中国科学院大学

试 题 专 用 纸

课程编号：B0912022H

课程名称：智能计算系统

任课教师：郭崎、李威

姓名 孙家豪 学号 2023K8009929008 成绩

《深度学习处理器运算器设计》实验报告

一、实验目的

本实验旨在理解深度学习处理器（DLP）以矩阵乘法/矩阵乘向量等线性代数算子作为核心计算单元的加速思想，并掌握其关键矩阵运算器的基本设计方法与验证流程。

具体目标包括：

1) 理解深度学习处理器加速深度学习运算的原理，理解本实验和深度学习处理器基本模块间的关系。

2) 使用 Verilog 语言实现内积运算器，理解内积运算器的基本结构，掌握基础的硬件设计流程。

3) 在内积运算器的基础上，使用 Verilog 语言实现矩阵乘向量运算器，加深对深度学习处理器加速原理的理解。

4) 在内积运算器、矩阵乘向量运算器的基础上，使用 Verilog 语言实现矩阵乘法运算器，加深对矩阵运算单元的理解。

二、实验内容与原理

1. 实验内容概述

实验采用“由简到繁”的设计路线：先给出内积运算器示例用于熟悉 Verilog 与仿真调试流程，再引导完成矩阵乘向量运算器与矩阵乘法运算器设计，从而掌握 DLP 核心运算部件的基本设计方法。

2. 内积运算器

内积运算器完成长度为 4 的向量点积。其硬件结构要点如下：

① 输入寄存，当 enable 有效时，在时钟上升沿将 4 组激活值与权重同步锁存到输入寄存器。

② 并行乘法+加法树累加：同一周期内 4 个乘法器并行产生 4 个 32 位乘积，再通过加法树完成累加

③ 两拍固定流水：第一拍完成采样与寄存，第二拍完成累加结果锁存并输出，因此结果固定延迟两拍；且该原型不提供“结果有效”信号，需要上层控制器自行对齐时序。

3. 矩阵乘向量运算器

矩阵乘向量实现每周期完成 4×4 权重矩阵与 4 维向量的乘法。实现思路：并行实例化 4 个内积运算器，每个内积负责计算输出向量的一行结果。

关键原理是广播：同一个输入激活向量同时送入 4 个内积单元，而 4 行权重分别送到对应内积单元。这样相比“4 个分离内积单元分别取数”，外部数据接口更省、同等存储带宽下运算密度更高。

接口与控制信号包含 clk/reset/enable，并采用 64 位输入向量、256 位权重矩阵与 128 位输出结果向量。

4. 矩阵乘法运算器

矩阵乘法运算器实现 4*4 矩阵与 4*4 矩阵的乘法，输出 4*4 结果矩阵。实现结构如下：

通过实例化 4 个矩阵乘向量运算器（等价于 16 个内积运算器）完成。各实例共享同一套 cl

k/reset/enable 与广播的权重矩阵，但分别接收不同批次/行的激活向量。

5. 功能仿真

实验使用 Verilator 将 Verilog 与 C++ 测试程序编译为仿真可执行文件，并生成波形文件 waveform.vcd 供 GTKWave 查看。

三、实验环境

运行环境： WSL 下的 Ubuntu
操作系统版本：Ubuntu 22.04 LTS
工程工作目录：/mnt/c/CODE/dlp_ex_student
Verilator:Verilator 5.020（Debian 5.020-1）
GTKWave：GTKWave Analyzer v3.3.116

四、实验实现

1. Testbench 数据路径适配（WSL 工程目录）

为适配 WSL 下工程实际位置/mnt/c/CODE/dlp_ex_student，本实验对仿真 Testbench 的数据加载路径进行了统一修改，使其能够正确读取 data/目录下的数据文件：

① sim/tb_inner_product.cpp：将输入数据改为从 /mnt/c/CODE/dlp_ex_student/data/mv/* 加载；同时放宽数据量检查逻辑，允许读取到的激活值与权重数据数量 ≥ 4，以便从现有数据文件中取前 4 个元素进行内积验证。

