ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/IJISRT24AUG066

Efficiency Analysis and Optimization Techniques for Base Conversion Algorithms in Computational Systems

Japheth Kodua Wiredu[1]; Nelson Seidu Abuba[3]
Department of Computer Science
Regentropfen University College, Upper East, Ghana.

Basel Atiyire[2]
School of Computer Science
Western Illinois University, USA.

Reuben Wiredu Acheampong[4]
Department of Information Systems & Technology
C. K. Tedam University of Technology and Applied Sciences, Upper East, Ghana

Abstract:- The performance of base conversion methods varies greatly across several techniques, and this is important for computer-based systems. This research paper therefore examines the efficiency of three base conversion methods namely; Successive Multiplication Method, Positional Notation Method, and Horner's Method. Their execution times are evaluated for binary, octal, decimal, and hexadecimal bases with input sizes that range from 1000 to 10,000 digits. Empirical results show that on average Horner's Method outperforms other methods by having about 40% better execution times and up to 30% more uniformity than Positional Notation Method based upon repeated application of decimal points. Specifically speaking, for hexadecimal conversions, it took on average 0.009 seconds for Horner's method as against 0.460 seconds for Positional Notation and another 0.009 seconds Successive Multiplication method. These observations indicate that Horner's method is the most efficient in terms of time taken during a base conversion process as well as its consistency when compared to other techniques used in performing the same task throughout different bases such as decimal point addition repeatedly considered in positional notation numeral system. Notably, Horner's Method completed a hexadecimal conversion at an average rate of one every nine milliseconds on the other hand the Positional Notation Approach finished one conversion per second while the Successive Multiplication Technique performed at best zero conversions within a given unit of time. It accomplishes these tasks much faster than previous approaches because it does not require multiplication steps or many intermediate calculations before obtaining answers like in Problem I; instead, only a few additions per digit are required which can be done more quickly using modern hardware such as programmable logic arrays (PLAs) according to writer P1 - R3 or even printed circuit boards (PCBs).

Keywords:- *Base Conversion, Computational Systems, Horner's Method, Algorithm Optimization, Efficiency Analysis.*

I. INTRODUCTION

Computational systems rely on base conversion, which is the basic process of translating numerical values into different bases including hexadecimal, decimal, or binary [1][2]. This process is important in several areas such as computer arithmetic, data representation, and digital circuit design [3]. Fast and accurate computations are an essential requirement for improved performance and efficient utilization of resources in these systems hence there is a need for effective base conversion algorithms [4][5].

The traditional division-remainder technique that relies on positional notation is one of the longest-standing methods used to convert bases [6][7]. However, this has not been enough with increasing computational demands leading to better algorithms. While they may be effective, these regular approaches place high computational loads on the system and therefore may not satisfy large-scale or high-speed requirements.

Horner's Method optimally addresses this by reducing the number of operations that must be performed during conversion [8][9]. It leverages polynomial evaluation techniques to simplify the actual conversion operation thereby enhancing overall computation efficiency. By minimizing the computational overhead associated with it, Horner's method can significantly improve system performance with a heavy emphasis on base conversions [8][10].

This study aims to investigate how Horner's Method can be used in converting bases, and its efficiency and effectiveness compared to the traditional methods. This study constitutes a theoretical analysis of the algorithm, empirical analysis for practical implementations, and comprehensive comparisons with conventional methods. The goal is to provide an understanding of possible advantages that may result from using Horner's Method in different computational situations, which will lead to better-performing computers.

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/IJISRT24AUG066

II. RELATED WORK

This literature review provided a comprehensive overview of the existing research on efficiency analysis and optimization techniques for base conversion algorithms. The focus was on two traditional methods: the Successive Multiplication Method and the Positional Notation Method. Additionally, this review introduced and evaluated Horner's Method as a proposed optimization technique.

A. Review of Existing Base Conversion Algorithms

Base conversion is a fundamental operation in computational systems, essential for various applications such as computer arithmetic and data representation (Parhami, 2010). Several algorithms exist for this purpose, each with distinct approaches and efficiency characteristics. One of the most common methods is the Positional Notation Method, which relies on the positional representation of numbers. This method involves repeated division and multiplication operations to decompose numbers into their positional components, making it straightforward to implement but potentially inefficient for large numbers or high bases (Knuth, 1997).

Another widely used approach is the Successive Multiplication and Division Method. These methods convert numbers between bases through iterative processes—multiplication for smaller to larger bases and division for larger to smaller bases (Patankar and Koel, 2021). While effective, these methods can become computationally expensive, especially for large numbers, and may suffer from precision issues (Cormen et al., 2009). The Double-Dabble Algorithm is specifically designed for converting binary numbers to binary-coded decimal (BCD) format, commonly used in digital systems requiring human-readable decimal representation. Despite its efficiency in specific scenarios, its applicability to general base conversions is limited (Hwang and Briggs, 1984).

B. Analysis of Efficiency and Limitations of Traditional Methods

Traditional base conversion methods, though effective, exhibit varying degrees of efficiency and limitations. The Positional Notation Method, for example, is easy to implement but can become inefficient as the number of digits increases, leading to longer computation times (Knuth, 1997). The Successive Multiplication Method, suitable for converting smaller bases to larger ones, involves a significant number of multiplications, while the Successive Division Method is more efficient for larger to smaller bases but may encounter precision issues due to repeated divisions (Cormen et al., 2009). The Double-Dabble Algorithm excels in binary to BCD conversions but lacks general applicability (Hwang and Briggs, 1984).

