arXiv:2601.07827v1 [cs.MS] 12 Jan 2026

张量代数处理原语（TAPP）：迈向张量运算标准

JanBrandejs,*,† NiklasH¨ornblad,‡ Edward F.Valeev,¶ Alexander Heinecke,§ Jeff Hammond,‖ Devin Matthews,⊥ and Paolo Bientinesi‡

†*Laboratoire de Chimie et Physique Quantique*，法国国家科学研究中心 *UMR 5626* — 图卢兹大学，*118 route de Narbonne*，*F-31062* 图卢兹，法国

‡计算机科学系，于默奥大学，*Ume ̊a*，*901 87*，瑞典

¶化学系，弗吉尼亚理工大学，布莱克斯堡，*VA 24061*，美国§英特尔先进技术集团，英特尔公司，圣克拉拉，美国。‖*NVIDIA*，*2788 San Tomas Expressway*，圣克拉拉，*CA 95051*⊥化学系，南卫理公会大学，达拉斯，*TX 75275*，美国

电子邮件：jbrandejs@irsamc.ups-tlse.fr

1 引言

张量运算推动着人工智能（AI）和量子科学建模等领域的发展。在这些领域中，计算开销最大的运算通常是张量缩并。张量缩并的性能直接影响材料科学、[1]量子化学、[2]药物发现、[3]生命科学等领域的进展。[4]这是因为在这些应用领域中，张量缩并的成本决定了可负担的最大模型大小。对于这些领域中的某些方法而言，张量缩并性能的提升一直是成功的基石，例如深度学习、量子计算模拟技术、张量网络方法以及许多量子化学

方法。如今，张量运算对整个世界的重要性和影响令人瞩目，尤其是随着人工智能技术的爆炸式增长。举例来说，预计未来几年全球每年用于张量缩并的电力将远超德国的总耗电量。[5]

尽管张量缩并软件在众多广泛应用中发挥着关键作用，但其发展还不如矩阵运算领域那样成熟且组织完善。张量软件开发者和用户的多样性及其数量的快速增长，导致算法和代码出现了明显的碎片化。事实上，最近的一项调查显示，现有大量分散的库，并且代码重复程度很高。[6] 造成这一现象的主要原因之一，是缺乏张量运算原语的标准化。广泛采用的张量运算标准将使开发者能够模块化其代码，并复用库，而无需担心难以替换的依赖项。遗憾的是，目前尚无此类标准。

近期，由张量软件开发者、应用专家和行业代表组成的社群齐聚一堂，共同应对这一问题。[7,8] 与会者普遍认为，多重线性代数运算（张量代数）的现状已经足够成熟，可以支持一套标准原子构建模块的形成。本文朝这一方向迈出了第一步：我们仔细定义了张量缩并问题，提出了标准接口，并提供了参考实现。

1.1 演进

四十多年来，**BLAS** （基本线性代数子程序）一直是基本线性代数运算事实上的标准接口。最初的规范现称为一级 BLAS，于1979年发布，旨在“支持”标量和向量运算，并被 LINPACK 库直接采用。[9,10] 鉴于在具有存储层次结构的现代架构上，仅执行向量运算存在性能局限，人们提出了二级 BLAS（矩阵-向量运算）规范，该提案于 1988,[11] 迅速

随后，1990年又提出了3级 BLAS（矩阵-矩阵运算），并被 LAPACK 库[12] 等采用。与1级和2级 BLAS 相比，3级 BLAS 的算术运算与内存访问之比（算术强度）更高，因此能够掩盖内存访问缓慢的问题——随着处理器速度超过内存，这一特性变得至关重要。[13]

BLAS 接口的初步成功得益于两个因素。一方面，固定接口使开发者能够提供高度优化的实现，从而实现性能可移植性。另一方面，BLAS 凭借其在 LINPACK 基准测试[14] 以及高度依赖高强度 3级运算的 LAPACK 中的广泛应用而受到关注。最终，BLAS 成为了矩阵运算的通用语言；尽管缺乏对张量缩并的支持，其地位仍基本未受到挑战。

BLAS 的标准化是自然形成的；它始于一个解决即时问题的参考实现（用于 LINPACK 的 1级），随后接口逐步扩展到矩阵运算，以最大限度地提高算术强度。这种自然发展是由自下而上的社区共识以及 LINPACK 和 LAPACK 等关键应用的实际需求推动的；这些应用通过广泛的实用性，而非僵化的自上而下指令，巩固了这一标准。

在其他标准化工作中，事件的顺序有所不同。在消息传递接口（MPI）建立之前，曾有大约 9 次标准化分布式通信的尝试未获成功。[15] 来自工业界、学术界和研究机构、具有不同背景的专家齐聚 MPI 论坛；经过多年努力，他们才就如何封装基本操作达成共识。

在节点内并行化标准化方面，最早的大规模努力是 POSIX 线程（pthreads）[16]，它为显式线程管理确立了低层标准接口。然而，pthreads 提供的接口复杂且层次较低。为了实现高性能计算应用的并行化，OpenMP 规范[17] 应运而生并得到广泛采用，这可

归功于其通过编译器指令实现增量并行化的高级编程模型。这表明，即使存在功能更强大但更复杂的替代方案，高性能计算社区仍倾向于选择高效且易于使用的标准。

借鉴以往和正在进行的标准化工作，我们认为，不应试图制定若干公理化规则，再在此基础上人为地构建标准。标准化本身是一个基于经验、社群互动与共识而自然演进的过程，而非可以由个人规划的确定性流程。一个关键的观察是，这一过程由社群内有针对性的讨论、经验分享和软件复用所推动。

这些软件标准（BLAS、POSIX 线程、OpenMP、消息传递接口）对计算领域产生了深远影响。BLAS 一旦被广泛采用，成为更高效利用硬件的方式，反过来也影响了硬件设计。经典计算硬件经过调优，能够执行通用矩阵乘法，因为 LINPACK 是用于评估，更重要的是用于比较机器性能的基准测试。[18] 高性能计算硬件与软件的发展相辅相成。科学计算领域的软件架构也受到这些标准的推动。例如，在量子化学中，张量缩并这一基本的多线性运算，长期以来一直通过将其映射为矩阵乘法来实现，这种模式通常称为“转置-转置-通用矩阵乘法-转置”（TTGT）。[19]

虽然张量缩并可以用矩阵运算来表示，但并不建议这样做。TTGT 方法虽然可行，但往往并非最优，因为数据重塑会带来额外开销，而且对于某些缩并来说效率不高。TTGT 将负担交给开发者，要求他们手动协调数据布局；此外，它还可能掩盖直接实现本可利用的性能优化机会。而且，将运算转换为矩阵乘积后，其底层多线性结构便会丢失。这促使人们开发专用的张量软件，以适合相关性能优化的方式封装张量运算。在

下文中，我们将回顾其中一些具有重要概念里程碑意义的工作。

早期探索

POOMA 于 1996年 发布，是最早在分布式多维数组之上使用表达式模板的 C 库，但不支持张量缩并。FTensor 于 1998 年首次发布，是最早的 C 张量缩并库之一，同样基于表达式模板构建。2003 年，张量缩并引擎（TCE）实现了张量缩并高效代码生成流程的自动化，并率先将循环遍历 GEMM 技术用于分布式张量，之后该技术也用于节点级优化。

分布式内存

Cyclops 张量框架（CTF）引入了一种分布式内存方案，使得在并行计算系统上能够缩并非常大的张量。Elemental 与 CTF 同期开发，致力于在分布式内存系统上构建统一的高性能线性代数框架，其中包含一些张量运算，但不包括张量缩并。此后，Elemental 通过 DiHydrogen 库扩展了对人工智能（ AI ）领域张量缩并的支持。TiledArray 支持在分布式内存环境中进行分块稀疏张量计算。

以硬件为中心的设计

Fastor[27] 将人们的注意力引向了以硬件为中心的设计，从底层构建，借助单指令多数据（SIMD）向量化来执行大小在编译时已知的张量缩并。Loops-over-GEMMs[23,28] 将张量切分为一系列二维子张量（矩阵），并通过通用矩阵乘法对其进行缩并。TBLIS[29] 提出了一种新颖的张量缩并方法：在调用矩阵乘法内核之前，通过在处理器缓存卸载期间对数据进行置换，隐藏转置步骤的成本。

与此同时，GETT 无转置方案[19] 也得到了验证，随后尤其针对图形处理器在 cuTENSOR[30] 中实现；cuTENSOR 为其他功能更完善的张量库提供了一组优化原语。cuTENSOR 使用精细的成本模型，通过即时编译生成针对给定缩并的最优代码，并再次通过在内核调用之前重新映射子张量，将转置与缩并步骤融合起来。

高级框架

虽然 NumPy[31] 并非专门的张量库，但它引入的$ndarray$ 对象及其强大的索引和广播功能，使其成为 Python 数值计算领域事实上的标准。针对该对象的$einsum$ 函数正日益广泛地得到采用，并成为 Python 社区中张量缩并接口的有力候选。TensorFlow[32] 和 PyTorch,[33] 是为机器学习开发的框架，它们通过让更广泛的用户群体能够进行高性能张量计算，脱颖而出。

张量编译器

张量编译器的基础思想在 Halide,[34] 中得以实现，尤其是将算法与调度解耦。张量代数编译器 (TACO)[35] 是首个能够处理涉及稀疏张量的任意张量代数表达式的张量编译器。张量虚拟机 (TVM)[36] 通过构建统一的编译栈，解决了机器学习面临的关键挑战——部署中的碎片化与不兼容问题。此外，所有面向人工智能的领域特定语言（领域特定语言）都基于 MLIR.[37]

现状与展望

张量缩并库开发的当前趋势包括新增稀疏计算、图形处理器加速、领域专用语言（DSL）、张量分解、[4]张量网络（TN）、[38] 以及面向神经网络的张量编译器等功能。[39,40]

除了此处列出的张量软件外，还有许多其他相关工作。[6] 我们想提醒读者，已经涌现出大量张量库，其中绝大多数都是独立、重复且欠优化地开发出来的。[6] 鉴于软件领域如此拥挤且缺乏条理，我们呼吁开展张量原语的识别与标准化工作，并将其作为可复用模块，供社区在此基础上构建。我们首先聚焦于张量缩并。

