# Optimizing FPGA-based Accelerator Design for Deep Convolutional Neural Networks

作者：张宸 孙广宇 关义金 丛京生

会议： *International Symposium on Field-Programmable Gate Arrays (FPGA)*

(Best Paper Award Nomination)

年份：2015

被引量：427

计算机 张德正 19112012

# 作者简介

- 张宸
  - 北京大学 计算机系统结构专业博士研究生

- 关义金
  - 北京大学 计算机系统结构专业博士研究生

- 孙广宇
  - 高能效计算与应用中心

- 丛京生
  - UCLA协理副教务长，北大高能效计算中心主任

# 被引情况

- 百度学术：427

**Optimizing fpgabased accelerator design for deep convolutional neural networks**

来自 ResearchGate | ♡ 喜欢 0　阅读量：1436

作者：C. Zhang，P. Li，G. Sun...

摘要：Convolutional neural network (CNN) has been widely employed for image recognition because it can achieve high accuracy by emulating behavior of optic nerves in living creatures. Recently, rapid growth of modern applications based on deep learning algorithms has further improved research and implementations. Especially, various accelerators for deep CNN have been proposed based on FPGA platform because it has advantages of high performance, reconfigurability, and fast development round, etc. Although current FPGA accelerators 展开

关键词：acceleration　convolutional neural network　fpga　roofline model

DOI：10.1145/2684746.2689060

被引量：427

年份：2015

- 谷歌学术：1153

Optimizing fpga-based accelerator design for deep convolutional neural networks

C Zhang, P Li, G Sun, Y Guan, B Xiao... - Proceedings of the 2015 ..., 2015 - dl.acm.org

Convolutional neural network (CNN) has been widely employed for image recognition because it can achieve high accuracy by emulating behavior of optic nerves in living creatures. Recently, rapid growth of modern applications based on deep learning algorithms ...

☆ 99 被引用 1153 次　相關文章　[文獻分析]

- 微软学术：1144

Optimizing FPGA-based Accelerator Design for Deep Convolutional Neural Networks

2015 Field Programmable Gate Arrays | pp 161-170 | DOI: 10.1145/2684746.2689060

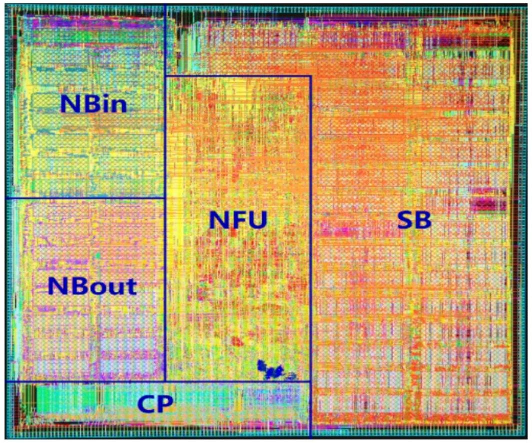
Chen Zhang [1], Peng Li [2], Guangyu Sun [1], Yijin Guan [1] +2

[1] Peking University, [2] University of California, Los Angeles

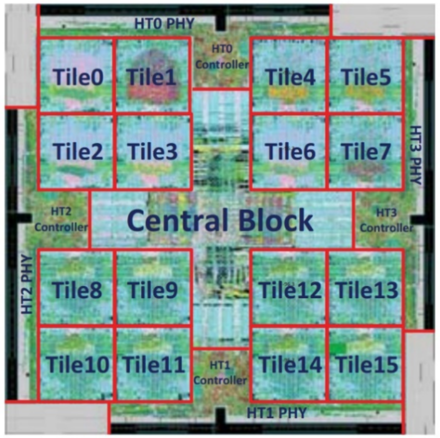
14 References　99 1,144 Citations*

# 摘要

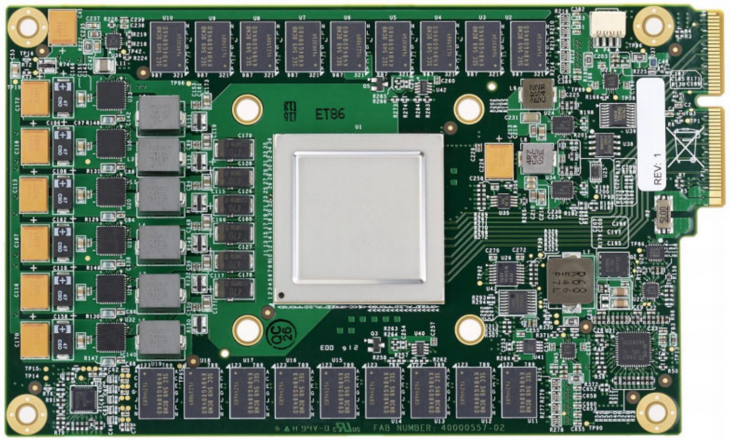
- 基于深度卷积神经网络的机器学习方法在很多应用领域取得了瞩目进步，但是在部署时存在以下困难：
  - 计算量大、算法困难
  - 使用GPU进行云端加速时，牺牲了实时性且能耗较高
- 因此工业界和学术界投入了大量的人力物力进行深度神经网络加速芯片设计




DianNao




DaDianNao



TPU

# 摘要

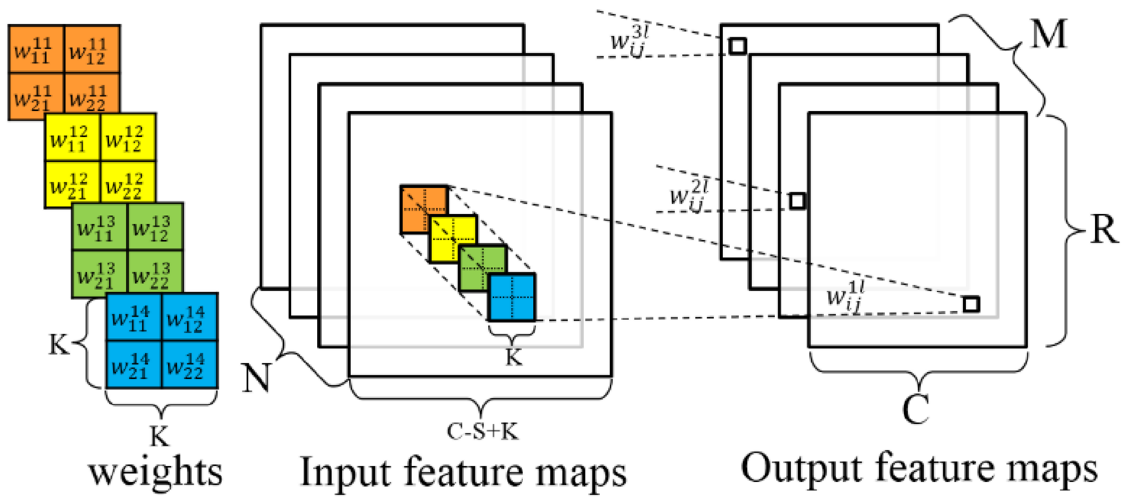
- 专用芯片在性能和功耗方面均超越传统GPU，但仍存在以下问题：
  - 设计周期长、前期投入大
  - 流片后架构和功能无法修改
  - 深度学习算法不断演进，算法的更新可能会导致处理器架构的更新

- 因此，工业界、学术界均倾向采用FPGA进行CNN应用加速
  - 腾讯云中心部署了FPGA计算资源
  - Xilinx 推出ALVEO系列板卡进行数据加速

- 但是基于FPGA的加速器存在着以下问题
  - 加速器的**计算吞吐量**无法和**内存带宽**很好的匹配
  - 在加速器上，有可能**计算资源没有充分利用**，要么**内存带宽没有被充分利用**

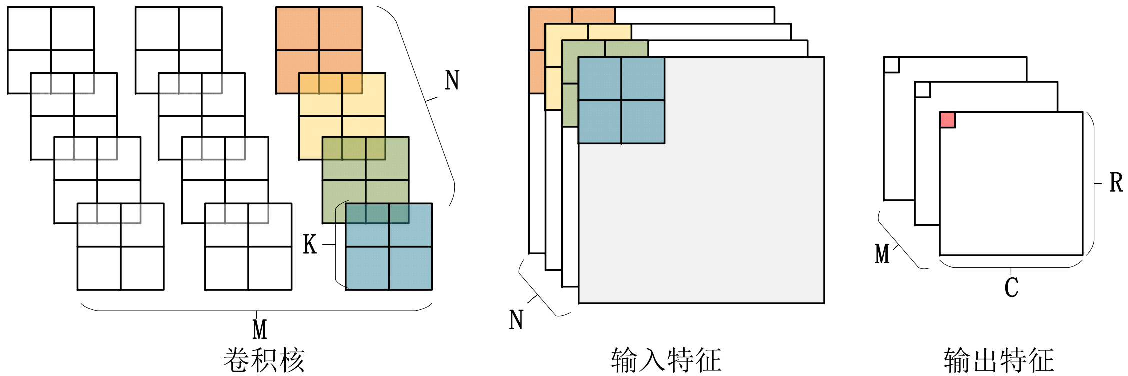
- 为了解决计算和访存不匹配的问题，本文做了以下工作点：
  - 使用Roofline **模型**对访存和计算瓶颈进行了刻画
  - 量化了计算吞吐量和内存带宽之间的关系
  - 使用**循环分块等**手段优化了计算吞吐量和访存之间的关系

# 卷积运算回顾




```
for(row=0; row<R; row++) {
 for(col=0; col<C; col++) {
  for(to=0; to<M; to++) {
   for(ti=0; ti<N; ti++) {
    for(i=0; i<K; i++) {
     for(j=0; j<K; j++) {
   L:   output_fm[to][row][col] +=
              weights[to][ti][i][j]*
              input_fm[ti][S*row+i][S*col+j];
} } } } } }
```

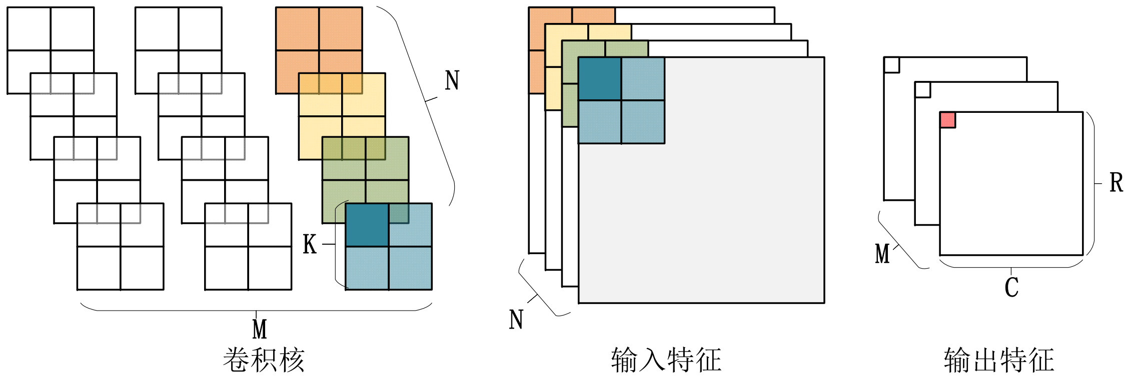
# 卷积运算回顾




```
for(row=0; row<R; row++) {
 for(col=0; col<C; col++) {
  for(to=0; to<M; to++) {
   for(ti=0; ti<N; ti++) {
    for(i=0; i<K; i++) {
     for(j=0; j<K; j++) {
    L:   output_fm[to][row][col] +=
            weights[to][ti][i][j]*
            input_fm[ti][S*row+i][S*col+j];
} } } } } }
```

- 最外的3层循环和输出特征图三个维度有关
- 最内的3层循环和每个输出特征点有关
- 例如要计算输出特征图的一个点（红点）需要彩色标出的特征图和权重

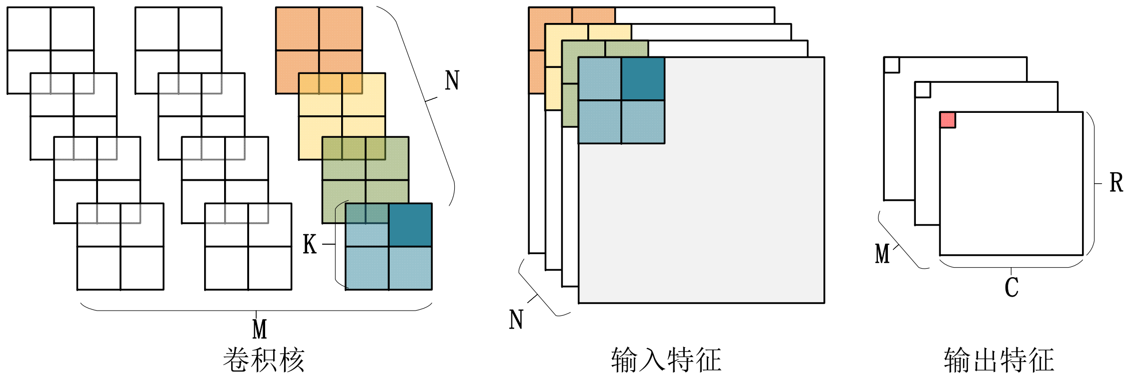
# 卷积运算回顾




```
for(row=0; row<R; row++) {
 for(col=0; col<C; col++) {
  for(to=0; to<M; to++) {
   for(ti=0; ti<N; ti++) {
    for(i=0; i<K; i++) {
     for(j=0; j<K; j++) {
    L:   output_fm[to][row][col] +=
            weights[to][ti][i][j]*
            input_fm[ti][S*row+i][S*col+j];
} } } } } }
```

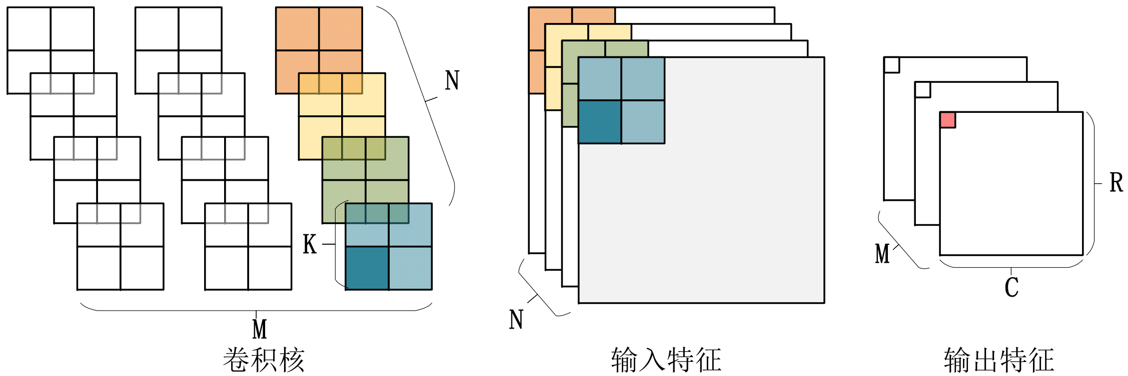
注：颜色深的部分是经过卷积运算的特征点/权重

# 卷积运算回顾



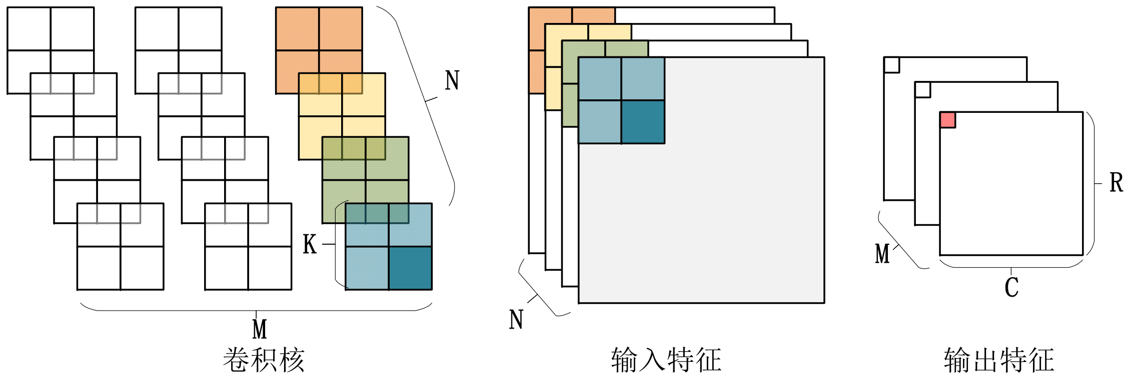

```
for(row=0; row<R; row++) {
 for(col=0; col<C; col++) {
  for(to=0; to<M; to++) {
   for(ti=0; ti<N; ti++) {
    for(i=0; i<K; i++) {
     for(j=0; j<K; j++) {
    L:   output_fm[to][row][col] +=
            weights[to][ti][i][j]*
            input_fm[ti][S*row+i][S*col+j];
} } } } } }
```

