https://crad.ict.ac.cn
63(2): 419−433, 2026

# 面向国产多核 DSP 的张量转置并行优化技术

刘根程[1,2,3] 王庆林[1,2,3] 洪楚河[1,2,3] 彭 兴[1,2,3] 夏 睿[1,2,3] 梁亚玲[1,2,3] 张庆阳[1,2,3] 车永刚[1] 刘 杰[1,2,3]

[1]（国防科技大学计算机学院 长沙 410073）

[2]（高端装备数字化软件湖南省重点实验室（国防科技大学） 长沙 410073）

[3]（并行与分布计算全国重点实验室（国防科技大学） 长沙 410073）

（liugencheng@nudt.edu.cn）

# Parallel Optimization Technology of Tensor Transposition for Domestic Multi-Core DSPs

Liu Gencheng[1,2,3], Wang Qinglin[1,2,3], Hong Chuhe[1,2,3], Peng Xing[1,2,3], Xia Rui[1,2,3], Liang Yaling[1,2,3], Zhang Qingyang[1,2,3], Che Yonggang[1], and Liu Jie[1,2,3]

[1]（*College of Computer Science and Technology*, *National University of Defense Technology*, *Changsha* 410073）

[2]（*Laboratory of Digitizing Software for Frontier Equipment* (*National University of Defense Technology*), *Changsha* 410073）

[3]（*National Key Laboratory of Parallel and Distributed Computing* (*National University of Defense Technology*), *Changsha* 410073）

**Abstract** Tensor transposition is a fundamental computational primitive extensively employed in signal processing, scientific computing, and deep learning applications. With the growing emphasis on energy efficiency in high-performance computing, digital signal processors (DSPs) have emerged as promising accelerators. However, existing tensor transposition libraries designed for conventional CPU/GPU architectures fail to fully exploit the architectural advantages of modern DSP platforms, particularly their specialized memory hierarchies and vector processing capabilities. To address the architectural characteristics of domestic DSP platforms, we proposes ftmTT, a high-performance tensor transposition algorithm specifically optimized for multi-core DSP architectures. The key technical contributions include: 1) An intelligent tiling strategy that systematically decomposes high-dimensional tensor operations into efficient matrix transposition kernels native to DSP hardware; 2) A novel DMA-based data access optimization scheme that significantly reduces memory transfer overhead through smart data block consolidation; 3) A double-buffering design that asynchronously overlaps transposition computation with DMA transfers, enabling computation-communication overlap and ultimately achieving high-performance parallel tensor transposition on DSP platforms. Experiments on the domestic multi-core DSP platform FT-M7032 demonstrate that ftmTT tensor transposition algorithm achieves up to 75.96% of the theoretical bandwidth and 99.23% of the experimental platform's STREAM bandwidth.

**Key words** tensor transposition; DMA transfer; multi-core DSP; vectorization; memory access merging

**摘 要** 张量转置（tensor transposition）作为基础张量运算原语，广泛应用于信号处理、科学计算以及深度学习等各种领域，在张量数据密集型应用及高性能计算中具有重要作用。随着能效指标在高性能计算系统中的重要性日益凸显，基于数字信号处理器（digital signal processors, DSPs）的加速器已被集成至通用

**收稿日期**：2025-03-01；**修回日期**：2025-08-27

**基金项目**：国家重点研发计划项目 (2023YFA1011704, 2021YFB0300101, 2022YFB4501702)

This work was supported by the National Key Research and Development Program of China (2023YFA1011704, 2021YFB0300101, 2022YFB4501702).

**通信作者**：王庆林（wangqinglin@nudt.edu.cn）

计算系统。然而,传统面向多核 CPU 和 GPU 的张量转置库因架构差异无法充分适配 DSP 架构。一方面,DSP 架构的向量化计算潜力尚未得到充分挖掘;另一方面,其复杂的片上存储体系与多层次共享内存结构为张量并行程序设计带来了显著挑战。针对国产多核 DSP 的架构特点,提出 ftmTT 算法,并设计实现了一个面向多核 DSP 架构的通用张量转置库。ftmTT 算法通过设计适配 DSP 架构的高效内存访问模式充分挖掘其并行化和向量化潜力,其核心创新包括:1)采用分块策略将高维张量转置转化为多核 DSP 平台所提供的矩阵转置内核操作;2)提出基于 DMA 点对点传输的张量数据块访存合并方案来降低数据搬运开销;3)通过双缓冲设计异步重叠转置计算与 DMA 传输实现计算通信隐藏,最终面向多核 DSP 实现高性能并行张量转置。在国产多核 DSP 平台 FT-M7032 的实验表明,ftmTT 张量转置算法取得了最高达理论带宽 75.96% 的性能,达到 FT-M7032 平台 STREAM 带宽 99.23% 的性能。

**关键词** 张量转置;DMA 传输;多核 DSP;向量化;访存合并

**中图法分类号** TP311.13; TP309

**DOI:** 10.7544/issn1000-1239.202550130 **CSTR:** 32373.14.issn1000-1239.202550130

张量转置是一种关键的张量操作原语,用于数据布局的转换,在机器学习、深度学习[1]、量子化学计算[2]和多维傅里叶变换等以张量为核心的领域中有着广泛的应用。特别是在使用转置-转置-GEMM-转置(TTGT)[3-5]方法进行张量收缩时,张量转置往往成为性能瓶颈。

随着高性能计算系统对能效要求的不断提升,具有高计算密度特性的数字信号处理器(digital signal processor, DSP)正逐渐成为该领域的重要计算单元[6-7]。然而,现有研究表明,当前张量转置优化研究存在 2 个主要局限:其一,研究多集中于低阶张量转置[8-13];其二,优化方法主要面向 CPU/GPU 平台[14-18],而针对多核 DSP 架构的专用优化方法较为匮乏。这种研究空白源于多核 DSP 与 CPU/GPU 在架构上的本质差异,即 DSP 采用超长指令字架构,虽提升了指令级并行性,却以牺牲传统缓存结构为代价,这使得面向多核 DSP 的算法设计依赖显式的片上内存管理策略。据目前已有研究,现有面向 CPU/GPU 平台的张量转置优化方法普遍采用缓存友好型设计,这种设计思路难以满足多核 DSP 架构的特殊需求。因此,设计一种针对多核 DSP 架构优化的张量转置算法具有重要的研究价值和实际意义。

面向 DSP 架构的高效张量转置算法设计需要克服其体系结构特性造成的存储管理和计算资源利用两方面的技术挑战。在存储管理方面,DSP 采用软件显式控制的存储子系统架构设计,此设计允许通过精细化的数据布局规划提升计算数据局部性,但是使得传统基于循环顺序优化的算法范式(如 CPU/GPU 平台广泛采用的循环分块技术)的性能优化效果受到制约[14]。因此,要求算法设计必须从根本上重构数据访问模式,而非简单移植现有平台的优化策略。在计算资源利用方面,DSP 架构虽然集成了指令级并行(ILP)、专用向量化计算单元和大容量片上存储等有利于数据密集型计算的硬件特性[19-20],但其多层次存储体系(包含寄存器文件、共享存储等不同层级)要求算法必须精确设计数据在存储层次间的分布策略与访存序列,以实现计算流水线与存储带宽的最优协同。特别值得注意的是,这种显式存储管理与异构并行机制的组合特性,使得传统依赖硬件缓存(如 CPU/GPU 平台)的隐式数据传输方案难以有效适配 DSP 架构[6-7,21-22]。其中,DSP 架构中计算单元依赖 DMA 引擎数据传输的工作机制,要求算法创新性地设计数据搬运与计算任务的耦合机制,以充分适配 DMA 引擎的传输特性。

针对上述挑战,本文提出了一种面向多核 DSP 的高效张量转置算法 ftmTT,并在国产 FT-M7032 平台上实现了完整的算法库。该算法采用高维抽取维度到 2 维的降维策略,通过将高维张量分解为 2 维张量切片,充分利用 DSP 平台上优化的向量化矩阵转置内核来处理高阶张量转置问题。为优化内存访问效率,算法创新性地设计了内存合并访问方案,该方案不仅实现了最优的内存访问模式,还被抽象为通用范式以保证在不同 DSP 平台间的可移植性。进一步地,通过引入双缓冲任务调度机制,实现了片上/片外数据传输与转置内核计算操作的高效并行。实验评估表明,在 FT-M7032 平台上,相较于与其 DSP 簇共享主存的 16 个 CPU 核心上实现的 HPTT[16]和 TTC[15]算法结果,访问峰值性能一致的情况下,ftmTT 算法仅使用 8 个 DSP 核心即实现了显著性能提升:平均加速比分别达到 2.04 倍和 1.92 倍,最大加速比分别提

升至 4.29 倍和 7.11 倍。在带宽利用效率方面，ftmTT 峰值带宽利用率达 75.96%，平均利用率保持在 59.17%。

本文主要贡献有 3 方面：

1）面向多核 DSP 架构实现高效张量转置库 ftmTT，针对其显式内存管理特性与高频 DMA 数据交换场景进行张量数据访存优化。实验表明该库显著优于传统方案，峰值带宽达到实验平台 FT-M7032 STREAM 带宽的 99.23%。

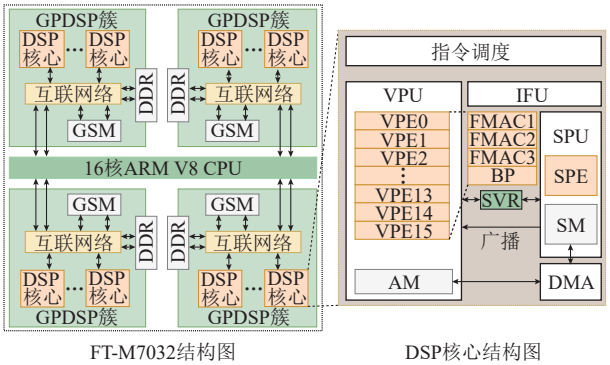
2）提出了一种基于 DSP 向量化矩阵转置的任意维度张量转置方法。通过对高效矩阵转置内核适配，充分发挥 DSP 的向量化矩阵转置能力，显著提升高维张量转置性能。

