Codesters: Design and Implementation of slow and Fast Divison algorithm in computer Architecture

Mr. B. Sriman
Department of Computer Science
Rajalakshmi intistute of Technology
Sriman.b@ritchennai.edu.in

Kousick krishna S
Computer Science
Rajalakshmi Institute of Technology
Chennai, India
kousickkrishna.s.2021.cse@ritchennai.edu.in

Keerthika R
Computer Science
Rajalakshmi Institute of Technology
Chennai,India
keerthika.r.2021.cse@ritchennai.edu.in

Hareene D
Computer Science
Rajalakshmi Institute of Technology
Chennai, India
hareene.d.2021.cse@ritchennai.edu.in

***Abstract*—The basic arithmetic operation of division is commonly applied in a variety of computer applications. For high-performance computations in computer architecture, efficient division algorithms are essential. Fast division and slow division algorithms are two frequently used division methods, both with benefits and drawbacks. Fast division algorithms make use of sophisticated technology and mathematical techniques to deliver quick results. Compared to slow division algorithms, these techniques often demand more complicated circuitry and computational resources but offer faster execution speeds. In applications where speed is crucial, such real-time signal processing, scientific simulations, and cryptography, fast division techniques are especially helpful. Newton-Raphson division is one of the most well-known fast division algorithms. It is based on Newton's approach from calculus and uses iterative approximation. By repeatedly improving a starting estimate, the algorithm quickly approaches the quotient. In order to divide the workload and accelerate the overall calculation, fast division algorithms frequently take advantage of parallelism by using several processing units or parallel pipelines.**

Keywords—Newton_Raphson, iterative approximation, parallel pipelines, simulations.

I. INTRODUCTION

Fundamental methods in computer architecture for effectively conducting division tasks include fast and slow division algorithms. The rapid division algorithm reduces the amount of computation needed to provide accurate division results by utilizing advanced arithmetic methods and optimization techniques. The slow division algorithm, on the other hand, prioritizes accuracy over speed and uses simpler arithmetic operations to guarantee accurate division results.

The importance of both algorithms in computer design is highlighted in this abstract, along with their unique traits and trade-offs between speed and accuracy. The performance and effectiveness of division operations in a variety of computer tasks can be significantly improved by comprehending and putting these techniques into practice

Fast division algorithms make use of sophisticated technology and mathematical techniques to deliver quick results. Compared to slow division algorithms, these techniques often demand more complicated circuitry and computational resources but offer faster execution speeds. In applications where speed is crucial,such real-time signal processing, scientific simulations, and cryptography, fast division techniques are especially helpful.

In contrast, slow division algorithms, sometimes referred to as non-restoring division algorithms, are easier to use and need less technology than their rapid counterparts. They are frequently employed in applications where speed is subordinated to hardware cost and simplicity. Slow division algorithms often entail repeated subtraction operations, which makes them less time efficient but better suited for systems with limited resources**.**

II. LITERATURE REVIEW

Fast division algorithms are made to maximize computation efficiency by reducing the number of steps needed. These algorithms are designed to deliver approximations of outcomes rapidly, making them ideal for applications where performance is of utmost importance. One such strategy uses an iterative process to discover the reciprocal of the divisor and multiply it by the dividend. This algorithm is known as the Newton-Raphson division algorithm. Due to the repetitive calculations required, the algorithm's complexity may be relatively high even if it delivers quick convergence. In situations when quick approximation is crucial, the Newton-Raphson technique is still useful.

The Goldschmidt division algorithm is another renowned rapid division algorithm. To compute division quickly, this iterative approach uses multiplicative and subtractive operations. It accelerates convergence by using reciprocal approximations to increase the correctness of the outcome.

Slow division algorithms are frequently used in situations where precision is of the utmost significance because they value accuracy over speed. For manual division calculations, the long division algorithm is a well-known and simple technique. Even if it produces precise findings, it might still take a while, especially when working with huge quantities. The long division algorithm ensures accuracy by repeatedly removing multiples of the dividend from the divisor until the remainder is smaller than the divisor.