# 卷积运算回顾



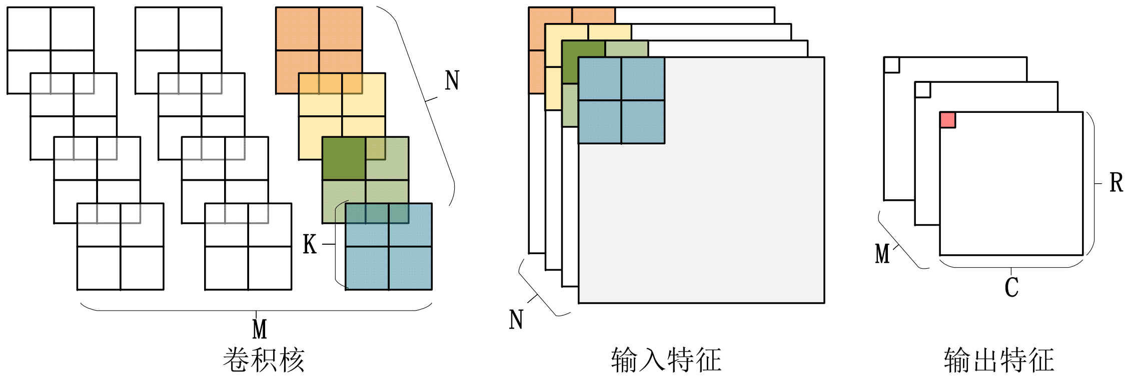

```
for(row=0; row<R; row++) {
 for(col=0; col<C; col++) {
  for(to=0; to<M; to++) {
   for(ti=0; ti<N; ti++) {
    for(i=0; i<K; i++) {
     for(j=0; j<K; j++) {
   L:   output_fm[to][row][col] +=
           weights[to][ti][i][j]*
           input_fm[ti][S*row+i][S*col+j];
} } } } } }
```

# 卷积运算回顾



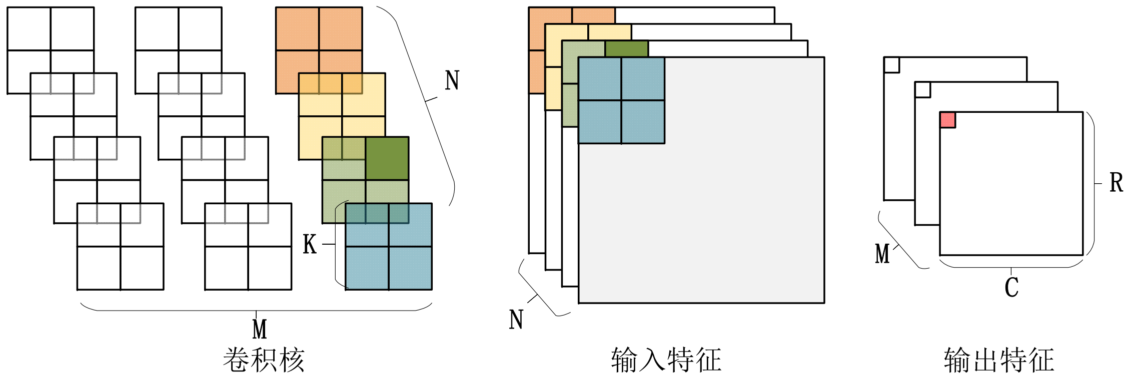

```
for(row=0; row<R; row++) {
 for(col=0; col<C; col++) {
  for(to=0; to<M; to++) {
   for(ti=0; ti<N; ti++) {
    for(i=0; i<K; i++) {
     for(j=0; j<K; j++) {
    L:   output_fm[to][row][col] +=
            weights[to][ti][i][j]*
            input_fm[ti][S*row+i][S*col+j];
} } } } } }
```

# 卷积运算回顾



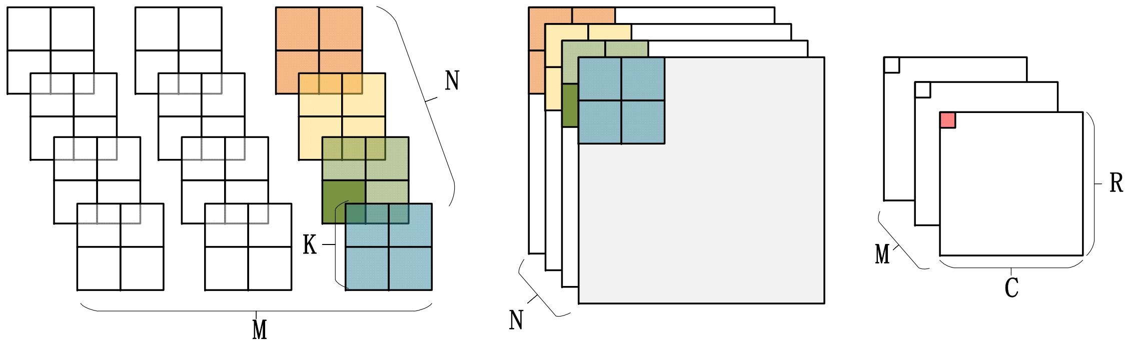

```
for(row=0; row<R; row++) {
 for(col=0; col<C; col++) {
  for(to=0; to<M; to++) {
   for(ti=0; ti<N; ti++) {
    for(i=0; i<K; i++) {
     for(j=0; j<K; j++) {
   L:    output_fm[to][row][col] +=
            weights[to][ti][i][j]*
            input_fm[ti][S*row+i][S*col+j];
} } } } } }
```

# 卷积运算回顾



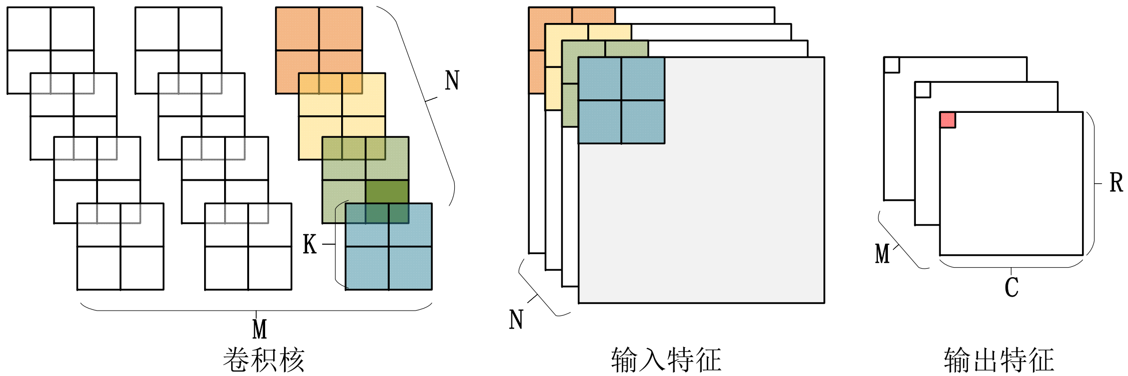

```
for(row=0; row<R; row++) {
 for(col=0; col<C; col++) {
  for(to=0; to<M; to++) {
   for(ti=0; ti<N; ti++) {
    for(i=0; i<K; i++) {
     for(j=0; j<K; j++) {
    L:    output_fm[to][row][col] +=
             weights[to][ti][i][j]*
             input_fm[ti][S*row+i][S*col+j];
} } } } } }
```

# 卷积运算回顾



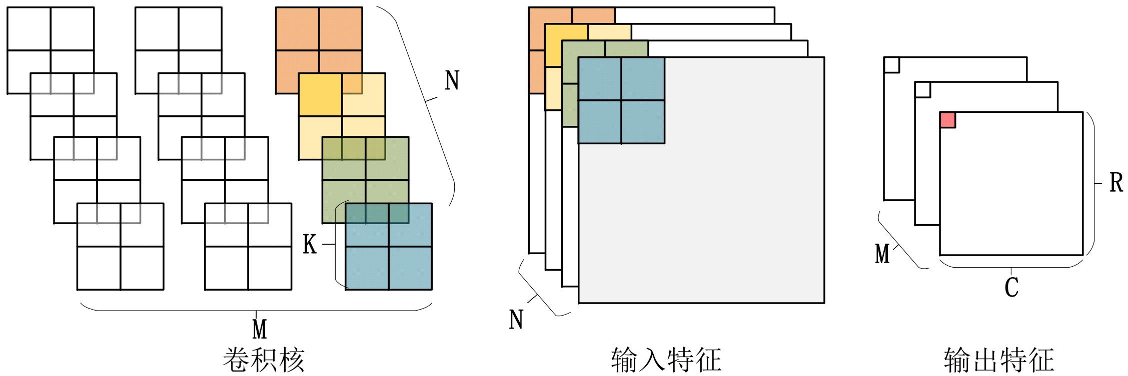

```
for(row=0; row<R; row++) {
 for(col=0; col<C; col++) {
  for(to=0; to<M; to++) {
   for(ti=0; ti<N; ti++) {
    for(i=0; i<K; i++) {
     for(j=0; j<K; j++) {
    L:    output_fm[to][row][col] +=
             weights[to][ti][i][j]*
             input_fm[ti][S*row+i][S*col+j];
} } } } } }
```

# 卷积运算回顾



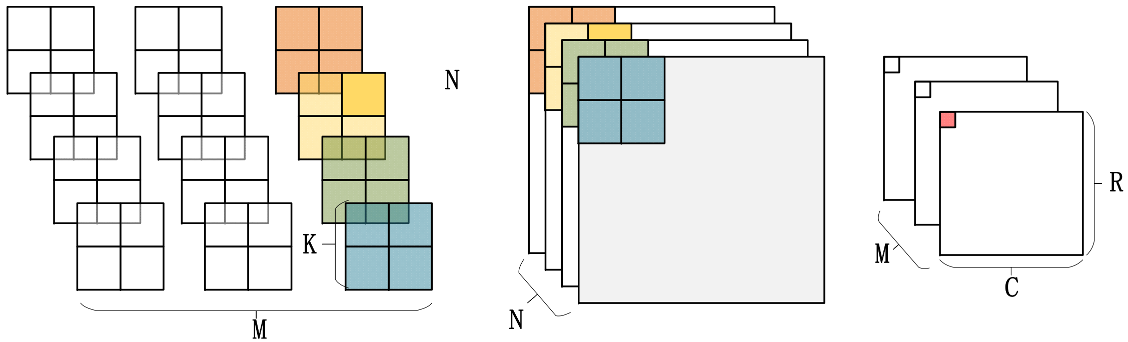

```
for(row=0; row<R; row++) {
 for(col=0; col<C; col++) {
  for(to=0; to<M; to++) {
   for(ti=0; ti<N; ti++) {
    for(i=0; i<K; i++) {
     for(j=0; j<K; j++) {
    L:    output_fm[to][row][col] +=
             weights[to][ti][i][j]*
             input_fm[ti][S*row+i][S*col+j];
} } } } } }
```

# 卷积运算回顾



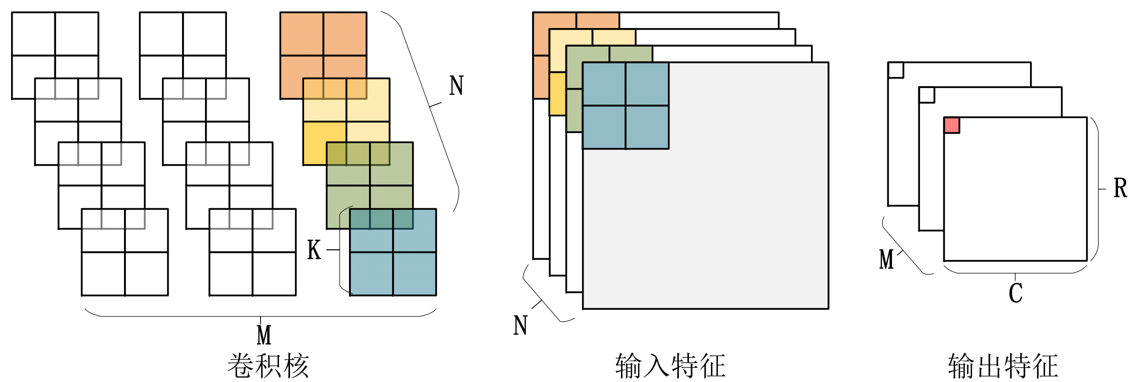

```
for(row=0; row<R; row++) {
 for(col=0; col<C; col++) {
  for(to=0; to<M; to++) {
   for(ti=0; ti<N; ti++) {
    for(i=0; i<K; i++) {
     for(j=0; j<K; j++) {
    L:   output_fm[to][row][col] +=
            weights[to][ti][i][j]*
            input_fm[ti][S*row+i][S*col+j];
} } } } } }
```

# 卷积运算回顾



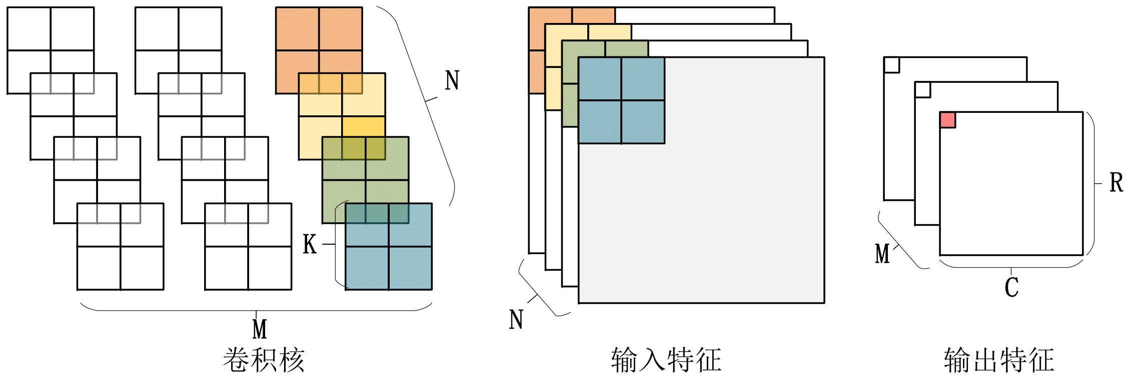

```
for(row=0; row<R; row++) {
 for(col=0; col<C; col++) {
  for(to=0; to<M; to++) {
   for(ti=0; ti<N; ti++) {
    for(i=0; i<K; i++) {
     for(j=0; j<K; j++) {
    L:   output_fm[to][row][col] +=
            weights[to][ti][i][j]*
            input_fm[ti][S*row+i][S*col+j];
} } } } } }
```

# 卷积运算回顾



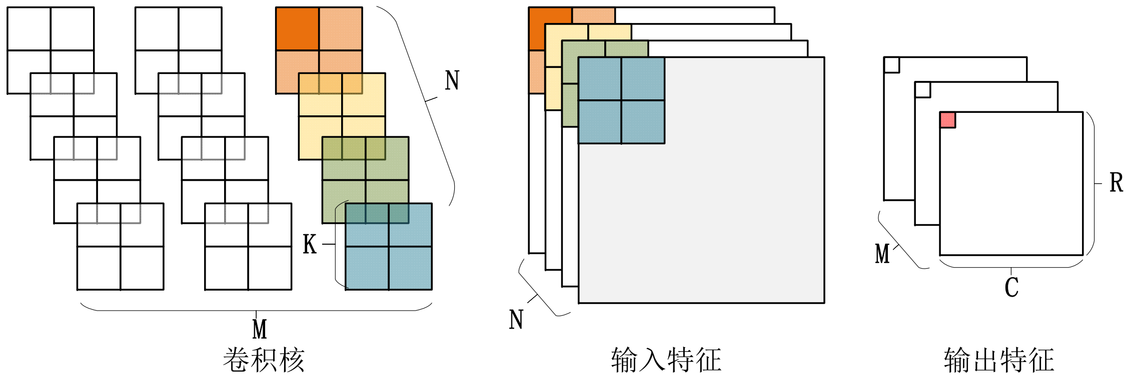

```
for(row=0; row<R; row++) {
 for(col=0; col<C; col++) {
  for(to=0; to<M; to++) {
   for(ti=0; ti<N; ti++) {
    for(i=0; i<K; i++) {
     for(j=0; j<K; j++) {
    L:   output_fm[to][row][col] +=
           weights[to][ti][i][j]*
           input_fm[ti][S*row+i][S*col+j];
} } } } } }
```

