Group Weekly Talk

TENET: A Framework for Modeling Tensor Dataflow Based on Relation-centric Notation

Yun (Eric) Liang

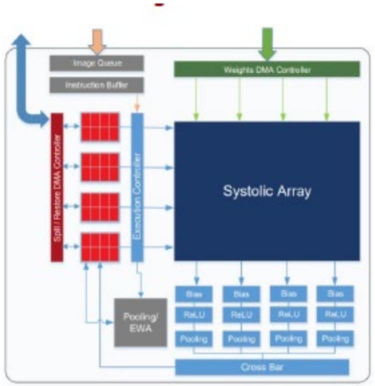
- Room 518
- Science Building #5
- Peking University
- +86 10 6276 0779
- ericlyun-at-pku.edu.cn
- Google Scholar

Liqiang Lu (B.S from PKU, Ph.D, started 2017)

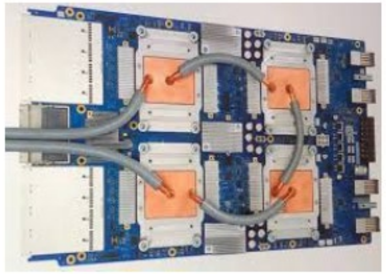
Liancheng Jia (B.S from PKU, Ph.D, started 2018)

辛杰 2021-5-26

Many Accelerators…



Xilinx xDNN



Google TPU

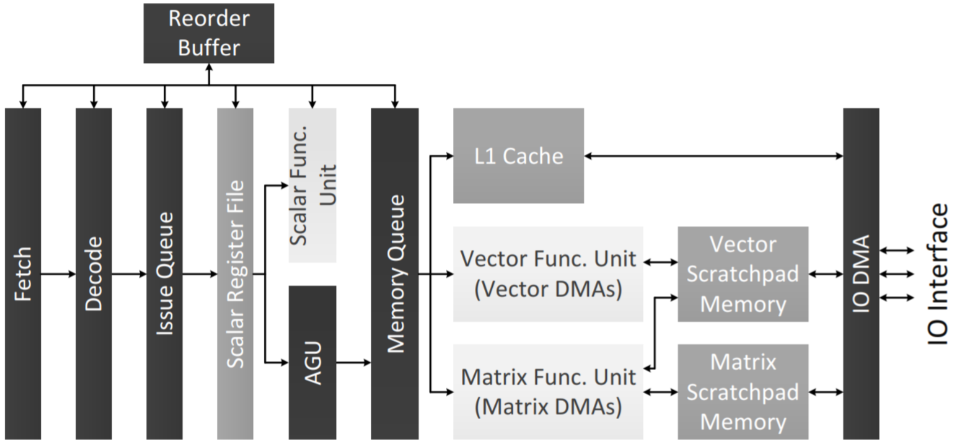
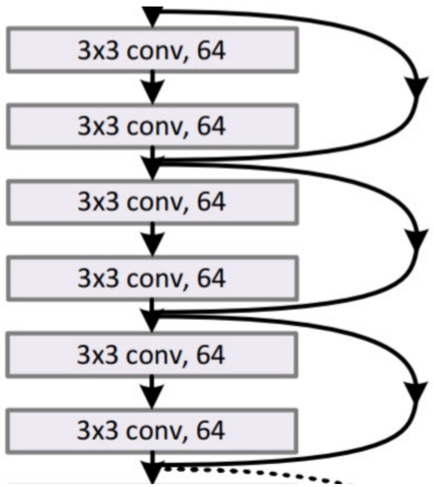


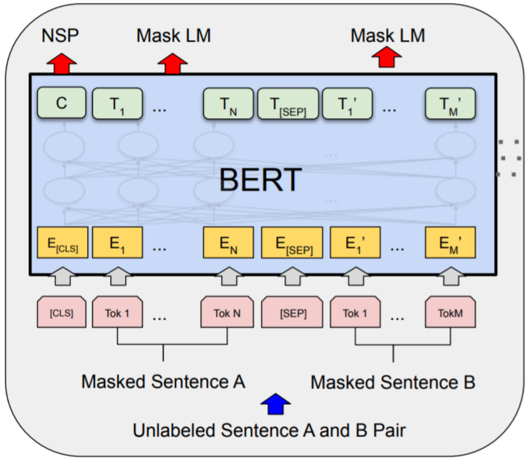
Figure 8. A prototype accelerator based on Cambricon.

Problem

Many Algorithms/Models...



ResNet



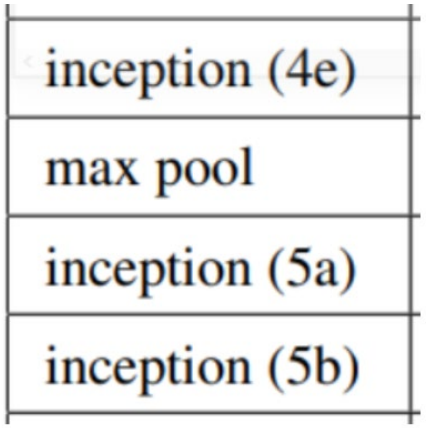
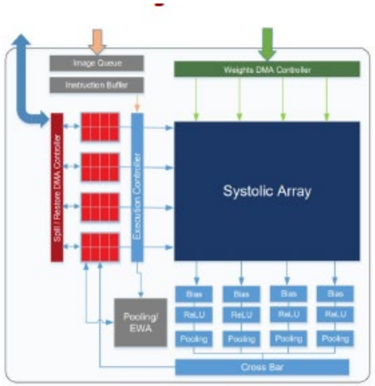
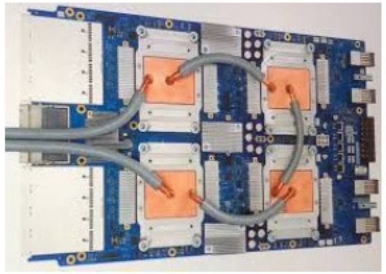


Figure 3: Partition of GoogLeNet

Which One is best?



Xilinx xDNN



Google TPU

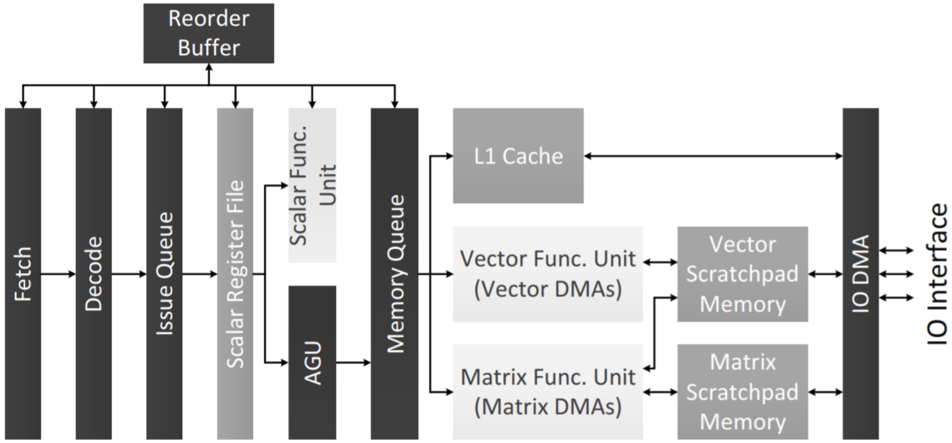
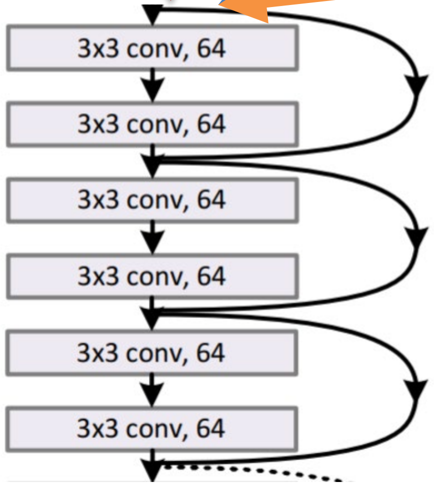
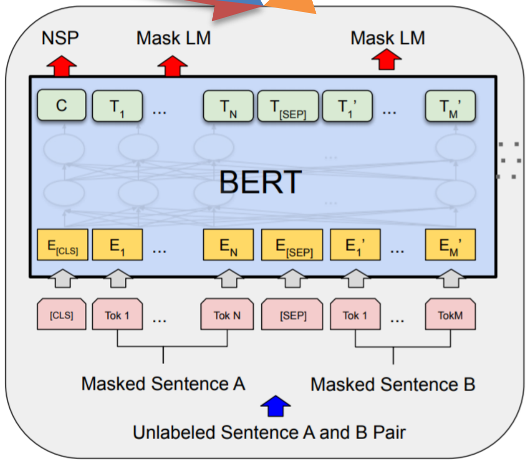


