FourierPIM：高吞吐量存内快速傅里叶变换与多项式乘法

Orian Leitersdorf[1,*], Yahav Boneh[1], Gonen Gazit[1], Ronny Ronen[1], Shahar Kvatinsky[1]

[a] *Viterbi Faculty of Electrical and Computer Engineering, Technion – Israel Institute of Technology, Haifa, 3200003, Israel*

arXiv:2304.02336v1 [cs.AR] 5 Apr 2023

摘要

离散傅里叶变换（DFT）对于从信号处理到卷积和多项式乘法等各种应用都至关重要。突破性的快速傅里叶变换（FFT）算法将 DFT 的时间复杂度从朴素的 $O(n^2)$ 降至 $O(n\log n)$。近期研究则尝试通过 GPU 等并行架构进一步加速。不幸的是，随着内存访问成为瓶颈，GPU 等加速器无法充分发挥其计算能力。因此，本文利用数字存内处理（PIM）架构加速 FFT 算法，通过能够实现存储和逻辑运算的物理器件（如忆阻器）将计算转移到内存中。我们提出一种 $O(\log n)$ 存内快速傅里叶变换算法，该算法还可在多个阵列上并行执行，以实现高吞吐量批量执行，并支持定点数和浮点数。借助卷积定理，我们将该算法扩展至 $O(\log n)$ 多项式乘法，这是一项密码学等应用中的基础任务。我们在公开可用的周期精确模拟器上评估 FourierPIM，验证其正确性和性能；结果表明，在 FFT 和多项式乘法方面，相较于先进 GPU 上的 NVIDIA cuFFT 库，吞吐量提升 5–15×，能耗改善 4–13×。

1. 引言

傅里叶变换在信号处理和密码学等众多领域中都至关重要 [1, 2]。例如，傅里叶变换可以根据不同频率的振幅对输入信号（如音频）进行分解，也可用于实现快速卷积和多项式乘法。尤其是，离散傅里叶变换（DFT）是一种作用于有限复数序列的可逆线性变换。然而，直接计算n维离散傅里叶变换需要 $O(n^2)$ 时间。幸运的是，1965年该领域的一项突破性发现是快速傅里叶变换（FFT） [3]算法，它利用递归分治法，以 $O(n\log n)$ 的时间复杂度计算离散傅里叶变换。此前的研究尝试使用 GPU 进行大规模并行计算，进一步将 FFT 算法加速至优于 $O(n\log n)$ [4]。

尽管 GPU 等架构提供了强大的计算并行能力，但由于存在内存墙瓶颈，它们无法充分发挥利用率因此无法充分利用 FFT算法。图 1 展示了基于英伟达 Nsight Compute 性能分析工具推导出的屋顶线模型，该模型能够准确评估 GPU 的底层性能。模型表明，由于应用受内存带宽限制，cuFFT [4] 的有效浮点性能仅为理论最大值的 9%（1.6 TFLOPS，而理论值为 17.7 TFLOPS）；因此，GPU 核心与 GPU 内存之间的初始 $O(n)$ 数据传输构成了瓶颈。因此，在本文中，我们转而研究能够应对内存墙瓶颈的新兴架构。

存内处理（PIM）通过在存储器内部执行计算，克服内存瓶颈。我们聚焦于数字处理器内存架构，它们利用底层物理器件本身同时支持存储和计算的特性（例如，忆阻器），使存储器内部能够执行基本的按位逻辑运算。近期研究展示了具有极高吞吐量的存内向量算术；AritPIM 提出了一套适用于定点数和浮点数的算术运算。尽管此前的研究尝试使用 PIM 实现类似的变换，但它们需要复杂的定制近存外设，不支持并行批量执行，因此整体吞吐量低于现有 GPU 方案。

*本研究部分经费由欧洲研究委员会通过欧盟“地平线2020”研究与创新计划（资助协议编号757259）提供，部分经费由欧洲研究委员会通过欧盟“地平线研究与创新计划”（资助协议编号101069336）提供，另有部分经费由以色列科学基金会（资助协议编号1514/17）提供。

*通讯作者*Email addresses:* orianl@campus.technion.ac.il (Orian Leitersdorf), yhb.bo@campus.technion.ac.il(Yahav Boneh),gonn.g@campus.technion.ac.il (Gonen Gazit), ronny.ronen@technion.ac.il(Ronny Ronen), shahar@ee.technion.ac.il (Shahar Kvatinsky)

[1]存内处理也可用于高效的、 *approximate*、在 *analog* 域 [9]中的矩阵-向量乘法。此前的工作已基于这一概念实现了快速的 *approximate* 离散傅里叶变换， [10], 但本文聚焦于 *exact* 离散傅里叶变换（具有浮点精度）。

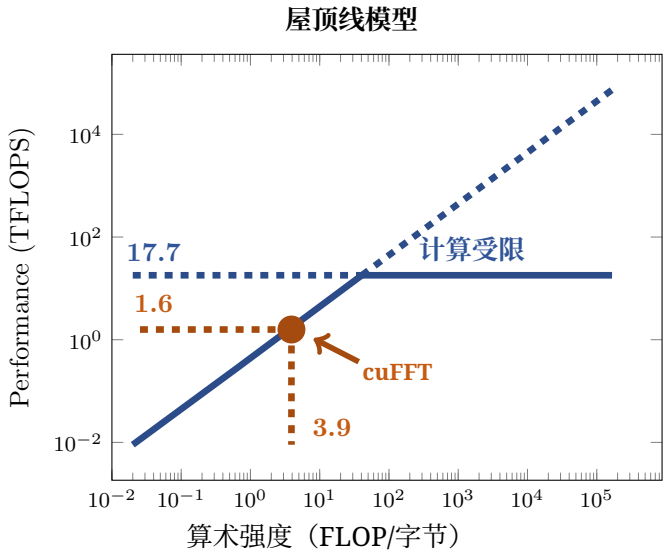


图1：NVIDIA RTX 3070 GPU 上 cuFFT 的Roofline模型；通过Nsight Compute 分析器为批处理的 $n = 8192$ 快速傅里叶变换生成。