Two further examples of slow division algorithms are the restoring and non-restoring division algorithms. Both algorithms carry out division by repeatedly subtracting numbers, although they do it in slightly different ways. When the remainder turns negative, the restoring division algorithm recovers the original value by comparing it to the divisor. The non-restoring division algorithm, on the other hand, divides the result and then modifies the remainder according to its sign. These algorithms achieve a reasonable level of computing economy while still providing results that are reasonably precise, balancing accuracy and speed. They are valuable in particular situations due to their simple hardware implementation.

III. OBJECTIVE

The basic purpose of fast division algorithms is to optimize the efficiency of division computations by decreasing the number of computational steps required. These algorithms prioritize speed and aim to provide approximate results quickly, making them particularly suitable for applications where speed is a critical factor.

- Minimize computational steps: Fast division algorithms' main goal is to minimize the number of calculation steps needed to conduct division computations. These algorithms try to reduce the amount of operations required to obtain quicker execution speeds.

- Provide approximate results quickly: Fast division algorithms put speed before accuracy. They are appropriate for applications where a perfect solution is not required or when a trade-off between speed and accuracy is acceptable because their goal is to swiftly produce approximative results.

- Optimize efficiency: One of the main goals of rapid division algorithms is the effectiveness of division computations. These algorithms optimize the use of computing resources by reducing computational overhead, allowing for quicker overall execution times for jobs involving frequent or repetitive divisions.

- Balance between speed and accuracy: The goal of fast division algorithms is to balance speed and precision. These algorithms try to retain an acceptable level of accuracy that is enough for the intended application or computational activity while focusing on delivering rapid results.

- Suitable for real-time and high-performance computing: Fast division algorithms are very helpful in situations requiring high-performance or real-time processing. Their goal is to make it possible to process enormous datasets, difficult mathematical operations, or time-sensitive applications that need quick division calculations in an effective manner.

Overall, the objective of fake news detection is to safeguard the integrity of information and protect society from the potentially harmful consequences of false news.

IV. OUTCOMES

- The reduction in calculation execution time for division operations is the main benefit of fast division algorithms. These methods enable quicker processing of division operations, resulting in enhanced overall computational efficiency, by decreasing the number of computational steps necessary.
- Fast division techniques reduce the computational overhead associated with division calculations, maximizing the use of computing resources. This improves computing efficiency, making it possible to handle jobs that require frequent or repetitive divisions more quickly.
- Hardware optimization options are frequently presented by fast division algorithms. They can take use of the algorithm's efficiency to increase the speed and effectiveness of division computations in hardware-intensive systems by being implemented in specialized hardware designs or bespoke circuits.
- Algorithms for fast division are well suited for applications that value efficiency and speed over absolute precision. These algorithms are useful in a variety of fields, including as scientific simulations, numerical analysis, digital signal processing, encryption, graphics rendering, and real-time systems

V. CHALLENGES

Implementing fast and slow division algorithm can encounter several challenges that must be addressed. Some key challenges in this context include:

1. Accuracy Trade-off:

Fast division algorithms frequently favor speed over accuracy. While they deliver fast results, the division computation's precision may suffer in comparison to slower, more accurate division algorithms. It can be difficult to strike the ideal balance between speed and accuracy, particularly in applications where precision is essential

2. Iterative Calculations:

Iterative calculations are used in some fast division methods, such as the Newton-Raphson algorithm. The complexity and computational burden added by iterative algorithms may reduce the algorithm's overall effectiveness.

3. Algorithm Complexity:

Fast division algorithms might be difficult to use and comprehend. They might call for sophisticated computational techniques and complex mathematical concepts. Correctly designing and implementing these algorithms can be difficult, especially for novices or those without a solid mathematical foundation.

4. Hardware Limitations:

Fast division algorithms that are designed for quick computations may be difficult to implement in hardware. Hardware limitations, such as scarce resources or slow clock speeds, may make it difficult for rapid division algorithms to operate effectively, thus reducing their performance advantages.

5. Numerical Instabilities:

Some algorithms for doing quick division might be prone to numerical instability or convergence problems. These algorithms' approximation strategies may introduce flaws or behave in an unpredictable manner under certain circumstances, producing erroneous or doubtful results.

6. Application Compatibility:

Not all applications may be suited for fast division algorithms. The speed offered by fast algorithms might not be sufficient for some computational workloads that call for precise and accurate division computations. It might be difficult to determine the applicability and restrictions of quick division algorithms in certain contexts or domains.

