SparseTrain: Leveraging Dynamic Sparsity in Software for Training DNNs on General-Purpose SIMD Processors

Zhangxiaowen Gong, Houxiang Ji, Christopher W. Fletcher
Christopher J. Hughes*, Josep Torrellas

University of Illinois at Urbana-Champaign, *Intel Labs

PACT 2020

ILLINOIS intel

Sparsity and Work Skipping

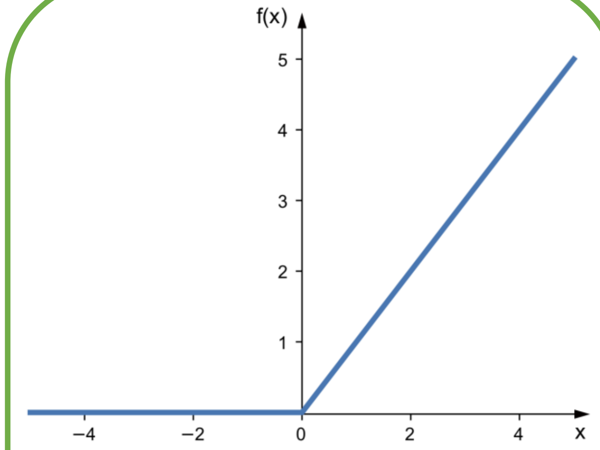
- Sparsity: zeros inside a data structure
- Multiply-Accumulate Operation (MAC): $c = c + a \times b$
- Zero in multiplicands: skippable!

skippable

$$c + 0 \times \color{red}{b} = c$$

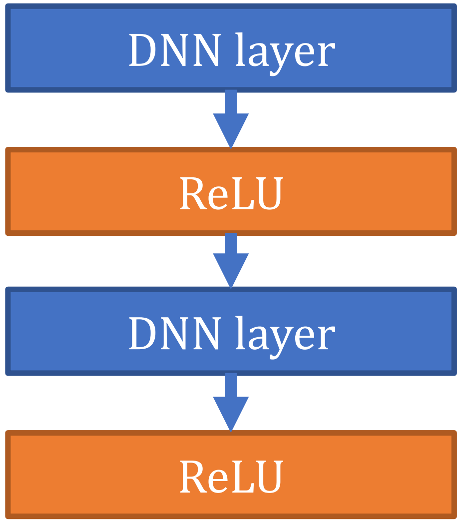
Dynamic Sparsity in DNNs

- Zeros appear dynamically during training/inference
- Main source: ReLU
  - Activates the neurons of a DNN layer
  - Produces 40-90% sparsity
- Hard to exploit
  - Unstructured
  - Changes over time
  - Moderate sparsity level



$$f(x) = \max(0, x)$$

$$f'(x) = \begin{cases} 1, if\ x > 0 \\ 0, otherwise \end{cases}$$



ILLINOIS intel

Contribution: SparseTrain

- The first software-only algorithm to speedup DNN training by exploiting dynamic sparsity
  - Skips computation at runtime
  - Uses a regular (dense) representation
  - Generates JIT kernels
  - Vectorized
- Applicable to all training phases
  - Forward propagation **(same as inference)**
  - Backward input propagation
  - Backward weight propagation

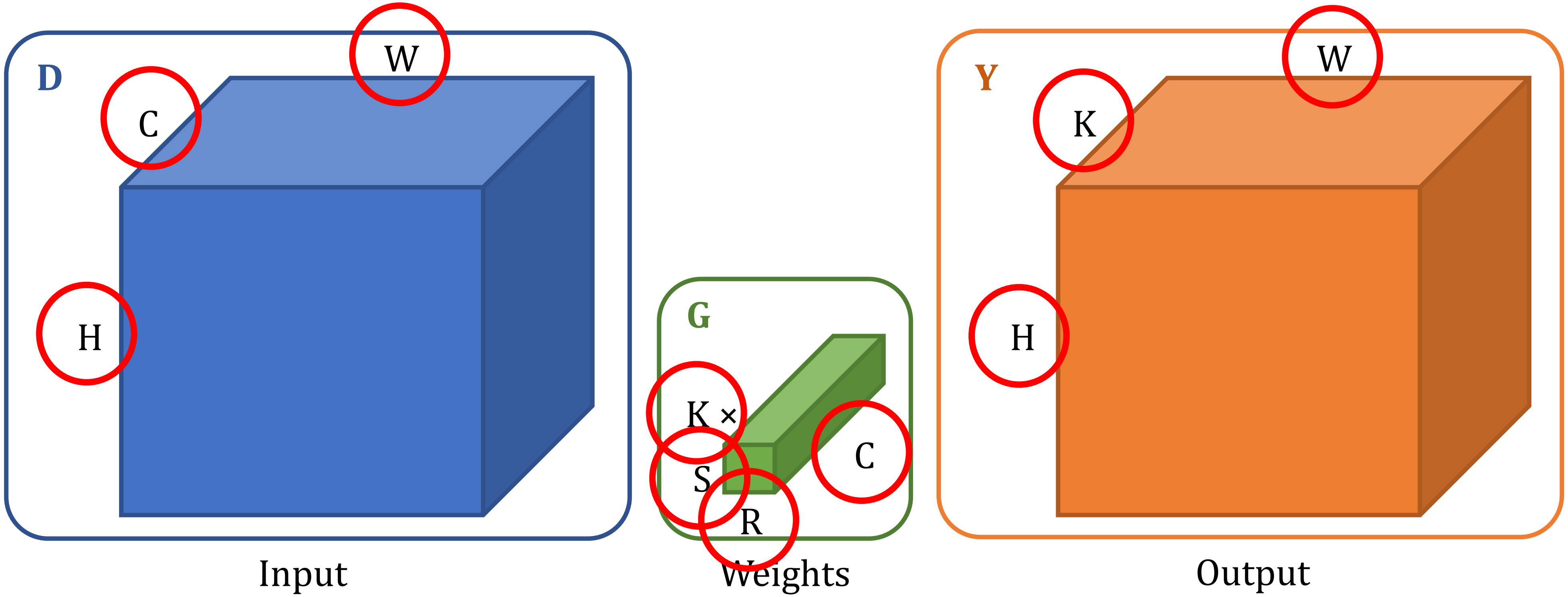
SparseTrain for Convolutional Neural Networks (CNNs)

- SparseTrain is applicable to GEMM-like computation; we focus on CNNs.
- Convolutional layers:
  - High compute-to-memory ratio
  - Most time consuming component
  - Usually activated by ReLU
- Accelerates training by 1.3x-2.2x and inference by 1.4x-1.9x
- In this talk, we present *forward propagation*
  - Backward input/weight propagations are similar

Forward Convolution

$$Y_{i,k,x,y} = \sum_{c=0}^{C-1} \sum_{u=0}^{R-1} \sum_{v=0}^{S-1} D_{i,c,x+u,y+v} \times G_{k,c,u,v}$$

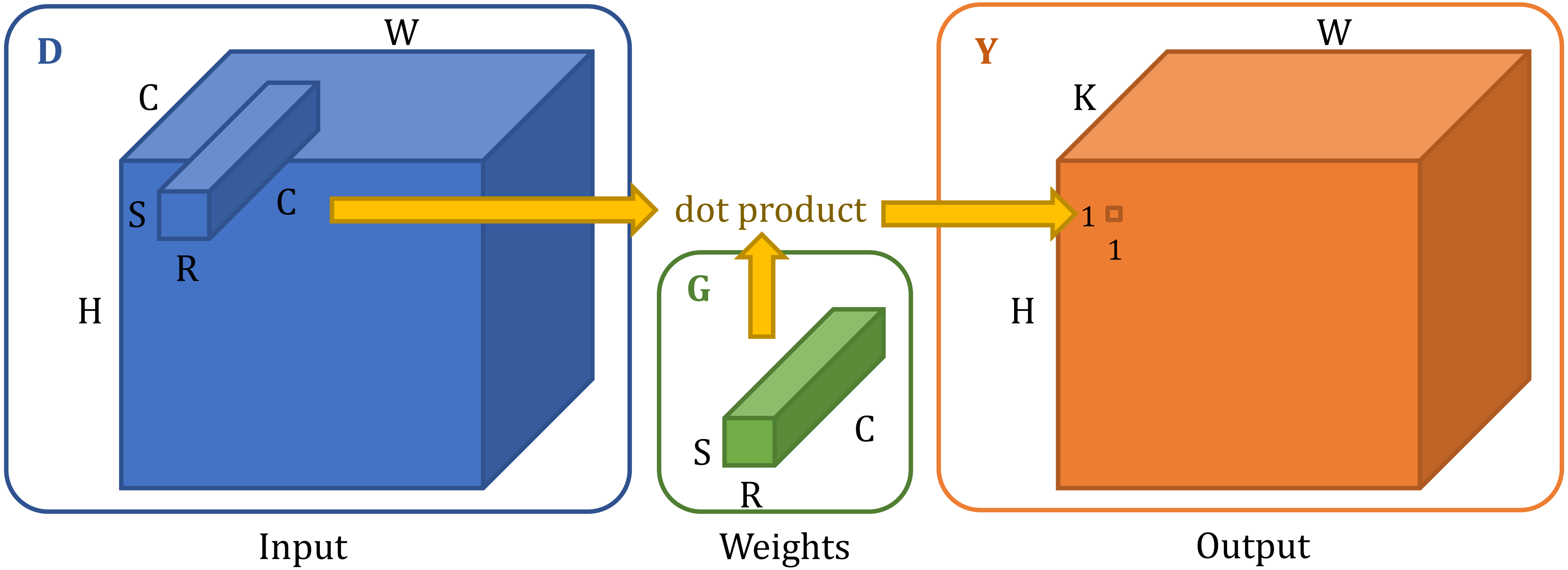
$i \in N \quad k \in K \quad c \in C$
$y \in H \quad x \in W \quad u \in R$
$v \in S$



Forward Convolution

$$Y_{i,k,x,y} = \sum_{c=0}^{C-1} \sum_{u=0}^{R-1} \sum_{v=0}^{S-1} D_{i,c,x+u,y+v} \times G_{k,c,u,v}$$

