Distributed Memory Sparse Inverse Covariance Matrix Estimation on High-Performance Computing Architectures

Aryan Eftekhari
Institute of Computational Science
Università della Svizzera italiana
Lugano, Switzerland
aryan.eftekhari@usi.ch

Matthias Bollhöfer
Institute of Computational Mathematics
TU Braunschweig
Braunschweig, Germany
m.bollhoefer@tu-bs.de

Olaf Schenk
Institute of Computational Science
Università della Svizzera italiana
Lugano, Switzerland
olaf.schenk@usi.ch

***Abstract*—We consider the problem of estimating sparse inverse covariance matrices for high-dimensional datasets using the ℓ_1-regularized Gaussian maximum likelihood method. This task is particularly challenging as the required computational resources increase superlinearly with the dimensionality of the dataset. We introduce a performant and scalable algorithm which builds on the current advancements of second-order, maximum likelihood methods. The routine leverages the intrinsic parallelism in the linear algebra operations and exploits the underlying sparsity of the problem. The computational bottlenecks are identified and the respective subroutines are parallelized using an `MPI-OpenMP` approach. Experiments conducted on a $5{,}320$ node Cray XC50 system at the Swiss National Supercomputing Center show that, in comparison to the state-of-the-art algorithms, the proposed routine provides significant strong scaling speedup with ideal scalability up to 128 nodes. The developed framework is used to estimate the sparse inverse covariance matrix of both synthetic and real-world datasets with up to 10 million dimensions.**

***Index Terms*—inverse covariance matrix, sparse matrices, approximate matrix inverse, high-performance computing.**

I. Introduction

In statistical analysis, one is often faced with the problem of estimating the sparse inverse covariance matrix of a high-dimensional dataset or, equivalently, a dataset with a high number of random variables. This type of analysis is useful in elucidating the association between the random variables. If the data samples are drawn from a Gaussian distribution, the inverse covariance matrix encodes the graph structure of a Gaussian Markov random field (GMRF). This general problem description finds applications in many fields such as finance, biology, health sciences, and many more [1]–[3]. There has been recent interest, and significant success, in adapting and extending ideas from statistical learning via Gaussian process (GP) regression to optimization via simulation problems [4]. At the heart of all such methods is a GP representing knowledge about the objective function. Calculating the conditional distribution requires inverting a large, dense covariance matrix, and this is the primary bottleneck for applying GP learning to large-scale problems. If the GP is a GMRF, then the precision matrix (inverse of the covariance matrix) can be constructed to be sparse. However, in a high-dimensional setting, this problem becomes especially challenging as (i) the number of data samples is usually limited in comparison to the number of random variables and (ii) the computational resources required increase superlinearly with the number of random variables. There exist multiple methods to address the mathematical soundness when dealing with large datasets with a limited number of samples; however, the computability of such methods remains a challenge when dealing with a high number of dimensions. We will focus our attention on this computational challenge for large-scale applications (a million dimensions or more) where the underlying inverse covariance matrix is sparse, or at least it is assumed to be sufficiently approximated as sparse. This assumption is commonly made in the literature [4]–[7] and is a necessary attribute when dealing with high-dimensional inverse covariance matrices.

Two common approaches for estimating a sparse inverse covariance matrix are the ℓ_1-regularized maximum likelihood (ML)- and pseudolikelihood (PL)-based methods. For the ML method one minimizes the ℓ_1-regularized negative log-likelihood function; see, e.g., [8]–[10]. The resulting problem is nonsmooth but convex and, thus, there are a larger number of approaches from convex optimization which have been developed such as (inexact) interior point methods [11]–[13], blockwise descent methods [14]–[17], iterative thresholding [18], alternating linearization [19], projected subgradients [20], greedy-type descent methods [21], and more recently developed second-order methods [22]–[26]. The QUadratic Inverse Covariance algorithm (QUIC) is a computationally efficient second-order technique which has multiple desirable properties; see [23] for further details. With this said, the applicability of the algorithm is limited to problem sizes of a few thousand dimensions. A variant of the algorithm, aptly named BigQUIC, is introduced in [25] for large-scale applications. This routine uses shared-memory parallelism and avoids the explicit construction of the larger dense matrices, thus reducing the overall runtime and memory footprint. However, even with this added performance and efficiency, the time-to-solution quickly becomes impractical when working with datasets with millions of dimensions. To address this

SC18, November 11-16, 2018, Dallas, Texas, USA
978-1-5386-8384-2/18/$31.00 ©2018 IEEE

issue, an extended version of QUIC called Sparse QUIC (SQUIC) has been proposed in [27]. SQUIC makes heavy use of sparse matrix operations with a greater focus on utilizing state-of-the-art high-performance sparse matrix routines. The resulting code uses multithreaded level-3 BLAS and has been shown to be significantly faster than BigQUIC but the memory requirements are higher due to the explicit representation of large sparse matrices. Both BigQUIC and SQUIC share a common problem in that they are limited to the performance and scalability of a single compute node.

Another approach for recovering sparse inverse covariance matrices are the PL-based methods. These methods utilize computationally simpler objective functions in an attempt to have a more efficient and direct exploitation of the underlying graphical structure [28]. Though the developments in PL-based methods have lagged in comparisons with ML methods, recent theoretical and computational advancements have led to the introduction of new algorithms with encouraging results; see, for example, [28]–[33]. With regard to large-scale applications, the High Performance CONvex CORrelation selection methoD (HP-CONCORD) is the state-of-the-art PL-based method introduced in [34]. HP-CONCORD is a distributed memory framework which has been shown to be highly scalable and performant. Single node tests show that in comparison to BigQUIC, HP-CONCORD exhibits a superior or equivalent performance for various datasets; refer to [34] for details.

Here we introduce an MPI–OpenMP parallelized version of SQUIC referred to as PRL-SQUIC, which is shown to have significantly better performance than the above noted ML- and PL-based methods, and ideal scalability for sufficiently large problems. The major computational bottlenecks of SQUIC are identified and a parallelization scheme is introduced to eliminate the scalability issues caused by the key restrictive subroutines. The developed framework is deployed at the Swiss National Supercomputing Centre (CSCS) where comparative performance tests are conducted. We illustrate the performance and scalability of PRL-SQUIC by using the framework to recover the approximate inverse covariance matrices of synthetic datasets of unprecedented dimensionality of up to 10 million variables. Finally, to motivate the applicability of the proposed method, we apply the PRL-SQUIC on a high-dimensional functional magnetic resonance imaging (fMRI) dataset of the human brain [35]. We emphasize that although the introduced algorithm provides impressive performance and scalability results for both the synthetic and the real-world datasets, the overall efficiency of the algorithm is dependent on the sparsity of intermediary matrix operations.

In section II we introduce the ML method for inverse covariance matrix estimation and the PL-based HP-CONCORD algorithm. We proceed in section III with a high-level description of the quadratic approximation method, followed by an outline of the QUIC and BigQUIC algorithms. In section IV we provide a detailed description of the SQUIC algorithm. Section V will be dedicated to the main contribution of the paper where the bottleneck of the SQUIC program and the developed parallelization scheme are described. In section VI, we outline the experimental setup. Finally, we present our numerical results in section VII and conclude in section VIII.

II. Background

We aim to solve the following problem: given the dataset $\mathbf{Y} \in \mathbb{R}^{p\times n}$ comprised of n independently drawn samples from a p-variate Gaussian distribution $\mathcal{N}(\boldsymbol{\mu}^*, \boldsymbol{\Sigma}^*)$, recover an estimate of the true inverse covariance matrix $\boldsymbol{\Theta}^* := (\boldsymbol{\Sigma}^*)^{-1} \in \mathbb{R}^{p\times p}$, where $\boldsymbol{\mu}^* \in \mathbb{R}^p$ is the true mean. Specifically we are interested in a setting where $\boldsymbol{\Theta}^*$ is sparse and $p \gg n$. A starting point is to define the sample covariance matrix

$$\mathbf{S} = \frac{1}{n}\sum_{j=1}^{n}(\mathbf{Y}_{:,j} - \boldsymbol{\mu})(\mathbf{Y}_{:,j} - \boldsymbol{\mu})^\top,\ \boldsymbol{\mu} = \frac{1}{n}\sum_{j=1}^{n}\mathbf{Y}_{:,j}, \qquad (1)$$

where $\mathbf{Y}_{:,j}$ denotes the jth column of $\mathbf{Y}$ and $\boldsymbol{\mu}$ is the sample mean. For simplicity, in the remaining text we will assume that $\mathbf{Y}$ is both scaled by $1/\sqrt{n}$ and has the mean value deducted, such that $\mathbf{S} = \mathbf{Y}\mathbf{Y}^\top$. Notice that due to the limited number of samples, $\mathbf{S}$ is both singular and contains significant noise. In the subsections to follow we highlight two different methods to solve this problem: the ℓ_1-regularized ML- and PL-based HP-CONCORD methods. Focus will be put on the prior as this will be the theoretical basis of our contribution.

We will adopt the following notation: for a vector or matrix, $\|\cdot\|_0$, $\|\cdot\|_{\max}$, $\|\cdot\|_F$, and $\|\cdot\|_1$ denote the number of nonzeros, elementwise maximum absolute value, Frobenius norm, and elementwise 1-norm, respectively. Finally, we use $\|\cdot\|_\#$ for the cardinality of a set or length of an array.

