AMLA：以加代乘的高性能昇腾MLA算子

计算数学算法专家　廖崎臣

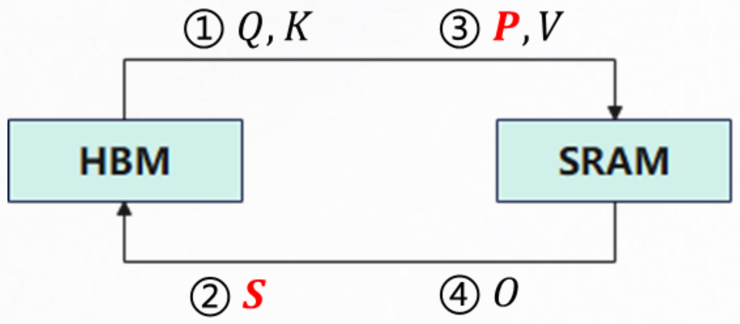
2025年12月20日

Content
目录

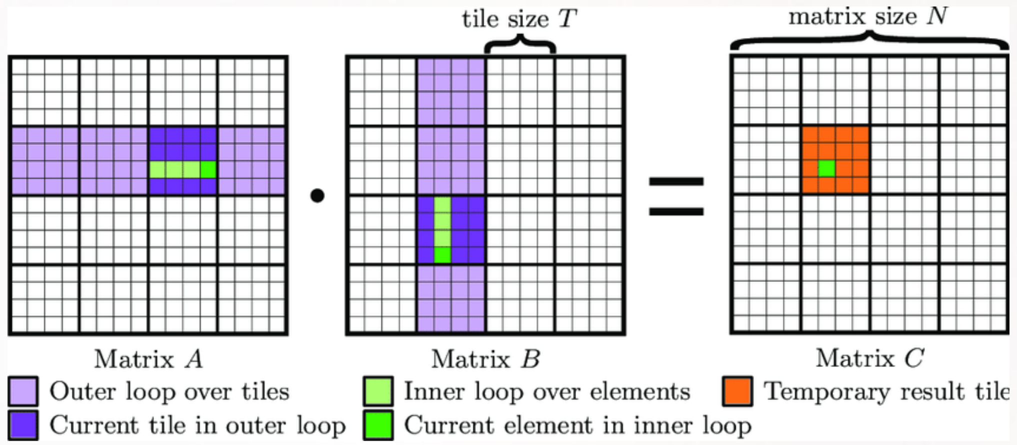
01 背景及motivation

Flash Attention：解耦计算降低峰值显存占用

- Attention计算为：$\mathrm{Softmax}(QK^T)V$，其中$\mathrm{Softmax}(\boldsymbol{x}) = \frac{1}{l}[e^{x_1-m}, e^{x_2-m}, \cdots, e^{x_N-m}], m = \max_i x_i, l = \sum_i e^{x_i-m}$



■ 规模为(N,N)

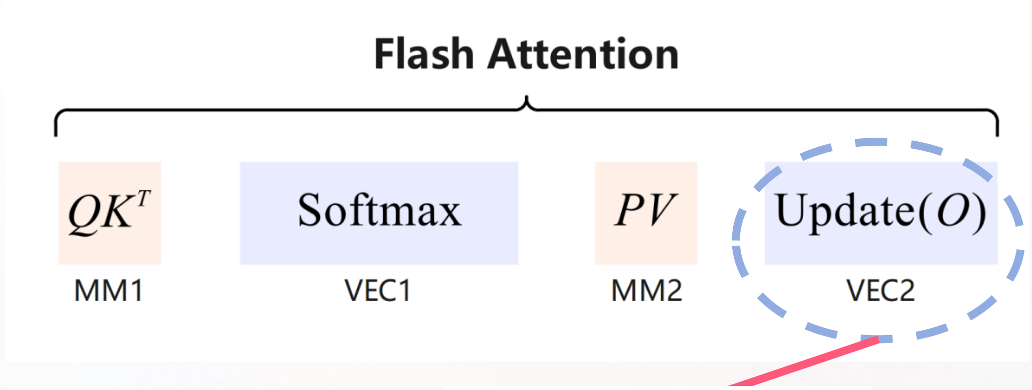


- $\mathrm{Softmax}([\boldsymbol{x},\boldsymbol{y}])V = \left[e^{m(\boldsymbol{x})-m} \times \frac{l_x}{l} \times \mathrm{Softmax}(\boldsymbol{x}), e^{m(\boldsymbol{y})-m} \times \frac{l_y}{l} \times \mathrm{Softmax}(\boldsymbol{y})\right] V$

其中$m(\boldsymbol{x}) = \max_i x_i, m(\boldsymbol{y}) = \max_i y_i, m = \max\big(m(\boldsymbol{x}), m(\boldsymbol{y})\big), l_x = \sum_i e^{x_i-m(\boldsymbol{x})}, l_y = \sum_i e^{y_i-m(\boldsymbol{y})}, l = e^{m(\boldsymbol{x})-m} l_x + e^{m(\boldsymbol{y})-m} l_y$

