国家超级计算无锡中心
National Supercomputing Center in Wuxi

swDNN: A Library for Accelerating Deep Learning Applications on Sunway TaihuLight Supercomputer

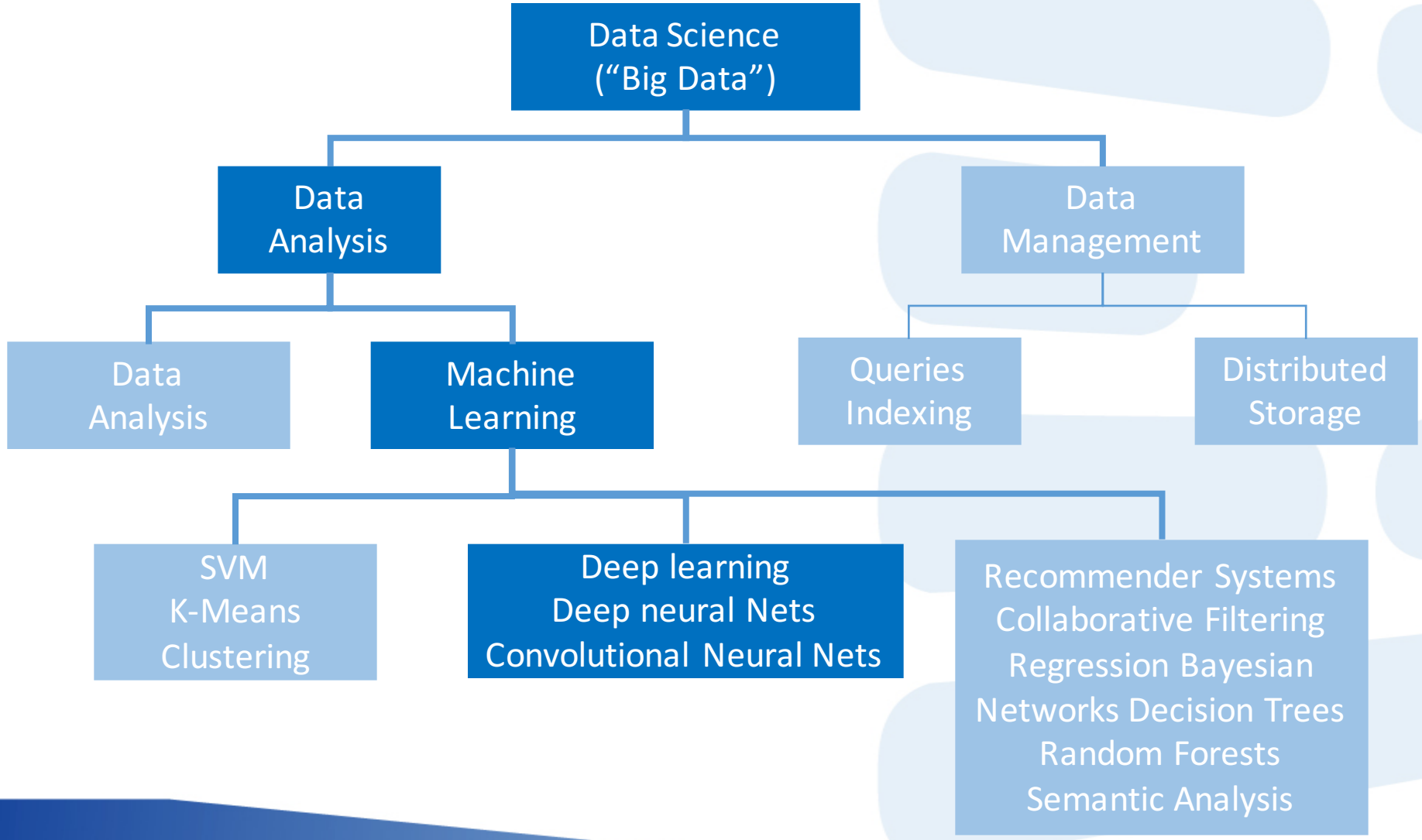
Jiarui Fang*†, *Haohuan Fu* †, Wenlai Zhao*†, Bingwei Chen*†, Weijie Zheng*†, Guangwen Yang*†*

