Paper Review: Efficient GPU Parallel Implementation and Optimization of ARIA Block Cipher in CTR Mode and Exhaustive Key Search

isomo

April 20, 2025

1 Quick Look at the Paper

Abstract

- To address this, we merged ARIA's four individual Sbox tables into a single unified 32-bit table, effectively reducing the total memory size from 4KB to 1KB
- the CUDA built-in function __byte_perm() was utilized to efficiently reconstruct the intended output from the reduced unified table without incurring additional computational overhead.
- For exhaustive key search scenarios, we adopted an on-the-fly key expansion approach, significantly minimizing memory usage per thread and enhancing parallel efficiency.

1.1 Introduction

To date, there has been no dedicated study(to our knowledge) on optimizing ARIA in CTR mode or on GPU-accelerated key search for ARIA.

> Note
>
> This paper how to optimizing ARIA in CTR mode?

- We implemented an optimized CTR mode for the ARIA algorithm. Previous researc [6] could not employ a table-copying technique to avoid bank conflicts due to the large size of the Sbox table. In this work, we **reduced the size of the Sbox table.**
- We extensively utilized CUDA built-in function **__byte_perm()**. Encryption processes often require state transformations such as permutations.
- Exhaustive Key Search(ES) in block cipher modes like CTR are suitable for parallel implementations due to their independent block computations. performing key expansion for each thread individually creates a memory burden for storing round keys. To address this, we implemented an **on-the-fly** approach that computes round keys as needed.

> Note
>
> the ECB is also independent block computations, not only the CTR mode.

1.2 Background

1.2.1 Graphic Processing Unit and CUDA Basics

The main memory spaces on Nvidia GPUs include global(device) memory, shared memory,constant memory, and registers. *Global memory* is large(several GB on modern GPUs) but resides in VRAM and has the highest access latency. *Shared memory*, on the other hand, is a smaller on-chip memory(typically 48 KB per SM) that offers much lower latency(comparable to L1 cache). However, shared memory is

divided into banks; if multiple threads in a warp access the same memory bank, a bank conflict arises, serializing those accesses. Another useful memory space is *constant memory*, which is a read-only cache optimized for broadcast. Constant memory is small(e.g., 64 KB) and actually resides in global memory but is cached such that if all threads in a warp read the same address, the value is served from a fast cache. Finally, *registers* are used for per-thread storage of local variables and see the fastest access.

1.3 Optimization Strategy of ARIA on GPU

1.3.1 Shared Memory Utilization through Optimized Sbox Table

To effectively mitigate latency concerns, we propose a strategic approach of placing these expanded Sbox tables into GPU *shared memory*. By replicating the tables, each individual thread is afforded independent and conflict-free access to separate table instances. GPUs inherently execute threads in groups known as warps, each warp comprising 32 threads. Therefore, *replicating lookup tables exactly 32 times* ensures that each thread within a warp accesses a separate memory bank independently.

> Note
>
> on the Figure 3, not show how to reduce the tables size from 4KB down to 1KB. it seems the table element rearrangement.

1.3.2 Efficient Output Reconstruction Using the __byte_perm() Function

directly using the integrated table causes an issue, as it outputs a single 8-bit result instead of the originally intended full 32-bit output. Thus, using the reduced table without additional processing leads to a mismatch between the intended and actual output forms. The CUDA environment offers efficient support for byte-level operations and rearrangements through the __byte_perm() function.

> Note
>
> The specific instruction used on the GPU, is like the rearrangement again do one time.

1.4 Evaluation

> Note
>
> on Table 2 the RTX 3060 it has the 344 Gbps, but only approaches 504 Gbps band on the PCIe 4.0 $\times$ 16, so on the experiment, the throughput is only on the GPU internal memory, not transformed to the PCIe bus?

2 Issues

- The paper does not clearly explain how the Sbox tables are reduced from 4KB to 1KB; Figure 3 appears to show only element rearrangement rather than actual size reduction.
- The optimization and independence of block computations are not unique to CTR mode; ECB mode also allows independent block processing.
- In Table 2, the reported throughput is based on GPU internal memory bandwidth and does not account for PCIe transfer to host memory, which may affect real-world performance.

3 Review Comments

This paper presents an efficient GPU parallel implementation and optimization of the ARIA block cipher in CTR mode, with a focus on memory reduction and parallel key search. The work demonstrates practical engineering efforts, such as Sbox table size reduction and on-the-fly key expansion, and provides experimental results on modern GPUs. However, the main innovation appears limited, as the core optimization relies primarily on rearranging the Sbox table and leveraging existing CUDA features. The

writing is clear and the experiments are well-organized, but the novelty and technical depth may not be sufficient for publication.

- The reduction of the Sbox table from 4KB to 1KB is not clearly explained; it seems to be a rearrangement rather than a true reduction in information or computational complexity.
- The use of the CUDA __byte_perm() function is practical, but this is a standard technique for byte-level operations on GPUs and does not represent a significant algorithmic contribution.
- The overall contribution is more of an engineering optimization than a novel cryptographic or architectural advancement.