

寒武紀の MLU03芯片架构



ZOMI



Talk Overview

I. AI 计算体系

- 深度学习计算模式
- 计算体系与矩阵运算

2. AI 芯片基础

- 通用处理器 CPU
- 从数据看 CPU 计算
- 通用图形处理器 GPU
- AI 专用处理器 NPU/TPU
- 计算体系架构的黄金10年

I. 华为昇腾 NPU

- 达芬奇架构
- 昇腾AI处理器

2. 谷歌 TPU

- TPU 核心脉动阵列
- TPU 系列架构

3. 特斯拉 DOJO

- DOJO 架构

4. 国内外其他AI芯片

- AI芯片的思考

Talk Overview

I. 国内其他 AI 芯片

- 壁仞 芯片剖析
- 寒武纪 芯片剖析
- 麟原科技 芯片剖析
- AI 芯片架构的思考

Talk Overview

I. 国内其他 AI 芯片

- 壁仞 芯片剖析
 - 寒武纪 芯片剖析
 - 麟原科技 芯片剖析
 - AI 芯片架构的思考
- 
- 寒武纪的产品形态
 - 寒武纪MLU03芯片架构
 - 寒武纪软件栈和通信

I. 架构发展



历经5代AI芯片架构



智能加速IP核

- 麒麟980
- 麒麟970

多核架构

- 思元100

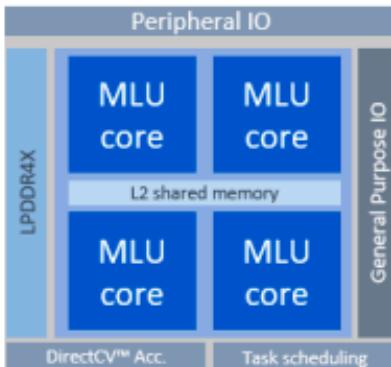
多核共享片内存储

- 思元220
- 思元270
- 思元290

多芯粒架构

