Parallel Univariate Real Root Isolation on Multicores

Changbo Chen, Marc Moreno Maza and Yuzhen Xie

ORCCA, University of Western Ontario, Canada

AMMCS 2011, Waterloo
June 29, 2011

Solving for the Real Solutions of Polynomial Systems

Practical importance

- Most applications of polynomial system solving require real solving
  - equilibria (and their stability analysis) of dynamical systems,
  - motion planning, such as the piano mover problem,
  - loop invariants, reachable states in program verification,
  - inverse kinematics problem in robotics, etc.

Solving for the Real Solutions of Polynomial Systems

Practical importance

- Most applications of polynomial system solving require real solving
  - equilibria (and their stability analysis) of dynamical systems,
  - motion planning, such as the piano mover problem,
  - loop invariants, reachable states in program verification,
  - inverse kinematics problem in robotics, etc.

Need of a symbolic approach

- In each of the above problems certifying the number of real solutions or avoiding errors due to approximation may be necessary.
- Moreover, the above problems are often parametric.
- Therefore, symbolic (thus exact) computation is often the way to go.

Solving Symbolically for the Real Solutions

Symbolic representation of real numbers

- A real number that is a solution of a (univar.) polynomial is said *algebraic*. For instance $\sqrt{2}$ and $-\sqrt{2}$ are algebraic, but π is not.
- Let $\alpha \in \mathbb{R}$ be algebraic as solution of $f(x) = 0$, for some $f \in \mathbb{R}[x]$.
- We represent α by a pair $(f, (a, b))$ with $a, b \in \mathbb{Q}$ such that
  - either $a = b = \alpha$
  - or α is the only real root x_0 of f satisfying $a < x_0 < b$.

Solving Symbolically for the Real Solutions

Symbolic representation of real numbers

- A real number that is a solution of a (univar.) polynomial is said *algebraic*. For instance $\sqrt{2}$ and $-\sqrt{2}$ are algebraic, but π is not.
- Let $\alpha \in \mathbb{R}$ be algebraic as solution of $f(x) = 0$, for some $f \in \mathbb{R}[x]$.
- We represent α by a pair $(f, (a, b))$ with $a, b \in \mathbb{Q}$ such that
  - either $a = b = \alpha$
  - or α is the only real root x_0 of f satisfying $a < x_0 < b$.

Usage

- Most algorithms manipulating the real solutions of polynomial systems (cylindrical algebraic decomposition, real root classification, semi-algebraic system decomposition) rely on this representation although others are available (continued fractions, Thom's encoding)
- In fact, these algorithms rely on a core routine: **real root isolation**.

Real Root Isolation

Input: A univariate polynomial $f(x) := a_d x^d + \cdots + a_1 x + a_0$ with rational number coefficients

Output: A list of pairwise disjoint intervals $[\alpha_1, \beta_1], \ldots, [\alpha_e, \beta_e]$ with rational endpoints such that

- each real zero of $f(x)$ belongs to some interval $[\alpha_i, \beta_i]$
- each $[\alpha_i, \beta_i]$ contains one and only one real root of $f(x)$

Real Root Isolation

Input: A univariate polynomial $f(x) := a_d x^d + \cdots + a_1 x + a_0$ with rational number coefficients

Output: A list of pairwise disjoint intervals $[\alpha_1, \beta_1], \ldots, [\alpha_e, \beta_e]$ with rational endpoints such that

- each real zero of $f(x)$ belongs to some interval $[\alpha_i, \beta_i]$
- each $[\alpha_i, \beta_i]$ contains one and only one real root of $f(x)$

Example:

Input: $f(x) := x^6 - 3x^4 + 2x^3 - 1$

Output: $[-\frac{5}{2}, -2],\ [1, \frac{3}{2}]$

Computational Challenges of Real Root Isolation

Complexity issues

- Let $f \in \mathbb{Z}[x]$ be a polynomial with integer coefficients. Let d be its degree and δ be the maximum bit size of a coefficient.
- Isolating the real roots of f requires $O(d^6 + d^4\delta^2)$ bit operations.

Computational Challenges of Real Root Isolation

Complexity issues

- Let $f \in \mathbb{Z}[x]$ be a polynomial with integer coefficients. Let d be its degree and δ be the maximum bit size of a coefficient.
- Isolating the real roots of f requires $O(d^6 + d^4\delta^2)$ bit operations.

