

Project 0: Ancient Multiplications Efficiency

Binjie Liang, 2023064642, bliang@estudiantec.cr and Esteban Secaida, 2019042589, esecaida@estudiantec.cr

Abstract—The quest for efficient algorithms has led to re-visiting historical methods like Russian Peasant multiplication, which uses a binary-like process of doubling and halving. Despite its ancient origins, its performance relative to conventional multiplication algorithms remains underexplored. This paper investigates the implementation and computational efficiency of ancient multiplications methods compared to standard methods.

Index Terms—Bits, Binary, Least Significant Bit

I. INTRODUCTION

Addition, subtraction, multiplication, and division—these four operations form the basic arithmetic of mathematics, serving as the foundation for more advanced mathematical concepts and applications. Among these, multiplication is particularly crucial, over time, various multiplication techniques have emerged, each with unique historical and cultural significance. One such method, known as *Russian Peasant Multiplication* [1], finds its roots in ancient *Egyptian mathematics* [2] and employs a binary-like process of iterative doubling and halving.

The primary objective of this research is to analyze the computational efficiency of the Russian Peasant Multiplication algorithm. By implementing both this algorithm and the standard multiplication function in a controlled environment, we aim to measure and compare their execution times. This comparison will shed light on the practical implications of utilizing historical algorithms in contemporary computing, particularly in scenarios where computational efficiency is paramount.

II. DEVELOPMENT OF THE RUSSIAN PEASANT MULTIPLICATION ALGORITHM

A. What's Russian Peasant Method?

In the present day, most people learn multiplication using standard methods, such as the long multiplication algorithm (example in Fig.1), which involves multiplying digits and summing intermediate products. However, historical methods like the Russian Peasant Multiplication offer an alternative approach with a unique process based on iterative doubling and halving. This ancient technique, originating from ancient Egyptian mathematics, leverages a binary-like procedure that can be quite effective.

The Russian Peasant Multiplication algorithm operates by repeatedly halving one of the multiplicands and doubling the other until the former reaches zero. At each step, if the halved value is odd, the corresponding doubled value is added to the result.

$$\begin{array}{r}
 12 \\
 \times 32 \\
 \hline
 24 \\
 20 \\
 60 \\
 300 \\
 \hline
 384
 \end{array}$$

Fig. 1. An example of the standard "long multiplication".

B. How this method works?

Given the numbers A and B

- 1) Initialize `result` as 0.
- 2) While B is not equal to 0:
 - a) If B is odd, add A to `result`.
 - b) Double A.
 - c) Divide B by 2.
- 3) At the end, `result` will contain the product of A and B.

This approach not only highlights the algorithm's simplicity but also its potential efficiency in certain computational contexts. By exploring this historical method, we can gain insights into its practical applications and performance compared to modern multiplication algorithms.

In the following figures, we will demonstrate a short example of how to perform the Russian Peasant Multiplication algorithm.

$$\begin{array}{cc}
 A & B \\
 13 & 24
 \end{array}$$

Fig. 2. Initially, we have numbers A and B.

Initially, as shown in Fig. 2, we have number A (13) and number B (24). The result is initialized to 0.

The next step is to check if B is greater than 0. If it is, we check if B is an odd number. If B is odd, we add A to the result. Then, we double the value of A and halve the value of B. This process repeats until B equals 0 when rounded down.

A	B
13	24
26	12
52	6
104	3
208	1

Fig. 3. Doubling A and halving B.

As shown in Fig. 3, after each step, we double the value of A and halve the value of B.

A	B
13	24
26	12
52	6
104	3
208	1

Fig. 4. Ruling out the rows where B is even.

After repeating the process, we rule out the rows where B is even, as illustrated in Fig. 4.

A	B
13	24
26	12
52	6
104	3
+	208
208	1
result = 312	

Fig. 5. Result of the product between A and B.

Finally, as shown in Fig. 5, we sum all the remaining values in the A column. This sum gives us the result of the product of A and B.

C. Relationship with binary system

You may be curious about the connection between this method and the binary system. The Russian Peasant Multiplication algorithm inherently utilizes the principles of the binary

number system through its process of iterative doubling and halving.

In this method, when A is doubled, in binary it represents a left shift. While on the other hand when we halve B, we are actually doing a right shift in bits. To know if our number in B is odd, we just need to make a AND operation with the least significant bit, if the LSB is equal to 1, then is 1.

For a better understanding, we will revisit the example previously shown, but using binary representation.

A	B
1101	11000

Fig. 6. Figure 2 numbers in binary representation

Converting 13 and 24 into binary, we get 1101 and 11000, respectively. The next step involves performing a left shift on A and a right shift on B.

A	B
1101	11000
11010	1100
110100	110
1101000	11
11010000	1

Fig. 7. Doubling and halving using left and right shifts

After shifting both numbers, the next step is to exclude the rows where the least significant bit (LSB) of B is 0.

A	B
1101	11000
11010	1100
110100	110
1101000	11
+	11010000
11010000	1
result = 100111000	

Fig. 8. Binary result after excluding even numbers

By excluding the rows where B's LSB is 0, the remaining task is to sum all the remaining numbers in the A column. This results in a binary value of 100111000, which is the final product.

Check out that the process is exactly the same, demonstrating that the Russian Peasant Multiplication algorithm leverages

the principles of binary arithmetic, further underscoring its efficiency and relevance in computational contexts.

III. HANDS-ON APPLICATION

Algorithm efficiency is key in computer science. As we tackle more complex problems, evaluating even basic operations like multiplication becomes crucial. Efficiency isn't just about speed, it also involves memory use and how well methods scale. We'll examine two multiplication approaches: the conventional method and Russian Peasant Multiplication, comparing their performance across various metrics.

The standard multiplication method, taught in schools and widely used, relies on basic arithmetic. It's fast and efficient, especially on modern processors. However, techniques like Russian Peasant Multiplication offer interesting alternatives. This method, based on binary principles, might be more efficient in certain computational contexts.

1) *Methodology*: To compare these methods, we set up a thorough experiment. We use eight random numbers to create a 4x4 table, covering various multiplication scenarios. Each operation is repeated N times to get accurate average execution times. We also swap the order of multiplication (a*b and b*a) to check if it affects performance.

This comparison aims to reveal the pros and cons of each method, offering insights into their practical uses by analyzing the timestamps of their performances.

IV. EMPTY VERSION

To better understand the project before analyzing the results, it is important to consider why there is an empty return version of the multiplication algorithm.

A. What's the purpose of the empty method?

Beyond being a requirement, it serves as a benchmark to determine the minimal time it takes for a function to receive two arguments and return a result. By using the empty version, which returns 0, we can establish the quickest time for a function to receive two numbers and produce an output. This benchmark allows us to compare it with the time taken to return the result of a multiplication using both the normal operation and the ancient algorithm.

After understanding the methodology and why are we using an empty version, it is time to run the program and see the timestamps of the different versions of the algorithms so we can go on and see which is the fastest and answer why by analyzing the results.

B. Empty Version in C

First, we have the results of the empty version. We have timestamps that range from 30.051ms in the 40 * 605 case to 37.815ms in the 424 * 798 case for the individual times, with the function being called 50000 times per operation.

