Electronic Journal of Mathematical Analysis and Applications
Vol. 13(1) Jan. 2025, No.14.
ISSN: 2090-729X (online)
ISSN: 3009-6731(print)
http://ejmaa.journals.ekb.eg/

CHRONOLOGICAL VERIFICATION OF THE COLLATZ CONJECTURE USING THEORETICALLY PROVEN SIEVES

S. DUTTA

Abstract. Lothar Collatz proposed a conjecture in number theory in 1937. The widely known Collatz conjecture has not been proven or disproven till date. There are several algorithmic approaches for verification of the conjecture. The sieve of Collatz is a new and popular algorithm to trace back the non linear problem to a linear cross back algorithm, speeding up the verification process. This paper presents a novel algorithmic approach to generate mathematically proven sieve bitsets of $O(2^m)$ elements, where $m \in \mathbb{N}$. The paper further presents a multi-core distributed approach for computational convergence verification of the Collatz conjecture using the pre-computed sieve. Our multi-threaded CPU implementation can verify 1.3×10^9 128-bit integers per second on Intel(R) Core(TM) i7-11850H CPU.

1. Introduction

The Collatz conjecture is one of the most famous unsolved problems in mathematics. It states that given any arbitary positive integer n, the function $f(n)$, defined as $n/2$ if x is even and $3n + 1$ if n is odd, generates a finite sequence that eventually converges to the trivial cycle passing through the value of 1. This finite sequence is also known as the Collatz sequence. There is no theoretical proof of this conjecture till date despite many authors' significant contributions [1–3] in solving this conjecture. There is no counter example to disprove the conjecture either. However, there are experimental evidence [4, 5] and heuristic arguments that support it. The conjecture has been checked by computers and found to follow the conjecture for all numbers $n \leq 2^{68}$ [1].

The function $f(n)$ can be written more elegantly as:

$$f(n) = \begin{cases} 3n+1 & \text{if n is odd} \\ n/2 & \text{if n is even} \end{cases} \tag{1}$$

2020 *Mathematics Subject Classification.* 11Y16.
Key words and phrases. Computational Number Theory, Collatz Conjecture, Sieve, Parallal Computing, Algorithm.
Submitted Nov. 18, 2024.

Note that the result of the odd branch will always be even, ensuring the next iteration going through the even branch. The formulation can be expressed in terms of iterations of the function

$$T(n) = \begin{cases} (3n+1)/2 & \text{if } n \equiv 1 \pmod 2 \\ n/2 & \text{if } n \equiv 0 \pmod 2 \end{cases} \tag{2}$$

The conjecture can be also represented as a weakly connected directed graph whose vertices are positive integers n and the edges are directed from n to $T(n)$ [2]. A portion of the Collatz graph is presented in Fig 1.

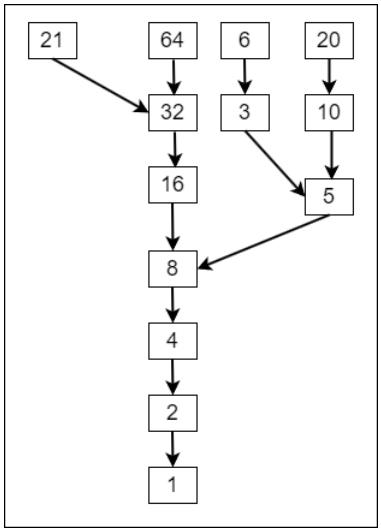

FIGURE 1. A portion of the Collatz graph.

The iterates of T can be defined in a simple manner. $T^0(n) = n$, $T^1(n) = T(n)$, and for $k \in \mathbb{N}$, $T^k(n) = (T(T^{k-1}(n)))$. The T-trajectory of n will be the sequence iterates $(T^0(n), T^1(n), T^2(n), ...)$ [6]. For example, the T trajectory of 5 will be:

$5, 16, 8, 4, 2, 1, ...$

The trajectory of n can show three possible behaviours where $n \in \mathbb{N}$ [2]:

- $T^k(n) = 1$ for some $k \in \mathbb{N}$
- The trajectory becomes periodic and $T^k(n) \neq 1$ for any $k \in \mathbb{N}$.
- $\lim_{k\to\infty} T^k(n) = \infty$

The conjecture states that for every $n \geq 1$, there is an iterate $T^k(n) = 1$ [7]. In other words, all trajectories are convergent for every $n \in \mathbb{N}$. For all $n > 1$, $T^k(n)$ can not occur without $T^k(n) < n$ occurring [2]. So, we can conclude that the trajectory of n converges if $T^k(n) < n$ for any finite positive integer value of k. The stopping time of n, denoted by $\sigma(n)$, is the minimum iteration after which a value smaller than the starting number n is reached. In other words, $\sigma(n)$ is the smallest value of k for which $T^k(n) < n$, if it exists [8]. Otherwise, infinity.

