计算机科学
COMPUTER SCIENCE

基于RISC-V指令扩展的神经网络计算加速架构

蔡成欢, 王一品, 许嘉滨, 张逢喆, 周学功, 曹伟, 张帆, 余新胜

引用本文

蔡成欢, 王一品, 许嘉滨, 张逢喆, 周学功, 曹伟, 张帆, 余新胜. 基于RISC-V指令扩展的神经网络计算加速架构[J]. 计算机科学, 2025, 52(12): 1-8.

CAI Chenghuan, WANG Yipin, XU Jiabin, ZHANG Fengzhe, ZHOU Xuegong, CAO Wei, ZHANG Fan, YU Xinsheng. Neural Network Acceleration Architecture Based on RISC-V Instruction Set Extension[J]. Computer Science, 2025, 52(12): 1-8.

计算机科学 COMPUTER SCIENCE
http://www.jsjkx.com
DOI:10.11896/jsjkx.250600014

基于 RISC-V 指令扩展的神经网络计算加速架构

蔡成欢[1]　王一品[1]　许嘉滨[2]　张逢喆[3]　周学功[3]　曹　伟[3]　张　帆[3]　余新胜[4]
1 复旦大学软件学院　上海 200433
2 复旦大学计算机科学技术学院　上海 200433
3 复旦大学大数据研究院　上海 200433
4 中国电子科技集团公司第三十二研究所　上海 201899
(18034065482@163.com)

摘　要　针对现阶段以 RISC-V 为核心的神经网络加速器对 Transformer 架构模型中矩阵计算及非线性计算加速不足的问题，开展了基于 RISC-V 指令扩展的神经网络计算加速架构研究，提出名为 Taurus 的神经网络加速器架构。针对模型架构特点，进行了矩阵指令扩展，并使用脉动阵列进行矩阵乘累加计算；为支持非线性计算加速，进行向量指令扩展，并设计特殊向量单元完成 LayerNorm 和 Softmax 的计算；为保证数据供给平衡，优化访存指令扩展，以保证矩阵计算单元、向量计算单元的数据供给，在进行指令扩展时采用标量寄存器的扩展方式，将运算数据信息存入寄存器中增大了寻址空间，以保证进行大规模数据运算时生成较少的指令条数。Taurus 神经网络加速器架构在 Gem5 平台上完成了周期精确的模拟仿真，与开源加速器 Gemmini 相比，进行通用矩阵乘法运算时，脉动阵列利用率提高 80%；在 ResNet50 和 BERT 模型推理中，Taurus 与 Gemmini 相比，分别获得 1.3 倍和 31.3 倍的加速；与 RISC-V 相比，性能分别获得 1467 倍和 4513 倍的加速。

关键词：神经网络；矩阵计算；非线性计算；指令扩展
中图分类号　TP302

Neural Network Acceleration Architecture Based on RISC-V Instruction Set Extension

CAI Chenghuan[1], WANG Yipin[1], XU Jiabin[2], ZHANG Fengzhe[3], ZHOU Xuegong[3], CAO Wei[3], ZHANG Fan[3] and YU Xinsheng[4]
1 School of Software Engineering, Fudan University, Shanghai 200433, China
2 School of Computer Science and Technology, Fudan University, Shanghai 200433, China
3 Institute of Big Data, Fudan University, Shanghai 200433, China
4 The 32nd Research Institute, China Electronics Technology Group Corporation (CETC), Shanghai 201899, China

Abstract　To address the current shortcomings of RISC-V-based neural network accelerators in accelerating matrix computations and nonlinear operations within Transformer-based models, a neural network acceleration architecture based on RISC-V instruction set extension, named Taurus, is proposed. This architecture introduces matrix instruction extensions tailored to the characteristics of Transformer models and employs a systolic array to perform matrix multiply-accumulate operations. To accelerate nonlinear computations, vector instruction extensions are added, along with the design of specialized vector units to efficiently compute operations such as LayerNorm and Softmax. To ensure balanced data supply, memory access instruction extensions are optimized to provide sufficient data throughput to the matrix and vector computation units. The instruction set extensions adopt a scalar register expansion approach, embedding operand data information directly into the registers. This increases the addressing space and reduces the number of instructions required for large-scale data computations. The Taurus neural network accelerator architecture is cycle-accurately simulated on the Gem5 platform. Compared with the open-source accelerator Gemmini, Taurus achieves an 80% improvement in systolic array utilization during general matrix multiplication. For inference tasks on ResNet50 and BERT models, Taurus delivers 1.3× and 31.3× speedups respectively over Gemmini. Compared with the baseline RISC-V, Taurus achieves 1467× and 4513× performance improvements respectively.

Keywords　Neural networks, Matrix computation, Nonlinear computation, Instruction set extension

到稿日期：2025-06-03　返修日期：2025-09-04
基金项目：国家重点研发计划(2022YFB4500903)
This work was supported by the National Key R&D Program of China (2022YFB4500903).
通信作者：张逢喆(fzzhang@fudan.edu.cn)

1 引言

随着人工智能领域的快速发展，基于自注意力机制的Transformer模型架构在自然语言处理和计算机视觉方面取得巨大成就[1]。该模型架构的显著特点为计算量巨大，需要专用加速器架构进行推理工作[2]。为支持模型高效推理，谷歌推出TPU[3]用于加速矩阵乘累加计算；华为推出昇腾系列处理器[4]加速矩阵运算及向量运算；Liu等[5]提出Vision Transformer大模型专用芯片设计。现阶段，神经网络加速器的相关研究更专注于模型中矩阵运算的加速，对非线性加速的支持不足，Kim等[6]指出非线性计算会涉及多次访存操作，造成较大的时间开销，导致模型推理延缓。

为实现模型推理时更完整的加速，需采用处理器与加速器相结合的设计。开源指令集RISC-V发展迅速并具有低成本、易扩展等优点[7]，为加速器架构设计提供了便利。以RISC-V为核心扩展指令[8-10]加速神经网络模型时存在以下问题：矩阵指令数据寻址空间较小，进行大规模矩阵运算时产生指令条数多，增大了指令解码、指令传输的时间开销，同时造成矩阵运算单元利用率低；对非线性计算支持不足，访存频繁，计算速度慢，增加了RISC-V负载；单条访存指令寻址空间小，无法满足计算单元数据供给平衡；神经网络加速器架构功能验证、性能评估周期较长。