# 卷积运算回顾



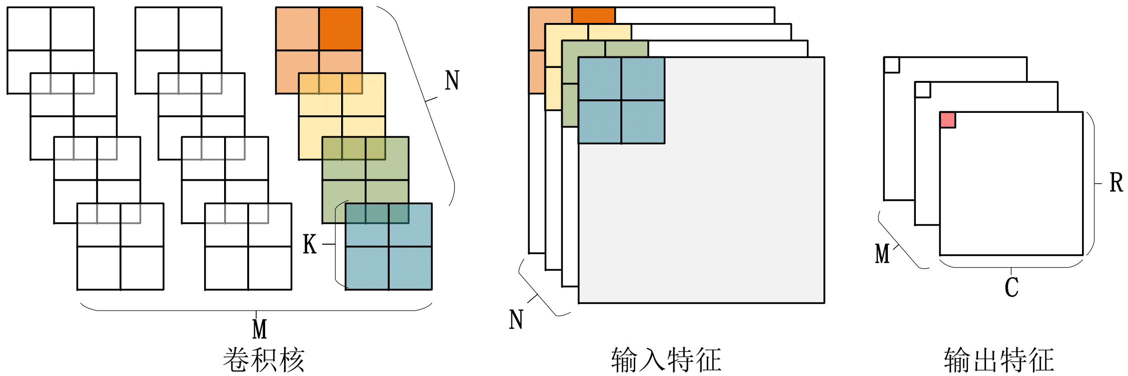

```
for(row=0; row<R; row++) {
 for(col=0; col<C; col++) {
  for(to=0; to<M; to++) {
   for(ti=0; ti<N; ti++) {
    for(i=0; i<K; i++) {
     for(j=0; j<K; j++) {
    L:   output_fm[to][row][col] +=
           weights[to][ti][i][j]*
           input_fm[ti][S*row+i][S*col+j];
} } } } } }
```

# 卷积运算回顾



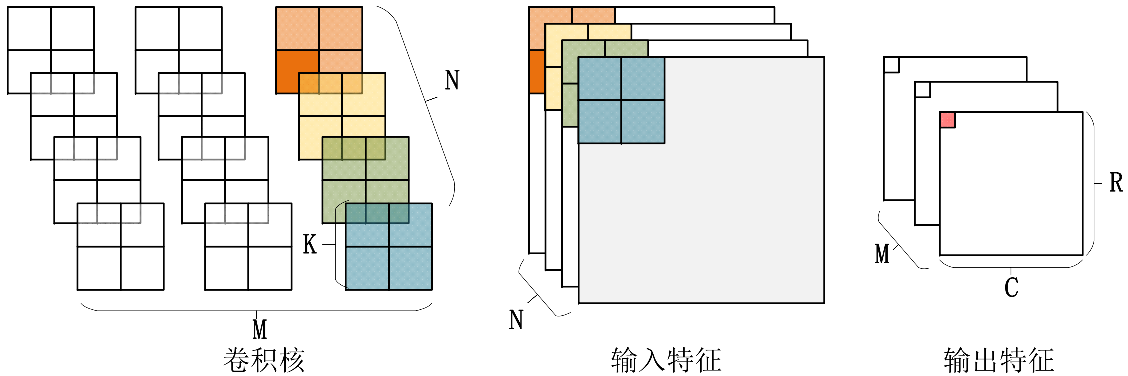

```
for(row=0; row<R; row++) {
 for(col=0; col<C; col++) {
  for(to=0; to<M; to++) {
   for(ti=0; ti<N; ti++) {
    for(i=0; i<K; i++) {
     for(j=0; j<K; j++) {
    L:   output_fm[to][row][col] +=
            weights[to][ti][i][j]*
            input_fm[ti][S*row+i][S*col+j];
} } } } } }
```

# 卷积运算回顾



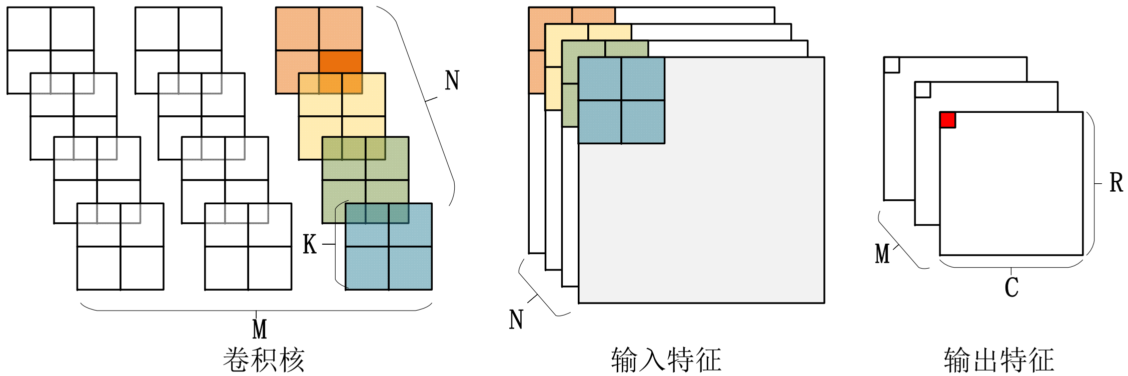

```
for(row=0; row<R; row++) {
 for(col=0; col<C; col++) {
  for(to=0; to<M; to++) {
   for(ti=0; ti<N; ti++) {
    for(i=0; i<K; i++) {
     for(j=0; j<K; j++) {
    L:    output_fm[to][row][col] +=
             weights[to][ti][i][j]*
             input_fm[ti][S*row+i][S*col+j];
} } } } } }
```

# 卷积运算回顾



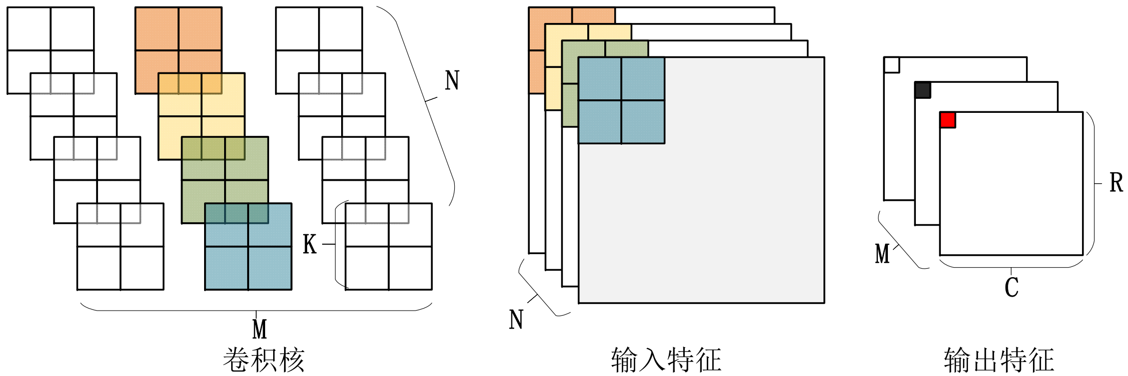

```
for(row=0; row<R; row++) {
 for(col=0; col<C; col++) {
  for(to=0; to<M; to++) {
   for(ti=0; ti<N; ti++) {
    for(i=0; i<K; i++) {
     for(j=0; j<K; j++) {
    L:   output_fm[to][row][col] +=
            weights[to][ti][i][j]*
            input_fm[ti][S*row+i][S*col+j];
} } } } } }
```

# 卷积运算回顾




```
for(row=0; row<R; row++) {
 for(col=0; col<C; col++) {
  for(to=0; to<M; to++) {
   for(ti=0; ti<N; ti++) {
    for(i=0; i<K; i++) {
     for(j=0; j<K; j++) {
    L:   output_fm[to][row][col] +=
           weights[to][ti][i][j]*
           input_fm[ti][S*row+i][S*col+j];
} } } } } }
```

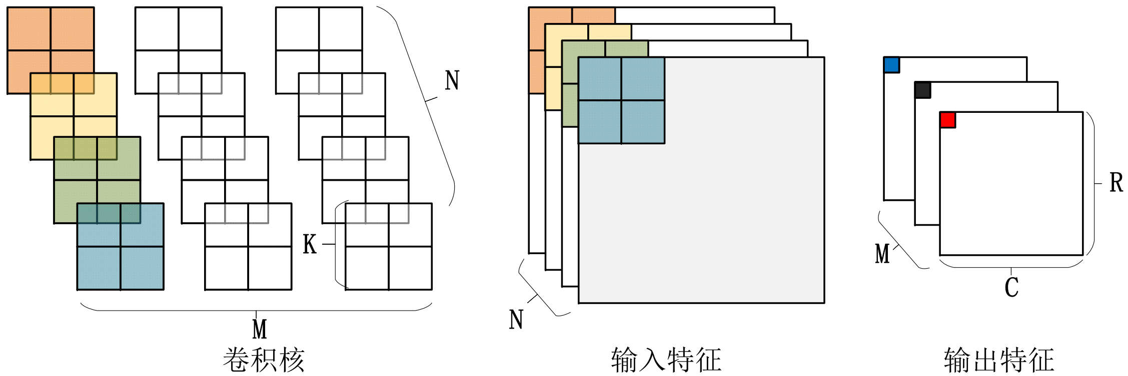
# 卷积运算回顾




```
for(row=0; row<R; row++) {
 for(col=0; col<C; col++) {
  for(to=0; to<M; to++) {
   for(ti=0; ti<N; ti++) {
    for(i=0; i<K; i++) {
     for(j=0; j<K; j++) {
    L:   output_fm[to][row][col] +=
           weights[to][ti][i][j]*
           input_fm[ti][S*row+i][S*col+j];
} } } } } }
```

- 接下来用类似的方式计算出第二个通道的特征点（黑色）

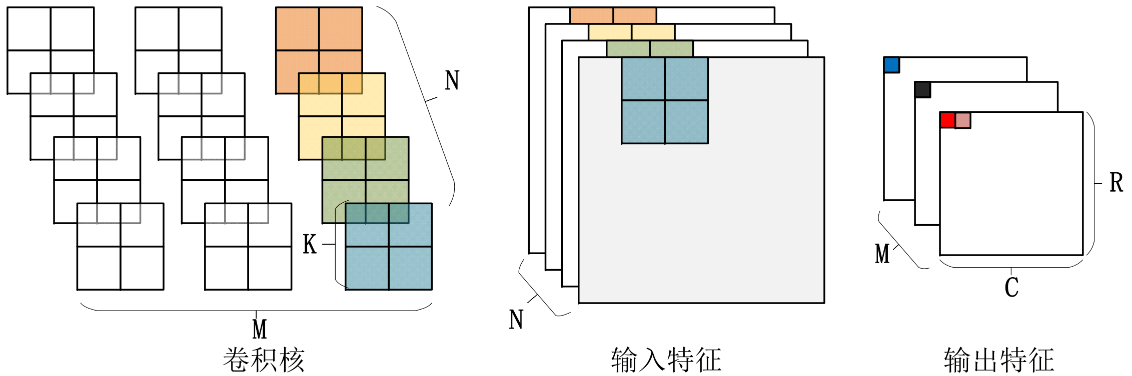
# 卷积运算回顾




```
for(row=0; row<R; row++) {
 for(col=0; col<C; col++) {
  for(to=0; to<M; to++) {
   for(ti=0; ti<N; ti++) {
    for(i=0; i<K; i++) {
     for(j=0; j<K; j++) {
    L:   output_fm[to][row][col] +=
            weights[to][ti][i][j]*
            input_fm[ti][S*row+i][S*col+j];
} } } } } }
```

- 接下来用类似的方式计算出第三个通道的特征点（蓝色）

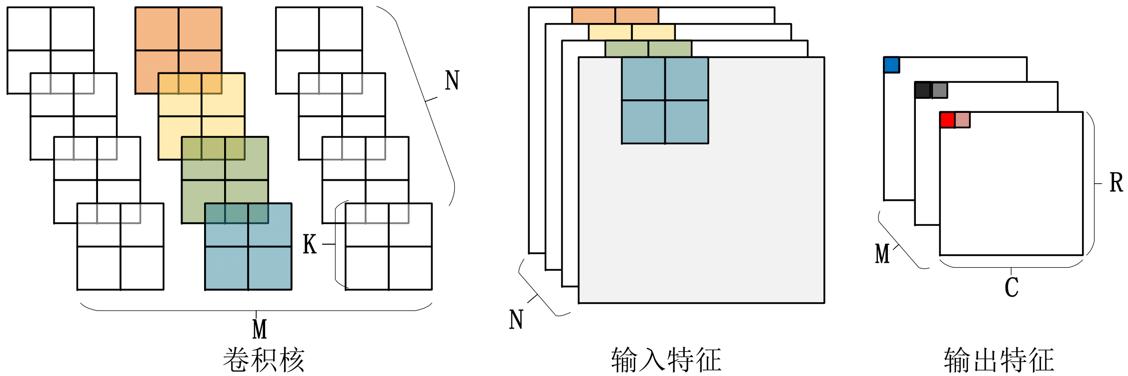
# 卷积运算回顾




```
for(row=0; row<R; row++) {
 for(col=0; col<C; col++) {
  for(to=0; to<M; to++) {
   for(ti=0; ti<N; ti++) {
    for(i=0; i<K; i++) {
     for(j=0; j<K; j++) {
    L:    output_fm[to][row][col] +=
             weights[to][ti][i][j]*
             input_fm[ti][S*row+i][S*col+j];
} } } } } }
```

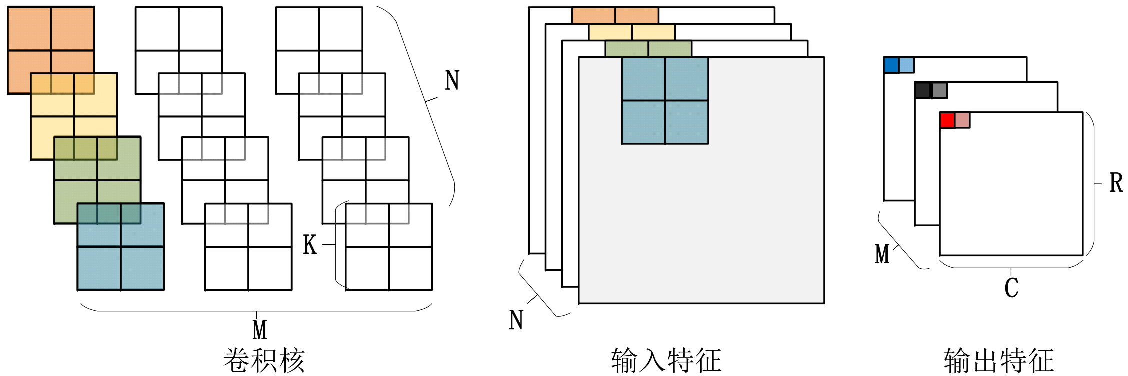
- 接下将滑动窗右移得到右侧的输出特征（粉色）

# 卷积运算回顾



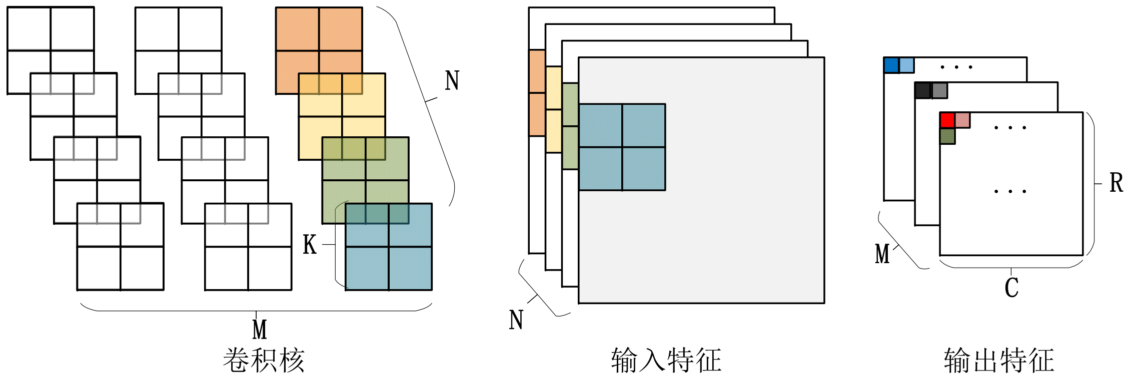

```
for(row=0; row<R; row++) {
 for(col=0; col<C; col++) {
  for(to=0; to<M; to++) {
   for(ti=0; ti<N; ti++) {
    for(i=0; i<K; i++) {
     for(j=0; j<K; j++) {
    L:   output_fm[to][row][col] +=
            weights[to][ti][i][j]*
            input_fm[ti][S*row+i][S*col+j];
} } } } } }
```