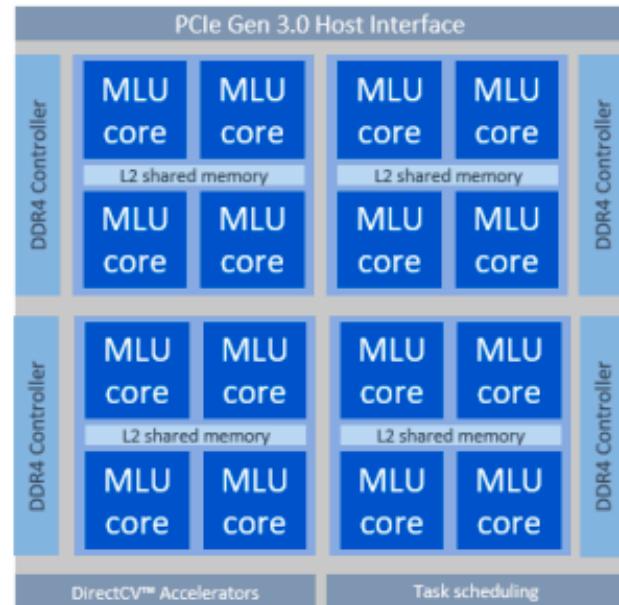
- 思元370

整体架构



MLU220

6MB SRAM
LPDDR4x



MLU270

32MB SRAM
DDR4

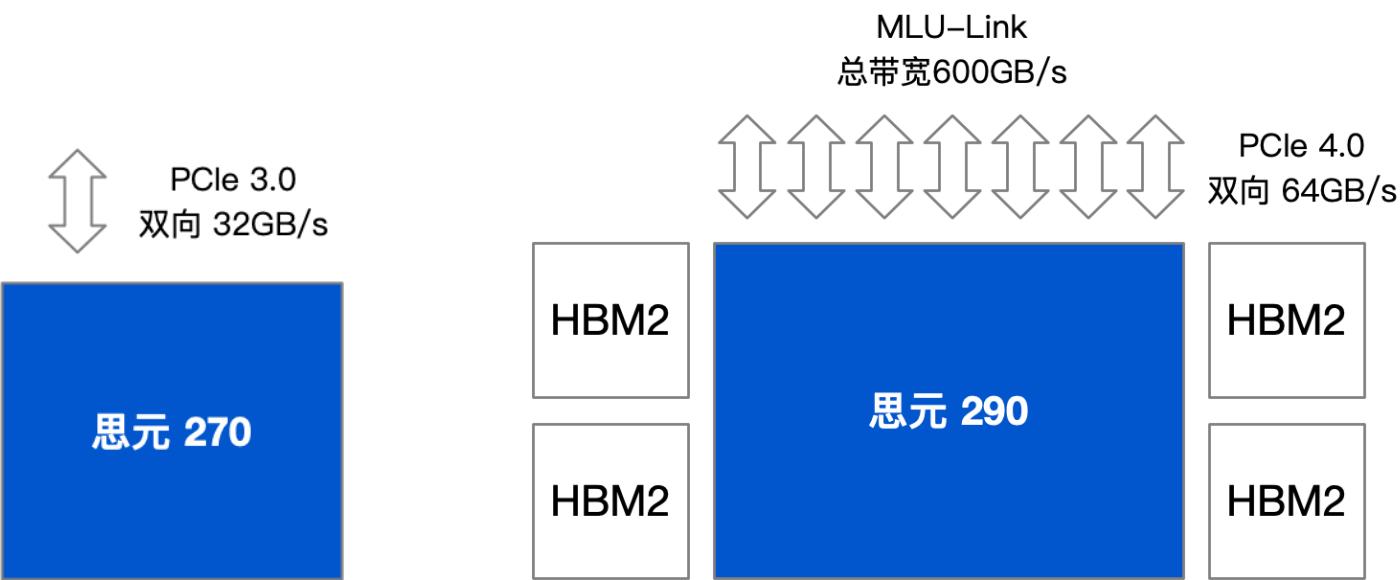


MLU290

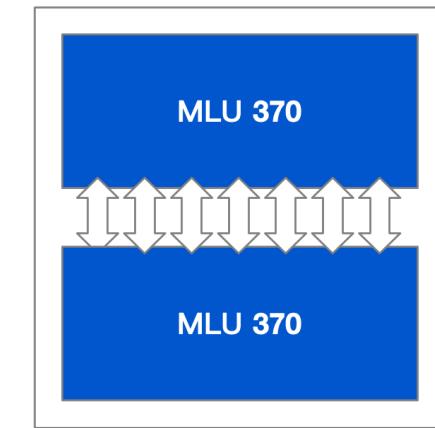
96MB SRAM
HBM2
MLU-Link™



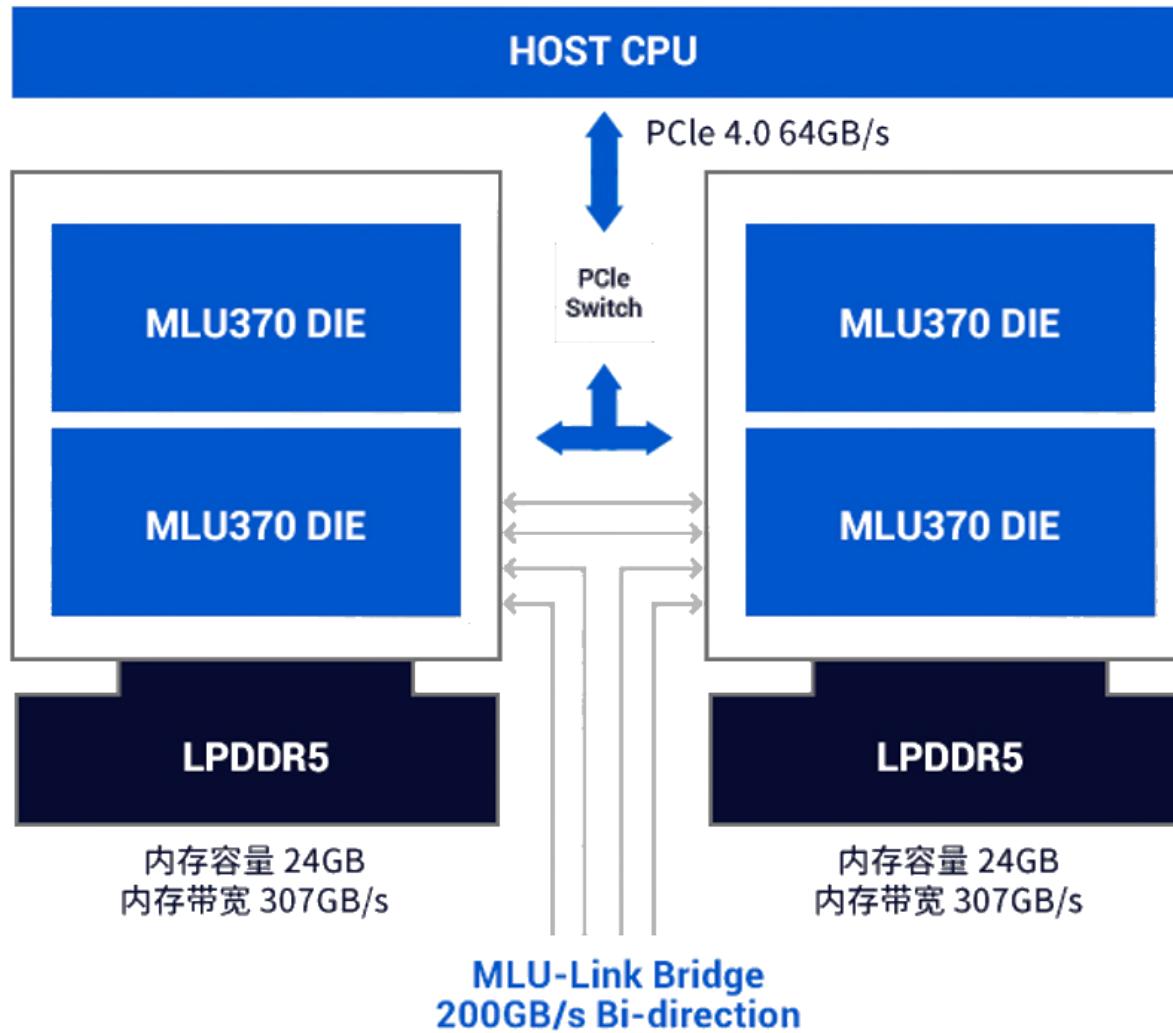
芯颗封装



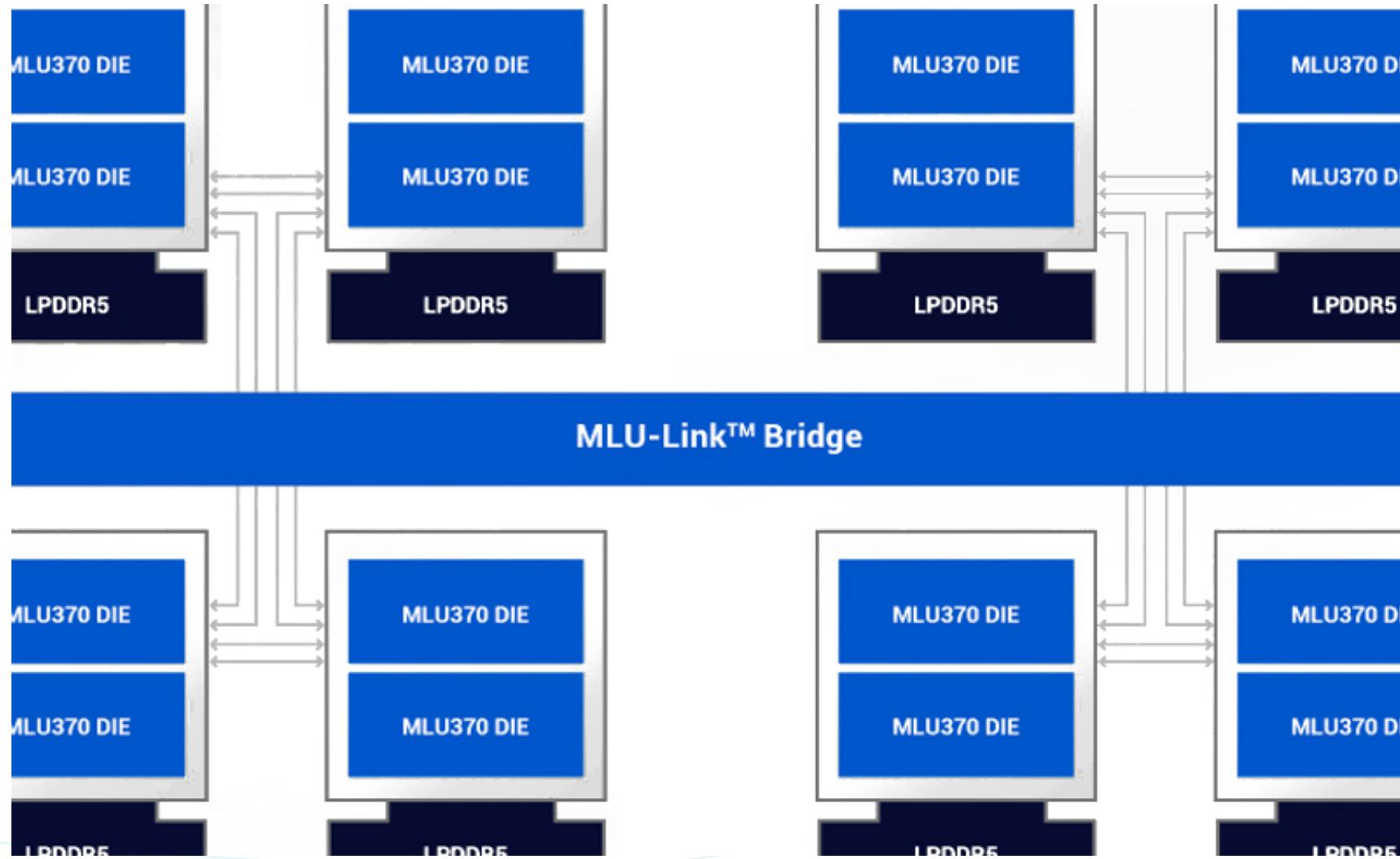
- 2颗DIE封装
- 全局UMA内存访问
- 低功耗、低时延、高带宽



产品形态 I



产品形态 II



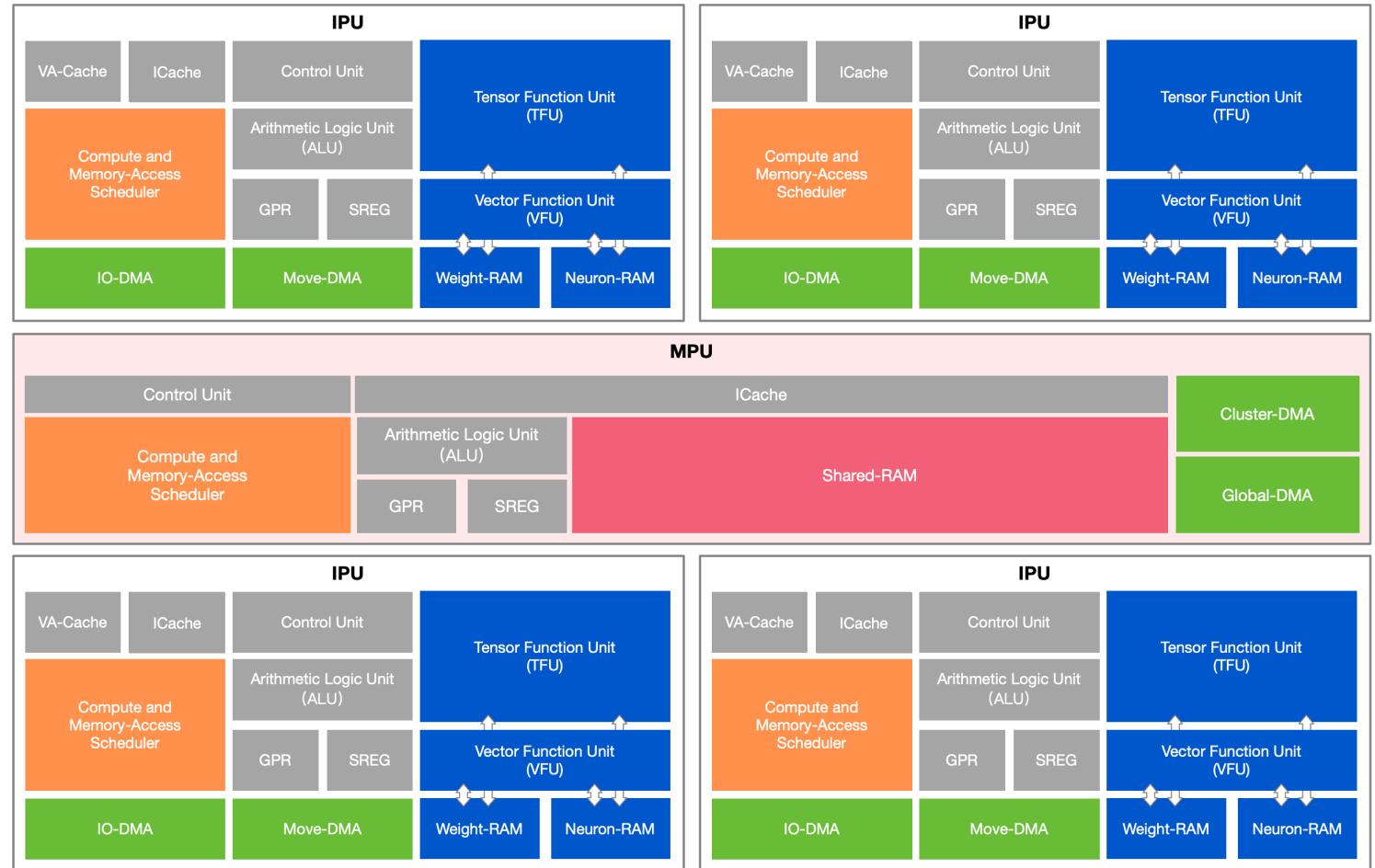
2. MLU03

整体架构



MLUv03 - Cluster

- IPU : Intelligence , 作为 MLU 核心模块，负责 AI 应用的计算、访存和控制指令执行。
- MPU : Memory , MLU中负责片上 Shared-RAM相关协处理器核心，对数据进行处理。
- Cluster : NLUv03架构 , 4个IPU和1个MPU作为核心构成1个Cluster。多个Cluster组合不同产品形态。