算法 1 Flash Attention 计算流程

1: 初始化：$O \leftarrow 0, \ell \leftarrow 0, m \leftarrow -\infty$，$K/V$ 分为 N 个子块
2: **for** $i = 1$ to N **do**
3: $S_i \leftarrow QK_i^T$ ▷ MM1 阶段
4: $m_i \leftarrow \max(m, \mathrm{rowmax}(S_i/\sqrt{D_k}))$ ▷ VEC1 阶段
$P_i \leftarrow \exp(S_i/\sqrt{D_k} - m_i)$
$\ell_i \leftarrow \ell \times \exp(m - m_i) + \mathrm{rowsum}(P_i)$
5: $P_i V_i$ ▷ MM2 阶段
6: $O \leftarrow O \times \exp(m - m_i) + P_i V_i$ ▷ VEC2 阶段
7: $m \leftarrow m_i, \ell \leftarrow \ell_i$
8: **end for**
9: 输出 O/ℓ



对齐指数上的偏移

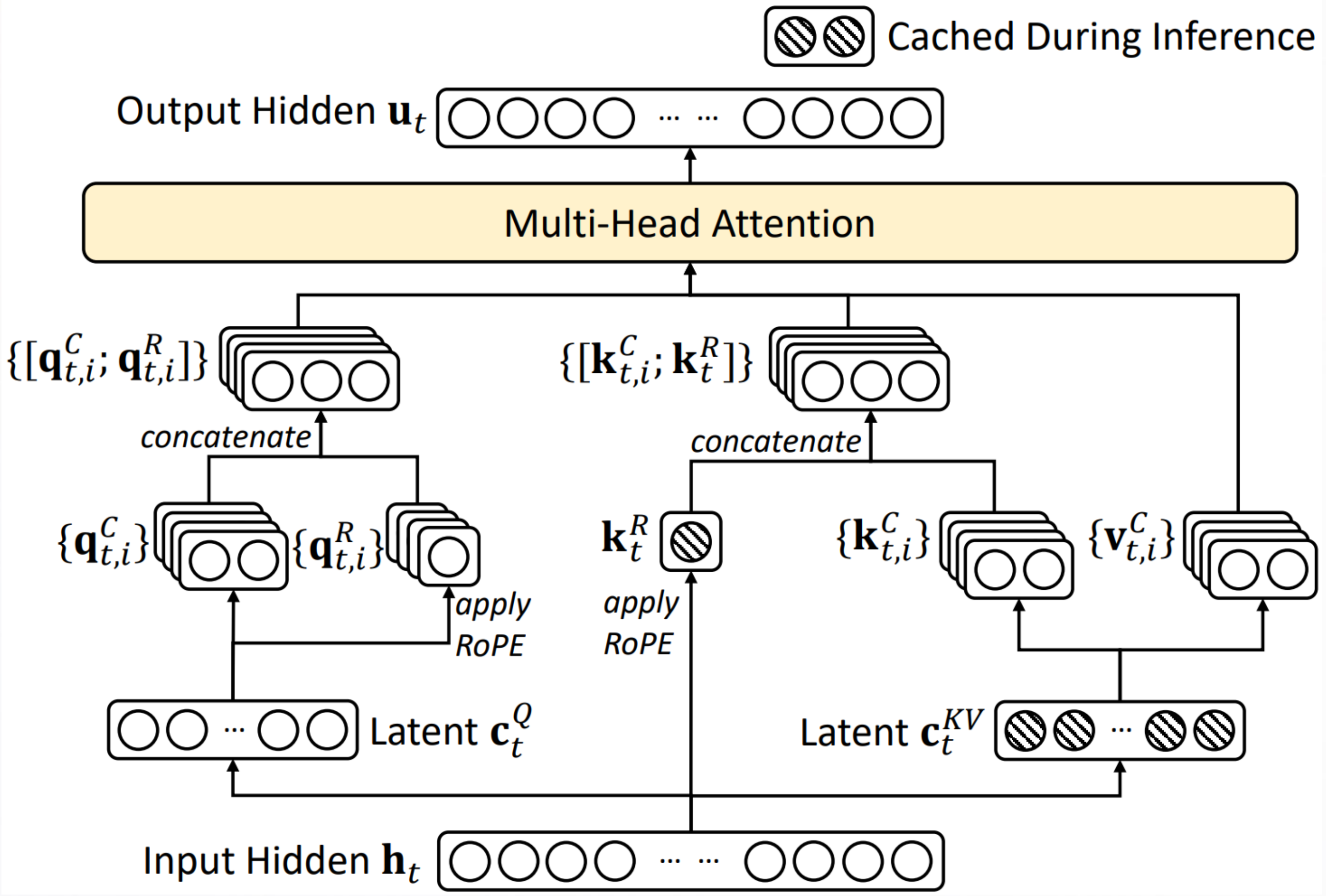
$$\boldsymbol{O} \leftarrow \boldsymbol{O} \times \exp(\boldsymbol{m} - \boldsymbol{m}_i) + \boldsymbol{P}_i \boldsymbol{V}_i$$

