An efficient algorithm for global interval solution of nonlinear algebraic equations and its GPGPU implementation

LIN Dang · CHEN Liangyu

Received: date / Accepted: date

Abstract  Solving nonlinear algebraic equations is a classic mathematics problem, and common in scientific researches and engineering applications. There are many numeric, symbolic and numeric-symbolic methods of solving (real) solutions. Unlucky, these methods are constrained by some factors, e.g., high complexity, slow serial calculation, and the notorious intermediate expression expansion. Especially when the count of variables is larger than six, the efficiency is decreasing drastically. In this paper, according to the property of physical world, we pay attention to nonlinear algebraic equations whose variables are in fixed constraints, and get meaningful real solutions. Combining with parallelism of GPGPU, we present an efficient algorithm, by searching the solution space globally and solving the nonlinear algebraic equations with real interval solutions. Furthermore, we realize the Hansen-Sengupta method on GPGPU. The experiments show that our method can solve many nonlinear algebraic equations, and the results are accurate and more efficient compared to traditional serial methods.

Keywords  nonlinear algebraic equations · branch and bound · interval arithmetic · GPGPU · Hansen-Sengupta method

1 Introduction

Construction theory and algorithm for algebraic equations is very classical and important. Many problems in scientific researches and engineering fields, eventually are transferred into equations to be solved. For linear equations, there
are lots of mature researches and tools. By the common desktop computers, one can easily solve linear equations with thousands of variables or higher. For nonlinear algebraic equations, there are also many symbolic and numeric methods of solving solutions. The methods of exact solutions for nonlinear algebraic equations can be divided into two categories, symbolic algorithm and symbolic-numerical algorithm. The symbolic methods include Groebner base [13], Wu’s method [24], resultant elimination method [25] and others. In recent years, some symbolic-numerical methods have also been developed, to preserve the accuracy of symbol calculation and enjoy the benefits of numerical computation. Many of the above methods are also implemented into specific software packages, such as Groebner package in Maple, WSolve [21], Discoverer [20], GAS [25], Epsilon [23] and more. However, with the rapid development of physical world, such as applications in space control, intelligent transportation, program verification, internet of things and etc, it has brought big challenges for solving the nonlinear algebraic equations. The current methods and tools have the following limitations. The first is insufficient computation. Most of the specific packages described above, currently, can only be executed with one core. Worse, it is not easy to extend these packages for execution on cluster computers or other high performance environments. The second is insufficient memory utilization caused by intermediate expression expansion. In successive computation, it generates many intermediate symbolic expressions, which cannot be omitted due to preserve strict accuracy. There is also no floating number truncation and rounding error. Thus, these intermediate expressions make the physical memory exhausted quickly and computation aborted.

Currently, the computer hardware has passed the multi-core age, and stridden to the many-core age. Many desktop computers have equipped with multiple CPU cores, usually four or eight cores, and the Intel CPU with MIC architecture has 32 cores or more. More lucky, CPU is not the sole calculation part in modern computers. GPU (Graphic Processing Unit) Computing has also been widely applied and popularized. The optimized GPGPU program can work very fast. However, most of traditional programs in symbol calculation still follow the old serial executive rule even with good CPU and GPU devices. Therefore, combining parallel computing and symbolic computation, and using parallel computing to accelerate symbolic computation process, have become more important in current researches, e.g., Wang’s Parallel polynomial operations on SMPs [19], Lin’s parallel computation for polynomial GCD [22], Sato’s on the parallel computation of comprehensive groebner systems [10], Moreno Maza’s multithreaded parallel implementation of arithmetic operations modulo a triangular set [17], Kobayashi’s work of the parallel implementation of the formal verification language-Isabelle. Using GPGPU technology to solve symbolic computation problems, is also emerging with the utilization of thousands of cores. Some researchers have make progress on theoretical analysis and calculation under many-core environments, like [15,18].

In this paper, based on the branch and bound method [2] and interval arithmetic [11,10,28], we present an efficient algorithm to get potential real
solutions for nonlinear algebraic equation with integer or floating number coefficients in GPGPU environment. It’s easily observe that, many problems in physical world can be transformed into nonlinear algebraic equations with two characteristics. One is each variable has limited value ranges. The other is only real solutions are emphasized and required. Therefore, we apply the simple idea of branch and bound method, and do successively iterative partition in the global solution space to remove unfeasible parts. We also use the Hansen-Sengputa method \[4,5,7,13\] to accelerate the convergence of solution space. The main advantage of our algorithm is complete and sound, namely, it can quickly find all subdivisions who potentially contain real solutions. We globally search the solution space, and use reliable interval arithmetics without any floating number truncation or rounding error to keep interval solutions right. Moreover, whole calculations are executed not only in numerical way but also in parallel, with the support from thousands of cores in GPGPU. This also avoids the notorious problem of intermediate expression expansion.

The structure of the paper is listed as follows. In section 2, we introduce the fundamental knowledge of interval arithmetics and Hansen-Sengputa method. In section 3, the main algorithm and related sub-algorithms are presented. In section 4 we present experiments and data statistics. The conclusion is in section 5.

2 Preliminaries

2.1 Polynomial equation

A polynomial is an expression consists of variables (or indeterminates) and coefficients, involving only the operations of addition, subtraction, multiplication, and variables with non-negative integer exponents. A typical example of a single indeterminate \(x\) is \(x^2 - 3x + 4\).

A polynomial function is a function that can be defined by evaluating a polynomial. For example, the function

\[P = a_0 + a_1x + \cdots + a_nx^n.\]  

(1)

The corresponding polynomial function can construct the equation

\[f_P(x) = a_0 + a_1x + \cdots + a_nx^n = 0.\]  

(2)

If \(r \in \mathbb{R}\), and \(f_P(r) = 0\), it can call \(r\) the root of the polynomial equation, or the zero of the associated function. The relation of the root of polynomial function and the polynomial is: if \(r \in \mathbb{R}\) is a root of \(f_P\) if and only if \(X - r\) divides \(f_P\), that is, there exist another polynomial \(Q(x)\), such as: \(f_P = (X - r)Q(x)\). If a positive integer \(k\), which \(f_P = (X - r)^kQ(x)\), then call \(r\) a multiple root of \(f_P\).