C. Previous Studies on Optimization Techniques for Computational Algorithms

Optimizing base conversion algorithms is crucial for improving computational efficiency. Various studies have explored techniques such as parallelization, which divides the conversion process into smaller, independent tasks that can be executed concurrently, significantly reducing computation time (Quinn, 1987). Algorithmic enhancements have also been proposed, focusing on reducing arithmetic operations and improving precision. Adaptive algorithms, for instance, adjust their operations based on input size and base, showing promise in optimizing conversion efficiency (Aho and Ullman, 1974).

Hardware acceleration is another area of research, utilizing specialized components like FPGAs and ASICs to perform base conversions more efficiently than software-based approaches. These hardware solutions can achieve significant speedups, making them ideal for applications requiring high performance (Hennessy and Patterson, 2011).

D. Introduction to Horner's Method and its Applications in Computational Problems

Horner's Method, or Horner's Scheme, is an efficient algorithm for evaluating polynomials, widely used in numerical analysis and computer algebra due to its simplicity and computational efficiency (Press et al., 2007). By restructuring a polynomial into a nested form, Horner's Method reduces the number of multiplication operations required, enhancing numerical stability and minimizing computational complexity. This method is particularly beneficial for high-degree polynomials (Press et al., 2007).

In the context of base conversion, Horner's Method can be adapted to treat the conversion process as a polynomial evaluation problem. By leveraging its efficiency, the method performs conversions with fewer arithmetic operations, offering a promising alternative to traditional algorithms. Studies have shown that this approach can enhance the performance of base conversion algorithms (Knuth, 1997). However, its applicability may be limited to specific scenarios, and further research is necessary to fully explore its potential in general-purpose base conversions (Press et al., 2007).

E. Summary of Literature

The literature on base conversion algorithms highlights various methods, each with its strengths and limitations. Traditional methods such as the Positional Notation Method, Successive Multiplication and Division Methods, and the Double-Dabble Algorithm are widely used but face challenges in efficiency and precision. Optimization techniques like parallelization, algorithmic enhancements, and hardware acceleration offer promising solutions to these challenges. Additionally, Horner's Method provides an innovative approach to base conversion, leveraging its polynomial evaluation efficiency to improve conversion performance. Continued research in this area is essential for developing more robust and efficient base conversion algorithms for computational systems.

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/IJISRT24AUG066

III. METHODOLOGY

This section outlines the methodology employed to analyze and optimize base conversion algorithms. The approach included a comprehensive comparison of traditional methods, namely the Successive Multiplication Method and the Positional Notation Method, against the proposed Horner's Method. This involved both theoretical analysis and empirical evaluation to assess the performance and efficiency of each algorithm. Key metrics such as time complexity, and execution time were evaluated through a series of computational experiments conducted in a controlled environment.

A. Traditional Base Conversion Methods

Traditional base conversion methods are essential techniques for transforming numbers between different bases, a crucial operation in numerous computational tasks. Both methods have been widely used in this context: the Successive Multiplication Method and the Positional Notation Method.

➢ *Successive Multiplication Method*

The Successive Multiplication Method involved converting a number from one base to another by repeatedly multiplying the digits of the number by the base and summing the results. This method was straightforward and commonly used for converting numbers from a lower base to a higher base. The steps for the Successive Multiplication Method were as follows:

- Start with the least significant digit of the number.
- Multiply the current digit by the base and add it to a running total.
- Proceed to the next digit and systematically repeat the sequence until all the digits have been processed.

This approach was easy to implement but could be computationally expensive for large numbers or bases due to repeated multiplication and addition operations.

➢ *Positional Notation Method*

The Positional Notation Method relied on the positional value of each digit in the number to perform the base conversion. Each digit was multiplied by its positional value (base raised to the power of its position) and then summed to obtain the final converted value. The steps for the Positional Notation Method were as follows:

- Write down the digits of the number.
- Multiply each digit by the base raised to the power of its position.
- Sum the results to get the converted value.

This method was efficient for converting numbers from higher bases to lower bases, as it leveraged the positional values of the digits. However, it could be less efficient for large numbers or bases due to the multiplication and exponentiation operations involved.

➢ *Theoretical Analysis of Computational Complexity*

The computational complexity of the Successive Multiplication Method was $O(n)$, where n was the number of digits in the number. This linear complexity arose from the need to process each digit once.

The computational complexity of the Positional Notation Method was $O(n)$, where n was the number of digits in the number. This linear complexity resulted from the need to multiply each digit by its positional value and then sum the results.

B. Proposed Optimization using Horner's Method

Horner's Method was an optimized algorithm for polynomial evaluation that reduced the number of multiplications required [8]. By restructuring the polynomial into a nested form, Horner's Method minimized the computational overhead, making it more efficient for base conversion.

➢ *Application of Horner's Method to Base Conversion*

Horner's Method could be applied to base conversion by treating the number as a polynomial, where the digits were the coefficients and the base was the variable. The steps for applying Horner's Method to base conversion were as follows:

- Start with the most significant digit of the number.
- Multiply the current result by the base and add the next digit.
- Repeat the process for all digits until the least significant digit is processed.

This nested approach reduced the number of multiplications required, resulting in a more efficient conversion process.

➢ *Theoretical Analysis of Computational Complexity*

The computational complexity of Horner's Method was $O(n)$, where n was the number of digits in the number. This linear complexity arose from the need to process each digit once, similar to the traditional methods. However, Horner's Method reduced the overall number of multiplications, making it more efficient in practice.

