Solving Large-Scale Robust Stability Problems by Exploiting the Parallel Structure of Polya's Theorem

Reza Kamyar, Matthew Peet and Yulia Peet

Abstract

In this paper, we propose a distributed computing approach for large-scale robust stability problems. First, we design a decentralized algorithm which uses Polya's algorithm to convert a robust stability problem into a set of highly structured linear matrix inequalities. Then, we design a decentralized primal-dual interior-point algorithm capable of exploiting this structure. Both algorithms have fixed computation, memory, and communication complexity. Numerical tests on multi-core computers, cluster computers, and supercomputers demonstrate the scalability of the algorithms for hundreds of processors and 100+ dimensional state space.

I. Introduction

The design of a control system is based on the model of the system we want to control. Often, the process of obtaining a model is more art than science and there is considerable uncertainty in the resulting set of differential equations. Other times, the models we obtain are nonlinear or otherwise complex. The process of reducing a complex or nonlinear model to a tractable approximation often introduces additional uncertainty or parametric dependence. Finally, the dynamics of a system itself may depend on changing environmental conditions or operating points. In order to design a controller which will respond appropriately in all circumstances, we must account for this dependence. The result is often a dynamic model with dozens of states and uncertain parameters.

R. Kamyar is a Ph.D student with the Department of Mechanical, Materials, and Aerospace Engineering, Illinois Institute of Technology, 10 West 32nd Street, Cybernetic Systems and Control Lab E1-030, Chicago, IL 60616, U.S.A. rkamyar@iit.edu

M. M. Peet is an assistant professor with the Department of Mechanical, Materials, and Aerospace Engineering, Illinois Institute of Technology, 10 West 32nd Street, E1-252B, Chicago, IL 60616, U.S.A. mpeet@iit.edu

Y. Peet is a computational scientist at Argonne National Laboratory, Argonne, IL peet@mcs.anl.gov

In this paper, we address the problem of stability and control of large systems with parameters whose values are known within a tolerance. The problem of control of systems with uncertainty has been widely studied. See, e.g. the texts [1], [2], [3], [4]. A limitation of existing methods of controller synthesis for systems with uncertainty is the computational complexity of these methods. For systems with hundreds of states or uncertain parameters, standard approaches using desktop computers inevitably fail. For this reason, we would like to utilize more powerful computing resources such as cluster-computing, supercomputing and GPU-computing. The problem with all these computing architectures is that they all highly parallel - composed of hundreds or thousands of networked computing units. This is a problem because the most common tool for solving robust control problems - Linear Matrix Inequalities (LMIs) - is known to be an inherently sequential problem [5]. This means that unless the LMI has significant structure, parallel computing platforms cannot be used efficiently to speed up the calculations. In this paper, we propose a structured approach to robust analysis and synthesis which can fully exploit the power of parallel computing architectures to handle systems with hundreds of states or dozens of uncertain parameters.

We focus on methods which are based on the existence of Lyapunov functions, the most basic of which is quadratic stability [6], [7]. Quadratic stability uses a fixed Lyapunov function to obtain Linear Matrix Inequality conditions. To reduce the conservatism of quadratic stability, affine parameter-dependent Lyapunov functions for systems with time-invariant uncertain parameters were used in [8] and expanded in, e.g. [9], [10], [11], [12]. In general, it is known that stability of a parameter-dependent system implies the existence of a parameter-dependent Lyapunov function. However, the search for a general-form parameter-dependent Lyapunov function is equivalent to feasibility of a parameter-dependent LMI - an intractable problem [13]. The problem can be simplified through the use of Lyapunov functions which are polynomial in the uncertain parameters. In [14], it was shown that any feasible parameter-dependent LMI with parameters inside a compact set has polynomial solution. Using this result, [15] provides a set of conditions for robust stability with polynomially parameter-dependent quadratic Lyapunov functions. Necessary and sufficient stability conditions for linear systems with one uncertain parameter are derived in [16], providing an explicit bound on the degree of the polynomial-type Lyapunov function. The result is extended to multi-parameter dependent linear systems in [17]. An interesting special case is the homogeneous polynomially parameter-dependent quadratic

Lyapunov functions for the systems with uncertain parameters inside a unit simplex. Using this type of Lyapunov functions, [18] provides sufficient stability conditions using Polya's theorem and [19] provides sufficient stability conditions using Complete Square Matricial Representation (CSMR).

The feasibility of polynomially-parameter-dependent LMIs is known to be NP-hard [13]. A common algorithm for solving parameter-dependent LMIs is Sum of Squares (SOS). This algorithm optimizes over positive polynomials with scalar or matrix coefficients [20], [21], [22]. Unfortunately, the number of variables in Semi-Definite Programming (SDP) problems associated with the SOS algorithm can easily grow beyond ten thousand as the number of uncertain parameters and/or the degree of the 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 [23].

In this paper, we would like to efficiently exploit the power of cluster-computing and super-computing resources. However, standard LMI solvers [24], [25] can only utilize a single processor of cluster computers. Moreover, because semidefinite programming is an inherently sequential problem, Amdahl's law [26] dictates that the speed of general-purpose parallel SDP solvers [27], [28] will inevitably saturate as the number of processors increases. To take full advantage of the computational power of advanced computing architectures, 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 [22]. A performance comparison of the SOS algorithm and Polya's algorithm in robust stability analysis can be found in [18]. An extension of the Polya approach for stability analysis of linear uncertain systems with simultaneous time-invariant and time-varying uncertain parameters inside a multi-simplex can be found in [29].

The first main result of this paper is a highly scalable parallel algorithm with no centralized computation which uses Polya's theorem to construct a large-scale structured SDP problem which will solve general polynomially-parameter-dependent LMI problems. The second contribution is

to show how to use the structure of the large-scale 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 parameter-dependent 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.

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 $\gamma, \eta \in \mathbb{N}^l$. We define the lexicographical ordering of the monomials as α^γ precedes δ^η if the left most non-zero entry of $\gamma - \eta$ is positive. We use $W_d \subset \mathbb{N}^l$ to denote the circle of radius d, $W_d := \left\{\gamma \in \mathbb{N}^l \,:\, \sum_{i=1}^{l} \gamma_i = d\right\}$. The subspace of partially ordered symmetric matrices in $\mathbb{R}^{n\times n}$ is denoted by $\mathbb{S}_n$. The standard basis for $\mathbb{S}_n$ is defined as

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

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

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

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

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 where $\mathrm{A}(\alpha)$ is a homogeneous polynomial 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 [18].

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 [13]. 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 [30]. 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 [31] 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 homogeneous 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 $W_d \subset \mathbb{N}^l$ is the circle of radius d, $W_d := \left\{\gamma\in\mathbb{N}^l : \sum_{i=1}^{l}\gamma_i = d\right\}$, where l is the dimension of the vector α. Since A is 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

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

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

for some set of coefficients $\{\beta_{h,\gamma}\}$ and $\{H_{h,\gamma}\}$.

Example: 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

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

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

$$\begin{aligned}(\alpha_1+\alpha_2)\left(A^T(\alpha)P(\alpha)+P(\alpha)A(\alpha)\right) &= \left(A_{[1,0]}^T P_{[1,0]}+P_{[1,0]}A_{[1,0]}\right)\alpha_1^3\\ &+\left(A_{[1,0]}^T P_{[1,0]}+P_{[1,0]}A_{[1,0]}+A_{[0,1]}^T P_{[1,0]}+P_{[1,0]}A_{[0,1]}+A_{[1,0]}^T P_{[0,1]}+P_{[0,1]}A_{[1,0]}\right)\alpha_1^2\alpha_2\\ &+\left(A_{[0,1]}^T P_{[1,0]}+P_{[1,0]}A_{[0,1]}+A_{[1,0]}^T P_{[0,1]}+P_{[0,1]}A_{[1,0]}+A_{[0,1]}^T P_{[0,1]}+P_{[0,1]}A_{[0,1]}\right)\alpha_1\alpha_2^2\\ &+\left(A_{[0,1]}^T P_{[0,1]}+P_{[0,1]}A_{[0,1]}\right)\alpha_2^3\end{aligned}$$

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

$$H_{[1,0],[3,0]} = A_{[1,0]}, \quad H_{[0,1],[3,0]} = \mathbf{0}, \quad H_{[1,0],[2,1]} = A_{[1,0]} + A_{[0,1]}, \quad H_{[0,1],[2,1]} = A_{[1,0]},$$

$$H_{[1,0],[1,2]} = A_{[0,1]}, \quad H_{[0,1],[1,2]} = A_{[1,0]} + A_{[0,1]}, \quad H_{[1,0],[0,3]} = \mathbf{0}, \quad H_{[0,1],[0,3]} = A_{[0,1]}.$$

General Case: We define $\{\beta_{h,\gamma}\}$ recursively as follows. Set the initial values for $\{\beta_{h,\gamma}\}$ as

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

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

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

Finally, set $\{\beta_{h,\gamma}\} = \{\beta^d_{h,\gamma}\}$. To obtain $H_{h,\gamma}$, set the initial values for $\{H_{h,\gamma}\}$

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

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

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

Finally, set $\{H_{h,\gamma}\} = \{H^d_{h,\gamma}\}$. In order to avoid excessive monomial subscripts, in our algorithm, we replace the monomial subscript with the index of the monomial within the circle using lexicographical ordering. Thus with some abuse of notation, for $j, k \in \mathbb{N}$ we will occasionally use $\beta_{k,j}$ to denote β_{h_k,γ_j} and P_k to denote P_{h_k}, where h_k is the k^{th} element of W_{d_p} and γ_j is the j^{th} element of W_{d_p+d} in lexicographical ordering. Likewise, we denote $H_{k,j} = H_{h_k,\gamma_j}$, where h_k is the k^{th} element of W_{d_p} and γ_j is the j^{th} element of $W_{d_p+d_a+d}$ in lexicographical ordering.

