Journal of Symbolic Computation 79 (2017) 516–534

Contents lists available at ScienceDirect

Journal of Symbolic Computation

www.elsevier.com/locate/jsc

Mixed cell computation in Hom4PS-3

Tianran Chen [a,1], Tsung-Lin Lee [b,2], Tien-Yien Li [a,1,3]

[a] Department of Mathematics, Michigan State University, East Lansing, MI, USA
[b] Department of Applied Mathematics, National Sun Yat-sen University, Taiwan ROC

ARTICLE INFO

Article history:
Received 1 July 2015
Accepted 19 August 2015
Available online 16 July 2016

Keywords:
Mixed volume
Mixed cells
Polyhedral homotopy
Polynomial system
Parallel computing

ABSTRACT

This article presents recent efforts in improving the efficiency and scalability of the mixed cell computation step in the context of the Polyhedral Homotopy method.

Solving systems of polynomial equations is an important problem in applied mathematics. The Polyhedral Homotopy method is an important numerical method for this task. In this method, a necessary preprocessing step, known as the "mixed cell computation" problem has been the main bottleneck in the parallel efficiency and scalability. This article presents recent remarkable improvements in the parallel scalability of the algorithm that are applicable to a wide range of hardware architectures including multi-core systems, NUMA systems, computer clusters, and GPUs devices.

1. Introduction

The problem of solving systems of polynomial equations, or polynomial systems, has been, and will continue to be, a subject of great importance in both pure and applied mathematics. The need to solve polynomial systems arises naturally and frequently in various fields of science and engineering as documented in Allgower and Georg (2003), Morgan (2009), Sommese and Wampler (2005). In 1990s, a considerable research effort in Europe had been directed to the problem of solving polynomial systems in two consecutive major projects, PoSSo (**Po**lynomial **S**ystem **So**lving) and FRISCO

E-mail addresses: chentia1@msu.edu (T. Chen), leetsung@math.nsysu.edu.tw (T.-L. Lee), li@math.msu.edu (T.-Y. Li).

[1] Research supported in part by NSF under Grant DMS 11-15587.
[2] Research supported in part by MOST under Grant 103-2115-M-110-002.
[3] Research supported in part by NSFC under Grant 11171052.

http://dx.doi.org/10.1016/j.jsc.2016.07.017
0747-7171/

(**FR**amework for **I**ntegrated **S**ymbolic and numerical **CO**mputation), supported by the European Commission. Those research projects focused on the development of the well-established Gröbner basis methods within the framework of computer algebra. Their reliance on symbolic manipulation limits those methods to relatively small problems. In 1977, Garcia and Zangwill (1979) and Drexler (1977) independently discovered that the *homotopy continuation* could be used to find the full set of isolated solutions to a polynomial system numerically. In the ensuing years, the method saw extensive development and sparked the creation of the exciting new field of *Numerical Algebraic Geometry* (Sommese and Wampler, 1996). It has become one of the most reliable and efficient class of numerical methods for finding the full set of isolated solutions to a general polynomial system. There are many software implementing this method, including Bertini (Bates et al., 2013), Hom4PS-2.0 (Lee et al., 2008), HOMPACK (Morgan et al., 1989; Watson et al., 1987), NAG4M2 (Leykin, 2009; Leykin, 2011), PHCpack (Verschelde, 1999), etc. See Allgower and Georg (2003), Attardi and Traverso (1995), Li (2003), Morgan (2009), Sommese and Wampler (2005) for basic references.

One important family of homotopy for solving polynomial systems is the *polyhedral homotopy* method initiated by B. Huber and B. Sturmfels (1995) which has a distinctive advantage in solving sparse polynomial systems. The method has been successfully implemented in the software packages PHCpack (Verschelde, 1999) developed by J. Verschelde at University of Illinois at Chicago Circle, and Hom4PS-2.0 (Lee et al., 2008) developed by the authors, etc. The efficiency and reliability in the applications of Hom4PS-2.0 in many problems in mathematics, science, and engineering have been documented in Bozóki et al. (2015), Lee et al. (2008), Lee and Santoprete (2009), Li (2003), Li and Tsai (2009), Mehta (2009), Mehta (2011a), Mehta (2011b), Mehta et al. (2012), Mehta et al. (2014), Mehta et al. (2009). Based on Hom4PS-2.0, we have built a new and further improved numerical solver for polynomial systems — Hom4PS-3 — around the same core mathematical algorithms.

In applications from science and engineering, there is no shortage in the demand of solving larger and larger polynomial systems. Homotopy continuation methods, in general, are particularly suited to handle these large polynomial systems due to their *pleasantly parallel* nature: each isolated solution is computed independently of the others. However, in the polyhedral homotopy method which plays the main role in our Hom4PS-3 package, an important preprocessing step, the "enumeration of mixed cells" (Chen et al., 2014; Emiris and Canny, 1995; Gao and Li, 2000; Gao and Li, 2003; Gao et al., 1999; Gao et al., 2005; Huber and Sturmfels, 1995; Lee and Li, 2011; Li and Li, 2001; Mizutani and Takeda, 2008; Mizutani et al., 2007; Takeda et al., 2000; Verschelde et al., 1996) (which will be discussed in §3) is a major bottleneck in terms of its scalability in parallel computation and can severely limit its ability in handling large polynomial systems. A fully parallel algorithm for this step based on a graph-theoretic formulation has been attempted by the authors in Chen et al. (2014) to solve this problem. Encouraged by the results, Hom4PS-3 integrates this parallel algorithm and consolidates many important improvements developed since Chen et al. (2014). Of particular importance are the improvements in the parallel scalability that enables the computation of problems of much larger scale. In this article, we outline the underlying parallel algorithm and describe the technical details of the recent improvements. Impressive results of a few experimental features are also presented with the hope to inspire future investigations.

The rest of the article is structured as follows: First, to motivate the rest of the discussions, we briefly review the polyhedral homotopy continuation method in §2. The mathematical description of mixed cell enumeration process is given in §3. A fully parallel algorithm based on a graph-theoretic formulation is outlined in §4. Technical but critical aspects of adapting the parallel algorithm to different computing hardware architectures are discussed in §4.1, §4.2, §4.3 and §4.4.

2. Polyhedral homotopy

In the middle of 1990s, a major computational advancement has emerged in solving polynomial systems by the homotopy continuation method. Taking advantage of the Bernstein's theorem (Bernshtein, 1975), the *polyhedral homotopy method* was introduced by B. Huber and B. Sturmfels (1995). The polyhedral homotopy method is designed to find all complex isolated zeros of a "square" polynomial system of the form

$$P(x_1,\ldots,x_n)=P(\mathbf{x})=\begin{cases} p_1(\mathbf{x}) &= \sum_{\mathbf{a}\in S_1} c_{1,\mathbf{a}}\,\mathbf{x}^{\mathbf{a}} \\ &\vdots \\ p_n(\mathbf{x}) &= \sum_{\mathbf{a}\in S_n} c_{n,\mathbf{a}}\,\mathbf{x}^{\mathbf{a}} \end{cases} \tag{1}$$

where $\mathbf{x}=(x_1,\ldots,x_n)$, $\mathbf{a}=(a_1,\ldots,a_n)^\top\in\mathbb{N}_0^n=(\mathbb{N}\cup\{0\})^n$, and $\mathbf{x}^{\mathbf{a}}=x_1^{a_1}\cdots x_n^{a_n}$. Here S_j is the *support* of p_j: a finite subset of $\mathbb{N}_0^n$ defined by the exponent vectors of the monomials appearing in the polynomial p_j. For fixed supports $S_1,\ldots,S_n$, it is a basic fact in algebraic geometry that for generic choices of the complex coefficients $c_{j,\mathbf{a}}\in\mathbb{C}^*=\mathbb{C}\setminus\{0\}$ the number of isolated solutions of the system $P(\mathbf{x})=\mathbf{0}$ in $(\mathbb{C}^*)^n$ is a constant. The word "generic" here can be understood as the existence of an open and dense subset of coefficients among all possible complex coefficients for which the previous statement holds. For instance, if the coefficients are chosen at random (using any continuous probability density), then the statement holds with probability one. Moreover this fixed number of isolated solutions also serves as an upper bound on the number of isolated solutions $P(\mathbf{x})=\mathbf{0}$ can have in $(\mathbb{C}^*)^n$ among all choices of coefficients.

In Bernshtein (1975), this number (sometimes called the *BKK bound*) was shown to be a particular *mixed volume* (Minkowski, 1911): For convex polytopes $\mathcal{Q}_1,\ldots,\mathcal{Q}_k\subset\mathbb{R}^k$, let $\lambda_1\mathcal{Q}_1,\ldots,\lambda_k\mathcal{Q}_k$ represent their scaled version, by factors of positive $\lambda_1,\ldots,\lambda_k$ respectively. The *Minkowski sum* $\lambda_1\mathcal{Q}_1+\cdots+\lambda_k\mathcal{Q}_k$ is also a convex polytope. It can be shown that the volume $\mathrm{Vol}_k(\lambda_1\mathcal{Q}_1+\cdots+\lambda_k\mathcal{Q}_k)$ in $\mathbb{R}^k$ is a homogeneous polynomial in $\lambda_1,\ldots,\lambda_k$ when λ_i's are nonnegative. The *mixed volume*, denoted by $\mathrm{MVol}(\mathcal{Q}_1,\ldots,\mathcal{Q}_k)$, is defined to be the coefficient of $\lambda_1\times\lambda_2\times\cdots\times\lambda_k$ in this polynomial.

Theorem 1 *(Bernshtein (1975)). The number of isolated solutions of $P(\mathbf{x})=\mathbf{0}$ in $(\mathbb{C}^*)^n$ is less than or equal to the mixed volume of the convex hull of the supports of $P(\mathbf{x})$, that is,*

$$\mathrm{MVol}(\mathrm{conv}\,S_1,\ldots,\mathrm{conv}\,S_n).$$

Moreover, this bound is exact for generic choices of the coefficients of $P(\mathbf{x})$.

Based on an alternative proof of the Bernshtein's Theorem from a combinatorial point of view, B. Huber and B. Sturmfels (1995) constructed a nonlinear "polyhedral homotopy" for solving polynomial systems which we shall outline below.