A real root means the root of an equation is a real number. Then the real root isolation of the polynomial system is presenting the intersection intervals of the real number line, which contain all the real roots of the polynomial, and each interval has one and only one real root.
2.2 Interval arithmetic

A real interval is a set of real numbers with the property that any number that lies between two numbers is also included in the set. For example, let $X$ be a real interval and let

$$ X = [x, \overline{x}] = \{ x \in \mathbb{R} : x \leq x \leq \overline{x} \}, $$

(3)

where $x$ is the lower bound, $\overline{x}$ is the upper bound respectively. The set of all interval numbers can also be defined as

$$ \mathbb{IR} = \{ [x, \overline{x}] : x, \overline{x} \in \mathbb{R}, x \leq \overline{x} \}. $$

(4)

Let $c$ be the midpoint of an interval, we have

$$ c = \text{mid}(x) = (x + \overline{x})/2. $$

(5)

Let $r$ be the radius of an interval, we also have

$$ r = \text{rad}(x) = (x - \overline{x})/2. $$

(6)

The basic arithmetic operations of interval computing are defined in $\mathbb{IR}$, so the result of interval calculation contains all possible real solutions, $x = [x, \overline{x}], y = [y, \overline{y}]$. These basic operations are listed as follow.

$$ x + y = [x + y, \overline{x} + \overline{y}], $$

(7)

$$ x - y = [x - \overline{y}, \overline{x} - y], $$

(8)

$$ x \times y = [\min\{xy, \overline{x}y, \overline{x}\overline{y}, \overline{x}\overline{y}\}, \max\{xy, \overline{x}y, \overline{x}\overline{y}, \overline{x}\overline{y}\}], $$

(9)

$$ 1/x = \begin{cases} [1/\overline{x}, 1/x] & \text{if } x > 0 \text{ or } \overline{x} < 0, \\ -\infty & \text{if } x < 0 \text{ or } \overline{x} > 0. \end{cases} $$

(10)

$$ x \div y = x \times 1/y. $$

(11)

In the above division operations, the case of zero division is undefined.

We make use of extended interval arithmetic introduced by Hanson [8] and Kahan [29] and deal the division with

$$ x/y = \begin{cases} [\overline{x}/y, +\infty] & : \overline{x} \leq 0 \text{ and } y = 0, \\ [-\infty, \overline{x}/y] \cup [\overline{x}/y, +\infty] & : \overline{x} \leq 0 \text{ and } y < 0 < \overline{y}, \\ [-\infty, +\infty] & : x < 0 < \overline{y}, \\ [-\infty, \overline{x}/y] & : x \geq 0 \text{ and } y = 0, \\ [-\infty, \overline{x}/y] \cup [\overline{x}/y, +\infty] & : x \geq 0 \text{ and } y < 0 < \overline{y}, \\ \overline{x}/y, +\infty] & : x \leq 0 \text{ and } y = 0. \end{cases} $$

(12)
Definition 1 Let $f$ be an arithmetic representation in $\mathbb{R}[x_1, x_2, \ldots, x_n]$, and transform all operands to intervals, thus all arithmetic operations can be seen as the corresponding interval operations, denote as $F$. Then $F : I(\mathbb{R})^n \rightarrow I(\mathbb{R})$ can be called an interval evaluation, simply, $Y \supseteq f(x) : x \in X$ define the evaluation of $f(x)$ in $X$.

Generally, the methods of interval evaluation of a function in a given region can be classified into three categories:

1) Using the algebra method or analysis directly, calculate the interval value of the function in a given region.

2) Dividing an given interval into a number of subintervals, calculate the interval function value on each subinterval, and choose the maximum and minimum as the result of the original function evaluation bound.

3) Using the optimization methods for function in a given range, and choose the maximum and minimum as evaluation bound.

Definition 2 Let $f$ be an arithmetic representation in $\mathbb{R}[x_1, x_2, \ldots, x_n]$. Given a space $I = [x_1, x_1], [x_2, x_2], \ldots, [x_n, x_n]$ with $n$ tuples, the function $f$ can be calculated by interval evaluation and the result is also in interval, namely, $[f, \bar{f}]$. If $0 \in [f, \bar{f}]$, it means $f$ may contain real solutions, and we call the tuple $C$ as an configuration of $f$; otherwise, $I$ is an infeasible part of $f$.

Definition 3 An interval matrix is a matrix whose elements are intervals. We denote the interval matrix $[X]$ with its elements $[X]_{ij}$, and have

$$[X] = \begin{bmatrix}
[x_{11}, \bar{x}_{11}] & [x_{12}, \bar{x}_{12}] & \cdots & [x_{1n}, \bar{x}_{1n}] \\
[x_{21}, \bar{x}_{21}] & [x_{22}, \bar{x}_{22}] & \cdots & [x_{2n}, \bar{x}_{2n}] \\
\vdots & \vdots & \ddots & \vdots \\
[x_{n1}, \bar{x}_{n1}] & [x_{n2}, \bar{x}_{n2}] & \cdots & [x_{nn}, \bar{x}_{nn}]
\end{bmatrix}. \quad (13)$$

2.3 Hansen-Sengupta method

Hansen-Sengupta method is an algorithm using interval arithmetic to compute and bound the zeros of nonlinear algebraic equations. It provides guaranteed bounds on all zeros in a given region. Hansen-Sengupta method is closely related to the interval Newton method [5,8,9].

Let $f$ be a function of $n$ variables, $x = (x_1, x_2, \ldots, x_n)^T$, and function $f(x) = 0$. \quad (14)

We expand $f$ by using Taylor’s theorem and expanding $f(y)$ about $x$, then obtain

$$f(x) + J(\xi)(z - x) = f(y) = 0, \quad (15)$$

where $J(\xi)$ is a Jacobian matrix evaluated at a point $\xi \in X$, and the point set $z$ contains all solution of $y$. We would like to know the set $z$, but it’s difficult to represent this set.
Moore [10] first found out to calculate the inverse of the Jacobian matrix $J(X)$ in interval Newton method. But it require an interval matrix $M$ containing every interval element. Hansen [14] pointed out that it is not necessary to find an interval inverse, instead, a Gaussian elimination procedure could be used as in the real counterpart.

Define $J_c$ as the center of $J(X)$, and each element of $J_c$ is the midpoint of the corresponding interval element of $J(X)$. A nonlinear algebraic equations with interval coefficients such as [15] is best solved by multiplying an approximate inverse of $J_c$. Let $B$ be this approximation matrix.

We thus rewrite (15) as

$$Bf(x) + BJ(X)(z - x) = 0.$$  \hfill (16)

The products $Bf(x)$ and $BJ(X)$ are computed in interval arithmetic to bound rounding errors.

Krawczyk introduced a variation of interval Newton method to avoid Gaussian elimination on interval matrix by not attempting to obtain a sharp solution of (16). Based on this method, we can compute the box with

$$K(X) = x - Bf(x) + [I - BJ(X)][X - x].$$  \hfill (17)

If a solution $y$ is contained in a box $X$, then it is also contained in $K(X)$. Since $K(X)$ may not be contained in $X$, we can use the iteration

$$X^{(i+1)} = X^i \cap K(X^i) \quad (i = 0, 1, \ldots, n).$$  \hfill (18)

The Krawczyk method computes the box $K(X)$ in each iteration. However, it is not the smallest box.