Computing $\beta_{k,i}$ and $H_{k,j}$ for $i = 1, \cdots, L$, $j = 1, \cdots, M$ and $k = 1, \cdots, L_0$ is a significant challenge. For a given d, the number of $\beta_{k,i}$ 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 $H_{k,j}$ 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 [32], we proposed a decentralized computing approach to the calculation of the coefficients $\beta_{k,j}$. In the present work, we extend this method to the calculation of $H_{k,j}$ and the SDP elements which will be discussed in the following section (Section IV-B).

In the following section, we express the LMIs associated with conditions (9) and (10) in the format of an SDP using both primal and dual form. 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 dual format.

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

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

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

$$B(X) = \begin{bmatrix} tr(B_1X) & tr(B_2X) & \cdots & tr(B_KX) \end{bmatrix}^T. \tag{18}$$

$X \in \mathbb{S}_m$ is the primal variable. Given a primal SDP, the associated **dual problem** is

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

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

$$B^T(y) = \sum_{i=1}^{K} y_i B_i, \tag{20}$$

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

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

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

where

$$C_j = \begin{cases} \delta I_n \cdot \left(\sum_{k=1}^{L_0} \beta_{k,j} \frac{d_p!}{\prod_{i=1}^{l} (h_k)_i!} \right), & 1 \le j \le L \\ 0_n, & L+1 \le j \le L+M, \end{cases} \tag{22}$$

where recall that L_0 is the number of monomials in $P(\alpha)$, $h_k \in \mathbb{N}^l$ is the k^{th} element of W_{d_p} in lexicographical ordering, L is the cardinality of W_{d_p+d}, M is the cardinality of $W_{d_p+d_a+d}$, n

is the dimension of system (1), I_n and 0_n are the identity and zero matrices of dimension n, l is the number of uncertain parameters and δ is a small positive parameter.

For $i = 1, \cdots, K$, the elements B_i are defined as

$$B_i = \text{diag}(B_{i,1}, \cdots B_{i,L}, B_{i,L+1}, \cdots B_{i,L+M}). \tag{23}$$

The number of dual variables is

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

Define

$$V_k(y) = \sum_{j=1}^{\tilde{N}} E_j \; y_{j+\tilde{N}(k-1)} \quad \text{for} \quad k = 1, \cdots, L_0,$$

where recall E_j is the basis for $\mathbb{S}_n$ as defined in Section II and $\tilde{N} = \frac{n(n+1)}{2}$. Then

$$B_{i,j} = \begin{cases} \sum_{k=1}^{L_0} \beta_{k,j} V_k(e_i), & 1 \le j \le L \\ -\sum_{k=1}^{L_0} H_{k,j-L}^T V_k(e_i) + V_k(e_i) H_{k,j-L}, & L+1 \le j \le L+M. \end{cases} \tag{25}$$

Finally, set

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

This completes the SDP problem associated with Polya's algorithm.

C. A Parallel Algorithm and Complexity Analysis

In Appendix A, we describe in detail a distributed, iterative algorithm for calculating $\{\beta_{k,j}\}$ and $\{H_{k,j}\}$ and subsequently the C_j and $B_{i,j}$ as defined in (22) and (25). This algorithm distributes the $\{\beta_{k,j}\}$ and $\{H_{k,j}\}$ among the processors. In an ideal case, where the number of available processors is sufficiently large, only one β and one H coefficient is assigned to each processor. This algorithm is included in the appendix to streamline the presentation and because the complexity of the setup problem is typically significantly less than that of solving the SDP and hence is a secondary focus of the paper.

The most computationally expensive part of the set-up algorithm in Appendix A is the calculation of the coefficients $B_{k,j}^i$ and $\widehat{B}_{k,j}^i$ in step 12 of the algorithm. If the number of processors N is equal to the number of monomials in $\left(\sum_{i=l} \alpha_i\right)^d P(\alpha)$; i.e. $N = L$, it can be shown that the number of operations per processor at each iteration is proportional to $l^{2d_p + d_a + d} n^5$. Since $2d_p + d_a + d$ is often greater than 5, the number of operations grows more slowly in n than in l. In the worst case, where every processor sends all of its assigned coefficients to other processors, the communication complexity per processor at each iteration is proportional to $\left(\texttt{floor}(\frac{L}{N}) + \texttt{floor}(\frac{M}{N})\right) n^2 l$, where the number of processors $N \le L$. This indicates that increasing the number of processors (not greater than L) leads to less communication overhead per processor and improves the scalability of the algorithm.

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 [33], primal-dual [34], [35], [36] and dual scaling [37]. In this paper, we use the central-path-following primal-dual algorithm described in [36] and [27]. 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

$$\Delta \widehat{y} = O^{-1} \left(-a + B(Z^{-1} G X) \right)$$

$$\Delta \widehat{X} = -X + Z^{-1} G B^T(\Delta \widehat{y}) X \tag{33}$$

$$\Delta \widehat{Z} = B^T(y) - Z - C + B^T(\Delta \widehat{y}), \tag{34}$$

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

$$G = -B^T(y) + Z + C \tag{35}$$

$$O = \left[B(Z^{-1} B^T(e_1) X) \ \cdots \ B(Z^{-1} B^T(e_k) X) \right]$$

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

$$\Delta\overline{y} = O^{-1}\left(B(\mu Z^{-1}) - B(Z^{-1}\Delta\widehat{Z}\Delta\widehat{X})\right)$$

$$\Delta\overline{X} = \mu Z^{-1} - Z^{-1}\Delta\widehat{Z}\Delta\widehat{X} - Z^{-1}\Delta\overline{Z}X \tag{36}$$

$$\Delta\overline{Z} = B^T(\Delta\overline{y}). \tag{37}$$

The stopping criterion is $|a^T y - tr(CX)| \leq \epsilon$. Information regarding the convergence of different variants of interior-point primal-dual algorithm are presented in [34] and [35].

B. Structure of SDP Variables

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

$$S_{l,m,n} := \{Y \subset \mathbb{R}^{(l+m)n\times(l+m)n} : \ Y = \text{diag}(Y_1, \cdots Y_l, Y_{l+1}, \cdots Y_{l+m}) \quad \text{for} \quad Y_i \in \mathbb{R}^{n\times n}\} \tag{38}$$

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 (38).

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 $k \in \mathbb{N}$

$$X_k \in S_{L,M,n}, \quad Z_k \in S_{L,M,n}. \tag{39}$$

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

$$\Delta X_k = \Delta\widehat{X}_k + \Delta\overline{X}_k \quad \text{for all } \ k \in \mathbb{N}. \tag{40}$$

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

$$\Delta\widehat{X}_k = -X_k + Z_k^{-1}G_kB^T(\Delta\widehat{y}_k)X_k \quad \text{for all } \ k \in \mathbb{N}. \tag{41}$$

where according to (35), G_k is

$$G_k = C - B^T(y_k) + Z_k \quad \text{for all } \ k \in \mathbb{N}. \tag{42}$$

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

$$C \in S_{L,M,n}, \quad B^T : \mathbb{R}^K \mapsto S_{L,M,n}. \tag{43}$$

Since all the terms on the right hand side of (42) are in $S_{L,M,n}$ and the structure of matrices in $S_{L,M,n}$ are preserved through algebraic addition, we conclude

$$G_k \in S_{L,M,n}. \tag{44}$$

Returning to (41), using our assumption in (39) and noting that the structure of the matrices in $S_{L,M,n}$ is preserved through multiplication and inversion, we conclude

$$\Delta\widehat{X}_k \in S_{L,M,n}. \tag{45}$$

Using (36), the second term in (40) is

$$\Delta\overline{X}_k = \mu Z_k^{-1} - Z_k^{-1}\Delta\widehat{Z}_k\Delta\widehat{X}_k - Z_k^{-1}\Delta\overline{Z}_k X_k \quad \text{for all } k \in \mathbb{N}. \tag{46}$$

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 (34) and (37) we have

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

$$\Delta\overline{Z}_k = B^T(\Delta\overline{y}_k) \qquad \text{for all } k \in \mathbb{N}. \tag{48}$$

Since all the terms in the right hand side of (47) and (48) are in $S_{L,M,n}$, then

$$\Delta\widehat{Z}_k \in S_{L,M,n}, \quad \Delta\overline{Z}_k \in S_{L,M,n}. \tag{49}$$

Recalling (45), (46) and our assumption in (39), we have

$$\Delta\overline{X}_k \in S_{L,M,n}. \tag{50}$$

According to (45), (49) and (50), the total step directions are in $S_{L,M,n}$,

$$\Delta X_k = \Delta\widehat{X}_k + \Delta\overline{X}_k \in S_{L,M,n}$$

$$\Delta Z_k = \Delta\widehat{Z}_k + \Delta\overline{Z}_k \in S_{L,M,n},$$

and it follows that

$$X_{k+1} = X_k + t_p\Delta X_k \in S_{L,M,n}$$

$$Z_{k+1} = Z_k + t_p\Delta Z_k \in S_{L,M,n}.$$

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 $J = \big(\texttt{floor}(\frac{L}{N}) + \texttt{floor}(\frac{M}{N})\big)$. According to the parallel set-up algorithm presented in Appendix A, processor i has access to $\overline{\mathbf{C}}_i$ and $\overline{\mathbf{B}}_{j,i}$ defined in (61) and (62) for $j = 1, \cdots, K$. 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 $i = 1, \cdots, L+M$, $B_{j,i}$ for $i = 1, \cdots, L+M$ and $j = 1, \cdots, K$ (provided by set-up algorithm), degree of $P(\alpha)$ and the number of uncertain parameters.