Our approach to initiate the convergence verification process is to apply the Collatz function (3) on a range of numbers and monitor their eventual convergence towards 1. To estimate the verification time for a given sequence of numbers, we drew upon our previous checks of number sequences known to satisfy the conjecture. These prior examinations enabled us to establish a baseline for anticipated verification duration, keeping the increment in verification time with increment in the size of the integers in consideration.

Sequences with number(s) deviating from the conjecture would result in infinitely prolonged verification periods. This process is extended to encompass several sequences of numbers, continuing until a sequence exhibits an unbounded verification time that significantly exceeds the anticipated duration. Upon encountering such a sequence of numbers, we plan to partition that sequence into smaller sub-sequences, allowing more targeted investigations, ultimately identifying the non-conforming number.

An important acceleration technique to test the convergence of numbers is the usage of sieve [9]. Kaiser [10] proposed the sieve algorithm to trace back the non-linear $3n + 1$ problem to a linear cross out algorithm. Using a sieve, only those numbers are checked that can not be shown to converge within a few steps. Using the current sieve strategy, using a sieve of 2^{16} (a sieve has the size of 2^m entries where $m \in \mathbb{N}$) leaves only 1720 out of every 2^{16} (65536) numbers to be checked [4]. The sieve method looks for numbers that can be crossed out, starting with the simplest consideration of crossing out even numbers as they are bound to be united with the odd numbers [10].

The past and ongoing competitive projects verifying the convergence of numbers use huge pre-computed lookup tables to calculate multiple iterates in a single step [9]. Barina [9] used a different approach to avoid the additive step in the Collatz function by switching between n and $n+1$ domains in a smart way when calculating the function iterates. Both of these approaches to speed up the verification process is out of the scope of our study, hence, not used.

The rest of the paper is organised as follows. Section 2 reviews related works, including competitive algorithms and heuristic proofs. Section 3 lead to the construction of a novel algorithm to generate the sieve and an efficient algorithm for convergence verification. The convergence verification algorithm, that can be distributed across multiple cores of a CPU, or multiple CPUs, is discussed in Section 4. The results of the sieve algorithm and the exhaustive convergence verification algorithm is reported in section 5. Finally, section 6 concludes the paper.

2. Related Works

In the past decades, several mathematicians have tried to prove and disprove the conjecture, giving us a greater insight into the conjecture. Lagarias et al. [2] discusses the 3x + 1 problem in great detail and points out the generalizations of the problem, like heuristic arguments and behaviour of the stopping time. Schwob et al. [11] presented novel theorems and generalizations that explore a mapping that follows the Collatz conjecture in two sections. The first section focuses on calculating the number of Collatz iterations for a natural number to reach 1 while the second section analyses the peak values. Rahn et al.'s work [12] proposes a methodology to linearize the Collatz convergence using complete binary tree and the complete ternary tree.

Diverse algorithmic approaches have been presented by a number of authors for the convergence verification of the Collatz conjecture [9, 13–16]. Several past projects, as well as some ongoing ones are trying to verify the convergence for all numbers up to some upper bound, or disprove it. The ongoing project by David Barina [5] claims that the convergence of all numbers below 1450×2^{60} has been verified. Honda et al. [17] claims that their GPU implementation can verify 1.31×10^{12} 64-bit numbers per second and the sequential CPU implementation can verify 5.25×10^9 64-bit numbers per second. Long ago, in 1992, Leavens et al. [6] verified the convergence for all numbers up to 5.61013. In a recent study, Ren et al. [16] proposed new algorithms that can verify very large numbers than known algorithms. Their algorithm is claimed to be able to verify numbers as large as 100000 bits.