# 卷积运算回顾




```
for(row=0; row<R; row++) {
 for(col=0; col<C; col++) {
  for(to=0; to<M; to++) {
   for(ti=0; ti<N; ti++) {
    for(i=0; i<K; i++) {
     for(j=0; j<K; j++) {
    L:   output_fm[to][row][col] +=
           weights[to][ti][i][j]*
           input_fm[ti][S*row+i][S*col+j];
} } } } } }
```

# 卷积运算回顾

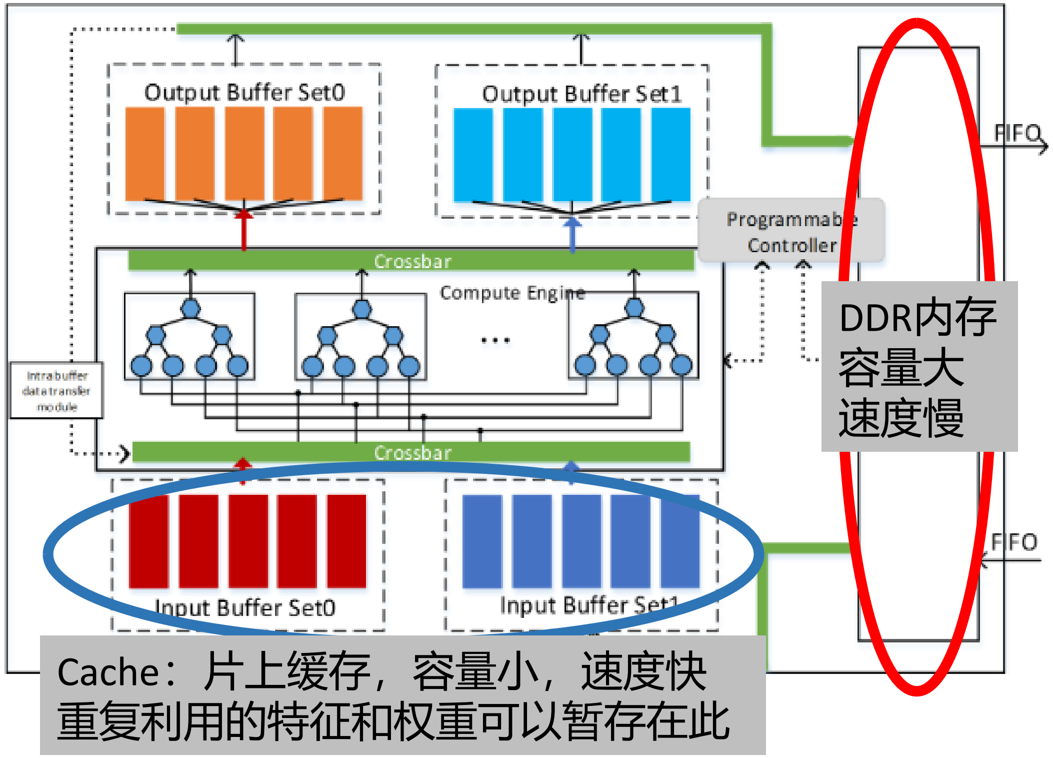


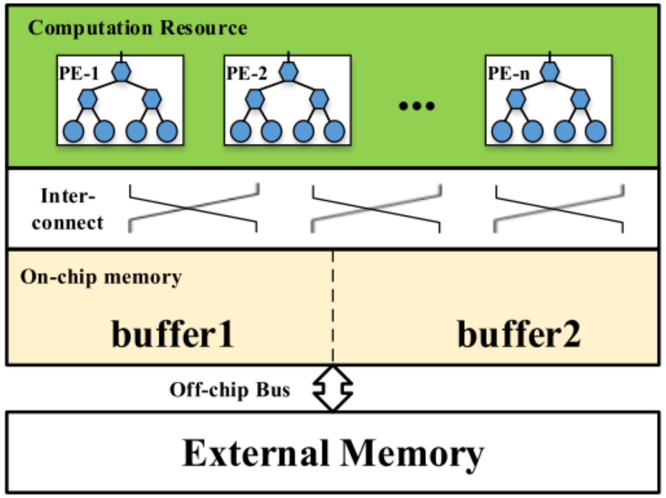

```
for(row=0; row<R; row++) {
 for(col=0; col<C; col++) {
  for(to=0; to<M; to++) {
   for(ti=0; ti<N; ti++) {
    for(i=0; i<K; i++) {
     for(j=0; j<K; j++) {
   L:    output_fm[to][row][col] +=
             weights[to][ti][i][j]*
             input_fm[ti][S*row+i][S*col+j];
} } } } } }
```

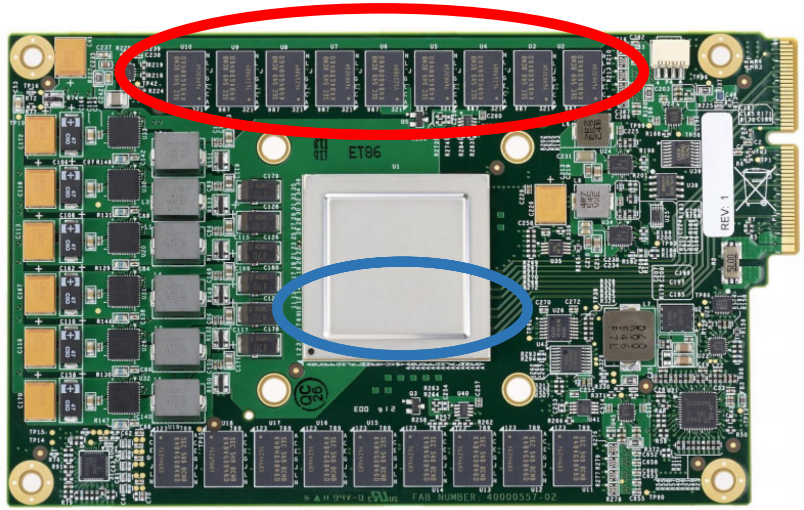
- 类似地可以得到整个地输出特征

# 输入特征和权重从哪里来？

- 加速器的存储层次如右上图：
  - 片外存储，容量大，速度慢（External Memory）
  - 片上存储，容量小，速度快（On-chip Memory）

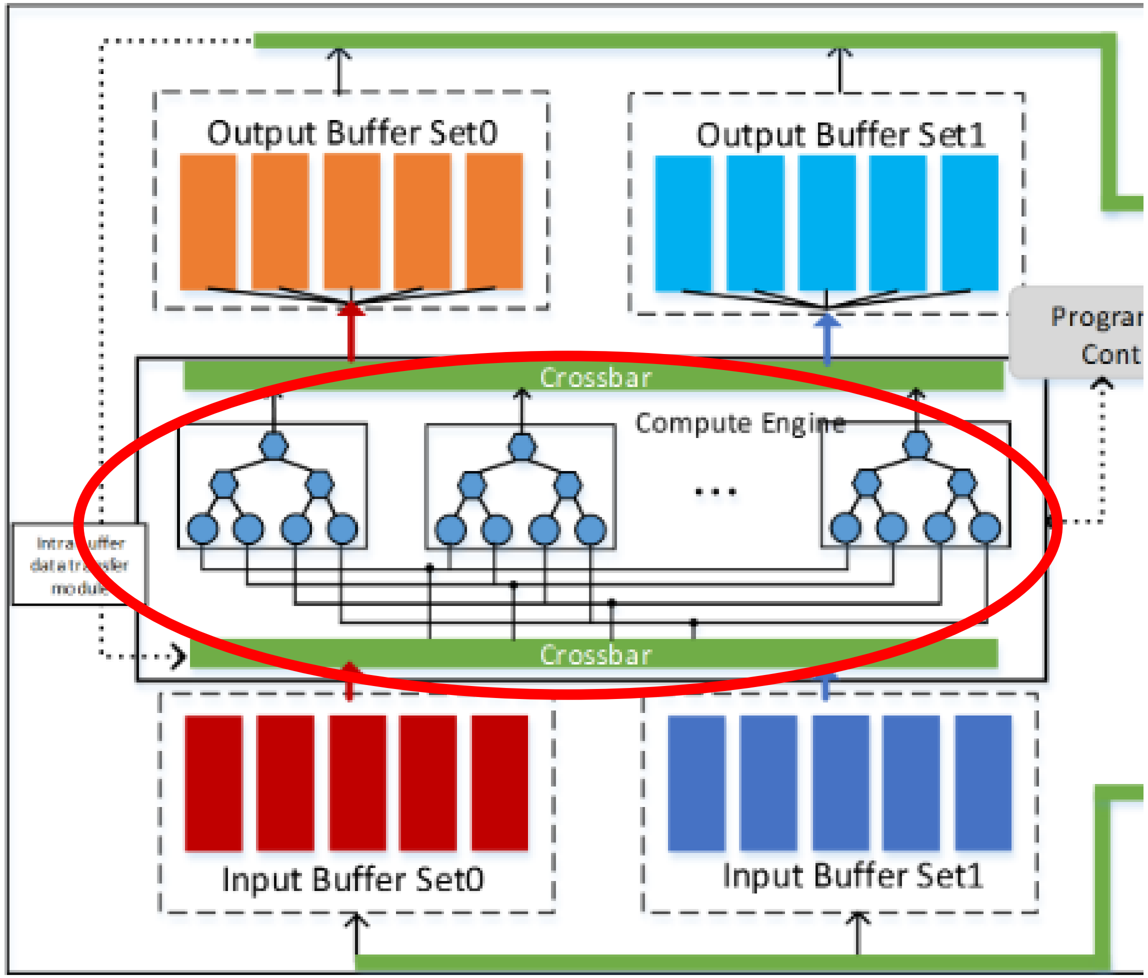


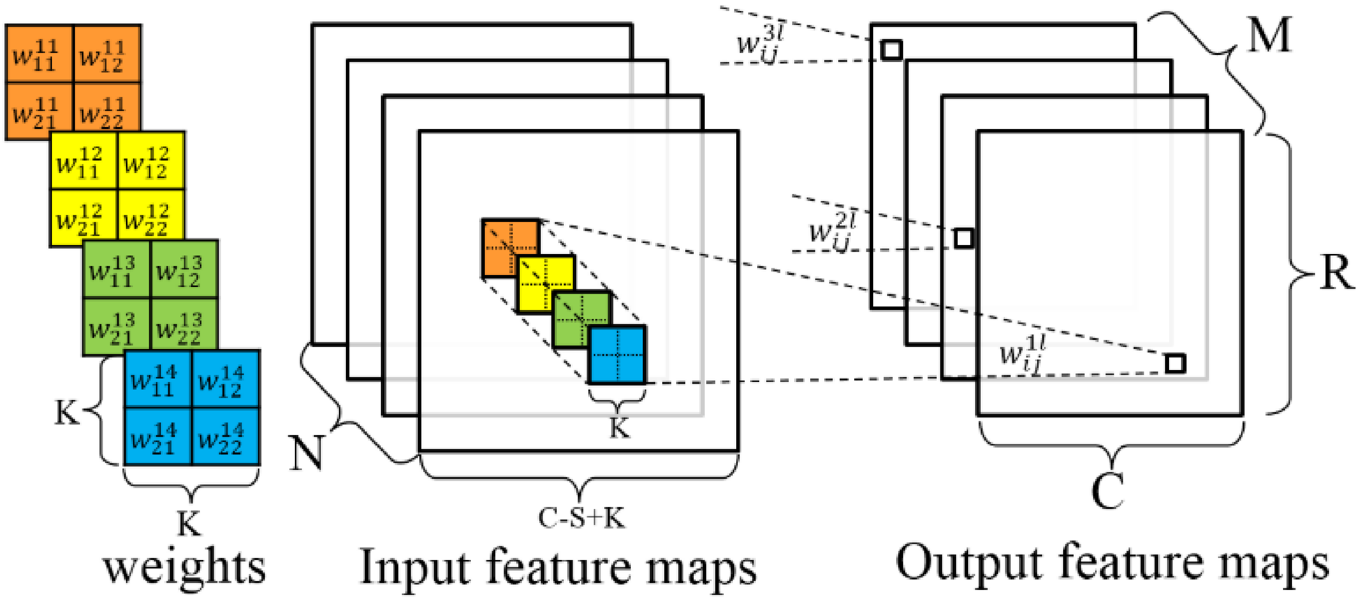
# 卷积算法的并行性

- 除了对**数据读写**的优化外，为了提高加速器的吞吐率，还应该充分利用**卷积运算的并行性**。
  - 能否同时进行多个卷积核的运算
  - 卷积核内的并行性的提取



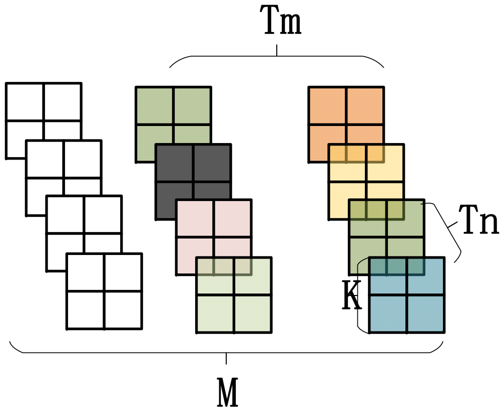
# 优化方式

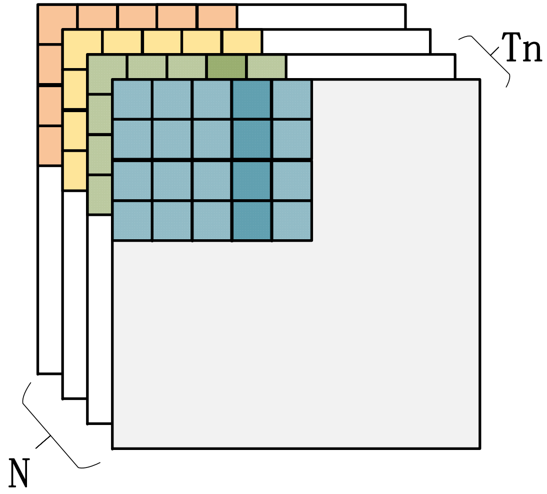
```
for(row=0; row<R; row+=Tr) {
 for(col=0; col<C; col+=Tc) {
  for(to=0; to<M; to+=Tm) {
   for(ti=0; ti<N; ti+=Tn) {
   //load output feature maps
   //load weights
   //load input feature maps

   //on-chip data computation
   for(i=0; i<K; i++) {
    for(j=0; j<K; j++) {
     for(trr=row; trr<min(row+Tr,R); trr++){
      for(tcc=col; tcc<min(col+Tc,C); tcc++){
       for(too=to; too<min(to+Tm,M); too++){
   #pragma HLS UNROLL
        for(tii=ti; tii<min(ti+Tn,N); tii++){
   #pragma HLS UNROLL
        L:  output_fm[too][trr][tcc] +=
              weights[too][tii][i][j]*
              input_fm[tii][S*trr+i][S*tcc+j];
   } } } } } }
} } } }
```