†Tsinghua University

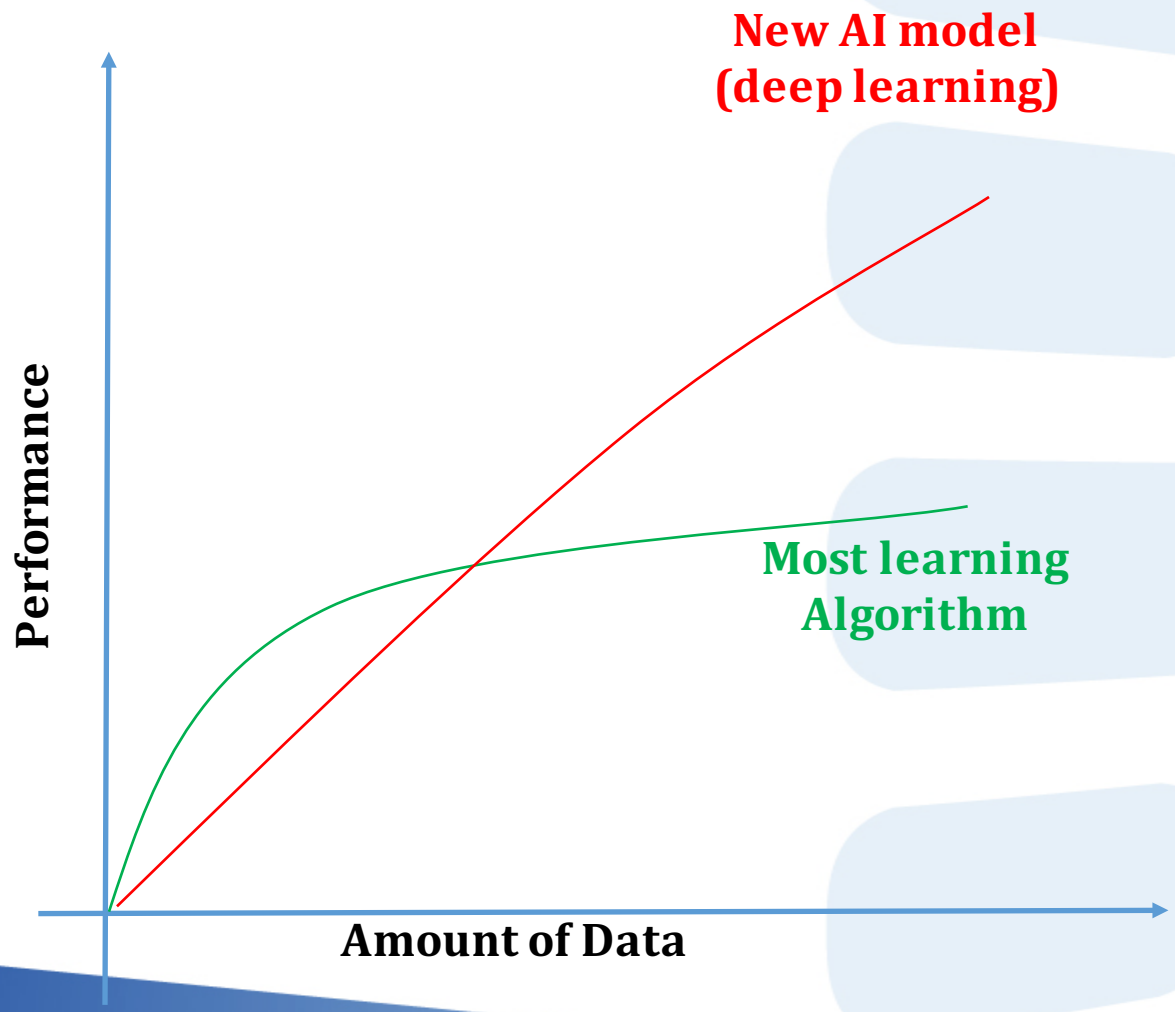
*National Supercomputing Center in Wuxi

2017. 05. 31@Orlando IPDPS17

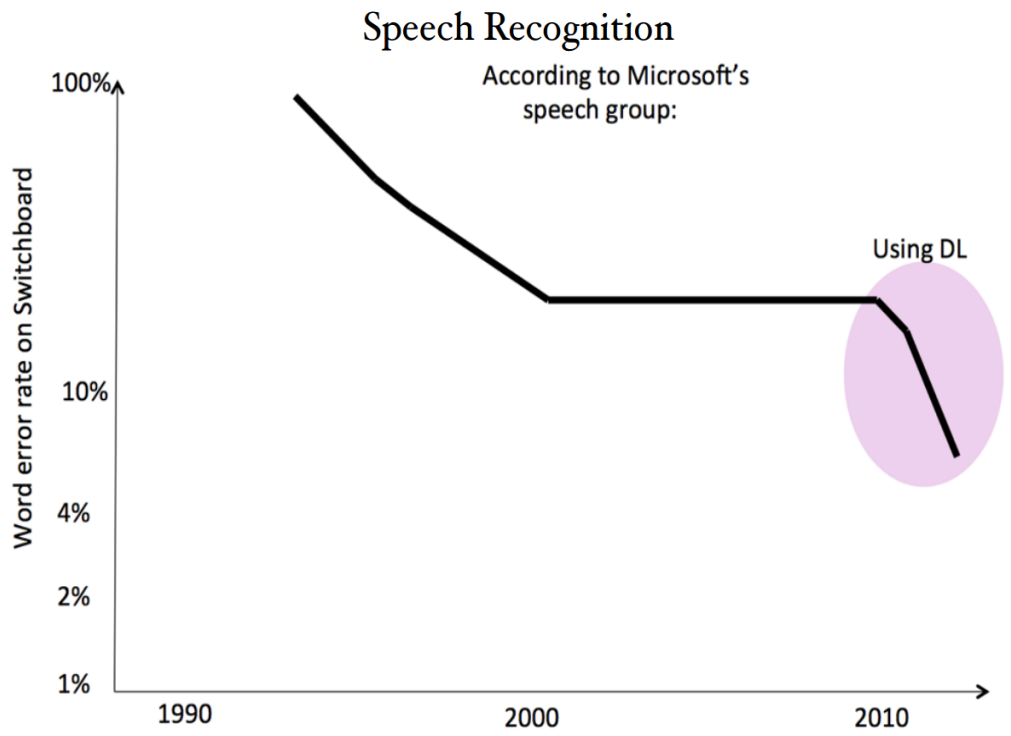
Deep Leaning from a Big Picture



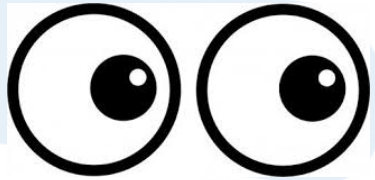
Deep Leaning vs Traditional Machine Learning

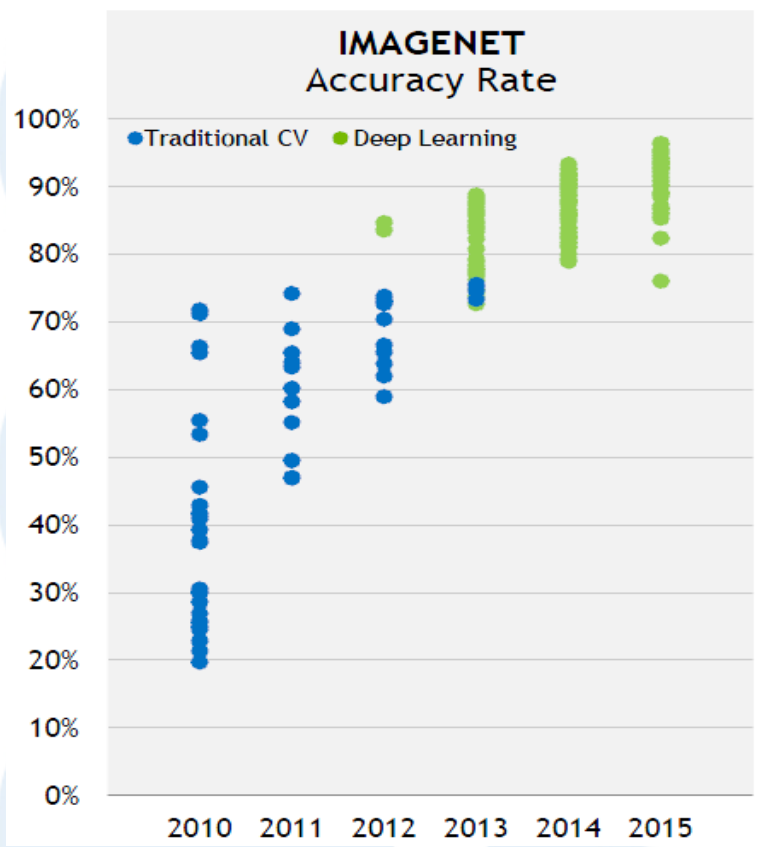


What is Deep Leaning good at?



