Research Reports on Mathematical and Computing Sciences

PHoMpara – Parallel Implementation of the Polyhedral Homotopy Continuation Method for Polynomial Systems

Takayuki Gunji, Sunyoung Kim, Katsuki Fujisawa and Masakazu Kojima

October 2005, B–419

Department of Mathematical and Computing Sciences
Tokyo Institute of Technology

SERIES B: Operations Research

B-419 PHoMpara – Parallel Implementation of the Polyhedral Homotopy Continuation Method for Polynomial Systems

Takayuki Gunji[⋆], Sunyoung Kim[†], Katsuki Fujisawa[‡], Masakazu Kojima[♯],

October 2005

Abstract.

The polyhedral homotopy continuation method is known to be a successful method for finding all isolated solutions of a system of polynomial equations. PHoM, implementation of the method in C++, finds all isolated solutions of a polynomial system by constructing a family of polyhedral-linear homotopy functions, tracing the solution curves of the homotopy equations, and verifying the obtained solutions. A software package PHoMpara parallelizes PHoM to solve a polynomial system of large size. Many characteristics of the polyhedral homotopy continuation method make parallel implementation efficient and provide excellent scalability. Numerical results include some large polynomial systems that had not been solved.

AMS subject classification.

Primary: 65H10 Systems of equations

Secondary: 65H20 Global methods, including homotopy approaches

Key words.

Polynomials, parallel computation, homotopy continuation methods, equations, polyhedral homotopy, numerical experiments, software package.

⋆ Department of Mathematical and Computing Sciences, Tokyo Institute of Technology, 2-12-1 Oh-Okayama, Meguro-ku, Tokyo 152-8552 Japan. *gunji1@is.titech.ac.jp*

† Department of Mathematics, Ewha Women's University, 11-1 Dahyun-dong, Sudaemoon-gu, Seoul 120-750 Korea. A considerable part of this work was conducted while this author was visiting Tokyo Institute of Technology. Research was supported by KRF 2004-042-C00014. *skim@ewha.ac.kr*

‡ Department of Mathematical Sciences, Tokyo Denki University, Ishizuka, Hatoyama, Saitama 350-0394 Japan. Research supported by Grant-in-Aid for Scientific Research on Priority Areas 16016234 and JST Grant-in-Aid for Applying Advanced Computational Science and Technology(ACT-JST). *fujisawa@r.dendai.ac.jp*

♯ Department of Mathematical and Computing Sciences, Tokyo Institute of Technology, 2-12-1 Oh-Okayama, Meguro-ku, Tokyo 152-8552 Japan. Research supported by Grant-in-Aid for Scientific Research on Priority Areas 16016234. *kojima@is.titech.ac.jp*

1 Introduction

We consider solving a polynomial system $\boldsymbol{f}(\boldsymbol{x}) = \boldsymbol{0}$ by the polyhedral homotopy (continuation) method, where $\boldsymbol{f}(\boldsymbol{x}) = (f_1(\boldsymbol{x}), f_2(\boldsymbol{x}), \ldots, f_n(\boldsymbol{x})) \in \mathbb{C}^n$ and $f_j(\boldsymbol{x}) \in \mathbb{C}$ denotes a polynomial in a variable vector $\boldsymbol{x} = (x_1, x_2, \ldots, x_n)$ in the n-dimensional complex space $\mathbb{C}^n$. The polyhedral homotopy continuation method has been successful to find all isolated solutions of many polynomial systems as shown in [7, 21] using software such as PHCpack [24] and PHoM [7]. As the size of a polynomial system becomes larger, we need more computing resources to solve the polynomial system efficiently. Parallelizing the polyhedral homotopy method is a natural way to obtain computing resources. The polyhedral homotopy method is well-suited to parallelization when performing each important task of the method.

The polyhedral homotopy method provides computational advantages over the linear homotopy (continuation) method [2, 5, 10] since the number of homotopy curves to be traced is smaller than the number of homotopy curves in the linear homotopy method. More precisely, the number of homotopy curves in the polyhedral homotopy method is determined by the mixed volume of the polynomial system based on the Bernshtein theory [3, 8, 11, 18, 22], which bounds the number of the isolated solutions of $\boldsymbol{f}(\boldsymbol{x}) = 0$ tighter than the Bézout bound for the number of homotopy curves in the linear homotopy method.

When we deal with a polyhedral or linear homotopy function $\boldsymbol{h}'(\boldsymbol{x}, t)$ for a polynomial system of equations $\boldsymbol{f}(\boldsymbol{x}) = \boldsymbol{0}$, the homotopy parameter t usually changes from 0 to 1, and we impose the assumptions that $\boldsymbol{h}'(\boldsymbol{x}, 0) = \boldsymbol{0}$ can be easily solved and that $\boldsymbol{h}'(\boldsymbol{x}, 1)$ coincides with $\boldsymbol{f}(\boldsymbol{x})$. In the paper [9], the nonlinear scaling $t \in (0, 1] \rightarrow s = \log t \in (-\infty, 0]$ or $s \in (-\infty, 0] \rightarrow t = \exp(s) \in (0, 1]$ was proposed and used in the current version of software package PHoM [7] to increase numerical stability. We utilize a homotopy function of the form $\boldsymbol{h}(\boldsymbol{x}, s) = \boldsymbol{h}'(\boldsymbol{x}, \exp(s))$ throughout the paper. For simplicity of notation, we write $\boldsymbol{h}(\boldsymbol{x}, -\infty) = \lim_{s \rightarrow -\infty} \boldsymbol{h}(\boldsymbol{x}, s) = \boldsymbol{h}'(\boldsymbol{x}, 0)$. It should be noted that the scaled homotopy system $\boldsymbol{h}(\boldsymbol{x}, s) = 0$ with the parameter $s \in [-\infty, 0]$ is equivalent to the original homotopy system $\boldsymbol{h}'(\boldsymbol{x}, t) = 0$ with the parameter $t \in [0, 1]$, and that if we choose a sufficiently large positive number s^0, then each solution $\boldsymbol{x}^0$ of $\boldsymbol{h}'(\boldsymbol{x}, 0) = 0$ serves as an accurate solution of $\boldsymbol{h}(\boldsymbol{x}, -s^0) = 0$. Hence, we can start tracing a solution curve of the scaled homotopy system $\boldsymbol{h}(\boldsymbol{x}, s) = \boldsymbol{0}$ from $(\boldsymbol{x}^0, -s^0)$.

Implementation of the polyhedral homotopy method is carried out as follows:

(i) Compute the fine mixed cells of a given polynomial system [6, 12, 19] to construct a family of polyhedral (or polyhedral-linear) homotopy functions

$$\boldsymbol{h}(\boldsymbol{x}, s) = (h_1(\boldsymbol{x}, s), h_2(\boldsymbol{x}, s), \ldots, h_n(\boldsymbol{x}, s)) \in \mathbb{C}^n \ ((\boldsymbol{x}, s) \in \mathbb{C}^n \times (-\infty, 0]).$$

(ii) Trace all solution curves of every homotopy system $\boldsymbol{h}(\boldsymbol{x}, s) = \boldsymbol{0}$ in the family, which are called homotopy (solution) curves, numerically varying the homotopy parameter s from $s = -s^0$ to $s = 0$ with a sufficiently large positive number s^0 [7, 9, 11, 23].

(iii) Verify the obtained solutions at $s = 0$ [7].

Here each homotopy function $\boldsymbol{h}(\boldsymbol{x}, s)$ in the family satisfies the following three conditions: (a) all solutions of *the starting polynomial system* $\boldsymbol{h}(\boldsymbol{x}, -\infty) = \boldsymbol{0}$ are easy to obtain, (b) for all s in $[-\infty, 0)$, the homotopy system $\boldsymbol{h}(\boldsymbol{x}, s) = \boldsymbol{0}$ has only nonsingular solutions (this

ensures that the set of solutions $(\boldsymbol{x}, s) \in \mathbb{C}^n \times [-\infty, 0)$ of $\boldsymbol{h}(\boldsymbol{x}, s) = \boldsymbol{0}$ consists of disjoint homotopy solution curves), (c) *the target polynomial system* $\boldsymbol{h}(\boldsymbol{x}, 0) = \boldsymbol{0}$ coincides with $\boldsymbol{f}(\boldsymbol{x}) = \boldsymbol{0}$. More details on these properties of the polyhedral-linear homotopy used in PHoM is discussed in Section 2.

A polynomial system of increasing size, such as noon-n and economic-n, has an increasing number of fine mixed cells to compute and homotopy curves to trace as n grows. The software package PHCpack [24] or PHoM [7] has been used to solve such polynomial systems. The size of polynomial systems that could be solved by these software packages has been limited to, for example, noon-5 and economic-8 by PHCpack and noon-8 and economic-12 by PHoM. The main reason of not being able to extend the size of polynomial systems further is that the numbers of fine mixed cells to compute and homotopy curves to trace are too large to handle on a single computer.

The aim of this paper is to present the description of PHoMpara, parallel implementation of PHoM, and to show with numerical results that parallelization provides computing resources to solve polynomial systems of larger size; it can compute the solutions of noon-12 and economic-16 as shown in Section 4.

Each stage of the polyhedral homotopy method is parallelized. In the initial stage (i) of computing fine mixed cells to construct a family of polyhedral homotopy functions, all possible candidates of fine mixed cells are described as solutions of linear systems of equations. Each linear system of the collection of the linear systems provides a fine mixed cell if and only if it has a feasible solution. With linear subsystems of the linear systems, we construct an enumeration tree whose root node is the empty linear subsystem, and each node represents a linear subsystem. As we go down from the root node to child nodes, the size of linear subsystems assigned to child nodes becomes larger, and the linear systems describing all possible candidates of fine mixed cells are located at the leaf nodes of the tree. Under this construction of the enumeration tree, the root node with the empty linear subsystem is feasible, and if a node (more precisely, a linear subsystem assigned to a node) is infeasible, then so are all of its child nodes. Therefore, computing all fine mixed cell is executed with the depth-first search applied to the enumeration tree from the root node: for a feasible node, check the feasibility of its child node. This process is parallelized since each node can be tested independently. The computational time required at a node for this process is different from that of other nodes. The gap in the computational time should be reduced for efficient parallel implementation. We propose a technique of redistributing the nodes to handle the difference.

