PALMA：面向基于 ARM 的嵌入式系统的轻量级热带代数库

基于 SIMD 加速的半环线性代数：谱分析与嵌入式案例研究

Gnankan Landry Regis N'guessan[1,2,3]

[1]Axiom Research Group[2]纳尔逊·曼德拉非洲科技研究所（NM-AIST）应用数学与计算科学系，阿鲁沙，坦桑尼亚[3]非洲数学科学研究所（AIMS）研究与创新中心（RIC），基加利，卢旺达邮箱：*rnguessan@aimsric.org*

arXiv:2601.17028v1 [cs.MS] 17 Jan 2026

摘要

热带代数包括极大加法、极小加法及相关的幂等半环，它提供了一个统一框架，使得许多在经典代数中呈非线性的优化问题能够转化为线性问题。这一特性使热带方法特别适用于最短路径、调度、吞吐量分析和离散事件系统。尽管热带代数在理论上已相当成熟，且具有实际应用价值，现有实现主要面向桌面或服务器环境；而在资源受限的嵌入式平台上，这类优化问题尤为突出，但相关实现仍难以获得。我们提出 Palma（**并行代数库最大加法应用**），这是一款轻量、无依赖的 C 语言库，可将热带线性代数引入基于 ARM 的嵌入式系统。Palma 实现了通用的半环抽象，并配备 SIMD 加速内核，使单一计算框架能够支持最短路径、瓶颈路径、可达性、调度和吞吐量分析。该库支持五种热带半环、稠密和稀疏（压缩稀疏行）表示、热带闭包，以及通过最大循环均值计算进行的谱分析。我们在 Raspberry Pi 4 上评估了 Palma，展示了其峰值性能达到 2,274 百万次操作每秒；与经典贝尔曼-福特算法相比，单源最短路径的加速比最高可达 11.9 倍；针对实时控制工作负载的调度求解耗时不到 10 微秒。无人机控制、物联网路由和制造系统的案例研究表明，热带代数能够直接在嵌入式硬件上实现高效、可预测且统一的优化。Palma 以 MIT 许可证作为开源软件发布。

关键词： 热带代数、最大加法代数、幂等半环、ARM NEON、Raspberry Pi、嵌入式系统、实时系统、最短路径算法、图算法、稀疏矩阵、离散事件系统、开源软件

1 引言

1.1 动机与背景

热带数学已发展成为一种强大的框架，通过将优化问题转化为幂等半环上的代数运算来求解。在热带代数中，传统加法被取最大值（或最小值）所取代，而传统乘法则被加法所取代。这种看似简单的重新定义带来了深远影响：许多在经典数学中本质上是非线性的问题，例如

寻找最短路径、优化调度以及计算系统吞吐量，都变得在线性热带半环上，可表示为矩阵乘积和不动点迭代 [1, 2]。

考虑经典的最短路径问题：给定一个加权图，求两个顶点之间总权重最小的路径。在标准代数中，这需要迭代执行比较和选择运算，难以用矩阵形式表示。然而，在最小加热带半环中，最短路径计算可化简为矩阵乘法：

$$d_{ij}^{(k)} = \min_{\ell} \left(d_{i\ell}^{(k-1)} + w_{\ell j} \right) \tag{1}$$

其中，min 扮演加法的角色，而 $+$ 扮演乘法的角色。这种“线性化”使得能够将强大的线性代数技术直接应用于优化问题 [8, 16]。在 Palma 中，该递推关系可表示为一次 API 调用：`palma_matvec(A,d、 d_new、 PALMA_MINPLUS)`，而全源最短路径解只需 `palma_matrix_closure(A,PALMA_MINPLUS)`。

尽管自 Cuninghame-Green 的奠基性工作 [3] 以及 Baccelli 等人 [1]，的综合论著问世以来，热带代数已经历了四十年的理论发展，但其实际实现仍然少得出奇。现有工具，如 ScicosLab MaxPlus 工具箱 [6] 和 polymake 系统 [7, 10]，面向桌面计算环境，不适用于资源受限的嵌入式系统。这一空白尤其令人担忧，因为嵌入式系统（机器人技术、自动驾驶车辆、工业控制器、物联网网络）正是实时调度和最优路由最为关键的应用领域。

1.2 嵌入式系统面临的挑战

以树莓派系列及类似的基于 ARM 的系统为代表的现代嵌入式平台，面临着独特的限制：

1. **计算资源有限**：尽管 ARM Cortex-A72 等现代嵌入式处理器性能强劲，但其运行频率明显低于桌面 CPU，且缓存容量更小。2. **内存限制**：嵌入式系统的随机存取存储器（RAM）通常有限，因此内存高效的数据结构至关重要。3. **实时要求**：许多嵌入式应用要求确定性执行时间，截止期限通常在微秒到毫秒范围内。4. **能效**：电池供电且受热约束的系统要求运算尽可能降低能耗。5. **部署简便**：嵌入式软件应尽量减少外部依赖，并能轻松移植到不同平台。

这些限制促使人们开发 Palma——一款从设计之初就面向嵌入式系统的热带代数库。Palma 利用 ARM NEON SIMD 指令实现并行性，采用稀疏矩阵表示提升内存效率，并使用纯整数算术确保可预测性，将热带数学的强大能力带到能够产生最大实际影响的平台上。

1.3 主要贡献

本文作出以下贡献：

1. **数学基础**：我们全面、完整地介绍了热带代数，包括严谨的定义、附有证明的重要定理，以及基础算法。这既可作为教程，也可作为实现的参考。

2. **L库设计**：我们介绍 Palma，这是一个功能完备的热带代数库，提供：

   - 五种热带半环：极大加法 ($\mathbb{R}_{\max}$)、极小加法 ($\mathbb{R}_{\min}$)、极大极小、极小极大和布尔

   - 稠密和稀疏（压缩稀疏行格式）矩阵表示
   - ARM NEON 单指令多数据优化运算
   - 通过 Karp 算法计算热带特征值
   - 用于调度和图算法的高级应用程序接口

3. **实现**：我们使用约 2,000 行可移植的 C99 代码实现，并详细讨论了 NEON 优化策略和稀疏矩阵技术。

4. **实验评估**：我们在 Raspberry Pi 4 上进行了全面的基准测试，测量以下内容：

   - 不同矩阵尺寸下 NEON 与标量的性能
   - 从 16×16 到 1024×1024 矩阵的可扩展性分析
   - 全部五种半环的比较
   - 稠密与稀疏交叉分析
   - 与经典算法（Floyd-Warshall算法、贝尔曼-福特算法）的性能比较

5. **案例研究**：我们通过三个详细的案例研究展示实际应用：

   - 实时无人机控制系统调度
   - 物联网传感器网络路由优化
   - 制造生产线吞吐量分析

平台与术语背景。 </style}在本文中，基于 ARM 的嵌入式系统指以 ARM 架构为核心构建的高能效计算平台，常用于机器人技术、物联网和信息物理系统，例如部署在自主平台、工业控制器和边缘计算节点中的 ARM Cortex-A 系列片上系统设备。这些处理器侧重于可预测的执行、适度的并行性和受限的内存资源，而非高时钟频率或大容量缓存。单指令多数据 （Single Instruction, Multiple Data）是一种并行执行模型，其中一条指令可同时作用于多个数据元素。在 ARM 平台上，单指令多数据功能由*NEON*扩展提供，该扩展配备 128 位向量寄存器，可对四个 32 位整数执行并行运算。在 Palma 中，NEON 仅用作底层优化机制，以加速核心热带半环运算（如最大值、最小值和加法）；所有代数定义和算法语义均独立于底层硬件和数值表示。

超越 ARM 的可扩展性。 尽管 Palma 的当前实现面向基于 ARM 的嵌入式平台，但该库采用了与硬件无关的半环抽象进行设计。这一架构选择使其未来能够支持其他嵌入式架构，例如 RISC-V、基于 DSP 的系统，以及带有 SIMD 或向量扩展的微控制器级平台，而无需改变数学或算法核心。Palma 的长期目标是发展成为面向异构硬件环境中实际嵌入式优化问题的可移植计算基础。

1.4 论文结构

本文其余部分的组织如下。第 2 节介绍热带代数的数学基础，包括半环理论、热带线性代数和特征值计算。第 3 节描述 Palma 库的架构与设计决策。第 4 节详细介绍实现过程，重点说明 NEON 优化和稀疏矩阵技术。第 5 节介绍 Palma 中实现的核心算法。第 6 节给出在 Raspberry Pi 4 上进行的全面实验评估。第 7 节介绍三个详细的案例研究，展示实际应用。第 8 节讨论相关工作。最后，第 9 节总结全文并概述未来方向。

2 数学基础

本节将严谨地阐述热带代数，为 Palma 奠定理论基础。我们先介绍半环的一般理论，再将其具体化为热带半环，进而发展热带线性代数，最后讨论谱理论及其与图算法的重要联系。

符号约定。 本文使用 $(S, \oplus, \otimes, \mathbf{0}, \mathbf{1})$ 表示一般的幂等半环，其中 $\oplus$ 表示“加法”，$\otimes$ 表示“乘法”。随后，我们将讨论五种具体的半环：极大加法 $(\mathbb{R}_{\max})$、极小加法 $(\mathbb{R}_{\min})$、极大极小、极小极大和布尔。矩阵用粗体大写字母表示（$\mathbf{A}, \mathbf{B}$），向量用粗体小写字母表示（$\mathbf{x}$,$\mathbf{y}$），标量用斜体表示（a, λ）。上下文明确时，热带矩阵乘积与标量乘法使用相同的符号 $\otimes$。

2.1 半环理论

定义 2.1（半环）。 半环是一种代数结构 $(S, \oplus, \otimes, \mathbf{0}, \mathbf{1})$，其中 S是一个集合，$\oplus$ 和 $\otimes$ 是定义在 S上的二元运算，且 $\mathbf{0}$，$\mathbf{1} \in S$满足：

(i) $(S, \oplus, \mathbf{0})$ 是一个交换幺半群：

- 结合律：$(a \oplus b) \oplus c = a \oplus (b \oplus c)$
- 交换律：$a \oplus b = b \oplus a$
- 单位元：$a \oplus \mathbf{0} = a$

(ii) $(S, \otimes, \mathbf{1})$ 是一个幺半群：

- Associativity: $(a \otimes b) \otimes c = a \otimes (b \otimes c)$
- Identity: $a \otimes \mathbf{1} = \mathbf{1} \otimes a = a$

(iii) 乘法对加法满足分配律：

- Left: $a \otimes (b \oplus c) = (a \otimes b) \oplus (a \otimes c)$
- Right: $(a \oplus b) \otimes c = (a \otimes c) \oplus (b \otimes c)$

(iv) 零元具有湮灭性：

- $a \otimes \mathbf{0} = \mathbf{0} \otimes a = \mathbf{0}$

定义 2.2 (幂等半环)。 若半环满足 $a \oplus a = a$ 对所有 $a \in S$，则称其为幂等。

幂等半环是热带数学的核心，因为它们诱导出一个自然偏序。

命题 2.3（自然序）。 在幂等半环中，关系 $a \leq b \Longleftrightarrow a \oplus b = b$在 S上定义了一个偏序。

证明。 我们验证序公理：

- **自反性**：由幂等律可得 $a \oplus a = a$，因此 $a \leq a$。
- **反对称性**：若 $a \leq b$且 $b \leq a$，则 $a \oplus b = b$且 $b \oplus a = a$。由交换律可得，$a = b \oplus a = a \oplus b = b$。
- **传递性**：若 $a \leq b$且 $b \leq c$，则 $a \oplus b = b$且 $b \oplus c = c$。我们有 $a \oplus c = a \oplus (b \oplus c) = (a \oplus b) \oplus c = b \oplus c = c$，因此 $a \leq c$。 □

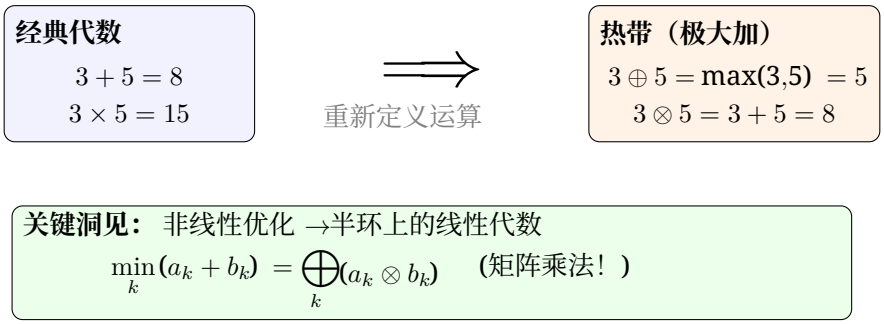


图1：经典代数与热带（最大加）代数。将加法重新定义为取最大值、将乘法重新定义为 $+$，即可将优化问题转化为线性代数运算。

2.2 热带半环

现在，我们定义 Palma 中实现的五种热带半环。

定义 2.4（极大加半环 $\mathbb{R}_{\max}$）。 该最大加半环 为 $\mathbb{R}_{\max} = (\mathbb{R} \cup \{-\infty\}, \max, +, -\infty, 0)$，其中：