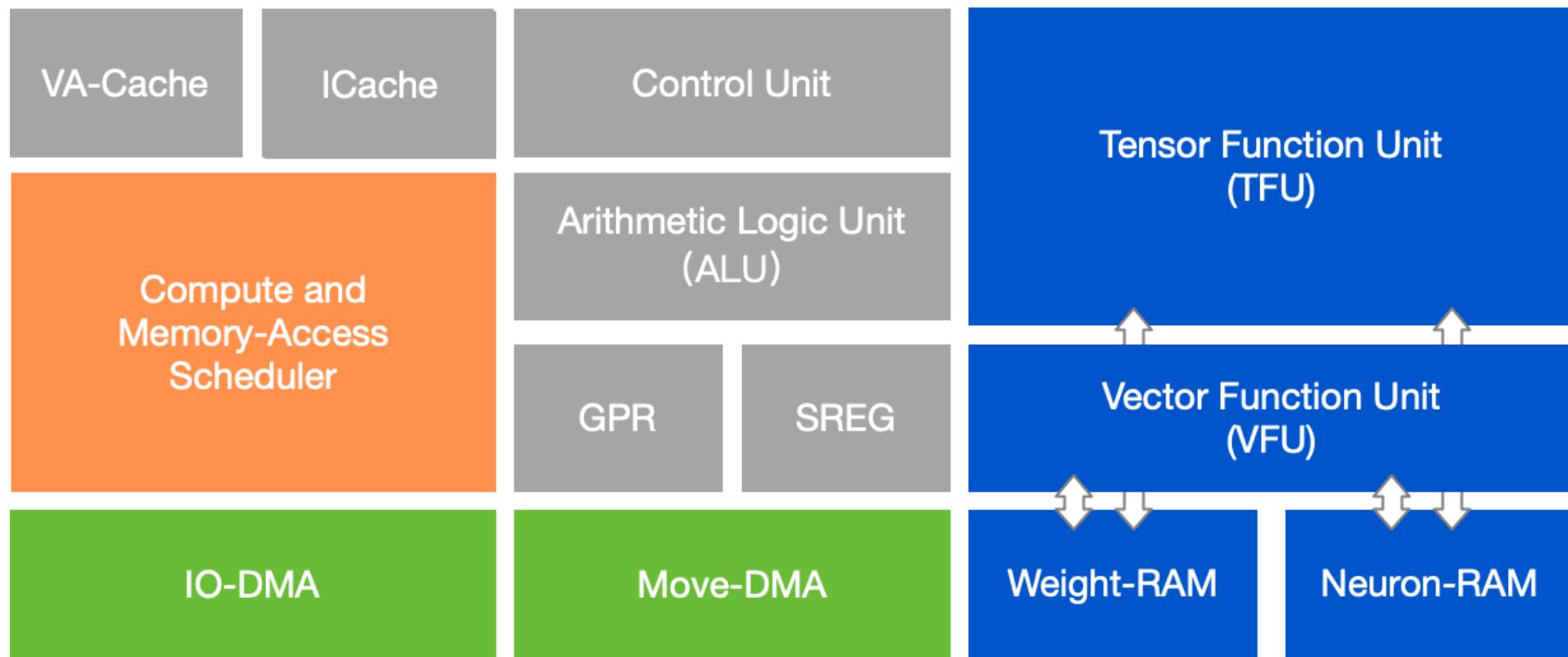
3. MLU03-IPU

核心



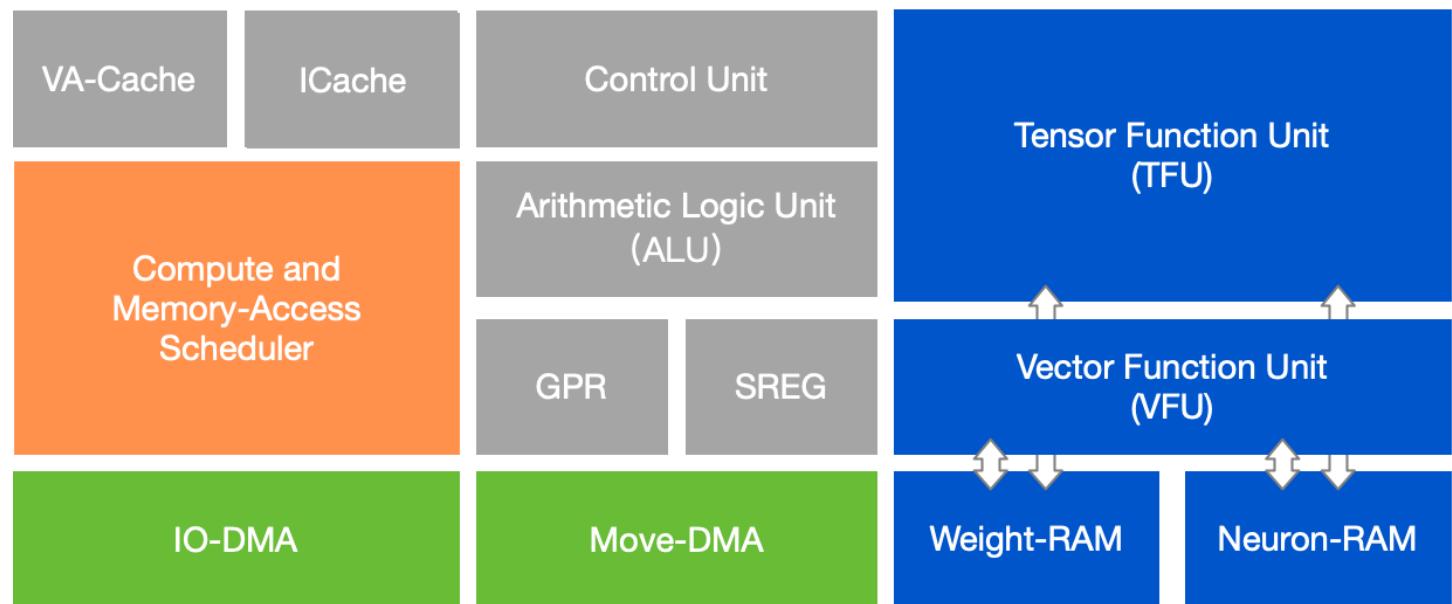
MLUv03 – IPU 核心模块

执行指令控制 > 数据搬运 > 具体 AI 计算



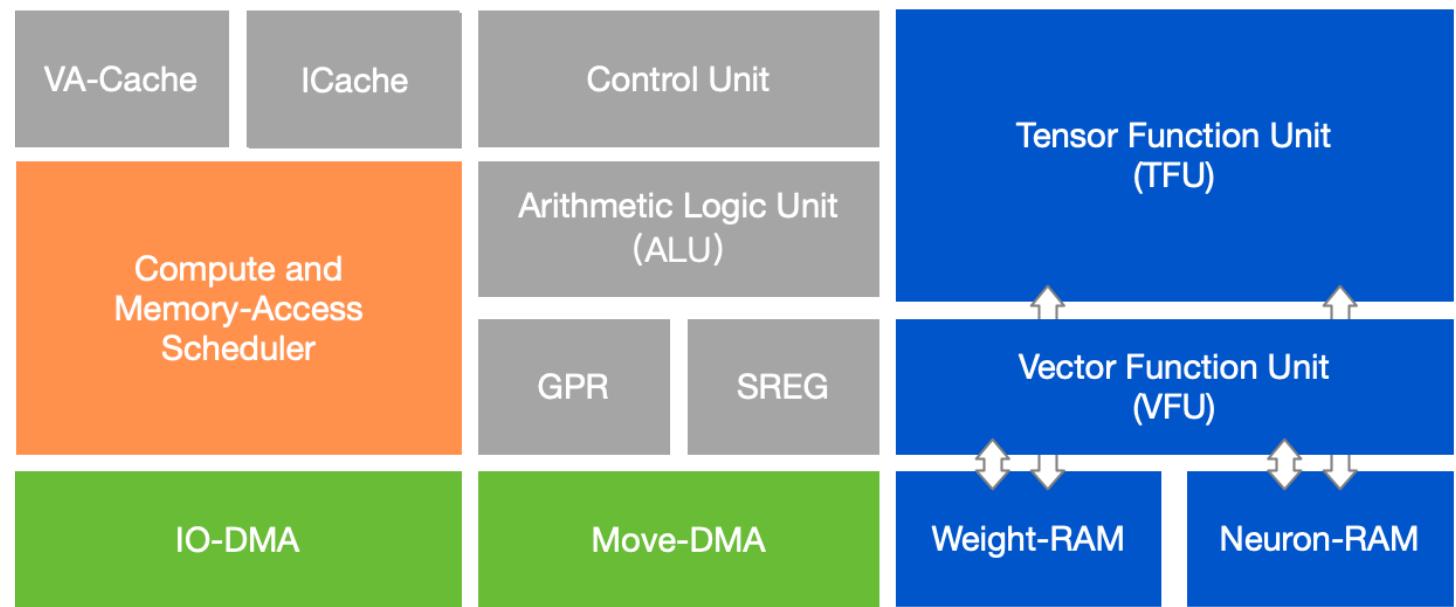
MLUv03 – IPU 核心模块

- **Control Unit**：负责指令读取，译码和发射 Dispatch。