When a homotopy solution curve is traced successfully from $s = -s^0$ for some large positive number s^0 to $s = 0$ in the stage (ii), each curve arrives at an isolated solution of $\boldsymbol{f}(\boldsymbol{x}) = 0$ if it does not diverge. We have increasingly large number of homotopy solution curves as we solve larger polynomial systems, and it takes longer time to compute all isolated solutions. Parallel implementation of tracing homotopy curves can reduce the computational time significantly.

After tracing homotopy curves, the stage (iii) of verifying the obtained solutions follows. Sorting solutions with respect to some norm is necessary at this stage and it is also parallelized.

Thus the main three stages of the polyhedral homotopy method are all parallelized to reduce the total computational time for solving polynomial systems of large size. A common feature of the three stages is that each important task on a single cpu is successively sub-

divided into multiple subtasks, which can be processed independently from others without communicating with them. This feature of the polyhedral homotopy method fits better to a simple master-worker type parallel computation environment, where one master cpu communicates with all worker cpus and the communication among the workers is not allowed, than message passing libraries such as MPI [14], where asynchronous communication between the workers is allowed. PHoMpara utilizes Ninf [17], a middleware which provides communication library functions for parallel computation in master-worker type PC clusters. Ninf is easy to use and efficient for parallel implementation of the polyhedral homotopy method satisfying the above feature.

This paper is organized as follows: In Sections 2, we introduce the polyhedral homotopy method, and briefly describe the three modules of PHoM, StartSystem, CMPSc and Verify, which correspond to the three stages (i), (ii) and (iii), respectively. Section 3 contains the description of parallelization of PHoM for each of the three modules. In Section 4, we present numerical results on economic-n, katsura-n, noon-n and reimer-n polynomial systems. Finally, Section 5 is devoted to concluding remarks.

We introduce notation and symbols for the subsequent discussions. Let $\mathbb{R}$ and $\mathbb{Z}_+$ denote the set of real numbers and the set of nonnegative integers, respectively. For every vector variable $\boldsymbol{x} \equiv (x_1, x_2, \ldots, x_n) \in \mathbb{C}^n$ and every $\boldsymbol{a} \equiv (a_1, a_2, \ldots, a_n) \in \mathbb{Z}^n_+$, we use the notation $\boldsymbol{x}^{\boldsymbol{a}}$ for the term $x_1^{a_1} x_2^{a_2} \cdots x_n^{a_n}$. Then we can write each component polynomial $f_j(\boldsymbol{x})$ of $\boldsymbol{f}(\boldsymbol{x})$ as $f_j(\boldsymbol{x}) \equiv \sum_{\boldsymbol{a} \in \mathcal{A}_j} c_j(\boldsymbol{a})\, \boldsymbol{x}^{\boldsymbol{a}}$ $(j = 1, 2, \ldots, n)$ for some finite subset $\mathcal{A}_j$ of $\mathbb{Z}^n_+$ $(j = 1, 2, \ldots, n)$ and some $c_j(\boldsymbol{a}) \in \mathbb{C}$ $(\boldsymbol{a} \in \mathcal{A}_j,\ j = 1, 2, \ldots, n)$. We call $\mathcal{A}_j$ the *support* of the polynomial $f_j(\boldsymbol{x})$.

2 The Polyhedral Homotopy Continuation Method

PHoM employs a polyhedral-linear homotopy, a combination of the polyhedral homotopy and the linear homotopy, which was called as the cheater's homotopy in the paper [11]. PHoM is written in C++, and has three modules StartSystem, CMPSc, and Verify.

2.1 Polyhedral-linear homotopy functions

The fine mixed cells of the polynomial system $\boldsymbol{f}(\boldsymbol{x})$ should be computed to construct a class of polyhedral-linear homotopy functions. We define a finite family of polyhedral-linear homotopy functions as [9], $\boldsymbol{h}^p : \mathbb{C}^n \times [-\infty, 0] \to \mathbb{C}^n$ $(p = 1, 2, \ldots, p^*)$ by

$$\begin{aligned} h^p_j(\boldsymbol{x}, s) \equiv & \sum_{\boldsymbol{a} \in \mathcal{A}_j} \big(\tilde{c}_j(\boldsymbol{a}) \exp(\rho^p_j(\boldsymbol{a}) s) \\ & + (c_j(\boldsymbol{a}) - \tilde{c}_j(\boldsymbol{a})) \exp((\rho^p_j(\boldsymbol{a}) + 1) s) \big)\, \boldsymbol{x}^{\boldsymbol{a}} = 0 \ (j = 1, 2, \ldots, n), \end{aligned} \tag{1}$$

where $\tilde{c}_j(\boldsymbol{a})$ $(\boldsymbol{a} \in \mathcal{A}_j, j = 1, 2, \ldots, n)$ are randomly chosen numbers. Positive integer p^* and nonnegative numbers $\rho^p_j(\boldsymbol{a})$ $(p = 1, 2, \ldots, p^*, \boldsymbol{a} \in \mathcal{A}_j, j = 1, 2, \ldots, n)$ are first obtained by computing fine mixed cells described in Section 2.2, and then $\rho^p_j(\boldsymbol{a})$ is recomputed as described in Section 2.3. We notice that $\boldsymbol{h}^p(\boldsymbol{x}, 0) = \boldsymbol{f}(\boldsymbol{x})$ for every $\boldsymbol{x} \in \mathbb{C}^n$ $(p = 1, 2, \ldots, p^*)$. The family of polyhedral-linear homotopy functions (1) should satisfy the followings:

(a) Each component $h_j^p(\boldsymbol{x}, -\infty)$ $(j = 1, 2, \ldots, n)$ is a binomial, and all solutions of the starting polynomial system $\boldsymbol{h}^p(\boldsymbol{x}, -\infty) = \boldsymbol{0}$ can be computed easily.

(b) For every $p = 1, 2, \ldots, p^*$ and every fixed $s \in [-\infty, 0)$, the polynomial system $\boldsymbol{h}^p(\boldsymbol{x}, s) = \boldsymbol{0}$ has only nonsingular solutions, hence, each connected component of $\{(\boldsymbol{x}, s) \in \mathbb{C}^n \times [-\infty, 0) : \boldsymbol{h}^p(\boldsymbol{x}, s) = \boldsymbol{0}\}$ that intersects with $\mathbb{C}^n \times \{-\infty\}$ forms a smooth curve $\{(\boldsymbol{\xi}(s), s) : s \in [-\infty, 0)\}$, which is called as a homotopy curve of $\boldsymbol{h}^p(\boldsymbol{x}, s) = \boldsymbol{0}$.

(c) For each isolated solutions $\boldsymbol{x}^1$ of $\boldsymbol{f}(\boldsymbol{x}) = \boldsymbol{0}$, there exist an index p and a solution $\boldsymbol{x}^0$ of the starting polynomial system $\boldsymbol{h}^p(\boldsymbol{x}, -\infty) = \boldsymbol{0}$ such that $(\boldsymbol{x}^0, -\infty)$ is connected to $(\boldsymbol{x}^1, 0)$ through a homotopy curve of $\boldsymbol{h}^p(\boldsymbol{x}, s) = \boldsymbol{0}$.

2.2 Computing fine mixed cells in StartSystem

The module StartSystem computes fine mixed cells, which determines the coefficients $\rho_j^p(\boldsymbol{a})$ of the continuation parameter s in (1), balances those coefficients, and solves binomial systems to obtain starting points for tracing homotopy curves.

We first give a definition of fine mixed cells of the polynomial system $\boldsymbol{f}(\boldsymbol{x})$. Let $\langle \boldsymbol{a}, \boldsymbol{\alpha} \rangle$ denote the inner product of two vectors $\boldsymbol{a}$, $\boldsymbol{\alpha} \in \mathbb{R}^n$ and let $\omega_j(\boldsymbol{a})$ be a random number, which is called *lifting*, chosen from a bounded open interval of $\mathbb{R}$ $(\boldsymbol{a} \in \mathcal{A}_j,\ j = 1, 2, \ldots, n)$. We consider a problem of finding all solutions $(\boldsymbol{\alpha}, \boldsymbol{\beta}) \equiv (\alpha_1, \alpha_2, \ldots, \alpha_n, \beta_1, \beta_2, \ldots, \beta_n) \in \mathbb{R}^{2n}$ which satisfy the linear inequality system

$$\omega_j(\boldsymbol{a}) + \langle \boldsymbol{a}, \boldsymbol{\alpha} \rangle - \beta_j \geq 0 \ (\boldsymbol{a} \in \mathcal{A}_j,\ j = 1, 2, \ldots, n) \tag{2}$$

and the additional condition

$$\text{equalities hold in (2) with some } \boldsymbol{\gamma}^j \in \mathcal{A}_j \text{ and } \boldsymbol{\delta}^j \in \mathcal{A}_j,\ \boldsymbol{\gamma}^j \neq \boldsymbol{\delta}^j, \text{ for each } j. \tag{3}$$

We assume a nondegeneracy condition on the linear inequality system (2) such that at most $2n$ equalities hold at any solution. This condition is satisfied generically by randomly choosing liftings $\omega_j(\boldsymbol{a})$ $(\boldsymbol{a} \in \mathcal{A}_j,\ j = 1, 2, \ldots, n)$. Then exactly two equalities hold for each j at every solution $(\boldsymbol{\alpha}, \boldsymbol{\beta})$ of (2) and (3). Let $(\boldsymbol{\alpha}^1, \boldsymbol{\beta}^1), (\boldsymbol{\alpha}^2, \boldsymbol{\beta}^2), \ldots, (\boldsymbol{\alpha}^{p^*}, \boldsymbol{\beta}^{p^*})$ denote all solutions of (2) and (3). Then the fine mixed cells $\boldsymbol{C}^p$ $(p = 1, 2, \ldots, p^*)$ and $\rho_j^p(\boldsymbol{a})$ $(p = 1, 2, \ldots, p^*,\ \boldsymbol{a} \in \mathcal{A}_j,\ j = 1, 2, \ldots, n)$ are obtained as following.