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 $V(x, \alpha) = x^T P(\alpha) x$.

Processors Initialization step:

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

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

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

   where $I_n \in \mathbb{R}^{n\times n}$ is the identity matrix.
2) Compute $TR_i^0 \in \mathbb{R}$ as $TR_i^0 = tr(\mathbf{Z}_i^0 \mathbf{X}_i^0)$.
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 $a = \vec{1} \in \mathbb{R}^K$.

4) 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.

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

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

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

$$D_1 = \begin{pmatrix} \sum_{i=0}^{N-1} TR_{i,1}^1 \\ \sum_{i=0}^{N-1} TR_{i,2}^1 \\ \vdots \\ \sum_{i=0}^{N-1} TR_{i,K}^1 \end{pmatrix} - a \quad \text{and} \quad O = \left[\begin{pmatrix} \sum_{i=0}^{N-1} TR_{i,1,1}^2 \\ \sum_{i=0}^{N-1} TR_{i,2,1}^2 \\ \vdots \\ \sum_{i=0}^{N-1} TR_{i,K,1}^2 \end{pmatrix}, \cdots, \begin{pmatrix} \sum_{i=0}^{N-1} TR_{i,1,K}^2 \\ \sum_{i=0}^{N-1} TR_{i,2,K}^2 \\ \vdots \\ \sum_{i=0}^{N-1} TR_{i,K,K}^2 \end{pmatrix}\right]$$

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.

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

$$\Delta\widehat{\mathbf{Z}}_i^m = \sum_{j=1}^{K} y_j^m \overline{\mathbf{B}}_{j,i} - \mathbf{Z}_i^m - \overline{\mathbf{C}}_i + \sum_{j=1}^{K} \Delta\widehat{y}_j^m \overline{\mathbf{B}}_{j,i}$$

3) Compute $TR^3_{i,k} \in \mathbb{R}$ and $TR^4_{i,k} \in \mathbb{R}$ according to

$$TR^3_{i,k} = tr(\overline{\mathbf{B}}_{k,i}(\mathbf{Z}_i^m)^{-1}) \quad \text{for} \quad k = 1, \cdots, K$$

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

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

Root step 2:

Root processor:

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

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

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.

$$\Delta\overline{\mathbf{Z}}_i^m = \sum_{j=1}^{K} \Delta\overline{y}_j^m \overline{\mathbf{B}}_{j,i} \quad , \quad \Delta\overline{\mathbf{X}}_i^m = -(\mathbf{Z}_i^m)^{-1}(\Delta\overline{\mathbf{Z}}_i^m\mathbf{X}_i^m + \Delta\widehat{\mathbf{Z}}_i^m\Delta\widehat{\mathbf{X}}_i^m) + \mu(\mathbf{Z}_i^m)^{-1}$$

3) Compute primal dual step total step directions according to

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

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

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

5) Update $\mathbf{X}_i$ as $\mathbf{X}_i^{m+1} = \mathbf{X}_i^m + t_p\Delta\mathbf{X}_i^m$.
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 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 $TR_i^5 = tr\left(\overline{\mathbf{C}}_i \mathbf{X}_i^{m+1}\right)$ and $TR_i^6 = tr\left(\mathbf{Z}_i^{m+1} \mathbf{X}_i^{m+1}\right)$.
2) Send TR_i^5 and TR_i^6 to root processor.
3) Increment $m = m + 1$.

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 $\phi = \sum_{i=0}^{N-1} TR_i^5$ and $\psi = a^T y^m$.
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

$$P_i = \sum_{j=1}^{\tilde{N}} E_j y^m_{(j+\tilde{N}(i-1)),}$$

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

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 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 is $N = L + M$ (equal to the number of sub-blocks in C as defined in (21)), then each processor performs

$$\simeq \frac{((d_p + l - 1)!)^2}{(d_p!)^2((l-1)!)^2} n^7. \tag{51}$$

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 (51), the number of processor operations is independent of d. The number of root operations

$$\simeq \frac{((d_p + l - 1)!)^3}{(d_p!)^3((l-1)!)^3} n^6, \tag{52}$$

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

$$SP_N = \frac{N}{D + NS}, \tag{53}$$

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 (51) and (52), 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

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

According to (15) and (16) the number of processors $N = L+M$ is independent of n; Therefore

$$\lim_{n\to\infty} D = 1 \quad \text{and} \quad \lim_{n\to\infty} S = 0.$$

By substituting D and S in (53) 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. For different values of n, the theoretical algorithm speed-up versus the number of processors is illustrated in Fig. 1. As shown in Fig. 1, for problems with large n, by using $N \leq L + M$ processors the parallel algorithm solves the robust stability problems approximately N times faster than the sequential algorithm. 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.

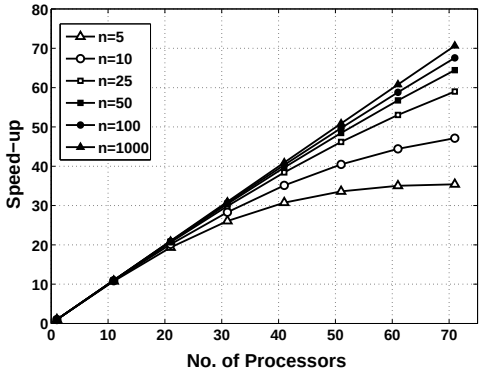

Fig. 1. Theoretical speed-up vs. No. of processors for different system dimensions n for $l = 10$, $d_p = 2$, $d_a = 3$ and $d = 4$, where $L + M = 53625$

VI. EXAMPLES AND RESULTS

We evaluate the scalability and accuracy of the proposed parallel set-up algorithm and SDP solver in the following examples.

Example 1: A simplified model for the poloidal magnetic flux gradient in a Tokamak reactor [38] is

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

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 $(0,1)$. The state-space model is

$$\dot{\psi}_x(t) = A(\eta(x))\psi_x(t), \tag{54}$$

where $A(\eta(x)) \in \mathbb{R}^{N\times N}$ has the following non-zero entries.

$$a_{11} = \frac{-4}{3\mu_0\Delta x^2 a^2}\left(\frac{\eta(x_{\frac{3}{2}})}{x_{\frac{3}{2}}} + \frac{2\eta(x_{\frac{3}{4}})}{x_{\frac{3}{4}}}\right) \quad , \quad a_{12} = \frac{4}{3\mu_0\Delta x^2 a^2}\left(\frac{\eta(x_{\frac{3}{2}})x_2}{x_{\frac{3}{2}}}\right)$$

$$a_{j,j-1} = \frac{1}{\Delta x^2\mu_0 a^2}\left(\frac{\eta(x_{j-\frac{1}{2}})}{x_{j-\frac{1}{2}}}x_{j-1}\right) \qquad \text{for } j = 2,\cdots,N-1$$

$$a_{j,j} = \frac{-1}{\Delta x^2\mu_0 a^2}\left(\frac{\eta(x_{j+\frac{1}{2}})}{x_{j+\frac{1}{2}}} + \frac{\eta(x_{j-\frac{1}{2}})}{x_{j-\frac{1}{2}}}\right)x_j \quad \text{for } j = 2,\cdots,N-1$$

$$a_{j,j+1} = \frac{1}{\Delta x^2\mu_0 a^2}\left(\frac{\eta(x_{j+\frac{1}{2}})}{x_{j+\frac{1}{2}}}x_{j+1}\right) \qquad \text{for } j = 2,\cdots,N-1$$

$$a_{N,N-1} = \frac{4}{3\Delta x\mu_0 a^2}\frac{\eta(x_{N-\frac{1}{2}})x_{N-1}}{x_{N-\frac{1}{2}}\Delta x} \quad , \quad a_{N,N} = \frac{-4}{3\Delta x\mu_0 a^2}\left(\frac{2\eta(x_{N+\frac{1}{4}})x_N}{x_{N+\frac{1}{4}}\Delta x} + \frac{\eta(x_{N-\frac{1}{2}})x_N}{x_{N-\frac{1}{2}}\Delta x}\right),$$

where $\Delta x = \dfrac{1}{N}$ and $x_j := (j - \frac{1}{2})\Delta x$.

We discretize the model at $N = 7$ points. Typically $\eta(x_k)$ are not precisely known, so we substitute $\eta(x_k)$ in (54) with $\widehat{\eta}(x_k) + \alpha_j$, where $\widehat{\eta}(x_k)$ are the nominal values of $\eta(x_k)$ and α_j are the uncertain parameters. At $x_k = 0.036, 0.143, 0.286, 0.429, 0.571, 0.714, 0.857, 0.964$ corresponding to $k = 0.75, 1.5, 2.5, 3.5, 4.5, 5.5, 6.5, 7.25$, $\widehat{\eta}(x_k)$ is equal to $1.775 \cdot 10^{-8}, 2.703 \cdot 10^{-8}, 5.676 \cdot 10^{-8}, 1.182 \cdot 10^{-7}, 2.058 \cdot 10^{-7}, 3.655 \cdot 10^{-7}, 1.076 \cdot 10^{-6}, 8.419 \cdot 10^{-6}$. The uncertain system can be written as

$$\dot{\psi}_x(t) = A(\alpha)\psi_x(t). \tag{55}$$

The uncertain parameters α_j belong to S_ρ which is defined as

$$S_\rho := \{\alpha \in \mathbb{R}^8 : \sum_{i=1}^{8} \alpha_i = -6|\rho|, -|\rho| \leq \alpha_i \leq |\rho|\},$$

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

$$\begin{aligned} \max \quad & \rho \\ \text{s.t.} \quad & \text{system (55) is stable for all } \alpha \in S_\rho. \end{aligned} \tag{56}$$

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

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

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 $\dot{\psi}_x(t) = \left(\sum_{i=1}^{8} A_i \alpha'_i\right) \psi_x(t)$, where the A_i matrices are in Appendix B. We solve the optimization problem in (56) using bisection search on ρ. For each trial value of ρ, we use the proposed parallel SDP solver to solve the associated SDP obtained by the parallel set-up algorithm. The SDP problems have 224 constraints with the primal variable $X \in \mathbb{R}^{1092 \times 1092}$. The normalized optimal value of ρ with respect to $\eta(x_j)_{max}$ is found to be 0.0019. In this particular example, the optimal value of ρ does not change with the degrees of $P(\alpha)$ and Polya's exponent d.