依靠编写可综合的寄存器代码验证加速器架构的正确性是一个复杂且耗时的过程，文献[11]指出利用周期精确级的模拟仿真器对神经网络加速器进行功能验证和性能评估是发展的主流方向。Gem5[12]是一个时钟周期精确的模块化事件驱动的体系结构模拟器，gem5-Aladdin[13]，gem5-SALAM[14]和Gem5-accel[15]均利用Gem5实现架构仿真验证，在时序方面得到了与Vivado[16]工具相当的效果。

针对加速器设计存在的问题，本文就基于RISC-V的指令扩展的神经网络加速计算架构展开研究，提出名为Taurus的加速器架构，并在Gem5平台上进行模拟仿真。在满足加速矩阵等线性计算和激活类非线性计算的条件下，提高矩阵运算单元利用率并快速完成架构验证工作。本文的主要工作包括以下几个方面：

1)针对神经网络模型计算负载进行矩阵运算类指令的扩展，将运算数据信息存入寄存器中，扩大了寻址空间。

2)扩展向量指令并设计相应的特殊向量运算单元，以支持LayerNorm和Softmax等非线性计算，优化数据流向。

3)增大访存指令寻址空间，保证矩阵运算单元、特殊向量运算单元数据的供给平衡。

4)使用Gem5模拟器中对所提出的神经网络计算架构进行周期精确的模拟仿真及参数化设计。实验结果表明，Taurus神经网络加速器与Gemmini[17]相比，矩阵运算单元利用率提升80%；在ResNet50和BERT模型推理中与Gemmini相比，性能分别获得1.3倍和31.3倍的加速；与RISC-V相比，性能分别获得1467倍和4513倍的加速。

本文第2章对RISC-V神经网络加速器的相关工作和RISC-V指令扩展方法进行介绍；第3章对面向神经网络加速的RISC-V指令扩展的具体信息及相应功能单元进行介绍；第4章对加速器模拟仿真方法进行介绍；第5章通过实验证明了Taurus加速器的有效性；最后总结全文并展望未来。

2 基于RISC-V的神经网络加速器的相关工作和指令扩展方法

2.1 基于RISC-V的神经网络加速器的相关工作

为加速神经网络模型推理，学者近年来围绕RISC-V进行了多项研究，通过对RISC-V完成向量扩展、矩阵扩展来加速模型运行，显著提升了性能。

Ara[18]通过RISC-V接口进行向量协处理器扩展，在RISC-V核心与协处理器之间采用轻量级接口，核心中的调度器将解码后的指令发射至协处理器。在向量指令执行时，协处理器采用了可变数量的计算通道设计，每个计算通道具有独立的通道序列器，可以跟踪多达8个的并行向量指令，且每个通道还具有一组向量寄存器文件和功能单元。

Gaurav等[19]对RISC-V进行矩阵乘法扩展，为满足矩阵乘法需求，设计了一种低功耗且高性能的乘加单元，该单元采用修改过的Booth乘法器和先行进位加法器。其设计是对RISC-V进行了协处理器的扩展，添加少量矩阵运算相关指令，采用单指令多数据的驱动方式来完成矩阵乘法运算。

Tai[20]在RISC-V协处理器的基础上开展矩阵运算扩展研究，其研究核心在于矩阵运算单元的指令调度，通过指令调度掩盖流水线开销，以达到高效的运算效果。其采用一个名为RVM的流式表系统来进行指令的调度，旨在通过异步执行流程和拆分位操作来加速基本矩阵扩展。流式表可以将具有多个位操作的长延迟算术指令分解，从而最小化算术指令的延迟。

OpenGeMM[21]提出一种高利用率的矩阵乘法加速器架构，其中的矩阵乘法阵列的数据通路被概念化为一个3D MAC阵列，该阵列对大小为(M,K)的切片矩阵$\boldsymbol{A}$和大小为(K,N)的切片矩阵$\boldsymbol{B}$进行空间处理，以生成大小为(M,N)的切片矩阵$\boldsymbol{C}$。3D MAC阵列被组织为一个大小为(M,N)的网格，其中包含了大小为K的向量点积单元(DotProd)，以空间展开矩阵$\boldsymbol{A}$,$\boldsymbol{B}$和$\boldsymbol{C}$的所有维度。具体而言，来自$\boldsymbol{A}$矩阵的向量和来自$\boldsymbol{B}$矩阵的向量在DotProd阵列中水平和垂直广播，从而最大化空间数据重用。在单个DotProd内，k次乘法结果被组合累加以获得一个$\boldsymbol{C}$的结果，整个架构中片上缓存采用多BANK缓冲设计，在数据输出到矩阵乘法阵列时实现高并行。

Gemmini是一个开源的全栈DNN加速器设计基础架构，提供了矩阵运算扩展的新思路。其设计采用了空间阵列形式，具有两级层次结构。空间阵列首先由tiles(计算块)组成，tiles通过显式流水寄存器连接。每个单独的tile可以进一步分解为PEs(处理单元)数组，其中同一个tile内的PEs通过组合逻辑连接。每个PE在每个周期执行一次乘加(MAC)操作，使用权重固定或输出固定的数据流。Tiles由矩形数组的PEs组成，同一tile内的PEs通过组合逻辑连接，没有流水寄存器，每个PE仅与其直接相邻的单元共享输入和输出。

先前的以RISC-V进行指令扩展的神经网络加速器在大

规模矩阵运算时支持不友好，对非线性计算支持不足，造成频繁的数据搬运，导致模型推理速度慢。

2.2 RISC-V 指令扩展方法

RISC-V 核心支持标量计算和向量计算[22]，其两类指令均具有各自的寄存器，即标量寄存器和向量寄存器。在进行矩阵类指令扩展时，按照是否增加专属矩阵寄存器分为以下两类。

1）增添矩阵寄存器：该方法通过增加专用矩阵寄存器来支持矩阵运算指令扩展。达摩院玄铁系列处理器[23]提供了典型参考，其扩展指令的设计融入了向量扩展的设计思想和实现，并可以与向量扩展解耦。由于其指令扩展采用完全独立的矩阵处理单元，需要设计矩阵寄存器。为支持矩阵数据的高效存储和操作，该架构采用二维结构的矩阵寄存器设计，确保行数固定、列数自适应不同位宽。

