Grenze International Journal of Engineering and Technology, June Issue

Karatsuba Algorithm: Parallelization using CUDA

Rakshith Kumar D[1], Amit Kumar[2], N Gopalakrishna Kini[3] and Ashwath Rao B[4]
[1-4]Department of Computer Science and Engineering, Manipal Institute of Technology
Manipal Academy of Higher Education, Manipal-576 104, Karnataka, India
Email: ng.kini@manipal.edu, {rakshithkademane, akakumar635, ashwath.rao.b}@gmail.com

***Abstract*— In arbitrary arithmetic computation and computational science, multiplying large integers is a widely used operation. Numerous cryptographic techniques involve operations on extremely large subsets of the integer numbers, including the Diffie-Hellman key exchange, RSA, ECC, and others. These techniques employ safe message encryption, decryption, and key exchange using security keys with a size of at least 1024 bits. Exponentiation and multiplication are necessary in order to carry out encryption, decryption, and key exchange. The Karatsuba algorithm is a fast and efficient method for multiplying large numbers, that reduces the number of multiplications from four to three at each recursive step. In this paper, we conduct a comprehensive assessment of the Karatsuba algorithm's performance when applied to both sequential and parallel contexts. We utilize the power of Nvidia Graphics Processing Unit (GPU) with Compute Unified Device Architecture (CUDA) programming to gauge the speedup of the parallel implementation and processor configurations. The speedup achieved by the Karatsuba algorithm running on Nvidia GPU CUDA platform over the sequential is 30.12. There exists an improvement in performance by making use of available GPU cores. The findings emphasize the potential advantages of parallelization in reducing the overall computation time.**

***Index Terms*— Karatsuba, Multiplication, Compute Unified Device Architecture, Nvidia Graphics Processing Unit, Speedup**

I. INTRODUCTION

Multiplication is one of the most fundamental and frequently used operations in mathematics and computer science. However, multiplying large numbers can be quite time-consuming and computationally intensive, especially when using the traditional multiplication algorithms that require n^2 time for single-digit multiplications of two n-digit numbers. Therefore, finding faster and more efficient methods for multiplying large numbers is an important and challenging problem. A major advancement with multiplication techniques in identifying quicker multiplication of huge integers is with Karatsuba algorithm.

In various cryptographic schemes such as Elliptic Curve Cryptography, RSA, and Pairing-Based Cryptography, a lengthy integer multiplication takes the longest to execute algorithms. Because of this, creating quick long integer multiplication algorithms becomes the focus of study in order to provide a higher level of security. This is where Karatsuba algorithm plays its role of multiplying larger numbers. Karatsuba algorithm can significantly reduce the time complexity of multiplication process.

CUDA is a parallel computing platform developed by Nvidia that enables software developers to harness the parallel computational power of Nvidia GPUs for general-purpose computing. CUDA introduces a distinct memory hierarchy and a large number of CUDA cores, which are the fundamental building blocks for parallel computing on Nvidia GPUs. The cryptographic applications can be expedited by using Nvidia GPU CUDA

Grenze ID: 01.GIJET.10.2.735

platform to implement and carry out large integer multiplication operations in parallel. The work in using Nvidia GPU CUDA platform, particularly for lengthy integer multiplication, has been the main focus of this paper.
On developing the Karatsuba algorithm on Nvidia GPU CUDA platform, optimizes multiplication by substituting certain multiplications with less computationally expensive addition and subtraction operations. The Karatsuba algorithm makes use of divide-and-conquer strategy that splits the two numbers into two halves each, and then recursively multiplies the halves using only three multiplications instead of four at each step. This reduces the number of multiplications from n^2 to n^{log3}, which approximates to about $n^{1.585}$. Karatsuba algorithm is more effective for large numbers in comparison to smaller numbers.
The main contribution of this paper is:

- To present how a parallel approach is investigated to perform multiplication with Karatsuba algorithm.
- To show how Karatsuba algorithm implementation turns out very efficient on graphics processing units due to inherent parallelism. This parallel implementation exhibits better timing performance as compared to serial implementation.
- To analyze the speedup performance of parallel algorithms with classical implementation on single CPU-core.