$$\left.\begin{array}{lcl} C_j^p & \equiv & \{\boldsymbol{a} \in \mathcal{A}_j : \omega_j(\boldsymbol{a}) + \langle \boldsymbol{a}, \boldsymbol{\alpha} \rangle - \beta_j = 0\} \ \ (j = 1, 2, \ldots, n), \\ \boldsymbol{C}^p & \equiv & (C_1^p, C_2^p, \ldots, C_n^p) \subseteq \mathcal{A}, \\ \rho_j^p(\boldsymbol{a}) & \equiv & \omega_j(\boldsymbol{a}) + \langle \boldsymbol{a}, \boldsymbol{\alpha}^p \rangle - \beta_j^p \ \ (\boldsymbol{a} \in \mathcal{A}_j,\ j = 1, 2, \ldots, n), \end{array}\right\} \tag{4}$$

where $\mathcal{A} \equiv (\mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_n)$. Note that each C_j^p consists of two elements and for every $p = 1, 2, \ldots, p^*$, the starting polynomial system (1) with $s = -\infty$ turns out to be a binomial system

$$h_j^p(\boldsymbol{x}, -\infty) \ \equiv \ \sum_{\boldsymbol{a} \in C_j} \tilde{c}_j(\boldsymbol{a}) \boldsymbol{x}^{\boldsymbol{a}} = 0 \ (j = 1, 2, \ldots, n),$$

which can be solved by a method from linear algebra.

In order to find all solutions of (2) and (3), we use the method [19], which is outlined as follows:

Algorithm 2.1. *Step 1. Define*

$$\mathcal{S}(k) \equiv \left\{ \boldsymbol{N} = (N_1, N_2, \ldots, N_n) : \begin{array}{ll} N_j \subseteq \mathcal{A}_j, & \#N_j = 2 \ (j = 1, 2, \ldots, k), \\ & N_j = \phi \ (j = k+1, k+2, \ldots, n) \end{array} \right\}$$

$(k = 0, 1, \ldots, n)$. *Here* $\#N_j$ *denotes the number of elements in* N_j. *Note that* $\mathcal{S}(n)$ *includes all possible candidates for fine mixed cells.*

Step 2. Construct an enumeration tree, which has a cell $(\phi, \phi, \ldots, \phi) \in \mathcal{S}(0)$ *at the root node and every* $\boldsymbol{N} \in \mathcal{S}(n)$ *at leaf nodes with no child node. Any child node* $\boldsymbol{N}' \in$

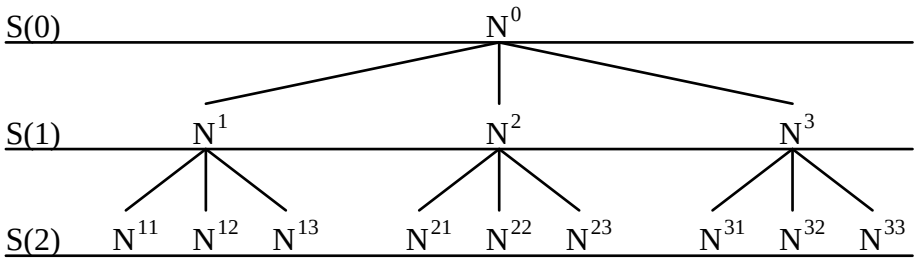

Figure 1: An enumeration tree for $n = 2$ and $\#\mathcal{A}_1 = \#\mathcal{A}_2 = 3$

$\mathcal{S}(k+1)$ *of node* $\boldsymbol{N} \in \mathcal{S}(k)$ *has the first* k *sets* $N'_1, N'_2, \ldots, N'_k$ *of* $\boldsymbol{N}'$ *equivalent to those of* $\boldsymbol{N}$.

Step 3. Apply the depth-first search to the enumeration tree until all solutions of (2) and (3) are enumerated. More precisely, with $\boldsymbol{N} \in \mathcal{S}(k)$, *check the feasibility of*

$$\left. \begin{array}{l} \beta_j - \langle \boldsymbol{a}, \boldsymbol{\alpha} \rangle = \omega_j(\boldsymbol{a}) \ \ (\boldsymbol{a} \in N_j, \ j = 1, 2, \ldots, n), \\ \beta_j - \langle \boldsymbol{a}, \boldsymbol{\alpha} \rangle \leq \omega_j(\boldsymbol{a}) \ \ (\boldsymbol{a} \in \mathcal{A}_j \backslash N_j, \ j = 1, 2, \ldots, n). \end{array} \right\} \tag{5}$$

If it turn out to be infeasible, ignore the subtree whose root is $\boldsymbol{N}$.

An enumeration tree defined in Step 2 for a polynomial system of 2 variables and 2 equations, and 3 elements in supports $\mathcal{A}_1$ and $\mathcal{A}_2$ is shown in Figure 1. The nodes represent $\boldsymbol{N}^0 = (\phi, \phi), \boldsymbol{N}^1 = (N_1^1, \phi), \boldsymbol{N}^{11} = (N_1^1, N_2^{11})$, and so on.

In Step 3, $\boldsymbol{N}$ is a fine mixed cell if and only if $\boldsymbol{N}$ is a leaf node included in $\mathcal{S}(n)$ and the system of inequalities (5) is feasible. The feasibility of the system (5) is tested for a linear program with the constraints (5) at each node $\boldsymbol{N} \in \mathcal{S}(k)$ of the enumeration tree with $k \in \{1, 2, \ldots, n\}$. If the system (5) is infeasible at some node $\boldsymbol{N} \in \mathcal{S}(k)$, so is at any child $\boldsymbol{N}' \in \mathcal{S}(k+1)$. Since the subtree whose root is $\boldsymbol{N}$ does not contain any fine mixed cell, the subtree is ignored. See [19] for more details.

2.3 Balancing the coefficients of the continuation parameter s in StartSystem

In the homotopy function (1), the large magnitude of the coefficients $\rho_j^p(\boldsymbol{a})$ $(\boldsymbol{a} \in \mathcal{A}_j, \ j = 1, 2, \ldots, n)$ of the homotopy parameter s affects numerical stability and computational efficiency when homotopy curves are traced. A technique called balancing [6] was introduced

to reduce the magnitude of the coefficients $\rho_j^p(\boldsymbol{a})$ ($\boldsymbol{a} \in \mathcal{A}_j,\ j = 1, 2, \ldots, n$). It decreases the ratio

$$\frac{\max\{\rho_j^p(\boldsymbol{a}) : \boldsymbol{a} \in \mathcal{A}_j,\ j = 1, 2, \ldots, n,\ p = 1, 2, \ldots, p^*\}}{\min\{\rho_j^p(\boldsymbol{a}) : \boldsymbol{a} \in \mathcal{A}_j,\ j = 1, 2, \ldots, n,\ p = 1, 2, \ldots, p^*\}} \tag{6}$$

while keeping the fine mixed cells $\boldsymbol{C}^p$ ($p = 1, 2, \ldots, p^*$) unchanged by solving the following linear program with variables M and $\boldsymbol{\omega} \equiv (\omega_j(\boldsymbol{a}) : \boldsymbol{a} \in \mathcal{A}_j,\ j = 1, 2, \ldots, n)$:

$$\left.\begin{array}{ll} \text{minimize} & M \\ \text{subject to} & 1 \le \big(\omega_j(\boldsymbol{a}) - \omega_j(\boldsymbol{\gamma}_j^p)\big) + \langle \boldsymbol{a} - \boldsymbol{\gamma}_j^p, \boldsymbol{\sigma}^p(\boldsymbol{\omega})\rangle \le M \\ & \qquad (\boldsymbol{a} \in \mathcal{A}_j \setminus C_j^p,\ j = 1, 2, \ldots, n,\ p = 1, 2, \ldots, p^*), \end{array}\right\} \tag{7}$$

where $C_i^p = (\boldsymbol{\gamma}_j^p, \boldsymbol{\delta}_j^p)$ and

$$\boldsymbol{\sigma}^p(\boldsymbol{\omega}) = -\begin{pmatrix} (\boldsymbol{\delta}_1^p - \boldsymbol{\gamma}_1^p)^T \\ \vdots \\ (\boldsymbol{\delta}_n^p - \boldsymbol{\gamma}_n^p)^T \end{pmatrix}^{-1} \begin{pmatrix} \omega_1(\boldsymbol{\delta}_1^p) - \omega_1(\boldsymbol{\gamma}_1^p) \\ \vdots \\ \omega_n(\boldsymbol{\delta}_n^p) - \omega_n(\boldsymbol{\gamma}_n^p) \end{pmatrix} \quad (p = 1, 2, \ldots, p^*).$$

The linear program (7) has $2p^* \sum_{j=1}^n (\#\mathcal{A}_j - 2)$ inequality constraints, which can grow exponentially as the dimension and/or the degree of the polynomial system $\boldsymbol{f}(\boldsymbol{x})$ becomes larger, while the number of variables M and $\omega_j(\boldsymbol{a})$ ($\boldsymbol{a} \in \mathcal{A}_j,\ j = 1, 2, \ldots, n$), which is $1 + \sum_j^n \#\mathcal{A}_j$, is usually very small. Since a cutting plane method [15] can effectively solve such a linear program, it is applied to the linear program (7).

The paper [9] proposed an additional technique to reduce the ratio (6), which was also incorporated in PHoM. But the details are omitted here since they are not relevant to the description of PHoMpara in Section 3.

