# 基于国产GPU的国产公钥密码SM2高性能并行加速方法

吴雯[1]，董建阔[1]，刘鹏博[1]，董振江[1]，胡昕[1,2]，张品昌[1]，肖甫[1]
（1.南京邮电大学计算机学院，江苏 南京 210023；2.江苏省公用信息有限公司，江苏 南京 210003）

**摘　要：**为了满足国家信息安全自主可控的战略需求，确保算法的透明性和安全性，提出基于国产GPU的国产公钥密码SM2数字签名算法的高性能并行加速方法。首先，设计适用于域运算的底层函数，优化有限域运算的效率，约减采用2轮进位消解以抵御计时攻击。其次，基于雅可比（Jacobian）坐标实现点加和倍点运算，充分利用寄存器和全局内存的特性，设计离线/在线预计算表以提高点乘计算效率。最后，根据海光深度计算单元（DCU）的特点进行实验设计，实现高性能的SM2签名和验签算法，分别达到6 816 kops/s的签名吞吐量和1 385 kops/s的验签吞吐量。研究验证了基于国产GPU的国产公钥密码SM2数字签名算法的可行性和有效性，为国内信息安全自主可控领域提供了重要的技术支持。

**关键词：**国家商用密码；数字签名；图形处理器；异构计算
**中图分类号：**TP309.7
**文献标志码：**A
**DOI：**10.11959/j.issn.1000−436x.2025089

# High-performance parallel acceleration method for domestic public key cryptographic algorithm SM2 based on domestic GPU

WU Wen[1], DONG Jiankuo[1], LIU Pengbo[1], DONG Zhenjiang[1], HU Xin[1,2], ZHANG Pinchang[1], XIAO Fu[1]
1. School of Computer Science, Nanjing University of Posts and Telecommunications, Nanjing 210023, China
2. Jiangsu Public Information Co., Ltd., Nanjing 210003, China

**Abstract:** To meet the strategic demand for independently controllable national information security and to ensure algorithm transparency and security, a high-performance parallel acceleration method for the domestic public key cryptographic SM2 digital signature algorithm based on domestic GPU was proposed. Firstly, low-level functions for field operations were designed to optimize the efficiency of finite field arithmetic, with two-round carry resolution adopted in modular reduction to resist timing attacks. Then, point addition and point doubling were implemented using Jacobian coordinates, leveraging register and global memory features, and offline/online precomputation tables were introduced to enhance scalar multiplication efficiency. Finally, experimental design was carried out based on the characteristics of the Hygon deep compute unit (DCU) platforms, achieving high-performance SM2 signature and verification with throughput of 6 816 kops/s and 1 385 kops/s, respectively. Results demonstrate the feasibility and effectiveness of implementing clemestic public key cryptographic SM2 based on domestic GPU, providing crucial support for national information security autonomy.

**Keywords:** state commercial cryptography, digital signature, GPU, heterogeneous computing

**收稿日期：**2025−02−07；**修回日期：**2025−04−30
**通信作者：**董建阔，djiankuo@njupt.edu.cn
**基金项目：**江苏省重点研发计划基金资助项目（No.BE2023025）；国家自然科学基金资助项目（No.62302238, No.62372245, No.62172258）；江苏省自然科学基金资助项目（No.BK20220388）；江苏省科技厅重点研发计划产业前瞻与关键核心技术基金资助项目（No.BE2022081）；CCF-腾讯犀牛鸟科研基金资助项目（No.CCF-Tencent RAGR20240129）
**Foundation Items:** The Key Research and Development Program of Jiangsu Province (No.BE2023025), The National Natural Science Foundation of China (No. 62302238, No. 62372245, No. 62172258), The Natural Science Foundation of Jiangsu Province (No.BK20220388), The Major Science and Technology Demonstration Project of Jiangsu Provincial Key Research and Development Program (No.BE2022081), The CCF-Tencent Rhino-Bird Open Research Fund (No.CCF-Tencent RAGR20240129)

## 0 引言

近年来，随着全球科技竞争的加剧，信息技术核心软硬件已逐渐演变为科技竞争和战略博弈的关键焦点。我国在获取高端计算芯片方面正面临日益严峻的技术封锁与出口限制。2023年，美国商务部工业和安全局实施新的出口限制规定，其中限制中国购买高端计算芯片及采用先进工艺的半导体设备[1]，禁售产品名录涵盖NVIDIA、AMD和Intel的多款芯片。此举措意在阻碍我国获取可能促进高性能计算领域突破性进展的先进半导体技术。

在此背景下，国产高性能计算芯片作为支撑国家信息技术自主可控战略的关键力量[2]，承载着突破“卡脖子”技术的重大使命。作为国内高端芯片设计的领军企业，海光公司的异构加速芯片深度计算单元（DCU, deep compute unit）已在国内多个计算中心得到部署应用，广泛应用于云端训练、推理等多个方向，是国内覆盖应用范围较广的一款通用异构加速芯片。作为高性能协处理器，DCU能够有效处理大量的通用计算任务，在大规模计算领域表现出明显的优势[3]，也成为在高并发环境下实现密码算法高性能加速的重要平台之一。

此外，信息安全事件频发也凸显了依赖国外密码算法可能带来的安全风险和可信度问题。美国国家标准与技术研究院（NIST, National Institute of Standards and Technology）制定的椭圆曲线家族[4]中的“双椭圆曲线确定性随机数发生器”（Dual_EC_DRBG, dual elliptic curve deterministic random bit generator）存在严重后门，引发全球对密码算法安全性的广泛质疑。因此，研究国产密码算法不仅可以有效避免外部潜在的安全威胁，还能够确保算法的透明性和安全性，符合国家信息安全自主可控的战略需求。中国国家密码管理局于2010年出台基于椭圆曲线密码（ECC, elliptic curve cryptography）算法的非对称密码方案——SM2数字签名算法[5]，已于2017年成为现行国家标准，同年成为国际标准[6]。SM2数字签名算法广泛应用于身份认证、电子政务、金融通信等[7]多个关键领域，在国家信息基础设施中发挥着重要作用。

然而，由于SM2数字签名算法包含复杂且计算量大的椭圆曲线运算，其密钥操作在计算成本上较对称加密算法更高[8]，实现高效可靠的SM2数字签名算法仍面临较大挑战。然而，现有SM2数字签名算法的高性能实现工作多依赖于英伟达图形处理单元（NVIDIA GPU, NVIDIA graphics processing unit）[9]。文献[10]基于开放计算语言（OpenCL, open computing language）编程框架采用“CPU-GPU”异构编程模型实现了AMD Radeon R9 290 GPU平台上的SM2并行算法，通过设计并优化有限域运算以及计算资源分配提高效率。文献[11]在GPU平台和现场可编程门阵列（FPGA, field programmable gate array）平台分别实现了SM2数字签名算法，在NVIDIA GTX 1080平台中分别从有限域运算、群运算、点乘运算进行优化，其中，有限域中采用蒙哥马利模乘、扩展欧几里得等加速域运算，群运算中采用雅可比（Jacobian）坐标形式进行计算以避免耗时的模逆运算，点乘中分别采用预计算表和非相邻形式（NAF, non-adjacent form）编码优化已知点点乘和未知点点乘。文献[12]基于NVIDIA RTX 3090使用计算统一设备架构（CUDA, compute unified device architecture）实现，并利用并行线程执行（PTX, parallel thread execution）指令优化域运算，采用费马小定理优化模逆运算，同时为避免线程束分叉按需计算点加和倍点，有效减少未知点点乘运算的计算量。

SM2数字签名算法中开销最大的为点乘运算，已有多项工作研究基于GPU实现并优化点乘运算。文献[13]中利用已知点保持不变的特性，为已知点点乘构建了64 MB大小的预计算表并将其存储在GPU的全局内存中，通过查表减少点运算次数以提高计算性能，但未知点点乘无法提前构建离线预计算表。文献[14]基于NVIDIA TITAN V的浮点数计算能力加速ECC算法，为抵御计时攻击使用窗口长度为4的固定窗口算法为每个线程生成在线预计算表，从而仅需要78个点加和252个倍点运算完成已知点点乘。点乘可以转换成点加和倍点运算，文献[10-14]中均采用Jacobian坐标计算以减少模逆运算带来的开销。

以上工作虽在性能上取得一定提升，但在当前技术封锁加剧的形势下，难以满足自主可控的安全需求。因此，基于国产GPU平台开展SM2数

字签名算法加速研究具有重要现实意义。已有研究初步验证了国产平台的可行性，文献[15]在我国自主研发的“嵩山”超级计算机的单个节点上，基于可移植异构计算接口（HIP）实现并验证了国家商用密码 SM2 加解密算法和 SM3 哈希算法在国产设备上的异构计算的可行性，计算效率相比串行实现提升数倍。

本文基于国产海光 DCU 对国家商用密码 SM2 数字签名算法高性能并行计算进行研究，旨在实现具有高吞吐量、高扩展性和高灵活性的算法优化方案，对完善国家商用密码生态、提升自主安全能力具有重要意义，具体贡献如下。