- $a \oplus b = \max(a, b)$
- $a \otimes b = a + b$
- $\mathbf{0} = \varepsilon = -\infty$（吸收元）
- $\mathbf{1} = e = 0$（单位元）

定义 2.5（极小加法半环 $\mathbb{R}_{\min}$）。 最小加半环 为 $\mathbb{R}_{\min} = (\mathbb{R} \cup \{+\infty\}, \min, +, +\infty, 0)$，其中：

- $a \oplus b = \min(a, b)$
- $a \otimes b = a + b$
- $\mathbf{0} = +\infty$
- $\mathbf{1} = 0$

定义 2.6（极大极小半环）。 最大最小半环 （也称为瓶颈半环）是 $(\mathbb{R} \cup \{-\infty, +\infty\}, \max, \min, -\infty, +\infty)$，其中：

- $a \oplus b =$ 最大值(a, b)

- $a \otimes b =$ 最小值$(a,\ b)$
- $\mathbf{0} = -\infty$
- $\mathbf{1} = +\infty$

定义 2.7（极大极小半环）。 极小极大半环 是 $(\mathbb{R} \cup \{-\infty, +\infty\}, \min, \max\ +\infty, -\infty)$，其中：

- $a \oplus b = \min(a,\ b)$
- $a \otimes b = \max(a,\ b)$
- $\mathbf{0} = +\infty$
- $\mathbf{1} = -\infty$

定义 2.8（布尔半环）。 布尔半环 是 $(\{0,\ 1\}\ \vee,\ \wedge,\ 0,\ 1)$，其中：

- $a \oplus b = a \vee b$（逻辑或）
- $a \otimes b = a \wedge b$（逻辑与）
- $\mathbf{0} = 0$ (假)
- $\mathbf{1} = 1$ (真)

定理 2.9（热带半环性质）。 上面定义的五个半环都是有效的半环。此外，它们都是幂等的。

证明。 我们对 $\mathbb{R}_{\max}$证明该结论；其他情况类似。

加法幺半群：$(\mathbb{R} \cup \{-\infty\},\ \max\ -\infty)$ 显然是交换幺半群，因为 max 具有结合性和交换性，并且 $\max(a,\ -\infty)\ = a$。

乘法幺半群：$(\mathbb{R} \cup \{-\infty\}\ +,\ 0)$ 是一个幺半群，约定对所有 $a,\ a+(-\infty)\ = -\infty$。

分配律：对于所有 $a,\ b,\ c \in \mathbb{R}_{\max}$：

$$a \otimes (b \oplus c) = a + \max(b, c) = \max(a + b, a + c) = (a \otimes b) \oplus (a \otimes c)$$

等式 $a + \max(b,\ c)\ = \max(a + b,\ a + c)$ 成立，因为加法保持顺序。

湮灭律：$a \otimes (-\infty)\ = a + (-\infty)\ = -\infty = \mathbf{0}$。幂等律：$\max(a, a)\ = a$，因此 $a \oplus a = a$。 □

表 1：Palma 中实现的热带半环汇总

半环	$\oplus$	$\otimes$	$\mathbf{0}$	$\mathbf{1}$	主要应用
极大加法	max	$+$	$-\infty$	0	调度、最长路径
极小加法	min	$+$	$+\infty$	0	最短路径、Dijkstra
极大极小	max	min	$-\infty$	$+\infty$	瓶颈／带宽路径
极小极大	min	max	$+\infty$	$-\infty$	可靠性路径
布尔	$\vee$	$\wedge$	0	1	可达性、连通性

2.3 热带线性代数

在确立半环结构之后，我们现在将在任意幂等半环（S，$\oplus$，$\otimes$，$\mathbf{0}$，$\mathbf{1}$）上发展热带线性代数。

定义 2.10（热带矩阵运算）。 设 $\mathbf{A} \in S^{m\times n}$ 和 $\mathbf{B} \in S^{n\times p}$ 是S上的矩阵。热带矩阵乘积 $\mathbf{C} = \mathbf{A} \otimes \mathbf{B} \in S^{m\times p}$ 定义如下：

$$C_{ij} = \bigoplus_{k=1}^{n} (A_{ik} \otimes B_{kj}) = \bigoplus_{k=1}^{n} A_{ik} \otimes B_{kj} \tag{2}$$

对于最大加半环，有：

$$C_{ij} = \max_{k=1}^{n} (A_{ik} + B_{kj}) \tag{3}$$

对于最小加半环：

$$C_{ij} = \min_{k=1}^{n} (A_{ik} + B_{kj}) \tag{4}$$

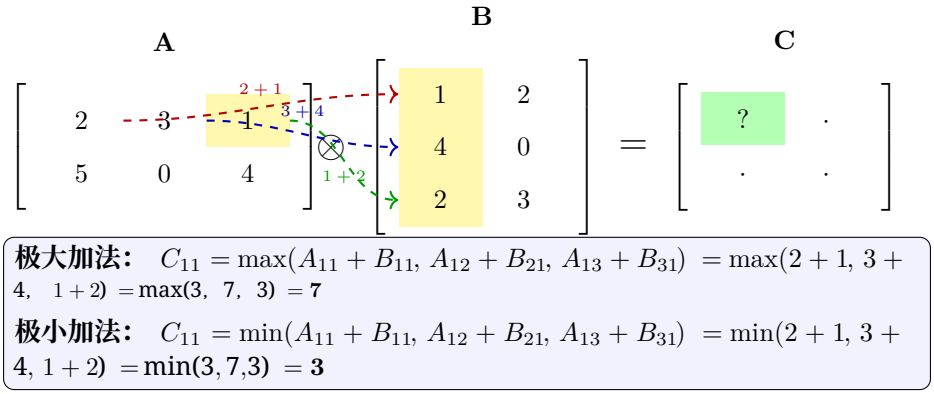


图 2：热带矩阵乘法示意图。计算 C_{ij}时，我们对从$\mathbf{A}$ 的第 i 行到$\mathbf{B}$的第 j 列的所有路径取热带加法（最大值或最小值），其中每条路径贡献其元素的热带乘积（$+$）。

定义 2.11（热带矩阵向量乘积）。 对于$\mathbf{A} \in S^{m\times n}$ 和$\mathbf{x} \in S^{n}$，乘积$\mathbf{y} = \mathbf{A} \otimes \mathbf{x} \in S^{m}$ 为：

$$y_i = \bigoplus_{j=1}^{n} A_{ij} \otimes x_j \tag{5}$$

定义 2.12（单位矩阵）。 该热带单位矩阵$\mathbf{E} \in S^{n\times n}$ 为：

$$E_{ij} = \begin{cases} \mathbf{1} & \text{if } i = j \\ \mathbf{0} & \text{if } i \neq j \end{cases} \tag{6}$$

命题 2.13（矩阵乘法性质）。 热带矩阵乘法满足：

(i) 结合律：$(\mathbf{A} \otimes \mathbf{B}) \otimes \mathbf{C} = \mathbf{A} \otimes (\mathbf{B} \otimes \mathbf{C})$ *(ii)* 单位元：$\mathbf{A} \otimes \mathbf{E} = \mathbf{E} \otimes \mathbf{A} = \mathbf{A}$

(iii) 分配律：$\mathbf{A} \otimes (\mathbf{B} \oplus \mathbf{C}) = (\mathbf{A} \otimes \mathbf{B}) \oplus (\mathbf{A} \otimes \mathbf{C})$

然而，一般而言：

(iv) 矩阵乘法不满足交换律：$\mathbf{A} \otimes \mathbf{B} \neq \mathbf{B} \otimes \mathbf{A}$

证明。 这些性质可直接由逐元素应用半环公理得出。对于结合律：

$$\begin{aligned}((\mathbf{A} \otimes \mathbf{B}) \otimes \mathbf{C})_{ij} &= \bigoplus_k \left(\bigoplus_\ell A_{i\ell} \otimes B_{\ell k} \right) \otimes C_{kj} \\ &= \bigoplus_k \bigoplus_\ell (A_{i\ell} \otimes B_{\ell k}) \otimes C_{kj} \\ &= \bigoplus_\ell \bigoplus_k A_{i\ell} \otimes (B_{\ell k} \otimes C_{kj}) \\ &= \bigoplus_\ell A_{i\ell} \otimes \left(\bigoplus_k B_{\ell k} \otimes C_{kj} \right) \\ &= (\mathbf{A} \otimes (\mathbf{B} \otimes \mathbf{C}))_{ij}\end{aligned}$$

这里用到了分配律以及 $\oplus$ 的交换律。 □

定义 2.14 （矩阵幂）。对于方阵 $\mathbf{A} \in S^{n \times n}$，定义如下：

$$\mathbf{A}^0 = \mathbf{E}, \quad \mathbf{A}^k = \mathbf{A}^{k-1} \otimes \mathbf{A} \text{ for } k \geq 1 \tag{7}$$

定理 2.15 （幂与路径）。设 $\mathbf{A}$ 是加权有向图G的邻接矩阵，其中 A_{ij} 表示从 i 到 j 的边的权重（若不存在边，则为$\mathbf{0}$ ）。那么：

(i) 在最小加半环中：$(\mathbf{A}^k)_{ij}$ 等于从 i 到 j 恰好经过 k 条边的最短路径的权重。

(ii) 在最大加半环中：$(\mathbf{A}^k)_{ij}$ 等于从 i 到 j 恰好经过 k 条边的最长路径的权重。

（*iii*）在最大最小半环中：（$\mathbf{A}^k$）$_{ij}$ 等于从 i 到 j 恰好经过 k 条边的路径中最大瓶颈值（路径上的最小边权）。

（*iv*）在布尔半环中：（$\mathbf{A}^k$）$_{ij} = 1$ 当且仅当存在一条从 i 到 j 且恰好经过 k 条边的路径。

证明。 我们通过归纳法证明（i）；其他情形类似。

基础情形 ($k = 1$)：$(\mathbf{A}^1)_{ij} = A_{ij}$ 是从 i 到 j 的直接边的权重，而该边是唯一的 1 条边路径。

归纳步骤：假设该结果对 $k-1$ 成立。则：

$$\begin{aligned}(\mathbf{A}^k)_{ij} &= \bigoplus_{\ell=1}^{n} (\mathbf{A}^{k-1})_{i\ell} \otimes A_{\ell j} \\ &= \min_{\ell=1}^{n} \left((\mathbf{A}^{k-1})_{i\ell} + A_{\ell j} \right)\end{aligned}$$

根据归纳假设，$(\mathbf{A}^{k-1})_{i\ell}$ 是从 i 到 ℓ的最短（$k-1$）边路径。添加边（ℓ，j）后，得到一条从 i 到 j 且经过 ℓ 的 k 边路径。对所有可能的中间顶点 ℓ 取最小值，即可得到最短的 k 边路径。

□

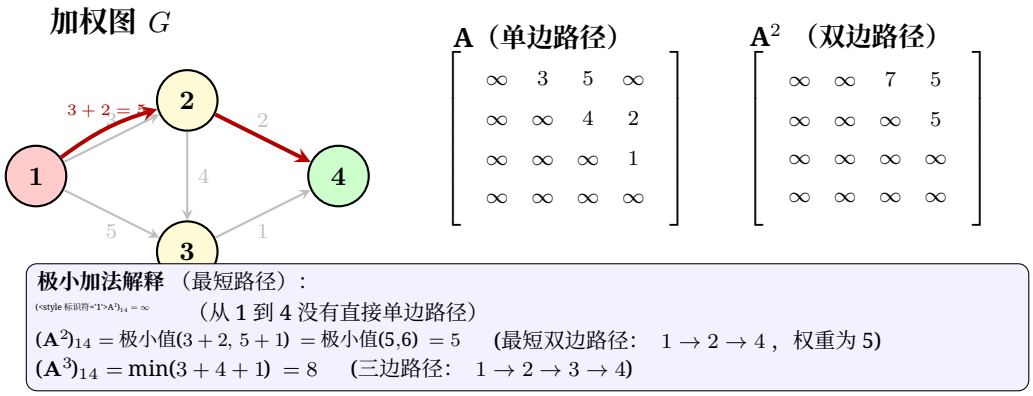


图 3：矩阵幂对应特定长度的路径。在最小加半环中，$(\mathbf{A}^k)_{ij}$ 给出从 i 到 j 恰好经过 k 条边的最短路径的权重。这里，$(\mathbf{A}^2)_{14} = 5$对应路径 $1 \to 2 \to 4$，其总权重为 $3+2=5$。

