Decentralized Computation for Robust Stability Analysis of Large State-Space Systems Using Polya's Theorem

Reza Kamyar and Matthew M. Peet

***Abstract*— In this paper, we propose a parallel algorithm to solve large robust stability problems. We apply Polya's theorem to a parameter-dependent version of the Lyapunov inequality to obtain a set of coupled linear matrix inequality conditions. We show that a common implementation of a primal-dual interior-point method for solving this LMI has a block diagonal structure which is preserved at each iteration. By exploiting this property, we create a highly scalable cluster-computing implementation of our algorithm for robust stability analysis of systems with large state-space. Numerical tests confirm the scalability of the algorithm.**

I. INTRODUCTION

Lyapunov-based methods have been used extensively in robust stability analysis of systems with uncertain parameters. These methods are often used to obtain stability conditions in the form of Linear Matrix Inequalities (LMIs). In the case of systems with uncertain parameters, the use of parameter-dependent Lyapunov functions as in [1], [2], [3] and [4] leads to parameter-dependent LMIs. The feasibility of such parameter-dependent LMIs implies the stability of the uncertain system. This Feasibility problem is known to be NP-hard [5]. A common algorithm for solving parameter-dependent LMIs is Sum of Squares (SOS). This algorithm optimizes over positive polynomials with scalar or matrix coefficients [6], [7], [8]. Unfortunately, the number of variables in Semi-Definite Programming (SDP) problems associated with SOS algorithm can easily grow beyond ten thousand as the number of uncertain parameters and/or the degree of squared polynomials increases. Current single-core and multi-core machines with shared memory architecture are incapable of solving such large SDP problems due to their insufficient memory capacity. Moreover, the single-core processors speed has not been increased significantly over the last few years and no further speed improvement is expected in near future [9]. In some applications, parallel computing can resolve the speed saturation problem and make a better use of the memory in hand. In particular, cluster computers can perform numerous tasks simultaneously and process large amounts of data by distributing the tasks and data among the processors and their individual memories.

Typical LMI solvers [10], [11] can only utilize a single processor of cluster computers. Moreover, due to the Amdahl's law [12] the speed of general-purpose parallel SDP solvers [13], [14] inevitably saturates as the number of processors increases. To take the full advantage of the computational power of cluster computers, we must use algorithms with highly decentralized structure. Unfortunately, the SDP conditions associated with the SOS algorithm do not have an obvious distributed structure. For this reason we pursue an alternative approach based on Polya's theorem, where the uncertain parameters are assumed to be inside a unit simplex. A modern version of this theorem for polynomials with matrix coefficients can be found in [15]. A performance comparison of SOS algorithm and Polya's algorithm in robust stability analysis has been done in [4]. The contribution of this paper is to show how to use the structure of the SDP which results from Polya's theorem to distribute the computation of step size and search direction in a primal-dual interior-point algorithm. We show that the largest tasks in this interior-point algorithm distribute with little communication overhead. We also show that if we have a sufficiently large number of processors, our algorithm solves large robust stability problems in the same time as it takes to solve the parameter-independent Lyapunov inequality. Moreover, if we have a sufficient number of processors the algorithm allows us to increase the accuracy of our approximation for the domain of uncertain parameters in which the system is stable, without adding any computation per processor or communication overhead. For uncertain systems with large number of states, by using N processors the parallel algorithm solves the associated robust stability problem approximately N times faster than a sequential algorithm, where N is shown to have a large upper bound. This implies that the proposed parallel algorithm is scalable.

This paper is arranged as follows. In Section III, the background materials for uncertain system characteristics, Lyapunov stability and Polya's theorem are presented. The LMIs obtained from Polya's algorithm and the associated SDP problem are discussed in Section IV. Then the parallel interior-point SDP solver, its implementation and complexity analysis are addressed in Section V. Finally, the performance of SDP solver in terms of the scalability and computational time is demonstrated in Section VI.

II. NOTATION

We represent a monomial as α^γ, where $\alpha \in \mathbb{R}^l$ is the vector of variables, $\gamma \in \mathbb{N}^l$ is the vector of exponents and $\alpha^\gamma = \prod_{i=1}^{l} \alpha_i^{\gamma_i}$. Consider α^γ and δ^η as two monomials, where $\alpha, \delta \in \mathbb{R}^l$ and $\gamma, \eta \in \mathbb{N}^l$. According to lexicographical ordering, α^γ precedes δ^η if the left most non-zero entry of $\gamma - \eta$ is positive. The subspace of ordered symmetric

Reza Kamyar is a Ph.D student with the Cybernetic Systems and Control Lab (CSCL), Department of Mechanical, Material and Aerospace Engineering, Illinois Institute of Technology, Chicago, IL, 60616 USA, rkamyar@iit.edu

Matthew M. Peet is an assistant professor with the department of Mechanical, Material and Aerospace Engineering, Illinois Institute of Technology, Chicago, IL, 60616 USA, mpeet@iit.edu

matrices in $\mathbb{R}^{n\times n}$ is denoted by $\mathbb{S}_n$. The standard basis for $\mathbb{S}_n$ is defined as

$$[E_k]_{ij} = \begin{cases} 1 & i=j=k \\ 0 & \text{otherwise} \end{cases}, \quad \text{for } k \le n$$

and

$$[E_k]_{ij} = [A_k]_{ij} + [A_k]_{ij}^T, \quad \text{for } k > n,$$

where

$$[A_k]_{ij} = \begin{cases} 1 & i = j-1 = k-n \\ 0 & \text{otherwise.} \end{cases}$$

The canonical basis for $\mathbb{R}^n$ is denoted by e_i for $i = 1, \cdots, n$, where

$$e_i = [0 \dots 0 \underbrace{1}_{i^{th}} 0 \dots 0].$$

$\vec{1} \in \mathbb{R}^k$ is a vector with all the entries equal to one. The trace of $A \in \mathbb{R}^{n\times n}$ is denoted by $tr(A) = \sum_{i=1}^n A_{ii}$. $\text{diag}(X_1, \cdots, X_m)$ is a block-diagonal matrix in $\mathbb{R}^{mn\times mn}$ whose diagonal blocks are $X_1, \cdots, X_m \in \mathbb{R}^{n\times n}$.

III. PRELIMINARIES

Consider the linear system

$$\dot{x}(t) = \mathrm{A}(\alpha)x(t), \tag{1}$$

where $\mathrm{A}(\alpha) \in \mathbb{R}^{n\times n}$ and $\alpha \in Q \subset \mathbb{R}^l$ is a vector of uncertain parameters. In this paper, we consider the case of a homogeneous polynomial $\mathrm{A}(\alpha)$ and $Q = \Delta_l \subset \mathbb{R}^l$ where Δ_l is the unit simplex:

$$\Delta_l = \left\{ \alpha \in \mathbb{R}^l, \sum_{i=1}^{l} \alpha_i = 1, \alpha_i \geqslant 0 \right\} \tag{2}$$

If $\mathrm{A}(\alpha)$ with degree d_a is not homogeneous, it can be made homogeneous by multiplying each monomial in $\mathrm{A}(\alpha)$ by $1 = (\sum_i \alpha_i)^b$, where $b = d_a - d_m$ and d_m is the degree of that monomial.

The following is a stability condition [4].

