IV within Figure 2.

When a node v_j receives a notification of inactivity from the pivot v_i, the OnReceiveInactive() procedure is triggered and the pivot is removed from the virtual neighborhood of v_j (viapoints, instead, are kept).

When a node v_j receives the message notifying that v_j is the new pivot, the OnPivot() routine is activated for v_j and the above procedures are iterated on the remaining $n-1$ nodes, while node v_i simply contributes to the routing of messages among virtual neighbors, by forwarding the messages it receives to the correct viapoint (e.g., see Phase V within Figure 2).

The above procedures are executed repeatedly, until the last diagonal entry is calculated, i.e., until the pivot v_i is not able to find a new pivot among its virtual neighbors. The pivot, in this case, transmits a termination message to its neighbors and terminates the execution of the algorithm. Upon reception of the termination message the OnReceiveTermination() procedure is activated, and the nodes broadcast the termination message to all their neighbors and terminate the execution of the algorithm.

IV. Characteristics of the Proposed Algorithm

A. Correctness

To prove the correctness of the proposed algorithm, let us first provide the following ancillary result, which is related to the soundness of the message routing among virtual neighbors.

Lemma 1: Let $\phi(i,j)$ be a function that maps a virtual neighbor $v_j \in \mathcal{M}_i$ of node v_i to a viapoint which is the one selected by agent v_i when it has to forward a cross-product message during the Distributed Cholesky Decomposition. Let us define $\phi_2(i,j) = \phi(\phi(i,j),j)$ while, in general, we denote the composition of the ϕ function k times as $\phi_k(i,j) = \phi(\phi(\dots\phi(i,j),j),j)$.

During the execution of the Distributed Cholesky Decomposition, for each node v_i and for each of its virtual neighbors $v_j \in \mathcal{M}_i$ there is an integer $k \geq 1$ such that $\phi_k(i,j) = j$.

Proof: Let us prove the result by induction. For the first pivot node v_1, each pair of nodes v_j, v_k involved in a cross-product are either neighbors themselves, or they are both neighbors of node v_i; in the first case we have $\phi(i,j) = j$, while in the second case node v_1 is a viapoint and it holds $\phi_2(i,j) = j$. Let us now suppose that the statement holds true until the m-th pivot v_m stops being the pivot and let us show that it holds true also until the $(m+1)$-st pivot v_{m+1} stops being the pivot. For each pair of nodes v_j, v_h involved in a cross-product related to the pivot v_{m+1}, the message intended for v_j is correctly received since all viapoints involved have been established before node v_{m+1} has become the pivot; as a consequence, there is an integer $k_1 \geq 1$ such that $\phi_{k_1}(m+1,j) = j$; similarly, it holds $\phi_{k_2}(m+1,h) = h$, for some integer $k_2 \geq 1$. While being forwarded from v_{m+1} to v_j, all intermediate nodes set the

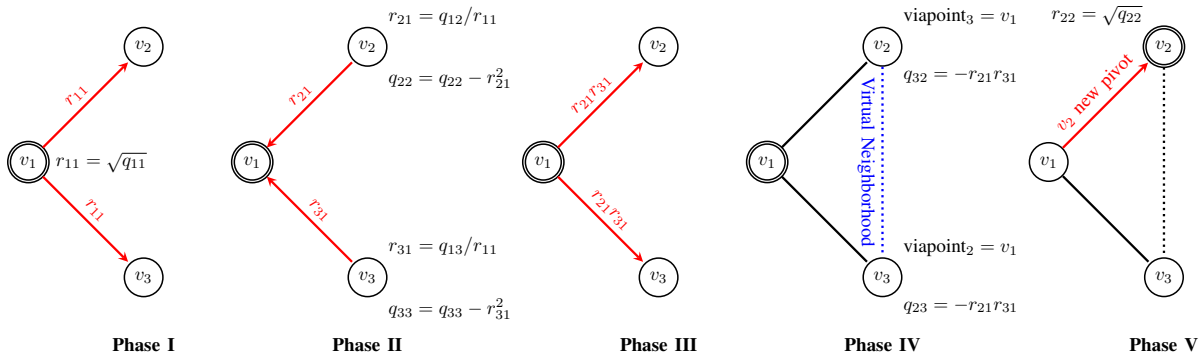

Fig. 2. Conceptual phases of the proposed distributed asynchronous algorithm.

