Date of publication xxxx 00, 0000, date of current version xxxx 00, 0000.

Digital Object Identifier 10.1109/ACCESS.2017.DOI

Exploiting Block Structures of KKT Matrices for Efficient Solution of Convex Optimization Problems

ZAFAR IQBAL [1], (Member, IEEE), SAEID NOOSHABADI [1], (Senior Member, IEEE), ICHITARO YAMAZAKI [2], (Member, IEEE), STANIMIRE TOMOV [2], (Senior Member, IEEE), and JACK DONGARRA [2], (Fellow, IEEE)

[1]Institute of Computing and Cybersystems, College of Computing, Michigan Technological University, Houghton, MI, USA 49931, emails: zafari,saeid@mtu.edu
[2]Innovative Computing Laboratory, College of Engineering, University of Tennessee, Knoxville, TN, USA 37996, emails: iyamazaki,tomov,dongarra@icl.utk.edu

Corresponding author: Saeid Nooshabadi (e-mail: saeid@mtu.edu).

This work was supported by National Science Foundation grant, Award No. 1709069

ABSTRACT Convex optimization solvers are widely used in the embedded systems that require sophisticated optimization algorithms including model predictive control (MPC). In this paper, we aim to reduce the online solve time of such convex optimization solvers so as to reduce the total runtime of the algorithm and make it suitable for real-time convex optimization. We exploit the property of the Karush–Kuhn–Tucker (KKT) matrix involved in the solution of the problem that only some parts of the matrix change during the solution iterations of the algorithm. Our results show that the proposed method can effectively reduce the runtime of the solvers.

INDEX TERMS Convex optimization, linear solver, Karush–Kuhn–Tucker (KKT), embedded systems.

I. INTRODUCTION

CONVEX optimization has emerged as an important mathematical tool in a wide range of science and engineering disciplines such as automatic control, machine learning, and statistical signal processing *etc*. Recent advances including those mentioned in [1], [2] has enabled its use as realtime solvers for embedded systems [3], [4], [5].

Unlike general-purpose solvers, a realtime embedded optimization imposes special requirements on the solver [6]. For example, it is critical that the linear solver obtains the solution of the desired accuracy within a specified amount of time at each step of solving the optimization problem. The solution time may have to be shorter than the sample rate of the embedded system, *e.g.*, tens to millions of samples per second. The solver must be robust. It is not acceptable for the solver to fail due to a fatal error such as division-by-zero or unreliable sensors. Furthermore, the solver should use simple code with a minimal dependency on dynamic libraries. On the other hand, the general-purpose solvers often depend on either an integrated environment like MATLAB, PYTHON, or external libraries such as basic linear algebra subprograms (BLAS) [7] and linear algebra package (LAPACK) [8], and Intel math kernel library (MKL) [9] for their programmability, usability, and performance. This makes it difficult to validate and port the solver for use in embedded applications. In addition these solvers run with human intervention and can fail occasionally.

Two recent works, CVXGEN [10] and ECOS [11] provide frameworks that generate a specific code for solving realtime convex optimization problems on an embedded system. Specifically, given a high-level description of the optimization problem, CVXGEN generates a simple, flat, and library-free C code using disciplined convex programming (DCP) [12]. The generated code is branch free and suitable for an embedded system, and can be compiled into a high performance solver for the specific family of the problems (e.g., all the matrices have the same sparsity structure). However, to meet the strict constraint enforced on the solution time in realtime applications, the dimension of the coefficient matrix for the linear system is currently limited to $O(100)$ in CVXGEN.

While embedded solvers come with certain requirements, they have certain features that can be exploited to reduce the complexity of the design. The accuracy required by the embedded solvers is often limited. For example, with model predictive control (MPC), even very low accuracy can result in acceptable control performance [13]. Another feature is that the structures of many problems do not change from one solve to the next, e.g., for Kalman filtering, the dimensions and structure of the system state, input and output vectors, and steady-state error covariance matrix are all fixed. Furthermore, system parameters remain unchanged with each realtime solution iteration. Therefore, each solver will perform many solves for a given problem instance (including the input data). Finally, the change in the numerical values of the solver parameters between two subsequent instances of the problem may be small. These features of realtime optimization provide the opportunity to significantly reduce the solution time. While CVXGEN generates the convex optimization solvers that take advantage of the structure of the problem family, e.g. the sparsity structure of the resulting Karush–Kuhn–Tucker (KKT) matrices [14], they fail to take the advantage of the fact that several blocks in the matrix do not change during the iterations of a given solve instance.

In this work, we aim to reduce the time to solve the family of the linear systems by taking advantage of the fact that many blocks of the KKT matrices do not change during the solution of the convex optimization problem [15].

II. QUADRATIC PROGRAMMING CONVEX PROBLEM

In DCP [16], [10], a quadratic programming (QP) convex problem is transformed into a standard form as follows,

$$\begin{aligned} &\text{minimize } \left(\frac{1}{2}\right) x^T Q x + q^T x \\ &\text{subject to } Gx \preceq h \text{ and } Ax = b \end{aligned} \tag{1}$$

where $x, q \in \mathbf{R}^n$, $Q \in \mathbf{S}^n_+ \succeq 0$ is a symmetric positive semidefinite matrix, $A \in \mathbf{R}^{m\times n}$, $G \in \mathbf{R}^{p\times n}$, $b \in \mathbf{R}^m$, and $h \in \mathbf{R}^p$. With each instance of the solve, the solver goes through several iterations of the solve until solution meets a certain level of pre-determined accuracy or the maximum number of iterations are reached. The iteration time of the optimization solver is dominated by the solution of the KKT linear system of equations, $Kx = c$, whose coefficient matrix K has the following block structure,

$$K = \left[\begin{array}{cc|cc} Q & 0 & G^T & A^T \\ 0 & S^{-1}Z & I_p & 0 \\ \hline G & I_p & 0 & 0 \\ A & 0 & 0 & 0 \end{array}\right] \tag{2}$$