First, the focus is restricted to solving a polynomial system $P(\mathbf{x})=\mathbf{0}$ as in (1) with "generic" (nonzero) complex coefficients $c_{j,\mathbf{a}}\in\mathbb{C}^*$. In this case, to solve $P(\mathbf{x})=\mathbf{0}$ in (1), consider, with a new variable t, the homotopy

$$H(x_1,\ldots,x_n,t)=H(\mathbf{x},t)=\begin{cases} h_1(\mathbf{x},t)=\sum_{\mathbf{a}\in S_1} c_{1,\mathbf{a}}\,\mathbf{x}^{\mathbf{a}}t^{\omega_1(\mathbf{a})} \\ \quad\vdots \\ h_n(\mathbf{x},t)=\sum_{\mathbf{a}\in S_n} c_{n,\mathbf{a}}\,\mathbf{x}^{\mathbf{a}}t^{\omega_n(\mathbf{a})} \end{cases} \tag{2}$$

with "lifting" functions $\omega_1,\ldots,\omega_n$, where each $\omega_k: S_k\to\mathbb{Q}$ has randomly chosen image. Note that when $t=1$, $H(\mathbf{x},1)=P(\mathbf{x})$ is exactly the target system. By a standard application of Generalized Sard's Theorem (Abraham and Robbin, 1967) and the Implicit Function Theorem, the zero set of $H(\mathbf{x},t)$ defines paths (known as *homotopy paths*) emanating from solutions of $P(\mathbf{x})=H(\mathbf{x},1)=\mathbf{0}$ and continue toward the solutions at $t=0$, and all these homotopy paths are smooth with no bifurcations. (The genericity of the coefficients and the randomness of the images of the ω_i are critical here.) Moreover, since for any fixed t value, $H(\mathbf{x},t)$, as a polynomial system in $\mathbf{x}$ only, has the same supports $S_1,\ldots,S_n$, the number of its roots agree with the BKK bound in Theorem 1. Therefore, the number of homotopy paths defined by (2) is precisely the BKK bound of $P(\mathbf{x})$.

However, at $t=0$, the solutions of the starting system $H(\mathbf{x},0)=\mathbf{0}$ cannot be identified since the terms of each h_i either vanish or blow up. This obstacle can be surmounted by the following device.

For $\mathbf{a} \in S_k$, write $\hat{\mathbf{a}} = (\mathbf{a}, \omega_k(\mathbf{a}))$. In Huber and Sturmfels (1995), it was shown that there exists at least one $\hat{\boldsymbol{\alpha}} = (\boldsymbol{\alpha}, 1) \in \mathbb{R}^{n+1}$ with $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_n)$ and an n-tuples of pairs $\{\mathbf{a}_1, \mathbf{a}_1'\} \subset S_1, \ldots, \{\mathbf{a}_n, \mathbf{a}_n'\} \subset S_n$ such that for each $k = 1, \ldots, n$

$$\langle \hat{\mathbf{a}}_k, \hat{\boldsymbol{\alpha}} \rangle = \langle \hat{\mathbf{a}}_k', \hat{\boldsymbol{\alpha}} \rangle < \langle \hat{\mathbf{a}}, \hat{\boldsymbol{\alpha}} \rangle \quad \text{for all } \mathbf{a} \in S_k \setminus \{\mathbf{a}_k, \mathbf{a}_k'\} \tag{3}$$

and

$$\kappa_{\boldsymbol{\alpha}} := \left| \det \begin{bmatrix} \mathbf{a}_1 - \mathbf{a}_1' & \cdots & \mathbf{a}_n - \mathbf{a}_n' \end{bmatrix} \right| > 0.$$

Here $\langle \cdot, \cdot \rangle$ stands for the standard inner product in Euclidean space. While the possible $\boldsymbol{\alpha}$ and $\kappa_{\boldsymbol{\alpha}}$ clearly depends on the liftings $\boldsymbol{\omega} = (\omega_1, \ldots, \omega_n)$, if $\mathcal{T}$ is the collection of all such distinct $\boldsymbol{\alpha}$'s, then

$$\sum_{\boldsymbol{\alpha} \in \mathcal{T}} \kappa_{\boldsymbol{\alpha}}$$

is independent of the choice of the lifting functions $\omega_1, \ldots, \omega_k$. In fact, this number agrees with the number of isolated solutions of the system $P(\mathbf{x}) = 0$ in $(\mathbb{C}^*)^n$ (counting multiplicities) known as the BKK bound mentioned before. The elements in the collection $\mathcal{T}$ along with their corresponding set of pairs $\{\mathbf{a}_1, \mathbf{a}_1'\} \subset S_1, \ \ldots, \ \{\mathbf{a}_n, \mathbf{a}_n'\} \subset S_n$ are known as the *mixed cells*. When there is no ambiguity, we will simply use the vector $\boldsymbol{\alpha} \in \mathcal{T}$ to represent the corresponding mixed cell. These mixed cells play a crucial role in the construction of polyhedral homotopy. Their computation will be discussed in §3.

For a fixed mixed cell, that is, an $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_n)$ in $\mathcal{T}$ along with its associated set of pairs $\{\mathbf{a}_1, \mathbf{a}_1'\} \subset S_1, \ldots, \{\mathbf{a}_n, \mathbf{a}_n'\} \subset S_n$, let

$$\beta_i := \langle \hat{\mathbf{a}}_i, \hat{\boldsymbol{\alpha}} \rangle = \langle \hat{\mathbf{a}}_i', \hat{\boldsymbol{\alpha}} \rangle \tag{4}$$

for $i = 1, \ldots, n$. Then by (3), for every $i = 1, \ldots, n$,

$$\beta_i < \langle \hat{\mathbf{a}}, \hat{\boldsymbol{\alpha}} \rangle \quad \text{for all } \mathbf{a} \in S_i \setminus \{\mathbf{a}_i, \mathbf{a}_i'\}. \tag{5}$$

By the change of variables $\mathbf{x} = t^{\boldsymbol{\alpha}} \cdot \mathbf{y}$, that is, for $\mathbf{y} = (y_1, \ldots, y_n)$

$$\begin{cases} x_1 = t^{\alpha_1} y_1 \\ \vdots \ \ \vdots \\ x_n = t^{\alpha_n} y_n, \end{cases} \tag{6}$$

we have, for $\mathbf{a} = (a_1, \ldots, a_n) \in S_k$ and $\hat{\mathbf{a}} = (\mathbf{a}, \omega_k(\mathbf{a}))$,

$$\begin{aligned} \mathbf{x}^{\mathbf{a}} t^{\omega_k(\mathbf{a})} &= x_1^{a_1} \cdots x_n^{a_n} t^{\omega_k(\mathbf{a})} \\ &= (t^{\alpha_1} y_1)^{a_1} \cdots (t^{\alpha_n} y_n)^{a_n} t^{\omega_k(\mathbf{a})} \\ &= y_1^{a_1} \cdots y_n^{a_n} t^{a_1\alpha_1 + \cdots + a_n\alpha_n + \omega_k(\mathbf{a})} \\ &= \mathbf{y}^{\mathbf{a}} t^{\langle (\mathbf{a}, \omega_k(\mathbf{a})), (\boldsymbol{\alpha}, 1) \rangle} \\ &= \mathbf{y}^{\mathbf{a}} t^{\langle \hat{\mathbf{a}}, \hat{\boldsymbol{\alpha}} \rangle} \end{aligned}$$

with $\hat{\boldsymbol{\alpha}} = (\boldsymbol{\alpha}, 1)$. Substituting the result into $H(\mathbf{x}, t)$ in (2), it follows that

$$\bar{H}^{\boldsymbol{\alpha}}(\mathbf{y}, t) := H(t^{\boldsymbol{\alpha}} \cdot \mathbf{y}, t) = \begin{cases} \bar{h}_1^{\boldsymbol{\alpha}}(\mathbf{y}, t) := h_1(t^{\boldsymbol{\alpha}} \cdot \mathbf{y}, t) = \sum_{\mathbf{a} \in S_1} c_{1,\mathbf{a}} \mathbf{y}^{\mathbf{a}} t^{\langle \hat{\mathbf{a}}, \hat{\boldsymbol{\alpha}} \rangle} \\ \vdots \\ \bar{h}_n^{\boldsymbol{\alpha}}(\mathbf{y}, t) := h_n(t^{\boldsymbol{\alpha}} \cdot \mathbf{y}, t) = \sum_{\mathbf{a} \in S_n} c_{n,\mathbf{a}} \mathbf{y}^{\mathbf{a}} t^{\langle \hat{\mathbf{a}}, \hat{\boldsymbol{\alpha}} \rangle}. \end{cases}$$

Though the above expression may contain positive or negative powers of t, the minimum exponents of t in each $\bar{h}_i^{\boldsymbol{\alpha}}$ is given by β_i (5). Therefore, by constructing

$$H^{\boldsymbol{\alpha}}(\mathbf{y},t) := \begin{cases} h_1^{\boldsymbol{\alpha}}(\mathbf{y},t) := t^{-\beta_1}\bar{h}_1^{\boldsymbol{\alpha}}(\mathbf{y},t) = \sum_{\mathbf{a}\in S_1} c_{1,\mathbf{a}}\,\mathbf{y}^{\mathbf{a}}\,t^{\langle\hat{\mathbf{a}},\hat{\boldsymbol{\alpha}}\rangle-\beta_1} \\ \quad\vdots \\ h_n^{\boldsymbol{\alpha}}(\mathbf{y},t) := t^{-\beta_n}\bar{h}_n^{\boldsymbol{\alpha}}(\mathbf{y},t) = \sum_{\mathbf{a}\in S_n} c_{n,\mathbf{a}}\,\mathbf{y}^{\mathbf{a}}\,t^{\langle\hat{\mathbf{a}},\hat{\boldsymbol{\alpha}}\rangle-\beta_n}, \end{cases} \tag{7}$$

each component $h_i^{\boldsymbol{\alpha}}$ of $H^{\boldsymbol{\alpha}}$ has exactly two terms having no powers of t corresponding to $\mathbf{a}_i$ and $\mathbf{a}_i'$ respectively while all other terms have positive powers of t. That is,

$$H^{\boldsymbol{\alpha}}(\mathbf{y},t) := \begin{cases} c_{1,\mathbf{a}_1}\mathbf{y}^{\mathbf{a}_1} + c_{1,\mathbf{a}_1'}\mathbf{y}^{\mathbf{a}_1'} + \sum_{\mathbf{a}\in S_1\setminus\{\mathbf{a}_1,\mathbf{a}_1'\}} c_{1,\mathbf{a}}\,\mathbf{y}^{\mathbf{a}}\,t^{\langle\hat{\mathbf{a}},\hat{\boldsymbol{\alpha}}\rangle-\beta_1} \\ \quad\vdots \\ c_{n,\mathbf{a}_1}\mathbf{y}^{\mathbf{a}_n} + c_{n,\mathbf{a}_1'}\mathbf{y}^{\mathbf{a}_n'} + \sum_{\mathbf{a}\in S_n\setminus\{\mathbf{a}_n,\mathbf{a}_n'\}} c_{n,\mathbf{a}}\,\mathbf{y}^{\mathbf{a}}\,t^{\langle\hat{\mathbf{a}},\hat{\boldsymbol{\alpha}}\rangle-\beta_n}. \end{cases}$$

This $H^{\boldsymbol{\alpha}}(\mathbf{y},t)$ is known as the *polyhedral homotopy induced by the mixed cell* $\boldsymbol{\alpha}$. Note that the desired property that $H^{\boldsymbol{\alpha}}(\mathbf{y},1)\equiv P(\mathbf{y})$ still holds, and hence $H^{\boldsymbol{\alpha}}$ still represents a deformation of the target system $P(\mathbf{x})=\mathbf{0}$. However, unlike the original H in (2) which becomes meaningless at $t=0$, here, when $t=0$, $H^{\boldsymbol{\alpha}}(\mathbf{y},0)=\mathbf{0}$ is the binomial system of equations

$$\begin{cases} c_{1,\mathbf{a}_1}\mathbf{y}^{\mathbf{a}_1} + c_{1,\mathbf{a}_1'}\mathbf{y}^{\mathbf{a}_1'} = 0 \\ \qquad\vdots \qquad\qquad\quad \vdots \\ c_{n,\mathbf{a}_n}\mathbf{y}^{\mathbf{a}_n} + c_{n,\mathbf{a}_n'}\mathbf{y}^{\mathbf{a}_n'} = 0 \end{cases}. \tag{8}$$

It is known that the solutions of this binomial systems in $(\mathbb{C}^*)^n$ are all isolated and nonsingular. The total number of these solutions is exactly $\kappa_{\boldsymbol{\alpha}} = |\det[\mathbf{a}_1-\mathbf{a}_1' \ \cdots\ \mathbf{a}_n-\mathbf{a}_n']|$ which is an integer by the Leibniz's determinant formula. Moreover, these solutions can be located accurately and efficiently via numerical methods (Chen and Li, 2014; Kahle, 2010; Li, 2003).

