Thread-Adaptive Optimized Parallel Architectures of SLH-DSA on GPUs-v0.11.0.pdf

by User Us

Thread-Adaptive: Optimized Parallel Architectures of SLH-DSA on GPUs

Jiahao Xiang and Lang Li.

***Abstract*—The imminent threat posed by quantum computing necessitates an urgent transition to Post-Quantum Cryptography (PQC) to safeguard sensitive data against future cryptanalytic attacks. The stateless hash-based digital signature algorithm (SLH-DSA) FIPS 205, while quantum-resistant, presents significant computational challenges for practical deployment. This research presents a GPU-accelerated implementation of SLH-DSA that employs a thread-adaptive parallelization methodology to maximize throughput. In contrast to conventional approaches utilizing fixed maximum thread allocation, the proposed implementation dynamically optimizes parallelism levels for individual cryptographic kernel functions, thereby establishing an equilibrium between thread utilization and execution efficiency. Furthermore, granular decomposition of signature components is implemented to enhance thread-level execution performance. Performance evaluation conducted on an NVIDIA RTX 4090 GPU demonstrates that the implementation attains a throughput of 62,239 signatures per second for the SPHINCS$^+$-128f parameter set, representing a significant performance improvement over existing methodologies. The empirical results establish GPUs as viable platforms for SLH-DSA acceleration in high-throughput environments, thus facilitating the practical transition to post-quantum cryptographic standards.**

***Index Terms*—FIPS 205, GPU, SPHINCS$^+$, Signature algorithm.**

I. INTRODUCTION

QUANTUM computers pose a significant threat to current cryptographic systems through their ability to efficiently solve mathematical problems that underpin modern security protocols. This threat materializes in the anticipated "Q-Day", when quantum computers attain sufficient computational power to compromise public encryption systems safeguarding digital communications, authentication mechanisms, and key exchange protocols. Widely deployed public-key cryptosystems such as RSA and ECC are particularly vulnerable to Shor's algorithm, which can efficiently factor large integers and compute discrete logarithms problems considered computationally infeasible using classical computing approaches [1]. In response to these vulnerabilities, the National Institute of Standards and Technology (NIST) initiated the Post-Quantum Cryptography (PQC) standardization process to develop cryptographic schemes resistant to quantum computing capabilities. The imminent threat of encrypted data being harvested now for future decryption once quantum computing matures makes the transition to post-quantum cryptography increasingly urgent for protecting sensitive information and maintaining long-term data security.

SPHINCS$^+$, a prominent stateless hash-based signature scheme and NIST standardization finalist [2], provides robust quantum-attack resistance through secure cryptographic hash functions [3]. This scheme subsequently formed the foundation for the Stateless Hash-based Digital Signature Algorithm (SLH-DSA), now standardized as FIPS 205 [4]. The computational intensity of these hash-based signatures has necessitated research into efficient implementations across various hardware platforms, including CPUs, FPGAs, and GPUs [5], to facilitate organizational transitions to post-quantum cryptographic solutions.

A. Related Work

GPU acceleration for SPHINCS$^+$ has evolved substantially in recent years. Lee and Hwang [6] established the fundamental parallel implementation techniques for hash-based signatures, demonstrating the viability of GPU acceleration for post-quantum cryptography. Building on this foundation, Kim et al. [7] developed parallel methods for critical SPHINCS$^+$ components—specifically FORS, WOTS$^+$, and Merkle tree computations. Their implementation on the NVIDIA RTX 3090 achieved significant throughput improvements, despite efficiency constraints from multiple kernel invocations.

Subsequently, Wang et al. [8] introduced CUSPX, a sophisticated three-level parallelism framework integrating algorithmic, data, and hybrid parallelization approaches. Their implementation featured optimized parallel Merkle tree construction algorithms and strategic load-balancing techniques, resulting in substantial performance enhancements.

B. Motivation

Existing GPU implementations of SLH-DSA reveals two fundamental efficiency constraints. First, conventional approaches apply uniform thread allocation across all cryptographic operations, neglecting the distinct computational characteristics of individual functions. This static resource allocation creates significant imbalances certain operations

This work is supported by the Hunan Provincial Natural Science Foundation of China (2022JJ30103), Postgraduate Scientific Research Innovation Project of Hunan Province (CX20240977), "the 14th Five-Year Plan" Key Disciplines and Application-oriented Special Disciplines of Hunan Province (Xiangjiaotong [2022] 351), the Science and Technology Innovation Program of Hunan Province (2016TP1020).