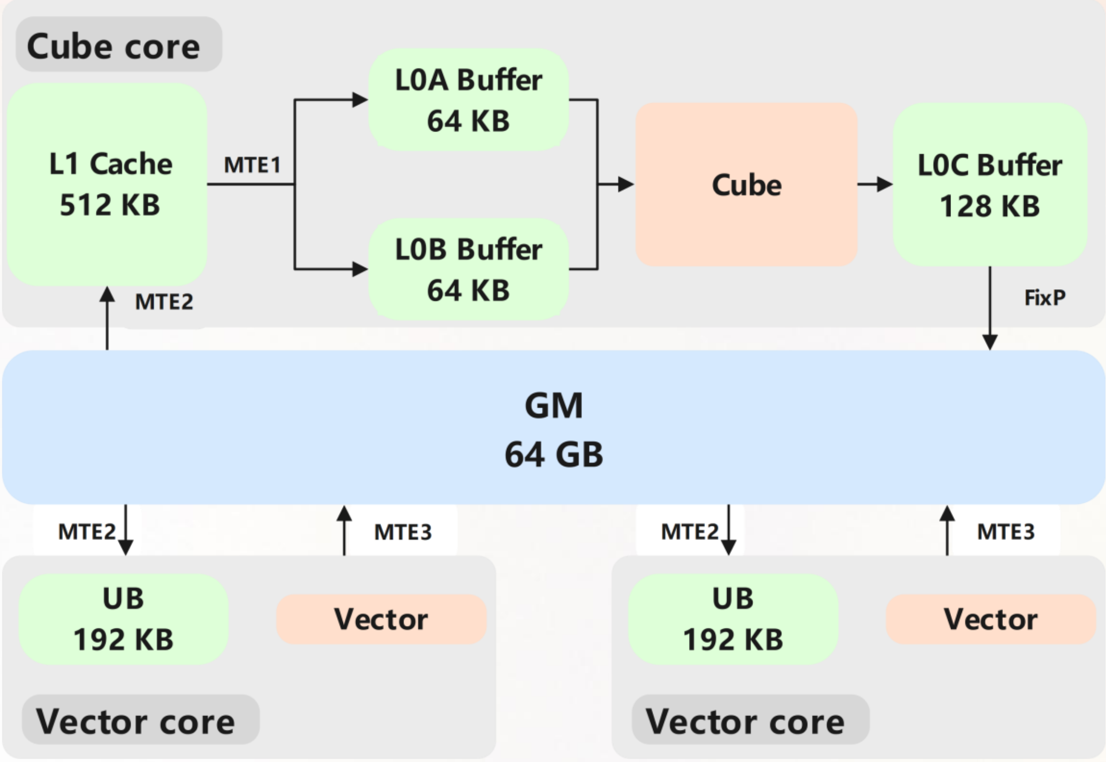
https://www.cs.sfu.ca/~ashriram/Courses/CS7ARCH/hw/hw4.html

此处预留直播画面LOGO
不放置内容

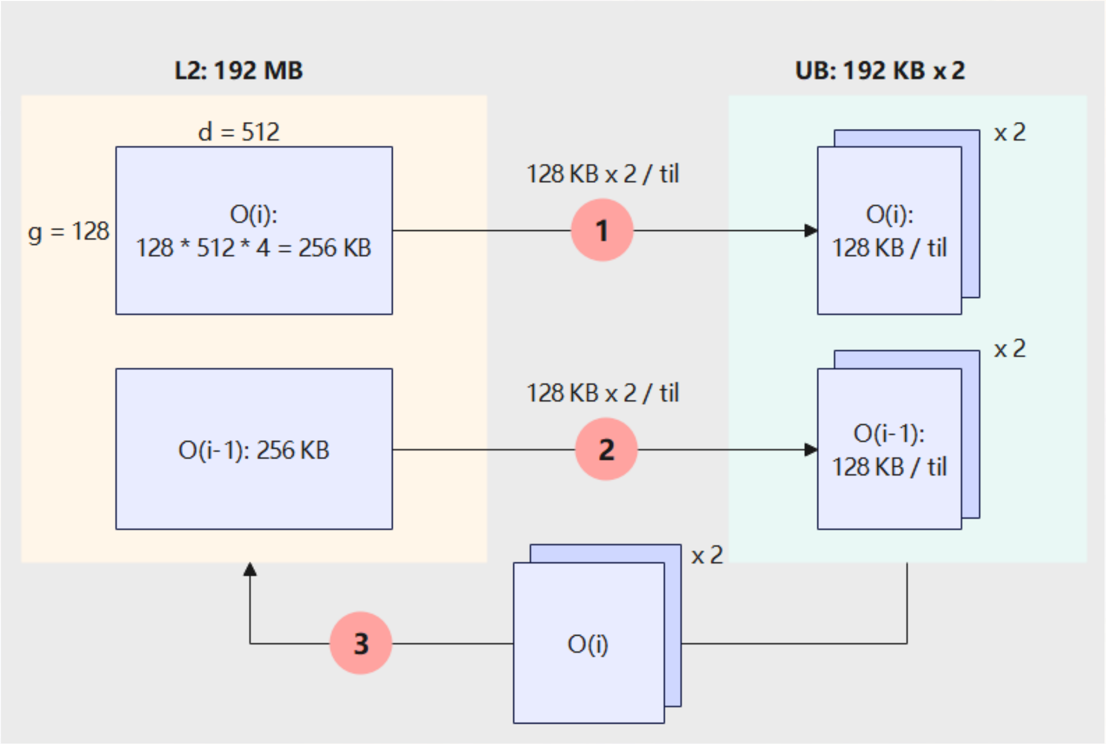
VEC2-Rescaling：变量尺寸过大

Multi-Head Latent Attention (MLA)