From Andrew Ng

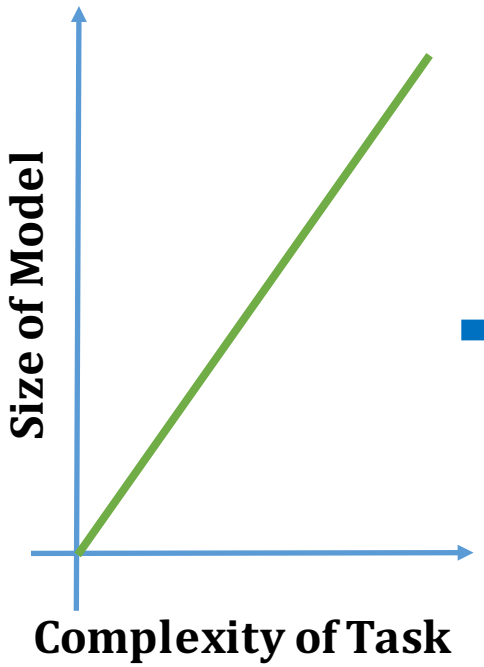


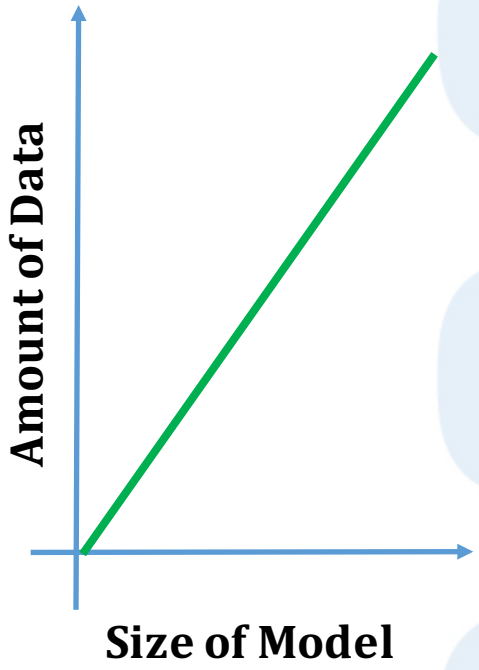


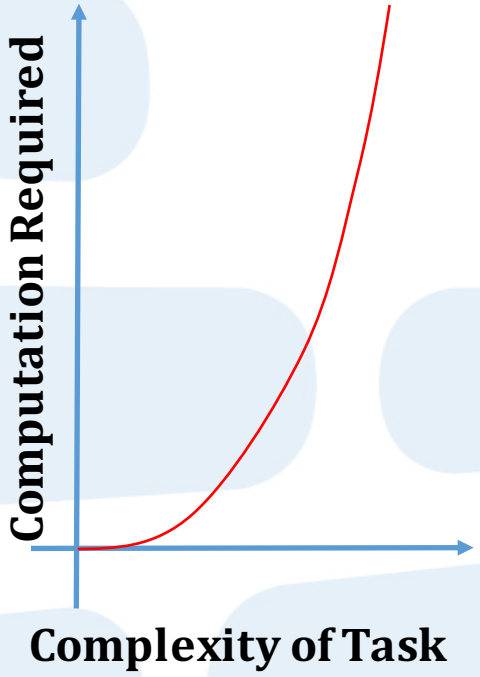
From NVIDIA

Deep Learning driven by scale

Better Performance = **Big Model** + **More Data** → **More Computing Power!**

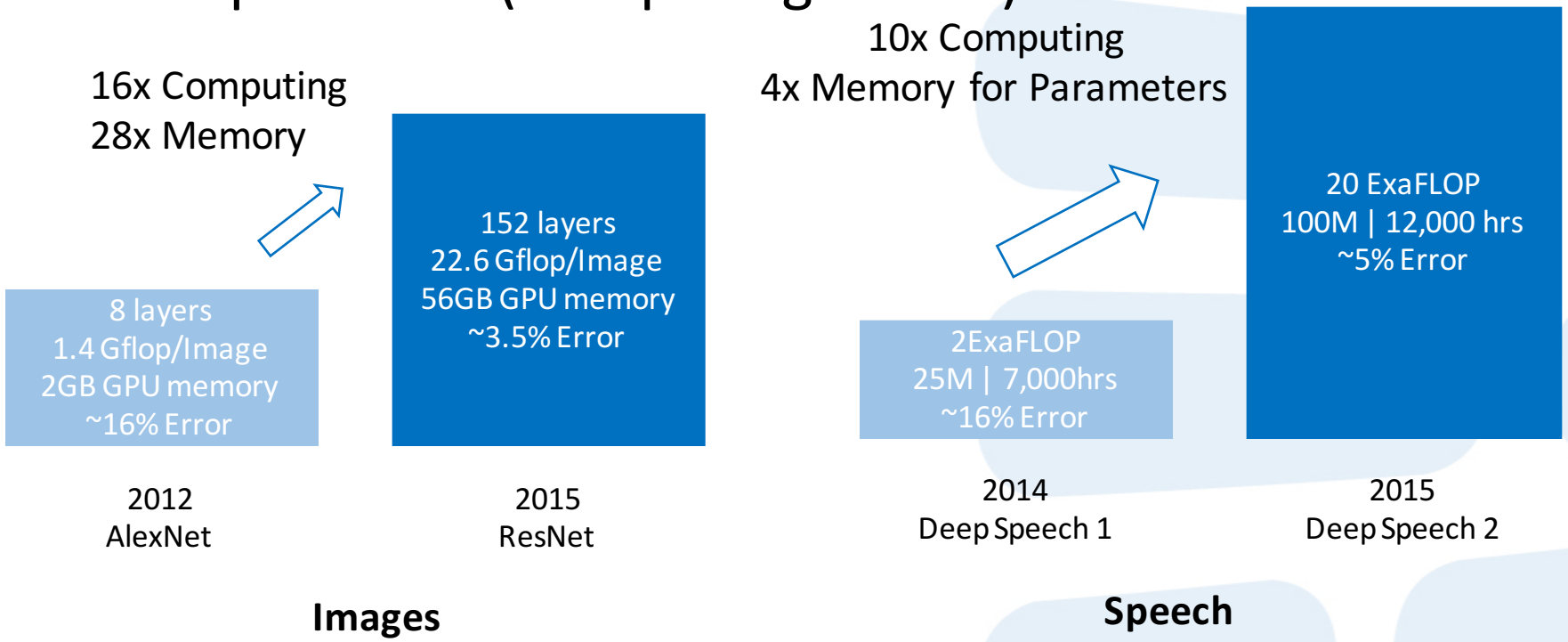






Deep Learning driven by scale

- Large Models Parameters and Data (Memory Space)
- More Float Point Operations (Computing Power)



Computing Capacity Tflops → Pflops

High Performance Deep Learning

Big data + Deep learning + High performance computing = Intelligence

GTC'14: Deep Learning Meets Heterogeneous Computing

Pioneers adopting hpc for deep learning

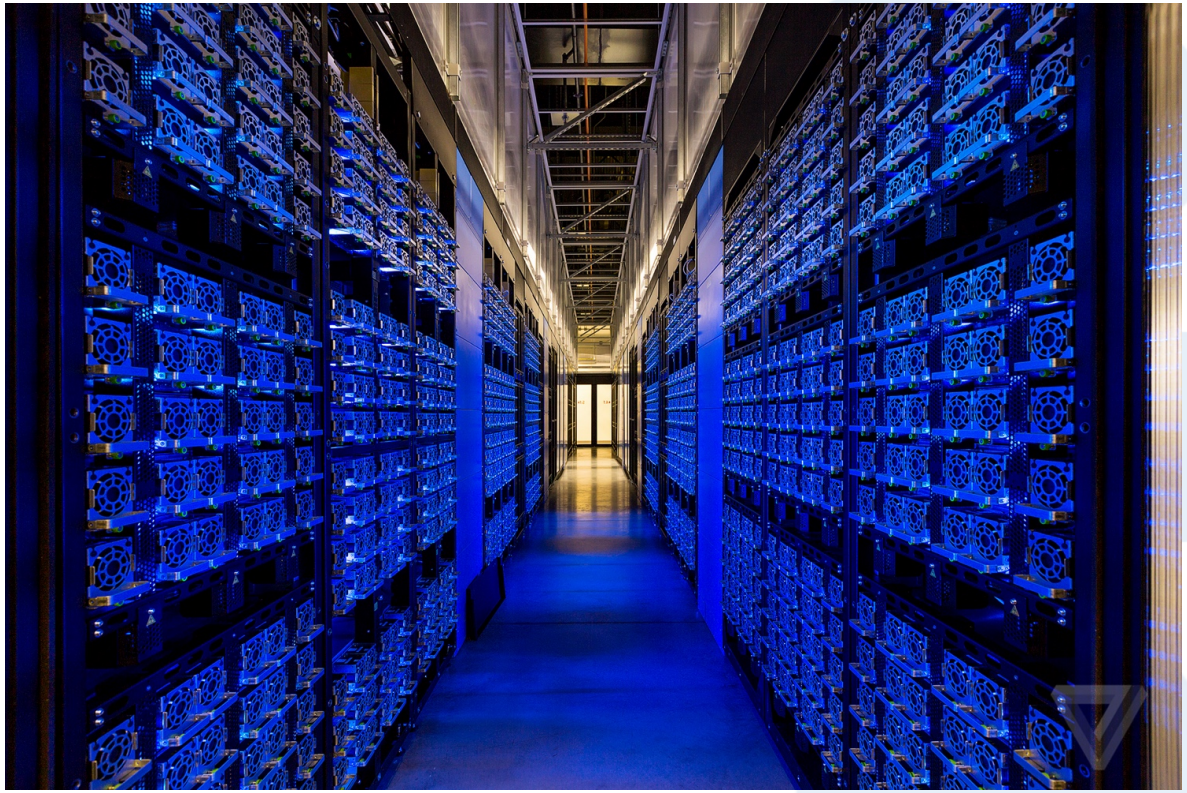
Dr. Andrew Ng, Chief Scientist, Baidu

"Investments in computer systems — and I think the bleeding-edge of AI, and deep learning specifically, is shifting to HPC — can cut down the time to run an experiment from a week to a day and sometimes even faster."

High Performance Deep Learning

Big Sur : Facebook's Supercomputer for Deep learning

40Pflops (single-precision)

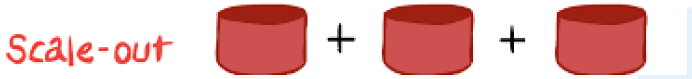


http://www.theverge.com/

Towards High Performance Deep Learning

- **Scale up**： leveraging hardware power inside a single machine

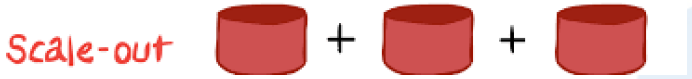
- **Scale out**： using multiple machines/nodes in a large cluster to increase the available computing power



Towards High Performance Deep Learning

- **Scale up**： leveraging hardware power inside a single machine .

- **Scale out**： using multiple machines/nodes in a large cluster to increase the available computing power



The Sunway TaihuLight Supercomputer

SW26010 Many-core processor

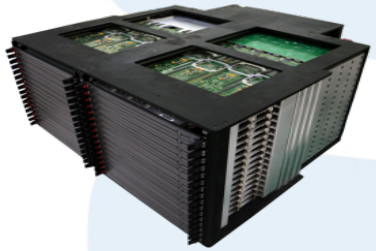
8x

Computing Node

4x

Computing Plugin

8x



Super Node

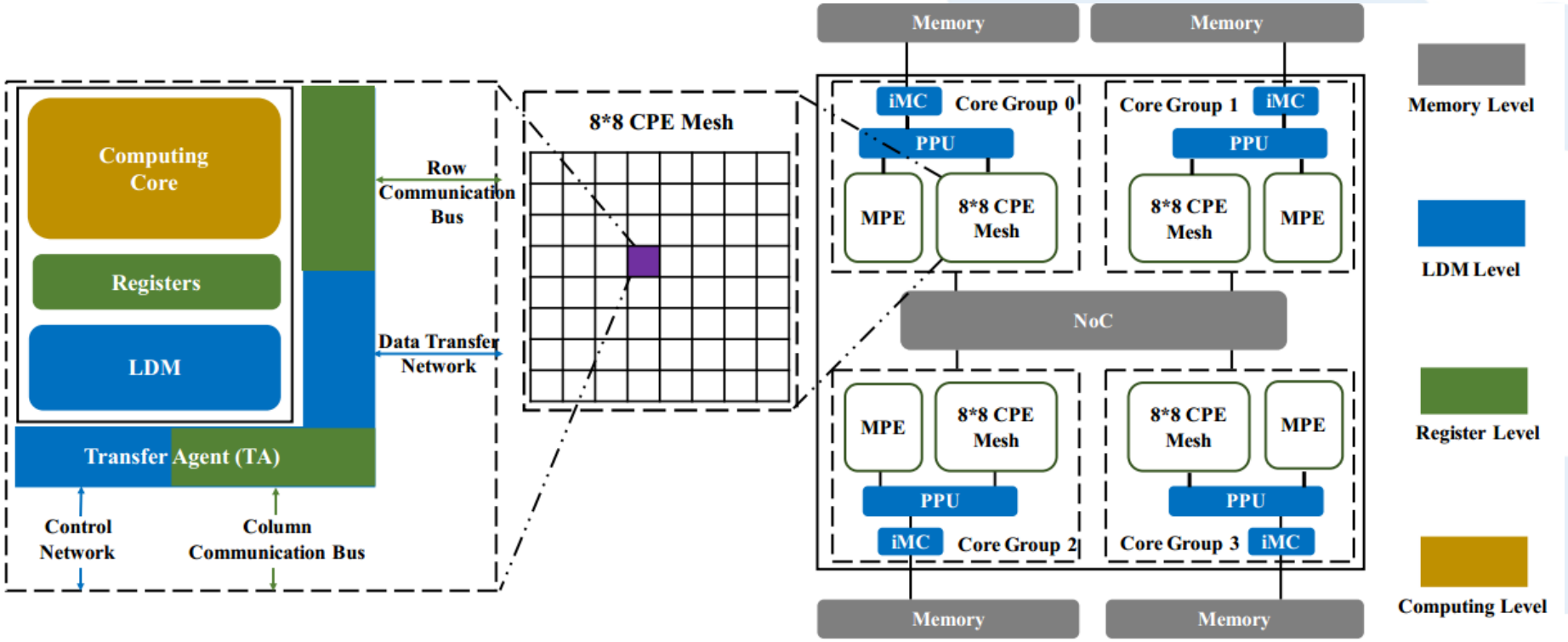
160x

SuperComputer

Perk Performance	:	125 PFLOPS
LINPACK Performance	:	93 PFLOPS
Performance per Watt	:	6.05 GFLOPS/W
Clock frequency of CPU	:	1.45GHz
Peak Performance of a CPU	:	3.06 TFLOPS
Total capacity	:	1024TB