A. Maximum Likelihood Method

ML-based estimation methods recover $\boldsymbol{\Theta}$ through minimizing the negative log-likelihood function

$$g(\boldsymbol{\Theta}) := -\log\det\boldsymbol{\Theta} + \mathrm{tr}[\mathbf{S}\boldsymbol{\Theta}]. \qquad (2)$$

Given the sparsity parameter $\lambda > 0$, a sparsity prior is imposed on $\boldsymbol{\Theta}$ by ℓ_1-regularization of (2). This results in the convex regularized log-determinate program

$$\operatorname*{argmin}_{\boldsymbol{\Theta}\succ 0}\Big\{f(\boldsymbol{\Theta})\Big\},\ \text{where } f(\boldsymbol{\Theta}) = g(\boldsymbol{\Theta}) + \lambda\|\boldsymbol{\Theta}\|_1, \qquad (3)$$

where $\boldsymbol{\Theta} \succ 0$ denotes positive-definiteness of $\boldsymbol{\Theta}$. The objective function f has two parts: the convex and smooth negative log-likelihood function g, and the nonsmooth but still convex regularization term $\lambda\|\boldsymbol{\Theta}\|_1$. A variety of methods exist for solving (3); a selected subset includes [11], [14], [18]–[20], and [21]. First-order methods benefit from lower time complexity making them popular in high-dimensional settings, while second-order methods exhibit superlinear convergence but have higher computational costs. The QUIC algorithm, described in section III-A, benefits from superlinear convergence while still being computationally efficient.

B. HP-CONCORD Pseudolikelihood Method

The HP-CONCORD algorithm is a first-order method and to our knowledge, it is the current state-of-the-art large-scale PL-based technique. The algorithm solves the following convex minimization problem

$$\underset{\mathbf{\Theta}}{\operatorname{argmin}}\Big\{-\log\det\mathbf{\Theta}_D^2+\operatorname{tr}[\mathbf{S}\mathbf{\Theta}^2]+\lambda_1\|\mathbf{\Theta}_X\|_1+\lambda_2\|\mathbf{\Theta}\|_F^2\Big\},\quad (4)$$

where $\mathbf{\Theta}_D$ and $\mathbf{\Theta}_X$ refers to the diagonal and off-diagonal elements of $\mathbf{\Theta}$ such that $\mathbf{\Theta}=\mathbf{\Theta}_D+\mathbf{\Theta}_X$. To solve (4) a proximal gradient method is used where the nondifferentiable term $\lambda_1\|\mathbf{\Theta}_X\|_1$ is separated and, in its place, a soft-thresholding operator is used. At each iteration an appropriate step size is selected and the new estimate is updated. For a detailed mathematical and algorithmic description refer to [32]–[34]. The HP-CONCORD objective function (4) bears a resemblance to (3), but the computation involved is simpler than the latter, specifically as the log–determinant is of a diagonal matrix.

For large-scale applications, the authors [34] identify the computational bottleneck to be the matrix multiplications required for $\mathbf{\Theta}\mathbf{S}$ at each iteration and the construction of $\mathbf{S}$. To address this, a highly optimized communication-avoiding matrix multiplication routine is developed; refer to [36] for further details. We will use the HP-CONCORD framework for performance tests outlined in section VII.

III. Quadratic Approximation Method

The QUIC algorithm [23] solves (3) following the same approach as [37], [38], that is, the quadratic approximation method. In this section, we will begin by giving a high-level formulation of the quadratic approximation technique and follow up with an outline of the QUIC algorithm. Finally, we will highlight the specific differences between QUIC and its large-scale implementation BigQUIC.

The full objective function f in (3) consists of the differentiable negative log-likelihood g and nondifferentiable ℓ_1-regularization term. We begin by generating a second-order Taylor expansion of g around $\mathbf{\Theta}$. The quadratic approximation of $g(\mathbf{\Theta}+\mathbf{\Delta})$ is defined as

$$\hat{g}(\mathbf{\Delta}):=g(\mathbf{\Theta})+\operatorname{tr}[(\mathbf{S}-\mathbf{\Theta}^{-1})\mathbf{\Delta}]+\frac{1}{2}\operatorname{tr}[\mathbf{\Theta}^{-1}\mathbf{\Delta}\mathbf{\Theta}^{-1}\mathbf{\Delta}].\quad (5)$$

In standard form, the Hessian $\nabla^2 g\in\mathbb{R}^{p^2\times p^2}$ is computationally intractable due to its size. The authors [23] leverage the specific structure of the formulation to rewrite the Hessian in terms of the $p\times p$ matrix products; this corresponds to the last term in the summation of (5). The Newton direction $\mathbf{H}$ of the approximated f around $\mathbf{\Theta}$ can be written as the solution of the following coordinate descent problem:

$$\mathbf{H}:=\underset{\mathbf{\Delta}}{\operatorname{argmin}}\Big\{\hat{g}(\mathbf{\Delta})+\lambda\|\mathbf{\Theta}+\mathbf{\Delta}\|_1\Big\}.\quad (6)$$

For the details on the solution to (6), we refer the reader to [23]. The principal idea of quadratic approximation is to solve (3) as a sequence of optimization problems. In each step we use the local approximation $\mathbf{\Theta}+\mathbf{H}$ to generate the quadratic expansion (5), and solve for the Newton step (6). The QUIC algorithm described in the next section will leverage this optimization technique to exploit the sparsity of the problem and significantly reduce the required computation.

A. QUIC Algorithm

The QUIC algorithm solves the problem outlined in the previous section through the following three steps: (i) start with an initial or current estimate $\mathbf{\Theta}$, generate the local approximation (5); (ii) solve the local minimization problem in (6); (iii) update the current estimate to be $\mathbf{\Theta}+\mathbf{H}$ and proceed with the next Newton iteration. Interestingly, in the one-dimensional case, in [23] it is shown that (6) has an analytical solution. With this said, a line search procedure is required to keep the updated estimate positive-definite (required by (3)). The line search procedure is outlined in the algorithmic description at the end of this section.

A hallmark of the QUIC algorithm is that only a subset of the elements of $\mathbf{H}$ and, in turn, $\mathbf{\Theta}$ need to be computed at each Newton iteration. The indices that need to be updated are referred to as *free* and those that remain unchanged are *fixed*. It has been proven in [23] that the collection of these indices form the following two disjoint sets:

$$\mathcal{I}_{fixed}:=\Big\{(i,j)\in\mathcal{I}:|\mathbf{S}_{ij}-\mathbf{\Theta}_{ij}^{-1}|\leq\lambda\text{ and }\mathbf{\Theta}_{ij}=0\Big\},\quad (7)$$
$$\mathcal{I}_{free}:=\mathcal{I}\backslash\mathcal{I}_{fixed},$$

where $\mathcal{I}:=\{1,2,\ldots,p\}\times\{1,2,\ldots,p\}$. The key assumption here is that for a properly selected λ, we will have $\|\mathcal{I}_{free}\|_{\#}\ll p^2$. In such a case the computation of the coordinate descent in (6) is reduced to a relatively small set of indices $(i,j)\in\mathcal{I}_{free}$.

The QUIC routine is summarized in Algorithm 1. The program is executed with the inputs $\mathbf{Y}$, λ, T, and ϵ_{term} corresponding to the dataset, sparsity parameter, maximum iteration, and termination tolerance, respectively. The routine begins in steps 1–3 by computing $\mathbf{S}$ and initializing the initial guess of $\mathbf{\Theta}$ to be identity $\mathbf{I}$. Entering the iterative portion of the algorithm, the Newton iteration in steps 3–19 is begun by computing $\mathcal{I}_{free}$, $f(\mathbf{\Theta})$, and $\mathbf{H}$. The updated estimate $\mathbf{\Theta}+\mathbf{H}$ is not constrained to be positive-definite as required by (3). Positive-definiteness is ensured in steps 7–14 by using a line search procedure, where an appropriate step size $\alpha\in(0,1]$ is selected such that the updated estimate $\mathbf{\Theta}+\alpha\mathbf{H}$ is positive-definite and sufficiently decreases the objective function. This procedure is done iteratively by first defining $\alpha:=0.5^m$, and starting at iteration $m=0$, the updated estimate is checked for positive-definiteness by attempting a Cholesky decomposition. In step 10, if the updated estimate is positive-definite, an Armijo-type criterion, denoted as `AC` (refer to [25] for further details) is used to check if the objective function has sufficiently decreased. If both of these conditions are satisfied, the update is accepted, else the update is deducted, and the procedure is repeated for $m=m+1$. The convergence is checked in step 15 by computing f at the updated $\mathbf{\Theta}$. Notice that having the Cholesky factors, we can easily compute the log-determinant term of f (see (11) in section IV-B for a

Algorithm 1 QUIC.

```
Require: $\mathbf{Y}, \lambda, T, \epsilon_{term}$
 1: $\mathbf{S} \leftarrow \mathbf{Y}\mathbf{Y}^\top$
 2: $\mathbf{\Theta} \leftarrow \mathbf{\Theta}^{-1} \leftarrow \mathbf{I}$
 3: for $t = 1$ to $T$ do
 4:     compute : $\mathcal{I}_{free}$
 5:     obj $\leftarrow f(\mathbf{\Theta})$
 6:     compute : $\mathbf{H}_{ij} \; \forall \; (i, j) \in \mathcal{I}_{free}$
 7:     for $m = 0$ to $\ldots$ do
 8:         $\alpha \leftarrow 0.5^m$
 9:         $\mathbf{\Theta} \leftarrow \mathbf{\Theta} + \alpha\mathbf{H}$
10:         if $\mathbf{\Theta} \succ 0 \; \vee$ AC($\mathbf{\Theta}$) then
11:             break
12:         end if
13:         $\mathbf{\Theta} \leftarrow \mathbf{\Theta} - \alpha\mathbf{H}$
14:     end for
15:     if |obj $- f(\mathbf{\Theta})| < \epsilon_{term}$ then
16:         break
17:     end if
18:     $\mathbf{\Theta}^{-1} \leftarrow$ inv($\mathbf{\Theta}$)
19: end for
20: return $\mathbf{\Theta}$
```

similar approach using LDL decomposition). Finally in step 18, to construct the quadratic expansion of the next iteration, $\mathbf{\Theta}^{-1}$ is computed straightforwardly by using the already available Cholesky factors of $\mathbf{\Theta}$.

