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

Reza Kamyar, Matthew M. Peet and Yulia Peet

Abstract—In this paper, we propose a distributed computing approach to solving large-scale robust stability problems on the simplex. Our approach is to formulate the robust stability problem as an optimization problem with polynomial variables and polynomial inequality constraints. We use Polya’s theorem to convert the polynomial optimization problem to a set of highly structured Linear Matrix Inequalities (LMIs). We then use a slight modification of a common interior-point primal-dual algorithm to solve the structured LMI constraints. This yields a set of extremely large yet structured computations. We then map the structure of the computations to a decentralized computing environment consisting of independent processing nodes with a structured adjacency matrix. The result is an algorithm which can solve the robust stability problem with the same per-core complexity as the deterministic stability problem with versions which is only a function of the number of processors available. Numerical tests on cluster computers and supercomputers demonstrate the ability of the algorithm to efficiently utilize hundreds and potentially thousands of processors and analyze systems with 100+ dimensional state-space. The proposed algorithms can be extended to perform stability analysis of nonlinear systems and robust controller synthesis.

Index Terms—Robust stability, Polynomial optimization, Large-scale systems, Decentralized computing

I. INTRODUCTION

This paper addresses the problem of stability of large-scale systems with several unknown parameters. Control system theory when applied in practical situations often involves the use of large state-space models, typically due to inherent complexity of the system, the interconnection of subsystems, or the reduction of an infinite-dimensional or PDE model to a finite-dimensional approximation. One approach to dealing with such large scale models has been to use model reduction techniques such as balanced truncation [1]. However, the use of model reduction techniques are not necessarily robust and can result in arbitrarily large errors. In addition to large state-space, practical problems often contain uncertainty in the model due to modeling errors, linearization, or fluctuation in the operating conditions. The problem of uncertainty and with hundreds of states, existing algorithms will fail with the primary point of failure usually being lack of unallocated memory.

In this paper, we seek to distribute the computation laterally over an array of processors within the context of existing computational resources. Specifically, we seek to utilize cluster-computing, supercomputing and Graphics Processing Unit (GPU)-computing architectures. When designing algorithms to run in a parallel computing environment, one must both synchronize computational tasks among the processors while minimizing communication overhead among the processors. This can be difficult, as each architecture has a specific communication graph. We account for communication by explicitly modeling the required communication graph between processors. This communication graph is then mapped to the processor architecture using the Message-Passing Interface (MPI) [8]. While there are many algorithms for robust stability analysis and control of linear systems, ours is the first which explicitly accounts for the processing architecture in the emerging multi-core computing environment.

Our approach to robust stability is based on the well-established use of parameter-dependent Quadratic-In-The-State (QITS) Lyapunov functions. The use of parameter-dependent Lyapunov QITS functions eliminates the conservativity associated with e.g. quadratic stability [9], at the cost of requiring some restriction on the rate of parameter variation. Specifically, our QITS Lyapunov variables are polynomials in the vector of uncertain parameters. This is a generalization of the use of QITS Lyapunov functions with affine parameter dependence as in [10] and expanded in, e.g. [11], [12], [13], [14]. The use of polynomial QITS Lyapunov variables can be motivated by [15], wherein it is shown that any feasible parameter-dependent LMI with parameters inside a compact set has a polynomial solution or [16] wherein it is shown that local stability of a nonlinear vector field implies the existence of a polynomial Lyapunov function.

There are several results which use polynomial QITS Lyapunov functions to prove robust stability. In most cases, the stability problem is reduced to the general problem of optimization of polynomial variables subject to LMI constraints - an NP-hard problem [17]. To avoid NP-hardness, the polyno-
mial optimization problem is usually solved in an asymptotic manner by posing a sequence of sufficient conditions of increasing accuracy and decreasing conservatism. For example, building on the result in [13, 19] provides a sequence of increasingly precise LMI conditions for robust stability analysis of linear systems with affine dependency on uncertain parameters on the complex unit ball. Necessary and sufficient stability conditions for linear systems with one uncertain parameter are derived in [19], providing an explicit bound on the degree of the polynomial-type Lyapunov function. The result is extended to multi-parameter-dependent linear systems in [20]. Another important approach to optimization of polynomials is the Sum of Squares (SOS) methodology which replaces the polynomial positivity constraint with the constraint that the polynomial admits a representation as a sum of squares of polynomials [21], [22], [23], [24]. A version of this theorem for polynomials with matrix coefficients can be found in [23]. While we have worked extensively with the SOS methodology, we have not, as of yet, been able to adapt algorithms for solving the resulting LMI conditions to a parallel-computing environment. Finally, there have been several results in recent years on the use of Polya’s theorem to solve polynomial optimization problems [25] on the simplex. An extension of the Polya’s theorem for uncertain parameters on the multisimplex or hypercube can be found in [26]. The approach presented in this paper is an extension of the use of Polya’s theorem for solving polynomial optimization problems in a parallel computing environment.

The goal of this project is to create algorithms which explicitly map computation, communication and storage to existing parallel processing architectures. This goal is motivated by the failure of existing general-purpose Semi-Definite Programming (SDP) solvers to efficiently utilize platforms for large-scale computation. Specifically, it is well-established that linear programming and semi-definite programming both belong to the complexity class P-Complete, also known as the class of inherently sequential problems. Although there have been several attempts to map certain SDP solvers to a parallel computing environment [27], [28], certain critical steps cannot be distributed. The result is that as the number of processors increases, certain bottleneck computations dominate, leading to a saturation in computational speed of these solvers (Amdahl’s law [29]). We avoid these bottlenecks and communications by exploiting the particular structure of the LMI conditions associated with Polya’s theorem. Note that, in principle, a perfectly designed general-purpose SDP algorithm could identify the structure of the SDP, as we have, and map the communication, computation and memory constraints to the parallel architecture. Indeed, there has been a great deal of research on creating programming languages which attempt to do just this [30], [31]. However, at present such languages are mostly theoretical and have certainly not been incorporated into existing SDP solvers.

In addition to parallel SDP solvers, there have been some efforts to exploit structure in certain polynomial optimization algorithms to reducing the size and complexity of the resulting LMI’s. For example, in [32] symmetry was used to reduce the size of the SDP variables. Specific sparsity structure was used in [33], [34], [35] to reduce the complexity of the linear algebra calculations. Generalized approaches to the use of sparsity in SDP algorithms can be found in [34]. Groebner basis techniques [36], [37] have been used by [38] to simplify the formulation of the SDPs associated with the SOS decomposition problems.

The paper is organized around two independent problems: setting up the sequence of structured SDPs associated with Polya’s theorem and solving them. Note that the problem of decentralizing the set-up algorithm is significant in that for large-scale systems, the instantiation of the problem may be beyond the memory and computational capacity of a single processing node. For the set-up problem, the algorithm that we propose has no centralized memory or computational requirements whatsoever. Furthermore, if a sufficient number of processors are available, the number of messages does not change with the size of the state-space or the number of Polya’s iterations. In addition, the ideal communication architecture for the set-up algorithm does not correspond to the communication structure of GPU computing or supercomputing. In the second problem, we propose a variant of a standard SDP primal-dual algorithm and map the computational, memory and communication requirements to a parallel computing environment. Unlike the set-up algorithm, the primal-dual algorithm does have a small centralized component corresponding to the update of the set of dual variables. However, we have structured the algorithm so that the size of this dual computation is solely a function of the degree of the polynomial QITS Lyapunov function and does not depend on the number of Polya’s iterations, meaning that the sequence of algorithms has fixed centralized computational and communication complexity. In addition, there is no communication between processors, which means that the algorithm is well suited to most parallel computing architectures. A graph representation of the communication architecture of both the set-up and SDP algorithms has also been provided in the relevant sections.

Combining the set-up and SDP components and testing the result of both in cluster computing environments, we demonstrate the capability of robust analysis and control of systems with 100+ states and several uncertain parameters. Specifically, we ran a series of numerical experiments using a local Linux cluster and the Blue Gene supercomputer (with 200 processor allocation). First, we applied the algorithm to a current problem in robust stability analysis of magnetic confinement fusion using a discretized PDE model. Next, we examine the accuracy of the algorithm as Polya’s iterations progress and compare this accuracy with the SOS approach. We show that unlike the general-purpose parallel SDP solver SDPARA [28], the speed-up - the increase in processing speed per additional processor - of our algorithm shows no evidence of saturation. Finally, we calculate the envelope of the algorithm on the Linux cluster in terms of the maximum state-space dimension, number of processors and Polya’s iterations.

**NOTATION**

We represent $l$-variate monomials as $\alpha^\gamma = \prod_{i=1}^{l} \alpha_i^{\gamma_i}$, where $\alpha \in \mathbb{R}^l$ is the vector of variables and $\gamma \in \mathbb{N}^l$ is the
vector of exponents and \( \sum_{i=1}^{l} \gamma_i = d \) is the degree of the monomial. We define \( W_d := \{ \gamma \in \mathbb{N}^l : \sum_{i=1}^{l} \gamma_i = d \} \) as the totally ordered set of the exponents of \( l \)-variate monomials of degree \( d \), where the ordering is lexicographic. In lexicographical ordering \( \gamma \in W_d \) precedes \( \eta \in W_d \), if the left most non-zero entry of \( \gamma - \eta \) is positive. The lexicographical index of every \( \gamma \in W_d \) can be calculated using the map \( \langle \cdot \rangle : \mathbb{N}^l \rightarrow \mathbb{N} \) defined as

\[
\langle \gamma \rangle = \sum_{j=1}^{l-1} \sum_{i=1}^{j} (l-j, d+1-\sum_{k=1}^{j} \gamma_k-i) + 1,
\]

where as in

\[
f(l, d) := \begin{cases} 0 & \text{for } l = 0, \\ \frac{(l+d-1)!}{l!} & \text{for } l > 0,
\end{cases}
\]

is the cardinality of \( W_d \), i.e., the number of \( l \)-variate monomials of degree \( d \). For convenience, we also define the index of a monomial \( \alpha \gamma \) to be \( \langle \gamma \rangle \). We represent \( l \)-variate homogeneous polynomials of degree \( d_p \) as

\[
P(\alpha) = \sum_{\gamma \in W_{d_p}} P_\gamma \alpha^\gamma,
\]

where \( P_\gamma \in \mathbb{R}^{n \times n} \) is the matrix coefficient of the monomial \( \alpha^\gamma \). We denote the element corresponding to the \( j \)-th row and \( j \)-th column of matrix \( A \) as \( A_{i,j} \). The subspace of symmetric matrices in \( \mathbb{R}^{n \times n} \) is denoted by \( \mathbb{S}^n \). We define a basis for \( \mathbb{S}^n \) as

\[
[E_k]_{i,j} := \begin{cases} 1 & \text{if } i = j = k, \\ 0 & \text{otherwise, for } k \leq n \text{ and } \end{cases}
\]

\[
[E_k]_{i,j} := \begin{cases} 1 & \text{if } i = j = k = n, \\ 0 & \text{otherwise.}
\end{cases}
\]

Note that this choice of basis is arbitrary - any other basis could be used. However, any change in basis would require modifications to the formulae defined in this paper. The canonical basis for \( \mathbb{R}^n \) is denoted by \( e_i \) for \( i = 1, \cdots, n \), \( e_i \).