2.4 CMPSc - Tracing homotopy curves

Homotopy continuation starts from a known approximate solution $\boldsymbol{x}^0$ of $\boldsymbol{h}^p(\boldsymbol{x}, -s^0) = \boldsymbol{0}$ with a sufficiently large positive s^0 and traces a solution curve of $\boldsymbol{h}^p(\boldsymbol{x}, s) = \boldsymbol{0}$ numerically in the space $\mathbb{C}^n \times [-s^0, 0]$ by increasing the value of s to obtain a solution of the target polynomial system $\boldsymbol{h}^p(\boldsymbol{x}, 0) \equiv \boldsymbol{f}(\boldsymbol{x}) = \boldsymbol{0}$ at $s = 0$. In PHoM, we take $s^0 = 20$ so that the magnitude of $\exp(-s^0)$ becomes less than 1.0e-8. We employ a predictor-corrector method to trace the homotopy curves. See [7, 9] for more details.

2.5 Verify

After tracing all homotopy curves, it is necessary to verify that the solutions obtained cover a correct set of all isolated solutions of the polynomial system with a given accuracy. We test whether there is any pair of starting points leading to an equivalent zero of $\boldsymbol{f}(\boldsymbol{x})$ due to an accidental jump while numerically tracing the homotopy curves. If there is any such a pair of starting points, the corresponding curves are retraced with a smaller step size.

Suppose that approximate solutions $\hat{\boldsymbol{x}}^1, \hat{\boldsymbol{x}}^2, \ldots, \hat{\boldsymbol{x}}^r$ of $\boldsymbol{f}(\boldsymbol{x}) = \boldsymbol{0}$ are obtained by tracing the homotopy curves from starting points $(\tilde{\boldsymbol{x}}^1, -s^0), (\tilde{\boldsymbol{x}}^2, -s^0), \ldots, (\tilde{\boldsymbol{x}}^r, -s^0)$, respectively. Notice that $r \le \sum_{p=1}^{p^*} q_p$ since not all homotopy curves traced may converge to isolated

solutions and some curves may diverge. Here q_p is the number of initial solutions from $\boldsymbol{h}^p$. The module Verify checks whether there exists any pair of different j and k such that

$$\frac{\|\hat{\boldsymbol{x}}^j - \hat{\boldsymbol{x}}^k\|_\infty}{\max\{\|\hat{\boldsymbol{x}}^j\|_\infty + \|\hat{\boldsymbol{x}}^k\|_\infty, 1\}} \leq \mathsf{verifyAccu}. \tag{8}$$

If such a pair is found, then either $\hat{\boldsymbol{x}}^j$ or $\hat{\boldsymbol{x}}^k$ might have been computed incorrectly, or there might have been a jump from one homotopy curve to another homotopy curve while tracing them from two different starting points $(\tilde{\boldsymbol{x}}^j, -s^0)$ and $(\tilde{\boldsymbol{x}}^k, -s^0)$. Let

$$\tilde{J} \equiv \{j : \quad (8) \text{ holds for some } k \neq j\}.$$

Thus $\{(\tilde{\boldsymbol{x}}^j, -s^0) : j \in \tilde{J}\}$ denotes a set of the starting points of the homotopy curves which might have been traced incorrectly. The aim of the module Verify is to find the set $\tilde{J}$.

The computational work involved in finding a pair of j and k in (8) is reduced if the computed solutions $\hat{\boldsymbol{x}}^i$ $(i = 1, 2, \ldots, r)$ are sorted according to some norm. We employ the ∞ norm and use the relation

$$\frac{\|\hat{\boldsymbol{x}}^j - \hat{\boldsymbol{x}}^k\|_\infty}{\max\{\|\hat{\boldsymbol{x}}^j\|_\infty + \|\hat{\boldsymbol{x}}^k\|_\infty, 1\}} > \mathsf{verifyAccu} \quad \text{if} \quad \frac{\|\hat{\boldsymbol{x}}^j\|_\infty - \|\hat{\boldsymbol{x}}^k\|_\infty}{\max\{\|\hat{\boldsymbol{x}}^j\|_\infty + \|\hat{\boldsymbol{x}}^k\|_\infty, 1\}} > \mathsf{verifyAccu}.$$

Then, we need to check whether the inequality (8) holds only when

$$\frac{\|\hat{\boldsymbol{x}}^j\|_\infty - \|\hat{\boldsymbol{x}}^k\|_\infty}{\max\{\|\hat{\boldsymbol{x}}^j\|_\infty + \|\hat{\boldsymbol{x}}^k\|_\infty, 1\}} \leq \mathsf{verifyAccu}$$

is satisfied. Now assume that $\|\hat{\boldsymbol{x}}^1\|_\infty \geq \|\hat{\boldsymbol{x}}^2\|_\infty \geq \cdots \geq \|\hat{\boldsymbol{x}}^r\|_\infty$. Then, if we find a pair of indices j and i $(j < i)$ such that

$$\frac{\|\hat{\boldsymbol{x}}^j\|_\infty - \|\hat{\boldsymbol{x}}^i\|_\infty}{\max\{\|\hat{\boldsymbol{x}}^j\|_\infty + \|\hat{\boldsymbol{x}}^i\|_\infty, 1\}} > \mathsf{verifyAccu}.$$

then (8) never holds for any $k \geq i$. This considerably reduces the computational time to obtain the set $\tilde{J}$.

3 Parallel Implementation

The modules StartSystem, CMPSc, and Verify of PHoM are parallelized on parallel CPUs (PC clusters). Ninf [17] that implements the master-worker type communication was used to communicate among PC clusters. The advantage of using Ninf system is that it is an easy-to-use software and implementing communication between the master and workers is simple. After the master PC assigns a job to a worker PC, the master can not communicate with the worker except killing the job on the worker PC until the worker finishes the job and reports a computational result to the master.

It is obvious that more computation is necessary as the size of the polynomial system grows since the number of fine mixed cells and the number of homotopy curves increase. For example, with PHoM [7], while 951.7 cpu seconds were spent to find 2,173 isolated nonsingular solutions of noon-7 [16], 4396.7 cpu seconds were consumed to compute 6,545 isolated nonsingular solutions of noon-8. We notice a sharp increase in cpu time as the dimension of the polynomial system becomes larger. Parallelization has a clear advantage to deal with increasingly heavy computational load.

3.1 Parallel computation of fine mixed cells in StartSystem

The key point of parallelizing computing fine mixed cell is how the computation of fine mixed cells are distributed to the workers. We first describe the method proposed in [19]. In the enumeration tree (see Figure 1 as an example), let $\boldsymbol{N}^1, \boldsymbol{N}^2, \ldots, \boldsymbol{N}^{m_\ell} \in \mathcal{S}(\ell)$ be the m_ℓ nodes at an ℓ level. Let w^* denote the number of worker PC's in the PC cluster. The level ℓ is the starting level for distributing nodes to workers and is decided by the number of nodes at the level and the number of worker PC's. More precisely, we choose a level ℓ such that $\#\mathcal{S}(\ell-1) \leq w^* \leq \#\mathcal{S}(\ell)$, where $\#\mathcal{S}(\ell)$ indicates the number of nodes at ℓ level; we assume that such an ℓ exists in the subsequent discussion. From the construction of the enumeration tree, if $\boldsymbol{N}^i$ and $\boldsymbol{N}^{i'}$ are two different nodes from $\mathcal{S}(\ell)$, then there is no overlap in the nodes between the two subtrees whose root are $\boldsymbol{N}^i$ and $\boldsymbol{N}^{i'}$. This makes it possible to assign each $\boldsymbol{N}^i$ in $\mathcal{S}(\ell)$ $(1 \leq i \leq m_\ell)$ to a different worker. It assigns $\boldsymbol{N}^1$ to $\boldsymbol{N}^{w^*}$ of $\mathcal{S}(\ell)$ to worker 1 to worker w^*, respectively. Assigning $\boldsymbol{N}^i$ $(w^*+1 \leq i \leq m_\ell)$ is described as follows:

Algorithm 3.1 (Takeda et al.[19]) :

Step 1. Set $i \leftarrow w^* + 1$.

Step 2. Wait until one worker, say worker w*, stops. The master assigns* $\boldsymbol{N}^i$ *to worker* w.

Step 3. If $i < m_\ell$*, then set* $i \leftarrow i+1$ *and go to Step 2.*

In Figures 2 and 3, we illustrate Algorithm 3.1. Worker w $(1 \leq w \leq w^*)$ to which the

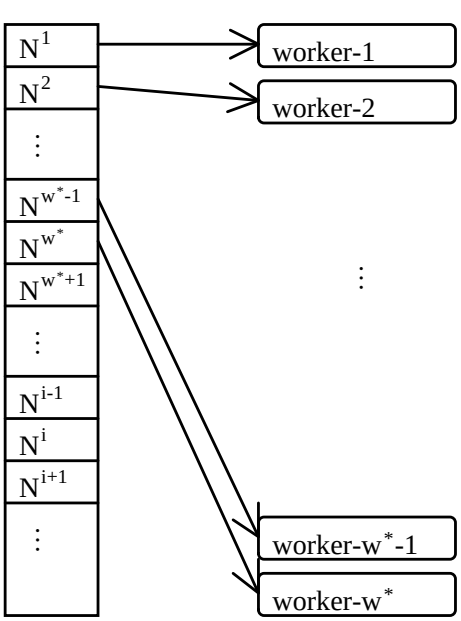

Figure 2: Initial assignment of nodes in $\mathcal{S}(\ell)$

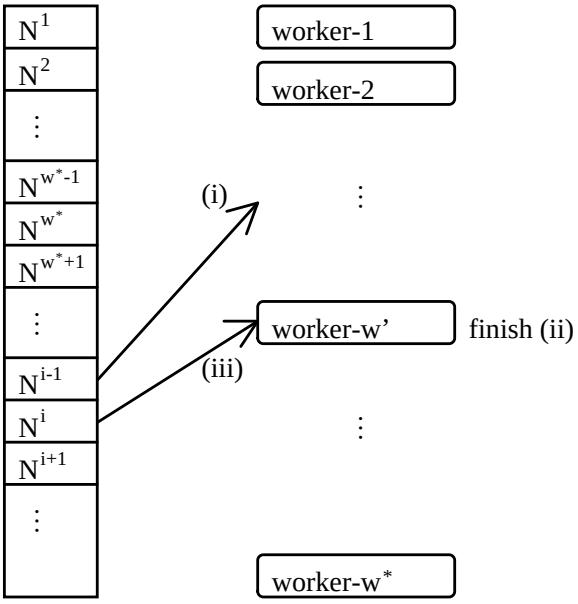

Figure 3: New nodes in $\mathcal{S}(\ell)$ are assigned after some workers finish their jobs.

master has assigned a node $\boldsymbol{N}^i$ executes Step 3 of Algorithm 2.1 for the subtree whose root

is $\boldsymbol{N}^i$. If worker w finishes processing the subtree, then it notifies the master fine mixed cells that have been found in the subtree.