② sim/tb_matrix_vector.cpp：将输入数据改为从 /mnt/c/CODE/dlp_ex_student/data/mv/* 加载，用于矩阵 × 向量运算验证。

③ sim/tb_matrix_mult.cpp：将输入数据改为从 /mnt/c/CODE/dlp_ex_student/data/mm/* 加载，用于矩阵 × 矩阵运算验证。

2. 数据集整理

① data/mv/（矩阵*向量数据集）：新增 config、neuron、weight、result 四类文件，用于矩阵乘向量测试。

② data/mm/（矩阵*矩阵数据集）：新增 config、neuron、weight、result 四类文件；并对 weight 的存储顺序进行了调整，使其与 DUT 在实现中使用的行/列顺序一致，从而避免由于行列主序不一致导致的结果转置或错位问题。

3. 矩阵乘向量运算器

并行例化 4 个内积单元并完成端口映射。

```
inner_product_4x16 ipu0 (
    .clk(clk),
    .reset(reset),
    .enable(enable),
    .activations(activations),
    .weights(weights[0]),
    .result(results[0])
);
inner_product_4x16 ipu1 (
    .clk(clk),
    .reset(reset),
    .enable(enable),
    .activations(activations),
    .weights(weights[1]),
    .result(results[1])
);
inner_product_4x16 ipu2 (
    .clk(clk),
    .reset(reset),
    .enable(enable),
    .activations(activations),
    .weights(weights[2]),
    .result(results[2])
);
inner_product_4x16 ipu3 (
    .clk(clk),
    .reset(reset),
    .enable(enable),
    .activations(activations),
    .weights(weights[3]),
    .result(results[3])
);
```

4. 矩阵乘法运算器

并行例化 4 个内积单元并完成端口映射。

```
matrix_vector_mult_4x4x16 mxv0 (
    .clk(clk),
    .reset(reset),
    .enable(enable),
    .activations(activations[0]),
    .weights(weights),
    .results(results[0])
);
matrix_vector_mult_4x4x16 mxv1 (
    .clk(clk),
    .reset(reset),
    .enable(enable),
    .activations(activations[1]),
    .weights(weights),
    .results(results[1])
);
matrix_vector_mult_4x4x16 mxv2 (
    .clk(clk),
    .reset(reset),
    .enable(enable),
    .activations(activations[2]),
    .weights(weights),
    .results(results[2])
);
matrix_vector_mult_4x4x16 mxv3 (
    .clk(clk),
    .reset(reset),
    .enable(enable),
    .activations(activations[3]),
    .weights(weights),
    .results(results[3])
);
```