910芯片



$$O_i \longleftarrow \exp(m_{i-1} - m_i) \times O_{i-1} + P_i V_i$$

O_i的shape是(G, D_v)， m_i的shape是$(G, 1)$

尺寸过大，无法常驻

MLA场景下

$$G = 128, D_v = 512$$

占用空间大小为$128 \times 512 \times 4Byte = 256KB$

DeepSeek-V3 Technical Report

AMLA算法

02

逆向更新的尝试

展开$O_i \leftarrow \exp(m_{i-1} - m_i) \times O_{i-1} + P_i V_i$可得

$$O_i = P_0 V_0 \times e^{m_0 - m_i} + P_1 V_1 \times e^{m_1 - m_i} + \cdots + P_i V_i,$$

如果我们希望乘法作用在$P_i V_i$而不是O_{i-1}上

$$e^{m_i - m_0} O_i = P_0 V_0 + P_1 V_1 \times e^{m_1 - m_0} + \cdots + P_i V_i \times e^{m_i - m_0},$$

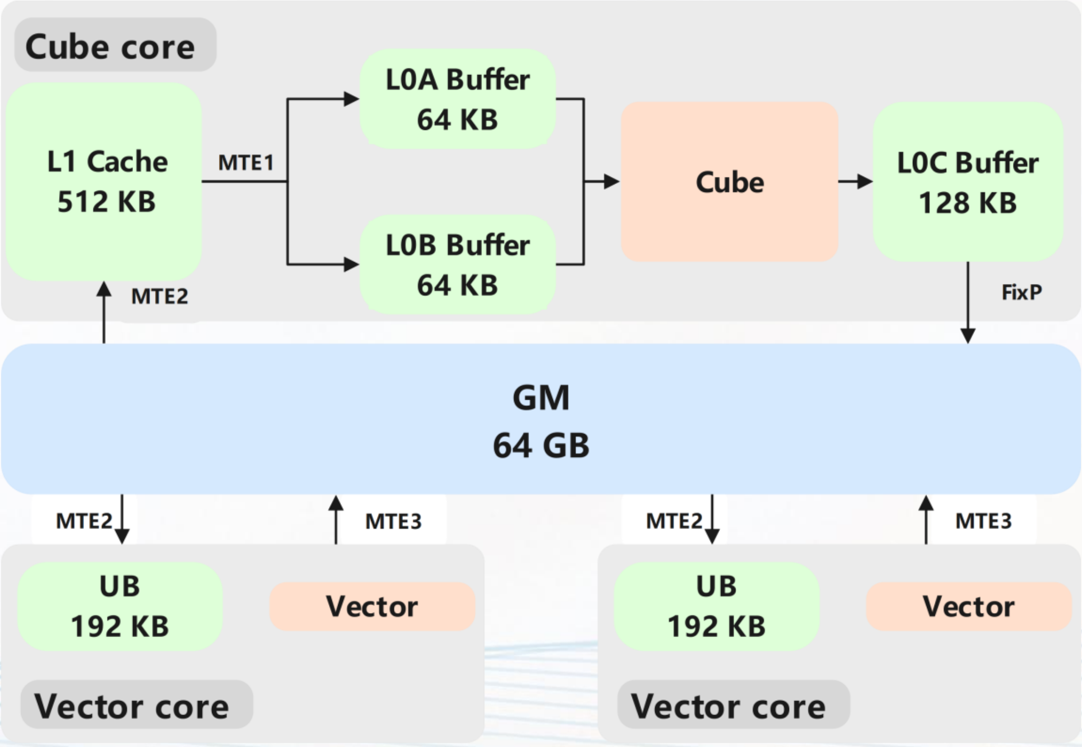
对应迭代式：

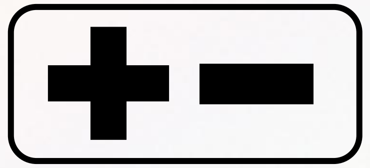
$$\widetilde{O}_i \leftarrow \widetilde{O}_{i-1} + P_i V_i \times \exp(m_i - m_0)$$

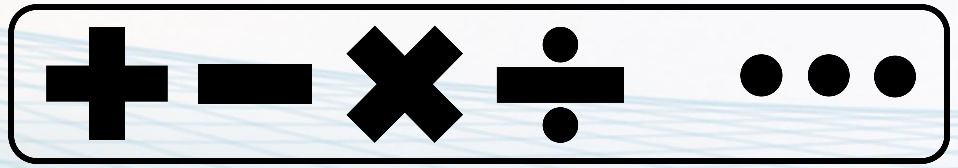
其中$\widetilde{O}_i = e^{m_i - m_0} O_i$

但由于$\mathbf{m_0 \le m_1 \le \cdots \le m_i}$，该迭代方式在数值上极不稳定，大概率数值上溢，不可行

对于O_{i-1}的修正无法避免，**如何用更高效的方式实现修正？**







以加代乘

符号位：**S** (1 bit)　指数位：**E** (8 bits)　尾数位：**M** (23 bits)

FP32

0	1	0	0	0	1	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

$$Y_{Float} = (-1)^S \times (1 + \frac{M}{2^{23}})2^{E-127} \xrightleftharpoons[FP32]{Int32} Y_{Int} = M - 2^{31}S + 2^{23}E$$