Fig.9 version presents a time of 491.896 ms in the A * B order using C language.

And Fig.10 presents a time of 522.372 ms in the B * A order using C language.

The total time (including the prints and variables initialization from both tables) for this version is 2027.123 ms.

```

N: 50000
nullMul(A * B)
38| 40| 424| 585|
590 | 590 * 38 = 0 / 30.252000 ms | 590 * 40 = 0 / 30.324000 ms | 590 * 424 = 0 / 31.004000 ms | 590 * 585 = 0 / 30.861000 ms |
767 | 767 * 38 = 0 / 30.211000 ms | 767 * 40 = 0 / 30.112000 ms | 767 * 424 = 0 / 30.063000 ms | 767 * 585 = 0 / 30.125000 ms |
798 | 798 * 38 = 0 / 30.111000 ms | 798 * 40 = 0 / 30.227000 ms | 798 * 424 = 0 / 34.540000 ms | 798 * 585 = 0 / 34.141000 ms |
605 | 605 * 38 = 0 / 30.154000 ms | 605 * 40 = 0 / 30.163000 ms | 605 * 424 = 0 / 30.099000 ms | 605 * 585 = 0 / 30.100000 ms |
Execution time multiplying A * B returning 0 in C: 491.896 ms

```

Fig. 9. Null multiplication between A and B using C

```

nullMul(B * A)
38| 40| 424| 585|
38 | 38 * 590 = 0 / 30.186000 ms | 38 * 767 = 0 / 30.433000 ms | 38 * 798 = 0 / 30.897000 ms | 38 * 605 = 0 / 30.081000 ms |
40 | 40 * 590 = 0 / 34.748000 ms | 40 * 767 = 0 / 30.269000 ms | 40 * 798 = 0 / 30.130000 ms | 40 * 605 = 0 / 30.051000 ms |
424 | 424 * 590 = 0 / 32.050000 ms | 424 * 767 = 0 / 30.113000 ms | 424 * 798 = 0 / 37.353000 ms | 424 * 605 = 0 / 33.940000 ms |
585 | 585 * 590 = 0 / 30.175000 ms | 585 * 767 = 0 / 37.485000 ms | 585 * 798 = 0 / 36.188000 ms | 585 * 605 = 0 / 37.117000 ms |
Execution time multiplying B * A returning 0 in C: 522.372 ms
Execution time taken from both tables: 2027.123 ms

```

Fig. 10. Null multiplication between B and A using C

C. Empty version in Assembly

Using NASM Assembler, we implement an empty version of the multiplication function. This function is called from the main C program. In terms of performance, using Assembly didn't result in a significant difference compared to the C implementation.

```

nullMulASM(A * B)
38| 40| 424| 585|
590 | 590 * 38 = 0 / 33.151000 ms | 590 * 40 = 0 / 32.708000 ms | 590 * 424 = 0 / 30.467000 ms | 590 * 585 = 0 / 30.241000 ms |
767 | 767 * 38 = 0 / 30.295000 ms | 767 * 40 = 0 / 30.364000 ms | 767 * 424 = 0 / 31.459000 ms | 767 * 585 = 0 / 30.613000 ms |
798 | 798 * 38 = 0 / 30.107000 ms | 798 * 40 = 0 / 30.406000 ms | 798 * 424 = 0 / 30.324000 ms | 798 * 585 = 0 / 30.320000 ms |
605 | 605 * 38 = 0 / 30.174000 ms | 605 * 40 = 0 / 30.475000 ms | 605 * 424 = 0 / 30.324000 ms | 605 * 585 = 0 / 30.527000 ms |
Execution time multiplying A * B returning 0 in ASM: 493.078 ms

```

Fig. 11. Null multiplication between A and B using ASM

Comparing the results of Fig. 9 and Fig. 11, both representing functions that return 0 for multiplications between A and B, we observe only a marginal difference in performance. The Assembly version executed all multiplications merely 2ms faster than its C counterpart.

While in the other-hand in Fig.12, multiplying B and A in Assembly returning 0, was 34ms faster than C, but these results are not always present, this minimal performance difference suggests that for simple operations like our null multiplication, the overhead of function calls and basic register operations dominates the execution time, leaving little room for optimization through assembly programming.

V. STANDARD VERSION

A. Standard version in C

Next, we have the standard version, with individual times that range from 30.009ms in the 590 * 585 case to 38.348ms in the 40 * 590 case. These times reflect the duration of performing the operation also 50000 times.

Fig.13 version presents a time of 497ms in the A * B order.

And Fig.14 presents a time of 517ms in the B * A order.

The total time for this version is 2020.89ms.

```

nullMulASM(A * B)
38| 40| 424| 585|
590 | 590 * 38 = 0 / 33.151000 ms | 590 * 40 = 0 / 32.708000 ms | 590 * 424 = 0 / 30.467000 ms | 590 * 585 = 0 / 30.241000 ms |
767 | 767 * 38 = 0 / 30.295000 ms | 767 * 40 = 0 / 30.364000 ms | 767 * 424 = 0 / 31.459000 ms | 767 * 585 = 0 / 30.613000 ms |
798 | 798 * 38 = 0 / 30.107000 ms | 798 * 40 = 0 / 30.406000 ms | 798 * 424 = 0 / 30.324000 ms | 798 * 585 = 0 / 30.320000 ms |
605 | 605 * 38 = 0 / 30.174000 ms | 605 * 40 = 0 / 30.475000 ms | 605 * 424 = 0 / 30.324000 ms | 605 * 585 = 0 / 30.427000 ms |
Execution time multiplying A * B returning 0 in ASM: 493.078 ms

nullMulASM(B * A)
38| 40| 424| 585|
38 | 38 * 590 = 0 / 30.245000 ms | 38 * 767 = 0 / 31.011000 ms | 38 * 798 = 0 / 30.245000 ms | 38 * 605 = 0 / 32.433000 ms |
40 | 40 * 590 = 0 / 32.593000 ms | 40 * 767 = 0 / 30.211000 ms | 40 * 798 = 0 / 30.217000 ms | 40 * 605 = 0 / 30.056000 ms |
424 | 424 * 590 = 0 / 30.140000 ms | 424 * 767 = 0 / 30.320000 ms | 424 * 798 = 0 / 30.224000 ms | 424 * 605 = 0 / 30.040000 ms |
585 | 585 * 590 = 0 / 30.140000 ms | 585 * 767 = 0 / 30.290000 ms | 585 * 798 = 0 / 30.224000 ms | 585 * 605 = 0 / 30.033000 ms |
Execution time multiplying B * A returning 0 in ASM: 488.582 ms
Execution time taken from both tables: 1968.887 ms