identifier of the sender of the message they receive as the viapoint towards v_h (and similarly the nodes in the path from v_{m+1} to v_h set a viapoint for v_j). When v_j receives the message, therefore, there is an integer $k^* \leq k_1 + k_2$ such that $\phi_{k^*}(j,h) = h$, and an analogous result holds for v_h. The proof is complete. ■

***Theorem** 1:* Within the proposed algorithm, each agent i successfully calculates the nonzero entries of the i-th row of the Cholesky matrix $\mathbf{R}$.

Proof: Let us prove the result by induction. The fact that, when the first pivot v_1 is active, the nonzero elements in the first column of $\mathbf{R}$ are successfully calculated by v_1 is trivial. Let us suppose that the nonzero elements in the first m columns of $\mathbf{R}$ have been calculated each by the corresponding agent; we will show that the nonzero elements in the $(m+1)$-st column of $\mathbf{R}$ are successfully calculated by v_{m+1}. According to Lemma 1, the routing schema for the virtual neighbors is correct, hence all messages are eventually received by the intended recipient. It follows that, since the procedure is successful for the first m iterations, the entries of $\mathbf{Q}$ for $v_{m+1}, \ldots, v_n$ correspond to those obtained in a centralized fashion via Eq (6). During the $(m+1)$-st iteration, therefore, when all cross-products have been received by the virtual neighbors of the pivot v_{m+1}, since for all pairs $v_j, v_k \in \mathcal{M}_{m+1}$ it holds

$$q_{kj} \leftarrow q_{kj} - r_{j,m+1} r_{k,m+1},$$

the entries of $\mathbf{Q}$ for $v_{m+1}, \ldots, v_n$ must correspond to those obtained via Eq (6). The thesis follows. ■

B. Computational Characteristics in the Worst Case

Let us discuss the computational characteristics of the proposed algorithm in the worst case.

Message size: Each message is dominated by an identifier and a real number (e.g., in the case of a cross-product message); as a consequence each message requires at most $O(\log n + \mathcal{W})$ bits (recall that n is the number of nodes/agents and $\mathcal{W}$ is the word size).

Bandwidth: As a consequence of the message size, the bandwidth is $O(\log n + \mathcal{W})$ bits per link per time instant.

Local memory: The memory occupation at each agent is dominated by the buffer where the off-diagonal entries are stored. The buffer is $O(|\mathcal{M}_i(i)|(\log n + \mathcal{W}))$ bits, where $\mathcal{M}_i(i)$ is the set of virtual neighbors of node v_i, when it becomes the pivot. Since, in the worst case, $|\mathcal{M}_i(i)| = V \backslash \{v_i\}$ it follows that the memory occupation is $O(n(\log n + \mathcal{W}))$ bits.

Local computational complexity: The computational complexity for each node is dominated by the need to calculate, when that node is the pivot, the cross-products. The cross-products are $O(|\mathcal{M}_i(i)|^2)$; hence, in the worst case, the computational complexity at each node is $O(n^2)$.

Total messages exchanged: While a specific node v_i is the pivot, the number of exchanged messages is dominated by the cross-product messages which, neglecting the forwarded messages, amounts to $O(|\mathcal{M}_i(i)|^2)$ messages. As a consequence, except for the forwarded messages, the total number of messages sent is $O(\sum_{i=1}^{n} |\mathcal{M}_i(i)|^2)$, which in the worst case is $O(n^3)$. It should be noted that the exact number of messages exchanged is nontrivial to assess, as it depends on the structure of the graph G.

V. Simulations

In order to illustrate the behavior of the proposed algorithm, we consider *random geometric graphs* where we chose the position of $n = 30$ nodes in the unit square and we connect them provided that their Euclidean distance is less than a threshold ρ; the parameter ρ, therefore, accounts for the sparsity of the graph (the smaller ρ is, the sparser is the resulting graph). We consider several values of $\rho \in [0.2, 1]$, and for each value we take $\mu = 100$ random topologies. For the simulations below, we assume a word size $\mathcal{W} = 16$ bits. For each topology, we select a matrix $\mathbf{M}$ with random coefficients.