五、实验步骤与复现流程

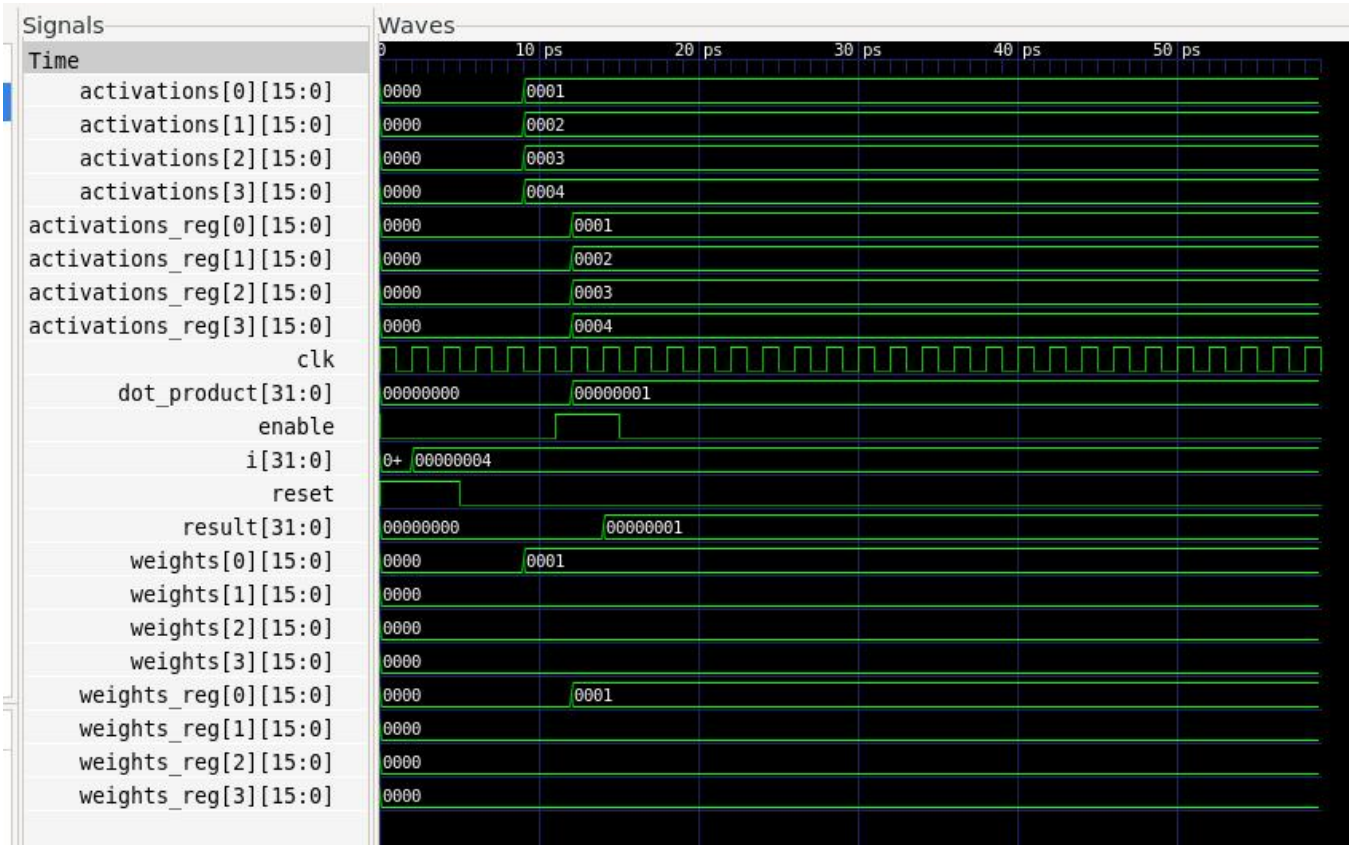
1. **进入工程仿真目录**
   cd /mnt/CODE/dlp_ex_student/sim
2. **编译命令：**
   make inner_product_4x16
   make matrix_vector_mult_4x4x16
   make matrix_mult_4x4x4x16
   作用：使用 Verilator 将 Verilog 代码编译成 C++可执行文件
3. **运行测试命令**
   ./obj_dir/inner_product_4x16/Vinner_product_4x16 # 测试内积运算器
   ./obj_dir/matrix_vector_mult_4x4x16/Vmatrix_vector_mult_4x4x16 # 测试矩阵乘向量
   ./obj_dir/matrix_mult_4x4x4x16/Vmatrix_mult_4x4x4x16 # 测试矩阵乘法
   作用：运行编译好的测试程序，验证硬件模块功能是否正确，对比结果
4. **查看波形命令**
   gtkwave wave_inner.vcd & # 查看内积运算器波形
   gtkwave wave_matrix_vector.vcd & # 查看矩阵乘向量波形
   gtkwave wave_matrix_mult.vcd & # 查看矩阵乘法波形