where I_p is the $p \times p$ identity matrix. In addition, for QP, the matrix K is quasisemidefinite (i.e. a symmetric matrix with (1,1) block diagonal positive semidefinite and (2,2) block a negative semidefinite block [2]), where $S = diag(s) \in \mathbf{R}^{p\times p}$ and $Z = diag(z) \in \mathbf{R}^{p\times p}$ are diagonal matrices, with $s \in \mathbf{R}^p$ and $z \in \mathbf{R}^p$, respectively, representing the slack variables and inequality multipliers in the KKT conditions. This special structure of KKT matrix is most interior-point methods [17]. To guarantee a reliable and stable performance, even for an ill-conditioned K, the linear solvers use the combination of static and dynamic regularization and iterative refinements [10]. The regularization is achieved through scalar parameter $\epsilon < 0$ to make the matrix K symmetric quasidefinite (e.g., $\epsilon = -10^{-7}$). The solution to the linear system $Kx = c \left(c \in \mathbf{R}^{n+m+2p}\right)$ is found through the LDL$^\text{T}$ decomposition, $PKP^T = LDL^T$, where P is a permutation matrix, L is a lower triangular matrix with unit diagonals, and D is a diagonal matrix. With the LDL$^\text{T}$ decomposition, the solution to $Kx = c$ is found through the sequence of forward substitution, diagonal scaling, and backward substitution.

In this paper, we aim to reduce the cost of solving the KKT linear systems by taking advantage of the property that only some of the sub-matrices in the KKT matrices change during the solution of the convex optimization problem. For example, for the online array weight design or for the adaptive filtering, only the matrix G in (1) change from one solve instance to the next. In many cases that we have studied, only the sub-matrix $S^{-1}Z$ changes from one iteration of one solve instance to the next [15].

III. ALGORITHM

We focus on the cases where only $S^{-1}Z$ changes during each iteration of a solution. Example of such applications include Kalman filtering, and sliding window smoothing and estimation [6]. We take advantage of the fixed Q, A, and G blocks. The matrix K is implicitly reordered as $\hat{K}$ and the resulting equivalent system $\hat{K}\hat{w} = \hat{c}$ is solved as follows,

$$\begin{bmatrix} Q & A^T & 0 & G^T \\ A & 0 & 0 & 0 \\ 0 & 0 & S^{-1}Z & I_p \\ G & 0 & I_p & 0 \end{bmatrix} \begin{bmatrix} \hat{w}_1 \\ \hat{w}_4 \\ \hat{w}_2 \\ \hat{w}_3 \end{bmatrix} = \begin{bmatrix} \hat{c}_1 \\ \hat{c}_4 \\ \hat{c}_2 \\ \hat{c}_3 \end{bmatrix} \tag{3}$$

A. INITIAL OFFLINE SETUP

The initial offline setup steps are the same as proposed in our previous work [15]. During the initial offline setup stage, we partially factorize the matrix K such that $K = LDL^T$, where

$$LD = \begin{bmatrix} L_{1,1} & 0 & 0 & 0 \\ L_{2,1} & L_{2,2} & 0 & 0 \\ 0 & 0 & I_p & 0 \\ L_{4,1} & L_{4,2} & 0 & I_p \end{bmatrix} \begin{bmatrix} D_{1,1} & 0 & 0 & 0 \\ 0 & D_{2,2} & 0 & 0 \\ 0 & 0 & S^{-1}Z & I_p \\ 0 & 0 & I_p & C \end{bmatrix}. \tag{4}$$

In the above the trailing block (2, 2) D is not yet fully factorized, and therefore, . This block will be factorized in the online factorization procedure. The above partial LDL^T factorization is computed as follows,

1) We compute the LDL^T factorization of Q such that $Q = L_{1,1}D_{1,1}L_{1,1}^T$.
2) We compute the off-diagonal blocks $L_{2,1}$ and $L_{4,1}$ in the first block column of L such that $L_{2,1} := A\left(D_{1,1}L_{1,1}^T\right)^{-1}$ and $L_{4,1} := G\left(D_{1,1}L_{1,1}^T\right)^{-1}$.
3) We compute the LDL^T factorization of the second diagonal block $\tilde{K}_{2,2}$ such that $\tilde{K}_{2,2} = L_{2,2}D_{2,2}L_{2,2}^T$, where $\tilde{K}_{2,2} := -\left(L_{2,1}D_{1,1}L_{2,1}^T\right)$. $\tilde{K}$ is used to distinguish the block from the corresponding block of the original matrix K.
4) We compute the off-diagonal block $L_{4,2}$ in the second block column of L such that $L_{4,2} := \tilde{K}_{4,2}\left(D_{2,2}L_{2,2}^T\right)^{-1}$, where $\tilde{K}_{4,2} := -\left(L_{4,1}D_{1,1}L_{2,1}^T\right)$.

5) We compute the last diagonal block C of D such that $C := -\left(L_{4,1}D_{1,1}L_{4,1}^T\right) - \left(L_{4,2}D_{2,2}L_{4,2}^T\right)$.

B. ONLINE FACTORIZATION

During the solution of the convex optimization problem, we only need to factorize the Schur complement of the matrix, given as follows,

$$\begin{bmatrix} S^{-1}Z & I_p \\ I_p & C \end{bmatrix} = \begin{bmatrix} I_p & 0 \\ L_{4,3} & L_{4,4} \end{bmatrix} \begin{bmatrix} D_{3,3} & 0 \\ 0 & D_{4,4} \end{bmatrix} \begin{bmatrix} I_p & L_{4,3}^T \\ 0 & L_{4,4}^T \end{bmatrix} \tag{5}$$

where $D_{3,3} = S^{-1}Z$, $L_{4,3} = Z^{-1}S$, $\tilde{C} = L_{4,4}D_{4,4}L_{4,4}^T$, and $\tilde{C} = C - \left(Z^{-1}S\right)$. Since both S and Z are diagonal matrices, the LDL^T factorization of $\tilde{C}$ is computationally the most expensive part of this factorization.