II. Related Work

In [1], the author explores two key advancements in cryptography. The first pertains to the demand for innovative cryptographic systems that reduce reliance on secure key distribution channels and offer a form of digital signature. The second addresses how evolving theories in communication and computation are enabling the resolution of longstanding cryptographic challenges. The paper proposes solutions to these on-going issues and delves into the transformative impact of recent advancements in communication and computation on the field of cryptograph. Author also addresses the complexity involved with multiplication of prime numbers in computing key information in cryptosystem.
The paper [2] discusses the techniques for performing multiplication of numbers with multiple digits using automata. Algorithms using homogeneous polynomials for extensive integer multiplication, utilizing homogeneity for simplification is presented in [3]. While a two-stage implementation and Tom-Cook algorithms double standard multiplications compared to a direct approach, the multistage implementation proves less efficient. However, the Generalized Horner rule introduced in the paper reduces the required algebraic additions by half.
In [4], the authors find the function's minimum computation time, asserting that multiplying decimal integers is inherently more challenging than adding them. In the first part introduces a precise conjecture while the second part proves a weakened form of the very same conjecture. Various other factors such as increased circuitry for fixed-point product computation and longer time for multiplication compared to addition are explored. Insights from longhand multiplication motivate the theory. The authors highlight that storage structure uniformity is sufficient for the arguments in provided in the second part.
Evaluating the effectiveness and scalability of algorithms for long integer multiplication is addressed in [5]. They investigate the implementation of these algorithms using parallel computing frameworks such as OpenMP and MPI. The results indicate improved speedup performance as the bit-length increases, attributed to the increased availability of CPU cores in the system.
In [6], the authors concentrate on enhancing the efficiency of cryptographic operations, specifically the critical task of long integer multiplication utilized in algorithms such as RSA and Diffie-Hellman. Their aim is to boost the speed of these operations by utilizing the parallel processing capabilities of multi-core CPUs and many-core GPUs. Drawing from prior research in both cryptography and parallel computing, the authors acknowledge the inherent computational complexities in long integer multi-plication and the challenges of effectively exploiting parallelism with conventional algorithms. To address these challenges, they propose the adoption of the Fast Fourier Transform (FFT) algorithm, well-known for its efficacy in handling large-scale multiplications. The authors delve into the practical implementation of parallel long integer multiplication on modern parallel processors, including GPUs, utilizing CUDA as a programming environment to harness the computational prowess of GPUs for accelerating cryptographic processes.
The implementation of a highly optimized systolic Montgomery multiplication algorithm using Nvidia's CUDA programming model for GPUs is discussed in [7]. The algorithm aims to improve the performance of modular multiplications, particularly in cryptographic applications. The speeds of modular multiplications for different bit numbers are provided, showcasing the performance capabilities of the GPUs in this context.

III. METHODOLOGY

To multiply two n-digit numbers, long multiplication method is employed, which involves $O(n^2)$ multiplication operations. To enhance efficiency the Karatsuba algorithm, which employs a divide-and-conquer approach is used. The Karatsuba algorithm achieves the product with only three multiplications. This algorithm is adopted by companies like Intel for faster multiplication due to its reduced number of steps. Moreover, the Karatsuba algorithm can be implemented either recursively or iteratively.

Let us consider two *n*-bit numbers X and Y. Partition both of them into *n*/2-bit numbers such that they contain higher bits and lower bits each. X_H contains the upper *n*/2 bits of X and X_L contains the lower *n*/2 bits of X whereas Y_H contains the upper *n*/2 bits of Y and Y_L contains the lower *n*/2 bits of Y.

The algorithmic steps of Karatsuba algorithm are as in Figure 1.

Algorithm: The algorithmic steps of Karatsuba

Let $X = x_3x_2x_1x_0$ and $Y = y_3y_2y_1y_0$ be two n-digit numbers.

Partition X and Y into two n/2-digit numbers such that $X_H = x_3x_2$ and $X_L = x_1x_0$, $Y_H = y_3y_2$ and $Y_L = y_1y_0$. Now compute the following:

Step 1: The product of X_H and Y_H gives A i.e. $A = X_H * Y_H$

Step 2: The product of X_L and Y_L gives B. i.e. $B = X_L * Y_L$

Step 3: The product of $(X_H + X_L)$ and $(Y_H + Y_L)$ gives C. i.e. $C = (X_H + X_L) * (Y_H + Y_L)$

Step 4: Subtract both A and B from C to obtain D. i.e. $D = C - A - B$

Step 5: The final product value is given by the formula: Product Value $= A * (b^n) + D * (b^{n/2}) + B$

Figure 1. Karatsuba Algorithm

Here '*n*' refers to the number of digits in the number and '*b*' refers to the base of the number. The *product value* is the result of multiplication.