We now present Hansen-Sengupta method. The box is generally smaller than $K(X)$. Each iteration of Hansen-Sengupta method tends to produce a greater reduction of the current box than Krawczyk’s method does, also fewer steps are required for convergence.

Let $g = Bf(x)$ and $P = BJ(X)$, thus (17) is rewritten as

$$g + P(z - x) = 0.$$  \hfill (19)

Note that the interval matrix $P$ is

$$P = L + D + U,$$  \hfill (20)

where the matrices $L$, $D$, and $U$ are lower triangular, diagonal, and upper triangular, respectively. We thus rewrite (19) as

$$Y = x - D^{-1}[g + L(X' - x) + U(X - x)],$$  \hfill (21)

$$X' = Y \cap X.$$  \hfill (22)

After each component $Y_i (i = 1, 2, \ldots, n)$ is obtained, it intersects with $X_i$ so that the new result $X'_i = Y_i \cap X_i$ can be used in obtain $Y_{i+1}, Y_{i+2}, \ldots, Y_n$, thus we compute componentwise, for $i = 1, 2, \ldots, n,$
An efficient algorithm for global interval solution of nonlinear algebraic equations

\[ Y_i = x_i - (D_{ii})^{-1} \left[ g_i + \sum_{j=1}^{i-1} P_{ij} (X'_j - x_j) + \sum_{j=i+1}^{n} P_{ij} (X_j - x_j) \right] \]  

(23)

\[ X'_i = Y_i \cap X_i. \]  

(24)

This step is done for each \( i = 1, 2, \ldots, n \) and the process is iterated until the new box is sufficiently small.

Note that even though \( P \) is supposed to approximate the identity matrix, the interval \( D_{ii} \) may contain zero. We simply use extended interval arithmetic to compute \( Y_i \). The intersection \[24\] then produces a finite result.

2.4 Gauss-Jordan algorithm

Gauss-Jordan Algorithm \[6\] is a classic method for matrix inversion. Let \( I_n \) be an identity matrix of size \( n \). Consider a matrix \( A \) of size \( n \), and an augment \( A \) to get the matrix \( [C] = [A|I_n] \), thus we can do column rank elimination of Gauss-Jordan method on \( C \) for \( k = 1, 2, \ldots, n - 1 \).

Firstly, select the main element by column maximum \( |A_{ik}| \). Exchange the elements of \( k \)-th row and \( ik \)-th row of \( [C] = [A|I_n] \).

Secondly, calculate the main element

\[ C_{kj} \leftarrow C_{kj} / A_{kk}, (j = k, k + 1 \ldots, 2n). \]  

(25)

Finally, do elimination calculation

\[ C_{ij} \leftarrow C_{ij} - A_{ik} \times C_{kj}, (i = 1, 2, \ldots, n; i \neq k; j = k + 1, k + 2, \ldots, 2n). \]  

(26)

By doing elimination calculation on each row successively, the left half becomes the \( I_n \), and the right half becomes the desired inverse of \( A \).

3 A nonlinear algebraic equations interval algorithm based on branch and bound method

3.1 The framework of our algorithm

Our goal is to solve the problem of nonlinear algebraic equations whose each variable is in a given region. Assume the variables are \( x_1, x_2, \ldots, x_n \), and their regions are \([d_i, \bar{d}_i]\) respectively. So an initial solution space can be constructed easily by combining these regions together. According to the branch and bound method, we can divide the solution space to \( 2^n \) rectangular subdivisions \[27\] by splitting the region to two half parts for each variable and making combinations on these new parts. Each of these subdivisions can be deemed as a potential solution space. Through certain mathematical calculation, some subdivisions can be proved no roots and removed. The next step is using the remained
subdivisions as original inputs. We can do the division and determination process iteratively until the interval width of subdivisions is sufficiently small.

Since the computation on intervals is “loose”, the final result may have too many configurations to be understood. Moreover, the intervals are very small, so that many subdivisions are nearly overlapped. Thus, we can do backtracking on these configurations to get more readable result.

Our algorithm is described as Algorithm 1 and its flowchart is showed in Fig.1.

\textbf{Algorithm 1:} Solving nonlinear algebraic equations system based on branch and bound method

\textbf{Input:} The algebraic equations \( f(x_1, x_2, \ldots, x_n) \), its variable regions and the desired width or result intervals

\textbf{Output:} The feasible solution set \( M_k \), where each element is an configuration with \( n \) intervals and may contain feasible solutions.

1. According to the original regions, construct an interval configuration as the initial solution set \( M_0 \).

2. For each element from \( M_0 \) Do
   (I) divide the configuration into \( 2^n \) subdivisions by using branch and bound method;
   (II) make evaluation on subdivisions and remove the infeasible ones;
   (III) use Hansen-Sengupta methods to compute and bound the zeros of nonlinear functions, and remove the infeasible ones;
   (IV) add the remained subdivisions into \( M_1 \).

3. Do the operations described in the step 2 iteratively and get the \( M_0, M_1, \ldots, M_k \), calculation is terminated if one of the conditions is satisfied:
   (I) the \( M_k \) is an empty set, then we can assert that there is no real solution for \( f(x_1, x_2, \ldots, x_n) \);
   (II) \( M_k \) is satisfied with the desired interval width, then \( M_k \) is the final result.

In successive iterations, for one configuration with \( n \) intervals, it generates \( 2^n \) new subdivisions according to the branch and bound method. So we need apply the Hansen-Sengupta method to remove more infeasible branches. While using Hansen-Sengupta method, we involve several sub-algorithms including interval matrix multiplication, Gauss-Jordan method for matrix inverse.

We obtain feasible interval configurations by iterate algorithms then merge the interval configurations by backtracking Algorithm 5 and isolate the final roots within intervals.
3.2 Polynomial interval evaluation and branch and bound method on GPGPU

Firstly, we consider the problem of polynomial interval evaluation. Our research goal is nonlinear algebraic equations with integer or floating number coefficients. To ensure the result strict accurate, all calculations are done in interval style. Detailedly, on GPGPU, we need carefully deal with the upper and lower bounds of basic calculations. In CUDA programming, there is two commands supported by IEEE standard to ensure the correctness of interval evaluation. One is \(\text{rd}\), round-down, and the other is \(\text{ru}\), round-up. Therefore, the basic calculations in GPGPU follow these rules:

\[
x + y = [\text{rd}(x + y), \text{ru}(x + y)],
\]

(27)
\[ x - y = [\text{rd}(x - y), \text{ru}(x - y)], \quad (28) \]

\[ x \times y = [\text{rd}(\min\{xy, yx, yx\}), \text{ru}(\max\{xy, yx, yx\})]. \quad (29) \]

Note that calculation in Algorithm 1 only involves polynomial evaluation, including addition, subtraction and multiplication, but no any division.