Jiahao Xiang and Lang Li are affiliated with the Hunan Provincial Key Laboratory of Intelligent Information Processing and Application, as well as the Hunan Engineering Research Center of Cyberspace Security Technology and Applications, both located at Hengyang Normal University, Hengyang 421002, China. They are also faculty members of the College of Computer Science and Technology at Hengyang Normal University. (e-mail: jiahaoxiang2000@gmail.com; lilang911@126.com)

suffer from excessive synchronization overhead while others underutilize available computational resources.

Second, the hierarchical structure of SLH-DSA, comprising multiple hash operations distributed across FORS, WOTS⁺, and Merkle tree components, permits decomposition into smaller computational units. However, current implementations emphasize maximizing thread count rather than optimizing computational efficiency, resulting in suboptimal per-thread performance despite high hardware utilization rates. These limitations necessitate an adaptive parallelization approach that dynamically determines optimal thread configurations for specific cryptographic functions while enabling fine-grained decomposition of signature components.

C. Contributions

This paper presents a thread-adaptive GPU-based implementation of SLH-DSA with the following key contributions:

1) A dynamic thread allocation methodology that optimizes thread configurations for individual cryptographic functions based on their computational characteristics, effectively balancing parallelism with execution efficiency to minimize synchronization overhead while maximizing throughput.
2) A sophisticated function-level parallelization approach that strategically decomposes cryptographic operations into independent computational tasks, significantly reducing latency and enhancing the performance of core SLH-DSA primitives.
3) Performance evaluation on NVIDIA GPU architecture demonstrating a throughput of 62,239 SLH-DSA signatures per second (for SPHINCS⁺-128f), representing substantial improvement over state-of-the-art implementations. The complete implementation is available as an open-source repository at https://github.com/jiahaoxiang2000/sphincs-plus.

The paper is structured as follows: Section II presents the fundamental concepts of the SLH-DSA signature scheme; Section III describes the architectural design and implementation details; Section IV analyzes performance results and comparative metrics; and Section V summarizes findings and discusses future research directions.

II. Preliminaries

A. SLH-DSA Overview

SLH-DSA represents a stateless hash-based signature scheme that provides post-quantum security through hierarchical certification architecture. The signature generation mechanism comprises three fundamental components:

- **WOTS⁺ (Winternitz One-Time Signature)**: A one-time scheme facilitating authentication paths and supporting Merkle tree construction
- **FORS (Forest Of Random Subsets)**: A few-time signature scheme utilizing k components, each containing t elements selected from pseudorandom subsets
- **Hypertree**: A multi-layer structure with height h divided into d layers, each containing Merkle trees of height h/d for WOTS⁺ public key authentication

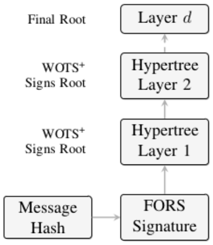

Fig. 1. SLH-DSA signature generation flow. A message hash undergoes FORS signing to produce k authentication paths, subsequently authenticated by a d-layer hypertree. Each layer employs WOTS⁺ to sign the previous layer's root, culminating in a final root signature.

The SLH-DSA signature generation process, illustrated in Fig. 1, implements a hierarchical authentication structure. Initially, a message digest is generated through hashing, followed by FORS few-time scheme signing, which produces k authentication paths comprising t elements each. The resulting FORS public key undergoes authentication via a d-layer hypertree, where each layer applies WOTS⁺ to sign the lower layer's root. This signature chain terminates at the final root node, enabling efficient verification while maintaining robust hash-based security.

SLH-DSA offers "simple" and "robust" operational modes to optimize speed-security trade-offs. Various parameter sets accommodate different requirements regarding signature size, security level, and computational efficiency. The scheme derives all security properties from underlying hash functions, rendering it resistant to quantum computational attacks.

B. GPU Computing Model

Graphics Processing Units (GPUs) incorporate numerous cores organized within Streaming Multiprocessors (SMs). This parallel architecture implements Single Instruction, Multiple Thread (SIMT) execution, organizing threads into warps that collectively form blocks. These blocks are distributed across available SMs, enabling thousands of concurrent threads to execute similar instructions simultaneously.

