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

Jiahao Xiang and Lang Li.

***Abstract*—Quantum computing threatens classical cryptography, creating urgency for Post-Quantum Cryptography adoption before encrypted data harvesting enables future decryption. The FIPS 205 Stateless Hash-based Digital Signature Algorithm (SLH-DSA) offers quantum resistance but faces performance bottlenecks impeding widespread implementation. This research presents a GPU-accelerated implementation of SLH-DSA utilizing two complementary optimization strategies. First, Adaptive Thread Allocation dynamically calibrates parallelism levels through empirical performance modeling, balancing computational acceleration against synchronization overhead for each cryptographic operation. Second, Function-Level Parallelism decomposes WOTS$^+$, FORS, and Hypertree components into fine-grained concurrent tasks with optimized memory access patterns, reducing execution latency. Performance evaluation on an NVIDIA RTX 4090 GPU demonstrates a throughput of 62,239 signatures per second for the SHA2-128f parameter set, representing a 1.16× improvement over state-of-the-art implementations.**

***Index Terms*—Post-quantum cryptography, FIPS 205, Stateless hash-based signature, Implementation, Parallel computing.**

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, 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 techniques have been extensively employed for cryptographic algorithms, with notable implementations for conventional primitives such as AES [6]. These acceleration methodologies have subsequently been adapted for post-quantum cryptography, particularly SLH-DSA, where implementation approaches have evolved substantially in recent years. Lee and Hwang [7] established the fundamental parallel implementation techniques for SLH-DSA, demonstrating the viability of GPU acceleration for post-quantum cryptography. Building on this foundation, Kim et al. [8] developed parallel methods for critical SLH-DSA 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. Contributions

Current GPU implementations of SLH-DSA exhibit two critical efficiency limitations. First, uniform thread allocation across cryptographic operations disregards their heterogeneous computational characteristics, resulting in resource contention for complex operations and under utilization for simpler functions. Second, performance bottlenecks persist throughout the

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

SLH-DSA hierarchy due to insufficient component-level parallelism. Without fine-grained parallelization of WOTS$^+$, FORS, and Hypertree components, inherent concurrency opportunities remain unexploited. These constraints necessitate an adaptive approach that dynamically configures thread allocation while decomposing operations into parallelizable tasks to maximize resource utilization across the signature generation workflow. This paper presents a Thread-Adaptive GPU-based implementation of SLH-DSA with the following key contributions:

1) An Adaptive Thread Allocation (ATA) methodology that optimizes thread configurations for individual cryptographic operations based on their computational characteristics, effectively balancing parallelism with execution efficiency to reduce resource contention and kernel launch overhead, thereby improving overall throughput.
2) A Function-Level Parallelism (FLP) approach that decomposes cryptographic operations into independent computational tasks, addressing inefficiencies not only at the top-level but also within fine-grained components, which further reduces latency and enhances 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, representing 1.16× 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 is a stateless hash-based signature scheme that achieves post-quantum security through a hierarchical authentication structure. The signature generation process consists of three main 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 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.

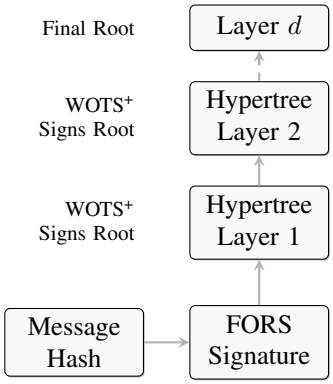

Fig. 1. SLH-DSA signature generation flow.

SLH-DSA offers "small" (s) and "fast" (f) operational modes, where "small" denotes reduced signature size and "fast" indicates higher signature generation speed, across different parameter sets and security levels. Various parameter sets accommodate different requirements regarding signature size, security level, and computational efficiency. All security properties are derived from underlying hash functions, providing resistance 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

The Thread-Adaptive parallelization architecture for SLH-DSA, illustrated in Fig. 2, consists of three hierarchical layers. The top layer represents cryptographic operations, with the Sign operation emphasized. The middle layer applies ATA, distributing dynamically optimized thread configurations across t parallel instances. The bottom layer employs FLP, decomposing operations into algorithmic components—WOTS$^+$, FORS, and Hypertree—with each component mapped to multiple GPU warps of 32 threads.