```

Fig. 12. Null multiplication between B and A using ASM

standardMul(A * B)									
598	598 * 38 = 22420 / 38.334000 ms	598 * 40 = 23680 / 38.425000 ms	598 * 424 = 258160 / 38.294000 ms	598 * 585 = 345150 / 38.800000 ms	767	767 * 38 = 29146 / 38.141000 ms	767 * 40 = 30680 / 38.170000 ms	767 * 424 = 325288 / 38.267000 ms	767 * 585 = 448695 / 38.412000 ms
798	798 * 38 = 30324 / 35.221000 ms	798 * 40 = 31920 / 38.728000 ms	798 * 424 = 338352 / 38.271000 ms	798 * 585 = 466838 / 38.356000 ms	685	685 * 38 = 25930 / 38.758000 ms	685 * 40 = 27400 / 35.540000 ms	685 * 424 = 289520 / 31.350000 ms	685 * 585 = 353925 / 38.716000 ms

Execution time multiplying A * B using standard method in C: 497.086 ms

Fig. 13. Standard version A * B in C

standardMul(B * A)									
38	38 * 598 = 22420 / 35.892000 ms	38 * 767 = 29146 / 38.401000 ms	38 * 798 = 30324 / 37.465000 ms	38 * 685 = 22990 / 39.452000 ms	40	40 * 598 = 23680 / 38.348000 ms	40 * 767 = 30680 / 38.897000 ms	40 * 798 = 31920 / 38.664000 ms	40 * 685 = 24200 / 38.227000 ms
424	424 * 598 = 258160 / 38.210000 ms	424 * 767 = 325288 / 38.347000 ms	424 * 798 = 338352 / 38.281000 ms	424 * 685 = 256520 / 38.179000 ms	585	585 * 38 = 22090 / 38.314000 ms	585 * 767 = 448695 / 31.835000 ms	585 * 798 = 466838 / 31.124000 ms	585 * 685 = 353925 / 38.286000 ms

Execution time multiplying B * A using standard method in C: 517.812 ms
Execution time taken from both tables: 2028.899 ms

Fig. 14. Standard version B * A in C

B. Standard Assembly version

Now, lets compare it with Assembly implementation.

In Fig.15 we get an execution time of 490ms after executing all the multiplications, this is a difference of 7ms from Fig.13.

standardMulASH(A * B)									
598	598 * 38 = 22420 / 33.492000 ms	598 * 40 = 23680 / 38.532000 ms	598 * 424 = 258160 / 29.946000 ms	598 * 585 = 345150 / 38.824000 ms	767	767 * 38 = 29146 / 29.972000 ms	767 * 40 = 30680 / 29.988000 ms	767 * 424 = 325288 / 29.941000 ms	767 * 585 = 448695 / 34.935000 ms
798	798 * 38 = 30324 / 38.435000 ms	798 * 40 = 31920 / 29.980000 ms	798 * 424 = 338352 / 38.637000 ms	798 * 585 = 466838 / 38.116000 ms	685	685 * 38 = 25930 / 38.493000 ms	685 * 40 = 27400 / 38.611000 ms	685 * 424 = 289520 / 38.956000 ms	685 * 585 = 353925 / 38.306000 ms

Execution time multiplying A * B using standard method in ASM: 490.648 ms

Fig. 15. Standard version A * B in Assembly

While multiplying B * A using Assembly, it performs 19ms better than its C counterpart. we can see it in the comparison between Fig.14 and Fig.16. This indicates that modern compilers are highly efficient at optimizing simple C code, often producing machine code that's comparable to hand-written assembly in performance.

standardMulASH(B * A)									
38	38 * 598 = 22420 / 30.195000 ms	38 * 767 = 29146 / 30.089000 ms	38 * 798 = 30324 / 37.504000 ms	38 * 685 = 22990 / 36.196000 ms	40	40 * 598 = 23680 / 30.144000 ms	40 * 767 = 30680 / 30.110000 ms	40 * 798 = 31920 / 30.150000 ms	40 * 685 = 24200 / 30.140000 ms
424	424 * 598 = 258160 / 38.151000 ms	424 * 767 = 325288 / 38.080000 ms	424 * 798 = 338352 / 38.098000 ms	424 * 685 = 256520 / 38.290000 ms	585	585 * 38 = 22090 / 31.788000 ms	585 * 767 = 448695 / 31.835000 ms	585 * 798 = 466838 / 30.907000 ms	585 * 685 = 353925 / 38.130000 ms

Execution time multiplying B * A using standard method in ASM: 498.216 ms
Execution time taken from both tables: 1973.818 ms

Fig. 16. Standard version B * A in Assembly

VI. ANCIENT VERSION

A. Ancient C Version

Finally, we have the ancient version, with times for individual runs ranging from the lowest, 30.628ms in the 424 * 767 case, to the highest, 39.13ms in the 585 * 605 case.

This version presents a time of 510ms in the A * B order.

And it presents a time of 523ms in the B * A order.

The total time for this version is 2045ms.

From this test case, we could infer that the ancient algorithm doesn't perform better than its counterpart, which is the standard method, there's a average of 4 ms of difference between them. This can be proved with other different test cases.

B. Ancient Assembly Version

While with Assembly, results are barely the same.

In Fig.19 the multiplication between AxB gives a execution time of 499ms, from the 510ms of Fig.17.

While on the other way, BxA gives an execution time of 521ms, just 2ms better than Fig.18.

russianMul(A * B)									
598	598 * 38 = 22420 / 33.540000 ms	598 * 40 = 23680 / 38.674000 ms	598 * 424 = 258160 / 36.350000 ms	598 * 585 = 345150 / 38.784000 ms	767	767 * 38 = 29146 / 38.957000 ms	767 * 40 = 30680 / 38.818000 ms	767 * 424 = 325288 / 38.734000 ms	767 * 585 = 448695 / 38.720000 ms
798	798 * 38 = 30324 / 38.734000 ms	798 * 40 = 31920 / 31.547000 ms	798 * 424 = 338352 / 34.900000 ms	798 * 585 = 466838 / 31.430000 ms	685	685 * 38 = 22990 / 38.753000 ms	685 * 40 = 24200 / 38.668000 ms	685 * 424 = 289520 / 38.713000 ms	685 * 585 = 353925 / 34.694000 ms

Execution time multiplying A * B using russian method in C: 508.343 ms

Fig. 17. AxB using C Russian Peasant Method

russianMul(B * A)									
38	38 * 598 = 22420 / 36.729000 ms	38 * 767 = 29146 / 31.181000 ms	38 * 798 = 30324 / 32.841000 ms	38 * 685 = 22990 / 31.867000 ms	40	40 * 598 = 23680 / 32.869000 ms	40 * 767 = 30680 / 31.672000 ms	40 * 798 = 31920 / 31.589000 ms	40 * 685 = 24200 / 32.152000 ms
424	424 * 598 = 258160 / 31.244000 ms	424 * 767 = 325288 / 36.426000 ms	424 * 798 = 338352 / 32.803000 ms	424 * 685 = 256520 / 32.188000 ms	585	585 * 38 = 22090 / 33.391000 ms	585 * 767 = 448695 / 31.728000 ms	585 * 798 = 466838 / 31.728000 ms	585 * 685 = 353925 / 35.379000 ms

Execution time multiplying B * A using russian method in C: 523.648 ms
Execution time taken from both tables: 2045.287 ms

Fig. 18. BxA using C Russian Peasant Method

VII. CONCLUSION - PERFORMANCE ANALYSIS

After seeing the timestamps of the program and different versions, results are:

1. Assembly implementations were consistently faster, but only by small margins (2-34ms).
2. The standard method slightly outperformed the Russian Peasant method in both C and Assembly.
3. A*B operations were generally slightly faster than B*A operations, but this is not always the case. More tests cases can be seen in the annexes section. Although there's no clear reason why this occur.