The nonsingular solutions obtained by solving (8) are then used as the starting points for following the homotopy paths $\mathbf{y}(t)$ defined by $H^{\boldsymbol{\alpha}}(\mathbf{y},t)=\mathbf{0}$, for which $H^{\boldsymbol{\alpha}}(\mathbf{y}(t),t)=\mathbf{0}$ from $t=0$ to $t=1$. Note that the change of variables $\mathbf{x}=t^{\boldsymbol{\alpha}}\cdot\mathbf{y}$ in (6) yields $\mathbf{x}\equiv\mathbf{y}$ at $t=1$. Therefore, each end point $\mathbf{y}(1)$ at $t=1$ of the homotopy path $\mathbf{y}(t)$ of $H^{\boldsymbol{\alpha}}(\mathbf{y},t)=\mathbf{0}$ is also an end point $\mathbf{x}(1)$ of the homotopy path $\mathbf{x}(t)$ defined by $H(\mathbf{x},t)=\mathbf{0}$ given in (2) which, in turn, provides a solution of the target system $P(\mathbf{x})=\mathbf{0}$ in (1). Altogether it yields $\kappa_{\boldsymbol{\alpha}}$ of the isolated solutions of $P(\mathbf{x})=\mathbf{0}$ in $(\mathbb{C}^*)^n$ along this route. In Huber and Sturmfels (1995), it was shown that as one follows the homotopy paths defined by $H^{\boldsymbol{\alpha}}(\mathbf{y},t)=\mathbf{0}$ for all individual $\boldsymbol{\alpha}\in\mathcal{T}$, one obtains all (isolated) solutions of $P(\mathbf{x})=\mathbf{0}$ in $(\mathbb{C}^*)^n$, justifying, indeed, the BKK bound agrees with $\sum_{\boldsymbol{\alpha}\in\mathcal{T}}\kappa_{\boldsymbol{\alpha}}$. We summarize the above derivation into the following theorem.

Theorem 2 *(Polyhedral homotopy (Huber and Sturmfels, 1995)). Assume the target system $P(\mathbf{x})=\mathbf{0}$ has generic coefficients. For each mixed cell $\boldsymbol{\alpha}\in\mathcal{T}$, along with its corresponding pairs $\{\mathbf{a}_1,\mathbf{a}_1'\}\subset S_1,\ldots,\{\mathbf{a}_n,\mathbf{a}_n'\}\subset S_n$, let $\beta_i=\langle\hat{\mathbf{a}}_i,\hat{\boldsymbol{\alpha}}\rangle=\langle\hat{\mathbf{a}}_i',\hat{\boldsymbol{\alpha}}\rangle$. Then the induced homotopy*

$$H^{\boldsymbol{\alpha}}(\mathbf{y},t) = \begin{cases} t^{-\beta_1}h_1(t^{\boldsymbol{\alpha}}\mathbf{y},t) = \sum_{\mathbf{a}\in S_1} c_{1,\mathbf{a}}\,\mathbf{y}^{\mathbf{a}}\,t^{\langle\hat{\mathbf{a}},\hat{\boldsymbol{\alpha}}\rangle-\beta_1} \\ \quad\vdots \\ t^{-\beta_n}h_n(t^{\boldsymbol{\alpha}}\mathbf{y},t) = \sum_{\mathbf{a}\in S_n} c_{n,\mathbf{a}}\,\mathbf{y}^{\mathbf{a}}\,t^{\langle\hat{\mathbf{a}},\hat{\boldsymbol{\alpha}}\rangle-\beta_n} \end{cases}$$

defines smooth homotopy paths with no bifurcations in $(\mathbb{C}^*)^n\times[0,1]$, and the number of paths is exactly

$$\kappa_{\boldsymbol{\alpha}} := \left|\det\left[\,\mathbf{a}_1-\mathbf{a}_1' \ \ \ldots\ \ \mathbf{a}_n-\mathbf{a}_n'\,\right]\right|. \tag{9}$$

The total number of paths defined by all $H^{\boldsymbol{\alpha}}$ for $\boldsymbol{\alpha}\in\mathcal{T}$ is

$$\sum_{\boldsymbol{\alpha}\in\mathcal{T}}\kappa_{\boldsymbol{\alpha}} = \mathrm{MVol}(\mathrm{conv}\,S_1,\ldots,\mathrm{conv}\,S_n) \tag{10}$$

agreeing with the number of isolated solutions of $P(\mathbf{x})=\mathbf{0}$ in $(\mathbb{C}^*)^n$. Moreover, every such isolated solution lies at the end of a unique path defined by a $H^{\boldsymbol{\alpha}}=\mathbf{0}$ for some $\boldsymbol{\alpha}\in\mathcal{T}$.

Remark 1. Recall that for polynomial systems having generic coefficients, the BKK bound (Theorem 1) is exact. For systems with specific (i.e. not generic in the above sense) coefficients, the number of isolated solutions in $(\mathbb{C}^*)^n$ may be less than the BKK bound and hence less than the number of paths defined by (2).

Remark 2. Though the above procedure only targets the isolated solutions in $(\mathbb{C}^*)^n$ of a polynomial system $P(\mathbf{x}) = \mathbf{0}$ with generic coefficients, these limitations can be resolved by the following modifications provided in Li et al. (1989), Li and Wang (1996), Morgan and Sommese (1989). Leaving aside the technical statements, it is sufficient to augment the polyhedral homotopy into

$$H(\mathbf{x},t) = \begin{cases} h_1(\mathbf{x},t) &= \sum_{\mathbf{a}\in S_1} [(1-t)c^*_{1,\mathbf{a}} + tc_{1,\mathbf{a}}]\,\mathbf{x}^{\mathbf{a}} t^{\omega_1(\mathbf{a})} + (1-t)b^*_1 \\ &\vdots \\ h_n(\mathbf{x},t) &= \sum_{\mathbf{a}\in S_n} [(1-t)c^*_{n,\mathbf{a}} + tc_{n,\mathbf{a}}]\,\mathbf{x}^{\mathbf{a}} t^{\omega_n(\mathbf{a})} + (1-t)b^*_n \end{cases}$$

for a generic set $\{c^*_{i,\mathbf{a}}\}_{i=1,\dots,n,\ \mathbf{a}\in S_i}$ of complex coefficients and complex constants $b^*_1,\dots,b^*_n$. It has been shown that this homotopy defines a finite number of smooth homotopy paths with no bifurcations and each isolated solution of $P(\mathbf{x}) = \mathbf{0}$ in $\mathbb{C}^n$ (which may be outside $(\mathbb{C}^*)^n$) lies at the end of one such path. That is, with this augmented homotopy, all complex isolated solutions (not necessarily in $(\mathbb{C}^*)^n$) of a polynomial system (not necessarily having generic coefficients) can be found.

Since its inception, this general method has achieved a great success. It is widely considered to be one of the most efficient, robust and reliable numerical methods for solving sparse polynomial systems. There is a rich and growing body of works devoted to different aspects in the efficient and stable implementation of the polyhedral homotopy method and homotopy methods in general including tracking of homotopy paths (e.g. Bates et al., 2011; Bates et al., 2008; Bates et al., 2009; Bates et al., 2006; Chen and Li, 2012; Morgan, 1986), handling singular end points (e.g. Kuo and Li, 2008; Morgan et al., 1990; Morgan et al., 1992), certifying results (e.g. Hauenstein et al., 2014; Hauenstein and Levandovskyy, 2011; Hauenstein and Sottile, 2012), as well as applying these methods to problems in science and engineering (e.g. Allgower and Georg, 2003; Bates et al., 2013; Morgan, 2009; Sommese and Wampler, 2005). Here we single out one computational aspect: the mixed cell computation.

3. Mixed cell enumeration

Critically important to the above construction of the polyhedral homotopies (7) are the mixed cells. Recall that the finite sets $S_1,\dots,S_n \subset \mathbb{N}^n_0 = (\mathbb{N}\cup\{0\})^n$ are the supports of the n polynomials $p_1,\dots,p_n$ containing points corresponding to the monomials appeared, and $\boldsymbol{\omega} = (\omega_1,\dots,\omega_n)$ with each $\omega_i : S_i \to \mathbb{Q}$ are the *generic* lifting functions that were used to construct the homotopy (2). Geometrically, $\boldsymbol{\omega}$ "lifts" the point sets $S_1,\dots,S_n$ to one higher dimension via $\mathbf{a} \mapsto \hat{\mathbf{a}} := (\mathbf{a}, \omega_i(\mathbf{a}))$ for each $\mathbf{a}\in S_i$ and $i = 1,\dots,n$. With the resulting "lifted" supports $\hat{S}_i = \{\hat{\mathbf{a}} \mid \mathbf{a}\in S_i\}$ in $\mathbb{R}^{n+1}$ for each $i = 1,\dots,n$, a mixed cell (with respect to the liftings) is represented by a vector $\boldsymbol{\alpha} = (\alpha_1,\dots,\alpha_n) \in \mathbb{Q}^n$ and its corresponding pairs $(\{\mathbf{a}_1,\mathbf{a}'_1\},\dots,\{\mathbf{a}_n,\mathbf{a}'_n\})$ with $\mathbf{a}_i,\mathbf{a}'_i \in S_i$ for each $i = 1,\dots,n$ such that the line segments $\mathrm{conv}\{\mathbf{a}_i,\mathbf{a}'_i\}$ are affinely independent and

$$\begin{cases} \langle \hat{\mathbf{a}}_1, \hat{\boldsymbol{\alpha}}\rangle &= \langle \hat{\mathbf{a}}'_1, \hat{\boldsymbol{\alpha}}\rangle < \langle \hat{\mathbf{a}}, \hat{\boldsymbol{\alpha}}\rangle \ \text{ for all } \mathbf{a} \in S_1 \setminus \{\mathbf{a}_1,\mathbf{a}'_1\} \\ &\vdots \qquad\qquad\qquad\qquad\qquad\qquad \vdots \\ \langle \hat{\mathbf{a}}_n, \hat{\boldsymbol{\alpha}}\rangle &= \langle \hat{\mathbf{a}}'_n, \hat{\boldsymbol{\alpha}}\rangle < \langle \hat{\mathbf{a}}, \hat{\boldsymbol{\alpha}}\rangle \ \text{ for all } \mathbf{a} \in S_n \setminus \{\mathbf{a}_n,\mathbf{a}'_n\} \end{cases} \tag{11}$$

where $\hat{\boldsymbol{\alpha}} = (\boldsymbol{\alpha}, 1) \in \mathbb{R}^{n+1}$ and for each $\mathbf{a}\in S_i$, $\hat{\mathbf{a}} = (\mathbf{a}, \omega_i(\mathbf{a})) \in \hat{S}_i$ is its "lifted" version in $\mathbb{R}^{n+1}$.