To solve a nonlinear algebraic equation, the branch and bound method is a good way to remove infeasible parts from global solution space. For a configuration with \( n \) interval tuples, we present Algorithm 2 to demonstrate one round of branch and bound method.

**Algorithm 2:** One round of branch and bound method on GPGPU

**Input:** \( f(x_1, x_2, \ldots, x_n) \), and the initial configuration \( I = [[x_1, l_1], [x_2, l_2], \ldots, [x_n, l_n]] \).

**Output:** Configurations contain potentially feasible solution.

1. Split each variable interval into two parts, namely, for \( I \) from \( n \) intervals to \( 2^n \) intervals, \( I_{11} = [x_1, \text{mid}(x_1 + x_1)], I_{12} = [\text{mid}(x_1 + x_1), x_1], \ldots, I_{n1} = [x_n, \text{mid}(x_n + x_n)], I_{n2} = [\text{mid}(x_n + x_n), x_n] \). Make combination on these new intervals for \( n \) variables, and get new \( 2^n \) configurations.
2. Put \( 2^n \) configurations into \( 2^n \) threads of GPGPU, and each thread calculates \( f(x_1, x_2, \ldots, x_n) \) with one configuration respectively.
3. According to the function interval evaluations, remove infeasible configurations (not contain zero) and output the remains.

It is remarked that if \( 2^n \) exceeds the limit of GPGPU, the task can be divided into several jobs, which can be submitted repeatedly.

After several rounds of branch and bound method, the infeasible parts of global solution space are removed, while those feasible configurations potentially contain solutions are left.

### 3.3 Hansen-Sengupta method based on GPGPU

Hansen-Sengupta method is a good method to determine whether a function has roots in a given box. If yes, this method can generate a more tighter box
for roots of input function. We present the Algorithm 3 to demonstrate this method implementation on GPGPU.

Algorithm 3: Hansen-Sengupta method on GPGPU

Input: $F = [f_1, f_2, \ldots, f_n] \in \mathbb{Q}[x_1, x_2, \ldots, x_n]$, the initial box $B = X_1 \times X_2 \times \cdots \times X_n$. 

Output: The new box $B^*$ where

1) $\forall $( $x_1, x_2, \ldots, x_n$) $\in B$, if $F(x_1, x_2, \ldots, x_n) = 0$, then ($x_1, x_2, \ldots, x_n$) $\in B^*$.
2) if $B^* \subseteq B$, there exists ($x_1, x_2, \ldots, x_n$) $\in B^*$, and $F(x_1, x_2, \ldots, x_n) = 0$.

Assume the Jacobian matrix of $F$ is $J$. Substitute the intervals $X_1, X_2, \ldots, X_n$ into $J$, then get a Jacobian matrix $J(X)$ whose entries are intervals.

2 Calculate center of $J(X)$, then calculate the approximate inverse $A = \text{Inv(mid}(J(X))))$ with Gauss-Jordan method.
3 Calculate $M = A \ast J(X)$, $R(x) = A \ast F(x)$ by interval matrix multiplication.
4 For $i = 1, 2, \ldots, n$, Do
\[ P_i = R(x) - \sum_{j=1}^{i-1} M_{ij}(X_j' - x_j) - \sum_{j=i+1}^{n} M_{ij}(X_j - x_j), \]
\[ N_i(x, X) = x_i + P_i/M_{ii}. \]
5 For $i = 1, 2, \ldots, n$, Do
\[ X_i' = X_i \cap N_i(X). \]
6 Output $B^* = X_1' \times X_2' \times \cdots \times X_n'$.

For interval matrix multiplication in Algorithm 3 it is different to common matrix multiplication since all elements in interval matrices are interval. We present the detail of interval matrix multiplication with Algorithm 4.

Algorithm 4: Interval matrix multiplication

Input: two interval matrices $A \in \mathbb{R}^{m \times k}, B \in \mathbb{R}^{k \times n}$, where each elements is an interval like $[d_{ij}, d_{ij}]$.

Output: the interval matrix $C = A \times B$.
1 Initialization interval matrix $C$.
2 For $i = 1, 2, \ldots, m$
3 For $j = 1, 2, \ldots, n$
4 For $u = 1, 2, \ldots, k$
\[ C_{ij} = C_{ij} = \min\{A_{iu}B_{uj}, A_{iu}B_{uj}, A_{iu}B_{uj}, A_{iu}B_{uj}\}, \]
\[ C_{ij} = C_{ij} = \max\{A_{iu}B_{uj}, A_{iu}B_{uj}, A_{iu}B_{uj}, A_{iu}B_{uj}\}. \]
5 Output $C$. 

1. Assume the Jacobian matrix of $F$ is $J$. Substitute the intervals $X_1, X_2, \ldots, X_n$ into $J$, then get a Jacobian matrix $J(X)$ whose entries are intervals.
2. Calculate center of $J(X)$, then calculate the approximate inverse $A = \text{Inv(mid}(J(X))))$ with Gauss-Jordan method.
3. Calculate $M = A \ast J(X)$, $R(x) = A \ast F(x)$ by interval matrix multiplication.
4. For $i = 1, 2, \ldots, n$, Do
   \[ P_i = R(x) - \sum_{j=1}^{i-1} M_{ij}(X_j' - x_j) - \sum_{j=i+1}^{n} M_{ij}(X_j - x_j), \]
   \[ N_i(x, X) = x_i + P_i/M_{ii}. \]
5. For $i = 1, 2, \ldots, n$, Do
   \[ X_i' = X_i \cap N_i(X). \]
6. Output $B^* = X_1' \times X_2' \times \cdots \times X_n'$.
3.4 Backtracking algorithm based on GPGPU

Through several successive rounds of branch and bound methods, the infeasible configurations are removed while the feasible ones are left. Since the box of these left configurations is too tiny to be recognized, we can do backtracking to get "bigger" and readable configurations. Because of bisection in the successive branch and bound method, we present the Algorithm 5 to demonstrate the backtracking process of feasible configurations from \(n\)-th round up to \((n - 1)\)-th round.

**Algorithm 5: Backtracking algorithm on GPGPU**

**Input:** A configuration of \(n\)-th round, \([a_1, \bar{a}_1], [a_2, \bar{a}_2], \ldots, [a_n, \bar{a}_n]\].

**Output:** A configuration of \(n - 1\)-th round.

1. **For** each variable interval **Do**
   
   \[x_i = \left\lceil \frac{a_i}{2(\bar{a}_i - a_i)} \right\rceil,\]
   \[y_i = \left\lfloor \frac{a_i}{2(\bar{a}_i - a_i)} \right\rfloor,\]

2. The \((n - 1)\)th interval \(b_i\) of \(x_i\) can be deduced with
   
   \[b_i = \begin{cases} 
   \frac{a_i + \bar{a}_i}{2}; & x \text{ is integer,} \\
   a_i; & x \text{ is not integer.}
   \end{cases}\]

3. output the new configuration \([b_1, \bar{b}_1], [b_2, \bar{b}_2], \ldots, [b_n, \bar{b}_n]\].

