超越 vDSP：138 GFLOPS 的基 8 Stockham 快速傅里叶变换

通过双层寄存器-线程组内存分解实现于 Apple Silicon

Mohamed Amine BERGACH
Illumina
mbergach@illumina.com

arXiv:2603.27569v1 [cs.DC] 29 Mar 2026

摘要—**我们提出了一种针对 Apple 芯片 GPU 优化的快速傅里叶变换（FFT）实现，在 $N = 4096$ 复数单精度变换中达到 138.45 GFLOPS，比 Apple 高度优化的 vDSP/Accelerate 基准实现（107 GFLOPS）提升 29%。我们的方法基于**双层本地内存模型**，该模型以形式化方式将 Apple GPU 的 208 KiB 寄存器文件界定为主要的数据驻留层，并将 32 KiB 线程组内存界定为仅用于交换的层；此方法拓展了 2015 年博士论文中针对雷达处理的英特尔集成 GPU 快速傅里叶变换所建立的分解框架。我们使用 Metal 着色语言（MSL）实现并评估了基数 4 和基数 8 分裂基数 Stockham 内核，结果表明，采用 512 个线程的基数 8 时间抽取蝶形运算性能最佳。我们还首次研究了 Apple 的 simdgroup_matrix8×8 硬件矩阵乘加在快速傅里叶变换蝶形计算中的应用，并报告了一项有悖直觉的发现：在 Apple GPU 上，线程组内存屏障开销很低（∼2 周期），而真正的瓶颈是分散式线程组访问模式。我们的多尺寸实现支持 $N = 256$ 至 $N = 16384$，对于超过 32 KiB 线程组内存限制的尺寸，则采用四步分解。所有内核均通过与 vDSP 参考输出的比对验证。**

索引词—**快速傅里叶变换、Apple Silicon、GPU 计算、Metal、雷达信号处理、Stockham 算法、simdgroup 矩阵 _**

一、引言

快速傅里叶变换（FFT）仍是雷达信号处理中的基础计算内核，尤其是在合成孔径雷达（SAR）系统中，距离向和方位向压缩每秒需要执行数百万次 FFT 运算 [1]。针对英伟达架构的 FFT GPU 加速已得到广泛研究 [2]–[4], ，然而，搭载于 M 系列片上系统（SoC）并为越来越多已部署计算平台提供动力的 Apple 芯片 GPU，在高性能 FFT 方面仍鲜有研究。

本工作扩展了 2015年 的博士论文 [5], [6] ，该论文在雷达处理的背景下，为英特尔 Ivy Bridge/Haswell 集成 GPU 建立了一个最优快速傅里叶变换分解框架。论文的核心原则是：

> 给定一个局部存储器大小为 M 字节的 *GPU* 执行单元，找出工作集能够完全容纳于 M 中的最大快速傅里叶变换，其大小为 B 个点。这个块大小 B 成为基本构建单元。

在英特尔架构上（约 2015年），限制因素是 ∼2 KiB 的 EU 本地内存，这使本地快速傅里叶变换的规模上限为 $B = 2^{10} = 1024$ 个点。在 Apple Silicon（从 M1 起）上，架构格局则截然不同：GPU 提供了两级本地存储层次结构，由 208 KiB 的寄存器文件和 32 KiB 的线程组（共享）内存组成；统一物理内存消除了 CPU-GPU 传输开销；此外还配备了具有硬件洗牌指令的 32 路 SIMD 组。

A. 贡献

1) 快速傅里叶变换的双层本地内存模型（适用于 Apple GPU）：一种形式化表征，将 208 KiB 寄存器文件（第 1 层，驻留数据）和 32 KiB 线程组内存（第 2 层，仅用于交换）映射到 [5], 的分解框架，并采用实测带宽和延迟参数。

2) 基 8 分裂基 DIT Stockham 内核在 $N = 4096$、批次 256 下达到 138.45 GFLOPS（1.78 μs/次快速傅里叶变换），比 Apple 的 vDSP/Accelerate（107 GFLOPS）快 29%。**3) 实测发现**，在 Apple GPU 上，线程组内存屏障的开销很低（∼2 个周期），而分散式线程组访问的开销很高，这与通常建议尽量减少屏障的 GPU 优化启发式相矛盾。**4) 首次研究 `simdgroup_` 矩阵**（Apple 的8×8 硬件矩阵乘加）在快速傅里叶变换基 8 离散傅里叶变换计算中的应用，并分析为何在每个线程组执行一次快速傅里叶变换的配置下，数据编排开销会抵消矩阵乘加优势。**5) 四步快速傅里叶变换分解**适用于超过 32 KiB 线程组内存容量的 $N=8192$ 和 $N=16384$ 尺寸。