3）提出了一种针对 DSP 架构下张量数据的内存访问合并方案。结合 DSP 架构以 DMA 传输为主的数据搬运模式，设计 DMA 传输过程主存访问合并算法，有效解决了张量数据切片后产生的高度碎片化内存访问的问题，最小化张量数据访问次数并降低数据传输启动开销，结合双缓冲算法进一步提高算法效率。

## 1 研究背景与研究动机

### 1.1 DSP 结构

FT-M7032 平台是国防科技大学面向通用计算自主研制的一款异构处理器[23-26]，集成了一个 16 核 ARMv8 CPU 和 4 个 GPDSP 簇，如图 1 所示。其多核 CPU 是飞腾 FT2000plus 处理器的精简版本[27-29]，主要负责进程级控制和组件间通信。每个 GPDSP 簇包含 8 个 DSP 核心，共享 6 MB 的片上全局共享内存（GSM）。集群内 DSP 核心之间的通信通过片上交叉网络实现，需构建软件级数据一致性机制来维护数据一致性。多核 CPU 与 GPDSP 簇共享统一的主存空间，其中 CPU 可以访问整个内存空间，而每个 GPDSP 簇仅能访问其指定的内存分区，带宽为 42.6 GBps。




FT-M7032结构图　　DSP核心结构图

Fig. 1 The diagram of FT-M7032 architecture and the micro-architecture of a DSP core

图 1 FT-M7032 架构及 DSP 核心微结构图

GPDSP 簇中的每个 DSP 核心均采用超长指令字（VLIW）架构，包含指令分发单元（IFU）、标量处理单元（SPU）、向量处理单元（VPU）以及 DMA 引擎等多个组件，如图 1 所示。IFU 每个时钟周期最多可分发 11 条指令，其中包括 5 条标量指令和 6 条向量指令。SPU 负责控制流和标量运算，配备 1 个标量处理单元（SPE）和 64 KB 的标量存储空间（SM），支持同时执行 5 条标量指令。VPU 作为主要计算资源，配备了 768 KB 的向量存储空间（AM）和 16 个以 SIMD 模式运行的向量处理单元（VPE）。每个 VPE 包含 64 个位宽为 64 b 的寄存器以及 3 个融合乘加（FMAC）单元，其中 1 个单元每个周期可执行 1 次 FP64 乘加运算。因此，FP64 操作的 SIMD 宽度为 16。AM 通过 2 个 Load/Store 向量单元，每个周期最多可向寄存器传输 512 B 的数据。SPU 与 VPU 之间的数据交换通过广播指令和共享寄存器实现。DMA 引擎负责管理包括主存、GSM、SM 和 AM 在内的内存层次结构之间的数据传输。

### 1.2 张量转置

张量转置是指对张量索引顺序的重新排列，其本质是对张量元素在内存中位置的重新排序：

$$\boldsymbol{A}_{i_0,i_1,\cdots,i_{N-1}} \rightarrow \boldsymbol{B}_{Dim(i_0,i_1,\cdots,i_{N-1})}, \tag{1}$$

其中 $\boldsymbol{A}$ 表示原始张量，$\boldsymbol{B}$ 表示转置后的张量。函数 $Dim(i_0,i_1,\cdots,i_{N-1})$ 定义了目标张量的新索引顺序，而 $i_0,i_1,\cdots,i_{N-1}$ 则代表原始张量的索引。这一过程通过重新排列张量的维度顺序实现了数据布局的转换，如算法 1 所示。

**算法 1.** 朴素张量转置。

输入：张量 $\boldsymbol{A}$，形状 $(i_0,i_1,\cdots,i_{N-1})$；目标张量 $\boldsymbol{B}$，形状 $Dim(i_0,i_1,\cdots,i_{N-1})$；

输出：张量 $\boldsymbol{B}$。

① for each element $A[i_0][i_1]\cdots[i_{N-1}]$ in $\boldsymbol{A}$ do

② $target \leftarrow$ 计算转置坐标$(i_0,i_1,\cdots,i_{N-1})$；

③ $B[target] \leftarrow A[i_0][i_1]\cdots[i_{N-1}]$；

④ end for

⑤ return $\boldsymbol{B}$。

张量转置的朴素实现通常需要多层嵌套循环，其中输入张量的元素被读取并写入到输出张量的相应索引位置。例如，将一个 3 维张量 $\boldsymbol{A}_{k,j,i}$ 转置为 $\boldsymbol{B}_{i,j,k}$ 可以通过以下循环实现：for$(i,j,k)B[i][j][k] = A[k][j][i]$ 在这种循环顺序下的最内层维度为 $k$，张量 $\boldsymbol{A}$ 的内存访问是高跨步的，而张量 $\boldsymbol{B}$ 的写入过程则是连续且

低跨步的。这种访问模式的不平衡会导致内存访问性能下降,从而对高性能张量转置程序的实现提出了挑战,优化内存访问模式是提升张量转置性能的关键[14]。

在本文工作中,张量以行优先格式存储于内存中。以$\boldsymbol{A}_{i_0,i_1,\cdots,i_{N-1}}$为例,索引$i_{N-1}$在内存中的跨步为1,对于$N$=2的情况,对应于矩阵的行方向,且将其称为最内层索引。本文张量转置算法将张量划分为张量切片,具体做法为按照一定的跨步对最内层索引指向的维度进行元素连续抽取,每次抽取都形成一个矩阵行,多次抽取得到的矩阵行形成矩阵。此时矩阵的行方向为最内层索引对应维度,列方向则为自由选择的维度。若以原始输入张量的最内层索引对应维度作为抽取元素行的维度,并将自由选择的维度确定为输出张量的最内层索引对应维度,在转置前的原始张量中该维度中相邻元素的内存跨步作为抽取元素行的跨步,形成的切片称为最大转置切片[14]。以2维张量为例,增加任意一行中最后1个元素的地址索引会移动到另一行,超出最内层索引大小限制后,将会导致相应元素不再位于同一连续行中。而转置过程的传输和向量化步骤十分依赖存储位置的连续性,因此,可以说抽取的张量切片达到了张量最内层索引对应维度大小限制下的上限。这一切片限制确保了在转置过程中,访问矩阵行方向元素时不会出现在原始张量中不在同一行的情况,这样可以保持访问模式一致性。

本文将以位数为64 b的双精度数据操作作为主要研究对象。除非特别说明,后续所有描述均针对双精度数据。这一选择基于双精度数据在高性能计算中的广泛应用及其对计算精度和性能的严格要求。

### 1.3 张量转置应用

张量转置在深度学习与神经网络中通过调整数据维度顺序,实现框架兼容性(如PyTorch的NCHW与TensorFlow的NHWC的格式转换)[30-31]、优化矩阵运算(如全连接层的权重转置和Transformer注意力机制中的键矩阵转置[32]),提升计算效率与内存访问连续性,同时在反向传播中确保梯度维度对齐[33],支持多模态数据处理(如视频时序调整)及模型优化(如计算图融合),是连接数据、算法与硬件性能的关键操作。

在张量收缩(tensor contraction)中,张量转置通过调整维度顺序优化缩并路径并降低计算复杂度。例如,对于2个3维张量$\boldsymbol{A}_{i,j,k}$和$\boldsymbol{B}_{l,m,n}$,若需缩并指标$j$和$l$,通过转置将$\boldsymbol{A}$的维度重排为$(k,i,j)$,$\boldsymbol{B}$的维度重排为$(l,m,n)$,可使得共享维度$j$和$l$对齐,利用高效矩阵乘法核(如BLAS)加速计算。在量子电路模拟或张量网络优化中[34],转置还能减少中间张量的内存占用,例如通过调整MPS(矩阵乘积态)中张量的物理指标顺序,避免冗余缩并操作[35]。

### 1.4 多核DSP矩阵转置

在DSP微内核架构中,由于功耗限制,不支持直接硬件转置,因此需要通过软件优化实现矩阵转置内核[36]。文献[36]提供的面向多核DSP的矩阵转置实现方式对本文多核DSP张量转置算法实现有所限制,因而在此着重介绍其具体实现。

得益于多核DSP平台VPU部件的Load/Store单元拥有支持模数存储指令,特别是VSTDW0M16和VSTDW1M16这2个指令。对于双精度数据,采用上述模16指令进行数据读取时,能够快速完成4×16矩阵的转置操作,过程如图2所示。

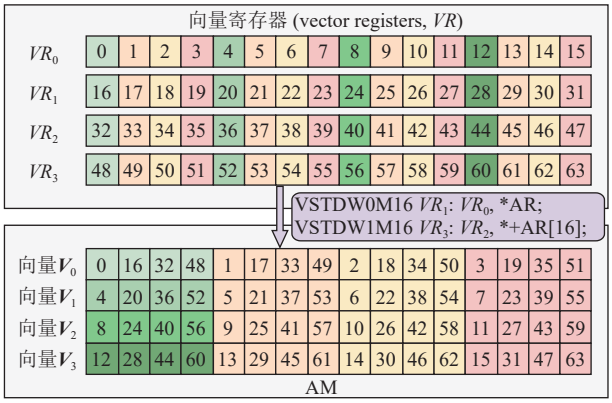



Fig. 2 4×16 matrix transposition core

图2 4×16矩阵转置核心

上述基于模16指令所实现的4×16转置方法核心流程是从一组向量中以16为步长循环读取,后将其存储为连续的向量元素。如图2所示过程,存储在向量$\boldsymbol{V}_0$中起始元素为0, 16, 32, 48。为了实现16×16的转置,朴素的方法是连续执行多次4×16转置操作。然而,使用VSTDW0M16和VSTDW1M16指令执行的转置操作均在向量单元上进行,且转置矩阵的多行数据会连续存储在同一向量中,如图2中$\boldsymbol{V}_0$中存储了4个长度为4的矩阵行,导致多行数据绑定到单个向量。这种存储方式不利于后续对向量行的操作。为此,使用了一种分组扩展方法,将4×16核心扩展为16×16核心,确保向量中的每个元素均来自矩阵的同一行,此条件对后续张量转置算法的数据搬运连续性有所限制。