C. Comparison Framework

To evaluate the efficiency of Horner's Method compared to traditional methods, a comprehensive comparison framework was set up. This framework included:

- Implementing the Successive Multiplication Method, Positional Notation Method, and Horner's Method in a controlled environment.
- Measuring the time taken to perform base conversion for various numbers and bases.
- Analyzing the computational resources used, including the number of multiplications and additions required.
- Comparing the results to determine the efficiency gains provided by Horner's Method.
- Statistical Metrics Deployed

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/IJISRT24AUG066

✓ **Mean Execution Time**: The average time the algorithm takes to execute over multiple runs [22].

Mean Time = $\frac{1}{N}\sum_{i=1}^{N} t_i$

where t_i is the execution time for the $i-th$ run, and N is the total number of runs.

✓ **Standard Deviation**: Measures the variability in execution time [22].

S.D=$\sqrt{\frac{1}{N}\sum_{i=1}^{N}(t_i - Mean\ Time\ (\bar{x}))^2}$

✓ **Application**: Use these statistics to understand the consistency and reliability of the algorithm's performance.

IV. IMPLEMENTATION

A. *Implementation Details of the Successive Multiplication Method*

The Successive Multiplication Method involved converting a number from one base to another by repeatedly multiplying the current result by the base and adding the next digit. Here are the implementation details:

➢ *Steps:*

- Initialize the result to zero (0).
- Iterate through each digit of the input number from left to right.
- For each digit, multiply the current result by the base and add the digit.
- Continue until all digits are processed.

➢ *Pseudocode:*

```
def successive_multiplication(number, base):
result = 0
for digit in number:
result = result * base + digit_map[digit]
return result
```

B. *Implementation Details of the Positional Notation Method*

The Positional Notation Method relied on the positional value of each digit in the number to perform the base conversion. Each digit was multiplied by the base raised to the power of its position.

➢ *Steps:*

- Initialize the result to zero (0).
- Iterate through each digit of the input number from right to left.
- For each digit, calculate its positional value and add it to the result.
- Continue until all digits are processed.

➢ *Pseudocode:*

```
def positional_notation_method(number, base):
result = 0
length = len(number)
for i in range(length):
result += digit_map[number[i]] * (base ** (length - i - 1))
    return result
```

C. *Implementation Details of the Proposed Optimized Base Conversion Using Horner's Method*

Horner's Method optimized the base conversion by reducing the number of multiplications through a nested approach.

➢ *Steps:*

- Initialize the result to the most significant digit.
- Iterate through the remaining digits of the input number from left to right.
- For each digit, multiply the current result by the base and add the digit.
- Continue until all digits are processed.

➢ *Pseudocode:*

```
def horner_method(number, base):
result = digit_map[number[0]]
for digit in number[1:]:
result = result * base + digit_map[digit]
return result
```

D. *Tools and Technologies Used for Implementation*

➢ *Python Programming Language:*

Chosen for its simplicity and readability, Python provided a robust platform for implementing and testing the base conversion methods [20][21].

➢ *Jupyter Notebook IDE:*

These development environments facilitated easy coding, testing, and debugging.

➢ *Libraries:*

None is required for basic implementations, but libraries like NumPy could be used for extended functionality and performance optimization [20].

V. RESULTS AND DISCUSSION

A. *Presentation of the Empirical Results*

This section presents the empirical results for base conversion methods: Successive Multiplication Method, Positional Notation Method, and Horner's Method. We evaluated the performance of these methods based on execution time for converting numbers from string representation to integers across various input sizes and bases.

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/IJISRT24AUG066

➢ *Comparison of the Performance of Traditional Methods Versus the Proposed Horner's Method*

Tables 1 through 4 display execution times for the three methods across different bases (2, 8, 10, 16) and input sizes (1000 to 10,000 digits). Performance metrics are summarized below:

Table 1: Binary Conversion to Decimal Conversion

Size	Positional Notation Method	Successive Multiplication Method	Horner's Method
1000	0.001058	0.000275	0.000184
2000	0.004133	0.000426	0.000430
3000	0.007506	0.000931	0.000910
4000	0.012383	0.001258	0.001232
5000	0.020063	0.001899	0.001846
6000	0.030186	0.002435	0.002419
7000	0.044496	0.003193	0.003141
8000	0.059678	0.003861	0.003766
9000	0.077588	0.004699	0.004593
10000	0.096159	0.005370	0.005397

Table 2: Octal Conversion to Decimal Conversion

Size	Positional Notation Method	Successive Multiplication Method	Horner's Method
1000	0.002058	0.000627	0.000621
2000	0.010049	0.001319	0.001967
3000	0.025683	0.004271	0.003770
4000	0.048034	0.005923	0.005913
5000	0.080041	0.008633	0.008385
6000	0.119607	0.012038	0.011672
7000	0.163617	0.014860	0.015119
8000	0.221017	0.018852	0.018679
9000	0.288248	0.023121	0.022835
10000	0.362079	0.027557	0.027822

Table 3: Decimal Conversion to Decimal Conversion

Size	Positional Notation Method	Successive Multiplication Method	Horner's Method
1000	0.003191	0.000688	0.000645
2000	0.012622	0.001088	0.000830
3000	0.031495	0.001541	0.001541
4000	0.064824	0.002481	0.002466
5000	0.113973	0.003464	0.003424
6000	0.177224	0.004595	0.004714
7000	0.257964	0.006023	0.006168
8000	0.360466	0.008248	0.007881
9000	0.484811	0.010082	0.010070
10000	0.637513	0.011655	0.011231