Theorem 1: The linear system (1) is stable if and only if there exists a polynomial matrix $P(\alpha)$ such that $P(\alpha) > 0$ and

$$\mathrm{A}^T(\alpha)P(\alpha) + P(\alpha)\mathrm{A}(\alpha) < 0 \tag{3}$$

for all $\alpha \in \Delta_l$.

A similar condition also holds for discrete-time linear systems.

The conditions associated with Theorem 1 are infinite-dimensional LMIs, meaning they must hold at an infinite number of points. Such problems are known to be NP-hard [5]. In this paper we derive a sequence of polynomial-time algorithms such that their outputs converge to the solution of the infinite-dimensional LMI. Key to this result is Polya's Theorem [16]. A variation of this theorem for matrices is listed below.

Theorem 2: (Polya's Theorem) The homogeneous polynomial $F(\alpha) > 0$ for all $\alpha \in \Delta_l$ if and only if for all sufficiently large d,

$$\left(\sum_{i=1}^{l} \alpha_i\right)^d F(\alpha) \tag{4}$$

has all positive definite coefficients.

Upper bounds for Polya's exponent d have been found [17] based on the properties of F. In this paper, we show that applying Polya's algorithm on Theorem 1 yields a semidefinite programming condition with a parallel structure. This condition will be discussed in detail in the following section.

IV. PROBLEM SET-UP

In this section, we show how Polya's theorem can be used to determine the robust stability of an uncertain system using linear matrix inequalities with a distributed structure.

A. Polya's Algorithm

We consider the stability of the system described by Equation (1). We are interested in finding a $P(\alpha)$ which satisfies the conditions of Theorem 1. According to Polya's theorem, the constraints of Theorem 1 are satisfied if for some sufficiently large d, the polynomials

$$\left(\sum_{i=1}^{l} \alpha_i\right)^d P(\alpha) \text{ and} \tag{5}$$

$$-\left(\sum_{i=1}^{l} \alpha_i\right)^d (\mathrm{A}^T(\alpha)P(\alpha) + P(\alpha)\mathrm{A}(\alpha)) \tag{6}$$

have all positive definite coefficients.

Let P be a polynomial of degree d_p. It is defined using coefficient matrices P_γ as

$$P(\alpha) = \sum_{\gamma \in W_{d_p}} P_\gamma \alpha^\gamma, \tag{7}$$

where W_d is the unit disk $W_d := \left\{\gamma \in \mathbb{N}^l : \sum_{i=1}^l \gamma_i = d\right\}$, where l is the dimension of the vector α. Likewise, let A be a homogeneous polynomial of degree d_a. It is defined using the matrices A_γ as

$$\mathrm{A}(\alpha) = \sum_{\gamma \in W_{d_a}} \mathrm{A}_\gamma \alpha^\gamma. \tag{8}$$

By substituting (7) and (8) into (5) and (6), the conditions of Theorem 2 can be represented as

$$\sum_{h \in W_{d_p}} \beta_{h,\gamma} P_h > 0; \quad \gamma \in W_{d_p+d} \text{ and} \tag{9}$$

$$\sum_{h \in W_{d_p}} (H_{h,\gamma}{}^T P_h + P_h H_{h,\gamma}) < 0; \quad \gamma \in W_{d_p+d_a+d}. \tag{10}$$

Before providing the formulas for the calculation of the sets of scalars $\{\beta_{h,\gamma}\}$ and matrices $\{H_{h,\gamma}\}$, let us obtain these coefficients for a simple case. Consider

$A(\alpha) = A_{[1,0]}\alpha_1 + A_{[0,1]}\alpha_2$ and $P(\alpha) = P_{[1,0]}\alpha_1 + P_{[0,1]}\alpha_2$.

By calculating (5) for $d = 1$ we have

$$(\alpha_1+\alpha_2)P(\alpha) = P_{[1,0]}\alpha_1^2 + (P_{[1,0]} + P_{[0,1]})\alpha_1\alpha_2 + P_{[0,1]}\alpha_2^2$$

and $\{\beta_{h,\gamma}\}$ can be extracted as

$$\beta_{[1,0],[2,0]} = 1, \beta_{[0,1],[2,0]} = 0, \beta_{[1,0],[1,1]} = 1,$$
$$\beta_{[0,1],[1,1]} = 1, \beta_{[1,0],[0,2]} = 0, \beta_{[0,1],[0,2]} = 1.$$

By calculating (6) for $d = 1$ we have

$$\begin{aligned}(\alpha_1+\alpha_2)\left(A^T(\alpha)P(\alpha) + P(\alpha)A(\alpha)\right) &= \left(A_1^TP_1 + P_1A_1\right)\alpha_1^3 \\ &+ \left(A_1^TP_1 + P_1A_1 + A_2^TP_1 + P_1A_2 + A_1^TP_2 + P_2A_1\right)\alpha_1^2\alpha_2 \\ &+ \left(A_2^TP_1 + P_1A_2 + A_1^TP_2 + P_2A_1 + A_2^TP_2 + P_2A_2\right)\alpha_1\alpha_2^2 \\ &+ \left(A_2^TP_2 + P_2A_2\right)\alpha_2^3\end{aligned}$$

and $\{H_{h,\gamma}\}$ can be extracted as

$$\begin{aligned} H_{[1,0],[3,0]} &= A_1, & H_{[0,1],[3,0]} &= \mathbf{0}, \\ H_{[1,0],[2,1]} &= A_1 + A_2, & H_{[0,1],[2,1]} &= A_1, \\ H_{[1,0],[1,2]} &= A_2, & H_{[0,1],[1,2]} &= A_1 + A_2, \\ H_{[1,0],[0,3]} &= \mathbf{0}, & H_{[0,1],[0,3]} &= A_2. \end{aligned}$$

We define $\{\beta_{h,\gamma}\}$ recursively as follows. Let

$$\beta^0_{h,\gamma} = \begin{cases} 1 & h=\gamma \\ 0 & \text{otherwise} \end{cases} \qquad \gamma \in W_{d_p},\ h \in W_{d_p} \tag{11}$$

Then, for $i = 1,\dots d$, let

$$\beta^i_{h,\gamma} = \sum_{\substack{\lambda \in W_{d_p+i-1} \\ \lambda=\gamma-e_j \\ j=1\dots l}} \beta^{i-1}_{h,\lambda}. \qquad \gamma \in W_{d_p+i},\ h \in W_{d_p} \tag{12}$$

Finally, $\beta_{h,\gamma} = \beta^d_{h,\gamma}$, where $\gamma \in W_{d_p+d}$. To obtain $H_{h,\gamma}$, let

$$H^0_{h,\gamma} = \sum_{\substack{\delta \in W_{d_a} \\ \delta+h=\gamma}} \mathrm{A}_\delta, \qquad \gamma \in W_{d_p+d_a}, \quad h \in W_{d_p}. \tag{13}$$

Then, for $i = 1,\dots d$, let

$$H^i_{h,\gamma} = \sum_{\substack{\lambda \in W_{d_p+d_a+i-1} \\ \lambda=\gamma-e_j \\ j=1\dots l}} H^{i-1}_{h,\lambda} \qquad \gamma \in W_{d_p+d_a+i} \quad h \in W_{d_p}. \tag{14}$$

Finally, set $H_{h,\gamma} = H^d_{h,\gamma}$, where $\gamma \in W_{d_p+d_a+d}$.