II. PRELIMINARIES

Consider the linear system

\[
\dot{x}(t) = A(\alpha)x(t),
\]

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

\[
\Delta_l = \left\{ \alpha \in \mathbb{R}^l, \sum_{i=1}^{l} \alpha_i = 1, \alpha_i \geq 0 \right\}.
\]

If \( A(\alpha) \) is not homogeneous, we can obtain an equivalent homogeneous representation in the following manner. Suppose \( A(\alpha) \) is a non-homogeneous polynomial with \( \alpha \in \Delta_l \), is of degree \( d_a \) and has \( N_a \) monomials with non-zero coefficients. Define

\[
D = \left( d_{a_1}, \cdots, d_{a_{N_a}} \right),
\]

where \( d_{a_i} \) is the degree of the \( i \)-th monomial of \( A(\alpha) \) according to lexicographical ordering.

Now define the polynomial \( B(\alpha) \) as per the following.

1) Let \( B = A \).

2) For \( i = 1, \cdots, N_a \), multiply the \( i \)-th monomial of \( B(\alpha) \), according to lexicographical ordering, by \( \left( \sum_{j=1}^{l} \alpha_j \right)^{d_{a_i} - d_{a_j}} \).

Then, since \( \sum_{j=1}^{l} \alpha_j = 1 \), \( B(\alpha) = A(\alpha) \) for all \( \alpha \in \Delta_l \) and hence all properties of \( \dot{x}(t) = A(\alpha)x(t) \) are retained by the homogeneous system \( \dot{x}(t) = B(\alpha)x(t) \).

1) Example: Construction of the homogeneous system \( \dot{x}(t) = B(\alpha)x(t) \).

Consider the non-homogeneous polynomial \( A(\alpha) = C\alpha^2 + D\alpha + E \alpha^3 + F \) of degree \( d_a = 2 \), where \( [\alpha_1, \alpha_2, \alpha_3] \in \Delta_3 \). Using the above procedure, the homogeneous polynomial \( B(\alpha) \) can be constructed as

\[
B(\alpha) = C\alpha^2 + D\alpha + E\alpha^3 + F \alpha^2 + (C + F) \alpha^2 + (D + 2F) \alpha \alpha^2 + (E + 2F) \alpha^3.
\]

The following is a stability condition [25] for System (6).

Theorem 1: System (6) is stable if and only if there exists a polynomial matrix \( P(\alpha) \) such that \( P(\alpha) > 0 \) and

\[
A^T(\alpha)P(\alpha) + P(\alpha)A(\alpha) \prec 0
\]

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 infinite number of points. Such problems are known to be NP-hard [17]. 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 [40]. A variation of this theorem for matrices is given as follows.

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 \( d \),

\[
\left( \sum_{i=1}^{l} \alpha_i \right)^d F(\alpha)
\]

has all positive definite coefficients.
Upper bounds for Polya’s exponent \( d \) can be found as in [31]. However, these bounds are based on the properties of \( F \) and are difficult to determine a priori. In this paper, we show that applying Polya’s Theorem to the robust stability problem, i.e., the inequalities in Theorem 1 yields a semi-definite programming condition with an efficiently distributable structure. This is discussed in the following section.

III. 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 distributable structure.

A. Polya’s Algorithm

We consider the stability of the system described by Equation 6. 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 \( d_1 \) and \( d_2 \), the polynomials

\[
\left( \sum_{i=1}^{l} \alpha_i \right)^{d_1} P(\alpha) \quad \text{and} \quad \left( \sum_{i=1}^{l} \alpha_i \right)^{d_2} (A^T(\alpha)P(\alpha) + P(\alpha)A(\alpha))
\]

have all positive definite coefficients.

Let \( P(\alpha) \) be a homogeneous polynomial of degree \( d_p \) which can be represented as

\[
P(\alpha) = \sum_{\gamma \in W_{d_p}} P(\gamma) \alpha^\gamma,
\]

where the coefficients \( P(\gamma) \in \mathbb{R}^{n \times n} \) and where we recall that \( W_{d_p} := \{ \gamma \in \mathbb{N}^n : \sum_{i=1}^n \gamma_i = d_p \} \) is the set of the exponents of all \( l \)-variate monomials of degree \( d_p \). Since \( A(\alpha) \) is a homogeneous polynomial of degree \( d_a \), we can write it as

\[
A(\alpha) = \sum_{\gamma \in W_{d_a}} A(\gamma) \alpha^\gamma,
\]

where the coefficients \( A(\gamma) \in \mathbb{R}^{n \times n} \). By substituting (13) and (14) into (11) and (12) and defining \( d_{pa} \) as the degree of \( P(\alpha)A(\alpha) \), the conditions of Theorem 1 can be represented in the form

\[
\sum_{h \in W_{d_p}} \beta_{(h),\gamma} P(h) > 0; \quad \gamma \in W_{d_p + d_a} \quad \text{and}
\]

\[
\sum_{h \in W_{d_p}} (H^T_{(h),\gamma}) P(h) + P(h) H_{(h),\gamma} < 0; \quad \gamma \in W_{dp + d_a + 2d_2}.
\]

Here \( \beta_{(h),\gamma} \) is defined to be the scalar coefficient which multiplies \( P(h) \) in the \( \gamma \)-th monomial of the homogeneous polynomial \( \left( \sum_{i=1}^l \alpha_i \right)^{d_1} P(\alpha) \) using the lexicographical ordering. Likewise \( H_{(h),\gamma} \) is the term which left or right multiplies \( P(h) \) in the \( \gamma \)-th monomial of \( \left( \sum_{i=1}^l \alpha_i \right)^{d_2} (A^T(\alpha)P(\alpha) + P(\alpha)A(\alpha)) \) using the lexicographical ordering. For an intuitive explanation as to how these \( \beta \) and \( H \) terms are calculated, we consider a simple example. Precise formulae for these terms will follow the example.

1) Example: Calculating the \( \beta \) and \( H \) coefficients.

Consider \( A(\alpha) = A_1 \alpha_1 + A_2 \alpha_2 \) and \( P(\alpha) = P_1 \alpha_1 + P_2 \alpha_2 \). By expanding Equation (11) for \( d_1 = 1 \) we have

\[
(\alpha_1 + \alpha_2) P(\alpha) = P_1 \alpha_1^2 + (P_1 + P_2) \alpha_1 \alpha_2 + P_2 \alpha_2^2.
\]

The \( \beta_{(h),\gamma} \) terms are then expressed as

\[
\beta_{(h),\gamma} = (1 + \alpha_2) P(\alpha). \quad (17)
\]

The \( H_{(h),\gamma} \) terms are then expressed as

\[
H_{(h),\gamma} = \begin{cases} 1 & \text{if } \gamma = h \in W_{d_p} \text{ and } h \in W_{d_a}, \\ 0 & \text{otherwise} \end{cases}
\]

(21)

Then, iterating for \( i = 1, \ldots, d_1 \), we let

\[
\beta_{(h),\gamma} = \sum_{\lambda \in W_{d_i}} \beta_{(h),\gamma} \delta_{i-1} \text{ for } \gamma \in W_{d_p + i} \text{ and } h \in W_{d_p}.
\]

(22)

Finally, we set \( \{ \beta_{(h),\gamma} \} = \{ \beta_{d_1} \} \). To obtain \( \{ H_{(h),\gamma} \} \), we set the initial values as

\[
H_{(h),\gamma} = \sum_{\lambda \in W_{d_a}} A(\lambda) \quad \text{for } \gamma \in W_{d_p + d_a} \text{ and } h \in W_{d_p}.
\]

(23)

Then, iterating for \( i = 1, \ldots, d_2 \), we let

\[
H_{(h),\gamma} = \sum_{\lambda \in W_{d_i}} H_{(h),\gamma} \delta_{i-1} \text{ for } \gamma \in W_{d_p + i} \text{ and } h \in W_{d_p}.
\]

(24)

Finally, set \( \{ H_{(h),\gamma} \} = \{ H_{d_1} \} \).

For the case of large-scale systems, computing and storing \( \{ \beta_{(h),\gamma} \} \) and \( \{ H_{(h),\gamma} \} \) is a significant challenge due to the number of these coefficients. Specifically, the number of terms increases with \( l \) (number of uncertain parameters in system (9)), \( d_p \) (degree of \( P(\alpha) \)), \( d_{pa} \) (degree of \( P(\alpha)A(\alpha) \)) and \( d_1, d_2 \) (Polya’s exponents) as follows.

3) Number of \( \beta_{(h),\gamma} \) coefficients: For given \( l, d_p \) and \( d_1 \), since \( h \in W_{d_p} \), and \( \gamma \in W_{d_p + d_1} \), the number of \( \beta_{(h),\gamma} \) coefficients is the product of \( L_0 := \text{card}(W_{d_p}) \) and \( L := \text{card}(W_{d_p + d_1}) \). Recall that \( \text{card}(W_{d_p}) \) is the number of all \( l \)-variate monomials of degree \( d_p \) and can be calculated using (2).
as follows.

\[ L_0 = f(l, d_p) = \begin{cases} 
0 & \text{for } l = 0 \\
\frac{(d_p + l - 1)!}{l!} & \text{for } l > 0.
\end{cases} \]

Likewise, \( \text{card}(W_{d_p+d_1}) \), i.e., the number of all \( l \)-variate monomials of degree \( d_p+d_1 \) is calculated using \( (2) \) as follows.

\[ L = f(l, d_p + d_1) = \begin{cases} 
0 & \text{for } l = 0 \\
\frac{(d_p + d_1 + l - 1)!}{(d_p + d_1)! l!} & \text{for } l > 0.
\end{cases} \]

The number of \( \beta_{(h),\langle\gamma\rangle} \) coefficients is \( L_0 \cdot L \).