The QUIC algorithm is not designed for large-scale applications as it uses dense linear algebra subroutines. On modern machines with 64 GB of memory, QUIC will be limited to roughly $p = 4{\cdot}10^4$ dimensions. In the section to follow we will highlight the key differences of BigQUIC which is designed for large-scale inverse covariance matrix approximation.

B. *BigQUIC Algorithm*

BigQUIC has been developed to address the performance and scalability issues of QUIC in high-dimensional applications. The authors in [25] have shown that BigQUIC is capable of dealing with problem sizes up to $p = 10^6$ dimensions. The underlying theme of BigQUIC is not to form $\mathbf{S}$ or $\mathbf{\Theta}^{-1}$, as they will be dense. To do this the required elements of $\mathbf{S}$ are computed on the fly. Positive-definiteness is checked by computing $\log\det\mathbf{\Theta}$, which is, in turn, calculated recursively using the Schur complement. The explicit construction $\mathbf{\Theta}^{-1}$ is avoided by computing it column by column. Using the conjugate gradient method, the jth column of $\mathbf{\Theta}^{-1}$ is computed as the solution to the system $\mathbf{\Theta}\mathbf{\Theta}^{-1}_{:j} = \mathbf{e}_j$, where $\mathbf{e}_j$ is the jth unit vector. This same technique is also used in computing the inverse of the submatrices for the Schur complement. We refer the reader to [25] for details of the BigQUIC algorithm.

IV. Sparse QUIC

In this section, we will describe the SQUIC algorithm which will form the basis of the main contribution of the paper. In comparison to BigQUIC, SQUIC takes a different approach which is distinguished by (i) computing a sparse representation of the sample covariance matrix, (ii) utilizing a sparse Cholesky factorization routine, and (iii) implementing an approximate inversion scheme. We elaborate each of these methods in the respective sections IV-A, IV-B, and IV-C.

A. *Sparse Sample Covariance Matrix*

The sample covariance matrix $\mathbf{S}$ is dense and, thus, it poses an issue when p is large. We noted in section III that BigQUIC sidesteps this issue by computing the elements of $\mathbf{S}$ as required. This, however, leads to cache inefficiencies and decreased performance, as the matrix $\mathbf{S}$ will be computed incrementally as the index set $\mathcal{I}_{free}$ grows. On the other hand, precomputing $\mathbf{S}$ entirely may be computationally efficient, but it will not be possible for large p. Here a composite approach is taken by partially computing $\mathbf{S}$ and updating missing values during the course of the routine. A sparse representation of $\mathbf{S}$ is constructed by accepting the values of indices (i, j), such that

$$\lambda \leq |\mathbf{S}_{ij}| \leq \sqrt{\mathbf{S}_{ii}\mathbf{S}_{jj}} \quad \text{or } i = j. \tag{8}$$

Notice that the diagonal elements of $\mathbf{S}$ are always computed, thus the Cauchy–Schwarz upper bound in (8) can be used as a computationally cheap initial check of $|\mathbf{S}_{ij}|$. As shown in (7), to update $\mathcal{I}_{free}$ we require $\mathbf{S}$ to have a nonzero pattern which overlaps $\mathbf{\Theta}^{-1}$. Any such missing values of $\mathbf{S}$ are computed on the fly. This method gives a balanced advantage of doing some of the computation of $\mathbf{S}$ in a cache efficient way, while at the same time not expending significant resource on indices $(i, j) \notin \mathcal{I}_{free}$.

B. *Sparse Cholesky Factorization*

Sparse Cholesky factorization is a critical operation in the overall SQUIC algorithm. With the Cholesky decomposition, we can validate positive-definiteness of $\mathbf{\Theta}$ and also, as we will see in this section, we can easily compute $\log\det\mathbf{\Theta}$. In section IV-C an efficient method for computing $\mathbf{L}^{-1}$ is shown. Having this inverse factor would, in turn, allow us to easily compute $\mathbf{\Theta}^{-1}$. There has been significant work put into the development of high-performance Cholesky factorization routines as they play a critical role in direct solvers used for systems of linear equations [39]. One well-known attribute of the Cholesky decomposition is the difference between the nonzero structure of the sparse matrix $\mathbf{\Theta}$ and its respective factors $\mathbf{L}$. The nonzero elements that appear in $\mathbf{L}$ but not in $\mathbf{\Theta}$ are called *fill-in*. Reducing the fill-in is a necessity for large-scale direct solvers as problems with millions of variables are common. The mechanism and theory of fill-in reduction are beyond the scope of this paper, and we refer the interested reader to [40] and, for more recent developments, to [41].

The authors in [27] argue that reduced fill-in matrix factorization routines are well suited for the case of sparse inverse covariance matrix approximation as they will be memory efficient and performant for large sparse matrices. The software package used in SQUIC is `cholmod` [42]; however, many other high-performance sparse direct solvers packages can be used [43]–[46]. Here we reformulate the Cholesky decomposition in a slightly different form, its variant, the LDL decomposition,

$$\mathbf{\Theta} = \mathbf{PLDL}^\top \mathbf{P}^\top, \tag{9}$$

where $\mathbf{P}$ is the permutation matrix, $\mathbf{D}$ a diagonal matrix with strictly positive values, and $\mathbf{L}$ is a lower triangular matrix with identity as its diagonal. The presumption here is that $\mathbf{\Theta}$ is sparse and thus we expect the fill-in of $\mathbf{L}$ to be low, that is, we expect $\mathbf{L}$ to also be sparse. This, of course, is dependent on both $\mathbf{\Theta}^*$ and the selected parameter λ. Using the factor in (9), it is easy to confirm the log-determinant and the inverse can be written as follows:

$$\log\det \mathbf{\Theta} = \sum_{i=1}^{p} \log \mathbf{D}_{ii}, \tag{10}$$

$$\mathbf{\Theta}^{-1} = \mathbf{PL}^{-\top}\mathbf{D}^{-1}\mathbf{L}^{-1}\mathbf{P}^\top. \tag{11}$$

To compute $\mathbf{\Theta}^{-1}$, the inversion of the factor $\mathbf{L}$ will pose a problem, as it will be dense. This motivates the section to follow, where we will describe an approximation scheme for sparse representation of the inverse factor.

C. Approximate Matrix Inversion

As discussed in the previous section the main computation required for $\mathbf{\Theta}^{-1}$ is $\mathbf{L}^{-1}$; see (11). Direct inversion, as implemented in QUIC, will pose a problem in a high-dimensional setting as $\mathbf{L}^{-1}$ may be dense. Here the approximated Neumann series is used to compute a sparse approximation of $\mathbf{L}^{-1}$. We begin by considering the Neumann series

$$\mathbf{A}^{-1} = \sum_{k=0}^{\infty} (\mathbf{I} - \mathbf{KA})^k \mathbf{K} \tag{12}$$
$$\text{s.t.} \lim_{k\to\infty} (\mathbf{I} - \mathbf{KA})^k = \mathbf{0},$$

where $\mathbf{A}$ and $\mathbf{K} \in \mathbb{R}^{p\times p}$ are a nonsingular and a diagonal matrix, respectively. For a convergent series, $\mathbf{K}$ must be selected such that the spectral radius of the matrix power is less than one, $\rho(\mathbf{I}-\mathbf{KA}) < 1$. In this case, $\mathbf{A} = \mathbf{L}$, we can set $\mathbf{K} = \mathbf{I}$ as the matrix $\mathbf{I} - \mathbf{L}$ is nilpotent. This will be a convergent series for which only the terms exponentiated to $k < p$ will have nonzero values (a property of nilpotency). All higher-order terms $k \geq p$ will not contribute to the summation in (12), and thus truncation at $k = p$ is not subject to approximation error. To attain a sparse approximation $\hat{\mathbf{L}}^{-1} \approx \mathbf{L}^{-1}$ we use a dropout rule to only compute values of $\hat{\mathbf{L}}_{ij}^{-1}$ which have significant update magnitude. We can compute (12) recursively by rewriting it as

$$\hat{\mathbf{L}}_{k+1}^{-1} = \hat{\mathbf{L}}_{k}^{-1}(\mathbf{I} - \mathbf{L}) + \mathbf{I}, \tag{13}$$
$$\text{s.t. } |(\hat{\mathbf{L}}_{k+1}^{-1})_{ij} - (\hat{\mathbf{L}}_{k}^{-1})_{ij}| > \epsilon_{inv},$$

where $\hat{\mathbf{L}}_0^{-1} = \mathbf{I}$ is the initial guess and $\epsilon_{inv} > 0$ is the dropout tolerance. Finally, using a similar dropout rule as in (13), the sparse approximation of $\mathbf{\Theta}^{-1}$ can be computed as

$$\hat{\mathbf{\Theta}}^{-1} = \mathbf{P}\hat{\mathbf{L}}^{-\top}\mathbf{D}^{-1}\hat{\mathbf{L}}^{-1}\mathbf{P}^\top \text{ s.t. } |\hat{\mathbf{\Theta}}_{ij}^{-1}| > \epsilon_{inv}. \tag{14}$$

This approach explicitly makes two key assumptions. First, the number of iterations required in the Neumann series should be very small. For applications where p is large, computing higher iterations of Neumann series will hinder performance; indeed for a high-dimensional dataset, the number of iterations must be $\ll p$. In section VII-A we validate this assumption for the test case presented. The second assumption is that $\mathbf{L}^{-1}$ and $\mathbf{\Theta}^{-1}$ can be sufficiently approximated as sparse. The validity of this hypothesis is dependent on the underlying true inverse covariance matrix. If the matrix cannot be approximated as sparse, for example, say all elements $\mathbf{\Theta}_{ij}^{-1} \approx \epsilon_{inv}$, then the required computational resources and memory footprint could increase significantly.

