Young Scientist- Tomorrow's Science Begins Today
http://eurekajournals.com/Young_Scientist.html
ISSN: 2581-4737

Computing to Find Zeros of the Zeta Function

Agastya Arora[1]

[1]Amity International School, Pushp Vihar.

Abstract

Mathematics and computing are closely related. Algorithms and programs are commonly used to solve historic problems or maybe just our homework.

One such example, of the former, is the Euler Equations. These are equations that model the motion of fluids. Everything from a ripple in the puddle of water to designing better airplanes, the equations are used. Euler equations predict how the fluid will evolve for any instant. Although producing precise values for the state of the fluid at any given moment, there may be one of these values which suddenly skyrockets to infinity. At that point, the Euler equations are said to give rise to a "singularity"-or, more radically, to "blow up."

Computation of the fluid's flow will no longer be possible.

But how did they reach this conclusion?

Well, through a computer simulation.

Mathematicians think of computers as a vital tool in this age. They are able to calculate any "stubborn" problems. One way of solving the RH is through an investigation of more efficient algorithms and computer programs.

We explore efficient computing methods and changes that can be made to them through two zero-finding, or root-finding, methods used in solving Riemann Hypothesis.

Keywords: Non-trivial zeros, conjecture, tangent, cache, parallel computing.

Introduction

The Riemann Hypothesis is a long-standing problem in mathematics. It was declared a Millenium Prize Problem in 2000 by Clay Mathematics Institute as one of the most important for the 21st century. Meaning, whoever is responsible for providing the proof of the same is awarded a million dollars.

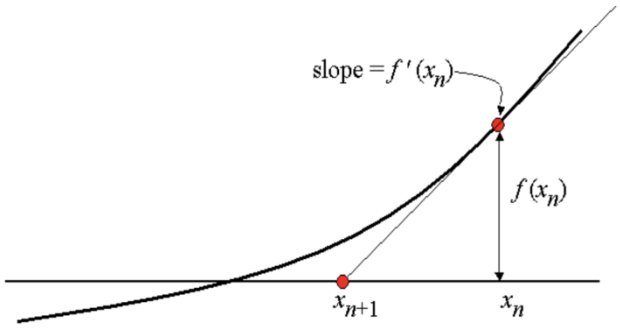
The RH is a deep mathematical conjecture based on the distribution of the primes. Primes are numbers that cannot be expressed as a product of two smaller numbers, eg: 2, 3, 19, 709, 999999937 etc.

The distribution of these prime numbers doesn't actually follow a regular pattern. However, Reimann asserted that their frequency is closely related to an intricate function- the Zeta function (ζ(s)). In his historic paper *Über die Anzahl der Primzahlen unter einer gegebenen Grösse*, german for "On the Number of Prime Numbers Less Than a Given Quantity", Reimann hypothesized that all the zeroes of this Zeta function lie on a certain vertical straight line. The Clay Institute claims that it has been checked for the first 10,000,000,000,000 solutions. Though, it is yet to be proved for every solution.

Newton-Raphson Method

Newton-Raphson method uses the tangent to estimate a more accurate root. We use an iterative method to get near to the actual root.

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$



The Newton-Raphson method is an apt method with regards to the perturbations of the zeta function. By studying into the perturbed zeta function, researchers are able to gain key insights on how the small changes or perturbations affect the original location and behavior of the roots. It gives us information about the sensitivity of these roots to these small changes in the complex plane.

The method also helps us consider approaching zero. Roots of perturbed function converging to zero can give us important insights into the original function.

Now, coming to the computer-y part of it.

Parallel Computing

Parallel computing breakdowns problems into smaller independant parts that can be executed simultaneously. Parallelism computing offers excellent scalability. Huge workloads are able to be computed in lesser times.

A possible parallelized exploration of the parameter space could be insightful. Individual processors can search for respective parameters in different regions.

When dealing with large-scale computations involving the non-trivial zeros, **Heterogeneous computing**, i.e, using a mix of different processing units, can aid the endeavor.

GPUs are apt for parallel processing. GPU clusters- even more so. The independent computing power of the thousands of cores of the GPU simultaneously is beneficial for RH. The aforementioned architecture is very well in line with the inherent parallel nature of most of the calculations for the RH.

Moreover, finding and examining the function at various points simultaneously helps better analyze the distribution of the non-trivial zeros of the critical strip.

Eigenvalue problems

Solving eigenvalue problems linked with the ζ(s) is often carried out while investigating the same. GPUs may expedite this process.

They can increase the speed of the iterative methods that require numerous matrix calculations. GPUs excel at matrix computations in parallel. The high memory bandwidth home to GPUs will also help in the same.

Researchers can write custom GPU kernels-parallelized functions- which tailor to the specific computations needed.

Some **hybrid CPU-GPU computing** could turn out to be a beneficial addition. Many iterative methods benefit from a hybrid method of computing. CPU cores can handle the parts that are not well-suited for parallelization by the GPUs, while GPUs handle the extremely parallel cases.

Machine Learning

Neural networks, a machine learning technique, can very well be deployed to better analyze the patterns and distribution of the non-trivial zeros. Machine learning models can help find anomalies that could possibly give interesting properties about the RH. Autoencoders, a machine learning model, are suited for this anomaly analysis.

