Revision History

Rev	Description	Authers	Date
v0.1	编码器架构完整设计与解码器的初步定义	孙雪健	2025.7.13

[TOC]

1. Parameters

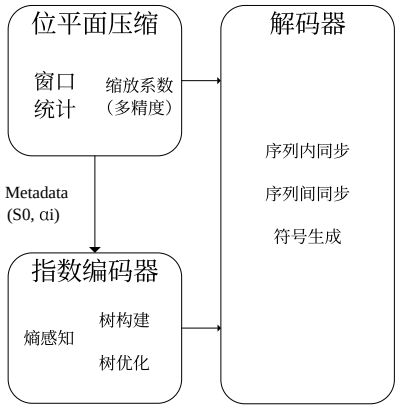
Full Spelling	Abbreviations	Description

2. Overview

Huffman编码因其接近熵界的压缩效率，已成为无损压缩系统的核心组件。传统的FP16/BF16 量化与软件实现已难以满足性能瓶颈，本设计采用了一种结合 Bit-Plane 分解的混合精度 Huffman 编解码器，面向FP16/FP8/FP4的LLM推理加速场景，力争低延迟与高吞吐的同时减小面积和功耗。

3. Interface

系统整体推理加速流程可划分为三个核心模块，分别完成压缩、传输与解压还原任务。系统总体结构如下图所示。



3.1 CMU-MXU Interface

Interface Structure

Name: `DsaRequest`

3.2 L2Xbar-MXU Interface

4. Micro Architecture Design

MXU由DSA接口、L2访存单元、译码单元、发射单元与执行单元组成。具体框图如下：

 Matrix硬件架构图

此架构具有以下特性：

- 每周期至多从DSA接口接收至多一条指令，基础译码器至多译码一条指令
- 具有可以配置数量的若干译码指令槽，并配有拆分计数器与拆分逻辑对指令进行拆分
- 含有可以配置大小的8个矩阵寄存器，每个矩阵寄存器含有4个读端口与2个写端口，支持以行为单位的读取与写入
- 发射级每周期至多发送与执行单元类型数目相同的指令至执行级，通过RF Mux选择相应的操作数送入执行级
- Load与Store操作在各自的Pipe中通过L2Bus对L2Cache进行访问，且通过id进行区分
- 执行单元执行完毕后，通过写回总线对Matrix寄存器进行写回，并将响应传送回译码单元进行状态更新

4.1 SymbolStat

> 确定矩阵指令的格式，并给出对应判断条件

sym_width：输入符号的位宽。 wtWidth：频率计数器的位宽。 depth：符号的总数。

data_in：输入数据流，使用 Decoupled 接口进行握手。 freq_out：输出频率数据，使用 Decoupled 接口进行握手。 start：启动信号，触发模块开始统计频率。 done：完成信号，指示频率统计完成。 busy：忙碌信号，指示模块正在处理数据。 flush：刷新信号，用于重置模块状态。

当前的状态转换逻辑： 依赖 start 信号进行初始化和状态转换 依赖 !io.start 判断输入阶段结束 移除后的问题：无法确定统计开始时机：仅凭 data_in.valid 无法区分是新的统计任务还是数据流的恢复 无法判断输入结束：Decoupled 接口无法表达"数据流结束"的概念 缺少明确的完成指示：上级模块无法知道何时可以读取完整的统计 结果保留 start/done 信号的优势 清晰的任务边界：明确定义统计任务的开始和结束 可预测的初始化：保证每次统计开始时的干净状态 模块间协调：为流水线中的模块协调提供明确的控制信号 状态管理：配合 busy 信号提供完整的模块状态信息

原本DSA接口的逻辑变化点：

- funct3（14-12位）不再表示rs1 、rs2 、rd是否有效的指示信号，在Matrix指令中恒为000
- 所有rs1 、rs2均有效，但当uop（27-25位）为011时，rs1的索引为17-15位
- 当27-25位为111时，协处理器需要写回寄存器。其他情况不写回寄存器

译码逻辑判断条件：

- ms3/md：inst[9:7]

- size：inst[11:10]
- ms1：inst[20:18] & algrithom
- ms2：inst[23:21] & algrithom
- uop：inst[27:25]
- func：inst[31:28]