Remark 3. Interpreted geometrically, a mixed cells consists of n of edges of $\operatorname{conv} S_1, \ldots, \operatorname{conv} S_n$ respectively whose liftings in $\operatorname{conv} \hat{S}_1, \ldots, \operatorname{conv} \hat{S}_n$ share a common inner normal vector ($\hat{\boldsymbol{\alpha}}$ in this case). Since we require the inner normal $\hat{\boldsymbol{\alpha}} = (\boldsymbol{\alpha}, 1)$ to have 1 as the last coordinate (i.e., "upward pointing" along the $\mathbf{e}_{n+1}$ direction), we call such edges "lower" edges. These n edges span an n-dimensional parallelepiped. According to (9) and (10), the mixed volume (i.e. the BKK bound) is exactly the sum of the volumes of all such parallelepiped.

Each mixed cell induces an augmented polyhedral homotopy of the form (7). A key step is therefore the enumeration of all possible mixed cells of the above form, and, as noted above the mixed volume of $(\operatorname{conv} S_1, \ldots, \operatorname{conv} S_n)$ will be produced as an important by-product. This process, known as "mixed cell enumeration", turns out to be the main bottleneck in the polyhedral homotopy method both in terms of efficiency and parallel potentials. A rich web of works has since been developed on this subject (Chen et al., 2014; Emiris and Canny, 1995; Gao and Li, 2000; Gao and Li, 2003; Gao et al., 2005; Lee and Li, 2011; Li and Li, 2001; Mizutani et al., 2007; Verschelde et al., 1996).

The above algebraic description (11) of mixed cells is the basis on which most search methods are developed. While a brute-force approach of checking all the possible combinations against the system of inequalities (11) is certainly possible, the "combinatorial explosion" will render it impractical for all but the most trivial cases. One of the most efficient and robust class of algorithms for enumerating mixed cells is based on the idea of systematic "extension of subfaces". In this scheme, instead of finding the n-tuples of subsets of $\hat{S}_1, \ldots, \hat{S}_n$ satisfying (11) directly, one progressively constructs the mixed cells by joining one point at a time. In the beginning, one focuses on the first support S_1 and enumerate all the possible $\mathbf{a}_1 \in S_1$ for which there exists an $\hat{\boldsymbol{\alpha}} = (\boldsymbol{\alpha}, 1) \in \mathbb{R}^{n+1}$ such that

$$\langle \hat{\mathbf{a}}_1, \hat{\boldsymbol{\alpha}} \rangle \leq \langle \hat{\mathbf{a}}, \hat{\boldsymbol{\alpha}} \rangle \quad \text{for all } \mathbf{a} \in S_1 \ .$$

Then for each of these possibilities, one continues to search for $\mathbf{a}'_1 \in S_1 \setminus \{\mathbf{a}_1\}$ for which there exists an $\hat{\boldsymbol{\alpha}} = (\boldsymbol{\alpha}, 1) \in \mathbb{R}^{n+1}$ such that

$$\langle \hat{\mathbf{a}}_1, \hat{\boldsymbol{\alpha}} \rangle = \langle \hat{\mathbf{a}}'_1, \hat{\boldsymbol{\alpha}} \rangle \leq \langle \hat{\mathbf{a}}, \hat{\boldsymbol{\alpha}} \rangle \quad \text{for all } \mathbf{a} \in S_1 \ .$$

Each of these possibilities, in turn, allows for the search of an additional point $\mathbf{a}_2 \in S_2$ and an $\hat{\boldsymbol{\alpha}} = (\boldsymbol{\alpha}, 1) \in \mathbb{R}^{n+1}$ such that

$$\begin{aligned} \langle \hat{\mathbf{a}}_1, \hat{\boldsymbol{\alpha}} \rangle \ &= \ \langle \hat{\mathbf{a}}'_1, \hat{\boldsymbol{\alpha}} \rangle \leq \langle \hat{\mathbf{a}}, \hat{\boldsymbol{\alpha}} \rangle \quad && \text{for all } \mathbf{a} \ \in \ S_1 \\ \langle \hat{\mathbf{a}}_2, \hat{\boldsymbol{\alpha}} \rangle \ &\leq \ \langle \hat{\mathbf{a}}, \hat{\boldsymbol{\alpha}} \rangle && \text{for all } \mathbf{a} \ \in \ S_2. \end{aligned}$$

Similarly, for each of the resulting possibilities further search attempts will be carried out to extend them. This self-sustaining process continues until one reaches all the possible n-tuples of pairs in $S_1, \ldots, S_n$ that satisfy (11) which are exactly all the mixed cells. We formalize this procedure via the concept of "subfaces".

Definition 1. Given supports $S_1, \ldots, S_n$ and liftings $\boldsymbol{\omega} = (\omega_1, \ldots, \omega_n)$, a $(k_1, \ldots, k_r)$**-subface** of $(\hat{S}_1, \ldots, \hat{S}_r)$ for some $r \leq n$ is an r-tuple of affinely independent sets of the form

$$\left(\left\{ \hat{\mathbf{a}}_0^{(1)}, \ldots, \hat{\mathbf{a}}_{k_1}^{(1)} \right\} \ , \ \ldots \ , \left\{ \hat{\mathbf{a}}_0^{(r)}, \ldots, \hat{\mathbf{a}}_{k_r}^{(r)} \right\} \right)$$

with each $\mathbf{a}_j^{(i)} \in S_i$ for which there exists an $\hat{\boldsymbol{\alpha}} = (\boldsymbol{\alpha}, 1) \in \mathbb{R}^{n+1}$ such that for each $i = 1, \ldots, r$,

$$\begin{cases} \langle \hat{\mathbf{a}}_0^{(i)}, \hat{\boldsymbol{\alpha}} \rangle = \langle \hat{\mathbf{a}}_1^{(i)}, \hat{\boldsymbol{\alpha}} \rangle & \text{for } j = 1 \ , \ldots, k_i \\ \langle \hat{\mathbf{a}}_0^{(i)}, \hat{\boldsymbol{\alpha}} \rangle \leq \langle \hat{\mathbf{a}}, \hat{\boldsymbol{\alpha}} \rangle & \text{for all } \mathbf{a} \ \in S_i. \end{cases} \tag{12}$$

Furthermore, we say a subface $(F_1, \ldots, F_r)$ **includes** the subface $(F'_1, \ldots, F'_r)$ if $F'_i \subseteq F_i$ for each i.

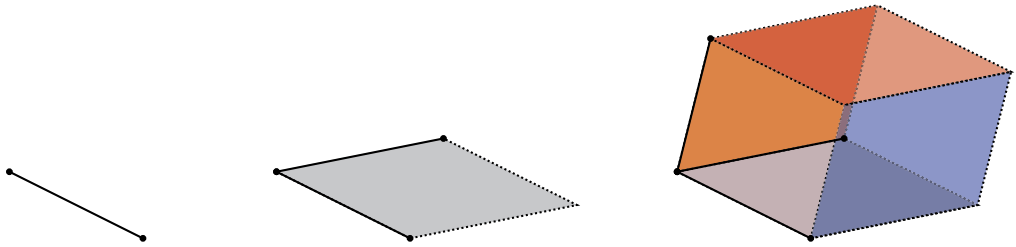


Fig. 1. The extension process builds mixed cells one edge at a time.

Comparing with (11), it is clear that mixed cells are simply $(1, \ldots, 1)$-subfaces. With the notion of subfaces, the basic scheme for the search of mixed cells can then be stated as the systematic extension of (0)-subfaces to (1)-subfaces and then to $(1, 0)$-subfaces, etc. The process terminates when one reaches all the $(1, \ldots, 1)$-subfaces of $(\hat{S}_1, \ldots, \hat{S}_n)$ which are precisely the mixed cells which we aimed to find.

The building block for the above scheme for enumerating mixed cells via systematic extension of subfaces is the "one-point test". Originally developed in Takeda et al. (2000) and independently in Li and Li (2001), this simple procedure has since been adopted by most software packages for mixed cells enumeration. Among a range of different formulations, for simplicity, we state a basic variation:

Definition 2 *(One-point test).* Given a $(k_1, \ldots, k_r)$-subface $(F_1, \ldots, F_r)$ with each $F_i = \{\hat{\mathbf{a}}_0^{(i)}, \ldots, \hat{\mathbf{a}}_{k_i}^{(i)}\} \subseteq \hat{S}_i$ and a point $\hat{\mathbf{b}} \in \hat{S}_r \setminus F_r$, the **one-point test** of $\hat{\mathbf{b}}$ with respect to $(F_1, \ldots, F_r)$ is the *linear programming problem*

$$LP(F_1, \ldots, F_r; \hat{\mathbf{b}}) : \begin{cases} \text{Minimize } \langle \hat{\mathbf{b}}, \hat{\boldsymbol{\alpha}} \rangle - h \text{ subject to} & \\ \langle \hat{\mathbf{a}}_0^{(i)}, \hat{\boldsymbol{\alpha}} \rangle = \langle \hat{\mathbf{a}}_1^{(j)}, \hat{\boldsymbol{\alpha}} \rangle & \text{for all } j = 1, \ldots, k_i \\ \langle \hat{\mathbf{a}}_0^{(i)}, \hat{\boldsymbol{\alpha}} \rangle \le \langle \hat{\mathbf{a}}, \hat{\boldsymbol{\alpha}} \rangle & \text{for all } \mathbf{a} \in S_i,\ i = 1, \ldots, r-1 \\ h = \langle \hat{\mathbf{a}}_j^{(r)}, \hat{\boldsymbol{\alpha}} \rangle & \text{for all } j = 0, \ldots, k_r \\ h \le \langle \hat{\mathbf{a}}, \hat{\boldsymbol{\alpha}} \rangle & \text{for all } \mathbf{a} \in S_r \end{cases} \tag{13}$$

in the $n+1$ variables $(\boldsymbol{\alpha}, h) = (\alpha_1, \ldots, \alpha_n, h)$. To include more general situations, we allow F_r to be empty in this definition.

Remark 4. As noted in Remark 3, the mixed cells can be interpreted as parallelepipeds whose edges lift to "lower" edges of $\operatorname{conv} \hat{S}_1, \ldots, \operatorname{conv} \hat{S}_n$. In the same vein, extension process via successive one-point tests can therefore be understood as the process of building such parallelepipeds one edge at a time. Fig. 1 illustrates this process.