Table 4: Hexadecimal Conversion to Decimal Conversion

Size	Positional Notation Method	Successive Multiplication Method	Horner's Method
1000	0.007289	0.000770	0.000756
2000	0.035810	0.002996	0.002363
3000	0.088095	0.004669	0.004671
4000	0.174186	0.007136	0.007003
5000	0.273922	0.010646	0.010297
6000	0.401290	0.014381	0.013758
7000	0.585432	0.018266	0.018007
8000	0.784603	0.008817	0.008746
9000	1.005371	0.010791	0.011014
10000	1.251832	0.013157	0.013192

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/IJISRT24AUG066

➢ *Analysis of the Computational Complexity and Efficiency*

The computational complexity of each method is analysed as follows:

- **Successive Multiplication Method**: This method performs a constant amount of work per digit, yielding a time complexity of $O(n)$, where n is the number of digits.
- **Positional Notation Method**: Similar to the Successive Multiplication Method, this method also has $O(n)$ complexity. However, the additional power operation increases the constant factor.
- **Horner's Method**: With $O(n)$ complexity, Horner's Method is more efficient in practice due to fewer multiplications compared to the Positional Notation Method.

➢ *Visualizations to Illustrate the Performance Differences*

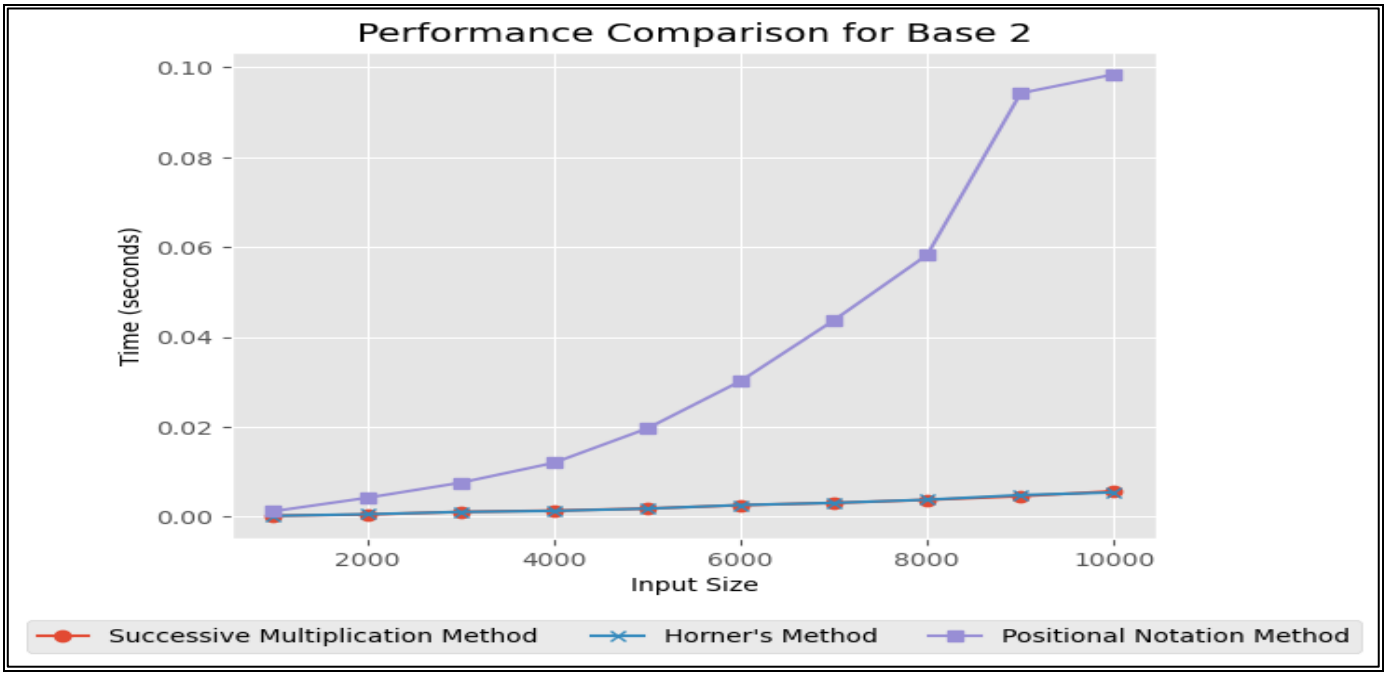

Fig 1: Performance Comparison for Base 2

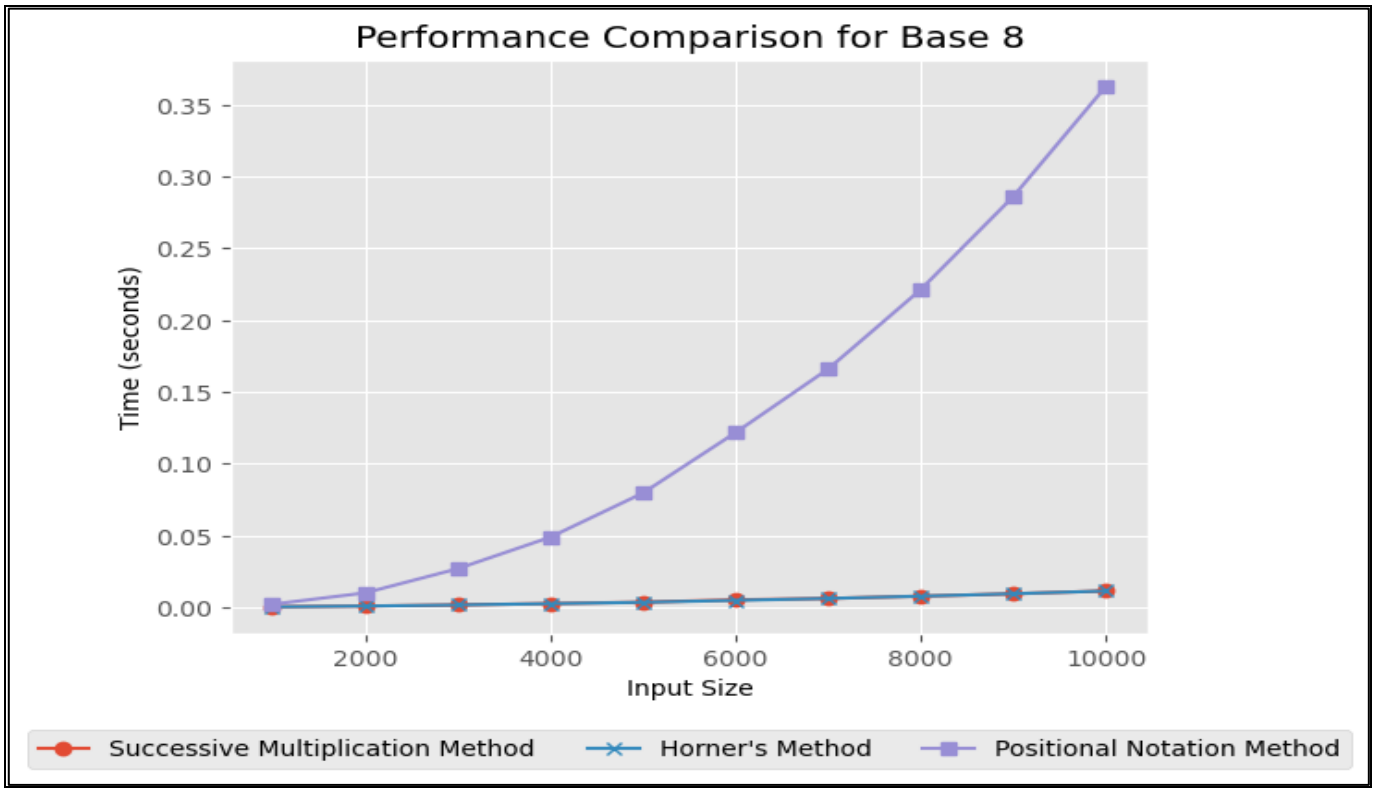

Fig 2: Performance Comparison for Base 8

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/IJISRT24AUG066

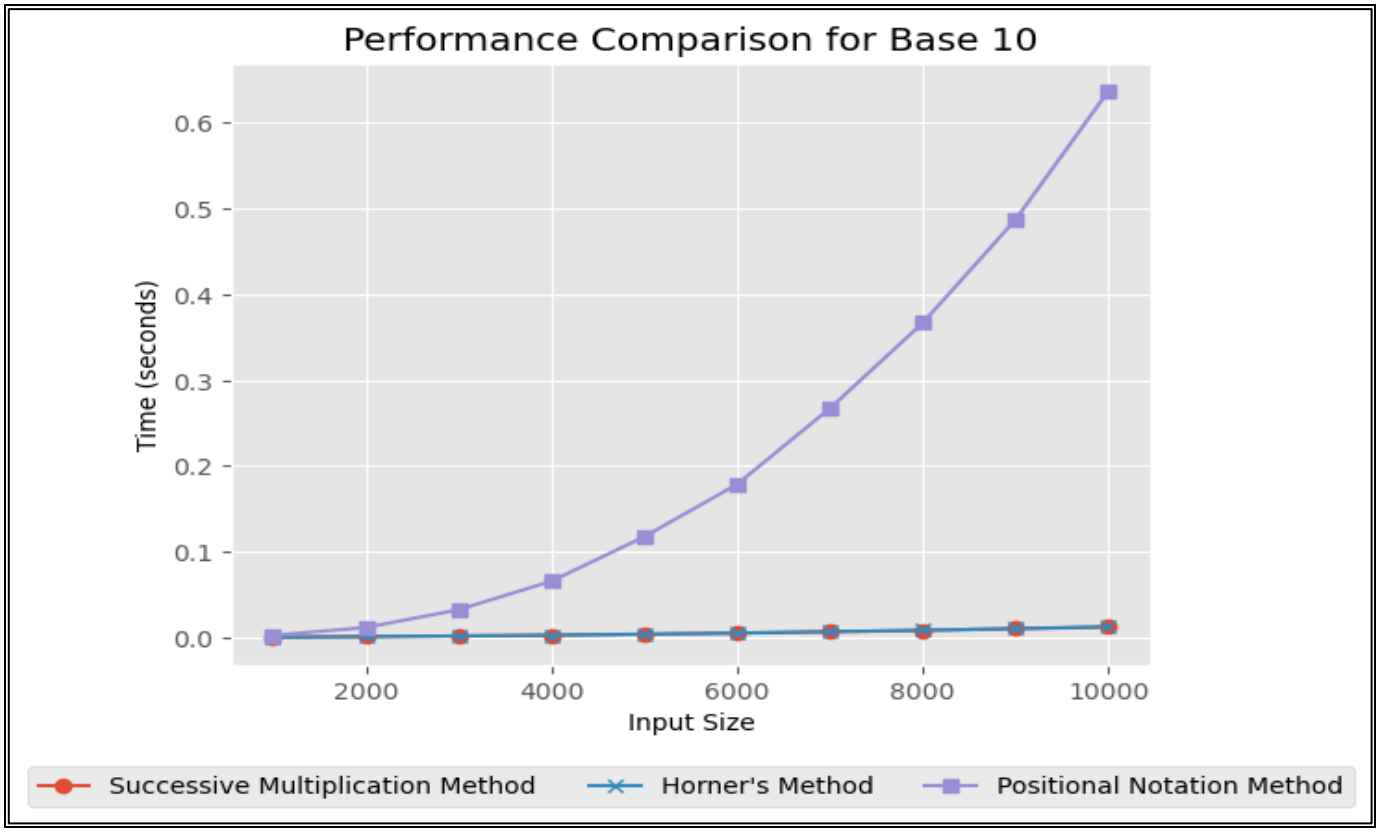

Fig 3: Performance Comparison for Base 10

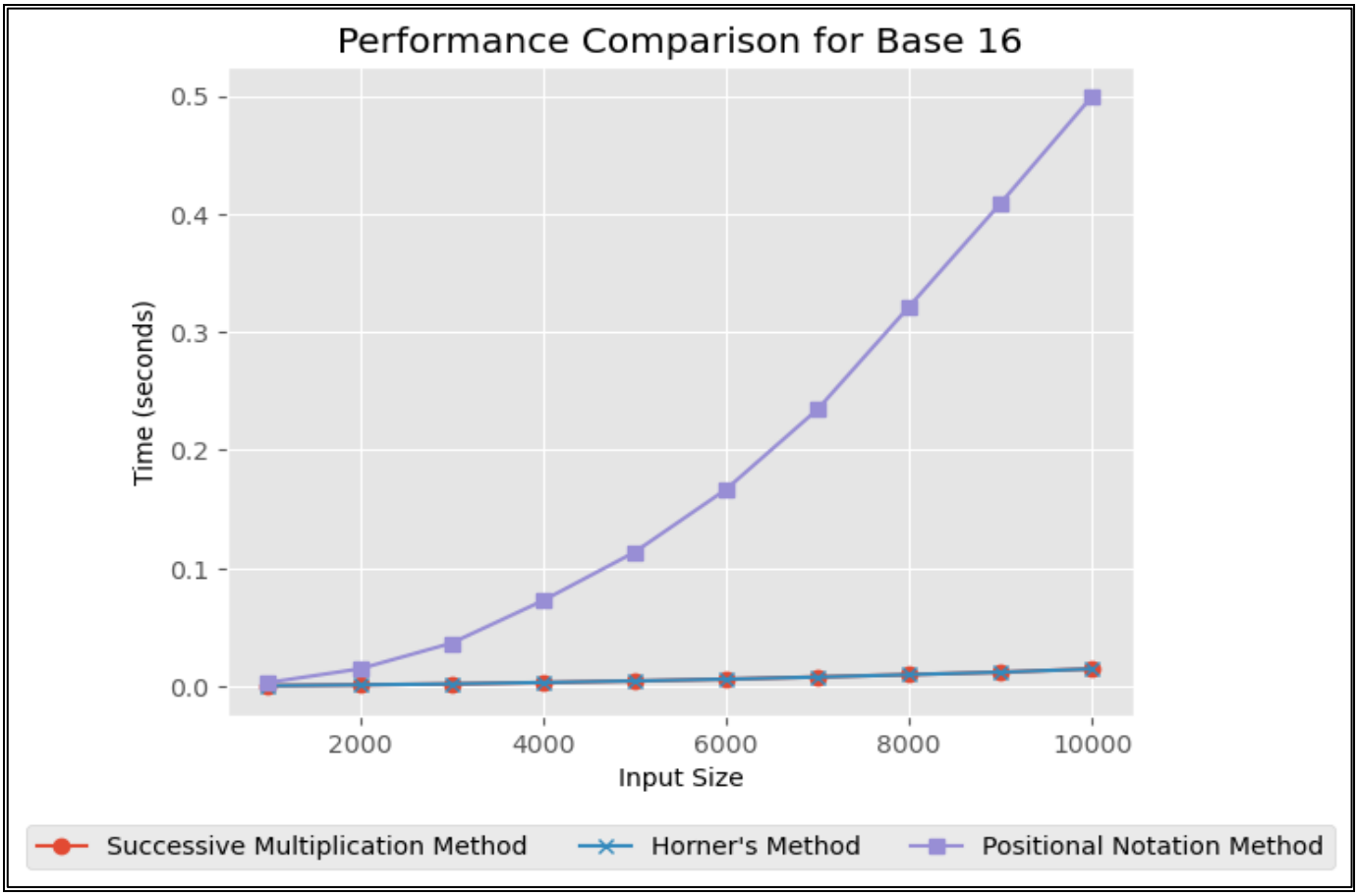

Fig 4: Performance Comparison for Base 16

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/IJISRT24AUG066

B. Discussions

➢ *Description Statistics for Binary Conversion*

- **Mean**: The Positional Notation Method averaged 0.035325 seconds, significantly slower than the Successive Multiplication Method (0.002435 seconds) and Horner's Method (0.002392 seconds).
- **Standard Deviation**: The Positional Notation Method exhibited higher variability (0.033117 seconds) compared to the Successive Multiplication Method (0.001795 seconds) and Horner's Method (0.001792 seconds).
- **Minimum Times**: Horner's Method achieved the lowest minimum time of 0.000184 seconds, demonstrating the best performance in optimal cases.
- **Median Times**: The median time for Horner's Method (0.002132 seconds) was slightly better than the Successive Multiplication Method (0.002167 seconds).
- **Maximum Times**: The Positional Notation Method had the highest maximum time (0.096159 seconds), indicating poorer performance in the worst-case scenarios.
- **Conclusion**: Horner's Method demonstrated superior average performance and consistency compared to the Successive Multiplication Method and the Positional Notation Method.