Obviously, multiple configurations of \(n\)-th rounds can restore up to the same configuration of \((n - 1)\)-th round. So we can remove the redundant configurations of \((n - 1)\)-th rounds and execute the backtracking algorithm successively to get more readable configurations.

3.5 Algorithm analysis

Our algorithm is based on branch and bound method to globally search the feasible solutions from initial space. Since the computation involves floating-point arithmetic, we use interval replacing number and use rounding down the lower bound and rounding up the upper bound, to ensure computation correctness. Hansen-Sengupta method is used to determine the root and reduce the solution space. The backtracking algorithm can isolate finite real roots. Therefore, our algorithm is right and complete. Additionally, the nonlinear algebraic equations studied in this paper have finite real solution, and the initial space is limited. So the algorithm will stop when the interval width of feasible configurations meets the desired goal.

In branch and bound method, \(n\) threads compute function interval evaluation in parallel. From view of parallel computing, the time cost of this algorithm is \(O(n^2)\). Similarly, the time complexity for each thread in Hansen-Sengupta algorithm is inverse matrix computing complexity \(O(n^3)\), plus with
interval matrix multiplication complexity $O(n^3)$, and Gaussian elimination complexity $O(n)$. So the total time complexity of Hansen-Sengupta algorithm is $T(n) = 2O(n^3) + O(n)$, as $O(n^3)$.

It is noted that our parallel algorithm does not decrease time complexity. Because we adopt coarse parallel pattern to improve whole performance. Each thread needs do whole calculation independently. There is no message communication and synchronization between different threads. This saves almost whole cost of communication.

4 Experiment

The detailed information of computational resource in our experiments is Intel Core i7-7700 CPU of 3.60GHz, 16GB memory, and GPGPU NVIDIA GeForce 1070 GTX with 8G video memory, 15 Multiprocessors, 1920 Cuda Cores. The version of CUDA programming is 7.5.

To show the feasibility and effectiveness of our algorithm, we collect 55 equations from real applications and academic references with one constraint that the variable count is not more than 9.

We select several typical equations for analysis and detailed description. The full equation descriptions and statistics are showed in Table 9, 10.

4.1 Analysis for typical equations

Example 1 Katsura8, a 9-dimensional economics problem, consider the system $F(x_1, \ldots, x_9)$:

\[
\begin{align*}
2x_9 + 2x_8 + 2x_7 + 2x_6 + 2x_5 + 2x_4 + 2x_3 + 2x_2 + 2x_1 & = 0, \\
2x_9^2 + 2x_8^2 + 2x_7^2 + 2x_6^2 + 2x_5^2 + 2x_4^2 + 2x_3^2 + 2x_2^2 + x_1^2 & = 0, \\
2x_9x_8 + 2x_9x_7 + 2x_8x_6 + 2x_7x_5 + 2x_6x_4 + 2x_5x_3 + 2x_4x_2 + 2x_3x_1 & = 0, \\
2x_9x_7 + 2x_8x_6 + 2x_7x_5 + 2x_6x_4 + 2x_5x_3 + 2x_4x_2 + 2x_3x_1 + x_2^2 - x_3 & = 0, \\
2x_9x_6 + 2x_8x_5 + 2x_7x_4 + 2x_6x_3 + 2x_5x_2 + 2x_4x_1 + 2x_3x_2 - x_4 & = 0, \\
2x_9x_5 + 2x_8x_4 + 2x_7x_3 + 2x_6x_2 + 2x_5x_1 + 2x_4x_2 + x_3^2 - x_5 & = 0, \\
2x_9x_4 + 2x_8x_3 + 2x_7x_2 + 2x_6x_1 + 2x_5x_2 + 2x_4x_3 - x_6 & = 0, \\
2x_9x_3 + 2x_8x_2 + 2x_7x_1 + 2x_6x_2 + 2x_5x_3 + x_4^2 - x_7 & = 0, \\
2x_9x_2 + 2x_8x_1 + 2x_7x_2 + 2x_6x_3 + 2x_5x_4 - x_8 & = 0.
\end{align*}
\]  

The initial interval for each variable is $[-1, 1]$. Using the branch and bound method iteratively, the number of interval boxes is decreasing after eight rounds. The statistics are showed in Table 11.
Table 1: B&B result of katsura8

| round | boxes         | time(s) | width of interval |
|-------|---------------|---------|-------------------|
| 1     | 1             | 0       | 2                 |
| 2     | 511           | 0.001   | 1                 |
| 3     | 4847          | 0.01    | 0.5               |
| 4     | 92880         | 0.2     | 0.25              |
| 5     | 2549461       | 8       | 0.125             |
| 6     | 76736464      | 38      | 0.0625            |
| 7     | 794760638     | 6556    | 0.03125           |
| 8     | 430650061     | 11203   | 0.015625          |

We continue to do branch and bound method and use Hansen-Sengupta method after the 8th round. We define the boxes number after Hansen-Sengupta method as new boxes. The statistics are showed in Table 2.

Table 2: Hansen-Sengupta result of katsura8

| round | origin boxes      | new boxes      | time(s) |
|-------|-------------------|----------------|---------|
| 8     | 430650061         | 4900601        | 1271    |
| 9     | 30098623          | 4347544        | 137     |
| 10    | 42617540          | 5532835        | 310     |
| 11    | 64267381          | 7946423        | 380     |
| 12    | 105056246         | 11571157       | 478     |
| 13    | 165616092         | 15474734       | 750     |
| 14    | 243760141         | 1871270        | 1013    |
| 15    | 295102270         | 20306174       | 1359    |
| 16    | 343840882         | 20242072       | 1624    |