译码信息Bundle：

load/store指令信息：

- 指令类型：load/store（利用Map表进行管理）
- baseAddr、stride（40bit）
- ms3/md（3bit）
- size（2bit）
- stream（1bit）
- sizeM、sizeN、sizeK
- busMask（L2Bus的位宽为128bit，需要对齐的Bus）（宽度为128/elementWidth）

mma指令信息：

- 指令类型：algrithom（利用Map表进行管理）
- 计算指令类型：fmmacc。。。。（也利用Map管理）
- ms1、ms2、ms3/md（3bit）
- size（2bit）
- sizeM、sizeN、sizeK

config指令信息：

- 指令类型：configi，config（利用Map表进行管理）
- index（3bit）
- Value（32bit）

4.2 Decode Unit

> 给出各个指令的拆分计数器的组成，拆分的逻辑表达式

译码器整体架构 译码拆分与发射架构图 译码拆分器架构图

4.2.1 拆分计数器

mma指令拆分逻辑：

由于目前架构下的数据通路宽度大于等于行宽度，因此对源寄存器不需要对K维度进行拆分

- counterM：指示目前计算的目标矩阵的行计数器，用于计算ms1和md的srcBlock与md的destBlock
- counterN：指示目前计算的目标矩阵的列计数器，用于计算ms2和md的srcBlock与md的srcMask和destMask

计数器自增逻辑（单独写一个模块）：

- 优先遍历counterN，在counterN满的时候自增counterM
- counterM每次自增1（行并行度为1），counterN每次自增基于数据通路位宽而定（列并行度展开）

ms1：

- srcReg：直接取指令信息中的ms1
- srcBlock：counterM >> log2(rowsPerMBlock)，其中rowsPerMBlock表示矩阵1中一个数据通路block中包含的行数（2次幂）
- srcMask：可以不设置srcMask，全部以整个block进行计算

ms2：

- srcReg：直接取指令信息中的ms2，但在16bit数据通路位宽下实际读取ms2与ms2+1两个寄存器
- srcBlock：counterN >> log2(rowsPerNBlock)，其中rowsPerNBlock表示矩阵2中一个数据通路block中包含的行数（2次幂）
- srcMask：可以不设置srcMask，全部以整个block进行计算

md/ms3：

- srcReg：直接取指令信息中的md/ms3
- srcBlock：(counterM >> log2(rowsPerMBlock)) * **(2 * maxSizeN >> log2(rowsPerNBlock))** + counterN >> log2(rowsPerNBlock)
  - (2 * maxSizeN >> log2(rowsPerNBlock))代表结果寄存器一行中存在几个Block，由于16bit宽度下会进行拼接，因此有乘2
- srcMask：可以不设置srcMask，以结果寄存器整体Block进行计算

mma硬件拆分方案

load/store指令拆分逻辑：

由于矩阵寄存器中每一行的地址能确保连续，因此目前架构下的访存指令均以寄存器行为单位进行拆分和执行

同时访存时需要考虑跨CacheLine的问题，需要把跨CacheLine的请求拆分为两个请求

- counterM：指示目前访存的矩阵寄存器行数，用于计算ms3的srcBlock与md的destBlock
- counterK：指示当前行处理到的位置，用于计算ms3的srcMask与md的destMask

计数器自增逻辑：

- counterK每次根据发射的宽度自增，当达到阈值后归零；发射宽度为maxSizeK - counterK和baseAddr距离下一个CacheLine边界的较小值
- counterM在每次counterK达到阈值后自增1

ms3/md：

- srcReg：直接取指令信息中的ms3/md
- srcBlock：counterM
- srcMask：
  - 起始点向量，通过全1掩码左移counterK位完成
  - 终点向量，通过与CacheLine大小对应全1掩码右移此行的起始地址低位来实现