Figure 8. A prototype accelerator based on Cambricon.



ResNet



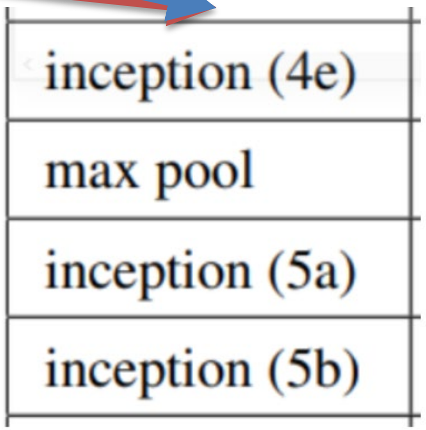
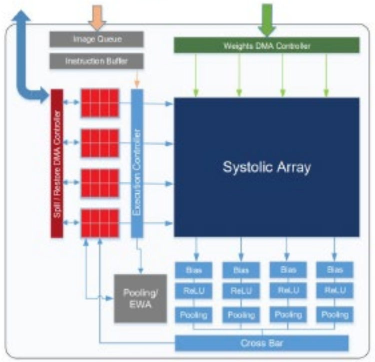


Figure 3: Partition of GoogLeNet

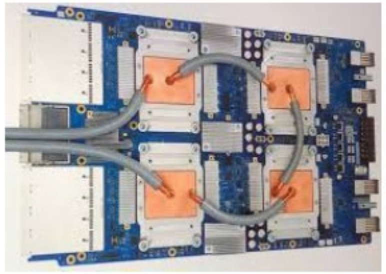
So, We need a performance model

1. 寻找适合这个应用的加速器
2. 指导加速器设计
3. …

So, We need a performance model



Xilinx xDNN



Google TPU

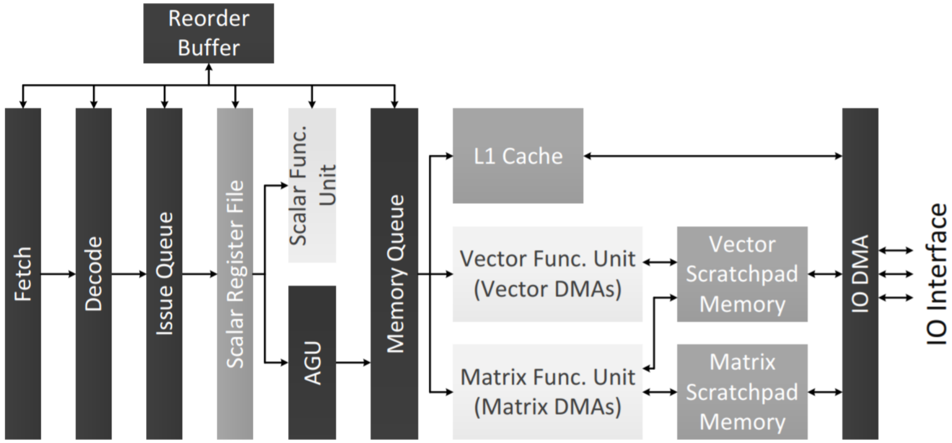
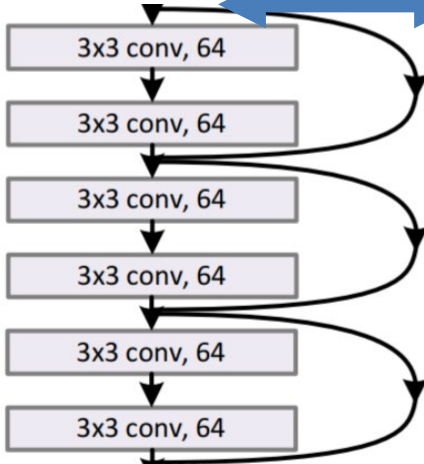
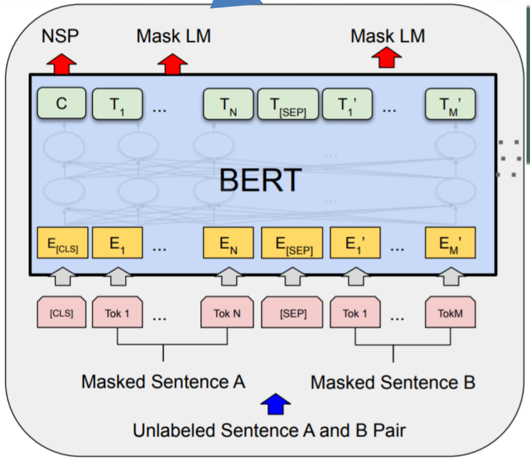


Figure 8. A prototype accelerator based on Cambricon.

指定一个架构，
指定一个算法
输出他的一些性能指标



ResNet



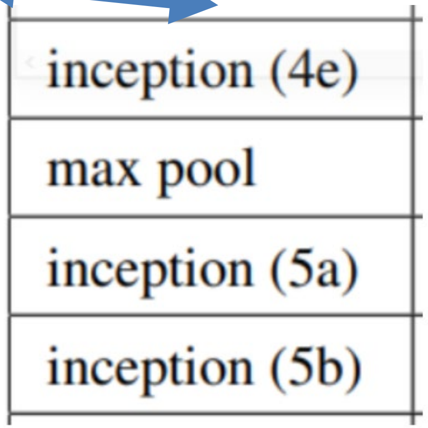
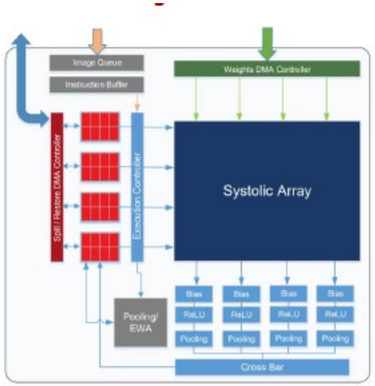


