



14th Workshop on Current
Trends in Cryptology

CTCrypt 2025

A Probabilistic Model for GPU-Accelerated Cryptographic Algorithm Execution

Fomin DENIS, Chuhno ANDREY



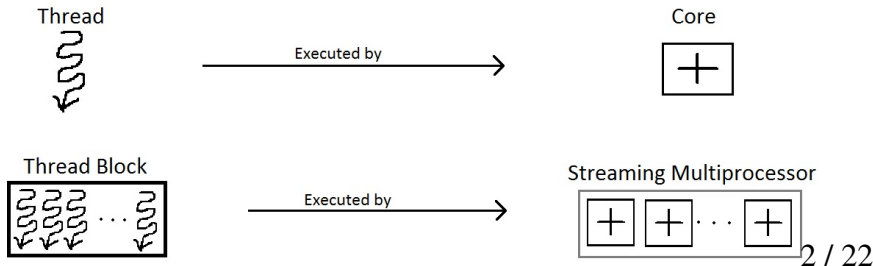
NVIDIA GPU's architecture

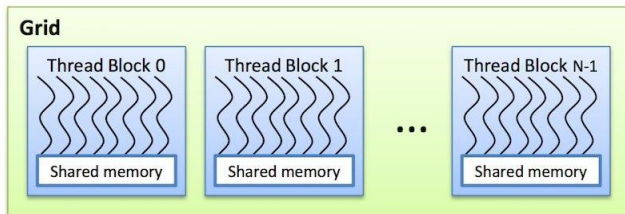
SIMT

SIMT (Single Instruction Multiple Threads): A model where the GPU executes a single instruction across multiple threads simultaneously.

All threads in a group (warp) perform the same operations (e.g., addition, multiplication), but with different data.

Threads are grouped into blocks of the same size and work in parallel.





Shared memory is a cache that is managed by the programmer. According to the CUDA specification¹, sequential 32-bit words in shared memory are assigned to sequential memory modules called *memory banks*.

¹CUDAC++ ProgrammingGuide — docs.nvidia.com ,NVIDIA Corporation & affiliates, 2024, <https://docs.nvidia.com/cuda/cuda-c-programming-guide/>,



NVIDIA GPU's architecture

Compute Capability

- ▶ **1.x:** the warp size is 32 threads, the number of banks is 16, but only 16 threads can access memory at a time;
- ▶ Starting with **2.x:** warp size is 32 threads, number of banks is 32, at one moment 32 threads access shared memory

threads in block – 32

number of shared memory banks – 32



NVIDIA GPU's architecture

Shared memory bank conflict

A **memory bank conflict** in CUDA occurs when multiple threads simultaneously access different addresses located in the same shared memory bank.

Bank 0	Bank 1	Bank 2	Bank 3	Bank 4	Bank 5	Bank 6	Bank 7
0x00	0x04	0x08	0x0C	0x10	0x14	0x18	0x1C
0x20	0x24	0x28	0x2C	0x30	0x34	0x38	0x3C
0x40	0x44	0x48	0x4C	0x50	0x54	0x58	0x5C
0x60	0x64	0x68	0x6C	0x70	0x74	0x78	0x7C
0x80	0x84	0x88	0x8C	0x90	0x94	0x98	0x9C
0xA0	0xA4	0xA8	0xAC	0xB0	0xB4	0xB8	0xBC
0xC0	0xC4	0xC8	0xCC	0xD0	0xD4	0xD8	0xDC
0xE0	0xE4	0xE8	0xEC	0xF0	0xF4	0xF8	0xFC

An example of conflicts in memory banks (bank 0 and bank 2). In the drawing ■ the addresses accessed by one thread are highlighted, ■ the addresses accessed by two threads are highlighted.



NVIDIA GPU's architecture

Time latency

The hardware splits a memory request that has bank conflicts into as many separate conflict-free requests as necessary, reducing effective throughput by a factor equal to the number of individual memory requests. As a result, accesses are processed sequentially, which reduces the overall throughput of the system.

The amount of **time latency** depends on the maximum number of cells in one bank that the threads have accessed.