1) 本文构建了适用于域运算的底层函数，利用该函数实现高效的模加、模减、模乘、模平方运算，优化计算次序以尽可能地减少临时变量的占用，并且分析了约减的计算过程，采用 2 轮进位消解以抵御计时攻击。

2) 基于 Jacobian 坐标设计并实现点加和倍点运算，充分利用寄存器高速存储和快速访问的特性；根据 DCU 具有大量全局内存的特性，分别为已知点点乘和未知点点乘设计了离线/在线预计算表，提高了点乘的计算效率。

3) 本文根据 DCU 特点设计实验和配置参数，达到最佳实验效果。最终，签名/验签吞吐量分别达到 6 816 kops/s 和 1 385 kops/s。通过对比分析，本文基于海光 DCU 实现的 SM2 数字签名算法的性能处于相对领先的地位。

表 1 汇总了包括本文在内的代表性异构 GPU 平台上 ECC 算法的加速实现情况，对比展示了各方案在平台选型、关键优化方案和安全性分析方面的差异。

## 1　预备知识

本节主要介绍 SM2 数字签名算法的基本原理、国产 DCU 硬件架构的关键特性，以及 HIP 异构编程模型的实现机制。

### 1.1　SM2 数字签名算法

SM2 数字签名算法由密钥生成、签名和验签 3 个部分组成，系统参数包括有限域 $\mathbb{F}_q$ 的规模 $q = 2^{256} - 2^{224} - 2^{96} + 2^{64} - 1$，椭圆曲线 $y^2 = x^3 + ax + b$ 的参数 $a$ 与 $b$，椭圆曲线上的非无穷远基点 $G(x_g,y_g)$ 及其阶 $n$。密钥生成算法负责生成用户的公私钥对 $(p_k,d_k)$，其中私钥 $d_k$ 用于数据签名，公钥 $p_k = d_kG = (x_k,y_k)$ 是椭圆曲线上的点，用于验证签名。SM2 签名和验签流程分别如算法 1 和算法 2 所示[16]。

**算法 1**　SM2 签名算法

**输入**　待签名消息 $M$，私钥 $d_k$

**输出**　签名 $(r,s)$

1) 生成随机数 $k \in [1,n-1]$；

2) 计算椭圆曲线上的点 $(x,y) = kG$；

3) 计算 $r = [\,\mathrm{hash}(M) + x\,] \bmod n$，若 $r = 0$ 或 $r + k = n$，则返回步骤 1)；

4) 计算 $s = [(d_k + 1)^{-1}(k - rd_k)] \bmod n$，若 $s = 0$，则返回步骤 1)；

5) 返回 $(r,s)$。

**算法 2**　SM2 验签算法

**输入**　消息 $M$，公钥 $p_k$，待验证签名 $(r,s)$

**输出**　验签结果是否通过

1) 计算 $t = (r + s) \bmod n$，若 $t = 0$，验签不通过；

2) 计算 $(\hat{x},\hat{y}) = sG + tp_k$；

**表 1**　与现有工作的差异分析

| 方案 | 平台 | 有限域 | 标量乘 | 安全性分析 |
|---|---|---|---|---|
| 文献[10] | AMD GPU (OpenCL) | 蒙哥马利模乘、费马小定理 | 梳状法、二进制展开法 | × |
| 文献[11] | GTX 1080 (CUDA) | 蒙哥马利模乘、拓展欧几里得 | 预计算表、NAF 算法 | × |
| 文献[12] | RTX 3090 (CUDA) | PTX 汇编指令、费马小定理 | 窗口法、构造预计算表 | × |
| 文献[14] | TITAN V (CUDA) | 浮点数实现、积和融加指令 | 分段算法、固定窗口法 | 仅侧信道理论分析 |
| 文献[15] | 海光 DCU (HIP) | 蒙哥马利模乘 | 宽度 $w$ 的 NAF 算法 | × |
| 本文 | 海光 DCU (HIP) | 构建底层函数、2 轮进位消解 | 分段算法、固定窗口法 | 多角度理论分析 |

3) 计算 $\hat{r}=[\mathrm{hash}(M)+\hat{x}]\bmod n$，若 $r=\hat{r}$，则验签通过，否则验签不通过。

在SM2数字签名算法中，需要对椭圆曲线上的点进行点乘运算，点乘可以分解为点加和倍点运算，而点加和倍点运算依赖于有限域上大整数的模加、模减、模乘、约减、模平方、模逆运算，如图1所示。

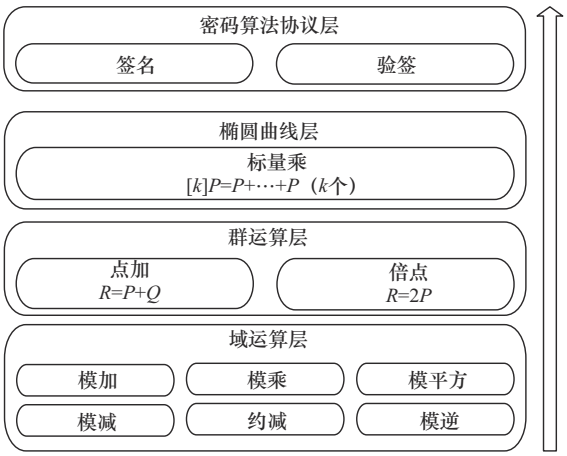



图1　SM2数字签名算法运算层次

## 1.2　DCU

DCU作为我国自主研发的新一代类GPU加速器[17]，现已部署在国内多个异构计算平台中，用于提供高效稳定的支撑，特别适用于数据密集型计算应用。DCU采用7 nm制造工艺，通过2.5 D中介层的系统级芯片（SoC）技术封装，配备高带宽存储器（HBM2, high bandwidth memory 2）片上高速内存，内存带宽可达1 TB/s，单精度峰值性能达13.3 TFLOPS，双精度峰值性能达6.5 TFLOPS。

计算单元（CU, compute unit）[18]是DCU加速器件的基本运算单元，每片DCU包含多个CU，每个CU由4个单指令多数据流（SIMD, single instruction multiple data）计算组和1个线程束调度器构成。每个SIMD计算组包含16 384个32 bit寄存器。线程束在SIMD上执行，单个SIMD最多支持64个线程并发执行，每个线程可使用最多256个寄存器资源。CU还配备了64 KB的本地数据共享（LDS, local data shared）内存，便于同一线程组中的线程进行数据交换，但不同CU之间的LDS内存相互独立。DCU架构如图2所示，任务管理器负责任务分发，指令处理器进行全局指令调度，其大量的计算单元和少量的指令调度部件，使其特别适合处理大规模计算密集型程序[19]。

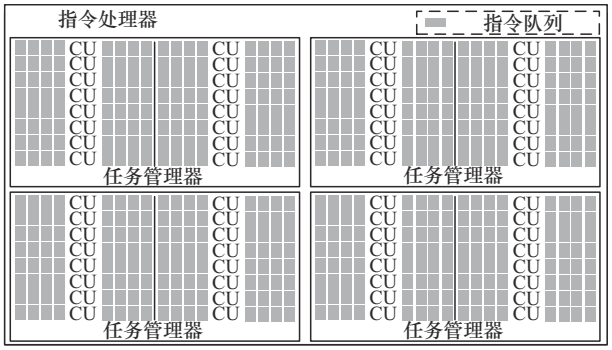



图2　DCU架构

## 1.3　HIP异构编程模型

国产DCU加速器采用HIP异构编程模型，在该模型中，用户使用HIP C/C++进行开发，并通过平台提供的编译工具Hipcc进行编译。HIP异构编程模型提供完善的运行时接口（Hip_runtime API），供主机端调用以实现内存分配、数据传输、核函数启动和内存释放等功能。整个HIP程序由主机端的串行执行和设备端的并行执行构成，主机端代码使用C/C++编写并在CPU上运行，而设备端代码由HIP C编写，并作为核（kernel）函数在DCU上运行。当设备端并行操作完成后，主机端继续执行串行代码，主机端和设备端之间的数据传输通过Hip-Memcpy()显式完成。

HIP异构编程模型中的线程组织方式按照三级结构进行：线程（Thread）、线程块（Block）和线程网格（Grid），如图3所示。一个线程网格对应单个核函数启动时的所有线程。线程块和线程网格可以是一维、二维或三维的[20]，该维度通过Hip-BlockDim和HipGridDim设置，每个执行内核的线程都有唯一的线程标识符（ID, identifier）[21]，用HipThreadIdx标识。此外，HIP异构编程模型具有层次性特征，线程可以从多层次存储空间进行数据访问。每个线程拥有私有本地内存，用于存储核函数中的溢出变量；线程块内所有线程共享片上高速缓存，以高带宽隐藏内存时延、提高访存速度；全局内存则是所有线程可访问的非片上内存，通过__global__关键字声明，最理想的访问模式是线程束中的线程访问连续内存块。