3. Sieve of Collatz

To ensure the convergence verification, the algorithm follows a sequential approach. For a positive integer n, the Collatz function (3) is repeatedly applied until a value smaller than n is reached. We consider the trajectory of n to be converged when we get $T^k(n) < n$ for any finite positive integer value of k. To ensure the aforementioned statement to be true, the sequences of numbers subjected to verification consist of numbers within a specific range in an ascending order with no numbers missing in the range. The starting element of the sequence is denoted by a_0 and it is established that all numbers from 1 to $(a_0 - 1)$ adhere to the conjecture. Suppose we are verifying a_0 for convergence and $T^k(a_0) < a_0$. This means, $T^k(a_0) \leq a_0 - 1$. Since it is previously established that all numbers from 1 to $a_0 - 1$ adheres the conjecture, we can conclude that the trajectory of a_0 converges. After verifying that a_0 converges, we can move to verifying a_1 for convergence in the same way, where $a_1 = a_0 + 1$, and so on. Due to the chronological nature of the verification process, it is ensured that a number converges when it reaches a value smaller than itself.

It is observed that even numbers can be crossed out because they unite with the odd numbers after a finite number of steps [10]. An alternative way to formulate this is as follows. Consider an even number $2p$ where p is a positive integer. After the first iteration of the Collatz function, we get $T^1(2p) = p$. We know that $p < 2p$ for all $p \geq 1$. Thus, we can conclude that all even numbers converge after a single iteration of the Collatz function.

With this understanding, we modified our algorithm to skip all even numbers during the verification process. To achieve this, we adapted an iterator that traverses the sequence of numbers to be verified by incrementing it by 2 after verifying each odd number. Starting the sequence of numbers to be verified with an odd number ensures incrementing the iterator by 2 always direct us to the subsequent odd number that necessitates convergence verification.

This sequence of numbers can be denoted as follows:

$$a_p = 2p + 1 \tag{3}$$

where $p \in \mathbb{N}$.

The aforementioned algorithm was implemented and the number of steps required for convergence of each number was recorded. Fig 2 illustrates the relationship between the number of steps and the frequency of numbers converging within each step count (for odd numbers from 1 to 1024). We can observe a discernible pattern emerging in this graphical representation.

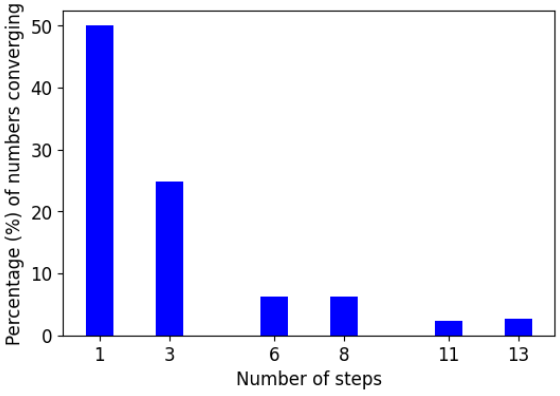

Figure 2. Convergence steps count and their frequency.

25% of the numbers (among all odd and even numbers) converged in just 3 steps. By running the algorithm for a sequence of numbers from 1 to 1000, we found a pattern in the numbers that converged in 3 steps. The numbers were of the form $4p+1$. Furthermore, we established a mathematical proof of convergence using the following converging sequence:

$4p + 1 \to 12p + 4 \to 6p + 2 \to 3p + 1$

Since $3p+1 < 4p+1$ for all $p \geq 1$, we can conclude that all numbers of the form $4p+1$ converges in 4 steps. As a result, it is evident that only one number out of every four numbers requires verification. Numbers of the form $4p + 0$ and $4p + 2$ can be skipped as they are even numbers. Numbers of the form $4p + 1$ converges in 3 steps, hence, can be skipped. This leaves number of the form $4p + 3$ requiring convergence verification.

We re-modified our algorithm to ensure all numbers of the form $4p + 1$ is skipped. We adapted an iterator that traverses the sequence of numbers to be verified by incrementing it by 4 after verifying each number of the form $4p + 3$. Starting the sequence with a number of the same form ensures incrementing the iterator by 4 always direct us to the subsequent number of the same form that necessitates convergence verification.

This sequence of numbers can be denoted as follows: $an = 4p + 3 + n$, where $p, n \in N$.

