1 Root finding problem

we consider a polynomial of degree n having coefficients in the complex C and zeros α_i, *i=1,...,n.*

$$p(x) = \sum a_i x^i = a_n \prod (x - \alpha_i), a_0 a_n \neq 0,$$

the root finding problem consist to find all n root of *p(x).* the problem of finding a root is equivalent to the problem of finding a fixed-point. To see this consider the fixed-point problem of finding the n-dimensional vector x such that

$$x = g(x).$$

where $g : C^n \longrightarrow C^n$. Note that 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

$$g(x) = f(x) - x$$

Often it will not be possible 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 numerically can be divided into two main groups: direct methods and iterative methods.
Direct methods exist only for $n \leqslant 4$,solved in closed form by G. Cardano in the mid-16th century. However, N.H. Abel in the early 19th century showed that polynomials of degree five or more could not be solved by directs methods. Since then researchers have concentrated on numerical (iterative) methods such as the famous Newton s method, Bernoulli s method of the 18th, and Graeffe s. With the advent of electronic computers, different methods has been developed such as the Jenkins-Traub method, Larkin s method, Muller s method, and several methods for simultaneous approximation of all the roots, starting with the Durand-Kerner method:

$$Z_i = Z_i - \frac{P(Z_i)}{\prod_{i \neq j}(z_i - z_j)}$$

This formula is mentioned for the first time from Weiestrass [12] as part of the fundamental theorem of Algebra and is rediscovered from Ilieff [2], Docev [3], Durand [4], Kerner [5]. Another method discovered from Borsch-Supan [6] and also described and brought in the following form from Ehrlich [7] and Aberth [1].

$$Z_i = Z_i - \frac{1}{\frac{P'(Z_i)}{P(Z_i)} - \sum_{i \neq j}(z_i - z_j)}$$

Aberth, Ehrlich and Farmer-Loizou [10] have proved that the above method has cubic order of convergence for simple roots.

Iterative methods raise several problem when implemented e.g. specific sizes of numbers must be used to deal with this difficulty.Moreover,the convergence time of iterative methods drastically increase like the degrees of high polynomials. The parallelization of these algorithms will improve the convergence time.

Many authors have treated the problem of parallelization of simultaneous methods. Freeman [13] has tested the DK method, EA method and another method of the fourth order proposed from Farmer and Loizou [10],on a 8- processor linear chain, for polynomial of degree up to 8. The third method often diverges, but the first two methods have speed-up 5.5 (speed-up=(Time on one processor)/(Time on p processors)). Later Freeman and Bane [14] consider asynchronous algorithms, in which each processor continues to update its approximations even although the latest values of other $z_i((k))$ have not received from the other processors, in difference with the synchronous version where it would wait. in [15]proposed two methods of parallelization for architecture with shared memory and distributed memory,it able to compute the root of polynomial degree 10000 on 430 s with only 8 pc and 2 communications per iteration. Compare to the sequential it take 3300 s to obtain the same results.

After this few works discuses this problem until the apparition of the Compute Unified Device Architecture (CUDA) [19],a parallel computing platform and a programming model invented by NVIDIA. the computing ability of GPU has exceeded the counterpart of CPU. It is a waste of resource to be just a graphics card for GPU. CUDA adopts a totally new computing architecture to use the hardware resources provided by GPU in order to offer a stronger computing ability to the massive data computing.

Indeed [16]proposed the implementation of the Durand-Kerner method on GPU (Graphics Processing Unit). The main result prove that a parallel implementation is 10 times as fast as the sequential implementation on a single CPU for high degree polynomials that is greater than about 48000.

The mean part of our work is to implement the Aberth method on GPU and compare it with the Durand Kerner implementation.................To be continued..................

References

[1] O. Aberth. Iteration methods for finding all zeros of a polynomial simultaneously. *Mathematics of Computation*, 27(122):339–344, 1973.