如图3所示,DSP矩阵转置算法将16个向量以

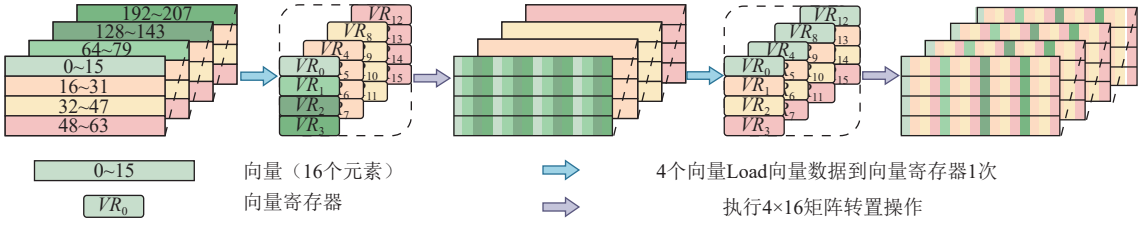



Fig. 3 Workflow of 16×16 matrix transposition kernel

图 3 16×16 矩阵转置核心工作流程

模 4 间隔分组，并将其存储在向量寄存器中。例如，实际存储在 $VR_0$~$VR_3$ 向量寄存器中的数据来自矩阵的第 0, 4, 8 和 12 行。每 4 个连续编号的寄存器存储 1 组向量，用于执行 VSTDW0M16 和 VSTDW1M16 指令，完成第 1 次重排操作。这一系列操作实际上将行方向元素的步长增加了 4 倍，第 2 次重排执行时又进一步增加 4 倍。因此，原始行方向元素的步长被扩展了 16 倍。对于 16×16 矩阵，同一列中相邻元素之间的步长恰好为 16，经过这一系列操作后，原本在同一行的元素将变为同一列中的元素，从而实现了向量化的 16×16 的 2 维转置。

当使用多核架构时，可将矩阵进行分块，将块分配到不同的 DSP 核心中进行并行处理，在 DSP 内核中的矩阵分块数据仍然可以继续分块以适应计算微内核 1 次处理的数据量限制，通过这种分块分批处理的方式，可以实现负载均衡的 DSP 平台矩阵转置算法，整体算法如算法 2 所示。

**算法 2.** 分层矩阵转置算法。

输入：形为 $M\times N$ 矩阵 $\boldsymbol{A}$，参数 $W_b$，$H_b$，$w_b$，$h_b$；

输出：形状为 $N\times M$ 的输出矩阵 $\boldsymbol{B}$。

① 矩阵分块为多个 $W_b\times H_b$ 大小矩阵块 $\boldsymbol{A}_{W_b,H_b}$；

② for each thread in parallel do

③ for each $\boldsymbol{A}_{w_b,h_b}$ of size $w_b\times h_b$ in $\boldsymbol{A}_{W_b,H_b}$ do

④ DMA-P2P: 传输 $\boldsymbol{A}_{w_b,h_b}$ 到 $AM_{in}$ 空间；

⑤ for each micro-block $\boldsymbol{A}_{16}$ of size 16×16 in $\boldsymbol{A}_{w_b,h_b}$ do

⑥ 加载 $\boldsymbol{A}_{16}$ 至向量寄存器；

⑦ 执行转置核心实现 $\boldsymbol{A}_{16}$ 转置；

⑧ 写回 $\boldsymbol{A}_{16}$ 到 $AM_{out}$ 地址；

⑨ end for

⑩ DMA- P2P: 传输 $\boldsymbol{A}_{w_b,h_b}$ 到 $\boldsymbol{B}$ 地址；

⑪ end for

⑫ end for

⑬ return $\boldsymbol{B}$。

实际上对于本文实验平台，其 VPU 部件中向量寄存器有 64 个，ftmTT 算法在具体实现时通过精确流水排布和循环操作，可以实现 32×32 形状的矩阵转置内核。至此，在本文中实现了适配任意阶数张量和任意转置模式的向量化矩阵转置内核，在算法 2 中承担关键作用。

### 1.5 研究动机

设计张量转置算法时本研究关注了以下 3 方面。

1）DSP 架构向量化潜力。得益于其特有的硬件设计：一方面，AM 与 VPU 为 2 维转置的向量化实现提供了硬件基础；另一方面，其片上内存带宽显著高于片外主存带宽的特性，加之片上 Load/Store 单元支持非阻塞访问机制，使得转置过程中的计算与访存操作能够并行执行，避免与片外访问产生资源争用。这种架构特性决定了关键优化路径：通过将高维张量转置转化为矩阵转置，充分挖掘 VPU 向量化能力以提升矩阵转置效率，设计高效的内存访问模式降低数据搬运开销。DSP 平台上已有一些高效的矩阵转置（2 维转置）工作[36]，因此，本文把如何设计高效的内存访问模式以适配向量化的矩阵转置内核作为高维张量转置性能提升的重要挑战。利用矩阵转置内核可以将矩阵转置计算过程与张量数据搬运解耦。适配高效向量化矩阵转置内核条件下，张量转置算法优化目标可聚焦于数据搬运效率，减少算法设计复杂程度。

2）大量的计算启动开销。参考 BLAS 库的算子内核化，本文将高维张量分解为大量 2 维切片，以充分利用优化的矩阵转置内核。然而，执行 1 次 2 维张量转置需单独启动 1 次 DSP 物理核心，大规模高维张量转置将产生显著的开销，主要体现在 DSP 核心与 CPU 通信交互开销。经典的 CPU 平台张量转置库 HPTT 在 CPU 上进行张量切片后需立即启动矩阵转置内核，此时如果直接调用 DSP 平台矩阵转置内核，整个程序将产生大量的转置内核启动开销，这使得 HPTT 算法不适合 DSP 架构。频繁的内核启动带来的额外开销将对 DSP 上负载平衡和数据布局优化所获得的收益进行稀释。因此，需要设计一种更适合 DSP 架构的转置策略，以减少内核启动开销并充

分利用硬件资源。这一优化方向对于提升高维张量转置性能至关重要。通过减少内核启动频率并优化数据局部性,可以显著提高整体计算效率。

3)软件管理的片上内存空间。与CPU基于硬件缓存的内存访问机制不同,多核DSP架构通常采用DMA引擎实现片外数据访问,这一特性使得传统面向缓存优化的优化技术(如张量索引循环变换等)难以直接适用。在软件显式管理片上内存的架构约束下,优化策略需重点关注以下2个核心方向:通过数据预取技术最大化片上内存空间利用率;最小化主存访问频次。值得注意的是,这一优化范式的转变与GPU平台的张量转置优化存在相似性,均从传统的缓存利用率优化转向共享片上内存的访问优化。然而,DSP平台的特殊性体现在:其一,受限于片内外带宽瓶颈及DMA传输模式,必须对主存访问操作进行深度合并优化;其二,不同于GPU通过大规模线程并行实现内存访问合并,DSP平台的线程资源有限且需严格绑定物理计算核心。基于以上分析,本文提出主存访问合并方案,主要依赖提前将多个线程所需张量数据放入DSP核心共享的片上内存中,增加单次DMA传输数据量,降低DMA数据传输频次,同时可结合高维张量数据布局优化提高DSP核心执行2维转置的效率。

## 2 算法设计

### 2.1 算法总览

ftmTT算法的核心创新在于协同优化了向量化转置计算与DMA数据访问2个关键环节:通过精心设计的高维张量转置方案,将DSP平台的向量化矩阵转置内核性能发挥到极致,显著提升了片上内存中的2维转置效率;同时针对DSP架构使用的DMA引擎访问模式,提出了创新的张量数据访存合并方案,有效降低了片内外数据搬运开销。这种“计算优化+访存优化”的双重设计策略,使得ftmTT能够充分利用DSP的向量化计算能力,同时最小化数据移动成本,从而在整体上实现了显著的性能提升。实验结果表明,该算法在典型张量转置任务中展现出优越的执行效率。

ftmTT主算法的工作流程如下:实验平台集成的CPU部分首先使用访存合并算法确定合并策略。随后,DMA引擎根据合并策略中的读取计划将数据搬运至DSP片上存储,将张量切片以数据块传输的方式加载到AM中。高效的矩阵转置内核负责在AM上执行2维转置操作,最终,合并策略中的写入计划指导DMA进行点对点传输,将张量写回主存,从而完成整个张量转置过程。

### 2.2 张量转置DSP平台一般实现

在高维张量转置过程中,若某些连续索引在转置前后保持不变,则可以在逻辑上将其视为单一索引。例如,给定一个张量转置$(a,b,c,d)\Rightarrow(d,c,a,b)$,得益于转置模式的统一性,维度$a$和$b$可以合并,将4维度张量转置转化为3维度$(A,c,d)\Rightarrow(d,c,A)$。因此,在后续讨论中,本文仅关注无法进一步合并的简化张量转置形式。

对于形状为$(a,b,c,d)$的张量,假设其在系统内存中按行优先方式存储,当沿索引$d$方向向以步长为1读取张量元素时,$d$被称为最内层索引。在本文的后续讨论中,均假设张量以此格式存储和定义。

通过区分张量转置前后最内层索引是否改变可以将转置模式分为2种,其中最内层索引不变的模式称为主轴不变转置,在2.5节中进行设计说明。

如图4所示展开过程,对于最内层索引转置前后发生变化的情况,算法通过将张量沿其最内层索引$j$和转置后输出张量最内层索引对应的索引$k$方向展开为矩阵。这一过程将张量展开为多个矩阵,随后对每个矩阵进行转置。由于DSP平台的2维转置内核需要AM的支持,我们使用DMA点对点传输将数据写入AM,在AM中完成2维转置操作,随后将转置结果通过DMA引擎写回主存。