We also observed 6.25% of the numbers converged in 6 and 8 steps. Further investigation revealed that numbers of the form $16n + 3$ converges in 6 steps. Also, numbers of the form $32n + 11$ and $32n + 23$ converges in 8 steps. Mathematical proof of these convergences are as follows:

$16p + 3 \to 48p + 10 \to 24p + 5 \to 72p + 16 \to 36p + 8 \to 18p + 4 \to 9p + 2$

Since $9p + 2 < 16p + 3$ for all $p \geq 1$, we can conclude that all numbers of the form $16p + 3$ converges in 6 steps. Similarly the convergence of $32p + 11$ and $32p + 23$ can be proved as follows:

- $32p + 11 \to 96p + 34 \to 48p + 17 \to 144p + 52 \to 72p + 26 \to 36p + 13 \to 108p + 40 \to 54p + 20 \to 27p + 10$
- $32p + 23 \to 96p + 70 \to 48p + 35 \to 144p + 106 \to 72p + 53 \to 216p + 160 \to 108p + 80 \to 54p + 40 \to 27p + 20$

Similar convergence patterns can be seen for the following number forms:

- $128p + 7$ converges with $81p + 5$ in 11 steps.
- $128p + 15$ converges with $81p + 10$ in 11 steps.
- $128p + 59$ converges with $81p + 38$ in 11 steps.
- $256p + 39$ converged with $243p + 38$ in 13 steps.
- $256p + 79$ converged with $243p + 76$ in 13 steps.
- $256p + 95$ converged with $243p + 91$ in 13 steps.
- $256p + 123$ converged with $243p + 118$ in 13 steps.
- $256p + 175$ converged with $243p + 167$ in 13 steps.
- $256p + 199$ converged with $243p + 190$ in 13 steps.
- $256p + 219$ converged with $243p + 209$ in 13 steps.

Let's have a look at the number forms discussed above. It was shown that there is a way to unite these numbers with a certain number of steps. These steps are the sieve of collatz. Using this sieve we will test only those numbers for convergence that are not mathematically proven to be converging in a certain number of steps. The number forms found to be converging can be represented as $2^m p+j$. Thus, a sieve bitset can be developed of size 2^m where $m \in \mathbb{N}$ and the i-th position of the bitset will represent the number form $2^m p + i$. The bitset will have values true for number forms that are proven to converge, and false otherwise. A converging number form $2^m p + j$ will populate the fields of the bitset of size $2^{m_{bitset}}$ with true for p=[0, $2^{m_{bitset}}/2^m$), $p \in N$. For example, the number form $4p + 1$ $(2^2 p + 1)$ will populate the following positions with true in a sieve bitset of size 2^5: $(2^2 \times 0) + 1$, $(2^2 \times 1) + 1$, $(2^2 \times 2) + 1$, $(2^2 \times 2) + 1$, $(2^2 \times 3) + 1$, $(2^2 \times 4) + 1$.

While checking a number n, we will first check if (n mod 2^m) is true in the bitset, if yes, we can conclude that n is convergent, thus, can be skipped. We will check n for

convergence otherwise. We observed that with each increasing value of k, new number forms are found to be convergent. So, we developed an algorithm to find all convergent number forms for any given value of k.

First, we start with a small value of m_1 and a bitset B_0 of size 2^{m_1}. To create the bitset of size 2^{m_1+1}, first we generate a new bitset B of the same size with all positions filled with false. We fill the position 0 with true, because numbers of form $2^{m_1}p + 0$ is an even number, thus, convergent. Next, we iterate from 1 to $2^{m_1+1} - 1$ and check for positions that can be marked as true. For a position i, if the position $i \bmod 2^{m_1}$ is marked as true in B_0, it is known that the number form $2^{m_1+1}p + i$ is convergent. Thus, position i is marked as true in B. Otherwise, further checking is done to determine if that number form mathematically converges.

The number is represented in the form $\alpha p + \beta$, where $\alpha = 2^{m_1+1}$ and $\beta = i$. This is to mimic how we represented a number as $2^m p + i$ while checking for convergence of number forms manually. Then we assume $p = 1$ and the Collatz function is applied on it repeatedly until $\beta_k < \beta$ and $\alpha_k \leq \alpha$, where α_k and β_k are the values of α and β after k steps. Thus, for every $\alpha p + \beta$, $\alpha_k p + \beta_k < \alpha p + p$, for $p \geq 0$. To apply to Collatz function on the number represented as $\alpha + \beta$, we make sure that α is even and check if β is even or odd to determine if $\alpha + \beta$ is even or odd. If α is found to be odd after any number of steps, we stop this check and populate the i-th position of the bitset with false. This is because, if *alpha* is odd, a number of the form $\alpha p + \beta$ can't be determined if it is even or odd based on just the value of β, it will depend on the value of p as well. This makes the convergence checking of the number form exceptionally difficult, thus, we avoid it by assuming that the certain number form does not converge.