The SDPs are constructed and solved on a parallel Linux-based cluster Cosmea of Argonne National Laboratory. Fig. 2 shows the algorithm speed-up vs. the number of processors. Note that solving this problem by SOSTOOLS [20] on the same machine is impossible due to the excessive amount of memory that SOS method requires and its centralized structure.

Example 2: In this example we investigate the effect of d_p as the degree of $P(\alpha)$ and d as the Polya's exponent on the accuracy of the algorithm. Consider the uncertain continuous-time system

$$\dot{x}(t) = \left(\sum_{i=1}^{4} A_i \alpha_i\right) x(t),$$

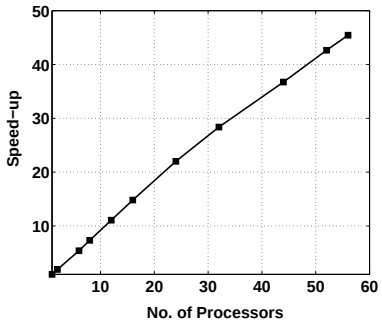

Fig. 2. Speed-up of the set-up and SDP algorithms vs. the number of processors

where

$$A_1 = \begin{bmatrix} -0.610 & -0.560 & 0.402 \\ -0.480 & -0.550 & 0.671 \\ -1.010 & -0.918 & 0.029 \end{bmatrix}, A_2 = \begin{bmatrix} -0.357 & 0.344 & -0.661 \\ -0.210 & -0.505 & 0.588 \\ 0.268 & 0.487 & -0.846 \end{bmatrix}$$

$$A_3 = \begin{bmatrix} -0.357 & 0.344 & -0.661 \\ -0.210 & -0.505 & 0.588 \\ 0.268 & 0.487 & -0.846 \end{bmatrix}, A_4 = \begin{bmatrix} -0.684 & -0.860 & -0.999 \\ -0.732 & -0.941 & -0.126 \\ 0.685 & 0.305 & -0.006 \end{bmatrix}$$

and α belong to S_L which is defined as

$$S_L := \{\alpha \in \mathbb{R}^4 : \sum_{i=1}^{4} \alpha_i = 3L + 1,\ L \leq \alpha_i \leq 1\}.$$

The problem is to solve

$$\begin{aligned} \min \quad & L \\ \text{s.t.} \quad & \dot{x}(t) = \left(\sum_{i=1}^{4} A_i' \alpha_i' \right) x(t) \text{ is stable for all } \alpha' \in \Delta, \end{aligned} \tag{57}$$

where α_i' are the new uncertain parameters in the unit simplex and are given by

$$\alpha_i' = \frac{\alpha_i - L}{1 - L}, \quad \text{for } i = 1, \cdots, 4. \tag{58}$$

In numerical experiments, we solved this problem by via bisection using several different polynomial values of d_p (degree of P) and the Polya exponent d. For different d_p and d, our upper bounds on the optimal L are shown in Fig. 3. Considering the optimal value of L to be $L_{\text{opt}} = -0.0044$ as taken from the calculations with high values of d_p and d, Fig. 3 shows how increasing d_p and/or d - when they are still relatively small - improves the accuracy of the algorithm. Fig. 4 demonstrates how the error in our upper bound for L_{opt} decreases by increasing d_p and/or d.

Example 3: To evaluate the scalability of the set-up algorithm, we run the algorithm on Blue Gene supercomputer of Argonne National Laboratory to obtain the SDP problems associated with three random linear systems with different state-space dimensions and numbers of uncertain parameters. Fig. 5 shows the computation time of the set-up algorithm with respect to the number of processors. The scalability of the algorithm is excellent in all cases.

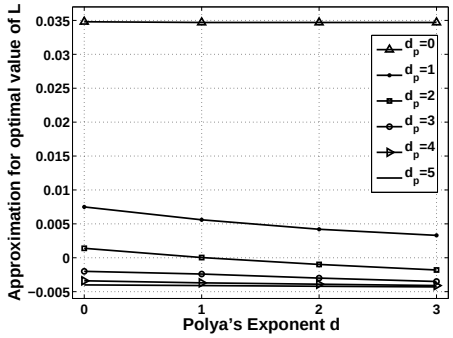

Fig. 3. Upper bound on the optimal L vs. the Polya's exponent, for different degrees of $P(\alpha)$

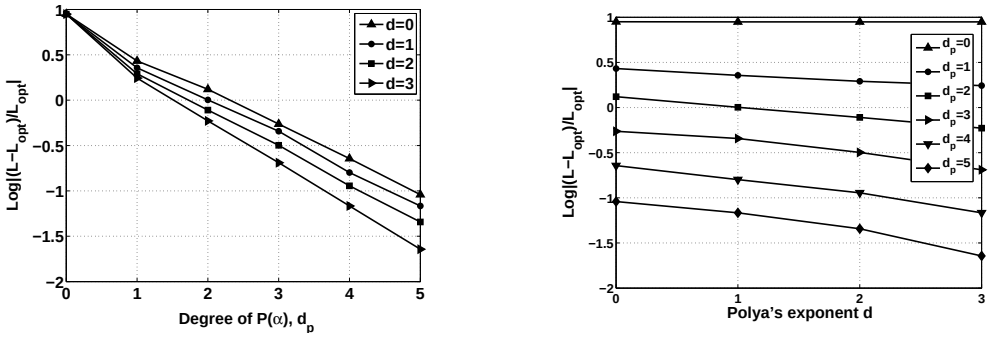

Fig. 4. Conservatism in the upper bound on the optimal L vs. the degrees of $P(\alpha)$, for different Polya's exponents (Left); Error of the approximation for the optimal value of L vs. Polya's exponent, for different degrees of $P(\alpha)$(Right)

To evaluate the scalability of the SDP algorithm, we solve three random SDP problems with different dimensions using the Karlin cluster. Fig. 6 demonstrates the computation time of the SDP algorithm with respect to the number of processors for SDP problems with different dimensions of the primal variable X and dual variable y. The dimensions of X are $(L+M)n = 200,\ 385$ and 1092, where L and M are defined in (15) and (16) and the dimensions of y are $K = 50,\ 90$ and 224, respectively. In all cases, $d_p = 2$ and $d = 1$. The linearity of the Time vs. Number of Processors curves in all three cases demonstrates the scalability of the SDP algorithm.

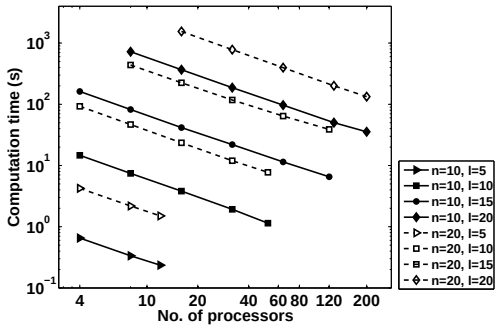

Fig. 5. Computation time of the parallel set-up algorithm vs. number of processors for different dimensions of linear system n and numbers of uncertain parameters l- executed on Blue Gene supercomputer of Argonne National Labratory

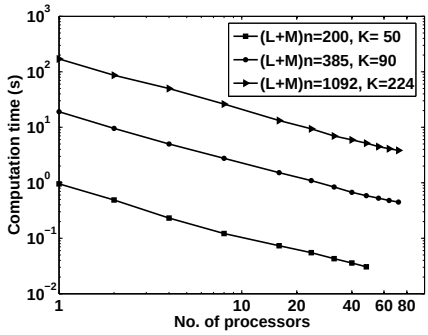

Fig. 6. Computation time of the parallel SDP algorithm vs. number of processors for different dimensions of primal variable $(L+M)n$ and of dual variable K- executed on Karlin cluster computer of Illinois Institute of Technology

Example 4: In this example, we ran the set-up and SDP algorithms to solve the robust stability problem with dimension n and l uncertain parameters on a computer with 24 Gig/node of RAM using one and nine nodes, respectively. The total memory access was thus 24 Gig and 216 Gig. Using trial and error, for different n and d we found the largest l for which the algorithms do not terminate due to insufficient memory (Fig. 7). In all of the runs $d_a = d_p = 1$.