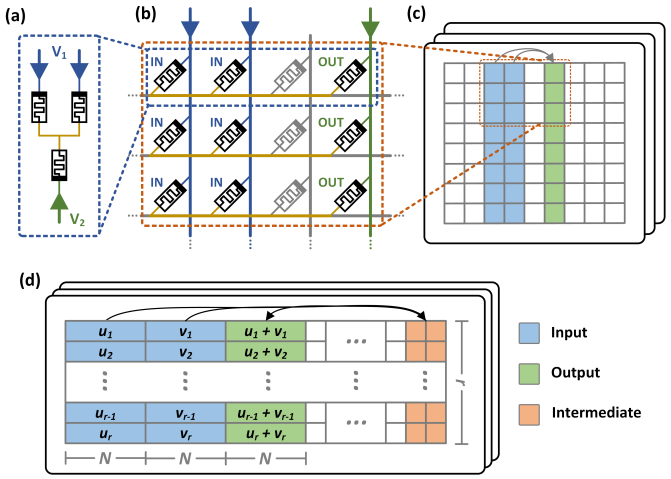


图 2：(a) 在输入（顶部）与输出（底部）忆阻器之间执行的有状态逻辑。(b) 通过在位线上施加电压，在交叉开关阵列的行内执行有状态逻辑。(c) 抽象计算模型，以 $O(1)$ 时间在列上提供按位运算。(d) 利用按位列操作实现向量算术。

本文利用数字处理器内存强大的算术吞吐能力，实现完全通过存内操作执行的高吞吐量快速傅里叶变换。我们设计了一种存内原位快速傅里叶变换算法，充分利用单个存储阵列内的全部并行性，并使 n 维快速傅里叶变换的时间复杂度达到 $O(\log n)$。此外，该算法还可在所有存储阵列间并行执行，从而为批量快速傅里叶变换提供高吞吐量。我们利用卷积定理将该算法扩展到阵列内多项式乘法，时间复杂度达到 $O(\log n)$。我们在一款公开可用的周期精确模拟器上评估所提出的算法；该模拟器既能验证正确性，也能测量性能（基于 RACER 架构的参数），并将其与先进 GPU（NVIDIA RTX 3070 和 NVIDIA A100）上的 cuFFT 进行比较。本文的贡献如下：

- 复数运算：将 AritPIM [12] 扩展至向量化存内算术运算，用于 复数。
- 存内快速傅里叶变换：提出了首个完全基于存内操作的高吞吐量FFT算法，用于存内处理，在达到 $O(\log n)$时间复杂度的同时，还支持并行批处理执行。
- 卷积和多项式乘法：将所提出的 FFT 算法扩展到卷积和多项式乘法，同时还利用了额外的优化。
- 评估：展示了 5–15× 倍吞吐量与 4–13× 倍能耗改进（优于 cuFFT） [4]。

本文组织如下。第 2 节介绍数字处理器内存技术的背景，第 3 节介绍傅里叶变换。第 4 节提出存内快速傅里叶变换算法，并在第 5 节通过卷积定理将其扩展至卷积和多项式乘法。第 6 节评估所提出的算法，第 7 节总结全文。

2. 数字存内处理（PIM）

存内处理架构正迅速兴起，通过将计算集成到内存本身中来应对内存瓶颈。存内处理架构使计算机能够在内存中执行向量算术运-算，而无需将数据传输到专用计算单元（例如CPU），从而几乎消-除了计算与内存之间的数据传输。存内处理架构通常基于物理器件，这些器件天然支持信息存储和基-本逻辑运算，为基本按位运算提供大规模并行性，进而扩展到高- 吞吐量的向量存内算术运算。

2.1. 数字忆阻器存内处理

数字忆阻器存内处理架构 [5, 6, 7] 利用忆阻器 [11]: 这是一种双端物理器件（类似电阻），具有可通过电流改变的内部电阻。忆阻器可通过电阻存储二进制信息（通常高电阻表示逻辑零，低电阻表示逻辑一）；写入数据时施加电流，读取数据时施加低电压并测量电流。忆阻器还支持在电阻域中实现数字逻辑 （有状态逻辑 [18]），并以忆阻器本身作为构建模块 [19, 20, 21]。对于如图 2(a)所示电路中的忆阻器，底部忆阻器的状态会根据顶部两个忆阻器的状态而改变（例如，或非 [20]）。

如图 2(b)所示，忆阻交叉阵列的单行中也存在相同电路（垂直方向为位线，水平方向为字线，所有交叉点均设有忆阻器）。因此，通过在位线上施加电压，最多可以让所有行同时在选定的忆阻器之间执行逻辑门运算。该模型的抽象形式是：交叉阵列是一种存储器二进制矩阵，支持对任意列执行按位逻辑，时间复杂度为 $O(1)$（例如，将存储在两列中的数据进行按位或非运算，并将结果存入第三列，只需一个时钟周期），见图 2(c)。此外，通过改为在字线上而非位线上施加电压，还可以对交叉阵列的行执行逻辑运算。存储器由许多交叉阵列组成，每个阵列的大小通常为 1024×1024 [22]。

2.2. 元素并行算术

元素并行算术将数字处理器内存提供的按位并行性扩展到向量算术 [12, 13, 6, 14]。例如，对向量$\boldsymbol{u} = (u_1, \ldots\ldots, u_r)$和 $\boldsymbol{v} = (v_1, \ldots\ldots, v_r)$进行加法，每个元素均为 N位数（定点数或浮点数），首先将这些向量排列在 $r \times c$ 交叉阵列中，其中 i^{th} 行包含 u_i和 v_i，见图 2(d)。元素并行算术通过在每一行中独立计算 $u_i + v_i$，并在所有行之间并行执行，从而计算向量加法 $\boldsymbol{u} + \boldsymbol{v}$。这通过位串行计算（也称为“单行” [13]）完成： N位加法被转换为一系列串行执行的基本逻辑门（例如，或非门），每个逻辑门在每一行内串行执行，但在所有行之间并行执行。其他列可用于存储算术运算所需的中间结果（例如，加法中的进位位）。[2] AritPIM [12] 最近提出了一套适用于定点数和浮点数的算术运算，我们采用了这些算法。