2.4 Kleene 星闭包（热带闭包）

定义 2.16 （Kleene 星闭包／热带闭包）。 对于方阵 $\mathbf{A} \in S^{n\times n}$，*Kleene* 星闭包或热带闭包定义为:

$$\mathbf{A}^* = \bigoplus_{k=0}^{\infty} \mathbf{A}^k = \mathbf{E} \oplus \mathbf{A} \oplus \mathbf{A}^2 \oplus \mathbf{A}^3 \oplus \cdots \tag{8}$$

定理 2.17 （Kleene 星闭包的收敛）。 对于具有非负权环的最小加半环，或者对于任意满足$\mathbf{A}$ 表示有向无环图的半环:

$$\mathbf{A}^* = \bigoplus_{k=0}^{n-1} \mathbf{A}^k \tag{9}$$

也就是说，无穷和在 $n-1$ 项后收敛。“边权非负”是“没有负权环”的充分（但非必要）条件。

证明。 在一个有 n 个顶点的图中，任意简单路径最多包含 $n-1$ 条边。对于没有负权环的最小加半环，经过一个圈不会降低路径权重，因此最优路径都是简单路径。因此，对于所有 $k \geq n-1$，$(\mathbf{A}^k)_{ij} = (\mathbf{A}^{n-1})_{ij}$，并且根据 min 的幂等律，额外的项不会改变总和。 □

定理 2.18 （Kleene 星闭包与全点对路径）。 对于最小加半环，$(\mathbf{A}^*)_{ij}$ 等于从 i 到 j 的最短路径权重（可使用任意数量的边）。这是全源最短路径（所有点对最短路径）问题的解。

推论 2.19 （Floyd-Warshall算法作为热带闭包）。 *Floyd-Warshall* 算法计算最小加半环中邻接矩阵的 *Kleene* 星闭包。

定理 2.20 （闭包复杂度）。 算法 *1* 在 $O(n^3)$ 时间内计算$\mathbf{A}^*$，空间复杂度为 $O(n^2)$。

2.5 热带特征值理论

热带特征值理论是分析周期系统以及计算制造业和调度应用中的吞吐量的基础 [1, 11]。

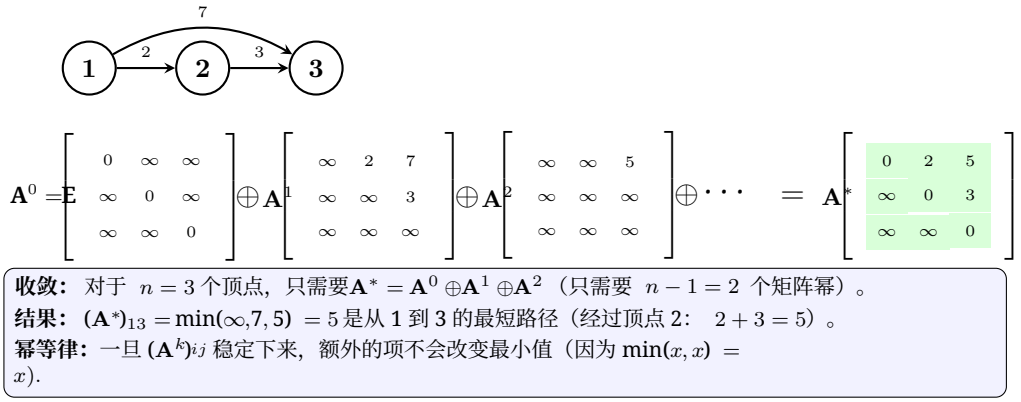


图 4：Kleene 星闭包 $\mathbf{A}^* = \bigoplus_{k=0}^{\infty} \mathbf{A}^k$在 $n-1$次迭代后收敛。每个幂 $\mathbf{A}^k$表示长度为k的路径；热带加法（取最小值）选出最短路径。元素 $(\mathbf{A}^*)_{ij}$ 给出从 i到 j的全点对最短路径。

算法 1 热带闭包（Floyd-Warshall算法风格）

输入： 邻接矩阵 $\mathbf{A} \in S^{n \times n}$
输出： 闭包矩阵 $\mathbf{A}^*$

D ←**A**
对于 $i = 1$ 至 n **执行**
    $D_{ii} \leftarrow D_{ii} \oplus \mathbf{1}$ ▷ 添加单位元
结束循环
对于 $k = 1$ 到 n **，执行**
    对于 $i = 1$ 到 n **，执行**
        对于 $j = 1$ 到 n **，执行**
            $D_{ij} \leftarrow D_{ij} \oplus (D_{ik} \otimes D_{kj})$
        结束循环
    结束循环
结束循环
返回 D

定义 2.21（热带特征值与特征向量）。 若存在一个非零向量 $\mathbf{v} \in S^n$（即特征向量），使得：标量 $\lambda \in S$ 是矩阵 $\mathbf{A} \in S^{n\times n}$ 的热带特征值。

$$\mathbf{A} \otimes \mathbf{v} = \lambda \otimes \mathbf{v} \tag{10}$$

在最大加半环中，这表示：

$$\max_{j=1}^{n}(A_{ij} + v_j) = \lambda + v_i \quad \text{for all } i \tag{11}$$

定义 2.22（带权有向图）。 矩阵 $\mathbf{A} \in \mathbb{R}_{\max}^{n\times n}$ 定义了一个加权有向图$G(\mathbf{A}) = (V, E, w)$，其中：

- $V = \{1, 2, \ldots, n\}$ 是顶点集
- $(i, j) \in E$ 当且仅当 $A_{ij} \neq -\infty$
- $w(i, j) = A_{ij}$ 是边权

也就是说，条目 A_{ij} 表示从顶点 i 到顶点 j的边的权重。

定义 2.23（圈均值）。 对于 $G(\mathbf{A})$ 中的一个圈 $\sigma = (i_1, i_2, \ldots, i_k, i_1)$，圈均值为：

$$\mu(\sigma) = \frac{1}{k}\sum_{j=1}^{k} A_{i_{j+1},i_j} = \frac{\text{weight}(\sigma)}{\text{length}(\sigma)} \tag{12}$$

其中，指标按模 k 计算。

定理 2.24（最大圈均值定理）。 对于一个矩阵 $\mathbf{A} \in \mathbb{R}_{\max}^{n\times n}$，若其关联图 $G(\mathbf{A})$是强连通的，则唯一的热带特征值为：

$$\lambda(\mathbf{A}) = \max_{\sigma \in \mathcal{C}} \mu(\sigma) \tag{13}$$

其中，$\mathcal{C}$ 是 $G(\mathbf{A})$ 中所有初等回路组成的集合。

证明概要。 完整证明见 [1, 2]。关键洞见在于，对于任意特征向量 $\mathbf{v}$：

$$\lambda = \max_{j}(A_{ij} + v_j) - v_i \tag{14}$$

沿任意圈求和，并利用 v_i 项的裂项相消性质，可知 λ 必定等于该圈的均值。所有圈中的最大值即为特征值。 □

定理 2.25（Karp 算法）。 最大回路平均值可在 $O(n^2 m)$时间内计算得出（对于稠密图则为 $O(n^3)$），使用公式：

$$\lambda(\mathbf{A}) = \max_{i \in V} \min_{0 \le k < n} \frac{(\mathbf{A}^n)_{ii} - (\mathbf{A}^k)_{ii}}{n - k} \tag{15}$$

其中，$(\mathbf{A}^k)_{ii}$ 表示从 i 到 i的长度为 k 的最长游走的权重。

证明。 参见 Karp [12]。该公式由以下观察得出：当 m足够大时，对于某个常数 C_i，$(\mathbf{A}^m)_{ii} \approx m \cdot \lambda + C_i$。作差可消去该常数，并揭示圈均值。

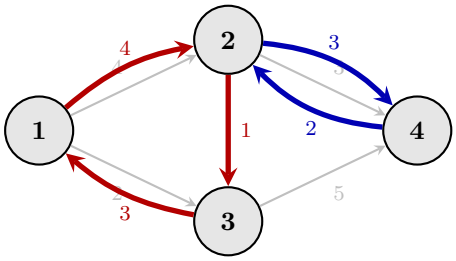
□

Th定理 2.26（渐近行为）。 对于最大回路平均值为 λ 的矩阵 $\mathbf{A}$：

$$\lim_{k\to\infty} \frac{(\mathbf{A}^k)_{ij}}{k} = \lambda \tag{16}$$

对所有 i, j 位于经过某个关键循环（达到最大平均值的圈）的路径上*).*

注 2.27（调度解释）。 在调度应用中，热带特征值具有重要的解释：它表示周期系统的最小可实现周期时间。如果任务按照由$\mathbf{A}$编码的优先约束重复执行，那么 λ 就是系统连续两次完成之间的最短时间，即吞吐量为 $1/\lambda$。



环路均值：

σ_1: $1 \to 2 \to 3 \to 1$
$\mu(\sigma_1) = \frac{4+1+3}{3} = \frac{8}{3} \approx 2.67$

σ_2: $2 \to 4 \to 2$
$\mu(\sigma_2) = \frac{3+2}{2} = \frac{5}{2} = 2.5$

σ_3: $3 \to 4 \to 3$ (若存在)
$\mu(\sigma_3) = \ldots$

热带特征值

$\lambda = \max_{\sigma} \mu(\sigma) = \frac{8}{3}$

（最大回路平均值）

物理意义：
在调度中，λ 是最小周期时间（吞吐量瓶颈）。

图5：热带特征值等于最大回路平均值。每个圈 σ 的均值为μ（σ） = 权重/长度。特征值 $\lambda = \max_\sigma \mu$（$\sigma$）决定系统的渐近行为，例如制造业中的最小周期时间或调度中的吞吐量上限。

算法 2 Karp 最大周期均值算法

要求： 矩阵 $\mathbf{A} \in \mathbb{R}_{\max}^{n \times n}$
确保： 最大回路平均值 λ

计算 $\mathbf{A}^0, \mathbf{A}^1, \ldots, \mathbf{A}^n$ 并存储对角线元素
$\lambda \leftarrow -\infty$
对于 $i = 1$ 到 n **执行**
    $\mu_i \leftarrow +\infty$ ▷初始化为 $+\infty$对于最小值
    对于 $k = 0$ 到 $n-1$ **执行**
        如果 $(\mathbf{A}^n)_{ii} \neq -\infty$ 且 $(\mathbf{A}^k)_{ii} \neq -\infty$ **则**
            $\mu_i \leftarrow$ 最小值$\left(\mu_i, \frac{(\mathbf{A}^n)_{ii} - (\mathbf{A}^k)_{ii}}{n-k}\right)$ ▷ 取最小值 k
        结束条件判断
    结束循环
    $\lambda \leftarrow$ 最大值(λ, μ_i) ▷ 取 i的最大值
结束循环
返回 λ

2.6 与经典图算法的联系

热带代数为经典图算法提供了一个统一框架。

定理 2.28 (算法统一)。 以下经典算法都是热带矩阵运算的特例：

(i) ***Floyd-Warshall***算法：在最小加半环中计算 *Kleene* 星闭包 $\mathbf{A}^*$，可得到全源最短路径。*(ii)* 贝尔曼-福特算法：在最小加半环中进行迭代 $\mathbf{d}^{(k+1)} = \mathbf{A} \otimes \mathbf{d}^{(k)} \oplus \mathbf{b}$，至多经过 $n-1$ 次迭代即可收敛到单源最短路径。*(iii)****Warshall*** 算法：在布尔半环中计算 $\mathbf{A}^*$，可得到传递闭包（可达性矩阵）。*(iv)* 最宽路径：在最大最小半环中计算 $\mathbf{A}^*$，可得到全源最大带宽路径。

证明。 这些结论可直接由定理 2.15 和 Kleene 星闭包的定义推出。对于 (ii)，注意：

$$\begin{aligned}\mathbf{d}^{(k)} &= \mathbf{A}^{k-1} \otimes \mathbf{b} \oplus \mathbf{A}^{k-2} \otimes \mathbf{b} \oplus \cdots \oplus \mathbf{A} \otimes \mathbf{b} \oplus \mathbf{b} \\ &= \left(\bigoplus_{j=0}^{k-1} \mathbf{A}^j \right) \otimes \mathbf{b}\end{aligned}$$

作为 $k \to n-1$，它会收敛到 $\mathbf{A}^* \otimes \mathbf{b}$，即从源点出发的最短路径。 □

3 库设计

本节介绍 Palma 的架构和设计原则。我们的主要目标是可移植性、在嵌入式平台上的高效性，以及让新手和专家用户都能轻松使用。

3.1 设计原则

Palma 遵循以下设计原则：

1. **零依赖**：该库使用纯 C99 实现，除标准库外没有任何外部依赖。这确保了它能够在所有配备 C 编译器的平台上移植。 2. **嵌入式优先设计**：所有设计决策均优先考虑嵌入式平台。我们采用定长整数算术运算，尽可能避免动态内存分配，并提供编译时配置选项。 3. **SIMD 透明性**：ARM 平台上会自动启用 NEON 优化，但对用户完全透明。同一应用程序接口在所有平台上的表现都完全一致。 4. **多种表示形式**：我们同时支持稠密和稀疏矩阵格式，用户可根据问题选择适当的表示形式。 5. **半环多态性**：所有运算都接受半环参数，因此可以在不同的热带代数之间复用代码。