3.1. **Algorithm I - Sieve Bitset Generation.**

Require: B_0 is a bitset of size 2^{m_1}
Require: B is a bitset of size 2^{m_1+1} filled with false
Require: Integer i, α, β

(1) $i \leftarrow 1$
(2) repeat
(3) $\quad \alpha \leftarrow 2^{m_1+1}$
(4) $\quad \beta \leftarrow i$
(5) $\quad \alpha_k \leftarrow \alpha$
(6) $\quad \beta_k \leftarrow \beta$
(7) $\quad$ repeat
(8) $\quad\quad$ if $B_0[\beta]$ is true:
(9) $\quad\quad\quad B[i] \leftarrow true$
(10) $\quad\quad\quad$ break
(11) $\quad\quad$ if $\alpha_k \leq \alpha$ and $\beta_k < \beta$:
(12) $\quad\quad\quad B[i] \leftarrow true$
(13) $\quad\quad\quad$ break
(14) $\quad\quad$ if β_k is even:
(15) $\quad\quad\quad \alpha_k \leftarrow \alpha_k/2$
(16) $\quad\quad\quad \beta_k \leftarrow \beta_k/2$
(17) $\quad\quad$ if β_k is odd:
(18) $\quad\quad\quad \alpha_k \leftarrow \alpha_k \times 3$
(19) $\quad\quad\quad \beta_k \leftarrow \beta_k \times 3 + 1$
(20) $\quad$ until α_k is odd
(21) $\quad i \leftarrow i + 1$
(22) until $i \leq 2^{m_1+1} - 1$

A sieve bitset of size 2^{16} leaves only 3.23028% of the numbers to be checked for convergence. Since each true or false boolean value takes only 1 bit of space, it takes 8KB

space to store the bitset. A sieve bitset of size 2^{32} leaves 0.9626% of the numbers to be checked for convergence and takes 512MB of space.

The bitsets can be saved as text files and distributed. They can be later opened by the convergence verification program and loaded into the main memory to speed up the verification process. The best way to save the bitsets as text files is to save 1 in place of true values and 0 otherwise. This requires 1 Byte of space for each character, requiring 64KB of space to save the bitset of size 2^{16}, and 4GB of space to save the bitset of size 2^{32}.

4. Convergence Verification

The convergence verification is executed on a sequence of numbers as discussed in section 3. We start checking from a_0 where a_0 is a number of the form $4p + 3$ upto a_n, incrementing the iterator by 4 after each number is checked. For a number a_i, it is first checked if the element at position $(a_i \bmod 2^k)$ in the sieve bitset B is true or false. If it is true, the checking is skipped and we increment the iterator by 4 to move to the next number. If it is false, the number is checked for convergence. To eliminate this extra step of calculating $(a_i \bmod 2^k)$ for every number, we shifted to using an arbitrary precision integer p to keep track of the value of $(a_i \bmod 2^k)$. For this, we first get the initial value of p for a_0 by calculating $a_0 \bmod 2^k$. Then we increment p by 4 after checking each number. For example, we can use a 8 bit integer as p with a bitset of size 2^8, which allows p to reset to 0 when $a_i \equiv 0 \pmod{2^8}$, effectively serving as a faster alternative to computing the modulo operation. This minimizes the computational overhead associated with repeated modulo calculations.

4.1. **Algorithm 2: Convergence Verification.**

Require: n_0, n_{max}, k are positive integers
Require: B is a sieve bitset of size 2^k
Require: p is a positive k-bit integer

(1) $l \leftarrow (n_0 \bmod 4)$
(2) $n_0 \leftarrow n_0 + (3 - l)$
(3) $p \leftarrow (n_0 \bmod 2^k)$
(4) repeat
(5) $\quad n \leftarrow n_0$
(6) $\quad$ if $B[p]$ is false:
(7) $\quad\quad$ repeat
(8) $\quad\quad\quad$ if n is even:
(9) $\quad\quad\quad\quad n \leftarrow n/2$
(10) $\quad\quad\quad$ if n is odd:
(11) $\quad\quad\quad\quad n \leftarrow 3n + 1$
(12) $\quad\quad$ until $n < n_0$
(13) $\quad n_0 \leftarrow n_0 + 4$
(14) $\quad p \leftarrow p + 4$
(15) until $n_0 \geq n_{max}$