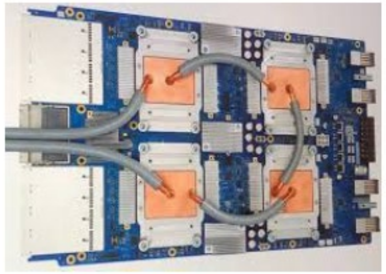
Figure 3: Partition of GoogLeNet

Insight

Simplify the hardware



Xilinx xDNN



Google TPU

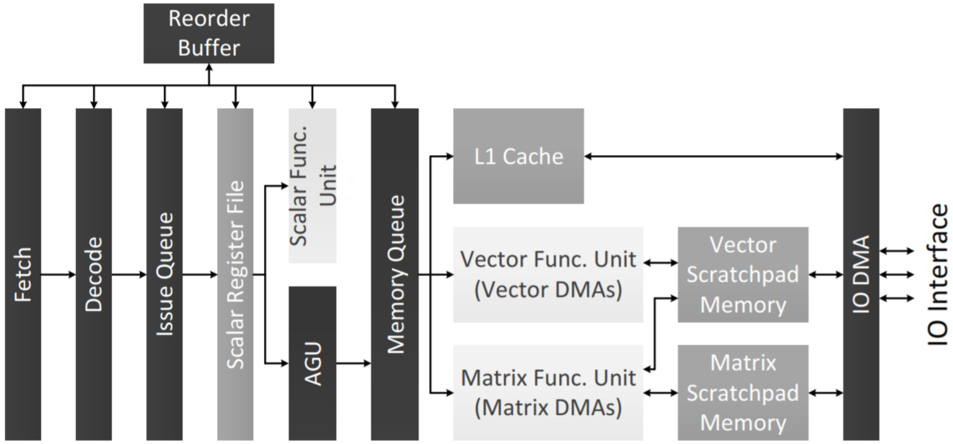
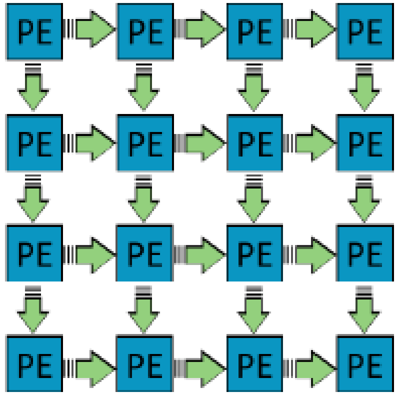


Figure 8. A prototype accelerator based on Cambricon.

