

# Building Smart SoCs

Using Virtual Prototyping for the Design and SoC Integration of Deep Learning Accelerators

Holger Keding

Solutions Architect





# Agenda



## Deep Learning Market and Technology Trends

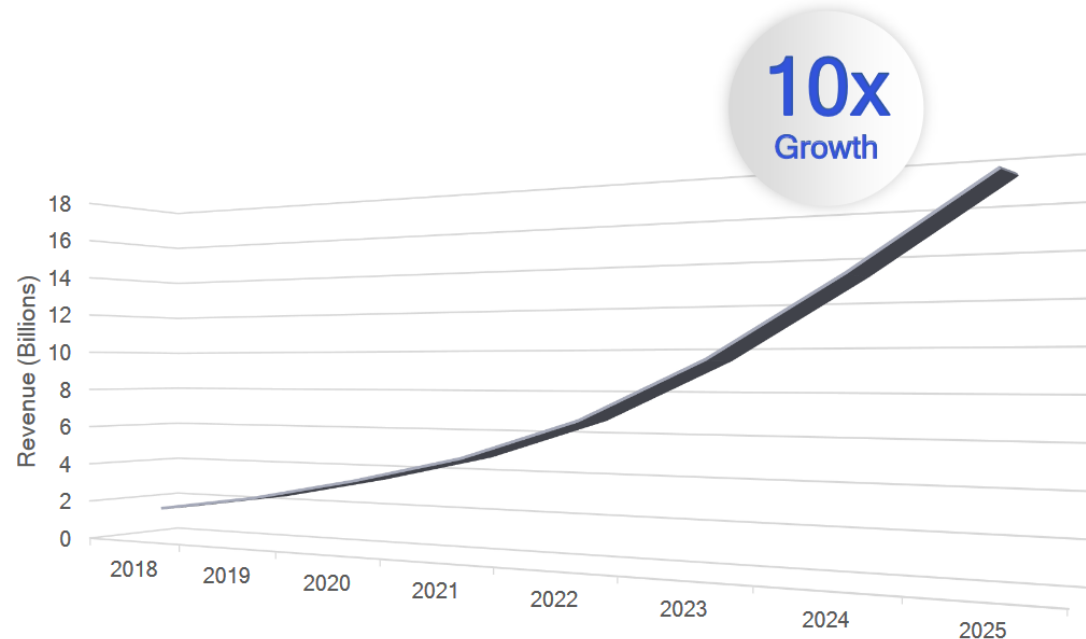
- How to Design a Deep Learning Accelerator (DLA)
  - Analytical Performance Modeling
  - Shift Left Architecture Analysis and Optimization with Virtual Prototyping
- Example
  - Importing Network Algorithms as prototxt + generate analytical model spreadsheet
  - Find suited configuration and scaling parameters in analytical model
  - Validate first results, and explore architecture for dynamic and power aspects using Virtual Platforms
- Summary



# Increasing number of AI Accelerators

By 2025,  
market ramp  
of AI in  
datacenters

**\$17  
Billion**



Source: Qualcomm AI Day Speaker Presentation 2019



# AI Chip Landscape

V0.5 August, 2019

S.T.



All information contained within this infographic is gathered from the internet and periodically updated, no guarantee is given that the information provided is correct, complete, and up-to-date.