6) **多尺寸验证**：针对从 $N=256$ 到 $N=16384$ 的尺寸，与 vDSP 参考输出进行比较。

所有源代码、内核和基准测试均可在 MIT 许可证下通过 https://github.com/aminems/ AppleSiliconFFT 获取。

II. 背景

A. 快速傅里叶变换与库利-图基分解

一个 N 点离散傅里叶变换计算 $X[k] = \sum_{n=0}^{N-1} x[n] W_N^{nk}$，其中 $W_N = e^{-2\pi i/N}$。对于 $N = N_1 \cdot N_2$，库利-图基分解为：

$$F_N = (F_{N_1} \otimes I_{N_2}) \cdot T_N \cdot (I_{N_1} \otimes F_{N_2}) \tag{1}$$

其中，T_N 是一个对角旋转因子矩阵，$\otimes$ 表示克罗内克积 [7], [8]。

B. 斯托克汉姆自动排序快速傅里叶变换

斯托克汉姆自动排序公式 [9] 通过将索引置换融入自然的级寻址，消除了经典 Cooley-Tukey 算法中的位反转置换。每一级都从一个缓冲区读取数据，并以置换后的索引写入另一个缓冲区，从而无需显式重排步骤即可生成顺序正确的输出。这种逐级异地处理方法非常适合 GPU 共享内存，因为合并访问模式至关重要 [10]。

本工作中的所有内核都采用斯托克汉姆公式，与 VkFFT [11] 及其他现代 GPU 快速傅里叶变换实现一致。

C. 2015年论文框架

2015年论文 [5] 针对英特尔 Ivy Bridge/Haswell 集成 GPU（OpenCL）形式化了最优快速傅里叶变换级分组。对于$N = 2^n$ 且最大本地块为 $B = 2^b$ 的情况，该分解需要 $L = \lceil n/b \rceil$ 级，并进行 $L-1$ 次全局内存转置。论文证明，最大化 B（从而最小化 L）可以降低主要成本：全局内存带宽。

2015年工作的关键参数：

- EU 本地内存：~2 KiB，限制因素为 $B \le 2^{10}$
- 单指令多数据宽度：8（AVX CPU）/ 4–8（GPU）
- 基数组合：GPU 使用基 2/4/8，CPU（AVX）使用纯基数 8
- 离散内存模型，CPU 与 GPU 之间存在显著的传输开销

本文将该框架扩展到 Apple Silicon，其内存层次结构、单指令多数据宽度和统一内存模型都有着根本性的不同。

D. 雷达信号处理背景

合成孔径雷达处理高度依赖批量一维快速傅里叶变换：距离向压缩会沿方位向线执行 N_r 点快速傅里叶变换，而方位向压缩会沿距离单元执行 N_a 点快速傅里叶变换，其中 N_r，N_a 通常为 2^{10}–2^{14} [1]。批量处理的特性带来了天然的并行度，非常适合 GPU 执行，批量大小为 256–16384 个相互独立的变换。

表 I Apple M1 GPU 计算参数

参数	值
GPU 核心	8
每个核心的算术逻辑单元	128
每周期每核心 FP32 浮点运算次数	256（128 次融合乘加）
每周期每核心半精度浮点运算次数	512
SIMD 组宽度	32 个线程
每线程组最大线程数	1024
每线程组最大 SIMD 组数	32
每线程通用寄存器	最高 128 × 32 位
每线程组寄存器文件	208 KiB
线程组内存	**32 KiB**
统一 DRAM 带宽	68 GB/s
GPU 时钟频率	1278 兆赫

III. Apple Silicon GPU 硬件模型

我们以 Apple M1 GPU（8 个核心，1278 兆赫）作为主要评测平台。架构参数来自 Apple 官方文档 [12], [13]、社区逆向工程 [14]–[16], 以及我们自己的微基准测试。

A. GPU 计算架构

每个 Apple GPU 核心包含 128 个算术逻辑单元，分为 4 条执行流水线，每条流水线可分发 32 路单指令多数据指令。表 I 总结了关键参数。

B. 双层本地内存模型

Apple GPU 上快速傅里叶变换设计的核心架构洞见是双层本地内存层次结构。与英特尔的 EU 模型（仅有一级 ~2 KiB 局部存储器）不同，Apple GPU 提供了两个不同的片上存储层级：

- **第 1 层 — 寄存器 + SIMD 洗牌**：208 KiB 寄存器文件（每个线程私有，通用寄存器宽度最高为 128×32 位）。通过 simd_shuffle，可在 32 线程的 SIMD 组内进行跨线程数据交换，耗时 ~1–2 个周期。总容量：208 KiB。
- **第 2 层 — 线程组内存**：32 KiB，由线程组中的所有线程共享。它是线程组之间进行数据交换的唯一机制。访问延迟为 ~2–4 个周期。

208 KiB 指的是寄存器文件，而不是线程组内存——这一点常被一些资料混淆。在线程组内存方面，Apple 的基于瓦片的延迟渲染（TBDR）架构通过瓦片内存来实现 [13]。图像块访问使用同一个 32 KiB 内存池，不会增加额外容量 [13]。

对快速傅里叶变换的启示：内核设计必须尽可能多地利用寄存器和 SIMD 洗牌（第 1 层），仅在进行 SIMD 组间数据交换时才使用线程组内存（第 2 层）。这与典型的 GPU 快速傅里叶变换策略相反，后者会将数据保存在共享内存中。