These results demonstrate that for these algorithms on modern hardware:

1. The performance gap between C and Assembly is minimal.
2. Modern C compilers produce highly optimized code, reducing the impact of hand-written Assembly optimizations.
3. The standard multiplication method slightly outperforms the Russian Peasant method.
4. The order of operands (A*B vs B*A) can affect performance, but not significantly.

A. How good is the Russian Peasant Method?

The Russian Peasant method was slightly slower than the standard multiplication method in both C and Assembly implementations.

The method didn't show any significant performance advantages over the standard multiplication algorithm for the tested range of numbers.

It performed consistently across different input values, suggesting stability in its execution time.

VIII. ANNEXES

This section provides different tests cases done during results recompilation.

The application performed less quickly when it ran on a Linux computer as opposed to a virtual machine. There are several possible reasons for this speed disparity, including variations in the distribution of resources, configurations of the system, and hardware optimizations at play. The virtual machine's environment was personalized for the particular application, which resulted in quicker execution speeds despite Linux's reputation for efficiency. This highlights the significance of customized configurations for obtaining optimal performance. It was necessary to bring up this distinction even if it isn't very pertinent to the project as a whole.

```

russianMul(A * B)
590 | 590 * 38 = 22420 / 31.208000 ms | 590 * 40 = 23600 / 31.658000 ms | 40 | 590 * 426 = 250100 / 31.849000 ms | 426 | 590 * 585 = 345150 / 30.950000 ms | 585 |
767 | 767 * 38 = 29146 / 30.790000 ms | 767 * 40 = 30680 / 30.285000 ms | 767 * 426 = 325288 / 30.692000 ms | 767 * 585 = 448695 / 30.442000 ms |
798 | 798 * 38 = 30324 / 30.584000 ms | 798 * 40 = 31320 / 30.952000 ms | 798 * 426 = 338352 / 30.596000 ms | 798 * 585 = 466830 / 30.587000 ms |
685 | 685 * 38 = 25940 / 30.523000 ms | 685 * 40 = 27400 / 30.253000 ms | 685 * 426 = 295520 / 30.693000 ms | 685 * 585 = 353925 / 30.350000 ms |
Execution time multiplying A * B using russian method in ASM: 499.498 ms

```

Fig. 19. AxB using Assembly Russian Peasant Method

```

russianMul(B * A)
590 | 38 * 590 = 22420 / 30.648000 ms | 38 * 767 = 29146 / 30.695000 ms | 38 * 798 = 30324 / 31.588000 ms | 38 * 685 = 25940 / 30.794000 ms |
40 | 40 * 590 = 23600 / 31.600000 ms | 40 * 767 = 30680 / 30.772000 ms | 40 * 798 = 31320 / 31.454000 ms | 40 * 685 = 27400 / 30.723000 ms |
426 | 426 * 590 = 250100 / 30.929000 ms | 426 * 767 = 325288 / 30.790000 ms | 426 * 798 = 338352 / 31.159000 ms | 426 * 685 = 295520 / 35.563000 ms |
585 | 585 * 590 = 345150 / 31.028000 ms | 585 * 767 = 448695 / 35.526000 ms | 585 * 798 = 466830 / 30.840000 ms | 585 * 685 = 353925 / 39.658000 ms |
Execution time multiplying B * A using russian method in ASM: 521.190 ms
Execution time taken from both tables: 2825.279 ms

```

Fig. 20. BxA using Assembly Russian Peasant Method

REFERENCES

- [1] "The Russian Peasant Method of Multiplication," Available: <https://www.jstor.org/stable/27949747?seq=2>. [Accessed: Jul. 26, 2024].
- [2] "Russian Peasant Multiplication: How and Why," Available: <https://www.themathdoctors.org/russian-peasant-multiplication-how-and-why/>. [Accessed: Jul. 26, 2024].

```

standardMul(A * B)
476 | 476 * 943 = 448868 / 30.524000 ms | 476 * 989 = 432684 / 31.543000 ms | 989 | 476 * 791 = 376516 / 30.454000 ms | 791 | 476 * 981 = 428876 / 30.150000 ms | 981 |
305 | 305 * 943 = 287615 / 30.413000 ms | 305 * 989 = 277245 / 30.380000 ms | 305 * 791 = 241255 / 37.681000 ms | 305 * 981 = 274805 / 31.422000 ms |
74 | 74 * 943 = 69782 / 30.378000 ms | 74 * 989 = 67266 / 30.843000 ms | 74 * 791 = 58534 / 30.466000 ms | 74 * 981 = 66674 / 30.154000 ms |
103 | 103 * 943 = 97129 / 30.596000 ms | 103 * 989 = 93627 / 30.150000 ms | 103 * 791 = 81473 / 30.450000 ms | 103 * 981 = 92803 / 30.760000 ms |
Execution time multiplying A * B using standard method in C: 582.548 ms

standardMul(B * A)
943 | 943 * 476 = 448868 / 31.301000 ms | 943 * 305 = 287615 / 31.573000 ms | 943 * 74 = 69782 / 30.473000 ms | 943 * 103 = 97129 / 30.321000 ms |
989 | 989 * 476 = 432684 / 30.528000 ms | 989 * 305 = 277245 / 30.320000 ms | 989 * 74 = 67266 / 30.431000 ms | 989 * 103 = 93627 / 31.722000 ms |
791 | 791 * 476 = 376516 / 30.383000 ms | 791 * 305 = 241255 / 30.631000 ms | 791 * 74 = 58534 / 30.473000 ms | 791 * 103 = 81473 / 30.327000 ms |
981 | 981 * 476 = 428876 / 30.728000 ms | 981 * 305 = 274805 / 30.458000 ms | 981 * 74 = 66674 / 30.472000 ms | 981 * 103 = 92803 / 30.530000 ms |
Execution time multiplying B * A using standard method in C: 496.945 ms

Execution time taken from both tables: 1993.254 ms

standardMul(A * B)
476 | 476 * 943 = 448868 / 31.205000 ms | 476 * 989 = 432684 / 32.040000 ms | 989 | 476 * 791 = 376516 / 30.454000 ms | 791 | 476 * 981 = 428876 / 30.150000 ms | 981 |
305 | 305 * 943 = 287615 / 30.548000 ms | 305 * 989 = 277245 / 30.451000 ms | 305 * 791 = 241255 / 30.466000 ms | 305 * 981 = 274805 / 30.392000 ms |
74 | 74 * 943 = 69782 / 30.388000 ms | 74 * 989 = 67266 / 30.848000 ms | 74 * 791 = 58534 / 30.466000 ms | 74 * 981 = 66674 / 30.210000 ms |
103 | 103 * 943 = 97129 / 30.397000 ms | 103 * 989 = 93627 / 30.629000 ms | 103 * 791 = 81473 / 30.472000 ms | 103 * 981 = 92803 / 31.220000 ms |
Execution time multiplying A * B using standard method in ASM: 582.457 ms

standardMul(B * A)
943 | 943 * 476 = 448868 / 32.385000 ms | 943 * 305 = 287615 / 31.174000 ms | 943 * 74 = 69782 / 30.766000 ms | 943 * 103 = 97129 / 30.431000 ms |
989 | 989 * 476 = 432684 / 30.588000 ms | 989 * 305 = 277245 / 31.260000 ms | 989 * 74 = 67266 / 30.431000 ms | 989 * 103 = 93627 / 30.807000 ms |
791 | 791 * 476 = 376516 / 31.108000 ms | 791 * 305 = 241255 / 31.805000 ms | 791 * 74 = 58534 / 30.473000 ms | 791 * 103 = 81473 / 30.877000 ms |
981 | 981 * 476 = 428876 / 30.872000 ms | 981 * 305 = 274805 / 30.394000 ms | 981 * 74 = 66674 / 31.680000 ms | 981 * 103 = 92803 / 30.232000 ms |
Execution time multiplying B * A using standard method in ASM: 582.459 ms

Execution time taken from both tables: 2007.003 ms

