MMU

MMU由N组Engine组成,每个Engine由一个Slice(矩阵切片模块),5个Fifo,M个PE,1个加法树组成

每个PE有T个计算通道.

暂时取N=3/4,M=4,T=2

左乘

$AS+E$, 设A为$m \times n$, S为$n \times \bar{n}$

```
for i in range(m/4): #最外层循环,A每次缓存4行
    a[0] = A[4*i]
    a[1] = A[4*i+1]
    a[2] = A[4*i+2]
    a[3] = A[4*i+3]
    for j in range(n/4N):#第二层循环,对单个PE,每次计算S的4行
        #单个PE计算 1*4  4*nbar
        #单个engin计算 4*4 4*nbar
        #整个MMU计算 4*4N 4N*nbar
        s[0] = S[4*j]
        s[1] = S[4*j+1]
        s[2] = S[4*j+2]
        s[3] = S[4*j+3]
        transrows = slice(s_T)
        #对于Frodo,transrow有40行,对于Scloud128/192,有16行,对于Scloud256,有22行(为什么会有11这种数
字)
        #准备4个累加器吧,写成类cache结构
        #4个PE用同一个transrows计算A的4行
        #不同Engine之间,计算S的不同行
        #结果为B的一行
```

输入带宽:

S: 输入S 1列的4N个，输入nbar次

A:一次内层循环需要4*4N个A。每个PE接A一行的4个

输出带宽

写成类cache结构只需最后写入(每轮外循环输出一次)

研究一下过程的B/E

每次其实就是计算了4*nbar

最多11种结果而已

整个过程只换A和S，B/E不动

右乘

$S'A+E'$, 设S'为$\bar{m} \times m$, A为$m \times n$

```
for i in range(m/4): #最外层循环,A每次缓存4行
    a[0] = A[4*i]
    a[1] = A[4*i+1]
    a[2] = A[4*i+2]
    a[3] = A[4*i+3]
    a = a^T
    #一次进行 mbar * 4 4*n
    #因此一次其实只能看到S'的4列
    for j in range(n/4N):#第二层循环,对单个PE,每次计算A的1列,一个transrow应该来自S'的一行
        #单个PE计算mbar *4  4*1
        #单个engine计算 mbar*4 4*4
        #整个MMU计算 mbar*4 4*4N
        s[0] = S[4*j]
        s[1] = S[4*j+1]
        s[2] = S[4*j+2]
        s[3] = S[4*j+3]
        transrows = slice(s)
        #对于Frodo,transrow有40行,对于Scloud128/192,有16行,对于Scloud256,24行
        #准备4个累加器吧,写成类cache结构
        #4个PE用同一个transrows计算A的4个不同列
        #不同Engine之间,transrows实际上是共用的,计算A的不同列
```

输入带宽

每次输入S'的1行的4个数，然后所有的共用，输入mbar次

每个PE需要A的一列的4个，一下要4N个

输出带宽

输出4N个一次

内层循环就需要输出,但是,内层循环的暗含循环每次严格4个累加结果

研究一下过程的B/E

每轮的最终结果是mbar*n（其实就是全部）

单个PE看到mbar个，单个engine看到mbar*4个

所以在一次transrow中，每个PE内部其实很简单，关键是算完后mbar*4N个的替换

double一下（）

计算时输出上次结果

有点幽默了

方案1： 寄存器double

4N*12*16bit = 3072!

方案2： 类cache结构

甚至能减少原需的寄存器，但是要考虑好替换策略，不然对latency影响很大

方案3： 大带宽B/S

需要伴随较为复杂的存储规划、地址生成

方案4： 不使用fifo,直接5个计算单元

lut的开销应该会很大，并且和hash速度严重不匹配（但是也许可以加速hash?）

engine数减半就可以了（但此时1344的差距比带fifo的大一点）

取engine=3倒是合理一点，但是控制麻烦

目前应该方案4更好一点

存储

S

以Frodo为主考虑，使用8个独立的ram，8*32*336（256+128=384）

位宽为32（4个数据）

在此基础上，考虑Scloud，为匹配MMU，我们一个32位存8个数据（即每个按4bit存储）

Scloud256的S还多了4列，之后找额外的空间存?

E/B

当前算法对E和B其实没有那么敏感，因为替换的不是非常频繁

类cache结构需要探索一下

其他的存储其实无所谓?

如果能实现mbar*4N个数据同时读取其实就无所谓了

使用16个独立的ram，16*128*84 -> 16*64*168

Scloud256多了4列