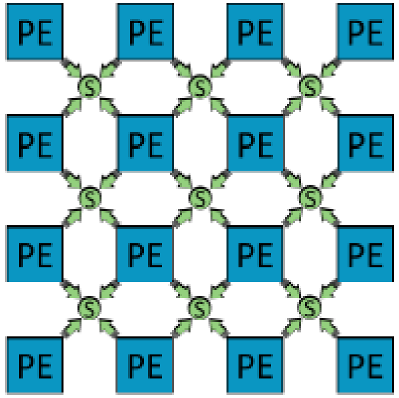
Spatial Architectural！！

Spatial Architectural

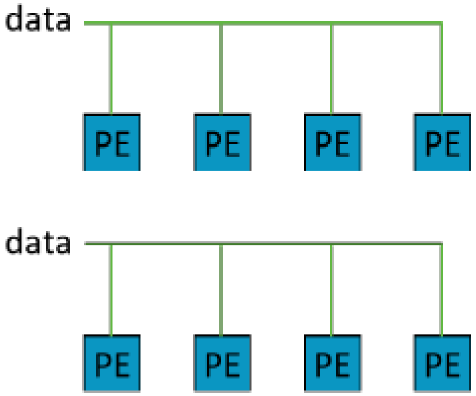
Contain a PE array



(a) 2D systolic



(b) Mesh NoC

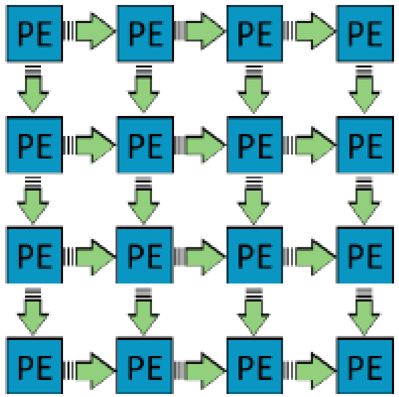


(c) Multicast

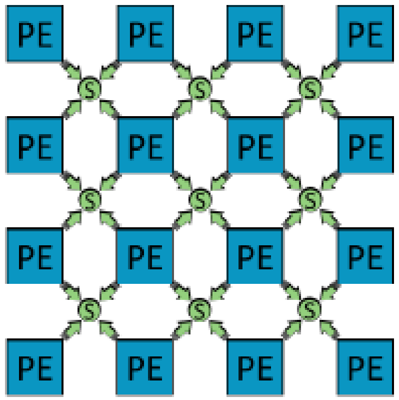
Spatial Architectural

Contain a PE array

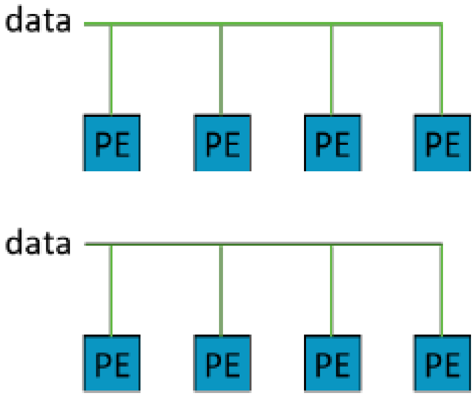
Connect via on-chip **interconnect**



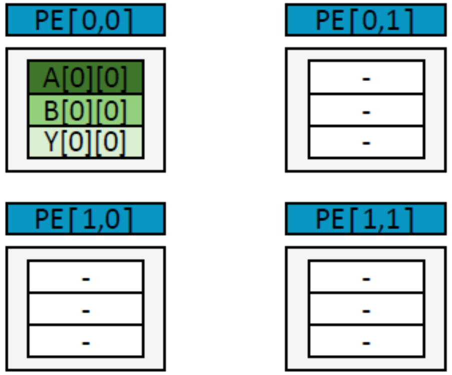
(a) 2D systolic



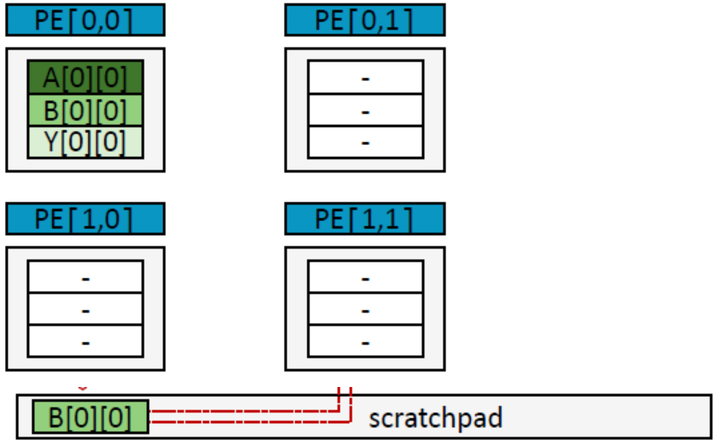
(b) Mesh NoC



(c) Multicast



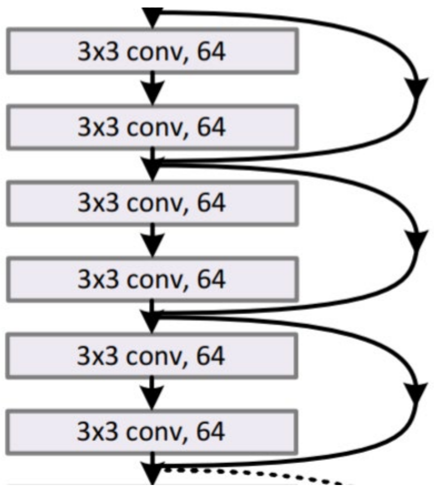
Each PE has their own Register file



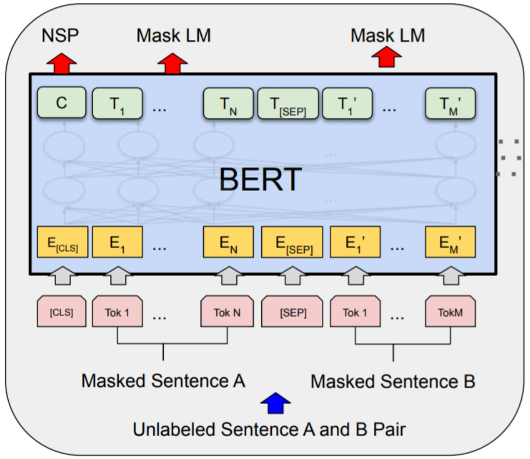
Share a on-chip scratchpad memory
And off-chip Global memory

So, It has a three level of
Memory Hierarchy

Simplify the Algorithm/model



ResNet



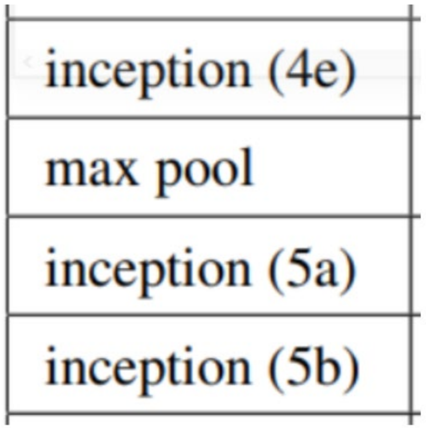


Figure 3: Partition of GoogLeNet

Tensor Operation！！

Tensor Operation

```
// Conv2D
for (int ioc = 0; ioc < OC; ioc++) {
    for (int ioh = 0; ioh < OH; ioh++) {
        for (int iow = 0; iow < OW; iow++) {
            Out[ioc][ioh][iow] = 0.f;
            for (int iic = 0; iic < IC; iic++) {
                for (int ikh = 0; ikh < KH; ikh++) {
                    for (int ikw = 0; ikw < KW; ikw++) {
                        Out[ioc][ioh][iow] +=
                          K[ioc][iic][ikh][ikw] *
                         In[iic][f(ioh)+ikh][f(iow)+ikw];
                    }
                }
            }
        }
    }
}
```

```
// BatchNorm
for (int ioc = 0; ioc < OC; ioc++) {
    for (int ioh = 0; ioh < OH; ioh++) {
        for (int iow = 0; iow < OW; iow++) {
            Out[ioc][ioh][iow] = (Out[ioc][ioh][iow] - mean) / var;
        }
    }
}
for (int ioc = 0; ioc < OC; ioc++) {
    for (int ioh = 0; ioh < OH; ioh++) {
        for (int iow = 0; iow < OW; iow++) {
            Out[ioc][ioh][iow] = Out[ioc][ioh][iow] * gamma + bias;
        }
    }
}
```

```
// GeMM
for (int m = 0; m < M; m++) {
    for (int n = 0; n < N; n++) {
        C[m][n] = beta / alpha;
        for (int k = 0; k < K; k++) {
            C[m][n] += A[m][k] * B[k][n];
        }
        C[m][n] *= alpha;
    }
}
```

Tensor Operation are usually described using a loop nest

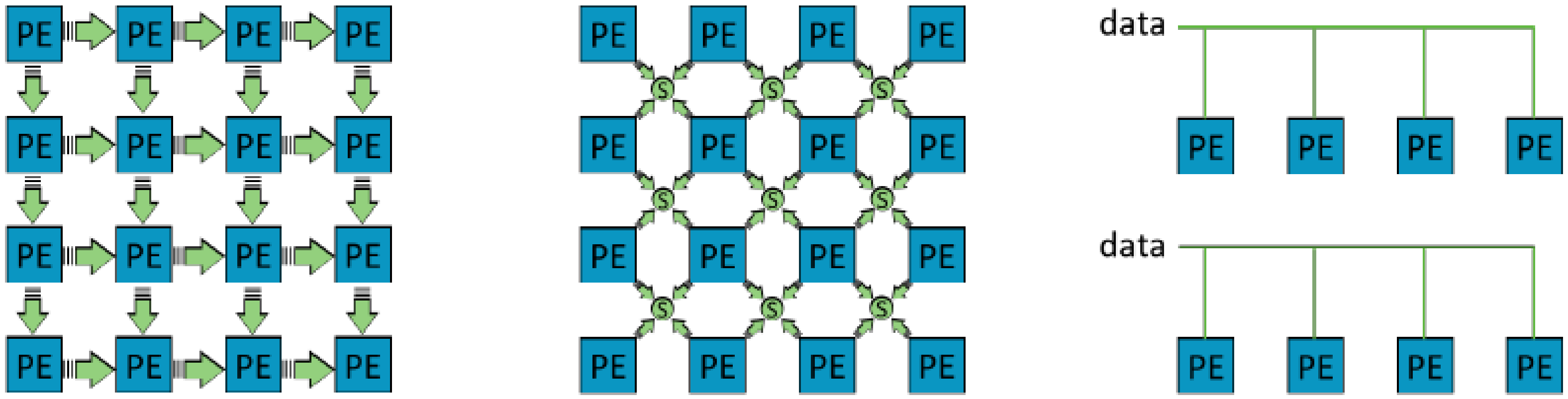
Challenge

```
// Conv2D
for (int ioc = 0; ioc < OC; ioc++) {
    for (int ioh = 0; ioh < OH; ioh++) {
        for (int iow = 0; iow < OW; iow++) {
            Out[ioc][ioh][iow] = 0.f;
            for (int iic = 0; iic < IC; iic++) {
                for (int ikh = 0; ikh < KH; ikh++) {
                    for (int ikw = 0; ikw < KW; ikw++) {
                        Out[ioc][ioh][iow] +=
                          K[ioc][iic][ikh][ikw] *
                         In[iic][f(ioh)+ikh][f(iow)+ikw];
                    }
                }
            }
        }
    }
}
```

```
// BatchNorm
for (int ioc = 0; ioc < OC; ioc++) {
    for (int ioh = 0; ioh < OH; ioh++) {
        for (int iow = 0; iow < OW; iow++) {
            Out[ioc][ioh][iow] = (Out[ioc][ioh][iow] - mean) / var;
        }
    }
}
for (int ioc = 0; ioc < OC; ioc++) {
    for (int ioh = 0; ioh < OH; ioh++) {
        for (int iow = 0; iow < OW; iow++) {
            Out[ioc][ioh][iow] = Out[ioc][ioh][iow] * gamma + bias;
        }
    }
}
```

```
// GeMM
for (int m = 0; m < M; m++) {
    for (int n = 0; n < N; n++) {
        C[m][n] = beta / alpha;
        for (int k = 0; k < K; k++) {
            C[m][n] += A[m][k] * B[k][n];
        }
        C[m][n] *= alpha;
    }
}
```

How to expression them?