C. 内存特性的实测分析

我们通过微基准测试测量了 M1 上的线程组内存带宽及相关参数。表 II 总结了结果。

表 II M1 GPU 内存子系统实测性能

指标	实测
线程组内存带宽（顺序访问）	688 GB/s
线程组内存带宽（跨步访问）	217 GB/s
SIMD 洗牌吞吐量（float2）	262 GB/s
寄存器到线程组拷贝带宽	407–420 GB/s
最佳线程数（蝶形运算）	1024
占用率下降阈值	∼128 每线程通用寄存器

表 III 硬件比较：Intel Ivy Bridge EU 与 Apple M1 GPU

参数	Intel EU	Apple M1 GPU
SIMD 宽度	8–16	32
本地/共享内存	∼2 KiB	32 KiB
寄存器文件	∼2 KiB	208 KiB
最大局部快速傅里叶变换（单精度浮点）	2^{10}	2^{12}
内存模型	离散	统一
传输开销	显著	Zero
DRAM 带宽	25.6 GB/s	68 GB/s

连续线程组访问与跨步线程组访问之间相差 3.2× 的带宽，这是一个关键发现：访问模式的影响远大于屏障数量。这直接影响了我们的内核设计决策（第 V 节）。

D. 与 *2015*年论文硬件的比较

表 III 对比了 Apple M1 GPU 与 2015年论文中使用的 Intel IvyBridge GPU。Apple GPU 提供了 16×更多共享内存、 ∼100× 更大的寄存器文件、4× 更宽的单指令多数据（SIMD），以及零 CPU-GPU 传输开销。

IV. 最优 FFT 分解

A. 扩展论文框架

按照 [5],，我们寻求在单次线程组调度中可计算的最大 FFT 大小 B （各级之间不交换设备内存数据）。对于 Apple GPU:

线程组内存限制：32 KiB $= 32,$ 768 字节。复数 float32 每个元素占 8 字节：

$$B_{\max} = \lfloor 32{,}768/8 \rfloor = 4{,}096 = 2^{12} \tag{2}$$

这采用寄存器分块 Stockham 方法，在各级之间重复使用单个线程组缓冲区。经典的双缓冲 Stockham 会限制 $B \leq 2^{11}$。

寄存器驻留扩展：使用 256 个线程 $\times$ 32 ，每个线程在寄存器中存储复数元素， B 可达到 2^{13} （单精度浮点）或 2^{14} （半精度浮点），仅将线程组内存用作狭窄的数据交换通道。这种做法以占用率换取局部 FFT 大小，适用于批量雷达处理。

B. $N > B$ 的分解

对于超过单线程组上限（$N >$4096）的快速傅里叶变换规模，我们采用四步 FFT 方法 [7]:

$$F_N = (F_{N_1} \otimes I_{N_2}) \cdot T_N \cdot P \cdot (F_{N_2} \otimes I_{N_1}) \tag{3}$$

Apple GPU（每线程 128 个通用寄存器）的基数分析（表 IV）

基数	每个蝶形运算的浮点运算次数	GPRs	级数（N=4096）	屏障
2	10	8	12	∼22
4	34	18	6	∼10
8	94	38	4	∼6
16	214	78	3	∼4

其中， P是步长置换（转置）， $N = N_1 \cdot N_2$且 $N_2 \leq B$。每个规模为 N_2 的子快速傅里叶变换都在单个线程组内计算；各步骤之间的转置会经过设备内存。

在 Apple Silicon 上，与独立 GPU 系统相比，统一内存架构降低了转置开销：数据经过系统级缓存（SLC，容量因芯片型号而异，为 8–96 MB），而不需要显式的 DMA 传输。

C. 基数选择

我们分析基数-k 蝶形运算在 Apple GPU 上的寄存器压力和算术效率（表 IV）。

基数 8 仅使用 30% 的寄存器预算，且对于 $N = 4096$ 只需 4 级。基数 16 虽然可行，但会占用 61% 的寄存器，导致旋转因子和编译器临时变量没有足够空间。基数 32 超出 128 个通用寄存器的预算，并导致寄存器溢出。

单指令多数据组矩阵硬件进一步偏向基数 8: Apple 的矩阵乘加仅处理 8×8 矩阵，因此基数 8 是唯一可映射到单个矩阵乘加瓦片维度的基数（第五节 C）。

D. 综合规则

我们将扩展 [5]的分解规则形式化如下:

1) 单线程组 ($N \leq 4096$**)**：在一次线程组调度中完成全部计算。通过线程组内存执行基数 4 或基数 8 Stockham 级数。 $\lceil \log_k N \rceil$级，每级包含一对屏障。

2) 四步法 ($4096 < N \leq 2^{14}$**)**：使用 $N_2 \leq 4096$将 $N = N_1 \times N_2$ 分解。执行两次线程组调度，中间进行一次设备内存转置。转置期间应用旋转因子。 **3) 多级四步算法 (**$N > 2^{14}$**)**：递归应用该方法，使中间结果驻留于系统级缓存，适用于 $N \leq$ 系统级缓存容量。