1.2 TAPP 的目标

我们观察到张量软件领域正在不断发展壮大，但尚未形成统一的张量运算标准接口。随着越来越多的研究人员进入这一领域，我们预计碎片化现象还会进一步加剧。目前唯一得到广泛采用的接口是 Python 风格的 `einsum`；虽然它是为高级脚本编写而设计的，但难以轻易推广到编译型语言或底层代码集成场景。

为弥补标准缺失的问题，我们定义了张量代数处理原语（TAPP），这是一种用于张量运算的基于 C 的标准接口。TAPP 的首要目标是为张量收缩运算定义接口，从而将应用层与实现层解耦。

我们的目标是制定类似 BLAS 的接口，以正确封装张量运算，同时实现性能可移植性。接口规范需要与清晰的参考实现同步制定。通过将特定张量内核提升为标准化的构建模块，TAPP 实现了明确的职责分离：

1. 应用开发者只需针对 TAPP 接口编写一次代码，即可使用多个后端；日后需要切换后端时，无需更改代码库。

2. 库开发者可以优化特定内核（TAPP 实现），而无需支持每一种应用框架。

3. 硬件供应商可以提供符合 TAPP 标准的库，确保更广泛的科学界能够使用其硬件。

该架构旨在借鉴 BLAS 的成功经验：建立一个透明的市场，在其中可以衡量和比较性能，并且后端中的优化能够自动传递到所有使用该标准的应用程序。

1.3 TAPP 生态系统

TAPP 倡议建立在学术界和工业界利益相关者的共识之上，提供一套资源，以实现相对简单的集成并确保长期可持续性。该生态系统的核心是本文介绍的参考实现。该实现优先考虑可读性，并在 TAPP 集成期间用于正确性检查。它使硬件供应商和库开发者能够将其优化内核与已知正确的结果进行验证；所使用的全面测试集还包含特殊的边界情况。参考实现的源代码已在 GitHub 上以宽松的 BSD-3 许可证公开提供。[41]

TAPP 社区旨在提供不断扩展的高性能后端。TBLIS 和 cuTENSOR 等库已经开发了 TAPP 接口，这表明该标准能够支持多种架构，而无需更改应用程序代码。

TAPP 采用开放治理模式，其发展方式类似于消息传递接口和 OpenMP。该标准由一个工作组负责维护，成员包括应用专家、库开发者和硬件供应商；他们曾于2024年5月在图卢兹参加 CECAM 张量缩并库标准化研讨会，并于2025年9月参加第二届图卢兹张量研讨会。这确保该标准能够根据实践反馈不断演进，而不是强行推行僵化的自上而下指令。

由此可见，TAPP 本身是一项范围更广的工作，除了本文之外，还包括一系列成果、会议和交流。不过，TAPP 刚刚迎来了一个重要

里程碑：参考实现已最终定稿。本文旨在此时向更广泛的社区介绍这一成果。参考实现本身是整个 TAPP 倡议的一项重要目标，也是本文的主要关注点，因为它将接口细化到具体细节，阐明了大多数模糊之处和边界情况；而在尝试严格定义接口时，这构成了一项重大挑战。

1.4 目标用例

DIRAC 的开发就是一个很有说服力的例子：它是一个历史悠久的 Fortran 量子化学软件包。最初，它是一个直接调用 BLAS 的旧代码库；后来，为了提升性能，关键模块经过现代化改造，开始依赖 ExaTENSOR 库。然而，ExaTENSOR 的主要开发者转投工业界后，该库便停止维护，随着时间推移，关键应用的开发因此陷入瓶颈。为解决这一问题，团队被迫重写整个代码库。这种“依赖地狱”凸显了制定标准的迫切性，也促使 DIRAC 团队通过 2024 年标准化研讨会向社区发出行动倡议。

如果 DIRAC 是基于 TAPP 接口编写的，那么从一个无人维护的后端切换到受支持的后端（例如 TBLIS 或供应商提供的库）只需在链接时作出选择，而不必重写代码。

这一情形并非量子化学领域独有。对于高级语言，TAPP 提供了稳定的应用程序接口，使 Julia、PyTorch 或 MATLAB 等框架能够安全地将计算量大的缩并操作卸载到用户可用的任意优化库中。

2 数学定义

由于两个张量的缩并是矩阵乘积的推广，因此自然可以将矩阵所用的记法扩展到张量上。具体而言，遵循 BLAS 接口，

矩阵乘积用以下三元运算表示：

$$C \coloneqq \alpha AB + \beta C, \tag{1}$$

其中 A, B 和 C 是大小兼容的矩阵，而 α 和 β 是标量。基于索引的记法明确地描述了如何计算 C 的元素：

$$\forall i, j: \; C_{i,j} \coloneqq \beta C_{i,j} + \alpha \sum_k A_{i,k} B_{k,j}. \tag{2}$$

张量的扩展似乎很直接：给定张量 A, B 和 C，将 A 与 B 缩并并更新 C，同样可以用方程 (1) 表示。然而，与二维情形（矩阵）不同，这种记法无法明确描述一般张量缩并中如何计算 C 的元素。事实上，表达式 (1) 背后的具体计算取决于不同张量的指标。

本节考虑方程 (1) 的一种广义形式：

$$D \coloneqq \alpha AB + \beta C, \tag{3}$$

并给出 TAPP 所支持的张量缩并及其变体运算的具体计算定义。

2.1 预备知识

用于描述张量及张量运算的术语，在不同学科之间（甚至同一学科内部）差异很大。本文选用一套“标准”术语——同义词及相关术语的术语表见附录 A。张量 T 由指标数量、指标的指标范围 以及元素（数据）来表征。具体而言，设 n 为整数，表示 T的指标数量。若 $n > 0$，则用正整数 d_i表示第 i个指标的范围，其中$i \in [1, \ldots, n]$。我们将对象 $T \in \mathbb{R}^{d_1 \times d_2 \times \cdots \times d_n}$

$_n$称为张量 T，其形状 为
$\bigcup\{d_i\}$。$_{i=1}$

注释 1 ： 本文档中介绍的所有内容均适用于定义在 $\mathbb{C}$ 上的张量 .

注释 2： 在此表示法中，标量对应于具有 $n = 0$ 个指标的张量，但也可以表示为所有范围均为 1 的张量。

注释 3： 此表示法不包含有关张量元素 T 在内存中的存储方式的信息。这意味着指标的任何排列都对应于同一个张量。例如，同一个张量 T 既可以定义为 $\mathbb{R}^{d_1\times d_2\times\cdots\times d_n}$，也可以定义为 $\mathbb{R}^{d_2\times d_1\times\cdots\times d_n}$ （指标经过排列）。

为方便起见，通常会为每个索引分配一个标签。 Id 是将索引映射到标签的函数。该函数满足条件：仅当 $d_i = d_j$ 时， $Id(i) = Id(j)$；这意味着，如果两个或更多索引的范围相同，那么它们可以共享同一个标签，也可以不共享。张量 T 的标签集记为 $\mathcal{T}$。它是 $\mathcal{T}$:
$= \bigcup\{Id(i)\}$、$_{i=1}$和 $1 \le |\mathcal{T}| \le n$。

2.2 运算

设 $n > 0$、 $p > 0$、 $r > 0$ 为整数，并考虑张量

$$A \in \mathbb{R}^{d_1\times d_2\times\cdots\times d_n},\ B \in \mathbb{R}^{e_1\times e_2\times\cdots\times e_p},\ \text{and}\ \ C, D \in \mathbb{R}^{f_1\times f_2\times\cdots\times f_r}.$$

注释 4： n、 p 和/或 r 也可以为 0。在这些情况下，运算 (3) 会变为张量缩放、两个标量的乘积或张量点积。为便于说明，我们重点讨论非退化情形。

注5： C 和 D 的指标仅相差一个排列。实际中，实现细节也会要求 C 和 D 具有相同的内存布局。

令 $\mathcal{A}$、$\mathcal{B}$、$\mathcal{C}$、$\mathcal{D}$ 分别表示 A、B、C、D的标签集，并令整数$o > 0$、$q > 0$、$s > 0$ 分别表示 $\mathcal{A}$、$\mathcal{B}$、$\mathcal{D}$的基数：$o := |\mathcal{A}|$、$q := |\mathcal{B}|$，以及$s := |\mathcal{C}| = |\mathcal{D}|$。方程（3）的含义取决于集合 $\mathcal{A}$、$\mathcal{B}$ 与 $\mathcal{D}$之间的关系及其基数。我们通过决定不支持 $\mathcal{D}$ 中的孤立索引，解决了这一问题（见第2.2.5节）。因此，在下文中，我们假设 $\mathcal{D}$ 中的每个索引也出现在 $\mathcal{A}$ 或 $\mathcal{B}$ 中（或同时出现在二者中），即：

$$\mathcal{D}/(\mathcal{A} \cup \mathcal{B}) = \{\}. \tag{4}$$

2.2.1 情形1："简单"缩并

这种情形是矩阵乘积在更高维度上的对应形式。$\mathcal{A}$ 中的指标分为两个互不相交的组：$\mathcal{P}$ 和 $\mathcal{F}_\mathcal{A}$，分别是收缩 指标和自由 指标。$\mathcal{B}$也同样如此：$\mathcal{P}$ 和 $\mathcal{F}_\mathcal{B}$ 分别包含收缩指标和自由指标。最后，$\mathcal{D}$（因此$\mathcal{C}$也如此）仅包含 A 和 B的自由指标：$\mathcal{D} \equiv \mathcal{F}_\mathcal{A} \cup \mathcal{F}_\mathcal{B}$。此外，同一张量中的任意两个指标都不共享同一个标签（所有指标标签均互不相同），即$n = o$、$p = q$、$r = s$.