2	0	3	0	1	0	0	0
Bank 0	Bank 1	Bank 2	Bank 3	Bank 4	Bank 5	Bank 6	Bank 7
0x00	0x04	0x08	0x0C	0x10	0x14	0x18	0x1C
0x20	0x24	0x28	0x2C	0x30	0x34	0x38	0x3C
0x40	0x44	0x48	0x4C	0x50	0x54	0x58	0x5C
0x60	0x64	0x68	0x6C	0x70	0x74	0x78	0x7C
0x80	0x84	0x88	0x8C	0x90	0x94	0x98	0x9C
0xA0	0xA4	0xA8	0xAC	0xB0	0xB4	0xB8	0xBC
0xC0	0xC4	0xC8	0xCC	0xD0	0xD4	0xD8	0xDC
0xE0	0xE4	0xE8	0xEC	0xF0	0xF4	0xF8	0xFC

- ▶ w – number of memory banks;
- ▶ the number of threads is set equal to the number of banks – w ;
- ▶ r – the number of memory cells in each of the memory banks;
- ▶ τ_j – the number of threads that entered the j th bank after they were distributed into cells, $j = \overline{1, w}$
- ▶ ξ_j – the number of different cells in the j th bank that were accessed appeals. In other words — ξ_j is the number of occupied cells in the j th bank.
- ▶ $\beta = \max_j \xi_j$ – the value that determines the maximum delay when accessing shared memory

Probabilistic Model for GPU-Accelerated

Model 1 – multinomial distribution

The set of random variables $\tau_1, \tau_2, \dots, \tau_w$ is described by the probability distribution of the number of particles in each of the banks.

For such $m_j \in \mathbb{N}, j \in 1, 2, \dots, w$ that

$$m_1 + m_2 + \dots + m_w = w,$$

the joint distribution of random variables $(\tau_1, \tau_2, \dots, \tau_w)$ is described by a multinomial distribution:

$$\mathcal{E}_{m_1, \dots, m_w} = \mathbb{P}(\tau_1 = m_1, \tau_2 = m_2, \dots, \tau_w = m_w) = \frac{1}{w^w} \frac{w!}{m_1! \dots m_w!}$$

Probabilistic Model for GPU-Accelerated

Model 1 – multinomial distribution

Because the delay value as a result of **memory bank conflict** is affected by the values of $(\xi_1, \xi_2, \dots, \xi_w)$.

We will need a distribution of the number of empty cells $\mu_0(m, r)$:

$$P(\mu_0(m, r) = k) = C_r^k \left(1 - \frac{k}{r}\right)^m \left(\sum_{i=0}^{r-k} C_{r-k}^i (-1)^i \left(1 - \frac{i}{r-k}\right)^m\right),$$

where $k \in \{0, 1, \dots, r-1\}$.

Using the distribution of the random variable $\mu_0(m, r)$, we can express the joint distribution of $(\xi_1, \xi_2, \dots, \xi_w)$.

Using the distribution of the random variable $\mu_0(m, r)$, we can express the joint distribution of $(\xi_1, \xi_2, \dots, \xi_w)$:

$$\begin{aligned} P(\xi_1 = x_1, \dots, \xi_w = x_w) &= \\ &= \sum_{\substack{i_1 + i_2 + \dots + i_w = w \\ i_j \geq 0}} \frac{w!}{i_1! \cdot \dots \cdot i_w!} \left(\frac{1}{w}\right)^w \prod_{k=1}^w P(\mu_0(i_k, r) = r - x_k) \end{aligned}$$

$$\begin{aligned} P(\beta \leq t) &= P(\xi_1 \leq t, \xi_2 \leq t, \dots, \xi_w \leq t) = \\ &= \sum_{0 < x_1, \dots, x_w < t} P(\xi_1 = x_1, \xi_2 = x_2, \dots, \xi_w = x_w) = \\ &= \sum_{\substack{i_1 + i_2 + \dots + i_w = w \\ i_j \geq 0}} \frac{w!}{i_1! \cdot \dots \cdot i_w!} \left(\frac{1}{w}\right)^w \prod_{k=1}^w P(\mu_0(i_k, r) \geq r - t). \end{aligned}$$

Model 2 is a simplification of Model 1. Basic assumptions of Model 2:

1. Independent distribution of particles by banks. Each value τ_j independent, equally distributed, $\tau_j \sim \text{Bi}\left(w, \frac{1}{w}\right)$.
2. Rejection of a fixed number of particles. In Model 2, the sum of the number of particles in all banks is not necessarily equal to w . This means, in particular, that an event can occur with a non-zero probability in which all $\tau_j = 0$ (no particles hit any pot).
3. Within each bank, the particle cell distribution remains the same as in Model 1.

