# 周报-向嘉豪 (2025 年 3 月 3 日)

**摘要**：本周主要工作为 SHA256 哈希算法的 GPU 并行实现与优化。通过线程束级别的细粒度并行方法，实现了哈希函数内部状态的高效并行处理。并进行了系统性能评估。测试结果显示，优化后的 SHA256 实现在数据处理时可达 160MB/s 的吞吐量，相比现有实现提升约 6 倍。此外，完成了论文中研究动机、主要贡献和哈希函数级并行化实现部分的修订与完善。

**下周计划**：1) 基于优化后的 SHA256 实现，完成 SPHINCS+ 签名方案中 WOTS+ 和 FORS 组件的 GPU 加速实现，着重对比密钥生成阶段；2) 设计并实现自适应线程分配策略，针对不同参数集优化计算资源分配。

## 1 SHA256 并行实现

本周主要工作为在 GPU 上实现 SHA256 哈希算法的高效并行计算。SHA256 作为 SPHINCS+ 签名方案的基础组件，其性能优化对整体签名速度至关重要。

### 1.1 线程束级并行实现

通过线程束（warp）级并行实现单个哈希计算，线程布局设计如下：线程 0 负责状态初始化工作；线程 0-15 协同并行加载消息字；线程 0-15 共同处理消息调度扩展；线程 0-7 分别管理轮计算中的不同状态变量；最后由线程 0 处理填充和最终输出操作。通过这种精细的任务分配，我们充分利用了线程束内的并行性，同时避免了不必要的线程间通信开销。

主要优势在于：更好地利用单个哈希操作的资源，通过同步操作减少线程束分化，并使用 `__shfl_sync()` 等线程束级原语实现高效的数据共享。测试表明在处理大量消息时能达到约 120MB/s 的吞吐量，相比 [WDC+25] 有6 倍的提升。

## 2 论文写作

### 2.1 研究动机更新

对论文第 1.2 节中的研究动机进行了更新和完善，主要从以下两个角度强化了我们工作的必要性：首先，指出现有实现主要关注于通过广泛并行化来最大化吞吐量，但往往忽视了单个线程执行的效率。Kim 等人的实现证明了并行处理的潜力，但由于多次内核启动而效率低下，而 Wang 等人的 CUSPX 虽然引入了全面的并行框架，但其线程利用和资源管理仍有优化空间。

其次，提出了驱动本研究的两个关键观察：（1）现有实现通常集中于并行化 SPHINCS+ 算法结构，而对构成该方案计算核心的底层哈希函数优化关注不足；（2）现有实现往往优先考虑最大线程并行性，而没有充分考虑线程数量和执行效率之间的权衡，导致由于同步开销增加、内存访问延迟和单线程计算效率降低而性能不佳。

### 2.2 主要贡献修订

修订了论文第 1.3 节中的前两点贡献，使其更加明确和具体：第一点贡献强调了我们提出的哈希函数级并行方法，通过细粒度任务分配减少延迟，显著加速了 SPHINCS+ 的核心计算原语。这一贡献直接针对当前 SPHINCS+ 实现中哈希函数处理效率低下的问题。

第二点贡献阐述了我们开发的自适应线程分配策略，优化了线程数量和内核函数效率之间的平衡，在 GPU 架构上最小化同步开销的同时最大化计算吞吐量。这一贡献解决了现有实现中对线程资源使用不当导致的性能问题。

### 2.3 哈希函数级并行化实现

编写并完善了第 3.1 节” 哈希函数级并行化” 的详细内容，这是本文的核心技术创新之一。该部分主要包括：我们的哈希函数级并行化主要从三个方面进行优化。首先是状态初始化优化，多个线程同时初始化哈希函数状态数组的不同部分，减少初始化开销。具体来说，线程 0 处理初始状态设置，线程 0-15 协作并行加载消息字。其次是轮函数优化，SHA256 置换的每一轮被分解为可由不同线程并发执行的通道操作。线程 0-15 处理消息调度扩展，线程 0-7 管理轮计算中的状态变量更新。最后是数据共享优化，通过 warp 级原语（如 `__shfl_sync()`）实现高效数据共享，无需昂贵的共享内存操作。同步操作确保 warp 内的线程可以在没有 warp 分化的情况下交换数据，提高计算效率。

## 参考文献

[WDC+25] Ziheng Wang, Xiaoshe Dong, Heng Chen, Yan Kang, and Qiang Wang. Cuspx: Efficient gpu implementations of post-quantum signature sphincs<sup>+</sup>. *IEEE Transactions on Computers*, 74(1):15–28, 2025.