六、实验结果

1. **内积运算器：**

```
=== 结果验证 ===
实际结果（十进制）：1
预期结果（十进制）：1
验证结果：通过
```

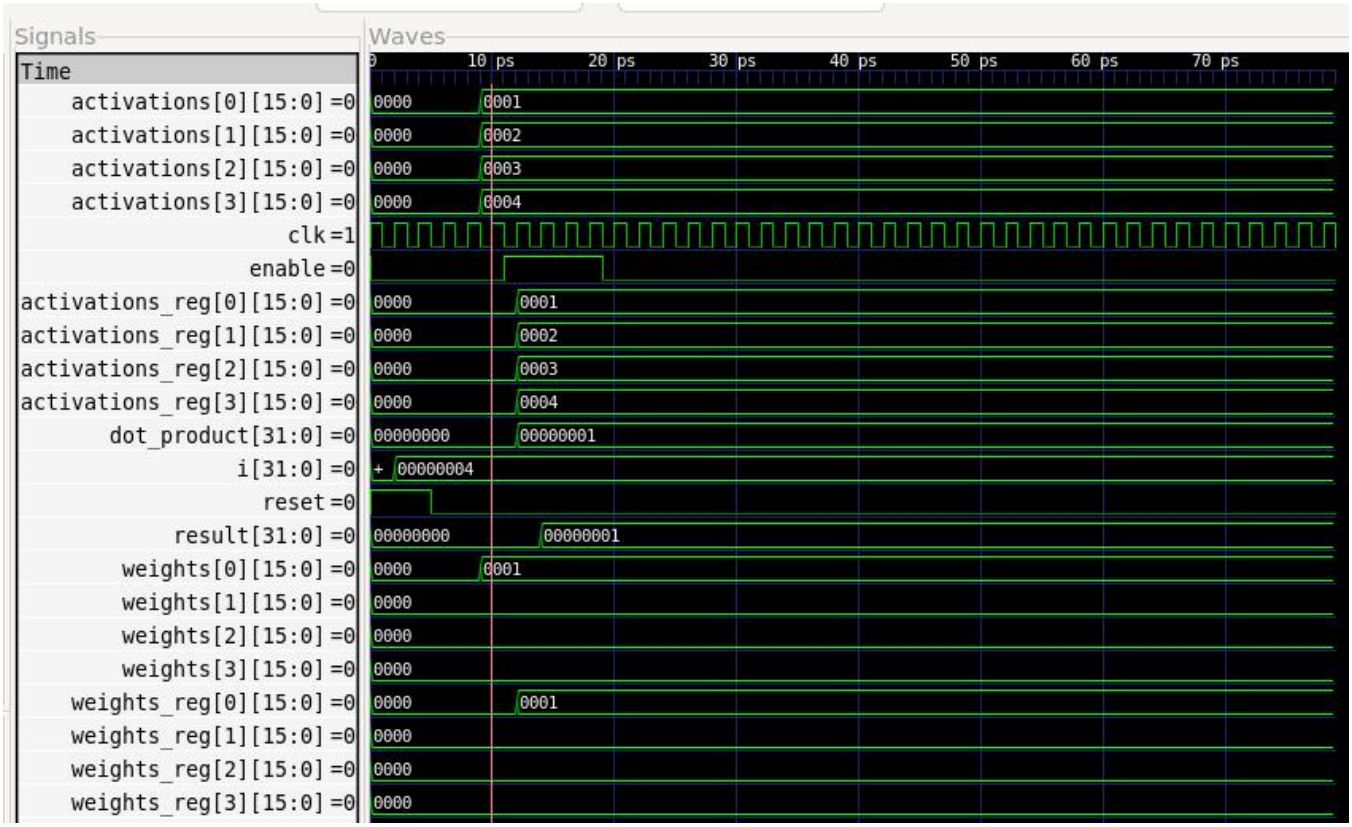
仿真结果如下：



2. **矩阵乘向量**

```
=== 输入数据 ===
矩阵（weights，4x4）：
1       0       0       0
0       1       0       0
0       0       1       0
0       0       0       1
向量（activations）：1 2 3 4
=== 结果验证 ===
结果1：实际=1，预期=1 [通过]
结果2：实际=2，预期=2 [通过]
结果3：实际=3，预期=3 [通过]
结果4：实际=4，预期=4 [通过]
```