- **指令发射**：自研指令可以 Control Unit 被负责计算和访存的调度器 Dispatch 到 ALU、VFU、TFU、IO-DMA、Move-DMA 四个队列。



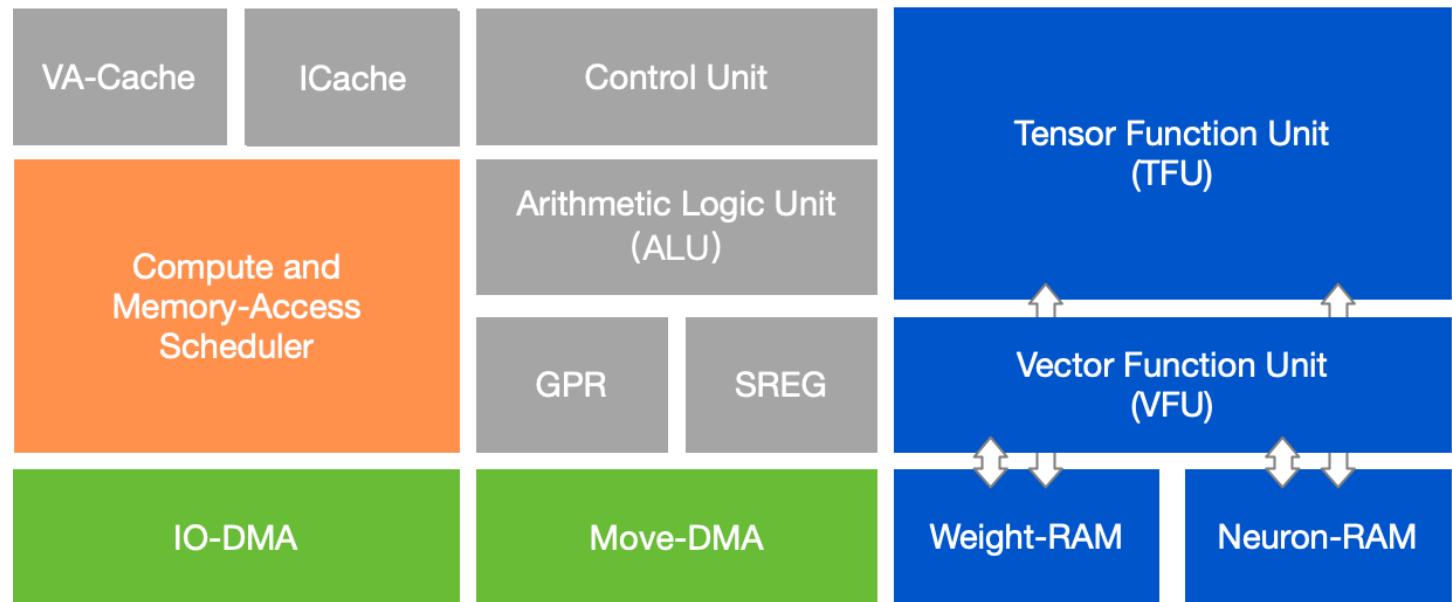
MLUv03 – IPU 核心模块

- **IO-DMA**：实现片外DRAM与W/N-RAM数据传输，也可以用于实现Register与片内RAM之间的Load/Store操作以及原子操作。
- **Move-DMA**：IPU中W/N-RAM与MPU S-RAM间数据传输和类型转换。