In this work, we attempt to improve the speed of the LDL^T factorization of $\tilde{C}$ by adapting a numerically stable product-form Cholesky factorization algorithm [18]. The LDL^T factorization of $\tilde{C}$ can be computed by updating the contribution from the $S^{-1}Z$ part of $\tilde{C}$ on each solve iteration, instead of the full LDL^T refactorization of $C + S^{-1}Z$.

We next present the technique to update the LDL^T factorization of a matrix on each solve iteration, when a symmetric matrix of the form IWI^T is added to it, where $W \in \mathbf{R}^{p\times p}$ is a diagonal matrix and $I \in \mathbf{R}^{p\times p}$ is an identity matrix. We rewrite the second term of $\tilde{C} = C - \left(Z^{-1}S\right)$ to $I\left(Z^{-1}S\right)I^T$ and set $W = -Z^{-1}S$. With also represent C in its factorized form as $C = LDL^T$. Assuming that $LDL^T + IWI^T$ is positive semidefinite, and since L is nonsingular, the updated LDL^T factorization of $\tilde{C}$ is given as,

$$\begin{aligned} \tilde{C} &= C - I\left(Z^{-1}S\right)I^T = LDL^T + IWI^T \\ &= L\left(D + YWY^T\right)L^T = L\tilde{L}\tilde{D}\tilde{L}^T L^T \end{aligned} \tag{6}$$

where $D + YWY^T = \tilde{L}\tilde{D}\tilde{L}^T$ represents the Cholesky factorization, $Y \in \mathbf{R}^{p\times p}$ is the solution of $LY = I$, (or equivalently $Y = L^{-1}$). The matrix inversion of L^{-1} needs to be computed only once in the offline setup phase and then used in the online solve process. Here $D \in \mathbf{R}^{p\times p}$ is a diagonal matrix which is also computed once in the offline setup phase. The computation of $\tilde{L}\tilde{D}\tilde{L}^T$ is performed by adapting the iterative rank-k update algorithm proposed by [18].

1) Rank-k Update

Let the transpose of $y^{(j)} \in \mathbf{R}^p$ be the jth row of Y and $Y^{(j)} \in \mathbf{R}^{(p-j)\times p}$ be the matrix consisting of the last $p-j$ rows of Y, let $D^{(j)} \in \mathbf{R}^{(p-j)\times(p-j)}$ be a diagonal matrix with diagonal elements $d_{j+1}, ..., d_p$, therefore $D^{(0)} = D$, and let $\Sigma_0 = Z^{-1}S$. Therefore,

$$\begin{aligned} Y^{(0)} &= Y \text{ and} \\ Y^{(j-1)^T} &= \left[y^{(j)}, Y^{(j)^T}\right] \end{aligned} \tag{7}$$

We use symmetric Gaussian elimination to compute the factorization of $D + Y\Sigma_0 Y^T$ and after the first step, the factorization state is given in (8), where $\tilde{d}_1 = d_1 + y^{(1)^T}\Sigma_0 y^{(1)}$, $\Sigma_1 = \Sigma_0 - \tilde{d}_1 q^{(1)} q^{(1)^T}$, and $q^{(j)} \in \mathbf{R}^p$ is given as,

$$q^{(1)} = \begin{cases} \left(\frac{1}{\tilde{d}_1}\right)\Sigma_0 y^{(1)}, & \text{if } \tilde{d}_1 \neq 0 \\ 0, & \text{if } \tilde{d}_1 = 0 \end{cases}. \tag{9}$$

Fig. 1 shows graphically, the computations involved in the first iteration of the factorization as given in (8) for the symmetric Gaussian elimination, $D^{(0)} + Y^{(0)}\Sigma_0 Y^{(0)^T}$ process. Fig. 1(a) shows the computation of the vector $r^{(1)}$ which is then used in subsequent computations. Since this is a multiplication of $\sigma_1^{(0)}$, which is the first diagonal element of Σ_0 by 1, therefore, there is no operation performed in this step. Fig. 1(b) shows the computation which results in the element $\tilde{d}_1 = d_1 + y^{(1)^T}\Sigma_0 y^{(1)}$ of $\tilde{D}$. Since $y^{(1)^T}$ is the first row of Y and Y being the inverse of a lower triangular matrix, only the first element of $y^{(1)^T}$ is 1 and the rest are zero. We can observe that this computation reduces to a scalar addition between two terms $r_1^{(1)}$ and d_1, as follows,

$$\tilde{d}_1 = d_1 + \left(y_1^{(1)^T} r_1^{(1)}\right). \tag{10}$$

Fig. 1(c) shows the computation of the vector $q^{(1)}$ of $\tilde{L}$ in the first iteration. Along the same lines, we can also observe that this computation requires a scalar division of $r_1^{(1)}$ by the term $\tilde{d}_1$ as follows,

$$q^{(1)} = \frac{1}{\tilde{d}_1} r_1^{(1)}. \tag{11}$$

Fig. 1(d) shows the computation of the matrix Σ_1 in the first iteration. Since at this stage, the vectors $q^{(1)}$ and $r^{(1)^T}$ both have only one element as nonzero, we can see that this computation also requires a scalar multiplication of two terms $q_1^{(1)}$, and $r_1^{(1)^T}$, and the resulting scalar term is subtracted from the matrix Σ_0, which is essentially a scalar operation since we only need to subtract from the first diagonal element, $\sigma_1^{(0)}$ because other elements will remain unchanged at this stage. The first diagonal element $\sigma_1^{(1)}$ of Σ_1 is computed as follows,

$$\sigma_1^{(1)} = \sigma_1^{(0)} - \left(q_1^{(1)} r_1^{(1)^T}\right). \tag{12}$$