3. 傅里叶变换

傅里叶变换对于从信号处理到卷积 [1, 2, 23]的众多任务都至关重要。离散傅里叶变换（DFT） [24] 作用于复数序列，在频域中生成替代表示。快速傅里叶变换（FFT） [3, 24] 以显著更低的时间复杂度执行 DFT： $O(n \log n)$，而不是 $O(n^2)$。

3.1. 离散傅里叶变换（*DFT*）

离散傅里叶变换（DFT）是一种可逆的线性变换，它根据以下公式将一个 n 维输入序列 $x_0, \ldots, x_{n-1} \in \mathbb{C}$ 转换为 $X_0, \ldots, X_{n-1}$ =DFT($x_0, \ldots, x_{n-1}$)：

$$X_k = \sum_{j=0}^{n-1} x_j \cdot {\omega_n}^{j \cdot k}, \tag{1}$$

其中 $\omega_n = e^{-2\pi i/n}$ 和 $i = \sqrt{-1}$。逆离散傅里叶变换则反向执行这一过程，通过以下公式计算 $x_0, \ldots, x_{n-1}$ = IDFT($X_0, \ldots, X_{n-1}$)：

$$x_j = \frac{1}{n} \cdot \sum_{k=0}^{n-1} X_k \cdot {\omega_n}^{-j \cdot k}. \tag{2}$$

直接计算 DFT 和逆离散傅里叶变换需要 $O(n^2)$ 次运算（$O(n)$ 次运算对应每个 $O(n)$ 个元素），而快速傅里叶变换（FFT）将其总体减少到 $O(n \log n)$ 次运算。

3.2. 快速傅里叶变换（*FFT*）

快速傅里叶变换基于一种递归的分治法，按偶数和奇数索引将输入分开，

$$\begin{aligned} X_k &= \sum_{j=0}^{n/2-1} x_{2j} \cdot {\omega_n}^{2j \cdot k} + \sum_{j=0}^{n/2-1} x_{2j+1} \cdot {\omega_n}^{(2j+1) \cdot k} \\ &= \underbrace{\sum_{j=0}^{n/2-1} x_{2j} \cdot {\omega_{n/2}}^{j \cdot k}}_{\mathrm{DFT}(x_0, x_2, \ldots)_{k \bmod n/2}} + {\omega_n}^{k} \cdot \underbrace{\sum_{j=0}^{n/2-1} x_{2j+1} \cdot {\omega_{n/2}}^{j \cdot k}}_{\mathrm{DFT}(x_1, x_3, \ldots)_{k \bmod n/2}}. \end{aligned} \tag{3}$$

因此，我们发现，DFT($x_0, \ldots, x_{n-1}$) 可以由 DFT($x_0, x_2, \ldots, x_{n-2}$)、DFT($x_1, x_3, \ldots, x_{n-1}$)以及 $O(n)$次额外的加法和乘法递归推导得到，最终实现 $O(n \log n)$ 的时间复杂度：

$$T(n) = 2 \cdot T(n/2) + O(n), \tag{4}$$

$$\implies T(n) = O(n) + \underbrace{2 \cdot O\left(\frac{n}{2}\right)}_{O(n)} + \underbrace{4 \cdot O\left(\frac{n}{4}\right)}_{O(n)} + \cdots, \tag{5}$$

$$\implies T(n) = O(n \log n), \tag{6}$$

因为共有 $\log(n)$ 个元素。图3展示了一种等效的 FFT 算法：通过递归计算较短序列的 FFT，并使用$O(n)$ 蝶形运算 [24] 合并结果。该方法首先对输入进行置换 [24] ，然后执行 $\log n$组 $n/2$ 蝶形运算（每组运算作用于序列中的两个元素）。由于每组包含相互独立的运算，因此可以在并行处理器上以 $O(\log n)$ 时间计算 FFT 算法。

4. 提出的存内快速傅里叶变换

在本节中，我们将并行快速傅里叶变换算法映射到忆阻器交叉阵列。通过充分利用忆阻逻辑提供的并行性，我们实现了批量执行的 $O(\log n)$ 时间复杂度，因为每组蝶形运算的时间复杂度为 $O(1)$，且共有 $\log n$ 组。我们关注 原位 快速傅里叶变换（输出覆盖输入）；如果需要保留输入，则可在开始前将输入复制到不同的列中。我们探讨了在一个 $r \times c$ 交叉阵列中存储输入序列的三种配置：

[2]为简洁起见，正文中不讨论中间算术单元，但会在仿真中对其进行评估。

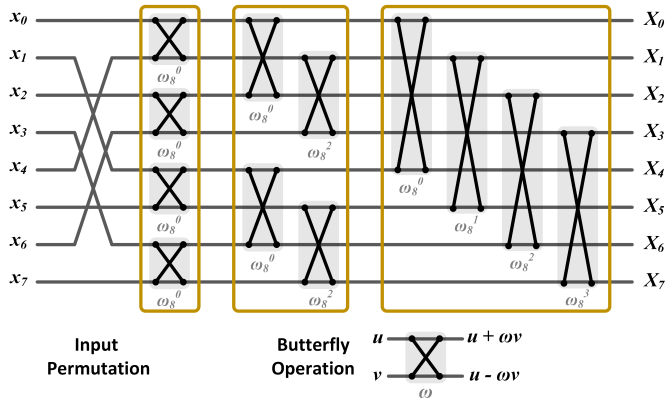


图3： $n = 8$ [24]的FFT算法并行表示。每次蝶形运算同时作用于序列中的两个元素，而每组（黄色） $n/2$ 运算都可以同时执行。

- r-配置 （$n = r$） ： x_0，.....， x_{n-1} 存储在 N 列中，每行存放一个元素。
- $2r$-配置 ($n = 2r$)： $x_0, \ldots, x_{n-1}$ 存储在 $2N$ 列中，每行两个元素（按照蛇形排列组织）。
- $2r\beta$-配置 ($n = 2r\beta$) 为 $\beta > 0$ 常数：输入序列 $x_0, \ldots, x_{n-1}$ 存储在 $2N\beta$列中，每行包含 2β 个元素（蛇形排列）。

本节组织如下。第4.1 节将 AritPIM [12] 扩展到复数运算，第 4.2节提出一种元素并行蝶形运算。最后，第4.3、4.4和4.5 节分别逐步构建针对 r、 $2r$ 和 $2r\beta$ 的所提出算法。