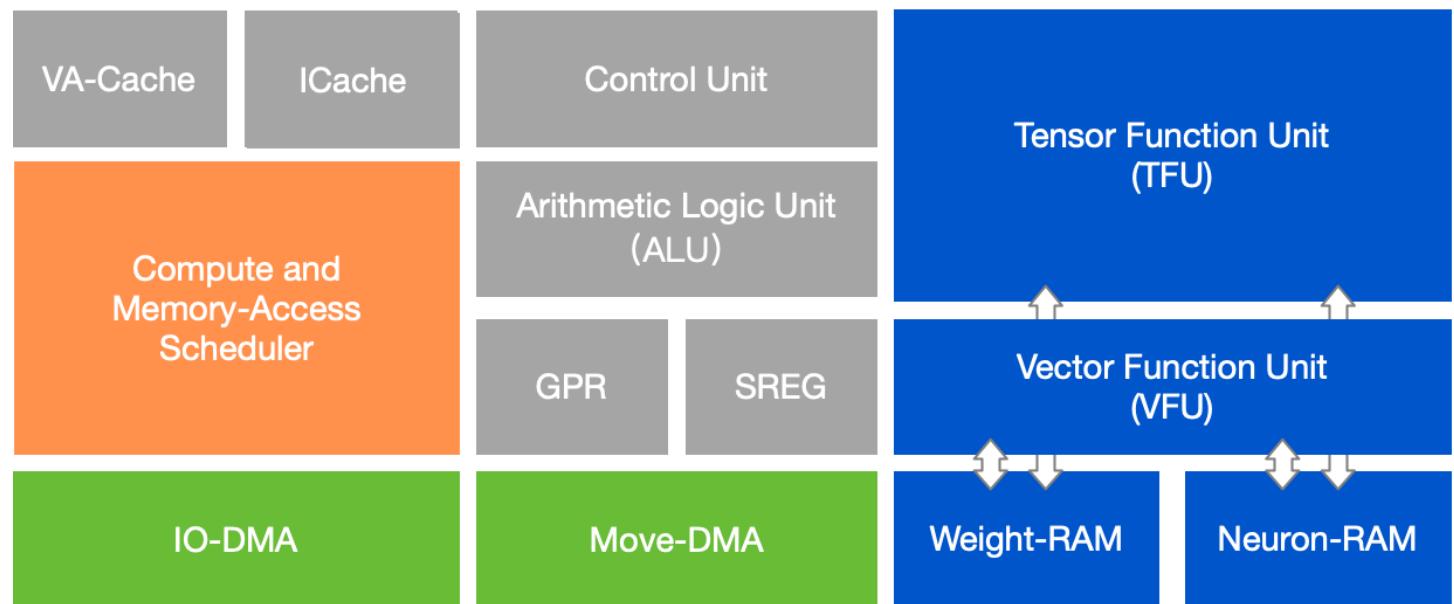
MLUv03 – IPU 核心模块

- **I-Cache** : 指令缓存 , 64KB , 如512bit指令可以缓存1024条。
- **VA-Cache**(Vector Addressing) : 离散数据访问指令的专用缓存。用于加速离散向量访问效率 , 减少对总线和存储单元读写次数。



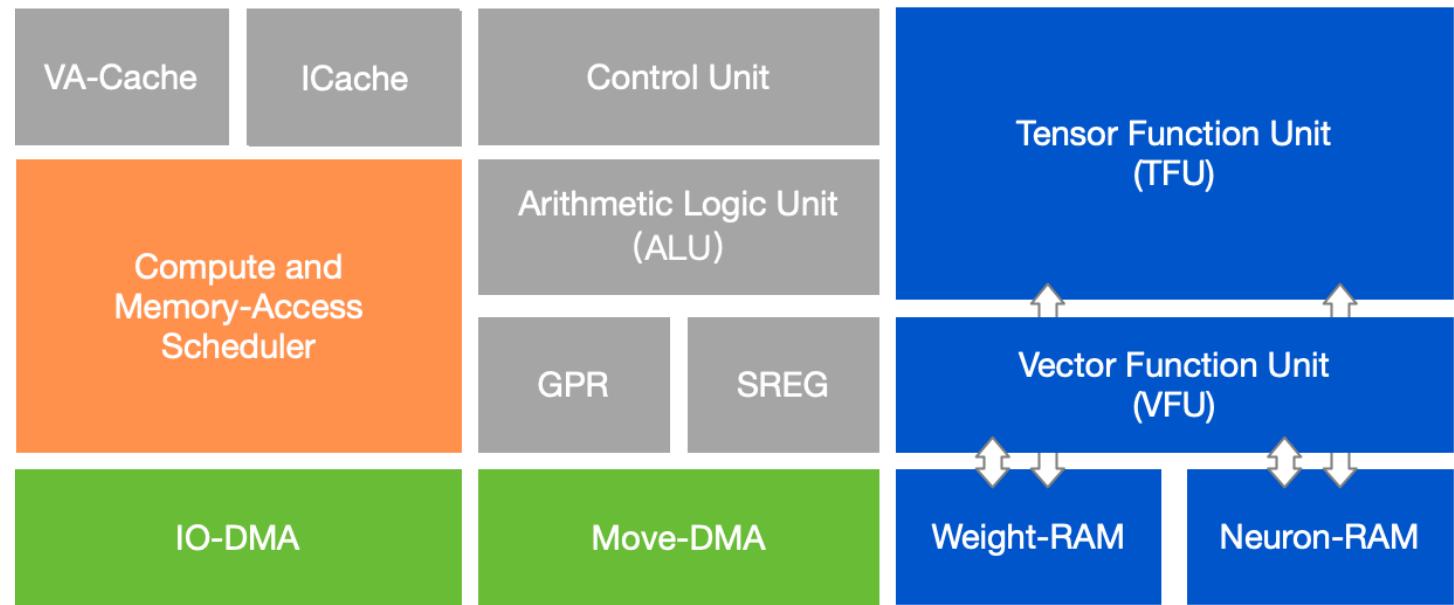
MLUv03 – IPU 存储

- **Neural-RAM** : 768KB , 需16byte对齐，存储Input和Feature Map数据。
- **Weight-RAM** : 1024KB , 需8byte对齐，存储权重和优化器数据。