The CUDA framework enhances computational throughput through memory optimization strategies including coalesced memory accesses, shared memory utilization, and constant memory buffering. These techniques facilitate extensive parallelization of SLH-DSA computations, yielding performance improvements through the combined application of thread-level, data-level, and algorithmic parallelism.

III. Optimized Implementation of SLH-DSA

This section presents a novel optimization framework for SLH-DSA that employs a dual-component architecture, as illustrated in Fig. 2. The framework integrates Adaptive Thread Allocation (ATA) with Function-Level Parallelism (FLP) to maximize computational efficiency while preserving cryptographic integrity.

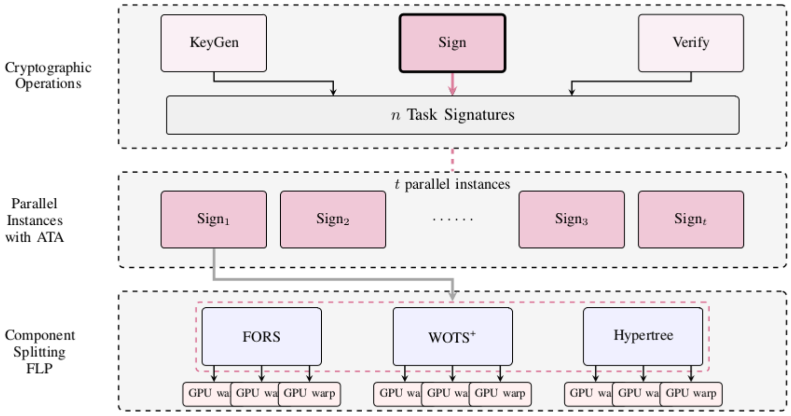

Fig. 2. Thread-adaptive parallelization architecture for SLH-DSA. The architecture comprises three hierarchical layers: the top layer represents cryptographic operations (with Sign highlighted); the middle layer implements Adaptive Thread Allocation (ATA) with dynamically optimized thread configurations distributed across t parallel instances; the bottom layer employs Function-Level Parallelism (FLP) by decomposing operations into algorithmic components (FORS, WOTS$^+$, and Hypertree). Each component distributes computation across multiple GPU warps, with each warp containing 32 threads as the fundamental execution unit.

A. Adaptive Thread Allocation

Conventional GPU implementations of cryptographic algorithms typically apply maximum thread allocation universally, neglecting the distinct computational profiles of individual operations within SLH-DSA. Our research introduces a thread-adaptive methodology that precisely calibrates parallelization based on function-specific characteristics to achieve optimal performance.

1) Performance Modeling: The execution time for each cryptographic function g_i in SLH-DSA is characterized by:

$$T(g_i, t) = \alpha_i + \frac{\beta_i}{t} + \gamma_i \cdot t, \tag{1}$$

This model captures three essential components: α_i represents invariant computational overhead independent of thread count, $\frac{\beta_i}{t}$ reflects the parallelizable workload component that scales inversely with thread count, and $\gamma_i{\cdot}t$ quantifies thread management overhead that increases linearly with thread allocation. The formulation encapsulates the fundamental parallelization tradeoff between computational acceleration and synchronization costs.

2) Optimal Thread Determination: By minimizing $T(g_i, t)$ with respect to t, the optimal thread allocation t_i^* for each function is derived as:

$$t_i^* = \sqrt{\frac{\beta_i}{\gamma_i}}, \tag{2}$$

The parameters α_i, β_i, and γ_i were determined through systematic profiling of each cryptographic operation across multiple thread configurations, establishing an empirical foundation for optimization decisions.

3) Dynamic Implementation: The thread allocation optimization follows the methodology outlined in Algorithm 1.

Algorithm 1 Adaptive Thread Allocation (ATA)

```
Input: Set of cryptographic functions $G = \{g_1, g_2, \ldots, g_m\}$
Output: Optimized thread configuration for each function
for each function $g_i \in G$ do
    Profile $g_i$ with varying thread counts $t \in \{2^j | j \in [5, 10]\}$
    Fit performance data to model $T(g_i, t) = \alpha_i + \frac{\beta_i}{t} + \gamma_i \cdot t$
    Calculate $t_i^* = \sqrt{\frac{\beta_i}{\gamma_i}}$
    Round $t_i^*$ to nearest power of 2 for GPU scheduler compatibility
end for
Store thread configuration in constant memory lookup table
return Thread configuration table
```

This architecture dynamically balances parallelism against synchronization overhead, yielding precise resource allocation tailored to each cryptographic operation. The thread configuration resides in GPU constant memory for low-latency access during execution, ensuring computational efficiency without introducing additional runtime overhead.