Algorithm 3.1 is not necessarily efficient in parallel implementation. In particular, cpu time required by the enumerating process of each subtree of $\mathcal{S}(\ell)$ can be disparate. If one node $\boldsymbol{N}^i \in \mathcal{S}(\ell)$ $(1 \le i \le m_\ell)$ takes much longer time to enumerate its subtree than all the other nodes $\boldsymbol{N}^{i'} \in \mathcal{S}(\ell)$ $(i' \neq i)$, then all the workers with $\boldsymbol{N}^{i'} \in \mathcal{S}(\ell)(i' \neq i)$ need to wait for the worker with node $\boldsymbol{N}^i$ to finish the job, and the overall efficiency of parallelization decreases. We need an efficient way in assigning jobs to the workers.

We propose Algorithms 3.2 and 3.3 to improve the efficiency of parallel computation of fine mixed cells. The basic ideas of Algorithms 3.2 and 3.3 are that a worker taking long time for checking the feasibility of the linear system (5) at the nodes of an assigned subtree is forced to stop without completing the job, and the nodes that are not checked for the feasibility are redistributed to other workers. In the master-worker type environment, the master does not know whether a specific worker has a lot of job to do while the worker is working. The master just waits for the notification of finishing jobs from the workers. For redistribution, a number ub is sent to the workers so that the workers can notify the master after completing the feasibility test of ub nodes. We present algorithms for the master and workers as follows:

Algorithm 3.2 (Computing fine mixed cells in parallel at the master) :
Input: ℓ for the starting level; Output: M that contains the set of mixed mixed cells.

Step 1. Choose ub and set $M \leftarrow \emptyset$. Take out all elements $\boldsymbol{N}^1, \boldsymbol{N}^2, \ldots, \boldsymbol{N}^{m_\ell}$ of $\mathcal{S}(\ell)$ from the enumeration tree. Set $Q \leftarrow \{\boldsymbol{N}^{w^*+1}, \boldsymbol{N}^{w^*+2}, \ldots, \boldsymbol{N}^{m_\ell}\}$ and $W \leftarrow \{1, 2, \ldots, w^*\}$.

Step 2. For $\forall w \in W$, send $\boldsymbol{N}^w$ and ub to worker w, and request to compute the fine mixed cell contained in the subtree with $\boldsymbol{N}^w$ as a root.

Step 3. Wait until one worker stops. If a worker, say worker $\tilde{w}$, stops, then receive stack R. and $M_{\tilde{w}}$ from worker $\tilde{w}$. Add the elements of stack R to Q. Set $W \leftarrow W \backslash \{\tilde{w}\}$ and $M \leftarrow M \cup M_{\tilde{w}}$.

Step 4. If $Q = \emptyset$, then go to Step 4A. Otherwise, go to Step 4B.

Step 4A. If $W = \emptyset$, then output M and finish. Otherwise, go to Step 3.

Step 4B. If $\sharp W = w^*$, then go to Step 3. Otherwise, go to Step 5.

Step 5. Choose one node $\boldsymbol{N}'$ from Q and let ℓ' be such that $\boldsymbol{N}' \in \mathcal{S}(\ell')$. Select w' from $\{1, 2, \ldots, w^*\} \backslash W$. Send $\boldsymbol{N}'$ and ub to worker w', and request to compute the fine mixed cell contained in the subtree whose root if $\boldsymbol{N}'$. Set $Q \leftarrow Q \backslash \{\boldsymbol{N}'\}$ and $W \leftarrow W \cup \{w'\}$. Go to Step 4.

Algorithm 3.3 (Computing fine mixed cells in parallel at worker w) :
Input: $\boldsymbol{N}$ and ub; Output: stack R and M_w.

Step 1. Set the counter $co \leftarrow 0$. Let R be the empty stack, and push $\boldsymbol{N}$ to stack R.

Step 2. Set $co \leftarrow co + 1$. Pop one node $\overline{\boldsymbol{N}}$ from stack R, and let $\bar{\ell}$ be such that $\overline{\boldsymbol{N}} \in S(\bar{\ell})$. Set $R \leftarrow R \backslash \{\overline{\boldsymbol{N}}\}$.

Step 3. If the linear system (5) with $\boldsymbol{N} = \overline{\boldsymbol{N}}$ is feasible, then go to Step 4. Otherwise go to Step 5.

Step 4. If $\overline{\boldsymbol{N}}$ is a leaf node of the enumeration tree, i.e. $\bar{\ell} = n$, then set $M_w \leftarrow M_w \cup \{\overline{\boldsymbol{N}}\}$. Otherwise, take out all the children nodes $\overline{\boldsymbol{N}}^1, \overline{\boldsymbol{N}}^2, \ldots, \overline{\boldsymbol{N}}^{m_{\bar{\ell}+1}} \in S(\bar{\ell}+1)$ of $\overline{\boldsymbol{N}}$ and stack them to R.

Step 5. If $R = \emptyset$, then send empty stack R and M_w to the master and stop. Elseif $co \geq ub$, then send stack R and M_w to the master and stop. Otherwise, go to Step 2.

A number ub is used in both Algorithms 3.2 and 3.3 for the redistribution. Algorithm 3.3 performs the depth first search for the enumeration tree using stack R.

We illustrate Algorithms 3.2 and 3.3 for the case shown in Figure 1. Suppose that a worker receives $\boldsymbol{N}^0$ and $ub = 3$ as input ((i) of Figure 4). At Step 2, $co = 1$ and $\boldsymbol{N}^0$ is popped from stack R. Assume that the linear system (5) with $\boldsymbol{N} = \boldsymbol{N}^0$ is feasible. Then, $\boldsymbol{N}^1, \boldsymbol{N}^2, \boldsymbol{N}^3$ are pushed to R at Step 4 ((ii) of Figure 4). Next, at Step 2, $co = 2$ and $\boldsymbol{N}^1$ is popped from R. Suppose that the linear system (5) with $\boldsymbol{N} = \boldsymbol{N}^1$ is feasible. At Step 4, $\boldsymbol{N}^{11}, \boldsymbol{N}^{12}, \boldsymbol{N}^{13}$ are pushed to R ((iii) of Figure 4). At Step 2, $co = 3$ and $\boldsymbol{N}^{11}$ is popped from R. If the linear system (5) with $\boldsymbol{N} = \boldsymbol{N}^{11}$ is infeasible, $\boldsymbol{N}^{11}$ is removed from R ((iv) of Figure 4). At Step 4, $co = ub$ and R has $\{\boldsymbol{N}^{12}, \boldsymbol{N}^{13}, \boldsymbol{N}^2, \boldsymbol{N}^3\}$. At Step 5, the worker sends R and $M_w = \emptyset$ to the master. In Step 3 of Algorithm 3.2, the master receives R and add it to Q.

In Algorithm 3.3, workers send stack R to the master. Instead, we can modify Algorithms 3.2 and 3.3 such that only the top node of stack R is sent to the master and then the master rebuilds all the elements of stack R. The advantage of this approach is that the communication cost can be reduced since it does not depends on the length of stack R. The master can retrieve stack R from its top node if the nodes at every level $\mathcal{S}(\ell)$ of the enumeration tree have been arranged according to some order when the enumeration tree is constructed. More precisely, we know that the jth element N_j of $\boldsymbol{N} = (N_1, N_2, \ldots, N_\ell, \emptyset, \ldots, \emptyset) \in \mathcal{S}(\ell)$ $(j = 1, 2, \ldots, \ell)$ is determined by two elements $\boldsymbol{a}, \boldsymbol{a}' \in \mathcal{A}_j$. If we use an ordering, for instance, the lexicographical ordering, for the family of all pairs of two different $\boldsymbol{a}, \boldsymbol{a}' \in \mathcal{A}_j$, we have an order among the children of a node $\boldsymbol{N} = (N_1, N_2, \ldots, N_\ell, \emptyset, \ldots, \emptyset)$. When a worker sends only the top node of stack R, the master can construct R by examining the top node, based on the order of the nodes and the level of the top node. Algorithms 3.2 and 3.3 are presented without rebuilding stack R at the master with its top node, and sending only the top node of stack R at worker w, for the simplicity of description, however, we have implemented rebuilding R using its top node and the order of the nodes in the numerical experiments shown in Section 4.

3.2 Parallelization of balancing the coefficients of the homotopy parameter s in StartSystem

A cutting plane method applied to (7) is implemented in parallel. Recall that the linear program (7) has a very large number of inequality constraints and a relatively small number of variables for polynomial systems of large size. Let the number of inequality constraints of (7) be d^*, and $g_d(\boldsymbol{v})$ $(d \in D^* \equiv \{1, 2, \ldots, d^*\})$ represent the linear inequality constraints,

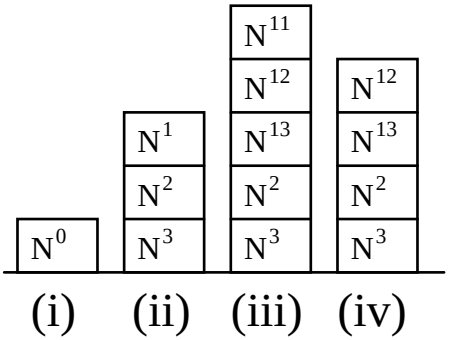

Figure 4: The stack at a worker

where $\boldsymbol{v}$ denotes a variable vector consisting of the real variables M and $\omega_j(\boldsymbol{a})$ ($\boldsymbol{a} \in \mathcal{A}_j,\ j = 1, 2, \ldots, n$). Then, we rewrite the linear program (7) as

$$\text{minimize} \ \ g_0(\boldsymbol{v}) \ \ \text{subject to} \ \ g_d(\boldsymbol{v}) \geq 0 \ (d \in D^*). \tag{9}$$

Since (9) has a large number of constraints, the cutting plane method first selects a certain small number of constraints. The size of the resulting subprogram with the selected constraints is smaller, as a result, the solution can be easily obtained. Next, the initial solution is tested for feasibility with the constraints that have not been selected in the beginning. We choose a set of constraints that the initial solution does not satisfy and add them to the initial subprogram. We then solve the new subprogram to find a solution. This process continues until a solution obtained from solving the subprogram satisfies all the constraints of (9).