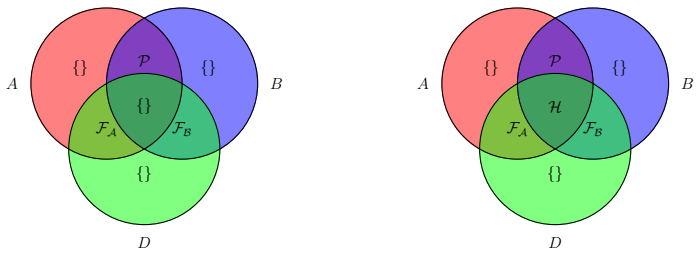

图1：标签集 $\mathcal{A}$、$\mathcal{B}$ 和 $\mathcal{D}$ 之间的关系。左：简单缩并（情形 1）；右：简单缩并和哈达玛积（情形 2）。

在这些条件下，乘积（3）通过与自由指标数量相同的 for循环计算

（$\mathcal{D} \equiv \mathcal{F}_{\mathcal{A}} \cup \mathcal{F}_{\mathcal{B}}$），并对缩并指标（$\mathcal{P} \equiv \mathcal{A} \cap \mathcal{B}$）进行求和：

$$D_{\mathcal{F}_{\mathcal{A}} \cup \mathcal{F}_{\mathcal{B}}} \coloneqq \alpha \cdot \sum_{\mathcal{P}} A_{\mathcal{P} \cup \mathcal{F}_{\mathcal{A}}} B_{\mathcal{P} \cup \mathcal{F}_{\mathcal{B}}} + \beta \cdot C_{\mathcal{F}_{\mathcal{A}} \cup \mathcal{F}_{\mathcal{B}}}. \tag{5}$$

示例 缩并 $D_{\delta\gamma} \coloneqq A_{\alpha\beta\gamma} B_{\beta\delta\alpha} + C_{\delta\gamma}$ 的计算方式为

$$\forall i \ \forall j \ D_{i,j} \coloneqq \alpha \sum_{k} \sum_{l} A_{k,l,j} B_{l,i,k} + \beta C_{i,j} \ , \tag{6}$$

其中，$i \in [1, \ldots, \text{extent}(\delta)]$、$j \in [1, \ldots, \text{extent}(\gamma)]$、$k \in [1, \ldots, \text{extent}(\alpha)]$, 以及$l \in [1, \ldots, \text{extent}(\beta)]$。

2.2.2 情形 2：哈达玛积与缩并

在第二种情形中，我们保留同一张量中任意两个指标都不共享同一标签这一约束（即 $n = o$、$p = q$、$r = s$.），但现在允许一个或多个指标同时出现在 $\mathcal{A}$、$\mathcal{B}$ 和 $\mathcal{D}$ 中。这组指标（$\mathcal{H} \coloneqq \mathcal{A} \cap \mathcal{B} \cap \mathcal{D}$）表示逐元素乘法，也称为哈达玛积。

具体而言，$\mathcal{A}$、$\mathcal{B}$ 和 $\mathcal{D}$ 中的指标分别拆分为三个互不相交集合：$\mathcal{A} \equiv \mathcal{P} \cup \mathcal{F}_{\mathcal{A}} \cup \mathcal{H}$、$\mathcal{B} \equiv \mathcal{P} \cup \mathcal{F}_{\mathcal{B}} \cup \mathcal{H}$ 和 $\mathcal{D} \equiv \mathcal{H} \cup \mathcal{F}_{\mathcal{A}} \cup \mathcal{F}_{\mathcal{B}}$。对于 $\mathcal{H}$ 中的所有指标，乘积 (3) 均按元素计算；其余情况下，则对指标 $\mathcal{P}$ 进行“情形 1 缩并”：

$$\forall \mathcal{H} : D_{\mathcal{H} \cup \mathcal{F}_{\mathcal{A}} \cup \mathcal{F}_{\mathcal{B}}} \coloneqq \alpha \cdot \sum_{\mathcal{P}} A_{\mathcal{H} \cup \mathcal{P} \cup \mathcal{F}_{\mathcal{A}}} B_{\mathcal{H} \cup \mathcal{P} \cup \mathcal{F}_{\mathcal{B}}} + \beta \cdot C_{\mathcal{H} \cup \mathcal{F}_{\mathcal{A}} \cup \mathcal{F}_{\mathcal{B}}}. \tag{7}$$

示例 缩并 $D_{\delta\alpha} \coloneqq A_{\alpha\beta} B_{\beta\delta\alpha} + C_{\delta\alpha}$ 的计算结果为

$$\forall i \ \forall j \ D_{j,i} \coloneqq \alpha \sum_{k} A_{i,k} B_{k,j,i} + \beta C_{j,i} \ , \tag{8}$$

其中 $i \in [1, \ldots, \text{范围}(\delta)]$、$j \in [1, \ldots, \text{范围}(\alpha)]$, 和 $k \in [1, \ldots, \text{范围}(\beta)]$。

2.2.3 情形 3：重复指标

与情形 2 一样， $\mathcal{A}$、 $\mathcal{B}$和 $\mathcal{D}$中的指标分别拆分为三个互不相交集合： $\mathcal{A} \equiv \mathcal{P} \cup \mathcal{F}_\mathcal{A} \cup \mathcal{H}$、 $\mathcal{B} \equiv \mathcal{P} \cup \mathcal{F}_\mathcal{B} \cup \mathcal{H}$和 $\mathcal{D} \equiv \mathcal{H} \cup \mathcal{F}_\mathcal{A} \cup \mathcal{F}_\mathcal{B}$。然而，与情形 2 不同的是，现在范围相同的指标可以共享同一个标签，即 $n \leq o$， $p \leq q$，和 $r \leq s$，其中 n， p和 r分别是 A， B和 D中指标的数量，而 $o := |\mathcal{A}|$、 $q := |\mathcal{B}|$和 $s := |\mathcal{D}|$ 分别是 $\mathcal{A}$、 $\mathcal{B}$和 $\mathcal{D}$中不同标签的数量。从计算角度看，此情形等价于情形 2（方程 (7)），循环数量与唯一索引的数量相同。

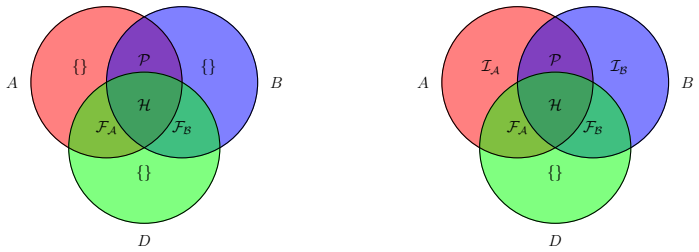

图 2：标签集 $\mathcal{A}$、 $\mathcal{B}$ 和 $\mathcal{D}$之间的关系。左：重复指标（情形 3）；右：孤立索引（情况 4）。

示例 缩并 $D_{\delta\alpha} := A_{\beta\alpha\beta}B_{\beta\delta\alpha} + C_{\delta\alpha}$ 计算如下

$$\forall i \; \forall j \; D_{j,i} := \alpha \sum_k A_{k,i,k}B_{k,j,i} + \beta C_{j,i} \; , \tag{9}$$

其中 $i \in [1, \ldots,$ 范围$(\delta)]$、 $j \in [1, \ldots,$ 范围$(\alpha)]$, 和 $k \in [1, \ldots,$ 范围$(\beta)]$。

注释 6： 具有重复指标标签和跨步布局 的张量 A ，可以等价地表示为具有唯一标签（因此指标更少）的张量 A'，两者的关系是： A' 中每个指标的步幅 等于 A 中所有具有相同标签的指标的步幅之和。

2.2.4 情况 4： $\mathcal{A}$ 或 $\mathcal{B}$中的孤立索引

这种情况比前几种情况更一般，因为现在某些标签可能只存在于 $\mathcal{A}$ 中，或只存在于 $\mathcal{B}$ 中： $\mathcal{I}_{\mathcal{A}} \coloneqq \mathcal{A}/(\mathcal{B} \cup \mathcal{D})$，以及 $\mathcal{I}_{\mathcal{B}} \coloneqq \mathcal{B}/(\mathcal{A} \cup \mathcal{D})$。这意味着 $\mathcal{A}$ 和 $\mathcal{B}$ 中的指标被分成四个互不相交集合： $\mathcal{A} \equiv \mathcal{H} \cup \mathcal{F}_{\mathcal{A}} \cup \mathcal{P} \cup \mathcal{I}_{\mathcal{A}}$，以及 $\mathcal{B} \equiv \mathcal{H} \cup \mathcal{F}_{\mathcal{B}} \cup \mathcal{P} \cup \mathcal{I}_{\mathcal{B}}$；而 $\mathcal{D}$ 中的指标则被分成三个互不相交集合： $\mathcal{D} \equiv \mathcal{H} \cup \mathcal{F}_{\mathcal{A}} \cup \mathcal{F}_{\mathcal{B}}$。

$\mathcal{I}_{\mathcal{A}}$ （$\mathcal{I}_{\mathcal{B}}$）中的指标表示在张量 A （B）中进行归约。在计算上，

$$\forall \mathcal{H} : D_{\mathcal{H} \cup \mathcal{F}_{\mathcal{A}} \cup \mathcal{F}_{\mathcal{B}}} \coloneqq \alpha \cdot \sum_{\mathcal{P}} \left(\sum_{\mathcal{I}_{\mathcal{A}}} A_{\mathcal{H} \cup \mathcal{F}_{\mathcal{A}} \cup \mathcal{P} \cup \mathcal{I}_{\mathcal{A}}} \right) \left(\sum_{\mathcal{I}_{\mathcal{B}}} B_{\mathcal{H} \cup \mathcal{F}_{\mathcal{B}} \cup \mathcal{P} \cup \mathcal{I}_{\mathcal{B}}} \right) + \beta \cdot C_{\mathcal{H} \cup \mathcal{F}_{\mathcal{A}} \cup \mathcal{F}_{\mathcal{B}}} \quad (10)$$

.