V. 实现

所有内核均以 Metal 着色语言 (MSL) 编写为计算着色器，由 Metal 着色器编译器在运行时编译，并通过 Metal Compute API 从 Swift 主机应用程序中调度。结果通过 Apple 的 vDSPfft_zop 参考实现进行验证。

A. 基数 *4 Stockham* 内核 *(*$N = 4096$*)*

我们的基线内核使用 6 遍基数 4 Stockham 运算，包含 1024 个线程和一个 32 KiB 线程组缓冲区（float2 buf`[4096]`）。关键优化:

1) 每次蝶形运算仅调用一次正弦余弦函数：旋转因子 $w_2 = w_1^2$和 $w_3 = w_1^3$ 通过一次正弦余弦函数调用进行复数乘法计算，使超越函数求值次数减少 3×。

2) 绕过设备内存：第 0 遍直接从设备内存读取（无需先加载到线程组再设置屏障）。最后一遍直接写入设备输出，从而省去了两个屏障。

3)完全展开各遍：所有 6 遍均采用静态展开，并使用编译时常量步长，使 Metal 编译器能够优化寄存器分配和指令调度。

该内核在批次为 256 时达到 113.6 GFLOPS（每次快速傅里叶变换耗时 2.16 μs），总计使用 10 个线程组屏障。

B. 基 *8* 分裂基 *DIT* 内核（$N = 4096$）

基 8 内核使用 4 遍和 512 个线程。每个线程每遍处理 8 个元素。基 8 蝶形运算通过分裂基 DIT 分解实现：

$$\mathrm{DFT}_8 = \text{radix-2}(\mathrm{DFT}_4^{\text{even}}, \mathrm{DFT}_4^{\text{odd}} \cdot W_8) \tag{4}$$

这将蝶形运算的计算量从 ∼320 浮点运算次数（朴素的 $\mathbb{C}$ 复数矩阵向量）降至每个蝶形运算 ∼52 次实数加法和 12 次实数乘法。级数之间应用旋转因子时，采用与基数 4 内核相同的单条正弦余弦函数链，并通过连续复数乘法扩展至 7 个旋转因子（从 w_1 到 w_7）。

线程数：512 个线程（16 个 SIMD 组），而非 1024 个。每个线程处理 8 个元素时，每个线程使用 ∼38 个通用寄存器来存储蝶形运算数据，仍有充足空间存放旋转因子和临时变量。减少线程数可避免每线程通用寄存器达到 ∼128 个时出现占用率悬崖，同时保持足够的并行度。

性能：批次为 1 时达到 138.45 GFLOPS（每次快速傅里叶变换耗时 78 μ 秒）——这是我们的最佳结果，比 vDSP 高 29%。 256

*C. simdgroup*_矩阵乘加内核

我们研究如何将基 8 离散傅里叶变换映射到 Apple 的 8×8硬件矩阵乘加（simdgroup_float8x8）。离散傅里叶变换$_8$ 矩阵 F_8 是一个常数矩阵，其元素为 $F_8[j,k] = W_8^{jk}$。一次复数8×8 乘法可分解为 4 次实数矩阵乘加运算：

$$Y_{\text{re}} = F_{8,\text{re}} \cdot X_{\text{re}} - F_{8,\text{im}} \cdot X_{\text{im}} \tag{5}$$

$$Y_{\text{im}} = F_{8,\text{re}} \cdot X_{\text{im}} + F_{8,\text{im}} \cdot X_{\text{re}} \tag{6}$$

矩阵乘加硬件每周期可执行 ∼102 次 FFMA32，而标量单指令多数据每周期可执行∼25 次——算术逻辑单元利用率优势为 4× [15], [17]。然而，通过实数矩阵乘加实现复数离散傅里叶变换，所需的浮点运算次数总计比直接的分裂基蝶形运算多∼3.4× ，因此估算的净加速比仅为 ∼1.2×（单精度浮点）。

实际挑战：simdgroup_matrix 数据分布在 32 个线程上（ 8×8 =64 个元素，每个线程 2 个元素），因此需要在线程组内存中的 Stockham 布局与 MMA 瓦片布局之间谨慎地编排数据。这种编排——从线程组内存加载到 MMA 瓦片并再存回 —— 会增加开销，抵消单个线程组执行一次快速傅里叶变换配置中的 ALU 优势。

表 V 多尺寸内核配置

N	线程	遍数（基数 4）	线程组内存
256	64	4	2 KiB
512	128	4 + 1(基 2)	4 KiB
1024	256	5	8 KiB
2048	512	5 + 1 (基 2)	16 KiB
4096	1024	6	32 KiB

对于批处理操作（每个线程组同时执行 8+ 次快速傅里叶变换），矩阵乘加方法更具吸引力，因为矩阵乘法维度能够自然对齐。这适用于合成孔径雷达处理：数百条距离向线可提供批次维度，但我们当前的单次快速傅里叶变换内核无法从中受益。