Computing $\{\beta_{h,\gamma}\}$ and $\{H_{h,\gamma}\}$ is a significant challenge. For a given d, the number of $\beta_{h,\gamma}$ coefficients is $L_0 \cdot L$, where

$$L_0 = \frac{(d_p+l-1)!}{d_p!(l-1)!}$$

is the number of monomials in $P(\alpha)$ and

$$L = \frac{(d_p+d+l-1)!}{(d_p+d)!(l-1)!} \tag{15}$$

is the cardinality of W_{d_p+d}, where recall l is the dimension of the uncertain parameters α in the uncertain system (1). The number of $H_{h,\gamma}$ coefficients is $L_0 \cdot M$, where

$$M = \frac{(d_p+d_a+d+l-1)!}{(d_p+d_a+d)!(l-1)!} \tag{16}$$

is the cardinality of $W_{d_p+d_a+d}$. In [18], we proposed a decentralized computing approach to the calculation of the coefficients $\beta_{h,\gamma}$.

In the following section, we express the LMIs associated with conditions (9) and (10) in primal and dual format. We also discuss the structure of the primal and dual SDP variables and constraints.

B. SDP Problem Elements

We express the LMI constraints of (9) and (10) as a semi-definite programming problem. We define semi-definite programming as the optimization of a linear objective function over the cone of positive definite matrices subject to linear equality constraints. This problem can be stated either in primal or in dual formulation.

Given $C \in \mathbb{S}_m$, $a \in \mathbb{R}^k$ and $A_i \in \mathbb{S}_m$, the **primal problem** is

$$\begin{aligned} \max \quad & tr(CX) \\ \text{subject to} \quad & a - A(X) = 0 \\ & X \succeq 0 \end{aligned} \tag{17}$$

where the linear operator $A : \mathbb{S}_m \to \mathbb{R}^k$ is defined as

$$A(X) = \begin{bmatrix} tr(A_1X) & tr(A_2X) & \cdots & tr(A_kX) \end{bmatrix}^T. \tag{18}$$

$X \in \mathbb{S}_m$ is the primal variable. Please note that, the operator A in (17) is different from the system matrix A in (1).

Given a primal SDP, the associated **dual problem** is

$$\begin{aligned} \min \quad & a^T y \\ \text{subject to} \quad & A^T(y) - C = Z \\ & Z \succeq 0\ ,\ y \in \mathbb{R}^k \end{aligned} \tag{19}$$

where the linear operator $A^T : \mathbb{R}^k \to \mathbb{S}_m$ is defined as

$$A^T(y) = \sum_{i=1}^{k} y_i A_i. \tag{20}$$

$y \in \mathbb{R}^k$ and $Z \in \mathbb{S}_m$ are the dual variables.

The elements C, A_i and a of SDP problem, associated with the LMIs in (9) and (10) are defined as follows. We define the element C as

$$C = \mathrm{diag}(C_1, \cdots C_L, C_{L+1}, \cdots C_{L+M}), \tag{21}$$

where

$$C_j = \begin{cases} \varepsilon I_n \cdot \left(\sum_{h \in W_{d_p}} \beta_{h,\lambda_j} \frac{d_p!}{\Pi_{i=1}^{l} h_i!} \right), & 1 \le j \le L \\ 0_n, & L+1 \le j \le L+M, \end{cases} \tag{22}$$

where $h_i \in \mathbb{N}^l$ is the i^{th} element of W_{d_p} using lexicographical ordering, λ_j is the j^{th} element of W_{d_p+d} using lexicographical ordering, L is the cardinality of W_{d_p+d}, M is the cardinality of $W_{d_p+d_a+d}$, I_n and 0_n are the identity and zero matrices of dimension n, and l is the number of uncertain parameters. For $i = 1, \cdots, K$, the elements A_i are defined as

$$A_i = \mathrm{diag}(A_{i,1}, \cdots A_{i,L}, A_{i,L+1}, \cdots A_{i,L+M}) \tag{23}$$

where

$$K = \frac{(d_p+l-1)!}{d_p!(l-1)!} \frac{n(n+1)}{2}, \tag{24}$$

is the number of independent elements in the coefficients of $P(\alpha)$ and $A_{i,j}$ is equal to

$$\begin{cases} \sum_{h \in W_{d_p}} \beta_{h,\lambda_j} P_h(e_i), & 1 \le j \le L \\ -\sum_{h \in W_{d_p}} H^T_{h,\gamma_{j-L}} P_h(e_i) + P_h(e_i) H_{h,\gamma_{j-L}}, & L+1 \le j \le L+M \end{cases} \tag{25}$$

where γ_j is the j^{th} element of $W_{d_p+d_a+d}$ using lexicographical ordering and

$$P_h(x) = \sum_{k=1}^{N} E_k\, x_{k+N(I_h-1)},$$

where E_k is the basis of $\mathbb{S}_n$ from Section II, I_h is the lexicographical index of h, $N = \frac{n(n+1)}{2}$ and n is the dimension of system (1). Finally, by setting

$$a = \vec{1} \in \mathbb{R}^K, \tag{26}$$

the SDP problem associated with Polya's algorithm is defined.

V. PARALLEL SDP SOLVER

In this section, we describe the steps of a primal-dual interior-point algorithm and show how, for the LMIs in (9) and (10), these steps can be distributed in a distributed-computing, distributed-memory environment.

A. Interior-point methods

Interior-point methods define a popular class of algorithms for solving linear and semi-definite programming problems. The three types of interior-point algorithm are: primal [19], primal-dual [20], [21], [22] and dual scaling [23]. In this paper, we use the central-path-following primal-dual algorithm described in [22] and [13]. In this algorithm, both primal and dual problems are solved simultaneously by iteratively calculating primal and dual step directions and step sizes, and applying these to the primal and dual variables. Let X

be the primal variable and y and Z be the dual variables. At each iteration, the variables are updated as

$$X_{k+1} = X_k + t_p \Delta X \tag{27}$$
$$y_{k+1} = y_k + t_d \Delta y \tag{28}$$
$$Z_{k+1} = Z_k + t_d \Delta Z, \tag{29}$$

where ΔX, Δy, and ΔZ are the search directions and t_p and t_d are primal and dual step sizes. We choose the step sizes using a line-search between 0 and 1 so that X_{k+1} and Z_{k+1} remain positive semi-definite. The Newton search direction we use is

$$\Delta X = \Delta \widehat{X} + \Delta \overline{X} \tag{30}$$
$$\Delta y = \Delta \widehat{y} + \Delta \overline{y} \tag{31}$$
$$\Delta Z = \Delta \widehat{Z} + \Delta \overline{Z}, \tag{32}$$

where $\Delta \widehat{X}$, $\Delta \widehat{y}$ and $\Delta \widehat{Z}$ are the predictor step directions and $\Delta \overline{X}$, $\Delta \overline{y}$, and $\Delta \overline{Z}$ are the corrector step directions. The predictor step directions are found as

$$\begin{aligned} \Delta \widehat{y} &= O^{-1}\left(-a + A(Z^{-1}GX)\right) \\ \Delta \widehat{X} &= -X + Z^{-1}GA^T(\Delta \widehat{y})X \\ \Delta \widehat{Z} &= A^T(y) - Z - C + A^T(\Delta \widehat{y}), \end{aligned} \tag{33}$$