```

Fig. 22. C outperforms Assembly in Standard method

```

russianMul(A * B)
476 | 476 * 943 = 448868 / 31.176000 ms | 476 * 989 = 432684 / 31.262000 ms | 989 | 476 * 791 = 376516 / 31.068000 ms | 791 | 476 * 981 = 428876 / 31.310000 ms | 981 |
305 | 305 * 943 = 287615 / 31.308000 ms | 305 * 989 = 277245 / 31.260000 ms | 305 * 791 = 241255 / 31.208000 ms | 305 * 981 = 274805 / 30.807000 ms |
74 | 74 * 943 = 69782 / 31.568000 ms | 74 * 989 = 67266 / 31.159000 ms | 74 * 791 = 58534 / 31.149000 ms | 74 * 981 = 66674 / 30.970000 ms |
103 | 103 * 943 = 97129 / 31.154000 ms | 103 * 989 = 93627 / 31.170000 ms | 103 * 791 = 81473 / 31.054000 ms | 103 * 981 = 92803 / 31.710000 ms |
Execution time multiplying A * B using russian method in C: 585.473 ms

russianMul(B * A)
943 | 943 * 476 = 448868 / 31.307000 ms | 943 * 305 = 287615 / 30.496000 ms | 943 * 74 = 69782 / 30.757000 ms | 943 * 103 = 97129 / 31.624000 ms | 103 |
989 | 989 * 476 = 432684 / 31.488000 ms | 989 * 305 = 277245 / 31.181000 ms | 989 * 74 = 67266 / 30.478000 ms | 989 * 103 = 93627 / 30.925000 ms |
791 | 791 * 476 = 376516 / 31.108000 ms | 791 * 305 = 241255 / 31.805000 ms | 791 * 74 = 58534 / 30.473000 ms | 791 * 103 = 81473 / 30.877000 ms |
981 | 981 * 476 = 428876 / 30.728000 ms | 981 * 305 = 274805 / 31.215000 ms | 981 * 74 = 66674 / 30.894000 ms | 981 * 103 = 92803 / 31.720000 ms |
Execution time multiplying B * A using russian method in C: 517.391 ms

Execution time taken from both tables: 2823.319 ms

russianMul(A * B)
476 | 476 * 943 = 448868 / 30.436000 ms | 476 * 989 = 432684 / 30.443000 ms | 989 | 476 * 791 = 376516 / 30.375000 ms | 791 | 476 * 981 = 428876 / 37.021000 ms | 981 |
791 | 791 * 476 = 376516 / 30.383000 ms | 791 * 305 = 241255 / 30.853000 ms | 791 * 74 = 58534 / 30.473000 ms | 791 * 103 = 81473 / 30.877000 ms |
103 | 103 * 943 = 97129 / 30.638000 ms | 103 * 989 = 93627 / 30.744000 ms | 103 * 791 = 81473 / 30.472000 ms | 103 * 981 = 92803 / 30.695000 ms |
Execution time multiplying A * B using russian method in ASM: 519.716 ms

russianMul(B * A)
943 | 943 * 476 = 448868 / 30.753000 ms | 943 * 305 = 287615 / 31.713000 ms | 943 * 74 = 69782 / 30.735000 ms | 943 * 103 = 97129 / 30.571000 ms | 103 |
989 | 989 * 476 = 432684 / 30.478000 ms | 989 * 305 = 277245 / 30.854000 ms | 989 * 74 = 67266 / 30.729000 ms | 989 * 103 = 93627 / 30.790000 ms |
791 | 791 * 476 = 376516 / 30.413000 ms | 791 * 305 = 241255 / 30.853000 ms | 791 * 74 = 58534 / 30.473000 ms | 791 * 103 = 81473 / 30.877000 ms |
981 | 981 * 476 = 428876 / 30.728000 ms | 981 * 305 = 274805 / 31.424000 ms | 981 * 74 = 66674 / 30.954000 ms | 981 * 103 = 92803 / 30.941000 ms |
Execution time multiplying B * A using russian method in ASM: 483.700 ms

Execution time taken from both tables: 2810.239 ms