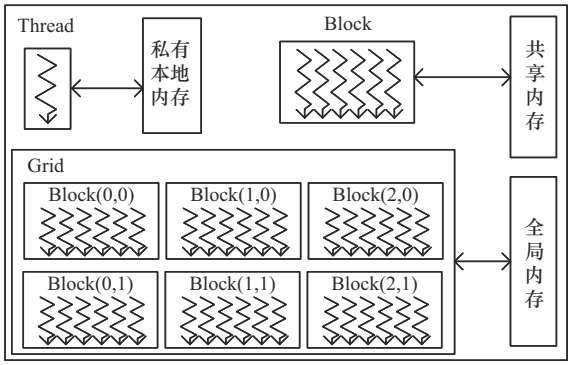



图3　HIP 异构编程模型

## 2　SM2 数字签名算法实现

本节首先提出“CPU-DCU”异构编程模型实现基于海光 DCU 的 SM2 数字签名/验签算法；其次详细介绍有限域运算的实现，具体包含大整数加法、减法、乘法、平方、约减；然后实现了 Jacobian 坐标下的点加和倍点算法；最后使用固定窗口方法计算未知点点乘，使用离线预计算表辅助计算已知点点乘，最终在海光 DCU 上完整实现了 SM2 数字签名算法。

### 2.1　整体框架

本文提出的“CPU-DCU”异构编程模型如图 4 所示，展示了基于海光 DCU 的 SM2 数字签名算法的整体实现框架。在海光 DCU 平台采用核融合方案，使用单个核函数完整实现 SM2 数字签名算法，显著减少计算过程中访问全局内存的次数，提高了算法的整体性能。由于 SM2 数字签名使用的比特长度较小，使用的寄存器资源较少，适合单个线程完成完整的签名/验签操作。

在该异构编程模型中，CPU 部分称为主机端（host），DCU 部分称为设备端（device），设备端作为主机端的协处理器。首先主机端根据需要处理的待签名/验签消息的数量和大小，使用 HipMalloc 函数为设备端分配足够大的全局内存空间；然后使用 HipMemcpy 函数将需要处理的消息传输至设备端的全局内存中；接着设备端启动核函数 kernel()开始并行执行 SM2 签名/验签算法，每个线程执行一个操作；当设备端执行完毕之后使用 HipMemcpy()将各个线程的签名/验签结果传输回主机端。本文提出的“CPU-DCU”异构编程模型具有高度的通用性和可扩展性。

该异构模型中，合理利用 DCU 的存储资源是提升签名/验签性能的重要因素。SM2 数字签名算法的点乘涉及已知点点乘和未知点点乘。对于已知点点乘，本文采用离线预计算表辅助计算，将离线预计算表存储在 DCU 的全局内存中，每个线程在计算已知点点乘时均通过读离线预计算表的方式减少点加和倍点的计算数量，从而提高点乘效率。对于未知点点乘，选择窗口大小为 4 的固定窗口算法生成在线预计算表辅助计算，通过实验分析，该固定窗口算法生成的在线预计算表大小为 1.5 KB，将其存放在临时内存中带来的性能提升相对更多，因为频繁的访问全局内存会带来较大的性能开销。

### 2.2　有限域运算

本文将有限域上 256 bit 的大整数 $A$ 拆分成 8 个 32 bit 的单字表示，即 $A[0:7]=\sum_{i=0}^{7}a_i2^{32i}$。考虑到 DCU 指令集的生态不及英伟达 CUDA 的指令集，为简化开发过程，本文使用自定义底层函数代替指令开发有限域运算，如表 2 所示。

#### 2.2.1　*加法与减法*

如图5所示，本文以“5字+5字”为例说明大整数加法的实现流程。在 SM2 有限域加法中，本文使用

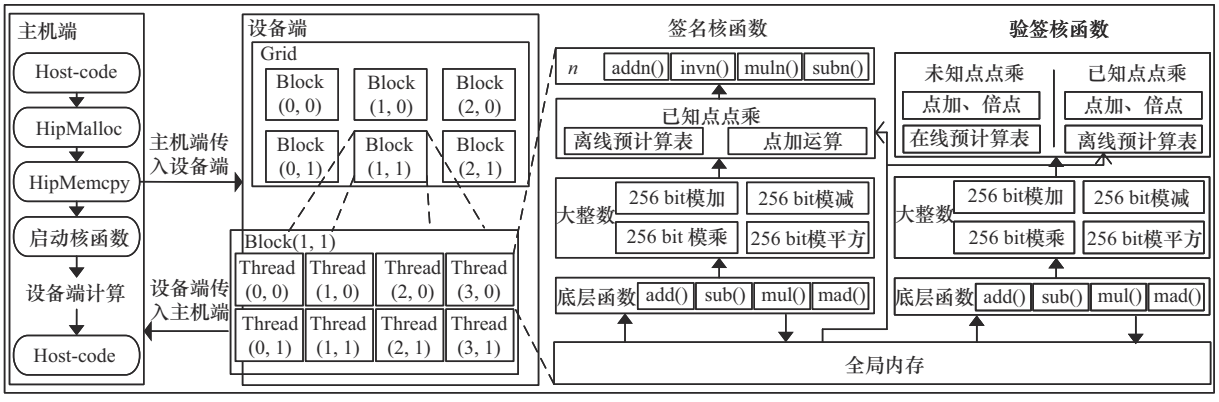



图4　“CPU-DCU”异构编程模型

1个add_cc($a$,$b$,CF,$c$)函数和7个addc_cc($a$,$b$,CF,$c$)函数实现大整数加法。

**表2　　自定义底层函数释义**

| 类别 | 具体函数 | 函数释义 |
|---|---|---|
| 加法 | add($a$,$b$,$c$) | $c = a + b$ |
| | add_cc($a$,$b$,CF,$c$) | $c = a + b$ |
| | addc($a$,$b$,CF,$c$) | $c = a + b + \mathrm{CF}$ |
| | addc_cc($a$,$b$,CF,$c$) | $c = a + b + \mathrm{CF}$ |
| 减法 | sub($a$,$b$,$c$) | $c = a - b$ |
| | sub_cc($a$,$b$,CF,$c$) | $c = a - b$ |
| | subc($a$,$b$,CF,$c$) | $c = a - b - \mathrm{CF}$ |
| | subc_cc($a$,$b$,CF,$c$) | $c = a - b - \mathrm{CF}$ |
| 乘法 | mul($a$,$b$,$c$,$d$) | $c = (a \times b)_{\mathrm{hi}}$ $d = (a \times b)_{\mathrm{lo}}$ |
| 乘加 | mad_lo_cc($a$,$b$,$c$,CF,$d$) | $d = (a \times b)_{\mathrm{lo}} + c$ |
| | madc_lo_cc($a$,$b$,$c$,CF,$d$) | $d = (a \times b)_{\mathrm{lo}} + c + \mathrm{CF}$ |
| | madc_hi($a$,$b$,$c$,CF,$d$) | $d = (a \times b)_{\mathrm{hi}} + c + \mathrm{CF}$ |
| | madc_hi_cc($a$,$b$,$c$,CF,$d$) | $d = (a \times b)_{\mathrm{hi}} + c + \mathrm{CF}$ |

注：函数若有后缀_cc，产生的进位将被写入进位寄存器(CF)。

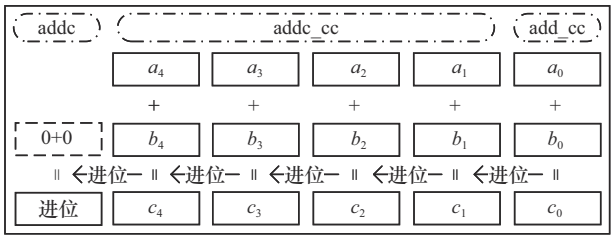



图5　大整数加法的流程实例

需要注意的是，大整数加法可能会产生进位，因此额外使用1个addc($a$,$b$,CF,$c$)函数处理进位，其中进位的数值不大于1。进位的约减方法将在下文详细描述。大整数减法的流程与加法的非常类似，本文使用自定义的减法函数实现，区别在于减法需要进一步处理产生的借位。

### 2.2.2　乘法

本文使用$A[0{:}7]=\sum_{i=0}^{7}a_i\cdot 2^{32i}$表示被乘数，$B[0{:}7]=\sum_{i=0}^{7}b_i\cdot 2^{32i}$表示乘数，$C[0{:}15]=\sum_{i=0}^{15}c_i\cdot 2^{32i}$表示乘积。如图6所示，本文以“3字×3字”为例说明大整数乘法的实现流程，每个块代表2个单字乘积的高/低位子积。为减少临时变量消耗寄存器资源，本文首先调整计算次序，将每行统一调整为高/低位子积行以便进行累加计算，并采用自上而下、自右向左的规则将每个计算块的结果累加至$C[0{:}5]$，以最后两行为例，展示累加的具体步骤。