Algorithm 3.4 (Solving the linear program (9) in parallel at the master) :
Input: the linear program (9); Output: $\boldsymbol{v}$.

Step 1. Partition D^* *into* D^*_w $(w = 1, 2, \ldots, w^*)$ *such that* $D^*_1 \cup D^*_2 \cup \cdots \cup D^*_{w^*} = D^*$ *and* $D^*_w \cap D^*_{w'} = \emptyset$ $(1 \leq w < w' \leq w^*)$. *Choose a set* $D^0 \subset D^*$. *Set* $ub \geq 1$ *and* $j \leftarrow 0$. *Send* D^*_w, g_d $(d \in D^*_w)$ *and* ub *to worker* w $(w = 1, 2, \ldots, w^*)$.

Step 2. Solve (9) with replacing D^* *by* D^j *to get an optimal solution* $\boldsymbol{v}^j$.

Step 3. Send $\boldsymbol{v}^j$ *to worker* w *and request to find a set* D^j_w *of constraint indices* $d \in D^*_w$ *of* g_d *that violates* $g_d(\boldsymbol{v}^j) \geq 0$ $(w = 1, 2, \ldots, w^*)$. *Wait until all* D^j_w $(w = 1, 2, \ldots, w^*)$ *are received from the workers.*

Step 4. If $D^j_w = \emptyset$ $(w = 1, 2, \ldots, w^*)$, *then output* $\boldsymbol{v} \leftarrow \boldsymbol{v}^j$ *and terminate. Otherwise, set* $D^{j+1} \leftarrow D^j \cup D^j_1 \cup D^j_2 \cup \cdots \cup D^j_{w^*}$ *and* $j \leftarrow j + 1$. *Go to Step 2.*

Algorithm 3.5 (Solving the linear program (9) in parallel at worker w**) :**
Input: $\boldsymbol{v}^j$*; Output a set of indices* $D^j_w \subset D^*_w$. *Initially, receive* D^*_w, ub *and* g_d $(d \in D^*_w)$ *from the master.*

Step 1. Set $D^j_w \leftarrow \emptyset, D' \leftarrow D^*_w$

Step 2. Select $\bar{d}$ *from* D' *and* $D' \leftarrow D' \backslash \{\bar{d}\}$. *If* $g_{\bar{d}}(\boldsymbol{v}^j) < 0$, *then* $D_w^j \leftarrow D_w^j \cup \{\bar{d}\}$

Step 3. If $\#D_w^j \geq ub$ *or* $D' = \emptyset$, *then send* D_w^j *to the master and stop. Otherwise go to Step 2.*

3.3 Parallelizing CMPSc

We obtain an isolated solution of the polynomial system by tracing a homotopy curve from $s = -s^0$ to $s = 0$ for some large positive number s^0 if the curve does not diverge. The job of tracing a homotopy curve is independent from tracing other curves. Hence, in parallelizing CMPSc, we only need to consider how we distribute the information on homotopy curves to the workers.

To trace a homotopy curve, we need the data of a polyhedral-linear function $\boldsymbol{h}^p$ described in (1) and the value of a known solution of the starting binomial system $\boldsymbol{h}^p(\boldsymbol{x}, -\infty) = 0$. Let $\boldsymbol{x}^{p1}, \boldsymbol{x}^{p2}, \ldots, \boldsymbol{x}^{pq_p}$ be the solutions of $\boldsymbol{h}^p(\boldsymbol{x}, -\infty) = 0$ $(p = 1, 2, \ldots, p^*)$. Then, with $(\boldsymbol{h}^p, \boldsymbol{x}^{p1})$, $(\boldsymbol{h}^p, \boldsymbol{x}^{p2}), \cdots, (\boldsymbol{h}^p, \boldsymbol{x}^{pq_p})$ $(p = 1, 2, \ldots, p^*)$, we have $\sum_{p=1}^{p^*} q_p$ homotopy curves to trace. Let $G^* = \{(\boldsymbol{h}^p, \boldsymbol{x}^{p1}), (\boldsymbol{h}^p, \boldsymbol{x}^{p2}), \cdots, (\boldsymbol{h}^p, \boldsymbol{x}^{pq_p})\}$ $(p = 1, 2, \ldots, p^*)$. One way of distributing the information on the homotopy curves to the workers is to assign $(\boldsymbol{h}^1, \boldsymbol{x}^{11})$ to worker 1, $(\boldsymbol{h}^1, \boldsymbol{x}^{12})$ to worker 2, and w^*th element of G^* to worker w^*. If we have more than w^* elements in G^*, then a worker that has finished the assigned tracing receives an element of G^* that is not sent to other worker from the master. The assignment occurs in the same way as computing fine mixed cells in parallel.

Notice that the master calls a function in Ninf communication library to send $(\boldsymbol{h}^p, \boldsymbol{x}^{pq})$ for some $q \in \{1, 2 \ldots, q_p\}$ and some $p \in \{1, 2, \ldots, p^*\}$ to a worker when it assigns the tracing job to the worker. The more elements G^* has, the more communication costs are needed to send the information. For example, noon-12 has more than 500,000 elements, and we need to call the Ninf library function to send the information more than 500,000 times. The extensive Ninf function calls to send elements of G^* to the workers should be avoided to save computational time.

The communication costs, cpu time consumed by Ninf function calls, are affected by the amount of information to send and the number of Ninf function calls. Reducing the amount of information to send and decreasing the number of Ninf function calls save total communication costs. We note that the master does not need to send the information of $\boldsymbol{h}^p$ repeatedly to trace the homotopy curves from the same homotopy system $\boldsymbol{h}^p$. If an element of G^* from the same homotopy system is to be sent to one worker, it can save communication cost of sending the information of $\boldsymbol{h}^p$. We mention that this may not improve the efficiency for all problems. For example, in reimer-n problems, there exists only one cell that contains all solutions of $\boldsymbol{h}^p(\boldsymbol{x}, -\infty) = 0$ $(p = 1)$, as a result, only one worker traces homotopy curves.

The inefficiency of sending the same information repeatedly can be decreased by choosing a positive number, ub, to balance the number of Ninf function calls and dividing the elements of G^* into some $k^* \leq ub$ groups such that the number of elements of G_k $(k = 1, 2, \ldots, k^*)$ satisfies the order of $\#G_1 \geq \#G_2 \geq \ldots \geq \#G_{k^*}$. We present Algorithm 3.6 as follows:

Algorithm 3.6 (Constructing groups of homotopy functions at the master) :
Input ub; Output k^* , $G_1, G_2, \ldots, G_{k^*}$.

Step 1. Construct $G_1, G_2, \ldots, G_{p^*}$ from G^* such that

$$\begin{aligned} G_1 &= \{(\boldsymbol{h}^1, \boldsymbol{x}^{11}), (\boldsymbol{h}^1, \boldsymbol{x}^{12}), \ldots, (\boldsymbol{h}^1, \boldsymbol{x}^{1q_1})\}, \\ G_2 &= \{(\boldsymbol{h}^2, \boldsymbol{x}^{21}), (\boldsymbol{h}^2, \boldsymbol{x}^{22}), \ldots, (\boldsymbol{h}^2, \boldsymbol{x}^{2q_2})\}, \\ &\vdots \\ G_{p^*} &= \{(\boldsymbol{h}^{p^*}, \boldsymbol{x}^{p^*1}), (\boldsymbol{h}^{p^*}, \boldsymbol{x}^{p^*2}), \ldots, (\boldsymbol{h}^{p^*}, \boldsymbol{x}^{p^*q_{p^*}})\}. \end{aligned}$$

Set $k^* \leftarrow p^*$.

Step 2. Sort G_k $(k = 1, 2, \ldots, k^*)$ by the number of elements in the descending order.

Step 3. If $k^* = ub$ or ($k^* < ub$ and $\#G_1 = 1$) then stop. If $k^* < ub$, go to Step 4. Otherwise go to Step 5.

Step 4. Divide G_1 into two sets G_1 and G_{k^*+1}. Set $k^* \leftarrow k^* + 1$. Go to Step 2.

Step 5. Merge G_{k^*-1} and G_{k^*} into G_{k^*-1}. Set $k^* \leftarrow k^* - 1$. Go to Step 2.

Algorithm 3.6 is based on the idea of sending similar number of elements of G^* to w^* workers. Now that we have $G_1, G_2, \ldots, G_{k^*}$, the master sends G_1 to worker 1, G_2 to worker 2, G_{w^*} to worker w^*. The remaining assignment is the same as computing fine mixed cells in parallel. Since G_k $(k = 1, 2, \ldots, k^*)$ are sorted by the number of elements in the descending order, G_1 has the largest number of curves and it takes longer to finish tracing the curves. We assign G_k's from 1 to k^* to the workers to minimize overall computing time.

3.4 Verifying solutions in parallel

In the module Verify, we compare whether the inequality (8) satisfies for $\forall j, k$ pair. A sorted list of the solutions $\hat{\boldsymbol{x}}^1, \hat{\boldsymbol{x}}^2, \ldots, \hat{\boldsymbol{x}}^r$ with respect to some norm is needed to efficiently execute the comparison. In parallel implementation, we use a parallel algorithm for quick sort. The quick sort algorithm that we used is based on Akl's algorithm [1], however, it differs in how it divides the list into smaller sublists. Once a sorted list is obtained, the comparison is performed by the master.