4.1. 复数运算

我们选择以直角坐标形式（a+bi）而非极坐标形式（r 和 θ）表示复数，因为这样可以简化加法。我们将 N 位复数浮点数存储在内存中，形式为两个 $N/2$ 位实数浮点数的串联，分别表示 a 和 b。复数加减法可由实数运算推导得出。

$$(a + bi) \pm (a' + b'i) = (a \pm a') + (b \pm b')i. \quad (7)$$

同样，乘法是按照

$$(a + bi) \cdot (a' + b'i) = (aa' - bb') + (ab' + a'b)i. \quad (8)$$

4.2. 蝶形运算

我们将复数运算扩展为按元素并行的蝶形运算：接收向量 $\boldsymbol{u}$， $\boldsymbol{v}$， $\boldsymbol{\omega}$ （每行存储一个元素），并输出 $\boldsymbol{u} + \boldsymbol{\omega} \odot \boldsymbol{v}$和 $\boldsymbol{u} - \boldsymbol{\omega} \odot \boldsymbol{v}$，其中 $\odot$表示逐元素乘法。该运算通过将 $\boldsymbol{\omega}$ 与 $\boldsymbol{v}$相乘，将结果加到 $\boldsymbol{u}$ 上得到 $\boldsymbol{u} + \boldsymbol{\omega} \odot \boldsymbol{v}$，再从 $\boldsymbol{u}$ 中减去该结果得到 $\boldsymbol{u} - \boldsymbol{\omega} \odot \boldsymbol{v}$。我们还支持原位计算，即用 $\boldsymbol{u} + \boldsymbol{\omega} \odot \boldsymbol{v}$， $\boldsymbol{u} - \boldsymbol{\omega} \odot \boldsymbol{v}$覆盖 $\boldsymbol{u}$， $\boldsymbol{v}$。请注意，由于向量化算术运算支持 $O(1)$ 时间复杂度 [12], ，因此我们发现，向量化蝶形运算的时间复杂度也是 $O(1)$，因为它涉及 $O(1)$ 次算术运算。

到目前为止，我们已经开发出一种高吞吐量的向量化蝶形运算：所有行都可以并行执行蝶形运算——每一行代表一次运算。本节将继续把FFT算法归约为在交叉开关阵列内执行的向量化蝶形运算，从而利用高吞吐量实现FFT加速。

4.3. r–快速傅里叶变换

我们将图3中的算法映射到 r- 配置交叉开关阵列，同时利用元素并行蝶形运算。

该算法的不变量是在图3的各组之间，以 r-配置存储序列x_0，. . . ， x_{n-1}。我们首先按照输入置换步骤对序列进行置换，方法是在交叉阵列的行上执行逻辑运算（例如，交换通过 NOT 门实现）。然后，对于图3中的每个黄色组，我们执行以下步骤（示例如图4所示）：

1.通过将序列的一半向右上方移动（非门），排列序列，使每个蝶形运算的输入位于同一行。 2. 在包含输入的 $r/2$ 行中并行执行蝶形运算。常量 ${\omega_n}^j$ 通过标准写入操作[3]生成。 3. 通过先向下移动，再向左移动，将序列移回 r-配置。

因此，我们发现所有分组均利用元素并行蝶形运算所提供的并行性来执行，而且整个快速傅里叶变换都在交叉开关阵列内完成。不过，这种方案还有两个方面可以改进：（1）占用面积是序列本身的两倍；（2）蝶形运算只能并行处理一半的行。所提出的 $2r$-配置算法利用了这两点，可实现 $n = 2r$ 维快速傅里叶变换，且面积和延迟几乎不变。

4.4. $2r$–快速傅里叶变换

$2r$配置的快速傅里叶变换通过避免组间转换时的中间表示，改进了 r配置的快速傅里叶变换。考虑用于 $n = 16$ 的并行快速傅里叶变换算法（如图 4(c) 所示），而不是 $n = 8$。我们不是从每行存储一个元素开始，而是从每行存储两个元素开始（蛇形格式）——这已经对应于第一次蝶形运算所需的格式。为了在连续蝶形运算的格式之间进行转换，需要在成对元素之间执行原位交换（见图 4(d)）。