示例 缩并 $D_{\delta\alpha} \coloneqq A_{\beta\alpha\beta} B_{\beta\gamma\delta\gamma\alpha} + C_{\delta\alpha}$ 的计算结果为

$$\forall i \ \forall j \ D_{j,i} \coloneqq \alpha \sum_{k} A_{k,i,k} \left(\sum_{l} B_{k,l,j,l,i} \right) + \beta C_{j,i} \ , \quad (11)$$

其中 $i \in [1, \ldots, \text{范围}(\delta)]$， $j \in [1, \ldots, \text{范围}(\alpha)]$， $k \in [1, \ldots, \text{范围}(\beta)]$,，且 $l \in [1, \ldots, \text{范围}(\gamma)]$。

2.2.5 情况 5： $\mathcal{D}$中的孤立索引

在最一般的情况下，一些标签也可能只存在于 $\mathcal{C}$中，因此 $\mathcal{D}$： $\mathcal{I}_{\mathcal{C}} \coloneqq \mathcal{C}/$（$\mathcal{A} \cup \mathcal{B}$）。这意味着 $\mathcal{C}$ 和 $\mathcal{D}$ 中的指标也分为四个互不相交集合， $\mathcal{C} = \mathcal{D} \equiv \mathcal{H} \cup \mathcal{F}_{\mathcal{A}} \cup \mathcal{F}_{\mathcal{B}} \cup \mathcal{I}_{\mathcal{C}}$，使互不相交集合的总数达到七个： $\mathcal{P}$、 $\mathcal{F}_{\mathcal{A}}$、 $\mathcal{F}_{\mathcal{B}}$、 $\mathcal{H}$、 $\mathcal{I}_{\mathcal{A}}$、$\mathcal{I}_{\mathcal{B}}$和 $\mathcal{I}_{\mathcal{C}}$，分别对应图 1 和图 2 中维恩图的全部七个区域。

$\mathcal{I}_{\mathcal{C}}$ 中的指标表示将 A 和 B 的乘积中的元素广播到 D 中的多个（假定唯一的）位置。从计算角度来看，

$$\forall \mathcal{I}_{\mathcal{C}} \ \forall \mathcal{H} : D_{\mathcal{H} \cup \mathcal{F}_{\mathcal{A}} \cup \mathcal{F}_{\mathcal{B}} \cup \mathcal{I}_{\mathcal{C}}} \coloneqq \alpha \cdot \sum_{\mathcal{P}} \left(\sum_{\mathcal{I}_{\mathcal{A}}} A_{\mathcal{H} \cup \mathcal{F}_{\mathcal{A}} \cup \mathcal{P} \cup \mathcal{I}_{\mathcal{A}}} \right) \left(\sum_{\mathcal{I}_{\mathcal{B}}} B_{\mathcal{H} \cup \mathcal{F}_{\mathcal{B}} \cup \mathcal{P} \cup \mathcal{I}_{\mathcal{B}}} \right) + \beta \cdot C_{\mathcal{H} \cup \mathcal{F}_{\mathcal{A}} \cup \mathcal{F}_{\mathcal{B}} \cup \mathcal{I}_{\mathcal{C}}}.$$

(12)

示例 缩并 $D_{\delta\alpha\epsilon} := A_{\beta\alpha\beta}B_{\beta\gamma\delta\gamma\alpha} + C_{\delta\alpha\epsilon}$ 的计算方式为

$$\forall m\ \forall i\ \forall j\ D_{j,i,m} := \alpha \sum_k A_{k,i,k} \left(\sum_l B_{k,l,j,l,i} \right) + \beta C_{j,i,m}\ , \tag{13}$$

其中 $i \in [1, \ldots,$ 范围(δ)]、 $j \in [1, \ldots,$ 范围(α)]、 $k \in [1, \ldots,$ 范围(β)]、 $l \in [1, \ldots,$ 范围(γ)]，以及 $m \in [1, \ldots,$ 范围(ϵ)]。

注 7： 对输入张量进行（部分）归约（情况 4），以及将输入张量广播到输出张量（情况 5），可能会导致冗余计算，或需要不受限制的工作空间。为了确保行为定义明确且接近最优，TAPP 目前不要求支持情形 4 和情形 5。

2.2.6 一元和二元运算

类比于缩并（三元张量运算），形式为 $C := \alpha A + \beta B$ 的二元运算可以根据索引标签字符串，以及标签在 A 和/或 B/C 中的出现情况所形成的不相交子集来理解（与缩并类似，必须满足$\mathcal{B} = \mathcal{C}$）： $\mathcal{A} \equiv \mathcal{H} \cup \mathcal{I}_\mathcal{A}$ 和 $\mathcal{B} = \mathcal{C} \equiv \mathcal{H} \cup \mathcal{I}_\mathcal{B}$。与之前一样， $\mathcal{I}_\mathcal{A}$ 中的指标表示归约，而 $\mathcal{I}_\mathcal{B}$ 中的指标表示广播。最后，一元运算（例如张量缩放（$B := \alpha A$））没有不同的标签子集，但重复标签可能表示对张量（半）对角元素的运算。虽然对于高性能计算而言不那么重要，但TAPP为方便性和一致性起见，提供了二元和一元运算的接口。

3 接口

3.1 相关工作

关于张量运算标准化，过去已有一些相关工作，但没有任何接口得到广泛采用。BTAS [46,47] 旨在提供一个标准和基础的CPU实现。在2022年Dagstuhl张量研讨会期间[48]，领域专用张量语言的开发工作已经启动，但尚未完成，也没有文档记录。此外，还有关于标准化其他方面技术规范的工作，例如张量内存分布[49,50]和随机多重线性代数[51]。最后但同样重要的是，TBLIS和cuTENSOR的作者之间曾就标准化展开讨论。他们的一份接口草案最终成为实践讨论期间的起始草案，而这些讨论促成了TAPP。

3.2 设计决策：可能的方案

编码张量代数有两种常见的高层次方式：

- **张量嵌入式领域专用语言**（嵌入式领域专用语言）：只有在支持运算符重载的语言中，才能通过应用程序接口实现更通用、类似数学表达式的接口；运算符重载使我们能够嵌入一种用于张量代数的领域专用语言（DSL）。

- **Einsum**：`einsum` 方法是一种广受欢迎的方式，用于编码单个张量网络的求值。张量网络是对多个张量进行更一般的缩并（超出上文讨论的三元运算）。这种接口的优点是可以在任何支持函数的语言中实现。

ITensor、CTF 和 TiledArray 都是提供嵌入式领域专用语言的张量框架。它们之间的主要区别在于，ITensor 会在构造时将指标永久绑定到张量上，

例如，公式（3）的以下实例，

$$D_{il} := \alpha A_{ijk} B_{jlk} + \beta C_{li} \tag{14}$$

在 ITensor 中可以编码为，

```
auto i = itensor::Index("i", i_len), j = itensor::Index("j", j_len);
auto k = itensor::Index("k", k_len), l = itensor::Index("l", l_len);
auto A = itensor::ITensor(i, j, k), B = itensor::ITensor(j, l, k);
auto C = itensor::ITensor(i, l);auto D = alpha * A * B + beta * C;
```

其中 alpha 和 beta 是浮点变量。CTF 和 TiledArray 则生成局部索引绑定（通过字符串编码，并在运行时解析），以表示表达式中的元素。这样做是必要的，例如，才能在表达式的多个部分或多个表达式中，使用具有不同索引绑定的同一个张量。例如，方程（14）在 CTF 中的编码，

```
Tensor<double> A(i_len, j_len, k_len), B(j_len, l_len, k_len);
Tensor<double> C(i_len, l_len), D(i_len, l_len);
D["il"] = alpha * A["ijk"] * B["jlk"] + beta * C["il"];
```

以及在 TiledArray 中的编码，极为相似：

```
TA::TiledRange shape_A{tr_i, tr_j, tr_k},  shape_B{tr_j, tr_l, tr_k};
TA::TiledRange shape_C{tr_i, tr_l};
TA::TArrayD A(world, shape_A),  B(world, shape_B);
TA::TArrayD C(world, shape_C),  D(world, shape_C);
D("i,l") = alpha * A("i,j,k") * B("j,l,k") + beta * C("i,l");
```

其中，一维范围 tr_i..... tr l 可选择进行分块（例如，用于支持块稀疏性），其定义如下： i len..... l len：

```
TA::TiledRange1 tr_i{0, [...] i_len};
```

libtensor[52] 提供了类似的接口，不同之处在于指标是独立对象，不过不像 ITensor 中那样永久绑定。方程 (14) 在 libtensor 中的一种编码示例如下：

```
bispace<1> i_sp(i_len), j_sp(j_len), k_sp(k_len), l_sp(l_len);
btensor<3> A(i_sp & j_sp & k_sp), B(j_sp & l_sp & k_sp);
btensor<2> C(i_sp & l_sp), D(i_sp & l_sp);letter i, j, k, l;
D(i|l) = alpha * mult(A(i|j|k), B(j|l|k)) + beta * C(i|l);


```

NumPy einsum 可能是应用最广泛的高级接口。它在概念上与 Cyclops CTF 的接口类似，基于简洁而富有表现力的字符串：

```
A = np.random.rand(i_len, j_len, k_len)
B = np.random.rand(j_len, l_len, k_len)
C = np.random.rand(i_len, l_len)
D = alpha * np.einsum('ijk,jlk->il', A, B) + beta * C
```

值得注意的是，除原有的嵌入式领域特定语言外，许多库（如 CTF 和 TiledArray）后来也加入了对 einsum的支持，不过这些支持通常构建在较低层级的嵌入式领域特定语言之上。一个重要区别是， einsum 支持几乎任意的运算字符串，可用于表示缩并（涵盖第 2.2 节讨论的任意情形），以及二元、一元或张量网络（多张量）运算。在 TAPP 中，低层接口旨在实现最高程度的优化，这意味着

针对 `einsum` 所支持的特定运算子类，然后将这些运算组合起来，以完成更复杂的运算。

3.3 接口组件

TAPP 接口由一系列相关的数据结构（以不透明的对象句柄表示）、用于创建和操作对象的应用程序接口，以及用于执行所需计算的计算应用程序接口组成。我们按照图3所示的一般工作流程的大致顺序讨论这些元素。图3展示了各种对象与计算应用程序接口之间的关系，并给出了应用程序接口签名示例。

3.3.1 库句柄

除了创建空的键值存储之外，所有 TAPP 运算都需要库句柄。该对象表示后端的内部状态，而这一状态可能是全局的，也可能不是。虽然并非强制要求，但我们鼓励实现方在创建库句柄时执行所有耗时的初始化操作，并在句柄释放时进行清理，从而节省用户资源。在更复杂的情况下，用户获取的多个库句柄可能代表不同的 TAPP 后端，其对其他 TAPP 对象、算法或其他属性采用不同（且不兼容）的表示形式。因此，每个不同的库句柄都必须与使用该句柄创建的所有对象分开管理。另一方面，较简单的后端可能只将单个库句柄定义为常量，用以表示全局后端状态。