Implementation issues

- Solvers (like `RegularChains:-Triangularize`) in MAPLE can compute the complex solutions of fairly large systems.
- Often the output is of the following triangle form

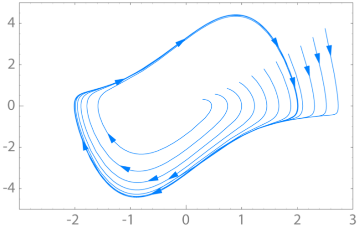
$$f(x_1) = 0, x_2 = R_2(x_1), \ldots, x_m = R_m(x_1),$$

  where f is a polynomial and $R_2, \ldots, R_m$ are rational functions.
- Thus, isolating the real roots of f computes the real solutions.
- Unfortunately, d and δ (as above) grow exponentially with m.
- Challenge: Isolating the real roots of f may require too much resource for the desktop where the complex solutions were computed!

A Driving Application: Limit Cycles of Dynamical Systems

Limit cycles

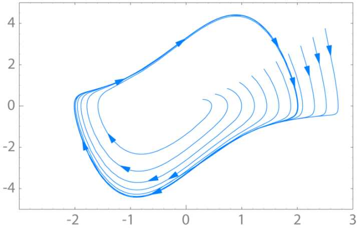
- For $\dot{x} = P(x, y), \quad \dot{y} = Q(x, y)$, this is a closed trajectory s.t. at least one other trajectory spirals into it as $t \to +/-\infty$.
- For P, Q polynomials of degree d, estimating the maximum number of limit cycles is Hilbert's 16th Problem.



A Driving Application: Limit Cycles of Dynamical Systems

Limit cycles

- For $\dot{x} = P(x,y), \quad \dot{y} = Q(x,y)$, this is a closed trajectory s.t. at least one other trajectory spirals into it as $t \to +/-\infty$.
- For P, Q polynomials of degree d, estimating the maximum number of limit cycles is Hilbert's 16th Problem.



Our challenge

- Solving a certain polynomial system in (P. Yu and R. Corless, 2009) estimates the number of limit cycles for $d = 3$ in a special case.
- `RegularChains:-Triangularize` computes the **852 complex roots** of this system within 19 days and 9GB of RAM on a desktop (Chen, Corless, Moreno Maza, Yu and Zhang).
- However, isolating the real roots require far more resources.

Real Root Counting: Vincent-Collins-Akritas Algorithm

Algorithm 1: RealRoots(p)

Input: a univ. squarefree poly. p
Output: the num. of real roots of p
begin
    Let $k \geq 0$ be an int such that the absolute value of all the real roots of p is less than or equal to 2^k;
    if $x \mid p$ **then** $m := 1$ **else** $m := 0$;
    $p_1 := p(2^k x)$;
    $p_2 := p_1(-x)$;
    $m' :=$ RootsInZeroOne(p_1);
    $m := m +$ RootsInZeroOne(p_2);
    return $m + m'$;
end

Algorithm 2: RootsInZeroOne(p)

Input: a univ. squarefree poly. p
Output: the num. of real roots of p in $(0, 1)$
begin
    $p_1 := x^d p(1/x)$;
    $p_2 := p_1(x + 1)$; //Taylor shift
    Let v be the num. of sign variations of the coeff. of p_2;
    if $v \leq 1$ **then** return v;
    $p_1 := 2^d p(x/2)$;
    $p_2 := p_1(x + 1)$; //Taylor shift
    if $x \mid p_2$ **then** $m := 1$ **else** $m := 0$;
    $m' :=$ RootsInZeroOne(p_1);
    $m := m +$ RootsInZeroOne(p_2);
    return $m + m'$;
end

Taylor Shift and Pascal's Triangle

Example: For $f(x) = a_3x^3 + a_2x^2 + a_1x + a_0$, we have, in Horner's rule,
$f(x+1) = a_3x^3 + (a_2 + 3a_3)x^2 + (a_1 + 2a_2 + 3a_3)x + (a_0 + a_1 + a_2 + a_3)$

This is a Pascal's triangle:

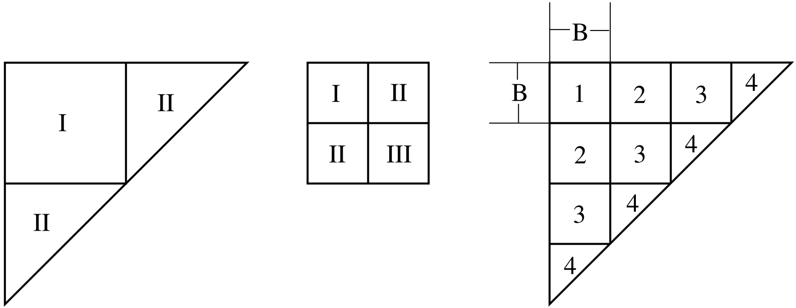
$$
\begin{array}{cccccccccc}
 & & 0 & & 0 & & 0 & & 0 & \\
 & & \downarrow & & \downarrow & & \downarrow & & \downarrow & \\
a_3 & \rightarrow & + & \rightarrow & + & \rightarrow & + & \rightarrow & + & \rightarrow c_3 \\
 & & \downarrow & & \downarrow & & \downarrow & & & \\
a_2 & \rightarrow & + & \rightarrow & + & \rightarrow & + & \rightarrow & c_2 & \\
 & & \downarrow & & \downarrow & & & & & \\
a_1 & \rightarrow & + & \rightarrow & + & \rightarrow & c_1 & & & \\
 & & \downarrow & & & & & & & \\
a_0 & \rightarrow & + & \rightarrow & c_0 & & & & &
\end{array}
$$

Key Observations

- There is certain parallelism in the recursive calls to RootsInZeroOne. However, the work among the recursive calls to RootsInZeroOne may not be balanced.
- The most costly operation is the Taylor shift.
- We should put effort in parallelizing Taylor shift.
- **Related work:**
  Collins and Akritas (1972),
  Collins, Johnson and Küchlin (1992),
  Gathen and Gerhard (1997),
  Decker and Krandick (1999),
  Johnson, Krandick and Ruslanov (2005),
  Boulier, Chen, Lemaire and Moreno Maza (2009), etc.

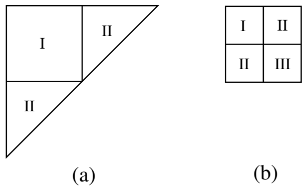
Our Two Strategies for Parallelizing Taylor Shift

"divide-and-conquer" in (a) and (b), and "blocking" in (c)



(a) (b) (c)

Divide-and-conquer: Work, Span and Parallelism



Let $n = d + 1$. For 2-way tableau, we have

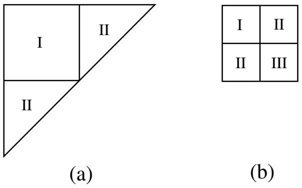
- **work:** $U_1(n) = 4U_1(n/2) + \Theta(1)$, so $U_1(n) = \Theta(n^2)$.
- **span:** $U_\infty(n) = 3U_\infty(n/2) + \Theta(1)$, so $U_\infty(n) = \Theta(n^{log_2 3})$.

For Pascal's triangle, we have

- **work:** $T_1(n) = 2T_1(n/2) + U_1(n/2)$, so $T_1(n) = \Theta(n^2)$.
- **span:** $T_\infty(n) = T_\infty(n/2) + U_\infty(n/2)$, so $T_\infty(n) = \Theta(n^{log_2 3})$.

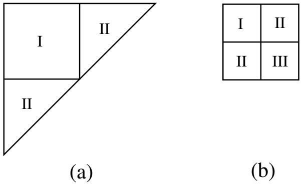
The parallelism for both is $\Theta(n^{0.45})$.

Divide-and-conquer: Space Complexity



Assuming each integer fits within a constant number C of bits, then relying on the fact that, when executed sequentially the whole algorithm can be done in-place within the space allocated to $2n$ integers, by induction we can deduce that the space needed in the divide-and-conquer scheme is $2nC$.

Divide-and-conquer: Cache Complexity



Use the ideal cache model (Frigo et al, 1999).

Z: cache size; L: cache line size.

For 2-way tableau, we have

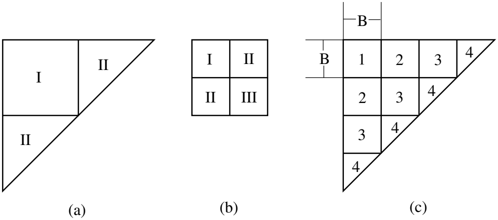
$$Q(n) = \begin{cases} 2n/L + 2 & n \leq \alpha Z \\ 4Q(n/2) + 1 & \text{otherwise} \end{cases} \quad \text{thus} \quad Q(n) = \Theta(n^2/ZL)$$

For Pascal's triangle:

$$Q(n) = \begin{cases} 2n/L + 2 & n \leq \alpha Z \\ 2Q(n/2) + \Theta(n^2/ZL) & \text{otherwise} \end{cases} \quad \text{thus} \quad Q(n) = \Theta(n^2/ZL)$$

Using the Hong-Kung lower bound one can prove that this is optimal.