Total Bandwidth	:	4473.16TB
Network Link bandwidth	:	14GB/S
Network Bisection bandwidth	:	56TB/S
Storage	:	20PB
Total I/O bandwidth	:	288GB/s
LINPACK Power	:	12.5MW

Architecture of SW26010

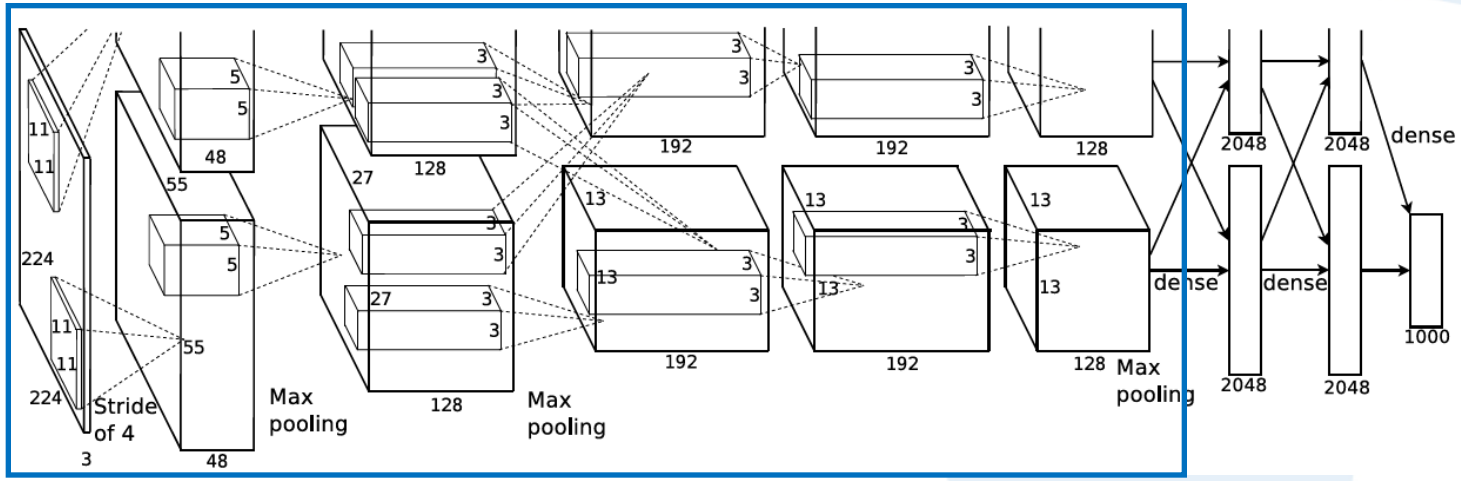


	Execution Pipelines	SIMD width	Clock	Data-Cache Pre CPE	Memory Bandwidth
CPE	2	256bit	1.45GHz	64KB LDM	136 GB/s
MPE	2	256bit	1.45GHz	256KB L2 + 32KB L1	

swDNN：A Library for Deep Learning

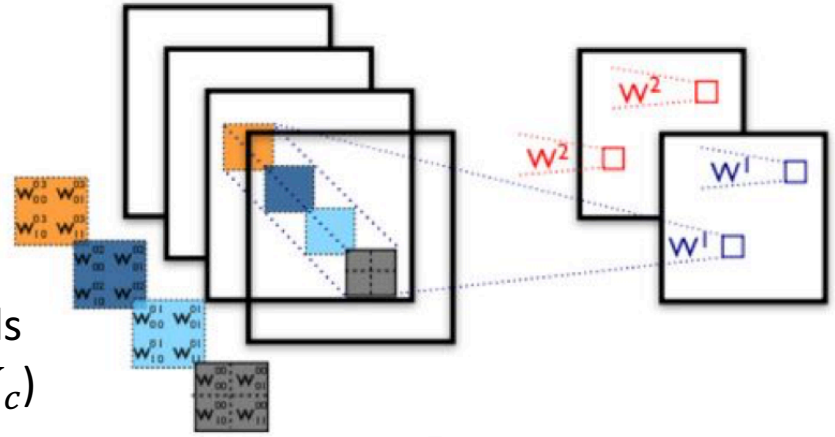
- **swDNN provides highly tuned implementations for standard routines for neuron layers of Deep Neural Networks**
  - BLAS (can accelerate most of layers)
  - ***Convolutional Layer** (occupy over 90% time in CNN)

Convolutional Layers



Input data
(B, N_i, C_i, R_i)

Output data
(B, N_o, C_o, R_o)



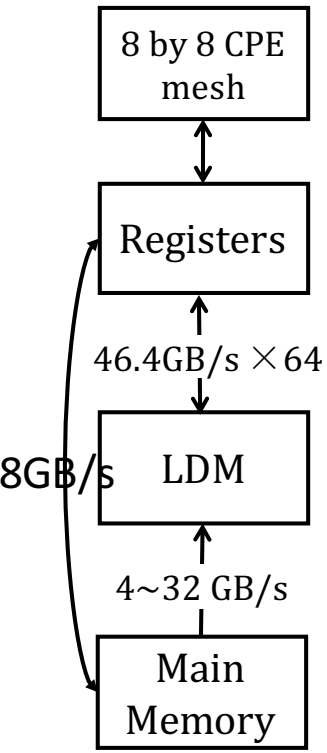
Filter Kernels
(N_o, N_i, K_r, K_c)

```
for(cB = 0; cB < B; ++cB)
 for(cCo = 0; cCo < Co; ++cCo)
  for(cRo = 0; cRo < Ro; ++cRo)
   for(cNi=0; cNi < Ni; ++cNi)
    for(cNo = 0; cNo < No; ++cNo)
     for(cKr = 0; cKr < Kr; ++cKr)
      for(cKc = 0; cKc < Kc; ++cKc)
       out[cRo][cCo][cNo][cB] +=
        in[cRo+cKr][cCo+cKc][cNo][cB]
       *filter[cKc][cKr][cCo][cRo];
```

swDNN：A Library for Deep Learning

- **swDNN provides highly tuned implementations for standard routines for neuron layers of Deep Neural Networks**
  - BLAS (can accelerate most of layers)
  - ***Convolutional Layer** (occupy 80%~90% time in CNN)
- **Challenges for parallel convolutional layer design on SW26010**
  - The relatively low memory bandwidth.
    - *The DDR3 memory interface provides a peak bandwidth of **144GB/s** (36 GB/s per CG). While the NVIDIA K80 GPU provides a bandwidth of **480 GB/s**.*
  - The algorithm of Convolutional Layer involves all-to-all connections between inputs, filter kernels, and outputs.
    - *SW26010, the CPEs do not have a shared buffer for such frequent data communications. While in NVIDIA K80, L1 cache can be shared in SPs in the same SMX and L2 cache can be shared by all SPs.*

A Performance Model for SW26010



Direct Memory Access

Peak Performance per CG

$Perf = 742.4\ Gflops$

Considering Execution Efficiency (EE)

$Perf = 742.4\ Gflops \cdot EE$

$$RBW_{MEM\to REG} = \frac{8Byte}{2flop/8\times 1.45GHz} = 139.2\ GBps$$

Considering Direct Memory Access

$$Perf = 742.4\ Gflops \cdot EE \cdot f\left(min\left(1, \frac{8}{RBW_{MEM\to REG}}\right)\right)$$

REG-LDM-MEM

Peak Performance per CG

$Perf = 742.4Gflops$

Considering Execution Efficiency (EE)

$Perf = 742.4\ Gflops \cdot EE$

Considering LDM – REG Exchange

$$Perf = 742.4Gflops \cdot EE \cdot f\left(min\left(1, \frac{46.4 \cdot 64}{RBW_{LDM\to REG}}\right)\right)$$

Considering MEM – LDM Exchange

$$Perf = 42.4Gflops \cdot EE \cdot f\left(min\left(1, \frac{46.4 \cdot 64}{RBW_{LDM\to REG}}\right)\right) \cdot f\left(min\left(1, \frac{MBW_{MEM\to LDM}}{RBW_{MEM\to LDM}}\right)\right)$$

Measured Bandwidth (MBW)

Required Bandwidth (RBW)$= \dfrac{Amount\ of\ Data\ Access}{time\ of\ Data\ Caculation\ with\ no\ Perforamance\ Loss}$

$$\frac{RBW}{MBW} = \frac{\frac{Amount\ of\ Data\ Access}{MBW}}{time\ of\ Data\ Caculation\ with\ no\ Perf\ Loss}$$

$$\frac{MBW}{RBW} = \frac{time\ of\ Data\ Caculation\ with\ no\ Perf\ Loss}{time\ of\ Data\ Access}$$

$$\boldsymbol{Discount} = \frac{perf\ with\ loss}{perf\ no\ loss} = f\left(\min\left(1, \frac{MBW}{RBW}\right)\right)$$

$f(\cdot)$ monotone increasing function with $f(1) = 1$

LDM-related optimizations

- ***LDM blocking : loop splitting and loop scheduling***
  - To decrease $RBW_{MEM \rightarrow LDM}$, we should reuse the data fetched by DMA operations as much as possible.
  - To increase $MBW_{\mathrm{MEM} \rightarrow LDM}$, we should increase leading dimensions of data accessed by DMA.