```

Fig. 23. Inverse performance in Russian Peasant Method

```

N: 50000
nullMul(A * B)
476 | 476 * 943 = 0 / 32.048000 ms | 476 * 989 = 0 / 37.527000 ms | 476 * 791 = 0 / 35.943000 ms | 476 * 981 = 0 / 33.142000 ms |
305 | 305 * 943 = 0 / 30.274000 ms | 305 * 989 = 0 / 30.268000 ms | 305 * 791 = 0 / 30.268000 ms | 305 * 981 = 0 / 30.145000 ms |
74 | 74 * 943 = 0 / 30.271000 ms | 74 * 989 = 0 / 31.320000 ms | 74 * 791 = 0 / 31.425000 ms | 74 * 981 = 0 / 30.345000 ms |
103 | 103 * 943 = 0 / 30.335000 ms | 103 * 989 = 0 / 30.281000 ms | 103 * 791 = 0 / 30.401000 ms | 103 * 981 = 0 / 33.551000 ms |
Execution time multiplying A * B returning 0 in C: 587.171 ms

nullMul(B * A)
943 | 943 * 476 = 0 / 32.048000 ms | 943 * 305 = 0 / 30.444000 ms | 943 * 74 = 0 / 30.376000 ms | 943 * 103 = 0 / 30.217000 ms | 103 |
989 | 989 * 476 = 0 / 30.355000 ms | 989 * 305 = 0 / 30.404000 ms | 989 * 74 = 0 / 30.398000 ms | 989 * 103 = 0 / 30.181000 ms |
791 | 791 * 476 = 0 / 30.462000 ms | 791 * 305 = 0 / 31.244000 ms | 791 * 74 = 0 / 30.428000 ms | 791 * 103 = 0 / 30.250000 ms |
981 | 981 * 476 = 0 / 30.440000 ms | 981 * 305 = 0 / 30.260000 ms | 981 * 74 = 0 / 30.468000 ms | 981 * 103 = 0 / 30.520000 ms |
Execution time multiplying B * A returning 0 in C: 488.087 ms

Execution time taken from both tables: 1989.586 ms

nullMul(A * B)
476 | 476 * 943 = 0 / 33.239000 ms | 476 * 989 = 0 / 30.558000 ms | 476 * 791 = 0 / 30.532000 ms | 476 * 981 = 0 / 30.321000 ms |
305 | 305 * 943 = 0 / 32.667000 ms | 305 * 989 = 0 / 30.670000 ms | 305 * 791 = 0 / 31.544000 ms | 305 * 981 = 0 / 30.561000 ms |
74 | 74 * 943 = 0 / 30.620000 ms | 74 * 989 = 0 / 31.520000 ms | 74 * 791 = 0 / 30.654000 ms | 74 * 981 = 0 / 30.712000 ms |
103 | 103 * 943 = 0 / 35.362000 ms | 103 * 989 = 0 / 30.434000 ms | 103 * 791 = 0 / 30.560000 ms | 103 * 981 = 0 / 30.348000 ms |
Execution time multiplying A * B returning 0 in ASM: 511.159 ms

nullMul(B * A)
943 | 943 * 476 = 0 / 31.291000 ms | 943 * 305 = 0 / 30.322000 ms | 943 * 74 = 0 / 30.318000 ms | 943 * 103 = 0 / 30.207000 ms | 103 |
989 | 989 * 476 = 0 / 30.355000 ms | 989 * 305 = 0 / 30.221000 ms | 989 * 74 = 0 / 30.268000 ms | 989 * 103 = 0 / 30.204000 ms |
791 | 791 * 476 = 0 / 30.286000 ms | 791 * 305 = 0 / 31.281000 ms | 791 * 74 = 0 / 30.478000 ms | 791 * 103 = 0 / 30.270000 ms |
981 | 981 * 476 = 0 / 30.450000 ms | 981 * 305 = 0 / 30.430000 ms | 981 * 74 = 0 / 30.413000 ms | 981 * 103 = 0 / 30.210000 ms |
Execution time multiplying B * A returning 0 in ASM: 490.980 ms

Execution time taken from both tables: 2004.639 ms

```

Fig. 21. C outperforms Assembly in Null method

```

@echo off && %workspace%\nullis\Project0 %* % /s /o /t
N: 100000
nullMul(A * B)
196 | 196 * 22 = 0 / 60.050000 ms | 196 * 799 = 0 / 60.412000 ms | 196 * 282 = 0 / 56.996000 ms | 196 * 445 = 0 / 65.214000 ms |
372 | 372 * 22 = 0 / 60.248000 ms | 372 * 799 = 0 / 60.422000 ms | 372 * 282 = 0 / 60.360000 ms | 372 * 445 = 0 / 64.396000 ms |
19 | 19 * 22 = 0 / 65.523000 ms | 19 * 799 = 0 / 60.390000 ms | 19 * 282 = 0 / 60.420000 ms | 19 * 445 = 0 / 60.611000 ms |
181 | 181 * 22 = 0 / 60.163000 ms | 181 * 799 = 0 / 60.980000 ms | 181 * 282 = 0 / 72.462000 ms | 181 * 445 = 0 / 60.613000 ms |
Execution time multiplying A * B returning 0 in C: 1000.763 ms

nullMul(B * A)
22 | 22 * 196 = 0 / 61.107000 ms | 22 * 372 = 0 / 60.108000 ms | 22 * 19 = 0 / 62.670000 ms | 22 * 181 = 0 / 74.840000 ms | 181 |
799 | 799 * 196 = 0 / 60.402000 ms | 799 * 372 = 0 / 60.380000 ms | 799 * 19 = 0 / 63.270000 ms | 799 * 181 = 0 / 59.920000 ms |
282 | 282 * 196 = 0 / 65.118000 ms | 282 * 372 = 0 / 60.170000 ms | 282 * 19 = 0 / 59.880000 ms | 282 * 181 = 0 / 59.950000 ms |
445 | 445 * 196 = 0 / 60.357000 ms | 445 * 372 = 0 / 60.177000 ms | 445 * 19 = 0 / 72.346000 ms | 445 * 181 = 0 / 67.167000 ms |
Execution time multiplying B * A returning 0 in C: 1008.011 ms

Execution time taken from both tables: 4013.956 ms

nullMul(A * B)
196 | 196 * 22 = 0 / 70.320000 ms | 196 * 799 = 0 / 60.850000 ms | 196 * 282 = 0 / 60.411000 ms | 196 * 445 = 0 / 60.480000 ms |
372 | 372 * 22 = 0 / 60.653000 ms | 372 * 799 = 0 / 61.967000 ms | 372 * 282 = 0 / 60.686000 ms | 372 * 445 = 0 / 60.582000 ms |
19 | 19 * 22 = 0 / 65.330000 ms | 19 * 799 = 0 / 60.551000 ms | 19 * 282 = 0 / 60.442000 ms | 19 * 445 = 0 / 60.643000 ms |
181 | 181 * 22 = 0 / 60.704000 ms | 181 * 799 = 0 / 60.830000 ms | 181 * 282 = 0 / 65.126000 ms | 181 * 445 = 0 / 63.470000 ms |
Execution time multiplying A * B returning 0 in ASM: 991.281 ms

nullMul(B * A)
22 | 22 * 196 = 0 / 63.273000 ms | 22 * 372 = 0 / 60.523000 ms | 22 * 19 = 0 / 67.490000 ms | 22 * 181 = 0 / 60.662000 ms | 181 |
799 | 799 * 196 = 0 / 61.105000 ms | 799 * 372 = 0 / 60.390000 ms | 799 * 19 = 0 / 70.250000 ms | 799 * 181 = 0 / 63.364000 ms |
282 | 282 * 196 = 0 / 64.555000 ms | 282 * 372 = 0 / 60.300000 ms | 282 * 19 = 0 / 59.580000 ms | 282 * 181 = 0 / 59.920000 ms |
445 | 445 * 196 = 0 / 60.650000 ms | 445 * 372 = 0 / 61.000000 ms | 445 * 19 = 0 / 63.254000 ms | 445 * 181 = 0 / 60.404000 ms |
Execution time multiplying B * A returning 0 in ASM: 997.506 ms

Execution time taken from both tables: 3975.469 ms