We can use multiple cores of a CPU for this verification process by distributing the sequence is multiple cores. Same thing can be done by distributing a large sequence among different CPUs. The chronological nature of the verification process allows us to do it. For a large sequence from a_0 to a_x, where all numbers from 0 to a_0 has been verified, we can distribute the sequence among d cores or d different CPUs to speed up the verification process. For ease of explanation, we will be referring to both different CPUs or different cores in a CPU as cores. Let's call the i-th core c_i, where $0 \leq i < d$ and $i \in \mathbb{N}$. Core c_i will be assigned to check numbers from $a_0 + \frac{x}{d}i$ to $a_0 + \frac{x}{d}(i + 1)$. For example, core c_0 will check numbers from a_0 to $a_0 + \frac{x}{d}$, i.e, a_0 to $a_{\frac{x}{d}}$. Core c_1 will check numbers from $a_{\frac{x}{d}}$

to $a_{\frac{2x}{d}}$, and so on. Here, each core will assume that all numbers from zero to the starting number the core starts checking from, has been verified. And each core will update the last number verified with each new number's verification.

This way, the core c_{i+1} will assume that all numbers from a_0 to $a_{\frac{xi}{d}}$ has been verified while the core c_i will still be busy verifying that range of numbers. This assumption will not result in we missing a non-converging number, as, if any number in the range a_0 to $a_{\frac{xi}{d}}$ actually does not follow the conjecture and the c_{i+1} core's assumption was wrong, the core c_i will find the number not following the number in it's run. Later, an infinite amount of non-converging number can be generated using that one non-convergent number, including the number we marked as verified as a false positive.

In case multiple cores of a CPU is used for the convergence verification, a single sieve bitset loaded in the main memory can be used by all of the cores, eliminating needs for excessive memory usage. Choosing a range of integers to be checked where the count of integers is divisible by the number of cores being used will ensure the sequence being divided equally among all cores.

5. Results and Discussions

We implemented our proposed algorithm for sieve bitset generation in Python programming language. As we needed more control and ease of implementation for this perticular program, and execution speed did not matter, we chose to write it in Python. The program can generate a sieve bitset of size 2^{16} in 46865 microseconds, and a bitset of size 2^{32} in 3083 seconds.

The convergence verification program is written in C++ and can verify units of 10^{13} 128-bit numbers in a single run. The program uses 128-bit arithmetic to represent numbers that are to be verified and for all major calculations. Although the program uses 64-bit arithmetic where 128-bit arithmetic is unnecessary. Table 1 shows the comparison between different sieve sizes, comparing time consumption in sieve generation and the speed of the convergence verification program using that sieve. We can see that a sieve of size 2^{16} gave us the best speed in the convergence verification program. A comparison of our program with other competing programs is displayed in Table 2. Note that the hardware used in these different programs are different. Also, note that all other programs uses lookup tables of size $\mathrm{O}(2^N)$ or $\mathrm{O}(N)$ [9]. Our program does not use any such lookup tables.

Table 1. Comparison between different sieve sizes.

Sieve size	Computational cost (ms)	Convergence verification speed (numbers/second)
2^{16}	46.865	1.4335×10^9
2^{32}	3083000	1.3105×10^9

From the comparison above, we can conclude that our algorithm showed decent performance despite having no lookup tables to speed up calculations during the verification process. Using a mathematically proven sieve ensures we don't mark any number as verified as a false positive. Using multiple cores and threads is one of the major factors in the speed of the algorithm. It also allows parallel verification in different machines. The algorithm can be further improved by introducing lookup tables, which can be a focus of future work.

6. Conclusion

This paper presents a novel method of generating a sieve bitset which can later be used in convergence verification of the Collatz conjecture. Existing approach to generating

TABLE 2. Comparison between different competing programs. Speed is given in numbers per second.