V. Implementation

In this section, we will begin by motivating the parallelization of SQUIC. An analysis and discussion of the bottlenecks of scalability and performance are provided. Based on this analysis, in section V-A, we will describe the proposed parallelization scheme and the developed PRL-SQUIC routine.

The SQUIC algorithm is highly dependent on the sparsity of the intermediary arithmetic. As discussed in sections IV-A, IV-B, and IV-C, the three major subroutines of the program are categorized as the sample covariance matrix, sparse Cholesky factorization, and approximate inversion. In Figure 1 we can see the runtimes of each key subroutine for a fixed sample size of $n = 100$ and varying problem size p. The program exhibits $\mathcal{O}(p^2)$ complexity with the majority of the compute time consumed by the construction of the sample covariance matrix. This computation is dense as the sparsity pattern cannot be assessed a priori. It is clear that the scalability of the covariance matrix subroutine poses an evident bottleneck for large-scale applications. This is seen in the test case $p = 10^5$, where over 90% of the total runtime is consumed by computing the sample covariance matrix. Due to the dense arithmetic, a mixed approach of distributed and shared-memory parallelism is fitting for an efficient parallelization of this subroutine.

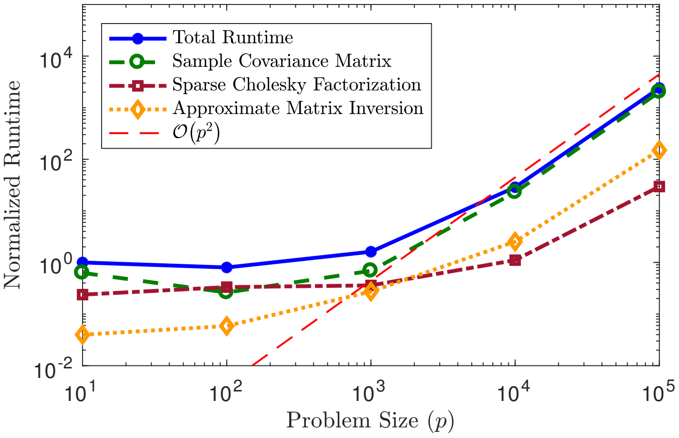


Fig. 1: Normalized runtimes for overall and major SQUIC subroutines with respect to the problem size p and a fixed $n = 100$. All tests use the datasets described in section VI.

In our experiments, the approximate matrix inversion is the second most time-consuming routine, but in comparison to the overall runtime, it takes roughly 9% of the computation time. The recursive formulation in (13) and (14) is not well suited for distributed memory parallelization as the computation is sparse and synchronization occurs at every iteration. Any performance gains from distributed parallelization would be penalized at each iteration with repeated communication and data reallocations. Also, from a strictly computational point of view, we do not expect major concerns with this routine as $\hat{\boldsymbol{\Theta}}^{-1}$ and $\hat{\mathbf{L}}^{-1}$ are maintained sparse throughout the computation. However, the validity of the imposed sparsity is problem dependent; we refer the reader to section IV-C for further discussions. With this said, we have adopted a shared-memory approach for a communication-free parallelization scheme.

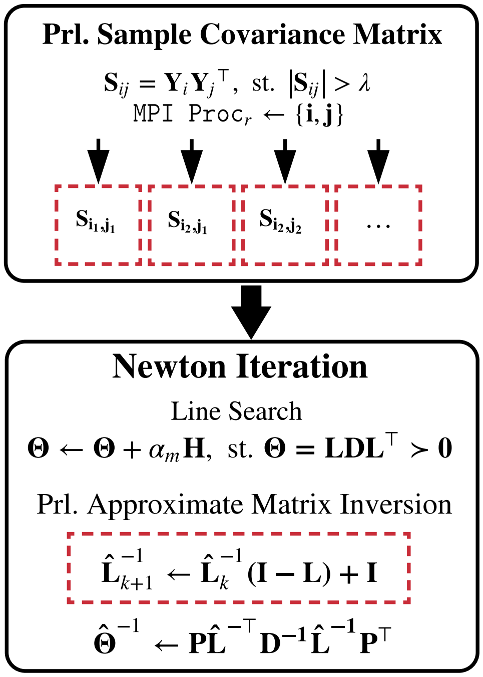
Finally, the sparse Cholesky factorization subroutine has the lowest impact on the overall runtime. It constitutes less than 1% of the total execution time. The factorization library `cholmod` provides sufficient performance such that this subroutine is not a primary bottleneck for large-scale applications.

A. Parallelization

Following the motivation of the previous section, we introduce the PRL-SQUIC algorithm which builds on SQUIC by parallelizing two key subroutines, the sample covariance matrix and approximate inversion. In Figure 2 we can see the overall computation scheme. The routine is broken into two parts, separated by total `MPI` synchronization between the top and bottom panel. The black arrows in the top panel box represent `MPI` processes, while `OpenMP` parallelism is isolated to within the dashed red boxes.

The algorithm begins by loading $\mathbf{Y}$ on each `MPI` process. This is required as we will need access to the data from any nodes to augment the sample covariance matrix during the update of $\mathcal{I}_{free}$ (see sections IV-A and III-A for further details). Notice that this approach necessitates that the dataset fits in the memory of a single node. Once loaded, we compute the diagonal elements of $\mathbf{S}$ and then proceed with block matrix multiplication. This operation is parallelized by partitioning $\mathbf{S}$ into submatrices and assigning each submatrix described by the collection of indices $\{\mathbf{i}, \mathbf{j}\}$ to an `MPI` process. In each process the computation is performed using `OpenMP` parallelism such that only the respective element $|\mathbf{S}_{ij}| > \lambda$ are saved. Upon completion, the data structures are globally synchronized. This provides an effective yet simple means to distribute the major bottleneck described in the last section. Notice that although the computation of the sample covariance matrix is dense, we construct a sparse approximation, and thus the size of the synchronized data structures are not a concern. The detailed algorithm for the parallelization of the sample covariance matrix subroutine will be described in section V-A1.

In the second part of the algorithm, we begin the Newton iteration which includes the line search, sparse matrix factorization, and the approximate inversion. As mentioned in the previous section, the factorization accounts for a small portion of the runtime and thus is not a performance bottleneck; however, for the approximate matrix inversion, we use `OpenMP`. The repeated evaluation of (13) is the major computation accounting for roughly 70% of the total time of the approximate inversion routine. The detailed algorithm for the parallelization of the approximate inversion subroutine will be described in section V-A2.



Prl. Sample Covariance Matrix
$\mathbf{S}_{ij} = \mathbf{Y}_i\mathbf{Y}_j^{\top}$, st. $|\mathbf{S}_{ij}| > \lambda$
MPI Proc$_r$ $\leftarrow \{\mathbf{i}, \mathbf{j}\}$
$\mathbf{S}_{\mathbf{i}_1,\mathbf{j}_1}$
$\mathbf{S}_{\mathbf{i}_2,\mathbf{j}_1}$
$\mathbf{S}_{\mathbf{i}_2,\mathbf{j}_2}$
$\dots$
Newton Iteration
Line Search
$\boldsymbol{\Theta} \leftarrow \boldsymbol{\Theta} + \alpha_m \mathbf{H}$, st. $\boldsymbol{\Theta} = \mathbf{LDL}^{\top} > \mathbf{0}$
Prl. Approximate Matrix Inversion
$\hat{\mathbf{L}}_{k+1}^{-1} \leftarrow \hat{\mathbf{L}}_k^{-1}(\mathbf{I} - \mathbf{L}) + \mathbf{I}$
$\hat{\boldsymbol{\Theta}}^{-1} \leftarrow \mathbf{P}\hat{\mathbf{L}}^{-\top}\mathbf{D}^{-1}\hat{\mathbf{L}}^{-1}\mathbf{P}^{\top}$

Fig. 2: Two parts of the PRL-SQUIC algorithm: distributed memory in the top and non distributed memory in the bottom box. Arrows in the top box signify `MPI` processes. Parallelization using `OpenMP` is symbolized by the dashed red boxes. Global `MPI` synchronization is required when transitioning from the top to the bottom panel.

1) Parallel Sample Covariance Matrix: The kernel operation in (1) is matrix-matrix multiplication, which is highly parallelizable. Though the sample covariance matrix is approximated as being sparse, the computation is dense, due to the undetermined sparsity pattern. The pseudocode shown in Algorithm 2 is executed with inputs $\mathbf{Y}$, λ, p, and the work block size b. In step 1 of the algorithm, we begin by assigning the workload array $\mathbf{q}$ containing the lower-triangular block indices of $\mathbf{S}$, where each corresponds to $b \times b$ submatrices. For simplicity we assume the number of processes, denoted by "proc_size," is a divisor of p and set $b =$ proc_size/ p. For example, given $p = 90$ and 3 processes, we would have $b = 30$ and $\mathbf{q} = \{\{1,1\},\{2,2\},\{3,3\},\{2,1\},\{3,1\},\{3,2\}\}$, where task $\mathbf{q}_4$ would identify the 30×30 submatrix with top left element $\mathbf{S}_{31,1}$. Next, from steps 2–4, we sequentially compute the diagonal of the sample covariance matrix which will be used to check the Cauchy–Schwarz condition outlined in section IV-A. In development we use compressed sparse column matrix format; however, for ease of notation in the pseudocode, we show coordinate list sparse matrix format. The arrays $\mathbf{c}$ and $\mathbf{v}$ are the collection of matrix indices and values,