4 Numerical Results

We present numerical results of parallel implementation on eco-n(economic-n) [13], katsura-n [4], noon-n [16], and reimer-n [20] polynomials. The numerical experiments were done on the SDPA cluster in Tokyo Denki University, which provided 40 workers (each CPU is AMD Athlon 2.0GHz). The parameters that control the performance of PHoM are described in [7]. Table 1 shows the values of the parameters used for the numerical experiments.

The size of the problems that were solved by PHCpack, PHoM and PHoMpara is summarized in Table 2. It is evident from Table 2 that PHoMpara solves larger size problems than PHCpack and PHoM.

In Table 3, we show 'Total' = total cpu time (in seconds), 'StSy' = cpu time spent by StartSystem, 'Tr.all' = $\sum_{i=1}^{6}$ 'Tr.i' = cpu time by CMPSc, 'Tr.i' = cpu time for the ith

parameters	katsura	eco	noon	reimer
accINfVal	1.0×10^{-10}	1.0×10^{-10}	1.0×10^{-10}	1.0×10^{-10}
accInNewtonDir	1.0×10^{-8}	1.0×10^{-8}	1.0×10^{-8}	1.0×10^{-8}
beta	1	1	1	1
divMagOFx	1.0×10^{10}	1.0×10^{10}	1.0×10^{10}	50
dTauMax	0.5	0.5	0.5	0.5
minEigForNonsing	1.0×10^{-12}	1.0×10^{-12}	1.0×10^{-12}	1.0×10^{-12}
NewtonDirMax	0.1	0.1	0.1	0.1
predItMax	2000	2000	2000	2000
verifyAccu	1.0×10^{-4}	1.0×10^{-4}	1.0×10^{-4}	1.0×10^{-4}
dTauMaxRedRate	0.1	0.1	0.1	0.1
NewtonDirMaxRedRate	1.0	1.0	1.0	1.0
predItMaxExpRate	10	10	10	10
divMagOFxExpRate	1	1	1	1
MindTauMax	1.0×10^{-5}	1.0×10^{-5}	1.0×10^{-5}	1.0×10^{-8}
MinNewtonDirMax	1.0×10^{-5}	1.0×10^{-5}	1.0×10^{-5}	1.0×10^{-8}
MaxpredItMax	1000000	1000000	1000000	1000000
MaxVerifyIter	5	5	5	7

Table 1: Parameters

Problem	PHCpack	PHoM	PHoMpara
katsura	10	11	15
eco	8	12	16
noon	5	8	12
reimer	5	6	7

Table 2: The maximum size of problems that can solved by PHCpack, PHoM and PHoMpara

tracing of homotopy curves by CMPSc after $(i-1)$th application of Verify, and '#Sol.' = the number of isolated solutions obtained.

Table 3 shows that noon-12 has 531,417 isolated solutions. It took 49,458 cpu seconds and some homotopy curves were traced four times. In katsura-n problem, the number of solutions is doubled as n grows from 11 to 15. We observe that the increase of cpu time in StartSystem is bigger than that in CMPSc. On the other hand, cpu time spent by StartSystem in noon-n, as n changes from 9 to 12, does not increase much while CMPSc takes increasingly longer cpu time. Although the number of isolated solutions of reimer-7 is 2880, which is not very large compared with other problems, some of homotopy curves were traced repeatedly as many as 6 times. From the fact that the number of homotopy curves of reimer-n problem is larger than the number of isolated solutions, we see that many divergent curves exist. In PHoMpara, a curve is determined to be divergent if it satisfies a divergence criterion twice after CMPSc traces the curve repeatedly. Numerical results show that reimer-n problem had some difficulty in curve tracing. We see that the increase in the average and the maximum cpu time for the first iteration of curve tracing is not sharp with growing size of the problem, for instance, 1.0 seconds of the average cpu time for noon-10, 2.1 seconds for noon-11, and 3.3 seconds for noon-12.

4.1 Scalability

We tested PHoMpara in view of scalability by varying the number of workers from 1 to 40 for katsura-11, noon-10, and economic-14. The numerical results are shown in Table 4. The column 'ratio' indicates the ratio of cpu time obtained with multiple workers to that of a single worker. Table 4 contains the ratios of total cpu time, cpu time for StartSystem and cpu time for CMPSc. In the case of katsura-11, CMPSc shows high scalability with the ratio 2.01, 5.02, to 38.63 as the number of workers increases from 2 to 40, while the ratio of StartSystem does not become larger compared with that of tracing curves by CMPSc. The total cpu time using 2 to 40 workers displays the ratio of 2.00 to 29.82.

In noon-10, curve tracing by CMPSc obtains very high scalability as shown in the ratio from 2 to 37.66, whereas the ratio of StartSystem does not change a lot. Total cpu time shows the ratio of 2 to 37.12 for the number of workers 2 to 40.

The curve tracing part of economic-14 achieves a great improvement as shown in the column of the ratio from 1.95 with 2 workers to 33.58 with 40 workers. Figure 5 displays the ratios of total cpu time, cpu time for StartSystem and cpu time for CMPSc with increasing number of workers. We can see that cpu time for CMPSc decreases almost linearly as the number of workers grows.

In all three problems tested, we observe that a better speedup with more workers is accomplished in CMPSc than StartSystem. This comes from the characteristics of tracing curves by CMPSc, that is, tracing one curve is basically independent from tracing others.

4.2 Testing effectiveness of the redistribution technique

We tested the redistribution technique described in Section 3.1 with katsura-14, katsura-15, eco-15 and eco-16 using 40 workers.

Prob.	Total	StSy	CMPSc								♯Sol.
			Tr.all	Tr.1		Tr.2	Tr.3	Tr.4	Tr.5	Tr.6	
	cpu	cpu	cpu	cpu ♯cur	ave. max.	cpu ♯cur	cpu ♯cur	cpu ♯cur	cpu ♯cur	cpu ♯cur	
eco-14	626	388	238	238 4096	2.1 6.7	- -	- -	- -	- -	- -	4096
eco-15	2964	2300	664	637 8192	2.9 10.4	8 6	19 6	- -	- -	- -	8192
eco-16	12051	10470	1581	1566 16384	3.6 19.7	15 8	- -	- -	- -	- -	16384
noon-9	314	20	294	279 19665	0.5 1.3	6 36	9 12	- -	- -	- -	19665
noon-10	1797	27	1770	1752 59029	1.0 2.6	18 2	- -	- -	- -	- -	59029
noon-11	10225	39	10186	9848 177125	2.1 5.5	94 180	97 66	147 12	- -	- -	177125
noon-12	49458	78	49380	46737 531417	3.3 9.1	860 333	871 127	912 26	- -	- -	531417
katsura-11	160	58	102	102 2048	1.9 4.6	- -	- -	- -	- -	- -	2048
katsura-12	604	142	462	324 4096	3.2 8.2	7 16	17 8	114 2	- -	- -	4096
katsura-13	2137	531	1606	799 8196	3.7 10.0	9 8	20 7	25 2	753 2	- -	8192
katsura-14	4187	2231	1956	1907 16384	4.5 10.4	17 34	32 10	- -	- -	- -	16384
katsura-15	18964	13638	5326	5224 32768	6.1 21.5	45 61	57 25	- -	- -	- -	32768
reimer-5	18	1	17	4 720	0.1 0.2	4 577	4 173	5 1	- -	- -	144
reimer-6	110	3	107	31 5040	0.2 0.5	30 4464	38 1695	8 4	- -	- -	576
reimer-7	4229	9	4220	398 40320	0.3 1.5	399 37448	524 15512	1363 5299	978 608	558 8	2880

Table 3: Cpu time in seconds for StartSystem and CMPSc and the number of isolated solutions obtained. ♯cur: the number of curves.

Prob.	♯wks	Total		StSy		CMPSc	
		cpu.t	ratio	cpu.t	ratio	cpu.t	ratio
katsura-11	1	4550	1.00	637	1.00	3923	1.00
	2	2298	1.97	318	2.00	1980	1.98
	5	932	4.88	140	4.45	792	4.95
	10	482	9.43	87	7.32	395	9.93
	20	279	16.30	68	9.36	211	18.59
	40	160	28.43	58	10.98	102	38.46
noon-10	1	62672	1.00	66	1.00	62606	1.00
	2	31244	2.00	35	1.88	31209	2.00
	5	12379	5.06	23	2.86	12356	5.06
	10	6235	10.05	24	2.75	6211	10.07
	20	3195	19.61	24	2.75	3171	19.74
	40	1797	34.87	27	2.44	1770	35.37
eco-14	1	22653	1.00	13620	1.00	9033	1.00
	2	11317	2.00	6804	2.00	4513	2.00
	5	4536	4.99	2727	4.99	1809	4.99
	10	2292	9.88	1383	9.84	909	9.93
	20	1178	19.23	718	18.96	460	19.63
	40	626	36.18	388	35.10	238	37.95

Table 4: Cpu time and ratio with varying number of workers for katsura-11, noon-10 and eco-14. ♯wks: the number of workers.

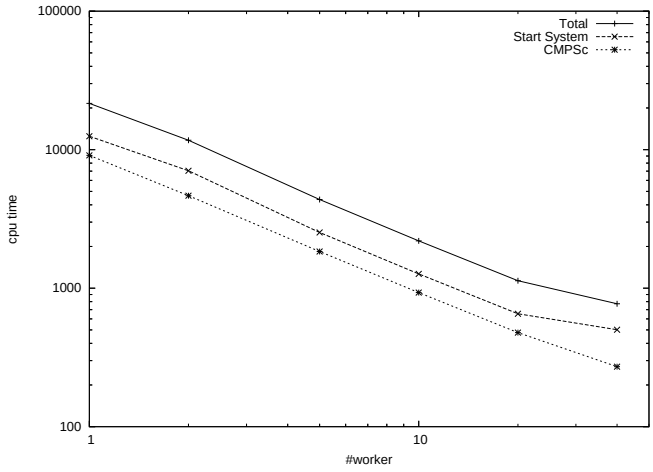

Figure 5: Cpu time with varying number of workers for economic-14.