where C and the operators A and A^T are as defined in the previous section and

$$\begin{aligned} G &= -A^T(y) + Z + C \\ O &= \left[A(Z^{-1}A^T(e_1)X) \ \cdots \ A(Z^{-1}A^T(e_k)X)\right] \end{aligned} \tag{34}$$

and recall $e_1, ..., e_k$ are the unit basis vectors in $\mathbb{R}^k$. Once we have the predictor step directions, we can calculate the corrector step directions. Let $\mu = \frac{1}{3}tr(ZX)$. The corrector step directions are

$$\begin{aligned} \Delta \overline{y} &= O^{-1}\left(A(\mu Z^{-1}) - A(Z^{-1}\Delta \widehat{Z}\Delta \widehat{X})\right) \\ \Delta \overline{X} &= \mu Z^{-1} - Z^{-1}\Delta \widehat{Z}\Delta \widehat{X} - Z^{-1}\Delta \overline{Z}X \end{aligned} \tag{35}$$
$$\Delta \overline{Z} = A^T(\Delta \overline{y}). \tag{36}$$

The stopping criterion is $|a^T y - tr(CX)| \leq \varepsilon$. Information regarding the convergence of different variants of interior-point primal-dual algorithm are presented in [20] and [21].

B. Structure of SDP Variables

In this section, the structure of the primal and dual variables of the SDP problem associated with Polya's algorithm is introduced. First, we define the following structured block-diagonal subspace.

$$\begin{aligned} S_{l,m,n} := \{Y \subset \mathbb{R}^{mn \times mn} : Y = \text{diag}(Y_1, \cdots Y_l, Y_{l+1}, \cdots Y_{l+m}) \\ \text{for} \quad Y_i \in \mathbb{R}^{n \times n}\} \end{aligned} \tag{37}$$

According to the following theorem, at each iteration the primal and dual variables of our SDP problem defined in Section IV-B have the same structure as in (37).

Theorem 3: Consider the SDP problem defined in (17) and (19) with elements given by (21), (23) and (26). Suppose L and M are the cardinalities of W_{d_p+d} and $W_{d_p+d_a+d}$. If (27), (28) and (29) are initialized by

$$X_0 \in S_{L,M,n}, \quad y_0 \in R^K, \quad Z_0 \in S_{L,M,n},$$

then for all $k \in \mathbb{N}$,

$$X_k \in S_{L,M,n}, \quad Z_k \in S_{L,M,n}.$$

Proof: First, suppose for some $k \in \mathbb{N}$

$$X_k \in S_{L,M,n}, \quad Z_k \in S_{L,M,n}. \tag{38}$$

We will show that this implies $X_{k+1}, Z_{k+1} \in S_{L,M,n}$. To see this, observe that according to (27)

$$X_{k+1} = X_k + t_p \Delta X_k \quad \text{for all } k \in \mathbb{N}.$$

From (30), ΔX_k can be written as

$$\Delta X_k = \Delta \widehat{X}_k + \Delta \overline{X}_k \quad \text{for all } k \in \mathbb{N}. \tag{39}$$

To find the structure of ΔX_k, we focus on the structure of $\Delta \widehat{X}_k$ and $\Delta \overline{X}_k$ individually. Using (33), $\Delta \widehat{X}_k$ is

$$\Delta \widehat{X}_k = -X_k + Z_k^{-1}G_k A^T(\Delta \widehat{y}_k)X_k \quad \text{for all } k \in \mathbb{N}. \tag{40}$$

where according to (34), G_k is

$$G_k = C - A^T(y_k) + Z_k \quad \text{for all } k \in \mathbb{N}. \tag{41}$$

First we examine the structure of G_k. According to the definition of C and A_i in (21) and (23), and the definition of $A^T(y)$ in (20), we know that

$$C \in S_{L,M,n}, \quad A^T : \mathbb{R}^K \mapsto S_{L,M,n}. \tag{42}$$

Since all the terms on the right hand side of (41) are in $S_{L,M,n}$ and the structure of matrices in $S_{L,M,n}$ are preserved through algebraic addition, we conclude

$$G_k \in S_{L,M,n} \quad \text{for all } k \in \mathbb{N}. \tag{43}$$

Returning to (40), using our assumption in (38) and noting that the structure of the matrices in $S_{L,M,n}$ is preserved through multiplication and inversion, we conclude

$$\Delta \widehat{X}_k \in S_{L,M,n} \quad \text{for all } k \in \mathbb{N}. \tag{44}$$

Using (35), the second term in (39) is

$$\Delta \overline{X}_k = \mu Z_k^{-1} - Z_k^{-1}\Delta \widehat{Z}_k \Delta \widehat{X}_k - Z_k^{-1}\Delta \overline{Z}_k X_k \quad \text{for all } k \in \mathbb{N}. \tag{45}$$

To determine the structure of $\Delta \overline{X}_k$, first we investigate the structure of $\Delta \widehat{Z}_k$ and $\Delta \overline{Z}_k$. According to (48) and (36) we have

$$\Delta \widehat{Z}_k = A^T(y_k) - Z_k - C + A^T(\Delta \widehat{y}_k) \quad \text{for all } k \in \mathbb{N} \tag{46}$$
$$\Delta \overline{Z}_k = A^T(\Delta \overline{y}_k) \quad \text{for all } k \in \mathbb{N}. \tag{47}$$

Since all the terms in the right hand side of (46) and (47) are in $S_{L,M,n}$, then

$$\Delta \widehat{Z}_k \in S_{L,M,n}, \quad \Delta \overline{Z}_k \in S_{L,M,n} \quad \text{for all } k \in \mathbb{N}. \tag{48}$$

Recalling (45) and our assumption in (38), we have

$$\Delta \overline{X}_k \in S_{L,M,n} \quad \text{for all } k \in \mathbb{N}. \tag{49}$$

According to (40), (44), (48) and (49), the total step directions are in $S_{L,M,n}$,

$$\Delta X_k = \Delta \widehat{X}_k + \Delta \overline{X}_k \in S_{L,M,n} \quad \text{for all } k \in \mathbb{N}$$
$$\Delta Z_k = \Delta \widehat{Z}_k + \Delta \overline{Z}_k \in S_{L,M,n} \quad \text{for all } k \in \mathbb{N},$$

and it follows that

$$X_{k+1} = X_k + t_p \Delta X_k \in S_{L,M,n} \quad \text{for all } k \in \mathbb{N}$$
$$Z_{k+1} = Z_k + t_p \Delta Z_k \in S_{L,M,n} \quad \text{for all } k \in \mathbb{N}.$$

Thus, we have shown that for any $k \in \mathbb{N}$ if $X_k, Z_k \in S_{L,M,n}$, then $X_{k+1}, Z_{k+1} \in S_{L,M,n}$. According to the theorem assumption, $X_0, Z_0 \in S_{L,M,n}$. Therefore, by induction we have proved that

$$X_k \in S_{L,M,n}, \quad Z_k \in S_{L,M,n} \quad \text{for all } k \in \mathbb{N}$$

■

C. Parallel Implementation

In this section, a parallel algorithm for solving the SDP problems associated with Polya's algorithm is provided. We show how to exploit the block-diagonal structure of SDP elements and primal and dual variables to decentralize the interior-point algorithm described in Section V-A.