Algorithm 2 Parallel sparse sample covariance matrix.

```
Require: $\mathbf{Y}, \lambda, p, b$
 1: $\mathbf{q} \leftarrow \{\{n, m\} : \{1 \leq m \leq n \leq b\} \in \mathbb{N}\}$
 2: for $i = 1$ to $p$ do
 3:     $\mathbf{S}_{ii} \leftarrow \mathbf{Y}_{i:}\mathbf{Y}_{i:}^{\top}$
 4: end for
 5: for $k = 1$ to $\|\mathbf{q}\|_{\#}$ do
 6:     if $k \pmod{\text{proc\_size}} = \text{proc\_id}$ then
 7:         $\{n, m\} \leftarrow \mathbf{q}_k$
 8:         #par region: static(default)
 9:         $\mathbf{v}^{tmp} \leftarrow \mathbf{j}^{tmp} \leftarrow \emptyset$
10:         for $j = (m-1)b + 1$ to $mb$ do
11:             for $i = (n-1)b + 1$ to $nb$ do
12:                 if $\lambda^2 \leq \mathbf{S}_{ii}\mathbf{S}_{jj} \wedge i > j$ then
13:                     $v \leftarrow \mathbf{Y}_{i:}\mathbf{Y}_{j:}^{\top}$
14:                     if $\lambda \leq |v|$ then
15:                         $\mathbf{v}^{tmp} \leftarrow \{\mathbf{v}^{tmp}, v\}$
16:                         $\mathbf{c}^{tmp} \leftarrow \{\mathbf{c}^{tmp}, \{i, j\}\}$
17:                     end if
18:                 end if
19:             end for
20:         end for
21:         critical : $\mathbf{v} \leftarrow \{\mathbf{v}, \mathbf{v}^{tmp}\}$
22:         critical : $\mathbf{c} \leftarrow \{\mathbf{c}, \mathbf{c}^{tmp}\}$
23:         #end par region
24:     end if
25: end for
26: synchronize $\mathbf{v}, \mathbf{c}$
27: $\mathbf{S} \leftarrow \{\mathbf{v}, \mathbf{c}\}$
28: return $\mathbf{S}$
```

respectively. The bulk of the algorithm begins at steps 5–25, where each task $\mathbf{q}_k$ is assigned to process "proc_id" using the modulo operator. This procedure ensures that the workload is well distributed over the processes and eliminates significant load imbalances. Using thread-level parallelism (denoted by "par region"), we compute the respective submatrix of $\mathbf{S}$ using the default OpenMP static scheduling. This scheduling type divides the computation across each thread as evenly as possible, in the largest possible contiguous chunks. Due to the lack of knowledge about the size of the data structures, efficient reallocation or resizing of shared data structures within a process's parallel region is not possible. As a solution, local temporary buffers $\mathbf{c}^{tmp}$ and $\mathbf{v}^{tmp}$ are used. Since the buffers are local to each thread, resizing can be done without issue. Furthermore, since $\mathbf{S}$ is symmetric and the diagonal is already computed, we only evaluate rows $i > j$. The accepted values are stored in the respective temporary data structures. Before exiting the parallel region the local temporary buffers must be consolidated; this is done serially and is denoted by "critical." After step 25, we exit the parallel region of each distributed memory process and globally synchronize the partially computed $\mathbf{S}$ on each node. Finally, the full sparse sample covariance matrix is reconstructed and returned on each node.

Algorithm 3 Parallel approximate inversion.

```
Require: $\mathbf{L}, \epsilon_{inv}$
 1: $\hat{\mathbf{L}}^{-1} \leftarrow 2\mathbf{I} - \mathbf{L}$
 2: $\mathbf{L} \leftarrow \mathbf{I} - \mathbf{L}$
 3: $\mathbf{u} \leftarrow \{\mathbf{0}\}$
 4: do
 5:     #par region: static(16)
 6:     for $j = 1$ to $p$ do
 7:         $r \leftarrow$ thread_id
 8:         if $\|\mathbf{L}_{:j}\|_0 > 0$ then
 9:             for $i = j$ to $p$ do
10:                 $v \leftarrow \hat{\mathbf{L}}_{i:}^{-1}\mathbf{L}_{:j} - \hat{\mathbf{L}}_{ij}^{-1}$
11:                 if $\epsilon_{inv} \leq |v|$ then
12:                     $\mathbf{u}_r \leftarrow max(\mathbf{u}_r, |v|)$
13:                     $\hat{\mathbf{L}}_{ij}^{-1} \leftarrow v + \hat{\mathbf{L}}_{ij}^{-1}$
14:                 end if
15:             end for
16:         end if
17:     end for
18:     #end par region
19: while $max(\mathbf{u}) > \epsilon_{inv}$
20: return $\hat{\mathbf{L}}^{-1}$
```

2) Parallel Approximate Matrix Inversion: The approximate matrix inversion is dominated by sparse matrix-matrix multiplication, but also requires intermediary logic for the dropout rule (see section IV-C for more details). Unlike the sample covariance matrix subroutine, the computation here begins sparse, as $\mathbf{L}$ is sparse, and is kept sparse. We execute Algorithm 3 with $\mathbf{L}$ and ϵ_{inv} as inputs. Here ϵ_{inv} is used as both a termination and dropout tolerance. In steps 1–3 of the algorithm the appropriate variables are initialized to their starting values. The value of $\mathbf{L}$ is written over as $\mathbf{I} - \mathbf{L}$ to simplify the computation (see (13) for more information). Notice that we know $\mathbf{L}_{ii}^{-1} = 1$, as it is a unit-diagonal triangular matrix, thus none of the arithmetic in (13) will be performed on the diagonal entries. Each element of the maximum absolute value array $\mathbf{u}$ is allocated to a thread id r. In step 19 of the algorithm we use $\mathbf{u}$ to assess convergence of the approximate Neumann series. The majority of the computation takes place in the shared-memory parallel region shown in steps 5–18. Here we use the OpenMP static scheduling with a chunk size of 16. Decreasing the chunk size will decrease load imbalances as it will be unlikely for clusters of columns with high computational load to be allocated to the single thread. With this said, larger chunk sizes will increase cache efficiency, but also increase potential load imbalances. The optimal selection of the chunk size is dependent on the problem and system specification. Next, on line step 8 if a given column of $\mathbf{L}^{-1}$ contains nonzeros, we proceed to compute and store the respective update values of $\hat{\mathbf{L}}_{ij}^{-1}$ in the local temporary variable v. If the update value surpasses the tolerance, we adjust both $\mathbf{u}$ and $\hat{\mathbf{L}}_{ij}^{-1}$ with the temporary variable. This process is repeated until convergence.

VI. Experimental Setup

In this section, we outline the experimental setup for results shown in section VII. Here we describe the procedure for generating the synthetic datasets and sparsity parameters used to evaluate the scalability and performance of PRL-SQUIC. For performance comparisons we use C++ packages of BigQUIC and HP-CONCORD.

To construct the synthetic datasets, we start by defining $\boldsymbol{\Theta}^*$ and computing its exact inverse $\boldsymbol{\Sigma}^*$. Given a matrix of normally distributed uncorrelated random variables $\mathbf{Z} \in \mathcal{N}(\mathbf{0}, \mathbf{I}) \in \mathbb{R}^{p \times n}$, the synthetic dataset is computed as $\mathbf{Y} = \mathbf{Z}\mathbf{L}$, where $\mathbf{L}$ is the lower-triangular Cholesky factor of $\boldsymbol{\Sigma}^*$. Generating large-scale test cases poses a problem as computing the exact inverse of $\boldsymbol{\Theta}^*$ is a non-trivial task. To sidestep this issue, we construct $\boldsymbol{\Theta}^*$ such that it is blockwise equal and thus inversion can be computed on each block $\boldsymbol{\theta}^* \in \mathbb{R}^{l \times l}$, where l is the block size. We assign $l = 10$ and define the symmetric block matrix

$$\boldsymbol{\theta}^*_{ji} = \boldsymbol{\theta}^*_{ij} := \begin{cases} 1, & i = j, \\ (11 - j)^{-1}, & i = 10, \\ 0, & \text{else}. \end{cases} \tag{15}$$

This matrix has an “arrowhead” sparsity structure and will have on average 2.8 nonzero entries per row. The full $p \times p$ matrix can be written as $\boldsymbol{\Theta}^* = \mathbf{I}_{p/l} \otimes \boldsymbol{\theta}^*$. Notice that $\boldsymbol{\Theta}^*$ encodes the GMRF of p/l correlated hub-like clusters. The inverse of $\boldsymbol{\Theta}^*$ can be easily computed as $\boldsymbol{\Sigma}^* = \mathbf{I}_{p/l} \otimes (\boldsymbol{\theta}^*)^{-1}$. We know that $(\boldsymbol{\theta}^*)^{-1}$ will be dense, but in a high-dimensional setting $\boldsymbol{\Sigma}^*$ will be sparse with a 10×10 dense block on the diagonal.

The synthetic datasets generated have $p = \{10, 10^2, 10^3, 10^4, 10^5, 10^6, 10^7\}$ random variables, all with $n = 100$ samples. We have selected the sparsity parameters for PRL-SQUIC and BigQUIC as $\lambda = \{0.65, 0.70, 0.75, 0.80, 0.85, 1.0, 1.2\}$ for the respective problem sizes. For HP-CONCORD, tests have been conducted for problem sizes up to $p = 10^4$, using $\lambda_1 = \{0.15, 0.20, 0.35, 0.45\}$ and $\lambda_2 = 0.2$. For the fMRI dataset, $\lambda = 0.95$ is used for PRL-SQUIC and for HP-CONCORD, $\lambda_1 = 0.8$ and $\lambda_2 = 0.01$ These parameters have been selected such that PRL-SQUIC, BigQUIC, and HP-CONCORD provide an equivalent number of nonzeros in the recovered inverse covariance matrix which are also closest to the ground truth in the synthetic test cases.