(a) 2D systolic (b) Mesh NoC (c) Multicast

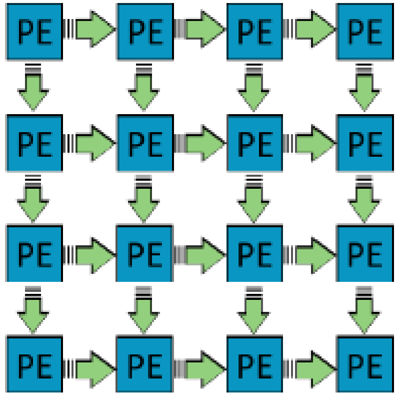
Method

```
// Conv2D
for (int ioc = 0; ioc < OC; ioc++) {
    for (int ioh = 0; ioh < OH; ioh++) {
        for (int iow = 0; iow < OW; iow++) {
            Out[ioc][ioh][iow] = 0.f;
            for (int iic = 0; iic < IC; iic++) {
                for (int ikh = 0; ikh < KH; ikh++) {
                    for (int ikw = 0; ikw < KW; ikw++) {
                        Out[ioc][ioh][iow] +=
                         K[ioc][iic][ikh][ikw] *
                        In[iic][f(ioh)+ikh][f(iow)+ikw];
                    }
                }
            }
        }
    }
}
```

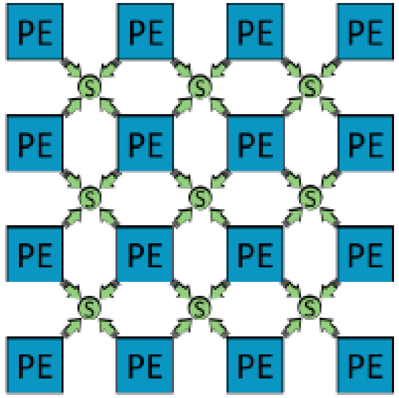
```
// BatchNorm
for (int ioc = 0; ioc < OC; ioc++) {
    for (int ioh = 0; ioh < OH; ioh++) {
        for (int iow = 0; iow < OW; iow++) {
            Out[ioc][ioh][iow] = (Out[ioc][ioh][iow] - mean) / var;
        }
    }
}
for (int ioc = 0; ioc < OC; ioc++) {
    for (int ioh = 0; ioh < OH; ioh++) {
        for (int iow = 0; iow < OW; iow++) {
            Out[ioc][ioh][iow] = Out[ioc][ioh][iow] * gamma + bias;
        }
    }
}
```

```
// GeMM
for (int m = 0; m < M; m++) {
    for (int n = 0; n < N; n++) {
        C[m][n] = beta / alpha;
        for (int k = 0; k < K; k++) {
            C[m][n] += A[m][k] * B[k][n];
        }
        C[m][n] *= alpha;
    }
}
```

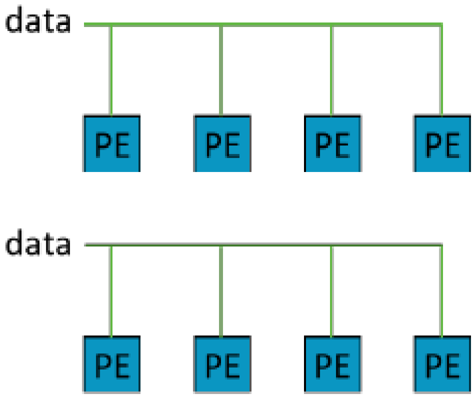
Relation-Centric Expression



(a) 2D systolic



(b) Mesh NoC

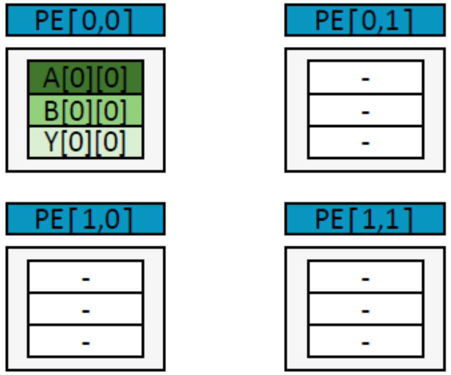


(c) Multicast

Example : input Tensor Operation

```
GEMM operation:

for  (i = 0; i < 2; i++)
  for  (j = 0; j < 2; j++)
    for  (k = 0; k < 4; k++)
          S: Y[i,j] +=
             A[i,k] * B[k,j];
```



Systolic Array

Example : Input dataflow relation

GEMM operation:

```
for  (i = 0; i < 2; i++)
  for  (j = 0; j < 2; j++)
    for  (k = 0; k < 4; k++)
          S: Y[i,j] +=
             A[i,k] * B[k,j];
```

Dataflow

space-stamp

{S[i,j,k] → PE[i,j]}   语句实例S[i, j, k]在PE[i, j]上执行

time-stamp

{S[i,j,k] → T[i+j+k]}   语句实例S[i, j, k]在时刻T[i + j + k]上执行

Example : this dataflow relation on systolic array

GEMM operation:

```
for  (i = 0; i < 2; i++)
  for  (j = 0; j < 2; j++)
    for  (k = 0; k < 4; k++)
          S: Y[i,j] +=
             A[i,k] * B[k,j];
```

Dataflow

space-stamp

{S[i,j,k] → PE[i,j]}　　语句实例S[i, j, k]在PE[i, j]上执行

time-stamp

{S[i,j,k] → T[i+j+k]}　　语句实例S[i, j, k]在时刻T[i + j + k]上执行

时刻T[0]的执行

T[i+j+k]=T[0]

⇓ Domain

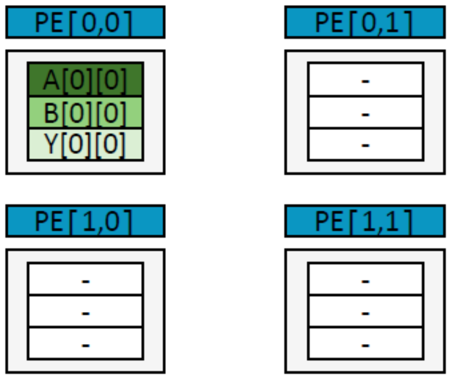
S[0,0,0]→PE[0,0]

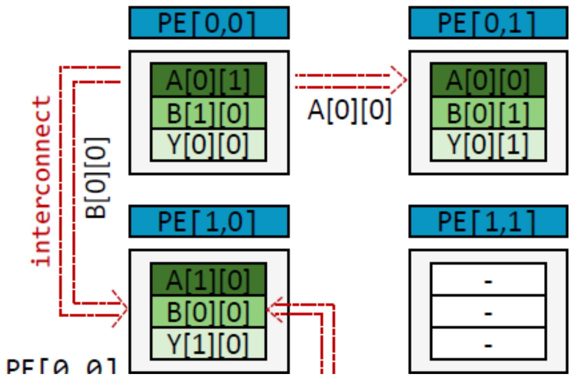
时刻T[1]的执行

T[i+j+k]=T[1]

⇓ Domain

S[0,0,1]→PE[0,0]
S[1,0,0]→PE[1,0]
S[0,1,0]→PE[0,1]





$$i+j+k=1$$

$$constraint: 0 \le i, j < 2, \quad 0 \le k < 4$$

$$loop\ instances: [i, j, k] = [0,0,1], [1,0,0], [0,1,0]$$

Example : data access

GEMM operation:

```
for  (i = 0; i < 2; i++)
  for  (j = 0; j < 2; j++)
    for  (k = 0; k < 4; k++)
          S: Y[i,j] +=
             A[i,k] * B[k,j];
```