此指令扩展方法及统一的矩阵寄存器设计虽简化了工具链设计，但考虑到矩阵计算单元的资源利用率，需要对矩阵寄存器进行合理调度以保证数据供给，尽可能保证计算单元处于忙碌状态，这对编程方面提出了更高要求，需要花费精力协调寄存器的复用。此外，玄铁处理器中矩阵寄存器的编号字段仅为三位，限制了片上缓存的寻址空间。另外，独立的矩阵寄存器堆栈会导致线程切换时性能损失较大[24]。

2）复用标量寄存器：RISC-V 指令扩展可以采用标量寄存器复用的方式，将加速器运算所需信息存入标量寄存器中，进行指令传输时将指令码和寄存器中的信息全部传输至加速器，这样的扩展方式将扩展指令穿插在标准的处理器指令中，在指令识别时只需要通过操作码的字段信息即可完成扩展指令的识别，指令的具体解码工作则交由加速器完成。

伯克利大学发布的开源项目 Gemmini 加速器扩展指令设计采用了标量寄存器复用的方式，其访存类指令和计算类指令对每次要操作的数据大小有严格限制，单条计算指令只驱动与计算阵列等大的数据进行计算，进行大规模矩阵乘法运算时会生成较多指令条数。RISC-V 设计手册中[25]指出，通过指令接口发出超过连续 4 条读指令时，开始显示出吞吐量限制，造成较大的延迟代价。过多的扩展指令生成，一方面在指令传输过程中会造成较大的时间开销；另一方面，扩展指令中涉及到的寄存器为标量寄存器，在处理器核心中，标量寄存器只有 32 个，大量扩展指令生成对寄存器的占用会对标量计算造成影响。

与引入专用矩阵寄存器的方法相比，采用标量寄存器复用方式在硬件实现上具有较低的复杂性。该方法无需对 RISC-V 核心的寄存器结构及指令译码逻辑进行结构行修改，而是通过标准的 Custom 接口指令将控制信息和数据地址编码到标量寄存器中，由加速器侧完成指令解码与执行，可在多个 RISC-V 内核间更好地复用，模块化程度高，保留了处理器核心的简洁性与通用性[26]。另外，通过复用标量寄存器进行指令设计时，其指令控制路径简单，控制逻辑资源开销低于独立矩阵寄存器设计方案。

综合来看，指令扩展时采用标量寄存器复用方式可降低指令扩展难度，且不会出现独立矩阵寄存器堆栈在线程切换的开销。此指令扩展方法需要解决指令生成数量过多的问题，对指令字段的设计提出了更高要求，需保证每条指令能尽可能填充较多的有效信息，扩大指令寻址空间，最终降低指令传输和译码的时间开销，并缓解标量寄存器长时间占用问题。指令设计的改进方法将在各类扩展指令设计的小节中进行详细介绍。

3 面向神经网络加速器的 RISC-V 指令扩展

3.1 整体架构设计

Taurus 神经网络加速器的整体架构如图 1 所示。加速器架构由指令控制模块（Inst_Control）、访存模块（Mem_Access）、片上缓存（On-chip Buffer）、转置模块（Transpose）、特征图变换模块（im2col）、脉动阵列（Systolic Array）、累加器（Accumulator）及特殊向量计算单元（Special_Vector）共同组成。

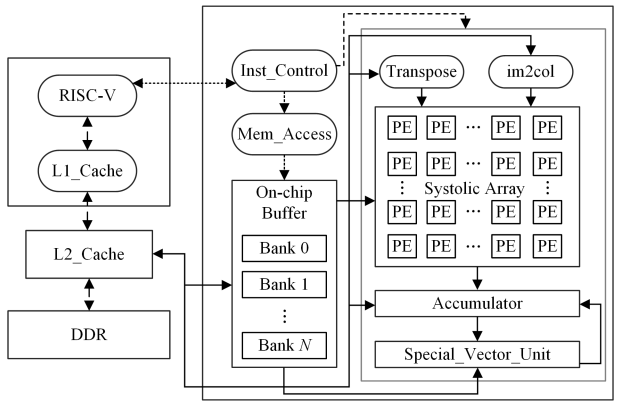

图 1 Taurus 加速器的整体架构图

Fig. 1 Overall architecture of the Taurus accelerator

加速器架构设计中，指令通路和数据通路的设计极为关键，图 1 中长虚线箭头代表 RISC-V 与内存之间的指令及数据交互，点虚线箭头表示 RISC-V 与加速器架构之间的指令流向，实线箭头则表示数据流向。加速器架构中，指令控制模块在接收到 RISC-V 核心传输的指令后，对指令进行微解码，再根据不同的指令控制数据变换单元、各类计算单元及访存模块进行工作。数据通路的设计是为了减少数据在 RISC-V 与加速器之间及加速器内部的移动，从而降低数据搬运的时间开销。由于在多头注意力机制中，两个矩阵进行乘法之后紧接着需要进行 Softmax 和 LayerNorm 的计算，因此在数据通路进行设计时要尽可能保证数据流经脉动阵列进行矩阵乘法之后能够流向特殊向量单元；同时考虑到单独进行激活函数的计算，还需要保证数据可以直接流向特殊向量单元并单独使用此计算单元。

根据加速器基本架构的设计，扩展指令设计需要保证数据正确流向和支持架构中各类单元的正常运行，同时尽可能减少内存与加速器的数据交换次数，以降低数据交互的时间开销。根据这些要求，本文的扩展指令采用复用标量寄存器并以 Gemmini 指令扩展为基础进行改进。除了无需进行独立寄存器的扩展外，还将更大的寻址空间信息填充到扩展指令中，从而能够更高效地支持加速器运算。

本文除了进行矩阵类指令扩展外，还支持 Softmax 和 LayerNorm 非线性运算指令，并合理设计了配置指令以支持

访存类、矩阵类、向量类指令的运行。

3.2 矩阵指令的扩展

本节将对扩展的矩阵运算类指令的指令码、名称、功能及个别矩阵类指令中寄存器涉及到的详细字段进行说明，同时介绍该类指令所驱动的脉动阵列单元的功能。

3.2.1 矩阵指令字段设计

矩阵相关的指令有 matmul. preload，matmul. compute. preload，matmul. compute. accumulated 和 madd，具体的指令字段码如图 2 所示。指令的功能分别为矩阵预加载、根据预加载值进行计算、根据上次矩阵计算结果进行累加和矩阵加法。