The number of occupied cells in the selected bank is uniquely determined by the number of empty cells and the joint distribution of random variables $(\xi_1, \xi_2, \dots, \xi_w)$ in model 2 is equal to:

$$\begin{aligned} P(\xi_1 = x_1, \xi_2 = x_2, \dots, \xi_w = x_w) = \\ = \prod_{j=1}^w \sum_{i=0}^w \left(C_w^i \left(\frac{1}{w} \right)^i \left(1 - \frac{1}{w} \right)^{w-i} \cdot P(\mu_0(i, r) = r - x_j) \right) \end{aligned}$$



Access to shared memory for the AES GPU's implementation

Motivation

The choice of AES as an example is due to the following factors:

- ▶ The AES algorithm is widely used;
- ▶ A significant number of papers are devoted to attacks on the implementation of the AES algorithm in runtime ²³, but mathematical models describing the impact of memory bank conflicts on algorithm execution time remain poorly understood.

²Gao Yiwen and Zhou Yongbin, Side-Channel Attacks With Multi-Thread Mixed Leakage, IEEE Transactions on Information Forensics and Security, 2021, 16, 770-785

³Jiang Zhen Hang and Fei Yungsi and Kaeli David R., A complete key recovery timing attack on a GPU, IEEE Computer Society, 2016, 394-405



Access to shared memory for the AES GPU's implementation

Shared memory required size

When implementing the AES algorithm, four replacement tables are used, containing the values of the composition of nonlinear and linear operations. The amount of memory for the tables is determined as follows:

$$M_{T\text{-boxes}} = 4 \cdot 256 \cdot 4 \text{ byte} = 4 \text{ KB},$$

where each of the four tables contains 256 32-bit words. The memory is organized into 32 banks, with 8 values stored in each bank, which follows from the ratio:

$$\frac{256 \text{ words}}{32 \text{ banks}} = 8 \text{ word/bank}.$$

Access to shared memory for the AES GPU's implementation

Probability estimations for β

In model 1 (with experiments)

i	1	2	3	4
$P(\beta = i)$	$1.98 \cdot 10^{-8}$	$1.07 \cdot 10^{-1}$	$6.58 \cdot 10^{-1}$	$2.12 \cdot 10^{-1}$
i	5	6	7	8
$P(\beta = i)$	$2.18 \cdot 10^{-2}$	$1.25 \cdot 10^{-3}$	$3.88 \cdot 10^{-5}$	$5.06 \cdot 10^{-7}$

$M\beta \approx 3.15$, $D\beta \approx 0.395$

In model 2

β	1	2	3	4
$P(\beta = i)$	$1.54 \cdot 10^{-4}$	$1.62 \cdot 10^{-1}$	$6.11 \cdot 10^{-1}$	$2.03 \cdot 10^{-1}$
β	5	6	7	8
$P(\beta = i)$	$2.17 \cdot 10^{-2}$	$1.25 \cdot 10^{-3}$	$3.88 \cdot 10^{-5}$	$5.06 \cdot 10^{-7}$

$M\beta \approx 3.09$, $D\beta \approx 0.457$.



A method for restoring the AES encryption key algorithm by runtime

Method description

The key is restored for the first round of the AES algorithm. The following property of shared memory is used for its recovery:

- ▶ memory access occurs simultaneously by 32 threads;
- ▶ the entire memory is divided into 32 banks, and the access speed is directly proportional to the maximum number of accesses among all memory banks;
- ▶ the memory access speed will be minimal when all threads access the same memory area.

A method for restoring the AES encryption key algorithm by runtime

Method description

1. special plaintexts consisting of M blocks are selected;

$$D_i = \begin{pmatrix} \theta & x_{0,1} & x_{0,2} & x_{0,3} \\ x_{1,0} & x_{1,1} & x_{1,2} & x_{1,3} \\ x_{2,0} & x_{2,1} & x_{2,2} & x_{2,3} \\ x_{3,0} & x_{3,1} & x_{3,2} & x_{3,3} \end{pmatrix} \cdot L^{-1},$$

$$x_{i,j} \in V_8$$

2. the execution time of the encryption of these texts is obtained by the violator;
3. The encryption time for D_i is compared with the encryption of modified blocks: $D'_i = D_i \oplus K$, $K \in V_{32}$

A method for restoring the AES encryption key algorithm by runtime

Random texts encryption

Based on the estimates of the distribution of β , we obtain estimates for its moments.

expectation $M\beta \approx 3.15$, and the variance $D\beta \approx 0.395$

The average and variance of the encryption time with an equally probable choice of M plaintexts can be estimated:

$$m_0 = M \cdot \mathcal{R} \cdot c_1 \cdot 16 \cdot M\beta + M \cdot C,$$

$$d_0 = M \cdot \mathcal{R} \cdot c_1^2 \cdot 16 \cdot D\beta.$$

where C is a constant, $\mathcal{R}$ is the number of AES algorithm iterations.