$i \in N$ $k \in K$ $c \in C$
$y \in H$ $x \in W$ $u \in R$
$v \in S$

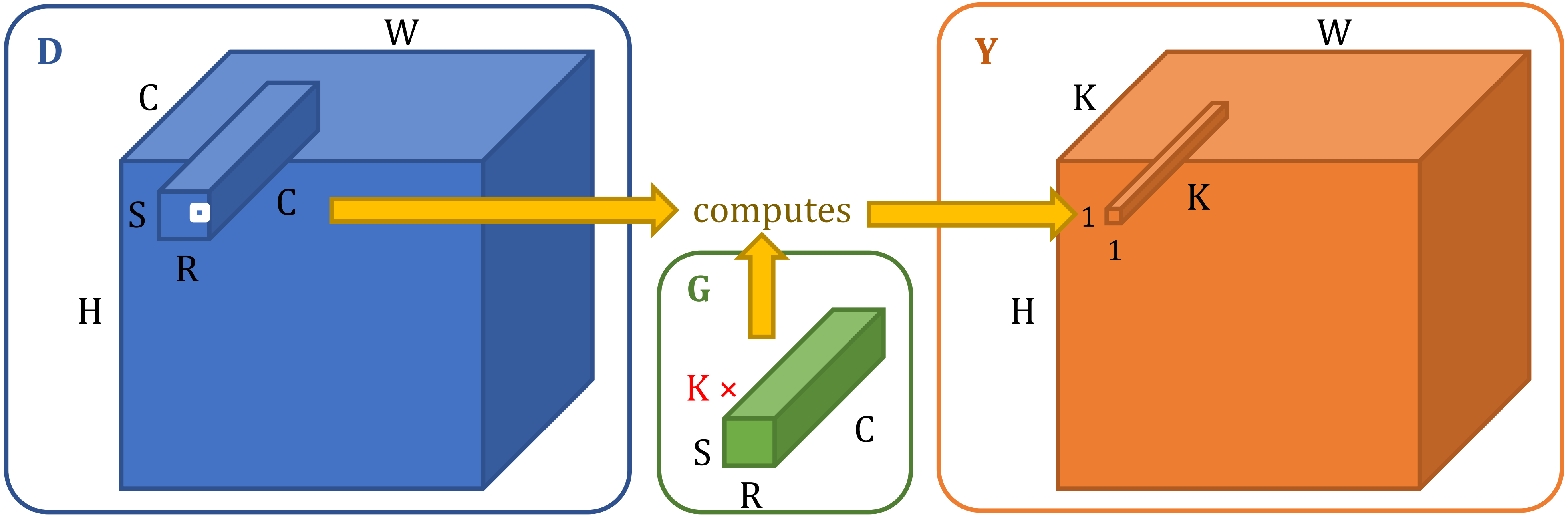


ILLINOIS intel

Forward Convolution

$$Y_{i,k,x,y} = \sum_{c=0}^{C-1} \sum_{u=0}^{R-1} \sum_{v=0}^{S-1} D_{i,c,x+u,y+v} \times G_{k,c,u,v}$$

$i \in N \quad k \in K \quad c \in C$
$y \in H \quad x \in W \quad u \in R$
$v \in S$



Input

Weights

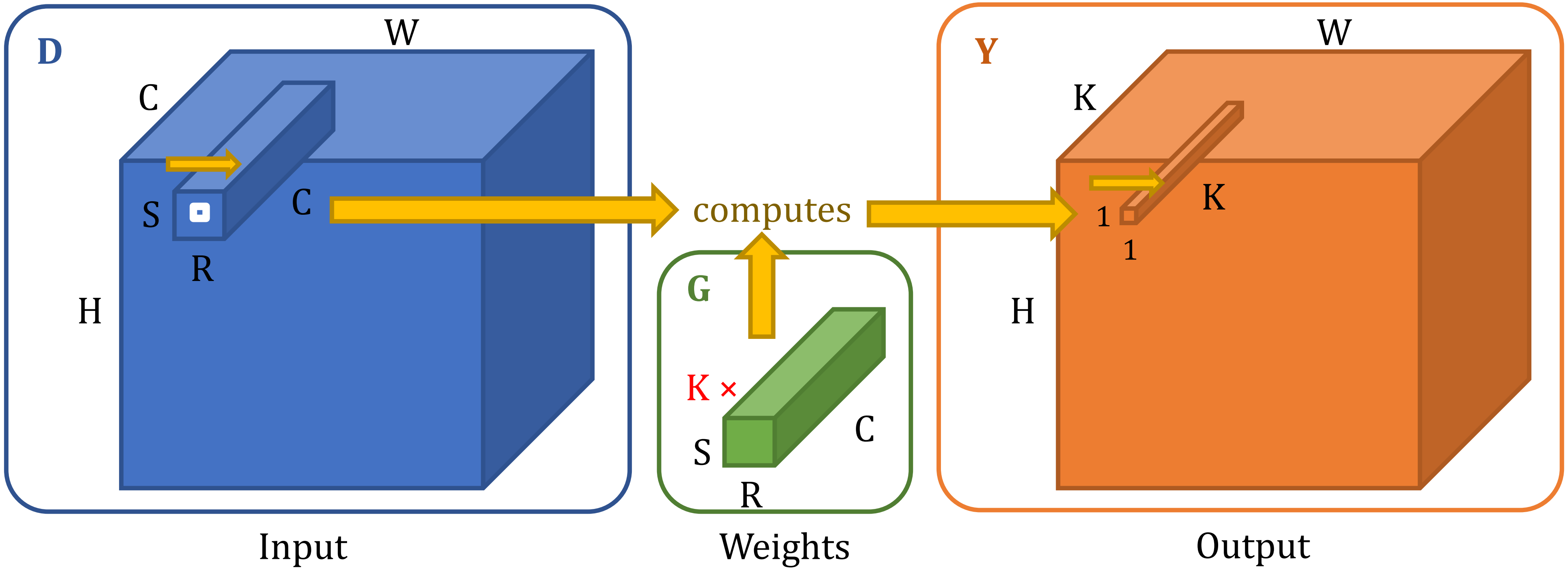
Output

ILLINOIS intel

Forward Convolution

$$Y_{i,k,x,y} = \sum_{c=0}^{C-1} \sum_{u=0}^{R-1} \sum_{v=0}^{S-1} D_{i,c,x+u,y+v} \times G_{k,c,u,v}$$

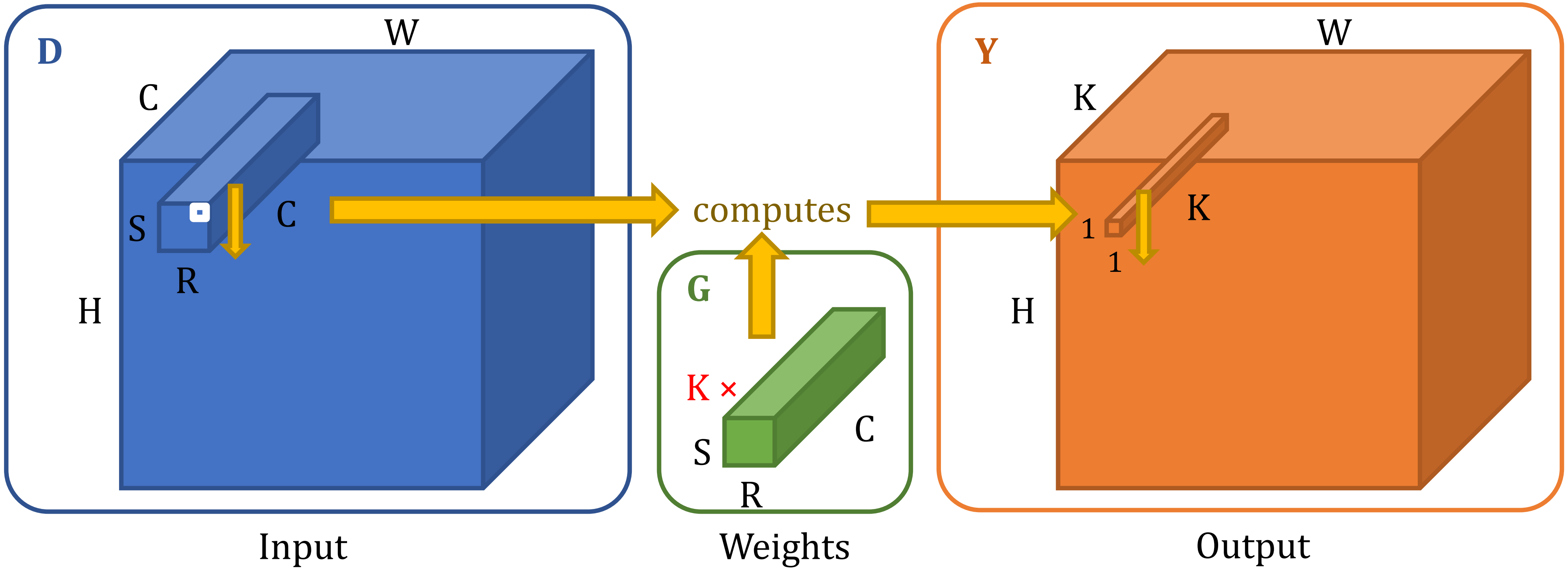
$i \in N \quad k \in K \quad c \in C$
$y \in H \quad x \in W \quad u \in R$
$v \in S$



Forward Convolution

$$Y_{i,k,x,y} = \sum_{c=0}^{C-1} \sum_{u=0}^{R-1} \sum_{v=0}^{S-1} D_{i,c,x+u,y+v} \times G_{k,c,u,v}$$

$i \in N \quad k \in K \quad c \in C$
$y \in H \quad x \in W \quad u \in R$
$v \in S$



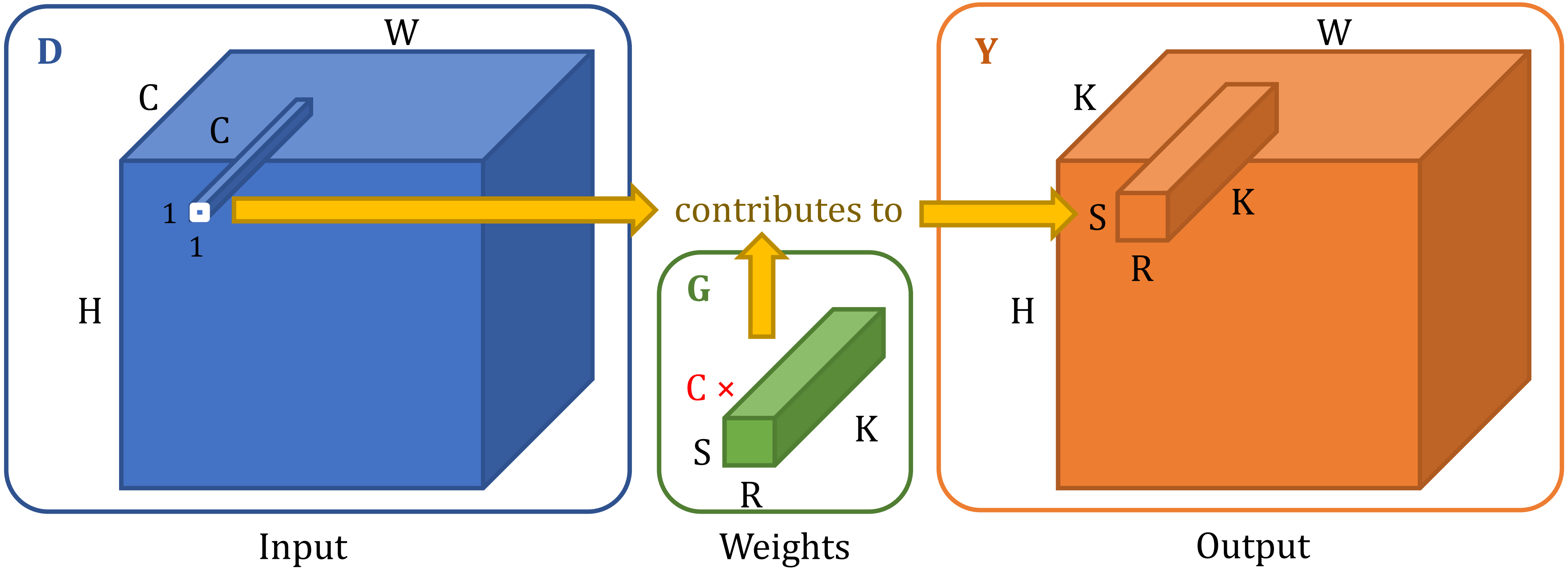
Forward Convolution

$$Y_{i,k,x,y} = \sum_{c=0}^{C-1} \sum_{u=0}^{R-1} \sum_{v=0}^{S-1} D_{i,c,x+u,y+v} \times G_{k,c,u,v}$$