A. Adaptive Thread Allocation

Thread-Adaptive allocation methodology through precise calibration of thread counts based on empirical performance

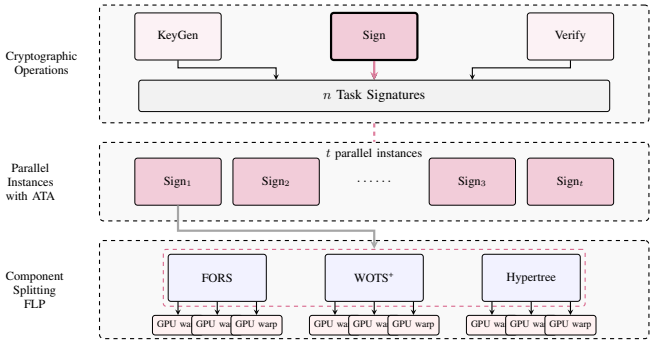

Fig. 2. Thread-adaptive parallelization architecture for SLH-DSA.

models, establishing function-specific optimal configurations that balance parallelizable workload against synchronization costs. Performance gains are achieved through the elimination of thread contention in complex operations and reduction of management overhead in simpler functions, resulting in improved overall system throughput.

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) Thread Configuration Optimization: The thread allocation optimization methodology is presented in Algorithm 1, which profiles each cryptographic function across thread configurations from 2^{11} to 2^{17}. Performance data is fitted to the analytical model in Equation (1), determining parameters α_i, β_i, and γ_i. The theoretically optimal thread count is calculated and rounded to the nearest power of two for GPU warp alignment. This approach optimizes resource allocation for each cryptographic operation, minimizing execution time across the signature generation workflow.

Algorithm 1 Thread Configuration Optimization

Input: Set of cryptographic functions $G = \{g_1, g_2, \ldots, g_m\}$
Output: Optimized thread configuration for each function

Initialize $\mathsf{ThreadConfig}[\,] \leftarrow \emptyset$
for $g_i \in G$ **do**
  $\mathsf{ThreadCounts} \leftarrow \{2^{11}, 2^{12}, 2^{13}, 2^{14}, 2^{15}, 2^{16}, 2^{17}\}$
  $\mathsf{PerfData} \leftarrow \mathsf{PROFILE}(g_i, \mathsf{ThreadCounts})$
  $(\alpha_i, \beta_i, \gamma_i) \leftarrow \mathsf{FIT\_MODEL}(\mathsf{PerfData})$
  $t_{\mathsf{opt}} \leftarrow \sqrt{\frac{\beta_i}{\gamma_i}}$
  $t_i^* \leftarrow \mathsf{ROUND\_POWER2}(t_{\mathsf{opt}})$
  $\mathsf{ThreadConfig}[g_i] \leftarrow t_i^*$
end for
return ThreadConfig

B. Function-Level Parallelism

Function-Level Parallelism decomposes cryptographic operations into granular computational tasks that can be executed concurrently rather than treating them as atomic units. This approach enables fine-grained parallelization across multiple execution threads, resulting in reduced cryptographic latency through optimal resource utilization. The implementation applies this methodology to the three core components of SLH-DSA: WOTS$^+$, FORS, and Hypertree.

1) WOTS$^+$ Parallelization: The WOTS$^+$ implementation facilitates concurrent computation of l independent hash chains by allocating individual chains or chain segments to distinct GPU threads. After hash chain computation completion, results are consolidated into leaf nodes using GPU shared memory rather than global memory to minimize access latencies. For hypertree structures, concurrent generation of multiple Merkle tree WOTS$^+$ public keys is facilitated through shared memory buffer utilization, thereby optimizing authentication path construction while significantly reducing memory access latency compared to previous implementations [8].