A method for restoring the AES encryption key algorithm by runtime

Special texts encryption

To encrypt selected texts, the average expectation time m_2 and Variance d_2 are expressed as:

$$m_2 = M \cdot c_1 \cdot ((\mathcal{R} - 1) \cdot 16 \cdot M\beta + 15 \cdot M\beta + 8) + M \cdot C$$

$$d_2 = M \cdot c_1^2 \cdot ((\mathcal{R} - 1) \cdot 16 \cdot D\beta + 15 \cdot D\beta) = M \cdot c_1^2 \cdot D\beta \cdot (16 \cdot \mathcal{R} - 1).$$

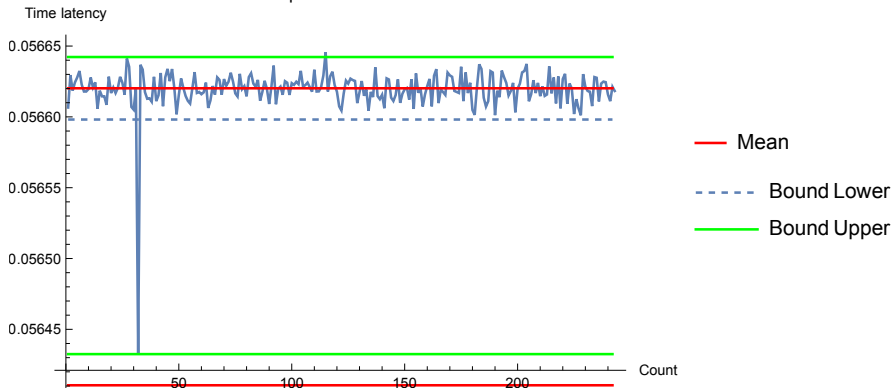
The execution time for specially selected texts is approximated by the normal distribution $\mathcal{N}(m_2, d_2)$, and for random texts by the normal distribution $\mathcal{N}(m_0, d_0)$. And the task of key recovery boils down to testing the hypothesis of the distribution of encryption execution time.



A method for restoring the AES encryption key algorithm by runtime

Example

Experimental Data



The background features the words "THANK YOU" in large, 3D, pink block letters. The letters are slightly offset and have a soft shadow beneath them, giving them a floating appearance. The text is centered horizontally and serves as a backdrop for the main message.

Thank you for your attention.

About model 1

Distribution of the occupied number cells in banks

The placement of w distinguishable particles in w banks, each of which contains r distinguishable cells, can be associated with the placement of these particles in $r \cdot w$ cells. Denote $\rho_1, \dots, \rho_{w \cdot r}$ – the number of particles trapped in each of the cells, respectively, with an equally probable placement.

Calculate the probability of an event

$\{\tau_1 = m_1, \tau_2 = m_2, \dots, \tau_w = m_w\}$, where τ_j is the number of particles trapped in j is the j th pot, and $\sum_{j=1}^w m_j = w$

$$\begin{aligned}
P(\tau_1 = m_1, \tau_2 = m_2, \dots, \tau_w = m_w) &= \\
&= \sum_{\substack{i_1 + \dots + i_{rw} = w \\ i_j \geq 0}} \frac{(w)!}{i_1! \cdot \dots \cdot i_{rw}!} \left(\frac{1}{wr}\right)^w = \\
&\quad \sum_{k=r(j-1)+1}^{rj} i_k = m_j, j = \overline{1, w} \\
&= \sum_{\substack{i_1 + \dots + i_r = m_1 \\ i_j \geq 0}} \sum_{\substack{i_{r+1} + \dots + i_{2r} = m_2 \\ i_j \geq 0}} \dots \sum_{\substack{i_{r(w-1)+1} + \dots + i_{rw} = m_w \\ i_j \geq 0}} \frac{(w)!}{i_1! \cdot \dots \cdot i_{rw}!} \left(\frac{1}{wr}\right)^w =
\end{aligned}$$