VII. Numerical Results

In this section, we will demonstrate the performance and scalability of PRL-SQUIC. We begin in section VII-A with an analysis of the approximate inversion scheme. In section VII-B we evaluate the single node performance of PRL-SQUIC with respect to BigQUIC and HP-CONCORD. In section VII-C the scalability results for PRL-SQUIC are provided for both distributed memory and single node configurations. Finally, in section VII-D we apply PRL-SQUIC on an fMRI dataset to recover the functional connectivity structure of the human brain.

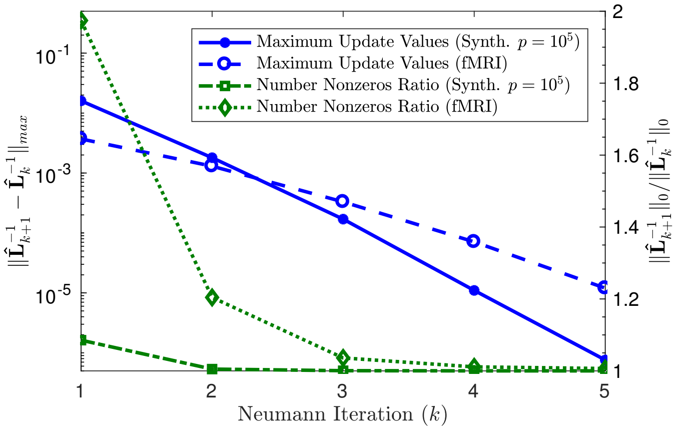


Fig. 3: The maximum absolute updated values of $\hat{\mathbf{L}}^{-1}_{k+1}$ (left axis) versus Neumann series iteration k; see Algorithm 2 and section IV-C for details. The ratio of the number of nonzero elements of $\hat{\mathbf{L}}^{-1}_{k+1}$ with respect to $\hat{\mathbf{L}}^{-1}_{k}$ (right axis) versus Neumann series iteration k. In all tests cases $\epsilon_{inv} = 10^{-12}$.

All tests have been conducted on “Piz Daint,” a Cray XC50 system that is installed at CSCS. Each of the compute nodes is equipped with a 12-core Intel(R) Xeon(R) E5-2690 v3 @ 2.60 GHz with 64 GB of memory.

A. *Sparse Inverse Matrix Approximation*

In this section, we look at the relationship between the number of Neumann series iterations and the number of nonzeros and maximum update values of $\hat{\mathbf{L}}^{-1}$. For this test we use the $p = 10^5$ synthetic and the fMRI datasets, with the approximate inversion tolerance set to $\epsilon_{inv} = 10^{-12}$.

The reported results in Figure 3 show that for both datasets, the ratio of the number of nonzero elements in $\hat{\mathbf{L}}^{-1}_{k+1}$ with respect to $\hat{\mathbf{L}}^{-1}_{k}$ approaches 1 after only 3 iterations. At this point the maximum update value of the Neumann series is sufficiently small, on the order of 10^{-3}, and the number of nonzeros per row of $\hat{\mathbf{L}}^{-1}$ is equal to approximately 60 and 1,400 for the synthetic and fMRI datasets respectively. This observation shows that for these synthetic and real-world database the Neumann series requires only a few iterations to capture both the structure and accuracy of the approximated inverse. Notice that if we select, say, $\epsilon_{inv} = 10^{-3}$, the number of nonzeros will be significantly less than what is observed in this test. This is because what we are reporting is the maximum update values and, thus, there could be many update values which fall below the threshold ϵ_{inv} threshold. In practice for the same test at $\epsilon_{inv} = 10^{-3}$, we observe only 3 and 20 nonzeros per row for the synthetic and fMRI datasets, respectively. As a result we will use $\epsilon_{inv} = 10^{-4}$ for the remaining tests outlined below. In the extreme cases where $\hat{\mathbf{L}}^{-1}$ might not be sparse and the majority of the elements are large, we might expect some deteriorating performance due to the increase of floating point operations and memory footprint. However, in such an extreme scenario, the impact of increased floating point operations can be mitigated due to the multithreading shared-memory parallelism of the Neumann series iteration; see section VII-C for details.

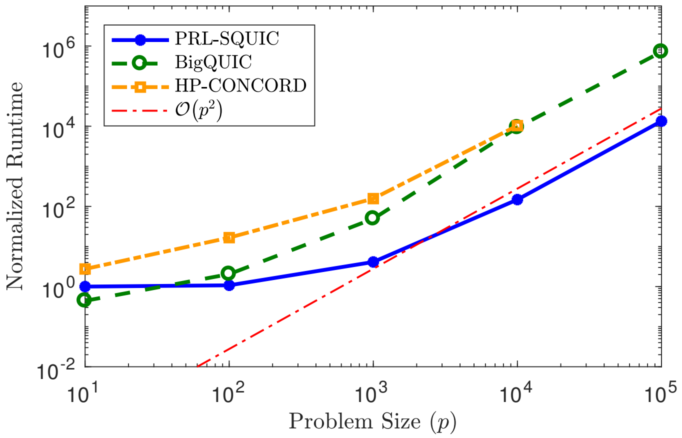


Fig. 4: The overall normalized runtime (single node) PRL-QUIC, BigQUIC, and HP-CONCORD (using pure `OpenMP`) versus problem size p. The thin dash-dotted line is the $\mathcal{O}(p^2)$ references. In all tests $\epsilon_{term} = \epsilon_{inv} = 10^{-4}$.

B. Single Node Performance

As BigQUIC is not a distributed memory software package, the performance comparison between PRL-SQUIC, BigQUIC, and HP-CONCORD will be conducted on a single node. We expect similar performance profiles between PRL-SQUIC and HP-CONCORD in a distributed memory deployment, as both programs exhibit efficient strong scalability (see VII-C and [34] for PRL-SQUIC and HP-CONCORD, respectively). In Figure 4 we report the total normalized runtime for the synthetic test cases with $p \leq 10^5$ and using a termination tolerance of $\epsilon_{term} = 10^{-4}$. We can observe that on a single node the runtime of BigQUIC and HP-CONCORD converge to be roughly the same. This observation corresponds to the "chain graphical model" test case outlined in [34]. Notice that for HP-CONCORD the problem size was limited to $p < 10^5$, though the distributed memory deployment does not have this restriction. PRL-SQUIC exhibits significantly faster runtimes in comparison to both BigQUIC and HP-CONCORD. For the test case of $p = 10^5$, PRL-SQUIC finished in 2 minutes, while BigQUIC took over 2 hours. For this problem size, PRL-SQUIC and BigQUIC converged in 4 and 8 Newton iterations, respectively. Each iteration of PRL-SQUIC took roughly 2 seconds while the remaining time was spent on the sample covariance matrix subroutine. For BigQUIC each iteration took the same time of 15 minutes. Tests larger than $p = 10^5$ could not be conducted at CSCS as the runtime of BigQUIC would exceed the 24 hour maximum execution time.

C. Strong Scalability

Strong scaling for distributed memory parallelism for all test cases is shown in the top panel of Figure 5. All experimental results are based on 4 Newton iterations. For the synthetic datasets the number of nonzeros per row recovered in the inverse covariance matrix is $\{1.5, 1.3, 1.1\}$ for $p = \{10^5, 10^6, 10^7\}$, respectively, and 20 for the fMRI dataset. In the $p = 10^7$ test case, we observe that PRL-SQUIC exhibits ideal scalability from 8 to 128 nodes. For the synthetic test case of $p = 10^6$ we observe almost ideal scalability, with slight deterioration at 128 nodes. On the 128 node tests, the total compute time of the $p = \{10^6, 10^7\}$ synthetic examples was 3 minutes and 1.3 hours, respectively. For the small test cases of $p = 10^5$ synthetic and the fMRI example, we have reduced strong scaling starting at roughly 32 nodes. At this point, the routine runs in 8 and 419 seconds for the respective tests and the critical bottleneck becomes the non distributed approximate matrix inversion subroutine. The observed efficient distributed memory scalability of PRL-SQUIC in the large examples is justified because the sample covariance matrix subroutine dominates the compute time in these high-dimensional examples. In contrast to the smaller test cases, the $p = \{10^6, 10^7\}$ synthetic examples, 32% and 92% of the respective total compute time was spent on the sample covariance matrix subroutine.

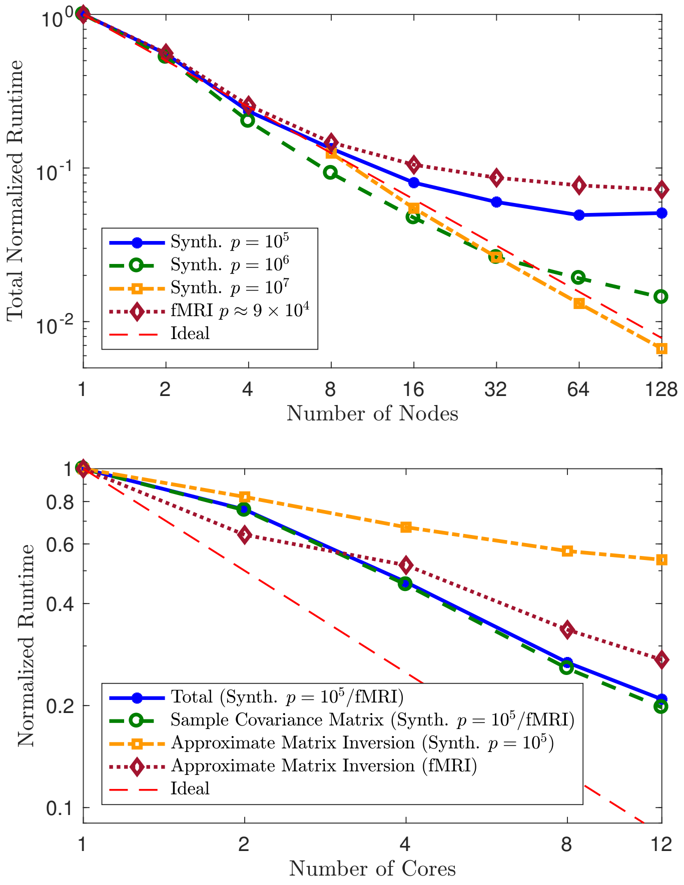


Fig. 5: Top panel—Strong scaling for distributed memory deployment of PRL-SQUIC. Bottom panel—Node-level strong scaling for PRL-SQUIC and respective subroutines. All tests run 4 Newton iterations with $\epsilon_{inv} = 10^{-4}$. Note the normalized total and sample covariance matrix runtimes of the fMRI test case are visually identical to the synthetic case.