[3]这些操作可以像处理常量一样在所有交叉开关阵列中并行执行，而且总体延迟可以忽略不计。

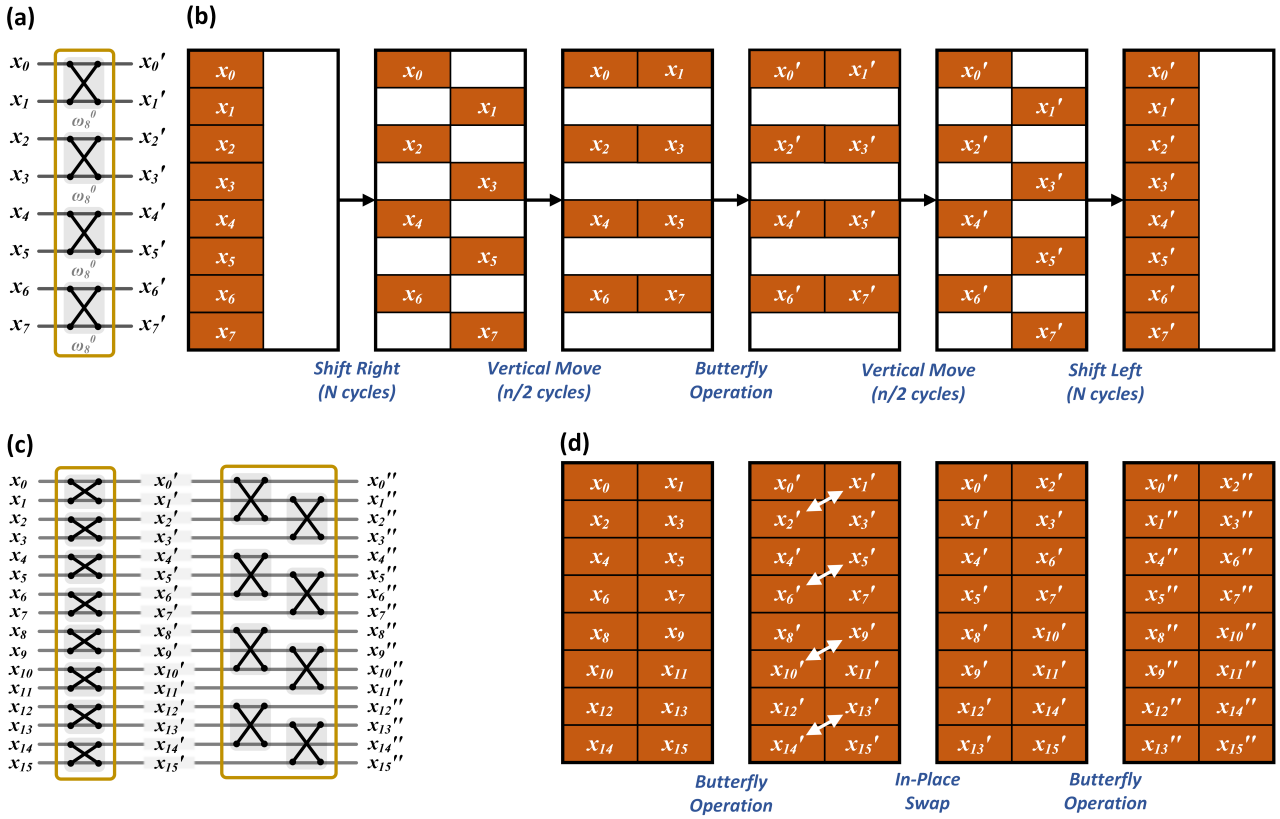


图4：(a, b)：当 $n = 8$时第一组的示例 r–快速傅里叶变换步骤。(c,d)： $n = 16$时第一组与第二组之间的示例 $2r$–快速傅里叶变换转换；这使维度翻倍，同时保持面积相同和相近的延迟。为清晰起见， $x_0, x_1, \ldots, x_{n-1}$ 图中所示内容均已进行输入置换。

4.5. $2r\beta$–快速傅里叶变换

在这种配置中，我们旨在通过将输入序列存储在多对 N 列中，最大化所支持的快速傅里叶变换维度。具体而言，对于 $n = 2r\beta$，我们以蛇形排列将输入存储在 $2N\beta$ 列中。每组 $2N$ 列构成一个执行蝶形运算的单元，而原位交换则在不同单元之间串行执行。我们还采用列操作形式的不同单元之间交换。在标准 $2r\beta$ 模型中，蝶形运算是在各单元之间串行执行的；此外，我们还评估了 $2r\beta$ 的一种扩展，该扩展利用分区 [25]来实现并行执行。

5. 提出的多项式乘法

本节将存内快速傅里叶变换算法扩展至一维卷积和多项式乘法，其时间复杂度为 $O(\log n)$。不失一般性，我们仅讨-论多项式乘法，因为一维卷积与其等价。接下来，我们将介绍卷积定理的背景知识，然后详细说明我们优化后的存内多项式乘法算法。

卷积定理描述了多项式乘法与傅里叶变换 [24] 之间的基本关系。次数为 $n-1$ 的多项式 A 由 $A(x) = \sum_{j=0}^{n-1} a_i x^i$ 定义，而多项式 A 与 B 的乘积是满足以下条件的多项式 C：对于所有 x， $C(x) = A(x) \cdot B(x)$。传统上，多项式乘法通过对系数进行卷积来实现（例如，$(2+3x)*(1+4x) = 2+(8+3)x+12x^2$），耗时为 $O(n^2)$。然而，如果根据多项式在 n 个固定点处的取值而非系数来表示多项式（例如，存储 $A(0)$ 和 $A(1)$，而不是 a_0 和 a_1），那么只需将多项式在这些固定点处的取值相乘，耗时为 $O(n)$。此外，若将固定点选为 n 单位根 ${\omega_n}^0, \ldots, {\omega_n}^{n-1}$，则系数表示与点值表示之间的转换恰好分别对应离散傅里叶变换和逆离散傅里叶变换（注意，式（1）是在 ${\omega_n}^k$ 处计算多项式的值），因此可在 $O(n\log n)$ 时间内完成。总之，

$$(c_0, \ldots, c_{n-1}) = \text{IDFT}(\text{DFT}(a_0, \ldots, a_{n-1}) \odot \text{DFT}(b_0, \ldots, b_{n-1})), \quad (9)$$

其中 $\odot$ 表示逐元素乘法。[4]

我们利用卷积定理和所提出的 FFT 算法，实现快速的存内多项式乘法。我们假设输入多项式

[4]请注意，这会生成一个次数最高为 n 的 C。如果需要次数最高达到 $2n$，则可以在 A 和 B 后额外填充 n 个零。

表 1：评估参数

配置	参数
FourierPIM–8/40（含分区）	内存大小：8/40GB 交叉开关阵列大小：1024×1024 [22] 时钟频率：333.3MHz [5] 门能耗（*ff*）：6.4fJ [5] （分区数量：最多 4。）
cuFFT（RTX 3070）	核心数：5888 内存大小：8GB 内存带宽：448GB/s
cuFFT（A100）	核心数：6912 内存大小：40GB 内存带宽：1,555GB/s

分别以 r、$2r$ 或 $2r\beta$ 配置提供，并执行以下步骤：(1) 计算每个多项式系数的快速傅里叶变换，(2) 计算快速傅里叶变换输出的逐元素乘积，以及 (3) 对该乘积进行逆快速傅里叶变换（逆快速傅里叶变换算法与快速傅里叶变换类似）。请注意，由于我们同时计算快速傅里叶变换和逆快速傅里叶变换，因此无需执行初始输入置换，因为它们会相互抵消 [24]。所提出的快速傅里叶变换具有 $O(\log n)$ 时间复杂度，因此该多项式乘法的时间复杂度也是 $O(\log n)$，因为它涉及常数次快速傅里叶变换，且使用 AritPIM [12] 可在 $O(1)$ 时间内完成逐元素乘积。