The number of interval boxes is basically stable, then we can merge the interval boxes by backtracking algorithm, and isolate 36 real interval roots, width is 0.125. The statistics are showed in Table 2.
Example 2 Noon9, a neural network Lotka-Volterra system. Consider the system \( F(x_1, \ldots, x_9) \)

\[
\begin{align*}
\{ & x_1^2 + x_1x_3 + x_1x_5 + x_1x_7 + x_1x_9 + x_1x_2 + x_1x_4 + x_1x_6 - 1.1x_1 + 1 = 0, \\
& x_2^2 + x_2x_3 + x_2x_4 + x_2x_5 + x_2x_6 + x_2x_7 + x_2x_8 + x_2x_9 - 1.1x_2 + 1 = 0, \\
& x_3^2 + x_3x_2 + x_3x_4 + x_3x_5 + x_3x_6 + x_3x_7 + x_3x_8 + x_3x_9 - 1.1x_3 + 1 = 0, \\
& x_4^2 + x_4x_2 + x_4x_3 + x_4x_5 + x_4x_6 + x_4x_7 + x_4x_8 + x_4x_9 - 1.1x_4 + 1 = 0, \\
& x_5^2 + x_5x_2 + x_5x_3 + x_5x_4 + x_5x_6 + x_5x_7 + x_5x_8 + x_5x_9 - 1.1x_5 + 1 = 0, \\
& x_6^2 + x_6x_2 + x_6x_3 + x_6x_4 + x_6x_5 + x_6x_7 + x_6x_8 + x_6x_9 - 1.1x_6 + 1 = 0, \\
& x_7^2 + x_7x_2 + x_7x_3 + x_7x_4 + x_7x_5 + x_7x_6 + x_7x_8 + x_7x_9 - 1.1x_7 + 1 = 0, \\
& x_8^2 + x_8x_2 + x_8x_3 + x_8x_4 + x_8x_5 + x_8x_6 + x_8x_7 + x_8x_9 - 1.1x_8 + 1 = 0, \\
& x_9^2 + x_9x_2 + x_9x_3 + x_9x_4 + x_9x_5 + x_9x_6 + x_9x_7 + x_9x_8 + x_9x_9 - 1.1x_9 + 1 = 0.
\end{align*}
\]

The initial interval for each variable is \([-8, 8]\). Using the branch and bound method iteratively, the number of interval boxes is decreasing after ten rounds.

The statistics are showed in Table 3.

| backtracking boxes | time(s) | width of interval |
|--------------------|---------|-------------------|
| 20242072           | 29      | 0.0006103515635   |
| 18711270           | 26      | 0.001230703135    |
| 11511157           | 18      | 0.00244140625     |
| 7665686            | 13      | 0.0048828125      |
| 2221856            | 7       | 0.00976525015     |
| 456692             | 2       | 0.01953125        |
| 76643             | 0.3     | 0.00390625        |
| 14006             | 0.01    | 0.00390625        |
| 2957              | 0.001   | 0.0078125         |
| 691               | 0.001   | 0.015625          |
| 246               | 0.001   | 0.03125           |
| 79               | 0.001   | 0.0625            |
| 36                | 0.001   | 0.125             |
| 36                | 0.001   | 0.25              |

Table 4 B&B result of noon9

| round | boxes | time(s) | width of interval |
|-------|-------|---------|-------------------|
| 1     | 1     | 0       | 16                |
| 2     | 512   | 0.001   | 8                 |
| 3     | 5120  | 0.001   | 4                 |
| 4     | 42176 | 0.01    | 2                 |
| 5     | 13511 | 0.2     | 1                 |
| 6     | 222941| 1.5     | 0.5               |
| 7     | 1156044| 8 | 0.25              |
| 8     | 11471246| 102 | 0.125             |
| 9     | 33925040| 300 | 0.0625            |
| 10    | 28775053| 285 | 0.03125           |
We continue to do branch and bound method and use Hansen-Sengupta method after the 10th round. The statistics are showed in Table 5.

| round | origin boxes | new boxes | time(s) |
|-------|--------------|-----------|---------|
| 10    | 28775053     | 477       | 78      |

It is easily seen that the Hansen-Sengupta method is efficient for the function., then we can merge the interval boxes by backtracking algorithm, and isolate 19 real interval roots, width is 0.0625. The statistics are showed in Table 6.

| backtracking boxes | time(s) | width of interval |
|--------------------|---------|-------------------|
| 477                | 0.001   | 0.03125           |
| 19                 | 0.001   | 0.0625            |
| 19                 | 0.001   | 0.125             |

Example 3 Kinema, a robot kinematics problem. Consider the system $F(x_1, \ldots, x_9)$:

$$
\begin{align*}
    x_1^2 + x_2^3 + x_3^2 - 12x_1 - 68 &= 0, \\
    x_2^2 + x_4^3 + x_5^2 - 12x_5 - 68 &= 0, \\
    x_6^2 + x_8^2 + x_9^2 - 24x_8 - 12x_9 + 100 &= 0, \\
    x_1x_4 + x_2x_5 + x_3x_6 - 6x_1 - 6x_5 - 52 &= 0, \\
    x_1x_7 + x_2x_8 + x_3x_9 - 6x_1 - 12x_8 - 6x_9 + 64 &= 0, \\
    x_4x_7 + x_5x_8 + x_6x_9 - 6x_5 - 12x_8 - 6x_9 + 32 &= 0, \\
    2x_2 + 2x_3 - x_4 - x_5 - 2x_6 - x_7 - x_9 + 18 &= 0, \\
    x_1 + x_2 + 2x_3 + 2x_4 + 2x_6 - 2x_7 + x_8 - x_9 - 38 &= 0, \\
    x_1 + x_3 - 2x_4 + x_5 - x_6 + 2x_7 - 2x_8 + 8 &= 0.
\end{align*}
$$

The initial interval for each variable is $[-32, 32]$. Using the branch and bound method iteratively, the number of interval boxes is decreasing after ten rounds. The statistics are showed in Table 7.
An efficient algorithm for global interval solution of nonlinear algebraic equations

Table 7  B&B result of kinema

| round | boxes | time(s) | width of interval |
|-------|-------|---------|-------------------|
| 1     | 1     | 0       | 64                |
| 2     | 288   | 0.001   | 32                |
| 3     | 5714  | 0.01    | 16                |
| 4     | 56036 | 0.3     | 8                 |
| 5     | 407380| 3       | 4                 |
| 6     | 4369735| 28    | 2                 |
| 7     | 26247657| 243 | 1                 |
| 8     | 87424485| 1284  | 0.5               |
| 9     | 33925040| 300   | 0.25              |
| 10    | 28775053| 285   | 0.125             |
| 11    | 238223456| 14930 | 0.0625            |
| 12    | 187213894| 12250 | 0.03125           |
| 13    | 158874057| 8450  | 0.015625          |
| 14    | 155682210| 7321  | 0.0078125         |
| 15    | 153442353| 7022  | 0.00390625        |
| 16    | 157313456| 7127  | 0.001953125       |

We can merge the interval boxes by backtracking algorithm, and isolate 8 real interval roots width is 0.03125. The statistics are showed in Table 8.

