Contents lists available at ScienceDirect

Journal of Computational Science

journal homepage: www.elsevier.com/locate/jocs

# Efficient high degree polynomial root finding using GPU

Kahina Ghidouche[a], Abderrahmane Sider[a], Raphaël Couturier[b,*], Christophe Guyeux[b]

[a] Laboratoire LIMED, Faculté des sciences exactes, Université de Bejaia, 06000 Bejaia, Algeria
[b] FEMTO-ST Institute, Univ. Bourgogne Franche-Comté (UBFC), France

ARTICLE INFO

*Article history:*
Received 7 November 2015
Received in revised form 28 October 2016
Accepted 9 December 2016
Available online 19 December 2016

*Keywords:*
Polynomial root finding
Iterative methods
Ehrlich–Aberth
Durand–Kerner
GPU

ABSTRACT

Polynomials are mathematical algebraic structures that play a great role in science and engineering. Finding the roots of high degree polynomials is computationally demanding. In this paper, we present the results of a parallel implementation of the Ehrlich–Aberth algorithm for the root finding problem for high degree polynomials on GPUs using CUDA and on multi-core processors using OpenMP. The main result we achieved is to solve high degree polynomials (up to 1,000,000) efficiently. We also compare the Ehrlich–Aberth method and the Durand–Kerner one on both full and sparse polynomials. Accordingly, our second result is that the first method is much faster and more efficient. Last, but not least, an original proof of the convergence of the asynchronous implementation for the EA method is produced.

## 1. The problem of finding the roots of a polynomial

Polynomials are mathematical algebraic structures used in science and engineering to capture physical phenomena and to express any outcome in the form of a function of some unknown variables. Formally speaking, a polynomial $p(x)$ of degree $n$ having $n$ coefficients in the complex plane $\mathbb{C}$ is:

$$p(x)=\sum_{i=0}^{n} a_i x^i, \qquad a_0 \neq 0. \tag{1}$$

The root finding problem consists in finding all the $n$ values of the variable $x$ for which $p(x)$ is nullified. Such values are called zeros of $p$. If zeros are $\alpha_i$, $i=1,\ldots,n$, the $p(x)$ can be written as:

$$p(x)=a_n\prod_{i=1}^{n}(x-\alpha_i), \qquad a_n \neq 0. \tag{2}$$

The problem of finding a root is equivalent to that of solving a fixed-point problem. To observe this, consider the fixed-point problem of finding the $n$-dimensional vector $X$ such that:

$$X=g(X)$$

where $g:\mathbb{C}^n \longrightarrow \mathbb{C}^n$. We can easily rewrite this fixed-point problem as a root-finding problem by setting $f(X)=X-g(X)$ and likewise we can recast the root-finding problem into a fixed-point problem by setting:

$$g(X)=f(X)-X.$$

It is often impossible to solve such nonlinear equation root-finding problems analytically. When this occurs, we turn to numerical methods to approximate the solution. Generally speaking, algorithms for solving problems can be divided into two main groups: direct methods and iterative methods.

Direct methods only exist for $n\leq 4$, solved in closed form by Cardano [1] in the mid-16th century. However, Abel [2] in the early 19th century proved that polynomials of degree five or more could not, in general, be solved by direct methods. Since then, mathematicians have focused on numerical (iterative) methods such as the famous Newton [3], and the Graeffe one [4].

Later on, with the advent of electronic computers, other methods have been developed such as Jenkins–Traub [5], Larkin [6],

* Corresponding author.
*E-mail addresses:* kahina.ghidouche@univ-bejaia.dz (K. Ghidouche), ar.sider@univ-bejaia.dz (A. Sider), raphael.couturier@univ-fcomte.fr (R. Couturier), christophe.guyeux@univ-fcomte.fr (C. Guyeux).

http://dx.doi.org/10.1016/j.jocs.2016.12.004
1877-7503/© 2016 Elsevier B.V. All rights reserved.

Muller [7], and several others for the simultaneous approximation of all the roots, starting with the Durand–Kerner (DK) method [8,9]:

$$\text{DK:}\quad z_i^{k+1} = z_i^k - \frac{p(z_i^k)}{\prod_{i\neq j}(z_i^k - z_j^k)}, \qquad i = 1, \ldots, n, \tag{3}$$

where $z_i^k$ is the $i$th root of the polynomial $p$ at the iteration $k$.

This formula was mentioned for the first time by Weiestrass [10] as part of the fundamental theorem of Algebra and was rediscovered by Ilieff [11], Docev [12], Durand [8], and Kerner [9]. Another method, discovered by Borsch-Supan [13], and also described and brought in the following form by Ehrlich [14] and Aberth [15], uses a different iteration formula given as:

$$\text{EA:}\quad z_i^{k+1} = z_i^k - \frac{1}{\frac{p'(z_i^k)}{p(z_i^k)} - \sum_{i\neq j}\frac{1}{(z_i^k - z_j^k)}}, \qquad i = 1, \ldots, n, \tag{4}$$

where $p'(z)$ is the polynomial derivative of $p$ evaluated in the point $z$.