仿真波形：



3. 矩阵乘法

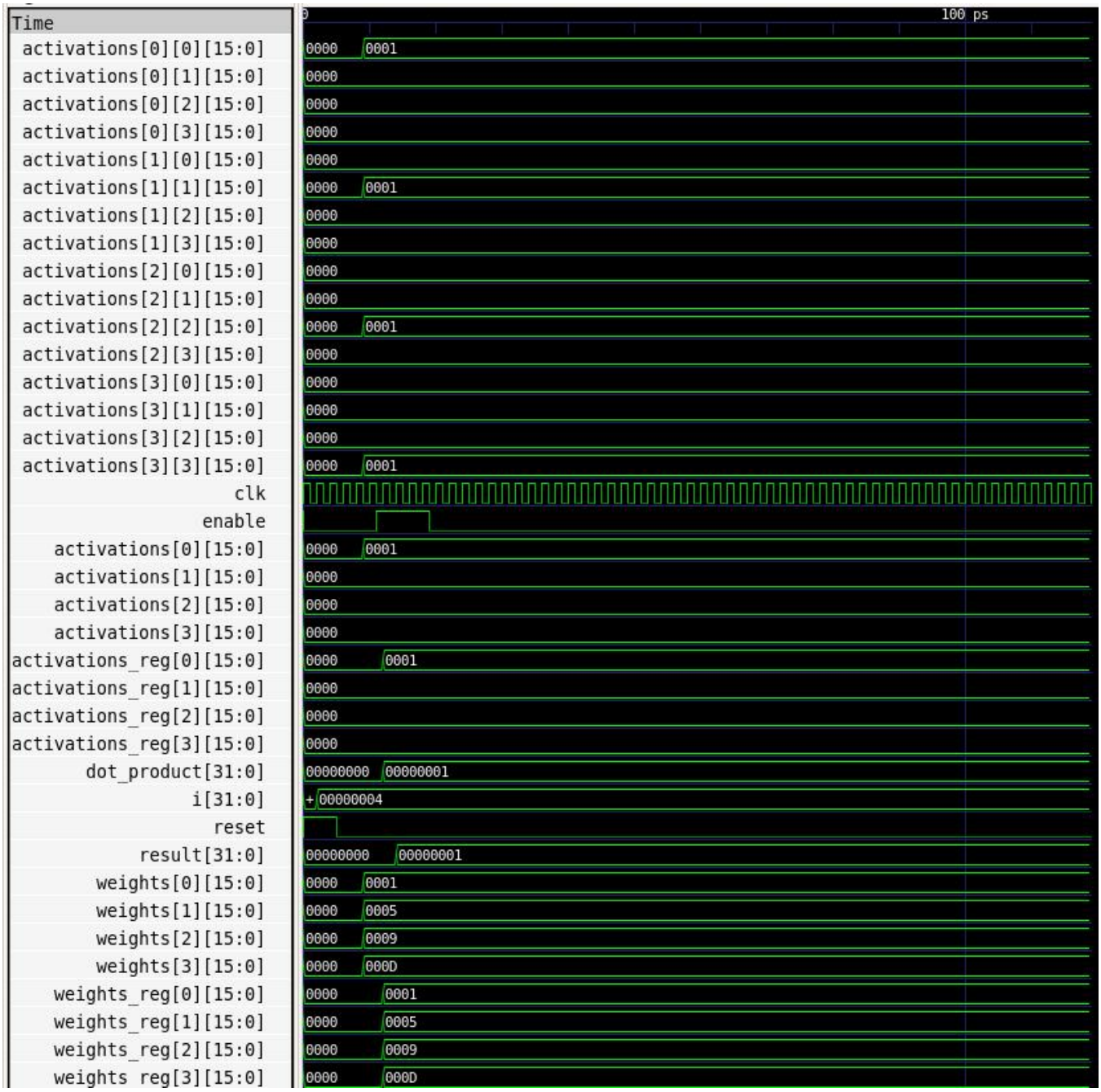
```
=== 输入数据 ===
激活值矩阵 (activations, 4x4) :
1       0       0       0
0       1       0       0
0       0       1       0
0       0       0       1

权重矩阵 (weights, 4x4) :
1       5       9       13
2       6       10      14
3       7       11      15
4       8       12      16

=== 结果验证 ===
结果[0][0]: 实际=1, 预期=1 [通过]
结果[0][1]: 实际=2, 预期=2 [通过]
结果[0][2]: 实际=3, 预期=3 [通过]
结果[0][3]: 实际=4, 预期=4 [通过]
结果[1][0]: 实际=5, 预期=5 [通过]
结果[1][1]: 实际=6, 预期=6 [通过]
结果[1][2]: 实际=7, 预期=7 [通过]
结果[1][3]: 实际=8, 预期=8 [通过]
结果[2][0]: 实际=9, 预期=9 [通过]
结果[2][1]: 实际=10, 预期=10 [通过]
结果[2][2]: 实际=11, 预期=11 [通过]
结果[2][3]: 实际=12, 预期=12 [通过]
结果[3][0]: 实际=13, 预期=13 [通过]
结果[3][1]: 实际=14, 预期=14 [通过]
结果[3][2]: 实际=15, 预期=15 [通过]
结果[3][3]: 实际=16, 预期=16 [通过]

验证结果: 全部通过
```