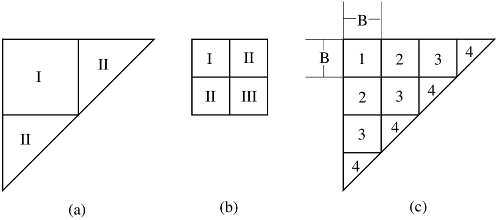
"blocking": Increase the Parallelism



(c) Partition the entire Pascal's triangle into $B \times B$ blocks. A block should fit in cache.

- Work is still $\Theta(n^2)$. Span is $\Theta(B^2) \times n/B = \Theta(Bn)$
- Parallelism is now $\Theta(n/B)$.
- Computation can be done again using $2nC$ space.

"blocking": Cache Complexity



- Assuming $B = \alpha Z$.
- The number of cache misses for each block is $2B/L + 1$.
- The total number of cache misses is

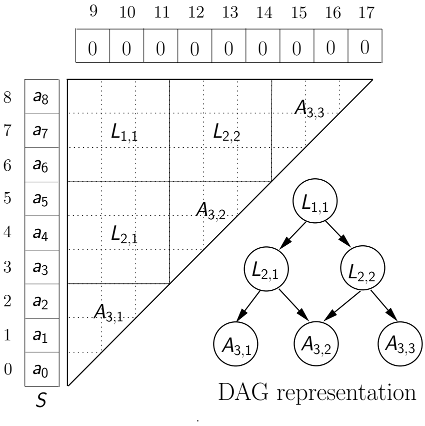
$$Q(n) = \Theta((n/B)^2(2B/L+1)) = \Theta(n^2/(BL)) = \Theta(n^2/(ZL)).$$

- Therefore, provided that $B = \alpha Z$ holds, we retrieve the optimal cache complexity result established for the divide-and-conquer approach.

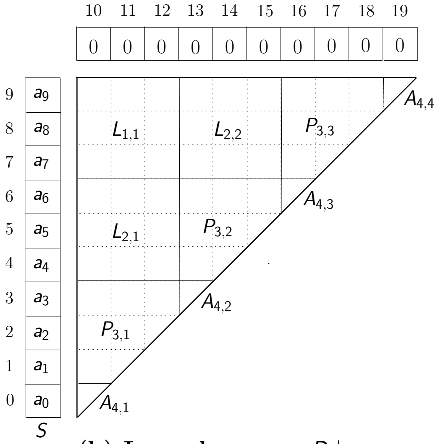
Optimizing the Multicore Implementation

Illustration for the "blocking" scheme:

$n = 9$ and $B = 3$ for Example (a); $n = 10$ and $B = 3$ for Example (b).



(a) Regular case: $B \mid n$

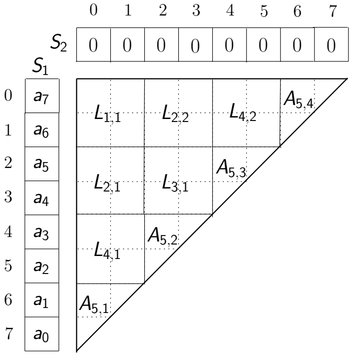


(b) Irregular case: $B \nmid n$

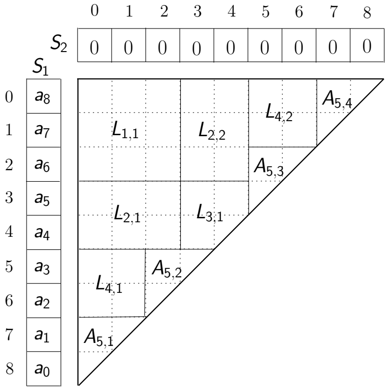
Optimizing the Multicore Implementation

Illustration for the "d-n-c" scheme:

$n = 8$ and $B = 3$ for Example (a); $n = 9$ and $B = 3$ for Example (b).