$Size(Byte)$	Get	Put	$Size(Byte)$	Get	Put
32	4.31	2.56	512	27.42	30.34
64	9.00	9.20	576	25.96	28.91
128	17.25	18.83	640	29.05	32.00
192	17.94	19.82	1024	29.79	33.44
256	22.44	25.80	2048	31.32	35.19
384	22.88	24.67	4096	32.05	36.01

Table I. Measured DMA Bandwidths (GBps) of 1 CG is affected by the the size of continuous memory access blocks of one CPE

LDM-related optimizations

	blocking dimensions	leading Dim. of DMA
Input data	C_o, B, N_i	$b_{C_o} \times b_B$
Filter kernels	N_i, N_o	N_i
Output data	C_o, B, N_o	$b_{C_o} \times b_B$

Inner Data Access

Input elements $N_i \times bCo \times b_B$

Filter elements $N_i \times N_o$

Amount of calculation

$2N_i \times N_o \times bCo \times b_B$ flop

Algorithm 1 Image Size Aware Version

```
for b_B Start = 0 : b_B : B do
    for Ro_Start = 0 : Ro do
        for Co_Start = 0 : bCo : Co do
            for cKr = 0 : K_r do
                for cKc = 0 : K_c do
                    DMA get D_i ← N_i × b_B channels input images (Co_Start +
                    cKc : Co_Start + cKc + bCo, Ro_Start + cKr) start at
                    b_B Start.  (Co_start: Co_start + K_c + bC_o, Ro_start + cKr)
                    DMA get W ← N_i × N_o channels filter kernels (cKc, cKr)
                    start at b_B Start
                    D_o += D_i × W
                end for
            end for
            DMA put b_B × N_o channels output images(Co_Start : Co_Start +
            bCo, Ro_Start) ← D_o
        end for
    end for
end for
```

$$RBW_{Mem \to LDM} = \frac{(N_o + b_{C_o} b_B) DataSize}{\frac{2 b_{C_o} b_B N_o}{perf\ no\ loss}} = \alpha \left(\frac{1}{b_{C_o} b_B} + \frac{1}{N_o} \right)$$

LDM-related optimizations

	Blocking dimensions	Leading Dim. of DMA
Input data	B, N_i	B
Filter kernels	N_i, N_o	N_i
Output data	C_o, B, N_o	B

Inner Data Access

Input elements	$N_i \times B$
Filter elements	$N_i \times N_o$

Amount of Calculation

$2N_i \times N_o \times B$ flop

Algorithm 2 Batch Size Aware Version

```
for Co_start = 0 : b_Co : C_o − 1 do
    for cR_o = 0 : R_o − 1 do
        for cK_r = 0 : K_r − 1 do
            cR_i = cR_o + cK_r
            for cC_i = Co_start : Co_start + b_Co + K_c − 1 do
                DMA get D_i ← N_i × B channels of input images(cC_i, cR_i)
                for cKc = 0 : K_c − 1 do
                    DMA get W ← N_i × N_o channels of filter kernels (cK_c, cK_r)
                    cC_o = cC_i − cK_c
                    if cC_o >= Co_start and cC_o < Co_start + K_c then
                        D_o(cC_o)+ = W * D_i
                    end if
                end for
            end for
        end for
        DMA put N_i × B channels of output images (Co_start : Co_start + b_Co, cR_o) ← D_o
    end for
end for
```

$\times K_c$

$(:, cK_r)$

$$RBW_{Mem \to LDM} = \frac{(B + N_o)DataSize}{\frac{2BN_o}{perf\ no\ loss}} = \alpha\left(\frac{1}{N_o} + \frac{1}{B}\right)$$

Register-Related Optimization

- **Coordinate GEMM (General Matrix Matrix Multiplication) operation on 64 CPEs**
  - How to distributed the data fetched from main memory onto LDM of 64 CPEs?
  - How to share data between 64 CPEs when computing?
  - How to enable $RBW_{LDM \rightarrow REG}$< 46.4GB/s?

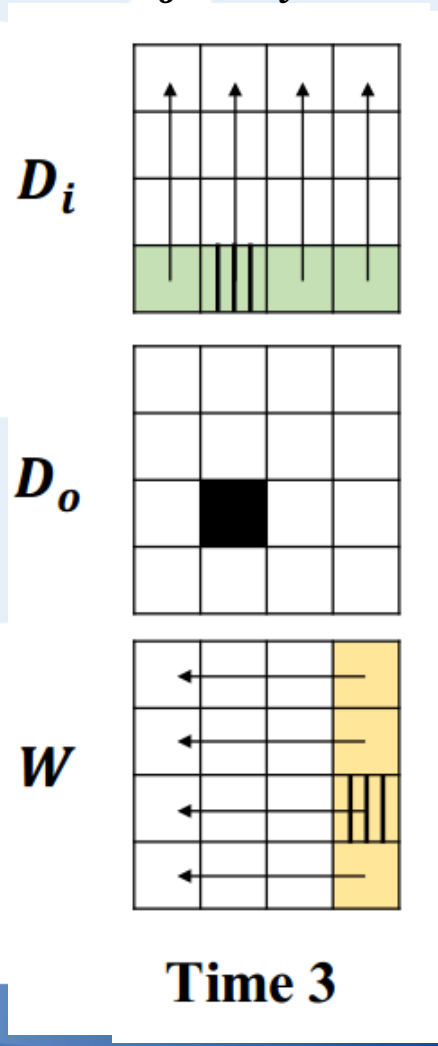
- **Solution**
  - Register Communication
  - Register Blocking + Vectorization

Register-Related Optimization – Register Communication

- **Data Layout**
  - Divide matrices structure in LDM along row and column into 8 parts respectively .i.e., each CPE maintains 1/8×1/8 data.
- **Data Sharing with Register Communications***
  - enables P2P/broadcast 256-bit data communications at the register level
  - each CPE can communication with other CPEs in the same row and column
  - follows an anonymous *producer-consumer* pattern with FIFO sending/receiving buffers with a latency of 10 or 11 cycles.