- 最终掩码向量由起点向量与终点向量低位相与得到

访存硬件拆分方案

4.2.2 依赖检查

本方案的依赖检查主要以矩阵寄存器为粒度进行检查，主要分为不同指令间依赖与同一指令内的不同微指令间的依赖检查

译码槽间的请求仲裁

不同指令间：

- 在指令进入指令槽时，更新指令槽中的年龄向量，并尝试占据对应寄存器（通过grant机制完成）
- 若寄存器占据成功则置位指令槽中对应的源寄存器有效位，并修改寄存器状态表（状态表中包含空闲、占据、执行完成三个状态）
- 对mma指令而言，需要为其目的寄存器（寄存器0和寄存器4）配置类移位寄存器形式的block列表，以指示无法bypass得到的block编号。其中移位寄存器数目为最终累加加法器的延时
- 当检查对象为mma和mma间的WAW依赖时，当目的寄存器冲突时，如果不存在block WAW冲突则可以尝试占据寄存器，如果成功则转换占据状态

同一指令内的不同微指令间：

- 只有mma可能会存在WAW的依赖问题，可以通过判断如上移位的表的形式来判断WAW依赖问题

4.3 Matrix Register

> 给出矩阵寄存器的组织形式，对不同类型指令的指示信号与读取位置的对应关系

矩阵寄存器读取示意图

矩阵寄存器的组织形式：

矩阵寄存器的组织形式表示会以何种方式读取矩阵寄存器的一部分，会影响矩阵寄存器读取时的选择器的数目与时序。本方案中的组织形式设计主要针对mma指令和访存进行区分（默认数据通路宽度大于矩阵寄存器的行宽度），组织形式如下所示：

- mma指令的源寄存器会根据架构中的M和N的并行度对ms1和ms2的M行和N行作为一个block，每次读取均以这个单位进行读取
- mma指令的源寄存器会根据架构中的M和N的并行度对ms3的M行N列的块作为一个block，每次读取均以这个单位进行读取
- 访存指令下会以每行作为Block进行管理，每次以单行作为读取和写入的单位

8个矩阵寄存器中每个寄存器在读取时会配备两套选择器，一套为mma指令，一套为访存指令。

> 其中mreg0和mreg4为mma的目的寄存器，配备目的寄存器的组织形式与load/store的组织形式
>
> 其余mreg均为mma源寄存器组织形式与load/store的组织形式

矩阵寄存器组织形式

矩阵寄存器的读取与写入：

矩阵寄存器的读取和写入分为两步：

- 第一步为将对应矩阵寄存器的对应Block选择出来
- 如果有掩码选择的逻辑，第二步则需要对数据进行移位等操作进行格式的对齐

需要提供三种读取的模式，以及两种写入的模式：

- 读取模式包括源寄存器组织形式的读取、结果寄存器组织形式的读取与store指令的读取
- 写入模式包括结果寄存器组织形式、load指令的写入

读取和写入模式的接口需要接收以下信息：

- 指令类型：用于识别选择器组
- 寄存器编号：用于指示寄存器选择器组的控制信号
- 寄存器Block编号：
  - 如果为mma源寄存器组织形式，取低N比特，N为log2（MLEN/32/Mpara）位
  - 如果为mma目的寄存器组织形式，取低N比特，N为log2((MLEN/32/Mpara) * (MLEN/EleWidth/NPara))
  - 如果为load和store下的组织形式，取低N比特，N为log2(MLEN/32)
- 读取写入掩码：
  - 仅在load和store下生效，掩码每一位代表EleWidth宽度的数据，因此掩码位数为MLEN/EleWidth
  - 读取和写入的数据需要根据掩码位置进行移位，移位位数为index(ff1) * EleWidth

4.4 Load/Store Pipe

> 需要明确与ACE总线的接口，并细化暂存Queue的逻辑与流水线逻辑

LSU架构图

ACE总线信号：

读总线信号列表

信号名	输入输出（位宽）	说明
dsa_req_rready	输入（1）	读请求握手
dsa_pad_req_rvalid	输出（1）	读请求握手
dsa_pad_req_raddr	输出（40）	读请求起始地址
dsa_pad_req_rlen	输出（2）	读请求长度，长度为128*（n+1）
dsa_pad_req_rid	输出（5）	读请求id
dsa_pad_resp_rready	输出（1）	读响应握手
pad_dsa_resp_rvalid	输入（1）	读响应握手
pad_dsa_resp_rid	输入（5）	读响应id
pad_dsa_resp_rdata	输入（128）	读响应数据
pad_dsa_resp_rlast	输入（1）	读响应结束信号