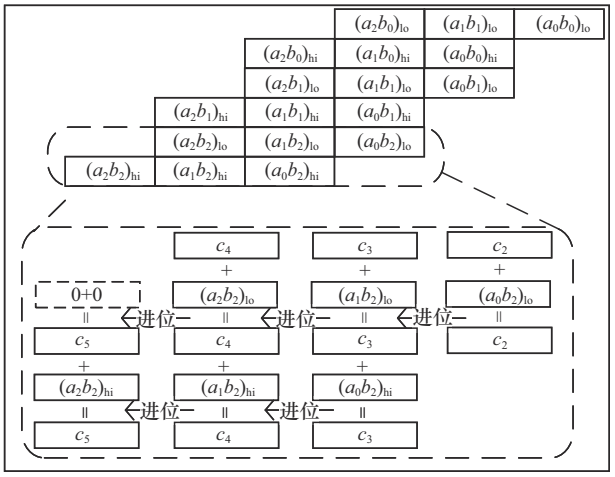



图6　大整数乘法的流程实例

低位子积所在行的累加过程采用madc_lo实现，该行中的每个块将低位子积与上一个块计算产生的进位累加到$C[0{:}5]$对应的单字中，然后将产生的进位存放在进位寄存器中，最后通过addc将进位累加到$C[0{:}5]$对应的单字中。而高位子积所在行的累加过程采用madc_hi实现，具体步骤与上述步骤类似，区别在于该行的最后一个块不会产生进位。具体实现如算法3所示。下面简单证明高位子积行的最后一个块不会产生进位。

**证明**　假设$A[0{:}m-1]$和$B[0{:}n-1]$分别为被乘数和乘数，乘积结果为$C[0{:}m+n-1]$。按照图6中的流程，当第$t$($0\leqslant t\leqslant n-1$)高位子积行完成计算时，临时结果是$m$字整数与$t$字整数的乘积$(2^{mr}-1)(2^{tr}-1)=2^{tr+mr}-2^{tr}-2^{mr}+1\leqslant 2^{tr+mr}-1$，这意味着该临时结果只会保存在$C[0{:}m+n-1]$中，而高位子积行的最后一个块累加的是$C$的第$(m+t-1)$字，因此不会产生进位。

**算法3**　大整数乘法

**输入**　被乘数$A[0{:}m-1]=\sum_{i=0}^{m-1}a_i\cdot 2^{32i}$，乘数$B[0{:}n-1]=\sum_{i=0}^{n-1}b_i\cdot 2^{32i}$，$m\geqslant n$

**输出**　乘积$C[0{:}m+n-1]=\sum_{i=0}^{m+n-1}c_i\cdot 2^{32i}$

1) $C[0{:}m+n-1]=0$；

2) for $i=0$ to $n-1$ do

3) for $j = 0$ to $m - 1$ do
4) CF = 0；
5) madc_lo_cc($a_j$,$b_i$,$c_{i+j}$,CF,$c_{i+j}$)；
6) end for；
7) if $i \neq 0$
8) addc(0,0,CF,$c_{i+n}$)；
9) end if；
10) CF = 0；
11) for $j = 0$ to $m - 1$ do
12) madc_hi_cc($a_j$,$b_i$,$c_{i+j+1}$,CF,$c_{i+j+1}$)；
13) end for；
14) end for；
15) return $C[0{:}m+n-1]$；

### 2.2.3 平方

大整数平方作为乘法的一种特殊情况，在计算中可以重用部分子积结果，从而提高其计算性能。以大整数 $A[0{:}7]=\sum_{i=0}^{7}a_i\cdot 2^{32i}$ 为例，平方运算可以表示为 $(A[0{:}7])^2 = 2M + N$，其中

$$M=\sum_{0\leqslant i<j<7}[2^{32(i+j+1)}(a_i\times a_j)_{\mathrm{hi}}+2^{32(i+j)}(a_i\times a_j)_{\mathrm{lo}}]$$

$$N=\sum_{0\leqslant i<7}[2^{32(2i+1)}({a_i}^2)_{\mathrm{hi}}+2^{32(2i)}({a_i}^2)_{\mathrm{lo}}] \tag{1}$$

具体实现如附录 1 所示。首先利用大整数乘法的实现策略计算式(1)中的乘积 $M$，接着使用加法函数实现乘积 $M$ 的翻倍计算，最后使用乘加函数将 $N$ 累加到对应的计算结果中。综合以上，大整数平方共使用 $8^2 + 8 = 72$ 个乘法函数和 15 个加法函数。

### 2.2.4 约减

在 SM2 数字签名算法中，有限域 $\mathbb{F}_q$ 的规模 $q = 2^{256} - 2^{224} - 2^{96} + 2^{64} - 1$。本文使用 8 个 32 bit 的单字表示该域上的大整数，从而可以将被乘数 $A=\sum_{i=0}^{7}a_i\cdot 2^{32i}$ 和乘数 $B=\sum_{i=0}^{7}b_i\cdot 2^{32i}$ 的取值范围放宽到 $[0,2^{256})$，则乘积结果 $C=\sum_{i=0}^{15}c_i\cdot 2^{32i}$ 的范围为 $[0,2^{512})$。根据 $q = 2^{256} - 2^{224} - 2^{96} + 2^{64} - 1$ 得

$$2^{256}\equiv 2^{224}+2^{96}-2^{64}+1(\mathrm{mod}\,q) \tag{2}$$

从而乘积 $C$ 有限域上可以进一步表示为

$$C\equiv\sum_{i=0}^{7}c_i\cdot 2^{32i}+\sum_{i=8}^{15}c_i\cdot 2^{32i}\ (\mathrm{mod}\,q)\equiv$$
$$\sum_{i=0}^{7}[c_i\cdot 2^{32i}+(2^{224}+2^{96}-2^{64}+1)c_{i+8}\cdot 2^{32i}]\ (\mathrm{mod}\,q) \tag{3}$$

式(2)反复代入式(3)进行计算直至 $C\ (\mathrm{mod}\,q)$ 被约减为 $\hat{C}\equiv\sum_{i=0}^{7}\widehat{c_i}\cdot 2^{32i}\ (\mathrm{mod}\,q)$，经过计算 $\hat{C}$ 中的每一项 $\widehat{c_i}$ 分别为

$$\begin{aligned}
\widehat{c_0} &= c_0 + c_8 + c_9 + c_{10} + c_{11} + c_{12} + 2c_{13} + 2c_{14} + 2c_{15}\\
\widehat{c_1} &= c_1 + c_9 + c_{10} + c_{11} + c_{12} + c_{13} + 2c_{14} + 2c_{15}\\
\widehat{c_2} &= c_2 + -c_8 - c_9 - c_{13} - c_{14}\\
\widehat{c_3} &= c_3 + c_8 + c_{11} + c_{12} + 2c_{13} + c_{14} + c_{15}\\
\widehat{c_4} &= c_4 + c_9 + c_{12} + c_{13} + 2c_{14} + c_{15}\\
\widehat{c_5} &= c_5 + c_{10} + c_{13} + c_{14} + 2c_{15}\\
\widehat{c_6} &= c_6 + c_{11} + c_{14} + c_{15}\\
\widehat{c_7} &= c_7 + c_8 + c_9 + c_{10} + c_{11} + 2c_{12} + 2c_{13} + 2c_{14} + 3c_{15}
\end{aligned} \tag{4}$$

此时，约减之后的结果仍然不在有限域 $\mathbb{F}_q$ 中，还需要经过 2 轮进位消解得到最终模运算结果。根据式(4)中 $\widehat{c_7}$ 的累加过程，约减之后至多会产生 4 bit 的进位（$\mathrm{carry}_1$），记约减结果 $\hat{C}$ 为 $(\mathrm{carry}_0|C_0)\in[0,2^{260})$，利用式(2)进行第一轮进位消解之后得到 $(\mathrm{carry}_1|C_1)$，其取值范围为

$$\begin{cases}[0,2^{256}), & \mathrm{carry}_0 = 0\\ (2^{224}-2^{64},2^{256}+2^{255}+2^{128}), & \mathrm{carry}_0 \neq 0\end{cases} \tag{5}$$

其中，$(\mathrm{carry}_1|C_1)\in[0,2^{256}+2^{255}+2^{128})$，新产生的进位（$\mathrm{carry}_1$）属于 $\{0,1\}$，接着进行第 2 轮进位消解，可得到 $(\mathrm{carry}_2|C_2)$ 为

$$\begin{cases}[0,2^{224}+2^{96}-2^{64}], & (\mathrm{carry}_1|C_1) < 2^{256}\\ [0,2^{255}+2^{224}+2^{129}), & (\mathrm{carry}_1|C_1) \geqslant 2^{256}\end{cases} \tag{6}$$

通过 2 轮进位消解可以完全消除进位，且将结果约减至有限域 $\mathbb{F}_q$ 中。在大整数模加运算中也是采用上述的进位消解方法完成取模操作。为抵御计时攻击[22]，本文在实现过程中不考虑进位取值的分支情况，各个线程计算过程都进行 2 轮进位消解，不会造成线程束分叉。

## 2.3 群运算

模逆是有限域 $\mathbb{F}_q$ 中最耗时的运算，为减少复杂的求逆运算，可以使用 Jacobian 坐标计算点加和倍点[23]。椭圆曲线中点 $(x,y)$ 在 Jacobian 坐标上是 $(X,Y,Z)$，其中 $x=\frac{X}{Z^2}$，$y=\frac{Y}{Z^3}$。Jacobian 坐标上的点加和倍点不涉及求逆操作，但计算完成之后将其转换回仿射坐标需要一次求逆操作。

在 DCU 中，寄存器作为可直接访问的高速存储资源，较全局内存和共享内存具备更低的访问时

延和更高的数据传输带宽。每个线程可以独立地使用寄存器，而不会受到其他线程的影响，这种并行访问性质使得寄存器能够有效支持大规模的并行计算。因此本文在计算点加和倍点运算时，将点的坐标加载至寄存器以提升计算效率。具体实现方案如算法4和算法5所示。