3.2 数据类型

3.2.1 数值表示

Palma 的所有热带值均使用 32 位有符号整数：

```
typedef int32_t palma_val_t;
```

清单 1：数值类型定义

与浮点表示相比，这种选择具有以下几项优势：

- **确定性**：整数算术精确，且在不同平台上结果可复现。
- **单指令多数据效率**：ARM NEON 整数运算的延迟通常低于浮点运算。
- **比较简单**：整数比较比浮点比较简单，尤其是在接近无穷大时。
- **范围足够**：对于大多数实际应用，范围 $[-2^{31}, 2^{31} - 1]$ 已经足够。

3.2.2 半环枚举

```
typedef enum {
    PALMA_MAXPLUS = 0,  // (max, +) - scheduling
} palma_semiring_t;
```

清单 2：半环类型枚举

3.2.3 稠密矩阵结构

```
typedef struct {
    size_t rows;         // Number of rows
    size_t cols;         // Number of columns
    palma_val_t *data;   // Row-major storage
} palma_matrix_t;
```

图 3：稠密矩阵结构

稠密矩阵使用行优先存储，以提高逐行操作期间的缓存效率；逐行操作在矩阵-向量乘积中占主导地位。

3.2.4 稀疏矩阵结构（压缩稀疏行）

F对于边密度较低的图，我们提供压缩稀疏行（CSR）格式 ：

```
typedef struct {
    size_t rows;           // Number of rows
    size_t cols;           // Number of columns
    size_t nnz;            // Number of non-zeros
    palma_val_t *values;   // Non-zero values [nnz]
    palma_idx_t *col_idx;  // Column indices [nnz]
```

```
    palma_idx_t *row_ptr;  // Row pointers [rows+1]
    palma_semiring_t semiring;  // Associated semiring
} palma_sparse_t;
```

图表 4：CSR稀疏矩阵结构

CSR 提供 $O(1)$ 行访问和高效的稀疏矩阵-向量乘法，这对迭代图算法至关重要。

3.3 API概述

Palma 在三个抽象层级提供 API：

3.3.1 Level1：半环运算

每种半环的基本标量运算：

```
// Get zero and identity elements
```

列表 5：半环运算应用程序接口

3.3.2 Level2：矩阵运算

核心线性代数运算：

```
// Matrix creation and destruction
palma_matrix_t* palma_matrix_create(size_t rows, size_t cols);
void palma_matrix_destroy(palma_matrix_t *mat);

// Matrix arithmetic
palma_matrix_t* palma_matrix_add(const palma_matrix_t *A,
    const palma_matrix_t *B, palma_semiring_t s);
palma_matrix_t* palma_matrix_mul(const palma_matrix_t *A,
    const palma_matrix_t *B, palma_semiring_t s);
palma_matrix_t* palma_matrix_closure(const palma_matrix_t *A,
    palma_semiring_t s);
palma_matrix_t* palma_matrix_power(const palma_matrix_t *A,
    int k, palma_semiring_t s);

// Matrix-vector operations
void palma_matvec(const palma_matrix_t *A, const palma_val_t *x,
    palma_val_t *y, palma_semiring_t s);
```

清单 6：矩阵操作 API

3.3.3 Level3：应用程序API

常见应用的高级应用程序接口：

```
// Graph algorithms
void palma_single_source_paths(const palma_matrix_t *A,
    size_t source, palma_val_t *dist, palma_semiring_t s);
```

```
palma_matrix_t* palma_all_pairs_paths(const palma_matrix_t *A,

p
p
/
]

]

]
```
层

4 实现

本节详细介绍 Palma 的实现，重点说明 ARM NEON 优化和稀疏矩阵技术。

目标架构背景。 Palma 面向基于 ARM 的嵌入式平台，例如 Raspberry Pi 4。这类平台的特点是采用顺序执行或轻度乱序执行的核心、缓存容量适中，以及针对整数算术优化的 128 位 NEON SIMD 单元。这些架构特性直接影响了 Palma 的设计选择：仅使用整数的半环运算可避免浮点流水线停顿；缓存感知分块策略可提高矩阵内核中的数据复用率；而对矩阵-向量乘积进行 SIMD 向量化，则能充分利用 NEON 寄存器中可用的四宽整数通道。

4.1 半环运算实现

核心半环运算通过带有 switch 语句的内联函数实现，现代编译器能够对此进行有效优化：

```
static inline palma_val_t palma_add(palma_val_t a, palma_val_t b,
                                    palma_semiring_t s) {
    switch (s) {
        case PALMA_MAXPLUS:
        case PALMA_MAXMIN:
            return (a > b) ? a : b;  // max
        case PALMA_MINPLUS:
        case PALMA_MINMAX:
            return (a < b) ? a : b;  // min
        case PALMA_BOOLEAN:
            return a | b;            // OR
    }
    return a;
}
```

清单 8：半环加法实现

乘法需要谨慎处理无穷大。请注意，每种半环只使用一个无穷大值：极大加法使用 $-\infty$ 作为吸收元，而极小加法使用 $+\infty$。另一个无穷大值在该半环的定义域中无效。

```c
static inline palma_val_t palma_mul(palma_val_t a, palma_val_t b,
                                    palma_semiring_t s) {
    switch (s) {
        case PALMA_MAXPLUS:
            // Domain: R union {-inf}. Absorbing: -inf + x = -inf
            if (a == PALMA_NEG_INF || b == PALMA_NEG_INF)
                return PALMA_NEG_INF;
            return a + b;
        case PALMA_MINPLUS:
            // Domain: R union {+inf}. Absorbing: +inf + x = +inf
            if (a == PALMA_POS_INF || b == PALMA_POS_INF)
                return PALMA_POS_INF;
            return a + b;
        case PALMA_MAXMIN:
            return (a < b) ? a : b;  // min
        case PALMA_MINMAX:
            return (a > b) ? a : b;  // max
        case PALMA_BOOLEAN:
            return a & b;            // AND
    }
    return a;
}
```

列表 9：处理无穷大的半环乘法

4.2 ARM NEON SIMD 优化

ARM NEON 提供能够同时处理四个 32 位整数的 128 位向量寄存器。Palma 利用 NEON 加速矩阵运算的最内层循环。

4.2.1 NEON 架构概述

Raspberry Pi 4 中的 ARM Cortex-A72 处理器具有以下特性：

- 32 128 位 NEON 寄存器（Q0–Q31）
- 单指令多数据与标量运算并行执行
- 大多数整数 NEON 指令均可实现单圈吞吐量 Palma 中使用的关键 NEON 内建函数：

- `vld1q_s32`：将 4 个整数加载到向量寄存器中
- `vst1q_s32`：将向量寄存器中的 4 个整数存储起来
- `vmaxq_s32`：并行求 4 对整数的最大值
- `vminq_s32`：并行求 4 对整数的最小值
- `vaddq_s32`：并行相加 4 对整数

4.2.2 向量化矩阵-向量乘法

矩阵-向量乘积是迭代算法中执行频率最高的操作。我们的 NEON 实现可同时处理 4 列：

```
void palma_matvec_maxplus_neon(const palma_matrix_t *A,
(
```

4.2.3 向量化矩阵乘法

矩阵乘法受益于分块带来的缓存利用率提升，并结合 NEON 向量化：

```
// Inner kernel: compute C[i,j] = max_k(A[i,k] + B[k,j])
```

```
        max_vec = vmaxq_s32(max_vec, sum);
    }
    return vmaxvq_s32(max_vec);  // Horizontal max
}
```

清单 11：经过 NEON 优化的矩阵乘法内核

4.2.4 NEON 优化分析

NEON 向量化带来的理论加速比受以下因素限制：

$$\text{Speedup} \leq \frac{\text{Vector Width}}{\text{Scalar Width}} = \frac{128}{32} = 4\times \tag{17}$$

在实践中，有几个因素会降低这一加速比：

- **内存带宽**：大型矩阵会受内存带宽限制
- **水平归约**：向量通道上的最终最大值/最小值计算是标量操作
- **非对齐访问**：在某些微架构上，非对齐加载会带来性能损失
- **循环开销**：对于无法被 4 整除的维度，需要进行设置和清理

我们的实验（第 6 节）表明，Palma 使用 NEON 可实现最高 1.80× 的加速比；峰值出现在矩阵大小适合 L2 缓存时（64×64至 128×128）。

4.3 稀疏矩阵实现

4.3.1 CSR 格式

压缩稀疏行（CSR）格式仅存储非零元素：

$$\mathbf{A} = \begin{bmatrix} 5 & & 3 & \\ & 2 & & 7 \\ 1 & & & \\ & & 4 & 6 \end{bmatrix}$$

```
values = [5, 3, 2, 7, 1, 4, 6]
col_idx = [0, 2, 1, 3, 0, 2, 3]
row_ptr = [0, 2, 4, 5, 7]
```

图6：稀疏矩阵的 CSR表示。空单元格表示半环零元（$-\infty$极大加法）。

$n \times n$ 矩阵的内存使用量比较，其中包含 nnz 个非零元素：

$$\text{稠密：} \quad n^2 \times 4 \text{ 字节} \tag{18}$$

$$\text{CSR：} \quad \text{nnz} \times 4 + \text{nnz} \times 4 + (n+1) \times 4 = 8\text{nnz} + 4n + 4 \text{ bytes} \tag{19}$$

在以下情况下，CSR更加内存高效：

$$8\text{nnz} + 4n + 4 < 4n^2 \implies \text{nnz} < \frac{n^2 - n - 1}{2} \approx \frac{n^2}{2} \tag{20}$$

因此，对于稀疏度（零元素所占比例）大于约 50% 的矩阵，压缩稀疏行格式可以节省内存。

4.3.2 稀疏矩阵-向量乘法

```
void palma_sparse_matvec(const palma_sparse_t *A,
```

代码清单 12：压缩稀疏行（CSR）稀疏矩阵-向量乘法

复杂度：稀疏情况下为 $O(\text{nnz})$，而稠密情况下为 $O(n^2)$，因此对于稀疏图可获得显著的加速比。

4.3.3 稀疏矩阵乘法

稀疏矩阵乘法更为复杂，因为必须动态计算结果稀疏模式：

```
palma_sparse_t* palma_sparse_mul(const palma_sparse_t *A,
            size_t k = A->col_idx[ja];
{
                palma_val_t prod = palma_mul(a_ik, B->values[jb],
```

代码清单 13：稀疏矩阵乘法概述

4.4 内存管理

Palma 提供两种内存管理模式：

1. **动态分配**（默认）：使用 `malloc/free`以提高灵活性。2. **静态分配**：对于硬实时系统，用户可以提供预分配缓冲区：

```
// User-provided buffer palma_val_t buffer[1024];


```

14：静态内存分配模式

4.5 构建系统与可移植性

Palma 使用简单的 Makefile，支持多种构建配置：

```
make all          # Default: auto-detect NEON
make scalar       # Force scalar (no NEON)
make openmp       # Enable OpenMP parallelization
make debug        # Debug build with sanitizers
```

清单 15：构建配置

编译时特性检测：

```
#if defined(__ARM_NEON) || defined(__ARM_NEON__)
    #define PALMA_USE_NEON 1
    #include <arm_neon.h>
#else
    #define PALMA_USE_NEON 0
#endif
```

清单 16：平台检测

5 算法

本节介绍 Palma 中实现的核心算法，并对其进行复杂度分析和说明实现细节。

5.1 热带闭包（全点对路径）

热带闭包 $\mathbf{A}^*$ 用于计算所有顶点对之间的最优路径。我们的实现采用 Floyd-Warshall 风格算法 1，并进行了以下优化：

1. **原地计算**：算法直接在矩阵中进行修改，仅需 $O(n^2$)
   用于恒等元加法的辅助空间。2. **提前终止**：对于有向无环图，我们会检测收敛并提前终止。3. **NEON 向量化**：内层循环同时处理 4 个元素。

定理 5.1 (闭包正确性)。 算法 *1* 能正确计算任意幂等半环的 *Kleene* 星闭包。

证明。 迭代 k 后，条目 D_{ij} 包含从 i 到 j 的最优路径，该路径仅使用来自 $\{1, \ldots, k\}$ 的中间顶点。这可通过归纳法得出：更新 $D_{ij} \leftarrow D_{ij} \oplus (D_{ik} \otimes D_{kj})$ 会考虑经过顶点 k 的路径。经过 n 次迭代后，所有中间顶点都已被考虑。 □

5.2 单源路径

对于单源最优路径，我们实现了贝尔曼-福特算法的热带变体 :