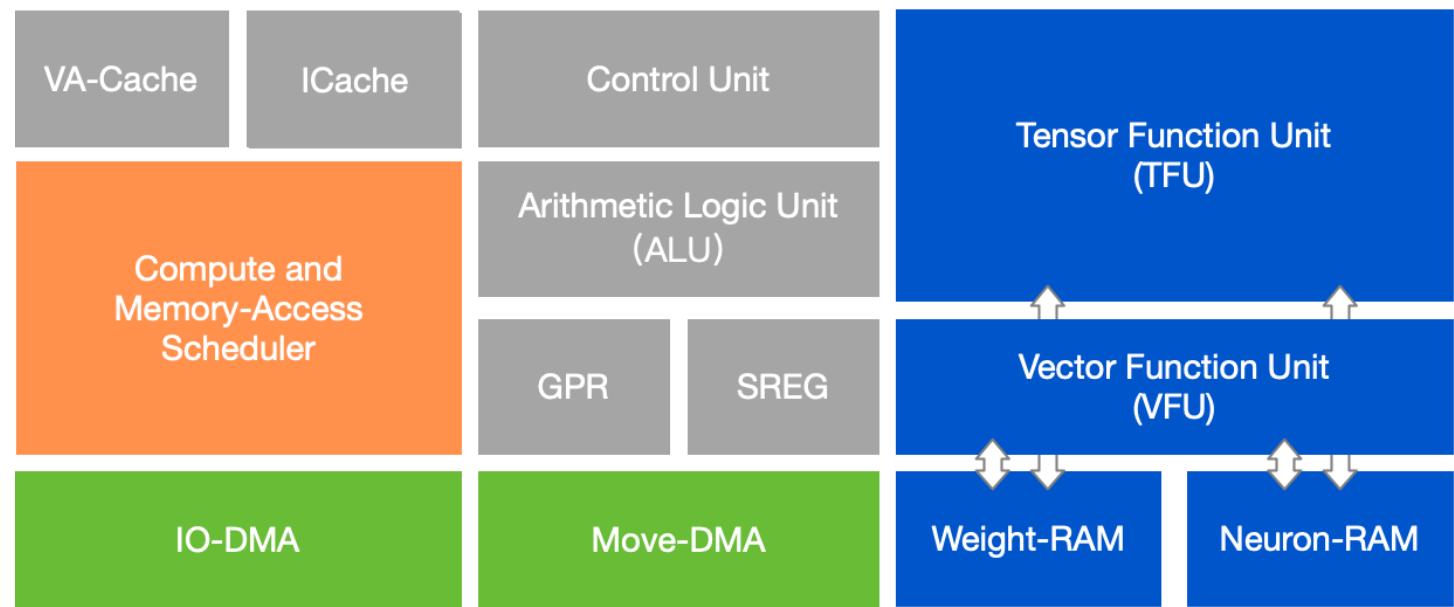
MLUv03 – IPU 核心模块

- **ALU**：标量 Scale 数据的算术逻辑运算。
- **GPR**：位宽48bit，IPU为Load/Store架构，除立即数以外所有操作加载到GPR才能参与算术逻辑运算。
- **SREG**：~位宽32bit，用于存储硬件特定属性，如Perf计数、任务索引等。



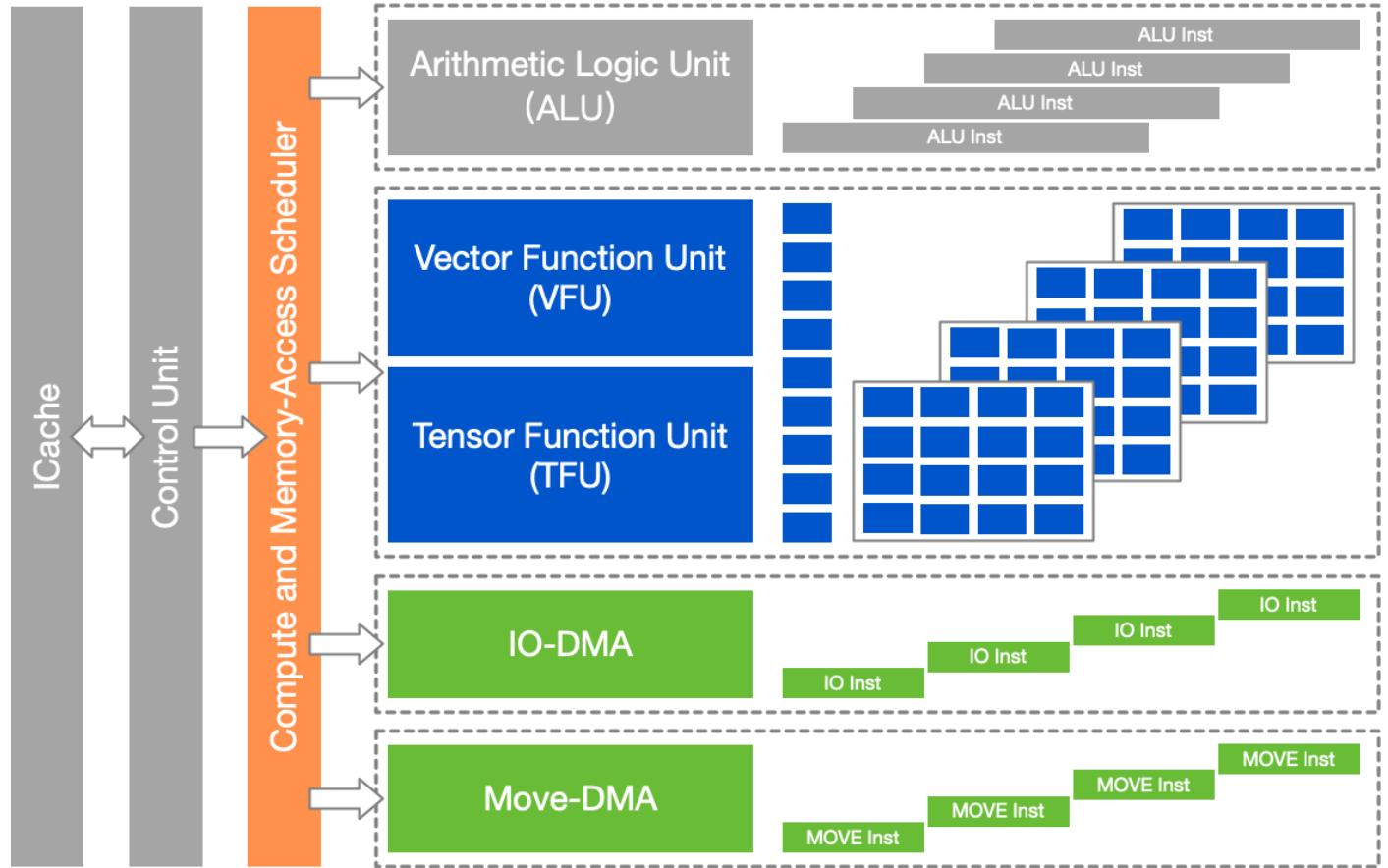
MLUv03 – IPU 核心模块

- **VFU/TFU** : 实现张量 Tensor 和向量 Vector 运算；
- **输入输出** : Vector 运算输入输出在Neuron-RAM ; Tensor 运算输入自Neuron-RAM和Weight-RAM , 输出根据调度情况到Neuron-RAM和Weight-RAM。



MLUv03 – IPU 指令流水

- **指令流水**：每个IPU计算核心包括三个指令队列，XFU-PIPE、 DMA-PIPE、 ALU-PIPE。
- **XFU-PIPE**：执行向量和张量单元指令。
- **DMA-PIPE**：支持双流同时进行数据搬运执行。
- **ALU-PIPE**：执行标量数据算术逻辑指令。