3.3.2 执行器

可以使用库句柄获取执行器对象，它代表一个或多个能够执行张量运算的资源。例如，基于 CPU 的实现可以提供一个或多个执行器，分别对应于（物理或逻辑）CPU 核心的各种分区、特定的核心位掩码、NUMA 域等。对于

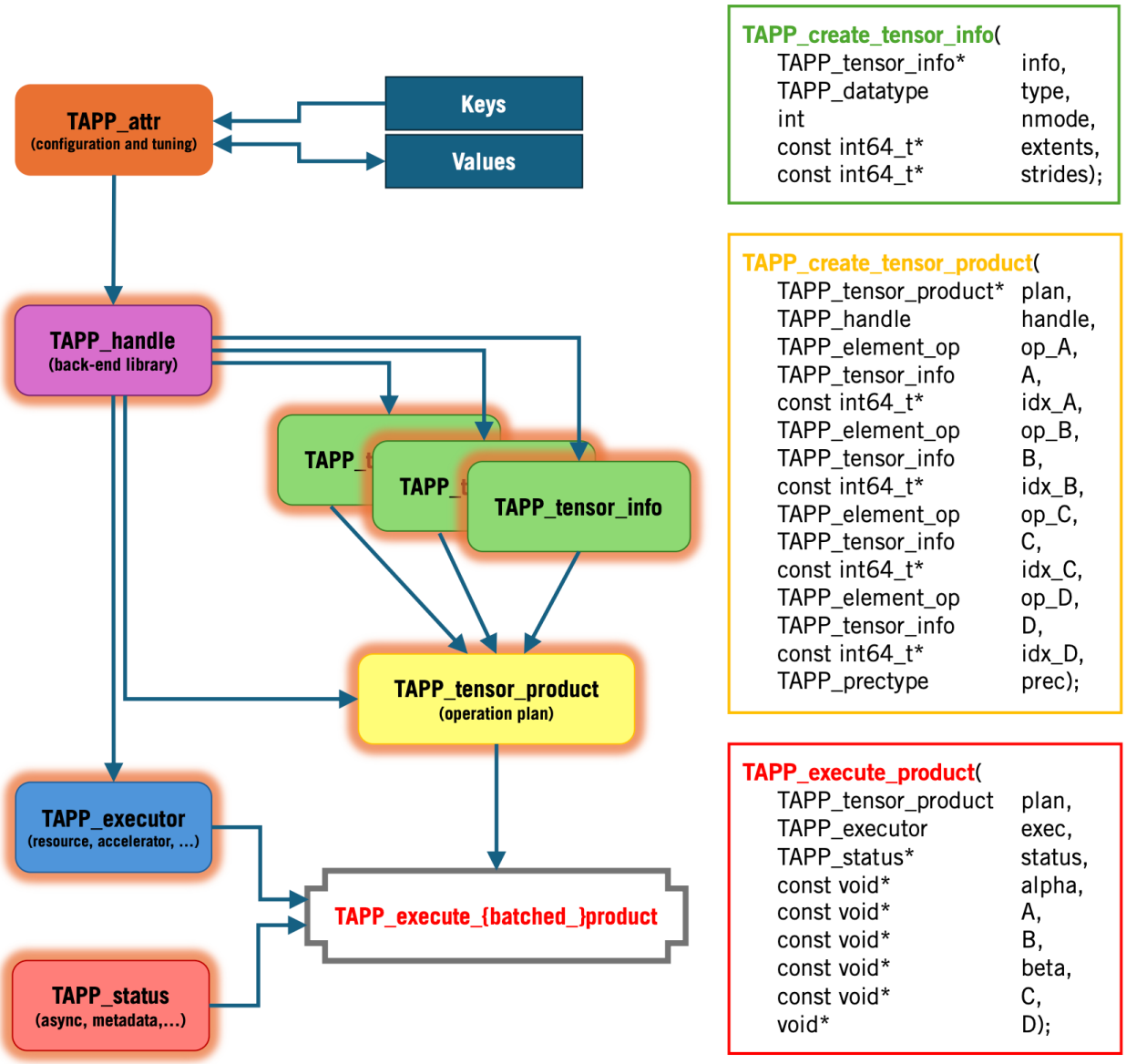

图3：TAPP接口组件的流程图，展示了 TAPP API 签名示例。除专用的 `TAPP attr` 对象外，所有 TAPP 对象都支持键值语义，以橙色光晕表示。

对于基于 GPU 的后端，执行器可以对应物理设备和/或流等抽象概念。为便于使用，库通常应定义一个常量“默认”执行器值，无需初始化即可将其传递给其他 TAPP 函数（类似于 CUDA 的 “ 流 0” 或消息传递接口的 `MPI COMM WORLD` ）。

- -

3.3.3 张量描述符

张量描述符提供有关张量逻辑形状和物理内存布局的信息。作为稠密张量库，TAPP 指定了一种通用步幅布局，类似于 `std::layout stride` 在 C++23 中的布局。张量描述符还定义张量的数据类型。不过，它不定义指向张量数据的指针。相反，张量描述符是对所有形状和结构相同的张量的一种抽象规范。这一特性使用户能够灵活地重复使用张量描述符来处理相似数据或执行批量张量运算。

3.3.4 张量运算描述符

与张量描述符一样，特定张量运算的抽象规范独立于该运算所处理的具体数据以及其执行的具体实例。这一抽象规范存储在张量操作描述符对象中（每种不同的张量运算对应一种类型）。这些对象基于库句柄和一组张量描述符构造，同时还包含其他特定于运算的信息，例如所需的计算精度。将张量操作描述符与实际的运算执行分离，并在初始化运算时而非执行时提供张量描述符，是使库后端能够采用一系列优化策略的重要一步。例如，后端可以检查运算及其操作数，并通过启发式方法（甚至根据经验）确定要使用的最优算法。后端甚至可以利用即时编译（JIT）技术构建特定的优化算法。在为此类优化提供实现机会的同时，将张量操作描述符与执行过程分离也使得这类优化的执行成本能够分摊到多次执行中。

将张量操作描述符与执行过程分离，也使得执行此类优化的成本能够分摊到多次执行中。

3.3.5 执行

获取库句柄、执行器和张量操作描述符（通过各个张量描述符）后，即可对特定数据执行运算。此时，只需要指向张量数据的指针（以及标量）。由于张量的数据类型已在张量描述符中单独指定，因此执行 API 自然与类型无关（不过，混合类型支持取决于后端的能力）。后续执行时，可以重复使用相同的张量操作描述和其他对象，并搭配相同或不同的数据。这鼓励用户缓存或以其他方式保存这些对象，而不是每次单独执行运算时都重新创建它们。

3.3.6 状态对象

执行张量运算时，可以选择创建状态对象。此对象可用于多种用途，而 TAPP 接口并未要求它具有某一种特定含义。例如，在异步环境中，状态对象可以提供一个句柄，用于等待运算完成、指定数据依赖等。状态对象还可以提供执行的元数据，例如性能计数器、执行时间或算法细节。状态对象与错误代码不同（参见第 3.3.8 节），不过它也可能提供与报告错误相关的更多数据。

3.3.7 虚拟键值存储（VKVs）

TAPP 接口旨在适配各种各样的张量后端，并提供多种选项来调整性能及其他设置。同样，随着新架构、实现技术和算法不断涌现，TAPP 接口本身也有望随时间演进。因此，TAPP 包含一种自由形式的机制，用于提供信息以供底层实现使用，

这些信息通过虚拟键值存储传递。从本质上说，键值存储表示从整数键（例如枚举值）到任意二进制值的映射。这些存储之所以称为“虚拟”，是因为尽管它们遵循传统映射数据结构的设置/获取 API，但实现方式不受限制，例如可以作为一层接口，用于访问分布在整个库中的配置值。

虽然 TAPP 目前未定义任何后端必须支持的必需键，但通过此机制可以支持多种用例：

- 通过包装器接口自定义库句柄的初始化（参见第 3.3.1 节），或选择特定后端。
- 根据 CPU 核心位掩码、设备 ID 或其他标准选择执行器（参见第 3.3.2 节）。
- 在张量描述符中提供有关数据局部性的信息（参见第 3.3.3 节），例如 NUMA 域、设备内存、共享/固定内存等。
- 选择特定算法，或为张量运算提供优化提示（参见第 3.3.4 节）。

TAPP 提供一种“裸”VKV 对象类型；但除非另有说明，所有其他 TAPP 对象也支持通过统一接口设置和获取键值对。

3.3.8 错误处理

几乎所有 TAPP API 函数都会返回一个整数错误代码。虽然没有为特定错误定义具体的整数值，但可以像 POSIX 中的 `strerror` 一样，获取错误代码的纯文本说明。同样，用户始终至少可以检查操作是否成功。由于每个函数都会返回错误代码，因此所有输出值都通过指针提供。在某些情况下，可以忽略输出值（或者说，要求不返回这些值），方法是传入

NULL。在其他一些情况下，出于安全或可靠性考虑，逻辑上本应仅作为输入的操作数会被视为输入/输出，并通过指针传递。例如，销毁 TAPP 对象时，会通过指针传递对象句柄，以便将其设为一个定义明确的值，表示对象尚未初始化（例如 NULL）。

3.3.9 可扩展性与兼容性

TAPP API 旨在同时保持向前兼容和向后兼容。其中一个重要方面是虚拟键值存储机制，它不仅允许存在特定于实现的行为，还能随着 TAPP 的发展演进，添加标准化键集合。与消息传递接口、C++ 和 OpenGL 等其他重要标准一样，TAPP 的扩展很可能会对常见的特定于实现的功能加以采纳并编纂成规范；这些功能可能源于趋同演进，也可能源于社区共识。除了标准化键值元数据外，TAPP 还计划通过新增张量运算、物理表示或更复杂的内部结构来不断发展，而实现这些功能的方式是添加新的 API 函数和/或 TAPP 对象。TAPP 将高度依赖社区意见和既有实践，来识别并定义新功能。

4 参考实现

参考实现旨在为 TAPP 当前支持的所有运算提供可靠且易于阅读的实现。因此，其主要关注点是正确性和简洁性，而不考虑性能。对于实现开发者而言，参考实现应作为支持 TAPP 时需要实现哪些内容的示例。其他实现可以有更多限制，但对于相同输入，仍应产生相同结果，除非该实现不支持此类输入。例如，出于技术或性能方面的原因，实现可以不支持负步幅，但通常预期实现支持