Dimensionality reduction techniques like t-Distributed Stochastic Neighbor Embedding (t-SNE or Principal Component Analysis (PCA) can help visualize data in lower dimensions while keeping the significant relationships safe.

Laguerre's Method

Edmond Nicolas Laguerre was a French Mathematician. He is responsible for Guelller's method. Gueller's method is a root finding algorithm which converges to a complex root from any given starting point. This is an attempt at a python code for the same:

```python
import numpy as n
import cmath

def laguerre(poly_coeff, initial_guess, max_iterations=100, tolerance=1e-10):

    # Converting the coefficients to a NumPy array
    poly_coeff = n.array(poly_coeff, dtype=complex)

    for iteration in range(max_iterations):
        # Calculating the polynomial and its derivatives at the current guess
        p = n.polyval(poly_coeff, initial_guess)
        p_prime = n.polyder(poly_coeff)
        p_prime_at_guess = n.polyval(p_prime, initial_guess)

        if abs(p) < tolerance:
            # Roots converged
            return initial_guess

        # Calculating the next guess
        G = p_prime_at_guess / p
        H = G**2 - (p_prime_at_guess / p) - (p_prime_at_guess / p)
        a = len(poly_coeff) - 1

        # Choosing the denominator
        if abs(G + cmath.sqrt((a - 1) * (a * H - G**2))) > abs(G - cmath.sqrt((a - 1) * (a * H - G**2))):
            a_denominator = a / (G + cmath.sqrt((a - 1) * (a * H - G**2)))
        else:
            a_denominator = a / (G - cmath.sqrt((a - 1) * (a * H - G**2)))

        # Updating the guess
        initial_guess -= a_denominator

    # If the method does not converge, we return the current guess
    return initial_guess
```

Cache optimization

Algorithms used for RH often operate on huge datasets, which might not -and usually don't- fit into cache. Optimal cache usage can help speed up matrix calculations which are very popular while solving the RH. Refining initial guesses is a fundamental part of Laguerre's method. We can reduce memory bandwidth requirements by caching intermediate and subsequently using them in iterations.

Loop blocking is useful for dividing a large loop iteration into smaller tiles. These smaller tiles or blocks are more likely to fit into caches, increasing data retrieval speeds.

Data structure alignment is also important.

When talking about **non-trivial zeros** of the Riemann Zeta Function, complex numbers are involved. One thing that can enhance the efficiency of numerical computations is proper alignment of these complex numbers. Cache-friendly access patterns lessen the time wasted finding data from the main memory. A useful way for the aforementioned would be **padding**- basically, the insertion of extra bytes or bits.

Reducing cache misses is necessary. This also increases overall program performance.

Prefetching does this by alleviating memory latency. It fetches the data into the cache preemptively by predicting which data will be needed in the near future.

Hardware Prefetching is one of the two types of prefetching. It does not require any programming intervention. It is especially effective when data patterns are accessible.

The other type **Software Prefetching** is where coders explicitly fetch instructions into the code to encourage the prefetching behavior. It is a sort of fine-grained control over prefetching. Though, it is useful for memory access patterns.

Code Vectorization

Code vectorization is commonly used in high-performance computing to take advantage of SIMD (single instruction, multiple data) abilities possible in modern processors.

Vectors of data used in numerical and iterative methods like Laguerre's method make it open to code vectorization.

With regards to the aforementioned, code vectorization also helps in cache optimization. It very well enhances spatial locality. Consequently, decreasing cache misses.

The most important point with this regard is the **instructionoverhead.** The instruction overhead associated with looping and data element retrieval are reduced. The same instructions are applied to multiple data elements in one individual operation.

A common way to do the same is loop unrolling where multiple iterations are combined into one iteration.

All these techniques have the main aim of increasing performance and lowering resource utilization.

Conclusion

It is readily apparent that computing is a helpful tool for any mathematician. The RH requires extensive computations which may not be possible today. However, we should not rule out the role of computing and AI for the same. The hypothesis remains one of the most challenging problems in mathematics and proving it relies on the combination of theoretical insights, rigorous mathematical analysis and obviously- computers.

References

Quanta magazine, *Computer Proof 'Blows Up' Centuries-Old Fluid Equations.*

David Joyner and Jon-Lark Kim. *Selected unsolved problems in coding theory*. Springer Science & Business Media, 2011.

The Newton-Raphson Method, University of British-Columbia.

The Riemann Hypothesis in Computer Science, Yu. V. Matiyasevich.

Dave Platt, Tim Trudgian : *The Riemann hypothesis is true up to 3x10^12.*

David J. Platt. Numerical Computations Concerning the GRH. arXiv:1305.3087.

Yitang Zhang. *On the zeros of ζ 0 (s) near the critical line*. Duke Math. J. 110 (2001), no. 3, 555–572.

Mekwi, W.: *Iterative Methods for Roots of Polynomials*. (Master's thesis, University of Oxford, 2001).

Press, W.H., Teukolsky, S.A., Vetterling, W.T., Flannery, B.P.: *Numerical Recipes: The Art of Scientific Computing*. (Cambridge University Press, New York, 3rd ed., 2007).

M¨oller, H.: *Report on the Visualization of Laguerre's Method*. 2014.

Kevin Buzzard: *What is the point of computers? A question for pure mathematicians.*