B. Function-Level Parallelism

This implementation decomposes hash functions into parallel tasks rather than treating them as atomic operations, reducing cryptographic latency through fine-grained parallelization. The following strategies are employed:

- **WOTS$^+$ Parallelization**: Concurrent computation of l independent hash chains by assigning chains or segments to distinct threads. For hypertree structures, multiple WOTS$^+$ public keys are generated simultaneously, with shared memory optimizing authentication path construction.
- **FORS Parallelization**: Parallel generation of $k \times 2^a$ secret key elements and leaf nodes, followed by concurrent construction of k Merkle trees. Shared memory aggregates tree roots to minimize global memory accesses during public key derivation.
- **Hypertree Parallelization**: Concurrent construction of multiple Merkle trees across all d layers. Threads compute nodes at identical levels but in different trees, enabling simultaneous execution of WOTS$^+$ key generation, hash chain computation, and node hashing operations.

These techniques, combined with coalesced memory access patterns and strategic shared memory utilization, significantly reduce execution latency compared to signature-level parallelism approaches.

IV. Performance Evaluation

This section presents empirical analyses of the SLH-DSA implementation, examining optimization effectiveness and performance characteristics across computational environments.

A. Experimental Setup

Evaluation was conducted on standardized hardware platforms to facilitate comparative analysis. The experimental configuration comprised:

TABLE I
Experimental Hardware Configuration

Component	GPU System
Processor	NVIDIA RTX 4090
Memory	24GB GDDR6X
Operating System	Ubuntu 24.04 LTS
Compiler	CUDA 12.5, GCC 13.3.0

Testing utilized standardized NIST parameter sets for SLH-DSA at security levels 1 and 3, corresponding to 128 and 192 bits of security. Each measurement was replicated 20 times, with statistical outliers removed using median absolute deviation techniques.

B. Thread Allocation Efficiency

ATA efficacy was evaluated through execution time measurements across varying thread configurations for core cryptographic functions. Table II presents experimentally derived model parameters and optimal thread allocations.

Results confirm that non-optimal thread counts introduce significant performance degradation. Implementation with thread counts rounded to powers of 2 (typically 32,768) demonstrated 10-20% performance improvements compared to fixed-thread approaches, validating the adaptive thread allocation model's effectiveness.

TABLE II
Thread Model Parameters and Optimal Allocations

Operation	α_i	β_i	γ_i	t_i^*
128F-keypair	52.06	506,000.57	1.26E-4	63,310
128F-sign	1386.01	13,231,567.75	3.60E-3	60,636
128F-verify	164.72	1,395,012.54	4.54E-4	55,407
128S-keypair	3317.74	32,046,199.26	7.15E-3	66,929
128S-sign	23716.81	248,632,501.64	6.59E-2	61,419
128S-verify	63.22	484,914.46	1.44E-4	57,968
192F-keypair	79.37	822,859.78	2.40E-4	58,560
192F-sign	2319.70	23,961,551.63	8.55E-3	52,932
192F-verify	267.63	2,342,878.75	8.91E-4	51,274

C. Function-Level Parallelism Impact

FLP effectiveness was assessed through component-level latency analysis within the signing process. Table III presents the latency distribution for SPHINCS$^+$-128f across FORS, WOTS⁺, and Hypertree operations.

TABLE III
Latency Breakdown of Signing Operation (SPHINCS$^+$-128f)

Component	Latency (ms)	Percentage of Total
WOTS⁺ Sign	1.857	0.35%
FORS Sign	29.371	5.58%
Hypertree Sign	495.252	94.07%
Total Sign Latency	526.48	100.00%

Analysis indicates Hypertree construction dominates signing latency at 94.07% of total execution time. Despite FLP optimizations of individual hash computations, inherent sequential dependencies and operation volume in the Hypertree constrain achievable parallelism compared to FORS and WOTS⁺ components. This highlights the necessity for targeted optimization of Hypertree structures for future performance enhancements.