算法 3 热带单源路径

要求： 邻接矩阵 $\mathbf{A}$，源顶点 s，半环 S
确保： 距离向量 $\mathbf{d}$，其中 $d_i =$ 到 i的最优路径权重（半环 s ）

$\mathbf{d} \leftarrow$ （$\mathbf{0}$,$\mathbf{0}$, . . . ,$\mathbf{0}$）▷初始化为半环零元
$d_s \leftarrow \mathbf{1}$ ▷到源点的距离是单位元
对于 $k = 1$ 至 $n - 1$ **执行**
  $\mathbf{d}_{\text{old}} \leftarrow \mathbf{d}$ ▷ 存储先前状态
  $\mathbf{d}' \leftarrow \mathbf{A} \otimes \mathbf{d}$ ▷ 热带矩阵向量乘积
  $\mathbf{d} \leftarrow \mathbf{d} \oplus \mathbf{d}'$ ▷ 松弛
  如果 $\mathbf{d} = \mathbf{d}_{\text{old}}$ **，则跳出循环。** ▷已达到不动点
  end if
结束循环
返回 $\mathbf{d}$

定理 5.2 (单源最短路径复杂度)。 算法 *3* 在稠密图中的最坏情况时间复杂度为 $O(n^3)$，在采用 *CSR* 格式的稀疏图中则为 $O(n \cdot \text{nnz})$。每次迭代都要执行一次矩阵-向量乘积，开销为 $O(n^2)$ （稠密图）或 $O(\text{nnz})$ （稀疏图）。提前终止在实践中可能减少迭代次数。

5.3 特征值计算

我们实现了 Karp 算法（算法 2），并进行了以下改进：

1. **内存高效的矩阵幂计算**：我们不存储所有 $n + 1$ 矩阵幂，而只存储对角线元素，从而将空间占用从 $O(n^3)$ 降至 $O(n^2)$。2. **增量计算**：矩阵幂按增量方式计算：$\mathbf{A}^{k+1} = \mathbf{A}^k \otimes \mathbf{A}$。

```
1  double palma_eigenvalue(const palma_matrix_t *A,
2                          palma_scratch_t *s) {
3
6
8
9
10
13
14
15
17
18
19
21
```

```
    // Apply Karp's formula: lambda = max_i min_k [(A^n)_ii - (A^k)_ii
]/
k),                    if (ratio < min_ratio) min_ratio = ratio;
```

列表 17：Karp 算法实现

5.4 特征向量计算（幂迭代）

我们使用带归一化的幂迭代计算热带特征向量：

算法 4 热带幂迭代

要求： 矩阵 $\mathbf{A}$，特征值 λ，容差 ϵ，最大迭代次数 K
确保： 满足条件的特征向量 $\mathbf{v}$ $\mathbf{A} \otimes \mathbf{v} = \lambda \otimes \mathbf{v}$

$\mathbf{v} \leftarrow (0, 0, \ldots, 0)$ ▷初始猜测
对于 $k = 1$ 到 K **执行**
  $\mathbf{u} \leftarrow \mathbf{A} \otimes \mathbf{v}$
  $\mathbf{v}' \leftarrow \mathbf{u} \ominus \lambda$ ▷ 归一化：减去特征值
  如果 $\|\mathbf{v}' - \mathbf{v}\|_\infty < \epsilon$ **则跳出循环**
  结束如果
  $\mathbf{v} \leftarrow \mathbf{v}'$
结束循环
返回 $\mathbf{v}$

5.5 调度求解器

调度器使用热带代数求解优先约束时序问题：

备注 5.3（调度器假设）。 算法 5 假设任务图是有向无环图（DAG）。对于有向无环图，该算法最多经过 $|V| - 1$ 次迭代即可收敛。对于循环任务系统（例如周期调度），必须使用 Karp 算法单独计算$\mathbf{A}$ 的特征值 λ；该值表示可达到的最小周期时间。随后，要得到一致的周期调度，就需要根据稳态周期 λ调整开始时间。

6 实验评估

我们呈 在树莓派 4 Model 上对 Palma 进行评估的全面实验结果 B.

算法 5 优先约束调度 要求：任务图 $G = (V, E, w)$，就绪时间 r 确保：开始时间 s，完成时间 c 1: 根据 G 构造邻接矩阵 A 2: s $\leftarrow$r ▷ 用就绪时间初始化 3: 对于 $k = 1$ 到 $|V| - 1$，执行 4: $s_{\text{old}} \leftarrow$s ▷ 保存先前状态 5: s′ $\leftarrow$ A $\otimes$s ▷ 极大加乘积 6: s $\leftarrow$ s $\oplus$s′ ▷取最大值 7: 如果 s $=s_{\text{old}}$，则跳出 ▷ 已达到不动点 8: 结束 if 9: 结束循环 10: c $\leftarrow$ s + d ▷ 加上持续时间 11: 返回 s, c

6.1 实验设置

6.1.1 硬件平台

- **处理器**：Broadcom BCM2711，四核 ARM Cortex-A72 @ 1.5 GHz
- **单指令多数据**：ARM NEON（128 位向量）
- **内存**：8 GB LPDDR4-3200同步动态随机存取存储器
- **存储**：32 GB SD卡
- **一级缓存**：每个核心配备 32 KB 指令缓存和 + 48 KB 数据缓存
- **L2 缓存**：1 MB 共享
- **操作系统**：树莓派操作系统（64 位）

6.1.2 方法

- 所有实验均重复 3–5 次；报告平均值
- 为确保一致性，CPU频率锁定为 1.5 GHz
- 计时方式： `clock_gettime(CLOCK_MONOTONIC)`
- 矩阵使用范围为 $[-1000, 1000]$ 的伪随机整数值进行初始化
- 稀疏度：常规测试为 30%；稀疏实验为 0–95% 不等
- 编译器：GCC 10.2，使用 `-O3 -march=native`
- 热管理：被动冷却，调速器设置为“性能”

6.1.3 正确性验证

所有实现均已通过与参考标量实现的对比验证：

- NEON 内核已针对随机矩阵与标量代码进行验证，矩阵规模最大为 512×512
- 将热带最短路径问题与标准贝尔曼-福特算法进行对比，测试对象为小型图（$n \leq 64$）
- 已通过迭代矩阵幂计算验证闭包结果

- 特征值针对小型图，将特征值计算与显式环枚举进行比较

百万次操作每秒（MOPS）定义为：对于耗时t 微秒的 $n \times n$ 矩阵乘法， $2n^3/t$ 将一次半环乘加计为一次运算。

6.2 实验 1：NEON 与标量性能对比

我们比较了不同矩阵尺寸下经过 NEON 优化的实现与标量实现。

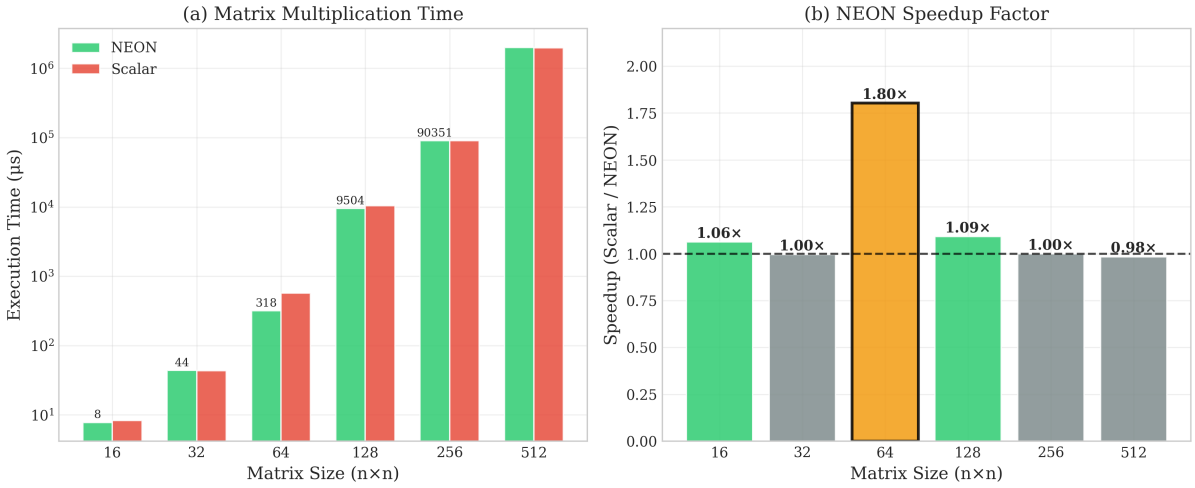


图7：NEON 优化结果。(a) 矩阵乘法的执行时间对比。(b) 加速比（标量执行时间 / NEON 执行时间）。在 64×64时达到峰值加速比 1.80× 。

表2：矩阵乘法的 NEON 与标量性能对比

Size	NEON（μ秒）	标量（μ秒）	加速比
16×16	7.8	8.3	1.06×
32×32	43.8	43.6	1.00×
64×64	317.8	573.1	**1.80×**
128×128	9,504	10,376	1.09×
256×256	90,351	90,291	1.00×
512×512	2,005,422	1,973,033	0.98×

分析：峰值加速比 1.80× 出现在 64×64 个矩阵时，此时数据可完全装入 L2 缓存。大小较小时，循环开销占主导；大小较大时，内存带宽成为瓶颈，缓存未命中抵消了 NEON 的优势。

6.3 实验 2：可扩展性分析

我们分析执行时间如何随矩阵大小变化。

分析：矩阵乘法遵循预期的 $O(n^3)$复杂度。MOPS（百万次操作每秒）峰值为 64×64，随后因缓存压力而下降。对于大于 512×512的矩阵，闭包计算的开销变得难以承受。

6.4 实验 3：半环比较

我们比较了全部五种半环在 64×64矩阵乘法方面的性能 .

分析：极大极小速度最快（2,274 百万次操作每秒），因为它完全避免了加法运算，只使用比较运算。布尔速度最慢，因为其位运算效率较低，在

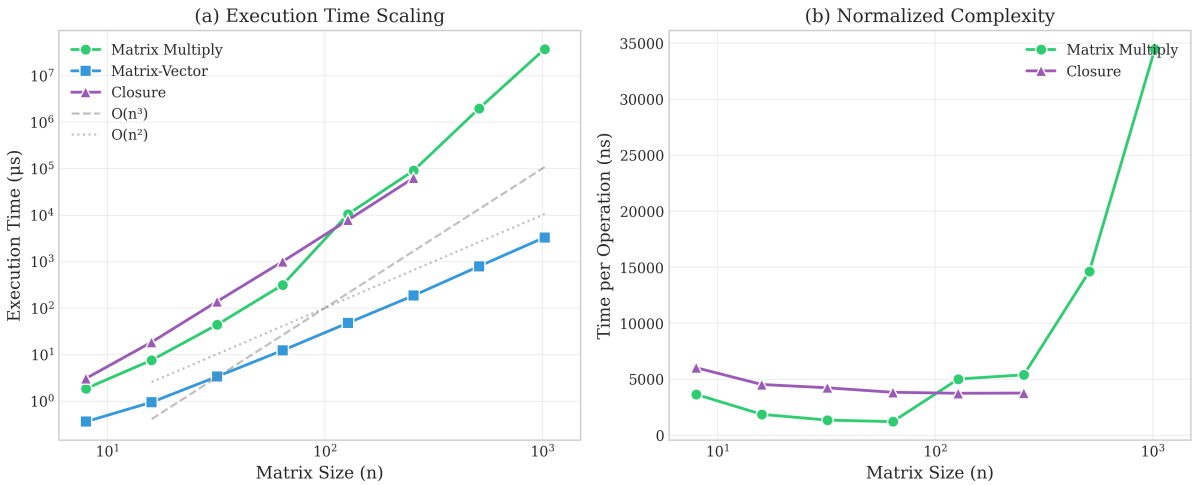


图 8：可扩展性分析。(a) 双对数尺度下执行时间与矩阵大小的关系。参考线显示理论上的 $O(n^3)$ 和 $O(n^2)$ 复杂度。(b) 每次操作时间（时间/n^3），显示缓存效应。

Tab表 3：极大加法半环的可扩展性结果：平均执行时间（微秒） s

Size	矩阵乘法	矩阵向量乘法	闭包	MOPS
8×8	1.9	0.4	3.1	539
16×16	7.6	0.9	18.5	1,076
32×32	43.8	3.4	138.4	1,495
64×64	314.3	12.3	1,001	1,668
128×128	10,498	47.9	7,827	399
256×256	90,296	185.6	62,909	372
512×512	1,961,718	791.2	n/a	137
1024×1024	37,018,442	3,302.5	n/a	58

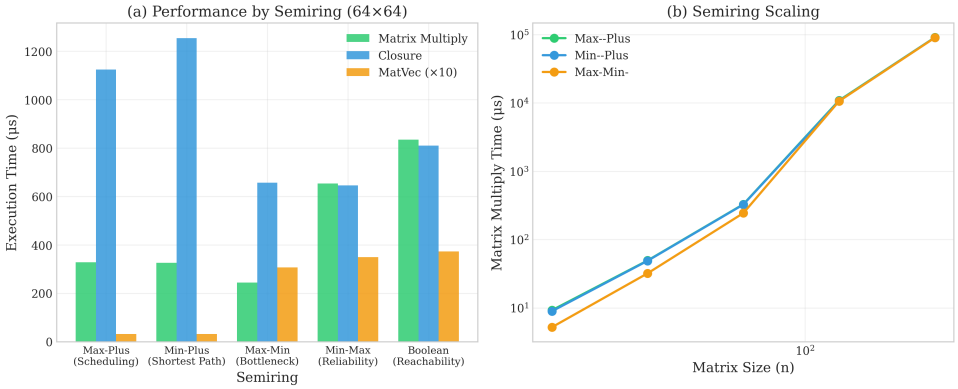


图9：半环性能比较。（a）不同运算下 64×64 矩阵的执行时间。（b）不同半环的扩展行为。

表 4：半环比较： 64×64矩阵乘法

半环	时间（μs）	MOPS	相对值	应用
极大加法	351.4	1,492	1.00×	调度
极小加法	342.9	1,530	0.98×	最短路径
极大极小	230.6	**2,274**	0.66×	瓶颈路径
极小极大	653.0	803	1.86×	可靠性
布尔	833.9	629	2.37×	可达性

ARM 架构。极大加法和极小加法的性能几乎相同，这符合其对称定义的预期。

6.5 实验 4：稠密与稀疏性能

我们分析稀疏表示开始具有优势的交叉点 s.