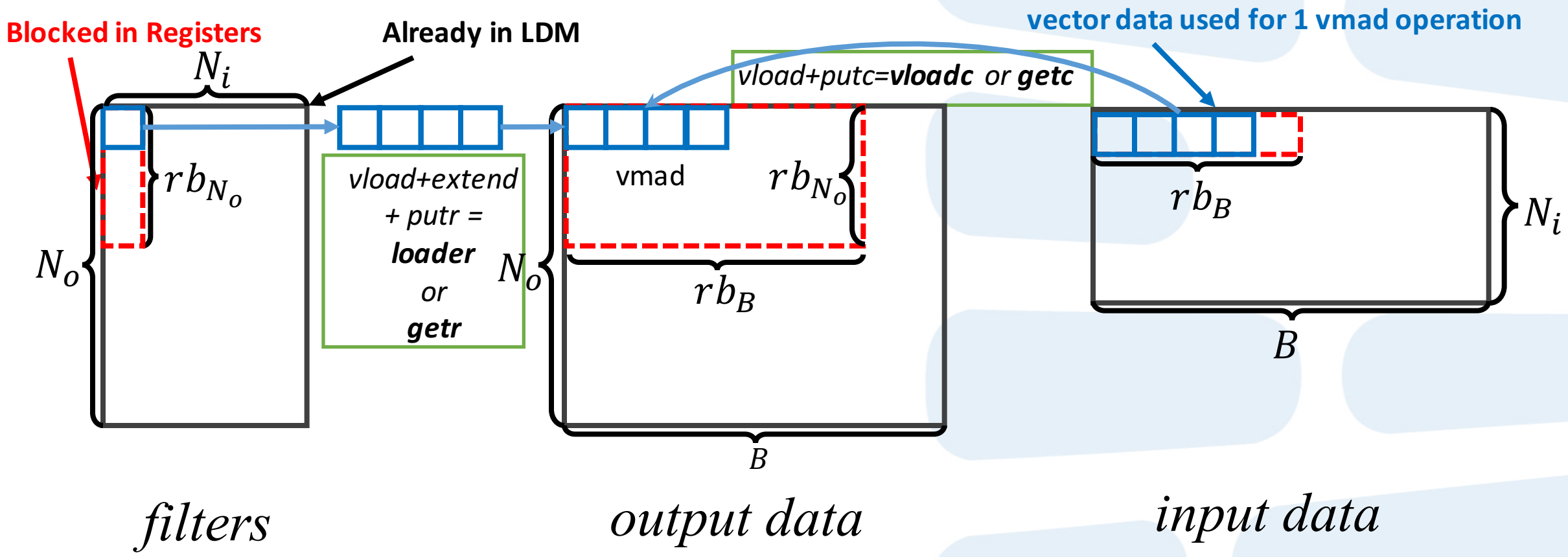
A 4x4 CPE mesh demo

$D_o = D_i \times W$



* Zhigeng Xu, at el. Benchmarking SW26010 Many-core Processor

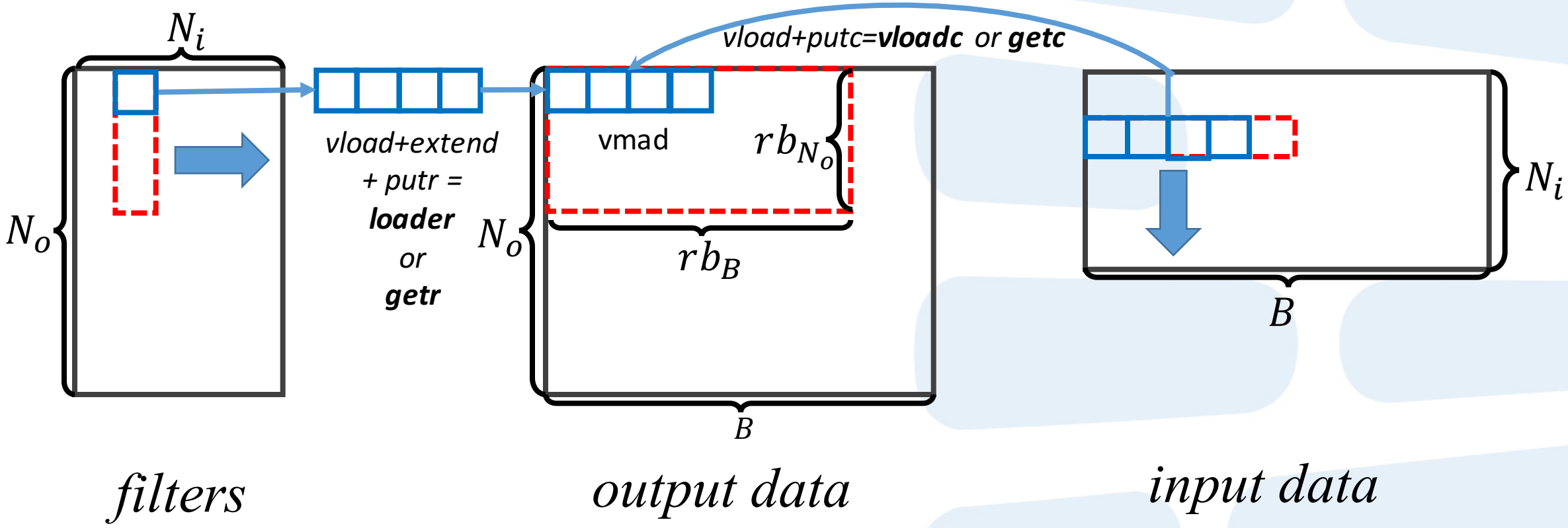
Register-Related Optimization – Register Blocking + Vectorization



Block filters data in column and input data in row, Update a submatrix of output data

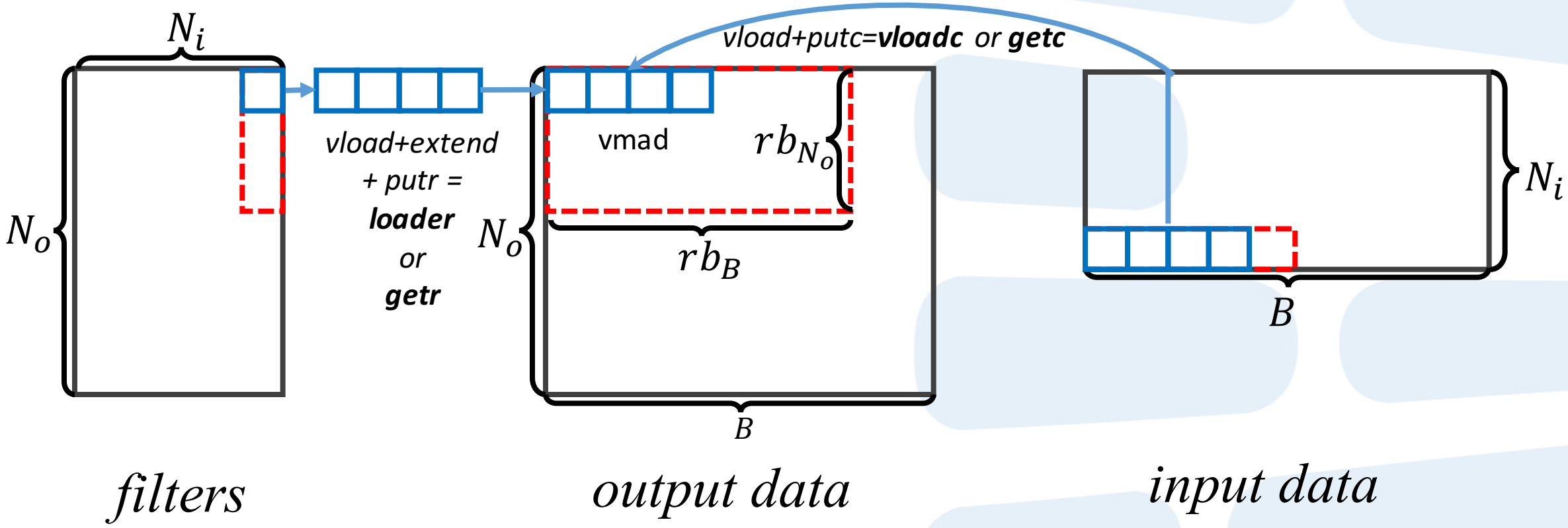
Register-Related Optimization – Register Blocking + Vectorization

block next filters data and input data registers, Update the same output data



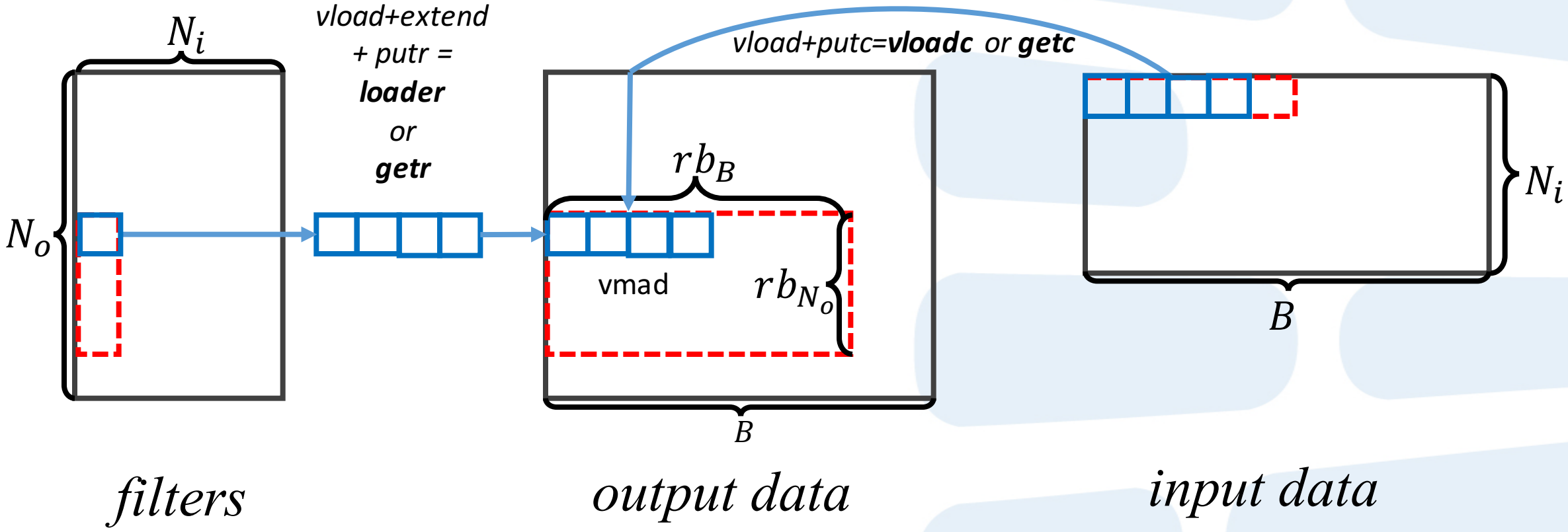
Register-Related Optimization – Register Blocking + Vectorization