Clearly, since the constraints already require $h \le \langle \hat{\mathbf{a}}, \hat{\boldsymbol{\alpha}} \rangle$ for all $\mathbf{a} \in S_r$, the objective function of the above linear programming problem is therefore bounded below by zero. If the minimum reaches this lower bound, namely,

$$h = \langle \hat{\mathbf{b}}, \hat{\boldsymbol{\alpha}} \rangle \le \langle \hat{\mathbf{a}}, \hat{\boldsymbol{\alpha}} \rangle \text{ for all } \mathbf{a} \in S_r$$

holds in addition to the imposed constraints, then the $(k_1, \ldots, k_r)$-subface $(F_1, \ldots, F_r)$ can be extended to a $(k_1, \ldots, k_r+1)$-subface $(F_1, \ldots, F_r \cup \{\hat{\mathbf{b}}\})$. On the other hand, if the LP problem is infeasible or the minimum is strictly greater than zero, then the subface $(F_1, \ldots, F_r \cup \{\hat{\mathbf{b}}\})$ cannot exist and hence the extension fails. Using such one-point tests, the algorithm for mixed cell enumeration via systematic extension of subfaces of $\hat{S}_1, \ldots, \hat{S}_n$ can be constructed.

Remark 5. An important observation is that not every one-point test problem must be solved when the classical *simplex method* is used to solve the linear programming problems (13) listed above. The result of certain one-point test problems can be determined with minimum computational costs using the fruitful information generated by the "simplicial pivoting" performed in solving other one-point tests. Indeed, in practice, it is often the case that only a small fraction of one-point test problems need to be solved. In the last decades, a great number of important techniques have been developed to take advantage of this special feature. Here we refer to Chen et al. (2014), Lee and Li (2011), Li (2003), Mizutani and Takeda (2008), Mizutani et al. (2007), Takeda et al. (2000) for discussions of the more advanced techniques.

Serial algorithms based on this basic scheme has been extensively studied (Gao and Li, 2000; Gao and Li, 2003; Gao et al., 2005; Gunji et al., 2004; Lee and Li, 2011; Li and Li, 2001; Mizutani and Takeda, 2008; Mizutani et al., 2007; Takeda et al., 2000). MixedVol-2.0 (Lee and Li, 2011), a part of Hom4PS-2.0 is widely considered as one of the most efficient serial implementation based on this scheme. Discussed in the following sections will be the parallel revision of this scheme attempted in Chen et al. (2014) as well as the recent efforts in improving the parallel scalability.

4. Parallel mixed cells enumeration

While the "path tracking" part of the polyhedral homotopy continuation method is *pleasantly parallel*, the mixed cell enumeration process is potentially a major bottleneck in terms of parallel scalability. An important revision in Hom4PS-3, compared to Hom4PS-2.0 (Lee et al., 2008), is the integration of a fully parallel mixed cell enumeration algorithm that is efficient, robust, and highly scalable. It greatly extends the reach of polyhedral homotopy method in the realm of large polynomial systems.

Developed in Chen et al. (2014) with direct inspiration from Gao and Li (2000), Lee and Li (2011), Li and Li (2001), Mizutani and Takeda (2008), the parallelization of the mixed cell enumeration is based on a reformulation as a graph-theoretic search problem for which parallel algorithm can achieve great efficiency and scalability.

Recall that in the above scheme, the search for mixed cells is performed by checking candidates of subfaces (n-tuples of subsets of lifted supports $\hat{S}_1, \ldots, \hat{S}_n$) using one-point tests (13). The candidates of subfaces naturally form a *direct acyclic graph* or DAG via the (componentwise) inclusion relationship among them: The nodes in this graph are the candidates for subfaces, i.e., r-tuples of the form $(F_1, \ldots, F_r)$ where $F_i \subseteq \hat{S}_i$ for $i = 1, \ldots, r$. It has edges $(F_1, \ldots, F_r) \to (F'_1, \ldots, F'_k)$ whenever $F_i \subseteq F'_i$ for $i = 1, \ldots, r$. Fig. 2 illustrates a portion of such a graph.

Remark 6. For simplicity, in the illustration of the DAG of candidates of subfaces we have implicitly fixed the ordering of the supports: from $S_1, S_2, \ldots$ to S_n. It is, however, clear that both the notion of subfaces and their extension process are independent from the ordering. That is, one can perform

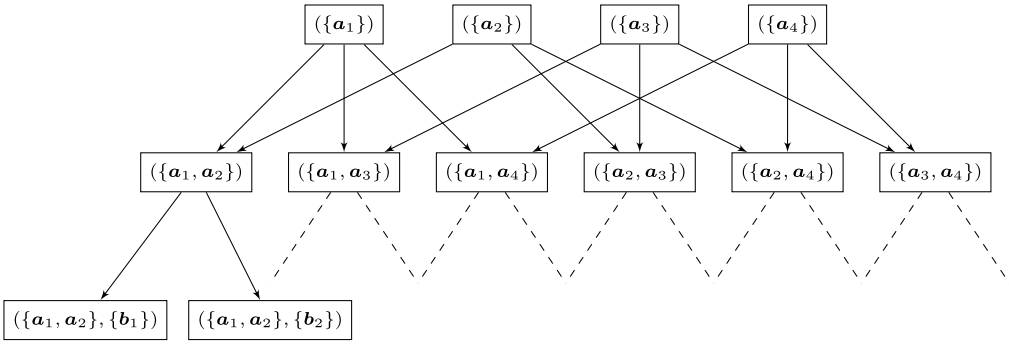


Fig. 2. A DAG containing candidates of subfaces with edges representing component-wise inclusion. The set of subfaces forms a subgraph known as the feasible subgraph.

the extension process in any ordering of the supports. Extensive experiments suggest the ordering of the supports can greatly influence the overall efficiency of the algorithm. This was first discovered and exploited in Mizutani et al. (2007) and was adopted, with modification, by MixedVol-2.0 (Lee and Li, 2011). The same technique is further improved in Chen et al. (2014) and now an integral part of Hom4PS-3.

In this graph, some nodes, including the mixed cells, are subfaces while others correspond to tuples that do not define subfaces (Definition 1). The subgraph containing all the subfaces shall be called the **feasible subgraph**, referring to the fact that their corresponding systems of inequalities of the form (12) are feasible. In this context, the mixed cell enumeration process can then be considered as the problem of gradually exploring the feasible subgraph through the connections among them.

In graph theory, this is a classical *graph traversal problem* (or a specialized *search problem*). Clearly, each node only needs to be visited once. That is, one only needs to explore a *spanning tree* of the feasible subgraph. To this end, the algorithm keeps track of all the nodes that are discovered but not yet completely explored, that is, known nodes whose connections to other nodes are not yet fully explored. The main task of the algorithm is then to take a discovered (but not yet completely explored) node and explore its connection via one-point tests (13). Each new node in the feasible subgraph is then recorded for further exploration. Once all such connections are explored, the original node is discarded. This process repeats until the entire feasible subgraph has been explored.

This point of view reveals the inherent parallelism in the mixed cell enumeration process: nodes on different branches of the spanning tree can be explored in parallel. To start the algorithm, an initial set of nodes corresponding to (0)-subfaces (the topmost nodes in Fig. 2) are generated by solving one-point test problems of the form $LP(\{\}; \hat{\mathbf{a}})$ for each $\hat{\mathbf{a}} \in \hat{S}_1$. For each (0)-subface $(\{\hat{\mathbf{a}}\})$ discovered, the algorithm then explores other potentially connected nodes by solving one-point tests of the form $LP(\{\hat{\mathbf{a}}\}; \hat{\mathbf{a}}')$ for $\hat{\mathbf{a}}' \in \hat{S}_1 \setminus \{\hat{\mathbf{a}}\}$. From each resulting subface the algorithm can continue the process and potentially discover more connected nodes. This self-sustaining process continues until all connected nodes are discovered including the mixed cells which we aim to enumerate. The detail of this class of algorithms can be found in standard textbooks such as Skiena (2009). However, as noted in Remark 5, in practice a large number of nodes can generally be discovered without direct exploration.

In order to keep track of the progress of the exploration and coordinate multiple threads, the collection of discovered (but not yet completely explored) nodes must be stored in a data structure. However, to conserve memory, all one-point tests with respect to the same subface are grouped together, called a **task**. That is, a task represents all the one-point tests of the form $LP((F_1, \ldots, F_r); \hat{\mathbf{b}})$ where $(F_1, \ldots, F_r)$ is a subface already discovered. In the CPU-based parallel algorithms, a task will be the smallest unit in the parallelization in the sense that they are always performed together by a single thread. The parallelism within a task can be exploited on GPUs and is discussed in §4.4.

The parallel algorithm maintains a dynamic pool of tasks, or simply the task pool. Though a number of data structures can be used, Hom4PS-3 chose to use a *priority queue* to maintain the task pool which provides fine-grained control of the exploration process, e.g., *depth-first-search* versus *breadth-first-search*. We refer to Chen et al. (2014) for the details.

Multiple threads will operate on the task pool and perform one-point tests simultaneously: Each thread repeatedly fetches a single task from the pool and explores it by performing a series of one-point tests. This "fetch-and-explore" procedure continues until the task pool is empty and there is no tasks that are currently being explored. At this point the feasible subgraph is completely explored and all the mixed cells have been obtained. The algorithm then terminates. This algorithm is summarized in the following EXPLORE subroutine.

```
1: function EXPLORE( TaskPool )
2:     while TaskPool ≠ ∅ do
3:         (F_1, ..., F_r) ← DEQUEUE( TaskPool )
4:         for all b̂ ∈ Ŝ_r \ F_r do
5:             if ISUNKNOWN( (F_1, ..., F_r ∪ {b̂}) ) then
6:                 if ONEPOINTTEST( (F_1, ..., F_r); b̂ ) then
7:                     F' ← (F_1, ..., F_r ∪ {b̂})
```

```
                if F′ is a mixed cell then
                    MixedCells ← MixedCells ∪ {F′}
                else
                    ENQUEUE( TaskPool, F′)
                end if
            end if
        end if
    end for
  end while
end function
```

Here the DEQUEUE subroutine removes a single item from a priority queue, while the ENQUEUE subroutine append an item into the priority queue. Since multiple threads will access the priority queue concurrently, this function must be made thread-safe to prevent a *race condition* (Herlihy and Shavit, 2012) (a condition in which multiple threads simultaneously alter the same data structure resulting in catastrophic data corruption). The ISUNKNOWN function determines if a node has already been discovered. That is, it returns whether or not the status of a candidate for a subface has already been determined. This is necessary to prevent a node from being visited more than once. It is particularly important since, as noted in Remark 5, many nodes can be discovered using information generated by other one-point tests.

4.1. Parallel mixed cell enumeration on multicore architectures

The work in Chen et al. (2014) has demonstrated that when properly implemented, the parallel algorithm described above can achieve great efficiency on small to medium sized multicore systems. In particular, nearly linear speedup has been achieved on systems with up to 12 cores. The integration of the parallel mixed cell enumeration algorithm into Hom4PS-3 focused on the scalability beyond 12 cores.