Let N be the number of available processors and $J = \texttt{floor}(\frac{L+M}{N})$. Processor i has access to $\overline{\mathbf{C}}_i$ and $\overline{\mathbf{A}}_{j,i}$ for $j = 1,\cdots,K$, where

$$\overline{\mathbf{C}}_i = \begin{cases} \text{diag}(C_{(i-1)(J+1)+1},\cdots,C_{i(J+1)}) & \text{if } 0 \le i < L+M-NJ \\ \text{diag}(C_{(2i-1)(J+1)},\cdots,C_{(2J+1)i+1}) & \text{if } L+M-NJ \le i < N \end{cases}$$

and

$$\overline{\mathbf{A}}_{j,i} = \begin{cases} \text{diag}(A_{j,(i-1)(J+1)+1},\cdots,A_{j,i(J+1)}) & \text{if } 0 \le i < L+M-NJ \\ \text{diag}(A_{j,(2i-1)(J+1)},\cdots,A_{j,(2J+1)i+1}) & \text{if } L+M-NJ \le i < N, \end{cases}$$

where C and A matrices are calculated using (22) and (25), and $\{\beta_{h,\gamma}\}$ and $\{H_{h,\gamma}\}$ are calculated using an algorithm similar to the algorithm in [18]. The parallel algorithm consists of processors and root initialization steps, five processors steps and five root steps. The inputs, steps and outputs are as follows.

Inputs:
The inputs to the algorithm are C_i for $i = 1,\cdots,L+M$, $A_{j,i}$ for $i = 1,\cdots,L+M$ and $j = 1,\cdots,K$, degree of $P(\alpha)$ and the number of uncertain parameters.

Processors Initialization step:
For $i = 0,\cdots,N-1$, processor i:

1) set $\mathbf{X}_i^0$, $\mathbf{Z}_i^0$ and y^0 as

$$\mathbf{X}_i^0 = \begin{cases} I_{(J+1)n}, & 0 \le i < L+M-NJ \\ I_{Jn}, & L+M-NJ \le i < N, \end{cases}$$
$$\mathbf{Z}_i^0 = \mathbf{X}_i^0 \quad \text{and} \quad y^0 = \vec{0} \in \mathbb{R}^K,$$

where $I_n \in \mathbb{R}^{n\times n}$ is the identity matrix.

2) compute $TR_i^0 \in \mathbb{R}$ as follows.

$$TR_i^0 = tr(\mathbf{Z}_i^0 \mathbf{X}_i^0)$$

3) send TR_i^0 to processor root.
4) set $m = 0$.

Root Initialization step:
Root processor:

1) for $i = 0,\cdots,N-1$, receive TR_i^0 from processor i.
2) set μ as $\mu = \frac{1}{3}\sum_{i=0}^{N-1} TR_i^0$.
3) set $a = \vec{1} \in \mathbb{R}^K$ and $y_0 = \vec{0} \in \mathbb{R}^K$.
4) set primal and dual step sizes as $t_p = 1$ and $t_d = 1$.
5) receive d_p as the degree of $P(\alpha)$ and n_u as the number of uncertain parameters from user.

Processors step 1:
For $i = 0,\cdots,N-1$, processor i:

1) Compute $TR_{i,k}^1 \in \mathbb{R}$ and $TR_{i,k,l}^2 \in \mathbb{R}$ as follows.

$$TR_{i,k}^1 = tr\left(\overline{\mathbf{A}}_{k,i}(\mathbf{Z}_i^m)^{-1}\left(-\sum_{j=1}^K y_j^m \overline{\mathbf{A}}_{j,i} + \mathbf{Z}_i^m + \overline{\mathbf{C}}_i\right)\mathbf{X}_i^m\right)$$
$$\text{for} \quad k = 1,\cdots,K$$
$$TR_{i,k,l}^2 = tr\left(\overline{\mathbf{A}}_{k,i}(\mathbf{Z}_i^m)^{-1}\overline{\mathbf{A}}_{l,i}\mathbf{X}_i^m\right)$$
$$\text{for} \quad k = 1,\cdots,K \quad \text{and} \quad l = 1,\cdots,K$$

2) send $TR_{i,k}^1$ and $TR_{i,k,l}^2$ for $k = 1,\cdots,K$ and $l = 1,\cdots,K$ to root processor.

Root step 1:
Root processor:

1) receive $TR_{i,k}^1$ and $TR_{i,k,l}^2$ for $k = 1,\cdots,K$ and $l = 1,\cdots,K$ from processor i, where $i = 0,\cdots,N-1$.
2) compute D_1 and O according to

$$D_1 = \begin{pmatrix} \sum_{i=0}^{N-1} TR_{i,1}^1 \\ \sum_{i=0}^{N-1} TR_{i,2}^1 \\ \vdots \\ \sum_{i=0}^{N-1} TR_{i,K}^1 \end{pmatrix} - a$$

and

$$O = \left[\begin{pmatrix} \sum_{i=0}^{N-1} TR_{i,1,1}^2 \\ \sum_{i=0}^{N-1} TR_{i,2,1}^2 \\ \vdots \\ \sum_{i=0}^{N-1} TR_{i,K,1}^2 \end{pmatrix},\cdots,\begin{pmatrix} \sum_{i=0}^{N-1} TR_{i,1,K}^2 \\ \sum_{i=0}^{N-1} TR_{i,2,K}^2 \\ \vdots \\ \sum_{i=0}^{N-1} TR_{i,K,K}^2 \end{pmatrix}\right]$$

3) solve the following system of linear equations for $\Delta\widehat{y}^m \in \mathbb{R}^K$.

$$O\Delta\widehat{y}^m = D_1$$

4) send $\Delta\widehat{y}^m$ to all processors.

Processors step 2:
For $i = 0,\cdots,N-1$, processor i:

1) receive $\Delta\widehat{y}^m$ from root.
2) compute predictor step directions as follows.

$$\Delta\widehat{\mathbf{X}}_i^m = -\mathbf{X}_i^m + (\mathbf{Z}_i^m)^{-1}\left(-\sum_{j=1}^K y_j^m \overline{\mathbf{A}}_{j,i} + \mathbf{Z}_i^m + \overline{\mathbf{C}}_i\right)\sum_{j=1}^K \Delta\widehat{y}_j^m \overline{\mathbf{A}}_{j,i}\,\mathbf{X}_i^m$$
$$\Delta\widehat{\mathbf{Z}}_i^m = \sum_{j=1}^K y_j^m \overline{\mathbf{A}}_{j,i} - \mathbf{Z}_i^m - \overline{\mathbf{C}}_i + \sum_{j=1}^K \Delta\widehat{y}_j^m \overline{\mathbf{A}}_{j,i}$$

3) compute $TR_{i,k}^3 \in \mathbb{R}$ and $TR_{i,k}^4 \in \mathbb{R}$ according to

$$TR_{i,k}^3 = tr(\overline{\mathbf{A}}_{k,i}(\mathbf{Z}_i^m)^{-1}) \quad \text{for} \quad k = 1,\cdots,K$$
$$TR_{i,k}^4 = tr(\overline{\mathbf{A}}_{k,i}(\mathbf{Z}_i^m)^{-1}\Delta\widehat{\mathbf{Z}}_i^m\Delta\widehat{\mathbf{X}}_i^m) \quad \text{for} \quad k = 1,\cdots,K.$$

4) send $TR_{i,k}^3$ and $TR_{i,k}^4$ for $k = 1,\cdots,K$ to root processor.