D. Comparative Performance Analysis

The implementation was benchmarked against state-of-the-art alternatives using SPHINCS$^+$-128f parameters. Table IV presents latency and throughput metrics across comparable hardware platforms.

Consistent performance improvements were observed across all cryptographic operations compared to reference implementations on identical hardware. Key generation, signature generation, and verification latencies decreased by 9.55%, 13.55%, and 10.03% respectively, corresponding to throughput increases of 10.60%, 15.68%, and 11.14%. Signature generation operations exhibited the most substantial performance enhancements.

E. Scalability Analysis

Implementation scalability was evaluated across varying security parameters and computational complexities. Table V presents execution metrics for signature generation across different parameter sets.

TABLE IV
PERFORMANCE COMPARISON OF SLH-DSA IMPLEMENTATIONS

Parameter Sets, Year [Work], Tasks	**Latency (ms)**			**Throughput (tasks/sec)**			**Device**
	KG	**Sign**	**Verify**	**KG**	**Sign**	**Verify**	
SPHINCS$^+$-128f, 2024 [7], 512	0.71	11.53	1.79	725,118 (55%)	44,391 (97%)	285,681 (81%)	RTX 3090
SPHINCS$^+$-128f, 2025 [8], 41,984	32.07	924.24	119.16	1,309,136 (100%)	45,425 (100%)	352,333 (100%)	RTX 3090
SPHINCS$^+$-128f, 2025 [8]†, 32,768	22.82	609.03	72.51	1,435,690 (109.7%)	53,804 (118.4%)	451,883 (128.3%)	RTX 4090
SPHINCS$^+$-128f, This work, 32,768	**20.64**	**526.48**	**65.24**	**1,587,849 (121.3%)**	**62,239 (137.0%)**	**502,243 (142.5%)**	RTX 4090

†: Results obtained by executing previously published implementations on the RTX 4090 test environment for direct hardware-equivalent comparison.

TABLE V
SCALABILITY ACROSS SLH-DSA SIGNATURE PARAMETER SETS

Parameter Set	**Latency (ms)**	**Throughput (tasks/sec)**
SPHINCS$^+$-128s, [8]	12,185.35	2,689
SPHINCS$^+$-128s	9,125.94	3,591
SPHINCS$^+$-192f, [8]	1,067.26	30,703
SPHINCS$^+$-192f	977.45	33,524
SPHINCS$^+$-192s, [8]	21,252.11	1,542
SPHINCS$^+$-192s	18,711.43	1,751

Superior performance was observed across all parameter configurations, with notable improvements for robust variants. The SPHINCS$^+$-128s parameter set exhibited a 25% latency reduction and 33.5% throughput enhancement. Thread-adaptive techniques proved most effective for computationally intensive operations with smaller hash sizes. Diminishing returns were observed at higher security levels due to increased synchronization overhead, reflecting an inherent trade-off between security parameters and parallel execution efficiency.

V. CONCLUSION

A thread-adaptive GPU implementation of SLH-DSA was developed that dynamically allocates computational resources based on cryptographic function profiles while decomposing operations into finely-grained parallel tasks. Performance evaluation on an NVIDIA RTX 4090 GPU demonstrated a throughput of 62,239 signatures per second for SPHINCS$^+$-128f parameters, representing a 15.68% improvement over state-of-the-art implementations. Empirical analysis identified Hypertree construction as the primary performance bottleneck, accounting for 94% of signing latency despite component-level optimizations. The implementation exhibited particular efficacy for robust parameter variants, with SPHINCS$^+$-128s showing a 25% latency reduction and 33.5% throughput enhancement. These results establish GPU platforms as viable acceleration frameworks for post-quantum cryptographic schemes. Future work will focus on Hypertree structure optimization and performance evaluation across diverse GPU architectures.

REFERENCES

[1] Z. Yang, M. Zolanvari, and R. Jain, "A survey of important issues in quantum computing and communications," *IEEE Commun. Surv. Tutorials*, vol. 25, no. 2, pp. 1059–1094, 2023.

[2] M. V. Yesina, Y. V. Ostrianska, and I. D. Gorbenko, "Status report on the third round of the nist post-quantum cryptography standardization process," *Radiotekhnika*, no. 210, pp. 75–86, 2022.

