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

Jiahao Xiang and Lang Li.

***Abstract*—The Post-Quantum Cryptography (PQC) standardization process has led to the development of the stateless hash-based digital signature algorithm (SLH-DSA) FIPS 205. The high computational cost of SLH-DSA necessitates efficient implementations for practical deployment. This work presents a GPU-based implementation of SLH-DSA that achieves high throughput through a thread-adaptive parallelization strategy. Unlike conventional approaches that allocate maximum thread counts, the implementation dynamically determines optimal parallelism levels for each cryptographic kernel function, balancing thread utilization with execution efficiency. Additionally, fine-grained decomposition of signature components enables more efficient thread-level execution. Performance evaluation on an NVIDIA RTX 4090 GPU demonstrates the implementation achieves a throughput of XXX signatures per second, significantly outperforming existing approaches. The results establish GPUs as effective platforms for accelerating SLH-DSA operations in high-throughput environments, facilitating practical transition to post-quantum standards.**

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

I. Introduction

THE quantum computers leverage quantum-mechanical phenomena to process data, raising significant concerns about the resilience of classical cryptographic methods. The security offered by widely deployed public-key cryptosystems, such as RSA and ECC, is jeopardized by Shor's algorithm [1], motivating comprehensive research on alternative cryptographic solutions. In response, the National Institute of Standards and Technology (NIST) initiated the Post-Quantum Cryptography (PQC) standardization process to develop novel schemes that withstand quantum computing capabilities [2].

SPHINCS$^+$ is a representative stateless hash-based signature scheme and a finalist in the ongoing NIST standardization effort [3]. Long-term security against advanced quantum attacks is targeted by employing robust cryptographic hash functions [4]. The high computational cost of SPHINCS$^+$ has motivated further investigations into efficient implementations across CPUs, FPGAs, and GPUs [5] to facilitate smooth adoption by organizations transitioning to post-quantum cryptography.

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)

A. Related Work

Significant progress has been made in GPU-based implementations of SPHINCS$^+$. Lee and Hwang [6] pioneered the exploration of GPU acceleration for post-quantum cryptographic schemes, establishing foundational techniques for parallel implementation of hash-based signatures. Building upon this foundation, Kim et al. [7] introduced parallel methods for key components of SPHINCS$^+$, including FORS, WOTS$^+$, and MSS tree computations. Their implementation on an RTX 3090 GPU demonstrated significant throughput improvements, though it faced efficiency limitations due to multiple CUDA kernel launches.

Most recently, Wang et al. [8] presented CUSPX, introducing a comprehensive three-level parallelism framework that integrates algorithmic, data, and hybrid parallelization strategies. Their implementation incorporated novel parallel Merkle tree construction algorithms and multiple load-balancing approaches, achieving substantial performance improvements over previous implementations. Additionally, Ning et al. [9] proposed GRASP, which further optimized GPU-based SPHINCS$^+$ implementation through adaptive parallelization strategies and kernel fusion technology.

B. Motivation

While previous implementations have made significant strides in GPU acceleration of SPHINCS$^+$, several critical limitations remain unaddressed. Existing approaches predominantly focus on maximizing throughput through extensive parallelization, often overlooking the efficiency of individual thread execution. The implementation by Kim et al. [7] demonstrated the potential of parallel processing but suffered from inefficiencies due to multiple kernel launches. Although CUSPX [8] introduced a comprehensive parallelization framework, its approach to thread utilization and resource management could be further optimized.

Two key observations motivate our work. First, current implementations typically concentrate on parallelizing the SPHINCS$^+$ algorithm structure while paying insufficient attention to the optimization of underlying hash functions, which constitute the computational core of the scheme. A more holistic approach that addresses both algorithmic levels could yield substantial performance improvements. Second, existing implementations often prioritize maximum thread parallelism

without adequately considering the trade-off between thread count and execution efficiency. This frequently leads to suboptimal performance due to increased synchronization overhead, memory access latency, and reduced computational efficiency per thread.

These observations indicate the need for a more balanced approach that optimizes both the degree of parallelism and the computational efficiency of individual threads. Our work addresses these limitations by developing an implementation that not only leverages GPU parallelism effectively but also ensures efficient utilization of computational resources through careful thread allocation and optimization of the underlying hash functions. By focusing on both algorithmic structure and computational primitives, our implementation achieves superior performance.