Root step 2:
Root processor:

1) for $k = 1,\cdots,K$, receive $TR_{i,k}^3$ and $TR_{i,k}^4$ from processor i, where $i = 0,\cdots,N-1$.
2) compute D_2 according to

$$D_2 = \mu\begin{pmatrix} \sum_{i=0}^{N-1} TR_{i,1}^3 \\ \sum_{i=0}^{N-1} TR_{i,2}^3 \\ \vdots \\ \sum_{i=0}^{N-1} TR_{i,K}^3 \end{pmatrix} - \begin{pmatrix} \sum_{i=0}^{N-1} TR_{i,1}^4 \\ \sum_{i=0}^{N-1} TR_{i,2}^4 \\ \vdots \\ \sum_{i=0}^{N-1} TR_{i,K}^4 \end{pmatrix}$$

3) solve the following system of linear equations for $\Delta\overline{y}^m$.

$$O\Delta\overline{y}^m = D_2$$

4) send $\Delta\overline{y}^m$ to all processors.

Processors step 3:

For $i = 0, \cdots, N-1$, processor i:

1) receive $\Delta\overline{y}^m$ from root processor.
2) compute corrector step directions as follows.

$$\Delta\overline{\mathbf{Z}}_i^m = \sum_{j=1}^{K} \Delta\overline{y}_j^m \overline{\mathbf{A}}_{j,i}$$

$$\Delta\overline{\mathbf{X}}_i^m = -(\mathbf{Z}_i^m)^{-1}(\Delta\overline{\mathbf{Z}}_i^m \mathbf{X}_i^m + \Delta\widehat{\mathbf{Z}}_i^m \Delta\widehat{\mathbf{X}}_i^m) + \mu(\mathbf{Z}_i^m)^{-1}$$

3) compute primal dual step total step directions according to

$$\Delta\mathbf{X}_i^m = \Delta\widehat{\mathbf{X}}_i^m + \Delta\overline{\mathbf{X}}_i^m \ , \ \Delta\mathbf{Z}_i^m = \Delta\widehat{\mathbf{Z}}_i^m + \Delta\overline{\mathbf{Z}}_i^m,$$

$$\Delta y^m = \Delta\widehat{y}^m + \Delta\overline{y}^m.$$

4) set primal and dual step sizes equal to one.

$$t_p = 1, \quad t_d = 1$$

5) update $\mathbf{X}_i$ as follows.

$$\mathbf{X}_i^{m+1} = \mathbf{X}_i^m + t_p \Delta\mathbf{X}_i^m$$

6) check the positive definiteness of $\mathbf{X}_i^{m+1}$. If $\mathbf{X}_i^{m+1} \succ 0$, then set $k_{x_i} = 1$; Otherwise set $k_{x_i} = 0$.
7) send k_{x_i} to root processor.
8) update $\mathbf{Z}_i$ and y_i as follows.

$$\mathbf{Z}_i^{m+1} = \mathbf{Z}_i^m + t_d \Delta\mathbf{Z}_i^m \quad , \quad y^{m+1} = y^m + t_d \Delta y^m$$

9) check the positive definiteness of $\mathbf{Z}_i^{m+1}$. If $\mathbf{Z}_i^{m+1} \succ 0$, then set $k_{z_i} = 1$; Otherwise set $k_{z_i} = 0$.
10) send k_{z_i} to root processor.

Root step 3:

Root processor:

1) for $i = 0, \cdots, N-1$ receive k_{x_i} and k_{z_i} from processor i.
2) for $i = 0, \cdots, N-1$, If any of $k_{x_i} = 0$, then set $t_p = 0.8 t_p$, send t_p to all processors and repeat processors steps 3.5, 3.6 and 3.7.
3) for $i = 0, \cdots, N-1$, If any of $k_{z_i} = 0$, then set $t_d = 0.8 t_d$, send t_d to all processors and repeat processors steps 3.8, 3.9 and 3.10.

Processors step 4:

For $i = 0, \cdots, N-1$, processor i:

1) compute $TR_i^5 \in \mathbb{R}$ and $TR_i^6 \in \mathbb{R}$ as follows.

$$TR_i^5 = tr\left(\overline{\mathbf{C}}_i \mathbf{X}_i^{m+1}\right) \quad , \quad TR_i^6 = tr\left(\mathbf{Z}_i^{m+1} \mathbf{X}_i^{m+1}\right)$$

2) send TR_i^5 and TR_i^6 to root processor.
3) sets the new value of $m = m+1$.

Root step 4:

Root processor:

1) receive TR_i^5 and TR_i^6 from processor i, where $i = 0, \cdots, N-1$.
2) update μ as $\mu = \frac{1}{3}\sum_{i=0}^{N-1} TR_i^6$. and send it to all processors.
3) calculate primal and dual costs as follows.

$$\phi = \sum_{i=0}^{N-1} TR_i^5, \quad \psi = a^T y$$

4) check the stopping criterion: If $|\phi - \psi| > \varepsilon$, then return to Processors step 1; Otherwise the algorithm converges. In this case, compute

$$L_0 = \frac{(d_p + n_u - 1)!}{(d_p)!(n_u - 1)!}.$$

For $i = 1, \cdots, L_0$ compute

$$P_i = \sum_{j=1}^{N} E_j y_{(j+N(i-1))}^{m-1},$$

where recall E_j is the standard basis for $\mathbb{S}_n$ and $N = \frac{1}{2}n(n+1)$.

Outputs:

If the algorithm converges, the outputs of the algorithm are $P_i \in \mathbb{R}^{n\times n}$ for $i = 1, \cdots, L_0$, which are the coefficients of monomials in $P(\alpha)$ in lexicographical order. In this case the system is stable and the Lyapunov function is $V(x,\alpha) = x^T P(\alpha) x$.

D. Computational Complexity Analysis

To show the computational advantages of the proposed parallel algorithm in solving large-scale robust stability problems, the computational complexity of the algorithm is investigated in the following cases.

Case 1: *Systems with large number of states*

Consider the following uncertain linear system

$$\dot{x}(t) = \mathrm{A}(\alpha) x(t)$$

where $\mathrm{A} \in \mathbb{R}^{n\times n}$ and $\alpha \in \mathbb{R}^l$ is the vector of uncertain parameters. If the number of available processors N is sufficiently large (equal or greater than the number of sub-blocks in C as defined in (21)), then each processor performs

$$\simeq \frac{((d_p + l - 1)!)^2}{(d_p!)^2((l-1)!)^2} n^7. \tag{50}$$

operations. Therefore, for systems with large n and fixed d_p and l, the number of operations per processor required to solve the SDP associated with parameter-dependent feasibility problem $\mathrm{A}(\alpha)^T P(\alpha) + P(\alpha)\mathrm{A}(\alpha) \prec 0$, is proportional to n^7. Solving the LMI associated with the parameter-independent problem

$$\mathrm{A}^T P + P\mathrm{A} \prec 0.$$

also requires $O(n^7)$ operations per processor. Therefore, if we have a sufficient number of processors, the algorithm solves both the stability and robust stability problems performing $O(n^7)$ operations per processor.