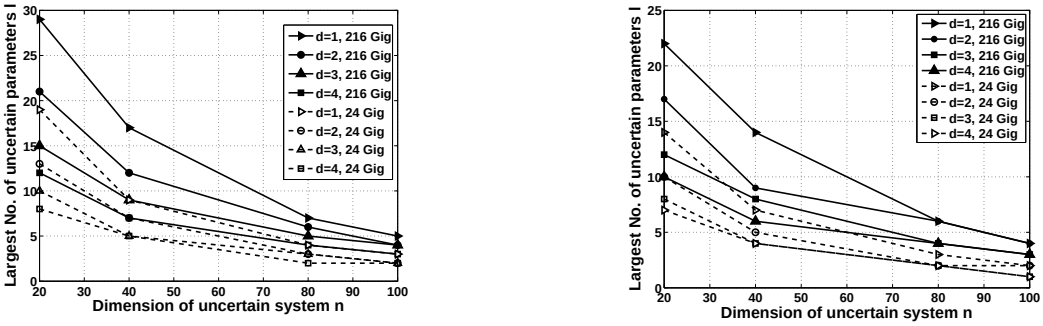

Fig. 7. Largest number of uncertain parameters of n-dimensional systems for which the set-up algorithm (Left) and SDP solver (Right) can solve the robust stability problem of the system using 24 and 216 Gig of RAM

VII. CONCLUSION

In this paper, we present a cluster-computing and supercomputing approach to solving the robust stability problems with parametric uncertainty for systems with large-dimensional state-space or many uncertain parameters. The approach is based on the structure of the LMI conditions associated with Polya's theorem. It is shown that in sufficiently large parallel environments, the proposed algorithm solves the intractable robust stability problem with the same per-core computation and communication cost as the tractable parameter-independent stability problem. The theoretical and experimental results for speed-up verify the scalability of the algorithms. Numerical examples demonstrate the ability to perform robust analysis of systems with 100+ states and several uncertain parameters. The algorithms can also be extended to nonlinear stability analysis and controller synthesis, which is left for future work.

ACKNOWLEDGMENTS

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

APPENDIX A
PARALLEL IMPLEMENTATION OF SET-UP ALGORITHM

Suppose N is the number of available processors, N_p is the number of monomials in $P(\alpha) \in \mathbb{R}^{n\times n}$, N_a is the number of monomials in $A(\alpha) \in \mathbb{R}^{n\times n}$, N_{pa} is the number of monomials in $P(\alpha)A(\alpha)$, $N_{p'}$ is initially the minimum number of monomials in $P(\alpha)$ assigned to each processor, $N_{a'}$ is initially the minimum number of monomials in $A(\alpha)$ assigned to each processor and $N_{(pa)'}$ is initially the minimum number of monomials in $P(\alpha)A(\alpha)$ assigned to each processor. Recall $\alpha \in \mathbb{R}^l$ is the vector of uncertain parameters in system (1). The indicator function $\mathbf{1}(x) : \mathbb{Z} \to \{0, 1\}$ is defined as

$$\mathbf{1}(x) = \begin{cases} 0, & \text{for } x = 0 \\ 1, & \text{for } x \neq 0. \end{cases}$$

The number of monomials in a homogeneous polynomial is

$$f(m, d) = \begin{cases} 0 & \text{for } m = 0 \\ \dfrac{(d+m-1)!}{d!(m-1)!} & \text{for } m > 0, \end{cases}$$

where d is the degree and m is the number of variables in the polynomial. We use the following functions in the algorithm.

$$f_1(x_1, x_2, R) = \texttt{floor}(\frac{x_1}{x_2+1}) + \mathbf{1}(R) - 1,$$

where R is the remainder of dividing x_1 by $x_2 + 1$.

$$f_2(x_1, x_2, x_3, x_4, R_1) = x_1 + \texttt{floor}(\frac{x_2 - x_3 - x_4 - x_1x_3}{x_3}) + \mathbf{1}(R_1)$$

where R_1 is the remainder of dividing $x_2 - x_3 - x_4 - x_1x_3$ by x_3.

$$f_3(x_1, x_2, x_3, x_4, R_2, R_3) = \begin{cases} x_1 + \texttt{floor}(\dfrac{x_2 - (x_3+1)x_1}{x_3}) + \mathbf{1}(R_2) - 1 & \text{if } x_2 > (x_3+1)x_1 \\ x_4 + \texttt{floor}(\dfrac{x_2 - (x_3+1)(x_4+1)}{x_3+1}) + \mathbf{1}(R_3) & \text{if } x_2 \leq (x_3+1)x_1 \end{cases}$$

where R_2 is the remainder of dividing $x_2 - (x_3+1)x_1$ by x_3 and R_3 is the remainder of dividing $x_2 - (x_3+1)(x_4+1)$ by $x_3 + 1$.

$$f_4(x_1, x_2, x_3, R_2, R_4) = \begin{cases} x_1 + \texttt{floor}(\dfrac{x_2 - (x_3+1)x_1}{x_3}) + \mathbf{1}(R_2) - 1 & \text{if } x_2 > (x_3+1)x_1 \\ \texttt{floor}(\dfrac{x_2}{x_3+1}) + \mathbf{1}(R_4) - 1 & \text{if } x_2 \leq (x_3+1)x_1 \end{cases}$$

where R_4 is the remainder of dividing x_2 by x_3+1.

1) **Reading data and initialization:**

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

i) Reads the degree d_p of $P(\alpha)$, the degree d_a of $A(\alpha)$, the dimension of uncertain parameters, l, and the maximum number of iterations, d_{max}.

ii) Defines the following variables.

$$N_p = f(l,d_p), \quad N_a = f(l,d_a), \quad N_{pa} = f(l,d_p+d_a) \tag{59}$$

$$N_{p'_1} = \texttt{floor}(\frac{N_p}{N}), \quad N_{a'} = \texttt{floor}(\frac{N_a}{N}), \quad N_{(pa)'_1} = \texttt{floor}(\frac{N_{pa}}{N})$$

$$r_p = N_p - N_{p'_1}\,N, \quad r_a = N_a - N_{a'}\,N, \quad r_{pa} = N_{pa} - N_{(pa)'_1}\,N \tag{60}$$

iii) Sets $N_{p_0}=N_p$, $d_{p_0}=d_p$, $N_{p'_2}=0$, $N_{(pa)'_2}=0$ and $d=0$.

2) **Creating Z_{p_0}:**

i) For $i=0,\cdots,N-1$, Processor i creates $Z^i_{p_0}\in\mathbb{Z}^{N_{p_0}\times l}$ with entries $z^i_{p_{0j,k}}$, where $z^i_{p_{0j,k}}$ is the degree in variable k of j^{th} monomial of $P(\alpha)$ in lexicographical ordering.

Note: To initialize β coefficients, we use equation (11). In the following step of the algorithm, for $i=0,\cdots,N-1$, the rows of Z^i_p correspond to $h\in W_{d_p}$ and $\gamma\in W_{d_p}$, where the cardinality of W_{d_p} is N_{p_0}.

3) **Initializing β:**

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

- If $i\in\{0,\cdots,r_p-1\}$, processor i:

i) Initializes $Z^i_p\in\mathbb{Z}^{(N_{p'_1}+1)\times l}$ with entries $z^i_{p_{j,k}}$, where $z^i_{p_{j,k}}$ is the degree in variable k of $[(N_{p'_1}+1)\,i+j]^{th}$ monomial of $P(\alpha)$ in lexicographical ordering.

ii) Initializes $B^i\in\mathbb{Z}^{N_{p_0}\times(N_{p'_1}+1)}$ with entries $\beta^i_{k,j}$, where

$$\beta^i_{k,j} = \begin{cases} 1, & j=(N_{p'_1}+1)\,i+k \\ 0, & j\neq(N_{p'_1}+1)\,i+k \end{cases}$$

- If $i\in\{r_p,\cdots,N-1\}$, processor i:

i) Initializes $Z^i\in\mathbb{Z}^{N_{p'_1}\times l}$ with entries $z^i_{j,k}$, where $z^i_{j,k}$ is the degree in variable k of $[N_{p'_1}i+r_p+j]^{th}$ monomial of $P(\alpha)$ in lexicographical ordering.

ii) Initializes $B^i \in \mathbb{Z}^{N_{p_0} \times N_{p'_1}}$ with entries $\beta^i_{k,j}$, where

$$\beta^i_{k,j} = \begin{cases} 1, & j = N_{p'_1} i + r_p + k \\ 0, & j \neq N_{p'_1} i + r_p + k \end{cases}$$

Note: To initialize H coefficients, we use equation (13). In steps 4 and 5 of the algorithm, for $i = 0, \cdots, N-1$, the rows of Z^i_a correspond to $\delta \in W_{d_a}$, where the cardinality of W_{d_a} is N_a. For $i = 0, \cdots, N-1$, the rows of Z^i_{pa} correspond to $\gamma \in W_{d_p+d_a}$, where the cardinality of $W_{d_p+d_a}$ is N_{pa}.

4) **Creating Z_a and Reading $A(\alpha)$ coefficients:**

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

- If $i \in \{0, \cdots, r_a - 1\}$ processor i: Initializes $Z^i_a \in \mathbb{Z}^{(N_{a'}+1)\times l}$ with entries $z^i_{a_{j,k}}$, where $z^i_{a_{j,k}}$ is the degree in variable k of $[(N_{a'}+1)\, i + j]^{th}$ monomial of $A(\alpha)$ in lexicographical ordering; For $j = (N_{a'}+1)i + 1, \cdots, (N_{a'}+1)(i+1)$ reads $A_j \in \mathbb{R}^{n\times n}$, where A_j is the coefficient of the j^{th} monomial in $A(\alpha)$ in lexicographical ordering.
- If $i \in \{r_a, \cdots, N-1\}$, processor i Initializes $Z^i_a \in \mathbb{Z}^{N_{a'}\times l}$ with entries $z^i_{a_{j,k}}$, where $z^i_{a_{j,k}}$ is the degree in variable k of $[N_{a'} i + r_a + j]^{th}$ monomial of $A(\alpha)$ in lexicographical ordering; For $j = N_{a'} i + r_a + 1, \cdots, N_{a'} i + r_a + N_{a'}$, reads $A_j \in \mathbb{R}^{n\times n}$, where A_j is the coefficient of the j^{th} monomial in $A(\alpha)$ in lexicographical ordering.