写总线信号列表：

信号名	输入输出（位宽）	说明
pad_dsa_req_wready	输入（1）	写请求握手
dsa_pad_req_wvalid	输出（1）	写请求握手
dsa_pad_req_waddr	输出（40）	写请求起始地址
dsa_pad_req_wlen	输出（2）	写请求长度，与读相同
dsa_pad_req_wid	输出（5）	写请求id
dsa_pad_req_fullline	输出（1）	写请求是否为满cacheline
dsa_pad_resp_wready	输出（1）	写响应握手
pad_dsa_resp_wvalid	输入（1）	写响应握手
pad_dsa_resp_wid	输入（5）	写响应id
pad_dsa_req_wdready	输入（1）	写数据请求握手
dsa_pad_req_wdvalid	输出（1）	写数据请求握手
dsa_pad_req_wdata	输出（128）	写数据请求数据
dsa_pad_req_wstrb	输出（16）	写数据掩码，以byte为单位
dsa_pad_req_wlast	输出（1）	写数据请求结束信号

读通道包含读请求和读响应通道，写通道包含写地址请求、写数据请求与写响应通道。其中读通道和写通道的大部分信号一致，下面是读通道的信号解读：

- 读请求通道包含握手信号与请求信息的信号，包括请求的地址，请求的长度和请求的id
    - 请求长度以128bit为单位，同时请求的地址与请求长度不能超过一个Cacheline
    - 请求的id用于识别有可能乱序返回的响应，同时上一个id响应未返回，不能发送新的同id请求
    - 读写的id是独立的
- 读响应通道包含握手信号、响应信息与响应数据
    - 响应返回时会携带id信息，用于识别响应所对应的请求，响应有可能乱序返回
    - 响应数据以128bit每周期的速率进行返回，最后的128bit数据会伴随着结束信号一同返回
- 写请求信号的特殊信号包括fullline信号，用于表示此次是否为完整Cacheline的请求（wlen为3，wstrb全为1），处理效率会比普通写要高
- 写数据请求通道包括写数据，数据掩码与数据结束信号，数据掩码以8bit为单位表示

访存单元的处理流程：

- 译码单元将uop信息发送至RF级，store指令会在RF进行寄存器block的读取，而load指令则为了与store时序对齐而空打此处空打一拍
- store指令的RF级处会读取寄存器的值，并存储在RF级中的数据buffer中
- 若L2 Ace Bus可以接收请求，则RF级会将请求和数据发送至Ace总线。同时根据load和store指令类型将相关信息存储至load/store Queue中
- 若L2 Ace Bus无法接收请求，则RF级会产生反压信号阻止译码单元向访存单元发送新的uop

- 当L2的访存请求响应返回时，会将请求的id广播至load Queue和store Queue，比对成功的项会弹出并返回响应至WB Bus

4.5 Mac Array

> 暂时采用乘加树实现，需要明确最后累加数的读取与RF Mux的交互方式

Mac Array主要完成mma指令的乘累加操作，主要结构和主要特性如下所示：

- 整个Mac Array会包含若干个sub mac array，而子阵列的个数由N维度和M维度的并行度决定
- 子阵列由流水化的乘加压缩树和最终结果的累加加法器组成，每个子阵列会完成mlen长度的ms1和ms2的乘加压缩，并与md进行累加
- 每个子阵列的ms1和ms2输入在指令发射的起始点进行读取，而ms3/md则是在子阵列乘加压缩后进行读取，因此分为两个RF Mux

 MacArray

Mac Array运作流程：

- 当mma的uop成功发射时，首先会进入到RF级进行ms1和ms2操作数的读取
- 在mma uop发射后一周期，会进入到MacArray中进行计算，并会携带valid信号随着流水线流动
- 当valid有效的uop进入到最终结果累加的流水级前一周期时，需要向ms3/md RF Mux发出读取结果寄存器的请求
- 在最终结果累加流水级会完成结果的累加，并发出向写回总线发出相应的写回信号

Mac Array的输入输出格式：

- 输入主要由ms1、ms2、ms3组成，由于不同的源寄存器的block格式不同，因此使用平铺的方式进行组织
- 输出主要为md寄存器的写回，也采用平铺的方式进行组织；在WB Mux中选择进入到结果寄存器中