4) Number of \( H_{(h),\langle\gamma\rangle} \) coefficients: For given \( l, d_p, d_a \) and \( d_2 \), since \( h \in W_{d_h} \) and \( \gamma \in W_{d_m+d_2} \), the number of \( H_{(h),\langle\gamma\rangle} \) coefficients is the product of \( L_0 := \text{card}(W_{d_p}) \) and \( M := \text{card}(W_{d_a+d_2}) \). By using \( (2) \), we have

\[ M = f(l, d_{pa} + d_2) = \begin{cases} 
0 & \text{for } l = 0 \\
\frac{(d_{pa} + d_2 + l - 1)!}{(d_{pa} + d_2)! l!} & \text{for } l > 0.
\end{cases} \]

The number of \( H_{(h),\langle\gamma\rangle} \) coefficients is \( L_0 \cdot M \).

The number of \( \beta_{(h),\langle\gamma\rangle} \) and \( H_{(h),\langle\gamma\rangle} \) coefficients and the required memory to store these coefficients are shown in Figs. 1 and 2 in terms of the number of uncertain parameters \( l \) and for different Polya’s exponents. In all cases \( d_p = d_a = 2 \).

It is observed from Fig. 2 that, even for small \( d_p \) and \( d_a \), the required memory is in the Terabyte range. In [18], we proposed a decentralized computing approach to the calculation of \( \{\beta_{(h),\langle\gamma\rangle}\} \) on large cluster computers. In the present work, we extend this method to the calculation of \( \{H_{(h),\langle\gamma\rangle}\} \) and the SDP elements which will be discussed in the following section. We express the LMIs associated with conditions \( (13) \) and \( (16) \) as an SDP in both primal and dual forms. We also discuss the structure of the primal and dual SDP variables and the constraints.
where $B^T : \mathbb{R}^K \to \mathbb{S}^m$ is the transpose operator and is given by

$$B^T(y) = \sum_{i=1}^{K} y_i B_i$$  \hspace{1cm} (31)

and where $y \in \mathbb{R}^K$ and $Z \in \mathbb{S}^m$ are the dual variables. The elements $C$, $B_i$, and $\alpha$ of the SDP problem associated with the LMIs in (15) and (16) are defined as follows. We define the element $C$ as

$$C := \text{diag}(C_1, \ldots, C_L, C_{L+1}, \ldots, C_{L+M}),$$  \hspace{1cm} (32)

where

$$C_i := \left\{ \begin{array}{ll} \delta_{n,i} \cdot \left( \sum_{h \in W_{d_p}} \beta(h,i) \frac{d_p}{n_1! \cdots n_l!} \right), & 1 \leq i \leq L \\ 0, & L + 1 \leq i \leq L + M \end{array} \right.$$  \hspace{1cm} (33)

with recall that $L = \text{card}(W_{d_p+d_s})$ is the number of monomials in $\left( \sum_{i=1}^{l} \alpha_i \right)^{d_p} P(\alpha)$, $M = \text{card}(W_{d_p+d_s}) + 1$, the number of uncertain parameters and $\delta$ is a small positive parameter.

For $i = 1, \ldots, K$, define $B_i$ elements as

$$B_i := \text{diag}(B_{i,1}, \ldots, B_{i,L}, B_{i,L+1}, \ldots, B_{i,L+M}),$$  \hspace{1cm} (34)

where $K$ is the number of dual variables in (30) and is equal to the product of the number of upper-triangular elements in each $P_\ell \in \mathbb{S}^n$ (the coefficients in $P(\alpha)$) and the number of coefficients in $P(\alpha)$ (i.e., the cardinality of $W_{d_p}$). Since there are $f(l, d_p) = \binom{d_p + l - 1}{l - 1}$ coefficients in $P(\alpha)$ and each coefficient has $N := \frac{n(n + 1)}{2}$ upper-triangular elements, we find

$$K = \frac{(d_p + l - 1)!}{d_p!(l - 1)!} N.$$  \hspace{1cm} (35)

To define the $B_{i,j}$ blocks, first we define the function $V_{(\ell)} : \mathbb{Z}^K \to \mathbb{Z}^{n \times n}$,

$$V_{(\ell)}(x) := \sum_{j=1}^{N} E_j x_{j+N(\ell(n+1)-1)} \quad \text{for all} \quad h \in W_{d_p},$$  \hspace{1cm} (36)

which maps each variable to a basis matrix $E_j$, where recall that $E_j$ is the basis for $\mathbb{S}^n$. Note that a different choice of basis would require a different function $V_{(\ell)}$. Then for $i = 1, \ldots, K$,

$$B_{i,j} := \left\{ \begin{array}{ll} \sum_{h \in W_{d_p}} \beta(h,j) V_{(\ell)}(e_i), & 1 \leq j \leq L (I) \\ - \sum_{h \in W_{d_p}} \left( H_{(h,j-L)} V_{(\ell)}(e_i) + V_{(\ell)}(e_i) H_{(h,j-L)} \right), & L + 1 \leq j \leq L + M. (II) \end{array} \right.$$  \hspace{1cm} (37)

Finally, to complete the SDP problem associated with Polya’s algorithm set

$$a = \bar{1} \in \mathbb{R}^K.$$  \hspace{1cm} (38)

C. Parallel Set-up Algorithm

In this section, we propose a decentralized, iterative algorithm for calculating the terms $\{\beta(h_{(\ell)}, \gamma)\}$, $\{H_{(h_{(\ell)}, \gamma)}\}$, $C$ and $B_i$ as defined in (22), (24), (22) and (34). The algorithm has been implemented in C++, using MPI (Message Passing Interface) and is available at: www.sites.google.com/a/asu.edu/kamyar/software. We present an abridged description of this algorithm in Algorithm 11, wherein $N$ is the number of available processors.

Note that we have only addressed the problem of robust stability analysis, using the polynomial inequality

$$P(\alpha) > 0, A^T(\alpha)P(\alpha) + P(\alpha)A(\alpha) \prec 0$$  \hspace{1cm} (39)

for $\alpha \in \Delta_l$. However, we can generalize the decentralized set-up algorithm to consider a more general class of feasibility problems, i.e.,

$$\sum_{i=1}^{N} \left( \hat{A}_i(\alpha) \hat{X}_i(\alpha) + \hat{B}_i(\alpha) \hat{X}_i(\alpha) A_i^T(\alpha) + R_i(\alpha) \right) < 0$$  \hspace{1cm} (40)

for $\alpha \in \Delta_l$. One motivation behind the development of such generalized set-up algorithm is that the parameter-dependent versions of the LMIs associated with $H_\infty$ and $H_\infty$ synthesis problems in (42), (43) can be formulated in the form of (40).

D. Set-up algorithm: Complexity Analysis

Since checking the positive definiteness of all representatives of a square matrix with parameters on proper real intervals is intractable [7], the question of feasibility of (23) is also intractable. To solve the problem of inherent intractability we establish a trade off between accuracy and complexity.

In fact, we develop a sequence of decentralized polynomial-time algorithms whose solutions converge to the exact solution of the NP-hard problem. In other words, the translation of a polynomial optimization problem to an LMI problem is the main source of complexity. This high complexity is unavoidable and, in fact, is the reason we seek parallel algorithms.

Algorithm 11 distributes the computation and storage of $\{\beta(h_{(\ell)}, \gamma)\}$ and $\{H_{(h_{(\ell)}, \gamma)}\}$ among the processors and their dedicated memories, respectively. In an ideal case, where the number of available processors is sufficiently large (equal to the number of available processors. Thus the per processor computational cost of calculating the $B_{i,j}$ at each Polya’s iteration is

$$\sim \text{card}(W_{d_p}) \cdot n^3 \cdot K \left( \text{floor} \left( \frac{L}{N} \right) + \text{floor} \left( \frac{M}{N} \right) \right).$$  \hspace{1cm} (42)
Algorithm 1: The parallel set-up algorithm

**Inputs:** $d_p$: degree of $P(\alpha)$, $d_d$: degree of $A(\alpha)$, $n$: number of states, $l$: number of uncertain parameters, $d_1, d_2$: number of Polya’s iterations, Coefficients of $A(\alpha)$.

**Initialization:** Set $d_1 = d_2 = 0$ and $d_{pa} = d_p + d_d$.
Calculate $L_0$ as the number of monomials in $P(\alpha)$ using (25) and $M$ as the number of monomials in $P(\alpha)A(\alpha)$ using (27). Set $L = L_0$. Calculate $L’ = \text{floor}(\frac{L}{N})$ and $M’ = \text{floor}(\frac{M}{N})$ as the number of monomials in $P(\alpha)$ and $P(\alpha)A(\alpha)$ assigned to each processor.

for $i = 1, \ldots, N$, processor $i$ do
  Initialize $\beta_{k,j}$ for $j = (i-1)L’+1$, $\ldots, iL’$ and $k = 1, \ldots, L_0$ using (21).
  Initialize $H_{k,m}$ for $m = (i-1)M’+1$, $\ldots, iM’$ and $k = 1, \ldots, L_0$ using (23).

Calculating $\beta$ and $H$ coefficients:
while $d_1 \leq d_2$ or $d_2 \leq d_2$ do
  if $d_1 \leq d_1$ then
    for $i = 1, \ldots, N$, processor $i$ do
      Set $d_p = d_p + 1$ and $d_1 = d_1 + 1$. Update $L$ using (20).
      Calculate $\beta_{k,j}$ for $j = (i-1)L’+1$, $\ldots, iL’$ and $k = 1, \ldots, L_0$ using (22).
  if $d_2 \leq d_2$ then
    for $i = 1, \ldots, N$, processor $i$ do
      Set $d_{pa} = d_{pa} + 1$ and $d_2 = d_2 + 1$. Update $M$ using (27).
      Calculate $H_{k,m}$ for $m = (i-1)M’+1$, $\ldots, iM’$ and $k = 1, \ldots, L_0$ using (23).