As shown in the pseudo code above, the computation intensive part of the EXPLORE subroutine is pleasantly parallel in the sense that threads do not interact with one another. However, multiple threads must access the same priority queue (`TaskPool`) via the DEQUEUE and ENQUEUE subroutine. Consequently, synchronization mechanisms must be in place to prevent race condition. In Chen et al. (2014), "mutex" (mutual exclusion, a standard mechanism commonly used to ensure only one thread has access to a data structure) was proposed to guard the task pool and prevent race conditions from happening. On multicore systems of small to medium size, this solution exhibits acceptable efficiency and scalability. However subsequent studies reveal that the use of mutex severely limits the overall scalability of the algorithm on larger systems, that is, it turns out to be the key factor that limits more processor cores from performing at their peak efficiency. Hom4PS-3, these functions are implemented based on the *concurrent data structure* provided by the Intel TBB library which has shown much better scalability.[4]

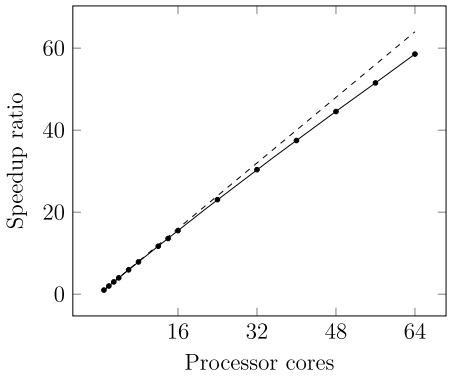
On multi-core systems, the implementation of this algorithm, based on Intel TBB, in Hom4PS-3 has achieved remarkable efficiency and scalability far superior to the original algorithm proposed in Chen et al. (2014). Nearly n-fold linear speedups scalable up to 64 processor cores have been observed in experiments on standard test suite problems. Table 1 shows the speedup ratio observed on the standard benchmark problem cyclic-15 (Björck and Fröberg, 1991).

4.2. On-the-fly NUMA optimization

Modern shared-memory systems with a large number of processor cores usually adopt a Non-Uniform Memory Access (Herlihy and Shavit, 2012), or NUMA, architecture in which each processor

[4] On systems that Intel TBB does not support, (e.g. certain legacy Unix systems) an alternative implementation based on *non-blocking linked lists* is used instead. This implementation, however, will not preserve the priority. That is, tasks will be fetched in a random order.

Table 1
Speedup ratio (solid line) achieved on a 64 core system (AMD Opteron processors with a total of 64 cores together with 512GB memory) for the cyclic-15 (Björck and Fröberg, 1991) problem showing close to n-fold linear speedups for up to 64 cores. Here the dashed line represents the ideal n-fold linear speedup. The speedup is computed in comparison with the fastest serial implementations published: MixedVol-2.0 (Lee and Li, 2011) and DEMiCs (Mizutani and Takeda, 2008).



core can access all the available memory but potentially at different speeds depending on the relative closeness between the core and memory. Developed in the 1990s as an answer to the scalability limitation in the traditional SMP (symmetric multiprocessing) architectures, it has gained great popularity in the world of high performance computing especially with AMD and Intel adopting the technology under the names HyperTransport (2003) and QPI (2007) respectively.

Fig. 3 shows the "memory-processor topology" of a NUMA system that consists of 8 nodes. Each node contains 4 processor cores as well as their "local" memory which they can access at full speed. The edges between nodes indicate the direct connectedness between nodes and determines the speed at which processor cores on one node can access memory on other nodes. For instance a processor on node 1 can access the memory on node 2 at a slower rate than it could access its local memory on node 1. The same core can access memory on node 3 at a even slower rate due to the minimum two jumps required (through node 2 or 4). Similarly there are at least three jumps between node 1 and node 7. Consequently, that processor core would have the slowest memory access to memory on node 7.

Recall that most of the data required by our algorithm for mixed cell enumeration reside in the task pool. On the NUMA system, it is therefore crucial to split a single task pool into several task pools shared by the threads in an optimized pattern that ideally matches the underlying memory-processor topology. The planning of this pattern is governed by two conflicting constraints:

1. Each thread should access a task pool that is placed as close as possible in terms of memory-processor topology to optimize the memory access time.
2. Each task pool should be shared by as many threads as possible to avoid load balancing issues (to be discussed in detail in §4.3).

Unfortunately there is currently no standardized method to determine precisely the memory-processor topology (Herlihy and Shavit, 2012). In particular, while one generally could inquire from the operating system which memory access patterns are slow, but not how slow. Coupled with the fact that the operating system can, at any time, migrate a running thread from one processor core to another, a dynamic and on-the-fly planning of the task pool placement and sharing pattern is therefore a necessity. Here we briefly outline the procedure.

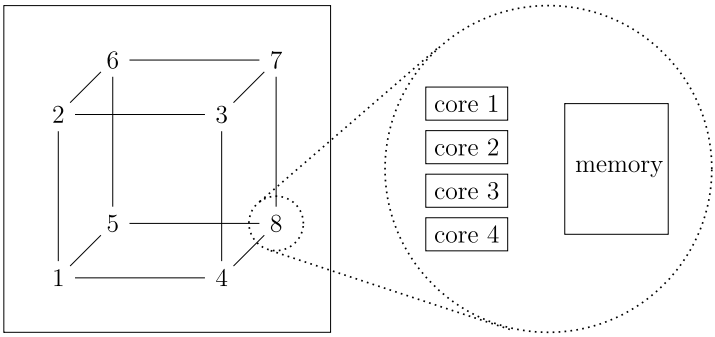
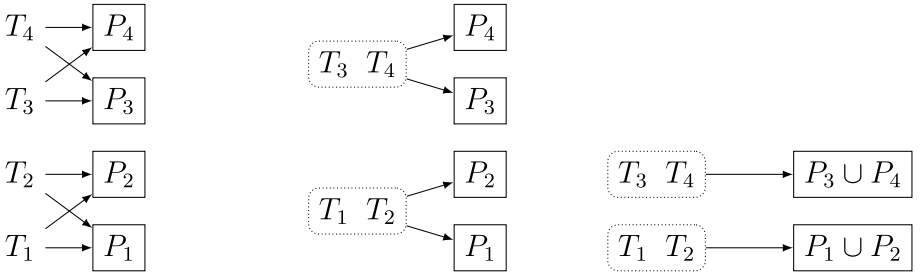


Fig. 3. An example of a NUMA node structure.



(a) Four threads T_1,T_2,T_3,T_4 with their preferred task pools P_1, P_2, P_3, P_4.

(b) Two clusters of threads are formed based on common preference on task pools.

(c) Task pools preferred by the same clusters of threads are merged together.

Fig. 4. Dynamic optimization of the task pool sharing pattern on NUMA architectures.

At the beginning of the extension process, the program starts with an initial "evaluation phase" in which each thread is spawned with its own task pool in the memory local to the processor core that the thread runs on.[5] During the extension process, each thread will access tasks from *all* task pools. The average time for accessing each task pool is monitored, and the resulting data is used to construct a list of "preferred" task pools for having best access time. (see for example Fig. 4a). Then threads that prefer the same task pool are grouped into "thread clusters" (see Fig. 4b for the formation of clusters). Conversely task pools that are preferred by the same cluster of threads are merged (see Fig. 4c for the merger of task pools). This process is repeated until the threads' preference of task pools stabilizes. At this point each cluster of threads has a single preferred task pool that has the best access time and this "evaluation phase" is terminated. Afterwards, this task pool sharing pattern is fixed and each thread only access its own preferred task pool. This optimization procedure can be performed again whenever threads have migrated to different processor cores or certain task pool becomes empty before others.

Incurring minimum additional computational cost at the initial "evaluation phase", this technique substantially improves the memory access time on NUMA systems. As shown in Table 2, in experiments on standard benchmark problems: the cyclic family (Björck and Fröberg, 1991), the five-body central configuration problem (fivebody) (Albouy and Kaloshin, 2012; Hampton and Jensen, 2011; Lee and Santoprete, 2009), and the 6-vortex problem (vortexAC6) (Hampton and Moeckel, 2009;

[5] On Linux, this is done via the standard library libnuma provided by most Linux distributions. On Unix this step requires the correct configuration to be set by the user.

Table 2
The memory access speedup (with errors within ± 0.05) and overall reductions in run time in the extension process over the basic algorithm observed in experiments with a few large systems in standard test suites on a NUMA system consisting of 8 node each having 8 quad-core AMD Opteron processor with a total of 256 cores. Experiments marked by '*' only used 4 out of 8 nodes (128 of the 256 cores) due to smaller size. The time spent on the computation of cell volume (9) and the accumulation of the mixed volume are not included since they are not affect by the memory access pattern.

System	Mem. acc. speedup	Overall reduction
cyclic-14*	1.40	4.9%
cyclic-15*	2.40	8.2%
cyclic-16*	4.50	9.5%
fivebody	17.55	33.2%
vortex-6	19.95	34.5%

von Helmholtz, 1858), approximately $1.5\times$ to $20\times$ speedup (150% to 2000%) in memory access time[6] have been observed which resulted in 5% to 35% overall speedups in the extension process.

Remark 7. The sensitivity in the overall run time to memory access pattern exhibited in Table 2 highlights the possibility that for sufficiently large systems, the mixed cell enumeration problem will become memory-bound. That is, the run time will no longer be dominated by the number of floating point operations but will instead be dominated by the memory access latency, an important factor that is often ignored in complexity analysis.

4.3. Extending to computer clusters

While the above referenced NUMA architecture allows shared-memory systems to scale to tens or even more than 100 processor cores, their scalability is still limited by the inherently high cost. Larger systems that contain several hundreds or even thousands of cores generally take the form of distributed-memory systems in which nodes, connected by some network, do not directly share memory spaces but communicate with one another by passing messages instead. The parallel algorithm described above can be extended to distributed-memory systems including *computer clusters* in which nodes are connected by dedicated high speed network.

In such distributed-memory systems, a *master-worker* model is chosen to extend the parallel algorithm described above. In this model, the "master" runs on a single node in the system. It first populates its own task pool with an initial set of subfaces. The number of initial subfaces is determined based on the number of nodes available within the system (a prescribed multiple of the number of nodes). This initial task pool is then divided into equal portions and sent to each of the remaining nodes as seeds for exploration. Each worker executes the EXPLORE algorithm described in §3 and explores the subgraph accessible from the initial set of nodes sent by the master until its task pool becomes empty. At the end each worker would have a collection of mutually exclusive mixed cells. These are then passed back to the master to form a final set of mixed cells. This basic scheme was proposed in Li and Tsai (2009) and significantly improved in Chen et al. (2014). Fig. 5 shows a typical setup in which arrows indicate the passing of tasks between nodes.

The Message Passing Interface, or MPI, is a specification that allows nodes to communicate with one another in a cluster. Though not sanctioned by any major standards body, MPI has became a *de facto* standard for scientific computation on computer clusters. In Hom4PS-3, this protocol is used for the communication between the master and workers.