Dataflow

space-stamp

{S[i,j,k] → PE[i,j]}

time-stamp

{S[i,j,k] → T[i+j+k]}

$$A_{D,F_Y} = \{(PE[i,j] \mid T[i+j+k]) \to Y[i,j]\}$$

PE[i, j]在时刻T[i + j + k]访问Y[i, j]

这个式子表达出了 一个PE，在不同的时刻，会访问相同的Y，因此，Y可以在一个PE内部进行复用

Example : Network

$I_{PE_1,PE_2} = \{PE[\vec{p}_1] \rightarrow PE[\vec{p}_2]\} \;\; : c_1, \cdots, c_k$ 表示 p1 和 p2 之间有连接

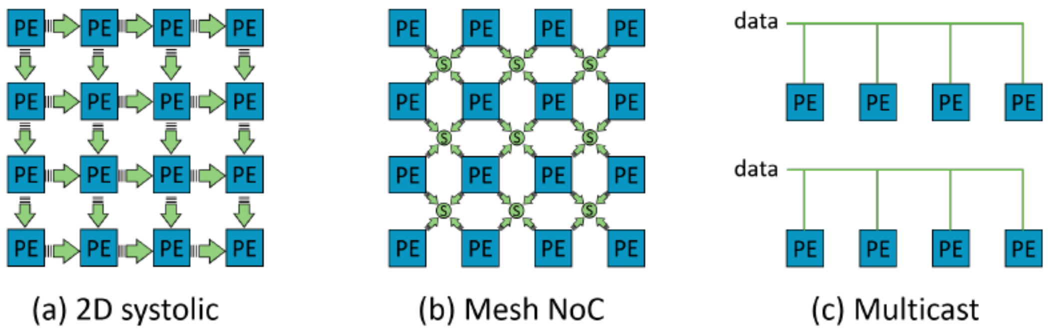


Fig. 4: Different interconnect topologies.

$$\begin{aligned}
\textbf{Interconnection}: &\quad \{\mathrm{PE[i,j]} \rightarrow \mathrm{PE[i',j']}\} \\
\textbf{2D-systolic}: &\quad (i' = i, j' = j+1) \; or \; (i' = i+1, j' = j) \\
\textbf{Mesh}: &\quad abs(i'-i) \leq 1 \; and \; abs(j'-j) \leq 1 \\
\textbf{1D-Multicast}: &\quad abs(i'-i) \leq 3 \quad (4 \; PEs)
\end{aligned}$$

Performance Model: Total Data Access

GEMM operation:

```
for  (i = 0; i < 2; i++)
  for  (j = 0; j < 2; j++)
    for  (k = 0; k < 4; k++)
          S: Y[i,j] +=
             A[i,k] * B[k,j];
```

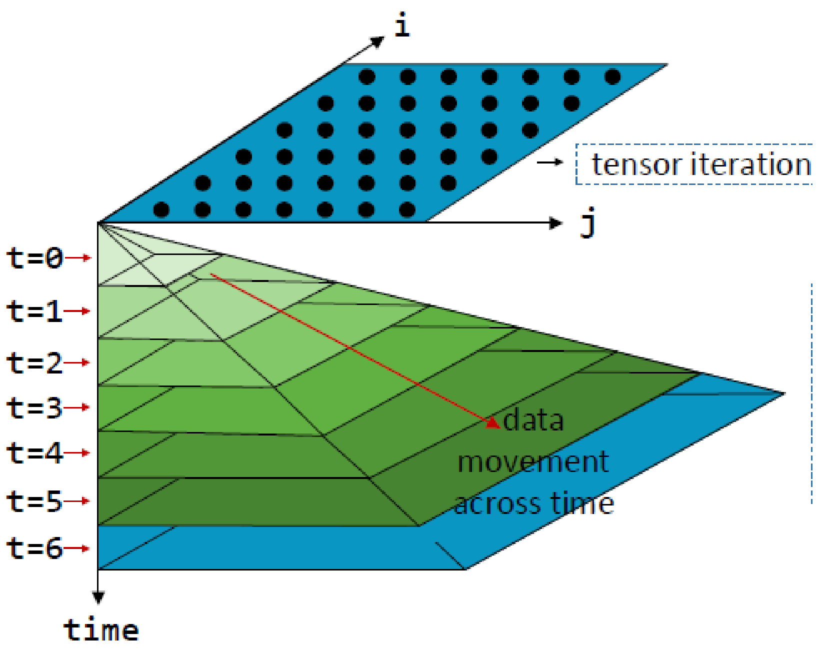
Dataflow

space-stamp

{S[i,j,k] → PE[i,j]}

time-stamp

{S[i,j,k] → T[i+j+k]}



time-stamp 0. used data $A[0][0]$

time-stamp 1. used data $A[0][1],\ A[0][0]\ A[1][0]$

time-stamp 2. used data $A[0][2],\ A[0][1],\ A[1][1],\ A[1][0]$

time-stamp 3. used data $A[0][3],\ A[0][2],\ A[1][2],\ A[1][1]$

$\textbf{TotalVolume} = 1 + 3 + 4 + 4 = 12$

Performance Model: Reused Data

GEMM operation:

```
for  (i = 0; i < 2; i++)
  for  (j = 0; j < 2; j++)
    for  (k = 0; k < 4; k++)
          S: Y[i,j] +=
             A[i,k] * B[k,j];
```

Dataflow

space-stamp

{S[i,j,k] → PE[i,j]}

time-stamp

{S[i,j,k] → T[i+j+k]}

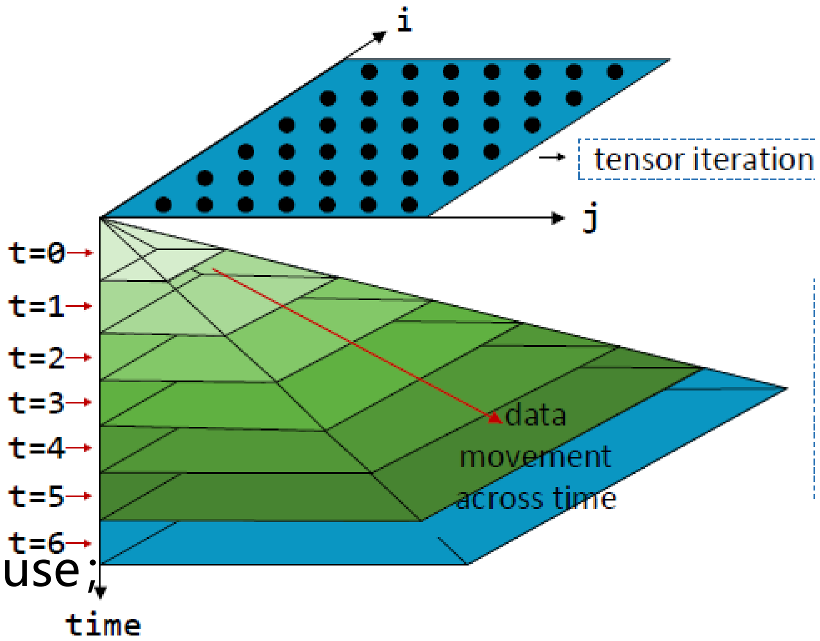
time-stamp 0. used data $A[0][0]$

time-stamp 1. used data $A[0][1], A[0][0]\ A[1][0]$

time-stamp 2. used data $A[0][2], A[0][1], A[1][1], A[1][0]$

time-stamp 3. used data $A[0][3], A[0][2], A[1][2], A[1][1]$

$\textbf{TotalVolume} = 1+3+4+4 = 12$



在一定时间间隔 interval 的两个时间戳访问了相同的数据，代表可能的Reuse；

如果，这两个数据相关联的语句在 相邻 的PE上执行/经过interval跳可以到达的PE上，代表Reuse

Performance Model: Reused Data

GEMM operation:

```
for  (i = 0; i < 2; i++)
  for  (j = 0; j < 2; j++)
    for  (k = 0; k < 4; k++)
          S: Y[i,j] +=
             A[i,k] * B[k,j];
```