仿真波形：



4. 结论

所有信号波形都符合预期，时钟信号工作时，呈现周期性翻转；复位信号值为 1 时，输入输出值都为 0；两个输入信号所显示的值与输入文件保持一致；输出值 result 与输入值进行相应的运算得到的最终结果一致。

七、问题记录与解决过程

1. Verilator 编译时报错”No rule to make target 'tb_inner_product.cpp”

现象：在 sim/obj_dir/...目录里 make 找不到 tb_innnr_product.cpp

原因：Makefile 在子目录编译时按相对路径找 TB，导致找不到源文件。

解决：在 sim/Makefile 里用 SIM_DIR 显式路径指定 TB 源文件路径。

2. 矩阵乘法大量 FAIL，但是对角线 PASS

现象：输入 A=单位矩阵时，输出像权重矩阵的转置（对角线正确，非对角线错位）。

原因：行列主序/索引对调：可能是 TB 打包 weights 时把 (r,c) 写成 (c,r)，或 Verilog 输出拼接 results[row][col]与 TB 解包顺序不一致。

解决：统一 row-major 约定；并调整 data/mm/weight 的顺序或修正打包/拼接索引。

3. WSL 路径与工程根目录问题

现象：硬编码/home/...找不到数据

解决：改成自身路径/mnt/c/CODE/dlp_ex_student/data/...

八、思考与总结

1. 对比分析内积运算器、矩阵乘向量运算其和矩阵乘法运算器内部的乘法器数量、加法器数量和对外数据接口数量，计算并对比每种设计中这些元素的比例。如果增大三种运算器的规模（例如，向量长度从 4 增加到 16 乃至 128），这些设计的比例分别会发生什么样的变化？

（1）内积运算器：

① 乘法器：N
② 加法器：N-1
③ 外部数据：2N
④ 比例：乘法器/输入数 = 0.5

（2）矩阵*向量

① 乘法器：N^2
② 加法器：N(N-1)
③ 外部数据：$N+N^2$
④ 比例：乘法器/输入数 = N/N+1

（3）矩阵*矩阵

① 乘法器：N^3
② 加法器：$N^2(N-1)$
③ 外部数据：N^2+N^2
④ 比例：乘法器/输入数 = N/2

当规模从 4 增大时：

内积：计算比例基本不变；矩阵*向量的计算比例逐渐逼近 1；矩阵*矩阵的计算比例随 N 线性变大，体现加速器优势。

2. 请说明深度学习处理器加速深度学习计算的基本原理是什么？

1. 用大量的乘加运算阵列提升吞吐
2. 利用矩阵运算的数据复用，提高运算能力
3. 把数据存储到片上存储中，减少对外部的频繁访问。