5) **Creating Z_{pa} and initializing H:**

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

- If $i \in \{0, \cdots, r_{pa} - 1\}$, processor i:

i) Initializes $Z^i_{pa} \in \mathbb{Z}^{(N_{(pa)'_1}+1)\times l}$ with entries $z^i_{(pa)_{j,k}}$, where $z^i_{(pa)_{j,k}}$ is the degree in variable k of $[(N_{(pa)'_1}+1)\, i + j]^{th}$ monomial of $P(\alpha)A(\alpha)$ in lexicographical ordering.

ii) Sets $t = N_{(pa)'_1} + 1$, if $i \in \{0, \cdots, r_a - 1\}$; Otherwise sets $t = N_{(pa)'_1}$.

iii) For $j = 1, \cdots, t$ and for $k = 1, \cdots, N_{p_0}$, processor i initializes $H^i_{k,j} = \mathbf{0} \in \mathbb{R}^{n\times n}$ and:

- If the lexicographical index I of the monomial $\prod_{m=1}^{l} \alpha_m^{(z^i_{p_{0_{k,m}}} + z^i_{a_{r,m}})}$ is between $(N_{(pa)'_1}+1)i+1$ and $(N'_a+1)(i+1)$, then sets $H^i_{k,I} = H^i_{k,I} + A_{(N'_a+1)i+j}$, where for $m = 1, \cdots, N_a$, A_m is the coefficient of m^{th} monomial in $A(\alpha)$ in lexicographical ordering.
- If the lexicographical index I of the monomial $\prod_{m=1}^{l} \alpha_m^{(z^i_{p_{0_{k,m}}} + z^i_{a_{r,m}})}$ is less than $(N_{pa'}+1)i+1$, then sends I and $A_{(N'_a+1)i+j}$ to processor with index

$$J_1 = f_1(I, N_{(pa)'_1}, R).$$

Processor J_1 receives I and $A_{(N'_a+1)i+j}$ and updates $H^{J_1}_{k,I}$ by setting $H^{J_1}_{k,I} = H^{J_1}_{k,I} + A_{(N'_a+1)i+j}$.

- If the lexicographical index I of the monomial $\prod_{m=1}^{l} \alpha_m^{(z^i_{p0_{k,m}}+z^i_{a_{r,m}})}$ is larger than $(N_{pa'}+1)(i+1)$, then sends I and $A_{(N'_a+1)i+j}$ to processor with index
$$J_1 = f_3(r_{pa}, I, N_{(pa)'_1}, i, R_2, R_3).$$
Processor J_1 receives I and $A_{(N'_a+1)i+j}$ and updates $H^{J_1}_{k,I}$ by setting $H^{J_1}_{k,I} = H^{J_1}_{k,I} + A_{(N'_a+1)i+j}$.

- If $i \in \{r_{pa}, \cdots, N-1\}$, processor i:

i) Initializes $Z^i_{pa} \in \mathbb{Z}^{N_{(pa)'_1} \times l}$ with entries $z^i_{(pa)_{j,k}}$, where $z^i_{(pa)_{j,k}}$ is the degree in variable k of $[N_{(pa)'_1} i + r_{pa} + j]^{th}$ monomial of $P(\alpha)A(\alpha)$ in lexicographical ordering.

ii) Sets $t = N_{(pa)'_1} + 1$, if $i \in \{0, \cdots, r_a - 1\}$; Otherwise sets $t = N_{(pa)'_1}$.

iii) For $j = 1, \cdots, t$ and for $k = 1, \cdots, N_{p_0}$, processor i initializes $H^i_{k,j} = \mathbf{0} \in \mathbb{R}^{n \times n}$ and:

- If the lexicographical index I of the monomial $\prod_{m=1}^{l} \alpha_m^{(z^i_{p0_{k,m}}+z^i_{a_{r,m}})}$ is between $N_{(pa)'_1} i + r_{pa} + 1$ and $N_{(pa)'_1} i + r_{pa} + N_{(pa)'_1}$, then sets $H^i_{k,I} = H^i_{k,I} + A_{N_{(pa)'_1} i + r_{pa} + j}$, where for $m = 1, \cdots, N_a$, A_m is the coefficient of m^{th} monomial in $A(\alpha)$ in lexicographical ordering.

- If the lexicographical index I of the monomial $\prod_{m=1}^{l} \alpha_m^{(z^i_{p0_{k,m}}+z^i_{a_{r,m}})}$ is less than $N_{(pa)'_1} i + r_{pa} + 1$, then sends I and $A_{N_{(pa)'_1} i + r_{pa} + j}$ to processor with index
$$J_2 = f_4(r_{pa}, I, N_{(pa)'_1}, R_2, R_4).$$
Processor J_2 receives I and $A_{N_{(pa)'_1} i + r_{pa} + j}$ and updates $H^{J_2}_{k,I}$ by setting $H^{J_2}_{k,I} = H^{J_2}_{k,I} + A_{N_{(pa)'_1} i + r_{pa} + j}$.

- If the lexicographical index I of the monomial $\prod_{m=1}^{l} \alpha_m^{(z^i_{p0_{k,m}}+z^i_{a_{r,m}})}$ is larger than $N_{(pa)'_1} i + r_{pa} + j$, then sends I and $A_{N_{(pa)'_1} i + r_{pa} + j}$ to processor with index
$$J_2 = f_2(i, I, N_{(pa)'_1}, r_{pa}, R_1).$$
Processor J_2 receives I and $A_{N_{(pa)'_1} i + r_{pa} + j}$ and updates $H^{J_2}_{k,I}$ by setting $H^{J_2}_{k,I} = H^{J_2}_{k,I} + A_{N_{(pa)'_1} i + r_{pa} + j}$.

Note: To update β and H coefficients we use equations (12) and (14). In steps 6 to 10 of the algorithm, for $i = 0, \cdots, N-1$, the rows of Z^i_p correspond to $\gamma \in W_{d_p+k}$ in (12), $k = 1, \ldots, d$, where the cardinality of W_{d_p+k} is N_p. For $i = 0, \cdots, N-1$, the rows of

Z^i_{pa} correspond to $\gamma \in W_{d_p+d_a+k}$ in (14), $k = 1, \ldots, d$, where the cardinality of $W_{d_p+d_a+k}$ is N_{pa}.

6) **Computing E:**

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

- If $i \in \{0, \cdots r_p - 1\}$, processor i computes $E^i \in \mathbb{Z}^{(N_{p'_1}+1)\times l}$ with entries $e^i_{j,k}$, where $e^i_{j,k}$ is the lexicographical order of the monomial corresponding to the j^{th} row of Z^i_p multiplied by α_k; i.e., $\alpha_k . \prod_{m=1}^{l} \alpha_m^{z^i_{p_{j,m}}}$.
- If $i \in \{r_p, \cdots N-1\}$, processor i computes $E^i \in \mathbb{Z}^{N_{p'_1}\times l}$ with entries $e^i_{j,k}$, where $e^i_{j,k}$ is the lexicographical order of the monomial corresponding to the j^{th} row of Z^i_p multiplied by α_k; i.e., $\alpha_k . \prod_{m=1}^{l} \alpha_m^{z^i_{p_{j,m}}}$

7) **Computing F:**

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

- If $i \in \{0, \cdots r_{pa} - 1\}$, processor i computes $F^i \in \mathbb{Z}^{(N_{(pa)'_1}+1)\times l}$ with entries $f^i_{j,k}$, where $f^i_{j,k}$ is the lexicographical order of the monomial corresponding to the j^{th} row of Z^i_{pa} multiplied by α_k; i.e., $\alpha_k . \prod_{m=1}^{l} \alpha_m^{z^i_{(pa)_{j,m}}}$.
- If $i \in \{r_{pa}, \cdots N-1\}$, processor i computes $F^i \in \mathbb{Z}^{N_{(pa)'_1}\times l}$ with entries $f^i_{j,k}$, where $f^i_{j,k}$ is the lexicographical order of the monomial corresponding to the j^{th} row of Z^i_{pa} multiplied by α_k; i.e., $\alpha_k . \prod_{m=1}^{l} \alpha_m^{z^i_{(pa)_{j,m}}}$.

8) **Updating parameters:**

For $i = 0, ..., N-1$, processor i:

i) Sets $d = d + 1$, $d_p = d_p + 1$ and $d_{pa} = d_{pa} + 1$.

ii) Updates N_p using (59), sets $N_{p'_2} = \texttt{floor}(\frac{N_p}{N})$ and updates r_p using (60).

iii) Updates N_{pa} using (59), sets $N_{(pa)'_2} = \texttt{floor}(\frac{N_{pa}}{N})$ and updates r_{pa} using (60).

9) **Creating β coefficients:**

For $i = 0, ..., N-1$:

- If $i \in \{0, \cdots r_p - 1\}$, processor i:

i) Updates $Z^i_p \in \mathbb{Z}^{(N_{p'_2}+1)\times l}$, where the entry $z^i_{p_{j,k}}$ is the degree in variable k of $[(N_{p'_2} + 1)i + j]^{th}$ monomial of $(\sum_{i=1}^{l} \alpha_i)^d P(\alpha)$.

ii) For $j = 1, \cdots, N_{p'_1} + 1$ and $k = 1, \cdots, l$, sets $I = e^i_{j,k}$ and:

- If $(N_{p_2'}+1)i+1 \leq I \leq (N_{p_2'}+1)(i+1)$, then sets $\beta^i_{m,I} = \beta^i_{m,I}+\beta^i_{m,j}$ for $m = 1, \cdots, N_{p_0}$.
- If $I < (N_{p_2'}+1)i+1$, then sends I and $\beta^i_{1,j}, \cdots, \beta^i_{N_{p_0},j}$ to processor with index
$$J_1 = f_1(I, N_{p_2'}, R).$$
Processor J_1 receives I and $\beta^i_{1,j}, \cdots, \beta^i_{N_{p_0},j}$, and updates $B^{J_1} \in \mathbb{Z}^{N_{p_0}\times(N_{p_1'}+1)}$ by setting $\beta^{J_1}_{m,I} = \beta^{J_1}_{m,I} + \beta^i_{m,j}$ for $m = 1, \cdots, N_{p_0}$.
- If $I > (N_{p_2'}+1)(i+1)$, then sends I and $\beta^i_{1,j}, \cdots, \beta^i_{N_{p_0},j}$ to processor with index
$$J_1 = f_3(r_p, I, N_{p_2'}, i, R_2, R_3).$$
Processor J_1 receives I and $\beta^i_{1,j}, \cdots, \beta^i_{N_{p_0},j}$, and updates $B^{J_1} \in \mathbb{Z}^{N_{p_0}\times(N_{p_1'}+1)}$ by setting $\beta^{J_1}_{m,I} = \beta^{J_1}_{m,I} + \beta^i_{m,j}$ for $m = 1, \cdots, N_{p_0}$.