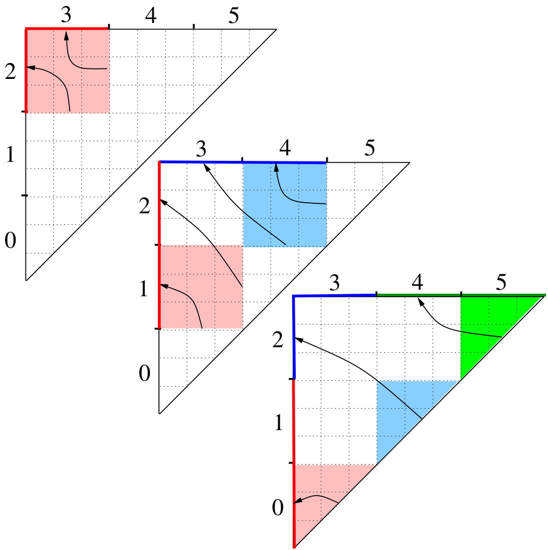
(a) Regular case: n is a power of 2



(b) Irregular case: n is not a power of 2

In-place Operation and Good Data Locality

Illu. for the "blocking" regular case: example of $n = 9$ and $B = 3$.



Experimental Results (I): Parallel Taylor Shift

Table 1. Timings for some benchmark polynomials (in seconds).

n $\times 10^3$	k $\times 10^3$	B	method	Bnd t_1	Bnd t_1/t_8	Cnd t_1	Cnd t_1/t_8	Random t_1	Random t_1/t_8
5	5	50	blocking	6.5	4.9	2.3	2.5	6.5	4.9
5	5	8	d-n-c	6.6	4.6	2.3	2.5	6.63	4.6
10	10	50	blocking	50.8	6.6	17.5	4.0	50.78	6.5
10	10	8	d-n-c	51.7	6.0	17.6	4.2	51.65	6.1
25	25	50	blocking	779	7.5	261	6.1	778.7	7.5
25	25	8	d-n-c	790	7.2	262	6.3	789.7	7.2

- The implementation is in Cilk++.
- The machine has 8 cores, 8 GB memory and 6MB of L2 cache.
- Each processor is Intel Xeon X5460 @3.16 GHz.
- In the table, n and k denote the degree and coefficient size (number of bits) of an input polynomial.

Experimental Results (II): Parallel Real Root Isolation

Table 2. Timings for Chebychev and Mignotte polynomials (in seconds).

n	B		Chebychev polynomial		Mignotte polynomial	
		method	t_1	t_1/t_8	t_1	t_1/t_8
400	50	blocking	413.87	7.0	564.91	3.4
400	8	d-n-c	420.18	7.1	572.65	4.5
500	50	blocking	1269.61	7.3	not enough	
500	8	d-n-c	1279.05	7.4	memory	

Table 3. Timings for random polynomials (in seconds).

n	k	d-n-c			blocking		
		B	t_1	t_1/t_8	B	t_1	t_1/t_8
1000	1000	8	3.26	3.5	50	3.21	3.7
2000	2000	8	18.84	5.4	50	18.58	5.7
3000	3000	8	23.33	5.7	50	22.89	6.0
4000	4000	8	246.34	6.4	50	243.82	6.8
5000	5000	8	1372.70	6.8	50	1340.95	7.3

Experimental Results (III)

Parallel real root isolation of a large polynomial coming from the study of limit cycles of dynamical systems, a simplified version of the Hilbert's 16th problem for the cubic case.

Table 4. Features of the polynomial.

degree	coefficient size	#real roots	processing time
426	1900	78	15 minutes

*The simplified system has 9 limit cycles.

- The 32-core machine consists of 8 Quad Core AMD Opteron 8354 @2.2 GHz connected by 8 sockets; each core has 64 KB L1 data cache and 512 KB L2 cache; every four cores share 2 MB of L3 cache; the total memory is 126 GB.

Concluding Remarks

- We provide a software tool for parallel real root isolation on multicore processors. The kernel routine, Taylor shift, is parallelized with two schemes: "d-n-c" and "blocking".
- For benchmark examples of relatively large size, the speedup is close to linear on 8-cores. For the large polynomial derived from a simplified Hilbert's 16th problem, we can quickly isolate its real roots on a 32-core with 128 GB RAM.

- We have shown an effective approach to empower real root isolation on multicore processors.
- Our software tool enlighten the potential of symbolic methods for solving large real-world applications.
- Work in progress: take into account the growth of the intermediate data and use dynamically sized blocks to balance works; Study how to adopt fast Taylor shift methods into this parallel framework.