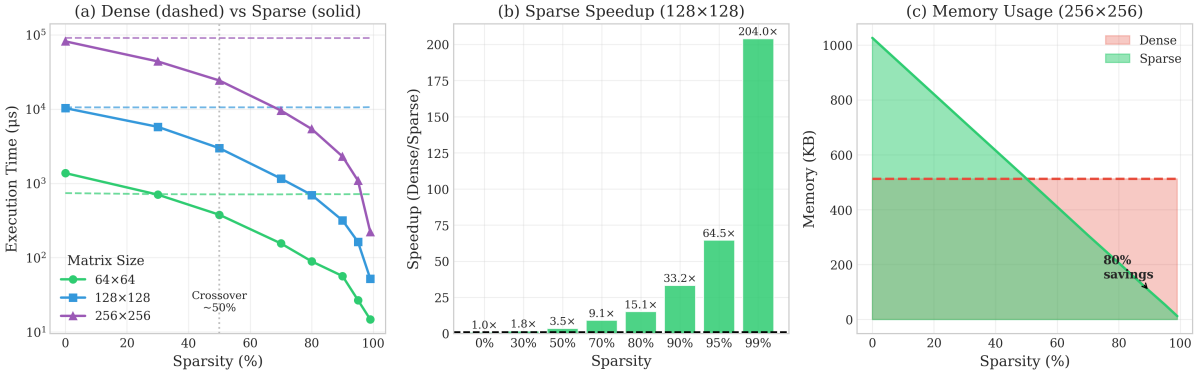


图 10：稠密与稀疏比较。(a) 不同矩阵大小下，执行时间与稀疏度的关系。(b) 稀疏相对于稠密的加速比。(c) 内存使用量比较。

表 5： 128×128矩阵的稠密与稀疏交叉分析

稀疏度	稠密（μs）	稀疏（μs）	加速比	节省的内存
0%	10,555	21,547	0.49×	−116%
30%	10,555	10,893	0.97×	12%
50%	10,555	2,980	**3.54**×	47%
70%	10,555	1,218	8.67×	68%
90%	10,555	312	33.8×	89%
95%	10,555	156	67.7×	94%

关键发现：稀疏度约为**50%**时，稀疏表示的速度开始超过稠密表示。真实世界图（通常有 70–95% 的稀疏度）可从 CSR 格式中显著受益。

6.6 实验 5：算法比较

我们将 Palma 的热带实现与经典算法进行比较。

关键发现：对于单源最短路径，Palma 的热带迭代性能优于经典的贝尔曼-福特算法，最高可达**11**。**9**× 这是因为热带矩阵向量乘积比逐边松弛具有更好的缓存局部性。

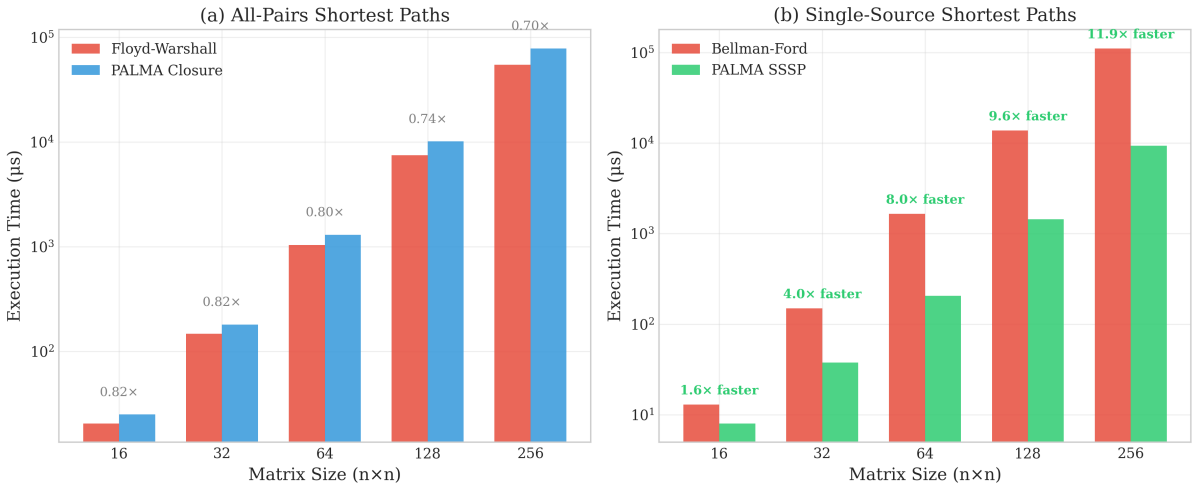


图11：算法比较。(a) 全源最短路径：Floyd-Warshall算法与PALMA热带闭包。(b) 单源最短路径：贝尔曼-福特算法与PALMA热带迭代。

表6：单源最短路径：贝尔曼-福特算法与PALMA

Size	贝尔曼-福特算法（μ秒）	PALMA（μ秒）	加速比
16×16	13	8	1.63×
32×32	149	38	3.95×
64×64	1,650	206	8.03×
128×128	13,816	1,444	9.57×
256×256	111,093	9,353	**11.88**×

对于全源最短路径，热带闭包和 Floyd-Warshall算法的复杂度相近，其中 Floyd-Warshall算法稍快（0.70–0.82×），因为其内层循环更简单。

6.7 实验6：实时可行性

我们分析哪些矩阵大小能够满足实时期限。

表7：满足实时约束的最大矩阵大小

期限	频率	最大矩阵规模
100 μ秒	10 千赫	16×16
1 ms	1 千赫	64×64
10 毫秒	100 赫兹	128×128
100 毫秒	10 赫兹	256×256

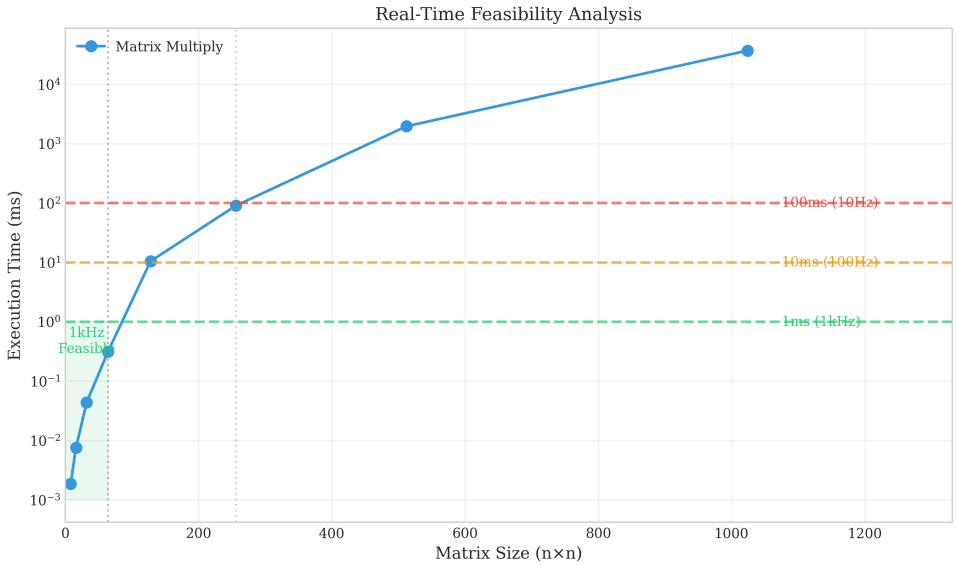


图12：实时可行性分析。横线表示常见的控制环路截止时间。对于 1 kHz 控制回路，矩阵大小不超过 64×64 时具有可行性。

6.8 关键结果总结

实验亮点

- **峰值性能**：在 64×64 矩阵（最大最小半环）上达到 2,274 百万次操作每秒
- **NEON 加速比**：相较于标量代码，最高可达 1.80×
- **单源最短路径加速比**：比贝尔曼-福特算法快至 11.9× 倍
- **稀疏效率**：稀疏度为 50% 时加速比达 3.5×
- 内存节省</style}：通过稀疏表示，最高可达 94%
- **实时**：对于最多包含 64 个节点的图，可实现 1 kHz

7 案例研究

我们通过三个详细的案例研究展示 Palma 的实际应用价值，这些案例分别代表不同的嵌入式系统定义域。

7.1 案例研究 1：实时无人机控制系统

7.1.1 问题描述

四旋翼无人机需要高频控制回路（通常为 250–1000 Hz）来保持稳定。控制流程涉及多个相互依赖的任务：

1. **传感器采集**：惯性测量单元、气压计、全球定位系统读数 2. **状态估计**：卡尔曼滤波、姿态/位置估计

3. **控制计算**：用于姿态、高度和位置的 PID控制器

4. **执行器驱动**：电机混控和 PWM 输出

5. **通信**：遥测传输

挑战在于确定给定的任务集是否能够达到目标控制频率，并找出限制吞吐量的关键路径。

7.1.2 热带代数建模

我们将无人机控制系统建模为最大加线性系统。设有 $n = 12$ 个任务，其优先约束构成一个有向无环图（DAG）。状态方程为：

$$\mathbf{x}(k+1) = \mathbf{A} \otimes \mathbf{x}(k) \oplus \mathbf{r} \tag{21}$$

其中，$\mathbf{x}(k)$包含第 k次迭代时的任务完成时间，$\mathbf{A}$ 编码优先约束，边权表示任务持续时间，而$\mathbf{r}$包含就绪时间。

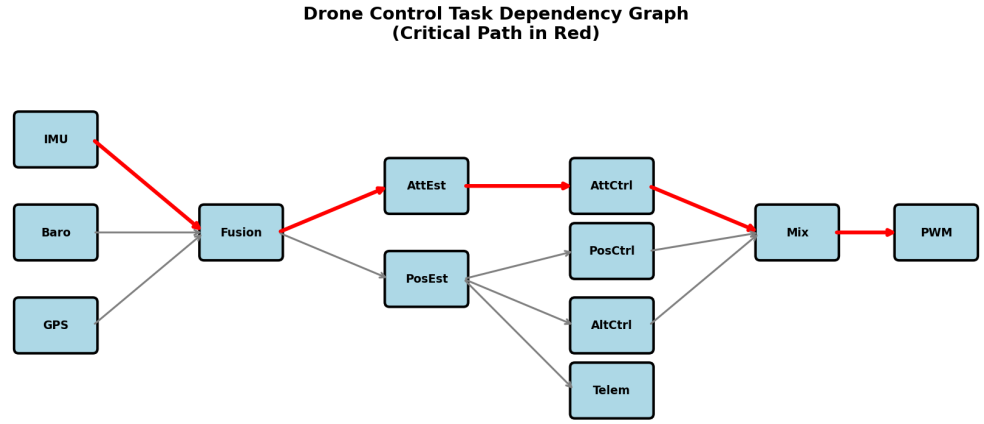


图13：由Palma生成的任务依赖关系图示例：系统启动序列包含HW_INIT → KERNEL → 并行阶段（DRIVERS、NETWORK、FILESYSTEM） →SERVICES。边权表示以毫秒为单位的执行时间。这展示了Palma对优先约束任务系统进行建模的能力。

7.1.3 任务规范

7.1.4 结果

使用 Palma 的调度 API：

```
palma_scheduler_t *sched = palma_scheduler_create(12, true);
```