Calculating the SDP elements:
for $i = 1, \ldots, N$, processor $i$ do
  Calculate the number of dual variables $K$ using (33).
  Set $T’ = \text{floor}(\frac{L’}{M’})$.
  Calculate the blocks of the SDP element $C$ as $C_j$ using (35) for $j = (i-1)L’+1$, $\ldots, iL’$.
  $C_j = 0_n$ for $j = L + (i-1)M’+1$, $\ldots, L + iM’$

Set the sub-blocks of the SDP element $C$ as
$$\mathcal{C}_i = \text{diag} \left( C_{k,(i-1)T’+1} \right).$$

for $j = 1, \ldots, K$ do
  Calculate the blocks of the SDP elements $B_j$ as
$$B_{j,k} = \text{diag} \left( B_{j,k}^{T’} \right)$$
  for $k = (i-1)L’+1$, $\ldots, iL’$.
Set the sub-blocks of the SDP element $B_j$ as
$$\mathcal{B}_{j,i} = \text{diag} \left( B_{j,k}^{T’} \right).$$

**Outputs:** Sub-blocks $\mathcal{C}_i$ and $\mathcal{B}_{j,i}$ of the SDP elements for $i = 1, \ldots, N$ and $j = 1, \ldots, K$.

By substituting for $K$ from (35), $\text{card}(W_{d_p})$ from (25), $L$ from (26) and $M$ from (27), the per processor computation cost at each iteration is
$$\sim \left( \frac{(d_p + l - 1)!}{d_p!(l-1)!} \right)^2 n^4 \left( \frac{(d_p + d_1 + l - 1)!}{d_p!(d_1+l)!} \right) \left( \frac{(d_p + d_1+l-1)!}{N} \right) \text{floor} \left( \frac{(d_p + d_1+l-1)!}{(d_p + d_1 + l)!} \right) + \text{floor} \left( \frac{(d_p + d_1 + l - 1)!}{(d_p + d_1 + l)!} \right) \left( \frac{(d_p + d_1 + l - 1)!}{N} \right)$$

assuming that $l > 0$ and $N \leq M$. For example, for the case of large-scale systems (large $n$ and $l$), the computation cost per processor at each iteration is $\sim (l^{2d_p+d_1} + l^{2d_p+d_1+d_2})n^5$ having $N = L_0$ processors, $\sim (l^{2d_p+d_1} + l^{2d_p+d_1+d_2})n^5$ having $N = L$ processors and $\sim l^{2d_p+d_1+d_2}d_1n^5$ having $M$ processors. Thus for the case where $d_p \geq 3$, the number of operations grows more slowly in $n$ than in $l$.

2) Communication complexity analysis: Communication between processors can be modeled by a directed graph $G(V, E)$, where the set of nodes $V = \{1, \ldots, N\}$ is the set of indices of the available processors and the set of edges $E = \{(i, j): i, j \in V\}$ is the set of all pairs of processors that can communicate with each other. For every directed graph we can define an adjacency matrix $G$. If processor $i$ communicates with processor $j$, then $[G]_{i,j} = 1$, otherwise $[G]_{i,j} = 0$.

In this section, we only define the adjacency matrix for the part of the algorithm that performs Polya’s iterations on $P(\alpha)$. For Polya’s iterations on $P(\alpha)A(\alpha)$, the adjacency matrix can be defined in a similar manner. For simplicity, we assume that at each iteration, the number of available processors is equal to the number of monomials in $\sum_{i=1}^{l} \alpha_i d_i P(\alpha)$. Using (26), let us define $r_d$ and $r_{d+1}$ as the number of monomials in $\sum_{i=1}^{l} \alpha_i d_i P(\alpha)$ and $\sum_{i=1}^{l} \alpha_i d_i+1 P(\alpha)$. For $I = 1, \ldots, r_d$, define
$$\mathcal{E}_I := \{\text{lex. indices of monomials in } \sum_{i=1}^{l} \alpha_i \mathcal{V}^\gamma: \gamma \in W_{d_p+d_1} \text{ and } (\gamma) = I\}.$$

Then for $i = 1, \ldots, r_{d+1}$ and $j = 1, \ldots, r_d+1$,
$$[G]_{i,j} := \begin{cases} 1 & \text{if } i \leq r_d \text{ and } j \in \mathcal{E}_I \text{ and } i \neq j \\ 0 & \text{otherwise.} \end{cases}$$

Note that this definition implies that the communication graph of the set-up algorithm changes at every iteration. To help visualize the graph, the adjacency matrix for the case where
Communication directed graph for the case \( \alpha \in \Delta_3, d_p = 2 \).
(b) Communication directed graph for the case \( \alpha \in \Delta_4, d_p = 2 \).

\[
\alpha \in \Delta_2 \text{ is } \quad T_G := \begin{bmatrix}
0 & 1 & 0 & \cdots & 0 & 0 \\
0 & 0 & 1 & \cdots & 0 & 0 \\
\vdots & \vdots & \ddots & \ddots & \vdots & \vdots \\
\vdots & \vdots & \cdots & \ddots & 0 & 0 \\
0 & 0 & \cdots & \cdots & 1 & 0 \\
0 & \cdots & \cdots & \cdots & \cdots & 0 \\
\end{bmatrix} \in \mathbb{R}^{d_1 \times d_2 + 1},
\]

where the nonzero sub-block of \( T_G \) lies in \( \mathbb{R}^{d_1 \times d_2} \). We can also illustrate the communication graphs for the cases \( \alpha \in \Delta_3 \) and \( \alpha \in \Delta_4 \) with \( d_p = 2 \) as seen in Fig. 3(a) and 3(b).

For a given algorithm, the communication complexity is defined as the sum of the size of all communicated messages. For simplicity, let us consider the worst case scenario, where each processor is assigned more than one monomial and sends all of its assigned \( \beta_{(h), (\gamma)} \) and \( H_{(h), (\gamma)} \) coefficients to other processors. In this case, the algorithm assigns \( \text{floor}(\frac{L}{N}) \cdot \text{card}(W_{d_p}) \) of the \( \beta_{(h), (\gamma)} \) coefficients, each of size 1, and \( \text{floor}(\frac{M}{N}) + \text{floor}(\frac{L}{N}) \cdot \text{card}(W_{d_p}) \) of the \( H_{(h), (\gamma)} \) coefficients, each of size \( n^2 \), to each processor. Thus the communication complexity of the algorithm per processor and per iteration is

\[
\text{card}(W_{d_p}) \left( \text{floor} \left( \frac{L}{N} \right) + \text{floor} \left( \frac{M}{N} \right) n^2 \right). \quad (44)
\]

This indicates that increasing the number of processors (up to \( M \)) actually leads to less communication overhead per processor and improves the scalability of the algorithm. By substituting for \( \text{card}(W_{d_p}) \) from (25), \( L \) from (26) and \( M \) from (27) and considering large \( l \) and \( n \), the communication complexity per processor at each Polya’s iteration is \( \sim l^d p_0 + d_2 n^2 \) having \( N = L_0 \) processors, \( \sim l^d p_0 + d_2 n^2 \) having \( N = L \) processors and \( \sim l^d p_0 + d_2 n^2 \) having \( N = M \) processors.

IV. PARALLEL SDP SOLVER

In this section, we describe the steps of our primal-dual interior-point algorithm and show how, for the LMIs in (15) and (16), 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 most widely accepted interior-point algorithms are dual scaling \([44, 45]\), primal-dual \([46, 47, 48]\) and cutting-plane/spectral bundle \([49, 50, 51]\). In this paper, we use the central-path-following primal-dual algorithm described in \([48\) and \([27\). Although we found it possible to use dual-scaling algorithms, we chose to pursue a primal-dual algorithm because, in general, primal-dual algorithms converge faster \([48, 45]\) while still preserving the structure of the solution (see (59)) at each iteration. We prefer primal-dual to cutting plane/spectral bundle methods because, as we show in Section IV-D, the centralized part of our primal-dual algorithm consists of solving a symmetric system of linear equations (see (80)), whereas for the cutting plane/spectral bundle algorithm, the centralized computation would consist of solving a constrained quadratic program (see \([50, 51]\) with number of variables equal to the size of the system of linear equations. Because centralized computation is the limiting factor in a parallel algorithm, and because solving symmetric linear equations is simpler than solving a quadratic programming problem, we chose the primal-dual approach.

The choice of a central path-following primal-dual algorithm as in \([48\) and \([52\) was motivated by results in \([53\) which demonstrated better convergence, accuracy and robustness over the other types of primal-dual algorithms. More specifically, we chose the approach in \([48\) over \([52\) because unlike the Schur complement matrix (SCM) approach of the algorithm in \([52\), the SCM of \([48\) is symmetric and only the upper-triangular elements need to be sent/received by the processors. This leads to less communication overhead. The other reason for choosing \([48\) is that the symmetric SCM of the algorithm in \([48\) can be factorized using Cholesky factorization, whereas the non-symmetric SCM of \([52\) must be factorized by LU factorization (LU factorization is roughly twice as expensive as Cholesky factorization). Since factorization of SCM comprises the main portion of centralized computation in our algorithm, it is crucial for us to use computationally cheaper factorization methods to achieve better scalability.