$$\begin{aligned}
&= \frac{w!}{m_1! \dots m_w!} \sum_{\substack{i_1 + \dots + i_r = m_1 \\ i_j \geq 0}} \frac{m_1!}{i_1! \dots i_r!} \left(\frac{1}{wr} \right)^{i_1 + \dots + i_r} \\
&\quad \cdot \sum_{\substack{i_{r+1} + \dots + i_{2r} = m_2 \\ i_j \geq 0}} \frac{m_2!}{i_{r+1}! \dots i_{2r}!} \left(\frac{1}{wr} \right)^{i_{r+1} + \dots + i_{2r}} \dots \\
&\quad \dots \sum_{\substack{i_{r(w-1)+1} + \dots + i_{wr} = m_w \\ i_j \geq 0}} \frac{m_w!}{i_{r(w-1)+1}! \dots i_{wr}!} \left(\frac{1}{wr} \right)^{i_{r(w-1)+1} + \dots + i_{wr}} = \\
&= \frac{w!}{m_1! \dots m_w!} \left(\frac{r}{rw} \right)^{m_1} \dots \left(\frac{r}{rw} \right)^{m_w} = \mathcal{E}_{m_1, \dots, m_w}
\end{aligned}$$

Note that for a scheme of equally probable placement of n distinguishable particles in N distinguishable cells, the distribution of μ_0 can be expressed by the sum

$$P(\mu_0(n, N) = k) = \sum_{\substack{i_1 + \dots + i_N = n \\ i_j \geq 0}} \frac{n!}{i_1! \dots i_N!} \frac{1}{N^n}$$

$$\sum_{l=1}^N I\{i_l > 0\} = N - k$$

We get the expression

$$\begin{aligned}
 P(\xi_1 = x_1, \dots, \xi_w = x_w) &= \sum_{\substack{i_1 + \dots + i_{rw} = w \\ i_j \geq 0}} \frac{(w)!}{i_1! \cdot \dots \cdot i_{rw}!} \left(\frac{1}{wr} \right)^w = \\
 &\quad \sum_{k=r(j-1)+1}^{rj} I\{i_k > 0\} = x_j, j=1, \overline{w} \\
 &= \sum_{\substack{m_1 + \dots + m_w = w \\ m_j \geq 0}} \sum_{\substack{i_1 + \dots + i_r = m_1 \\ i_j \geq 0}} \dots \sum_{\substack{i_{r(w-1)+1} + \dots + i_{wr} = m_w \\ i_j \geq 0}} \frac{(w)!}{i_1! \cdot \dots \cdot i_{rw}!} \left(\frac{1}{wr} \right)^w = \\
 &\quad \sum_{k=1}^r I\{i_k > 0\} = x_1 \quad \sum_{k=r(w-1)+1}^{wr} I\{i_k > 0\} = x_1
 \end{aligned}$$

$$\begin{aligned}
&= \sum_{\substack{m_1+\dots+m_w=w \\ m_j \geq 0}} \frac{w!}{m_1! \dots m_w! w^w} \sum_{\substack{i_1+\dots+i_r=m_1 \\ i_j \geq 0 \\ \sum_{k=1}^r \mathbb{I}\{i_k > 0\} = x_1}} \frac{m_1!}{i_1! \dots i_r!} \left(\frac{1}{r}\right)^{m_1} \dots \\
&\quad \dots \sum_{\substack{i_{r(w-1)+1}+\dots+i_{wr}=m_w \\ i_j \geq 0 \\ \sum_{k=r(w-1)+1}^{wr} \mathbb{I}\{i_k > 0\} = x_w}} \frac{m_w!}{i_{r(w-1)+1}! \dots i_{wr}!} \left(\frac{1}{r}\right)^{m_w} = \\
&= \sum_{\substack{m_1+i_2+\dots+m_w=w \\ i_j \geq 0}} \frac{w!}{m_1! \cdot \dots \cdot m_w!} \left(\frac{1}{w}\right)^w \prod_{k=1}^w \mathbb{P}(\mu_0(m_k, r) = r - x_k)
\end{aligned}$$

About AES choosing

In the process of performing a round-by-round conversion, 32 calls are made to the replacement tables. This assumes a uniform distribution of accesses when 32 threads access addresses in 32 memory banks. The most preferred way to implement AES is «bitslice implementation»⁴⁵ (time constant). However, this implementation method will not be considered in this paper because it does not use shared memory.