图4过程中,以3维张量转置过程$(k,i,j)\Rightarrow(j,i,k)$为例,其维度大小为4×4×4。首先,根据输入张量和输出张量的最内层索引$j$和$k$,对输入张量进行切片,得到$i$个$k\times j$的矩阵。这种切片方式正是前文中定义的最大转置切片。通过执行$i$次2维转置操作实现所有切片的$k$和$j$换位,并注意写回时每个矩阵行的跨步变化,$\boldsymbol{S}_0$在主存中的跨步应该对应转置后$j$维度的跨步大小。利用这种方式可以实现所需的3维张量转置,且可以推广到任意阶张量转置。

### 2.3 冲突检测的访存合并算法

在图4所示的张量转置方案中,每次2维转置操作处理一个维度为$k\times j$的矩阵切片。当该矩阵切片中的元素总数较小时,存在多个因素影响算法的效率。首先,切片尺寸小于矩阵转置核心的处理能力时,由于其向量化处理方式,向量长度不足时,转置核心仍然需要执行同样周期,这意味着每个时钟周期转置核心处理的数据量未能达到最大化。其次,每个切片需要2次DMA引擎启动:一次是从主内存

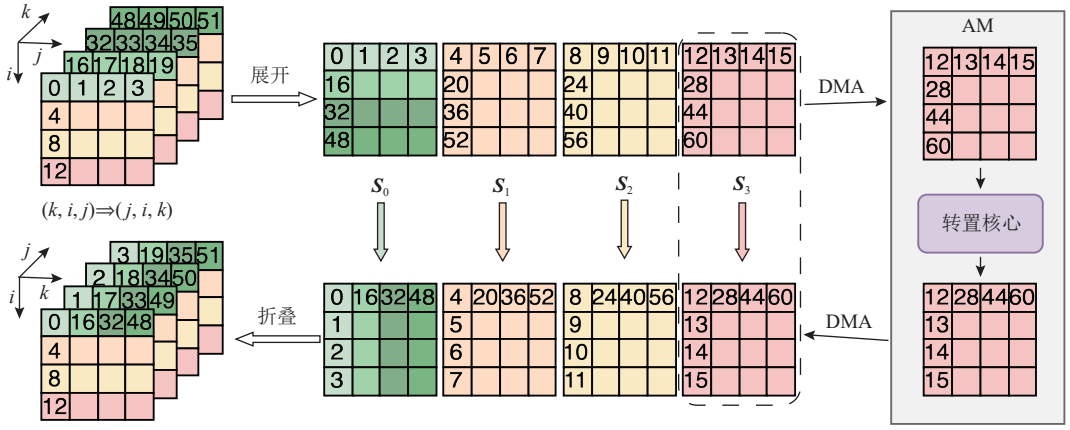



Fig. 4 Tensor transposition process

图 4 张量转置流程

加载到 AM，另一次是从 AM 存储回主内存。如果切片尺寸过小，数据量一定的情况下，启动次数增多，启动 DMA 引擎时间的占比会显著增加，从而降低带宽利用率。

针对这一问题，本文分析了张量元素的分布特性，并结合 DSP 微内核架构分层存储结构提出了一种基于向量内存空间的合并方案。通过合并张量切片访存操作，增大 1 次数据搬运过程的数据量，并增大转置核心可操作的切片大小，最理想状况下每次转置核心运行时都处理最大尺寸切片。访存合并后，能够提升核心计算的饱和程度，并减少核心启动开销，从多方面提升传输效率。

本研究的访存合并方案基于对转置前后矩阵切片位置映射关系的深入分析，其核心设计思想是通过建立输出张量块与输入张量矩阵块的逆向映射关系，推导出主存中连续张量切片的最大合并边界。该方案具有以下关键技术特征：首先，通过分析转置过程中数据块的局部性特征，实现了输出-输入张量块分布的精确逆向映射，从而确定了理论上的最大合并限度；其次，在具体实现时需根据 AM 存储容量限制，在数据读取和写回阶段智能平衡合并程度；最后，利用转置模式下各张量切片地址偏移的固有规律性，将合并方案设计为一次性预计算模式，即通过预先计算并存储各切片的相对地址偏移量，在实际执行时仅需确定起始地址即可快速推导所有可合并切片的地址信息，这一设计不仅保证了方案的通用性，同时完全避免了运行时动态计算带来的额外开销。

合并方案具体实现如图 5 所示的算法流程，图 5 中示例张量转置的索引变换模式为 $(i,j,k,m,n,v)\Rightarrow(m,v,n,j,i,k)$，转置前张量索引大小为 $\{3,2,2,2,2,2\}$。

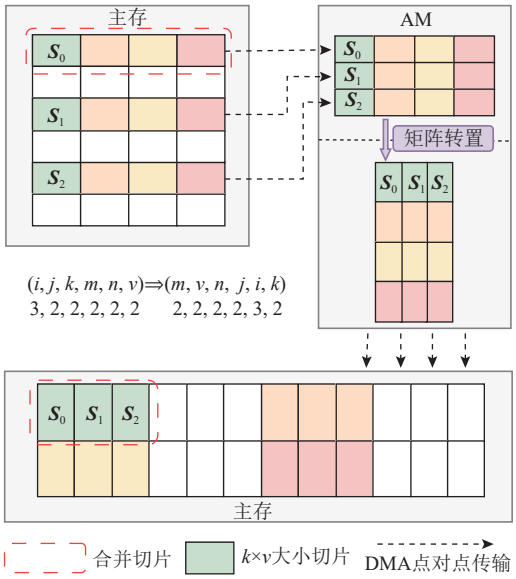



Fig. 5 An example for tensor transposition of merging scheme

图 5 合并方案张量转置示例

如果我们定义对应维度大小用函数 $extent()$ 进行表达，$i$ 维度的大小为 $extent(i)$。如果未进行访存合并，需将大小为 $k\times v$ 的 $extent(i)\times extent(j)\times extent(m)\times extent(n)$ 个张量切片（总共 24 个切片）传输到 AM 中，并执行 24 次转置函数，导致总共 48 次 DMA 激活。如果采用图 5 中所示的合并方案，则只需将 4 个切片合并为 1 个切片读入 AM，并在写回过程中合并 3 个切片为 1 个更大的切片。将这一方法扩展到其他行，总共需要 6 次读取和 8 次写回，即 14 次 DMA 激活，相比于未进行张量切片合并的 48 次 DMA 激活，显著减少了 DMA 启动开销。

合并方案在实现阶段中有诸多限制条件，核心为在 AM 大小和 DMA 引擎特性的约束下实现高效的矩阵合并算法，本文考虑了 3 个关键条件。

1)读写过程行方向元素的连续性度量。在写入过程中,张量切片需要在物理内存中保证连续存储,以提高 DMA 点对点传输效率。为实现这一目标,在矩阵合并过程中必须确保连续性。例如,考虑以下变换:

$$\boldsymbol{A}_{i_0,i_1,\cdots,i_m(j),\cdots,i_{N-1}(v)} \rightarrow \boldsymbol{B}_{Dim(i_0,i_1,\cdots,i_{N-1}(j))}。\tag{2}$$

在输入张量中,最内层维度$i_{N-1}$被定位为输出张量中的第$p$维。输出张量的最内层维度为$Dim(i_{N-1})$,位于输入张量中的第$q$维。张量$\boldsymbol{A}$中每个索引$i_n$的大小表示为$extA(i_n)$,从而形成一个$v\times j$矩阵的最大转置切片。单次 DMA 传输的最大数据帧可通过式(3)~(6)确定。

$$maxRF=\prod_{k=q+1}^{n-1} extA(i_k),\tag{3}$$

$$maxRB=extA\left(i_q\right)\times maxRF,\tag{4}$$

$$maxWF=\prod_{k=p+1}^{n-1} extB(Dim(i_k)),\tag{5}$$

$$maxWB=extB\left(Dim\left(i_p\right)\right)\times maxWF。\tag{6}$$

在 DMA 传输中,最大读取帧长度($maxRF$)和最大写回帧长度($maxWF$)分别表示单次 DMA 传输中能够读取和写回的最大数据帧长度。同时,单次 DMA 传输的最大读取块($maxRB$)和最大写回块($maxWB$)也分别由$maxRF$和$maxWF$以及数据帧数决定,简单理解就是$maxRF$是最大矩阵行,数据帧数目为行数$n$,共同决定整个矩阵切片大小为$n\times maxRF$。所以,$maxRB$和$maxWB$描述单次 DMA 传输中能够处理的最大张量数据块大小。

2)读写合并的冲突维度检测。在实际的合并操作中,往往难以同时实现$maxWB$与$maxRB$。具体而言,在合并过程中,为了保证合并对每个维度只发生 1 次,保证合并后处理数据与未合并条件下数据的一致性,参与读合并的切片将被禁止参与写合并,如图 5 中的$\boldsymbol{S}_0$, $\boldsymbol{S}_1$, $\boldsymbol{S}_2$这 3 个矩阵切片在读入 AM 空间时并未合并,仅在写回时进行合并。为此,合并方案需要首先确定读写合并冲突的维度$c$,实现方式为:从最内层维度开始,依次遍历输入和输出张量的索引序列,以定位首个破坏读写操作连续性的冲突值。该过程可通过式(7)实现。

$$c=i_k=Dim(i_m),\tag{7}$$

$$\text{where } k=\arg\max_{j\in[q,n-1]}(i_j=Dim(i_{m'}),\ m'\in[0,N-1])。\tag{8}$$