Fig. 2 shows graphically, the computations involved in the second iteration after (8). Fig. 2(a) shows the computation of the vector $r^{(2)}$ where $y(2)$ now has a nonzero element $y_1^{(2)}$ and a 1 at $y_2^{(2)}$. We notice that Σ_1 is still a diagonal matrix but with the first diagonal element $\sigma_1^{(1)}$, updated in the previous iteration. By inspection, we can see that this requires a single multiplication of $\sigma_1^{(1)}$, which is the first diagonal element of Σ_0 by $y_1^{(2)}$. Fig. 2(b) shows the second step of symmetric Gaussian elimination, $D^{(1)} + Y^{(1)}\Sigma_1 Y^{(1)^T}$ computation, which results in the element $\tilde{d}_2 = d_2 + y^{(2)^T} r^{(2)}$ of $\tilde{D}$. Since $y^{(2)^T}$ is the second row of the lower triangular matrix Y, only the first two elements are required to compute this term. Therefore, the number of computation required in this step include one scalar multiplication between $y_1^{(2)^T}$, and $r_1^{(2)}$, and two scalar additions as follows,

$$\tilde{d}_2 = d_2 + \left(y_1^{(2)^T} r_1^{(2)} + r_2^{(2)}\right). \tag{13}$$

$$D^{(0)} + Y^{(0)}\Sigma_0 Y^{(0)^T} = \begin{bmatrix} d_1 + y^{(1)^T}\Sigma_0 y^{(1)} & y^{(1)^T}\Sigma_0 Y^{(1)^T} \\ Y^{(1)}\Sigma_0 y^{(1)} & D^{(1)} + Y^{(1)}\Sigma_0 Y^{(1)} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ Y^{(1)}q^{(1)} & I \end{bmatrix} \begin{bmatrix} \tilde{d}_1 & 0 \\ 0 & D^{(1)} + Y^{(1)}\Sigma_1 Y^{(1)^T} \end{bmatrix} \begin{bmatrix} 1 & q^{(1)}Y^{(1)^T} \\ 0 & I \end{bmatrix} \tag{8}$$

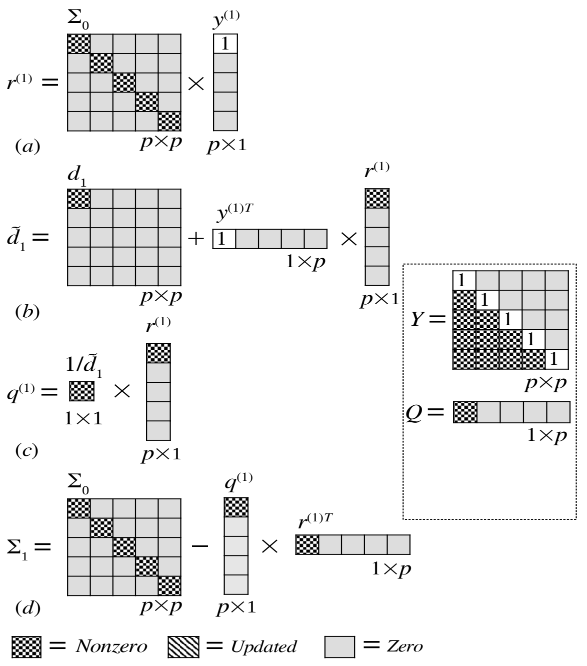


FIGURE 1. Graphical representation of the computations involved in the first iteration of the recurrence relations given in (20).

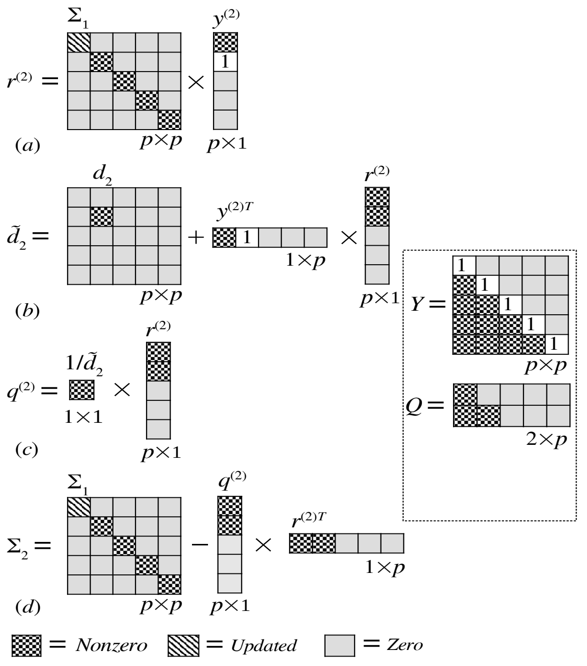


FIGURE 2. Graphical representation of the computations involved in the second iteration of the recurrence relations given in (20).

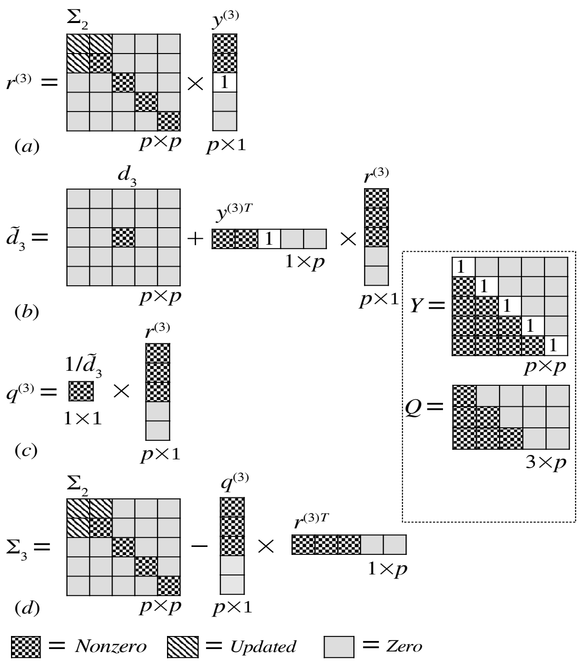


FIGURE 3. Graphical representation of the computations involved in the third iteration of the recurrence relations given in (20).