Table 8  backtracking result of kinema

| backtracking boxes | time(s) | width of interval |
|--------------------|---------|-------------------|
| 6587702            | 24      | 0.001953125       |
| 328006             | 1.5     | 0.00390625        |
| 14876              | 0.1     | 0.0078125         |
| 453                | 0.01    | 0.015625          |
| 8                  | 0.001   | 0.03125           |
| 8                  | 0.001   | 0.0625            |

In Table 9 we present the description of whole test set. In Table 10 we list the variable dimension, iteration rounds, number of isolated solutions of backtracking algorithm, and cost time. Some equations cannot be computed in finitely reasonable time and space, and we mark them as blank. See the full statistics for all equations at: http://github.com/eviloan/Nonlinear-Equation-Test-Result.
Table 9 description of test set

| name      | variable | description                                                                 |
|-----------|----------|-----------------------------------------------------------------------------|
| mickey    | 2        | mickey-mouse example to illustrate homotopy continuation                    |
| barry     | 3        | barry from PoSSo                                                            |
| arnborg   | 3        | lazard, auxiliary in cyclic 7 roots, from PoSSo                              |
| noon3     | 3        | neural network, Lotka-Volterra system, n=3                                  |
| rediff3   | 3        | 3-dimensional reaction-diffusion problem                                    |
| conform1  | 3        | conformal analysis of cyclic molecules, instance 1                          |
| morgan    | 3        | from PoSSo                                                                  |
| guakwa2   | 4        | Gaussian quadrature formula 2 knots,2 weights                               |
| katsura3  | 4        | a problem of magnetism in physics n=4                                       |
| liu       | 4        | from PoSSo                                                                  |
| cyclic4   | 4        | cyclic 4-roots problem                                                      |
| capprasse | 4        | the system caprassse of the PoSSo test suite                                |
| lorentz   | 4        | equilibrium of 4-dimensional Lorentz attractor                              |
| moeller5  | 4        | moeller example 5, from PoSSo                                                |
| noon4     | 4        | neural network, Lotka-Volterra system, n=4                                  |
| reimer4   | 4        | the 4-dimensional system of Reimer                                           |
| solotarev| 4        | from PoSSo                                                                  |
| katsura4  | 5        | a problem of magnetism in physics n=5                                       |
| noon5     | 5        | neural network, Lotka-Volterra system, n=5                                  |
| eco5      | 5        | 5-dimensional economics problem                                             |
| redeco5   | 5        | reduced 5-dimensional economics problem                                     |
| sparse5   | 5        | 5-dimensional sparse symmetric polynomial system                             |
| wright    | 5        | system of A.H.Wright                                                        |
| cyclic5   | 5        | cyclic 5-roots problem                                                      |
| reimer5   | 5        | the 5-dimensional system of Reimer                                           |
| boon6     | 6        | neurophysiology, posted by Sjirk Boon                                        |
| eco6      | 6        | 6-dimensional economics problem                                             |
| katsura5  | 6        | a problem of magnetism in physics n=7                                       |
| noon6     | 6        | neural network, Lotka-Volterra system, n=6                                  |
| gaukwa3   | 6        | Gaussian quadrature formula 2 knots,2 weights                               |
| trinks1   | 6        | system of Trinks from the PoSSo test suite                                  |
| cyclic6   | 6        | cyclic 6-roots problem                                                      |
| extcyc6   | 6        | extended cyclic 6-roots problem, to exploit the symmetry                    |
| redeco6   | 6        | reduced 6-dimensional economics problem                                     |
| romin     | 6        | romin robot inverse model, from PoSSo                                       |
| eco7      | 7        | 7-dimensional economics problem                                             |
| cyclic7   | 7        | cyclic 7-roots problem                                                      |
| redeco7   | 7        | reduced 7-dimensional economics problem                                     |
| katsura6  | 7        | a problem of magnetism in physics n=7                                       |
| noon7     | 7        | neural network, Lotka-Volterra system, n=7                                  |
| eco8      | 8        | 8-dimensional economics problem                                             |
| redeco8   | 8        | hand position and orientation of PUMA robot                                 |
| cyclic8   | 8        | cyclic 8-roots problem                                                      |
| redeco8   | 8        | reduced 8-dimensional economics problem                                     |
| gaukwa4   | 8        | Gaussian quadrature formula 2 knots,2 weights                               |
| noon8     | 8        | neural network, Lotka-Volterra system, n=8                                  |
| heart     | 8        | heart-dipole problem                                                        |
| SHEPWM    | 8        | selective harmonic eliminated pulse width modulation                        |
| s91       | 8        | small system from constructive Galois theory, called s91                   |
| eco9      | 9        | 9-dimensional economics problem                                             |
| katsura8  | 9        | a problem of magnetism in physics n=9                                       |
| kinema    | 9        | robot kinematics problem                                                    |
| noon9     | 9        | neural network, Lotka-Volterra system, n=9                                  |
| cyclic9   | 9        | cyclic 9-roots problem                                                      |
### Table 10: Statistics of Test Set

| name      | dimension | initial interval | iterations | real roots | time(s) |
|-----------|-----------|------------------|------------|------------|---------|
| mickey    | 2         | [-2 2]           | 7          | 2          | 0.001   |
| barry     | 3         | [-32 32]         | 14         | 2          | 0.001   |
| arnborg   | 3         | [-16 16]         | 13         | 8          | 0.020   |
| noon3     | 3         | [-8 8]           | 12         | 7          | 0.001   |
| reff]3    | 3         | [-1 1]           | 9          | 2          | 0.001   |
| conform1  | 3         | [-2 2]           | 3          | 0          | 0.001   |
| morgan    | 3         | [-32 32]         | 13         | 2          | 0.001   |
| guialwa2  | 4         | [-4 4]           | 10         | 2          | 0.001   |
| katsura3  | 4         | [-1 1]           | 9          | 2          | 0.001   |
| liu       | 4         | [-1 1]           | 9          | 1          | 0.001   |
| cyclic4   | 4         | [-16 16]         | 20         | 8          | 0.430   |
| caprasse  | 4         | [-4 4]           | 11         | 18         | 0.020   |
| lorentz   | 4         | [-2 2]           | 8          | 3          | 0.001   |
| moeller4  | 4         | [-2 2]           | 11         | 8          | 0.001   |
| moeller5  | 4         | [-2 2]           | 11         | 6          | 0.02    |
| noon4     | 4         | [-8 8]           | 15         | 15         | 0.020   |
| reimer4   | 4         | [-1 1]           | 13         |            | 0.300   |
| solotarev| 4         | [-8 8]           | 14         | 6          | 0.001   |
| katsura4  | 5         | [-1 1]           | 8          | 8          | 0.001   |
| noon5     | 5         | [-8 8]           | 13         | 11         | 9.8     |
| eco5      | 5         | [-8 8]           | 8          | 4          | 0.020   |
| redeco5   | 5         | [-8 8]           | 12         | 4          | 0.001   |
| sparse5   | 5         | [-2 2]           | 3          | 0          | 0.001   |
| wright    | 5         | [-8 8]           | 8          | 32         | 0.030   |
| cyclic5   | 5         | [-16 16]         | 11         | 10         | 0.05    |
| reimer5   | 5         | [-1 1]           |            |            |         |
| boon      | 6         | [-2 2]           | 9          | 8          | 0.01    |
| eco6      | 6         | [-8 8]           | 9          | 4          | 0.004   |
| katsura5  | 6         | [-1 1]           | 8          | 8          | 0.03    |
| noon6     | 6         | [-8 8]           | 13         | 13         | 120.6   |
| guakwa3   | 6         | [-4 4]           | 12         | 0          | 66      |
| trinks1   | 6         | [-8 8]           | 11         | 8          | 0.003   |
| cyclic6   | 6         | [-16 16]         | 15         | 24         | 42      |
| extcyc6   | 6         | [-16 16]         |            |            |         |
| redeco6   | 6         | [-16 16]         | 12         | 4          | 0.02    |
| roinin    | 6         | [-1 1]           | 13         | 4          | 0.003   |
| eco7      | 7         | [-8 8]           | 8          | 8          | 0.02    |
| cyclic7   | 7         | [-16 16]         | 15         | 2          | 3722    |
| redeco7   | 7         | [-16 16]         |            |            |         |
| katsura6  | 7         | [-1 1]           | 8          | 16         | 36.2    |
| noon7     | 7         | [-8 8]           | 12         | 15         | 264     |
| eco8      | 8         | [-8 8]           | 7          | 8          | 0.05    |
| puma      | 8         | [-1 1]           | 13         | 16         | 0.02    |
| cyclic8   | 8         | [-16 16]         |            |            |         |
| redeco8   | 8         | [-16 16]         |            |            |         |
| guakwa4   | 8         | [-16 16]         |            |            |         |
| katsura7  | 8         | [-1 1]           | 9          | 16         | 7150    |
| noon8     | 8         | [-8 8]           | 11         | 17         | 924     |
| heart     | 8         | [-16 16]         |            |            |         |
| SHEPWM    | 8         | [-1 1]           |            |            |         |
| s9t       | 8         | [-8 8]           | 11         | 4          | 0.001   |
| eco9      | 9         | [-8 8]           |            |            |         |
| katsura8  | 9         | [-1 1]           | 16         |            | 13360   |
| kinema    | 9         | [-32 32]         | 16         | 8          | 56334   |
| noon9     | 9         | [-8 8]           | 10         | 19         | 1021    |
| cyclic9   | 9         | [-16 16]         |            |            |         |
5 Conclusions