Int32

0	1	0	0	0	1	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

$$Y_{Float} \times 2^n$$

Y_{Int}

0	1	0	0	0	1	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

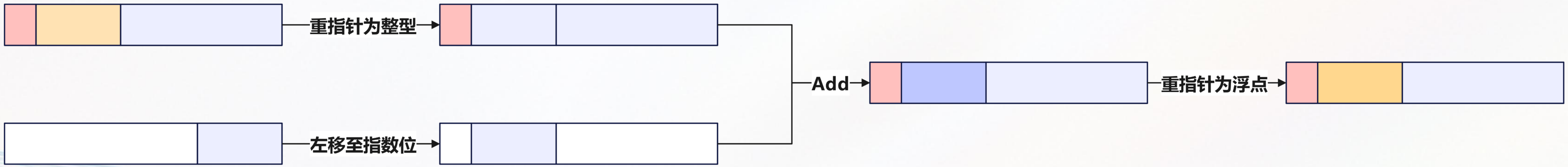
+

$n \times 2^{23}$

0	0	1	1	0	0	1	0	1

$$Y_{Float} \times 2^n = (-1)^S \times (1 + \frac{M}{2^{23}})2^{E+n-127} \xrightleftharpoons[FP32]{Int32} Y_{Int} + n \times 2^{23} = M - 2^{31}S + 2^{23}(E+n)$$

浮点数与2的整数次幂之间的乘法，可通过加法实现



AMLA

- 显然，$O_i \leftarrow \exp(m_{i-1} - m_i) \times O_{i-1} + P_i V_i$中的乘数$\exp(m_{i-1} - m_i)$可以是任意正实数，大概率不是2的整数次幂
- 但是任意正实数都可以用2的整数次幂逼近：

引理 2. 给定 $x \in \mathbb{R}^+$，令 $n = \mathrm{round}\left(-\frac{x}{\ln 2}\right) \in \mathbb{Z}$，有

$$\frac{1}{\sqrt{2}} \le 2^n \exp(x) \le \sqrt{2}$$

证明. 由于 $n = \mathrm{round}\left(-\frac{x}{\ln 2}\right)$，有

$$\left|n + \frac{x}{\ln 2}\right| \le 0.5,$$

根据幂函数的单调性，

$$\frac{1}{\sqrt{2}} \le 2^n \exp(x) \le \sqrt{2}.$$

- 这个乘数的意义是对齐不同块在指数上的偏移：

$$\times \exp(m_{i-1} - m_i)$$

$$O_{i-1} = [\exp(S_0 \boxed{- \boldsymbol{m_{i-1}}}), \exp(S_1 \boxed{- \boldsymbol{m_{i-1}}}), \cdots \exp(S_{i-1} \boxed{- \boldsymbol{m_{i-1}}})] \begin{bmatrix} V_0 \\ V_1 \\ \vdots \\ V_{i-1} \end{bmatrix}, P_i = \exp(S_i - \boldsymbol{m_i})$$

$$\bar{O}_{i-1} = [\exp(S_0) \times \boxed{\boldsymbol{2^{n_{i-1}}}}, \exp(S_1) \times \boxed{\boldsymbol{2^{n_{i-1}}}}, \cdots \exp(S_{i-1}) \times \boxed{\boldsymbol{2^{n_{i-1}}}}] \begin{bmatrix} V_0 \\ V_1 \\ \vdots \\ V_{i-1} \end{bmatrix}, P_i = \exp(S_i) \times \boldsymbol{2^{n_i}}$$

$$\times 2^{n_i - n_{i-1}}$$

基于2的整数次幂放缩同样可以达到防溢出的效果

AMLA

用2的整数次幂逼近行最大值：$n_i = round(-m_i/\ln 2)$，定义指数偏移的转换因子$r_i = \exp(-n_i \ln 2 - m_i)$，有

$$\exp(m_{i-1} - m_i) = 2^{n_i - n_{i-1}} \times \frac{r_i}{r_{i-1}}$$

那么递归式可写为：

$$\bar{O}_i \leftarrow \bar{O}_{i-1} \times 2^{n_i - n_{i-1}} + \frac{1}{r_i} \times P_i V_i$$

其中$\bar{O}_i = \frac{O_i}{r_i}$，具体流程：

$$n_0 \leftarrow \mathrm{round}\left(-\frac{m_0}{\ln 2}\right), O_0 \leftarrow \cancel{\exp(m_0 + n_0 \ln 2) \times \exp(S_0} - m_0)\, V_0 = 2^{n_0} \exp(S_0)\, V_0$$

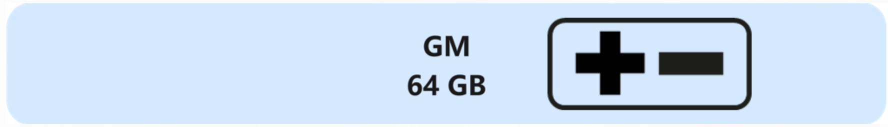
$$n_1 \leftarrow \mathrm{round}\left(-\frac{m_1}{\ln 2}\right), 2^{n_1 - n_0} O_0 + \exp(m_1 + n_1 \ln 2) \exp(S_1 - m_1)\, V_1 = 2^{n_1} \exp(S_0)\, V_0 + 2^{n_1} \exp(S_1)\, V_1$$