**算法4** 基于DCU的Jacobian坐标点加运算

**输入** 分别存放在寄存器$(X_1,Y_1,Z_1)$、$(X_2,Y_2,Z_2)$的点$P_1$、$P_2$的坐标

**输出** 点加结果$P_3=P_1+P_2=(Z_2,T,Z_1)$的坐标，其中$T$为中间变量

1) $T=Z_1^2$ 2) $X_2=X_2T$
3) $Y_2=Y_2Z_1$ 4) $Y_2=Y_2T$
5) $T=Z_2^2$ 6) $X_1=X_1T$
7) $Y_1=Y_1Z_2$ 8) $Y_1=Y_1T$
9) $X_1=X_1-X_2$ 10) $Z_1=Z_1Z_2$
11) $Z_1=X_1Z_1$ 12) $Y_1=Y_1-Z_2$
13) $T=X_1^2$ 14) $Z_2=X_1^2$
15) $X_2=TX_2$ 16) $X_1=TX_1$
17) $Z_2=Z_2-X_1$ 18) $T=2X_2$
19) $Z_2=Z_2-T$ 20) $X_2=X_2-Z_2$
21) $Y_1=Y_1X_2$ 22) $T=Y_2X_1$
23) $T=Y_1-T$ 24) return $(Z_2,T,Z_1)$

**算法5** 基于DCU的Jacobian坐标倍点运算

**输入** 存放在寄存器$(X_1,Y_1,Z_1)$的点$P_1$的坐标

**输出** 倍点结果$P_2=2P_1=(Z_1,X_1,Y_1)$的坐标

1) $T_1=Y_2^2$ 2) $T_2=X_1T_1$
3) $T_1=T_1^2$ 4) $Y_1=Y_1Z_1$
5) $Z_1=Z_1^2$ 6) $X_1=X_1+Z_1$
7) $Z_1=2Z_1$ 8) $Z_1=X_1-Z_1$
9) $X_1=X_1Z_1$ 10) $Z_1=2X_1$
11) $X_1=X_1+Z_1$ 12) $X_1=X_1/2$
13) $Z_1=X_1^2$ 14) $Z_1=Z_1-T_2$
15) $Z_1=Z_1-T_2$ 16) $T_2=T_2-Z_1$
17) $X_1=X_1T_2$ 18) $X_1=X_1-T_1$
19) return $(Z_1,X_1,Y_1)$

然而，寄存器资源是有限的，过多的寄存器资源消耗会导致资源竞争和性能下降，也会限制同时运行的线程数量，从而降低并行性能。本文单个点加和倍点计算使用的寄存器数量少于100个，能够驻留在DCU寄存器中，因此可以尽量使用寄存器并避免不必要的寄存器占用来提高计算速度。从算法4中可以看出，基于DCU实现Jacobian坐标上的点加运算需要12次模乘、4次模平方、6次模减、1次模加，除了必需的坐标之外仅用到一个中间变量$T$，总计$7\times8=56$个32 bit寄存器。倍点的执行步骤相对于点加较少，详细的计算步骤在算法5中给出，完成该计算仅需要5组寄存器$(X_1,Y_1,Z_1,T_1,T_2)$，总计$5\times8=40$个32 bit寄存器。DCU加速器中每个线程块可以运行1 024个线程，这样每个线程块中最多需要56×1 024=57 344个寄存器，而一个CU拥有65 536个寄存器，因此可以完全驻留在一个CU中。

### 2.4 点乘运算

SM2数字签名算法中的点乘包括已知点点乘和未知点点乘。已知点点乘是指使用已知的固定点进行计算，而未知点点乘则是在运算过程中动态生成点，并进行相应的计算。

#### 2.4.1 已知点点乘

已知点点乘常用梳状（Comb）算法[24]和分段算法[25]，相比于Comb算法，分段算法需要的访存次数和点算数运算步骤更少，但需要大量的存储资源。鉴于DCU加速器拥有足够大的全局内存来存放预计算表，本文选用分段算法完成已知点点乘。

设点$G$为已知点，将二进制位数为$k$的标量$d$分成$n$组，即$d=d_{n-1}|\cdots|d_1|d_0$，令$m=\frac{k}{n}$，则每组包含$2^m$个点。第1组的$2^m$个点为$O,G,2G,\cdots,(2^m-1)G$，第2组的$2^m$个点为$2^mG,(2^m+1)G,\cdots,(2^{m+1}-1)G$，以此类推，第$n$组的$2^m$个点为$2^{(n-1)m}G,(2^{(n-1)m}+1)G,\cdots,(2^{nm}-1)G$。将计算生成的预计算表存储在DCU的全局内存中，从而$dG=\sum_{i=0}^{n-1}d_i2^{im}G$，实现如算法6所示。需要注意的是，需要选取合适的$n$，$n$越小则点加运算越少，但需要离线预计算的点越多，生成的离线预计算表所占的DCU全局内存就越大。若选取$n=16$，每组包含16 bit，则离线预计算表大小约为128 MB。

**算法6** 已知点点乘算法

**输入** 已知点$G$，全局内存中的预计算表$R$，

被分成 $n$ 组且二进制位数为 $k$ 的标量 $d = d_{n-1}|\cdots|d_0$，令 $m = \frac{k}{n}$，则 $d = \sum_{i=0}^{n-1} d_i 2^{im} G$

**输出**　已知点点乘结果 $Q = dG$

1) 从预计算表 $R$ 加载 $d_0 G$ 结果到寄存器 $Q$；
2) for $i$ = 1 to $n$ - 1 do
3)　　从预计算表 $R$ 加载 $d_i 2^{im} G$ 至寄存器；
4)　　$Q = Q + d_i 2^{im} G$；
5) end for
6) return $Q$

#### 2.4.2　未知点点乘

未知点点乘 $Q = dP$ 中，点 $P$ 是未知且变化的，无法提前生成预计算表来减少计算步骤。未知点点乘常用的算法包括加倍累加算法[25]、固定窗口算法[25]、滑动窗口算法[26]等，加倍累加算法效率低且不能抵御计时攻击，滑动窗口算法效率一般比固定窗口算法高，但也不能抵御计时攻击，因此本文选择固定窗口算法计算未知点点乘。

在固定窗口算法中，设 $\omega$ 为窗口大小，每次扫描标量 $d$ 特定的 $k$ bit，生成包含 $2^{\omega}$ 个点的在线预计算表，总计需要扫描 $n = \frac{k}{\omega}$ 次。具体实现如算法 7 所示，首先利用 $2^{\omega} - 1$ 次点加生成在线预计算表，然后使用 $(n-1)\omega$ 次倍点和 $n-1$ 次点加计算得到最终结果。同样，需要合理选择窗口 $\omega$ 的大小，才能实现最优的未知点点乘性能。本文得出最佳窗口大小为 4 时，在线预计算表的大小为 1.5 KB，通过实验分析发现，若将其存放在全局内存中，频繁地访问全局内存会带来一定的性能开销，因此本文将其存放在临时内存中以提升计算效率。

**算法 7**　未知点点乘算法

**输入**　未知点 $P$，窗口大小 $\omega$，$k$ bit 标量 $d$，扫描次数 $n = \frac{k}{\omega}$，则 $d = d_{n-1}|\cdots|d_1|d_0$

**输出**　未知点点乘结果 $Q = dP$

1) 将点 $O$、$P$ 存储至在线预计算表 $R$；
2) $Q = P$；
3) for $i$ = 2 to $2^{\omega} - 1$ do
4)　　$Q = Q + P$；
5)　　将 $Q$ 存储至在线预计算表 $R$；
6) end for
7) 从在线预计算表 $R$ 加载 $d_{n-1} P$ 至 $Q$；
8) for $i = n - 2$ to 0 do
9)　　for $j$ = 0 to $\omega - 1$ do
10)　　$Q = 2Q$；
11)　　end for
12)　　从预计算表 $R$ 加载 $d_i P$ 至临时变量 $T$；
13)　　$Q = Q + T$；
14) end for
15) return $Q$

## 3　性能评估

本节主要分析 SM2 数字签名算法在海光 DCU 上的实验性能，DCU 的硬件与软件配置信息如表 3 所示。在硬件方面，CPU 为海光 C86 7285 主处理器，运行主频为 2.0 GHz；协处理器为海光 DCU 加速器，其内部集成 60 个计算单元，可有效支撑大规模并行任务的执行。在软件环境方面，实验采用 CentOS 7.6 作为操作系统，配备 Hipcc 2.9 作为主要的开发工具链，并支持 HIP 异构编程模型的完整构建与运行。

**表 3**　**DCU 信息**

| 类别 | 名称 | 型号 |
|---|---|---|
| 硬件 | CPU | Hygon C86 7285 Processor@ 2.0GHz |
| | DCU | 海光 DCU（60 个 CU） |
| 软件 | 操作系统 | CentOS 7.6 |
| | 工具链 | Hipcc 2.9 |