我们通过将步骤 (1) 中的两次快速傅里叶变换一并执行，进一步加速实系数多项式乘法，即将两个 n维度实系数快速傅里叶变换合并为一次 n维度复数快速傅里叶变换。具体而言，对于实输入序列 $x_1, \ldots, x_n \in \mathbb{R}$ 和$y_1, \ldots y_n \in \mathbb{R}$，我们根据 $z_k = x_k + i \cdot y_k$定义复数序列 $z_1, \ldots, z_n \in \mathbb{C}$。初始序列的快速傅里叶变换根据以下方式导出，

$$X_k = \frac{1}{2} \cdot \left[\overline{Z_{n-k}} + Z_k\right], \qquad Y_k = \frac{i}{2} \cdot \left[\overline{Z_{n-k}} - Z_k\right]. \quad (10)$$

关于存内实现：（1）复共轭通过翻转虚部的符号位来实现；（2）乘以 i 通过模拟实部和虚部的交换，再对虚部的符号进行反转来实现；（3）除以 2通过将指数减1来实现；（4）Z_{n-k} 的值通过使用交换操作反转 Z 的顺序来导出。

6. 评估

我们将所提出的算法与英伟达 GPU 库 cuFFT [4]进行比较。对于 GPU 基线，我们在 NVIDIA RTX 3070 和 NVIDIA A100 上评估 cuFFT（见表 1），并使用 cudaEvents API 测量吞吐量、使用 nvidia-smi 测量功耗。所提出的算法通过一个

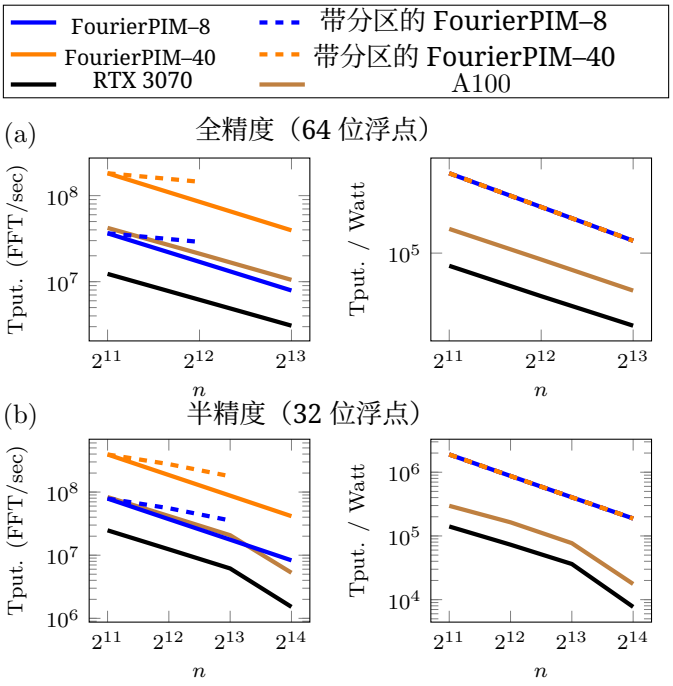


图5：所提出的 FFT 算法与基准 cuFFT 的对比：(a) 全精度；(b) 半精度。

公开的周期精确模拟器[5] 进行验证，该模拟器对忆阻器交叉阵列进行逻辑建模，并执行与所提出算法相对应的一系列操作。[6] 验证正确性时，输入随机数据，并将输出与基线实现的真实输出进行比较；性能则通过模拟器中的计数器测量，这些计数器记录延迟和能耗。根据源自 Bitlet [22] 和 RACER [5], 的数字忆阻器存内处理架构，按批处理设置评估这些算法，如表 1 所示；架构提供 8 GB 和 40 GB 两种配置（分别对应 RTX 3070 和 A100 的内存大小）。此外，我们还评估了最多采用 4 个分区 [25] 的 FourierPIM，以展示分区的潜力——见第 4.5 节。

我们生成了三种基准测试：（1）快速傅里叶变换，（2）多项式乘法，以及（3）实系数多项式乘法。这些基准测试涵盖多种维度 $n \in \{2K, 4K, 8K, 16K\}$，分别对应 $2r$，$2r \cdot 2$，$2r \cdot 4$ 和 $2r \cdot 8$ 种算法[7]，，并针对全精度和半精度浮点数进行测试。GPU 批大小设为能够装入 GPU 内存的最大值，而 Fouri- erPIM 的批大小则为交叉开关阵列的总数。我们比较吞吐量（每秒执行的快速傅里叶变换或多项式乘法次数）和每瓦吞吐量（能耗）。

[5]可访问 https://github.com/oleitersdorf/FourierPIM。[6]由于 FourierPIM 符合抽象计算模型（图 2），外围电路的有效性遵循先前的工作 [5, 6]。[7]在某些情况下（例如使用分区时），维度会受到限制，这是因为单个交叉开关阵列的宽度限制了中间忆阻器面积。