Fig. 2(c) shows the computation of the vector $q^{(2)}$ of $\tilde{L}$ in the second iteration. In order to compute the first two elements of $q^{(2)}$, we require the first two elements of $r^{(2)}$, for division by the scalar term $\tilde{d}_2$ as follows,

$$q^{(2)} = \frac{1}{\tilde{d}_2}\begin{bmatrix} r_1^{(2)} \\ r_2^{(2)} \end{bmatrix}. \tag{14}$$

Fig. 2(d) shows the computation of the matrix Σ_2 in the second iteration. At this stage the vectors $q^{(2)}$ and $r^{(2)^T}$ both have the first two elements as nonzero, and we require four multiplications to compute the resulting 2×2 matrix $q^{(2)}r^{(2)^T}$, and then subtracting the resulting matrix from Σ_1. This computation results in Σ_2 with the initial 2×2 diagonal block updated and the rest of the diagonal elements remain unchanged as follows,

$$\Sigma_2 = \Sigma_1 - \begin{bmatrix} q_1^{(2)}r_1^{(2)^T} & q_1^{(2)}r_2^{(2)^T} & \cdots \\ q_2^{(2)}r_1^{(2)^T} & q_2^{(2)}r_2^{(2)^T} & \cdots \\ \vdots & \vdots & \ddots \end{bmatrix}. \tag{15}$$

Fig. 3 shows graphically, the computations involved in the third iteration of the factorization. Fig. 3(a) shows the computation of the vector $r^{(3)}$ where $y(3)$ now has two nonzero elements $y_1^{(3)}$, $y_2^{(3)}$ and a 1 at $y_3^{(3)}$. We notice that Σ_2 now has the initial 2×2 block of nonzero elements updated in the previous iteration. By inspection, we can see that this computation requires four multiplications and two additions as follows,

$$r^{(3)} = \begin{bmatrix} \sigma^{(2)}_{(1,1)} y^{(3)}_1 + \sigma^{(2)}_{(1,2)} y^{(3)}_2 \\ \sigma^{(2)}_{(2,1)} y^{(3)}_1 + \sigma^{(2)}_{(2,2)} y^{(3)}_2 \\ \sigma^{(2)}_{(3,3)} \end{bmatrix}. \tag{16}$$

Fig. 3(b) shows the third step of symmetric Gaussian elimination, $D^{(2)} + Y^{(2)} \Sigma_2 Y^{(2)^T}$ computation, which results in the element $\tilde{d}_3 = d_3 + y^{(3)^T} r^{(3)}$ of $\tilde{D}$. Therefore, computing the term $\tilde{d}_3$ requires two scalar multiplications and three additions as follows,

$$\tilde{d}_3 = d_3 + \left(y^{(3)^T}_1 r^{(3)}_1 + y^{(3)^T}_2 r^{(3)}_2 + r^{(3)}_3 \right) \tag{17}$$

Fig. 3(c) shows the computation of the vector $q^{(3)}$ of $\tilde{L}$ in the third iteration. Here, we require three divisions as follows,

$$q^{(3)} = \frac{1}{\tilde{d}_3} \begin{bmatrix} r^{(3)}_1 \\ r^{(3)}_2 \\ r^{(3)}_3 \end{bmatrix}. \tag{18}$$

Fig. 3(d) shows the computation of the matrix Σ_3 in the third iteration. At this stage the vectors $q^{(3)}$ and $r^{(3)^T}$ both have the first three elements as nonzero, and we require nine multiplications to compute the resulting 3×3 matrix $q^{(3)} r^{(3)^T}$, and then subtracting the resulting matrix from Σ_2. This computation results in Σ_3 with the initial 3×3 diagonal block updated and the rest of the diagonal elements remain unchanged as follows,

$$\Sigma_3 = \Sigma_2 - \begin{bmatrix} q^{(3)}_1 r^{(3)^T}_1 & q^{(3)}_1 r^{(3)^T}_2 & q^{(3)}_1 r^{(3)^T}_3 & \cdots \\ q^{(3)}_2 r^{(3)^T}_1 & q^{(3)}_2 r^{(3)^T}_2 & q^{(3)}_2 r^{(3)^T}_3 & \cdots \\ q^{(3)}_3 r^{(3)^T}_1 & q^{(3)}_3 r^{(3)^T}_2 & q^{(3)}_3 r^{(3)^T}_3 & \cdots \\ \vdots & \vdots & \vdots & \ddots \end{bmatrix}. \tag{19}$$

Similarly, each corresponding term of D and a column of Y is being used to compute a term $\tilde{d}$ for $\tilde{D}$ and the subsequent steps result in the computation of column vector $q^{(j)}$, which is then used in the formation of $\tilde{L}$. In the next steps, the $(p-j) \times (p-j)$ matrix $D^{(j)} + Y^{(j)} \Sigma_j Y^{(j)^T}$ is identical in form to the matrix $D^{(0)} + Y^{(0)} \Sigma_0 Y^{(0)^T}$. Therefore, by performing the above-mentioned symmetric Gaussian elimination procedure on the matrix $D^{(j)} + Y^{(j)} \Sigma_j Y^{(j)^T}$ for $j = 1, 2, ..., p-2$, we obtain the factorization $\tilde{L}\tilde{D}\tilde{L}^T$ give in (6), where $\tilde{L}$ is given in (24) and $\tilde{D} = \text{diag}\{\tilde{d}_1, ..., \tilde{d}_p\}$. The terms $q^{(j)}$ and $\tilde{d}_j$ can be computed using the following recurrence relations, proposed in [18]. Note that the term $\Sigma_{j-1} y^{(j)}$ is used twice in the iterations, thus we compute it only once as $r^{(j)}$ and it results in eliminating the multiplication of the term $\tilde{d}_j$ to compute Σ_j.