Finish updating the a block of output data, store it into LDM

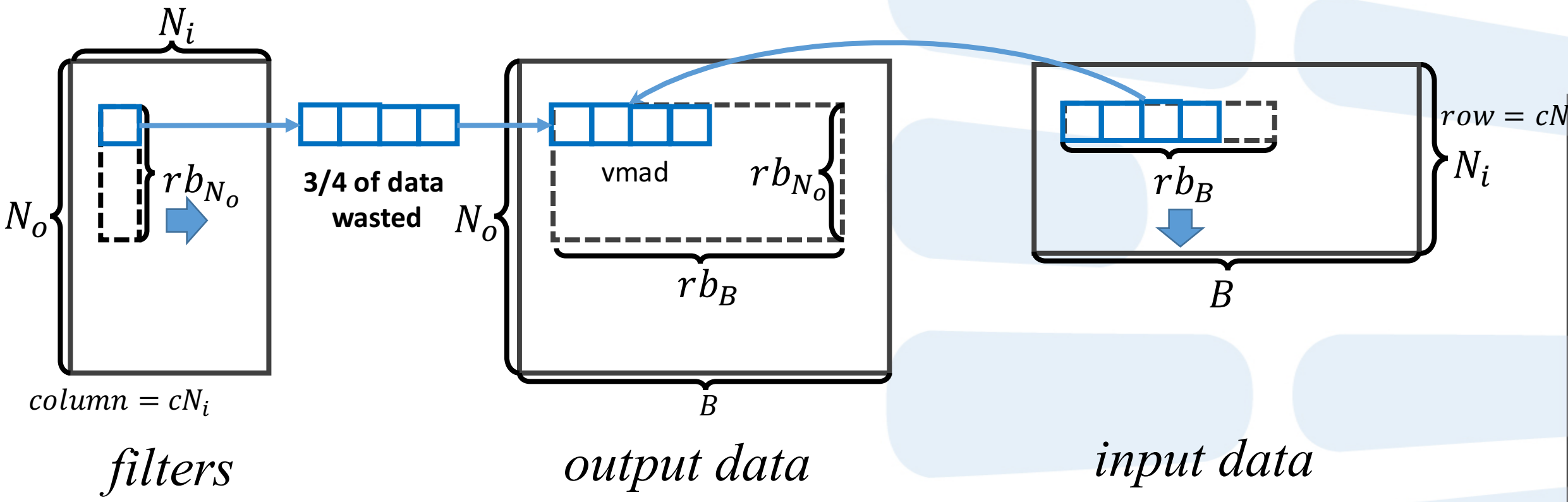


Register-Related Optimization – Register Blocking + Vectorization

Begin update next block of output data



Register-Related Optimization – Register Blocking + Vectorization



$$RBW_{LDM\to REG} = \frac{(rb_B + 4rb_{N_o})DataSize}{\frac{2rb_B rb_{N_o}}{Perf\ o\ loss}}$$

$$rb_{N_o} = rb_B = 4$$

$$RBW_{LDM\to REG} < \frac{(4 + 4\times4)\times8Byte}{2\times4\times4/(1.45GHz\times8flop)} = 23.2GB/s < 46.4GB/s$$

Computing-Unit-Related Optimization

Principles :

- **Reduce Read After Write (RAW) Hazard :**

  postpone issuing of dependent instructions

- **Increase Instruction Level Parallelism:**

  pairing loads/stores with flops to maximize dual-issue

EE = 16/26 = **61.5%**

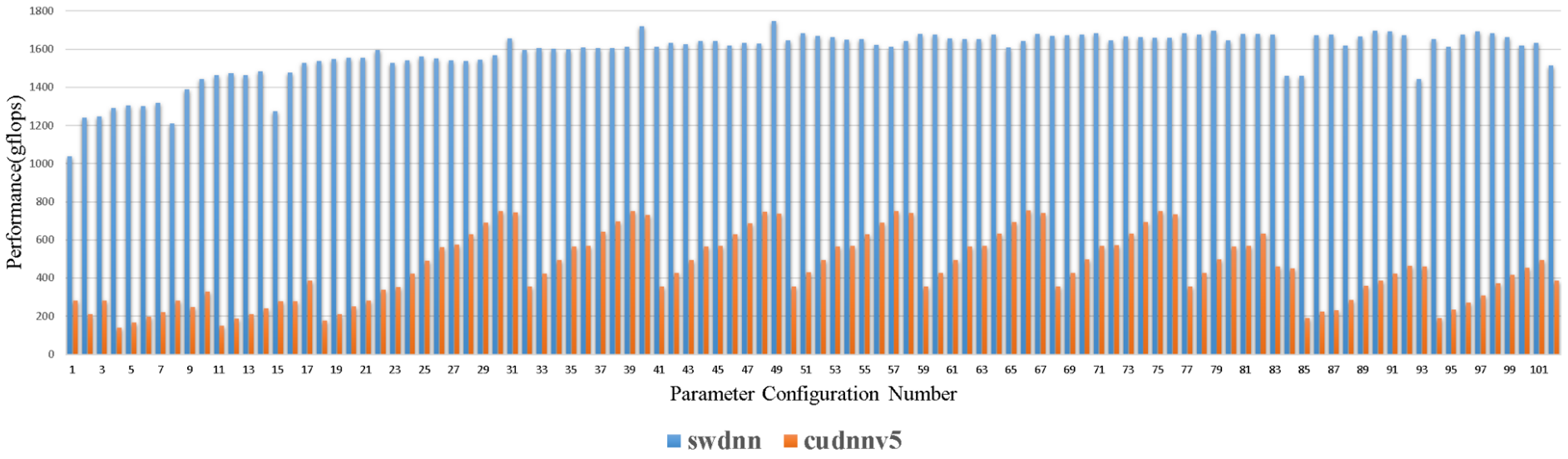
```
InLoop:
1  vldr(getr) A[0],ptrA,0
2  vldr(getr) A[1],ptrA,4
3  vldr(getr) A[2],ptrA,8
4  add ptrA, offsetA, ptrA
    vldr(getr) A[3],ptrA,12
5  vlddec(getc)   B[0],ptrB,0
6  vlddec(getc)   B[1],ptrB,4
7  vlddec(getc)   B[2],ptrB,8
8  add ptrB, offsetB, ptrB
    vlddec(getc) B[3],ptrB,12
9  vfmadd A[0], B[0], C[0]
10 vfmadd A[1], B[0], C[1]
11 vfmadd A[2], B[0], C[2]
12 vfmadd A[3], B[0], C[3]
13 vfmadd A[0], B[1], C[4]
14 vfmadd A[1], B[1], C[5]
15 vfmadd A[2], B[1], C[6]
16 vfmadd A[3], B[1], C[7]
17 vfmadd A[0], B[2], C[8]
18 vfmadd A[1], B[2], C[9]
19 vfmadd A[2], B[2], C[10]
20 vfmadd A[3], B[2], C[11]
21 vfmadd A[0], B[3], C[12]
22 vfmadd A[1], B[3], C[13]
23 vfmadd A[2], B[3], C[14]
     cmp cNi, (Ni/8-1)
24 vfmadd A[3], B[3], C[15]
     add cNi, 1, cNi
25 cmp cNi, Ni
26 bne InLoop
```

EE = 16/17 = **94.1%**

```
InLoop:
1  vfmadd A[0], B[0], C[0]
     vlddec(getc) B[1],ptrB,4
2  vfmadd A[1], B[0], C[1]
     vlddec(getc) B[2],ptrB,8
3  vfmadd A[2], B[0], C[2]
     vlddec(getc) B[3],ptrB,12
4  vfmadd A[3], B[0], C[3]
     add ptrB, offsetB, ptrB
5  vfmadd A[0], B[1], C[4]
     add cNi, 1, cNi
6  vfmadd A[1], B[1], C[5]
     cmp cNi, (Ni/8-1)
7  vfmadd A[2], B[1], C[6]
8  vfmadd A[3], B[1], C[7]
9  vfmadd A[0], B[2], C[8]
10 vfmadd A[1], B[2], C[9]
11 vfmadd A[2], B[2], C[10]
12 vfmadd A[3], B[2], C[11]
     vlddec(getc) B[0],ptrB,0
13 vfmadd A[0], B[3], C[12]
     vldr(getr) A[0],ptrA,0
14 vfmadd A[1], B[3], C[13]
     vldr(getr) A[1],ptrA,4
15 vfmadd A[2], B[3], C[14]
     vldr(getr) A[2],ptrA,8
16 vfmadd A[3], B[3], C[15]
     vldr(getr) A[3],ptrA,12
17 add ptrA, offsetA, ptrA
     bne InLoop
```