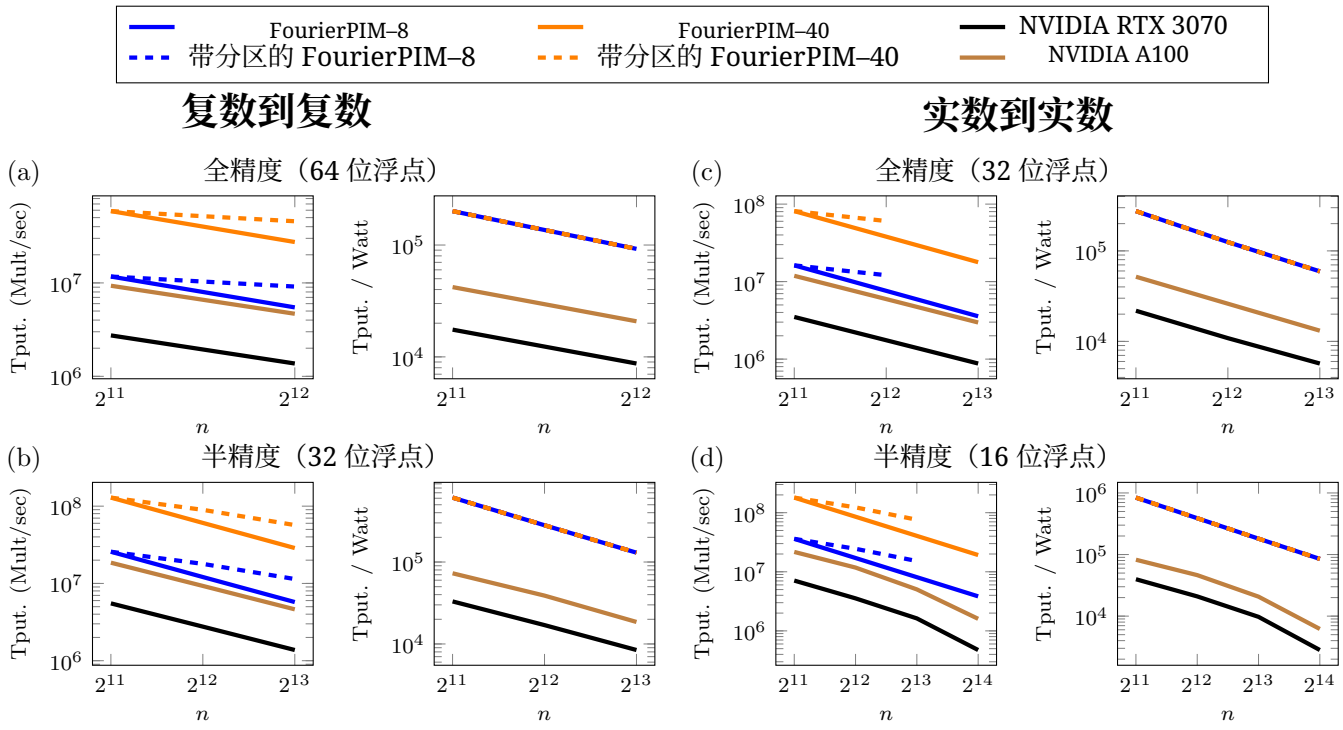


图6：在 NVIDIARTX3070 和A100上，将 (a, b) *complex* 和 (c,d) *real* 多项式乘法与 cuFFT进行比较。请注意，重叠的曲线以交替虚线格式显示，以突出显示两个曲线均存在。

图5展示了快速傅里叶变换基准测试的结果。对于全精度算法，每个复数由 64 位表示（其中每个a 和 b占 32 位浮点）。我们注意到，Fouri- erPIM 和 cuFFT 的吞吐量均随 n近似线性下降，而采用分区的 FourierPIM 吞吐量则随 n 呈对数下降（因为时间复杂度为 $O(\log n)$）。例如，对于全精度，我们发现仅使用两个分区，吞吐量最高可提升 $1.7\times$；并且预计这些结果可推广到具有更多分区的大型交叉开关阵列。这表明忆阻分区有望提高存内计算算法的吞吐量。总体而言，对于采用分区的全精度算法，与 RTX 3070 相比，吞吐量（能耗）最高可提升 $5\times$（$8\times$）；与 A100 相比，吞吐量（能耗）最高可提升 $7\times$（$4\times$）。对于半精度数，与 RTX 3070 和 A100 相比，吞吐量（能耗）分别提升 $6\times$（$13\times$）和 $9\times$（$6\times$）。[8] 比率的提升源于 GPU 的性能仅提升 $2\times$（因为其受内存带宽限制，且数据量缩小了 $2\times$），而存内处理在内存中执行，因此可将计算量减少超过 $2\times$倍。

对于多项式乘法基准测试，我们预计相对提升幅度与快速傅里叶变换大致相同，因为 cuFFT 和 Fouri- erPIM 都利用卷积定理。然而，我们发现相对提升幅度有所增加；例如，与 A100 相比，实数半精度多项式乘法的吞吐量和能耗最高分别提升了 $15\times$ 和 $10\times$，优于快速傅里叶变换中 $9\times$ 和 $6\times$的比值。这一现象是因为 GPU 执行的逐元素乘法也受内存带宽限制，因此它对 GPU 延迟和能耗的影响比对存内处理的影响更大。

7. 结论

傅里叶变换是许多应用的基础，涵盖从信号处理到密码学。因此，本文旨在利用存内计算技术加速快速傅里叶变换（FFT），以避免内存瓶颈。我们首先将存内算术运算扩展到复数，接着展示高吞吐量的存内蝶形运算，然后探讨将 FFT 高效映射到交叉阵列的方法。我们还通过卷积定理，将存内快速傅里叶变换扩展到存内卷积和多项式乘法。随后，我们使用面向所提算法、可公开获取的周期精确模拟器评估结果，以验证正确性并测量性能；并使用 NVIDIA cuFFT 库作为先进 GPU 的基准。我们发现，相较于先进 GPU，吞吐量提升 5–$15\times$ 倍，能耗降低 4–$13\times$ 倍。未来将研究多交叉阵列快速傅里叶变换。

[8] cuFFT 在 $n = 16K$ 处的下降属于另一种线性趋势，该趋势从 $n = 16K$ 开始延续，因为 cuFFT 在中型和大型快速傅里叶变换上的表现有所不同。

参考文献

[1] N. Samardzic、A. Feldmann、A. Krastev、S. Devadas、R. Dres-linski、C. Peikert、D. Sanchez，《F1：一种用于全同态加密的快速可编程加速器》，载于：MICRO-54：第54届 IEEE/ACM 国际微体系结构研讨会，2021年，第238–252页。 doi:10.1145/3466752.3480070。[2] L. Jiang、Q. Lou、N. Joshi，《MATCHA：一种用于环面上全同态加密的快速节能加速器》，载于：第59届 ACM/IEEE 设计自动化大会论文集，

2022， 235 240 doi:10.1145/3489517. 页码：–。3530435

[3]J. W. Cooley、J. W. Tukey，《一种用于机器计算复傅里叶级数的算法》，《计算数学》

19 90 1965 297 301 doi:10.1090/ （） S0025-5718-1965-0178586-1 （）–。[4] 英伟达，《面向英伟达 GPU 的快速傅里叶变换》（2023）。网址：https://developer.nvidia.com/cufft。[5] M. S. Q. Truong、E. Chen、D. Su、L. Shen、A. Glass、L. R. Carley、J. A. Bain、S. Ghose，《RACER：采用电阻式存储器的位流水线处理》，载于：MICRO-54：第54届 IEEE/ACM 国际微体系结构研讨会，2021年，第100–116页。 doi:10.1145/3466752.3480071。[6] N. Talati、S. Gupta、P. Mane、S. Kvatinsky，《利用忆阻器辅助逻辑（MAGIC）在忆阻器存储器中进行逻辑设计》，《IEEE 纳米技术汇刊》15（4）（2016年）635–650。doi:10.1109/TNANO.2016.2570248。[7] S. Kvatinsky，《采用忆阻器存储器处理单元（mMPU）的真正存内处理》，载于：IEEE 第30届专用系统、架构与处理器国际会议（ASAP），第