- If $i \in \{r_p, \cdots, N-1\}$, processor i:

i) Updates $Z^i_p \in \mathbb{Z}^{N_{p_2'}\times l}$, where the entry $z^i_{p_{j,k}}$ is the degree in variable k of $[N_{p_2'}i+r_p+j]^{th}$ monomial of $(\sum_{i=1}^{l}\alpha_i)^d P(\alpha)$.

ii) For $j = 1, \cdots, N_{p_1'}$ and for $k = 1, \cdots, l$, sets $I = e^i_{j,k}$ and:

- If $N_{p_2'}i+r_p+1 \leq I \leq N_{p_2'}i+r_p+N_{p_2'}$, then sets $\beta^i_{m,I} = \beta^i_{m,I}+\beta^i_{m,j}$ for $m = 1, \cdots, N_{p_0}$.
- If $I < N_{p_2'}i+r_p+1$, then sends I and $\beta^i_{1,j}, \cdots, \beta^i_{N_{p_0},j}$ to processor with index
$$J_2 = f_4(r_p, I, N_{p_2'}, R_2, R_4).$$
Processor J_2 receives I and $\beta^i_{1,j}, \cdots, \beta^i_{N_{p_0},j}$, and updates $B^{J_2} \in \mathbb{Z}^{N_{p_0}\times N_{p_1'}}$ by setting $\beta^{J_2}_{m,I} = \beta^{J_2}_{m,I} + \beta^i_{m,j}$ for $m = 1, \cdots, N_{p_0}$.
- If $I > N_{p_2'}i+r_p+N_{p_2'}$, then sends I and $\beta^i_{1,j}, \cdots, \beta^i_{N_{p_0},j}$ to processor with index
$$J_2 = f_2(i, I, N_{p_2'}, r_p, R_1).$$
Processor J_2 receives I and $\beta^i_{1,j}, \cdots, \beta^i_{N_{p_0},j}$, and updates $B^{J_2} \in \mathbb{Z}^{N_{p_0}\times N_{p_1'}}$ by setting $\beta^{J_2}_{m,I} = \beta^{J_2}_{m,I} + \beta^i_{m,j}$ for $m = 1, \cdots, N_{p_0}$.

10) **Creating H coefficients:**

For $i = 0, ..., N-1$:

- If $i \in \{0, \cdots r_{pa}-1\}$, processor i:

i) Updates $Z^i_{pa} \in \mathbb{Z}^{(N_{(pa)_2'}+1)\times l}$, where the entry $z^i_{(pa)_{j,k}}$ is the degree in variable k of $[(N_{(pa)_2'}+1)i+j]^{th}$ monomial of $(\sum_{i=1}^{l}\alpha_i)^d P(\alpha)A(\alpha)$.

ii) Sets $t = N_{(pa)_1'}+1$, if $i \in \{0, \cdots, r_a-1\}$; Otherwise sets $t = N_{(pa)_1'}$.

iii) For $j = 1, \cdots, t$ and $k = 1, \cdots, l$, sets $I = f^i_{j,k}$ and:

- If $(N_{(pa)'_2}+1)i+1 \leq I \leq (N_{(pa)'_2}+1)(i+1)$, then sets $H^i_{m,I} = H^i_{m,I} + H^i_{m,j}$ for $m = 1, \cdots, N_{p_0}$.
- If $I < (N_{(pa)'_2}+1)i+1$, then send I and $H^i_{1,j}, \cdots, H^i_{N_{p_0},j}$ to processor with index
$$J_1 = f_1(I, N_{(pa)'_2}, R).$$
Processor J_1 receives I and $H^i_{1,j}, \cdots, H^i_{N_{p_0},j}$, and updates $H^{J_1}_{m,I}$ by setting $H^{J_1}_{m,I} = H^{J_1}_{m,I} + H^i_{m,j}$ for $m = 1, \cdots, N_{p_0}$.
- If $I > (N_{(pa)'_2}+1)(i+1)$, then sends I and $H^i_{1,j}, \cdots, H^i_{N_{p_0},j}$ to processor with index
$$J_1 = f_3(r_{pa}, I, N_{(pa)'_2}, i, R_2, R_3).$$
Processor J_1 receives I and $H^i_{1,j}, \cdots, H^i_{N_{p_0},j}$, and updates $H^{J_1}_{m,I}$ by setting $H^{J_1}_{m,I} = H^{J_1}_{m,I} + H^i_{m,j}$ for $m = 1, \cdots, N_{p_0}$.

- If $i \in \{r_{pa}, \cdots, N-1\}$, processor i:

i) Updates $Z^i_{pa} \in \mathbb{Z}^{N_{(pa)'_2} \times l}$, where the entry $z^i_{(pa)_{j,k}}$ is the degree in variable k of $[N_{(pa)'_2} i + r_{pa} + j]^{th}$ monomial of $(\sum_{i=1}^{l} \alpha_i)^d P(\alpha) A(\alpha)$.

ii) Sets $t = N_{(pa)'_1} + 1$, if $i \in \{0, \cdots, r_a - 1\}$; Otherwise sets $t = N_{(pa)'_1}$.

iii) For $j = 1, \cdots, t$ and for $k = 1, \cdots, l$, sets $I = f^i_{j,k}$ and:

- If $N_{(pa)'_2} i + r_{pa} + 1 \leq I \leq N_{(pa)'_2} i + r_{pa} + N_{(pa)'_2}$, then sets $H^i_{m,I} = H^i_{m,I} + H^i_{m,j}$, for $m = 1, \cdots, N_{p_0}$.
- If $I < N_{(pa)'_2} i + r_{pa} + 1$, then sends I and $H^i_{1,j}, \cdots, H^i_{N_{p_0},j}$ to processor with index
$$J_2 = f_4(r_{pa}, I, N_{(pa)'_2}, R_2, R_4).$$
Processor J_2 receives I and $H^i_{1,j}, \cdots, H^i_{N_{p_0},j}$, and updates $H^{J_2}_{m,I}$ by setting $H^{J_2}_{m,I} = H^{J_2}_{m,I} + H^i_{m,j}$, for $m = 1, \cdots, N_{p_0}$.
- If $I > N_{(pa)'_2} i + r_{pa} + N_{(pa)'_2}$, then sends I and $H^i_{1,j}, \cdots, H^i_{N_{p_0},j}$ to processor with index
$$J_2 = f_2(i, I, N_{(pa)'_2}, r_{pa}, R_1)$$
Processor J_2 receives I and $H^i_{1,j}, \cdots, H^i_{N_{p_0},j}$, and updates $H^{J_2}_{m,I}$ by setting $H^{J_2}_{m,I} = H^{J_2}_{m,I} + H^i_{m,j}$ for $m = 1, \cdots, N_{p_0}$.

11) **Updating parameters:**

For $i = 0, ..., N-1$, processor i sets $N_{p'_1} = N_{p'_2}$ and $N_{(pa)'_1} = N_{(pa)'_2}$.

12) **Creating the SDP elements:**

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

- If $i \in \{0, \cdots r_p - 1\}$, then processor i sets $t = N_{p'_2} + 1$; Otherwise sets $t = N_{p'_2}$.

- For $j = 1, \cdots, t$ and $k = 1, \cdots, K$ processor i computes

$$C_j^i = \delta I_n \cdot \left(\sum_{m=1}^{N_{p_0}} \beta_{m,j}^i \frac{d_{p_0}!}{\prod_{n=1}^{l} z_{p_{0m,n}}!} \right) \quad \text{and} \quad B_{k,j}^i = \sum_{m=1}^{N_{p_0}} \beta_{m,j}^i V_m(e_k),$$

where e_k is the canonical basis for $\mathbb{R}^K$ and $V_m(x) = \sum_{r=1}^{\tilde{N}} E_r x_{r+\tilde{N}(m-1)}$, where E_r is standard basis of $\mathbb{S}_n$, n is the dimension of system (1) and $\tilde{N} = \frac{n(n+1)}{2}$.