$i \in N \quad k \in K \quad c \in C$

$y \in H \quad x \in W \quad u \in R$

$v \in S$



ILLINOIS intel

Naïve Sparse Forward Convolution

$$Y_{i,k,x,y} = \sum_{c=0}^{C-1} \sum_{u=0}^{R-1} \sum_{v=0}^{S-1} D_{i,c,x+u,y+v} \times G_{k,c,u,v}$$

```
for i in 0 to N-1
  for k in 0 to K-1
    for y in 0 to H-1
      for x in 0 to W-1
        for c in 0 to C-1
          for v in 0 to S-1
            for u in 0 to R-1
              Y[i,k,x,y] += D[i,c,x+u,y+v] G[k,c,u,v]
```

stationary

Naïve Sparse Forward Convolution

- Step 1: permute the computation loop nest to station the input

```
for i in 0 to N-1
  for k in 0 to K-1
    for y in 0 to H-1
      for x in 0 to W-1
        for c in 0 to C-1
          for v in 0 to S-1
            for u in 0 to R-1
              Y_{i,k,x,y} += D_{i,c,x+u,y+v} G_{k,c,u,v}
```

transform

```
for i in 0 to N-1
  for c in 0 to C-1
    for y in S-1 to H+S-2
      for x in R-1 to W+R-2

        for k in 0 to K-1
          for v in 0 to S-1
            for u in 0 to R-1
              Y_{i, k ,x-u,y-v} += D_{i,c,x,y} G
```

stationary

Naïve Sparse Forward Convolution

- Step 1: permute the computation loop nest to station the input
- Step 2: check the input for zero and skip compute accordingly

```
for i in 0 to N-1
  for k in 0 to K-1
    for y in 0 to H-1
      for x in 0 to W-1
        for c in 0 to C-1
          for v in 0 to S-1
            for u in 0 to R-1
              Y_{i,k,x,y} += D_{i,c,x+u,y+v} G_{k,c,u,v}
```

transform

```
for i in 0 to N-1
  for c in 0 to C-1
    for y in S-1 to H+S-2
      for x in R-1 to W+R-2
        if D_{i,c,x,y} ≠ 0
          for k in 0 to K-1
            for v in 0 to S-1
              for u in 0 to R-1
                Y_{i,k,x-u,y-v} += D_{i,c,x,y} G_{k,c,u,v}
```

Naïve Sparse Forward Convolution

- Step 1: permute the computation loop nest to station the input
- Step 2: check the input for zero and skip compute accordingly
- Step 3: vectorize the compute along the output channel dimension K

```
for i in 0 to N-1
  for k in 0 to K-1
    for y in 0 to H-1
      for x in 0 to W-1
        for c in 0 to C-1
          for v in 0 to S-1
            for u in 0 to R-1
              Y_{i,k,x,y} += D_{i,c,x+u,y+v} G_{k,c,u,v}
```

transform

```
for i in 0 to N-1
  for c in 0 to C-1
    for y in S-1 to H+S-2
      for x in R-1 to W+R-2
        if D_{i,c,x,y} ≠ 0
          for k in 0 to K-V step V
            for v in 0 to S-1
              for u in 0 to R-1
                Y_{i,[k:k+V-1],x-u,y-v}
                      += D_{i,c,x,y} G_{[k:k+V-1],c,u,v}
```

ILLINOIS intel

Naïve Sparse Forward Convolution: Problems

- Hard to parallelize

```
for i in 0 to N-1 in parallel
  for c in 0 to C-1
    for y in S-1 to H+S-2
      for x in R-1 to W+R-2
        if D_{i,c,x,y}≠0
          for k in 0 to K-V step V
            for v in 0 to S-1
              for u in 0 to R-1
                Y_{i,[k:k+V-1],x-u,y-v}
                      +=D_{i,c,x,y}G_{[k:k+V-1],c,u,v}
```

too coarse

update multiple output vectors

ILLINOIS intel

Naïve Sparse Forward Convolution: Problems

- Hard to parallelize
- Register spilling

```
for i in 0 to N-1
  for c in 0 to C-1
    for y in S-1 to H+S-2
      for x in R-1 to W+R-2
        if D_{i,c,x,y}≠0
          for k in 0 to K-V step V
            for v in 0 to S-1
              for u in 0 to R-1
                Y_{i,[k:k+V-1],x-u,y-v}
                        +=D_{i,c,x,y}G_{[k:k+V-1],c,u,v}
```

update R×S×K/V output vectors

Naïve Sparse Forward Convolution: Problems

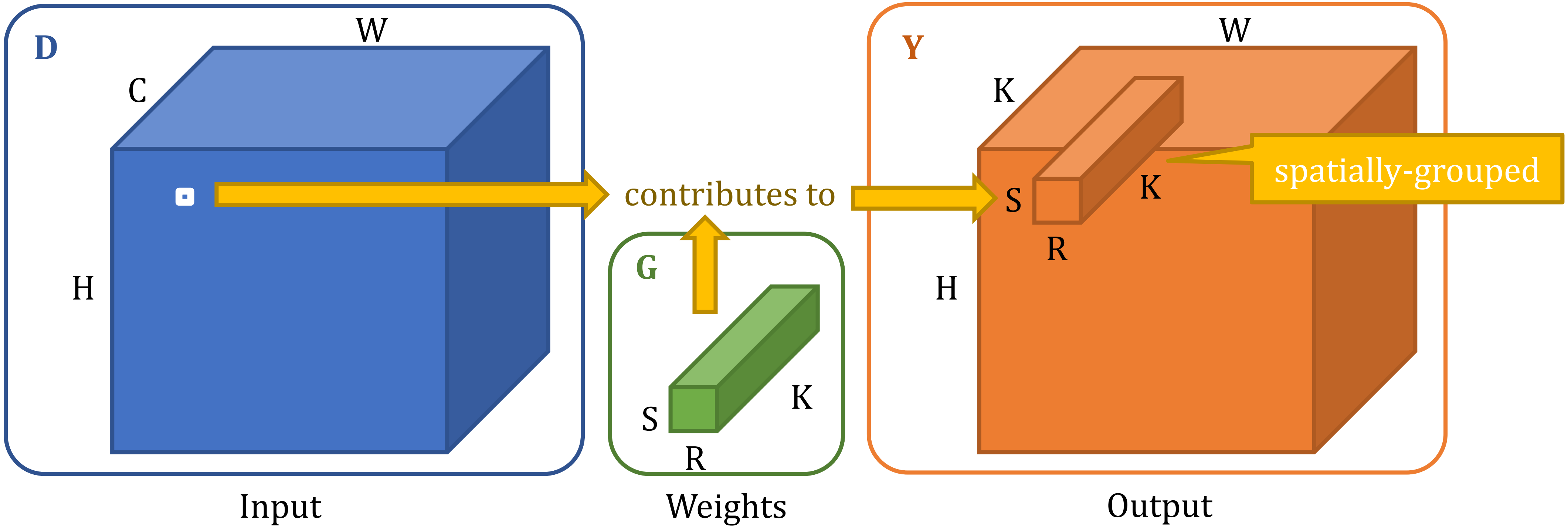
- Hard to parallelize
- Register spilling
- Branch misprediction

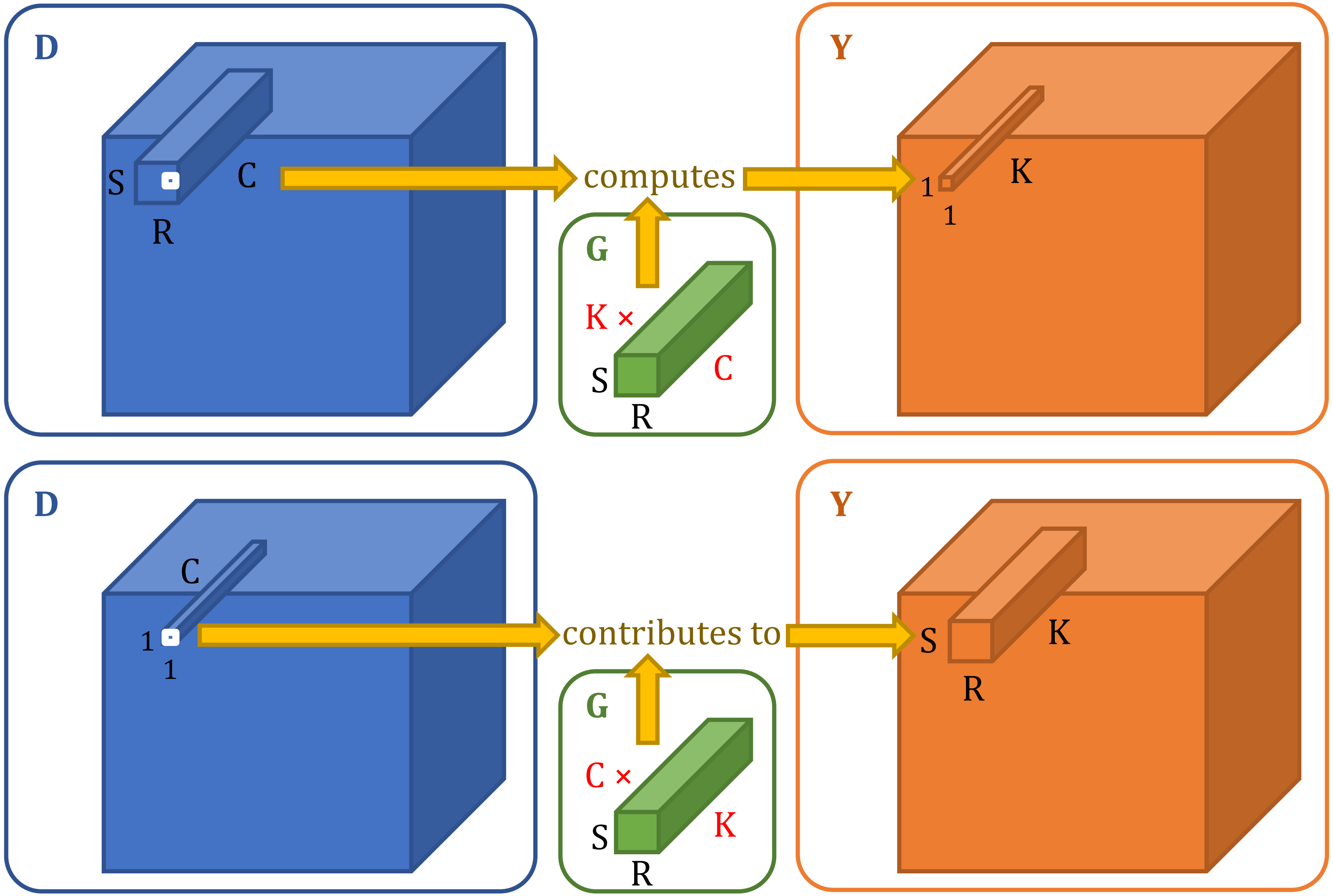
unpredictable branch

```
for i in 0 to N-1
  for c in 0 to C-1
    for y in S-1 to H+S-2
      for x in R-1 to W+R-2
        if D_{i,c,x,y}≠0
          for k in 0 to K-V step V
            for v in 0 to S-1
              for u in 0 to R-1
                Y_{i,[k:k+V-1],x-u,y-v}
                     +=D_{i,c,x,y}G_{[k:k+V-1],c,u,v}
```