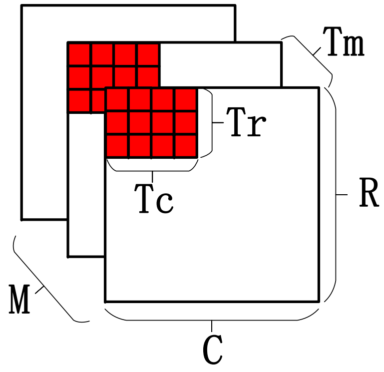



- 左图为论文提出的数据流（如何组织卷积运算）

# 优化方式




External data transfer
To be discussed in Section 3.2

On-chip data computation
To be discussed in Section 3.1

```
for(row=0; row<R; row+=Tr) {
 for(col=0; col<C; col+=Tc) {
  for(to=0; to<M; to+=Tm) {
   for(ti=0; ti<N; ti+=Tn) {
   //load output feature maps
   //load weights
   //load input feature maps

    //on-chip data computation
    for(i=0; i<K; i++) {
     for(j=0; j<K; j++) {
      for(trr=row; trr<min(row+Tr,R); trr++){
       for(tcc=col; tcc<min(col+Tc,C); tcc++){
        for(too=to; too<min(to+Tm,M); too++){
#pragma HLS UNROLL
         for(tii=ti; tii<min(ti+Tn,N); tii++){
#pragma HLS UNROLL
          L:  output_fm[too][trr][tcc] +=
                weights[too][tii][i][j]*
                input_fm[tii][S*trr+i][S*tcc+j];
    } } } } } }
} } } }
```

论文的主要思想：将输入/输出特征图和权重分块。

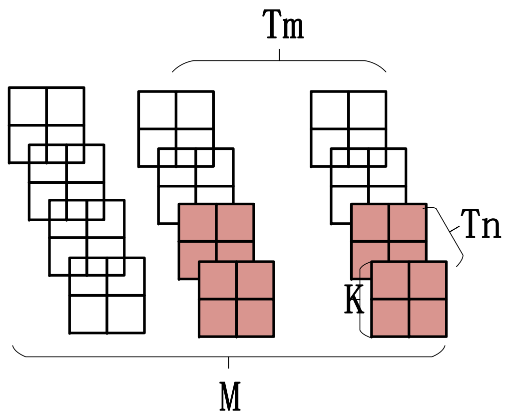
为了计算**标红**的输出特征图（右），所需的**输入特征**（中）和**权重**（左）均以彩色标出。

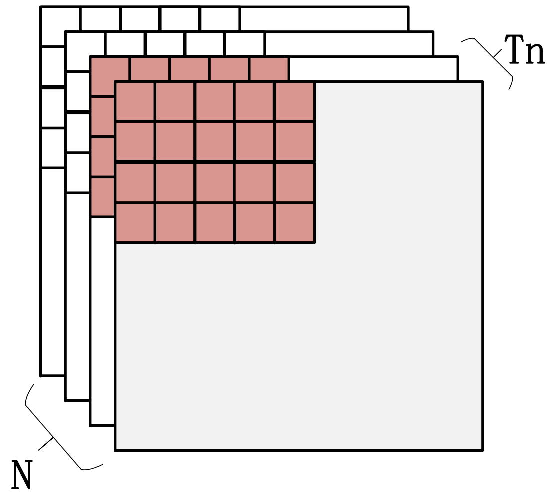
最简单的思路：将这些彩色标出的输入全部存入片上缓存，那么输入特征，权重，均可在片上复用。但是N很大，所以要对通道分块。

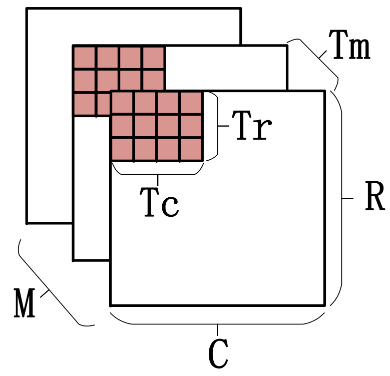
达到效果：输入特征图，权重，输出特征图均可**重用**

卷积核之间**并行**，卷积核内部**并行**

# 优化方式




External data transfer
To be discussed in Section 3.2

```
for(row=0; row<R; row+=Tr) {
 for(col=0; col<C; col+=Tc) {
  for(to=0; to<M; to+=Tm) {
   for(ti=0; ti<N; ti+=Tn) {
   //load output feature maps
   //load weights
   //load input feature maps
```

On-chip data computation
To be discussed in Section 3.1

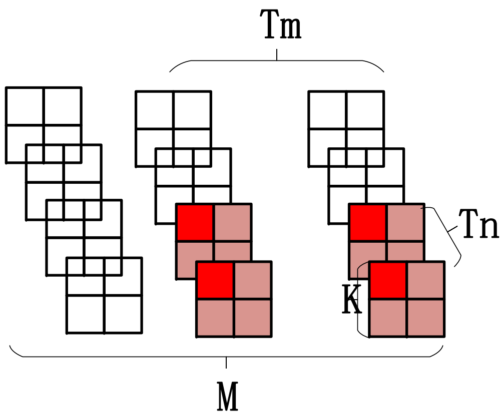
```
//on-chip data computation
for(i=0; i<K; i++) {
 for(j=0; j<K; j++) {
  for(trr=row; trr<min(row+Tr,R); trr++){
   for(tcc=col; tcc<min(col+Tc,C); tcc++){
    for(too=to; too<min(to+Tm,M); too++){
#pragma HLS UNROLL
     for(tii=ti; tii<min(ti+Tn,N); tii++){
#pragma HLS UNROLL
      L:  output_fm[too][trr][tcc] +=
            weights[too][tii][i][j]*
            input_fm[tii][S*trr+i][S*tcc+j];
} } } } } }
} } } }
```

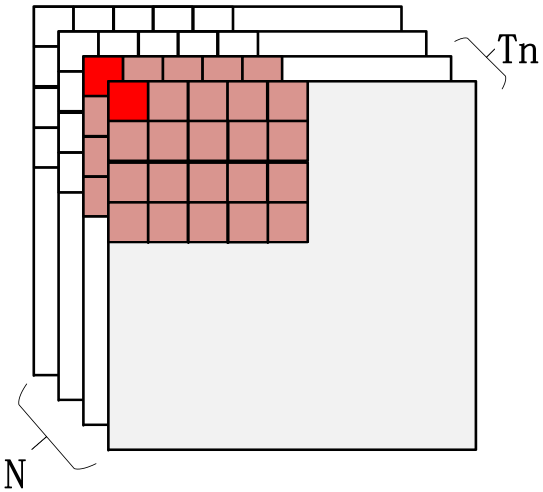
注：红色的代表**片上缓存**中的权重，特征，卷积结果
　　其他颜色的代表片外存储中的权重，特征，卷积结果

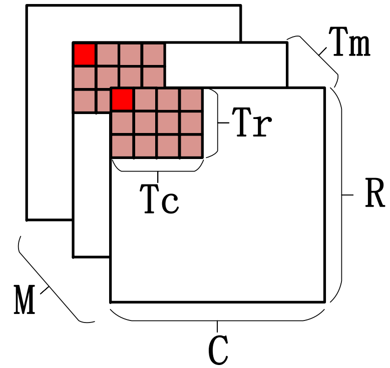
卷积运算分块对应着左侧伪代码的最外层4个循环

首先将Tm*Tn*K*K大小的权重和Tn*Tr’*Tc’大小的输入特征缓存到片上cache

# 优化方式




External data transfer
To be discussed in Section 3.2

```
for(row=0; row<R; row+=Tr) {
 for(col=0; col<C; col+=Tc) {
  for(to=0; to<M; to+=Tm) {
   for(ti=0; ti<N; ti+=Tn) {
   //load output feature maps
   //load weights
   //load input feature maps
```

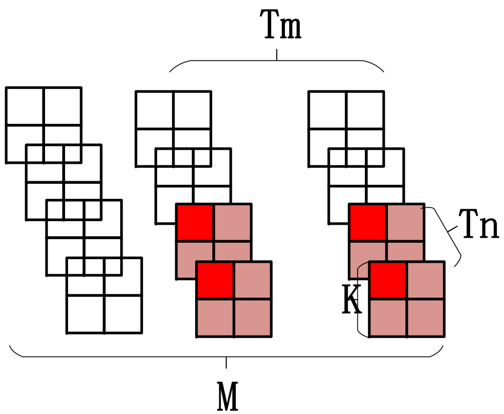
On-chip data computation
To be discussed in Section 3.1

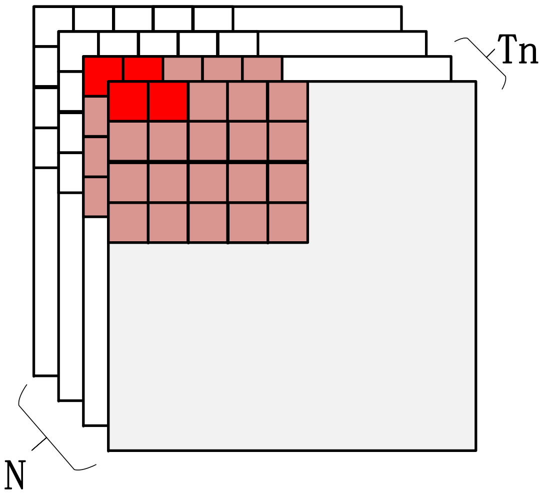
```
   //on-chip data computation
   for(i=0; i<K; i++) {
    for(j=0; j<K; j++) {
     for(trr=row; trr<min(row+Tr,R); trr++){
      for(tcc=col; tcc<min(col+Tc,C); tcc++){
       for(too=to; too<min(to+Tm,M); too++){
#pragma HLS UNROLL
        for(tii=ti; tii<min(ti+Tn,N); tii++){
#pragma HLS UNROLL
         L:  output_fm[too][trr][tcc] +=
               weights[too][tii][i][j]*
               input_fm[tii][S*trr+i][S*tcc+j];
   } } } } } }
} } } }
```

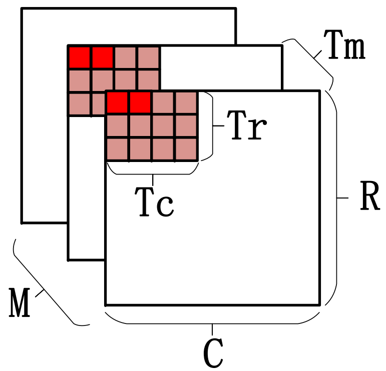
注：突出显示的代表进行卷积的权重，输入特征，卷积结果

接下来，Tm个卷积核同时进行卷积，在每个卷积核内部同时进行Tn个元素的乘累加

# 优化方式



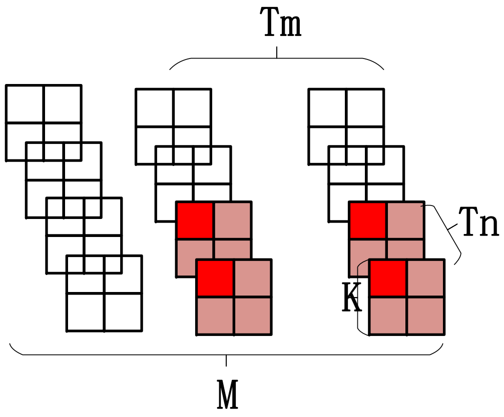



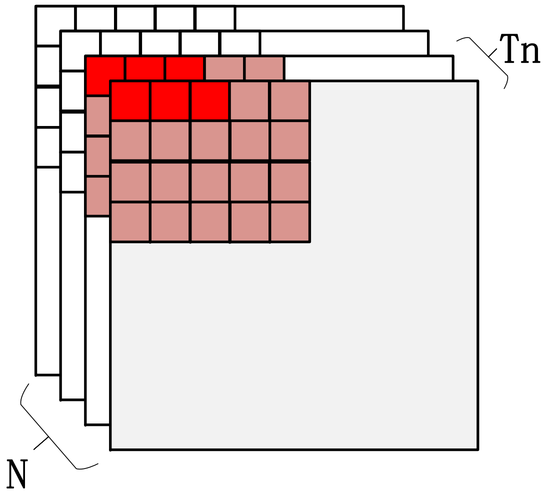

```
for(row=0; row<R; row+=Tr) {                External data transfer
 for(col=0; col<C; col+=Tc) {               To be discussed in Section 3.2
  for(to=0; to<M; to+=Tm) {
   for(ti=0; ti<N; ti+=Tn) {
   //load output feature maps
   //load weights
   //load input feature maps
                                            On-chip data computation
                                            To be discussed in Section 3.1
    //on-chip data computation
    for(i=0; i<K; i++) {
     for(j=0; j<K; j++) {
      for(trr=row; trr<min(row+Tr,R); trr++){
       for(tcc=col; tcc<min(col+Tc,C); tcc++){
        for(too=to; too<min(to+Tm,M); too++){
#pragma HLS UNROLL
         for(tii=ti; tii<min(ti+Tn,N); tii++){
#pragma HLS UNROLL
          L:  output_fm[too][trr][tcc] +=
                weights[too][tii][i][j]*
                input_fm[tii][S*trr+i][S*tcc+j];
    } } } } } }
} } } }
```

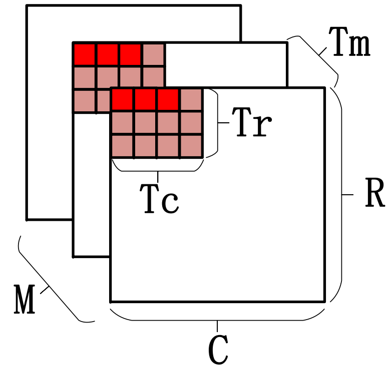
注：突出显示的代表进行卷积的权重，输入特征，卷积结果

接下来，Tm个卷积核同时进行卷积，在每个卷积核内部同时进行Tn个元素的乘累加

# 优化方式




External data transfer
To be discussed in Section 3.2

```
for(row=0; row<R; row+=Tr) {
 for(col=0; col<C; col+=Tc) {
  for(to=0; to<M; to+=Tm) {
   for(ti=0; ti<N; ti+=Tn) {
   //load output feature maps
   //load weights
   //load input feature maps
```

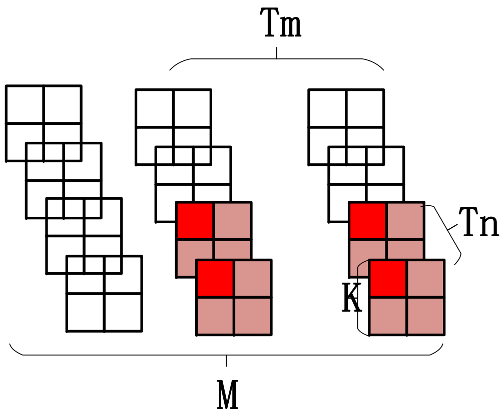
On-chip data computation
To be discussed in Section 3.1

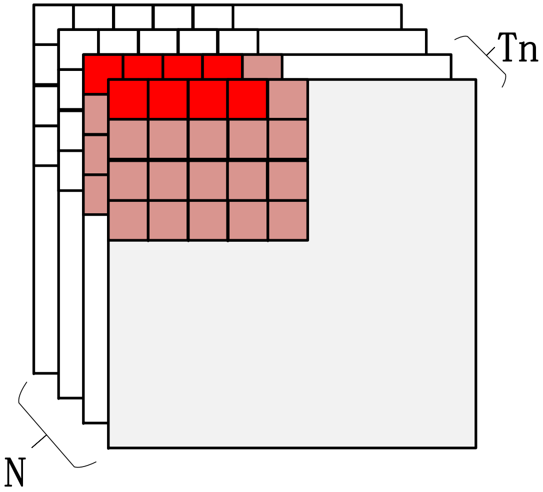
```
    //on-chip data computation
    for(i=0; i<K; i++) {
     for(j=0; j<K; j++) {
      for(trr=row; trr<min(row+Tr,R); trr++){
       for(tcc=col; tcc<min(col+Tc,C); tcc++){
        for(too=to; too<min(to+Tm,M); too++){
#pragma HLS UNROLL
         for(tii=ti; tii<min(ti+Tn,N); tii++){
#pragma HLS UNROLL
      L:  output_fm[too][trr][tcc] +=
            weights[too][tii][i][j]*
            input_fm[tii][S*trr+i][S*tcc+j];
    } } } } } }
} } } }
```