In this paper, we propose a global interval algorithm, to solve the nonlinear algebraic equations and get the feasible interval solutions. Our algorithm is implemented on GPGPU. Our future work is to improve the computational efficiency of the algorithm. One of the hopeful direction is finding an optimization method to quickly find the maximum and the minimum of the function in a given region to obtain a more tighter interval, improve the speed of branch and bound method.

References

1. Neumaier, Arnold, Interval methods for systems of equations, Vol.37, Cambridge university press(1990).
2. Boyd, Stephen, and Jacob M, Branch and bound methods, Notes for EE364b, 2006-07, Stanford University(2007).
3. Hansen E, Eldon R, Computing zeros of functions using generalized interval arithmetic, Interval Computations, 3, 3–28(1995).
4. Hansen E, Sengupta S, Bounding solutions of systems of equations using interval analysis. BIT Numerical Mathematics, 21(2), 203–211(1981).
5. Hansen E, Interval forms of newtons method. Computing, 20(2), 153–163(1978).
6. Sharma G, Agarwala A, Bhattacharya B, A fast parallel gauss jordan algorithm for matrix inversion using CUDA. Computers & Structures, 128, 31–37(2013).
7. Hansen E, Global optimization using interval analysis the multi-dimensional case, Numerische Mathematik, 34(3), 247–270(1980).
8. Hanson R J, Interval arithmetic as a closed arithmetic system on a computer, Jet Propulsion Laboratory Report, 197(1968).
9. Beck P D, Nehmeier M, Parallel interval newton method on CUDA, Proceedings of Para, 454–464(2012).
10. Moore R E, Kearfott R B, Cloud M J, Introduction to interval analysis, Society for Industrial and Applied Mathematics(2009).
11. Stradi-Granados B A, Interval arithmetic for nonlinear problem solving, International Journal of Engineering Mathematic, (2013).
12. Stahl, Volker, Interval methods for bounding the range of polynomials and solving systems of nonlinear equations, 135–180(1995).
13. Hansen E, On solving systems of equations using interval arithmetic, Mathematics of Computation, 22(102), 374–384(1968).
14. Buchberger B, An Algorithm for Finding the Basis Elements the Residue Class Ring of a Zero Dimensional Polynomial Idea, PhD thesis, Mathematical Institute, University of Innsbruck, Austria, English translation published in the Journal of Symbolic Computation, 41, 475–511(2006).
15. Emeliyanenko P, A complete modular resultant algorithm targeted for realization on graphics hardware, Proceedings of ACM PASCO 35–43(2010).
16. Inoue S, Sato Y, On the parallel computation of comprehensive groebner systems. Proceedings of PASCO 99–101(2007).
17. Li X, Maza M, Multithreaded parallel implementation of arithmetic operations modulo a triangular set, Proceedings of the 2007 international workshop on Parallel symbolic computation, London, Ontario, Canada July 27–28(2007).
18. Stussak C, Schenzel P, Parallel computation of bivariate polynomial resultants on graphics processing units, Proceedings of Para 2010, Iceland(2010).
19. Wang P, Parallel polynomial operations on SMPs: an Overview, Journal of Symbolic Computation, 21, 397–410(1996).
20. Yang L, Hou X, Zeng Z, A complete discrimination systems for polynomials, Science in China Series E 39, 628–646(1996).
21. GAO X S, WANG D K, PEI Z Y, YANG H, Equation solving and machine proof: problem solving based on MMP, Science Press, Beijing(2006).
22. LIN D D, WU Y W, YANG H, Parallel computation for polynomial GCD, CNCC 2003, 900–905(2003).
23. WANG D M, Elimination of the method and its application, Science Press, Beijing(2002).
24. WU W J, Mathematics mechanization, Science Press, Beijing(2003).
25. YANG L, ZHANG J Z, HOU X Y, Nonlinear algebraic equation systems and automated theorem proving, Shanghai Technology Education Press, Shanghai(1996).
26. YANG L, XIA B C, The inequality machine proves and the automatic discovery, Science Press, Beijing(2008).
27. ZENG Z B, ZHANG J Z, A mechanical proof to a geometric inequality of zirakzadeh through rectangular partition of polyhedra, Journal of System Science and Mathematical, 11, 1430–1458(2010).
28. SHAN M J, study on solving real nonlinear algebra systems by hybrid method, Ph.D Thesis, East China Normal University(2008).
29. Kahan W M, A more complete interval arithmetic, Lecture notes for a summer course at the University of Michigan(1968).
30. Hansen E, On linear algebraic equations with interval coefficients, Topics in Interval Analysis, Oxford University Press, London(1969).