$$\begin{aligned} \Sigma_0 &= W, r^{(1)} = \Sigma_0 y^{(1)} \\ \tilde{d}_1 &= d_1 + y^{(1)^T} r^{(1)} \\ &\text{for } j = 1, 2, ..., p-1 \\ &\quad q^{(j)} = \begin{cases} \left(\frac{1}{\tilde{d}_j}\right) r^{(j)}, & \text{if } \tilde{d}_j \neq 0 \\ 0, & \text{if } \tilde{d}_j = 0 \end{cases} \\ &\quad \Sigma_j = \Sigma_{j-1} - q^{(j)} r^{(j)^T} \\ &\quad r^{(j+1)} = \Sigma_j y^{(j+1)} \\ &\quad \tilde{d}_{j+1} = d_{j+1} + y^{(j+1)^T} r^{(j+1)} \end{aligned} \tag{20}$$

2) Rank-1 Update

The term $LDL^T + IWI^T$ given in (6) can also be computed using the rank-1 update mechanism performing a sequence of p rank-1 updates. We let $I = [e_1, e_2, e_3, ..., e_p]$, $\tilde{D}_0 = D$, and $\tilde{L}_0 = L$, then,

$$\begin{aligned} \tilde{C} &= LDL^T + \sum_{j=1}^{p} w_j e_j e_j^T \\ &= \tilde{L}_0 \tilde{L}_1 \tilde{L}_2 ... \tilde{L}_p \tilde{D}_p \tilde{L}_p^T ... \tilde{L}_2^T \tilde{L}_1^T \tilde{L}_0^T \end{aligned} \tag{21}$$

where $j = 1, 2, ..., p$. Each successive $\tilde{L}_j \tilde{D}_j \tilde{L}_j^T$ is the result of Cholesky factorization of $\tilde{D}_{j-1} + w_j y^{(j)} y^{(j)^T}$ as given below,

$$\tilde{D}_{j-1} + w_j y^{(j)} y^{(j)^T} = \tilde{L}_j \tilde{D}_j \tilde{L}_j^T \tag{22}$$

and each $y^{(j)} \in \mathbf{R}^p$ is obtained from the solution of $\tilde{L}_0 \tilde{L}_1 \tilde{L}_2 ... \tilde{L}_{j-1} y^{(j)} = e_j$. In order to compute $q^{(j)}$ and $\tilde{d}_j$, we use the rank-1 update mechanism proposed in [18]. Let us use y to denote the vector $y^{(j)}$, y_j to denote its j^{th} component. Therefore, the vectors $y^{(j)}$, $q^{(j)}$, and matrices Σ_j and W become scalars, denoted by y_i, q_i, σ_i, and w_j respectively. The following algorithm is used to obtain the terms q_j and $\tilde{d}_j$ for each successive $\tilde{L}_j$ in computing (21).

$$\begin{aligned} &\text{for } j = 1, 2, ..., p \\ &\quad \text{Compute } y^{(j)}: \\ &\qquad y^{(j)} = \tilde{L}_{j-1}^{-1}, \tilde{L}_{j-2}^{-1}, ..., \tilde{L}_0^{-1} e_j \\ &\quad \text{Compute } q_i, \tilde{d}_i: \\ &\qquad y = y^{(j)}, \sigma_0 = w_j, r_1 = \sigma_0 y_1 \\ &\qquad \tilde{d}_1 = d_1 + y_1 r_1 \\ &\qquad \text{for } i = 1, 2, ..., p-1 \\ &\qquad\quad q_i = \begin{cases} \left(\frac{1}{\tilde{d}_i}\right) r_i, & \text{if } \tilde{d}_i \neq 0 \\ 0, & \text{if } \tilde{d}_i = 0 \end{cases} \\ &\qquad\quad \sigma_i = \sigma_{i-1} - q_i r_i \\ &\qquad\quad r_{i+1} = \sigma_i y_{i+1} \\ &\qquad\quad \tilde{d}_{i+1} = d_{i+1} + y_{i+1} r_{i+1} \end{aligned} \tag{23}$$

Thus, we can rewrite (24) as given in (25).

$$\tilde{L} = \tilde{L}\left(Y^{(1)}, Q\right) = \begin{bmatrix} 1 & & & & & \\ {y^{(2)}}^T q^{(1)} & 1 & & & & \\ {y^{(3)}}^T q^{(1)} & {y^{(3)}}^T q^{(2)} & 1 & & & \\ \vdots & \vdots & \vdots & \ddots & & \\ {y^{(p-1)}}^T q^{(1)} & {y^{(p-1)}}^T q^{(2)} & {y^{(p-1)}}^T q^{(3)} & \cdots & 1 & \\ {y^{(p)}}^T q^{(1)} & {y^{(p)}}^T q^{(2)} & {y^{(p)}}^T q^{(3)} & \cdots & {y^{(p)}}^T q^{(p-1)} & 1 \end{bmatrix} \tag{24}$$

$$\tilde{L} = \tilde{L}\left(y, q\right) = \begin{bmatrix} 1 & & & & & \\ y_2 q_1 & 1 & & & & \\ y_3 q_1 & y_3 q_2 & 1 & & & \\ \vdots & \vdots & \vdots & \ddots & & \\ y_{(p-1)} q_1 & y_{(p-1)} q_2 & y_{(p-1)} q_3 & \cdots & 1 & \\ y_p q_1 & y_p q_2 & y_p q_3 & \cdots & y_p q_{(p-1)} & 1 \end{bmatrix} \tag{25}$$