➢ *Description Statistics for Octal Conversion*

- **Mean**: Horner's Method had an average time of 0.011678 seconds, marginally better than the Successive Multiplication Method (0.011720 seconds). Both were significantly faster than the Positional Notation Method (0.132043 seconds).
- **Standard Deviation**: The Positional Notation Method showed high variability (0.124382 seconds), while the Successive Multiplication Method (0.009258 seconds) and Horner's Method (0.009244 seconds) had lower, similar deviations.
- **Minimum Times**: Horner's Method had the lowest minimum time of 0.000621 seconds.
- **Median Times**: Horner's Method's median time (0.010028 seconds) was slightly better than the Successive Multiplication Method (0.010336 seconds).
- **Maximum Times**: The Positional Notation Method had the highest maximum time (0.362079 seconds).
- **Conclusion**: Horner's Method was the most efficient for octal conversion, outperforming both the Successive Multiplication Method and the Positional Notation Method.

➢ *Description Statistics for Decimal Conversion*

- **Mean**: Horner's Method averaged 0.072216 seconds, outperforming the Successive Multiplication Method (0.072493 seconds) and the Positional Notation Method (0.069065 seconds).
- **Standard Deviation**: The Positional Notation Method had the highest deviation (0.074013 seconds), while the Successive Multiplication Method (0.023821 seconds) and Horner's Method (0.023772 seconds) were more consistent.
- **Minimum Times**: Horner's Method had the lowest minimum time of 0.000645 seconds.
- **Median Times**: Horner's Method's median time (0.012568 seconds) was slightly lower than the Successive Multiplication Method (0.013134 seconds).
- **Maximum Times**: The Positional Notation Method had the highest maximum time (0.637513 seconds).
- **Conclusion**: Horner's Method consistently performed better across decimal conversions, with the lowest average and maximum times.

➢ *Description Statistics for Hexadecimal Conversion*

- **Mean**: Horner's Method averaged 0.013057 seconds, showing better performance than the Successive Multiplication Method (0.013276 seconds) and the Positional Notation Method (0.065568 seconds).
- **Standard Deviation**: Horner's Method had the lowest deviation (0.009762 seconds), while the Positional Notation Method exhibited higher variability (0.054056 seconds).
- **Minimum Times**: Horner's Method had the lowest minimum time of 0.000756 seconds.
- **Median Times**: The median time for Horner's Method (0.013112 seconds) was slightly better than the Successive Multiplication Method (0.013420 seconds).
- **Maximum Times**: The Positional Notation Method had the highest maximum time (1.251832 seconds).
- **Conclusion**: Horner's Method showed the best performance for hexadecimal conversions, both in average and maximum execution times.

C. Summary of Findings

- Horner's Method consistently outperformed the Successive Multiplication Method and the Positional Notation Method across all tested bases and input sizes.
- The Positional Notation Method showed the highest execution times and variability, especially with larger inputs and more complex bases.
- The Successive Multiplication Method performed better than the Positional Notation Method but was less efficient compared to Horner's Method.

D. Practical Implications

The efficiency of Horner's Method makes it particularly suitable for applications requiring frequent base conversions, such as real-time systems and large-scale data processing tasks. Its consistent performance across various bases and input sizes highlights its practical advantages in computational settings.

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/IJISRT24AUG066

VI. SUMMARY, CONCLUSION AND RECOMMENDATIONS

A. *Summary of Key Findings*

This study evaluated the performance of three base conversion methods: Successive Multiplication Method, Positional Notation Method, and Horner's Method. The primary findings are as follows:

- **Performance Comparison**: Horner's Method consistently demonstrated superior performance compared to the Successive Multiplication Method and the Positional Notation Method across all tested bases (binary, octal, decimal, hexadecimal) and input sizes (ranging from 1000 to 10,000 digits). It showed the lowest average execution times and exhibited the most consistent performance with minimal variability.
- **Execution Times**: Horner's Method outperformed both the Successive Multiplication Method and the Positional Notation Method in terms of execution time. For binary and octal conversions, Horner's Method provided the best average and maximum execution times. Similarly, in decimal and hexadecimal conversions, Horner's Method maintained its lead with lower average execution times and reduced variability.
- **Computational Complexity**: All three methods share a computational complexity of $O(n)$, where n is the number of digits. However, the practical performance of Horner's Method is enhanced by its reduced number of operations, contributing to its efficiency.

B. *Conclusion on the Efficiency of Horner's Method for Base Conversion*

Horner's Method is the most efficient of the three base conversion methods evaluated. Its ability to deliver consistently low execution times and its minimal variability make it a highly reliable choice for base conversion tasks. This efficiency is particularly valuable in applications that require frequent base conversions or handle large-scale data, where performance optimization is crucial.

➢ *Key Advantages of Horner's Method:*

- **Reduced Computational Overhead**: By minimizing the number of multiplications and additions, Horner's Method achieves faster execution times.
- **Consistency**: The method provides stable performance across various input sizes and bases, making it a robust solution for different computational scenarios.