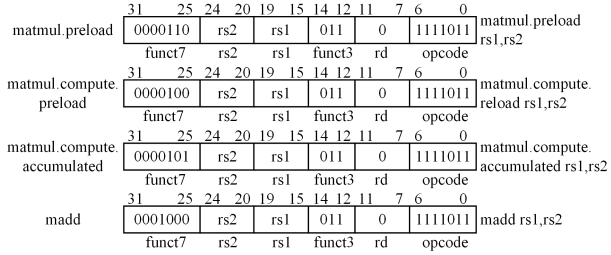

图 2 矩阵指令字段码

Fig. 2 Matrix instruction field codes

为支持大规模矩阵运算，扩大寻址空间，将矩阵大小信息存入指令中的标量寄存器中，以矩阵运算 $\boldsymbol{C}=\boldsymbol{A}*\boldsymbol{B}+\boldsymbol{D}$ 为例，对涉及到的矩阵预加载指令 matmul. preload 进行详细的字段说明。

1）rs1[31:0]字段存储 $\boldsymbol{D}/\boldsymbol{B}$ 矩阵的片上缓存地址，如果使用输出固定计算模式，则该地址为 $\boldsymbol{D}$ 矩阵地址；若使用权重固定计算模式，则该地址为 $\boldsymbol{B}$ 矩阵地址。

2）rs1[47:32]字段存储 $\boldsymbol{D}/\boldsymbol{B}$ 矩阵列数。

3）rs1[63:48]字段存储 $\boldsymbol{D}/\boldsymbol{B}$ 矩阵行数。

4）rs2[31:0]字段存储 $\boldsymbol{C}$ 矩阵的片上缓存地址，该字段若设置为全高位，则表示 $\boldsymbol{C}$ 矩阵不会写入片上缓存。

5）rs2[47:32]字段存储 $\boldsymbol{C}$ 矩阵列数。

6）rs2[63:48] 字段存储 $\boldsymbol{C}$ 矩阵行数。

由扩展的矩阵运算指令可知，为支持大规模数据计算并尽可能减少生成指令的数量，在存储相关运算矩阵信息时，矩阵起始地址大小为 32 位，矩阵的行列数大小为 16 位，在数据类型为 8 位整型的情况下，单条矩阵运算指令可支持行列数最大为 2^{16} 的矩阵大小运算。

3.2.2 矩阵指令运算单元

矩阵运算时，本文相对应的单元采用脉动阵列的形式，并支持权重固定(WS)与输出固定(OS)两种数据流形式[27-28]。

WS 的数据流计算模式如图 3 所示。以 2 * 2 矩阵乘法为例，权重矩阵($\boldsymbol{B}$)被读取并预先加载到相应的处理元素中，保持静止，同时另一个矩阵($\boldsymbol{A}$)在脉动阵列中进行传播，以执行尽可能多的使用相同权重的操作。当对同一输入矩阵进行多次操作时，这种计算模式非常适用，能够最小化对片上缓存中权重矩阵的访问，从而减少对缓存的访问。同时，为了确保传播的矩阵 $\boldsymbol{A}$ 的每个元素在正确的时钟周期到达处理元素，在脉动阵列行的输入端设计了移位寄存器。

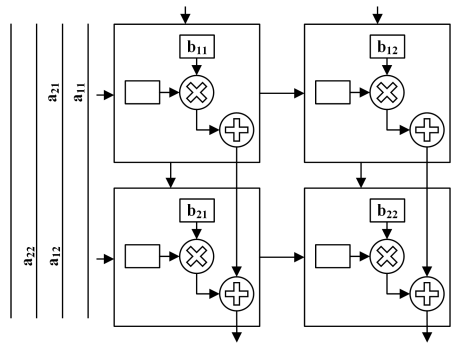

图 3 WS 数据流计算模式

Fig. 3 WS dataflow computation model

OS 的数据流计算模式如图 4 所示。进行矩阵乘法计算时，$\boldsymbol{A}$ 和 $\boldsymbol{B}$ 两个矩阵都会被传播，即 $\boldsymbol{A}$ 矩阵从左向右进行数据传播，$\boldsymbol{B}$ 矩阵从上向下进行数据传播，确保在相同的脉动阵列地址中保持部分和的累加。这种数据流计算模式中，处理元素会将乘法结果直接发送到相应的累加器中，在矩阵完全传播后，即可达到最终结果。在需要进行 $\boldsymbol{C}=\boldsymbol{A}*\boldsymbol{B}+\boldsymbol{D}$ 的计算时，矩阵 $\boldsymbol{D}$ 可以被预先加载到累加器中。这种数据流适用于连续的矩阵乘法，其中这些乘法的结果会进行累加，该模式允许中间结果被累积，直到最终结果计算完成。

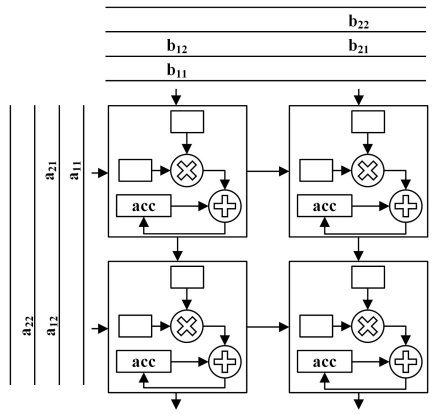

图 4 OS 数据流计算模式

Fig. 4 OS dataflow computation model

3.3 向量指令的扩展

本节将对涉及到的向量指令和特殊向量执行单元进行介绍，说明每条向量指令的作用以及对应特殊向量单元的工作原理。由于 Softmax 和 LayerNorm 计算过程复杂，需要单独进行指令扩展，执行 ReLU 和 GeLU 激活计算时，计算过程只与其元素本身相关，实现简单，只需要在配置指令中设置标志位即可在矩阵输出的累加器中完成计算，无需进行单独的指令扩展。