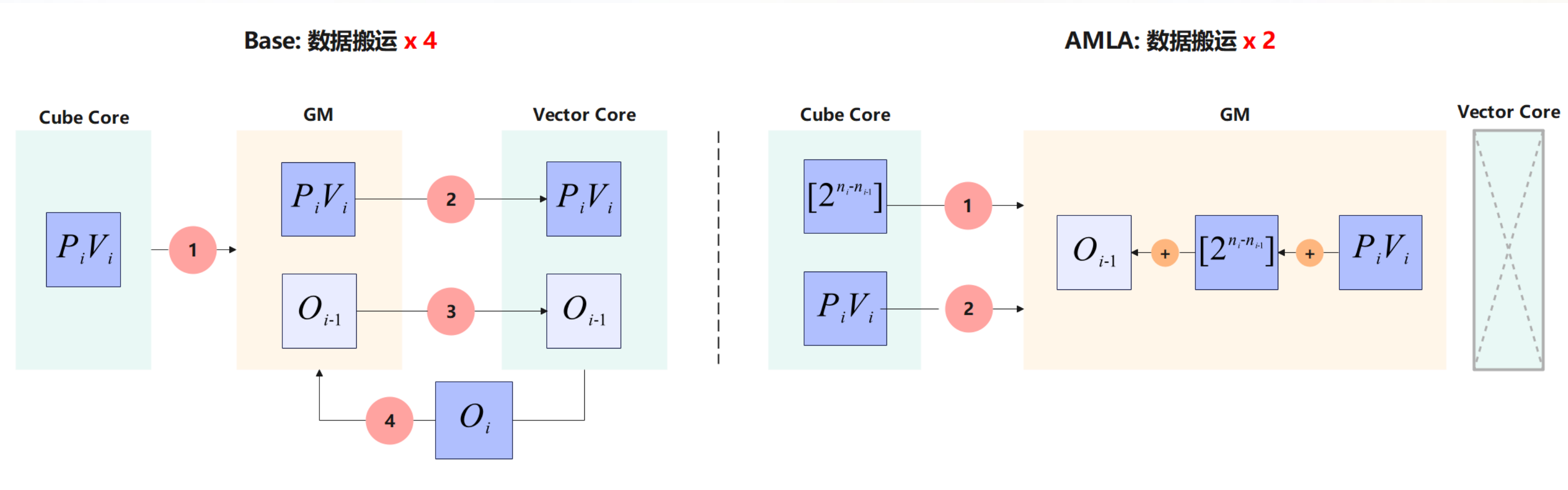
逼近最大值
$2^{n_i} \approx \exp(-m_i)$

以加代乘，原子累加实现对O的更新

二次更新P_i结果提取因子

原子累加实现更新





算法泛化性

乘积累加形式的递归格式都可以基于上述算法进行存上更新

假设需要对初始值x^0进行N次更新，即对应N次乘积累加操作：

$$x^1 \leftarrow a^1 \times x^0 + y^1, x^2 \leftarrow a^2 \times x^1 + y^2, \cdots, x^N \leftarrow a^N \times x^{N-1} + y^N$$

采用新的更新方式

$$\frac{x^1}{T^1} \leftarrow \frac{a^1}{T^1} \times x^0 + \frac{y^1}{T^1}, \frac{x^2}{T^2} \leftarrow \left(a^2 \times \frac{T^1}{T^2}\right) \times \left(\frac{x^1}{T^1}\right) + \frac{y^2}{T^2}, \cdots, \frac{x^N}{T^N} \leftarrow \left(a^N \times \frac{T^{N-1}}{T^N}\right) \times \left(\frac{x^{N-1}}{T^{N-1}}\right) + \frac{y^N}{T^N}$$

即每一次的递归更新公式为：

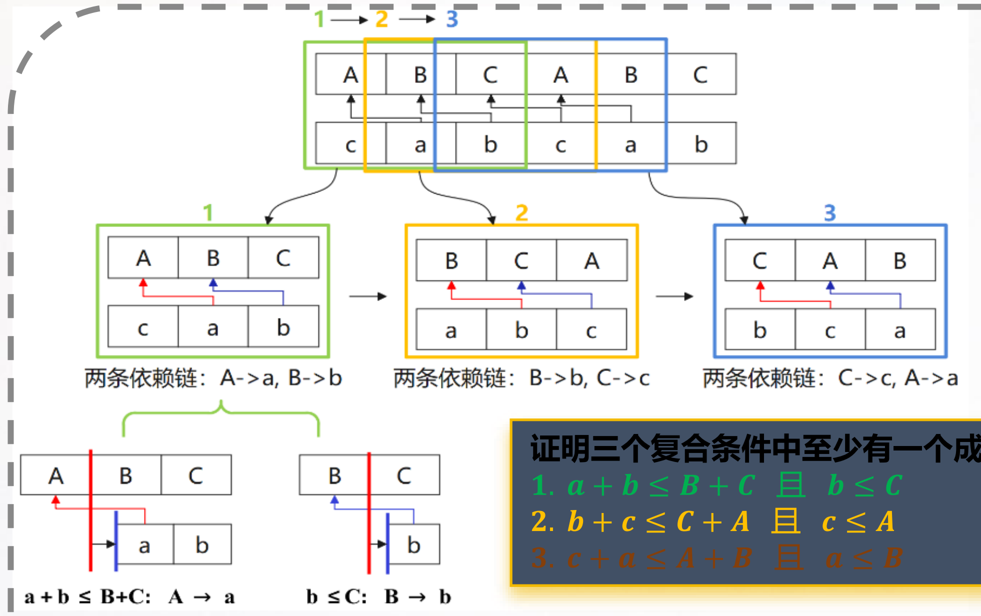
$$\frac{x^i}{T^i} \leftarrow \left(a^i \times \frac{T^{i-1}}{T^i}\right) \times \left(\frac{x^{i-1}}{T^{i-1}}\right) + \frac{y^i}{T^i}$$