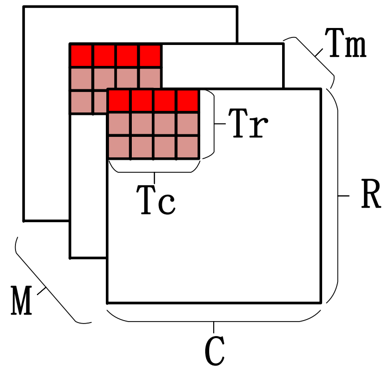
注：突出显示的代表进行卷积的权重，输入特征，卷积结果

接下来，Tm个卷积核同时进行卷积，在每个卷积核内部同时进行Tn个元素的乘累加

# 优化方式




External data transfer
To be discussed in Section 3.2

```
for(row=0; row<R; row+=Tr) {
 for(col=0; col<C; col+=Tc) {
  for(to=0; to<M; to+=Tm) {
   for(ti=0; ti<N; ti+=Tn) {
   //load output feature maps
   //load weights
   //load input feature maps
```

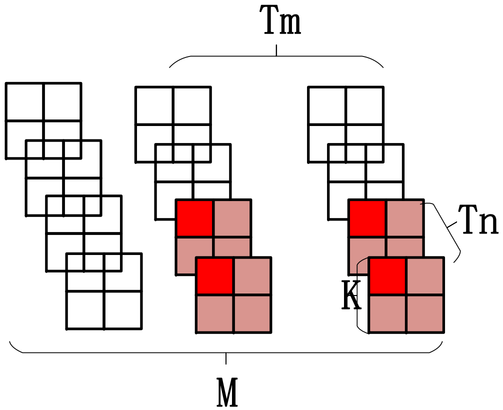
On-chip data computation
To be discussed in Section 3.1

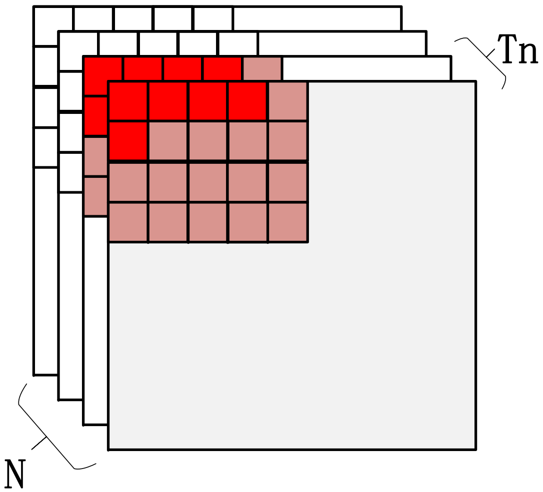
```
    //on-chip data computation
    for(i=0; i<K; i++) {
     for(j=0; j<K; j++) {
      for(trr=row; trr<min(row+Tr,R); trr++){
       for(tcc=col; tcc<min(col+Tc,C); tcc++){
        for(too=to; too<min(to+Tm,M); too++){
#pragma HLS UNROLL
         for(tii=ti; tii<min(ti+Tn,N); tii++){
#pragma HLS UNROLL
          L: output_fm[too][trr][tcc] +=
               weights[too][tii][i][j]*
               input_fm[tii][S*trr+i][S*tcc+j];
    } } } } } }
} } } }
```

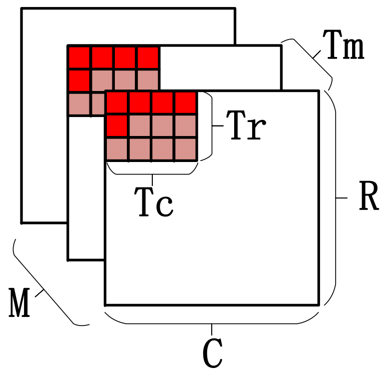
注：突出显示的代表进行卷积的权重，输入特征，卷积结果

接下来，Tm个卷积核同时进行卷积，在每个卷积核内部同时进行Tn个元素的乘累加

# 优化方式




External data transfer
To be discussed in Section 3.2

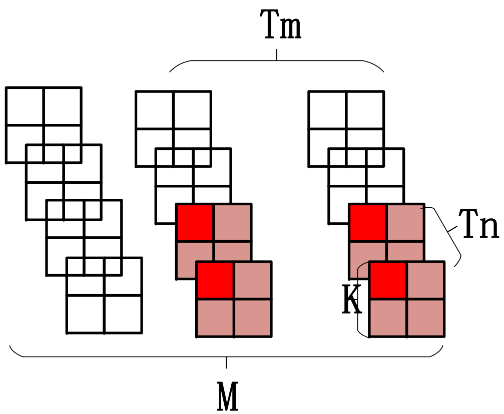
On-chip data computation
To be discussed in Section 3.1

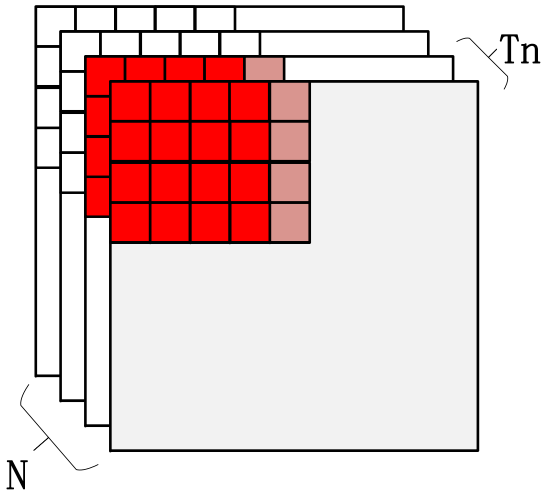
```
for(row=0; row<R; row+=Tr) {
 for(col=0; col<C; col+=Tc) {
  for(to=0; to<M; to+=Tm) {
   for(ti=0; ti<N; ti+=Tn) {
   //load output feature maps
   //load weights
   //load input feature maps

    //on-chip data computation
    for(i=0; i<K; i++) {
     for(j=0; j<K; j++) {
      for(trr=row; trr<min(row+Tr,R); trr++){
       for(tcc=col; tcc<min(col+Tc,C); tcc++){
        for(too=to; too<min(to+Tm,M); too++){
#pragma HLS UNROLL
         for(tii=ti; tii<min(ti+Tn,N); tii++){
#pragma HLS UNROLL
          L:  output_fm[too][trr][tcc] +=
                 weights[too][tii][i][j]*
                 input_fm[tii][S*trr+i][S*tcc+j];
    } } } } } }
} } } }
```

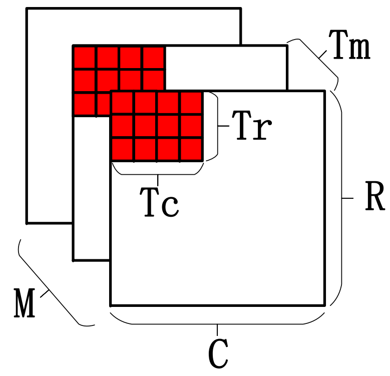
注：突出显示的代表进行卷积的权重，输入特征，卷积结果

接下来，Tm个卷积核同时进行卷积，在每个卷积核内部同时进行Tn个元素的乘累加

# 优化方式




External data transfer
To be discussed in Section 3.2

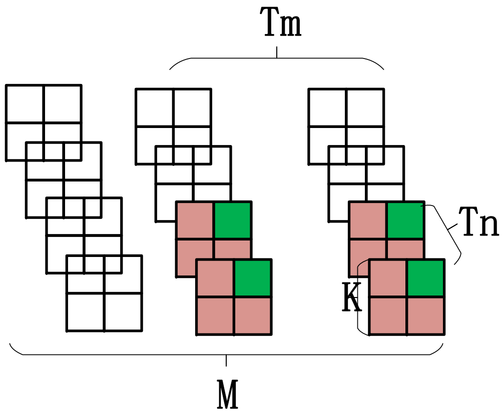
On-chip data computation
To be discussed in Section 3.1

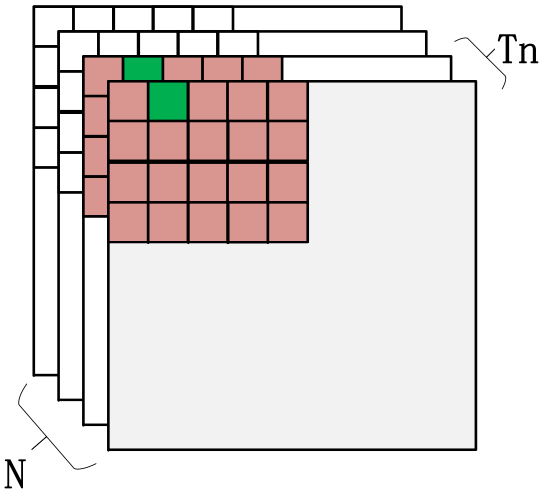
```
for(row=0; row<R; row+=Tr) {
 for(col=0; col<C; col+=Tc) {
  for(to=0; to<M; to+=Tm) {
   for(ti=0; ti<N; ti+=Tn) {
   //load output feature maps
   //load weights
   //load input feature maps

    //on-chip data computation
    for(i=0; i<K; i++) {
     for(j=0; j<K; j++) {
      for(trr=row; trr<min(row+Tr,R); trr++){
       for(tcc=col; tcc<min(col+Tc,C); tcc++){
        for(too=to; too<min(to+Tm,M); too++){
    #pragma HLS UNROLL
         for(tii=ti; tii<min(ti+Tn,N); tii++){
    #pragma HLS UNROLL
        L:  output_fm[too][trr][tcc] +=
              weights[too][tii][i][j]*
              input_fm[tii][S*trr+i][S*tcc+j];
    } } } } } }
} } } }
```

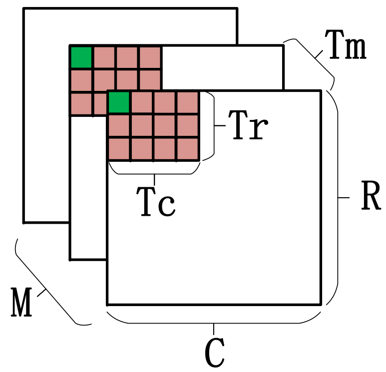
注：突出显示的代表进行卷积的权重，输入特征，卷积结果

接下来，Tm个卷积核同时进行卷积，在每个卷积核内部同时进行Tn个元素的乘累加

# 优化方式




External data transfer
To be discussed in Section 3.2

```
for(row=0; row<R; row+=Tr) {
 for(col=0; col<C; col+=Tc) {
  for(to=0; to<M; to+=Tm) {
   for(ti=0; ti<N; ti+=Tn) {
   //load output feature maps
   //load weights
   //load input feature maps
```

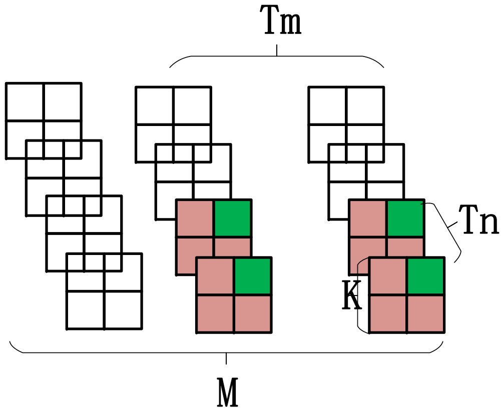
On-chip data computation
To be discussed in Section 3.1

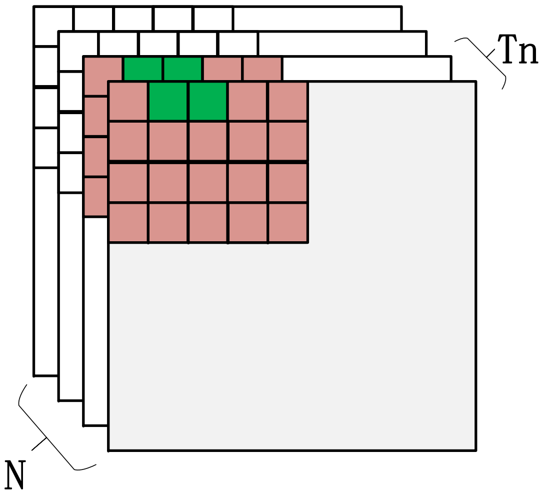
```
   //on-chip data computation
   for(i=0; i<K; i++) {
    for(j=0; j<K; j++) {
     for(trr=row; trr<min(row+Tr,R); trr++){
      for(tcc=col; tcc<min(col+Tc,C); tcc++){
       for(too=to; too<min(to+Tm,M); too++){
#pragma HLS UNROLL
        for(tii=ti; tii<min(ti+Tn,N); tii++){
#pragma HLS UNROLL
         L: output_fm[too][trr][tcc] +=
              weights[too][tii][i][j]*
              input_fm[tii][S*trr+i][S*tcc+j];
   } } } } } }
} } } }
```

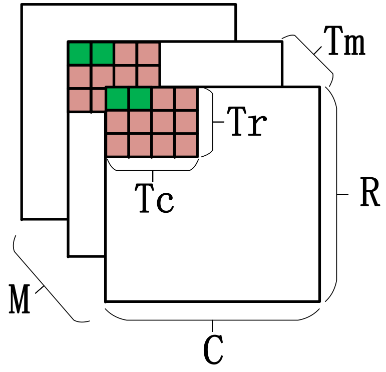
注：突出显示的代表进行卷积的权重，输入特征，卷积结果

接下来，Tm个卷积核同时进行卷积，在每个卷积核内部同时进行Tn个元素的乘累加

# 优化方式




External data transfer
To be discussed in Section 3.2

```
for(row=0; row<R; row+=Tr) {
 for(col=0; col<C; col+=Tc) {
  for(to=0; to<M; to+=Tm) {
   for(ti=0; ti<N; ti+=Tn) {
   //load output feature maps
   //load weights
   //load input feature maps
```

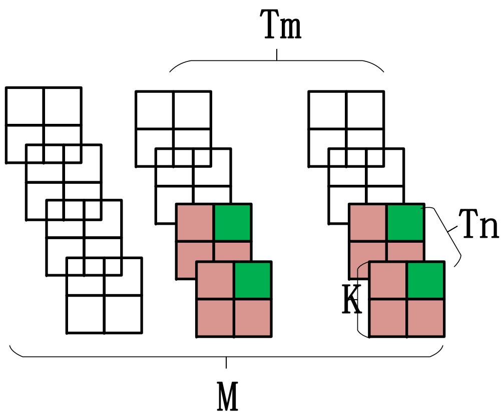
On-chip data computation
To be discussed in Section 3.1

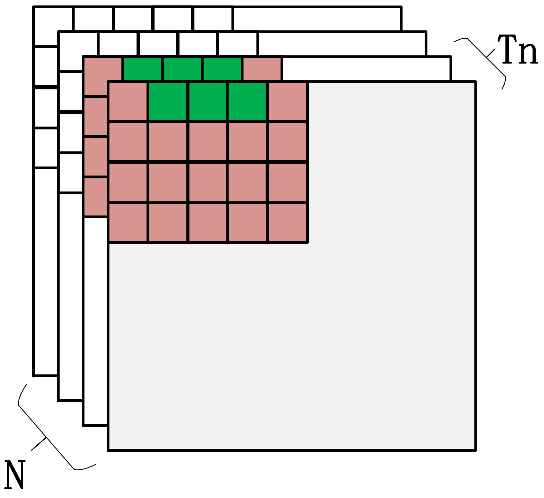
```
   //on-chip data computation
   for(i=0; i<K; i++) {
    for(j=0; j<K; j++) {
     for(trr=row; trr<min(row+Tr,R); trr++){
      for(tcc=col; tcc<min(col+Tc,C); tcc++){
       for(too=to; too<min(to+Tm,M); too++){
   #pragma HLS UNROLL
        for(tii=ti; tii<min(ti+Tn,N); tii++){
   #pragma HLS UNROLL
     L:  output_fm[too][trr][tcc] +=
           weights[too][tii][i][j]*
           input_fm[tii][S*trr+i][S*tcc+j];
   } } } } } }
} } } }
```