3.3.1 向量指令设计

Softmax 的计算式为：

$$\mathrm{Softmax}(\boldsymbol{z})_i=\frac{\exp(z_i-z_{\max})}{\sum_{j=1}^{K}\exp(z_j-z_{\max})} \tag{1}$$

根据其计算步骤，涉及到 4 条相关指令的使用，分别为 softmax. max，softmax. sum，softmax. compute 和 softmax. clear。Softmax 具体的指令字段码如图 5 所示。

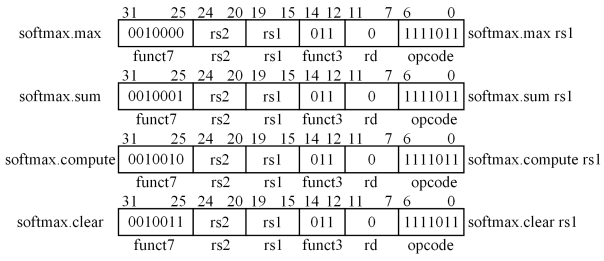

图 5 Softmax 指令字段码

Fig. 5 Softmax instruction field codes

softmax. max 用来更新最大值缓存；softmax. sum 更新分母缓存；softmax. compute 利用最大值和分母进行最终的激活计算；softmax. clear 清空所有缓存。

LayerNorm 的计算式为：

$$\mathrm{LayerNorm}(\boldsymbol{x})=\frac{\boldsymbol{x}-\mu}{\sigma}\cdot\gamma+\beta \tag{2}$$

其中，μ 为平均值，σ 为方差，γ 为权重，β 为偏置。进行 LayerNorm 计算时，涉及到 3 条相关指令，分别为 layernorm. mean. var，layernorm. compute 和 layernorm. clear。LayerNorm 指令具体的字段码信息如图 6 所示。

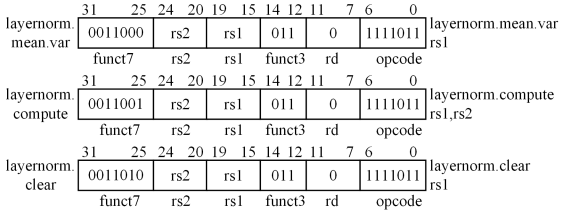

图 6 LayerNorm 指令字段码

Fig. 6 LayerNorm instruction field codes

layernorm. mean. var 用来更新平均值和方差缓存；layernorm. compute 利用平均值缓存、方差缓存、权重和预加载的偏置值计算最终结果；layernorm. clear 清空所有缓存。

3.3.2 特殊向量单元

Softmax 的计算单元设计以及计算流程如图 7 所示。Softmax 运算模块包括最大值查找单元(Max Value)、最大值缓存单元(Max Value Buffer)以及 Softmax 计算单元。最大值查找单元是对 Softmax 的计算进行优化，防止数据溢出，当完成矩阵乘法后，该单元会接收到结果矩阵的数值，得到每一行特征矩阵的最大值并暂存至最大缓存单元中。进行 Softmax 计算时，所在行的最大值从最大值缓存单元中获取，特征数据则从片上缓存或累加器中获取，这些数据可以并行发送至 Softmax 运算模块，几个周期后，可以并行得到多个 Softmax 值；每个时钟周期送入一批新的计算值，其依次流入 Softmax 运算模块，形成流水线，从而提高该模块的吞吐率。

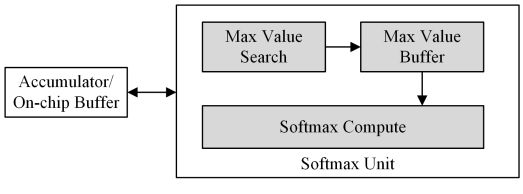

图 7 Softmax 计算单元

Fig. 7 Softmax calculation unit

LayerNorm 的计算单元设计以及计算流程如图 8 所示。LayerNorm 模块包括均值计算单元(Mean Compute)、均值缓存单元(Mean Buffer)、方差计算单元(Var Compute)、方差缓存单元(Var Buffer)及 LayerNorm 计算单元。均值计算单元用于计算脉动阵列中的输出向量的每一行元素的均值；均值缓存单元用于缓存均值；方差计算单元用于计算脉动阵列的输出向量的每一行元素的方差；方差缓存模块用于缓存方差；LayerNorm 计算单元用于从片上缓存中获取偏置矩阵缓存数据和参数缓存数据，并根据偏置矩阵缓存数据、参数缓存数据、均值及方差进行层归一化运算。

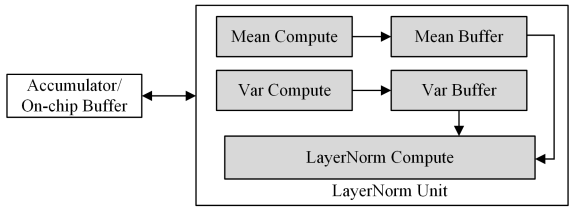

图 8 LayerNorm 计算单元

Fig. 8 LayerNorm calculation unit

3.4 访存指令的扩展

本节将对访存指令及指令中涉及到的寄存器的具体字段进行介绍，同时说明片上存储单元的存储结构。

3.4.1 访存指令设计

RISC-V 精简指令集的设计中，需要专门的指令来访问存储器，即 load 与 store 指令，其他的指令均无法访问存储器，进行矩阵数据的读写时，需要设计适配的访存指令。本文设计了 mvin 和 mout 两条指令控制数据的读写：mvin 指令负责将数据从内存搬运到加速器的片上缓存；mout 指令完成片上缓存到内存的数据搬运工作。访存类指令的具体字段信息如图 9 所示。

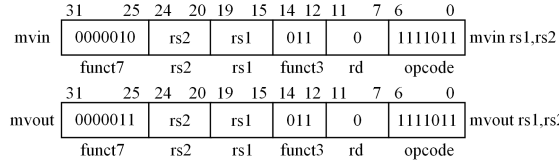

图 9 访存指令字段码

Fig. 9 Memory access instruction field codes

在访存类指令字段中，具体字段含义如下。

1) rs1 寄存器中存储虚拟内存地址。

2) rs2[31:0] 字段存储加速器片上缓存地址。

3) rs2[47:32] 字段存储加载/存储矩阵列数。

4) rs2[63:48] 字段存储加载/存储矩阵行数。

为保证数据供给平衡，缩短运算时数据的等待时间，进行访存类指令设计时，扩大数据寻址空间。访存类指令中，标量寄存器 rs1 中的信息均为虚拟内存地址，寄存器 rs2 中存储的矩阵行列数大小为 16 位，进行访存时，单条访存类指令的执行可完成较大规模的数据迁移，从而减少访存类指令的生成。