Performance Evaluation of Convolutional Layers

Convolution performance is around 1.6 Tflops in double-precision floating-point

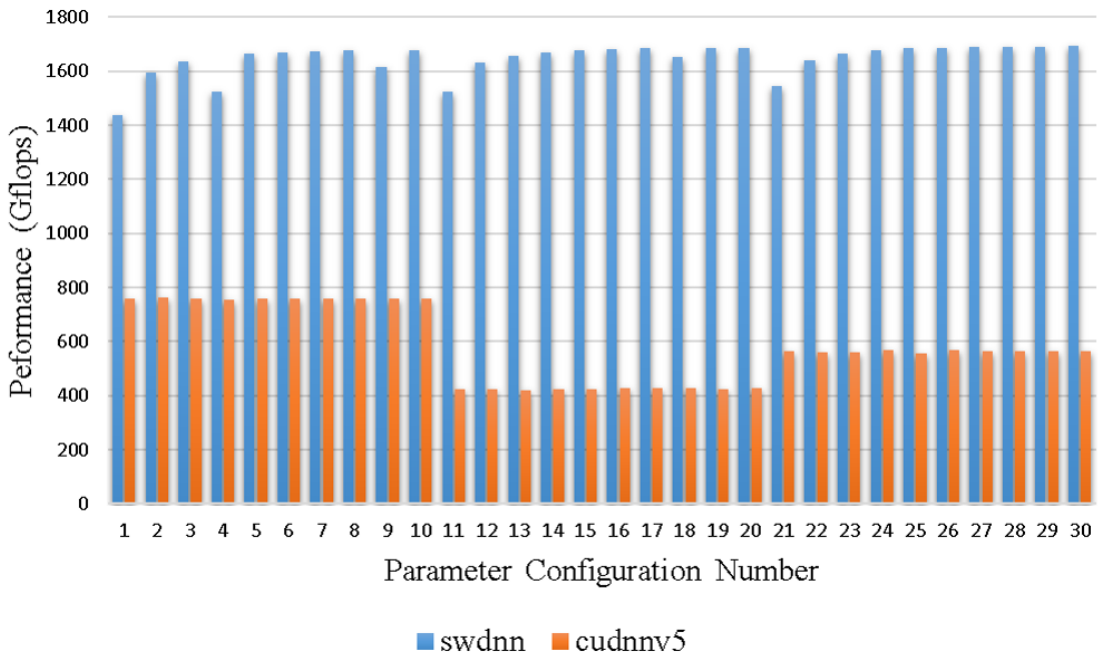


Different input and output channels Tests

Double-precision performance results of our convolution kernels with different (Ni,No) ranging from (64,64) to (384, 384), compared with the K40m GPU results with cuDNNv5. (B = 128, output image =64 × 64, filter = 3 × 3)

Performance Evaluation of Convolutional Layers

Convolution performance is around 1.6 Tflops in double-precision floating-point



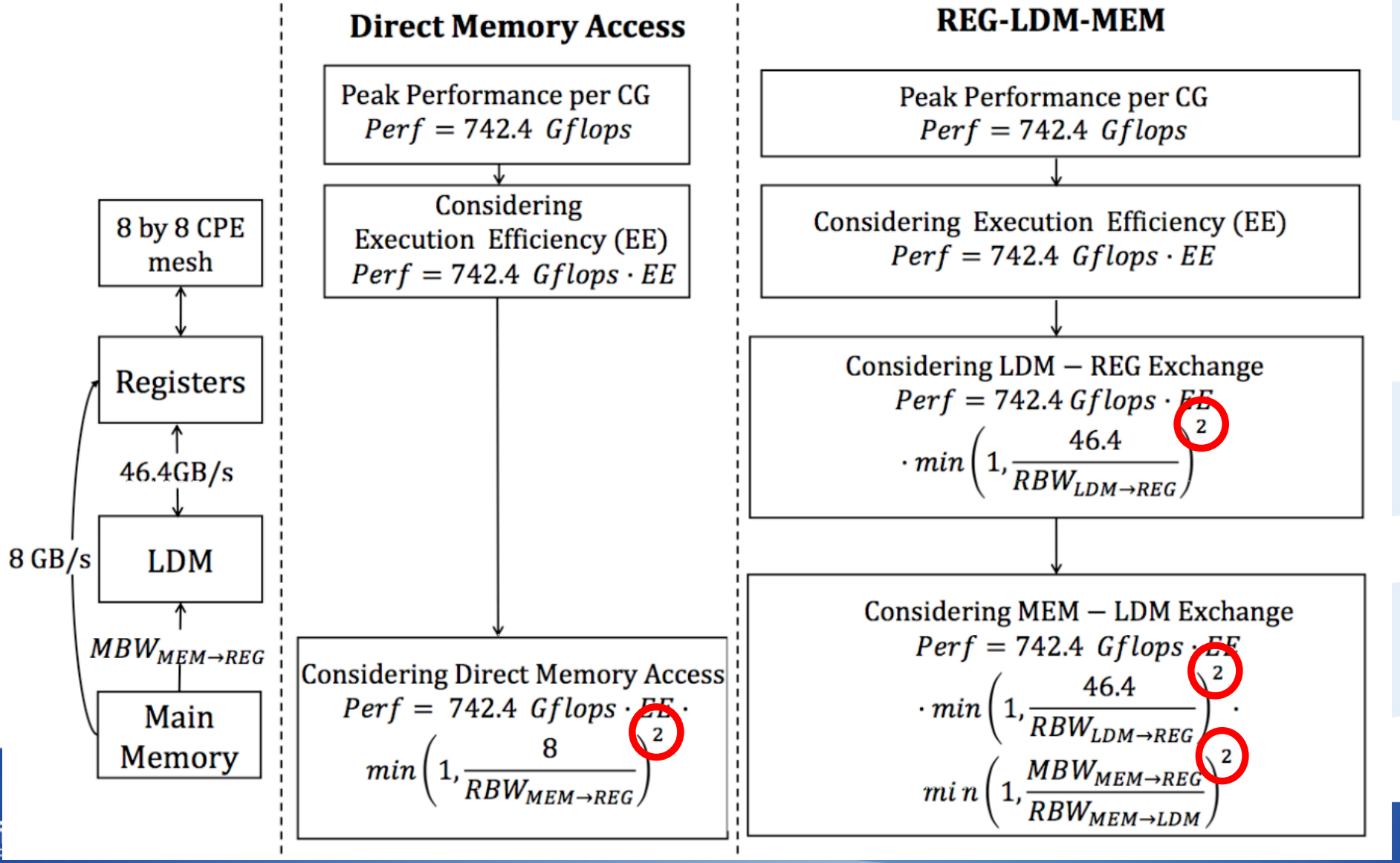
Different filter kernel sizes Tests

Double-precision performance results of our convolution kernels with different (Kr, Kc) ranging from (3, 3) to (21, 21) and Ni ranging rom 128 to 384, , compared with the K40m GPU results with cuDNNv5. (B = 128, output image =64 × 64)

Performance Evaluation of Convolutional Layers

- Speedup ranging from 1.91x to 9.75x compared with cuDNNv5.1 on NVIDIA Tesla K40 with double-precision floating-point.
- Performance is insensitive to parameter configurations – more stable than cuDNN.
- swDNN is about **54%** of the peak performance, while cuDNN is around **40%**.

Revisiting the Performance Model for SW26010

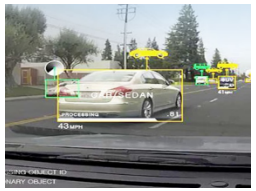


国家超级
National Superco

Evaluation of our Performance Model

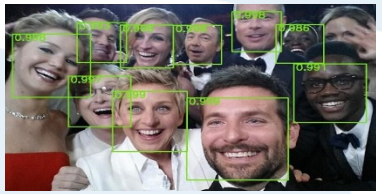
Plan	K_c	b_B	b_{C_o}	N_i	N_o	RBW	MBW	modeled	measured
img	3	32	16	128	128	29.0	21.9	**368**	**350**
img	3	32	8	128	256	23.2	18.2	**397**	**375**
batch	3	-	-	256	256	27.1	21.2	**422**	**410**
batch	3	-	-	128	384	25.7	21.2	**407**	**392**

App


无人驾驶


对弈

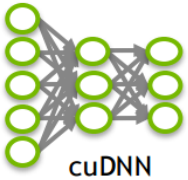

语音

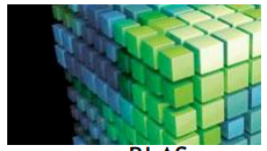

视觉

Framework

KERAS
Microsoft CNTK
MatConvNet
Caffe
Chainer
DL4J Deeplearning4j
TensorFlow
theano
torch
MINERVA
mxnet
Purine

Lib


cuDNN

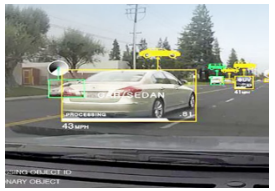

cuBLAS

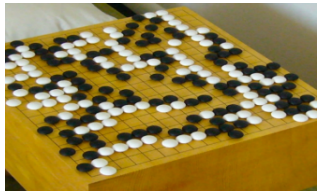
INTEL® MKL
DEEPLEARNING4J

hardware

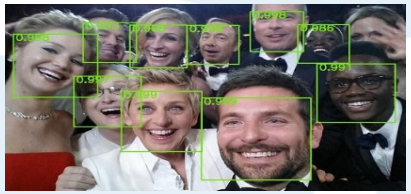
amazon web services
Microsoft Azure
IBM SOFTLAYER
阿里云计算 Alibaba Cloud Computing

App


无人驾驶


对弈


语音


视觉

Framework

KERAS
Microsoft CNTK
MatConvNet
Caffe
Chainer
DL4J Deeplearning4j
SWCaffe
TensorFlow
theano
torch
MINERVA
mxnet
Purine

Lib

cuDNN
cuBLAS
INTEL® MKL
DEEPLEARNING4J
SWDNN

hardware

amazon web services
Microsoft Azure
IBM SOFTLAYER
阿里云计算 Alibaba Cloud Computing

国家超级计算无锡中心
National Supercomputing Center in Wuxi

Thank you for your listening

swDNN: A Library for Accelerating Deep Learning Applications on Sunway TaihuLight Supercomputer

Jiarui Fang*†, *Haohuan Fu* †, Wenlai Zhao*†, Bingwei Chen*†, Weijie Zheng*†, Guangwen Yang*†*

†Tsinghua University

*National Supercomputing Center in Wuxi