图 18：使用 PALMA 进行无人机调度分析

7.1.5 分析

关键路径为：惯性测量单元 → 融合 → 姿态估计 → 姿态控制 → 混控 → 脉宽调制，总计440 μ秒。系统有 130 μ秒松弛时间，因此能够轻松实现 1 kHz 控制回路。Palma

表 8：无人机控制任务参数

ID	Task	持续时间（μ 秒）	依赖项
T1	惯性测量单元读取	50	（无）
T2	气压计读取	30	(none)
T3	GPS读取	100	(none)
T4	传感器融合	200	T1, T2, T3
T5	姿态估计	80	T4
T6	位置估计	120	T4
T7	高度控制	40	T6
T8	姿态控制	60	T5
T9	位置控制	50	T6
T10	电机混控	30	T7, T8, T9
T11	PWM 输出	20	T10
T12	遥测	150	T6

表 9：无人机调度分析结果

指标	数值
调度求解时间	6.28 μ秒
求解器迭代次数	6
循环总时间（总工期）	570 μ秒
关键路径长度	440 μ秒
松弛时间	130 μ秒
最大可实现频率	**1,754 Hz**
1 kHz 可行性	**Yes**
内存占用	672 字节

求解器仅用 6.28 μs 就完成了计算，开销可以忽略不计；如果任务计时发生变化，每个控制周期都可以执行一次。

对于周期性（循环）运行，周期时间为 78 μs，吞吐量为 12.82 次迭代/ms，表明系统完全满足实时约束。

7.2 案例研究 2：物联网传感器网络路由

7.2.1 问题描述

物联网应用中的无线传感器网络（WSN）需要高效的路由协议，以优化各项指标：

- **延迟**：尽量缩短时间关键型数据的端到端延迟
- **带宽**：最大化数据聚合吞吐量
- **能量**：平衡负载以延长网络寿命

我们分析了一个由 50 个节点组成的传感器网络，该网络部署在面积为 100m $\times$ 100m 的区域内，通信范围为 30m。

7.2.2 热带代数公式

每种路由指标都对应一种不同的热带半环：

- **最小延迟**：最小加半环，其中 A_{ij} 是从 i到 j的传输延迟
- **最大带宽**：最大最小半环，其中 A_{ij} 表示链路容量
- **连通性**：用于可达性分析的布尔半环

7.2.3 结果

表 10：传感器网络路由结果（50 个节点）

指标	数值	Unit
计算时间		
最短路径（最小加闭包）	678.4	μs
瓶颈路径（最大最小闭包）	395.1	μs
可达性（布尔闭包）	420.8	μs
网络指标		
到网关的平均延迟	652	μs
最低保证带宽	345	kbps
可达节点	49/49	节点
连通率	100%	（不适用）
内存效率（稀疏）		
稠密内存	10,000	字节
稀疏内存	4,540	字节
内存节省	**54.6%**	（不适用）
矩阵稀疏度	78.3%	（不适用）

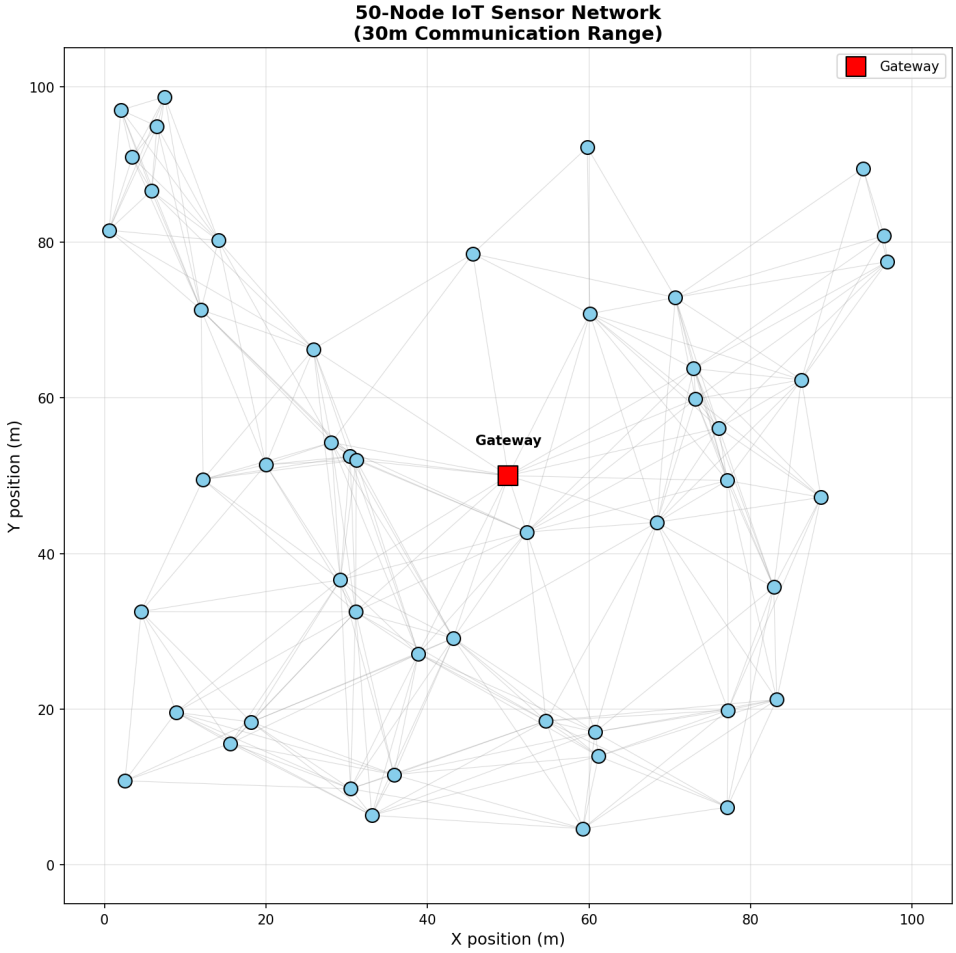


图14：由 Palma 生成的示例网络拓扑：一个包含服务器、两个路由器（Router_A、Router_B）和四个客户端的 7节点网络。边权表示双向链路延迟。此示例展示了使用不同热带半环计算最短路径和瓶颈路径。

表11：路由计算时间与网络规模

节点	最短路径	瓶颈	可达性	节省的内存
10	12.4 μ秒	8.2 μ秒	9.1 μ秒	42.1%
20	78.6 μ秒	51.3 μ秒	55.8 μ秒	48.3%
30	241.2 μ秒	156.7 μ秒	168.4 μ秒	51.2%
50	678.4 μ秒	395.1 μ秒	420.8 μ秒	54.6%

7.2.4 可扩展性分析

我们测试了节点数为 10 至 50 的网络：

7.2.5 分析

对于包含 50 个节点的网络，所有路由计算都能在 1 毫秒内完成，从而支持实时路由调整。稀疏表示可节省超过 50% 的内存，这对资源受限的传感器节点至关重要。统一的 Palma API 只需更改半环参数，即可在不同优化目标（延迟、带宽、能量）之间切换。

7.3 案例研究 3：制造生产线

7.3.1 问题描述

一家制造工厂运营着一条包含并行处理阶段和同步点的生产线。其目标是：

1. 计算单件产品的总工期（总生产时间） 2. 确定连续生产的周期时间（吞吐量） 3. 找出限制吞吐量的瓶颈工作站

7.3.2 ProductionLine 模型

这条生产线由七个工作站组成，结构如下：

- **输入**（5 秒）：原材料进入
- **并行阶段**：铣削（15 秒）、钻床（12 秒）、磨床（10 秒），均从输入工位接收原材料
- **焊接** (20 秒)：接收来自铣削、钻床和磨床的工件（同步点）
- **精加工** (8 秒)：接收来自焊接的工件
- **质量控制** (6 秒)：最终质量控制；反馈至输入以进行循环生产

关键路径为输入 → 铣削 → 焊接 → 精加工 → 质量控制，总工期为 54 秒。循环生产时，已完成的工件会反馈至输入。

7.3.3 结果

7.3.4 假设分析

使用 Palma，我们分析将焊接时间从 20 秒缩短至 15 秒的影响：
热带特征值计算支持快速假设分析，让生产工程师能够在微秒级评估改进方案。

7.4 总结：案例研究仪表板

8 相关工作

我们将相关工作分为四类，并指出 Palma 在这些类别交汇处填补的空白。

表 12：生产线分析结果

指标	数值	解释
单件生产		
总工期	54 s	单件总时间
关键路径	输入 → 铣削 →焊接 → 精加工 → 质量控制	最长路径
连续生产		
周期时间（λ）	10.8 秒	最大循环均值（特征值）
吞吐量	0.093 项/秒	333 项/小时
瓶颈	经过铣削的关键循环	确定 λ
计算性能		
调度求解时间	6.31 μs	（不适用）
特征值计算时间	6.28 μ秒	计算 λ 所需的时间

表13：瓶颈改进的影响

指标	之前	之后
焊接时间	20 s	15 s
周期时间	10.8 秒	9.8 秒
吞吐量	333 项/小时	367 项/小时
改进	（不适用）	+10%

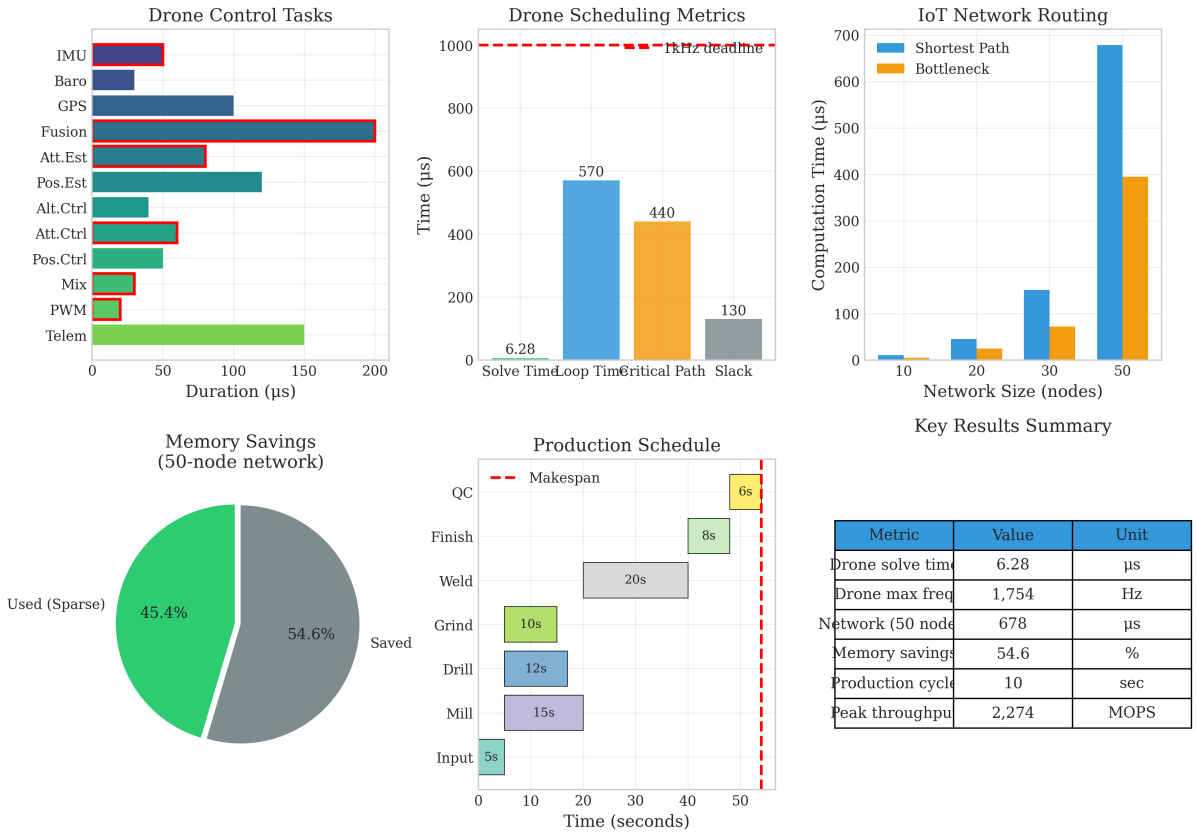


图15：案例研究仪表板，总结了三个应用中的结果。(a) 突出显示关键路径的无人机任务持续时间。(b) 无人机调度指标。(c) 网络规模与网络路由时间。(d) 网络内存节省。(e) 生产线甘特图。(f) 关键结果汇总表。

8.1 热带代数软件

热带代数的理论基础由 Cuninghame-Green [3] 奠定，并由 Baccelli 等人 [1]全面发展。Butkovič [2] 提供了算法方面的视角，而 Heidergott 等人 [11] 则给出了通俗易懂的入门介绍。