7. Implementation Overhead:

The goal of rapid division algorithms is to minimize the number of processing steps, however there may be additional implementation cost due to the need to manage convergence criteria, set up beginning conditions, or handle special instances. The overall effectiveness and usefulness of the algorithm may be impacted by these overheads.

VI. ARCHITECTURE

The architecture/algorithm model proposed for fast and slow division algorithm using Newton-Raphson consists of several key components that work together to achieve accurate results. The following is an overview of the architecture:

1. Define Input and Output Requirements

Define the required output format, such as the quotient and remainder, as well as the input requirements, such as the dividend and divisor.Think about any particular input restrictions, such as data size, the range of acceptable numbers, or any exceptional instances that must be addressed.

2. Select a Suitable Fast Division Algorithm

Select a certain quick division algorithm that satisfies the conditions and limitations of your application.Think about things like implementation ease, hardware compatibility, speed, and accuracy.

3. Algorithm Initialization

The logistic regression model may undergo further optimization techniques, such as feature selection or regularization, to improve its performance and generalization capabilities. This step helps refine the model and enhance its ability to handle unseen data.

4. Perform Preprocessing

Preprocessing the dividend or divisor may be necessary to assure compatibility or increase computing performance, depending on the chosen algorithm and input conditions. The input values may need to be scaled, normalized, or modified in order to achieve this.

5. Iterative Computation

Compute using the fast division algorithm's fundamental phases. This usually entails doing a series of calculations repeatedly until a termination condition is satisfied. Consider any tweaks or optimizations needed for your application as you follow the precise procedures and formulas connected with the selected approach.

6. Handling Special Cases

Recognize and address any unique situations that may occur during the division process. Division by zero, division with negative values, and addressing overflow or underflow circumstances are a few examples. Implement suitable error or exception handling procedures to ensure robustness.

7. Postprocessing

Execute any necessary postprocessing procedures when the division computation is finished to get the output format you want. Rounding, formatting, or changing the results' representation may be included in this.

8. Validating and Testing

To validate accuracy and correctness, compare the built rapid division method to known test cases.Verify the algorithm's performance and dependability by thoroughly testing it with a range of input possibilities, including edge cases.

9. Optimize for Hardware

Depending on the intended hardware platform and limitations, examine the implemented algorithm for any options for hardware optimization, such as parallel processing, pipelining, or custom circuit design.Boost the quick division algorithm's performance by implementing hardware enhancements.

10. Document and Maintain

The architecture that was used should be documented, along with any optimizations that were made and algorithmic details.Keep using the fast division algorithm you've already created, making sure it can be adjusted for future software, hardware, or hardware-related improvements.