Case 2: *Accuracy improvement*

Consider the definition of simplex as follows.

$$\Delta_l = \left\{ \alpha \in \mathbb{R}^l, \sum_{i=1}^{l} \alpha_i = r, \alpha_i \geqslant 0 \right\}$$

The accuracy of the algorithm is defined as the largest value of r found by the algorithm (if exists) such that if the uncertain parameters are inside the corresponding simplex, the stability of the system is verified. Typically, increasing Polya's exponent d in (4) improves the accuracy of the algorithm. According to (50), the number of processor operations is independent of d. The number of root operations

$$\simeq \frac{((d_p + l - 1)!)^3}{(d_p!)^3((l-1)!)^3} n^6, \tag{51}$$

and the number of communication operations

$$\simeq N \frac{((d_p + l - 1)!)^2}{(d_p!)^2((l-1)!)^2} n^4.$$

are also independent of d. Therefore, for a fixed d_p and sufficiently large N, improving the accuracy by increasing d does

not add any computation per processor and communicational overhead.

Case 3: *Algorithm scalability*
The speed-up of a parallel algorithm is defined as

$$SP_N = \frac{T_s}{T_N},$$

where T_s is the execution time of the sequential algorithm and T_N is the execution time using N processors. The speed-up is governed by

$$SP_N = \frac{N}{D+NS}, \tag{52}$$

where D is defined as the ratio of total operations performed by all processors except root to total operations performed by all processors and root. S is the ratio of operations performed by root to total operations performed by all processors and root. Suppose that the number of available processors is equal to the number of sub-blocks in C defined in (21). Using the definitions of D and S and equations (50) and (51), D and S can be approximated as

$$D \simeq \frac{N\frac{((d_p+l-1)!)^2}{(d_p!)^2((l-1)!)^2}n^7}{N\frac{((d_p+l-1)!)^2}{(d_p!)^2((l-1)!)^2}n^7 + \frac{((d_p+l-1)!)^3}{(d_p!)^3((l-1)!)^3}n^6}$$

and

$$S \simeq \frac{\frac{((d_p+l-1)!)^3}{(d_p!)^3((l-1)!)^3}n^6}{N\frac{((d_p+l-1)!)^2}{(d_p!)^2((l-1)!)^2}n^7 + \frac{((d_p+l-1)!)^3}{(d_p!)^3((l-1)!)^3}n^6}.$$

The number of sub-blocks in our algorithm is $L+M$, where according to (15) and (16), L and M are independent of n. Therefore, the number of processors $N = L+M$ is also independent of n. Therefore,

$$\lim_{n\to\infty} D = \lim_{n\to\infty} \frac{Nn^7}{Nn^7} = 1 \quad \text{and} \quad \lim_{n\to\infty} S = \lim_{n\to\infty} \frac{n^6}{Nn^7+n^6} = 0.$$

By substituting D and S in (52) with their limit values, we have

$$\lim_{n\to\infty} SP_N = N.$$

Thus, for large n, by using $L+M$ processors the present parallel algorithm solves large robust stability problems $L+M$ times faster than the sequential algorithms. Generally, for problems with large n, it can be shown that by using $N \le L+M$ processors the parallel algorithm solves the robust stability problems approximately N times faster than the sequential algorithm. For different values of n, the algorithm speed-up versus number of processors is illustrated in Fig 1. According to Fig 1, as n increases, the trend of speed-up becomes more linear. Therefore, in case of problems with large number of states, n, our algorithm is highly scalable.

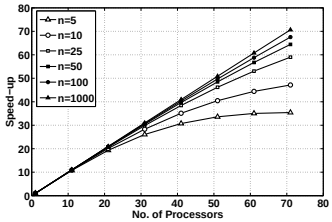

Fig. 1. Theoretical speed-up vs. No. of processors for different system dimensions n

TABLE I
COMPUTATION TIME T OF THE SDP ALGORITHM FOR DIFFERENT NUMBER OF PROCESSORS N

N	1	2	3	4	5
$T(s)$	247.32	129.13	87.35	66.24	53.16

VI. EXAMPLE

We evaluate the computation time and scalability of the proposed parallel SDP solver in the following examples.

Example 1: A simplified model for the poloidal magnetic flux gradient in a Tokamak reactor [24] is

$$\frac{\partial \psi_x(x,t)}{\partial t} = \frac{1}{\mu_0 a^2}\frac{\partial}{\partial x}\left(\frac{\eta(x)}{x}\frac{\partial}{\partial x}(x\psi_x(x,t))\right)$$

with the boundary conditions $\psi_x(0,t)=0$ and $\psi_x(1,t)=0$, where ψ_x is the deviation of the flux gradient from a reference flux gradient profile, μ_0 is the permeability of free space, $\eta(x)$ is the plasma resistivity and a is the radius of the last closed magnetic surface (LCMS). To obtain the finite-dimensional state-space representation of the PDE, we discretize the PDE in the spatial domain $(0,1)$ at 7 points. The linear state-space model is

$$\dot{\psi}_x(t) = A(\eta(x))\psi_x(t), \tag{53}$$

where $A(\eta(x)) \in \mathbb{R}^{7\times 7}$. Typically $\eta(x)$ is not precisely known; Therefore at each discretization point we substitute $\eta(x)$ in (53) with $\widehat{\eta}(x)+\alpha$, where $\widehat{\eta}(x)$ is the known nominal value of $\eta(x)$ at that point and α is the uncertainty. Then the uncertain system can be written as

$$\dot{\psi}_x(t) = A(\alpha)\psi_x(t). \tag{54}$$

The uncertain parameters α_j belong to S_B, which is defined as

$$S_B := \{\alpha \in \mathbb{R}^8 : \sum_{i=1}^{8}\alpha_i = -6|B|, -|B| \le \alpha_i \le |B|\},$$

where the optimal value of B is to be found by solving the following optimization problem.

$$\begin{aligned} &\max \quad B \\ &\text{s.t.} \quad \text{system (54) is stable for all } \alpha \in S_B \end{aligned} \tag{55}$$

To transform S_B into the unit simplex defined in (2), we use the map $f : S_B \to \Delta_8$ defined as

$$\alpha' = f(\alpha) = \frac{1}{2|B|}[\alpha_1+|B|, \cdots, \alpha_8+|B|].$$

By writing all α_j in terms of α'_j using the map f, the linear uncertain state-space model with the uncertain parameters inside a unit simplex can be written as $\dot{\psi}_x(t) = \left(\sum_{i=1}^{8} A_i\alpha'_i\right)\psi_x(t)$. We have omitted the A_i matrices due to the limitation of space. We solve the optimization problem in (55) using bisection search on B. For each trial value of B, we use the proposed parallel SDP solver to solve the associated SDP with the elements defined in (21), (23) and (26). The SDP problems have 224 constraints with the primal variable $X \in \mathbb{R}^{1092\times 1092}$. Optimal B is found to be 1.60×10^{-8}. In this particular example, the optimal value of B does not change with the degrees of $P(\alpha)$ and Polya's exponent d. The SDP problems are solved on a Core i7 machine. The computation time of SDP algorithm for different number of processors is presented in Table I. Note that solving this problem by SOSTOOLS [6] on the same machine is impossible due to the excessive amount of memory that SOS method requires.