Authors	Sieve size	Size of numbers	Speed	Hardware
Honda et al. [17]	2^{37}	64-bit	1.31×10^{12}	NVIDIA GeForce GTX TITAN X
Honda et al. [17]	2^{37}	64-bit	5.25×10^{9}	Intel Core i7-4790
Barina et al. [9]	2^{34}	128-bit	4.21×10^{9}	Intel Xeon Gold 5218
Barina et al. [9]	2^{24}	128-bit	2.20×10^{11}	NVIDIA GeForce RTX 2080
This paper	2^{16}	128-bit	1.43×10^{9}	Intel Core i7-11850H

sieve depends on an empirical proof whereas our method is based on mathematical proof using number forms. In addition, the paper presents a new method of using multiple cores and threads, as well as multiple different CPUs for the convergence verification. Our program can also process 128-bit numbers while most of other competitive programs can only process 64-bit numbers. Usage of lookup tables can significantly speed up our current algorithm and can be considered as a topic for future work.

ACKNOWLEDGEMENT

The computational resources were provided by IBM India Systems Development Lab.

REFERENCES

[1] H. N. Crooks Jr and C. Nwoke, "Collatz conjecture: Patterns within," *arXiv preprint arXiv:2209.05995*, 2022.
[2] J. C. Lagarias, "The 3 x+ 1 problem and its generalizations," *The American Mathematical Monthly*, vol. 92, no. 1, pp. 3–23, 1985.
[3] ——, *The Ultimate Challenge: The* $3x+1$ *Problem*. American Mathematical Society, 2023.
[4] Eric Roosendaal. Technical Details (On the 3x + 1 problem). Accessed on April 6th, 2023. [Online]. Available: http://www.ericr.nl/wondrous/techpage.html
[5] David Barina. Convergence verification of the Collatz problem. Accessed on October 5th, 2023. [Online]. Available: https://pcbarina.fit.vutbr.cz/
[6] G. T. Leavens and M. Vermeulen, "3x+ 1 search programs," *Computers & Mathematics with Applications*, vol. 24, no. 11, pp. 79–99, 1992.
[7] J. C. Lagarias, "The 3x+ 1 problem: An annotated bibliography, ii (2000-2009)," *arXiv preprint math/0608208*, 2006.
[8] T. O. E. Silva *et al.*, "Maximum excursion and stopping time record-holders for the 3x+ 1 problem: computational results," *Mathematics of Computation*, vol. 68, no. 225, pp. 371–384, 1999.
[9] D. Barina, "Convergence verification of the collatz problem," *The Journal of Supercomputing*, vol. 77, no. 3, pp. 2681–2688, 2021.
[10] J. Kaiser, "Sieve of collatz," p. 11, 2016. [Online]. Available: https://vixra.org/pdf/1611.0224v1.pdf(AccessedonApril6th,2023)
[11] M. R. Schwob, P. Shiue, and R. Venkat, "Novel theorems and algorithms relating to the collatz conjecture," *International Journal of Mathematics and Mathematical Sciences*, vol. 2021, pp. 1–10, 2021.
[12] A. Rahn, E. Sultanow, M. Henkel, S. Ghosh, and I. J. Aberkane, "An algorithm for linearizing the collatz convergence," *Mathematics*, vol. 9, no. 16, p. 1898, 2021.

[13] M. Venkatesulu and C. D. Parameswari, "Verification of collatz conjecture: An algorithmic approach," *WSEAS Transactions on Engineering World*, vol. 2, pp. 71–75, 2020.
[14] E. Yolcu, S. Aaronson, and M. J. Heule, "An automated approach to the collatz conjecture," *Journal of Automated Reasoning*, vol. 67, no. 2, p. 15, 2023.
[15] V. Mandadi and D. Paramwswari, "Verification of collatz conjecture: An algorithmic approach based on binary representation of integers," *arXiv preprint arXiv:1912.05942*, 2019.
[16] W. Ren, S. Li, R. Xiao, and W. Bi, "Collatz conjecture for 2^ 100000-1 is true-algorithms for verifying extremely large numbers," in *2018 IEEE SmartWorld, Ubiquitous Intelligence & Computing, Advanced & Trusted Computing, Scalable Computing & Communications, Cloud & Big Data Computing, Internet of People and Smart City Innovation (SmartWorld/SCALCOM/UIC/ATC/CBDCom/IOP/SCI)*. IEEE, 2018, pp. 411–416.
[17] T. Honda, Y. Ito, and K. Nakano, "Gpu-accelerated exhaustive verification of the collatz conjecture," *International Journal of Networking and Computing*, vol. 7, no. 1, pp. 69–85, 2017.

Samrat Dutta
India Systems Development Lab, IBM, Bangalore, India
Email address: samrat.dutta2@ibm.com