In the primal-dual algorithm, both primal and dual problems are solved 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

\[
X_{k+1} = X_k + t_y \Delta X \quad (45)
\]
\[
y_{k+1} = y_k + t_d \Delta y \quad (46)
\]
\[
Z_{k+1} = Z_k + t_d \Delta Z, \quad (47)
\]

where \( \Delta X \), \( \Delta y \), and \( \Delta Z \) are Newton’s search direction and \( t_p \) and \( t_d \) are primal and dual step sizes. We choose the step sizes using a standard line-search between 0 and 1 with the
constraint that $X_{k+1}$ and $Z_{k+1}$ remain positive semi-definite. We use a Newton's search direction given by
\[ \Delta X = \Delta \tilde{X} + \Delta \tilde{Z}, \]
\[ \Delta y = \Delta \tilde{y} + \Delta \tilde{Z}, \]
\[ \Delta Z = \Delta \tilde{Z} + \Delta Z, \]
where $\Delta \tilde{X}$, $\Delta \tilde{y}$ and $\Delta \tilde{Z}$ are the predictor step directions and $\Delta \tilde{X}$, $\Delta \tilde{y}$, and $\Delta \tilde{Z}$ are the corrector step directions. As per [48], the predictor step directions are found as
\[ \Delta \tilde{y} = \Omega^{-1} (-a + B(Z^{-1}GX)) \]
\[ \Delta \tilde{X} = -X + Z^{-1}GB^T(\Delta \tilde{y})X \]
\[ \Delta \tilde{Z} = B^T(y) - Z - C + B^T(\Delta \tilde{y}), \]
where $C$ and the operators $B$ and $B^T$ are as defined in the previous section.
\[ G = -B^T(y) + Z + C, \]
and
\[ \Omega = [B(Z^{-1}B^T(e_1)X) \cdots B(Z^{-1}B^T(e_k)X)]. \]
Recall that $e_1, \ldots, e_k$ are the standard basis for $\mathbb{R}^k$. Once we have the predictor step directions, we can calculate the corrector step directions as per [48]. Let $\mu = \frac{1}{3} tr(ZX)$. The corrector step directions are
\[ \Delta \tilde{y} = \Omega^{-1} \left( B(Z^{-1}) - B(Z^{-1}Z \Delta \tilde{X}) \right) \]
\[ \Delta \tilde{X} = \mu Z^{-1} - Z^{-1} \Delta \tilde{Z} \Delta \tilde{X} - Z^{-1} \Delta \tilde{Z} X \]
\[ \Delta \tilde{Z} = B^T(\Delta \tilde{y}). \]
The stopping criterion is $|a^T y - tr(CX)| \leq \varepsilon$. Information regarding the selection of starting points and convergence of different variants of interior-point primal-dual algorithm, including the algorithm we use in this paper are presented in [46], [47] and [48].

B. Structure of SDP Variables
The key algorithmic insight of this paper which allows us to use the primal-dual approach presented in [48] is that by choosing an initial value for the primal variable with a certain block structure corresponding to the distributed structure of the processors, the algorithm will preserve this structure at every iteration. Specifically, we define the following structured block-diagonal subspace where each block corresponds to a single processor.
\[ S_{l,m,n} := \{ Y \subset \mathbb{R}^{(l+m)n \times (l+m)n} : Y = \text{diag}(Y_1, \ldots Y_l, Y_{l+1}, \ldots Y_{l+m}) \} \]
According to the following theorem, the subspace $S_{l,m,n}$ is invariant under Newton's iteration in the sense that when the algorithm in [48] is applied to the SDP problem defined by the polynomial optimization problem with initial value of the primal variable $X_0 \in S_{l,m,n}$, then the primal variable remains in the subspace at every Newton's iteration $X_k$.

Theorem 3: Consider the SDP problem defined in [28] and (30) with elements given by (32), (34) and (38). Suppose $L$ and $M$ are the cardinalities of $W_{d_1+d_1}$ and $W_{d_2+d_2}$. If (45), (46) and (47) are initialized by
\[ X_0 \in S_{L,M,n}, \quad y_0 \in \mathbb{R}^K, \quad Z_0 \in S_{L,M,n}, \]
then for all $k \in \mathbb{N}$,
\[ X_k \in S_{L,M,n}, \quad Z_k \in S_{L,M,n}. \]
Proof: We proceed by induction. First, suppose for some $k \in \mathbb{N}$,
\[ X_k \in S_{L,M,n} \quad \text{and} \quad Z_k \in S_{L,M,n}. \]
We would like to show that this implies $X_{k+1}, Z_{k+1} \in S_{L,M,n}$. To see this, observe that according to (45),
\[ X_{k+1} = X_k + t_p \Delta X_k \quad \text{for all} \quad k \in \mathbb{N}. \]
From (48), $\Delta X_k$ can be written as
\[ \Delta X_k = \Delta \tilde{X}_k + \Delta \tilde{Z}_k \quad \text{for all} \quad k \in \mathbb{N}. \]
To find the structure of $\Delta X_k$, we focus on the structures of $\Delta \tilde{X}_k$ and $\Delta \tilde{Z}_k$ individually. Using (52), $\Delta \tilde{X}_k$ is
\[ \Delta \tilde{X}_k = -X_k + Z_k^{-1}G_k B^T(\Delta \tilde{y}_k)X_k \quad \text{for all} \quad k \in \mathbb{N}. \]
where according to (53), $G_k$ is
\[ G_k = C - B^T(y_k) + Z_k \quad \text{for all} \quad k \in \mathbb{N}. \]
First we examine the structure of $G_k$. According to the definition of $C$ and $B_i$ in (32) and (34), and the definition of $B^T(y)$ in (61), we know that
\[ C \in S_{L,M,n}, \quad B^T : \mathbb{R}^K \mapsto S_{L,M,n}. \]
Since all the terms on the right hand side of (66) are in $S_{L,M,n}$ and $S_{L,M,n}$ is a subspace, we conclude
\[ G_k \in S_{L,M,n}. \]
Returning to (65), using our assumption in (62) and noting that the structure of the matrices in $S_{L,M,n}$ is also preserved through multiplication and inversion, we conclude
\[ \Delta \tilde{X}_k \in S_{L,M,n}. \]
Using (57), the second term in (64) is
\[ \Delta \tilde{X}_k = \mu Z_k^{-1} - Z_k^{-1} \Delta \tilde{Z}_k \Delta \tilde{X}_k - Z_k^{-1} \Delta \tilde{Z}_k X_k \quad \text{for all} \quad k \in \mathbb{N}. \]
To determine the structure of $\Delta \tilde{X}_k$, first we investigate the structure of $\Delta \tilde{Z}_k$. According to (53) and (58) we have
\[ \Delta \tilde{Z}_k = B^T(y_k) - Z_k - C + B^T(\Delta \tilde{y}_k) \quad \text{for all} \quad k \in \mathbb{N}. \]
Since all the terms in the right hand side of (71) and (72) are in $S_{L,M,n}$, then
\[ \Delta \tilde{Z}_k \in S_{L,M,n}, \quad \Delta \tilde{Z}_k \in S_{L,M,n}. \]
Recalling (69), (70) and our assumption in (62), we have
\[ \Delta \tilde{X}_k \in S_{L,M,n}. \]
According to (69), (73) and (74), the total step directions are in $S_{L,M,n}$,
\[
\Delta X_k = \Delta \hat{X}_k + \Delta \hat{Z}_k \in S_{L,M,n} \quad (75)
\]
\[
\Delta Z_k = \Delta \hat{Z}_k + \Delta \hat{Z}_k \in S_{L,M,n} \quad (76)
\]
and it follows that
\[
X_{k+1} = X_k + t_p \Delta X_k \in S_{L,M,n} \quad (77)
\]
\[
Z_{k+1} = Z_k + t_p \Delta Z_k \in S_{L,M,n} \quad (78)
\]
Thus, for any $y \in \mathbb{R}^K$ and $k \in \mathbb{N}$, if $X_k, Z_k \in S_{L,M,n}$, we have $X_{k+1}, Z_{k+1} \in S_{L,M,n}$. Since we have assumed that the initial values $X_0, Z_0 \in S_{L,M,n}$, we conclude by induction that $X_k \in S_{L,M,n}$ and $Z_k \in S_{L,M,n}$ for all $k \in \mathbb{N}$.

C. Parallel Implementation

In this section, a parallel algorithm for solving the SDP problems associated with Poly's algorithm is provided. We show how to map the block-diagonal structure of the primal variable and Newton updates described in Section XV-A to a parallel computing structure consisting of a central root processor with $N$ slave processors. Note that process steps are simultaneous and transitions between root and processor steps are synchronous. Processors are idle when root is active and vice versa. A C++ implementation of this algorithm, using MPI and Blas/Lapack libraries is provided at: www.sites.google.com/a/asu.edu/kamyar/software. Let $N$ be the number of available processors and $J = \lceil \log_2(N) \rceil$. As per Algorithm 1, we assume processor $i$ has access to the sub-blocks $\mathbf{C}_i$ and $\mathbf{B}_{i,j}$, for $j = 1, \ldots, K$. Be aware that minor parts of Algorithm 2 have been abridged in order to simplify the presentation.

D. Computational Complexity Analysis: SDP Algorithm

NC $\subseteq$ P is defined to be the class of problems which can be solved in a poly-logarithmic number of steps using a polynomially number processors and is often considered to be the class of problems that can be parallelized efficiently. The class P-complete is a set of problems which are equivalent up to an NC reduction, but contains no problem in NC and is thought to be the simplest class of “inherently sequential” problems. It has been proven that Linear Programming (LP) is P-complete [54] and SDP is P-hard (at least as hard as any P-complete problem) and thus is unlikely to admit a general-purpose parallel solution. Given this fact and given the observation that the problem we are trying to solve is NP-hard, it is important to thoroughly understand the complexity of the algorithms we are proposing and how this complexity scales with various parameters which define the size of the problem. To better understand these issues, we have broken our complexity analysis down into several cases which should be of interest to the control community. Note that the cases below do not discuss memory complexity. This is because in the cases when a sufficient number of processors are available, for a system with $n$ states, the memory requirements per block are simply proportional to $n^2$.