吞吐量和时延是评估 DCU 加速器上 SM2 数字签名算法计算性能的 2 个关键指标[12,14,27]，吞吐量是指每秒完成的签名/验签次数（kops/s），时延是指完整处理一批计算请求需要的时间（ms）。

### 3.1　实现结构

在 DCU 计算中，高吞吐量和低时延往往相互矛盾，一般情况下，为了追求低时延则需要将一个完整的算法拆分，使其运行在数个并行执行的线程中。但这种方案不适合 SM2 数字签名算法，因为非对称密码算法往往无法拆分成完全独立的线程，从而各个线程之间需要数据交互和线程同步，从而造成性能损失，无法获得最高的吞吐量，此外算法拆分成的线程数越多，性能消耗越明显。这个结论在文献[28-31]中得到了证实。

本文实现的 SM2 数字签名算法面向高并发场景，设计原则为具有可接受的时延下达到尽可能大

的吞吐量。通过实验发现，使用一个线程完成一个SM2数字签名/验签算法的时延仅为数十毫秒，在可接受范围内，因此本文采用一个线程处理一个完整的算法，不存在任何形式的数据交互和线程同步操作。下面是一些可配置参数，不同参数对算法性能会产生不同的影响。

1)线程块大小：每个线程块中包含的线程数。

2)线程总数：DCU中为SM2签名/验签开启的总线程数量，表示为启动的线程块数量×线程块大小。

### 3.2　实现结果

本节主要分析基于国产DCU平台实现SM2数字签名算法的性能与安全性评估。

#### 3.2.1　性能分析

本文使用的DCU平台具有60个CU，应设置线程块的数量为60的倍数以尽可能将其均匀分配至各CU以提高利用率，经过实验分析发现，当线程块数量为480时已近乎充分调动DCU计算资源，若线程块数量继续增加，吞吐量趋于稳定，而时延随着计算请求的增加而增大。为探究不同线程块大小对SM2签名/验签计算性能的影响，对SM2签名/验签分别测试不同线程块大小配置下的吞吐量和时延，实验吞吐量结果的数值均标注在图7和图8中。从图7和图8中可以看出，SM2签名/验签实现在线程块大小为512时取得性能峰值，分别为6 816 kops/s和1 385 kops/s。然而更大的线程块大小也意味着需要更多的计算和存储资源，在达到性能峰值之后继续增加线程块中线程的数量，会导致DCU的寄存器资源不足，理应存放在寄存器中的数据会“溢出”到访存更慢的内存中。当资源需求超过较大批处理规模带来的性能提升时，性能会下降，时延也会大幅度增长，具体表现为当线程块大小为640时，性能下降的同时时延增长几乎近一倍。

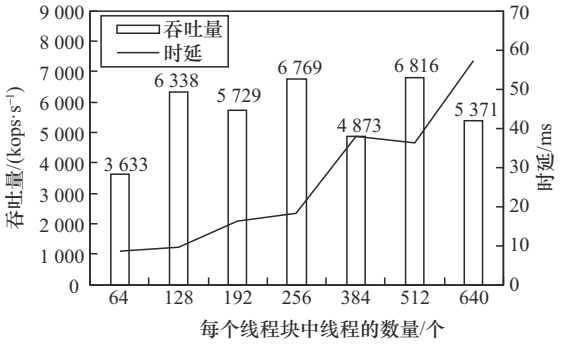



图7　SM2签名性能

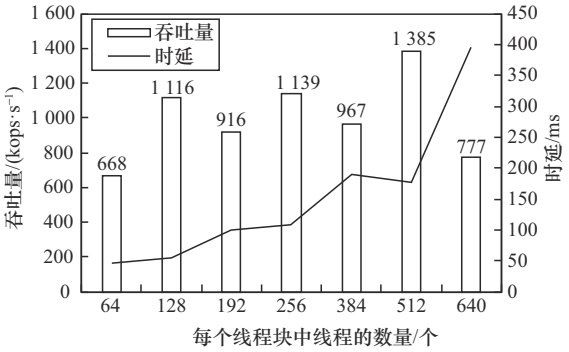



图8　SM2验签性能

#### 3.2.2　安全性分析

侧信道攻击指通过分析算法执行过程中泄露的物理信息（如时间、功耗、电磁泄漏等）来破解密码系统的攻击方式。计时攻击是侧信道攻击的一种常见类型，其主要专注于通过分析密码算法需要的时间推断出系统的秘密信息。本文实现过程中规避数据依赖型分支和不恒定的内存访问模式，实现恒定执行路径，从理论上可有效抵御计时攻击。具体而言，模块化的有限域运算不涉及可能导致执行时间变化的数据依赖型分支，点乘中的内存访问和跳转模式也不依赖标量的任一比特，进一步削弱侧信道泄露的可能性。

此外，为提升对GPU缓存类侧信道攻击的抵抗能力，本文避免使用共享内存及依赖缓存行为的线程调度策略，提升整体的抗侧信道鲁棒性。本文在算法实现中采用时延一致性和恒定控制流等防护策略，在实现结构上具备理论防护效果。

除侧信道攻击外，本文还考虑其他几类常见的攻击模型。首先，曲线级安全是通过曲线算法本身实现的，SM2数字签名算法基于椭圆曲线离散对数问题，目前无有效的多项式时间算法可破解该问题，因此具备较强的数学安全性。其次，本文在实现国产中避免数据相关的条件跳转，关键中间变量亦不暴露于潜在故障易感区域，减少了故障攻击的利用面，具备一定的抗故障攻击能力。最后，SM2数字签名算法在国家商用密码标准中已定义防止选择消息攻击的机制，本文遵循其标准规范实现，并未引入新的数据处理流程，不会引入新的攻击面。

### 3.3　性能对比

本节将DCU平台高性能SM2数字签名算法的实现与相关文献的多种实现方案[10-12,32-34]进行对比，如表4所示，主要包括CPU平台、FPGA平台和GPU平台。

表 4　　相关文献的性能对比

| 文献 | 平台 | 签名吞吐量/(kops·s$^{-1}$) | 验签吞吐量/(kops·s$^{-1}$) |
|---|---|---|---|
| 文献[10] | AMD Radeon R9 290 | 935.533 | 114.081 |
| 文献[11] | Intel Core i7-7700k @4.2 GHz (8 线程) | 2.302 | 1.624 |
| | NVIDIA GTX1080 | 320.942 | 152.887 |
| 文献[12] | FPGA Znyq7000 @150 MHz | 3.984 | 2.309 |
| | NVIDIA RTX 3090 | 76 097 | 3 449 |
| | NVIDIA T1000 | 6 348.632 | 345.248 |
| 文献[32] | Intel Core i7-1165G7 @ 2.80 GHz | 5.827 | 5.753 |
| 文献[33] | FPGA Virtex-7 @122.05 MHz | 0.132 | 0.057 |
| 文献[34] | FPGA Alveo @250 MHz | — | 2.717 |
| 本文 | i5-12600KF(GmSSL) | 22.564 | — |
| | 海光 DCU (HIP) | 6 816 | 1 385 |

### 3.3.1　与 CPU 平台实现对比

文献[11]在 Intel Core i7-7700k @ 4.2 GHz 上采用 8 线程各完成 10 000 次签名/验签计算，得到 CPU 实现的性能峰值分别为 2.302 kops/s 和 1.624 kops/s；文献[32]利用改进滑动窗口算法来优化点乘运算，优化改进蒙哥马利模乘算法来优化模乘计算，最终在 Intel Core i7-1165G7 @ 2.80 GHz 上实验得到签名/验签的最佳性能为 5.827 kops/s 和 5.753 kops/s。此外，本文在综合已有文献数据的基础上，进一步补充了在通用 CPU 平台上使用主流国产密码工具库 GmSSL[35]对 SM2 数字签名算法的性能进行实测。GmSSL 自 3.1.1 版本以来在其测试工具 tests 模块中集成了最先进的国产商用密码算法性能测试功能。实测结果显示，在搭载第 12 代 Intel Core i5-12600KF 处理器的主机上，SM2 数字签名操作的吞吐性能达 22.564 kops/s。

相比之下，本文基于 DCU 提出的高性能并行加速方案，在充分利用多核并发特性基础上，显著提升了签名与验签的处理能力签名/验签性能达到百万次/秒。本文方法在处理批量签名请求时可达到 6 816 kops/s 的吞吐量，性能相比 GmSSL 在 CPU 上的实现提升超过 300 倍。相比于 CPU 采用的串行计算方式，在处理请求数量达到一定规模时，DCU 的大规模并行计算优势得以体现。尽管 DCU 的主频相对较低，处理单个计算任务所需的时间可能更多，但在面对大规模运算请求的场景，其并行处理能力更具优势，能够更高效地处理大量的计算任务。

### 3.3.2　与 FPGA 平台实现对比

FPGA[36]是一种极具灵活性和可重配置性的数字电路集成芯片。通过优化其硬件架构，能够提高并行度和吞吐量，减少计算时延，因此在密码工程领域得到广泛的应用。文献[11,33-34]研究 SM2 数字签名算法在 FPGA 平台上的实现，其中文献[11]取得的性能最佳，在 Znyq7000 平台上签名/验签吞吐量为 3.984 kops/s 和 2.309 kops/s，该性能远远低于本文在 DCU 平台的实验结果。此外，FPGA 在功耗方面更具优势，但本文基于 DCU 平台开发的 SM2 数字签名算法更具灵活性和可重配置性，可以扩展到不同型号的 DCU 平台，也可以改写成 CUDA 异构编程模型部署至英伟达 GPU 平台。