在确定读写合并冲突的维度$c$后,我们可以进一步从该维度扩展到可合并的维度。通过这种方式,$maxRB$和$maxWB$可以演化为式(9)(10):

$$maxRB=extA\left(i_q\right)\times\prod_{k=c+1}^{n-1} extA(i_k),\tag{9}$$

$$\max WB=extB\left(Dim\left(i_q\right)\right)\times\prod_{k=c}^{n-1} extB(Dim(i_k))。\tag{10}$$

3)AM 空间大小限制。合并块的大小不得超过 AM 空间的容量,以确保在读取和计算过程中不会发生内存溢出。具体而言,每个合并块的大小必须在 AM 的容量限制内进行计算。在此条件下,读入 AM 的数据块的总大小为$maxRWblock$,如式(11)所示。

$$maxRWblock=\lambda\times maxRF\times maxWF,\tag{11}$$

其中$\lambda$是防止超过 AM 大小的缩放因子,用于具体实现时的进一步任务分解,即根据缩放因子大小进行多次分批以应对过大的转置任务。

考虑上述条件下的合并方案整体算法如算法 3 所示。

**算法 3.** 访存合并算法。

输入:张量$\boldsymbol{A}$,形状$(i_0,i_1,\cdots,i_{N-1})$计为$A_{\text{size}}$;张量$\boldsymbol{B}$,形状$Dim(i_0,i_1,\cdots,i_{N-1})$计为$B_{\text{size}}$;AM 空间大小$AM_{\text{capacity}}$;

输出:经过合并后的后分块合集$FusedBlocks$。

① 初始化$FusedBlocks$结构用于存储合并张量切片信息;

② for each 潜在合并数据块 in $\boldsymbol{A}$ do

③ 寻找读写过程冲突维度$c$;

④ $R_{\text{start}}\leftarrow\max(c,q)$;

⑤ $maxRF=\prod_{k=R_{\text{start}}+1}^{n-1} extentA(i_k)$;

⑥ $W_{\text{start}}\leftarrow\max(c,p)$;

⑦ $maxWF=\prod_{k=W_{\text{start}}}^{n-1} extentB\left(d_{Dim(i_k)}\right)$;

⑧ $maxRWblock=\lambda\times maxRF\times maxWF$;

⑨ if $maxRWblock>AM_{\text{capacity}}$ then

⑩ 调整$\lambda$使$maxRWblock$小于$AM_{\text{capacity}}$;

⑪ end if

⑫ Add in $FusedBlocks$;

⑬ end for

⑭ 每个合并数据块在 DMA-P2P 的过程参数存储在$FusedBlocks$中;

⑮ return $FusedBlocks$。

本文合并方案设计的优势在于它有效地将高效的矩阵转置核心与 DMA 传输方案相结合,最大限度发挥向量化矩阵转置内核利用率的同时提高访存效

率。最终整个张量转置过程如算法 4 描述。

**算法 4.** ftmTT 算法。

输入：张量 $\boldsymbol{A}$，形状 $(i_0,i_1,\cdots,i_{N-1})$ 计为 $A_{\text{size}}$；张量 $\boldsymbol{B}$，形状 $Dim(i_0,i_1,\cdots,i_{N-1})$ 计为 $B_{\text{size}}$；

输出：形状为 $Dim(i_0,i_1,\cdots,i_{N-1})$ 的输出张量 $\boldsymbol{B}$。

① $FusedBlocks \leftarrow MergePlan(\boldsymbol{A},\boldsymbol{B},AM_{\text{capacity}})$;

② for each ***Block*** in $FusedBlocks$ do

③ 加载 ***Block*** 到 AM;

④ 在 AM 中执行 2 维转置核心;

⑤ 存储转置后的 ***Block*** 至 $\boldsymbol{B}$;

⑥ end for

⑦ return $\boldsymbol{B}$。

通过 2.4 节中描述的双缓冲技术以及合并块的连续读取可以进一步掩盖 2 维转置内核的执行时间。

### 2.4 双缓冲技术

为了进一步提升内存访问效率，我们采用双缓冲技术来掩盖 2 维转置核的执行时间。如图 6 所示，使用单缓冲的计算过程包括 3 个阶段：数据加载、2 维转置和数据存储。单核 DSP 执行时间的测量结果表明，数据传输可以完全掩盖 2 维转置的时间。通过使用优化后的向量化核，转置操作时间占总时间的 10%~20%。详细数据将在 3.6 节中提供。

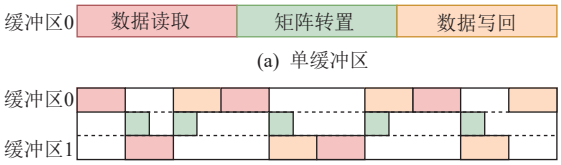



Fig. 6 Buffers allocation for ftmTT algorithm

图 6 ftmTT 算法缓冲区分配

本文考虑 2 种方法：1）单核双缓冲技术，以减少核间干扰；2）多核缓冲技术，以掩盖核间数据传输时间。ftmTT 高度依赖 AM 空间分配；在第 1 种方法中，将 AM（辅助存储器）空间划分为 4 个部分：2 个用于输入张量空间，2 个用于输出张量空间。从而将数据使用限制在 AM 空间的 1/4。这种划分减少了对更大张量切片的容量支持，与我们之前最大化切片大小以最小化传输开销的目标相矛盾。尽管计算可以完全重叠，但这种方法增加了传输开销，不利于在带有访存合并方案的场景。

第 2 种方法使用了 AM 的 1/2 空间。即使 2 维转置操作无法完全被掩盖，这种方法也最小化了 DMA 传输的启动开销，使其在理论上更加高效。因此，本文选择了多核掩盖方法，以实现更好的并行处理性能。实验部分的结果表明，尽管存在时间差异，但在多个核心上重叠执行 2 维转置操作能够实现完全掩盖。

### 2.5 主轴不变张量转置

对于主轴不变张量转置，张量转置前后对应的最内层索引不变，其对应的张量元素具有天然的局部性，例如在最内层索引维度中的相邻元素，在实际内存空间中转置前后仍然保持相邻，因此利用这种初始张量数据局部性可以实现更高效的张量转置。

通过使用 DMA 点对点传输，最内层索引对应的元素行作为最小的数据帧，而传输次数则由张量的“最大索引”决定。在读取时，帧步长与输入张量中最大索引对应维度中相邻元素的步长一致；在写入时，帧步长则转换为输入张量中最大索引在输出张量中对应索引的维度中相邻元素步长。这种方法由于最内层维度未改变，而不需要通过转置核心来改变最内层维度，因而这种方法能够高效利用 DMA 传输实现快速的张量转置。

上述过程也可以被视为将张量沿着最内层索引 $j$ 和最大索引 $v$ 展开。展开后，张量变为多个 $v\times j$ 的矩阵。与之前的转置方法不同，这里无需对这些切片进行转置，这些大小为 $v\times j$ 的张量切片可以直接从主存中读取并写回。由于不需要进行 2 维转置，这一过程不涉及 AM 空间中的中间传输。因此，总的 DMA 启动开销和传输次数减少为通用转置方法的一半，从而显著提升了性能。

## 3 实验与评估

### 3.1 实验设置

本文对在 FT-M7032 上实现的 ftmTT 进行了全面的测试和分析，并将其与运行在同平台集成 CPU 上的开源算法 TTC[15]，HPTT[16] 进行了对比，算法在与 DSP 共享主存的 CPU 核心上运行，操作系统为 Linux（内核版本 5.4.0）。TTC，HPTT 使用 GCC 版本 9.3 进行编译，并选择了最优的编译选项。

1）数据集。本研究使用了 TTC，HPTT 算法所采用的开源基准测试集。该数据集包含 57 个输入张量，总数据量约为 200 MB。这些张量数据包含 2~6 阶。基准测试中张量转置模式已达最简，避免索引融合对算法的影响，确保了测试的全面性和公平性。

2）测试标准。对于每次转置操作，进行 10 次测试并取平均值。平均带宽（*Bandwidth*）单位为 GBps，计算公式为：

$$Bandwidth = \frac{(ReadTimes + WriteTimes) \times TV}{1\,024^3 \times Time}, \quad (12)$$

其中$ReadTimes$表示张量数据读取次数，$WriteTimes$表示张量写回的次数，$TV$表示张量的体积(即张量中元素的总数)。

引入能效测试指标$BPW$(bandwidth per watt)，单位为GBps/W，对算法能效进行评估，计算公式为：

$$BPW = \frac{Bandwidth}{Power}, \quad (13)$$

其中$Bandwidth$为平均带宽，$Power$则为算法实际执行时所测量的平均功率。

在本文的测试中，CPU平台的算法HPTT使用了FT-M7032平台CPU的全部16个CPU核心，而本文提出的ftmTT算法则使用了平台中4个可用DSP簇中的1个，该DSP簇包含8个DSP核心，这样的设置可保证实验平台FT-M7032的CPU部分与DSP簇共享主存空间的访问带宽极限值一致。

使用STREAM基准测试[37]计算访存操作($\alpha x + y \to z$)的极限带宽，反映了实验平台带宽的理论极限值。对于实验平台CPU侧中的ARMv8架构CPU，STREAM基准测试达到了31.97 GBps的带宽，而在DSP上则达到了32.6 GBps，与理论极限带宽基本一致。

### 3.2 DSP端张量转置朴素算法实现

本节对DSP张量转置算法朴素实现进行测试，面向多核DSP实现算法1，即使用多重循环对张量数据遍历，并对最外层循环进行并行化处理，内层循环将被分发到8个DSP核中，进一步处理张量元素位置变换。与实验平台CPU端实现的朴素算法对比，结果如图7所示。