Node-level scaling for the parallelized subroutines and overall program are shown in the bottom panel of Figure 5. Tests have been conducted for a synthetic problem size of $p = 10^5$ and the fMRI dataset. The normalized total and sample covariance matrix runtimes of the fMRI test cases exhibit visually identical parallelization in comparison to the synthetic test case, and thus are ignored for clarity. For both test cases, the overall routine attains a 5× speedup at 12 cores with a majority of the performance gains attributed to the sample covariance matrix subroutine. The approximate matrix inversion subroutine exhibits $1.9\times$ and $3\times$ speedup at 12 cores, for the synthetic and fMRI test cases, respectively. The lower performance gains of the approximate matrix inversion using the Neumann iteration are expected as the multithreaded sparse matrix-matrix multiplication has low arithmetic intensity. For scenarios where $\hat{\mathbf{L}}^{-1}$ is less sparse, for example, in the fMRI test case, we can see that the parallelism in this subroutine is more effective as the arithmetic intensity of the computation is higher. With this said, for both subroutines, there is visibly minimal deterioration in the parallelization with increasing number of cores.

D. Case Study: fMRI dataset

The testing procedure outlined in this section follows the approach of the authors of HP-CONCORD [34]. We use the "HCP_1200" fMRI dataset which is part of the Human Connectome project [35] found at `https://db.humanconnectome.org`, which contains $p = 91{,}282$ random variables corresponding to the left and right hemisphere, and subcortical regions of the brain. Mapping the connectivity of the brain is objective in this case study, and although the covariance matrix will provide insight on this marginal connectivity [47], the inverse covariance matrix is of particular interest for modeling direct associations [48].

We will use PRL-SQUIC to map direct functional connectivity of the brain and compare it in Table I to HP-CONCORD on 128 nodes. First, we can see that both routines return approximately the same number of nonzeros per row for the recovered inverse covariance matrix $\hat{\mathbf{\Theta}}$. In Figure 6 the sparsity structure of $\hat{\mathbf{\Theta}}$ recovered by PRL-SQUIC is visualized. We can see two distinct block diagonal groups of connectivity which correspond to the left and right hemisphere of the brain. The lower portion of the matrix is associated with the subcortical regions of the brain. The recovered sparsity structure is coherent with results presented in HP-CONCORD [34]. Further, we can see that our approximation for $\hat{\mathbf{\Theta}}^{-1}$ is sparse with only $2.5\times$ more nonzeros per row than $\hat{\mathbf{\Theta}}$. Although both methods used the same initial guess, we can see in Table I that PRL-SQUIC required fewer iterations than HP-CONCORD for the same termination tolerance. This is expected as HP-CONCORD is a first-order method with slower convergence, while the PRL-SQUIC utilizes second-order information. The total runtime of PRL-SQUIC is approximately $10\times$ faster than HP-CONCORD in this dataset.

TABLE I. Functional Magnetic Resonance Imaging Dataset Comparison.

	HP-CONCORD	PRL-SQUIC
$\|\hat{\mathbf{\Theta}}\|_0/p$	21	20
$\|\hat{\mathbf{\Theta}}^{-1}\|_0/p$	–	50
Iterations	65	6
Total compute time (sec)	4,948	419
Sample covariance matrix time (sec)	–	116
Approximate matrix inversion time (sec)	–	10
Sparse Cholesky factorization time (sec)	–	12

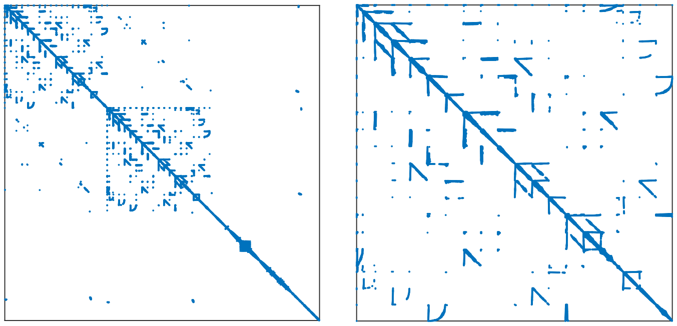


Fig. 6: Left panel—sparsity structure of recovered inverse covariance matrix (91,282 × 91,282) for fMRI dataset. Right panel—zoom in of the sparsity structure of the top left block of the recovered inverse covariance matrix (29,696 × 29,696), corresponding to the left hemisphere of the brain.

VIII. Conclusion

In this work, we present PRL-SQUIC, a highly scalable inverse covariance matrix estimation method. Based on the ℓ_1-regularized ML method, our parallel algorithm leverages distributed memory systems to recover the underlying graph structure of large-scale datasets. Building on the current advancements of ML-based methods [23], [25], [27] we introduce PRL-SQUIC, a novel hybrid `MPI-OpenMP` parallel framework, which exhibits almost ideal strong scaling up to thousands of cores. Our contributions exploit the intrinsic parallelism of various advanced sparse linear algebra techniques in SQUIC, specifically, in the sample covariance matrix and in the approximate matrix inversion of the Neumann series iteration.

For single node performance, we have compared PRL-SQUIC to BigQUIC [25] and HP-CONCORD [33], [34], [36]. The results are favorable for PRL-SQUIC with significant performance gains that reduce runtimes from hours to minutes. Furthermore, the distributed memory parallelization of PRL-SQUIC is highly efficient, with almost ideal strong scaling to up to 128 nodes. We show for the first time that PRL-SQUIC is capable of approximating sparse inverse covariance matrices of datasets up to 10 million dimensions, which is far beyond what has been reported in the literature thus far.

We also tested PRL-SQUIC on a fMRI dataset, where the recovered inverse covariance matrix encodes direct functional connectivity of different regions of the human brain. Our results show that PRL-SQUIC exhibits good strong scalability and performance for this dataset as well, while at the same time equivalent recovery of the inverse covariance matrix in terms of structure and density.

Acknowledgments

We gratefully acknowledge the collaboration of Penporn Koanantakool (UC Berkeley, Google Brain) and Sang-Yun Oh (UC Santa Barbara) for their guidance on the fMRI dataset and the comparative performance results of HP-CONCORD. We also acknowledge support by the Swiss Platform for Advanced Scientific Computing which is a structuring project supported by the Swiss Council of Federal Institutes of Technology.

References

[1] J. Fan, Y. Fan, and J. Lv, “High dimensional covariance matrix estimation using a factor model,” *Journal of Econometrics*, vol. 147, pp. 186–197, 2008.

[2] M. O. Kuismin and M. J. Sillanpää, “Estimation of covariance and precision matrix, network structure, and a view toward systems biology,” *Wiley Interdisciplinary Reviews: Computational Statistics*, vol. 9, no. 6, p. e1415, Nov 2017.

[3] J. Ye and J. Liu, “Sparse Methods for Biomedical Data.” *SIGKDD explorations : newsletter of the Special Interest Group (SIG) on Knowledge Discovery & Data Mining*, vol. 14, no. 1, pp. 4–15, Jun 2012.

[4] M. Semelhago, B. L. Nelson, A. Wächter, and E. Song, “Computational methods for optimization via simulation using gaussian markov random fields,” in *2017 Winter Simulation Conference (WSC)*, Dec 2017, pp. 2080–2091.

[5] P. J. Bickel and E. Levina, “Covariance Regularization by Thresholding,” *The Annals of Statistics*, vol. 36, no. 6, pp. 2577–2604, 2008.

[6] C. Lam and J. Fan, “Sparsistency And Rates Of Convergence In Large Covariance Matrix Estimation,” *The Annals of Statistics*, vol. 37, no. 6B, pp. 4254–4278, 2009.

[7] N. E. Karoui, “High-dimensionality Effects In The Markowitz Problem And Other Quadratic Programs With Linear Constraints: Risk Underestimation,” *The Annals of Statistics Institute of Mathematical Statistics in The Annals of Statistics*, vol. 38, no. 1, pp. 3487–3566, 2010.

[8] O. Banerjee, L. E. Ghaoui, and A. Edu, “Model Selection Through Sparse Maximum Likelihood Estimation for Multivariate Gaussian or Binary Data Alexandre d’Aspremont,” *Journal of Machine Learning Research*, vol. 9, pp. 485–516, 2008.

[9] J. Friedman, T. Hastie, and R. Tibshirani, “Sparse inverse covariance estimation with the graphical lasso,” 2007.

[10] A. J. Rothman, P. J. Bickel, E. Levina, and J. Zhu, “Sparse permutation invariant covariance estimation,” *Electronic Journal of Statistics*, vol. 2, pp. 494–515, 2008.

[11] M. Yuan and Y. Lin, “Model selection and estimation in the Gaussian graphical model,” *Biometrika*, vol. 94, pp. 19–35, 2007.

[12] L. Li and K.-C. Toh, “An inexact interior point method for l_1-regularized sparse covariance selection,” *Mathematical Programming Computation*, vol. 2, pp. 291–315, 2010.