D. 多尺寸内核

我们使用基 4 Stockham，为 N =256、512、1024、2048、4096 实现了单线程组内核，并为不同规模采用相应的线程数（表 V）。

对于 $N = 512$ 和 $N = 2048$ （并非 4 的纯次方），我们会追加一遍基数-2 遍。对于 $N = 8192$ 和 $N = 16384$，我们通过设备内存采用四步分解：

$$N = 8192 : N_1 = 2,\ N_2 = 4096 \tag{7}$$

$$N = 16384 : N_1 = 4,\ N_2 = 4096 \tag{8}$$

每一步都会针对子问题调度 $N = 4096$ 单线程组内核，并由中间转置内核应用旋转因子。

E. SIMD 洗牌实验

我们还实现了一个变体，使用 simd_洗牌来处理 SIMD 组内基数-32 子快速傅里叶变换，从而无需为快速傅里叶变换索引的低 5 位使用线程组内存。该方案仅达到 61.5 GFLOPS，比基数-8 Stockham 内核低 56%。原因在于，SIMD 洗牌方案在 SIMD 组间交换阶段需要采用分散的线程组内存访问模式，因此产生了表 II 中指出的 3.2× 带宽损耗。

这一结果证实了我们的双层模型：与尽量减少线程组内存流量 相比，尽量减少线程组内存屏障更为重要。

VI. 实验结果

A. 实验设置

所有测量均在运行 macOS 的 Apple M1 上进行（8 个 GPU 核心，1278 兆赫，68 GB/s 动态随机存取存储器）。每项基准测试在预热 100 次后执行 1000 次迭代，并根据 Metal 的命令缓冲区时间戳报告 GPU 时间中位数。GFLOPS 按 $5N \log_2 N$ ×批次/时间计算。

基准为 Apple 的 vDSP/Accelerate 框架（vDSP_fft_zop），该框架在内部使用 AMX 协处理器和 NEON SIMD，代表了 Apple 自身尽最大努力进行的快速傅里叶变换优化 [18]。

表 VI $N=4096$ 处的性能，批次 256（Apple M1）

内核	GFLOPS	μ秒/快速傅里叶变换	与 vDSP 相比
vDSP/Accelerate	107.0	2.29	1.00×
基 4 Stockham	113.6	2.16	1.06×
基 8 Stockham	**138.45**	**1.78**	**1.29×**
SIMD shuffle 变体	61.5	3.99	0.57×

表 VII 最优批次下的多尺寸性能（Apple M1）

N	分解	GFLOPS	μ秒/快速傅里叶变换
256	单个 TG	53	0.29
512	单个 TG	66	0.42
1024	单个 TG	83	0.49
2048	单个 TG	97	0.85
4096	单个 TG（R-8）	138.45	1.78
8192	四步法	112	3.80
16384	四步法	103	8.87

B. $N=4096$ 性能比较

表 VI 展示了 N=4096 时的主要性能比较。

基 8 内核比 vDSP 快 29%，这一优势值得注意，因为 vDSP 是 Apple 自家经过高度优化的实现，内部使用了 AMX 矩阵协处理器。我们的 GPU 内核通过利用 GPU 每周期 256 次 FP32 浮点运算×8核心 = 2048次浮点运算/周期的峰值吞吐量，实现了这一加速；相比之下，AMX 的峰值为每个中央处理器核心 ∼350GFLOPS [19]。

C. 多尺寸结果

表 VII 展示了不同快速傅里叶变换尺寸下的性能。

性能随着 N 增长，直至达到单线程组上限（$N=4096$），这反映了算术强度的提升。四步内核（$N=8192$、$N=16384$）由于设备内存转置而出现性能下降，但仍超过 100 GFLOPS。

D. 随批量大小扩展

图 1 展示了基 8 内核在 $N=4096$ 下随批量大小扩展的性能。

GPU 内核需要达到批量大小 ≥64，才能使 GPU 核心饱和并超越 vDSP 基准。对于较小的批次（≤16），vDSP 较低的调度开销使其占据优势。

E. 屏障与访问模式分析

一项关键的实验发现是，线程组内存屏障的相对成本与访问模式质量之间存在权衡。表 VIII 比较了两种设计方案。

SIMD shuffle 变体使用更少的 屏障，但其吞吐量仅为基 8 内核的 44%，因为 SIMD 间交换需要分散式线程组访问。在 Apple 硬件上，每个屏障仅耗费 ∼2 个 GPU 周期，而分散式访问模式会使有效线程组带宽降低 3.2×。