2) FORS Parallelization: The FORS component implementation employs fine-grained parallelism for generating $k \times 2^a$ secret key elements and leaf nodes concurrently. Multiple Merkle trees are constructed simultaneously, with each of the k trees assigned to independent thread blocks. Shared memory buffers are strategically employed during tree construction to aggregate intermediate roots, significantly reducing global memory access operations during public key derivation. This approach builds upon methodologies established in [9] while enhancing memory utilization patterns for the specific characteristics of the FORS structure.

3) Hypertree Parallelization: For the hypertree construction, parallel processing is applied across multiple Merkle trees spanning all d layers. The leaf nodes are initially generated through parallel hash chain computations before being consolidated into tree structures. Due to shared memory constraints preventing full parallelization of all nodes across all layers, a semi-parallel approach was implemented. This methodology differs from the full-node parallelization presented in [9] by employing a hierarchical execution strategy where each thread processes four-node combinations rather than two-node combinations.

TABLE I
PERFORMANCE COMPARISON OF SLH-DSA IMPLEMENTATIONS

Parameter Sets, Year [Work], Tasks	Latency (ms)			Throughput (tasks/sec)			Device
	KG	Sign	Verify	KG	Sign	Verify	
SHA2-128f, 2024 [8], 512	0.71	11.53	1.79	725,118 (55%)	44,391 (97%)	285,681 (81%)	RTX 3090
SHA2-128f, 2025 [9], 41,984	32.07	924.24	119.16	1,309,136 (100%)	45,425 (100%)	352,333 (100%)	RTX 3090
SHA2-128f, 2025 [9]†, 32,768	22.82	609.03	72.51	1,435,690 (109.7%)	53,804 (118.4%)	451,883 (128.3%)	RTX 4090
SHA2-128f, **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.

IV. PERFORMANCE EVALUATION

A. Experimental Setup

Evaluation was conducted on standardized hardware platforms to facilitate comparative analysis. The experimental configuration comprised the following components: an NVIDIA RTX 4090 GPU with 24GB GDDR6X memory, operating under Ubuntu 24.04 LTS. Compilation was performed using CUDA 12.5 and 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. Comparative Performance Analysis

The implementation was benchmarked against state-of-the-art alternatives using SHA2-128f parameters. Table I 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. For SHA2-128f, key generation, signature generation, and verification latencies were reduced, and corresponding throughput was increased. Specifically, compared to Kim et al. [8], the throughput of our implementation is 2.20× for key generation, 1.41× for signature generation, and 1.76× for verification. Compared to Wang et al. [9], the throughput is 1.11×, 1.16×, and 1.43× for key generation, signature generation, and verification, respectively, on the same hardware (RTX 4090).

C. 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.

The execution time model parameters reveal distinct characteristics across cryptographic operations. The α_i parameter quantifies non-parallelizable sequential overhead, with sign operations exhibiting substantially higher values (e.g., SHA2-128s-sign: 23716.81) than key generation and verification operations. The β_i coefficient represents parallelizable workload components, with higher values indicating greater potential performance gain from increased parallelism. The γ_i parameter measures synchronization and thread management overhead, which increases with thread count.

TABLE II
THREAD MODEL PARAMETERS AND OPTIMAL ALLOCATIONS

Operation	α_i	β_i	γ_i	t_i^*
SHA2-128f-keygen	52.06	506,000.57	1.26E-4	63,310
SHA2-128f-sign	1386.01	13,231,567.75	3.60E-3	60,636
SHA2-128f-verify	164.72	1,395,012.54	4.54E-4	55,407
SHA2-128s-keygen	3317.74	32,046,199.26	7.15E-3	66,929
SHA2-128s-sign	23716.81	248,632,501.64	6.59E-2	61,419
SHA2-128s-verify	63.22	484,914.46	1.44E-4	57,968
SHA2-192f-keygen	79.37	822,859.78	2.40E-4	58,560
SHA2-192f-sign	2319.70	23,961,551.63	8.55E-3	52,932
SHA2-192f-verify	267.63	2,342,878.75	8.91E-4	51,274