Karatsuba algorithm can be employed recursively by breaking down both the numbers, X and Y, into two parts, especially for the computation of larger-digit multiplication operations. Due to the higher computational cost of multiplication compared to addition and subtraction incurred, the Karatsuba algorithm aims to minimize the number of required multiplications.

The classical multiplication algorithm needs to perform 16 single-digit multiplications to multiply two 8-digit numbers, along with additional add and shift steps. On the other hand, the Karatsuba algorithm can multiply the same two 8-digit numbers using only 3 multiplications of smaller numbers, plus some add, subtract, and shift steps. So Karatsuba's method reduces the number of multiplication operations needed substantially compared to the classical approach. This reduction is achieved by breaking the big multiplication down recursively into smaller multiplications plus some clever rearrangement and recombination of the intermediate terms. Overall, Karatsuba algorithm decreases the computational load for large multiplications by reducing the number of digit-by-digit multiplications required. For instance, a sample example for the calculation of 1234 multiplied by 5678 using the Karatsuba algorithm is illustrated in Figure 2.

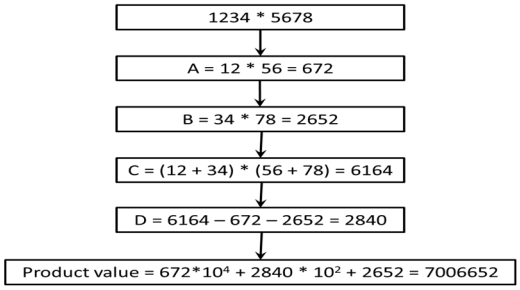

Figure 2. A sample example flowchart of Karatsuba Implementation

To provide the details on the Dataset used, the dataset comprising of two distinct files, denoted as *file*1 and *file*2 are used, where each file contains a collection of 100000 *n*-digit large numbers. The dataset is collected by running the program of random number generator. The dataset is intended to evaluate algorithm performance, where each entry of *file*1 corresponds to multiplicand and each entry of *file*2 corresponds to multiplier. When the algorithm is applied to the dataset, corresponding elements of *file*1 and *file*2 are multiplied and stored in another file *file*3 which is called the product file.

Both sequential and parallel Karatsuba implementation is carried out in Windows 10 operating system of version 22H2, hard disk capacity of 500GB and Nvidia GPU GeForce GTX 1050Ti. The number of CUDA cores we are using is 768, the VRAM size is 4GB.

A. Sequential Karatsuba Implementation

Two files of dataset are considered each file containing a large number of multiplicands and corresponding multipliers. A multiplicand is multiplied by the corresponding multiplier sequentially to meet the multiplication process as defined by the Karatsuba algorithm to give out the product. This product is stored in another file.
The time complexity for normal multiplication of two *n*-bit numbers sequentially is $O(n^2)$. In sequential Karatsuba algorithm, only one process is performing all the computations of the algorithm iteratively. The time complexity for multiplication of two *n*-bit numbers using Karatsuba algorithm is $O(n^{log3})$ which is approximately equal to $O(n^{1.585})$.

B. Karatsuba Implementation on CUDA platform

The same dataset generated for sequential implementation of Karatsuba algorithm containing a large number of multiplicands and corresponding multipliers are considered for Karatsuba Implementation on CUDA platform parallelization. A multiplicand is multiplied by the corresponding multiplier as defined by the Karatsuba algorithm to give out the product in another file. Here the whole event of generating the large number products of multiplicands with their corresponding multipliers are totally parallelized to exhibit inherent parallelism.
Parallelism is achieved by dividing data into threads, each running on a separate CUDA core. CUDA threads are used to access X_H, X_L, Y_H and Y_L components of two *n*-bit numbers X and Y. Each thread independently performs their part of the required arithmetic operations in parallel. Finally, the product is computed using these calculated values. The number of threads per block and the number of blocks per grid are determined by the size of the input dataset and the kernel is executed accordingly.
The steps involved in CUDA in achieving the parallelism of Karatsuba algorithm is:

- Transfer the X_H, X_L, Y_H and Y_L components of two *n*-bit numbers X and Y from CPU to GPU memory.
- Launch the CUDA kernel to perform respective part of the required arithmetic operations based on Karatsuba algorithm.
- The final product computed is transferred back to the CPU memory from GPU memory.
- The time required to perform the operation by Karatsuba algorithm is measured.