比参考实现少得多的混合数据类型组合；参考实现支持 64、32 和 16 位（如果编译器支持）实数和复数浮点数的任意混合。与接口一样，参考实现采用 C 语言开发，保持低层、简洁且可移植。

4.1 算法结构

收缩算法的结构首先从计划和张量信息对象中提取元素，例如指标、步长和范围。在提取出核心参数后，算法继续进行错误检查，以验证所有前置条件，例如共享同一标签的指标具有相同的范围。

在算法主体部分（图 4）之前，需要进行一些必要的预处理。在此预处理阶段，指标按照第 2.2 节中的说明分为哈达玛指标组、收缩指标组以及若干自由指标组。利用提取出的指标，可以计算循环大小（相关范围的乘积），并为每个张量及相关指标组提取步长和范围。虽然图 4 为每个自由指标组/哈达玛指标组展示了不同的循环，但实现会将 D 中的所有指标组合在一起（作为 $\mathcal{D} = \mathcal{H} \cup \mathcal{F}_A \cup \mathcal{F}_B$），置于一个循环结构中。对于重复指标，则通过移除重复项并合并其步长来进行处理。随后为每个张量分配步长数组：所有张量都具有自由指标步长数组，而 A 和 B 还会额外获得收缩指标步长。对于某个特定张量中未出现的指标所对应的步长（例如 B 中的 $\mathcal{F}_A$ 指标），则将其设置为零。

该算法的核心在逻辑上由四个主要循环组成，如图 4 所示。前三个（外层）循环遍历指标组合，这些指标会影响所有张量 A、B 和 D 中值的位置（因此也会影响 C，因为假设它们具有相同的结构）。随后，内层循环遍历仅影响 A 和 B 的指标组合，生成的值可以直接加到 D 的单个元素值上，而该元素最初被初始化为来自 C 的缩放值。递增到下一个组合的

```
function contract(alpha, beta, P, FA, FB, H, A, B, C) {
    size_FA <- prod(extent(i) for i in FA)
    size_FB <- prod(extent(j) for j in FB)
    size_P <- prod(extent(k) for k in P)
    size_H <- prod(extent(l) for l in H)

    L <- (0 for l in H)
    for (l in 0 to size_H-1) {
        I <- (0 for i in FA)
        for (i in 0 to size_FA-1) {
            J <- (0 for j in FB)
            for (j in 0 to size_FB-1) {
                off_D <- offset(L, stride(D, l) for l in H) +
                         offset(I, stride(D, i) for i in FA) +
                         offset(J, stride(D, j) for j in FB)
                D[off_D] <- beta * C[off_D]

                K <- (0 for k in P)
                for (k in 0 to size_P-1) {
                    off_A <- offset(L, stride(A, l) for l in H) +
                             offset(I, stride(A, i) for i in FA) +
                             offset(K, stride(A, k) for k in P)
                    off_B <- offset(L, stride(B, l) for l in H) +
                             offset(J, stride(B, j) for j in FB) +
                             offset(K, stride(B, k) for k in P)
                    D[off_D] <- D[off_D] + alpha * A[off_A] * B[off_B]

                    increment(K, extent(k) for k in P)
                }
                increment(J, extent(j) for j in FB)
            }
            increment(I, extent(i) for i in FA)
        }
        increment(L, extent(l) for l in H)
    }
}
```

图 4：一般张量缩并（情形 3）的实现伪代码。将指标串分解为子集 $\mathcal{P}$、 $\mathcal{F}_A$、……，以及加法、乘法等基本函数的实现未示出。

指标（此处按逆字典序排列）以及确定张量值在内存中的实际位置，分别由 `increment` 和 `offset` 辅助函数处理，如图 5 所示。如上所述，实际实现代码仅略有不同，同时还应用了几项显而易见的优化。此外，参考实现支持对输入张量 A 和 B（图 4 中未显示）进行归约；这会分别为 A 和 B 各增加一个循环，以便在乘法之前累积归约值。最后，通过类型标记和基本辅助函数（乘法、加法、零等）处理多种数据类型及其混合情况；这些函数穷尽地实现了所需的各种组合。

```
    function increment(indices, shape) {
        for (k in 0 to len(shape)-1) {
            indices[k] <- (indices[k] + 1) % shape[k]
            if (indices[k] > 0) return
        }
    }

    function offset(indices, strides) {
        offset = 0
        for (k in 0 to len(strides)-1) {
            offset <- offset + indices[k] * strides[k]
        }
    }
```

图 5：通用张量缩并实现所用辅助函数的伪代码。

4.2 操作覆盖范围

参考实现涵盖了第 2.2 节中所述、截至情况 4 的运算，而外部 TAPP 实现只需支持情形 3，图 4 也展示了情形 3。在参考实现中加入情况 4，是为了面向未来：若今后扩展操作覆盖范围，便可支持更多运算；此外，情况 4 相对容易处理。随着实现不断演进，为了

保证完备性，很可能也会加入情况 5。第 2.2.1 节所述的情形 1，由实现中的三个内层循环覆盖，这与众所周知的矩阵乘法三重循环实现完全对应。在这种情况下，属于张量 D 的一个索引只属于张量 A 和 B 中的一个。遍历张量 D 的空间时，每次只会影响张量 A 和 B 中一个张量内元素的位置。第 2.2.2 节所述的情形 2 引入了三个张量共享的索引。这些索引在算法的外层循环中作为哈达玛积处理，实质上形成一系列相互独立的缩并，每个缩并都属于情形 1 的类型。该算法允许这种处理方式：遍历张量 D 的空间时，同时遍历张量 A 和 B；这只需通过每个张量的步长数组进行编码即可（零表示某个索引不会影响该张量中元素的位置）。如上文简要提到的，第 2.2.3 节所述的情形 3 也通过预处理来处理：将重复指标的步幅相加并合并，从而将其有效地简化为情形 1 和情形 2 中的单个“简单”索引。情形 4 和情形 5 只需增加额外循环，不过如第 2.2.5 节所述，在高性能实现中这样做可能需要无法接受的额外工作空间或浮点运算，因此 TAPP 不要求实现必须支持它们。

4.3 挑战：数据类型

在实现的开发过程中遇到的一个挑战，是如何处理不同的数据类型。在 C 语言中，支持不同数据类型的输入以及混合数据类型颇具挑战，因为除此之外没有其他办法，只能为每组数据类型编写专门的代码。另一个挑战是实现 C 语言不原生支持的数据类型，例如 16 位脑浮点数和 16 位浮点数，这些类型目前仍未实现。

4.4 设计决策

在实现的开发过程中，围绕标准做出了一些设计决策。这些设计决策包括规定运算涵盖的内容，以及允许重复索引和孤立索引。此外，还决定了允许使用哪些类型的张量，包括具有负步幅的张量、零模张量和标量。

4.5 测试

为验证参考实现的正确性，我们开发了一组测试，将其与 TBLIS 进行对比。测试使用半随机张量，其中模态数、大小和值均以随机方式生成，同时确保仍能满足各项测试的目的。

值得一提的测试包括：一个测试负步幅的测试，以及一个测试长度为零的步长的测试。我们特别讨论了这些测试所涉及的特性，即接口是否应限制允许使用的步长类型。使用零模态张量进行测试是一个特殊的边界情况，因为这种张量本质上是标量，不包含任何结构信息。

完整的测试列表请参见附录 B。

5 结论

我们提出了张量缩并操作标准接口的详细草案，弥合了不同应用领域之间的差异，以及以往彼此脱节的软件之间的隔阂。我们定义了通用术语，提供了必要的软件，并成立了决策委员会以维护该标准。该接口已集成到多个高性能实现中，并在应用软件中展示了其用法。

5.1 软件集成

作为概念验证，TAPP 与多个张量库之间的绑定已成功开发。TBLIS 是首个在自身代码库中直接正式支持 TAPP 接口的库。此外，TAPP 代码库中还提供了 cuTENSOR 的绑定，该绑定由独立于供应商的团队开发。最后，基于 TAPP[53] 的泛化版本，为 Cyclops CTF 创建了面向分布式内存张量的实验性绑定。[24]

首个采用 TAPP 的主要软件套件是 DIRAC，[42] 具体而言，其大规模并行耦合簇模块 [43] 已更新为使用 TAPP 作为标准化抽象层，[53] 从而将应用程序与特定的后端依赖解耦，并加入了对分布式内存库的实验性支持。

5.2 展望与未来工作

TAPP 并不旨在提供高级张量代码可能需要的每项功能，而是专注于有效封装张量库所提供的性能关键操作。举例来说，可以在不改变多维数组元素在内存中的存储方式的情况下，对其进行形式上的重塑。我们将此视为一种便利功能，因为它并非性能关键操作。相比之下，如果需要执行会改变数组元素在内存中排列方式的转置，则可能需要调用高性能库来高效地完成此类重排。TAPP 旨在纳入前者，并排除后者。

尽管重点显然在于性能驱动，但目前的工作之后还计划开展两项工作。我们的目标是提供一套全面的随机基准测试，其中将包含来自主要应用领域的代表性问题，以代表我们的张量软件社区。为此，我们正在收集有关张量大小、结构以及缩并类型的数据。

与此相关　　在基准测试集方面，第二项开发称为 Multi-TAPP，是一项特性　　e

它允许使用 TAPP 的开发者在运行时切换张量库。这与近期发布的“BLAS/LAPACK 跳板”概念类似[54,55]，能够无缝地进行基准测试，并动态选择合适的张量库来执行给定任务。

致谢

J. Brandejs 获得了欧洲研究理事会（ERC）根据欧盟地平线 2020研究与创新计划提供的资助（资助协议编号 101019907）。Niklas H¨ornblad 开发参考实现的工作得到了 eSSENCE（瑞典电子科学战略研究计划）的支持。EFV 感谢美国国家科学基金会（奖项编号 2217081）和美国能源部（奖项编号 DE-SC0022327）提供的支持。DAM 感谢美国国家科学基金会（资助编号 CHE-2143725）和美国能源部（资助编号 DE-SC0022893）提供的支持。