用于热带计算的数学软件包包括：

- **ScicosLab MaxPlus 工具箱** [6]：类似 MATLAB 的离散事件系统环境。需要图形桌面环境和 Scilab 运行时。
- **polymake** [7, 10]：综合性计算几何系统。需要 Perl、C++编译器工具链以及数 GB 的依赖项。
- **Macaulay2 Tropical**：侧重代数几何。需要完整的计算机代数系统。

差距：所有现有热带代数工具都面向桌面或服务器环境，依赖项繁多，因此不适合嵌入式部署。

8.2 图算法的半环框架

Mohri [16] 将半环与最短路径算法之间的联系形式化，指出 Dijkstra 和贝尔曼-福特算法是半环的特例。这种代数视角使算法统一成为可能。

通用图形库包括：

- **Boost 图形库** [18]：C++ 模板支持自定义半环，但元编程复杂，二进制文件体积也很大。
- **LEMON** [4]：C++ 接口更简洁，但仍然大量依赖模板，且主要面向桌面环境。
- **cuASR**：适用于 CUDA 的 GPU 半环通用矩阵乘法。仅支持服务器级 GPU。

差距：通用半环框架虽然存在，但需要 C++模板或 GPU 硬件，不适用于嵌入式 C 环境。

8.3 EmbeddedGraph 与调度计算

实时系统和嵌入式系统需要高效的图算法：

- 面向传感器网络的贝尔曼–福特优化 [5] 侧重于能量消耗，而非代数通用性。
- 刘和莱兰 [14] 奠定了实时调度理论的基础，但经典算法缺乏能够实现统一处理的热带代数结构。
- Goverde [9] 将极大加法应用于铁路调度，证明了其工业应用价值，但使用的是桌面工具。

差距：嵌入式图算法被实现为彼此孤立的特例，缺乏统一的半环抽象。

8.4 ARM 上的 SIMD 线性代数

ARM NEON 优化已得到广泛研究：

- Mitra 等人 [15] 针对科学计算内核采用手工调优的 NEON，实现了 1.05–13.88× 倍加速比。

- Park 等人 [17]展示了密码学中矩阵运算性能提升 36.93%。
- 近期的快速傅里叶变换研究 [13] 在 Cortex-A72 上取得了显著的加速比。

差距：NEON 优化针对的是传统（域上的）线性代数，而非幂等半环运算。

8.5 总结：Palma 的贡献

表 14：Palma 相对于现有工作的定位

系统	热带		SIMD	稀疏
ScicosLab MaxPlus	✓			
polymake	✓			
Boost 图形库	✓			
cuASR	✓		✓(图形处理器)	✓
嵌入式 Bellman-Ford 算法		✓		
Palma	✓	✓	✓(NEON)	✓

Palma 将热带代数与嵌入式优化实现独特地结合起来，填补了这四个研究领域交汇处的空白。

8.6 范围与局限性

Palma 专为嵌入式和实时系统中出现的中小规模优化问题而设计。这类问题的矩阵维度通常从几十到几千不等，且对确定性执行、内存效率和低部署开销有严格要求。该库同时支持稠密和稀疏表示，并可受益于 SIMD 加速，但并不旨在与高度优化的高性能计算或基于图形处理器的线性代数框架竞争，尤其是在处理超大型图或矩阵时。具体而言，热带闭包等最坏情况下复杂度为立方的算法，无论如何优化实现，对于大规模输入仍然不切实际。此外，Palma 专注于权重为整数的幂等半环；需要全程保持浮点精度、大规模动态更新图，或渐近速度更快的并行算法的应用，均不在当前范围内。这些局限是有意为之，体现了设计上的取舍：优先保证资源受限平台上的正确性、可预测性和可移植性，而非追求高端硬件上的绝对峰值吞吐量。

9 结论与未来工作

9.1 总结

我们介绍了 Palma，这是一个专为基于 ARM 的嵌入式系统设计的综合性热带代数库。我们的核心见解是，热带代数为嵌入式优化提供了一种统一的计算抽象：最短路径、瓶颈分析、可达性、调度和吞吐量计算——这些任务传统上需要各自独立的算法——都可以归结为在适当半环上进行矩阵运算。在资源受限平台上，这种统一尤为重要，因为代码大小至关重要。

我们的技术贡献包括：

1. **理论基础**：全面阐述热带半环、线性代数和谱理论，并提供严谨的定义与证明。

2. **统一实现**：提供通用的半环接口，支持五种热带代数（极大加法、极小加法、极大极小、极小极大、布尔），并支持稠密和 CSR 稀疏表示。 3. **SIMD 加速**：针对 ARM NEON 优化的内核可实现高达 1.80× 的加速比；分析指出，向量化在缓存受限区间效果最佳。 4. **实验验证**：全面的基准测试表明，峰值性能达到 2,274 百万次操作每秒；凭借更高的缓存利用率，性能比贝尔曼-福特算法提升 11.9× 倍；采用稀疏表示可节省 54.6% 的内存。 5. **实际应用**：三个案例研究（无人机控制、物联网路由、制造业）表明，Palma 可在树莓派级硬件上实现 1.7+ kHz 的控制回路和亚毫秒级路由决策。

Palma 填补了热带数学与嵌入式系统交叉领域的重要空白，使信息物理系统能够在设备端进行优化。该库以 MIT 许可证作为开源软件发布，包含约 2,000 行可移植、无依赖的 C99 代码。通过将代数语义与硬件特定优化解耦，Palma 奠定了坚实基础，可自然扩展至未来的嵌入式架构，并适应不断演进的现实世界优化工作负载。

9.2 未来工作

Palma 当前的设计展现出若干值得探索的方向。

并行性与异构平台。 目前的实现面向单核 SIMD 加速，未来工作将探索跨多个核心的任务级和数据级并行，包括基于 OpenMP 的并行化，以及对矩阵运算进行细致调度，以满足实时约束。另一个顺理成章的方向是扩展 Palma，使其能够利用异构嵌入式平台，例如集成图形处理器或 DSP 加速器的 ARM 系统。

动态与增量更新。 尽管 Palma 目前专注于静态图和矩阵，但许多嵌入式应用都涉及缓慢变化或事件驱动的更新。支持增量更新和动态更新，例如在边插入或权重变化时局部重新计算热带闭包或特征值，将在保持效率的同时显著拓宽其适用范围。

算法改进。 从算法角度来看，未来的工作将探索渐近速度更快或更适用于嵌入式环境的结构感知算法，包括针对稀疏热带闭包的剪枝策略、依据图拓扑制定的提前终止准则，以及适用于大型或稠密实例且具有有界误差保证的近似方法。

更丰富的代数结构。 扩展半环接口以支持更丰富的代数结构颇具意义。这包括参数化半环、字典序或多准则热带代数，以及有望在统一框架内实现多目标优化的混合符号-数值表述。

领域特定集成。 最后，与领域特定应用进行更紧密的集成也是一个重要方向。在机器人技术的控制回路、实时调度框架或联网物联网中间件中嵌入 Palma，并在闭环系统中验证其影响，将进一步展现热带代数如何成为嵌入式优化的实用计算基础。

9.3 可用性

Palma 的源代码、文档和示例可在以下位置获取：

`https://github.com/ReFractals/palma`

致谢

作者感谢 Axiom Research Group 富有价值的讨论，感谢 NM-AIST 和 AIMS-RIC 提供机构支持，并感谢树莓派基金会让全球研究人员能够使用价格实惠的计算设备。

参考文献

[1] François Baccelli、Guy Cohen、Geert Jan Olsder 和 Jean-Pierre Quadrat。《同步与线性：离散事件系统的代数》。John Wiley & Sons，英国奇切斯特，1992年。在线访问：`https://www.rocq.inria.fr/metalau/cohen/SED/book-online.html`。[2] Peter Butkovič。《最大线性系统：理论与算法》。施普林格数学专著丛书。施普林格，伦敦，2010年。[3] Raymond A. Cuninghame-Green。《极小极大代数》，经济学与数学系统讲义第166卷。施普林格出版社，柏林，1979年。[4] Balázs Dezső、Alpár Jüttner 和 Péter Kovács。LEMON——一个开源 C++ 图模板库。《理论计算机科学电子札记》，264(5)：23–45，2011年。[5] 丁志国、沈连峰、陈洪斌、严峰和 Nirwan Ansari。面向物联网的软件定义无线传感器网络的基于中继选择的节能动态路由算法。《IEEE物联网杂志》，7(9)：9050–9065，2020年。[6] Stéphane Gaubert。用于离散事件系统、静态分析和零和博弈的极大加代数工具。载于《定时系统的形式化建模与分析（FORMATS 2009）》，施普林格计算机科学讲义第5813卷。施普林格，2009年。[7] Ewgenij Gawrilow 和 Michael Joswig。polymake：用于分析凸多胞体的框架。载于《多胞体——组合学与计算》，DMV研讨会第29卷，第43–73页。Birkhäuser，2000年。[8] Michel Gondran 和 Michel Minoux。《图、幂等半环与半环：新模型与算法》，运筹学与计算机科学接口系列第41卷。施普林格，纽约，2008年。[9] Rob M. P. Goverde。使用极大加系统理论进行铁路时刻表稳定性分析。《交通研究B辑：方法论》，41(2)：179–201，2007年。

[10] Simon Hampe 和 Michael Joswig。polymake 中的热带计算。载于《代数、几何与数论中的算法与实验方法》，第361–385页。施普林格，2018年。[11] Bernd Heidergott、Geert Jan Olsder 和 Jacob van der Woude。《工作中的最大加代数：同步系统的建模与分析》。普林斯顿应用数学丛书。普林斯顿大学出版社，新泽西州普林斯顿，2006年。[12] Richard M. Karp。有向图中最小环平均值的刻画。《离散数学》，23(3)：309–311，1978年。[13] Krit Lajpop。ARM Cortex-A72 上使用 NEON 指令的快速傅里叶变换性能。载于《ICICT 2025》，网络与系统讲义第1416卷。施普林格，2025年。[14] Chung Laung Liu 和 James W. Layland。硬实时环境下多道程序设计的调度算法。《ACM学报》，20(1)：46–61，1973年。[15] Gaurav Mitra、Beau Johnston、Alistair P. Rendell、Eric McCreath 和 Jun Zhou。利用 SIMD向量运算提升低功耗 ARM 和英特尔平台上的应用代码性能。载于《IEEE国际并行与分布式处理研讨会》，2013年。[16] Mehryar Mohri。用于最短距离问题的半环框架与算法。《自动机、语言与组合学杂志》，7(3)：321–350，2002年。[17] Taehwan Park、Hwajeong Seo、Yeoncheol Park 和 Howon Kim。现代 ARM处理器上基于格的密码学矩阵乘法的高效并行实现。《安全与通信网络》，2018:7012056，2018年。[18] Jeremy G. Siek、Lie-Quan Lee 和 Andrew Lumsdaine。《Boost 图形库：用户指南与参考手册》。艾迪生-韦斯利，2002年。

A API参考

本附录提供 Palma 公共API的完整参考。

A.1 核心类型

```
typedef int32_t palma_val_t;     // Tropical value type
typedef uint32_t palma_idx_t;    // Index type for sparse matrices
```

```
    size_t rows, cols, nnz;
```

清单 19：核心类型定义

A.2 矩阵运算

```
// Creation and destruction
palma_matrix_t* palma_matrix_create(size_t rows, size_t cols);
```

```
pa
```

```
pa
```

```
pa
```

名

A.3 图算法

```
void palma_single_source_paths(const palma_matrix_t *A, size_t src,
```

```
;
```

```
ma
```

```
12                                  palma_semiring_t  s);列表
```

21：图算法签名

B 附加定理的证明

定理 B.1（热带运算的分配律）。 在任何热带半环中，分配律都适用于向量和矩阵。

证明。 对于维度兼容的矩阵 $\mathbf{A}$、$\mathbf{B}$、$\mathbf{C}$：

$$\begin{aligned}(\mathbf{A} \otimes (\mathbf{B} \oplus \mathbf{C}))_{ij} &= \bigoplus_k A_{ik} \otimes (B_{kj} \oplus C_{kj}) \\ &= \bigoplus_k (A_{ik} \otimes B_{kj}) \oplus (A_{ik} \otimes C_{kj}) \\ &= \left(\bigoplus_k A_{ik} \otimes B_{kj}\right) \oplus \left(\bigoplus_k A_{ik} \otimes C_{kj}\right) \\ &= (\mathbf{A} \otimes \mathbf{B})_{ij} \oplus (\mathbf{A} \otimes \mathbf{C})_{ij}\end{aligned}$$

第二行使用标量分配律；第三行使用$\oplus$的交换律和结合律。

□

定理 B.2（不可约矩阵的热带特征值唯一性）。 若$\mathbf{A}$ 不可约（其图是强连通的），则$\mathbf{A}$ 有唯一的热带特征值。

证明。 参见 Baccelli 等人 [1], 第 3 章。关键洞见在于，所有强连通分量都对应同一个特征值，该值等于所有初等回路的最大回路平均值。

□