#nodes in a subtree	katsura-14 #subtrees	eco-15 #subtree
$[1, 1.0 \times 10^5]$	13342	6214
$(1.0 \times 10^5, 2.0 \times 10^5]$	151	1024
$(2.0 \times 10^5, 4.0 \times 10^5]$	179	820
$(4.0 \times 10^5, 8.0 \times 10^5]$	218	596
$(8.0 \times 10^5, 16.0 \times 10^5]$	189	427
$(16.0 \times 10^5, 32.0 \times 10^5]$	194	271
$(32.0 \times 10^5, 64.0 \times 10^5]$	130	150
$(64.0 \times 10^5, 128.0 \times 10^5]$	88	37
$(128.0 \times 10^5, \infty)$	29	16
avr.cpu.t for a node	4.36×10^{-5}	3.64×10^{-5}

Table 5: The number of subtrees classified according to the number of their nodes and the average of computing time for a node.

When Algorithm 3.1 was applied to katsura-14 and economic-15 problems, the subtrees of the enumeration tree were processed by the workers. Each subtree has a different number of nodes (of the enumeration tree) where the linear system (5) is checked for feasibility. This number is counted by *co* in Algorithm 3.3. Note that the number is not known in advance when a subtree is assigned to a worker; some workers may find many nodes infeasible in the upper levels of the assigned subtree so that a smaller number of nodes in the lower levels are checked for feasibility, while other workers may need to check a larger number of the lower level nodes because most of the upper level nodes are feasible. If the numbers of nodes to be checked for feasibility in the subtrees of a problem vary in a wide range, then the computing time needed to process a subtree can differ very much. The workers performing the assigned job on the subtrees with relatively small numbers of the nodes to be checked should wait for other workers processing the subtrees with large numbers of the nodes to be examined for feasibility, increasing total computing time. This imbalance of computational load among workers increases as the difference in the numbers of nodes for feasibility check in the subtrees becomes large.

Table 5 shows the number of the subtrees classified according to the number of the nodes that were checked for feasibility in the subtrees. For example, katsura-14 has 13,342 subtrees whose number of the checked nodes are in the range of $(1, 1.0 \times 10^5]$, and economic-15 has 1024 subtrees whose number of the checked nodes are $(1.0 \times 10^5, 2.0 \times 10^5]$. The last row of Table 5 indicates the computational time spent by a worker devided by the number of the checked nodes. We can estimate how much computing time was spent for a worker processing a subtree in terms of 'the number of the checked nodes' $\times$ 'the average cpu time for checking the feasibility of a node'. As shown in Table 5, katsura-14 problem has 13,342 subtrees with at most 1.0×10^5 checked nodes and 29 subtrees with the number of the checked nodes greater than 128.0×10^5. This large difference causes an imbalance in computational load among the workers.

We used Algorithms 3.2 and 3.3 for katsura-14, katsura-15, economic-15, and economic-16 to observe the effectiveness of the redistribution technique. In the algorithms, we introduced a number *ub* to denote the maximum number of nodes to be checked, which was set

Problem	#wkr	without redistribution				with redistribution			
		Aver.	Max.	Min.	wRatio	Aver.	Max.	Min.	wRatio
katsura-14	10	8650	8804	8577	0.982	8684	8700	8675	0.998
	20	4339	4596	4251	0.944	4338	4380	4312	0.990
	40	2164	2462	2087	0.878	2183	2209	2172	0.988
katsura-15	40	13459	17097	12269	0.787	13461	13536	13388	0.994
eco-15	10	8862	8919	8843	0.993	8888	8907	8881	0.997
	20	4427	4506	4399	0.982	4432	4457	4411	0.994
	40	2262	2446	2238	0.927	2253	2276	2247	0.990
eco-16	40	10324	10632	10257	0.971	10393	10428	10377	0.996

Table 6: Effectiveness of the redistribution technique with numerical results

to 2.00×10^6 in the numerical tests. In katsura-14, the counter *co*, which add up to the total number of the checked nodes, reaches the value of *ub* in 87.2 seconds, computed by $(4.36 \times 10^{-5}) \times (2.00 \times 10^6)$. In economic-15, it resulted in 72.8 seconds. We observe that each worker spends less than 100 seconds to perform the assigned job using the value of *ub*.

The numerical results are displayed in Table 6. The average, minimum and maximum cpu time spent by the specified number of the workers are shown in the columns of Aver. Max. and Min.. Note that the maximum cpu time affects overall computing time. The column of wRatio means

$$\frac{\sum_{w=1}^{w^*} \tau_w}{w^* \times \max_{w=1}^{w^*} \tau_w}$$

where τ_w is cpu time consumed by worker w. Hence, the larger value of wRatio is, the smaller number of idle workers exist. For example, if wRatio is 1, then all the workers finish their jobs simultaneously, and there is no waiting for others to stop. If there is only one worker working and all the others are idle, then wRatio is $1/w^*$.

Table 6 shows that the redistribution technique provided better wRatio, nearly 1, for most test problems. As a result, the maximum cpu time was decreased by 10% for katsura-14 and economic-15, in particular, katsura-15 could be solved much faster with the redistribution. We confirm that the redistribution technique is effective in achieving high scalability by reducing the maximum cpu time.

5 Concluding remarks

The parallel implementation of polyhedral homotopy continuation methods has been presented with numerical results for selected test problems. We have shown that the isolated solutions of large polynomial systems could be found with the parallelization of PHoM, which otherwise was quite difficult.

PHoMpara is implemented under the master-worker type of PC clusters. Although this type of clusters has an advantage of one master managing workers efficiently, a drawback can be expected as the solutions of even larger polynomial systems are targeted. Since more workers are needed to do increasing amount of computation, the work load of the master to communicate with workers increases. This may become a bottleneck for overall efficiency.

In order to avoid such bottleneck, we need to introduce middle-masters between master and workers, which can distribute jobs to workers and communicate master. This will be a future subject of study.

For the problems with many divergent homotopy curves such as reimer-n, we have experienced some difficulties of judging whether curves diverged or not because some curves started to diverge very near $s = 0$. Those curves was retraced many times with smaller step sizes, affecting the overall efficiency. We need to develop more careful and efficient ways to determine the divergence of homotopy curves for the problems.

References

[1] S. G. Akl, "Optimal Parallel Algorithms for Computing Convex Hulls and for Sorting," *Computing.* **33** (1984) 1–11.

[2] E. Allgower and K. Georg, *Numerical continuation methods,* Springer-Verlag, 1990.

[3] D. N. Bernshtein, "The number of roots of a system of equations," *Functional Analysis and Appl.* **9** (1975) 183–185.

[4] W. Boege, R. Gebauer, and H. Kredel, "Some examples for solving systems of algebraic equations by calculating Groebner bases," *J. Symbolic Computation* **2** (1986) 83–98.

[5] C. B. Garcia and W. I. Zangwill, "Determining all solutions to certain systems of nonlinear equations," *Mathematics of Operations Research* **4** (1979) 1–14.

[6] T. Gao, T. Y. Li, J. Verschelde and M. Wu, "Balancing the lifting values to improve the numerical stability of polyhedral homotopy continuation methods," *Applied Mathematics and Computation* **114** (2000) 233–247.

[7] T. Gunji, S. Kim, M. Kojima, A. Takeda, K. Fujisawa and T. Mizutani, "PHoM – a Polyhedral homotopy continuation method for polynomial systems," *Computing.* **73** (2004) 55–77.

[8] B. Huber and B. Sturmfels, "A Polyhedral method for solving sparse polynomial systems," *Mathematics of Computation* **64** (1995) 1541–1555.

[9] S. Kim and M. Kojima, "Numerical stability of path tracing in polynomial homotopy continuation methods", *Computing*, **73**, 329-348 (2004).

[10] T. Y. Li, "Solving polynomial systems," *The mathematical intelligencer* **9** (1987) 33–39.

[11] T. Y. Li, "Solving polynomial systems by polyhedral homotopies", *Taiwan Journal of Mathematics* **3** (1999) 251–279.

[12] T. Y. Li and X. Li, "Finding Mixed Cells in the Mixed Volume Computation," *Foundation of Computational Mathematics* **1** (2001) 161–181.

[13] A. Morgan, *"Solving polynomial systems using continuation for engineering and scientific problems,"* Prentice-Hall, 1987.

[14] MPI: http://www.mpi-forum.org.

[15] G.L. Nemhauser and L.A.Wolsey, Integer and Combinatorial Optimization, Wiley-Interscience, 1988

[16] V. W. Noonberg, A neural network modeled by an adaptive Lotka-Volterra system, *SIAM J. Appl. Math.* **49** (1989) 1779–1792.

[17] M. Sato, H. Nakada, S. Sekiguchi, S. Matsuoka, U. Nagashima and H. Takagi, "Ninf: A Network based Information Library for a Global World-Wide Computing Infrastructure", HPCN'97 (LNCS-1225), pp. 491-502, 1997.

[18] B, Sturmfels, *Solving systems of polynomial equations*, CBMS Regional Conference Series in Mathematics, No 97, American Mathematical Society, 2002.

[19] A. Takeda, M. Kojima, and K. Fujisawa, "Enumeration of all solutions of a combinatorial linear inequality system arising from the polyhedral homotopy continuation method," *J. of Operations Society of Japan* **45** (2002) 64–82.

[20] C. Traverso, The PoSSo test suite examples. Available at http://www.inria.fr/saga/POL.

[21] J. Verschelde, The database of polynomial systems is in his web site: "http://www.math.uic.edu/∼jan/".

[22] J. Verschelde, P. Verlinden and R. Cools, "Homotopies exploiting Newton polytopes for solving sparse polynomial systems," *SIAM J. Numerical Analysis* **31** (1994) 915–930.

[23] J. Verschelde, "Homotopy continuation methods for solving polynomial systems," *Ph.D. thesis, Department of Computer Science, Katholieke Universiteit Leuven,* 1996.

[24] J. Verschelde, "Algorithm 795: PHCpack: A general-purpose solver for polynomial systems by homotopy continuation," *ACM Trans. Math. Softw.* **25** (1999) 251–276.