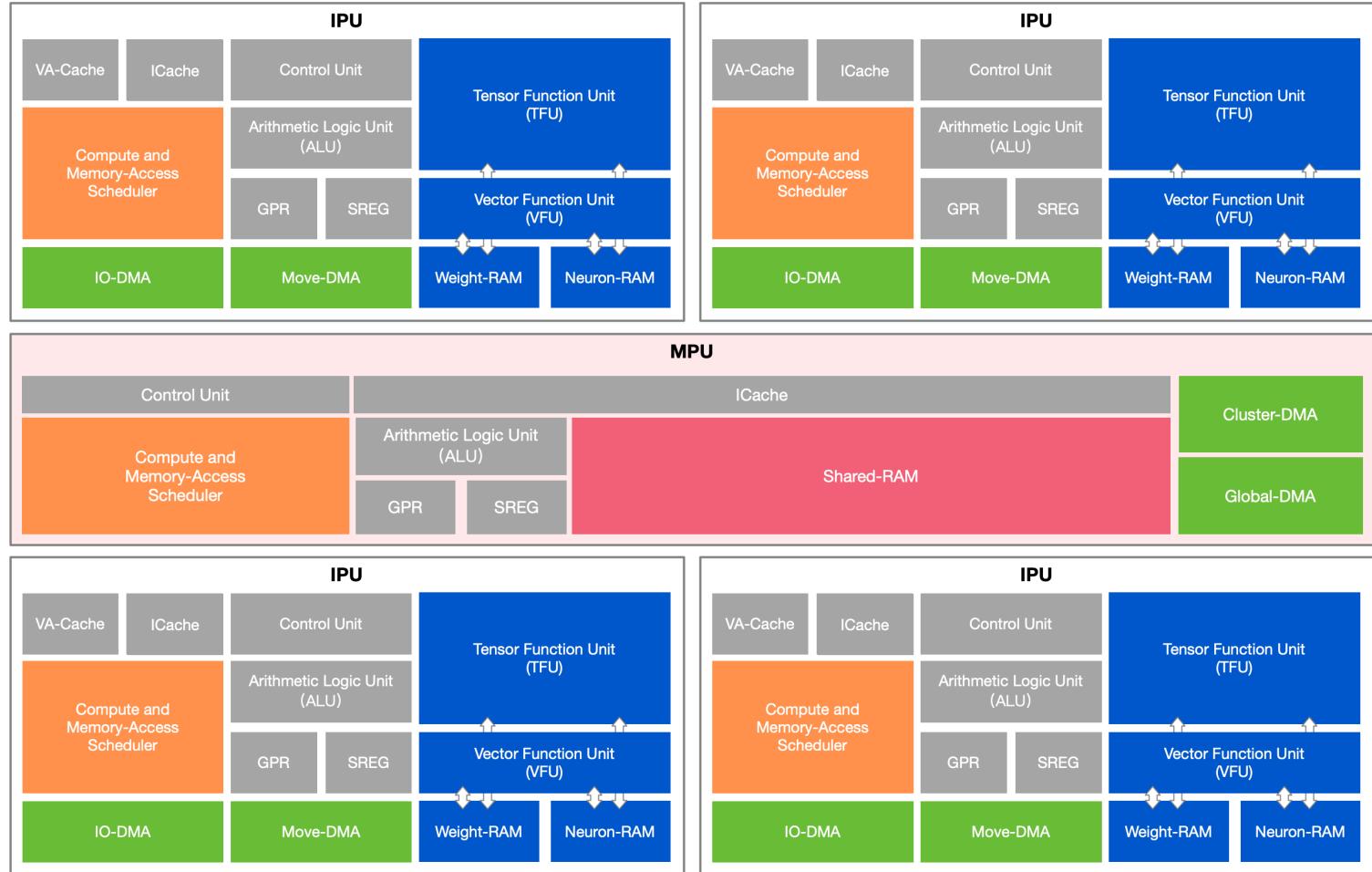
4. MLU03-MPU

核心



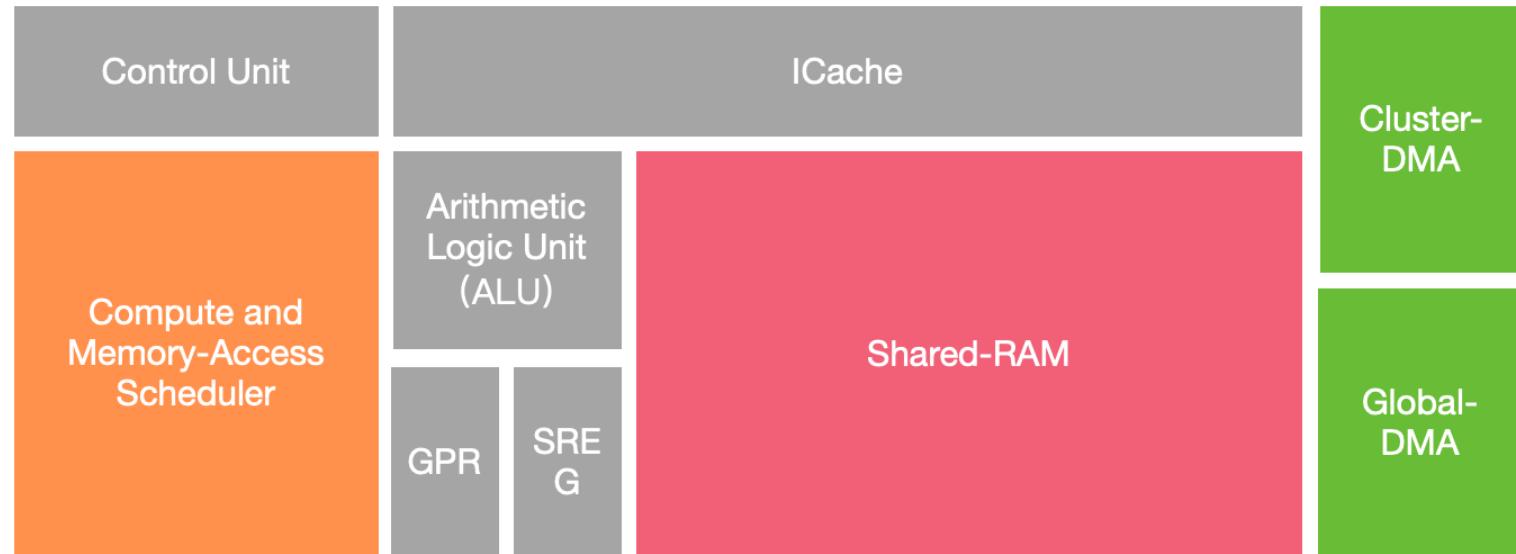
MLUv03 – MPU 核心

- 主要功能：单个 Cluster 内部 Shared-RAM 和多个 Cluster 间 Shared-RAM 的管理和通信。



MLUv03 – MPU 核心

- **Cluster-DMA** : 负责Cluster间和 Shared-RMA间数据传输。
- **Global-DMA** : 负责GPR与片外内存，GPR与Shared-RAM， Shared-RAM和DRAM间数据传输。
- **Shared-RAM** : 4MB，相当于L1级缓存，给具体计算提供数据。