IV. RESULTS

The datasets were randomly generated in two files as shown in the code snippet in Figure 3 and Figure 4 respectively and further they are imported for Karatsuba implementation on CUDA platform.

```
FILE *file = fopen("X_100000.txt", "w");
for (int i = 0; i < 100000; i++) {
    int random_number = rand() % 90000000 + 10000000;
    fprintf(file, "%d\n", random_number);
}
fclose(file);
```

Figure 3. Random generator for *file* 1

```
FILE *file = fopen("Y_100000.txt", "w");
for (int i = 0; i < 100000; i++) {
    int random_number = rand() % 90000000 + 10000000;
    fprintf(file, "%d\n", random_number);
}
fclose(file);
```

Figure 4. Random generator for *file* 2

After generating the dataset, in *file* 1 and *file* 2 Karatsuba algorithm for sequential and for parallel are executed separately and generated the output file is generated which contains the product of those two files used as dataset. The execution time of them are noted for multiple iterations.
Table I describes the results obtained for Karatsuba algorithm when run sequentially and in parallel.

TABLE I. SEQUENTIAL AND PARALLEL EXECUTION TIME

S.No.	Sequential execution Time (ms)	Parallel execution Time (ms)	Speedup
1	19.773	0.671	29.47
2	20.719	0.723	28.66
3	21.304	0.692	30.79
4	20.916	0.712	29.38
5	21.235	0.657	32.32

The average time taken for Karatsuba algorithm to run in sequential is 20.79ms. The average time for Karatsuba algorithm in parallel is 0.691ms. Speedup is a performance parameter measured as the improvement in execution

of a task when executed on two different architectures / environments. The speedup achieved by parallelizing Karatsuba algorithm over the sequential is equal to 30.12.

V. Conclusions

In this paper, it has been presented how multiplication takes place with Karatsuba algorithm in sequential and in CUDA environment. A detailed assessment has been carried out on Karatsuba algorithm to show how it turns out very efficient on implementing on GPU due to inherent parallelism. The computed speedup parameter certifies the improvement with the Karatsuba algorithm's performance. Due to the parallelization, the algorithm witnessed a good timing measure on multiple CUDA cores with respect to when ran sequentially. The parallel algorithm exhibits better timing performance as compared to serial implementation. It is attributed to the ability of the GPU to perform many computations simultaneously, taking advantage of its highly parallel architecture.

References

[1] Diffie, Whitfield, and Martin E. Hellman, "New directions in cryptography," Democratizing Cryptography: The Work of Whitfield Diffie and Martin Hellman, pp. 365-390, 2022.

[2] A. Karatsuba and Yu. Ofman, "Multiplication of Many-Digital Numbers by Automatic Computers," Russian Academy of Sciences, vol. 145(2), pp. 293- 294, 1962.

[3] L. Toom: The complexity of a scheme of functional elements realizing the multiplication of integers. Russian Academy of Sciences, 150(3), pp. 496- 498, 1963.

[4] S. A. Cook, "On minimum computation time of function," Ph.D. dissertation, Department of Mathematics, Harvard University, 1966.

[5] Jitendra V. Tembhurne, Shailesh R. Sathe, "Performance evaluation of long integer multiplication using OpenMP and MPI on shared memory architecture," 2014 Seventh International Conference on Contemporary Computing (IC3), pp. 283-288, IEEE, 2014.

[6] J. V. Tembhurne and S. R. Sathe, "High throughput long integer multiplication using Fast Fourier Transform on parallel workstation," 2014 Annual IEEE India Conference (INDICON), pp.1-6, IEEE, 2014.

[7] Robinson, Jeffrey, Brandon Dixon, and Jeffrey Galloway, "Montgomery multiplication using CUDA," Proceedings of the 2014 ACM Southeast Regional Conference, pp. 1-6, ACM, 2014.