TABLE 1. Computational Complexity of the Algorithm in FLOPS

Computing Step	FLOPS, proposed method	FLOPS, method in [15]
Offline Setup:		
LDL^T of Q	$\frac{2}{3}n^3 + \frac{1}{3}n - 1$	$\frac{2}{3}n^3 + \frac{1}{3}n - 1$
$L_{2,1}$	$n^2 m$	$n^2 m$
LDL^T of $\tilde{K}_{2,2}$	$m^2 n + \frac{2}{3}m^3 + \frac{1}{3}m - 1$	$m^2 n + \frac{2}{3}m^3 + \frac{1}{3}m - 1$
$L_{4,1}$	$n^2 p$	$n^2 p$
$L_{4,2}$	$2pmn + m^2 p$	$2pmn + m^2 p$
C	$p^2\left(n+m\right)$	$p^2\left(n+m\right)$
LDL^T of C	$\frac{2}{3}p^3 + \frac{1}{3}p - 1$	-
Y	p^3	-
Online Factorization:		
$\tilde{L}\left(Y, Q\right)$	$\frac{4k^3 - 3k^2 + 2k}{3}$	$\frac{2}{3}p^3 + \frac{1}{3}p - 1$

TABLE 2. Computational Complexity in FLOPS with Varying k. k drops to 1 Typically on the 7th Iteration

Dimension of KKT	Rank-k Update				Method in [15]			
	Offline Setup	Online Factor	Online Solve	Online Total	Offline Setup	Online Factor	Online Solve	Online Total
13 (p=1)	915	10	0	10	913	10	0	10
26 (p=2)	7,296	40	12	52	7,282	60	20	80
52 (p=4)	58,320	268	72	340	58,212	440	120	560
78 (p=6)	196,800	880	180	1,060	196,438	1,460	300	1,760
104 (p=8)	466,464	2,068	336	2,404	465,608	3,440	560	4,000
130 (p=10)	911,040	4,024	540	4,564	909,370	6,700	900	7,600
156 (p=12)	1,574,256	6,940	792	7,732	1,571,372	11,560	1,320	12,880
182 (p=14)	2,499,840	11,008	1,092	12,100	2,495,262	18,340	1,820	20,160
208 (p=16)	3,731,520	16,420	1,440	17,860	3,724,688	27,360	2,400	29,760

IV. COMPUTATIONAL COMPLEXITY

Considering that each solve instance consists of one offline setup and 10 online iterations to perform factorization of $\tilde{C}$ part of the KKT matrix. The complexity for online factorization is given in the form of k, which represents the rank of the $S^{-1}Z$ block. Performing simulations of multiple problem sizes, we have observed that the value of k drops to 1 typically around 6 or 7 iterations for all the problem sizes. Using the number of operations involved in each step as explained in Section III, the computational complexity in terms of floating point operations (FLOPS) is given in Table 1. In comparison, we also show the computational complexity for the method in [15]. Note that the complexity of online factorization for the proposed method is given in terms of k, which drops down to 1 around the 7th iteration. Even though the proposed method is iterative, reduction in the rank of the matrix to 1, reduces the complexity of the online factorization operation which contributes towards the efficiency of the proposed method. On the other hand, the complexity of offline setup has increased in the proposed method but since that is done only once for each solve instance, it does not contribute significantly toward the overall complexity and efficiency of the algorithm. The complexities in terms of number of floating-point operations (FLOPS) for various sizes of the KKT matrix are given in Table 2. It can be seen that for the largest KKT matrix dimension of 208 in Table 2 the number of FLOPS for the online factorization has reduced by 40%.

We observe that when the rank drops to 1, the product-form Cholesky factorization method requires the least amount of FLOPS. However, both the rank-k and rank-1 methods can take advantage of parallel computation to compute the term $LY = I$ or $Y = L^{-1}$, if k parallel processors can compute each column of Y.

V. CONCLUSION

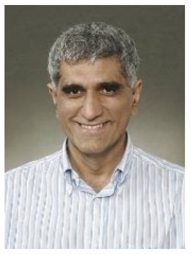
In this paper, we have proposed a method to factorize the KKT matrices for solving real-time convex optimization problems on an embedded system. We have used a product-form Cholesky factorization method to efficiently factorize the KKT matrix online especially when the rank of the matrix drops significantly to around 1 after 6 or 7 solve iterations. This approach can be used in combination with Cholesky factorization of the full-rank matrix to save significant number of operations in solving a problem of the same size, i.e. about 40% less number of FLOPS are required to solve a problem with a KKT matrix dimension of 208.