According to equation (2), operations with higher β_i/γ_i ratios yield higher optimal thread counts. Experimental validation confirms performance degradation of 18-23% when exceeding optimal thread allocation (t_i^*) and significant latency increases when using insufficient threads. Notable consistency in optimal thread allocation is observed within parameter sets (e.g., 55,407-63,310 for SHA2-128f), while security level transitions show distinct allocation patterns, reflecting fundamental algorithmic characteristics.

D. Function-Level Parallelism Impact

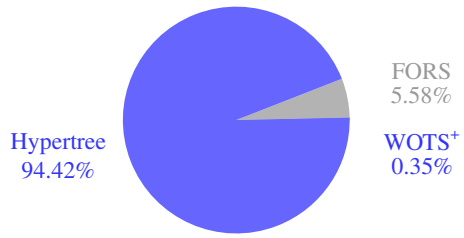

Fig. 3. Signing latency breakdown for SHA2-128f.

Analysis indicates that Hypertree construction dominates signing latency, accounting for 94.42% of total execution time (497.11 ms), while FORS and WOTS$^+$ contribute 5.58% (29.37 ms) and 0.35% (1.857 ms), respectively, as visually depicted in Fig. 3. 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 observation highlights the necessity for targeted optimization of Hypertree structures for future performance enhancements.

E. Scalability Analysis

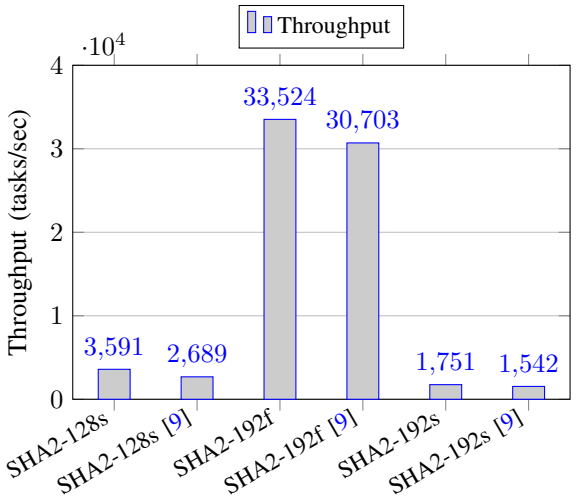

Fig. 4. Throughput across SLH-DSA parameter sets.

Implementation scalability was evaluated across varying security parameters and computational complexities. As shown in Fig. 4, throughput improves for all parameter configurations, with the SHA2-128s parameter set achieving 3,591 tasks/sec and SHA2-192f reaching 33,524 tasks/sec. These results indicate that Thread-Adaptive techniques are most effective for computationally intensive operations with smaller signature sizes, while diminishing returns are observed at higher security levels due to increased computing complexity.

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 SHA2-128f parameters, representing a 1.16× improvement over state-of-the-art implementations. The implementation exhibited particular efficacy for robust parameter variants, with SHA2-128s showing a 1.33× throughput enhancement and 1.25× latency reduction. These results establish GPU platforms as viable acceleration frameworks for post-quantum cryptographic schemes. Empirical analysis identified Hypertree construction as the primary performance bottleneck, accounting for 94% of signing latency despite component-level optimizations. 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] N. I. of Standards and Technology, "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, H. Seo, S. C. Seo, and S. O. Hwang, "Efficient implementation of AES-CTR and AES-ECB on gpus with applications for high-speed frodokem and exhaustive key search," *IEEE Trans. Circuits Syst. II Express Briefs*, vol. 69, no. 6, pp. 2962–2966, 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+," *IEEE Transactions on Computers*, vol. 74, no. 1, pp. 15–28, 2025.