3.4.2 片上存储单元

神经网络加速器的片上存储单元包括片上缓存和累加器，这两个存储模块均可进行单独寻址。通常，脉动阵列的输

入数据存储在片上缓存，输出数据即部分和及最终结果存储在累加器中。片上缓存的结构如图 10 所示。

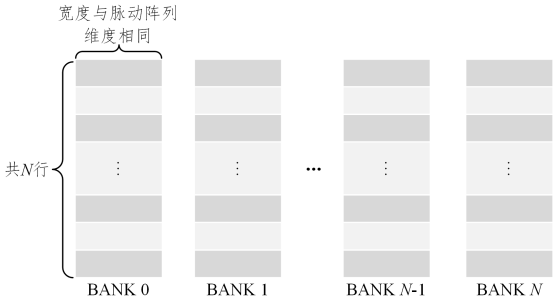

图 10 片上缓存结构示意图

Fig. 10 Diagram of on-chip buffer structure

片上缓存采用了多 BANK 的结构，允许多个存储单元同时被访问。多 BANK 结构显著提高了数据的读取速度，减少了访问延迟。此外，多 BANK 架构还提供了灵活性，能够根据需求优化布局，易于扩展，适应更高的容量和性能存储需求。每个 BANK 的设计，其宽度与脉动阵列大小相匹配，若脉动阵列大小为 DIM，则 BANK 的每一行都可以存储 DIM 个数据元素。

累加器的设计稍微复杂，底层数据存储与片上缓存保持一致，其宽度与片上缓存一致，均与脉动阵列相匹配；在功能上，累加器包含了一组加法器来支持原地累加。此外，还存在一组缩放器以及 ReLU 和 GeLU 的非线性计算单元。

4 加速器架构周期精确模拟仿真

本文使用 Gem5 体系结构模拟器进行了周期精确的模拟仿真。模拟仿真时使用 Ticked Object 组件，此组件在每个时钟周期触发特定操作，以确保模拟器能够以高精度的时间管理来调度各个硬件组件的操作。此机制不仅增强了模拟的真实性和可靠性，还为研究人员提供了灵活性，以便于实现复杂的硬件特性并进行深入的体系结构研究。

在进行加速器架构周期精确的模拟仿真时，本文将加速器架构中所涉及的各个组件视为独立对象，进行模拟仿真时，允许对象在每个时钟周期(Tick)执行特定操作。每个对象可以注册一个 tick 方法，该方法在每个 tick 被调用，以执行状态更新和事件处理。这种设计通过事件调度系统确保对象在正确的时间点触发事件，从而反映硬件的真实时间行为。

加速器架构的周期精确的模拟仿真实现在效果上是将模拟的 C++代码的顺序调用转化为了与 verilog[29] 一样的代码并行执行，或者类似高级语言的并发编程。在仿真实现的 ticked 基类代码中，存在一个每个时钟周期均会被触发的“事件”，“事件内容”则是调用一次 evaluate()函数。根据状态信号来源的差异，存在两种状态管理实现的 evalate()函数的代码逻辑。当 state 信号来自模块内部时，evaluate()函数逻辑则是实现了一个状态机，state 作为模块内部的逻辑信号，允许在每个时钟上升沿根据当前状态进行更新和控制，这种调用方式适合需要自主状态管理的应用；当 state 信号来自外部时，state 作为输入信号，模块的行为完全依赖于外部输入，无法自主更新状态，这种设计适合响应外部信号的场景。

对加速器架构中各单元进行周期精确的模拟仿真时，采用函数内部维护 state 变量的方法自主管理状态机，此方式能够更贴近硬件的顺序逻辑行为。该方法在每个时钟周期中自主更新内部状态，从而模拟数据在各单元之间按照时序推进的计算过程，避免外部控制导致的数据流不一致的问题。此状态机制是构建周期精确仿真的有效手段。

使用 Gem5 模拟器进行架构仿真时，对神经网络加速器中涉及到的各单元，如脉动阵列、特殊向量单元等，进行单独函数类的建立并编写功能代码，同时在每个类中正确调用 evaluate()函数，以确保功能按照时序进行，从而保证周期精确的架构模拟仿真。

脉动阵列的模拟仿真实现如算法 1 所示。

算法 1 周期精确的脉动阵列仿真实现

输入：两组输入数据，分别来自上方端口和左侧端口

输出：在 Systolic 网络中逐周期推进的数据处理结果

1. 初始化 state 为 0
2. 每个仿真周期调用 evaluate()函数，执行如下步骤：
3. 若 state == 1(数据注入阶段)：
   3.1. 若检测到左侧有新输入数据，则将其压入左侧延迟端口
   3.2. 若检测到上方有新输入数据，则将其压入上方延迟端口
   3.3. 从上方延迟端口读取一行新数据 B_new，并注入网络最上层各处理单元的 B 输入
   3.4. 从左侧延迟端口读取一列新数据 A_new，并注入网络最左侧各处理单元的 A 输入
   3.5. 状态更新：state ← 0
4. 否则(state == 0，数据流动阶段)：
   4.1. 自底向上将每列中上层单元的 B 数据下移一层
   4.2. 自右向左将每行中左侧单元的 A 数据右移一格
   4.3. 状态更新：state ← 1
5. 重复步骤 2－步骤 4，直至完成所有计算周期。

特殊向量单元的模拟仿真如算法 2 所示。

算法 2 周期精确特殊向量单元仿真实现

输入：包含 N 个元素的向量 X，控制标志(get_max，get_sum，get_ans，get_var，getmean)