### 3.3.3　与 GPU 平台实现对比

目前仅有文献[15]在国产 GPU 平台实现 SM2 加解密算法，其实现性能相比于 CPU 串行实现提升仅有 4~6 倍，且该研究并未给出具体的加解密性能吞吐量。文献[10]在 AMD Radeon R9 290 GPU 上实现 SM2 数字签名算法，性能达到 935.533 kops/s 和 114.081 kops/s。此外，文献[11-12]在英伟达不同型号的 GPU 上实现并优化了 SM2 数字签名算法，其中文献[12]在 NVIDIA RTX 3090 单卡上的实现是已知性能最佳的实验结果，达到 76 097 kops/s 和 3 449 kops/s。而本文基于国产 DCU 实现的 SM2 数字签名/验签算法的最佳性能为 6 816 kops/s 和 1 385 kops/s，相比之下性能稍弱，但考虑到两者硬件架构和软件生态的显著差异，该性能已具有一定代表性。

具体而言，RTX 3090基于Ampere架构，拥有84个流处理器（SM, streaming multiprocessor）、10 496个CUDA核心和约17 800 GIOPS的整数计算能力，同时拥有936 GB/s的超高内存带宽。相比之下，本文使用的海光DCU仅配备60个CU、3 840个计算核心，核心数量仅为RTX 3090的约36%。RTX 3090的更多核心支持更高的线程并行度，而DCU受限于3 840个计算核心，线程并发规模较小，这直接影响了并行任务的调度效率和吞吐率。此外，海光DCU的内存带宽约为1 TB/s，但由于调度机制、缓存结构等因素的不同，其实际带宽利用率和指令吞吐效率不及NVIDIA平台。在指令集优化方面，文献[12]借助成熟的PTX汇编指令进行低层次优化，如使用mad指令融合乘加运算，大幅减少访存和指令数量，显著优化有限域乘法等关键操作。而海光DCU目前虽然支持类似的内联汇编，但在软件工具链的支持和指令集稳定性方面仍待完善。

CUDA作为业界成熟的异构计算平台，其高度统一的指令集架构、驱动支持和调试工具，使得开发者能够在不同型号的NVIDIA GPU之间高效迁移和复用代码，并充分挖掘硬件性能。然而，当前国产GPU生态尚不成熟，各平台对CPU-GPU异构协同模型的支持程度存在差异。例如，本文采用基于开源计算（ROCm, radeon open compute）框架的HIP编程模型实现SM2数字签名算法的并行加速，其可以支持包括AMD GPU和海光、摩尔线程等部分国产GPU芯片，但由于不同厂商采用的底层架构和系统接口存在较大差异，实际的算法适配与优化仍面临较大挑战。为提升不同架构的适配性，本文通过解耦算法实现与硬件特性来提升可移植性，将有限域运算、椭圆曲线群运算及点乘等核心运算划分为与平台无关的模块，同时将CPU-GPU协同计算等硬件相关特性划分在独立的功能层，增强算法结构的可移植性，便于后续在不同国产GPU架构之间进行迁移。未来，本文将针对特定国产GPU架构使用特定的指令集实现有限域的底层运算，例如图形核心下一代（GCN, graphics core next）架构的MAD（Multiply-Add）指令以进一步提升SM2数字签名算法的计算性能。

## 4　结束语

本文选择国产DCU作为国产公钥密码SM2高性能并行加速的平台，围绕高吞吐量、可扩展和高灵活性等目标，构建底层函数，对有限域运算、群运算、点乘运算进行研究。基于DCU完整实现SM2数字签名/验签算法，在兼顾安全性的基础上取得显著性能优势，在单卡DCU上可以达到6 816 kops/s的签名吞吐量和1 385 kops/s的验签吞吐量。

## 附录1　大整数平方算法

**输入**　乘数 $A[0{:}n-1]=\sum_{i=0}^{n-1}a_i\cdot 2^{32i}$

**输出**　平方结果 $C[0{:}2n-1]=\sum_{i=0}^{2n-1}c_i\cdot 2^{32i}$

1) $C[0{:}2n-1]=0$，CF = 0；
2) for $i=0$ to $\left\lfloor\frac{n}{2}-2\right\rfloor$ do
3)　for $j=i+1$ to $n-i-2$ do
4)　　madc_lo_cc($a_j$,$a_i$,$c_{i+j}$,CF,$c_{i+j}$)；
5)　end for
6)　for $k=0$ to $i$ do
7)　　madc_lo_cc($a_{n-1-i+k}$,$a_{i+k}$,$c_{n+2k-1}$,CF,$c_{n+2k-1}$),
　　madc_hi_cc($a_{n-1-i+k}$,$a_{i+k}$,$c_{n+2k}$,CF,$c_{n+2k}$)
8)　end for
9)　CF = 0
10)　for $j=i+1$ to $n-i-2$ do
11)　　madc_hi_cc($a_j$,$a_i$,$c_{i+j+1}$,CF,$c_{i+j+1}$)
12)　end for
13)　for $k=0$ to $i$ do
14)　　madc_lo_cc($a_{n-1-i+k}$,$a_{i+k+1}$,$c_{n+2k}$,CF,$c_{n+2k}$),
　　madc_hi_cc($a_{n-1-i+k}$,$a_{i+k+1}$,$c_{n+2k+1}$,CF,
　　$c_{n+2k+1}$)
15)　end for
16) end for
17) if $n$为偶数, then
18)　CF = 0
19)　for $k=0$ to $\frac{n}{2}-1$ do
20)　　madc_lo_cc($a_{\frac{n}{2}+k}$,$a_{\frac{n}{2}+k-1}$,$c_{n+2k-1}$,CF,$c_{n+2k-1}$),
　　madc_hi_cc($a_{\frac{n}{2}+k}$,$a_{\frac{n}{2}+k-1}$,$c_{n+2k}$,CF,$c_{n+2k}$)
21)　end for
22) end if
23) CF = 0
24) for $i=1$ to $2n-1$ do
25)　addc($c_i$,$c_i$,CF,$c_i$)
26) end for
27) CF = 0
28) for $i=0$ to $n-1$ do
29)　madc_lo_cc($a_i$,$a_i$,$c_{2i}$,CF,$a_{2i}$)
30)　madc_hi_cc($a_i$,$a_i$,$c_{2i+1}$,CF,$a_{2i}+1$)
31) end for
32) return $C[0{:}2n-1]$

## 参考文献：


[1] 周秭沫 . 美国升级对华芯片限制砸了谁的脚[N]. 国际金融报, 2024-11-04.
ZHOU Z M. Who does the U. S. upgrading chip restrictions against China actually hurt?[N]. International Finance News, 2024-11-04.

[2] 薛路皓 . 国外芯片漏洞频发国产 CPU 替代加速[N]. 第一财经日报, 2024-11-19.
XUE L H. Frequent vulnerabilities in foreign chips accelerate domestic CPU replacement[N]. First Financial Daily, 2024-11-19.

[3] 孙永杰 . AI 芯片受限 海光信息 DCU 能否担起替代重任?[J]. 通信世界, 2024(6): 4.
SUN Y J. Can DCU, whose AI chip is limited by sea light information, take on the heavy responsibility of substitution? [J]. Communications World, 2024(6): 4.

[4] BERNSTEIN D J, LANGE T. Non-uniform cracks in the concrete: the power of free precomputation[C]//Proceedings of the 19th International Conference on the Theory and Application of Cryptology and Information Security. Berlin: Springer, 2013: 321-340.

[5] 国家密码管理局 . SM2 椭圆曲线公钥密码算法 第 2 部分：数字签名算法: GM/T 0003.2—2012[S]. 北京: 中国标准出版社, 2012.
State Cryptography Administration. Public key cryptographic algorithm SM2 based on elliptic curves: Part 2: Digital signature algorithm: GM/T 0003.2—2012[S]. Beijing: Standards Press of China, 2012.

[6] International Organization for Standardization. IT Security techniques - Digital signatures with appendix - Part 3: Discrete logarithm based mechanisms: ISO/IEC 14888-3: 2018[S]. Geneva: ISO, 2018.

[7] 袁波, 孙佳雯, 李婧怡 . SM2 算法在信息系统中的应用和商用密码应用安全性评估方法研究[J]. 网络安全技术与应用, 2025(4): 51-53.
YUAN B, SUN J W, LI J Y. Research on the application of SM2 algorithm in information system and the security evaluation method of commercial password application[J]. Network Security Technology & Application, 2025(4): 51-53.

[8] 孔团结, 郑昉昱, 郭润, 等 . 低功耗软硬结合 SM2 算法实现[J]. 密码学报(中英文), 2024, 11(6): 1354-1369.
KONG T J, ZHENG F Y, GUO R, et al. Low-power hardware-software co-design implementation of SM2 algorithm[J]. Journal of Cryptography (Chinese & English), 2024, 11(6):1354-1369.