# Deep Learning Technology Trends

## New Neural Network algorithms

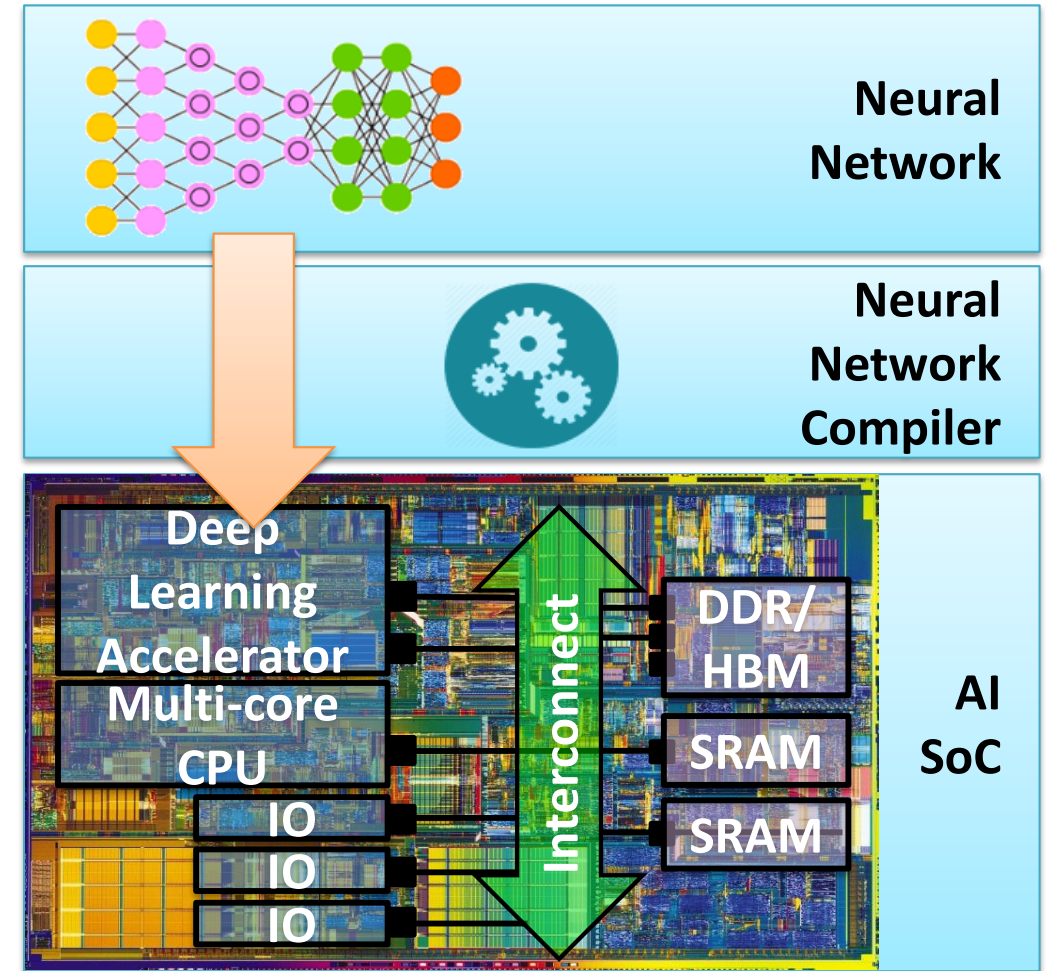
- Higher accuracy, lower size and less processing
- But: less data re-use, less cycles per byte

## Neural Network Compiler optimizations

- Loop-tiling, -unrolling, and -parallelization
- Splitting and fusing of Neural Network layers
- Memory layout optimization across layers
- Optimized code generation to utilize available hardware accelerators

## Deep Learning Accelerator optimizations

- Schedule workload on parallel hardware engines
- Optimize and reduce data transfers to and from memory





# AI SoC Design Challenges

Brute-force Processing of Huge Data Sets

- **Choosing the right algorithm and architecture: CPU, GPU, FPGA, vector DSP, ASIP**
  - CNN graphs evolving fast, need short time to market, cannot optimize for one single graph
  - Joint design of algorithm, compiler, and target architecture
  - Joint optimization of power, performance, accuracy, and cost
- **Highly parallel compute drives memory requirements**
  - High on-chip and chip to chip bandwidth at low latency
  - High memory bandwidth requirements for parameters and layer to layer communication
- **Performance analysis requires realistic workloads to consider dynamic effects**
  - Scheduling of AI operators on parallel processing elements
  - Unpredictable interconnect and memory access latencies