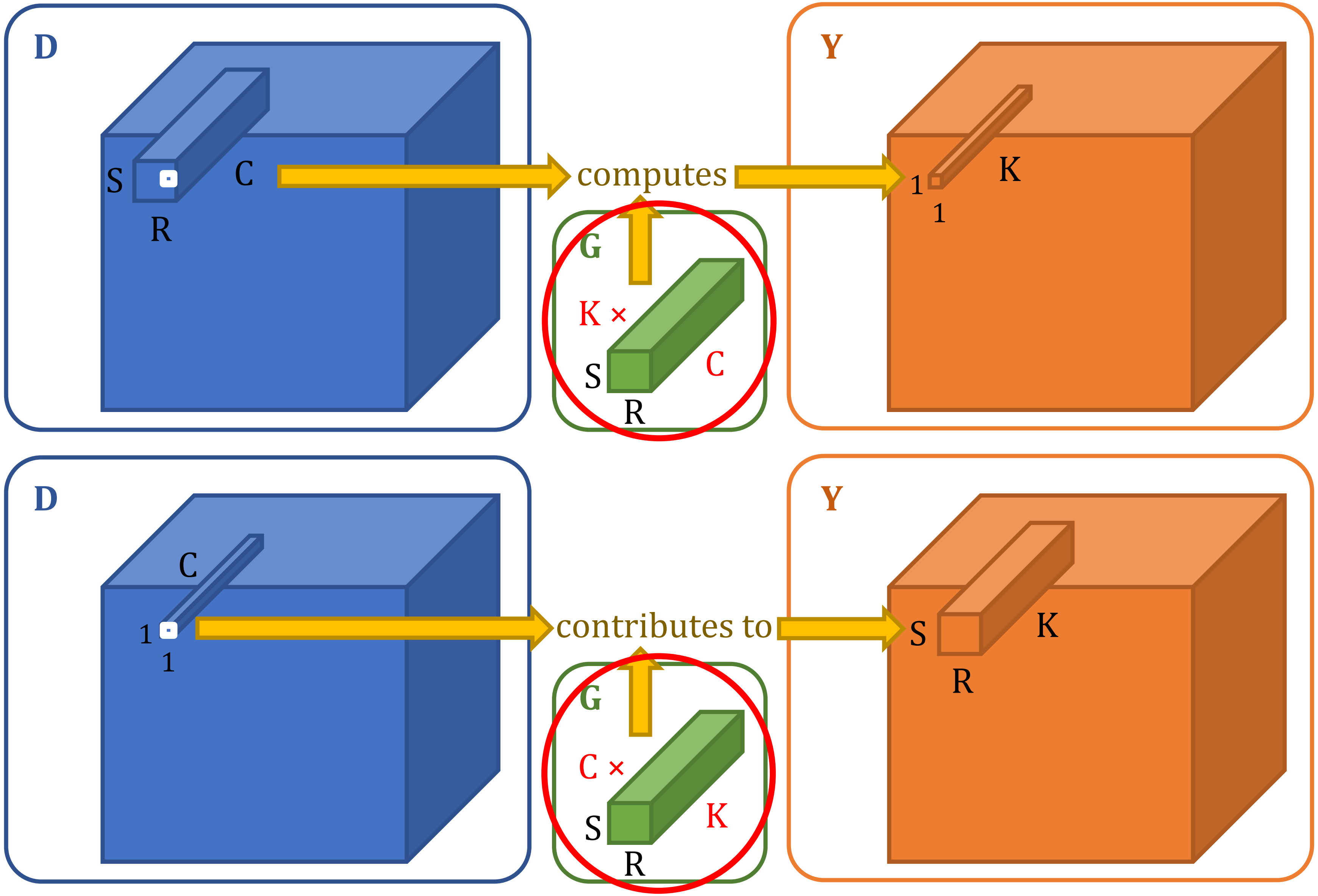
Optimization 1: Building Parallel Tasks

- Each input contributes to spatially-grouped outputs
- Reduce skippable computation per zero-checking