输出：Softmax 和 LayerNorm 计算后的结果向量

1. 初始化 state 为 0，初始化计算器 cnt 为 0
2. 每个仿真周期调用 evaluate()函数，执行如下步骤：
3. 若 state == 1(初始化阶段)：
   3.1. 将 vector_state ← COMPUTING
   3.2. 将临时参数转入工作参数：
      -size_n ← tmp _size_n
      -X← tmp_X
      -get_max←tmp_get_max，get_sum←tmp_get_sum，get_ans←tmp_get_ans，get_var←tmp_get_var，get_mean←tmp_get_mean
   3.3. 初始化迭代控制器：
      -current_X_line ← 0
   3.4. 初始化结果向量 v 为全 0
4. vector_state == COMPUTING(计算阶段)：
   4.1. 若 cnt < size，执行以下步骤：
      -进行最大值、累加计算；进行平均值、方差计算
      -更新计数器：cnt++
   4.2. 否则(当前列处理完成)：

-清零 cnt

-vector_state ←0，并设置 is_done ← true

-清空所有缓存

5. 重复步骤 3、步骤 4，直至完成所有向量计算。

5 实验及结果

Taurus 神经网络加速器在架构和扩展指令设计方面，均在 Gemmini 开源加速器上进行了改进。对 Taurus 进行了性能测试，在实验时所采用的 RISC-V 内存层次结构、Taurus 架构中片上存储空间、脉动阵列大小均与 Gemmini 的默认配置保持一致。本文涉及的相关实验数据均来自于模拟仿真平台，Taurus 加速器的相关实验数据来自于本文提出的基于 Gem5 的周期精确的模拟仿真平台，Gemmni 实验数据来自其设计团队提供的基于 QEMU 的模拟仿真平台，实验结果对比均采用周期数对比方法。

本文以 $\boldsymbol{C}=\boldsymbol{A}*\boldsymbol{B}+\boldsymbol{D}$ 矩阵运算为例，其中参与运算的矩阵维度相等，在 Taurus 和 Gemmini 加速器上测试不同维度大小的矩阵乘累加计算，并进行了扩展指令生成条数和脉动阵列利用率的对比。

指令改进前后生成的扩展指令条数如表 1 所列，随着矩阵维度的增大，Gemmini 产生的指令条数与改进后指令条数的差值逐渐增大，在矩阵维度为 1 024 时，二者指令条数相差上百倍。

表 1 不同维度矩阵运算时，Gemmini 扩展指令与本文改进后的指令生成数量的对比

Table 1 Comparison of Gemmini extension instructions and the number of generated instructions with the proposed improved instructions for matrix operations in different dimensions

矩阵维度	Gemmini 扩展指令生成数量	改进后的扩展指令生成数量
64 * 64	158	40
128 * 128	1 125	40
256 * 256	8 716	149
512 * 512	68 643	566
1 024 * 1 024	544 900	5 151

脉动阵列利用率结果如图 11 所示。可以看出，Gemmini 加速器在进行小矩阵运算时利用率较低，随着矩阵维度的增大，脉动阵列的利用率增大，维持在 50% 左右。本文设计的 Taurus 加速器在计算相同维度矩阵乘法时，利用率均在 90% 左右，并随着矩阵维度的增大，利用率可达 92%，相对于 Gemmini，利用率提升了近 80%。

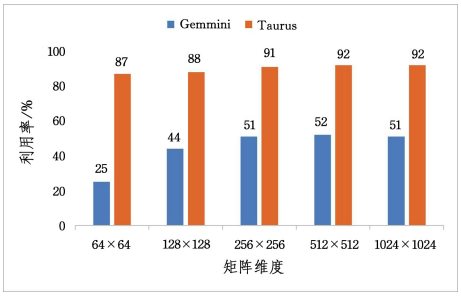

图 11 脉动阵列利用率的对比

Fig. 11 Comparison of utilization rates of systolic arrays

本文在基线 RISC-V 核心、Gemmini 及本文所设计的 Taurus 加速器上进行了 ResNet50 模型和 BERT 模型的推理，结果如图 12 所示，纵坐标为 RISC-V 与不同加速器组合相对于 RISC-V 的加速倍数。RISC-V 与 Gemmini 的组合使用，在 ResNet50 和 BERT 模型推理上分别获得1 130倍和 144 倍的加速；RISC-V 与 Taurus 的组合使用，在 ResNet50 和 BERT 模型推理中分别获得1 467倍和4 513倍的加速。根据 Gemmini 发表的文章中的数据，在 ResNet50 模型推理速度为 22.8 fps 时，使用本文设计的 Taurus 加速器，推理速度可达 29.6 fps，相对于 Gemmini 获得了 1.3 倍的加速；在 BERT 模型推理中，本文设计的 Taurus 加速器相对于 Gemmini 获得了 31.3 倍的加速。由此可知，Taurus 加速器架构能够更快加速 Transformer 模型的推理，架构设计合理。

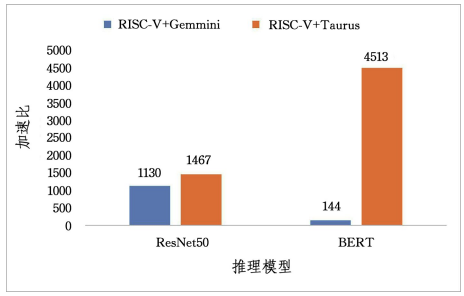

图 12 各模型在不同加速器组合上推理速度相较于 RISC-V 的加速比

Fig. 12 Inference speed acceleration ratios of various models on different accelerator combinations compared to RISC-V

结束语 针对神经网络模型的计算特点，本文基于 RISC-V 指令扩展设计了 Taurus 神经网络加速器。为满足计算的需要，进行了矩阵指令、向量指令及访存指令的扩展。进行指令扩展时使用标量寄存器复用的方式，将运算所需信息存入标量寄存器中，扩大了数据寻址空间，以支持大规模矩阵运算。最后，使用 Gem5 体系结构模拟仿真器进行周期精确的模拟仿真，与开源加速器 Gemmini 相比，脉动阵列利用率提高 80%。在 ResNet50 和 BERT 模型推理中，与 Gemmini 相比，性能分别获得 1.3 倍和 31.3 倍的加速；与 RISC-V 相比，性能分别获得 1467 倍和 4513 倍的加速。本文所设计的神经网络加速器架构在模拟评估时采用了参数化的设计，并未进行设计空间探索，之后的工作中将对参数化形成的设计空间进行探索以找到最优配置。

参 考 文 献

[1] YOU H，SUN Z，SHI H，et al. Vitcod：Vision transformer acceleration via dedicated algorithm and accelerator co-design[C]// 2023 IEEE International Symposium on High-Performance Computer Architecture(HPCA). IEEE，2023：273-286.