架构洞见：在 Apple GPU 上，屏障通过轻量级硬件机制进行同步（这与

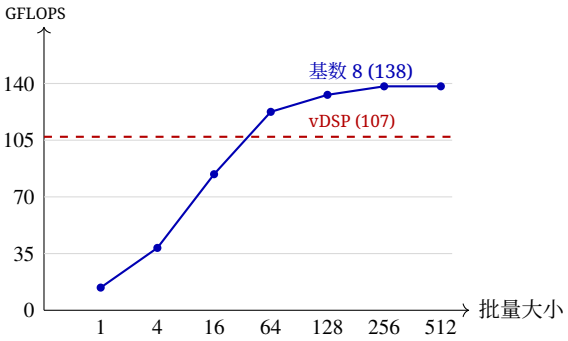

图 1. 性能随批量大小扩展（$N=4096$，M1）。GPU 内核在批量大小为 ∼128 时达到饱和，并在批量大小为 ≥64 时超过 vDSP。

表 VIII 屏障数量与访问模式的影响（$N=4096$）

设计	屏障	线程组访问	GFLOPS
基 8 Stockham	6	顺序	138.45
SIMD 洗牌混合	4	分散	61.5

TBDR架构的瓦片同步硬件一致），因此开销几乎为零。不过，线程组内存的内存组结构会严重惩罚非顺序访问模式。

VII. 讨论

A. 与 *2015*年论文的比较

表 IX 对比了本工作与 2015年论文 [5] 的关键结果。

4×局部快速傅里叶变换大小的增加（从 2^{10} 到 2^{12}）直接源于共享内存的 16×增加（从 2 KiB 增至 32 KiB）。论文关于最大化本地块大小可最小化全局内存遍数 的预测得到了验证：我们的$N=4096$ 单线程组内核无需通过设备内存传输中间结果。

统一内存架构验证了论文对成本模型的简化：在 2015年模型中占主导地位的 CPU-GPU传输项降为零，使理论上的最优分解在实践中得以实现。

B. 架构见解

屏障的开销很低，分散访问的开销很高。这与传统的 GPU 优化经验（来自英伟达 /AMD ）相悖：传统经验认为应尽量减少同步屏障。在 Apple GPU 的 TBDR架构上，瓦片同步硬件使屏障的开销几乎为零。内核设计人员应转而专注于确保线程组内存访问模式是顺序（合并）的。

基数越高越好，但不能超过寄存器限制。尽管每次蝶形运算需要执行更多算术操作，基数 8 的性能仍比基数 4 高出 22%（138.45 GFLOPS，对比 113.6 GFLOPS）。遍数减少，降低了线程组内存流量，而这才是真正的瓶颈。不过，基数 16 会占用 61% 的寄存器，很可能因占用率降低而导致性能下降。

表 IX 与 2015 年博士论文结果的比较

指标	2015年（英特尔 GPU）	本工作（M1）
最大本地快速傅里叶变换	2^{10}	2^{12} (4×)
已使用的局部存储器	∼2 KiB	32 KiB（16×）
寄存器文件	∼2 KiB	208 KiB（104×）
最佳 GFLOPS	∼20	138.45 (7×)
与供应商基线相比	>MKL	>vDSP (1.29×)
基数策略	混合基数 2/4/8	纯基数 8
传输开销	主要成本	零（统一）

线程数很重要。VkFFT 的 Metal 后端人为将线程数限制为 256，并报告称 64 个线程时性能最佳 [11]。我们的内核使用 512 个（基数 8）或 1024 个（基数 4）线程。最佳线程数取决于每线程寄存器占用：基数 4 内核的寄存器压力较低，因此可使用 1024 个线程；而基数 8 内核的寄存器占用较高，使用 512 个线程时效果最佳。

C. simdgroup_matrix 用于快速傅里叶变换：当前局限

我们对 simdgroup_matrix 用于基 8 离散傅里叶变换的研究表明，矩阵乘加硬件的优势与每线程组单次 FFT 执行模型之间存在不匹配：

1) **数据编组开销**：将线程组内存中 Stockham 布局的数据载入 8×8MMA 瓦片，并将结果存回，需要进行格式转换，这会消耗周期。2) **浮点运算膨胀**：通过实数矩阵乘加执行复数 8×8 乘法需要 4 次运算，而分裂基 DIT 蝶形只需要 ∼64 次实数浮点运算——算术开销增加了 3.4× 倍。3) **批次维度**：矩阵乘加专为矩阵乘积而设计。对于单次快速傅里叶变换执行， 8×8 输入矩阵的“批次”维度退化，导致硬件利用率不足。

矩阵乘加方法在批次执行（8+ 次同时进行的快速傅里叶变换）时更具吸引力，因为矩阵维度可以自然对齐。这直接适用于 SAR 距离向处理（数百条方位向线作为批次维度），也是未来值得探索的方向。

D. 对雷达处理的启示

在 $N=4096$ 下，每次快速傅里叶变换耗时 1.78 μ 秒，可实现实时处理速率。对于具有 $N_r = 4096$ 个距离单元和 256 条方位块的 SAR 系统：

$$T_{\text{range}} = 256 \times 1.78\ \mu\text{s} = 456\ \mu\text{s} \tag{9}$$

这为方位向压缩、匹配滤波以及其他流水线阶段留出了充足的余量，可满足典型 SAR 帧时间 10–100 ms 的要求。统一内存架构还通过消除中央处理器端控制与 GPU 端处理之间的数据暂存，进一步提升了流水线效率。