1) Case 1: Systems with large number of states

```
Algorithm 2: The parallel SDP solver algorithm

Inputs: $\mathbf{C}_i, \mathbf{B}_{i,j}$ for $i = 1, \ldots, N$ and $j = 1, \ldots, K$ - the sub-blocks of the SDP elements provided to processor $i$ by the set-up algorithm.

Processors Initialization step:
for $i = 1, \ldots, N$, processor $i$ do
    Initialize primal and dual variables $\mathbf{X}_i^0, \mathbf{Z}_i^0$ and $y_i^0$ as
    $\mathbf{X}_i^0 = \begin{cases}
    I_{(j+1)n}, & \text{if } 0 \leq i < L + M - NJ \\
    I_{Jn}, & \text{if } L + M - NJ \leq i < N,
    \end{cases}$
    $\mathbf{Z}_i^0 = \mathbf{X}_i^0$ and $y_i^0 = \mathbf{0} \in \mathbb{R}^K$.
    Calculate the complementary slackness [49] $S_i = tr(Z_i^0 X_i^0)$. Send $S_i$ to processor root.

Root Initialization step:
Root processor do
    Calculate the barrier parameter [49] $\mu = \frac{1}{3} \sum_{i=1}^{N} S_i$.
    Set the SDP element $a = \mathbf{I} \in \mathbb{R}^K$.

Processors step 1:
for $i = 1, \ldots, N$, processor $i$ do
    for $k = 1, \ldots, K$ do
        Calculate the elements of $\Omega_i$ (R-H-S of (80))
        $\omega_i,k = tr(\mathbf{B}_{k,i}(\mathbf{Z}_i)^{-1}(-\sum_{j=1}^{K} y_{j,i} \mathbf{B}_{j,i} + \mathbf{Z}_i + \mathbf{C}_i) \mathbf{X}_i)$
    for $l = 1, \ldots, K$ do
        Calculate the elements of the SCM as
        $\lambda_{i,k,l} = tr(\mathbf{B}_{k,i}(\mathbf{Z}_i)^{-1}\mathbf{B}_{l,i}\mathbf{X}_i)$
    Send $\omega_i,k$ and $\lambda_{i,k,l}$, $k = 1, \ldots, K$ and $l = 1, \ldots, K$ to root processor.

Root step 1:
Root processor do
    Construct the R-H-S of (80) and the SCM as
    $\Omega_i = \begin{pmatrix}
    \sum_{i=1}^{N} \omega_{i,1} \\
    \sum_{i=1}^{N} \omega_{i,2} \\
    \vdots \\
    \sum_{i=1}^{N} \omega_{i,K}
    \end{pmatrix} - a$ and
    $\Lambda = \begin{pmatrix}
    \sum_{i=1}^{N} \lambda_{i,1,1} \\
    \sum_{i=1}^{N} \lambda_{i,1,2} \\
    \vdots \\
    \sum_{i=1}^{N} \lambda_{i,1,K}
    \end{pmatrix}, \ldots, \begin{pmatrix}
    \sum_{i=1}^{N} \lambda_{i,1,K} \\
    \sum_{i=1}^{N} \lambda_{i,2,K} \\
    \vdots \\
    \sum_{i=1}^{N} \lambda_{i,K,K}
    \end{pmatrix}$
    Solve the following system of equations for the predictor dual step $\Delta \hat{y} \in \mathbb{R}^K$ and send $\Delta \hat{y}$ to all processors.
    $\Lambda \Delta \hat{y} = \Omega_i$ (80)
```
\begin{align*}
\text{Processors step 2:} \\
\text{for } i = 1, \ldots, N, \text{ processor } i \text{ do} \\
\text{Calculate the predictor step directions} \\
\Delta \tilde{X}_i = -X_i \\
+(Z_i)^{-1} \left( - \sum_{j=1}^{K} y_{j,i} \bar{B}_{j,i} + Z_i + \mathcal{C}_i \right) \sum_{j=1}^{K} \Delta \hat{y}_{j} \bar{B}_{j,i} X_i, \\
\Delta \hat{Z}_i = \sum_{j=1}^{K} y_{j,i} \bar{B}_{j,i} - Z_i - \mathcal{C}_i + \sum_{j=1}^{K} \Delta \hat{y}_{j} \bar{B}_{j,i}. \\
\text{for } k = 1, \ldots, K \text{ do} \\
\delta_{i,k} = tr(\bar{B}_{k,i}(Z_i)^{-1}), \tau_{i,k} = tr(\bar{B}_{k,i}(Z_i)^{-1} \Delta \hat{Z}_i \Delta \tilde{X}_i) \\
\text{Send } \delta_{i,k} \text{ and } \tau_{i,k}, \text{ } k = 1, \ldots, K \text{ to root processor.}
\end{align*}

Root step 2: 
Root processor do
Construct the R-H-S of (81) as
\[ \Omega_2 = \mu \left[ \sum_{i=1}^{N} \delta_{i,1} \sum_{i=1}^{N} \delta_{i,2} \cdots \sum_{i=1}^{N} \delta_{i,K} \right]^T - \left[ \sum_{i=1}^{N} \tau_{i,1} \sum_{i=1}^{N} \tau_{i,2} \cdots \sum_{i=1}^{N} \tau_{i,K} \right]^T. \]
Solve the following system of equations for the corrector dual variable \( \Delta \tilde{\varphi} \) and send \( \Delta \tilde{\varphi} \) to all processors.
\[ \Lambda \Delta \tilde{\varphi} = \Omega_2 \] (81)

\begin{align*}
\text{Processors step 3:} \\
\text{for } i = 1, \ldots, N, \text{ processor } i \text{ do} \\
\text{Calculate the corrector step directions as follows} \\
\Delta \tilde{Z}_i = \sum_{j=1}^{K} \Delta \tilde{y}_j \bar{B}_{j,i} \\
\Delta \tilde{X}_i = -(Z_i)^{-1} (\Delta \tilde{Z}_i X_i + \Delta \hat{Z}_i \Delta \tilde{X}_i) + (Z_i)^{-1} \\
\text{Calculate primal dual step total step directions as follows.} \\
\Delta X_i = \Delta \tilde{X}_i + \Delta \tilde{X}_i, \Delta Z_i = \Delta \hat{Z}_i + \Delta \hat{Z}_i, \Delta y = \Delta \hat{y} + \Delta \tilde{\varphi}. \\
\text{Set primal step size } t_p \text{ and dual step size } t_d \text{ using an appropriate line search method.} \\
\text{Update primal and dual variables as} \\
X_i = X_i + t_p \Delta X_i, \quad Z_i = Z_i + t_d \Delta Z_i, \quad y = y + t_d \Delta y
\end{align*}

\begin{align*}
\text{Processors step 4:} \\
\text{for } i = 1, \ldots, N, \text{ processor } i \text{ do} \\
\text{Calculate the contribution to primal cost} \\
\phi_i = tr(\mathcal{C}_i X_i) \text{ and the complementary slack} \\
S_i = tr(Z_i X_i). \text{ Send } S_i \text{ and } \phi_i \text{ to root processor.}
\end{align*}

Root step 4: 
Root processor do
Update the barrier parameter \( \mu = \frac{1}{2} \sum_{i=1}^{N} S_i. \) Calculate primal and dual costs as \( \phi = \sum_{i=1}^{N} \phi_i \) and \( \psi = a^T y. \) If \( |\phi - \psi| > \varepsilon, \) then go to Processors step 1; Otherwise calculate the coefficients of \( P(\alpha) \) as \( P_i = \sum_{j=1}^{N} E_j y_{(j+N-1)} \) for \( i = 1, \ldots, L_0. \)

Suppose we are considering a problem with \( n \) states. For this case, the most expensive part of the algorithm is the calculation of the Schur complement matrix \( \Lambda \) by the processors in Processors step 1 (and summed by the root in Root step 1, although we neglect this part). In particular, the computational complexity of the algorithm is determined by the number of operations required to calculate (79), restated here.
\[ \lambda_{i,k} = tr(\bar{B}_{k,i}(Z_i)^{-1} \bar{B}_{i,j} X_i) \] (82) 
for \( k = 1, \ldots, K \) and \( l = 1, \ldots, K. \)

Since the cost of \( n \times n \) matrix-matrix multiplication is \( \sim n^3 \) and each of \( X_i, Z_i, \bar{B}_{k,i} \) has \( \text{floor}(\frac{L+M}{N}) \) number of blocks in \( \mathbb{R}^{N \times n} \), the number of operations performed by the \( i^{th} \) processor to calculate \( \lambda_{i,k} \) for \( k = 1, \ldots, K \) and \( l = 1, \ldots, K \) is
\[ \sim \left( \text{floor} \left( \frac{L+M}{N} \right) \right)^2 n^3 \quad N < L + M \]
\[ \sim K^2 n^3 \quad N \geq L + M \] (83)
at each iteration, where \( i = 1, \ldots, N. \) By substituting \( K \) in (83) from (55), for \( N \geq L + M, \) each processor performs
\[ \sim ((d_p + l - 1)!)^2 \] (84)
operations per iteration. 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 \( A(\alpha)^T P(\alpha) + P(\alpha)A(\alpha) < 0, \) is proportional to \( n^7. \) Solving the LMI associated with the parameter-independent problem \( A^T P + PA < 0 \) using our algorithm or most of the SDP solvers such as [55], [27], [28] also requires \( O(n^7) \) operations per processor. Therefore, if we have a sufficient number of processors, the proposed algorithm solves both the stability and robust stability problems by performing \( O(n^7) \) operations per processor in this case.

2) Case 2: High Accuracy/Low Conservativity
In this case we consider the effect of raising Polya’s exponent. Consider the definition of simplex as follows.
\[ \tilde{\Delta}_i = \left\{ \alpha \in \mathbb{R}^l, \sum_{i=1}^{l} \alpha_i = r, \alpha_i \geq 0 \right\} \] (85)
Suppose we now define the accuracy of the algorithm as the largest value of \( r \) found by the algorithm (if it exists) such that if the uncertain parameters lie inside the corresponding simplex, the stability of the system is verified. Typically, increasing Polya’s exponent \( d \) in (10) improves the accuracy of the algorithm. If we again only consider Processor step 1, according to (84), the number of processor operations is independent of the Polya’s exponent \( d_p \) and \( d_2. \) Because this part of the algorithm does not vary with Polya’s exponent, we look at the root processing requirements associated with solving the systems of equations in (80) and (81) in Root step 1 using Cholesky factorization. Each of these systems consists of \( K \) equations. The computational complexity of Cholesky factorization is \( O(K^3). \) Thus the number of operations performed by the root processor is proportional to
\[ K^3 = \frac{(d_p + l - 1)!)^3}{(d_p)!^3(l - 1)!^3} n^6. \] (86)
In terms of communication complexity, the most significant operation between the root and other processors is sending and receiving $\lambda_{i,k,l}$ for $i = 1, \ldots, N$, $k = 1, \ldots, K$ and $l = 1, \ldots, K$ in Processes step 1 and Root step 1. Thus the total communication cost for $N$ processors per iteration is 
\[
\sim N \cdot K^2 = N \cdot \left( \frac{(d_p + l - 1)!}{(d_p)!^2((l-1)!)^2} \right)^{l_n^4}. \quad (87)
\]
From (84), (86) and (87) it is observed that the number of processors operations, root operations and communication operations are independent of Polya’s exponent $d_1$ and $d_2$. Therefore, we conclude that for a fixed $d_p$ and sufficiently large number of processors $N$ ($N \geq L + M$), improving the accuracy by increasing $d_1$ and $d_2$ does not add any computation per processor or communication overhead.