[13] T. Cai, W. Liu, and X. Luo, “A constrained l_1 minimization approach to sparse precision matrix estimation,” *Journal of the American Statistical Association*, vol. 106, no. 494, pp. 594–607, 2011.

[14] O. Banerjee, L. E. Ghaoui, and A. d’Aspremont, “Model selection through sparse maximum likelihood estimation for multivariate gaussian or binary data,” *The Journal of Machine Learning Research*, vol. 9, pp. 485–516, 2008.

[15] A. d’Aspremont, O. Banerjee, and L. E. Ghaoui, “First-order methods for sparse covariance selection,” *SIAM J. Matrix Analysis and Applications*, vol. 30, no. 1, pp. 56–66, 2008.

[16] J. Friedman, T. Hastie, and R. Tibshirani, “Sparse inverse covariance estimation with the graphical lasso,” *Biostatistics*, vol. 9, no. 3, pp. 432–441, 2008.

[17] J. Rothman, P. Bickel, E. Levina, and J. Zhu, “Sparse permutation invariant covariance estimation,” *Electron. J. Stat.*, vol. 2, pp. 494–515, 2008.

[18] B. Rolfs, B. Rajaratnam, D. Guillot, I. Wong, and A. Maleki, “Iterative thresholding algorithm for sparse inverse covariance estimation,” *Advances in Neural Information Processing Systems*, vol. 25, pp. 1574–1582, 2012.

[19] K. Scheinberg and I. Rish, “Learning sparse Gaussian Markov networks using a greedy coordinate ascent approach,” in *Machine Learning and Knowledge Discovery in Databases*, ser. Lecture Notes in Computer Science, J. Balczar, F. Bonchi, A. Gionis, and M. Sebag, Eds. Springer Berlin / Heidelberg, 2010, vol. 6323, pp. 196–212.

[20] J. Duchi, S. Gould, and D. Koller, “Projected subgradient methods for learning sparse Gaussians,” in *Proceedings of the Twenty-Fourth Conference Annual Conference on Uncertainty in Artificial Intelligence (UAI-08)*, 2008, pp. 153–160.

[21] K. Scheinberg and I. Rish, “Learning sparse Gaussian Markov networks using a greedy coordinate ascent approach,” in *Proceedings of the 2010 European Conference on Machine Learning and Knowledge Discovery in Databases: Part III*, 2010, pp. 196–212.

[22] J. Dahl, L. Vandenberghe, and V. Roychowdhury, “Covariance selection for non–chordal graphs via chordal embedding,” *Optimization Methods and Software*, vol. 23, no. 4, pp. 501–520, 2008.

[23] C.-J. Hsieh, M. A. Sustik, I. S. Dhillon, and P. K. Ravikumar, “Sparse inverse covariance matrix estimation using quadratic approximation,” in *Advances in Neural Information Processing Systems*, J. Shawe-Taylor, R. Zemel, P. Bartlett, F. Pereira, and K. Weinberger, Eds. Neural Information Processing Systems Foundation, 2011, vol. 24, pp. 2330–2338.

[24] F. Oztoprak, J. Nocedal, S. Rennie, and P. A. Olsen, “Newton-like methods for sparse inverse covariance estimation,” *Advances in Neural Information Processing Systems*, vol. 25, pp. 755–763, 2012.

[25] C.-J. Hsieh, M. A. Sustik, I. S. Dhillon, P. K. Ravikumar, and R. A. Poldrack, “BIG & QUIC: Sparse inverse covariance estimation for a million variables,” in *Advances in Neural Information Processing Systems*, C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Weinberger, Eds. Neural Information Processing Systems Foundation, 2013, vol. 26, pp. 3165–3173.

[26] J. Ballani and D. Kressner, “Sparse inverse covariance estimation with hierarchical matrices,” EPFL Technical Report, Tech. Rep., 2014.

[27] anonymous omitted due to double-blind review, “anonymous,” *anonymous*, anonymous.

[28] A. Ali, K. Khare, S.-Y. Oh, and B. Rajaratnam, “Generalized Pseudo-likelihood Methods for Inverse Covariance Estimation,” in *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*, ser. Proceedings of Machine Learning Research, A. Singh and J. Zhu, Eds., vol. 54. Fort Lauderdale, FL, USA: PMLR, 20–22 Apr 2017, pp. 280–288.

[29] J. Peng, P. Wang, N. Zhou, and J. Zhu, “Partial Correlation Estimation by Joint Sparse Regression Models,” *Journal of the American Statistical Association*, vol. 104, no. 486, pp. 735–746, Jun 2009.

[30] G. V. Rocha, P. Zhao, and B. Yu, “A path following algorithm for Sparse Pseudo-Likelihood Inverse Covariance Estimation (SPLICE),” *ArXiv e-prints*, Jul. 2008.

[31] J. Friedman, T. Hastie, and R. Tibshirani, “Applications of the lasso and grouped lasso to the estimation of sparse graphical models,” Stanford Technical Report, Tech. Rep., 2010.

[32] K. Khare, S. Oh, and B. Rajaratnam, “A convex pseudolikelihood framework for high dimensional partial correlation estimation with convergence guarantees,” *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, vol. 77, no. 4, pp. 803–825.

[33] S. Oh, O. Dalal, K. Khare, and B. Rajaratnam, “Optimization methods for sparse pseudo-likelihood graphical model selection,” in *NIPS 27*, 2014, pp. 667–675.

[34] P. Koanantakool, A. Ali, A. Azad, A. Buluc, D. Morozov, L. Oliker, K. Yelick, and S.-Y. Oh, “Communication-avoiding optimization methods for distributed massive-scale sparse inverse covariance estimation,” in *Proceedings of the Twenty-First International Conference on Artificial Intelligence and Statistics*, ser. Proceedings of Machine Learning Research, A. Storkey and F. Perez-Cruz, Eds., vol. 84. Playa Blanca, Lanzarote, Canary Islands: PMLR, 09–11 Apr 2018, pp. 1376–1386.

[35] S. M. Smith, C. F. Beckmann, J. Andersson, E. J. Auerbach, J. Bijsterbosch, G. Douaud, E. Duff, D. A. Feinberg, L. Griffanti, M. P. Harms, M. Kelly, T. Laumann, K. L. Miller, S. Moeller, S. Petersen, J. Power, G. Salimi-Khorshidi, A. Z. Snyder, A. T. Vu, M. W. Woolrich, J. Xu, E. Yacoub, K. UÄurbil, D. C. V. Essen, and M. F. Glasser, “Resting-state fmri in the human connectome project,” *NeuroImage*, vol. 80, pp. 144 – 168, 2013, mapping the Connectome.

[36] P. Koanantakool, A. Azad, A. Buluç, D. Morozov, S. Y. Oh, L. Oliker, and K. Yelick, “Communication-avoiding parallel sparse-dense matrix-matrix multiplication,” in *2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, May 2016, pp. 842–853.

[37] P. Tseng, S. Yun, P. Tseng, and S. Yun, “A coordinate gradient descent method for nonsmooth separable minimization,” *Math. Program., Ser. B*, vol. 117, pp. 387–423, 2009.

[38] S. Yun, K.-C. Toh, S. Yun, and K.-C. Toh, “A coordinate gradient descent method for 1 -regularized convex minimization,” *Comput Optim Appl*, vol. 48, pp. 273–307, 2011.

[39] T. A. Davis, S. Rajamanickam, and W. M. Sid-Lakhdar, “A survey of direct methods for sparse linear systems,” *Acta Numerica*, 2016.

[40] T. A. Davis, *Direct Methods for Sparse Linear Systems (Fundamentals of Algorithms 2)*. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 2006.

[41] O. Kaya, E. Kayaaslan, B. Uçar, and I. S. Duff, “Fill-in reduction in sparse matrix factorizations using hypergraphs,” INRIA, Research Report RR-8448, Jan. 2014.

[42] Y. Chen, T. A. Davis, W. W. Hager, S. Rajamanickam, and W. W. Hager, “Algorithm 887: CHOLMOD, Supernodal Sparse Cholesky Factorization and Update/Downdate,” *ACM Trans. Math. Softw*, vol. 35, no. 14, 2008.

[43] P. Hénon, P. Ramet, and J. Roman, “PaStiX: A High-Performance Parallel Direct Solver for Sparse Symmetric Definite Systems,” *Parallel Computing*, vol. 28, no. 2, pp. 301–321, 2002.

[44] D. Irony, G. Shklarski, and S. Toledo, “Parallel and fully recursive multifrontal supernodal sparse Cholesky,” *Future Generation Computer Systems — Special issue: Selected numerical algorithms archive*, vol. 20, no. 3, pp. 425–440, 2004.

[45] J. K. P. R. Amestoy, I. S. Duff and J.-Y. L’Excellent, “A fully asynchronous multifrontal solver using distributed dynamic scheduling,” *SIAM Journal of Matrix Analysis and Applications*, vol. 23, no. 1, pp. 15–41, 2001.

[46] O. Schenk and K. Gärtner, “Solving unsymmetric sparse systems of linear equations with PARDISO,” *Journal of Future Generation Computer Systems*, vol. 20, no. 3, pp. 475–487, 2004.

[47] S. B. Eickhoff, B. Thirion, G. Varoquaux, and D. Bzdok, “Connectivity-Based Parcellation: Critique and Implications,” *Human Brain Mapping*, p. 22, Jan. 2016.

[48] G. Marrelec, A. Krainik, H. Duffau, M. PÃ©lÃ©grini-Issac, S. LehÃ©ricy, J. Doyon, and H. Benali, “Partial correlation for functional brain interactivity investigation in functional mri,” *NeuroImage*, vol. 32, no. 1, pp. 228 – 237, 2006.