取$T^0 = 1$，在第i次更新时，计算因子T^i使得$a^i \times \frac{T^{i-1}}{T^i}$为2的幂次，即可在每一次更新时通过以加代乘的技术基于存内计算原子加法实现。最终的结果为$\frac{x^N}{T^N}$，只需要乘上T^N即可得到最终结果。

部分工程细节

03

Preload Pipeline & 分层Tiling



证明三个复合条件中至少有一个成立

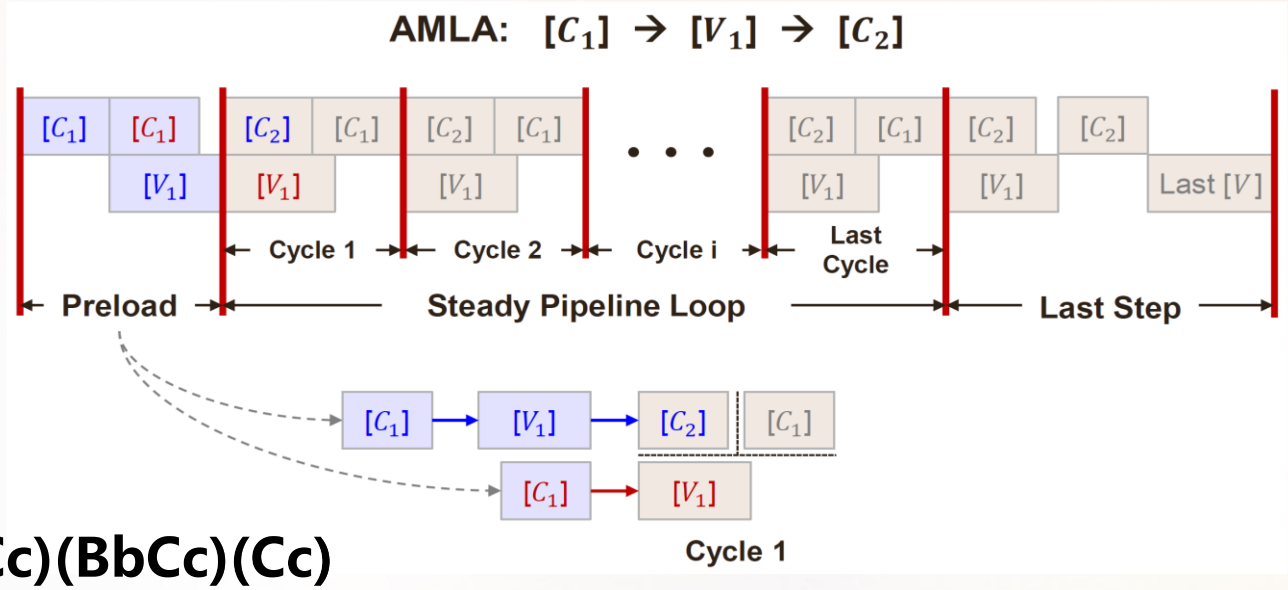
1. $a+b \le B+C$ 且 $b \le C$
2. $b+c \le C+A$ 且 $c \le A$
3. $c+a \le A+B$ 且 $a \le B$

Cube : A, B, C; Vector : a, b, c

依赖链： A→a→B→b→C→c

Preload流水：

(Aa)(AaBb)(AaBcCc)……(AaBcCc)(BbCc)(Cc)



- 计算依赖关系为 C1 → V1 → C2 → V2 → ... → Cn → Vn
- Cube总耗时 ≥ Vector 总耗时： $\sum_{i=1}^{n} v_i \le \sum_{i=1}^{n} C_i$
- 证明存在 $m \in \{1, 2, \dots, n\}$ 集合，使得循环条件成立：

$$\bigwedge_{j=1}^{n-1}\left(\sum_{i=0}^{j-1} v_{n-m-i} \le \sum_{i=0}^{j-1} C_{n+1-m-i}\right)$$