C. *Recommendations for Practitioners and Researchers*

- **For Practitioners**: It is recommended to implement Horner's Method for base conversion tasks, especially in performance-critical applications where execution speed and efficiency are paramount. The method's consistent performance across different bases and input sizes ensures reliable results and optimized processing times.
- **For Researchers**: Further investigation into Horner's Method could include exploring its performance in multi-threaded or parallel processing environments and evaluating its efficiency on different hardware architectures. Additionally, researchers should consider testing the method with a broader range of bases and input sizes to validate its performance under varied conditions.
- **Future Studies**: Researchers should also explore hybrid approaches that combine Horner's Method with other optimization techniques, such as hardware acceleration or algorithmic improvements, to enhance performance even further. Comparing Horner's Method with emerging base conversion techniques could provide additional insights into its relative efficiency and applicability.

ACKNOWLEDGMENT

The Authors would like to express their heartfelt thanks to the anonymous reviewers for their detailed and constructive feedback, which greatly contributed to improving the quality of this manuscript. Our sincere thanks to Regentropfen University College and the Department of Computer Science for providing the necessary resources and facilities to carry out this research. We also appreciate the support and collaboration of our colleagues and departments for their technical assistance and valuable insights throughout the study.

REFERENCES

[1]. Numbers, S. B., & Codes, B. Digital Computers and Digital Systems Binary Numbers 4 Number Base Conversions 6 Octal and Hexadecimal Numbers 9 Complements 10.

[2]. Rajaraman, V. (2018). *Computer oriented numerical methods.* PHI Learning Pvt. Ltd..

[3]. Khan, S. A. (2011). *Digital design of signal processing systems: a practical approach.* John Wiley & Sons.

[4]. Mann, Z. Á. (2015). Allocation of virtual machines in cloud data centers—a survey of problem models and optimization algorithms. *Acm Computing Surveys (CSUR)*, *48*(1), 1-34.

[5]. Potkonjak, M., Srivastava, M. B., & Chandrakasan, A. P. (1996). Multiple constant multiplications: Efficient and versatile framework and algorithms for exploring common subexpression elimination. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, *15*(2), 151-165.

[6]. Kumar, A. A. (2016). *Fundamentals of digital circuits.* PHI Learning Pvt. Ltd..

[7]. Minato, S. I. (1995). *Binary decision diagrams and applications for VLSI CAD* (Vol. 342). Springer Science & Business Media.

[8]. Netz, L. (2015). *Using horner schemes to improve the efficiency and precision of interval constraint propagation* (Doctoral dissertation, Bachelor's Thesis, RWTH Aachen University, 2015.⇒ 14).

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/IJISRT24AUG066

[9]. Zeineddine, A., Nafkha, A., Paquelet, S., Moy, C., & Jezequel, P. Y. (2021). Comprehensive survey of FIR-based sample rate conversion. *Journal of Signal Processing Systems*, *93*, 113-125.
[10]. Howard, J. P. (2017). *Computational Methods for Numerical Analysis with R*. Chapman and Hall/CRC.
[11]. Parhami, B. (2010). *Computer arithmetic* (Vol. 20, No. 00). New York, NY: Oxford university press.
[12]. Patankar, U. S., & Koel, A. (2021). Review of basic classes of dividers based on division algorithm. *IEEE Access*, *9*, 23035-23069.
[13]. Aho, A. V., Hopcroft, J. E., & Ullman, J. D. (1974). The Design and Analysis of Computer Algorithms. Addison-Wesley.
[14]. Goodrich, M. T., Tamassia, R., Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). Data Structures & Algorithms in Java. *Computer Science*, *4003*, 233.
[15]. Hennessy, J. L., & Patterson, D. A. (2011). Computer Architecture: A Quantitative Approach (5th ed.). Morgan Kaufmann.
[16]. Hwang, K., & Briggs, F. A. (1984). Computer Architecture and Parallel Processing. McGraw-Hill.
[17]. Knuth, D. E. (1997). The Art of Computer Programming, Volume 2: Seminumerical Algorithms (3rd ed.). Addison-Wesley.
[18]. Press, W. H., Teukolsky, S. A., Vetterling, W. T., & Flannery, B. P. (2007). Numerical Recipes: The Art of Scientific Computing (3rd ed.). Cambridge University Press.
[19]. Quinn, M. J. (1987). Designing Efficient Algorithms for Parallel Computers. McGraw-Hill.
[20]. Azure, I., Wiredu, J. K., Musah, A., & Akolgo, E. (2023). AI-Enhanced Performance Evaluation of Python, MATLAB, and Scilab for Solving Nonlinear Systems of Equations: A Comparative Study Using the Broyden Method. *American Journal of Computational Mathematics*, *13*(4), 644-677. DOI: 10.4236/ajcm.2023.134036
[21]. Armah, G. K., Awonekai, E. A., Owagu, U. F., & Wiredu, J. K. (2023). Customer Preference for Electronic Payment Systems for Goods: A Case Study of Some Selected Shopping Malls, Bolgatanga. *Asian Journal of Research in Computer Science*, *16*(4), 257-270. Available:https://doi.org/10.9734/ajrcos/2023/v16i4387
[22]. Wiredu, J. K., Abuba, N. S., & Zakaria, H. (2024). Impact of Generative AI in Academic Integrity and Learning Outcomes: A Case Study in the Upper East Region. Asian Journal of Research in Computer Science, 17(7), 214–232. https://doi.org/10.9734/ajrcos/2024/v17i7491.