Figure 3 summarizes the results obtained in terms of (from top to bottom): (1) overall number of messages sent (upper plot); (2) greatest integer time stamp associated to a message; every time a message is sent as a consequence of a received message with timestamp T, the sent message is associated with a timestamp $T+1$ (central plot); (3) number of virtual neighbor relations established (lower plot, we count one relation for each pair of virtual neighbors established). Specifically, we plot as box plots the statistical distribution of the different instances for each choice of ρ.

We notice that the median value for the total exchanged messages ranges from about 3×10^3 up to about 1.4×10^4; hence, also including the forwarded messages, it is considerably below the worst case limit, which does not include forwarded messages and amounts to $n^3 = 2.7 \times 10^4$ messages. It is noteworthy that both the total exchanged messages and the number of virtual neighbors created show a peak for $\rho \in [0.35, 0.45]$, meaning that most message forwarding occur in medium-sparse networks. As for the finish time (greatest integer timestamp), it should be noted that there is an evident reduction as ρ grows (from about $T = 250$ for sparse networks up to about $T = 100$ for dense networks). Moreover, by comparing the total number of messages sent with the greatest timestamp, we notice that there is a relevant gap, suggesting that there is a high degree of parallelism in the transmission of the messages (e.g., the cross-product messages).

VI. Conclusions

In this paper we develop an asynchronous and distributed algorithm to calculate the Cholesky decomposition of a matrix with the same structure as the graph that represents the interaction among the agents. The proposed approach has limited bandwidth, memory and complexity, and does not rely on a centralized scheduler. Future work will be aimed at characterizing the complexity of the proposed algorithm in terms of the structure of the underlying graph topology, also considering particular structures, such as Delaunay triangulations or chordal graphs. We will also develop distributed algorithms that take advantage of distributed Cholesky decomposition in order to solve otherwise complex matrix-related operations, such as inversion, solution of linear systems of equations and determinant calculation.

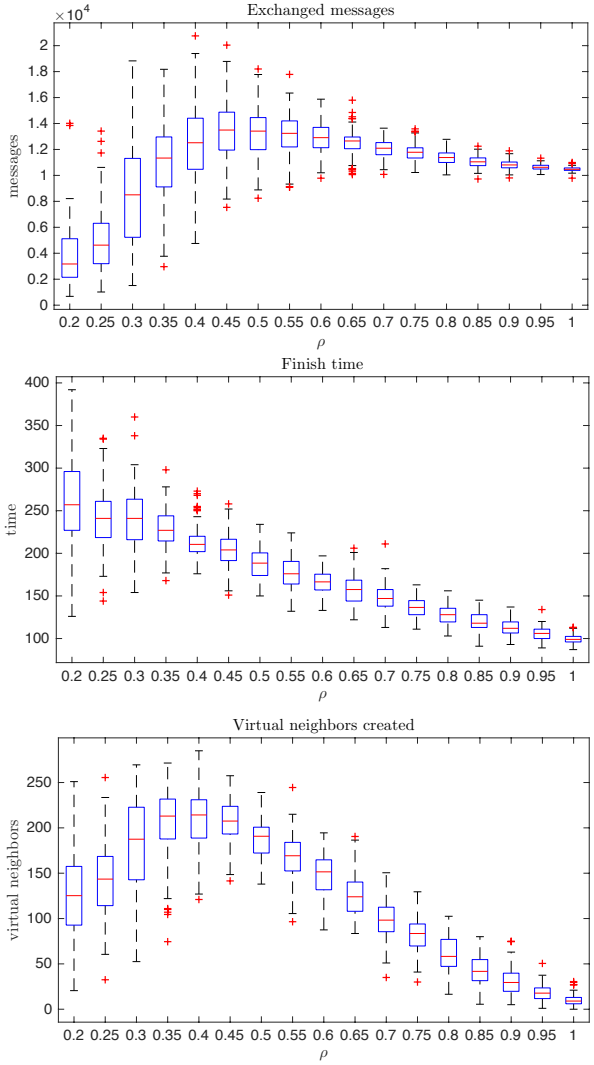

Fig. 3. Simulation results for random geometric graphs with $n = 30$ nodes and for different choices of the parameter ρ, which is proportional to the graph density. We consider $\mu = 100$ random topologies for each choice of ρ and we show as box plots the statistical distribution of (from top to bottom) the total number of exchanged messages, the timestamp associated to the last transmitted messages, and the number of virtual neighbors created.