➡

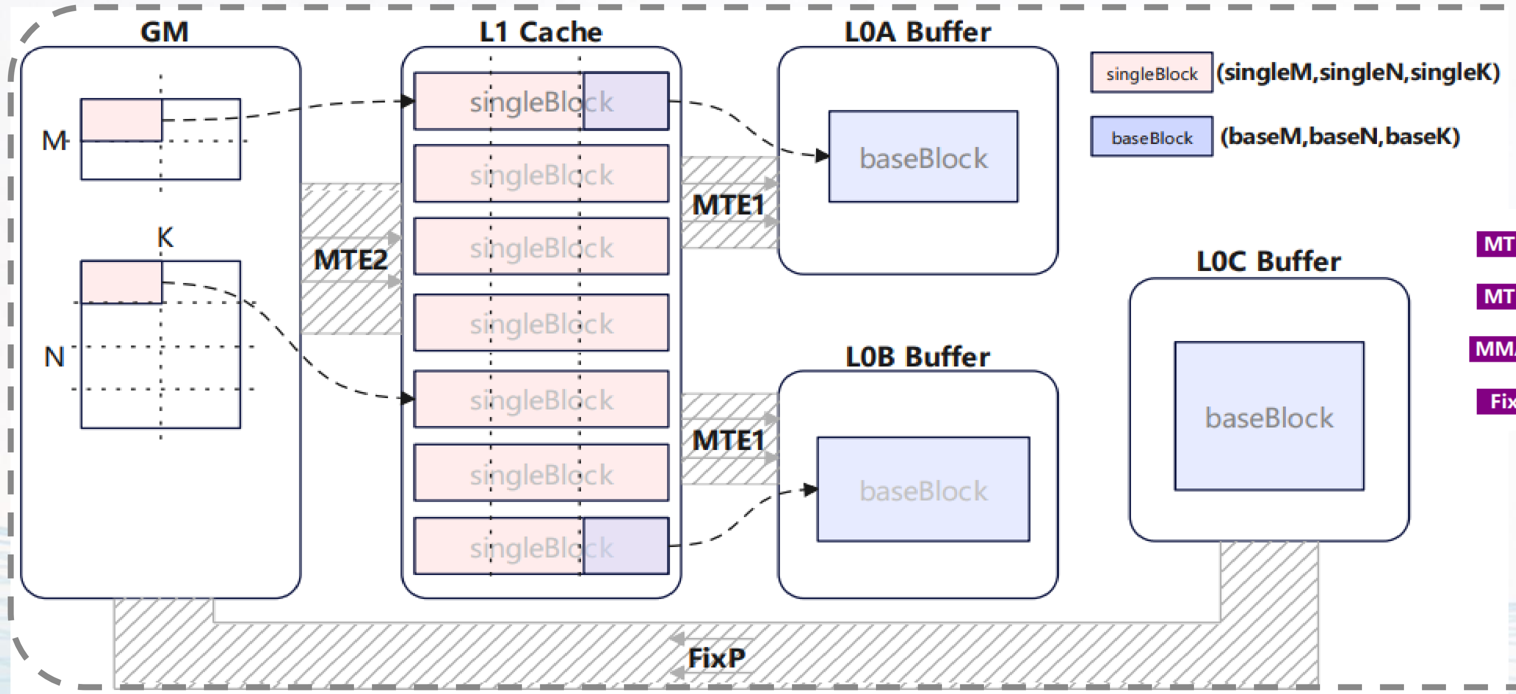
- 定义辅助序列：$c_i = v_i - C_{i+1}$， $c_{n+i} = c_i$
- 循环条件等价于：证明存在 $m \in \{1, 2, \dots, n\}$，使得所有 $j \in \{1, 2, \dots, n-1\}$，下述不等式均成立：

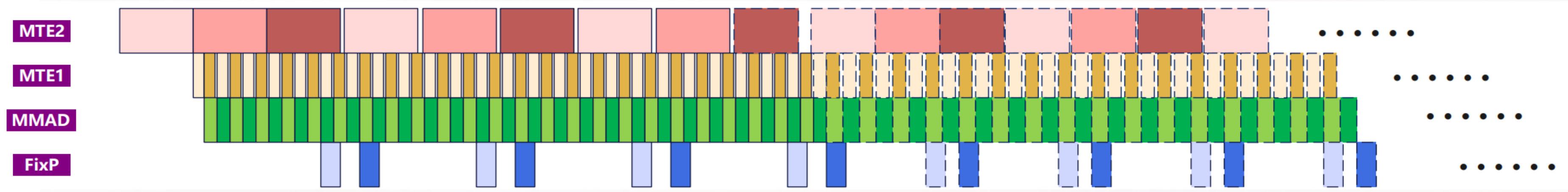
$$\sum_{i=0}^{j-1} c_{m-i} \le 0$$

➡

- 构造部分和序列： $F(l) = \sum_{i=1}^{l} c_i$， $l \in \{1, 2, \dots, n\}$
- $F_{min} = F_k$， $m = k$，对任意 $j \in \{1, 2, \dots, n-1\}$：

$$\sum_{i=0}^{j-1} c_{k-i} = \sum_{i=1}^{k} c_i - \sum_{i=1}^{k-j} c_i = F(k) - F(k-j) \le 0$$





04

性能结果

(a) Part 1

S_k		1024		2048		3072	
S_q	Hardware	duration (μs)	FU	duration (μs)	FU	duration (μs)	FU
1	910	95	40.9%	140	55.1%	186	62.4%
	GPU	85	32.6%	128	43.3%	173	48.0%
2	910	135	57.3%	219	70.7%	306	75.8%
	GPU	115	48.1%	196	56.5%	278	59.8%

(b) Part 2

S_k		4096		6144		16384	
S_q	Hardware	duration (μs)	FU	duration (μs)	FU	duration (μs)	FU
1	910	241	64.1%	331	70.2%	830	74.5%
	GPU	215	51.5%	316	52.6%	766	57.8%
2	910	388	79.7%	565	82.2%	1427	**86.8%**
	GPU	374	59.2%	527	63.0%	1314	67.4%

AMLA: MUL by ADD in FlashAttention Rescaling

Qichen Liao liaoqichen2@huawei.com
Chengqiu Hu hu.chengqiu@huawei.com
Fangzheng Miao miaofangzheng@huawei.com
Bao Li li.bao@huawei.com
Yiyang Liu liuyiyang16@huawei.com
Junlong Lyu lyujunlong@huawei.com
Lirui Jiang jianglirui1@huawei.com
Jun Wang hwjun.wang@huawei.com
Lingchao Zheng zhenglingchao@huawei.com
Jun Li lijun276@huawei.com
Yuwei Fan* fanyuwei2@huawei.com

Huawei

CANN-Transformer仓

AMLA论文

Thanks！

访问CANN开源社区

关注昇腾CANN公众号