[9] TEZCAN C, LEANDER G. GPU assisted brute force cryptanalysis of GPRS, GSM, RFID, and TETRA[J]. IACR Transactions on Symmetric Cryptology, 2025, 2025(1): 309-327.

[10] 闫闵 . 基于 GPU 的 SM2 数字签名和验证算法的快速实现与优化[D]. 上海: 上海交通大学, 2020.
YAN M. Fast implementation and optimization of SM2 digital signature and verification algorithm based on GPU[D]. Shanghai: Shanghai Jiao Tong University, 2020.

[11] 邓致远 . 基于异构计算的椭圆曲线算法加速的研究与实现[D]. 长春: 吉林大学, 2023.
DENG Z Y. Research and implementation of acceleration of elliptic curve algorithm based on heterogeneous computing[D]. Changchun: Jilin University, 2023.

[12] 朱辉, 黄煜坤, 王枫为, 等 . 一种基于图形处理器的高吞吐量 SM2 数字签名计算方案[J]. 电子与信息学报, 2022, 44(12): 4274-4283.
ZHU H, HUANG Y K, WANG F W, et al. A high throughput SM2 digital signature computing scheme based on graphics processing unit platform[J]. Journal of Electronics & Information Technology, 2022, 44(12): 4274-4283.

[13] PAN W Q, ZHENG F Y, ZHAO Y, et al. An efficient elliptic curve cryptography signature server with GPU acceleration[J]. IEEE Transactions on Information Forensics and Security, 2017, 12(1): 111-122.

[14] GAO L L, ZHENG F Y, EMMART N, et al. DPF-ECC: accelerating elliptic curve cryptography with floating-point computing power of GPUs[C]//Proceedings of the 2020 IEEE International Parallel and Distributed Processing Symposium (IPDPS). Piscataway: IEEE Press, 2020: 494-504.

[15] 刘勇杰 . 面向“嵩山” 超级计算机节点结构的国密算法实现与优化[D]. 郑州: 郑州大学, 2022.
LIU Y J. Implementation and optimization of state secret algorithm for node structure of Songshan supercomputer[D]. Zhengzhou: Zhengzhou University, 2022.

[16] 汪朝晖, 张振峰 . SM2 椭圆曲线公钥密码算法综述[J]. 信息安全研究, 2016, 2(11): 972-982.
WANG Z H, ZHANG Z F. Overview on public key cryptographic algorithm SM2 based on elliptic curves[J]. Journal of Information Security Research, 2016, 2(11): 972-982.

[17] 景川 . 海光信息：国产高端处理器希望之光[J]. 经理人, 2024(3): 48-50.
JING C. Hygon information: the light of hope for domestic high-end processors[J]. Manager, 2024(3): 48-50.

[18] 岳源源 . 面向 DCU 加速器件的国密算法实现与优化[D]. 郑州: 郑州大学, 2022.
YUE Y Y. Realization and optimization of state secret algorithm for DCU accelerator[D]. Zhengzhou: Zhengzhou University, 2022.

[19] FREYTAG G, SERPA M S, LIMA J V F, et al. Collaborative execution of fluid flow simulation using non-uniform decomposition on heterogeneous architectures[J]. Journal of Parallel and Distributed Computing, 2021, 152: 11-20.

[20] 郝萌, 田雪洋, 鲁刚钊, 等 . 基于国产 DCU 异构平台的图匹配算法移植与优化[J]. 计算机科学, 2024, 51(4): 67-77.
HAO M, TIAN X Y, LU G Z, et al. Transplantation and optimization of graph matching algorithm based on domestic DCU heterogeneous platform[J]. Computer Science, 2024, 51(4): 67-77.

[21] 梁正虹 . CPU+GPU 异构并行计算研究及其在可压缩流动中的应用[D]. 绵阳: 西南科技大学, 2021.
LIANG Z H. Research on heterogeneous parallel computing of CPU+GPU and its application in compressible flow[D]. Mianyang: Southwest University of Science and Technology, 2021.

[22] 赵良驹, 汤宇锋, 龚征 . 白盒密码实现的侧信道分析技术综述[J]. 网络空间安全科学学报, 2024(6): 57-73.
ZHAO L J, TANG Y F, GONG Z. A survey of side-channel analysis on white-box cryptography[J]. Journal of Cybersecurity, 2024(6): 57-73.

[23] LARA-NINO C A, DIAZ-PEREZ A, MORALES-SANDOVAL M. A comparison of differential addition and doubling in binary edwards curves for elliptic curve cryptography[C]//Proceedings of the 2021 Fifth World Conference on Smart Trends in Systems Security and Sustainability (WorldS4). Piscataway: IEEE Press, 2021: 12-18.

[24] ZHANG J, CHEN Z, MA M, et al. High-performance ECC scalar multiplication architecture based on comb method and low-latency window recoding algorithm[J]. IEEE Transactions on Very Large Scale Integration (VLSI) Systems, 2023, 32(2): 382-395.

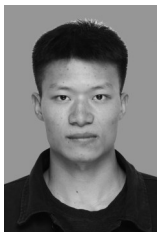
[25] HANKERSON D, MENEZES A J, VANSTONE S. Guide to elliptic curve cryptography[M]. New York: Springer Science & Business Media, 2006.

[26] COHEN H, FREY G, AVANZI R, et al. Handbook of elliptic and hyperelliptic curve cryptography[M]. Boca Raton: CRC Press, 2005.
[27] DONG J K, ZHENG F Y, CHENG J J, et al. Towards high-performance X25519/448 key agreement in general purpose GPUs[C]// Proceedings of the 2018 IEEE Conference on Communications and Network Security (CNS). Piscataway: IEEE Press, 2018: 1-9.
[28] SZERWINSKI R, GÜNEYSU T. Exploiting the power of GPUs for asymmetric cryptography[C]//Proceedings of the 10th International Workshop Cryptographic Hardware and Embedded Systems. Berlin: Springer, 2008: 79-99.
[29] BOS J W. Low-latency elliptic curve scalar multiplication[J]. International Journal of Parallel Programming, 2012, 40(5): 532-550.
[30] HARRISON O, WALDRON J. Efficient acceleration of asymmetric cryptography on graphics hardware[C]//Proceedings of the Second International Conference on Cryptology Progress in Cryptology. Berlin: Springer, 2009: 350-367.
[31] DONG J, ZHENG F, LIN J, et al. EC-ECC: accelerating elliptic curve cryptography for edge computing on embedded GPU TX2[J]. ACM Transactions on Embedded Computing Systems (TECS), 2022, 21(2): 1-25.
[32] 孙宏健. SM2算法快速实现研究[D]. 成都: 电子科技大学, 2021.
SUN H J. Research on fast implementation of SM2 algorithm[D]. Chengdu: University of Electronic Science and Technology of China, 2021.
[33] HU J, LI J J, XI Q S, et al. An optimization method for national cryptography algorithm[C]//Proceedings of the 2024 International Symposium on Parallel Computing and Distributed Systems (PCDS). Piscataway: IEEE Press, 2024: 1-5.
[34] 王国强, 张明明, 任凯琦, 等. 一种基于FPGA的ECC点乘优化设计[J]. 信息技术与信息化, 2024(2): 35-38.
WANG G Q, ZHANG M M, REN K Q, et al. An optimized design of ECC point multiplication based on FPGA[J]. Information Technology and Informatization, 2024(2): 35-38.
[35] KIEU-DO-NGUYEN B, PHAM-QUOC C, TRAN N T, et al. Low-cost area-efficient FPGA-based multi-functional ECDSA/EdDSA[J]. Cryptography, 2022, 6(2): 25.
[36] AGRAWAL R, YANG J, JAVAID H. Efficient FPGA-based ECDSA verification engine for permissioned blockchains[C]//Proceedings of the 2022 IEEE 33rd International Conference on Application-specific Systems, Architectures and Processors (ASAP). Piscataway: IEEE Press, 2022: 148-155.

[作者简介]

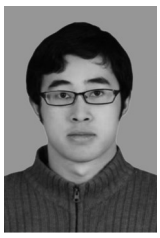
**吴雯**（2000– ），男，江苏东台人，南京邮电大学博士生，主要研究方向为公钥密码、密码工程学、数据安全等。

**董建阔**（1992– ），男，河北邢台人，博士，南京邮电大学副教授、硕士生导师，主要研究方向为公钥密码、后量子密码、并行计算等。



**刘鹏博**（1998– ），男，山西运城人，南京邮电大学硕士生，主要研究方向为密码工程学、并行计算等。

**董振江**（1970– ），男，江苏南京人，博士，南京邮电大学教授、博士生导师，主要研究领域为网络空间安全、人工智能等。



**胡昕**（1988– ），男，江苏南京人，南京邮电大学博士生，主要研究方向为应用密码学、网络入侵检测、云安全与虚拟化等。

**张品昌**（1985– ），男，安徽滁州人，博士，南京邮电大学副教授、硕士生导师，主要研究方向为6G无线网络安全、物理层安全认证、无人机通信安全、人工智能安全等。

**肖甫**（1980– ），男，湖南邵阳人，博士，南京邮电大学教授、博士生导师，主要研究方向为网络空间安全、物联网技术等。