[3] D. J. Bernstein, A. Hülsing, S. Kölbl, R. Niederhagen, J. Rijneveld, and P. Schwabe, "The sphincs$^+$ signature framework," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, CCS 2019, London, UK, November 11-15, 2019*, L. Cavallaro, J. Kinder, X. Wang, and J. Katz, Eds. ACM, 2019, pp. 2129–2146.

[4] "Stateless hash-based digital signature standard," 2024. [Online]. Available: https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.205.pdf

[5] D. Joseph, R. Misoczki, M. Manzano, J. Tricot, F. D. Pinuaga, O. Lacombe, S. Leichenauer, J. Hidary, P. Venables, and R. Hansen, "Transitioning organizations to post-quantum cryptography," *Nat.*, vol. 605, no. 7909, pp. 237–243, 2022.

[6] W. Lee and S. O. Hwang, "High throughput implementation of post-quantum key encapsulation and decapsulation on GPU for internet of things applications," *IEEE Trans. Serv. Comput.*, vol. 15, no. 6, pp. 3275–3288, 2022.

[7] D. Kim, H. Choi, and S. C. Seo, "Parallel implementation of SPHINCS+ with gpus," *IEEE Trans. Circuits Syst. I Regul. Pap.*, vol. 71, no. 6, pp. 2810–2823, 2024.

[8] Z. Wang, X. Dong, H. Chen, Y. Kang, and Q. Wang, "Cuspx: Efficient gpu implementations of post-quantum signature sphincs+," *IEEE Transactions on Computers*, vol. 74, no. 1, pp. 15–28, 2025.

Jiahao Xiang is pursuing a Master's degree in Electronic Information at Hengyang Normal University, China. His research focuses on cryptographic engineering and efficient implementations of block ciphers on resource-constrained devices. Publications include works on lightweight cryptography optimization and contributions to open-source cryptographic projects.

Lang Li received his Ph.D. and Master's degrees in computer science from Hunan University, Changsha, China, in 2010 and 2006, respectively, and earned his B.S. degree in circuits and systems from Hunan Normal University in 1996. Since 2011, he has been working as a professor in the College of Computer Science and Technology at the Hengyang Normal University, Hengyang, China. He has research interests in embedded system and information security.

Thread-Adaptive Optimized Parallel Architectures of SLH-DSA on GPUs-v0.11.0.pdf

ORIGINALITY REPORT

10%	9%	8%	4%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	Submitted to Veermata Jijabai Technological Institute Student Paper	2%
2	Lianrui Deng, Lang Li, Yu Ou, Jiahao Xiang, Shengcheng Xia. "Tripm: a multi-label deep learning SCA model for multi-byte attacks", International Journal of Machine Learning and Cybernetics, 2025 Publication	1%
3	eprint.iacr.org Internet Source	1%
4	thequantuminsider.com Internet Source	1%
5	www.mdpi.com Internet Source	1%
6	koreascience.kr Internet Source	<1%
7	www.frontiersin.org Internet Source	<1%
8	linnk.ai Internet Source	<1%
9	pure.tue.nl Internet Source	<1%
10	Sathwik GS, Barun Kumar Acharya, Bilal Ali, Deepu S. P., Sumam David S.. "Real-Time Hardware Implementation of 3D Sound Synthesis", 2020 IEEE Asia Pacific Conference on Circuits and Systems (APCCAS), 2020 Publication	<1%

11	Xiantong Huang, Lang Li, Jinling Yang. "IVLBC: An Involutive Lightweight Block Cipher for Internet of Things", IEEE Systems Journal, 2022 Publication	<1 %
12	qmro.qmul.ac.uk Internet Source	<1 %
13	web.archive.org Internet Source	<1 %
14	Wai-Kong Lee, Hwajeong Seo, Seong Oun Hwang, Ramachandra Achar, Angshuman Karmakar, Jose Maria Bermudo Mera. "DPCrypto: Acceleration of Post-Quantum Cryptography Using Dot-Product Instructions on GPUs", IEEE Transactions on Circuits and Systems I: Regular Papers, 2022 Publication	<1 %
15	mirror.funkfreundelandshut.de Internet Source	<1 %
16	Hassan Khodaiemehr, Khadijeh Bagheri, Chen Feng. "Navigating the Quantum Computing Threat Landscape for Blockchains: A Comprehensive Survey", Institute of Electrical and Electronics Engineers (IEEE), 2023 Publication	<1 %

Exclude quotes	Off	Exclude matches	Off
Exclude bibliography	On		