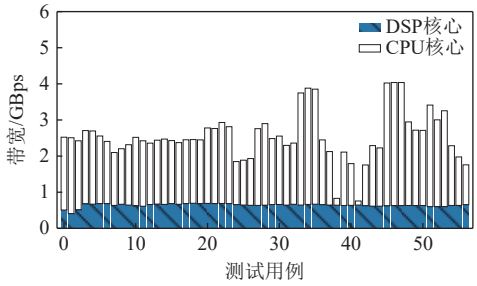



Fig. 7 Bandwidth effect of the naive implementation of tensor transposition

图7 张量转置朴素实现带宽效果

实验结果表明，DSP朴素转置算法虽然性能表现欠佳，但带宽数据保持稳定。其原因可能为DSP架构缺乏硬件缓存支持，其对主存的直接访问效率较低，但避免了因缓存缺失导致的数据布局敏感性，从而表现出稳定的带宽特性。相比之下，CPU端朴素转置实现虽然平均带宽达到2.54 GBps，高于DSP朴素转置实现，但仍远低于该平台STREAM基准测试31.97 GBps的理论极限。造成这一差距的主要原因在于：1)朴素转置算法中存在大量高跨步非连续内存访问，严重降低了缓存命中率；2)未对数据布局进行针对性优化，导致内存访问模式未能充分利用现代CPU的缓存层次结构优势。以上发现印证了1.5节关于不同架构特性对算法性能影响的分析，因此设计一种能够发挥DSP平台架构特性的高维转置算法是必要的。

### 3.3 对比其他算法

本实验使用实验平台中的1个DSP簇的8个DSP核执行ftmTT算法，使用与该簇共享主存的16个CPU核执行HPTT和TTC算法，保证了对主存访问期间的理论带宽极限一致性。

如图8所示，ftmTT算法实现了32.38 GBps的峰值带宽，达到75.96%的带宽利用率，并达到了STREAM基准测试的99.34%。其平均带宽为峰值带宽的59.1%。与HPTT和TTC相比，ftmTT分别实现了平均2.04倍和1.92倍的加速，单转置程序的最大加速比分别达到4.29倍和7.11倍。

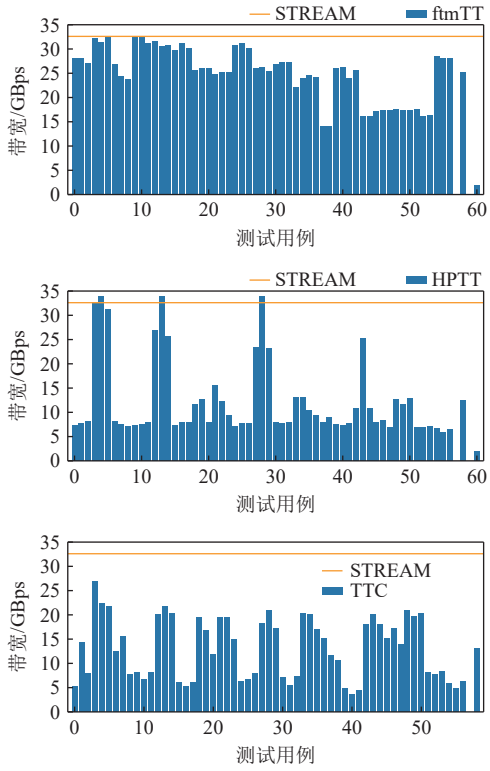



Fig. 8 Comparison of the bandwidth among ftmTT, HPTT, and TTC

图8 ftmTT与HPTT，TTC带宽对比

ftmTT 在测试中表现出更为稳定的性能。在用例 0~42 中保持了 25~32 GBps 的带宽；而在用例 43~53 中，尽管性能较用例 0~42 有所下降，但仍保持了相对稳定的表现，而 HPTT 算法和 TTC 算法的带宽则波动程度更大。

根据对比实验结果，可认为 ftmTT 算法提供了一种对 DSP 微内核架构更友好稳定的张量转置解决方案，能够高效利用 AM 空间以及发掘向量化能力。其关键原因在于，ftmTT 利用 DMA 引擎进行数据块的提取，并通过合并算法指导数据传输，从而更好地利用了数据局部性。此外，ftmTT 表现出更高的稳定性的原因在于不依赖于缓存命中，而是依赖于显式内存管理即数据搬运流程设计，并且在这种设计框架下使用 DMA 引擎进行数据搬运更具稳定性。

### 3.4 向量化内核适配评估

如图 9 所示，本实验使用单个 DSP 核心进行了测试，用于对比 ftmTT 算法在使用和未使用向量化 2 维转置内核时的性能差异。当最内层维度较大时（用例 0~27），适配向量化方法的性能显著更优；而当最内层维度较小时（例如用例 42~52），性能提升则较为有限。

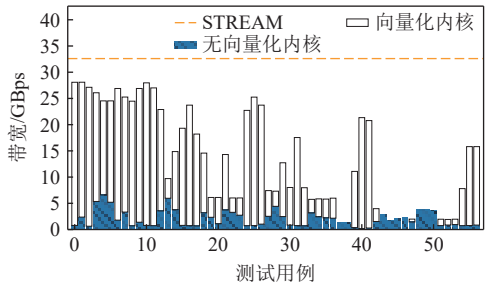



Fig. 9 Comparison of kernel effect of ftmTT algorithm with and without vectorization

图 9 ftmTT 算法有无向量化内核效果对比

如图 10 所示，较大的切片更有利于转置操作。当切片尺寸超过 24×24 时，带宽利用率趋于稳定。较大的切片意味着 DMA 传输启动开销的占比更低，且 DMA 传输次数更少。一旦切片尺寸达到一定阈值，开销对整体算法性能的影响将减弱。这与图 8 中 ftmTT 的整体表现一致。在用例 43~53 中，由于张量最内层索引较小，生成的切片也较小，导致性能低于支持较大张量切片的转置操作。

对于最内层维度较大的张量（例如用例 0~27），其对应的转置单元切片也更大，从而在相同数据量下减少了数据传输次数。此时，性能瓶颈在于转置核心算法而非传输过程，因此向量化带来了明显的

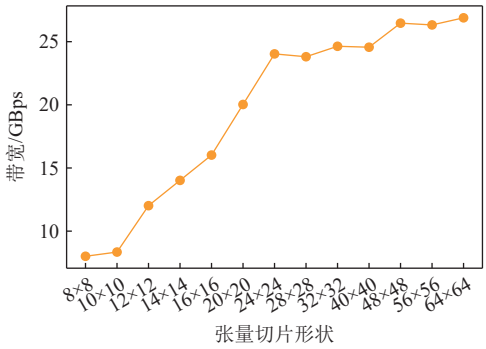



Fig. 10 Effect of ftmTT algorithm without memory access merging under different tensor slice configurations

图 10 无访存合并 ftmTT 算法在不同张量切片配置下的效果

性能提升，有效带宽得到了明显改善。

相比之下，对于最内层维度较小的张量（例如用例 42~53），根据查看 HPTT 开源库的数据，张量切片大小为 15×15，这小于设置中使用的最小转置核心。这导致向量化能力未能得到充分利用，且数据传输次数增加，而主要开销来自于 DMA 引擎传输的初始化过程。这解释了在最内层维度较小的张量上向量化操作的性能提升较为有限的原因。

### 3.5 合并算法评估

在本节中，本实验主要通过测试不同最小切片尺寸来说明切片合并的必要性。测试用例采用不可约简的 6 阶张量转置进行微调，在保持总数据量不变的同时，改变张量切片的最小单元尺寸，这相当于改变输入和输出张量的最内层维度。本实验测试了多组数据，切片尺寸从 8×8 到 64×64 不等。

图 11 中，将使用访存合并方案版本的 ftmTT 与无访存合并版本进行了对比，本实验仍然以 HPTT 开源数据集作为基准。如图 11 所示，张量阶数越高且切片尺寸越小时，合并算法的影响越显著。在 5 阶和 6 阶张量转置中（用例 42~53），平均带宽提升了 71.1%，通过观察数据集发现，这些用例中的大多数

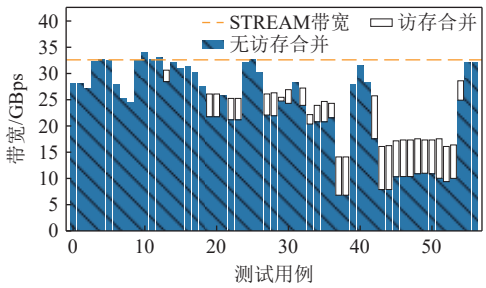



Fig. 11 Comparison of the effect of ftmTT algorithm with and without memory access merging scheme on bandwidth

图 11 ftmTT 算法有无访存合并方案对带宽影响对比

涉及切片尺寸为 15×15 的 2 维张量转置, 导致 DMA 传输启动开销过大。而 ftmTT 则可有效减少内存访问次数, 从而降低了启动开销, 提升转置性能。

### 3.6 双缓冲效果评估

如图 12 所示, 我们使用 HPTT 用例中的 5 阶和 6 阶张量转置来测试 ftmTT 算法中 2 维转置核心执行时间和 DMA 数据传输时间的运行时比例。可以观察到, ftmTT 是内存密集型算法, 2 维转置计算时间占总时间的比例不到 20%。因此, 在双缓冲策略下, 计算时间理论上能被完全掩盖。

如图 13 所示, 为了展示双缓冲的效果, 我们掩盖了核心转置计算, 仅以相同模式执行数据搬运。通过对比转置核心的带宽, 可见 2 条曲线基本重合, 差异小于 2%。这证明双缓冲完全掩盖了 2 维转置核心的执行时间, 而进一步证明了 DMA 引擎启动开销和 DMA 传输速率限制了整个算法的效率。