⁴Cheng Wangzhao and Zheng Fangyu and Pan Wuqiong and Lin Jingqiang and Li Huorong and Li Bingyu, Building Your Private Cloud Storage on Public Cloud Service Using Embedded GPUs, Security and Privacy in Communication Networks, 2018, 512–528

⁵Implementation of Bitsliced AES Encryption on CUDA-Enabled GPU. Lecture Notes in Computer Science, Springer, 2017

Memory size for AES implementation

Each original replacement table $T_i[256]$, where $i \in \{0, 1, 2, 3\}$, is replaced by an extended table $\tilde{T}_i[32][256]$, in which each row contains a complete copy of the original table:

$$\tilde{T}_i[j][k] = T_i[k] \quad \text{for all } j \in \{0, \dots, 31\}, k \in \{0, \dots, 255\}$$

In this case, the amount of memory used will be:

$$M_{\text{newT-boxes}} = 32 \cdot M_{\text{T-boxes}} = 128 \text{ KB.}$$

For the last round we also have to remember replacement table $S[256]$ is duplicated 32 times in $S[32][256]$. This will require:

$$M_{\text{newS-boxes}} = 32 \cdot 256 \cdot 4\text{bytes} = 32 \text{ KB of shared memory.}$$

The total amount of memory required for the specified implementation of the AES algorithm is:

$$M_{\text{newtotal}} = M_{\text{newT-boxes}} + M_{\text{newS-boxes}} = 160 \text{ KB.}$$

This amount of memory is available on modern GPUs with compute capability 9.0, which makes the proposed modification practically feasible.

Memory size for Kuznechik

For the Kuznechik algorithm, it is necessary to store 16 tables, each of which has 256 values of 128 bits and each stream has its own copy of all the tables. The total amount of memory will be:

$$32 \cdot 256 \cdot 16 \cdot 128 \text{bits} = 32 \cdot 4 \cdot 128 \text{Kbit} = 2048 \text{Kb},$$

which significantly exceeds the available shared memory resources (maximum 227 KB for devices with compute capability 9.0).

If implemented without composition with linear transformation, the encryption speed slows down significantly.

Expectation and variance expression

A random variable equal to the execution time of one j th round of the AES algorithm during processing of the k th packet can be expressed as follows

$$\zeta_{j,k} = c_1 \sum_{i=1}^{16} \beta_{i,j,k} + c_{2,j}$$

where:

- ▶ c_1 is a constant that depends on the clock frequency of the graphics processor used.,
- ▶ $c_{2,j}$ — the number of operations performed without accessing shared memory on the j th round
- ▶ $\beta_{i,j,k}$ is a random variable that characterizes the delay in accessing shared memory for the i th 8—bit block on the j th round when encrypting the k th packet.

Expectation and variance expression For $\mathcal{R}$ iterations of the AES algorithm, the encryption time of one k th data packet will be the sum of the execution time of each iteration

$$\nu_k = \sum_{j=1}^{\mathcal{R}} \zeta_{j,k}$$

Average and variance for encryption time:

$$m_0 = \mathbb{M} \left(\sum_{k=1}^M \nu_k \right) = M \cdot \mathcal{R} \cdot c_1 \cdot 16 \cdot \mathbb{M}\beta + M \cdot C,$$

$$d_0 = \mathbb{D} \left(\sum_{k=1}^M \nu_k \right) = M \cdot \mathcal{R} \cdot c_1^2 \cdot 16 \cdot \mathbb{D}\beta.$$

difference between the recovery method and the previously proposed one

In the analysis method described in the article⁶, the average execution time m_1 is expressed as:

$$m_1 = M \cdot c_1 \cdot ((\mathcal{R} - 1) \cdot 16 \cdot M\beta + 15 \cdot M\beta + 1) + M \cdot C.$$

The variance of the execution time d_1 in this case is expressed as:

$$d_1 = M \cdot c_1^2 \cdot ((\mathcal{R} - 1) \cdot 16 \cdot D\beta + 15 \cdot D\beta) = M \cdot c_1^2 \cdot D\beta \cdot (16 \cdot \mathcal{R} - 1).$$

⁶Fomin D. B., A timing attack on CUDA implementations of an AES-type block cipher, Mathematical issues of cryptography, 2016, 7, 2, 121–130

the plaintexts are chosen in such a way that for one of the 2^{32} cases after the first round of encryption in certain places (for example, the first 8 bits) there were values that differed from each other by a constant. For example, 0x00, 0x02, 0x04 and so on. This ensures that 32 warp threads will access 8 different addresses of the first block, resulting in an 8-fold slowdown. In this case, the mean and variance of the execution time will be m_2 and d_2 .

Since $|m_0 - m_1| < |m_0 - m_2|$, and $d_1 = d_2$, the modified method will generally be more efficient, since the difference between the average values of m_0 and m_2 is greater than between m_0 and m_1 . This reduces the number of M blocks needed to achieve the probability of success.