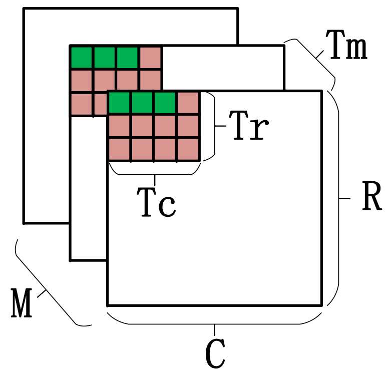
注：突出显示的代表进行卷积的权重，输入特征，卷积结果

接下来，Tm个卷积核同时进行卷积，在每个卷积核内部同时进行Tn个元素的乘累加

# 优化方式




External data transfer
To be discussed in Section 3.2

```
for(row=0; row<R; row+=Tr) {
 for(col=0; col<C; col+=Tc) {
  for(to=0; to<M; to+=Tm) {
   for(ti=0; ti<N; ti+=Tn) {
   //load output feature maps
   //load weights
   //load input feature maps
```

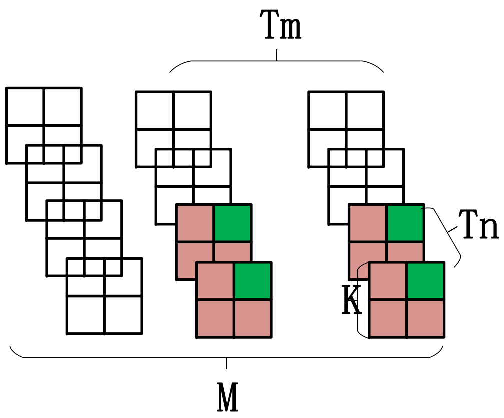
On-chip data computation
To be discussed in Section 3.1

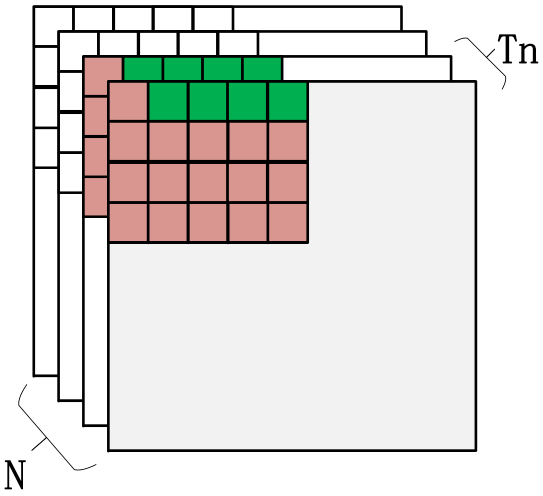
```
//on-chip data computation
for(i=0; i<K; i++) {
 for(j=0; j<K; j++) {
  for(trr=row; trr<min(row+Tr,R); trr++){
   for(tcc=col; tcc<min(col+Tc,C); tcc++){
    for(too=to; too<min(to+Tm,M); too++){
#pragma HLS UNROLL
     for(tii=ti; tii<min(ti+Tn,N); tii++){
#pragma HLS UNROLL
      L:  output_fm[too][trr][tcc] +=
            weights[too][tii][i][j]*
            input_fm[tii][S*trr+i][S*tcc+j];
} } } } } }
} } } }
```

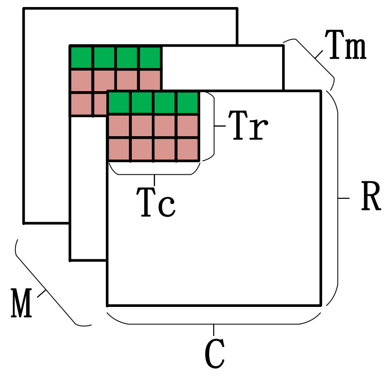
注：突出显示的代表进行卷积的权重，输入特征，卷积结果

接下来，Tm个卷积核同时进行卷积，在每个卷积核内部同时进行Tn个元素的乘累加

# 优化方式




```
for(row=0; row<R; row+=Tr) {
 for(col=0; col<C; col+=Tc) {
  for(to=0; to<M; to+=Tm) {
   for(ti=0; ti<N; ti+=Tn) {
   //load output feature maps
   //load weights
   //load input feature maps

   //on-chip data computation
   for(i=0; i<K; i++) {
    for(j=0; j<K; j++) {
     for(trr=row; trr<min(row+Tr,R); trr++){
      for(tcc=col; tcc<min(col+Tc,C); tcc++){
       for(too=to; too<min(to+Tm,M); too++){
#pragma HLS UNROLL
        for(tii=ti; tii<min(ti+Tn,N); tii++){
#pragma HLS UNROLL
         L: output_fm[too][trr][tcc] +=
              weights[too][tii][i][j]*
              input_fm[tii][S*trr+i][S*tcc+j];
   } } } } } }
} } } }
```

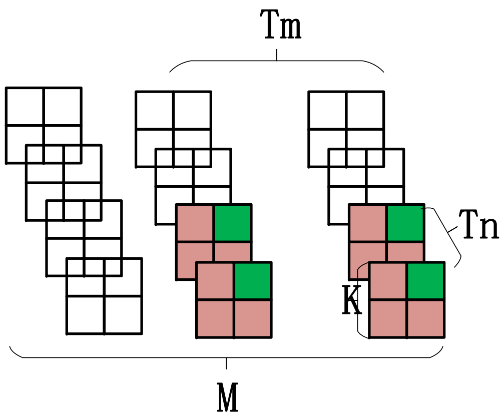
External data transfer
To be discussed in Section 3.2

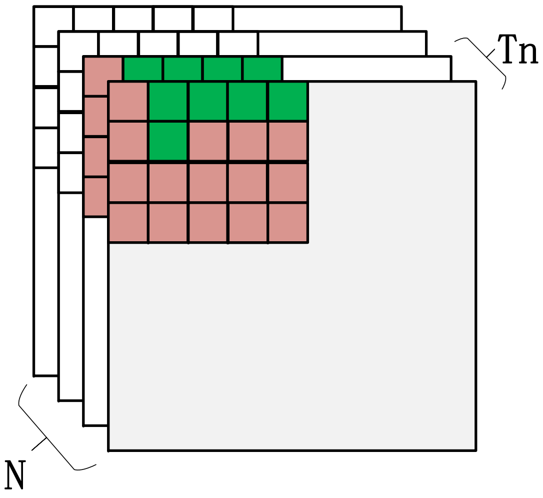
On-chip data computation
To be discussed in Section 3.1

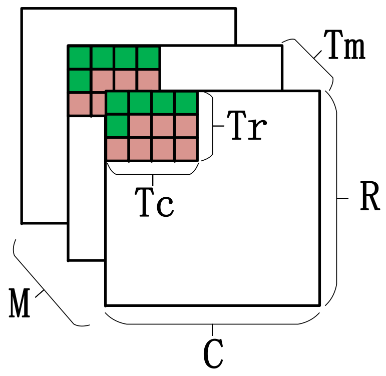
注：突出显示的代表进行卷积的权重，输入特征，卷积结果

接下来，Tm个卷积核同时进行卷积，在每个卷积核内部同时进行Tn个元素的乘累加

# 优化方式




External data transfer
To be discussed in Section 3.2

```
for(row=0; row<R; row+=Tr) {
 for(col=0; col<C; col+=Tc) {
  for(to=0; to<M; to+=Tm) {
   for(ti=0; ti<N; ti+=Tn) {
   //load output feature maps
   //load weights
   //load input feature maps
```

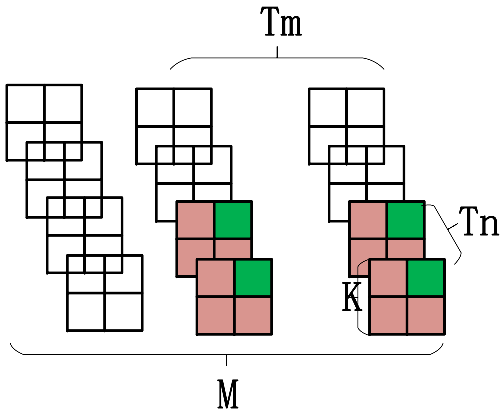
On-chip data computation
To be discussed in Section 3.1

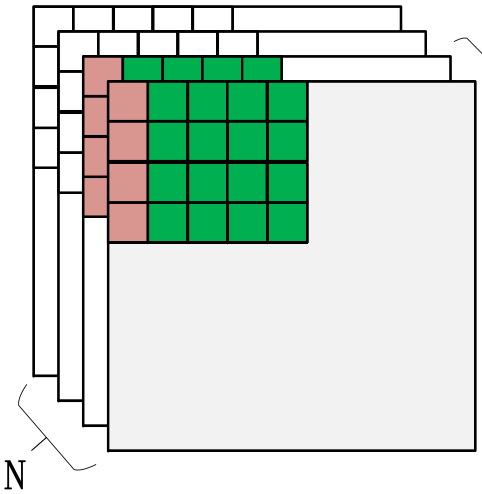
```
    //on-chip data computation
    for(i=0; i<K; i++) {
     for(j=0; j<K; j++) {
      for(trr=row; trr<min(row+Tr,R); trr++){
       for(tcc=col; tcc<min(col+Tc,C); tcc++){
        for(too=to; too<min(to+Tm,M); too++){
#pragma HLS UNROLL
         for(tii=ti; tii<min(ti+Tn,N); tii++){
#pragma HLS UNROLL
       L:  output_fm[too][trr][tcc] +=
             weights[too][tii][i][j]*
             input_fm[tii][S*trr+i][S*tcc+j];
    } } } } } }
} } } }
```

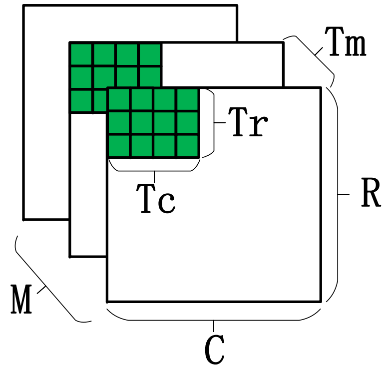
注：突出显示的代表进行卷积的权重，输入特征，卷积结果

接下来，Tm个卷积核同时进行卷积，在每个卷积核内部同时进行Tn个元素的乘累加

# 优化方式




External data transfer
To be discussed in Section 3.2

```
for(row=0; row<R; row+=Tr) {
 for(col=0; col<C; col+=Tc) {
  for(to=0; to<M; to+=Tm) {
   for(ti=0; ti<N; ti+=Tn) {
   //load output feature maps
   //load weights
   //load input feature maps
```

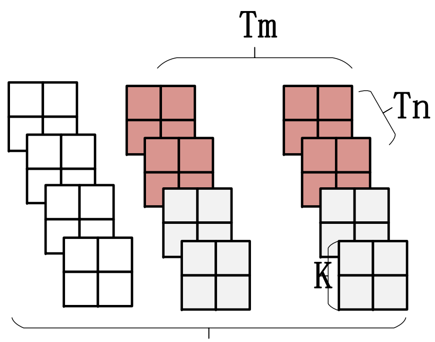
On-chip data computation
To be discussed in Section 3.1

```
//on-chip data computation
for(i=0; i<K; i++) {
 for(j=0; j<K; j++) {
  for(trr=row; trr<min(row+Tr,R); trr++){
   for(tcc=col; tcc<min(col+Tc,C); tcc++){
    for(too=to; too<min(to+Tm,M); too++){
#pragma HLS UNROLL
     for(tii=ti; tii<min(ti+Tn,N); tii++){
#pragma HLS UNROLL
     L:  output_fm[too][trr][tcc] +=
          weights[too][tii][i][j]*
          input_fm[tii][S*trr+i][S*tcc+j];
} } } } } }
} } } }
```

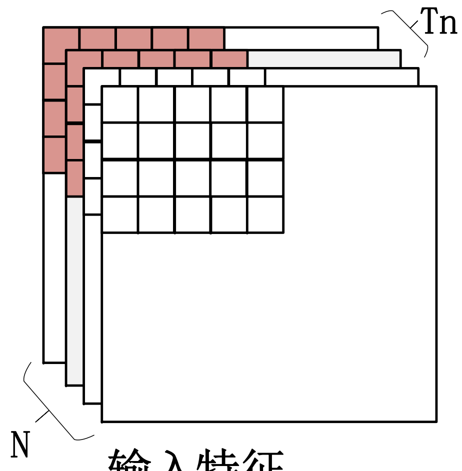
注：突出显示的代表进行卷积的权重，输入特征，卷积结果

接下来，Tm个卷积核同时进行卷积，在每个卷积核内部同时进行Tn个元素的乘累加

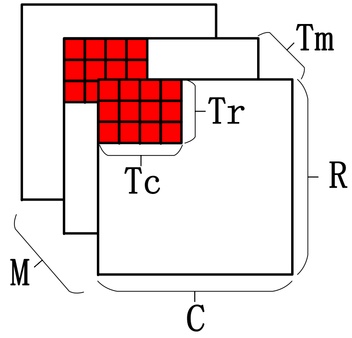
# 优化方式




卷积核




输入特征




输出特征

External data transfer
To be discussed in Section 3.2

```
for(row=0; row<R; row+=Tr) {
 for(col=0; col<C; col+=Tc) {
  for(to=0; to<M; to+=Tm) {
   for(ti=0; ti<N; ti+=Tn) {
   //load output feature maps
   //load weights
   //load input feature maps
```

On-chip data computation
To be discussed in Section 3.1

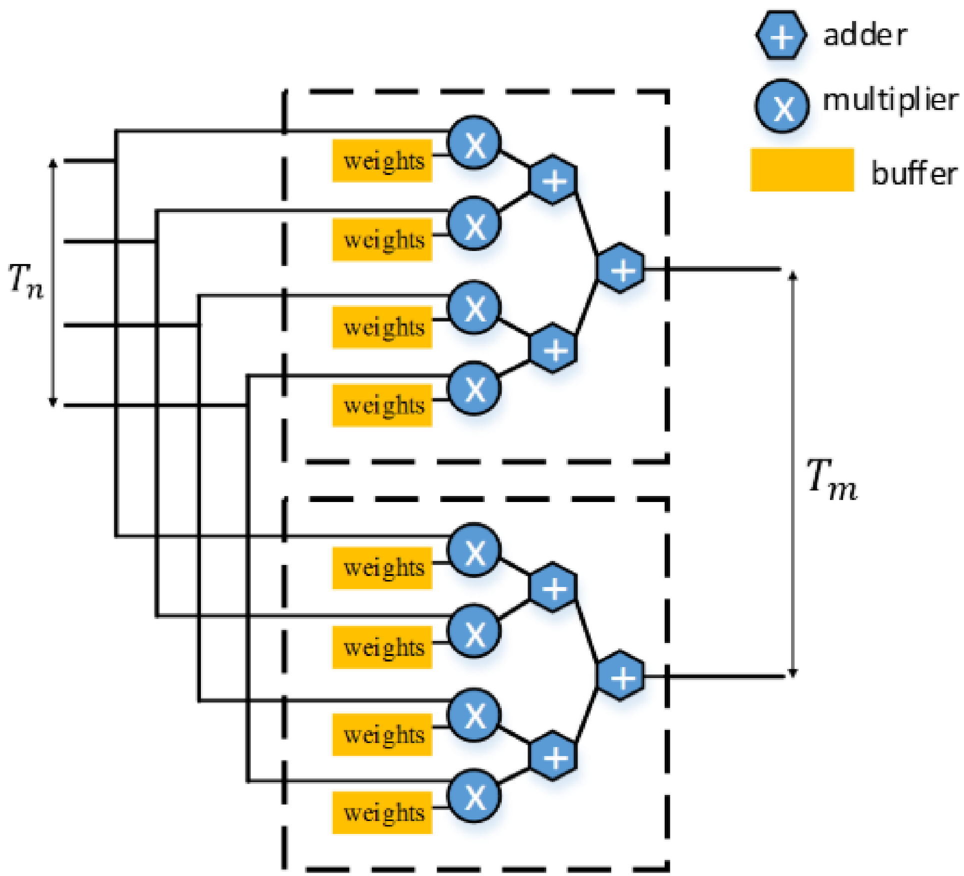
```
//on-chip data computation
for(i=0; i<K; i++) {
 for(j=0; j<K; j++) {
  for(trr=row; trr<min(row+Tr,R); trr++){
   for(tcc=col; tcc<min(col+Tc,C); tcc++){
    for(too=to; too<min(to+Tm,M); too++){
#pragma HLS UNROLL
     for(tii=ti; tii<min(ti+Tn,N); tii++){
#pragma HLS UNROLL
    L:  output_fm[too][trr][tcc] +=
          weights[too][tii][i][j]*
          input_fm[tii][S*trr+i][S*tcc+j];
} } } } } }
} } } }
```