2160 052 2019， 142 148 doi: 卷-X，第–10.1109/ASAP.2019.00-10 页。[8] N. Hajinazar、G. F. Oliveira、S. Gregorio、J. a. D. Ferreira、N. M. Ghiasi、M. Patel、M. Alser、S. Ghose、J. G´omez-Luna、O. Mutlu，《SIMDRAM：一种利用动态随机存取存储器进行位串行 SIMD 处理的框架》，载于：第26届 ACM 编程语言与操作系统架构支持国际会议论文集，2021年，第329–345页。doi:10.1145/3445814.3446749。[9] Y. Xi、B. Gao、J. Tang、A. Chen、M.-F. Chang、X. S. Hu、J. V. D. Spiegel、H. Qian、H. Wu，《模拟电阻式开关存储器上的存内学习：综述与展望》，《IEEE 会刊》

109 1 2021 14 42 doi: （） （）–。10.1109/JPROC.2020.3004543

[10] R. Cai、A. Ren、Y. Wang、B. Yuan，《基于忆阻器的离散傅里叶变换：提升性能和能效》，载于：IEEE 计算机学会年度超大规模集成电路研讨会（ISVLSI），2016年，第643–648页。 doi:10.1109/ISVLSI.2016.124。[11] L. Chua，《忆阻器——缺失的电路元件》，《IEEE 电路理论汇刊》

18 5 1971 507 519 doi:10.1109/ （） （）–。TCT.1971.1083337

[12] O. Leitersdorf、D. Leitersdorf、J. Gal、M. Dahan、R. Ronen、S. Kvatinsky，《AritPIM：高吞吐量存内算术运算》，arXiv 预印本 arXiv:2206.04218（2022年）。[13] R. Ben-Hur、R. Ronen、A. Haj-Ali、D. Bhattacharjee、A. Eli-ahu, N. Peled,S.Kvatinsky, SIMPLER MAGIC：在单行中执行的存内逻辑的综合与映射，以提升吞吐量，《IEEE计算机辅助设计的集成电路与系统学报》39(10) (2020) 2434–2447。doi:10.1109/TCAD.2019.2931188.[14] A.Haj-Ali, R. Ben-Hur,N.Wald, S.Kvatinsky，使用MAGIC进行存内定点乘法的高效算法，载于：IEEE国际电路与系统研讨会（ISCAS），2018年，第1–5页。doi:10.1109/ISCAS.2018.8351561。[15] H.Nejatollahi, S. Gupta, M. Imani,T. S.Rosing, R.Cam- marota,N.Dutt，CryptoPIM：基于格的密码硬件的存内加速，载于：第57届ACM/IEEE设计自动化会议（DAC），

2020， 1 6 doi: 第–10.1109/DAC18072.2020.9218730 页。。[16] D.Li,A. Pakala,K.Yang，MeNTT：紧凑且高效的存内处理数论变换（NTT）加速器，《IEEE超大规模集成（VLSI）系统学报》

30 5 2022 579 588 doi:10.1109/TVLSI. ( ) ( )–2022.3151321 。。[17] H. Cilasun, S. Resch, Z. I. Chowdhury,E. Olson,M. Zabihi, Z. Zhao, T. Peterson,J.-P. Wang, S. S.Sapatnekar,U. Karpuzcu, CRAFFT：自旋电子计算型随机存取存储器中的高分辨率快速傅里叶变换加速器，载于：第57届ACM/IEEE设计自动化会议（DAC），

2020， 1 6 doi:10.1109/ 第–DAC18072.2020.9218673 页。。[18] J. Reuben, R. Ben-Hur, N.Wald, N. Talati, A. H. Ali,P.-E. Gaillardon,S. Kvatinsky，忆阻逻辑：用于评估和比较的框架，载于：第27届功耗与时序建模、优化和仿真国际研讨会（PATMOS），2017年，第1–8页。 doi:10.1109/PATMOS.2017.8106959。[19] J. Borghetti, G. S. Snider, P. J. Kuekes, J. J. Yang, D.R.Stew-art, R.S.Williams，“忆阻”开关通过物质蕴涵实现“有状态”逻辑运算，《自然》464 (7290) (2010) 873–876。[20] S. Kvatinsky,D.Belousov, S. Liman, G. Satat, N. Wald, E. G. Friedman, A.Kolodny, U.C. Weiser，MAGIC——忆阻器辅助逻辑，《IEEE电路与系统学报II：简报》

61 11 2014 895 899 doi:10.1109/TCSII.2014. ( )( )–2357292 。。[21] S. Gupta,M.Imani,T. Rosing，FELIX：快速且高能效的存储器内逻辑，载于：IEEE/ACM计算机辅助设计国际会议（ICCAD），2018年，第1–7页。doi:10.1145/3240765.3240811.[22] R. Ronen, A. Eliahu, O. Leitersdorf, N. Peled,K.Korgaonkar, A. Chattopadhyay, B. Perach, S. Kvatinsky，bitlet模型：用于比较存内处理与CPU系统的参数化分析模型，《新兴计算技术杂志》18 (2)（2022年3月）。doi:10.1145/3465371.[23] H.J.Nussbaumer，快速傅里叶变换，载于：《快速傅里叶变换与卷积算法》，施普林格，1981年。[24] T. H. Cormen, C. E. Leiserson,R. L.Rivest, C. Stein，算法导论，麻省理工学院出版社，2022年。[25] O. Leitersdorf, R. Ronen, S. Kvatinsky, PartitionPIM：用于快速存内处理的实用忆阻分区，arXiv预印本 arXiv:2206.04200（2022年）。