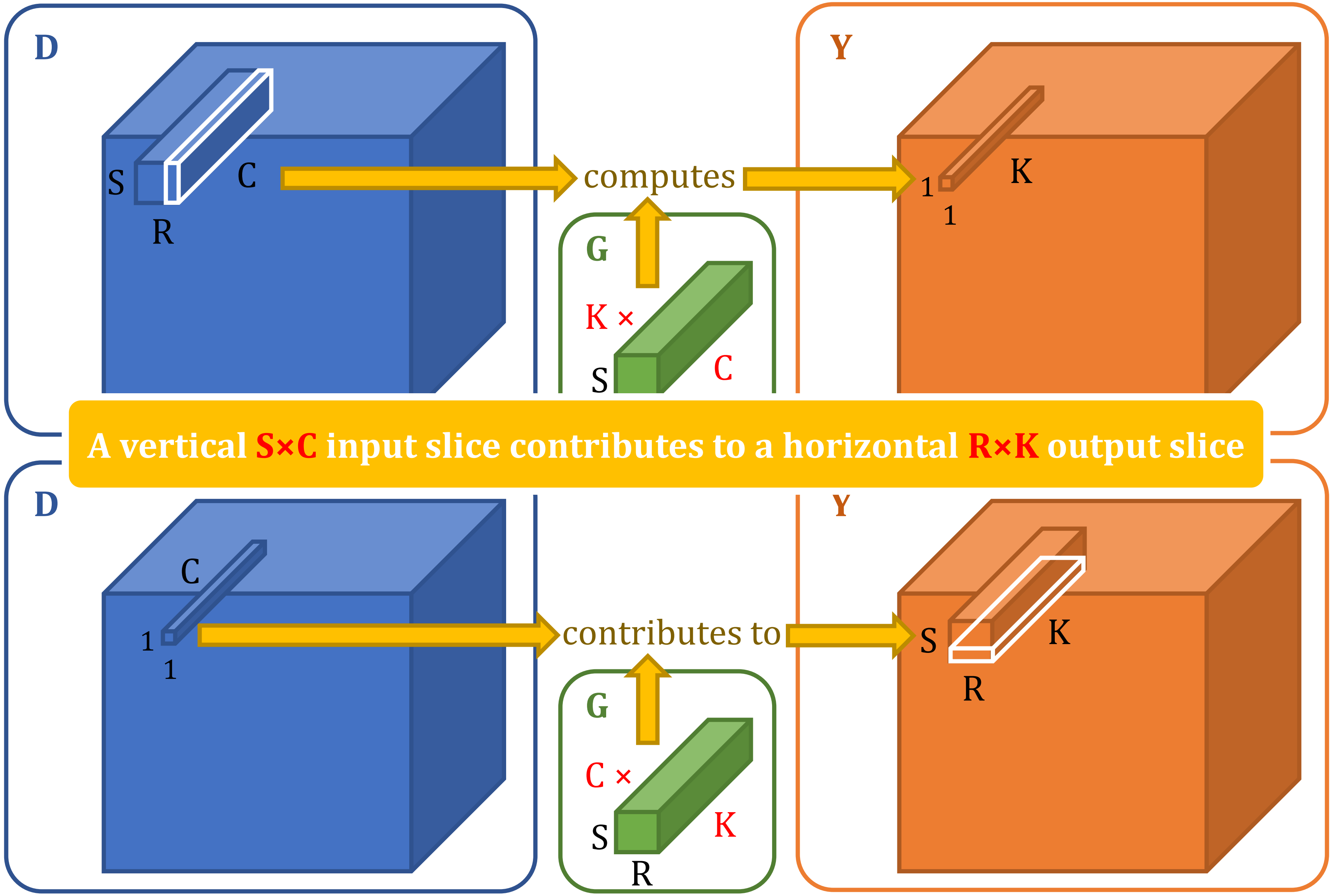




D
S
R
C
computes
G
K ×
S
R
C
Y
1
1
K
D
C
1
1
contributes to
G
C ×
S
R
K
Y
S
R
K



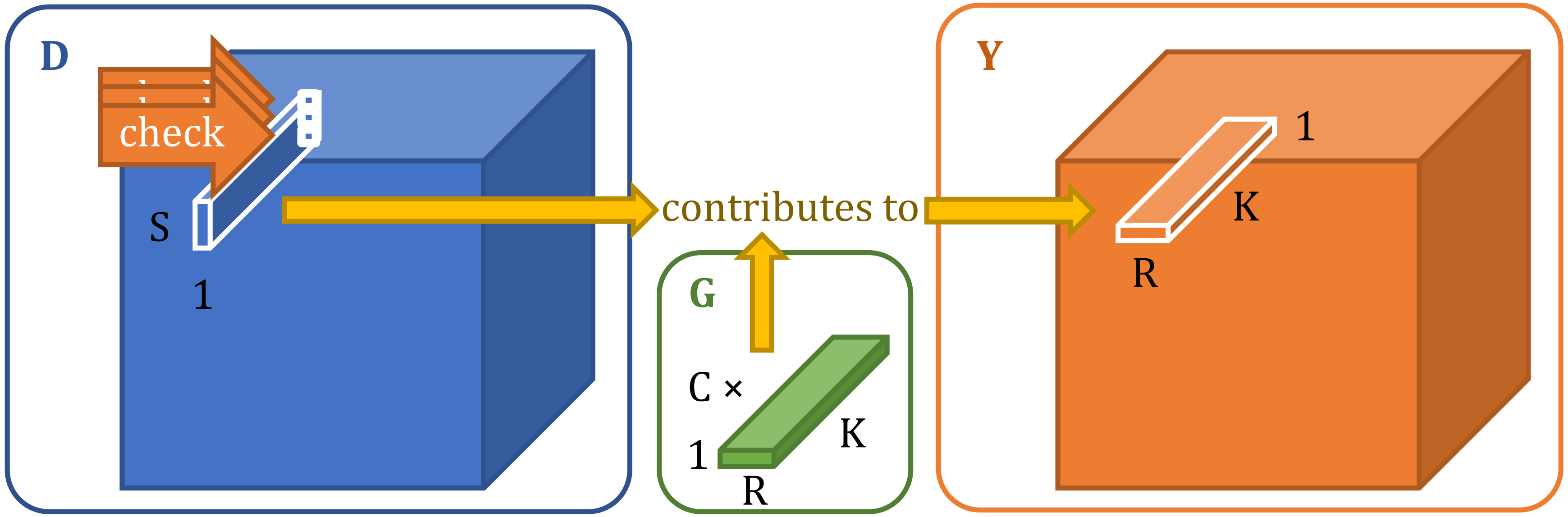
ILLINOIS intel



A vertical S×C input slice contributes to a horizontal R×K output slice

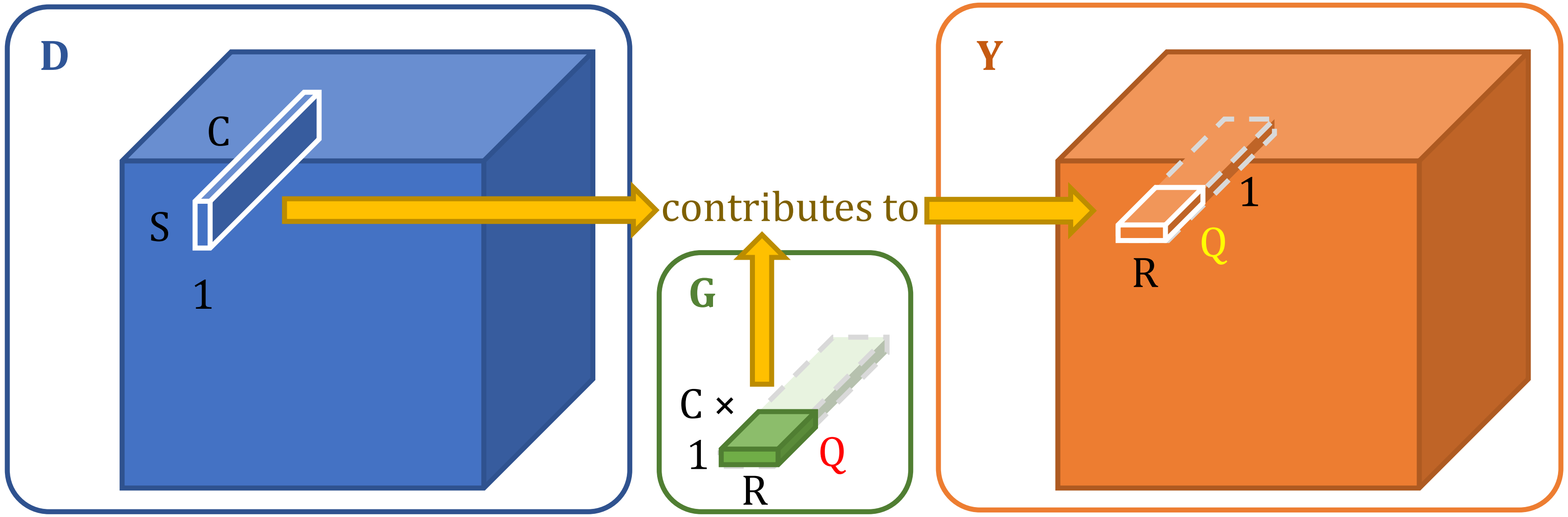
Optimization 1: Building Parallel Tasks

- Check each element in input slice before moving to next output slice



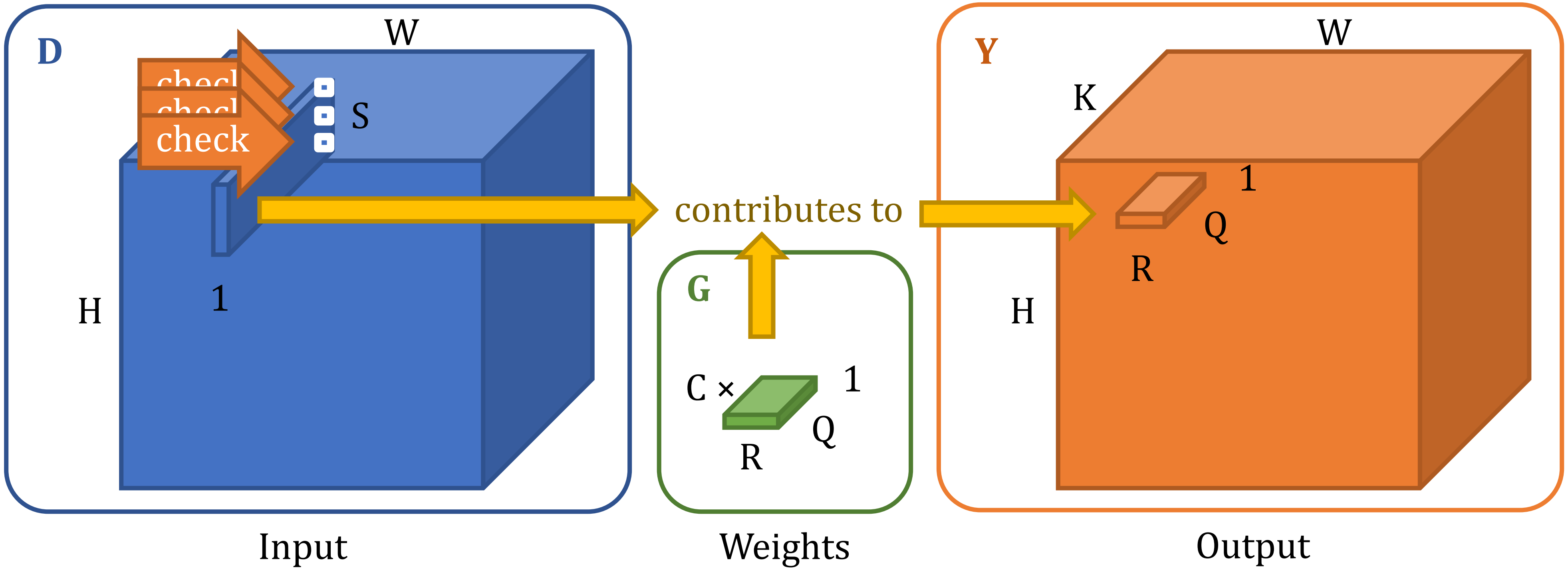
Optimization 1: Building Parallel Tasks

- Check each element in input slice before moving to next output slice
- Tackle register spilling: shrink output slice to R×Q
  - Q is a factor of K
  - Goal: R×Q < register budget