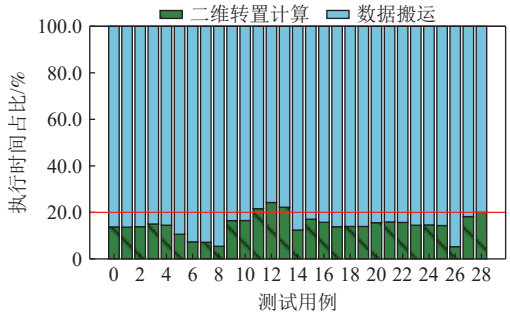



Fig. 12 Execution time proportion diagram of ftmTT

图 12 ftmTT 执行时间占比图

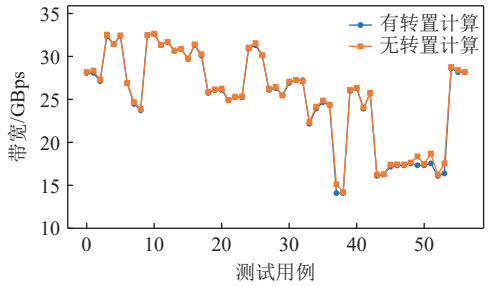



Fig. 13 Bandwidth of ftmTT algorithm with and without 2D transposition computation

图 13 有无 2 维转置计算 ftmTT 算法的带宽

### 3.7 算法能效分析

本节对 ftmTT, HPTT, TTC 这 3 个算法在 FT-M7032 实验平台的能效进行测试, 与前文实验环境保持一致, 使用实验平台中 1 个 DSP 簇的 8 个 DSP 核执行 ftmTT 算法, 使用与该簇共享主存的 16 个 CPU 核执行 HPTT 和 TTC 算法。测量算法运行时 *BPW*, 如式(13)描述, 可以反映算法的实际能效。

结果如图 14 所示, ftmTT 算法能效显著高于 HPTT 与 TTC, ftmTT 算法 *BPW* 达到 3.74 GBps/W, 远高于 HPTT 算法的 0.92 GBps/W 和 TTC 算法的 0.98 GBps/W, 验证了 ftmTT 算法在 DSP 平台上的能效优势。

### 3.8 实验结果讨论

实验结果表明, ftmTT 在多核 DSP 上的实现具有高效性, 得益于软件控制的片上内存管理和 DMA 传输模式, 能够在任意维度上保持一致的性能。在使用 HPTT 和 TTC 基准测试的实验中, ftmTT 实现了平均 59.1% 的带宽利用率, 峰值达到 75.96%, 并在平台上达到了 STREAM 基准测试的 99.34%。与在同一平台上共享相同内存的 CPU 上实现的 HPTT 和 TTC 算法相比, ftmTT 分别实现了 2.04 倍和 1.92 倍的加速。

向量化性能测试表明, 对于最内层维度较大的低维情况, 数据搬运并非主要限制因素, ftmTT 对向量化 2 维转置核心高度适配能够显著提升性能。而在最内层维度较小的高维情况下, ftmTT 算法主要受限于数据搬运过程中的传输开销, 此时访存合并算法带来了显著的性能提升。双缓冲测试表明, 在多核间采用双缓冲策略时, 整个 2 维转置时间可以完全被传输时间掩盖, 瓶颈限制为片内外访存带宽, 进

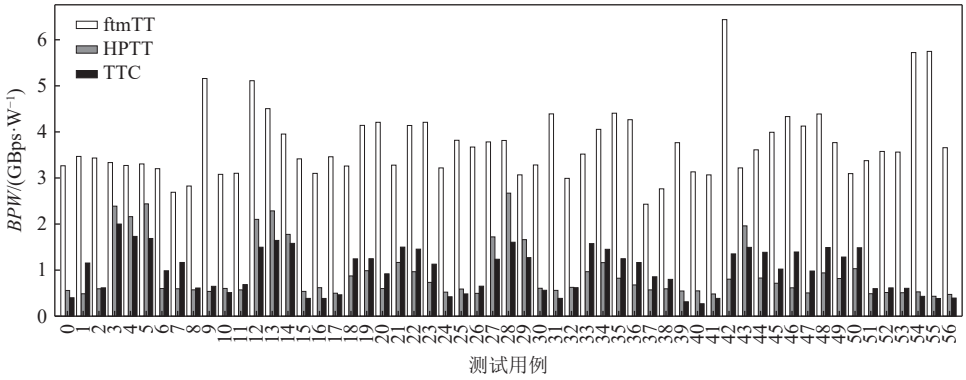



Fig. 14 Comparative chart for *BPW* of ftmTT, HPTT and TTC algorithms

图 14 ftmTT 与 HPTT, TTC 算法的 *BPW* 对比图

一步证明，本研究针对 DMA 数据搬运过程访存合并优化的必要性。算法运行能效测试则展现 ftmTT 算法在能效上的优秀表现，证明 ftmTT 设计发挥了国产 DSP 平台在低能耗方面的优势。

## 4 相关工作

多年来，针对 CPU 和 GPU 的张量转置问题，已经发展出许多成熟的解决方案。Lyakh[14] 提出了一种适用于多核 CPU, Intel Xeon Phi, NVIDIA Tesla GPU 的并行张量转置算法，与朴素方法相比实现了 2~3 倍的性能提升。然而，该解决方案仍未能实现最优带宽利用率，这表明张量转置操作中最大化硬件效率仍有提升空间。

相比之下，Springer 等人[16] 提出的 HPTT 算法通过提供更灵活的框架，解决了早期张量转置解决方案中的许多局限性。HPTT 旨在高效适应多种 CPU 架构，采用动态调度和有效的负载均衡来适应不同大小和形状的张量。然而，HPTT 缺乏对 GPU 或 DSP 架构的支持，这限制了其在涉及专用硬件的更广泛场景中的适用性。

在 GPU 张量转置方面，Vedurada 等人[18] 进一步推进了这一领域的发展，他们开发了 TTLG，一个用于 GPU 的张量转置库。TTLG 采用性能预测模型动态选择合适的转置内核和块大小，使其能够适应不同的工作负载特性。这种灵活性使 TTLG 在广泛的测试场景中实现了与其他 GPU 专用实现相当或更优的性能。TTLG 通过平衡工作负载分布和优化内存访问模式，在最大化 GPU 资源利用率方面发挥了关键作用。

本文工作从这些先前的研究中获得了重要启发。HPTT 工作的 2 维切片内核化方案，以及 cuTT[17] 和 TTLG[18] 对 GPU 共享内存的充分使用，启发了本研究针对 DSP 平台构建的张量转置方法。具体而言，本文采用 2 维切片方法来增强数据局部性并简化张量转置操作，此外，内存合并策略经过调整以适应 DSP 架构的独特特性，充分利用片上高速内存。在这些架构中，由于片上内存有限且需要有效的数据重用，高效的内存管理至关重要。本文针对 DSP 微内核架构进行思考，优化计算和内存效率，解决上述相关工作不适配 DSP 平台的问题，最终提供了一个面向的 DSP 平台的张量转置解决方案，且提出了一些关于 DSP 平台向量化发掘和访存优化的研究方法。

## 5 总结与展望

本文提出的 ftmTT 算法在多核 DSP 架构上展现了显著的性能，尤其在大规模转置任务中表现突出。算法通过设计一个充分发挥向量单元潜力的基于高速向量化转置核心的任意维度张量转置算法，通过张量切片访存合并方案高效利用片上内存以减少主存访问，并采用双缓冲技术进一步提高算法有效带宽，实现了张量转置在 DSP 平台的高性能。ftmTT 算法克服了传统张量转置算法对缓存不友好平台的局限性，有效利用了 DSP 的向量处理单元和软件显式管理的片上分层内存管理。在 FT-M7032 平台的实验结果表明，ftmTT 算法不仅在性能上优于现有的开源 HPTT 库，还在 DPS 平台上表现出广泛的适用性。未来我们的工作将重点进一步优化片上内存管理，整合张量收缩等强相关张量操作，并探索 ftmTT 算法在更广泛的异构计算平台上的潜力。

**作者贡献声明：**刘根程负责算法实现与实验测试并撰写论文；王庆林负责论文总体架构设计；洪楚河、彭兴、夏睿、梁亚玲负责对比实验的补充与论文的修改；张庆阳、车永刚、刘杰提出指导意见并修改论文。

## 参 考 文 献

[1] Anandkumar A, Ge R, Hsu D, et al. Tensor decompositions for learning latent variable models[J]. Journal of Machine Learning Research, 2014, 15(1): 2773−2832

[2] Khoromskaia V, Khoromskij BorisN. Tensor numerical methods in quantum chemistry: From Hartree–Fock to excitation energies[J]. Physical Chemistry Chemical Physics, 2015, 17(47): 31491−31509

[3] Kossaifi J, Khanna A, Lipton Z, et al. Tensor contraction layers for parsimonious deep nets[C]//Proc of the IEEE Conf on Computer Vision and Pattern Recognition Workshops. Piscataway, NJ: IEEE, 2017: 26−32

[4] Solomonik E, Matthews D, Hammond J, et al. Cyclops tensor framework: Reducing communication and eliminating load imbalance in massively parallel contractions[C]//Proc of the 27th IEEE Int Symp on Parallel and Distributed. Piscataway, NJ: IEEE, 2013: 813−824

[5] Springer P, Bientinesi P. Design of a high-performance GEMM-like tensor–tensor multiplication[J]. ACM Transactions on Mathematical Software, 2018, 44(3): 28: 1−28: 29

[6] Wang Yaohua, Li Chen, Liu Chang, et al. Advancing DSP into HPC, AI, and beyond: Challenges, mechanisms, and future directions[J]. CCF Transactions on High Performance Computing, 2021,

3: 114−125

[7] Igual F D, Ali M, Friedmann A, et al. Unleashing the high-performance and low-power of multi-core DSPs for general-purpose HPC[C]//Proc of the Int Conf on High Performance Computing, Networking, Storage and Analysis(SC'12). Piscataway, NJ: IEEE, 2012: 1−11

[8] Goldbogen G C. PRIM: A fast matrix transpose method[J]. IEEE Transactions on Software Engineering, 1981, 7(2): 255−257

[9] Mateescu G, Bauer G H, Fiedler R A. Optimizing matrix transposes using a POWER7 cache model and explicit prefetching[C]//Proc of the 2nd Int Workshop on Performance Modeling, Benchmarking and Simulation of High Performance Computing Systems. Piscataway, NJ: IEEE, 2011: 5−6