C. Contributions

In this brief, an optimized GPU-based implementation of SPHINCS$^+$ is presented, achieving high throughput and low latency. The main contributions are summarized as follows:

1) A hash-function-level parallelization approach is introduced that reduces latency through fine-grained task distribution, significantly accelerating the core computational primitives of SPHINCS$^+$.
2) An adaptive thread allocation strategy is developed that optimizes the balance between thread count and kernel function efficiency, minimizing synchronization overhead while maximizing computational throughput on GPU architectures.
3) The implementation is evaluated on an NVIDIA GPU, demonstrating a throughput of XXX SPHINCS$^+$ signatures per second, significantly exceeding the performance of state-of-the-art approaches. The complete source code and implementation details are available at https://github.com/jiahaoxiang2000/sphincs-plus.

The remainder of the brief is organized as follows. Section II provides an overview of the SPHINCS$^+$ signature scheme; Section III details the GPU-based implementation; Section IV presents the performance evaluation; and Section V concludes the brief.

II. Preliminaries

A. SPHINCS$^+$ Overview

SPHINCS$^+$ is a stateless hash-based signature scheme that delivers post-quantum security through a hierarchical certification structure. The signature generation process relies on three main components:

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

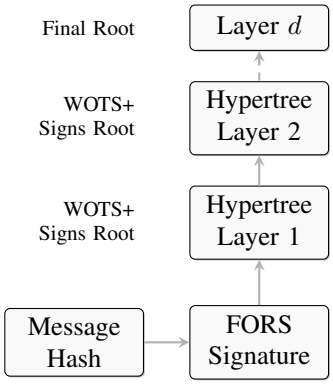

Fig. 1. SPHINCS$^+$ signature generation flow. A message hash is signed by FORS to produce k authentication paths, which are then authenticated by a d-layer hypertree. Each layer employs WOTS+ to sign the root of the previous layer, culminating in a final root signature.

The SPHINCS$^+$ signature generation process, shown in Figure 1, employs a hierarchical authentication structure. A message digest is first created via hashing, followed by signing with the FORS few-time scheme, producing k authentication paths of t elements each. The resulting FORS public key is authenticated through a hypertree of d layers, where each layer uses WOTS+ to sign the root of the layer below. This chain of signatures leads to the final root node, offering efficient verification with robust hash-based security.

Two operational modes, "simple" and "robust," are provided to balance speed and security. Parameter sets facilitate trade-offs among signature size, security level, and computational efficiency. All security properties derive from the hash functions, rendering SPHINCS$^+$ resistant to quantum attacks.

B. GPU Computing Model

Modern Graphics Processing Units (GPUs) incorporate a large number of cores organized within multiple Streaming Multiprocessors (SMs). This highly parallel structure supports Single Instruction, Multiple Thread (SIMT) execution, wherein threads are grouped into warps, and warps collectively form blocks. Each block is then scheduled across available SMs, ensuring that thousands of concurrent threads can execute similar instructions in parallel.

In the CUDA framework, memory optimization strategies such as coalesced accesses, shared memory buffering, and constant memory utilization further enhance throughput. Extensive parallelization of SPHINCS$^+$ computations is therefore facilitated, allowing performance improvements through a combination of thread-level, data-level, and algorithmic parallelism.

III. Optimized Implementation of SPHINCS$^+$

A. Hash-Function-Level Parallelization

The 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 operations into parallel tasks distributed across multiple threads.

1) Internal State Parallelism: The approach decomposes hash function operations into concurrent tasks that can be executed in parallel by multiple threads within a single warp. For SHA256, the internal state transformations are parallelized as follows:

- **State Initialization**: Multiple threads 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, reducing initialization overhead.
- **Round Functions**: 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.
- **Data Sharing**: 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 efficiency.

2) Task Distribution : A hierarchical task allocation scheme is implemented to efficiently distribute hash function operations across GPU threads:

Algorithm 1 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

The task distribution strategy is optimized for SHA256 operations within SPHINCS$^+$, where hash function calls account for 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
- At the grid level, optimal workload distribution is maintained with 128 blocks of 256 threads per block

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.

B. Adaptive Thread Allocation Strategy

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] 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¡sup¿+¡/sup¿," *IEEE Transactions on Computers*, vol. 74, no. 1, pp. 15–28, 2025.

[9] Y. Ning, J. Dong, J. Lin, F. Zheng, Y. Fu, Z. Dong, and F. Xiao, "GRASP: Accelerating hash-based PQC performance on GPU parallel architecture," Cryptology ePrint Archive, Paper 2024/1030, 2024. [Online]. Available: https://eprint.iacr.org/2024/1030