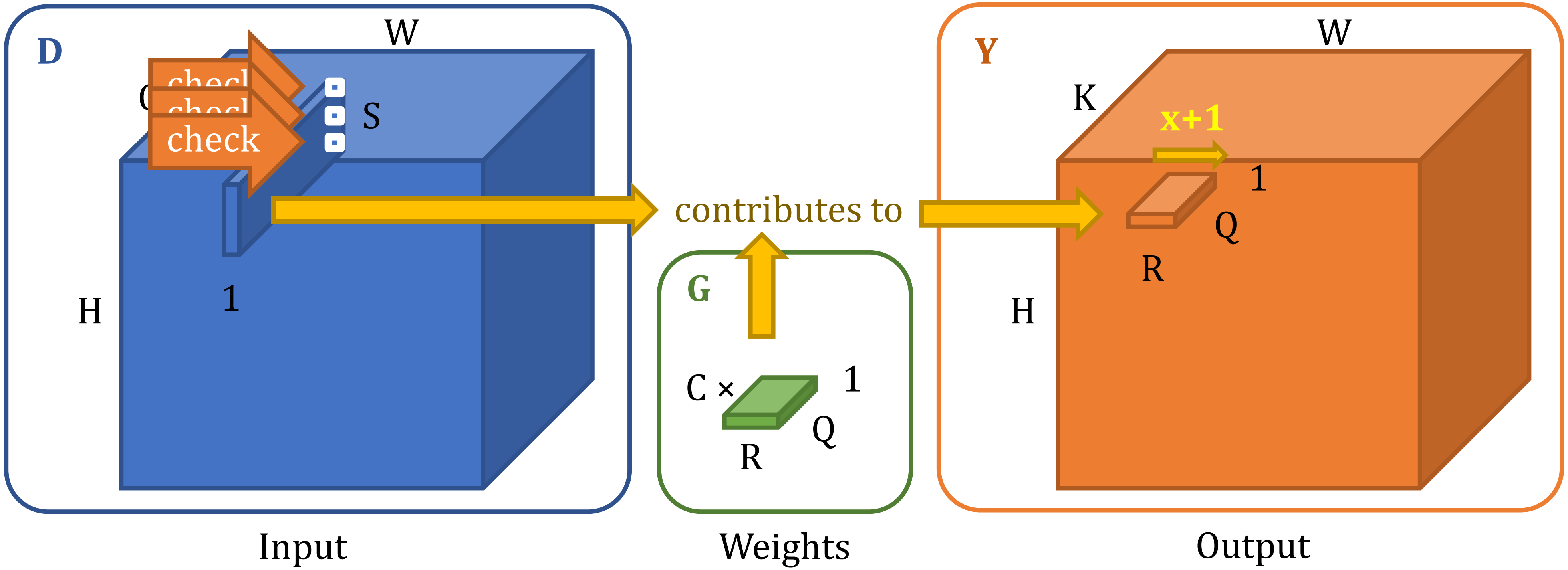
Optimization 1: Building Parallel Tasks

- Each parallel task executes a loop



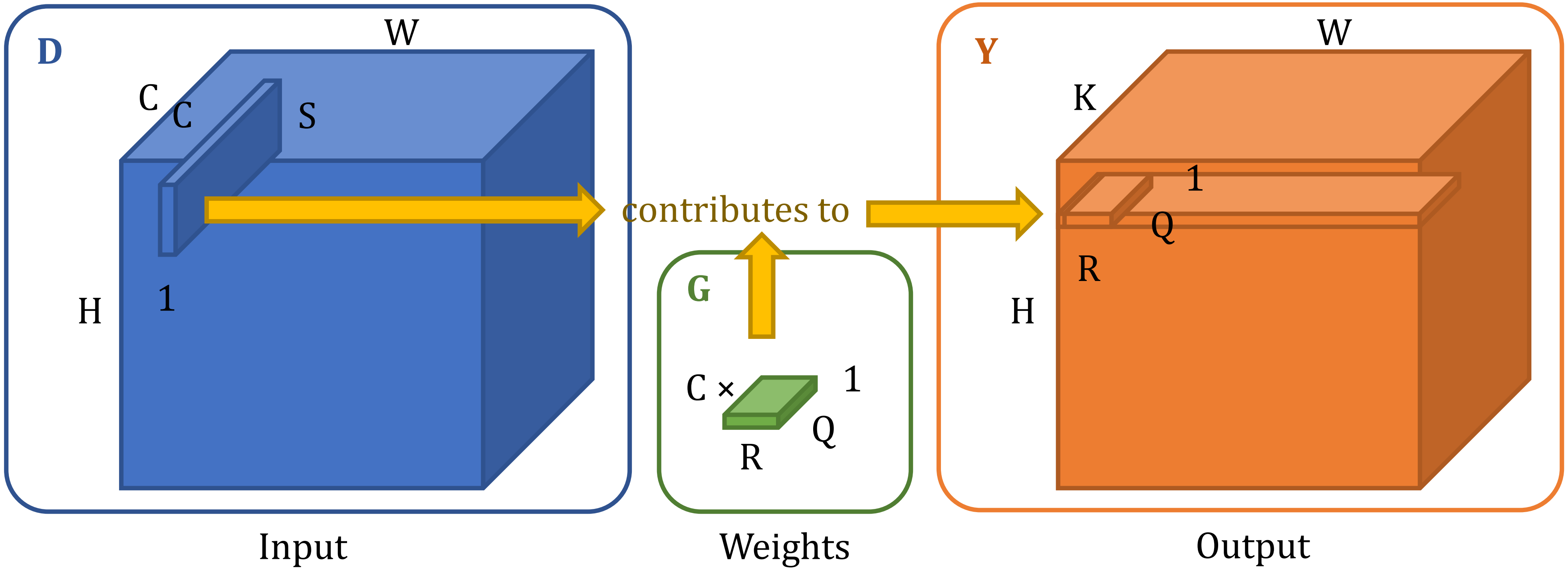
Optimization 1: Building Parallel Tasks

- Each parallel task executes a loop



Optimization 1: Building Parallel Tasks

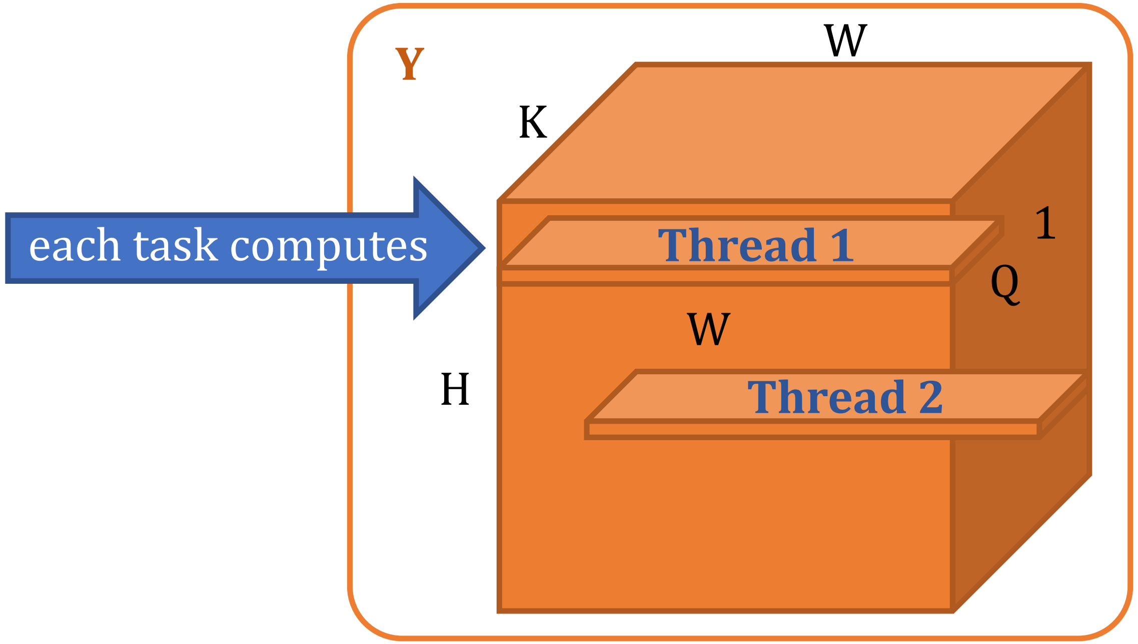
- Sweeps through the output width dimension W and updates a W×Q output row
  - Called row sweep



ILLINOIS intel

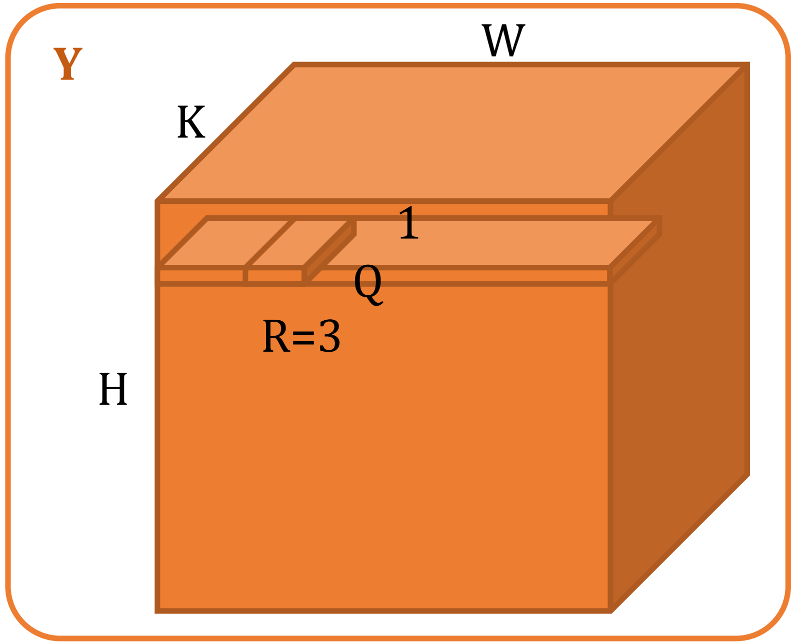
Optimization 1: Building Parallel Tasks

- Each task computes W×Q distinct outputs: no race condition
- Creates many parallel tasks to avoid load imbalance



Optimization 2: Efficient Vector Register Usage

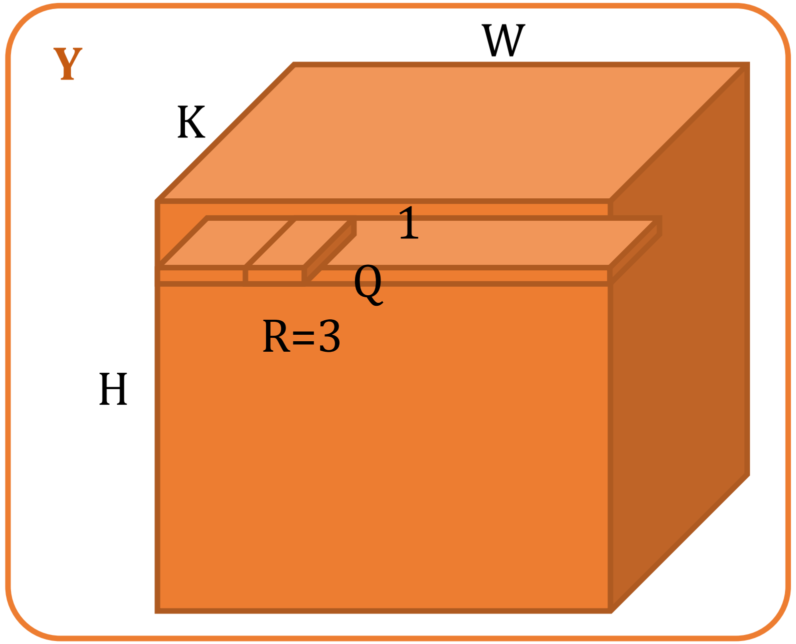
- Minimize memory accesses: keep outputs shared between iterations in registers
- Avoid register-to-register data transfer: cyclically rename registers



Optimization 2: Efficient Vector Register Usage

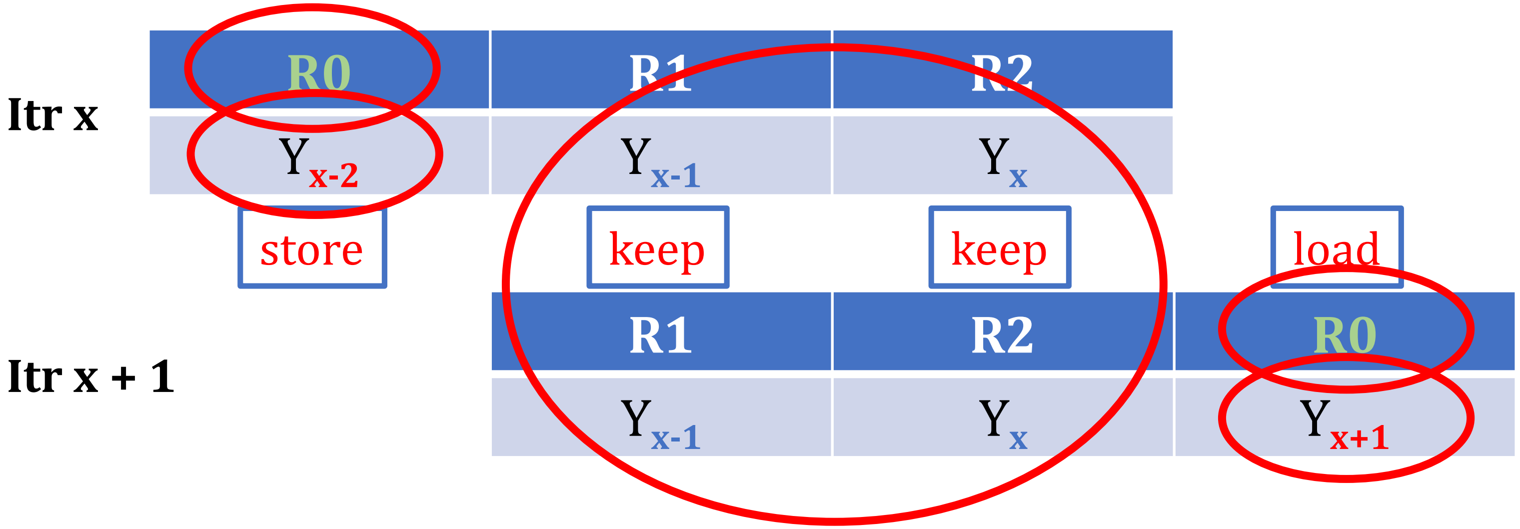
- Minimize memory accesses: keep outputs shared between iterations in registers
- Avoid register-to-register data transfer: cyclically rename registers

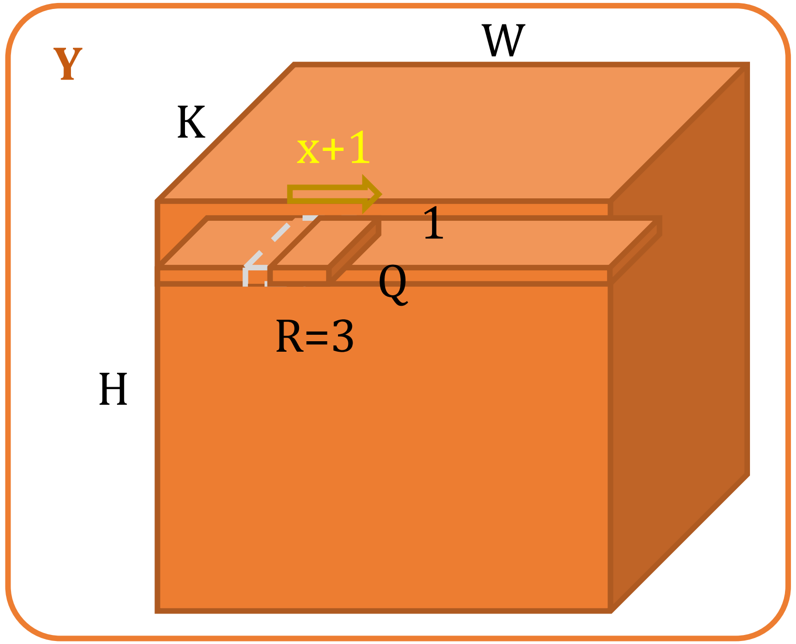
	R0	R1	R2
Itr x	Y_{x-2}	Y_{x-1}	Y_x



Optimization 2: Efficient Vector Register Usage

- Minimize memory accesses: keep outputs shared between iterations in registers
- Avoid register-to-register data transfer: cyclically rename registers