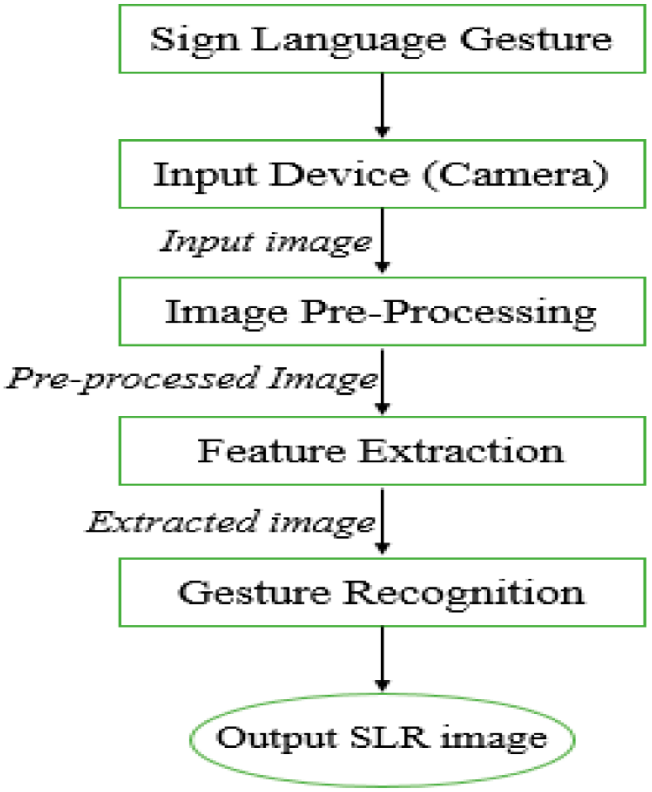


Fig . 1 **architecture of this algorithm**

VII. ACCURACY

Depending on the particular method chosen and how it is implemented, a rapid division algorithm's accuracy can change. Fast division algorithms frequently trade off accuracy for speed, thus there might be some accuracy trade-off. It's crucial to remember that quick division algorithms still make an effort to deliver outcomes that are precise enough for a variety of real-world applications.

The approximation methods, convergence requirements, and particular numerical values used in the division computation can all have an impact on how accurate a fast division algorithm is. Fast division algorithms generally seek to produce approximations that fall within a reasonable margin of the precise division result.

A fast division algorithm's performance on a variety of test cases, including both common scenarios and edge circumstances, must be taken into account in order to assess how accurate it is. Comparisons can be done with known exact results derived through division methods that are slower but more accurate, or by applying recognized mathematical formulas.

Overall, rapid division algorithms try to produce answers that are accurate enough for many real-world situations, even though they may make certain accuracy sacrifices for speed. When applying rapid division algorithms in real-world applications, it's crucial to do enough testing, validation, and evaluation of the algorithm's constraints and expected levels of accuracy.

VIII. RESULT CACHES

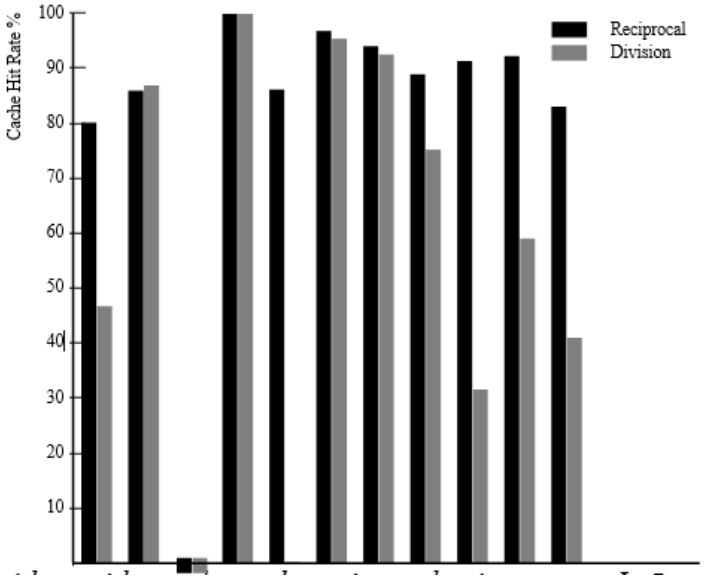
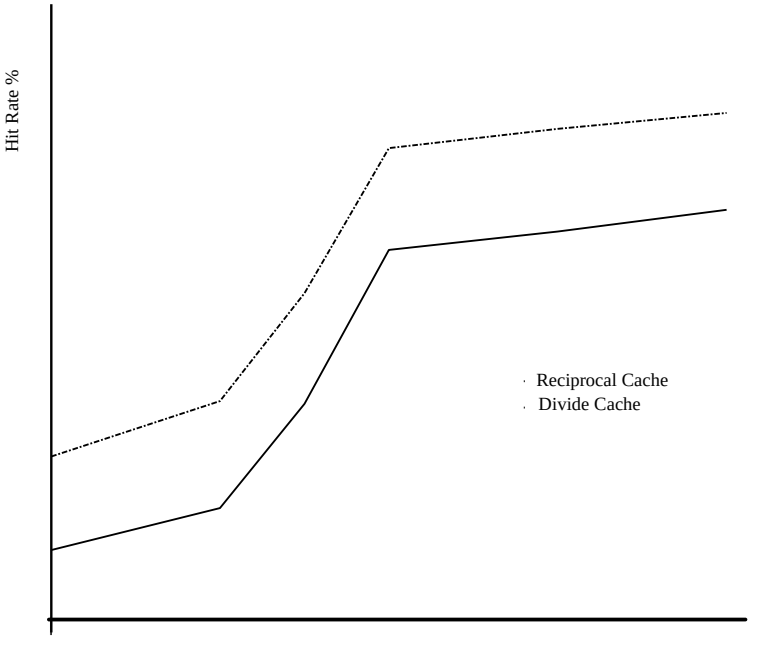


