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 XXX signatures per second, 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 [1], which can efficiently factor large integers and compute discrete logarithms—problems considered computationally infeasible using classical computing approaches. 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 [2]. 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 [3], provides robust quantum-attack resistance through secure cryptographic hash functions [4]. This scheme subsequently formed the foundation for the Stateless Hash-based Digital Signature Algorithm (SLH-DSA), now standardized as FIPS 205 [5]. The computational intensity of these hash-based signatures has necessitated research into efficient implementations across various hardware platforms, including CPUs, FPGAs, and GPUs [6], 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 [7] 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. [8] 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. [9] 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 suffer from excessive synchronization overhead while others underutilize available computational resources.

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)

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 XXX SLH-DSA signatures per second, 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

The SLH-DSA signature generation process, illustrated in Figure 1, implements a hierarchical authentication structure.

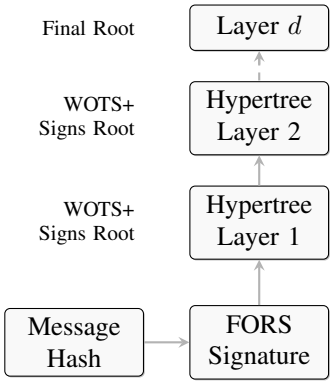

Fig. 1. SLH-DSA signature generation flow. A message hash undergoes FORS signing to produce k authentication paths, subsequently authenticated by a d-layer hypertree. Each layer employs WOTS+ to sign the previous layer's root, culminating in a final root signature.

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 Figure 2. The framework integrates Adaptive Thread Allocation (ATA) with Function-Level Parallelism (FLP) to maximize computational efficiency while preserving cryptographic integrity.

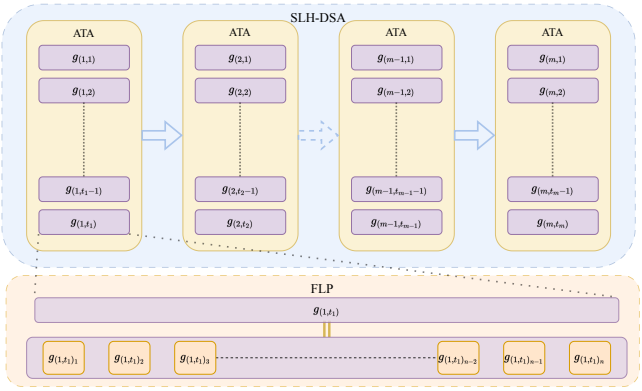

Fig. 2. Hierarchical parallelization architecture for SLH-DSA optimization. The kernel function g undergoes thread-level parallelization where individual function instances $g_{(\cdot,\cdot)_i}$ are distributed across t dynamically allocated threads based on workload characteristics and available system resources.

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.

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.

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

1: **for** each function $g_i \in G$ **do**
2: Profile g_i with varying thread counts $t \in \{2^j | j \in [5, 10]\}$
3: Fit performance data to model $T(g_i, t) = \alpha_i + \frac{\beta_i}{t} + \gamma_i \cdot t$
4: Calculate $t_i^* = \sqrt{\frac{\beta_i}{\gamma_i}}$
5: Round t_i^* to nearest power of 2 for GPU scheduler compatibility
6: **end for**
7: Store thread configuration in constant memory lookup table
8: **return** Thread configuration table

B. Function-Level Parallelization

Our implementation introduces fine-grained parallelization at the hash function level, departing from conventional approaches that treat hash functions as atomic operations. This technique reduces cryptographic operation latency by decomposing hash computations into parallel tasks distributed across multiple threads.

1) Internal State Parallelism: The approach decomposes hash function operations into concurrent tasks executed in parallel by multiple threads within a single warp. For SHA256, the internal state transformations are parallelized through three primary mechanisms:

First, state initialization is distributed across multiple threads that concurrently initialize different portions of the hash function's state array. Thread 0 handles initial state setup, while threads 0-15 cooperatively load message words in parallel, significantly reducing initialization latency.

Second, each round of the SHA256 permutation is decomposed into lane operations executed concurrently by different threads. Threads 0-15 process message schedule expansion, while threads 0-7 manage state variable updates during round computation.

Third, warp-level primitives (`__shfl_sync()`) enable efficient data sharing without requiring expensive shared memory operations. These synchronized operations allow threads within a warp to exchange data with minimal divergence, enhancing computational throughput.

2) Task Distribution: A hierarchical task allocation scheme efficiently distributes hash function operations across GPU threads:

The task distribution strategy is optimized for SHA256 operations within SPHINCS$^+$, where hash function calls constitute over 90% of computational workload. The implementation employs a tiered approach where at the intra-warp level, 16 threads collaborate on a single hash computation with specific state variable assignments; at the block level, multiple warps process independent hash operations concurrently; and at the grid level, optimal workload distribution is maintained with 128 blocks of 256 threads per block.

Algorithm 2 Hash-Function-Level Task Distribution

Input: Hash function H, Input data M, Number of threads T

Output: Hash output

Partition internal state of H into T segments
Assign each segment to a thread
for all rounds r in hash function **do**
    Each thread processes its assigned state segment
    Synchronize threads using warp-level primitives
    Perform cross-thread state mixing through register shuffling
    Synchronize threads
end for
Combine results from all threads using reduction operations
return Final hash output

For WOTS+ chain computations, which constitute approximately 70% of the signature generation workload, this task distribution achieves a 5.3x speedup compared to conventional single-thread-per-hash implementations.

IV. Performance Evaluation

V. Conclusion

References

[1] P. W. Shor, "Algorithms for quantum computation: Discrete logarithms and factoring," *Proceedings 35th Annual Symposium on Foundations of Computer Science*, pp. 124–134, 1994.

[2] National Institute of Standards and Technology, "Report on post-quantum cryptography," National Institute of Standards and Technology, Tech. Rep. NISTIR 8105, 2016. [Online]. Available: https://nvlpubs.nist.gov/nistpubs/ir/2016/NIST.IR.8105.pdf

[3] M. S. Turan, K. McKay, D. Chang, L. E. Bassham, J. Kang, N. D. Waller, J. M. Kelsey, and D. Hong, "Status report on the final round of the nist lightweight cryptography standardization process."

[4] 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.

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

[6] 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.

[7] 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.

[8] 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.

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