Optimization 3: Reducing Branch Mispredictions

- Step 1: vectorize zero-checking along input channel dimension C

```
for i in 0 to N-1
  for c in 0 to C-1
    for y in S-1 to H+S-2
      for x in R-1 to W+R-2
        if D_{i,c,x,y}≠0
          ...
```

vectorize

```
for i in 0 to N-1
  for c in 0 to C-V step V
    for y in S-1 to H+S-2
      for x in R-1 to W+R-2
        m_{[0:V-1]}=vect_cmp_neq_zero(D_{i,[c:c+V-1],x,y})
        for c′ in 0 to V-1
          if m_{c′} is true
          ...
```

$D_{i,[c:c+V-1],x,y}$

0	0	25	12	0	4	0	0

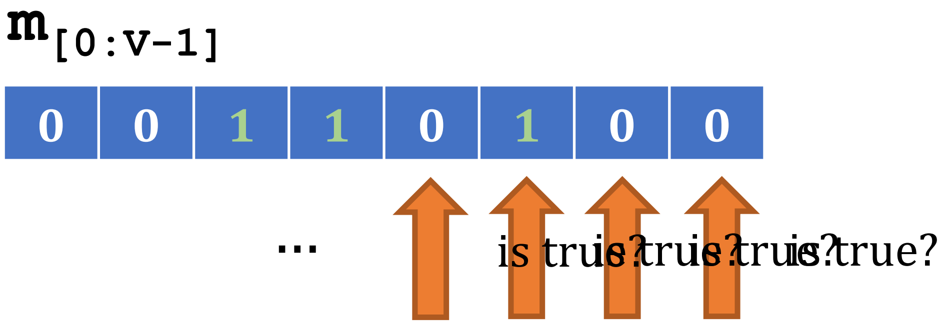
vector compare

$m_{[0:V-1]}$

0	0	1	1	0	1	0	0

Optimization 3: Reducing Branch Mispredictions

- Step 1: vectorize zero-checking along input channel dimension C
- Step 2: transform a series of `if` statements into a single loop

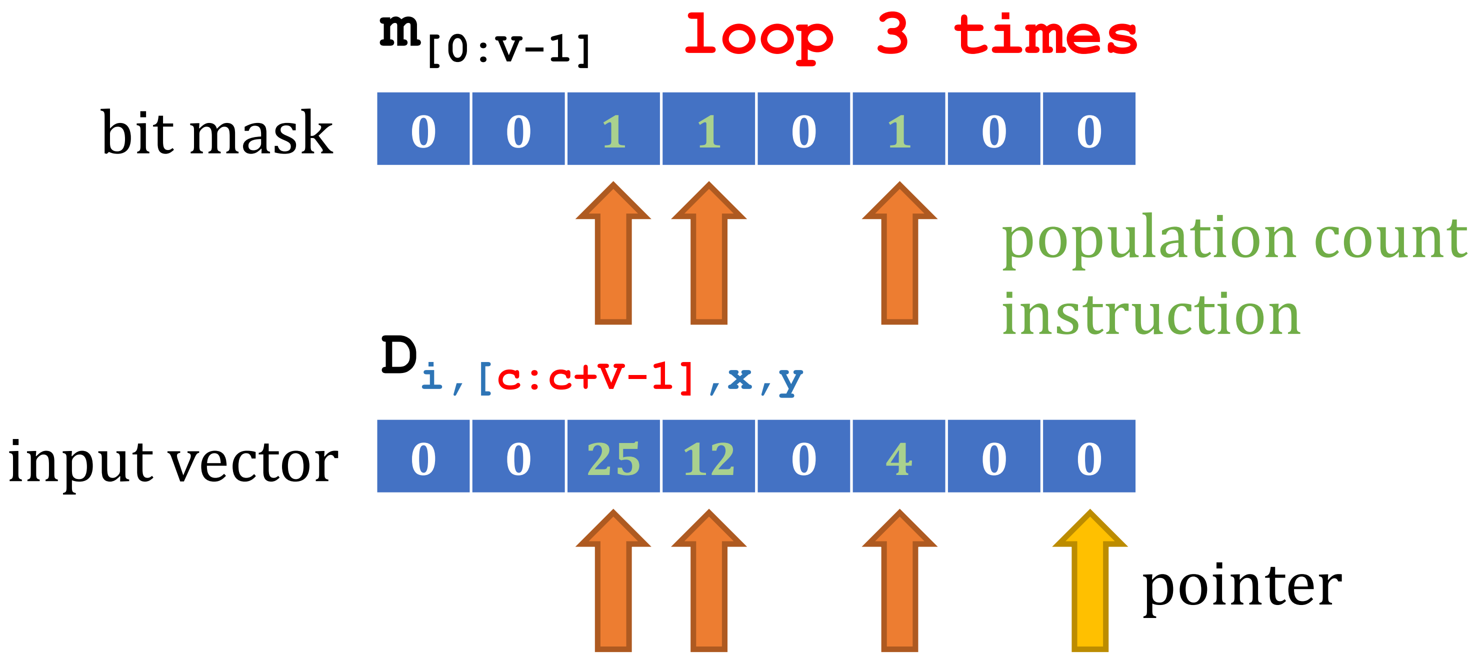


V `if` branches in total

```
for i in 0 to N-1
  for c in 0 to C-V step V
    for y in S-1 to H+S-2
      for x in R-1 to W+R-2
        m[0:V-1]=vect_cmp_neq_zero(D_{i,[c:c+V-1],x,y})
        for c′ in 0 to V-1
          if m_c′ is true
        ...
```

Optimization 3: Reducing Branch Mispredictions

- The transformation:



Optimization 3: Reducing Branch Mispredictions

- The transformation:

$\mathbf{m}_{[0:V-1]}$ **iteration 1**

bit mask	0	0	1	1	0	1	0	0

bit manipulation instruction

2 trailing zeros

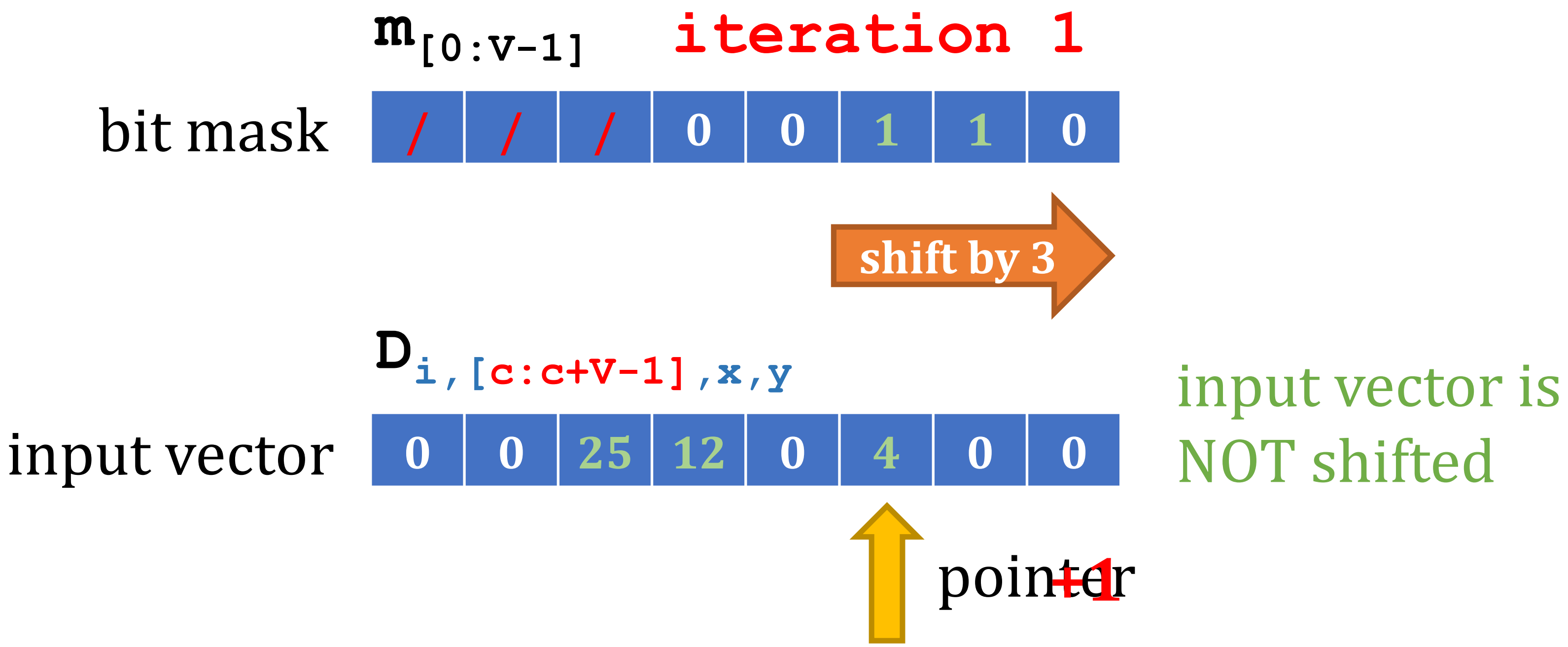
$\mathbf{D}_{i,[c:c+V-1],x,y}$

input vector	0	0	25	12	0	4	0	0

pointer +2

Optimization 3: Reducing Branch Mispredictions

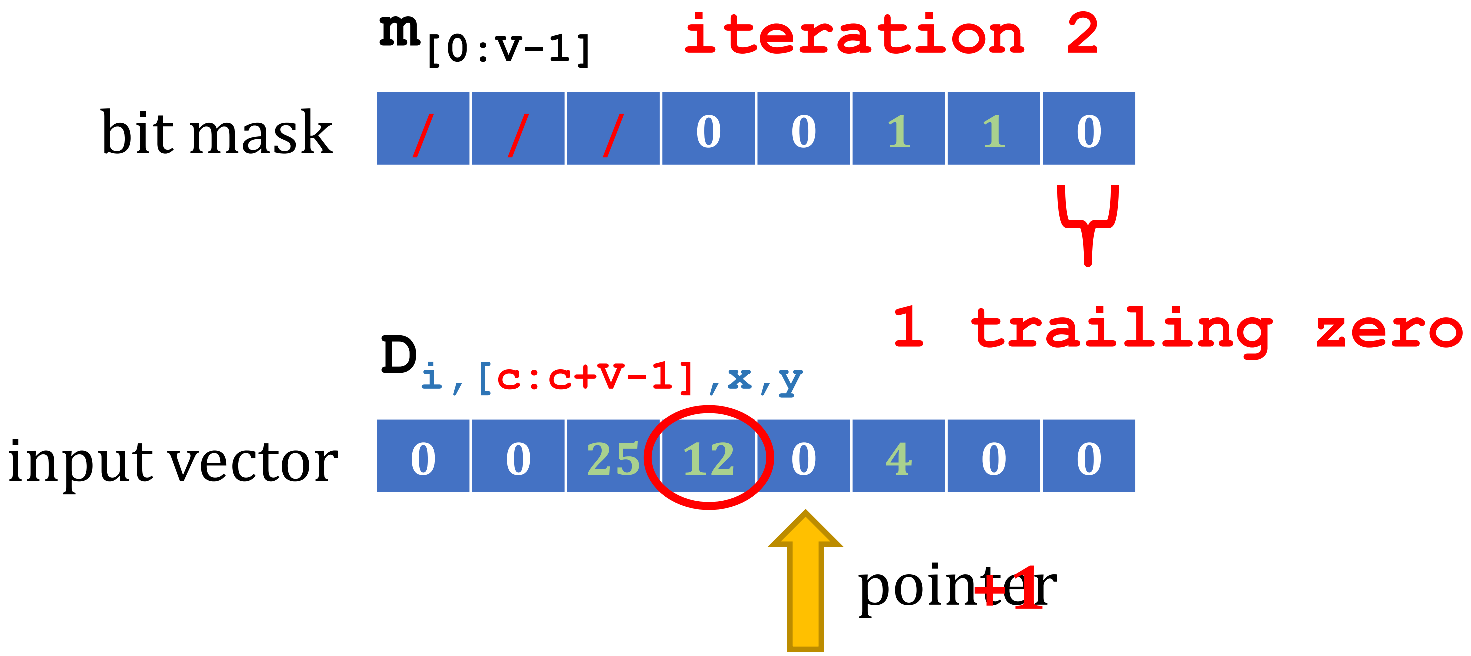
- The transformation:



ILLINOIS intel

Optimization 3: Reducing Branch Mispredictions

- The transformation:

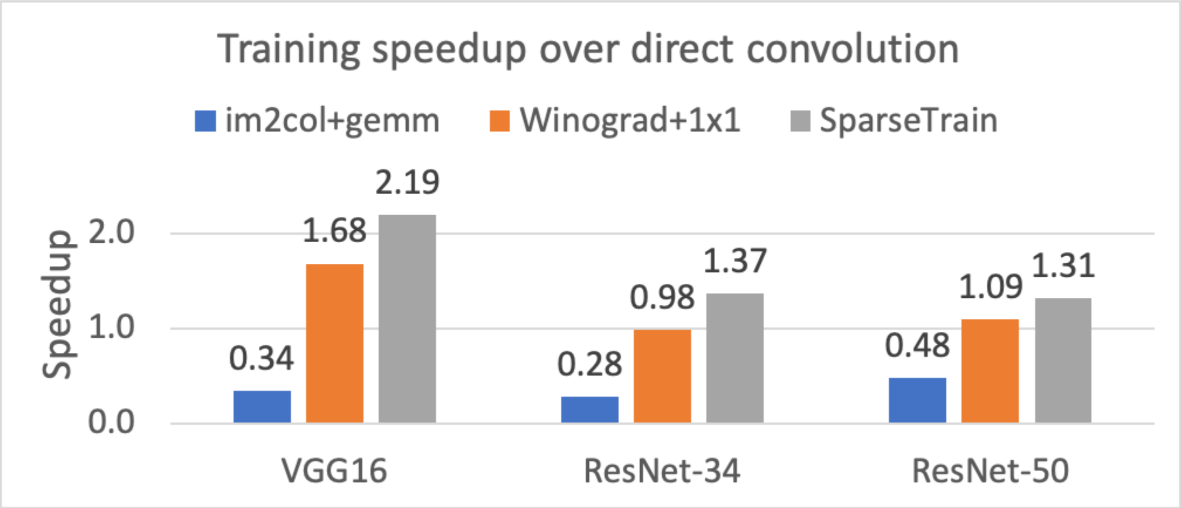


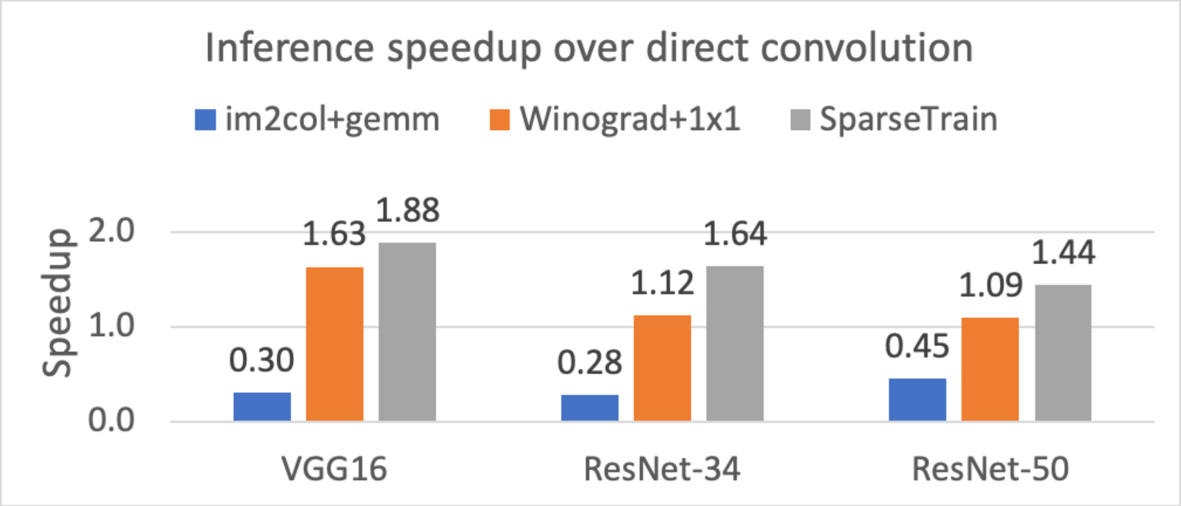
Experimental Setup

- Implemented SparseTrain in *MKL-DNN*
- Compared SparseTrain with *MKL-DNN*’s:
  - Direct convolution (baseline)
  - im2col+GEMM
  - Winograd 3x3 convolution
  - Specialized 1x1 convolution
- Experimented on 6-core Skylake-X server CPU with AVX-512
- Evaluated training/inference of VGG and ResNet

ImageNet Performance

- SparseTrain outperforms other algorithms
- SparseTrain speedup:
  - Training: 1.3x-2.2x
  - Inference: 1.4x-1.9x

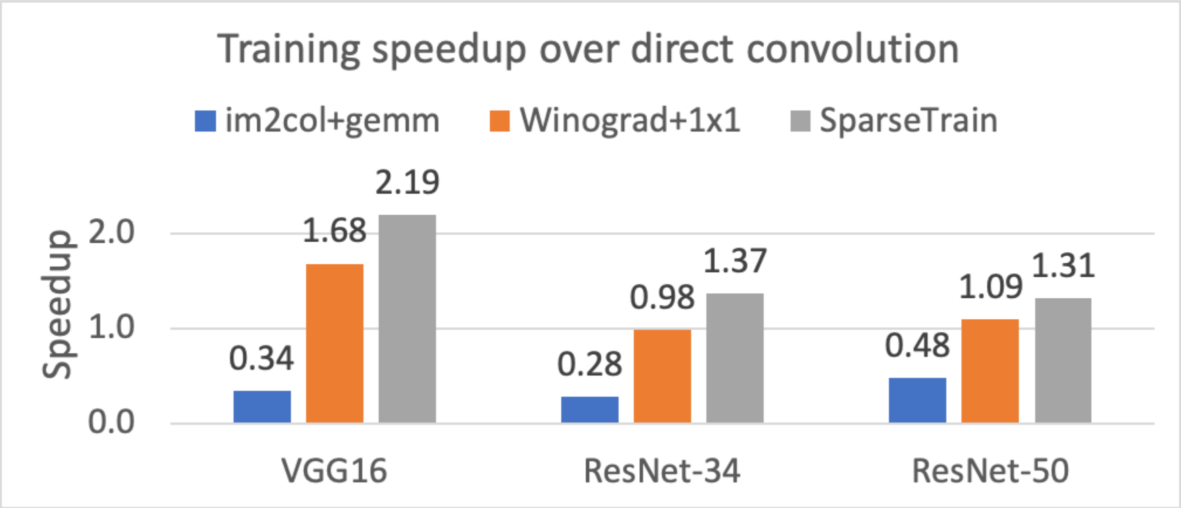


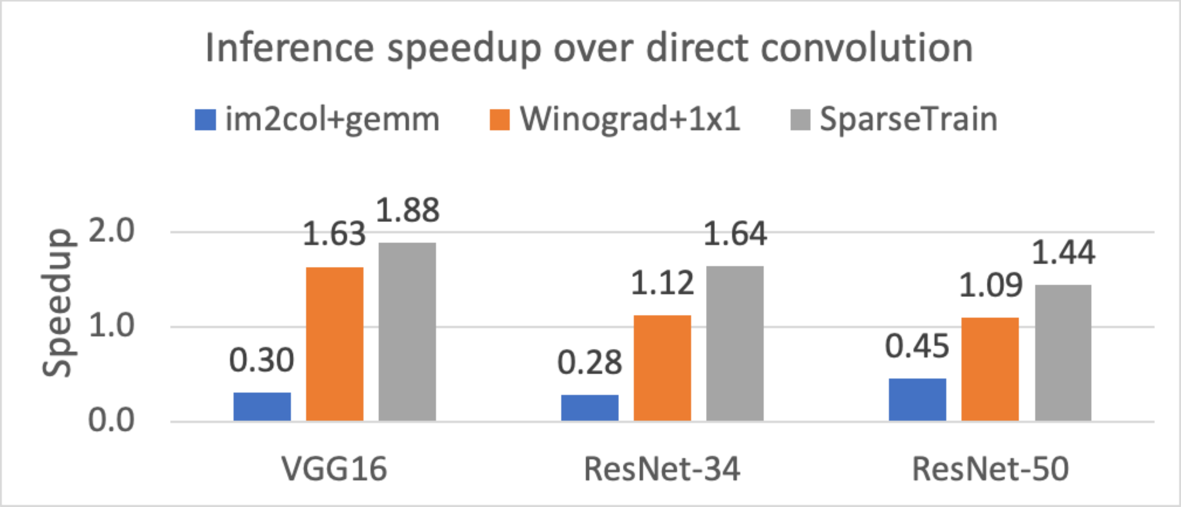


ILLINOIS intel

ImageNet Performance

- SparseTrain outperforms other algorithms
- SparseTrain speedup:
  - Training: 1.3x-2.2x
  - Inference: 1.4x-1.9x





ILLINOIS intel

Conclusion

- Presented SparseTrain:
  - First software-only algorithm to speedup DNN training by exploiting dynamic sparsity
  - Dynamically skips computation at runtime according to sparsity
  - Applicable to all three training phases
- Accelerates end-to-end training by 1.3x-2.2x and inference by 1.4x-1.9x

Thanks!

- Contact: Zhangxiaowen (Andy) Gong (gong15@Illinois.edu)
- Disclaimer: This presentation and recording belong to the authors. No distribution is allowed without the authors' permission.