An implementation based on this master-worker model would not be scalable without *load balancing* mechanisms. In exploring the spanning tree of the feasible subgraph, certain branches may require

[6] The memory access time is approximated by using the "memory access latency" provided by the Intel VTune software which closely correlates to the actual memory access time which is generally difficult to measure. For best accuracy, all CPU caches were disabled when measuring memory access latency.

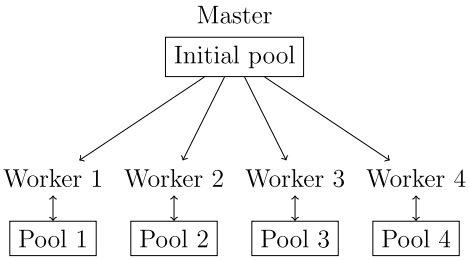


Fig. 5. A master-worker setup for performing parallel mixed cell enumeration on a computer cluster with 5 nodes. One node act as the master and the remaining nodes act as workers each having their own task pools. The initial tasks are passed to each worker for exploration.

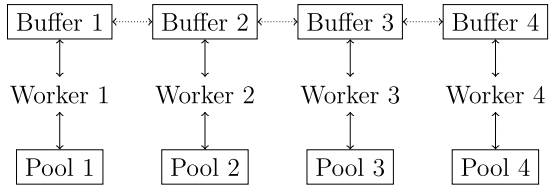


Fig. 6. Buffering and load balancing mechanisms among workers. Tasks are moved from one buffer to another in the background. In contrast to the simple master-worker setup illustrated in Fig. 5, this model relies mostly on direct (peer-to-peer) communication among worker nodes.

significantly more CPU time than other branches. Such imbalances generally cannot be detected easily ahead of time, a dynamic load balancing mechanism that actively shifts tasks from one worker to another is therefore critically essential to the overall efficiency and scalability of this algorithm. In Chen et al. (2014), this problem is resolved by requiring each worker to request more tasks from the master when it exhausted its own task pool. However, the waiting spent on message passing incurs a measurable and sometime significant cost. Indeed, in large clusters, experiments suggest that the waiting time often dominate the overall run time as the single master node can be easily overwhelmed by the large number of worker nodes.

A major improvement Hom4PS-3 provides over Chen et al. (2014) is the use of asynchronous message passing and buffering to further improve the load balancing mechanism: In addition to the task pool, each worker also maintains an "overflow buffer" which is filled with newly discovered tasks whenever the number of tasks in the task pool exceeds a prescribed threshold. The number of tasks in the buffer is periodically reported back to the master which maintains a dynamic tally of the imbalance of buffers among the workers. The master periodically broadcasts to all workers which buffer has the lowest number of tasks. Upon receiving the notice, each worker whose buffer has higher number of tasks then passes certain number of tasks to the buffers with lowest number of tasks. Here the passing of tasks from one buffer to another is performed via efficient *asynchronous message passing* provided by recent revisions of the MPI standard that are asynchronous in the sense that they do not interrupt the threads from their main computational intensive task. That is, the transmission of data takes place in the background and the workers have no need to wait for the sending or receiving. See Fig. 6 for the illustration. When a worker exhausted its own task pool, it moves tasks from its own buffer, which resides in the worker's local memory space, to its task pool so that the task pool reaches the original prescribed capacity. The worker then continues its exploration.

The asynchronous load balancing substantially improved the efficiency and scalability of the basic scheme developed in Chen et al. (2014). The implementation exhibits great scalability on clusters having between 32 and 200 nodes. It is expected that the speedup ratio cannot get close to those achieved on a multi-core system (as shown in Table 1) due to the inherently higher cost in communication. However it is possible to scale to many more processors cores than on multi-core or NUMA

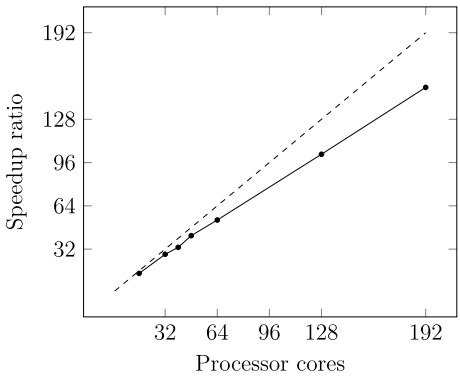


Fig. 7. Speedup ratios achieved by the distributed-memory variation of the parallel algorithm for mixed cell enumeration over the fastest serial implementations MixedVol-2.0 (Lee and Li, 2011) and DEMiCs (Mizutani and Takeda, 2008). Measurements are done in a cluster containing up to 192 processor cores.

system. For example, the speedup ratios achieved using multiple nodes in a cluster for the fivebody (five body central configuration) problem (Albouy and Kaloshin, 2012; Hampton and Jensen, 2011; Lee and Santoprete, 2009) is shown in Fig. 7.

4.4. GPU accelerated mixed cell enumeration algorithm

An exciting development in the world of computing is the advent of general purpose parallel computation on GPUs. While originally designed to handle graphics rendering only, over the years GPU devices have become sufficiently sophisticated to handle a much wider range of problems. Highly parallel by design, GPUs are more efficient than traditional CPUs in performing a variety of complex tasks (NVIDIA Corporation, 2011). In terms of raw computational power, GPU devices have now surpassed the fastest CPU available (NVIDIA Corporation, 2011). However, hurdles still exist along the path to fully employing GPU computing. In particular, both the memory layout and thread organization are very different from their counterparts in traditional CPUs. In NVidia's CUDA architecture, for example, threads are always organized in groups of 32 threads called "warps" (NVIDIA Corporation, 2011) which are the basic scheduling units in CUDA GPUs with all threads in a warp always perform the same instruction at the same time.

In the attempt to take advantage of GPU devices, Hom4PS-3 adopts the approach of GPU *accelerated* mixed cell enumeration algorithm where the CPUs still perform the main algorithm, and GPU devices provide assistance in computation intensive tasks that are best suited for GPUs. Though still an experimental part of the software with active development currently underway, the remarkable speedups obtained definitely merit further investigation on the idea. The preliminary results are therefore presented here.

The GPU accelerated mixed cell enumeration algorithm explores the parallelism inside individual "tasks" which is not directly utilized in the approaches discussed above. It comes in the form of a separated module that performs a specific set of operations inside the one-point test problems (13). As described in §3, in the parallel algorithm included in Hom4PS-3, the simplex method is preferred due to the great amount of additional information it generates which can be used to significantly accelerate the mixed cell enumeration process (see Remark 5).

The simplex method is an iterative method for solving the linear programming problems of the form (13). The implementation of the simplex method itself is outside the scope of this article, we therefore refer to Lee and Li (2011), Chen et al. (2014) for the detailed description. Leaving aside the technical detail, the key property being exploited here is that each iteration in the simplex method involves the manipulation of a fixed size matrix. In particular, the part that dominates the overall computational costs takes the form of a matrix-vector multiplication

$$\mathbf{y} \leftarrow A \cdot \mathbf{x}$$

Table 3
The average speedup ratio in computing $\mathbf{y} \leftarrow A\mathbf{x}$ and solving one-point test problems respectively observed in some standard test suite problems. The data reflect the average of 10000 one-point tests for each problem.

System (dimension)	Avg. speedup in $\mathbf{y} \leftarrow A\mathbf{x}$
cyclic15 (15)	3.55
cyclic19 (19)	3.95
cyclic23 (23)	8.09
cyclic27 (27)	9.00
cyclic31 (31)	31.54
cyclic47 (47)	39.90

where A is a fixed matrix with real entries whose number of rows is much greater than the number of columns. The potential parallelism in this operation is obvious: in principle, every scalar–scalar multiplication involved can be computed independently.

A straightforward approach is to utilize the standard NVidia cuBLAS library (NVIDIA Corporation, 2011) which has built-in functions designed specifically for handling this task. Unfortunately, the cuBLAS library incurs a small but measurable amount of additional cost upon each invocation. Recall that the enumeration of mixed cells involves a large number of one-point tests, the cumulative costs associated with cuBLAS often outweighs its benefits, as our experiments suggest.

A direct programming approach is therefore in place. The computation is divided into two steps. First, all the scalar–scalar products $\{a_{i,j} \cdot x_j\}$ are computed in parallel where $a_{i,j}$ and x_j are the entries of A and $\mathbf{x}$ respectively. GPU devices are generally capable of running hundreds or even thousands of threads simultaneously. Conforming to the organization of threads on GPU, this step is done in block of 16×16 threads. That is, the (i, j) block of 16×16 threads computes the array of products

$$\begin{bmatrix} a_{16i,16j} \cdot x_{16j} & a_{16i,16j+1} \cdot x_{16j+1} & \cdots & a_{16i,16j+15} \cdot x_{16j+15} \\ a_{16i+1,16j} \cdot x_{16j} & a_{16i+1,16j+1} \cdot x_{16j+1} & \cdots & a_{16i+1,16j+15} \cdot x_{16j+15} \\ \vdots & \vdots & & \vdots \\ a_{16i+15,16j} \cdot x_{16j} & a_{16i+15,16j+1} \cdot x_{16j+1} & \cdots & a_{16i+15,16j+15} \cdot x_{16j+15} \end{bmatrix}$$

with one thread computing each entry. Once the scalar–scalar products $\{a_{i,j} \cdot x_j\}$ are all computed, the standard "parallel reduction" algorithm is then applied to compute the row sums among blocks of the above form. Table 3 shows the speedup results of this algorithm observed on some standard test suite problems using a NVidia GTX 970 graphic card. Measuring this operation of computing $\mathbf{y} \leftarrow A\mathbf{x}$ alone, approximately $3.5\times$ to $40\times$ speedup ratio have been achieved on sufficiently large systems.

Though still limited in its functionality and portability, on sufficiently large systems the GPU accelerated part shows remarkably promising results. Developments in applying GPU to more operations in the mixed cell enumeration algorithm are currently underway.

5. Concluding remarks

The parallel polyhedral homotopy method is implemented in Hom4PS-3 as a major revision to its predecessor Hom4PS-2.0 which has already been proved to be efficient and robust in a wide variety of applications. Inheriting all the core strength of Hom4PS-2.0, the development of Hom4PS-3 focuses on the only major limiting factor of polyhedral homotopy in parallel computation: the mixed cell enumeration problem. Following the same general approach proposed in Chen et al. (2014) the current work integrates several important modifications tailored for individual hardware architectures which have brought substantial improvements in parallel scalability. This development eliminates the bottleneck of the polyhedral homotopy method in terms of scalability in parallel computation and greatly extends the reach of this method in dealing with large polynomial systems.

References

Abraham, R.H., Robbin, J.W., 1967. Transversal Mappings and Flows. Benjamin.

Albouy, A., Kaloshin, V., 2012. Finiteness of central configurations of five bodies in the plane. Ann. Math. 176 (1), 535–588.

Allgower, E., Georg, K., 2003. Introduction to Numerical Continuation Methods, vol. 45. Society for Industrial and Applied Mathematics.