注：突出显示的代表进行卷积的权重，输入特征，卷积结果

接下来，对输入通道方向上后续的分块进行卷积

# 优化方式

```
//on-chip data computation
for(i=0; i<K; i++) {
 for(j=0; j<K; j++) {
  for(trr=row; trr<min(row+Tr,R); trr++){
   for(tcc=col; tcc<min(col+Tc,C); tcc++){
    for(too=to; too<min(to+Tm,M); too++){
#pragma HLS UNROLL
     for(tii=ti; tii<min(ti+Tn,N); tii++){
#pragma HLS UNROLL
      L:   output_fm[too][trr][tcc] +=
              weights[too][tii][i][j]*
              input_fm[tii][S*trr+i][S*tcc+j];
} } } } } }
```




- 综合得出的硬件结构

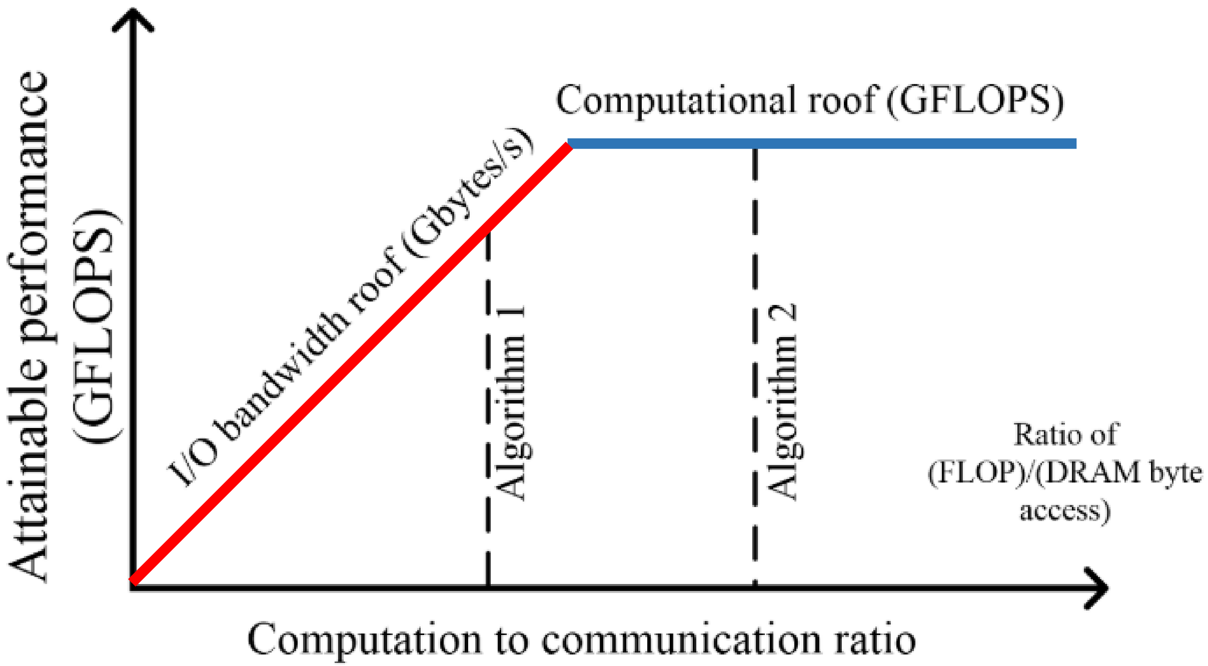
# 借助Roofline模型寻找最优参数

- 另一个问题产生了，如此多的分块参数：
  - Tr， Tc， Tm， Tn，如何选择这些参数，能够**优化数据重用，优化并行度** → 使用Roofline模型

- Roofline模型的简单理解：
  - 计算上限：硬件资源（DSP，逻辑资源）
  - 带宽上限：硬件通信带宽

例如：

1. DDR到FPGA带宽：1秒钟传送100个数字
2. 算法1中数据重用做的不好，从DDR读取100个数能进行180次运算，这时系统的计算能力即为180/s
3. 算法2中数据重用做的好，从DDR读取100个数能进行300次运算，但是系统的计算上限是200/s，这时系统的计算能力即为200/s



# 借助Roofline模型寻找最优参数

- 计算横坐标
  - 根据Tr， Tc， Tm， Tn，这些参数，计算**运算量和通信量之比**

$$
\begin{aligned}
& \textit{Computation to Communication Ratio} \\
= & \frac{\textit{total number of operations}}{\textit{total amount of external data access}} \\
= & \frac{2 \times R \times C \times M \times N \times K \times K}{\alpha_{in} \times B_{in} + \alpha_{wght} \times B_{wght} + \alpha_{out} \times B_{out}} \quad (4)
\end{aligned}
$$

where

$$B_{in} = T_n(ST_r + K - S)(ST_c + K - S) \quad (5)$$

$$B_{wght} = T_m T_n K^2 \quad (6)$$

$$B_{out} = T_m T_r T_c \quad (7)$$

$$0 < B_{in} + B_{wght} + B_{out} \leq BRAM_{capacity} \quad (8)$$

$$\alpha_{in} = \alpha_{wght} = \frac{M}{T_m} \times \frac{N}{T_n} \times \frac{R}{T_r} \times \frac{C}{T_c} \quad (9)$$

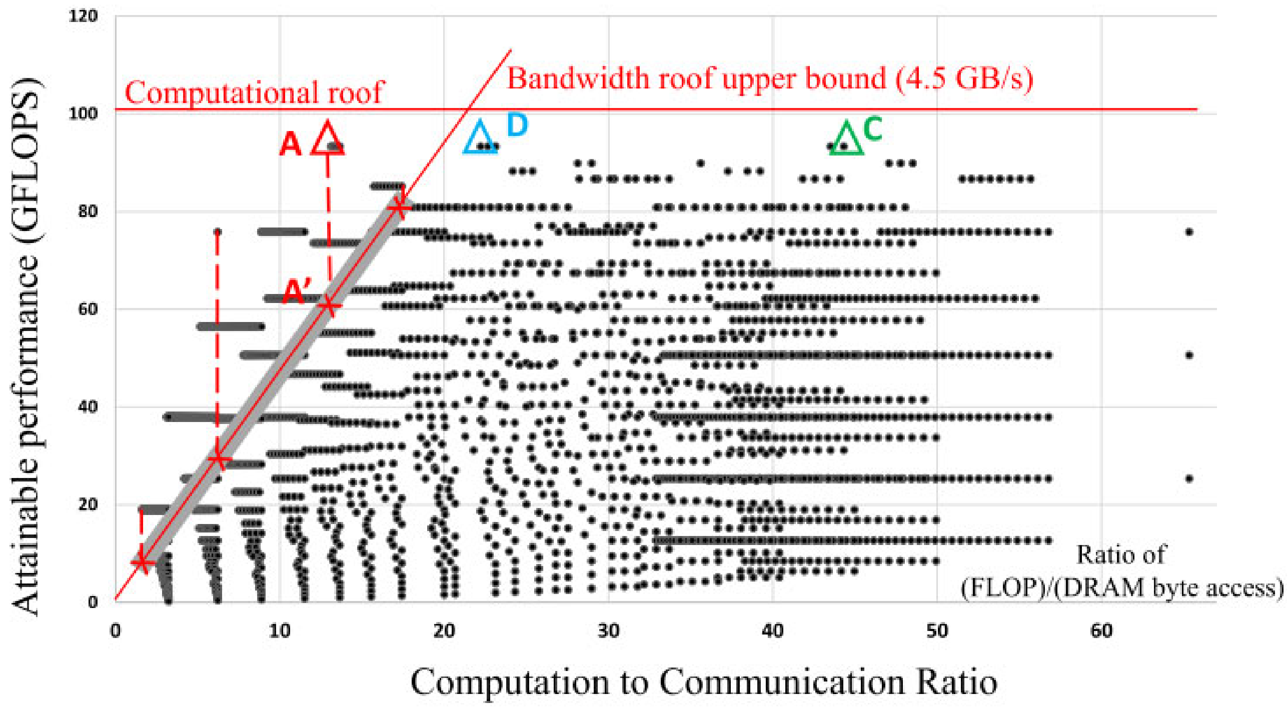
Without *output_fm*'s data reuse,

$$\alpha_{out} = 2 \times \frac{M}{T_m} \times \frac{N}{T_n} \times \frac{R}{T_r} \times \frac{C}{T_c} \quad (10)$$

With *output_fm*'s data reuse,

$$\alpha_{out} = \frac{M}{T_m} \times \frac{R}{T_r} \times \frac{C}{T_c} \quad (11)$$

Given a specific loop schedule and a set of tile size tuple $\langle Tm, Tn, Tr, Tc \rangle$, computation to communication ratio can be calculated with above formula.




(b) Design space of platform-supported designs

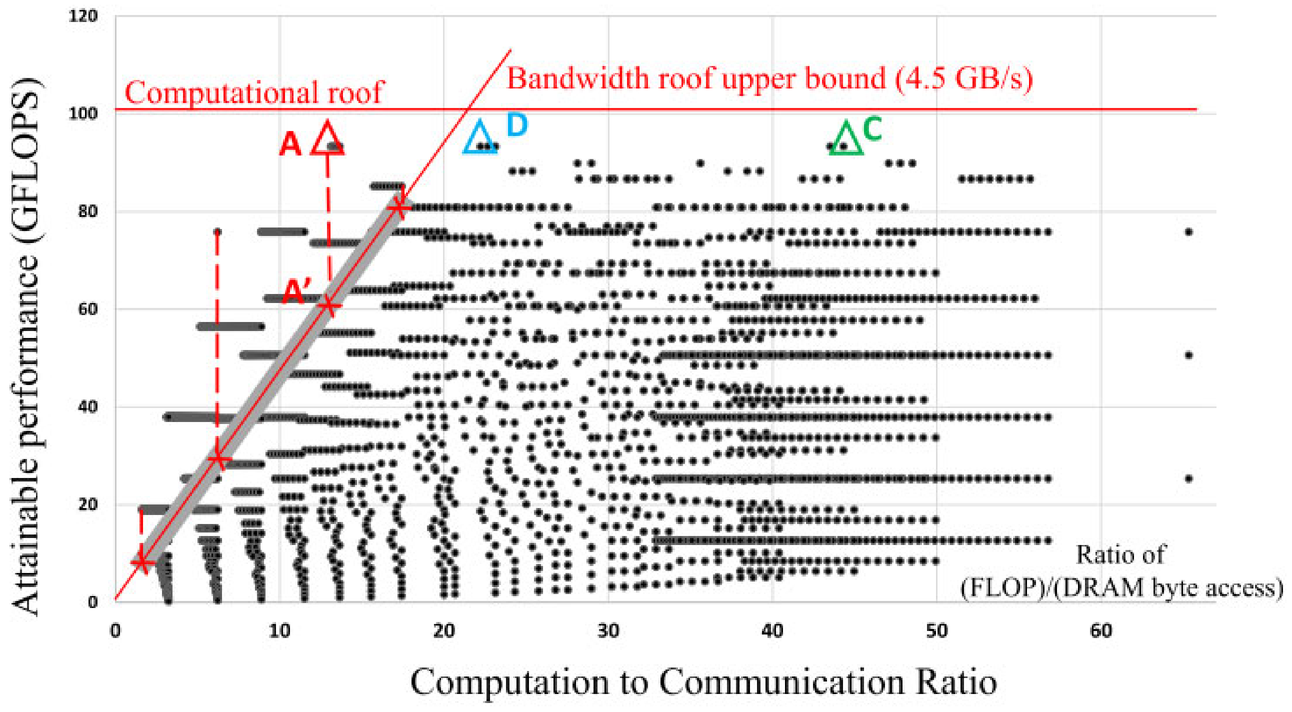
# 借助Roofline模型寻找最优参数

- 计算纵坐标
  - 根据Tr， Tc， Tm， Tn，这些参数，计算运算吞吐量

$$
\begin{aligned}
& \textit{computational roof} \\
= & \frac{\textit{total number of operations}}{\textit{number of execution cycles}} \\
= & \frac{2 \times R \times C \times M \times N \times K \times K}{\lceil \frac{M}{T_m} \rceil \times \lceil \frac{N}{T_n} \rceil \times \frac{R}{T_r} \times \frac{C}{T_c} \times (T_r \times T_c \times K \times K + P)} \\
\approx & \frac{2 \times R \times C \times M \times N \times K \times K}{\lceil \frac{M}{T_m} \rceil \times \lceil \frac{N}{T_n} \rceil \times R \times C \times K \times K}
\end{aligned} \tag{3}
$$

where $P = \textit{pipeline depth} - 1$.

- A点：受限于传输带宽
- D点：受限于计算资源
- C点：受限于计算资源




(b) Design space of platform-supported designs

# 实验结果

- 对AlexNet加速效果：
  - 对比CPU加速：17.4x
  - 对比CPU能耗：89.4x
- 存在的问题：
  - 只对卷积层进行加速
  - 没有进行低bit量化
- 较新进展：
  - 模型稀疏化
  - 模型低位宽量化
  - 可以计算出访存下界

**Table 7: Performance comparison to CPU**

| float | CPU 2.20GHz (ms) | | FPGA | |
|---|---|---|---|---|
| 32 bit | 1thd -O3 | 16thd -O3 | (ms) | GFLOPS |
| layer 1 | 98.18 | 19.36 | 7.67 | 27.50 |
| layer 2 | 94.66 | 27.00 | 5.35 | 83.79 |
| layer 3 | 77.38 | 24.30 | 3.79 | 78.81 |
| layer 4 | 65.58 | 18.64 | 2.88 | 77.94 |
| layer 5 | 40.70 | 14.18 | 1.93 | 77.61 |
| **Total** | 376.50 | 103.48 | 21.61 | - |
| **Overall GFLOPS** | **3.54** | **12.87** | **61.62** | |
| **Speedup** | **1.00x** | **3.64x** | **17.42x** | |

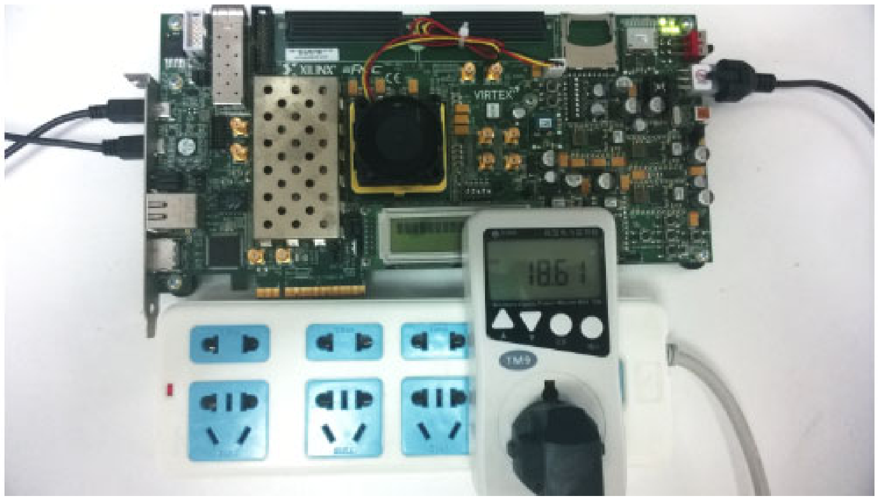


**Table 8: Power consumption and energy**

| | Intel Xeon 2.20GHz | | FPGA |
|---|---|---|---|
| | 1 thread -O3 | 16 threads -O3 | |
| Power (Watt) | 95.00 | 95.00 | 18.61 |
| Comparison | 5.1x | 5.1x | 1x |
| Energy (J) | 35.77 | 9.83 | 0.40 |
| Comparison | 89.4x | 24.6x | 1x |

谢谢大家