REFERENCES

[1] D. P. Bertsekas, “Convex optimization algorithms.” Athena Scientific, Belmont, MA., 2015.
[2] S. Boyd and L. Vandenberghe, “Convex optimization algorithms.” Cambridge University Press, Cambridge, 2012.
[3] J. Mattingley and S. Boyd, “Realtime convex optimization in signal processing,” *IEEE Signal Processing Magazine*, vol. 27, no. 3, pp. 50–61, May 2010.
[4] I. Das and W. Fuller, J, “Real-time quadratic programming for control of dynamical systems,” *US Patent 7328074, tech. Rep.*, 2008.
[5] D. Burns, W. Weiss, and M. Guay, “Realtime setpoint optimization with time-varying extremum seeking for vapor compression systems,” in *American Control Conference (ACC), 2015*, Jul. 2015, pp. 974–979.
[6] J. Mattingley and S. Boyd, “Automatic code generation for real-time convex optimization,” in *Conex Optimization in Signal Processing and Communications*. D. P. Palomar, and Y. C. Eldar, Eds. Cambridge University Press, 2009, pp. ch. 1, 1–41.
[7] (2012) BLAS: Basic linear algebra subprograms. [Online]. Available: http://www.netlib.org/blas
[8] (2016) LAPACK: Linear algebra PACKage. [Online]. Available: http://www.netlib.org/lapack
[9] Intel. (2012) Intel math kernel library (intel mkl) 11.0. [Online]. Available: http://software.intel.com/en-us/intel-mkl
[10] J. Mattingley and J. Boyd, “Cvxgen âĂŞ code generation for convex optimization,” *Optimization and Engineering*, vol. 13, no. 1, pp. 1–27, Mar. 2012. [Online]. Available: http://http://cvxgen.com
[11] A. Domahidi, E. Chu, and S. Boyd, “Ecos: An socp solver for embedded systems,” in *European Control Conference (ECC), Zurich, Switzerland*, Jul. 2013, pp. 3071–3076. [Online]. Available: https://www.embotech.com/ECOS
[12] M. Grant, S. Boyd, and Y. Ye, “Disciplined convex programming,” in *Global Optimization: From Theory to Implementation (Nonconvex Optimization and Its Applications)*. L. Liberti and N. Maculan, Eds. Springer Science and Business Media, 2006, pp. 155–210.
[13] Y. Wang and S. Boyd, “Fast model predictive control using online optimization,” in *World Congress of the International Federation of Automatic Control (IFAC), Jejudo, Korea*, Jun. 2001, pp. 6974–6997.
[14] J. Nocedal and S. Wright, “Numerical optimization, 2nd ed.” Springer Verlag, 2006.
[15] I. Yamazaki, S. Nooshabadi, S. Tomov, and J. Dongara, “Structure-aware linear solver for realtime convex optimization for embedded systems,” *IEEE Embedded Systems Letters*, vol. 9, no. 3, pp. 61–65, 2017.
[16] M. Grant and S. Boyd. (2015) CVX: MATLAB software for disciplined convex programming version 2.1. [Online]. Available: http://cvxr.com/cvx
[17] T. M., “A note on the ldlt decomposition of matrices from saddle-point problems,” *SIAM Journal on Matrix Analysis and Application*, vol. 23, no. 4, pp. 903–915, April 2002.
[18] D. Goldfarb and K. Scheinberg, “Product-form cholesky factorization in interior point methods for second-order cone programming,” *Mathematical Programming*, vol. 103, no. 1, pp. 153–179, Dec. 2004.

ZAFAR IQBAL (M’18) received his undergraduate degree in computer engineering from COMSATS University, Islamabad, Pakistan in 2005, M.S. in information and mechatronics, and Ph.D. in electrical engineering and computer science from the Gwangju Institute of Science and Technology (GIST), South Korea in 2010 and 2017, respectively. He was awarded the Korea IT Industry Promotion Agency scholarship for his M.S., and the Korean Government Scholarship for his Ph.D. study and research.

He was with ZTE Corporation, Shanghai R&D Center, from 2005 to 2008. He worked at Vieworks Co. Ltd. Korea, and Nokia Siemens Networks Co. Ltd., Shanghai, during 2011 and as a research assistant professor at Michigan Tech in 2017/18. Currently, he is a machine learning research scientist at Ford Motor Company. His research interests include wireless communications, signal processing, machine learning, high-performance computing, and computer vision systems.



SAEID NOOSHABADI received an MTech and a PhD in Electrical Engineering from the India Institute of Technology, Delhi, in 1986 and 1992, respectively.

In 1992, Nooshabadi was a research scientist at the CAD Laboratory, of the Indian Institute of Science in Bangalore. In 1996 and 1997, he was a visiting faculty member and researcher at the Center for Very High Speed Microelectronic Systems, of Edith Cowan University and at Curtin University of Technology, in Western Australia. From 2000 to 2007 he was with the School of Electrical Engineering and Telecommunications at the University of New South Wales, in Sydney. He was a professor in VLSI multimedia signal processing in the Department of Information and Communications at the Gwangju Institute of Science and Technology, Republic of Korea. Currently, he is a professor at the Department of Electrical and Computer Engineering, Michigan Tech.

Nooshabadi has extensive research experience and interests in the area of SoC design of multimedia systems, high-performance and low-power computing systems, application-specified integrated circuit design for information-processing systems, and embedded electronic systems.

ICHITARO YAMAZAKI received his PhD degree in Computer Science from the University of California at Davis in 2008. He has worked as a Postdoctoral Researcher in Scientific Computing Group at Lawrence Berkeley National Laboratory from 2008 to 2011. He was a research scientist in the Innovative Computing Laboratory at the University of Tennessee at Knoxville at the time of this research work. He is currently with Sandia National Labs, where his interests lie in high-performance computing, especially for linear algebra and scientific computing.

STANIMIRE TOMOV received a M.S. degree in Computer Science from Sofia University, Bulgaria, and Ph.D. in Mathematics from Texas A&M University. He is a Research Assistant Professor at the Innovative Computing Laboratory (ICL), University of Tennessee. Tomov's research interests are in parallel algorithms, numerical analysis, and high-performance scientific computing (HPC). Currently, his work is concentrated on the development of numerical linear algebra software for emerging architectures for HPC.

JACK DONGARRA received a Bachelor of Science in Mathematics from Chicago State University in 1972 and a Master of Science in Computer Science from the Illinois Institute of Technology in 1973. He received his Ph.D. in Applied Mathematics from the University of New Mexico in 1980. He worked at the Argonne National Laboratory until 1989, becoming a Senior Scientist. He now holds an appointment as University Distinguished Professor of Computer Science in the Department of Electrical Engineering and Computer Science at the University of Tennessee, has the position of a Distinguished Research Staff member in the Computer Science and Mathematics Division at Oak Ridge National Laboratory (ORNL), Turing Fellow in the Computer Science and Mathematics Schools at the University of Manchester, and an Adjunct Professor in the Computer Science Department at Rice University.