```

Fig. 24. Better performance in Assembly when N=100000, using null method

standardMul(A * B)					
196	196 * 22 = 4312 / 61.042000 ms	196 * 799 = 156604 / 62.226000 ms	196 * 282 = 55272 / 60.378000 ms	196 * 445 = 87220 / 72.850000 ms	
372	372 * 22 = 8184 / 60.154000 ms	372 * 799 = 297228 / 60.518000 ms	372 * 282 = 104904 / 60.250000 ms	372 * 445 = 165540 / 61.230000 ms	
19	19 * 22 = 418 / 61.848000 ms	19 * 799 = 15181 / 60.690000 ms	19 * 282 = 5358 / 64.056000 ms	19 * 445 = 8455 / 68.476000 ms	
181	181 * 22 = 2222 / 60.527000 ms	181 * 799 = 80699 / 60.730000 ms	181 * 282 = 28482 / 60.552000 ms	181 * 445 = 44945 / 60.494000 ms	
Execution time multiplying A * B using standard method in C: 991.095 ms					
standardMul(B * A)					
22	22 * 196 = 4312 / 62.080000 ms	22 * 372 = 8184 / 60.840000 ms	22 * 19 = 418 / 61.310000 ms	22 * 181 = 2222 / 60.690000 ms	
799	799 * 196 = 156604 / 60.118000 ms	799 * 372 = 297228 / 61.558000 ms	799 * 19 = 15181 / 60.360000 ms	799 * 181 = 80699 / 60.681000 ms	
282	282 * 196 = 55272 / 67.702000 ms	282 * 372 = 104904 / 60.670000 ms	282 * 19 = 5358 / 67.924000 ms	282 * 181 = 28482 / 74.201000 ms	
445	445 * 196 = 87220 / 60.378000 ms	445 * 372 = 165540 / 60.551000 ms	445 * 19 = 8455 / 60.487000 ms	445 * 181 = 44945 / 60.196000 ms	
Execution time multiplying B * A using standard method in C: 1001.472 ms					
Execution time taken from both tables: 3073.321 ms					
standardMulASM(A * B)					
196	196 * 22 = 4312 / 61.000000 ms	196 * 799 = 156604 / 60.070000 ms	196 * 282 = 55272 / 59.980000 ms	196 * 445 = 87220 / 59.902000 ms	
372	372 * 22 = 8184 / 60.080000 ms	372 * 799 = 297228 / 60.131000 ms	372 * 282 = 104904 / 60.018000 ms	372 * 445 = 165540 / 59.881000 ms	
19	19 * 22 = 418 / 60.770000 ms	19 * 799 = 15181 / 60.260000 ms	19 * 282 = 5358 / 60.120000 ms	19 * 445 = 8455 / 64.407000 ms	
181	181 * 22 = 2222 / 67.357000 ms	181 * 799 = 80699 / 61.990000 ms	181 * 282 = 28482 / 60.433000 ms	181 * 445 = 44945 / 59.894000 ms	
Execution time multiplying A * B using standard method in ASM: 977.830 ms					
standardMulASM(B * A)					
22	22 * 196 = 4312 / 62.782000 ms	22 * 372 = 8184 / 72.811000 ms	22 * 19 = 418 / 64.493000 ms	22 * 181 = 2222 / 60.133000 ms	
799	799 * 196 = 156604 / 60.211000 ms	799 * 372 = 297228 / 60.285000 ms	799 * 19 = 15181 / 60.362000 ms	799 * 181 = 80699 / 59.950000 ms	
282	282 * 196 = 55272 / 62.800000 ms	282 * 372 = 104904 / 60.470000 ms	282 * 19 = 5358 / 60.140000 ms	282 * 181 = 28482 / 64.620000 ms	
445	445 * 196 = 87220 / 60.378000 ms	445 * 372 = 165540 / 60.273000 ms	445 * 19 = 8455 / 60.203000 ms	445 * 181 = 44945 / 60.027000 ms	
Execution time multiplying B * A using standard method in ASM: 988.852 ms					
Execution time taken from both tables: 3029.933 ms					

Fig. 25. Better performance in Assembly when N=100000, using standard method

russianMul(A * B)					
196	196 * 22 = 4312 / 61.510000 ms	22]	799]	282]	445]
372	372 * 22 = 8184 / 61.752000 ms	372]	799]	282]	445]
19	19 * 22 = 418 / 61.945000 ms	19]	799]	282]	445]
181	181 * 22 = 2222 / 62.820000 ms	181]	799]	282]	445]
Execution time multiplying A * B using russian method in C: 1013.135 ms					
russianMul(B * A)					
22	22 * 196 = 4312 / 61.320000 ms	22]	372]	19]	181]
799	799 * 196 = 156604 / 61.077000 ms	799]	372]	19]	181]
282	282 * 196 = 55272 / 61.547000 ms	282]	372]	19]	181]
445	445 * 196 = 87220 / 63.125000 ms	445]	372]	19]	181]
Execution time multiplying B * A using russian method in C: 1014.518 ms					
Execution time taken from both tables: 4015.973 ms					
russianMulASM(A * B)					
196	196 * 22 = 4312 / 60.833000 ms	22]	799]	282]	445]
372	372 * 22 = 8184 / 60.473000 ms	372]	799]	282]	445]
19	19 * 22 = 418 / 59.990000 ms	19]	799]	282]	445]
181	181 * 22 = 2222 / 65.138000 ms	181]	799]	282]	445]
Execution time multiplying A * B using russian method in ASM: 1010.792 ms					
russianMulASM(B * A)					
22	22 * 196 = 4312 / 60.986000 ms	22]	372]	19]	181]
799	799 * 196 = 156604 / 60.080000 ms	799]	372]	19]	181]
282	282 * 196 = 55272 / 61.903000 ms	282]	372]	19]	181]
445	445 * 196 = 87220 / 71.133000 ms	445]	372]	19]	181]
Execution time multiplying B * A using russian method in ASM: 998.171 ms					
Execution time taken from both tables: 3089.295 ms					