VIII. 相关工作

VkFFT [11], [20] 是唯一一款具备原生 Metal 支持、可用于生产环境的快速傅里叶变换库。它采用运行时代码生成技术，在六种 GPU 后端上生成 Stockham 自动排序内核。在 Apple Silicon 上，VkFFT 将线程数限制为 256，并指出线程组内存带宽是主要瓶颈。我们的研究证实了这一发现，并表明采用更高的线程数（512–1024）和结合架构特点进行基数选择，能够显著提升性能。

tcFFT [4] 证明，FFT 蝶形可以重新表述为在英伟达张量核心上执行的矩阵乘法，并在半精度浮点下实现了相较 cuFFT 1.29–3.24× 倍的加速。其基 16 方法映射到英伟达的 16×16张量核心 tile。我们将这一概念应用于 Apple 的 8×8 simdgroup_矩阵，发现较小的 tile 尺寸使基数 8 成为自然选择，但数据编排开销限制了单次快速傅里叶变换的性能。

TurboFFT [3] 为 NVIDIA A100/T4 提出了具备容错能力、适配架构的快速傅里叶变换优化方案。其寄存器级优化策略与我们的双层模型相似，不过具体的架构参数（共享内存大小、线程束宽度、张量核心维度）差异显著。

MetalFFT [21] 是 Philip Turner 尝试使用 Metal 原生实现快速傅里叶变换的项目，但因未能超越 vDSP/Accelerate 而被放弃。我们的工作通过基 8 分裂基蝶形以及依据架构设计的线程/内存配置，主要在 MetalFFT 未能取得成果之处实现了突破。

FFTW [22] 和 **cuFFT** [2]分别代表了面向中央处理器和 NVIDIA 图形处理器的快速傅里叶变换领域的先进水平，但两者都未针对 Apple Silicon GPU 进行优化。

ThunderMittens [17] 将 ThunderKittens Tensor Core 框架从英伟达 16×16 移植到 Apple GPU 的 8×8simdgroup_matrix，为 Apple Silicon 上的矩阵乘加（MMA）性能提供了首次系统性表征。我们的矩阵乘加内核分析基于他们的性能测量展开。

IX. 结论与未来工作

我们提出了一种面向 Apple Silicon GPU 的高性能快速傅里叶变换实现，在 $N=$4096 规模下达到 138.45 GFLOPS，较 Apple 的 vDSP/Accelerate 提升了 29%。其关键在于双层本地内存模型：该模型将 208 KiB 寄存器文件和 32 KiB 线程组内存映射到快速傅里叶变换分解中的不同用途，并将一篇 2015 年英特尔集成 GPU 博士论文提出的框架扩展到了 Apple Silicon。

我们的实测发现——在 Apple GPU 上，屏障开销低而分散访问代价高；尽管算术运算更多，基数 8 仍优于基数 4；此外，simdgroup_matrix 矩阵乘加尚未使单快速傅里叶变换内核受益——为快速傅里叶变换以外的 Apple Silicon GPU 内核优化提供了切实可行的指导。

A. 未来工作

合成孔径雷达流水线集成：在单个 Metal 渲染/计算通道中融合快速傅里叶变换、加窗、匹配滤波和角转置操作，以利用

32 KiB 线程组内存。目前尚无任何 GPU 平台发布过内核融合的合成孔径雷达流水线。

批次 单指令多数据组_矩阵 快速傅里叶变换：通过矩阵乘加基数 8 蝶形运算，在每个线程组中同时处理 8+ 个快速傅里叶变换。SAR 距离向处理天然提供了这一批次维度。估算显示，相较于标量单指令多数据，单精度浮点的加速比为 1.2×（半精度浮点为 2.4×），因此这种方案特别适合注重吞吐量的工作负载。

混合精度快速傅里叶变换：利用 Apple GPU 原生支持的半精度浮点（2×吞吐量，转换不产生额外开销），适用于信噪比要求允许降低精度的雷达应用。32 KiB 的线程组内存支持在半精度浮点下执行最大为 2^{13} 的局部快速傅里叶变换。

AMX 协处理器集成：利用中央处理器端的 AMX 矩阵单元，将批次小型快速傅里叶变换（$N \le 16$）作为矩阵乘法进行处理，同时由 GPU 处理较大的快速傅里叶变换；统一内存支持零拷贝共享，使这一方案成为可能。

更大型的 Apple Silicon 芯片：M4 Max（40 个 GPU 核心，带宽为 546 GB/s）在批次工作负载下的性能应大致随核心数量成比例提升，批次 $N = 4096$ 快速傅里叶变换的性能有望超过 500 GFLOPS。

可复现性

所有源代码、Metal 计算着色器、基准测试工具以及验证脚本均已在 https://github.com/aminems/AppleSiliconFFT 公开发布，并采用 MIT 许可证。该代码库包含构建说明，可在任何 Apple 芯片 Mac（M1 或更新机型）上复现所有结果。