3) Case 3: Algorithm scalability/Speed-up

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 of the parallel algorithm using $N$ processors. The speed-up is governed by
\[
SP_N = \frac{N}{D + NS}, \quad (88)
\]
where $D$ is defined as the ratio of the total operations performed by all processors except root to total operations performed by all processors and root. $S$ is the ratio of the 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 (32). Using the above definitions for $D$ and $S$, Equation (84) as the decentralized computation and (86) as the centralized computation, $D$ and $S$ can be approximated as
\[
D \approx \frac{N \cdot ((d_p + l - 1))^2}{(d_p)!^2((l-1)!)^2} \cdot n^7 \quad \text{and} \quad S \approx \frac{((d_p + l - 1))^3}{(d_p)!^3((l-1)!)^3} \cdot n^6 \quad (89)
\]
According to (26) and (27) 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 (88) with their limit values, we have $\lim_{n \to \infty} SP_N = N$. Thus, for large $n$, by using $L + M$ processors the presented decentralized algorithm solves large robust stability problems $L + M$ times faster than the sequential algorithms. For different values of the state-space dimension $n$, the theoretical speed-up of the algorithm versus the number of processors is illustrated in Fig. 4. As shown in Fig. 4, 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 increasingly linear. Therefore, in case of problems with a large number of states $n$, our algorithm becomes increasingly efficient in terms of processor utilization.

4) Case 4: Synchronization and load balancing

The proposed algorithm is synchronized in that all processors must return values before the centralized step can proceed. However, in the case where we have fewer processors than blocks, some processors may be assigned one block more than other processors. In this case, some processors may remain idle while waiting for the more heavily loaded blocks to complete. In the worst case, this can result in a 50% decrease in speed. We have addressed this issue in the following manner:

1) We allocate almost the same number ($\pm 1$) of blocks of the SDP elements $C$ and $B$, to all processors, i.e., $\text{floor} \left( \frac{L + M}{N} \right) + 1$ blocks to $r$ processors and $\text{floor} \left( \frac{L + M}{N} \right)$ blocks to the other $N - r$ processors, where $r$ is the remainder of dividing $L + M$ by $N$.

2) We assign the same routine to all of the processors in the Processors steps of Alg. 2.

If $L + M$ is a multiple of $N$, then the algorithm assigns the same amount of data, i.e., $\frac{L + M}{N}$ blocks of $C$ and $B_i$ to each processor. In this case, the processors are perfectly synchronized. If $L + M$ is not a multiple of $N$, then according to (83), $r$ of $N$ processors perform $K^2n^3$ extra operations per iteration. This fraction is 
\[
\frac{1}{1 + \text{floor} \left( \frac{L + M}{N} \right)} \leq 0.5
\]
and $\frac{1}{1 + \text{floor} \left( \frac{L + M}{N} \right)} \leq 0.5$ of the operations per iteration performed by each of $r$ processors. Thus in the worst case, we have a 50% reduction, although this situation is rare. As an example, the load balancing (distribution of data and calculation) for the case of solving an SDP of the size $L + M = 24$ using different numbers of available processors, $N$, is demonstrated in Fig. 5. This figure shows the number of blocks that are allocated to each processor. According to this figure, for $N = 2, 12$ and 24, the processors are well-balanced, whereas for the case where $N = 18$, twelve processors perform 50% fewer calculations.

5) Case 5: Communication graph

The communication directed graph of the SDP algorithm (Fig. 5) is static (fixed for all iterations). At each iteration, root sends messages $(\hat{\Delta}_{i,j}^x$ and $\hat{\Delta}_{i,j}^y$) to all the processors and receives messages $(\lambda_{i,k,l}$ in (79)) from all of the processors. The adjacency matrix of the communication directed graph is

Fig. 4. Theoretical speed-up vs. No. of processors for different system dimensions $n$ for $l = 10$, $d_p = 2$, $d_a = 3$ and $d_1 = d_2 = 4$, where $L + M = 53625$
with the boundary conditions
\[ \psi(x) = A(\eta(x))\psi(x), \]
where \( A(\eta(x)) \in \mathbb{R}^{N \times N} \) has the following non-zero entries.
\[
a_{11} = -\frac{4}{3\mu_0 \Delta x^2 a^2} \left( \eta(x) + 2\eta(x) \right),
\]
\[
a_{12} = \frac{4}{3\mu_0 \Delta x^2 a^2} \left( \eta(x) \frac{x_2}{x^2} \right),
\]
\[
a_{j, j-1} = \frac{1}{\Delta x^2 \mu_0 a^2} \left( \eta(x) \frac{x_{j-1}}{x_j} \right) \text{ for } j = 2, \ldots, N - 1
\]
\[
a_{j, j} = \frac{-1}{\Delta x^2 \mu_0 a^2} \left( \eta(x) \frac{x_j + \frac{1}{2}}{x_j + \frac{1}{2}} + \eta(x) \frac{x_j - \frac{1}{2}}{x_j - \frac{1}{2}} \right) x_j
\]
\[
\text{for } j = 2, \ldots, N - 1
\]
\[
a_{j, j+1} = \frac{1}{\Delta x^2 \mu_0 a^2} \left( \eta(x) \frac{x_{j+1}}{x_{j+1}} \right) \text{ for } j = 2, \ldots, N - 1
\]
\[
a_{N, N-1} = \frac{4}{3 \Delta x \mu_0 a^2} \eta(x_{N-1}) x_{N-1},
\]
\[
a_{N, N} = \frac{-4}{3 \Delta x \mu_0 a^2} \left( \frac{2\eta(x_{N+\frac{1}{2}}) x_N}{x_{N+\frac{1}{2}} x_N} + \eta(x_{\frac{1}{2}} \eta(x)) \frac{x_N}{x_N - \frac{1}{2} \Delta x} \right),
\]
where \( \Delta x = \frac{1}{N} \) and \( x_j := \left( j - \frac{1}{2} \right) \Delta x \).

We discretize the model at \( N = 7 \) points. Typically the \( \eta(x_k) \) are not precisely known (they depend on other state variables), so we substitute for \( \eta(x_k) \) in (92) with \( \tilde{\eta}(x_k) + \alpha_j \), where \( \tilde{\eta}(x_k) \) are the nominal values of \( \eta(x_k) \) and \( \alpha_j \) are the uncertain parameters. At \( x_k = 0.036, 0.143, 0.286, 0.429, 0.571, 0.714, 0.857, 0.964 \), we use data from the Tore Supra reactor to estimate the \( \tilde{\eta}(x_k) \) as \( 1.775 \cdot 10^{-5}, 2.703 \cdot 10^{-5}, 5.676 \cdot 10^{-8}, 1.182 \cdot 10^{-7}, 2.058 \cdot 10^{-7}, 3.655 \cdot 10^{-7}, 1.076 \cdot 10^{-6}, 8.419 \cdot 10^{-6} \).

The uncertain system is then written as
\[
\dot{\psi}(x)(t) = A(\alpha)\psi(x)(t),
\]
where \( A(\alpha) = A_0 + \sum_{i=1}^8 A_i \alpha_i \) (the \( A_i \) are omitted for the sake of brevity). For a given \( \rho \), we restrict the uncertain parameters \( \alpha_j \) to \( S_\rho \), defined as
\[
S_\rho := \{ \alpha \in \mathbb{R}^8 : \sum_{i=1}^8 \alpha_i = -6|\rho|, -|\rho| \leq \alpha_i \leq |\rho| \},
\]
which is a simplex translated to the origin. We would like to determine the maximum value of \( \rho \) such that the system is stable by solving the following optimization problem.
\[
\max \rho
\]
s.t. System (98) is stable for all \( \alpha \in S_\rho \).

To represent \( S_\rho \) using the standard unit simplex defined in (7), we define the invertible map \( g : \Delta_8 \rightarrow S_\rho \) as
\[
g(\alpha) := [g_1(\alpha) \cdots g_8(\alpha)],
\]
\[
g_i(\alpha) := 2|\rho|/(\alpha_i - 0.5).
\]
Then, if we let $A'(\alpha) = A(g(\alpha))$, since $g$ is one-to-one, 
\{A'(\alpha) : \alpha \in S_p\} = \{A(g(\alpha)) : \alpha \in \Delta_S\} = \{A'(\alpha) : \alpha \in \Delta_S\}.$
Thus stability of $\dot{\psi}_x(t) = A'(\alpha)\dot{x}(t)$, for all $\alpha \in \Delta_i$ is equivalent to stability of Equation (29) for all $\alpha \in S_p$.

We solve the optimization problem in (100) using bisection. For each trial value of $\rho$, we use the proposed parallel SDP solver to solve the associated SDP obtained by the parallel setup algorithm. The SDP problems have 224 constraints with the primal variable $X \in \mathbb{R}^{1092 \times 1092}$. The normalized maximum value of $\rho$ is found to be 0.0019. In this particular example, the optimal value of $\rho$ does not change with the degrees of $P(\alpha)$ and Polya’s exponents $d_1$ and $d_2$, primarily because the model is affine.

The SDPs are constructed and solved on a parallel Linux-based cluster Cosmea at Argonne National Laboratory. Fig. 7 shows the algorithm speed-up vs. the number of processors. Note that solving this problem by SOSTOOLS [21] on the same machine is impossible due to the lack of unallocated memory.

2) Example 2: Accuracy and Convergence

The goal of this example is to investigate the effect of the degree of $P(\alpha)$, $d_p$, and the Polya’s exponents, $d_1$, $d_2$, on the accuracy of the algorithm. Given a computer with fixed amount of RAM, we compare the accuracy of the proposed algorithm with SOS algorithm. Consider the system $\dot{x}(t) = A(\alpha)x(t)$ where $A$ is a polynomial degree 3 defined as

\[ A(\alpha) = A_1\alpha_1^3 + A_2\alpha_1^2\alpha_2 + A_3\alpha_1\alpha_2\alpha_3 + A_4\alpha_1\alpha_2^2 + A_5\alpha_2^3 + A_6\alpha_3^3 \]

with the constraint

\[ \alpha \in S_L := \left\{ \alpha \in \mathbb{R}^3 : \sum_{i=1}^{3} \alpha_i = 2L + 1, L \leq \alpha_i \leq 1 \right\} \]

(102)

Using bisection in $L$, as in Example 1, we varied the parameters $d_p$, $d_1$, and $d_2$. The cluster computer Karlin at the Illinois Institute of Technology with 24 Gbytes/node of RAM (216 Gbytes total memory) was used to run our algorithm. The upper bounds on the optimal $L$ are shown in Fig. 5 in terms of $d_1$ and $d_2$ and for different $d_p$. Considering the optimal value of $L$ to be $L_{opt} = -0.111$, Fig. 8 shows how increasing $d_p$ and/or $d_1$, $d_2$ - when they are still relatively small - improves the accuracy of the algorithm. Fig. 9 demonstrates how the error in our upper bound for $L_{opt}$ decreases by increasing $d_p$ and/or $d_1$, $d_2$.