附录 A 术语表

批次索引	参见*Hadamard* 索引。
广播	如果输出张量中出现了某个标签，但该标签未出现在任何输入张量中，那么输出数据中的元素将按照该标签的范围重复相应次数。这种一对多的过程称为广播。也称为：复制。

列主序布局 张量在内存中的一种特定跨步布局，其中 n 步长 $\bigcup_{s=1}^{k-1}\{s_k\}$ 由 $s_k = \prod_{l=0}^{n} e_l$ 给出，而 $\bigcup_{s=1}^{k-1}\{e_k\}$ 是张量的形状。

缩并索引其标签仅在输入张量之间共享的索引。也称为：内部指标。

维度 参见范围。

范围索引可表示的不同整数值的数量，记作 d_i 第 i个索引的范围，或记作 extent(i)（对于标签为 i 的索引）。通常采用紧凑范围，例如从 0 开始的索引（ $i \in [0,$extent(i)），或从 1 开始的索引（ $i \in [1,$extent(i)]）。也称为：长度、维度、范围、大小（不过，大小也可以指所有索引范围的乘积，或等价地，张量元素的数量）。外部索引参见自由指标。

自由指标 其标签由输入张量和输出张量共享的索引。也称为：外部索引、未收缩指标。

Hadamard 索引　在三元运算中，标签出现在三个张量 A、B 和 C 中的索引 称为哈达玛索引。仅包含哈达玛索引的运算称为纯哈达玛缩并 或纯哈达玛积 （等同于逐元素乘法）；同时包含哈达玛索引和非哈达玛索引的运算则称为混合哈达玛缩并 或混合哈达玛积。也称为：批处理 或批处理指标。

索引　n 个标签之一，表示整数值范围或某个特定值。所有 n 指标共同指定一个 n阶张量的一个或多个元素。也称为：模式 （n-模式 张量）、维度 （n-维 张量、维数 n等）、秩 （秩-n 张量）、子空间。定义 n时，也称为 n-元 （对于较低的 n，如二元、三元等）、 n-路、 n-重、顺序–n和 n-元 张量。不要与标签混淆。

内部指标 参见缩并索引

标签　赋予张量一个或多个 n 指标的特定符号，表示一个整数值范围或某个特定值。所有 n 指标共同指定一个 n阶张量的一个或多个元素。也称为：索引（也就是说，索引 一词常同时指张量形状 中的元素以及赋予该元素的特定标签）。

布局 参见 形状。

长度 参见范围，形状。

Mode 参见 索引。

n进制， n元 参见索引。

n维 参见索引。

顺序 参见索引。

排列 参见转置。

范围 参见 范围。

Rank 参见 索引。

归约 如果某个标签 出现在一个输入张量中，但不出现在任何其他张量中，则会有多个元素（数量最多可达其范围该张量的标签）将在与另一个输入张量中的元素相乘进行三元运算，或累加到输出张量中进行二元运算。 也称为：求和（尽管归约也可能指除求和以外的二元运算加法）。

复制 参见 广播。

重塑 参见 转置。

行主序布局　　张量在内存中的一种特定跨步布局，其中 n 步长 $\bigcup_{k=1}^{n-1}\{s_k\}$ 由 $s_k = \prod_{l=k+1}^{n} e_l$ 给出，且 $\bigcup_{s=1}^{n}\{e_k\}$ 是张量的形状。

形状　　一个 n-索引张量的范围集合。在抽象用法中，该集合是无序的，反映了指标可能存在的排列。分配给指标的标签可以指代由相应范围定义的范围内的整数值，而整体形状决定了整个张量的数学结构。也称为：大小、布局、长度。

大小参见范围、形状。跨步布局张量在内存中的一种布局，其中具有索引 $\bigcup_{k=1}^{n}\{i_k\}$ 的元素的内存位置 n 由 $base + \sum_{k=0}^{n-1} i_k \cdot s_k$ 给出，和 $\bigcup_{s=1}^{n}\{s_k\}$ 是张量的步长。求和参见复制。转置具有相同标签集合但内存布局不同的两个张量被称为彼此的转置。对于广义列主序或行主序等典型布局，这等价于改变标签的顺序。也称为：重塑、排列。未收缩的-

参见自由指标。

dex

Way 参见索引。

附录 B 测试

TAPP 参考实现的测试集包含以下测试用例。除了测试的顺序外，指标串本身及其维度也会随机化：

1. 哈达玛积 2. 基本二元缩并 3. 交换性：交换 A 和 B 中的张量与指标 4. 检查更改张量 D 的指标顺序会产生不同的排列 5. 所有范围大小均相同的张量之间的缩并 6. 外积，无缩并指标 7. 所有指标均参与缩并的情况 8. 至少一个张量具有零个模态的缩并 9. 至少一个张量具有一个模态的缩并 10. 子张量的缩并，其模态数量与较大的张量相同 11. 子张量的缩并，其模态数量少于较大的张量 12. 使用负步幅的张量之间的缩并 13. 使用负步幅的子张量的缩并，且其数量与

f

模态数少于较大的张量

14. 使用负步幅的子张量进行缩并，且模态数少于较大的张量

15. 使用正、负步幅混合的张量进行缩并 16. 使用正、负步幅混合且模式数量与较大张量相同的张量进行缩并 17. 使用正、负步幅混合且模式数量少于较大张量的张量进行缩并 18. 使用双精度进行缩并 19. 使用复数进行缩并 20. 使用双精度复数进行缩并 21. 使用具有零步幅的张量进行缩并 22. 对具有孤立指标的张量进行缩并和归约 23. 对具有重复指标的张量进行缩并 24. 将哈达玛积与自由指标结合 25. 将哈达玛积与缩并指标结合 26. 测试同一索引的范围不匹配时的错误处理 27. 测试张量 C 与张量 D 不匹配时的错误处理 28. 测试张量 D 内部存在别名时的错误处理

参考文献

(1) Irmler, A.; Gallo, A.; Hummel, F.; Gr¨uneis, A. 环形图与梯形图的对偶性及其对多电子微扰理论的重要性。《物理评论快报》**2019年**, *123*。

(2) Calvin, J. A.; Lewis, C. A.; Valeev, E. F. 用于块秩稀疏矩阵乘法的可扩展基于任务的算法。第 5 届非规则应用：架构与算法研讨会论文集。2015；第 1–8 页。

(3) Kozinsky, B.; Musaelian, A.; Johansson, A.; Batzner, S. 将深度等变模型的领先精度扩展至真实大小的生物分子模拟。国际高性能计算、网络、存储与分析会议论文集。2023年；第 1–12 页。

(4) Kolda, T. G.; Bader, B. W. 张量分解及其应用。《*SIAM*评论》**2009年**, *51*, 455–500。

(5) 国际能源署, 《能源与人工智能》。2025年；保留所有权利。

(6) Psarras, C.; Karlsson, L.; Li, J.; Bientinesi, P. 张量计算软件概览。 `https://arxiv.org/abs/2103.13756`, 2022年。

(7) Brandejs, J.; Bientinesi, P.; Visscher, L.; Gomes, A.; Saue, T. CECAM 张量缩并库标准化研讨会。 `https://tensor.sciencesconf.org/`, 2024年。

(8) Brandejs, J.; Bientinesi, P.; Visscher, L.; Gomes, A.; Saue, T. 第二届图卢兹张量研讨会。 `https://tensor.sciencesconf.org/`, 2025年。

(9) Lawson, C. L.; Hanson, R. J.; Kincaid, D. R.; Krogh, F. T. 面向 Fortran 使用的基本线性代数子程序。*ACM Transactions on Mathematical Software* **1979年**, *5*, 308–323。

(10) 1979年； http://dx.doi.org/10.1137/1.9781611971811.fm。(11) Dongarra, J. J.; Du Croz, J.; Hammarling, S.; Hanson, R. J. 一组扩展的 FORTRAN 基本线性代数子程序。*ACM Transactions on Mathematical Software* **1988年**，*14*，1–17。(12) Demmel, J. LAPACK：面向高性能计算机的可移植线性代数库。*Concurrency: Practice and Experience* **1991年**，*3*，655–666。(13) Dongarra, J. J.; Du Croz, J.; Hammarling, S.; Duff, I. S. 一组三级基础线性代数子程序。*ACM Transactions on Mathematical Software* **1990年**，*16*，1–17。(14) Dongarra, J. J. 超级计算；Springer Berlin Heidelberg，1988年；第456–474页。(15) Gropp, W.; Lusk, E.; Doss, N.; Skjellum, A. 高性能、可移植的消息传递接口标准 MPI 实现。*Parallel Computing* **1996年**，*22*，789–828。(16) Nichols, B.; Buttlar, D.; Farrell, J. P. *POSIX* 线程编程；O'Reilly & Associates, Inc.：美国，1996年。(17) Chandra, R. *OpenMP* 并行编程；Morgan kaufmann，2001年。(18) Dongarra, J.; Gunnels, J.; Bayraktar, H.; Haidar, A.; Ernst, D. 影响科学应用中浮点计算的硬件趋势。2024年； https://arxiv.org/abs/2411.12090。(19) Springer, P.; Bientinesi, P. 高性能类 GEMM 张量乘张量乘法的设计。*ACM* 数学软件汇刊（*TOMS*） **2018年**，*44*，28:1–28:29。

(20) Reynders, J. V. W.; Hinker, P. J.; Cummings, J. C.; Atlas, S. R.; Banerjee, S.; Humphrey, W. F.; Karmesin, S. R.; Keahey, K.; Srikant, M.; Tholburn, M. 使用 *C* 进行并行编程*++*；麻省理工学院出版社，1996年；第 547–588 页。

(21) Landry, W. 高性能张量库的实现。*Scientific Programming***2003**，*11*，273–290。

(22) Hirata, S. 张量收缩引擎：组态相互作用、耦合簇和多体微扰理论的抽象与自动并行实现。*The Journal of Physical Chemistry A* **2003**，*107*，9887–9897。

(23) Di Napoli, E.; Fabregat-Traver, D.; Quintana-Ort´ı, G.; Bientinesi, P. 探索如何高效利用 BLAS 库进行多线性张量收缩。*Applied Mathematicsand Computation* **2014**，*235*，454–468。