致谢

作者感谢 Dougall Johnson（Apple 图形处理器指令集架构）、Alyssa Rosenzweig（Asahi Linux 图形处理器驱动程序）和 Philip Turner（Metal 基准测试）在开源逆向工程方面所做的工作，使本工作得以进行架构特性分析。

参考文献

[1] “机载合成孔径雷达处理的挑战：GPU 带来的机遇”，载于计算科学——*ICCS 2020*，丛书《计算机科学讲义》，第 12139 卷，2020 年，第 46–59 页。[2] 英伟达公司，*cuFFT 库用户指南*，2024 年。 [在线]。可访问：https://docs.nvidia.com/cuda/cufft/index.html [3] S. Wu、Y. Zhai、J. Liu、J. Huang、Z. Jian、H. Dai、S. Di、F. Cappello 和 Z. Chen，“TurboFFT：面向 GPU 的协同设计高性能容错快速傅里叶变换”，载于第 *30* 届美国计算机协会 *SIGPLAN* 并行编程原理与实践年会论文集（*PPoPP ’25*）。美国计算机协会，2025 年。

[4] S. Li 和 Y. Cheng，“tcFFT：面向英伟达张量核心的快速半精度快速傅里叶变换库”，载于 IEEE 国际并行与分布式处理研讨会专题会议（IPDPSW），2021 年。[5] M. A. Bergach，“混合 CPU/GPU 架构上快速傅里叶变换计算的适配”，博士论文，尼斯-索菲亚·安提波利斯大学，2015年，法国国家信息与自动化研究院。法语。

[ ] 01245958 在线。可访问：[6] https://theses.hal.science/tel- M. A. Bergach、E. Kofman、R. de Simone、S. Tissot 和 M. Syska，“面向雷达处理应用的 GPU 高效快速傅里叶变换映射：建模与实现”，arXiv 预印本 arXiv:1505.08067，2015年。

[ ] 1505 08067 在线。可访问：[7] https://arxiv.org/abs/。C. F. V. Loan，《快速傅里叶变换的计算框架》，丛书《应用数学前沿》。工业与应用数学学会，1992 年，第 10 号。[8] J. W. Cooley 和 J. W. Tukey，“一种用于计算复傅里叶级数的算法”，《计算数学》，第 19 卷，第 90 期，第 297–301 页，1965 年。[9] J. Thomas G. Stockham，“高速卷积与相关”，麻省理工学院林肯实验室，技术报告 AFCRL-66-55，1966 年。[10] N. K. Govindaraju、B. Lloyd、Y. Dotsenko、B. Smith 和 J. Manferdelli，“图形处理器上的高性能离散傅里叶变换”，载于 SC ’08：2008 年美国计算机协会/电气电子工程师学会超级计算会议论文集，2008 年。[11] D. Tolmachev，“VkFFT——高性能、跨平台的开源 GPU 快速傅里叶变换库”，《IEEE Access》，第 11 卷，第 12 039–12 058 页，2023 年。[12] 苹果公司，《Metal 着色语言规范》，第 4 版，2024 年。[在线]。可访问：https://developer.apple.com/metal/Metal-Shading-Language-Specification.pdf [13] ——，《Metal 功能集表》，2024 年。 [在线]。可访问：https://developer.apple.com/metal/Metal-Feature-Set-Tables.pdf [14] D. Johnson，“Apple G13 图形处理器架构参考”，在线文档。 [在线]。可访问：https://dougallj.github.io/applegpu/docs.html[15] P. Turner，“metal-benchmarks：Apple 图形处理器微架构”，GitHub 仓库。 [在线]。可访问：https://github.com/philipturner/metal-benchmarks[16] A. Rosenzweig，“剖析 Apple 图形处理器，第三部分”，博客文章，2022 年。[在线]。可访问：https://alyssarosenzweig.ca/blog/asahi-gpu-part-3.html[17] 斯坦福大学 Hazy Research，“为 ThunderKittens 打造的 ThunderMittens”，博客文章，2024 年。 [在线]。可访问：https://hazyresearch.stanford.edu/blog/2024-11-28-tk-mlx [18] 苹果公司，“vDSP.FFT——Accelerate 框架”，Apple 开发者文档。[在线]。可访问：https://developer.apple.com/documentation/accelerate/vdsp/fft [19] J. Zhou，“Apple AMX 矩阵加速器性能分析”，工程硕士论文，麻省理工学院，2025 年。[20] D. Tolmachev，“VkFFT：Vulkan/CUDA/HIP/OpenCL/Level Zero/Metal 快速傅里叶变换库”，GitHub 仓库，2020 年，MIT 许可证。[在线]。可访问：https://github.com/DTolm/VkFFT [21] P. Turner，“MetalFFT：快速傅里叶变换的一维、二维和三维变体”，GitHub 仓库，2022 年，已停止维护；性能无法超越 vDSP/Accelerate。 [在线]。可访问：https://github.com/philipturner/metal-fft [22] M. Frigo 和 S. G. Johnson，“FFTW3 的设计与实现”，《IEEE 论文集》，第 93 卷，第 2 期，第 216–231 页，2005 年。