5. MLU03-片内通信

MLUv03 – Cluster 核心

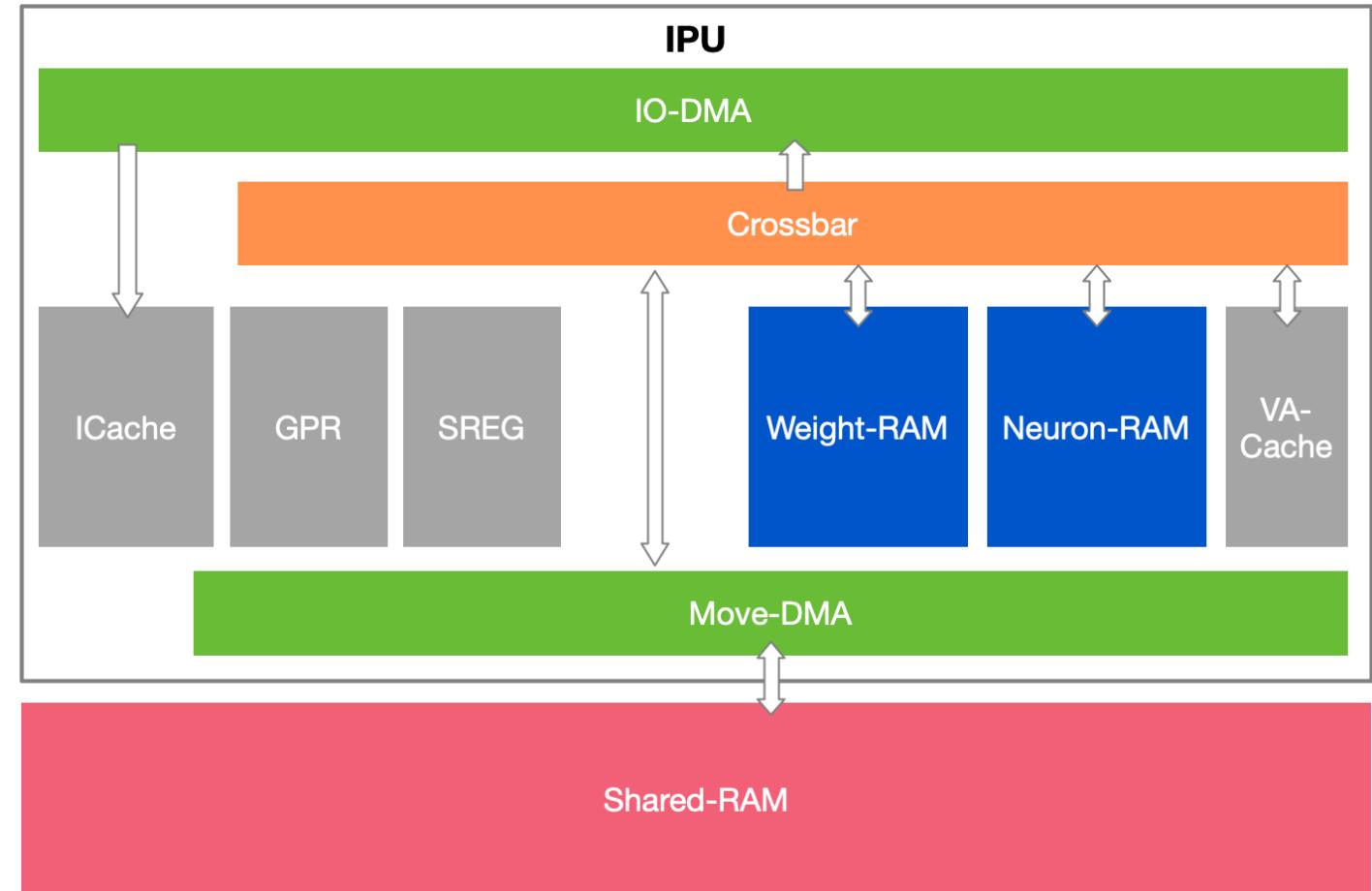
- Cluster内：MPU与片外DRAM交互，提供 Shared-RDMA 和 IPU 广播数据。
- Cluster间：向其他 MPU 传输数据。



MLUv03 – Cluster 核心之通信

数据通路过程：

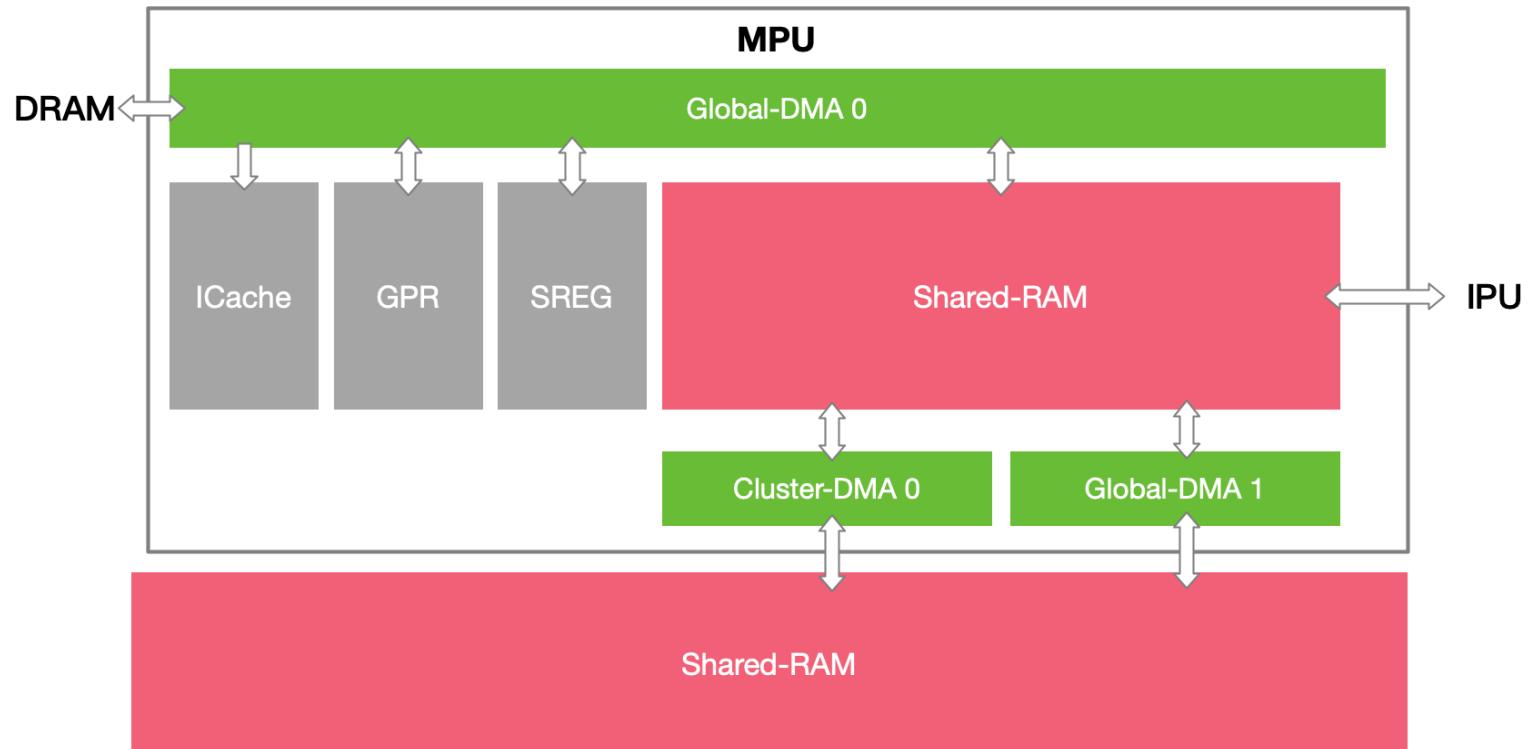
- **I Cache 访问 Global-DRAM :**
IO-DMA读取指令并保存大I Cache
- **DRAM 与 S/W/N-RAM 和 GPR 互相访问** : 通过 IO-DMA 实现
- **S/W/N-RAM 与 GPR 互相访问** : 通过 Move-DMA 实现



MLUv03 – Cluster 核心之通信

数据通路过程：

- **Global-DRAM 访问 ICache :**
Global-DRAM 读取指令并保存到 ICache。
- **Global-DRAM 访问 Shared-RAM 和 GPR :** 通过 Global-DMA 实现
- **Cluster Shared-RAM 和 Shared-RAM 相互访问 :** Cluster DMA 实现





Thank you

把AI系统带入每个开发者、每个家庭、
每个组织，构建万物互联的智能世界

Bring AI System to every person, home and
organization for a fully connected,
intelligent world.

Copyright © 2023 XXX Technologies Co., Ltd.
All Rights Reserved.

The information in this document may contain predictive statements including, without limitation, statements regarding the future financial and operating results, future product portfolio, new technology, etc. There are a number of factors that could cause actual results and developments to differ materially from those expressed or implied in the predictive statements. Therefore, such information is provided for reference purpose only and constitutes neither an offer nor an acceptance. XXX may change the information at any time without notice.



Course chenzomi12.github.io

GitHub github.com/chenzomi12/DeepLearningSystem