Aberth, Ehrlich, and Farmer–Loizou [16] have proven that the Ehrlich–Aberth method (EA) has a cubic order of convergence for simple roots whereas the Durand–Kerner has a quadratic order of convergence. Moreover, the convergence time of iterative methods drastically increases like the degrees of high polynomials, while it is expected that the parallelization of these algorithms will reduce the execution times.

Many authors have dealt with the parallelization of simultaneous methods, *i.e.*, that find all the zeros simultaneously. Freeman [17] implemented and compared DK, EA, and another method of the fourth order proposed by Farmer and Loizou [16], on an 8-processor linear chain, for polynomials of degree 8. The third method often diverges, but the first two methods have a speed-up factor equal to 5.5. Later, Freeman and Brankin [18] considered asynchronous algorithms, in which each processor continues to update its approximations even though the latest values of other roots have not yet been received from the other processors. In contrast, synchronous algorithms wait for the computation of all roots at a given iterations before making a new one. Couturier et al. [19] proposed two methods of parallelization for a shared memory architecture and for a distributed memory one. They were able to compute the roots of sparse polynomials of degree 10,000 in 430 seconds with only 8 personal computers and 2 communications per iteration. Compared to sequential implementations where it takes up to 3300 s to obtain the same results, the authors' work experiment shows an interesting speedup.

To our knowledge, no other work has been published regarding the parallelization of this method or other ones before the emergence of the Compute Unified Device Architecture (CUDA) [20], a parallel computing platform and a programming model invented by NVIDIA. The computing power of GPUs (Graphics Processing Units) has exceeded that of CPUs. However, CUDA adopts a totally new computing architecture to use the hardware resources provided by a GPU in order to offer a stronger computing ability to the massive data computing. First, Ghidouche et al. [21] proposed an implementation of the Durand–Kerner method for sparse polynomials on GPU. Their main result shows that a parallel CUDA implementation is much faster than the sequential implementation on a single CPU.

In this paper, we report on our ongoing research aiming at proposing, implementing, and improving the EA iterative function and the implementation of the Ehrlich–Aberth method to solve high degree polynomials accurately and rapidly on GPUs. The main contributions of this research work are:

- An adaptation of the exponential logarithm to improve the classical Ehrlich–Aberth iterative method, in order to be able to solve sparse and full polynomials of high degree.
- A parallel implementation of Ehrlich–Aberth method on GPU for sparse and full polynomials of high degree up to 1,000,000. This parallel implementation finds roots quite rapidly.
- An original proof of the convergence of the asynchronous implementation for the EA method.

The article is organized as follows. Initially, we recall the Ehrlich–Aberth method in Section 2. Improvements for the Ehrlich–Aberth method are proposed in Section 3. Our convergence proof of the EA asynchronous method is presented in Section 4. Research works related to the implementation of simultaneous methods using a parallel approach are presented in Section 5. In Section 6, we propose a parallel implementation of the Ehrlich–Aberth method on GPU and we discuss it. Section 7 presents and investigates our implementation and experimental study results. Section 8 presents a data analysis collected in the experiments. Finally, Section 9 concludes this article and gives some hints for future research directions in this topic.

## 2. The Ehrlich–Aberth method

It is a cubically convergent iterative method to find zeros of polynomials as proposed by Aberth [15] whose iterative function is:

$$\text{EA2:}\quad z_i^{k+1} = z_i^k - \frac{\frac{p(z_i^k)}{p'(z_i^k)}}{1 - \frac{p(z_i^k)}{p'(z_i^k)}\sum_{j=1, j\neq i}^{j=n}\frac{1}{(z_i^k - z_j^k)}}, \qquad i = 1, \ldots., n \tag{5}$$

It can be noticed that this equation is equivalent to Eq. (4), but we prefer the latter one, because we can use it to improve the Ehrlich–Aberth method and find the roots of high degree polynomials. More details are given in Section 3.

As for any iterative method, a convergence criterion must be checked after each iteration to decide whether to perform another step or to terminate the computations. When the termination happens, it means that the roots are sufficiently stable, *i.e.*, very close to the actual zeros. In the following, we consider that the method converges sufficiently when:

$$\forall i \in [1, n]; \quad \left|\frac{z_i^k - z_i^{k-1}}{z_i^k}\right| < \xi \tag{6}$$

where $|.|$ stands for the absolute value and $\xi$ is the error threshold. The definition of a polynomial $p(z)$ is done by setting each of the $n$ complex coefficients $a_i$. According to the *sparse* or *full* setting, some or all of the coefficients are set deterministically and not randomly so as to have reproducible and comparable results. More details are given in Section 7.

Finally, as for any iterative method, we need to choose $n$ initial guess points $z_i^0$, $i = 1, \ldots, n$. The initial guess is very important since the number of steps needed by the iterative method to reach a given approximation strongly depends on it. In [15] the Ehrlich–Aberth iteration is started by selecting $n$ equi-spaced points on a circle of center 0 and radius $\sigma$, where $\sigma$ is an upper bound to the moduli of the zeros. Later, Bini [22] improved this choice by selecting complex numbers along different circles which