Example 2: To evaluate the scalability of the algorithm, we solve three random SDP problems with different dimensions using a parallel Linux-based cluster Karlin at Illinois Institute of Technology. Fig. 2 demonstrates the algorithm speed-up with respect to the number of processors for SDP problems with different dimensions of the primal variable X. The dimensions of X are $(L+M)n = 80$, 200 and 500, where L and M are defined in (15) and (16). The linearity of the speed-up in all three cases implies the scalability of the parallel algorithm.

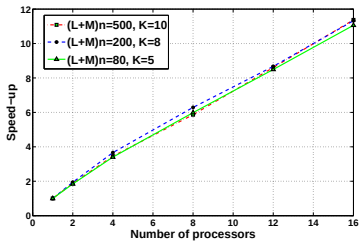

Fig. 2. Speed-up of SDP algorithm vs. the number of processors

Example 3: In this example, we ran our algorithm on a desktop computer with 24 Gig of RAM to solve the robust stability problem with dimension n and l uncertain parameters. Using trial and error, for different n and d we found the largest l for which the algorithm does not terminate due to insufficient memory (Fig. 3). In all of the runs $d_a = d_p = 1$.

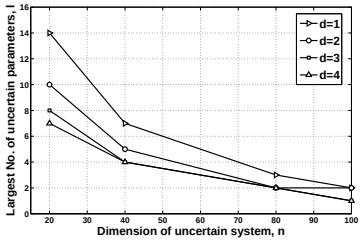

Fig. 3. Largest number of uncertain parameters of n-dimensional systems for which the algorithm can solve the robust stability problem of the system on a computer with 24 Gig of RAM

VII. CONCLUSIONS

A parallel optimization algorithm based on an interior-point primal-dual algorithm is proposed to solve large-scale LMIs involved in robust stability analysis. In the present work, these LMIs come from the application of Polya's algorithm on parameter-dependent Lyapunov inequalities. It is shown that the SDP problem associated with these LMIs has block-diagonal structure. By taking advantage of such structure, the data and operations of the algorithm are decentralized with little communication overhead. The algorithm was used to investigate the stability of a model for the gradient of the poloidal magnetic flux in a Tokamak reactor with a large number of uncertain parameters.

ACKNOWLEDGMENTS

The authors gratefully acknowledge funding from NSF with the award number CMMI-1100376 for the present work.

REFERENCES

[1] R. C. L. F. Oliveira and P. L. D. Peres, "Stability of polytopes of matrices via affine parameter-dependent Lyapunov functions: Asymptotically exact LMI conditions," *Linear Algebra Appl.*, vol. 405, pp. 209–228, Aug 2005.

[2] R. C. L. F. Oliveira and P. L. D. Peres, "A less conservative LMI condition for the robust stability of discrete-time uncertain systems," *Syst. Control Lett.*, vol. 43, pp. 371–378, Aug 2001.

[3] G. Chesi, A. Garulli, A. Tesi, and A. Vicino, "Robust stability of polytopic systems via polynomially parameter-dependent Lyapunov functions," Proceedings of the 42nd IEEE Conference on Decision and Control, Dec 2003.

[4] R. C. L. F. Oliveira and P. L. D. Peres, "Parameter-dependent LMIs in robust analysis: Characterization of homogeneous polynomially parameter-dependent solutions via LMI relaxations," *IEEE Transactions on Automatic Control*, vol. 52, pp. 1334–1340, Jul 2007.

[5] A. Ben-Tal and A. Nemirovski, "Robust convex optimization," *Math. Operat. Res.*, vol. 23, no. 4, pp. 769–805, 1998.

[6] P. Rajna, A. Papachristodoulou, and P. A. Parrilo, "Introducing SOSTOOLS: a general purpose sum of squares programming solver," Proceedings of IEEE Conference on Decision and Control, 2002.

[7] D. Henrion and J. B. Lassere, "Gloptipoly: Global optimization over polynomials with Matlab and SeDuMi," Proceedings of IEEE Conference on Decision and Control, Mar 2003.

[8] C. W. Scherer and C. W. J. Hol, "Matrix sum-of squares relaxations for robust semi-definite programs," *Math. programming Ser. B*, vol. 107, no. 1-2, pp. 189–211, 2006.

[9] S. Furber, "The future of computer technology and its implications for the computer industry," *The Computer Journal*, vol. 51, no. 6, 2008.

[10] J. Sturm, "Using sedumi 1.02, a MATLAB toolbox for optimization over symmetric cones," *Optimization Methods and Software*, vol. 11-12, pp. 625–653, 1999.

[11] K. Toh, M. Todd, and R. Tutuncu, "A Matlab software package for semidefinite programming," *Optimization Methods and Software*, vol. 11, pp. 545–581, 1999.

[12] G. M. Amdahl, "Validity of the single processor approach to achieving large-scale computing capabilities," No. 30, pp. 483–485, AFIPS Conference Proceedings, 1967.

[13] B. Borchers and J. G. Young, "Implementation of a primal dual method for SDP on a shared memory parallel architecture," *Computational Optimization and Applications*, vol. 37, no. 3, pp. 355–369, 2007.

[14] M. Yamashita, K. Fujisawa, and M. Kojima, "SDPARA: Semidefinite programming algorithm parallel version," *Parallel Computing*, vol. 29, pp. 1053–1067, 2003.

[15] C. W. Scherer, "Relaxations for robust linear matrix inequality problems with verifications for exactness," *SIAM Journal on Optimization*, vol. 27, no. 2, pp. 365–395, 2005.

[16] G. Hardy, J. E. Littlewood, and G. Pólya, *Inequalities*. Cambridge University Press, 1934.

[17] M. Castle, V. Powers, and B. Reznick, "A quantitative polya's theorem with zeros," *Effective Methods in Algebraic Geometry*, vol. 44, no. 9, pp. 1285–1290, 2009.

[18] M. M. Peet and Y. V. Peet, "A parallel-computing solution for optimization of polynomials," Proceedings of the American Control Conference, Jun-Jul 2010.

[19] S. J. Benson, Y. Ye, and X. Zhang, "Solving large-scale sparse semidefinite programs for combinatorial optimization," *SIAM Journal on Optimization*, vol. 10, pp. 443–461, 1998.

[20] F. Alizadeh, J. A. Haeberly, and M. Overton, "Primal-dual interior-point methods for semidefinite programming: Convergence rates, stability and numerical results," *SIAM Journal on Optimization*, vol. 8, no. 3, pp. 746–768, 1998.

[21] R. D. C. Monteiro, "Primal-dual path following algorithms for semidefinite programming," *SIAM Journal on Optimization*, vol. 7, no. 3, 1997.

[22] C. Helmberg, F. R. R. J. Vanderbei, and H. Wolkovicz, "An interior-point method for semidefinite programming," *SIAM Journal on Optimization*, vol. 6, pp. 342–361, 1996.

[23] S. J. Benson, "DSDP3: Dual scaling algorithm for general positive semidefinite programs," *Preprint ANL/MCS-P851-1000, Argonne National Labs*, 2001.

[24] E. Witrant, E. Joffrin, S. Brémont, G. Giruzzi, D. Mazon, O. Barana, and P. Moreau, "A control-oriented model of the current profile in tokamak plasma," *Plasma Physics and Controlled Fusion*, vol. 49, pp. 1075–1105, 2007.