[2] WANG T，GONG L，WANG C，et al. Via：A novel vision-transformer accelerator based on fpga[J]. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems，2022，41(11)：4088-4099.

[3] JOUPPI N P，YOUNG C，PATIL N，et al. In-datacenter performance analysis of a tensor processing unit[C]// Proceedings

of the 44th Annual International Symposium on Computer Architecture. 2017: 1-12.
[4] LIANG X Y. Ascend AI Processor Architecture and Programming: In-Depth Understanding of CANN Technology Principles and Applications [M]. Beijing: Tsinghua University Press, 2019.
[5] LIU M Y, LI C H, LIN C Y, et al. Matrix Accelerator Designed for Vision Transformer [C]// 2024 IEEE International Conference on Consumer Electronics-Asia (ICCE-Asia). IEEE, 2024: 1-2.
[6] KIM S, HOOPER C, WATTANAWONGT, et al. Full stack optimization of transformer inference: a survey [J]. arXiv: 2302.14017, 2023.
[7] CUI E, LI T, WEI Q. Risc-v instruction set architecture extensions: A survey [J]. IEEE Access, 2023, 11: 24696-24711.
[8] CAMMARATA D, PEROTTI M, BERTULETTI M, et al. Quadrilatero: A RISC-V programmable matrix coprocessor for low-power edge applications [J]. arXiv: 2504.07565, 2025.
[9] PUROHIT Y, PAREEK D, SAVANI V. Development of a System on Chip (SoC) for Matrix Multiplication Utilizing RISC-V and Vector Processor [C]// International Conference on Sustainable and Innovative Solutions for Current Challenges in Engineering & Technology. Springer, 2025: 1-12.
[10] JIAO Q, HU W, LIU F, et al. Risc-vtf: Risc-v based extended instruction set for transformer [C] // 2021 IEEE International Conference on Systems, Man, and Cybernetics (SMC). IEEE, 2021: 1565-1570.
[11] BUTKO A, GARIBOTTI R, OST L, et al. Accuracy evaluation of gem5 simulator system [C]// 7th International Workshop on Reconfigurable and Communication-centric Systems-on-chip (ReCoSoC). IEEE, 2012: 1-7.
[12] LOWE-POWER J, AHMAD A M, AKRAM A, et al. The gem5 simulator: Version 20.0+ [J]. arXiv: 2007.03152, 2020.
[13] SHAO Y S, XI S L, SRINIVASAN V, et al. Co-designing accelerators and SoC interfaces using gem5-Aladdin [C]// 2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO). IEEE, 2016: 1-12.
[14] ROGERS S, SLYCORD J, BAHARANI M, et al. gem5-salam: A system architecture for llvm-based accelerator modeling [C]// 2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO). IEEE, 2020: 471-482.
[15] VIEIRA J, ROMA N, FALCAO G, et al. Gem5-accel: A pre-RTL simulation toolchain for accelerator architecture validation [J]. IEEE Computer Architecture Letters, 2023, 23(1): 1-4.
[16] FEIST T. Vivado design suite [Z]. White Paper, 2012: 24.
[17] GENC H, KIM S, AMID A, et al. Gemmini: Enabling systematic deep-learning architecture evaluation via full-stack integration [C]// 2021 58th ACM/IEEE Design Automation Conference (DAC). IEEE, 2021: 769-774.
[18] CAVALCANTE M, SCHUIKI F, ZARUBAF, et al. Ara: A 1-GHz+ scalable and energy-efficient RISC-V vector processor with multiprecision floating-point support in 22-nm FD-SOI [J]. IEEE Transactions on Very Large Scale Integration Systems, 2019, 28(2): 530-543.
[19] GAURAV T, BHATT A, PAREKH R. Design and Implementation of low power RISC V ISA based coprocessor design for Matrix multiplication [C]// 2021 Second International Conference on Electronics and Sustainable Communication Systems (ICESC). IEEE, 2021: 189-195.
[20] TAI H Y. Enhanced RISC-V Matrix Extension Architecture [D]. Taiwan: National Yang Ming Chiao Tung University, 2023.
[21] YI X, ANTONIO R, DUMOULIN J, et al. OpenGeMM: A High-Utilization GeMM Accelerator Generator with Lightweight RISC-V Control and Tight Memory Coupling [J]. arXiv: 2411.09543, 2024.
[22] Working draft of the proposed RISC-V V vector extension [EB/OL]. https://github.com/riscv/riscv-v-spec.
[23] CHEN C, XIANG X, LIU C, et al. Xuantie-910: A commercial multi-core 12-stage pipeline out-of-order 64-bit high performance RISC-V processor with vector extension: Industrial product [C]// 2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA). IEEE, 2020: 52-64.
[24] KIRAN D C, GURUNARAYANAN S, MISRAJ P, et al. Register allocation for fine grain threads on multicore processor [J]. Journal of King Saud University-Computer and Information Sciences, 2017, 29(1): 85-92.
[25] PALA D. Design and programming of a coprocessor for a RISC-V architecture [D]. Torino: Politecnico di Torino, 2017.
[26] WATERMAN A, LEE Y, PATTERSON D A, et al. The RISC-V instruction set manual, volume I: User-level ISA, version 2.0: Tech. Rep.: UCB/EECS-2014-54 [R]. EECS Department, University of California, Berkeley, 2014: 4.
[27] SZE V, CHEN Y H, YANG T J, et al. Efficient processing of deep neural networks: A tutorial and survey [C]// Proceedings of the IEEE. 2017: 2295-2329.
[28] CAPRA M, BUSSOLINO B, MARCHISIO A, et al. Hardware and software optimizations for accelerating deep neural networks: Survey of current trends, challenges, and the road ahead [J]. IEEE Access, 2020, 8: 225134-225180.
[29] THOMASD, MOORBY P. The Verilog © hardware description language [M]. Springer Science & Business Media, 2008.

CAI Chenghuan, born in 1999, postgraduate. His main research interest is in domain-specific hardware-software co-design.

ZHANG Fengzhe, born in 1982, Ph.D, associate professor, Ph.D supervisor, is a member of CCF (No. 21012M). His main research interests include computer architecture and system software, and brain-inspired computing.

(责任编辑:李亚辉)