Fig 2 **Cache rate**



IX. CODE IMPLEMENTATION

Fast divison Algorithm(Newton-Raphson Method):

```
def fast_division(dividend, divisor, precision):
approx = 1.0 / divisor
# Initial approximation
for _ in range(precision):
 approx = approx * (2 - divisor * approx)
quotient = dividend * approx
return quotient
```

Slow division Algorithm(Long division Algorithm):

```
def slow_division(dividend, divisor):
quotient = 0
remainder = dividend
while remainder >= divisor:
quotient += 1
remainder -= divisor
 return quotient, remainder
```

X. CONCLUSION

In conclusion, by minimizing the number of computational steps and delivering prompt approximations, rapid division algorithms offer effective solutions for division computations. They may compromise some degree of accuracy, but they are useful in applications where speed is more important than perfect accuracy. The selection of a rapid division algorithm is based on the demands and limitations of the application. These techniques can considerably improve the performance and efficiency of division calculations in a variety of computational tasks through careful implementation, testing, and validation.

XI. REFERENCES

[1] D. E. Atkins. Higher-radix division using estimates of the divisor and partial remainders. IEEE Transactions on Computers, C-17(10), October 1968.

[2] W. S. Briggs and D. W. Matula. A 17x69 Bit multiply and add unit with redundant binary feedback and single cycle latency. In Proceedings of the 11th IEEE Symposium on Computer Arithmetic, pages 163{170, July 1993.

[3] J. Cortadella and T. Lang. Division with speculation of quotient digits. In Proceedings of the 11th IEEE Symposium on Computer Arithmetic, pages 87{94, July 1993.

[4] D. L. Fowler et al. An accurate, high speed implementation of division by reciprocal approximation. In Proceedings of the 9th IEEE Symposium on Computer Arithmetic, pages 60{67, September 1989.

[3] D. DasSarma and D. Matula. Measuring the accuracy of ROM reciprocal tables. IEEE Transactions on Computers, 43(8):932{940, August 1994.

[4] D. DasSarma and D. Matula. Faithful bipartite ROM reciprocal tables. In Proceedings of the 12th IEEE Symposium on Computer Arithmetic, pages 12{25, July 1995.

[5] M. D. Ercegovac and T. Lang. Radix-4 square root without initial PLA. IEEE Transactions on Computers, 39(8):1016{1024, August 1990.

[6] M. D. Ercegovac and T. Lang. Simple radix-4 division with operands scaling. IEEE Transactions on Computers, C-39(9):1204{1207, September 1990.

[7] M. D. Ercegovac and T. Lang. Division and Square Root: Digit-Recurrence Algorithms and Implementations. Kluwer Academic Publishers, 1994.

[8] H. Kabuo et al. Accurate rounding scheme for the Newton-Raphson method using redundant binary representation. IEEE Transactions on Computers, 43(1):43{51, January 1994.

[9] J. Mulder et al. An area model for on-chip memories and its application. IEEE Journal of Solid-State Circuits, 26(2), February 1991.

[10] M. D. Ercegovac et al. Very high radix division with selection by rounding and prescaling. In Proceedings of the 11th IEEE Symposium on Computer Arithmetic, pages 112{ 119, July 1993.

[11] M. Darley et al. The TMS390C602A oating-point coprocessor for Sparc systems. IEEE Micro, 10(3):36{47, June 1990.

[12] M. J. Schulte et al. Optimal initial approximations for the Newton-Raphson division algorithm. Computing, 53:233{242, 1994.

[13] D. E. Atkins. Higher-radix division using estimates of the divisor and partial remainders. IEEE Transactions on Computers, C-17(10), October 1968.

[14] W. S. Briggs and D. W. Matula. A 17x69 Bit multiply and add unit with redundant binary feedback and single cycle latency. In Proceedings of the 11th IEEE Symposium on Computer Arithmetic, pages 163{170, July 1993.