Fig. 26. Minimal difference using Russian Method, when N=100000

[root@localhost ~]# ./workspaces/analysis/Project (mln) \$./a.out					
N: 50000					
nullMul(A * B)					
	5367]	3586]	2942]	5299]	
2183	2183 * 5367 = 0 / 30.23 ms	2183 * 3586 = 0 / 30.45 ms	2183 * 2942 = 0 / 30.31 ms	2183 * 5299 = 0 / 30.06 ms	
1915	1915 * 5367 = 0 / 30.40 ms	1915 * 3586 = 0 / 30.30 ms	1915 * 2942 = 0 / 30.40 ms	1915 * 5299 = 0 / 30.61 ms	
3575	3575 * 5367 = 0 / 35.11 ms	3575 * 3586 = 0 / 29.52 ms	3575 * 2942 = 0 / 29.84 ms	3575 * 5299 = 0 / 29.58 ms	
6288	6288 * 5367 = 0 / 29.62 ms	6288 * 3586 = 0 / 29.60 ms	6288 * 2942 = 0 / 30.22 ms	6288 * 5299 = 0 / 30.96 ms	
Execution time multiplying A * B returning 0 in C: 496.646 ms					
nullMul(B * A)					
	2183]	1915]	3575]	6288]	
5367	5367 * 2183 = 0 / 34.63 ms	5367 * 1915 = 0 / 30.82 ms	5367 * 3575 = 0 / 29.48 ms	5367 * 6288 = 0 / 31.97 ms	
3586	3586 * 2183 = 0 / 31.36 ms	3586 * 1915 = 0 / 29.64 ms	3586 * 3575 = 0 / 32.15 ms	3586 * 6288 = 0 / 32.30 ms	
2942	2942 * 2183 = 0 / 29.60 ms	2942 * 1915 = 0 / 29.50 ms	2942 * 3575 = 0 / 29.67 ms	2942 * 6288 = 0 / 42.43 ms	
5299	5299 * 2183 = 0 / 33.25 ms	5299 * 1915 = 0 / 34.48 ms	5299 * 3575 = 0 / 30.52 ms	5299 * 6288 = 0 / 29.55 ms	
Execution time multiplying B * A returning 0 in C: 500.791 ms					
Execution time taken from both tables: 1093.448 ms					
nullMulASM(A * B)					
	5367]	3586]	2942]	5299]	
2183	2183 * 5367 = 0 / 30.66 ms	2183 * 3586 = 0 / 30.01 ms	2183 * 2942 = 0 / 29.77 ms	2183 * 5299 = 0 / 29.64 ms	
1915	1915 * 5367 = 0 / 29.77 ms	1915 * 3586 = 0 / 30.78 ms	1915 * 2942 = 0 / 29.37 ms	1915 * 5299 = 0 / 30.62 ms	
3575	3575 * 5367 = 0 / 29.69 ms	3575 * 3586 = 0 / 29.69 ms	3575 * 2942 = 0 / 30.85 ms	3575 * 5299 = 0 / 30.20 ms	
6288	6288 * 5367 = 0 / 29.06 ms	6288 * 3586 = 0 / 29.75 ms	6288 * 2942 = 0 / 29.04 ms	6288 * 5299 = 0 / 30.37 ms	
Execution time multiplying A * B returning 0 in ASM: 479.445 ms					
nullMulASM(B * A)					
	2183]	3575]	3575]	6288]	
5367	5367 * 2183 = 0 / 31.13 ms	5367 * 1915 = 0 / 29.52 ms	5367 * 3575 = 0 / 29.69 ms	5367 * 6288 = 0 / 29.51 ms	
3586	3586 * 2183 = 0 / 29.62 ms	3586 * 1915 = 0 / 31.94 ms	3586 * 3575 = 0 / 29.75 ms	3586 * 6288 = 0 / 29.57 ms	
2942	2942 * 2183 = 0 / 29.60 ms	2942 * 1915 = 0 / 29.51 ms	2942 * 3575 = 0 / 30.46 ms	2942 * 6288 = 0 / 31.45 ms	
5299	5299 * 2183 = 0 / 29.63 ms	5299 * 1915 = 0 / 29.52 ms	5299 * 3575 = 0 / 29.59 ms	5299 * 6288 = 0 / 31.55 ms	
Execution time multiplying B * A returning 0 in ASM: 492.517 ms					
Execution time taken from both tables: 1044.278 ms					

Fig. 27. Assembly outperforms C in null method

standardMul(A * B)						5367]	3586]	2942]	5299]
2183	2183 * 5367 = 11716151 / 30.77 ms	2183 * 3586 = 7828238 / 30.06 ms	2183 * 2942 = 6422386 / 30.11 ms	2183 * 5299 = 11567717 / 29.99 ms					
1915	1915 * 5367 = 10277805 / 30.18 ms	1915 * 3586 = 6867100 / 30.16 ms	1915 * 2942 = 5033930 / 30.47 ms	1915 * 5299 = 10147585 / 30.05 ms					
3575	3575 * 5367 = 21817025 / 32.32 ms	3575 * 3586 = 22819990 / 30.49 ms	3575 * 2942 = 10517650 / 30.22 ms	3575 * 5299 = 12045925 / 30.23 ms					
6288	6288 * 5367 = 33747696 / 30.21 ms	6288 * 3586 = 22548768 / 30.11 ms	6288 * 2942 = 18499296 / 30.27 ms	6288 * 5299 = 33320112 / 30.32 ms					
Execution time multiplying A * B using standard method in C: 485.563 ms									
standardMul(B * A)						2183]	3575]	3575]	6288]
5367	5367 * 2183 = 11716151 / 34.03 ms	5367 * 1915 = 10277805 / 30.42 ms	5367 * 3575 = 19187025 / 30.38 ms	5367 * 6288 = 33747696 / 30.22 ms					
3586	3586 * 2183 = 7828238 / 30.12 ms	3586 * 1915 = 6867100 / 30.31 ms	3586 * 3575 = 22819990 / 30.34 ms	3586 * 6288 = 22548768 / 30.10 ms					
2942	2942 * 2183 = 6422386 / 30.28 ms	2942 * 1915 = 5033930 / 30.17 ms	2942 * 3575 = 10517650 / 35.24 ms	2942 * 6288 = 18499296 / 30.22 ms					
5299	5299 * 2183 = 11567717 / 30.19 ms	5299 * 1915 = 10147585 / 30.15 ms	5299 * 3575 = 18943925 / 30.50 ms	5299 * 6288 = 33320112 / 30.20 ms					
Execution time multiplying B * A using standard method in C: 493.177 ms									
Execution time taken from both tables: 1951.807 ms									
standardMulASM(A * B)						5367]	3586]	2942]	5299]
2183	2183 * 5367 = 11716151 / 30.93 ms	2183 * 3586 = 7828238 / 34.05 ms	2183 * 2942 = 6422386 / 30.77 ms	2183 * 5299 = 11567717 / 30.24 ms					
1915	1915 * 5367 = 10277805 / 30.11 ms	1915 * 3586 = 6867100 / 29.94 ms	1915 * 2942 = 5033930 / 30.27 ms	1915 * 5299 = 10147585 / 29.93 ms					
3575	3575 * 5367 = 21817025 / 30.07 ms	3575 * 3586 = 22819990 / 29.96 ms	3575 * 2942 = 10517650 / 30.41 ms	3575 * 5299 = 12045925 / 30.07 ms					
6288	6288 * 5367 = 33747696 / 30.90 ms	6288 * 3586 = 22548768 / 29.96 ms	6288 * 2942 = 18499296 / 31.40 ms	6288 * 5299 = 33320112 / 30.04 ms					
Execution time multiplying A * B using standard method in ASM: 494.528 ms									
standardMulASM(B * A)						2183]	3575]	3575]	6288]
5367	5367 * 2183 = 11716151 / 32.62 ms	5367 * 1915 = 10277805 / 37.48 ms	5367 * 3575 = 19187025 / 32.48 ms	5367 * 6288 = 33747696 / 30.02 ms					
3586	3586 * 2183 = 7828238 / 30.00 ms	3586 * 1915 = 6867100 / 30.13 ms	3586 * 3575 = 22819990 / 29.98 ms	3586 * 6288 = 22548768 / 30.35 ms					
2942	2942 * 2183 = 6422386 / 30.21 ms	2942 * 1915 = 5033930 / 29.97 ms	2942 * 3575 = 10517650 / 30.41 ms	2942 * 6288 = 18499296 / 34.65 ms					
5299	5299 * 2183 = 11567717 / 30.13 ms	5299 * 1915 = 10147585 / 29.96 ms	5299 * 3575 = 18943925 / 30.16 ms	5299 * 6288 = 33320112 / 30.31 ms					
Execution time multiplying B * A using standard method in ASM: 499.365 ms									
Execution time taken from both tables: 1987.886 ms									

Fig. 28. C outperforms Assembly in standard method