- If $i \in \{0, \cdots r_{pa} - 1\}$, then processor i sets $s = N_{(pa)_2'} + 1$; Otherwise sets $s = N_{(pa)_2'}$.
- For $j = 1, \cdots, s$ and $k = 1, \cdots, K$ processor i computes

$$\widehat{C}_j^i = 0_n \quad \text{and} \quad \widehat{B}_{k,j}^i = -\sum_{m=1}^{N_{p_0}} {H_{m,j}^i}^T V_m(e_k) + V_m(e_k) H_{m,j}^i.$$

- Processor i constructs $\overline{\mathbf{C}}_i$ and $\overline{\mathbf{B}}_{k,i}$ for $k = 1, \cdots, K$ as follows.

$$\overline{\mathbf{C}}_i = \text{diag}\{C_1^i, \cdots, C_t^i, \widehat{C}_1^i, \cdots, \widehat{C}_s^i\} \tag{61}$$

$$\overline{\mathbf{B}}_{k,i} = \text{diag}\{B_{k,1}^i, \cdots, B_{k,t}^i, \widehat{B}_{k,1}^i, \cdots, \widehat{B}_{k,s}^i\} \tag{62}$$

ii) If $d < d_{max}$, then go to step 6; otherwise start the SDP algorithm in section V-C, while for $i = 0, \cdots, N-1$, processor i has access to $\overline{\mathbf{C}}_i$ and $\overline{\mathbf{B}}_{k,i}$ for $k = 1, \cdots, K$.

APPENDIX B

A_i MATRICES IN EXAMPLE 1

$$A_1 = \begin{bmatrix} -14.09 & 6.47 & 0 & 0 & 0 & 0 & 0 \\ 0.41 & -3.54 & 3.83 & 0 & 0 & 0 & 0 \\ 0 & 2.30 & -10.24 & 8.97 & 0 & 0 & 0 \\ 0 & 0 & 6.41 & -21.46 & 16.06 & 0 & 0 \\ 0 & 0 & 0 & 12.49 & -39.71 & 28.91 & 0 \\ 0 & 0 & 0 & 0 & 23.66 & -101.96 & 86.33 \\ 0 & 0 & 0 & 0 & 0 & 97.40 & -1.74e3 \end{bmatrix} \quad A_2 = \begin{bmatrix} -2.86 & 1.66 & 0 & 0 & 0 & 0 & 0 \\ 1.62 & -7.15 & 3.83 & 0 & 0 & 0 & 0 \\ 0 & 2.30 & -10.24 & 8.97 & 0 & 0 & 0 \\ 0 & 0 & 6.41 & -21.46 & 16.06 & 0 & 0 \\ 0 & 0 & 0 & 12.49 & -39.72 & 28.91 & 0 \\ 0 & 0 & 0 & 0 & 23.66 & -101.96 & 86.33 \\ 0 & 0 & 0 & 0 & 0 & 97.40 & -1.74e3 \end{bmatrix}$$

$$A_3 = \begin{bmatrix} -1.25 & 6.47 & 0 & 0 & 0 & 0 & 0 \\ 0.41 & -5.35 & 6.84 & 0 & 0 & 0 & 0 \\ 0 & 4.10 & -13.25 & 8.97 & 0 & 0 & 0 \\ 0 & 0 & 6.41 & -21.46 & 16.06 & 0 & 0 \\ 0 & 0 & 0 & 12.49 & -39.71 & 28.91 & 0 \\ 0 & 0 & 0 & 0 & 23.66 & -101.96 & 86.33 \\ 0 & 0 & 0 & 0 & 0 & 97.40 & -1.74e3 \end{bmatrix} \quad A_4 = \begin{bmatrix} -1.25 & 6.47 & 0 & 0 & 0 & 0 & 0 \\ 0.41 & -3.54 & 3.83 & 0 & 0 & 0 & 0 \\ 0 & 2.30 & -12.25 & 11.77 & 0 & 0 & 0 \\ 0 & 0 & 8.41 & -24.27 & 16.06 & 0 & 0 \\ 0 & 0 & 0 & 12.49 & -39.72 & 28.91 & 0 \\ 0 & 0 & 0 & 0 & 23.66 & -101.96 & 86.33 \\ 0 & 0 & 0 & 0 & 0 & 97.40 & -1.74e3 \end{bmatrix}$$

$$A_5 = \begin{bmatrix} -1.25 & 6.47 & 0 & 0 & 0 & 0 & 0 \\ 0.41 & -3.54 & 3.83 & 0 & 0 & 0 & 0 \\ 0 & 2.30 & -10.24 & 8.97 & 0 & 0 & 0 \\ 0 & 0 & 6.41 & -23.57 & 18.76 & 0 & 0 \\ 0 & 0 & 0 & 14.59 & -42.42 & 28.91 & 0 \\ 0 & 0 & 0 & 0 & 23.66 & -101.96 & 86.33 \\ 0 & 0 & 0 & 0 & 0 & 97.40 & -1.74e3 \end{bmatrix} \quad A_6 = \begin{bmatrix} -1.25 & 6.47 & 0 & 0 & 0 & 0 & 0 \\ 0.41 & -3.54 & 3.83 & 0 & 0 & 0 & 0 \\ 0 & 2.30 & -10.24 & 8.97 & 0 & 0 & 0 \\ 0 & 0 & 6.41 & -21.46 & 16.06 & 0 & 0 \\ 0 & 0 & 0 & 12.49 & -41.88 & 31.56 & 0 \\ 0 & 0 & 0 & 0 & 25.82 & -104.61 & 86.33 \\ 0 & 0 & 0 & 0 & 0 & 97.40 & -1.74e3 \end{bmatrix}$$

$$A_7 = \begin{bmatrix} -1.25 & 6.47 & 0 & 0 & 0 & 0 & 0 \\ 0.41 & -3.54 & 3.83 & 0 & 0 & 0 & 0 \\ 0 & 2.30 & -10.24 & 8.97 & 0 & 0 & 0 \\ 0 & 0 & 6.41 & -21.46 & 16.06 & 0 & 0 \\ 0 & 0 & 0 & 12.49 & -39.71 & 28.91 & 0 \\ 0 & 0 & 0 & 0 & 23.66 & -104.17 & 88.94 \\ 0 & 0 & 0 & 0 & 0 & 100.34 & -1.74e3 \end{bmatrix} \quad A_8 = \begin{bmatrix} -1.25 & 6.47 & 0 & 0 & 0 & 0 & 0 \\ 0.41 & -3.54 & 3.83 & 0 & 0 & 0 & 0 \\ 0 & 2.30 & -10.24 & 8.97 & 0 & 0 & 0 \\ 0 & 0 & 6.41 & -21.46 & 16.06 & 0 & 0 \\ 0 & 0 & 0 & 12.49 & -39.71 & 28.91 & 0 \\ 0 & 0 & 0 & 0 & 23.66 & -101.96 & 86.33 \\ 0 & 0 & 0 & 0 & 0 & 97.40 & -1.74e3 \end{bmatrix}$$

REFERENCES

[1] J. Ackermann, A. Bartlett, D. Kaesbauer, W. Sienel, and R. Steinhauser, *Robust Control: Systems with Uncertain Physical Parameters*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2001.

[2] S. P. Bhattacharyya, H. Chpellat, and L. H. Keel, *Robust Control: The Parametric Approach*. Prentice Hall, 1995.

[3] M. Green and D. J. N. Limebeer, *Linear robust control*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1995.

[4] K. Zhou and J. Doyle, *Essentials of Robust Control*. Prentice Hall, 1998.

[5] R. Greenlaw, H. J. Hoover, and W. L. Ruzzo, "Limits to parallel computation: P-completeness theory," 1995.

[6] A. Packard and J. Doyle, "Quadratic stability with real and complex perturbations," *IEEE Transactions on Automatic Control*, vol. 35, pp. 198–201, Feb 1990.

[7] P. Gahinet, P. Apkarian, and M. Chilali, "Affine parameter-dependent lyapunov functions and real parametric uncertainty," *IEEE Transactions on Automatic Control*, vol. 41, pp. 436–442, Mar 1996.

[8] B. R. Barmish and C. L. DeMarco, "A new method for improvement of robustness bounds for linear state equations," in Proceedings Conf. Inform. Sci. Syst. Princeton University, 1986.

[9] P. Gahinet, P. Apkarian, and M. Chilali, "Affine parameter-dependent lyapunov functions and real parametric uncertainty," *IEEE Transactions on Automatic Control*, vol. 41, pp. 436–442, Mar 1996.

[10] 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.

[11] 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.

[12] D. Ramos and P. Peres, "An LMI approach to compute robust stability domains for uncertain linear systems," Proceedings of the American Control Conference, Jun 2001.

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

[14] P. A. Bliman, "An existence result for polynomial solutions of parameterdependent LMIs," *Systems & Control Letters*, no. 3-4, pp. 165–169, 2004.

[15] P. A. Bliman, "A convex approach to robust stability for linear systems with uncertain scalar parameters," *SIAM J. Control Optim*, vol. 42, no. 3-4, pp. 2016–2042, 2004.

[16] X. Zhang and P. Tsiotras, "Parameter-dependent lyapunov functions for stability analysis of LTI parameter dependent systems," pp. 5168–5173, in Proceedings of the IEEE 42nd Conference on Decision and Control, 2003.

[17] X. Zhang, P. Tsiotras, and P. A. Bliman, "Multi-parameter dependent lyapunov functions for the stability analysis of parameter-dependent LTI systems," pp. 1263–1268, in Proceedings of IEEE International Symposium on, Mediterrean Conference on Control and Automation, 2005.

[18] 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.

[19] G. Chesi, A. Garulli, A. Tesi, and A. Vicino, "Polynomially parameter-dependent lyapunov functions for robust stability of polytopic systems: an LMI approach," *IEEE Transactions on Automatic Control*, vol. 50, pp. 365–370, Mar 2005.

[20] 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.

[21] 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.

[22] 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.

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

[24] 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.

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

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

[27] 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.

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

[29] R. C. L. F. Oliveira, P.-A. Bliman, and P. L. D. Peres, "Robust LMIs with parameters in multi-simplex: Existence of solutions and applications," pp. 2226–2231, Proceedings of IEEE Conference on Decision and Control, 2008.

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

[31] 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.

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

[33] 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.

[34] 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.

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

[36] 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.

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

[38] 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.