Dataflow

space-stamp

{S[i,j,k] → PE[i,j]}

time-stamp

{S[i,j,k] → T[i+j+k]}

time-stamp 0. used data $A[0][0]$

time-stamp 1. used data $A[0][1],\ A[0][0]\ A[1][0]$

time-stamp 2. used data $A[0][2],\ A[0][1],\ A[1][1],\ A[1][0]$

time-stamp 3. used data $A[0][3],\ A[0][2],\ A[1][2],\ A[1][1]$

$\mathbf{TotalVolume} = 1+3+4+4 = 12$

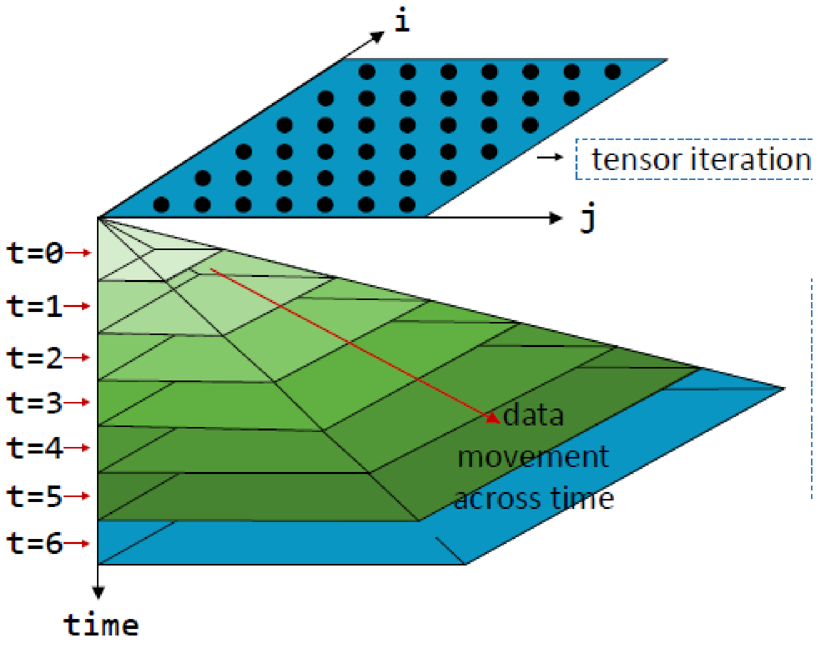
time-stamp 1. reused data from time-stamp 0 $A[0][0]$

time-stamp 2. reused data from time-stamp 1 $A[0][1],\ A[1][0]$

time-stamp 3. reused data from time-stamp 2 $A[0][2],\ A[1][1]$

$\mathbf{ReuseVolume} = 1+2+2 = 5$

|Reuse，
弋表Reuse



Performance Model: Reused Data

GEMM operation:

```
for  (i = 0; i < 2; i++)
  for  (j = 0; j < 2; j++)
    for  (k = 0; k < 4; k++)
          S: Y[i,j] +=
             A[i,k] * B[k,j];
```