(24) Solomonik, E.; Matthews, D.; Hammond, J. R.; Stanton, J. F.; Demmel, J. 面向耦合簇计算的大规模并行张量收缩框架。*Journalof Parallel and Distributed Computing* **2014**，*74*，3176–3190。

(25) Poulson, J.; Marker, B.; van de Geijn, R. A.; Hammond, J. R.; Romero, N. A. Elemental：一种用于分布式内存稠密矩阵计算的新框架。*ACMTransactions on Mathematical Software* **2013**，*39*，1–24。

(26) Maruyama, N.; Essen, B. V.; Dryden, N. J.; Benson, T. R.; Moon, T. Y.; Oyama, Y. DiHydrogen. [计算机软件] `https://doi.org/10.11578/dc.20200106.1`，2020；`https://doi.org/10.11578/dc.20200106.1`。

(27) Poya, R.; Gil, A. J.; Ortigosa, R. 高性能数据并行张量收缩框架：在耦合电-力学中的应用。《计算机物理通讯》**2017**，*216*，35–52。

(28) Li, J.; Battaglino, C.; Perros, I.; Sun, J.; Vuduc, R. 一种用于稠密张量乘矩阵的输入自适应原位方法。国际高性能计算、网络、存储与分析会议论文集。2015；第 1–12 页。

(29) Matthews, D. A. 无需转置的高性能张量缩并。*SIAM*科学计算杂志**2018年**，*40*，C1–C24。

(30) 2019年；https://developer.nvidia.com/cutensor，cuTENSOR，developer.nvidia.com/cutensor。

(31) Harris, C. R.；Millman, K. J.；van der Walt, S. J.；Gommers, R.；Virtanen, P.；Courna- peau, D.；Wieser, E.；Taylor, J.；Berg, S.；Smith, N. J.；Kern, R.；Picus, M.；Hoyer, S.；van Kerkwijk, M. H.；Brett, M.；Haldane, A.；del R´ıo, J. F.；Wiebe, M.；Peterson, P.；G´erard-Marchant, P.；Sheppard, K.；Reddy, T.；Weckesser, W.；Abbasi, H.；Gohlke, C.；Oliphant, T. E. 使用 NumPy 进行数组编程。《自然》**2020**，*585*，357–362。

(32) Abadi, M.; Agarwal, A.; Barham, P.; Brevdo, E.; Chen, Z.; Citro, C.; Corrado, G. S.; Davis, A.; Dean, J.; Devin, M.; Ghemawat, S.; Goodfellow, I.; Harp, A.; Irv- ing, G.; Isard, M.; Jia, Y.; Jozefowicz, R.; Kaiser, L.; Kudlur, M.; Levenberg, J.; Man´e, D.; Monga, R.; Moore, S.; Murray, D.; Olah, C.; Schuster, M.; Shlens, J.; Steiner, B.; Sutskever, I.; Talwar, K.; Tucker, P.; Vanhoucke, V.; Vasudevan, V.; Vi´egas, F.; Vinyals, O.; Warden, P.; Wattenberg, M.; Wicke, M.; Yu, Y.; Zheng, X. TensorFlow：异构系统上的大规模机器学习。2015； https://www.tensorflow.org/，软件可从 tensorflow.org 获取。

(33) Paszke, A.；Gross, S.；Massa, F.；Lerer, A.；Bradbury, J.；Chanan, G.；Killeen, T.；Lin, Z.；Gimelshein, N.；Antiga, L.；Desmaison, A.；K¨opf, A.；Yang, E.；DeVito, Z.；Raison, M.；Tejani, A.；Chilamkurthy, S.；Steiner, B.；Fang, L.；Bai, J.；Chintala, S.神经信息处理系统第*33*届国际会议论文集；Curran Associates公司：美国纽约州雷德胡克，2019年。

(34) Ragan-Kelley, J.; Barnes, C.; Adams, A.; Paris, S.; Durand, F.; Amarasinghe, S. Halide：一种用于优化图像处理流水线中并行性、局部性和重新计算的语言与编译器。*ACM SIGPLAN Notices* **2013**，*48*，519–530。

(35) Kjolstad, F.; Kamil, S.; Chou, S.; Lugato, D.; Amarasinghe, S. 张量代数编译器。*ACM*《编程语言》期刊论文集 **2017**，*1*，1–29。

(36) Chen, T.; Moreau, T.; Jiang, Z.; Zheng, L.; Yan, E.; Cowan, M.; Shen, H.; Wang, L.; Hu, Y.; Ceze, L.; Guestrin, C.; Krishnamurthy, A. TVM：一种用于深度学习的自动化端到端优化编译器。第13届USENIX操作系统设计与实现会议论文集。美国，2018年；第579–594页。

(37) Lattner, C.; Amini, M.; Bondhugula, U.; Cohen, A.; Davis, A.; Pienaar, J.; Riddle, R.; Shpeisman, T.; Vasilache, N.; Zinenko, O. MLIR：面向领域专用计算的可扩展编译器基础设施。2021年IEEE/ACM代码生成与优化国际研讨会（CGO）。2021年；第2–14页。

(38) Fishman, M.; White, S.; Stoudenmire, E. 用于张量网络计算的ITensor软件库。*SciPost*物理代码库 **2022年**，

(39) Li, M.; Liu, Y.; Liu, X.; Sun, Q.; You, X.; Yang, H.; Luan, Z.; Gan, L.; Yang, G.; Qian, D. 深度学习编译器：全面综述。*IEEE*《并行与分布式系统汇刊》**2021年**，*32*，708–727。

(40) Zheng, L.; 等。优秀张量编译器。 `https://github.com/merrymercy/awesome-tensor-compilers`，2024年。

(41) H¨ornblad, N.; Brandejs, J.; Valeev, E. F.; Heinecke, A.; Hammond, J.; Millette, C.; Matthews, D.; Bientinesi, P. TAPP。 `https://github.com/TAPPorg/reference-implementation`，2025年。

(42) Visscher, L.; Jensen, H. J. Aa.; Bast, R.; Gomes, A. S. P.; Saue, T.; Aucar, I. A.; Bakken, V.; Brandejs, J.; Chibueze, C.; Creutzberg, J.; Dyall, K. G.; Dubillard, S.; Ekstr¨om, U.; Eliav, E.; Enevoldsen, T.; Faßhauer, E.; Fleig, T.; Fossgaard, O.; Halbert, L.; Hedeg°ard, E. D.; Helgaker, T.; Helmich-Paris, B.; Henriksson, J.; van

Horn, M.; Iliaˇs, M.; Jacob, C. R.; Knecht, S.; Komorovsk´y, S.; Kullie, O.; Lærdahl, J. K.; Larsen, C. V.; Lee, Y. S.; List, N. H.; Nataraj, H. S.; Nayak, M. K.; Norman, P.; Nyvang, A.; Olejniczak, M.; Olsen, J.; Olsen., J. M. H.; Papadopoulos, A.; Park, Y. C.; Pedersen, J. K.; Pernpointner, M.; Pototschnig, J. V.; Di Remi- gio Eik˚as, R.; Repisk´y, M.; Ruud, K.; Salek, P.; Schimmelpfennig, B.; Senjean, B.; Shee, A.; Sikkema, J.; Sunaga, A.; Thyssen, J.; van Stralen, J.; Vidal, M. L.; Vil- laume, S.; Visser, O.; Winther, T.; Yamamoto, S.; Yuan, X. DIRAC24. https://zenodo.org/doi/10.5281/zenodo.10680560, 2024年.

(43) Pototschnig, J. V.; Papadopoulos, A.; Lyakh, D. I.; Repisky, M.; Halbert, L.; Gomes, A. S. P.; Jensen, H. J. Aa.; Visscher, L. 面向大规模并行 GPU 加速计算架构的相对论耦合簇理论实现。《化学理论与计算杂志》**2021**, *17*, 5509–5529.

(44) Bezanson, J.; Edelman, A.; Karpinski, S.; Shah, V. B. Julia：一种数值计算的新方法。《*SIAM*评论》**2017**, *59*, 65–98.

(45) Inc., T. M. M ATLAB 版本：9.13.0 (R2022b)。2022年； https://www.mathworks.com。

(46) Nakatani, N.; Verstichel, B.; Chan, G.; Calvin, J.; 等。基础张量代数子程序（BTAS）。 https://github.com/ValeevGroup/BTAS, 2016.

(47) Shi, Y.; Niranjan, U. N.; Anandkumar, A.; Cecka, C. 使用扩展 BLAS 内核在 CPU 和 GPU 上进行张量缩并。2016 年第23届 IEEE 高性能计算国际会议（HiPC）。2016年。

(48) Bientinesi, P.; Ham, D.; Huang, F.; Kelly, P. H. J.; Sadayappan, P. S.; Stow, E. 张量计算：应用与优化（Dagstuhl研讨会22101）。**2022年**,

(49) Valeev, E. TiledArray 基础编程。 https://valeevgroup.github.io/tiledarray/dox-master/_basic-_programming.html, [在线；访问日期：2026]年1月9日。

(50) cppreference.com 社区 C++ reference MDSPAN。 https://en.cppreference.com/w/cpp/container/mdspan, [在线；访问日期：2023]年7月13日。

(51) Murray, R.; Demmel, J.; Mahoney, M. W.; Erichson, N. B.; Melnichenko, M.; Ma- lik, O. A.; Grigori, L.; Luszczek, P.; Derezi´nski, M.; Lopes, M. E.; Liang, T.; Luo, H.; Dongarra, J. 随机数值线性代数：对该领域的展望，并着眼于软件。2023年; https://arxiv.org/abs/2302.11474。

(52) Ibrahim, K. Z.; Williams, S. W.; Epifanovsky, E.; Krylov, A. I. 多核架构上 libtensor 框架的分析与调优。2014年 第21届国际高性能计算会议 (HiPC)。2014年; 第 1–10 页。

(53) Brandejs, J.; Pototschnig, J.; Saue, T. 为现代分布式内存张量软件生成耦合簇代码。《化学理论与计算杂志》**2025年**, *21*, 7320–7334。

(54) K¨ohler, M.; Saak, J. FlexiBLAS——一个支持运行时切换后端的灵活 BLAS 库。**2013**,

(55) Saba, E.; Shah, V. B. libblastrampoline。 https://github.com/JuliaLinearAlgebra/libblastrampoline, 2025年。