For comparison, we solved the same stability problem using the SOS algorithm [21] using only a single node of the same cluster computer and 24 Gbytes of RAM. We used the Positivstellensatz approach based on [57] to impose the constraints $\sum_{i=1}^{3} \alpha_i = 2L + 1$ and $L \leq \alpha_i \leq 1$. Table I shows the upper bounds on $L$ given by the SOS algorithm using different degrees for $x$ and $\alpha$. By considering a Lyapunov function of degree two in $x$ and degree one in $\alpha$, the SOS algorithm gives $-0.102$ as the upper bound on $L_{opt}$ as compared with our value of $-0.111$. Increasing the degree of $\alpha$ in the Lyapunov function beyond degree two resulted in a failure due to lack of memory. Note that while relevant, this comparison may not be entirely fair as the SOS algorithm has not been decentralized and it can handle global nonlinear stability problems, which our algorithm cannot.

3) Example 3: Speed-up

In this example we evaluate the efficiency of the algorithm in using additional processors to decrease computation time. As mentioned in Section IV-D on computational complexity, the measure of this efficiency is termed speed-up and in Case 3, we gave a formula for this number. To evaluate the true speed-up, we first ran the set-up algorithm on the Blue Gene supercomputer at Argonne National Laboratory using three random linear systems with different state-space dimensions and numbers of uncertain parameters. Fig. 10 shows a log-log plot of the computation time of the set-up algorithm vs. the number of processors. As can be seen, the scalability of the algorithm is practically ideal for several different state-space dimensions and numbers of uncertain parameters.

To evaluate the speed-up of the SDP portion of the algorithm, we solved three random SDP problems with different dimensions using the Karlin cluster computer. Fig. 11 gives a log-log plot of the computation time of the SDP algorithm vs. the number of processors for three different dimensions of the primal variable $X$ and the dual variable $y$. As indicated in the figure, the three dimensions of the primal variable $X$ are 200, 385 and 1092, and the dimensions of the dual variable $y$ are $K = 50$, 90 and 224, respectively. In all cases,
\(d_p = 2\) and \(d_1 = d_2 = 1\). The linearity of the Time vs. Number of Processors curves in all three cases demonstrates the scalability of the SDP algorithm.

For comparison, we plot the speed-up of our algorithm vs. that of the general-purpose parallel SDP solver SDPARA 7.3.1 as illustrated in Fig. 12. Although similar for a small number of processors, for a larger number of processors, SDPARA saturates, while our algorithm remains approximately linear.

4) Example 4: Max state-space and parameter dimensions

for a 9-node Linux cluster computer

The goal of this example is to show that given moderate computational resources, the proposed decentralized algorithms can solve robust stability problems for systems with 100+ states. We used the Karlin cluster computer with 24 Gbytes/node RAM and nine nodes. We ran the set-up and SDP algorithms to solve the robust stability problem with dimension \(n\) and \(l\) uncertain parameters on one and nine nodes of Karlin cluster computer. Thus the total memory access was thus 24 Gig and 216 Gig, respectively. Using trial and error, for different \(n\) and \(d_1, d_2\) we found the largest \(l\) for which the algorithms do not terminate due to insufficient memory (Fig. 13). In all of the runs \(d_a = d_p = 1\). Fig. 13 shows that by using 216 Gbytes of RAM, the algorithms can solve the stability problem of size \(n = 100\) with 4 uncertain parameters in \(d_1 = d_2 = 1\) Polyá’s iteration and with 3 uncertain parameters in \(d_1 = d_2 = 4\) Polyá’s iterations.

VI. CONCLUSION

In this paper, we have presented a cluster-computing and supercomputing approach to stability analysis of large-scale linear systems of the form \(\dot{x}(t) = A(\alpha)x(t)\) where \(A\) is polynomial, \(\alpha \in \Delta \subset \mathbb{R}^l\) and \(x \in \mathbb{R}^n\) and where \(n \geq 100\) or \(\alpha \geq 10\). The approach is based on mapping the structure of the LMI conditions associated with Polyá’s theorem to a decentralized computing environment. We have shown that for a sufficient number of processors, the proposed algorithm can solve the NP-hard robust stability problem with the same per-core computation cost as solving the Lyapunov inequality for a system with no parametric uncertainty. Theoretical and experimental results verify near-perfect scalability and speed-up for up to 200 processors. Moreover, numerical examples demonstrate the ability of the algorithm to perform robust
analysis of systems with 100+ states and several uncertain parameters using a simple 9-node Linux cluster computer. We have also argued that our algorithms can also be extended to solve nonlinear stability analysis and robust controller synthesis problems, although this is left for future work.

REFERENCES

[1] S. Gugercin and A. Antoulas, “A survey of model reduction by balanced truncation and some new results,” International Journal of Control, vol. 77, no. 8, pp. 748–766, 2004.

[2] 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.

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

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

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

[6] V. Blondel and J. Tsitsiklis, “A survey of computational complexity results in systems and control,” Automatica, vol. 36, no. 9, pp. 1249–1274, 2000.

[7] A. Nemirovskii, “Several NP-hard problems arising in robust stability analysis,” Mathematics of Control, Signals, and Systems (MCSS), vol. 6, no. 2, pp. 99–105, 1993.

[8] D. Walker and J. Dongarra, “Mpi: a standard message passing interface,” Supercomputer, vol. 12, pp. 56–68, 1996.

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

[10] 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.

[11] 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.

[12] 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.

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

[14] 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.

[15] P. A. Bliman, “An existence result for polynomial solutions of parameter dependent LMI,” Systems & Control Letters, no. 3–4, pp. 165–169, 2004.

[16] M. Peet, “Exponentially stable nonlinear systems have polynomial lyapunov functions on bounded regions,” Automatic Control, IEEE Transactions on, vol. 54, no. 5, pp. 979–987, 2009.

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

[18] 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.

[19] X. Zhang and P. Tsitrotras, “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.

[20] X. Zhang, P. Tsitrotras, and P. A. Bliman, “Multi-parameter dependent lyapunov functions for stability analysis of parameter-dependent LTI systems,” pp. 1263–1268, in Proceedings of IEEE International Symposium on, Mediterranean Conference on Control and Automation, 2005.

[21] S. Prajna, 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.

[22] D. Henrion and J. B. Lasserre, “Gloptipoly: Global optimization over polynomials with Matlab and SeDuMi,” Proceedings of IEEE Conference on Decision and Control, Mar 2003.

[23] 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.

[24] 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.

[25] 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, no. 2, pp. 1334–1340, Jul 2007.

[26] R. C. L. F. Oliveira, P.-A. Bliman, and P. L. D. Peres, “Robust LMIs with multi-parametric dependencies: Existence of solutions and applications,” pp. 2226–2231, Proceedings of IEEE Conference on Decision and Control, 2008.

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

[30] L. Kalé, B. Ranumumar, A. Sinha, and A. Gursoy, “The charm parallel programming language and system: Part I–description of language features,” Parallel Programming Laboratory Technical Report No. 95-02, 1994.

[31] S. Deitz, High-level programming language abstractions for advanced and dynamic parallel computations. PhD thesis, Computer Science and Engineering Department, University of Washington, 2005.

[32] K. Gatermann and P. Parrilo, “Symmetry groups, semidefinite programs, and sums of squares,” Journal of Pure and Applied Algebra, vol. 192, no. 1, pp. 95–128, 2004.

[33] P. Parrilo, “Exploiting algebraic structure in sum of squares programs,” Positive polynomials in control, pp. 580–580, 2005.

[34] S. Kim, M. Kojima, and H. Waki, “Generalized lagrangian duals and sums of squares relaxations of sparse polynomial optimization problems,” SIAM Journal on Optimization, vol. 15, no. 3, pp. 697–719, 2005.
Reza Kamyar received the B.S. and M.S in aerospace engineering from Sharif University of Technology, Tehran, Iran in 2008, and 2010. He is currently a Ph.D student in the department of mechanical engineering of Arizona State University, Tempe, Arizona. He is a research assistant with Cybernetic Systems and Controls Laboratory (CSCL) in the School for Engineering of Matter, Transport and Energy (SEMTE) at Arizona State University. His research focuses on the development of decentralized algorithms applied to the problems of stability and control of large-scale complex systems.

Matthew M. Peet received the B.S. degrees in physics and in aerospace engineering from the University of Texas at Austin in 1999 and the M.S. and Ph.D. degrees in aeronautics and astronautics from Stanford University, Stanford, CA, in 2001 and 2006, respectively. He was a Postdoctoral Fellow at the National Institute for Research in Computer Science and Control (INRIA), Paris, France, from 2006 to 2008, where he worked in the SISYPHE and BANG groups. He was an Assistant Professor of Aerospace Engineering in the Mechanical, Materials, and Aerospace Engineering Department, Illinois Institute of Technology, Chicago, from 2008 to 2012. He is currently an Assistant Professor of Aerospace Engineering in the School for the Engineering of Matter, Transport, and Energy at Arizona State University, Tempe, and Director of the Cybernetic Systems and Controls Laboratory. His research interests are in the role of computation as it is applied to the understanding and control of complex and large-scale systems. Applications include fusion energy and immunology. Dr. Peet received an NSF CAREER award in 2011.

Yulia Peet is an Assistant Professor of Mechanical and Aerospace Engineering at the School for Engineering of Matter, Transport and Energy at Arizona State University. Her Ph.D. degree is in Aeronautics and Astronautics from Stanford (2006), M.S. in Aerospace Engineering (1999) and B.S. in Applied Mathematics and Physics (1997) from Moscow Institute of Physics and Technology in Russia. Her previous appointments include a postdoctoral position at the University of Pierre and Marie Curie in Paris in 2006-2008, and a dual appointment as an NSF research and teaching fellow at Northwestern University and assistant computational scientist at the Mathematics and Computer Science Division at Argonne National Laboratory in 2009-2012.