Attardi, G., Traverso, C., 1995. The PoSSo Library for Polynomial System Solving. In: Proc. of AIHENP95.

Bates, D.J., Hauenstein, J.D., Sommese, A.J., 2011. Efficient path tracking methods. Numer. Algorithms 58 (4), 451–459.

Bates, D.J., Hauenstein, J.D., Sommese, A.J., Wampler, C.W., 2008. Adaptive multiprecision path tracking. SIAM J. Numer. Anal. 46 (2), 722–746.

Bates, D.J., Hauenstein, J.D., Sommese, A.J., Wampler, C.W., 2013. Numerically Solving Polynomial Systems with Bertini. Society for Industrial and Applied Mathematics.

Bates, D.J., Hauenstein, J.D., Sommese, A.J., Wampler II, C.W., 2009. Stepsize control for path tracking. In: Interactions of Classical and Numerical Algebraic Geometry. In: Contemp. Math., vol. 496. Amer. Math. Soc., Providence, RI, pp. 21–31.

Bates, D.J., Sommese, A.J., Wampler, C.W., 2006. Multiprecision path tracking. Preprint, arXiv:math/0605105.

Bernshtein, D.N., 1975. The number of roots of a system of equations. Funct. Anal. Appl. 9 (3), 183–185.

Björck, G., Fröberg, R., 1991. A faster way to count the solutions of inhomogeneous systems of algebraic equations, with applications to cyclic n-roots. J. Symb. Comput. 12 (3), 329–336.

Bozóki, S., Lee, T.-L., Rónyai, L., 2015. Seven mutually touching infinite cylinders. Comput. Geom. 48 (2), 87–93.

Chen, T., Lee, T.-L., Li, T.-Y., 2014. Mixed volume computation in parallel. Taiwan. J. Math. 18 (1), 93–114.

Chen, T., Li, T.-Y., 2012. Spherical projective path tracking for homotopy continuation methods. Commun. Inf. Syst. 12 (3), 195–220.

Chen, T., Li, T.-Y., 2014. Solutions to systems of binomial equations. Ann. Math. Sil. 28, 7–34.

Drexler, F.-J., 1977. Eine methode zur Berechnung sämtlicher Lösungen von Polynomgleichungssystemen. Numer. Math. 29 (1), 45–58.

Emiris, I.Z., Canny, J.F., 1995. Efficient incremental algorithms for the sparse resultant and the mixed volume. J. Symb. Comput. 20 (2), 117–149.

Gao, T., Li, T.-Y., 2000. Mixed volume computation via linear programming. Taiwan. J. Math. 4 (4), 599–619.

Gao, T., Li, T.-Y., 2003. Mixed volume computation for semi-mixed systems. Discrete Comput. Geom. 29 (2), 257–277.

Gao, T., Li, T.-Y., Wang, X., 1999. Finding all isolated zeros of polynomial systems in $\mathbb{C}^n$ via stable mixed volumes. J. Symb. Comput. 28 (1–2), 187–212.

Gao, T., Li, T.-Y., Wu, M., 2005. Algorithm 846: MixedVol: a software package for mixed-volume computation. ACM Trans. Math. Softw. 31 (4), 555–560.

Garcia, C.B., Zangwill, W.I., 1979. Finding all solutions to polynomial systems and other systems of equations. Math. Program. 16 (1), 159–176.

Gunji, T., Kim, S., Kojima, M., Takeda, A., Fujisawa, K., Mizutani, T., 2004. PHoM–a polyhedral homotopy continuation method for polynomial systems. Computing 73 (1), 57–77.

Hampton, M., Jensen, A., 2011. Finiteness of spatial central configurations in the five-body problem. Celest. Mech. Dyn. Astron. 109 (4), 321–332.

Hampton, M., Moeckel, R., 2009. Finiteness of stationary configurations of the four-vortex problem. Trans. Am. Math. Soc. 361 (3), 1317–1332.

Hauenstein, J.D., Haywood, I., Liddell Jr., A.C., 2014. An a posteriori certification algorithm for newton homotopies. In: Proceedings of the 39th International Symposium on Symbolic and Algebraic Computation. ISSAC '14. ACM, New York, NY, USA, pp. 248–255.

Hauenstein, J.D., Levandovskyy, V., 2011. Certifying solutions to square systems of polynomial-exponential equations. arXiv: 1109.4547 [math].

Hauenstein, J.D., Sottile, F., 2012. Algorithm 921: AlphaCertified: certifying solutions to polynomial systems. ACM Trans. Math. Softw. 38 (4), 28:1–28:20.

Herlihy, M., Shavit, N., 2012. The Art of Multiprocessor Programming. Elsevier.

Huber, B., Sturmfels, B., 1995. A polyhedral method for solving sparse polynomial systems. Math. Comput. 64 (212), 1541–1555.

Kahle, T., 2010. Decompositions of binomial ideals. Ann. Inst. Stat. Math. 62 (4), 727–745.

Kuo, Y.C., Li, T.-Y., 2008. Determining dimension of the solution component that contains a computed zero of a polynomial system. J. Math. Anal. Appl. 338 (2), 840–851.

Lee, T.-L., Li, T.-Y., 2011. Mixed volume computation in solving polynomial systems. Contemp. Math. 556, 97–112.

Lee, T.-L., Li, T.-Y., Tsai, C.-H., 2008. HOM4ps-2.0: a software package for solving polynomial systems by the polyhedral homotopy continuation method. Computing 83 (2), 109–133.

Lee, T.-L., Santoprete, M., 2009. Central configurations of the five-body problem with equal masses. Celest. Mech. Dyn. Astron. 104 (4), 369–381.

Leykin, A., 2009. NAG4m2: numerical algebraic geometry for Macaulay2.

Leykin, A., 2011. Numerical algebraic geometry. J. Softw. Algebra Geom. 3 (1), 5–10.

Li, T.-Y., 2003. Numerical solution of polynomial systems by homotopy continuation methods. In: Ciarlet, P.G. (Ed.), Handb. Numer. Anal., vol. 11. North-Holland, pp. 209–304.

Li, T.-Y., Li, X., 2001. Finding mixed cells in the mixed volume computation. Found. Comput. Math. 1 (2), 161–181.

Li, T.-Y., Sauer, T., Yorke, J.A., 1989. The cheater's homotopy: an efficient procedure for solving systems of polynomial equations. SIAM J. Numer. Anal., 1241–1251.

Li, T.-Y., Tsai, C.-H., 2009. HOM4ps-2.0para: parallelization of HOM4ps-2.0 for solving polynomial systems. Parallel Comput. 35 (4), 226–238.
Li, T.-Y., Wang, X., 1996. The BKK root count in $\mathbb{C}^n$. Math. Comput. Amer. Math. Soc. 65 (216), 1477–1484.
Mehta, D., 2009. Lattice vs. continuum: Landau gauge fixing and 't Hooft–Polyakov monopoles. Ph.D. Thesis, Australasian Digital Theses Program. The Uni. of Adelaide.
Mehta, D., 2011a. Finding all the stationary points of a potential energy landscape via numerical polynomial homotopy continuation method. Phys. Rev. E 84, 025702.
Mehta, D., 2011b. Numerical polynomial homotopy continuation method and string vacua. Adv. High Energy Phys. 2011, 263937.
Mehta, D., He, Y.-H., Hauenstein, J.D., 2012. Numerical algebraic geometry: a new perspective on string and gauge theories. J. High Energy Phys. 1207, 018.
Mehta, D., Nguyen, H., Turitsyn, K., 2014. Numerical polynomial homotopy continuation method to locate all the power flow solutions. Preprint, arXiv:1408.2732.
Mehta, D., Sternbeck, A., von Smekal, L., Williams, A.G., 2009. Lattice Landau gauge and algebraic geometry. PoS QCD-TNT09, 025.
Minkowski, H., 1911. Theorie der konvexen Korper, insbesondere Begrundung ihres Oberflachenbegriffs. Gesammelte Abh., Hermann Minkowski 2, 131–229.
Mizutani, T., Takeda, A., 2008. DEMiCs: A software package for computing the mixed volume via dynamic enumeration of all mixed cells. In: Stillman, M., Verschelde, J., Takayama, N. (Eds.), Software for Algebraic Geometry. In: IMA Vol. Math. Its Appl., vol. 148. Springer, pp. 59–79.
Mizutani, T., Takeda, A., Kojima, M., 2007. Dynamic enumeration of all mixed cells. Discrete Comput. Geom. 37 (3), 351–367.
Morgan, A.P., 1986. A transformation to avoid solutions at infinity for polynomial systems. Appl. Math. Comput. 18 (1), 77–86.
Morgan, A.P., 2009. Solving polynomial systems using continuation for engineering and scientific problems. Class. Appl. Math., vol. 57. Society for Industrial and Applied Mathematics.
Morgan, A.P., Sommese, A.J., 1989. Coefficient-parameter polynomial continuation. Appl. Math. Comput. 29 (2), 123–160.
Morgan, A.P., Sommese, A.J., Wampler, C.W., 1990. Computing singular solutions to nonlinear analytic systems. Numer. Math. 58 (1), 669–684.
Morgan, A.P., Sommese, A.J., Wampler, C.W., 1992. A power series method for computing singular solutions to nonlinear analytic systems. Numer. Math. 63 (1), 391–409.
Morgan, A.P., Sommese, A.J., Watson, L.T., 1989. Finding all isolated solutions to polynomial systems using HOMPACK. ACM Trans. Math. Softw. 15 (2), 93–122.
NVIDIA Corporation. NVIDIA CUDA C Programming Guide. Technical report, July 2011.
Skiena, S.S., 2009. The Algorithm Design Manual. Springer Science & Business Media.
Sommese, A.J., Wampler, C.W., 1996. Numerical algebraic geometry. In: The Mathematics of Numerical Analysis. In: Lect. Appl. Math., vol. 32. AMS, pp. 749–763.
Sommese, A.J., Wampler, C.W., 2005. The Numerical Solution of Systems of Polynomials Arising in Engineering and Science. World Scientific Pub Co Inc.
Takeda, A., Kojima, M., Fujisawa, K., 2000. Enumeration of all solutions of a combinatorial linear inequality system arising from the polyhedral homotopy continuation method. J. Oper. Soc. Jpn. 45, 64–82.
Verschelde, J., 1999. Algorithm 795: PHCpack: a general-purpose solver for polynomial systems by homotopy continuation. ACM Trans. Math. Softw. 25 (2), 251–276.
Verschelde, J., Gatermann, K., Cools, R., 1996. Mixed-volume computation by dynamic lifting applied to polynomial system solving. Discrete Comput. Geom. 16 (1), 69–112.
von Helmholtz, H., 1858. Uber Integrale der hydrodynamischen Gleichungen welche den Wirbelbewegungen entsprechen. Crelle's J. Math. 55, 25–55.
Watson, L.T., Billups, S.C., Morgan, A.P., 1987. Algorithm 652: HOMPACK: a suite of codes for globally convergent homotopy algorithms. ACM Trans. Math. Softw. 13 (3), 281–310.