[10] Jodra J L, Gurrutxaga I, Muguerza J. Efficient 3D transpositions in graphics processing units[J]. International Journal of Parallel Programming, 2015, 43(5): 876−891

[11] Lu Qingda, Krishnamoorthy S, Sadayappan P. Combining analytical and empirical approaches in tuning matrix transposition[C]//Proc of the 15th Int Conf on Parallel Architectures and Compilation Techniques: New York: ACM , 2006: 233−242

[12] McCalpin J, Smotherman M. Automatic benchmark generation for cache optimization of matrix operations[C]//Proc of the 33rd Annual on Southeast Regional Conf (ACM-SE' 33). New York: ACM, 1995: 195−204

[13] Van Heel M. A fast algorithm for transposing large multidimensional image data sets[J]. Ultramicroscopy, 1991, 38(1): 75−83

[14] Lyakh D I. An efficient tensor transpose algorithm for multicore CPU, Intel Xeon Phi, and NVIDIA Tesla GPU[J]. Computer Physics Communications, 2015, 189: 84−91

[15] Springer P, Sankaran A, Bientinesi P. TTC: A tensor transposition compiler for multiple architectures[C]//Proc of the 3rd ACM SIGPLAN Int Workshop on Libraries, Languages, and Compilers for Array Programming(SOFT 2016). New York: ACM, 2016: 41−46

[16] Springer P, Su Tong, Bientinesi P. HPTT: A high-performance tensor transposition C++ library[C]//Proc of the 4th ACM SIGPLAN Int Workshop on Libraries, Languages, and Compilers for Array Programming. New York: ACM, 2017: 56−62

[17] Hynninen A P, Lyakh DmitryI. cuTT: A high performance tensor transpose library for CUDA compatible GPUs[J]. arXiv preprint, arXiv: 1705.01598, 2017

[18] Vedurada J, Suresh A, Rajam A S, et al. TTLG—An efficient tensor transposition library for GPUs[C]//Proc of the 2018 IEEE Int Parallel and Distributed Processing Symp (IPDPS). Piscataway, NJ: IEEE, 2018: 578−588

[19] Yin Shangfei, Wang Qinglin, Hao Ruochen, et al. Optimizing irregular-shaped matrix-matrix multiplication on multi-core DSPs[C]//Proc of the 2022 IEEE Int Conf on Cluster Computing (CLUSTER). Piscataway, NJ: IEEE, 2022: 451−461

[20] Hong Chuhe, Wang Qinglin, Mao Runzhang, et al. SaSpGEMM: Sorting-avoiding sparse general matrix-matrix multiplication on multi-core processors[C]//Proc of the 53rd Int Conf on Parallel Processing. New York: ACM, 2024: 1166−1175

[21] Ma Sheng, Liu Zhong, Chen Shenggang, et al. Coordinated DMA: Improving the DRAM access efficiency for matrix multiplication[J]. IEEE Transactions on Parallel and Distributed Systems, 2019, 30(10): 2148−2164

[22] Bi Deshun, Li Shengguo, Dong Dezun, et al. Optimizing SpMV on heterogeneous multi-core DSPs through improved locality and vectorization[C]//Proc of the 53rd Int Conf on Parallel. New York: ACM, 2024: 1145−1155

[23] Mo Haotian, Wang Qinglin, Liao Linyu, et al. Detailed analysis and optimization of irregular-shaped matrix multiplication on multi-core DSPs[C]//Proc of the 53rd Int Conf on Parallel Processing. New York: ACM, 2024: 1176−1186

[24] Wang Yang, Wang Qinglin, Pei Xiangdong, et al. High performance dilated convolutions on multi-core DSPs[J]. CCF Transactions on High Performance Computing, 2024, 6: 78−93

[25] Wang Yang, Wang Qinglin, Pei Xiangdong, et al. Optimizing pointwise convolutions on multi-core DSPs[C]//Proc of the 2023 Int Conf on Algorithms and Architectures for Parallel. Berlin: Springer, 2023: 209−223

[26] Zuo Xiaohan, Xiao Chunhua, Wang Qinglin, et al. Optimizing batched small matrix multiplication on multi-core DSP Architecture[C]//Proc of the 2024 IEEE Int Symp on Parallel and Distributed with Applications (ISPA). Piscataway, NJ: IEEE, 2024: 426−435

[27] Huang Xiandong, Wang Qinglin, Lu Shuyu, et al. NUMA-aware FFT-based convolution on ARMv8 many-core CPUs[C]//Proc of the 2021 IEEE Int Conf on Parallel & Distributed Proc with Applications, Big Data & Cloud Computing, Sustainable Computing & Communications, Social Computing & Networking (ISPA/BDCloud/SocialCom/SustainCom). Piscataway, NJ: IEEE, 2021: 1019−1026

[28] Zhao Xiaolei, Chen Zhaoyun, Shi Yang, et al. Kernel code automatic generation framework on FT-Matrix[J]. Journal of Computer Research and Development, 2023, 60(6): 1232−1245 (in Chinese)
(赵宵磊, 陈照云, 时洋, 等. 面向飞腾迈创数字处理器的内核代码自动生成框架[J]. 计算机研究与发展, 2023, 60(6): 1232−1245)

[29] Huang Xiandong, Wang Qinglin, Lu Shuyu, et al. Evaluating FFT-based algorithms for strided convolutions on armv8 architectures?[J]. ACM SIGMETRICS Performance Evaluation Review, 2022, 49(3): 28−29

[30] Paszke A, Gross S, Massa F, et al. PyTorch: An imperative style, high-performance deep learning library[C]//Proc of the 33rd Int Conf on Neural Information Processing Systems. Red Hook, NY: Curran Associates Inc, 2019: 8026−8037

[31] Abadi M, Barham P, Chen J M, et al. TensorFlow: A system for large-scale machine learning[C]//Proc of the 12th USENIX Symp on Operating Systems Design and Implementation (OSDI'16). Berkeley, CA: USENIX Association, 2016: 265−283

[32] Vaswani A, Shazeer N, Parmar N, et al. Attention is all you need[C]//Proc of the 31st Int Conf on Neural Information Processing Systems. Red Hook, NY: Curran Associates Inc, 2017: 6000−6010

[33] Baydin A G, Pearlmutter B A, Radul A A, et al. Automatic differentiation in machine learning: A survey[J]. Journal of Machine Learning Research, 2018, 18(1): 5595−5637

[34] Hackbusch W. Tensor Spaces and Numerical Tensor Calculus[M]. Berlin: Springer, 2012

[35] Stoudenmire E M, Schwab D J. Supervised learning with quantum-inspired tensor networks[C]//Proc of the 30th Int Conf on Neural

Information Processing Systems. Red Hook, NY: Curran Associates Inc, 2016: 4799−4807

[36] Pei Xiangdong, Wang Qinglin, Liao Linyu, et al. Optimization of parallel matrix transposition algorithm for multi-core DSPs[J]. Journal of National University of Defense Technology, 2023, 45(1): 57−66 (in Chinese)
(裴向东, 王庆林, 廖林玉, 等. 多核数字信号处理器并行矩阵转置算法优化[J]. 国防科技大学学报, 2023, 45(1): 57−66)

[37] McCalpin J D. Memory bandwidth and machine balance in current high performance computers[J]. IEEE Computer Society Technical Committee on Computer Architecture Newsletter, 1995, 2: 19−25



**Liu Gencheng**, born in 2000. PhD candidate. His main research interests include parallel computing and high-performance operator generation.
**刘根程**, 2000 年生。博士研究生。主要研究方向为并行计算、高性能算子生成。



**Wang Qinglin**, born in 1987. PhD, associate professor. Member of CCF. His main research interests include parallel algorithm, reconfigurable computing, and machine learning.
**王庆林**, 1987 年生。博士, 副研究员。CCF 会员。主要研究方向为并行算法、可重构计算、机器学习。



**Hong Chuhe**, born in 2001. PhD candidate. His main research interests include sparse matrix computation, high-performance computing, and heterogeneous acceleration.
**洪楚河**, 2001 年生。博士研究生。主要研究方向为稀疏矩阵计算、高性能计算、异构加速。



**Peng Xing**, born in 2001. PhD candidate. His main research interests include parallel discrete event simulation, machine learning, and heterogeneous computing.
**彭　兴**, 2001 年生。博士研究生。主要研究方向为并行离散事件仿真、机器学习、异构计算。



**Xia Rui**, born in 1999. PhD candidate. His main research interests include artificial intelligence, high-performance computing, and ocean acoustics.
**夏　睿**, 1999 年生。博士研究生。主要研究方向为人工智能、高性能计算、海洋声学。



**Liang Yaling**, born in 2001. Master candidate. Her main research interests include high-performance computing and artificial intelligence.
**梁亚玲**, 2001 年生。硕士研究生。主要研究方向为高性能计算、人工智能。



**Zhang Qingyang**, born in 1995. PhD, assistant professor. His main research interests include high-performance computing, fluid dynamics, and numerical simulation of nuclear reactors.
**张庆阳**, 1995 年生。博士, 助理研究员。主要研究方向为高性能计算、流体力学、核反应堆数值模拟。



**Che Yonggang**, born in 1973. PhD, professor. His main research interests include parallel algorithms, and high-performance applications and performance evaluation of computer systems.
**车永刚**, 1973 年生。博士, 研究员。主要研究方向为并行算法、高性能应用和计算机系统性能测评。



**Liu Jie**, born in 1969. PhD, professor, PhD supervisor. His main research interests include supercomputer, CFD, artificial intelligence, parallel algorithms, parallel applications, and high-performance computing.
**刘　杰**, 1969 年生。博士, 研究员, 博士生导师。主要研究方向为超级计算、计算流体力学、人工智能、并行算法、并行应用、高性能计算。