References

[1] A. L. Cholesky, "Note sur une méthode de résolution des équations normales provenant de l'application de la méthode des moindres carrésa un syteme d'équations linéaires en nombre inférieura celui des inconnues", (Published six years after Cholesky's death by Benoit)." Bull. Géodésique, vol. 2, pp. 67–77, 1924.

[2] G. H. Golub and C. F. Van Loan, *Matrix Computations*, JHU Press, 2012.

[3] S. Särkkä, "On unscented Kalman filtering for state estimation of continuous-time nonlinear systems", *IEEE Transactions on Automatic Control*, vol. 52, no. 9, pp. 1631–1641, 2007.

[4] H. M. Menegaz, J. Y. Ishihara, G. A. Borges and A. N. Vargas, "A systematization of the unscented Kalman filter theory", *IEEE Transactions on Automatic Control*, vol. 60, no. 10, pp. 2583–2598, 2015.

[5] S. Yousefi, X. W. Chang, and B. Champagne, "Mobile localization in non-line-of-sight using constrained square-root unscented Kalman filter", *IEEE Transactions on Vehicular Technology*, vol. 64, no. 5, pp. 2071–2083, 2015.

[6] A. Krishnamoorthy and M. Deepak, "Matrix inversion using Cholesky decomposition", arXiv preprint arXiv:1111.4144, 2011.

[7] J. Wang and N. Elia, "Solving systems of linear equations by distributed convex optimization in the presence of stochastic uncertainty", *Proceedings of the IFAC World Congress*, pp. 1210–1215, 2014.

[8] D. E. Marelli and M. Fu, "Distributed weighted least-squares estimation with fast convergence for large-scale systems", *Automatica*, vol. 51, pp. 27–39, 2015.

[9] A. S. Teixeira, P. T. Monteiro, J. A. Carriço, M. Ramirez and A. P. Francisco, "Spanning edge betweenness", *Proceeding of the Workshop on Mining and Learning with Graphs*, pp. 27-31, 2013.

[10] C. Godsil and G. F. Royle, *Algebraic graph theory*, Springer, 2001.

[11] C. Ashcraft, S. C. Eisenstat and J. W. H. Liu "A fan-in algorithm for distributed sparse numerical factorization", *SIAM Journal on Scientific and Statistical Computing*, vol. 11, no. 3, pp. 593–599, 1990.

[12] C. Ashcraft, "The fan-both family of column-based distributed Cholesky factorization algorithms", *in Graph Theory and Sparse Matrix Computation*, vol. 56, pp. 159–190, 1993.

[13] G. Bosilca, L. Hatem and J. Dongarra "Power profiling of Cholesky and QR factorizations on distributed memory systems", *Computer Science-Research and Development*, vol. 29, no. 2, pp. 139–147, 2014.

[14] S. Abdelhak, R. S. Chaudhuri, C. S. Gurram, S. Ghosh and M. Bayoumi,"Energy-aware distributed QR decomposition on wireless sensor nodes", *The Computer Journal*, vol. 54, no. 3, pp. 373–391, 2010.

[15] N. Mateev, V. Menon, and K. Pingali, "Left-looking to right-looking and vice versa: An application of fractal symbolic analysis to linear algebra code re-structuring", *Technical Report TR2000-1797*, Cornell University, Computer Science, 2000.

[16] S. Kutten, G. Pandurangan, D. Peleg, P. Robinson, and A. Trehan, "Sublinear bounds for randomized leader election," in *Distributed Computing and Networking*. Springer, pp. 348–362, 2013.