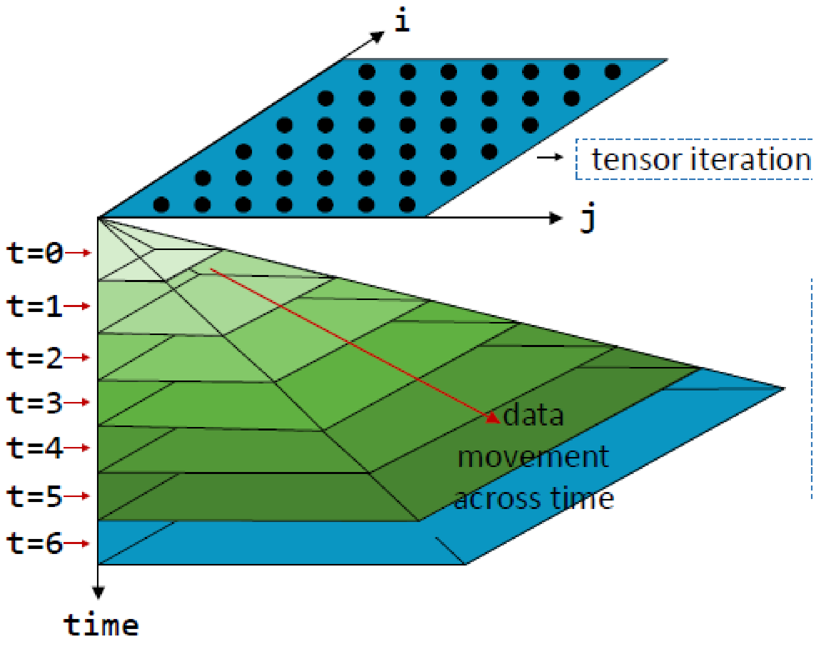
Dataflow

space-stamp

{S[i,j,k] → PE[i,j]}

time-stamp

{S[i,j,k] → T[i+j+k]}



time-stamp 1. reused data from time-stamp 0 $A[0][0]$

time-stamp 2. reused data from time-stamp 1 $A[0][1]$, $A[1][0]$

time-stamp 3. reused data from time-stamp 2 $A[0][2]$, $A[1][1]$

$\textbf{ReuseVolume} = 1 + 2 + 2 = 5$

发生在不同时间戳的复用，叫时间复用
发生在不同PE的复用，叫空间复用

Performance Model: Latency

$$Delay_{read} = \frac{UniqueVolume(Input)}{bandwidth}$$

$$Delay_{write} = \frac{UniqueVolume(Output)}{bandwidth}$$

Unique = 所有数据量 – reuse

访存延迟估算

$$Delay_{compute} = \frac{sum(D_S)}{Util_{PE} \times PE\ size}$$

计算延迟估算

Sum(Ds) 所有要执行的语句
UtilsPE = PE利用率

$$IBW = \frac{SpatialReuseVolume}{Delay_{compute}}$$

$$SBW = \frac{UniqueVolume}{Delay_{compute}}$$

所需要的带宽，

空间复用的总数据 / 计算延迟 = 网络带宽

独立的数据load / 计算延迟 = 片上内存带宽

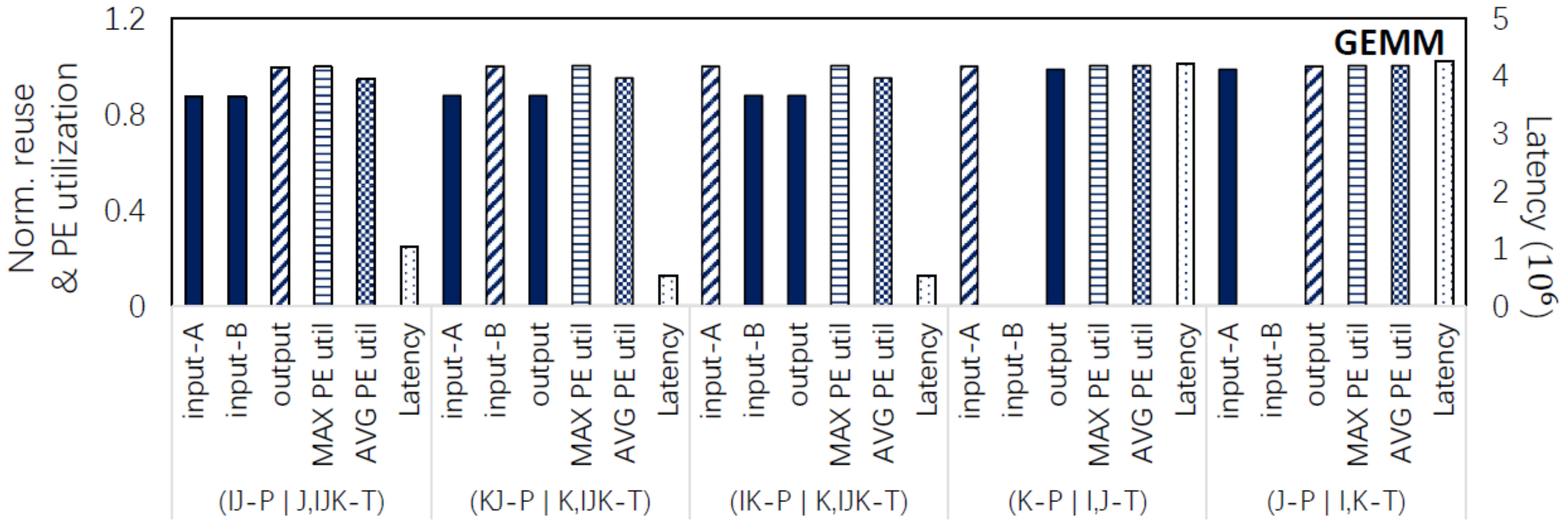
Evaluation

GEMM $Y(i,j) = A(i,k)B(k,j)$

Benchmark	Dataflow	Relation-centric
GEMM	(IJ-P | J,IJK-T) applied in TPU [22]	{S[i,j,k] → PE[i%8,j%8]} {S[i,j,k] → T[fl(i/8),fl(j/8),i%8+j%8+k]}
	(KJ-P | K,IJK-T)	{S[i,j,k] → PE[k%8,j%8]} {S[i,j,k] → T[fl(j/8),fl(k/8),i+j%8+k%8]}
	(IK-P | K,IJK-T)	{S[i,j,k] → PE[i%8,k%8]} {S[i,j,k] → T[fl(i/8),fl(k/8),j+i%8+k%8]}
	(K-P | I,J-T)	{S[i,j,k] → PE[k%64]} {S[i,j,k] → T[fl(k/64),i,j]}
	(J-P | I,K-T)	{S[i,j,k] → PE[j%64]} {S[i,j,k] → T[fl(j/64),i,k]}

fl() 代表向下取整,
IJ-P代表，i , j 给不同的PE
K，IJK –T 代表，时间戳 = T[k, I + j + k]

Evaluation



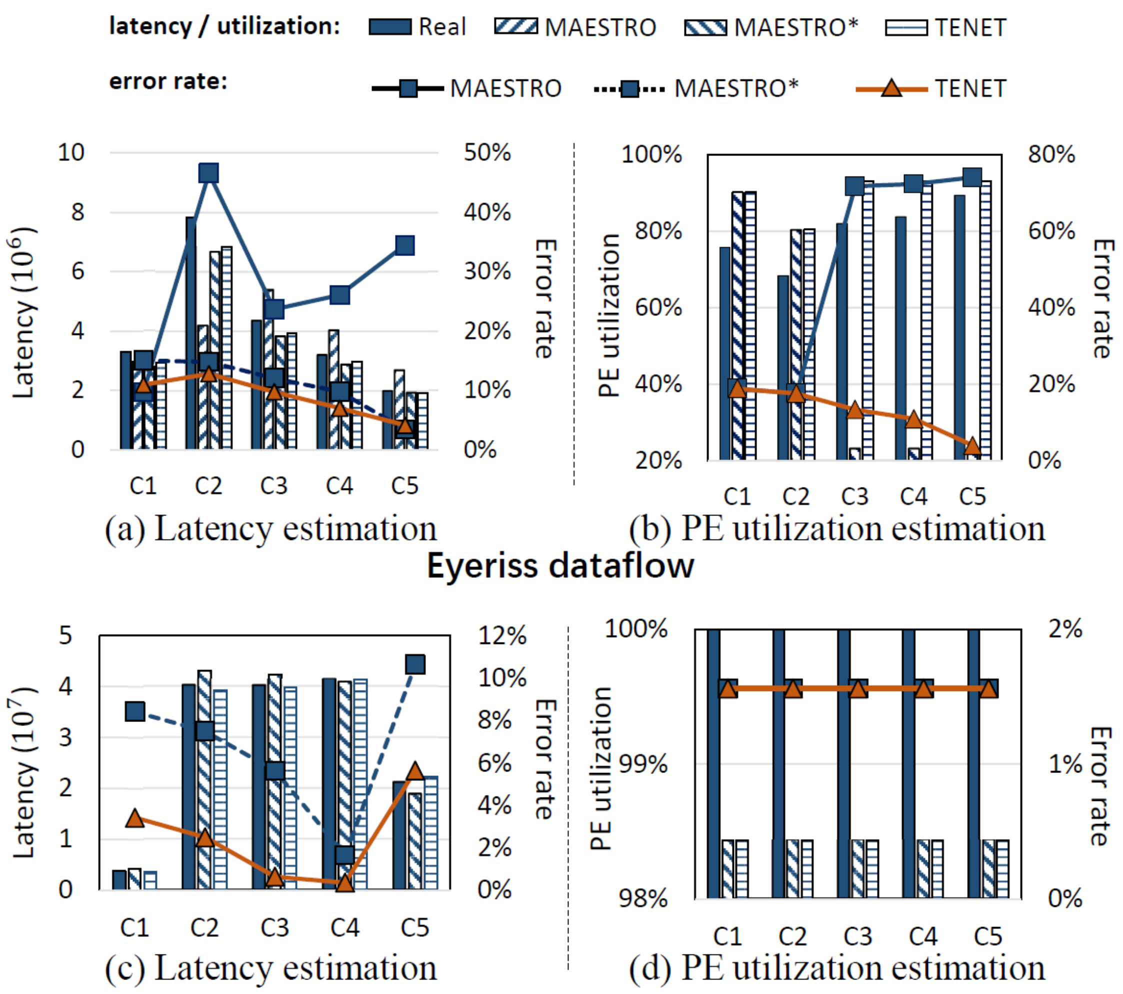
同样的算法，不同数据流，数据复用情况、latency都不同

Evaluation: 模拟准确率

use the reported latency
and PE utilization in Eyeriss and MAERI as the golden result

模拟错误率和之前类似的工作比下降很明显

主要由于，关系形式表达数据流的表达能力很强。



(a) Latency estimation
(b) PE utilization estimation
Eyeriss dataflow
(c) Latency estimation
(d) PE utilization estimation
MAERI dataflow

Thanks

Q&A