Large Design Space drives Differentiation by  
AI Algorithm & Architecture



# Agenda

- Deep Learning Market and Technology Trends

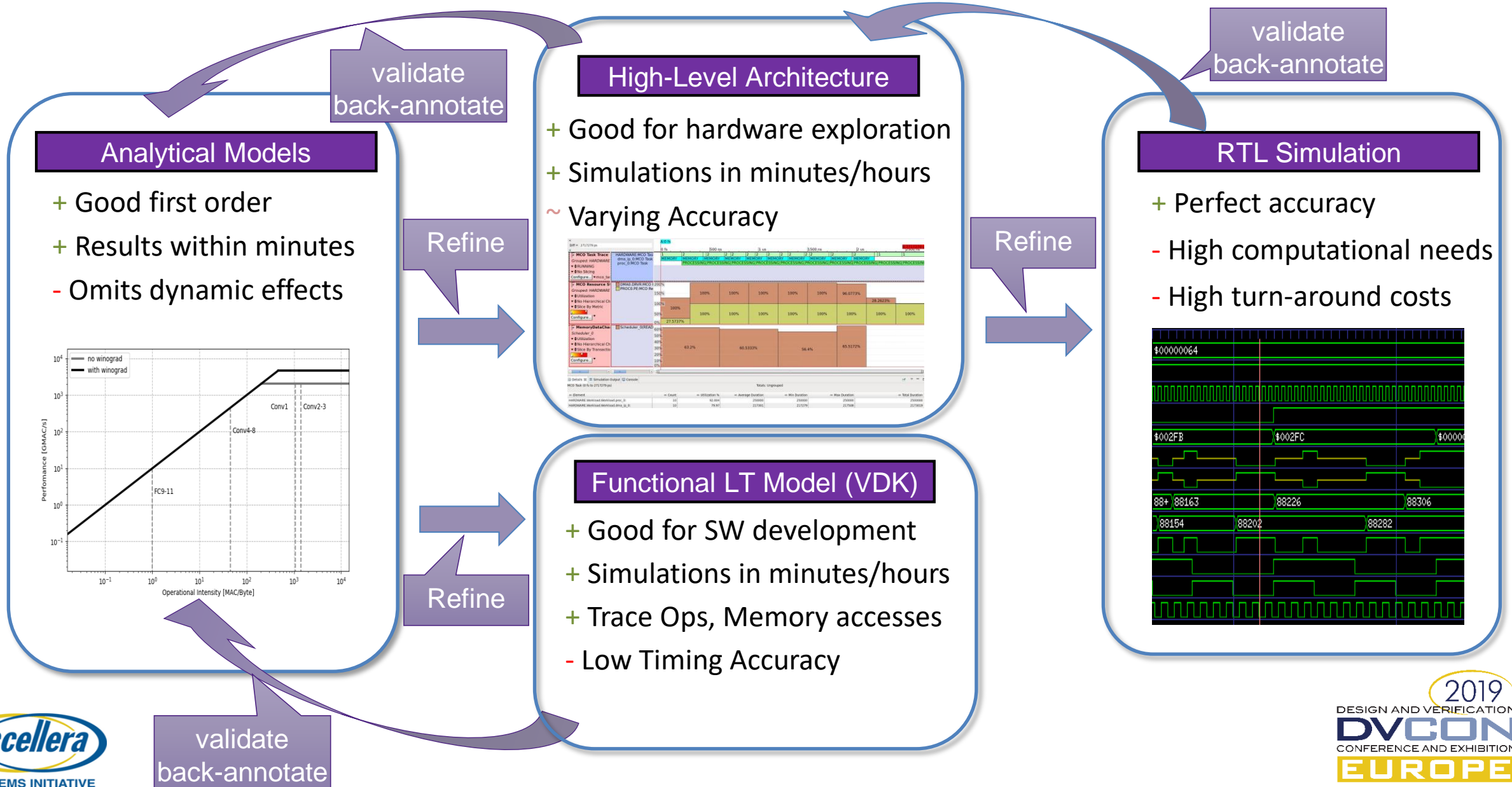


## How to Design a Deep Learning Accelerator (DLA)

- Analytical Performance Modeling
- Shift Left Architecture Analysis and Optimization with Virtual Prototyping
- Example
  - Importing Network Algorithms as prototxt + generate analytical model spreadsheet
  - Find suited configuration and scaling parameters in analytical model
  - Validate first results, and explore architecture for dynamic and power aspects using Virtual Platforms
- Summary



# How to design a DLA?



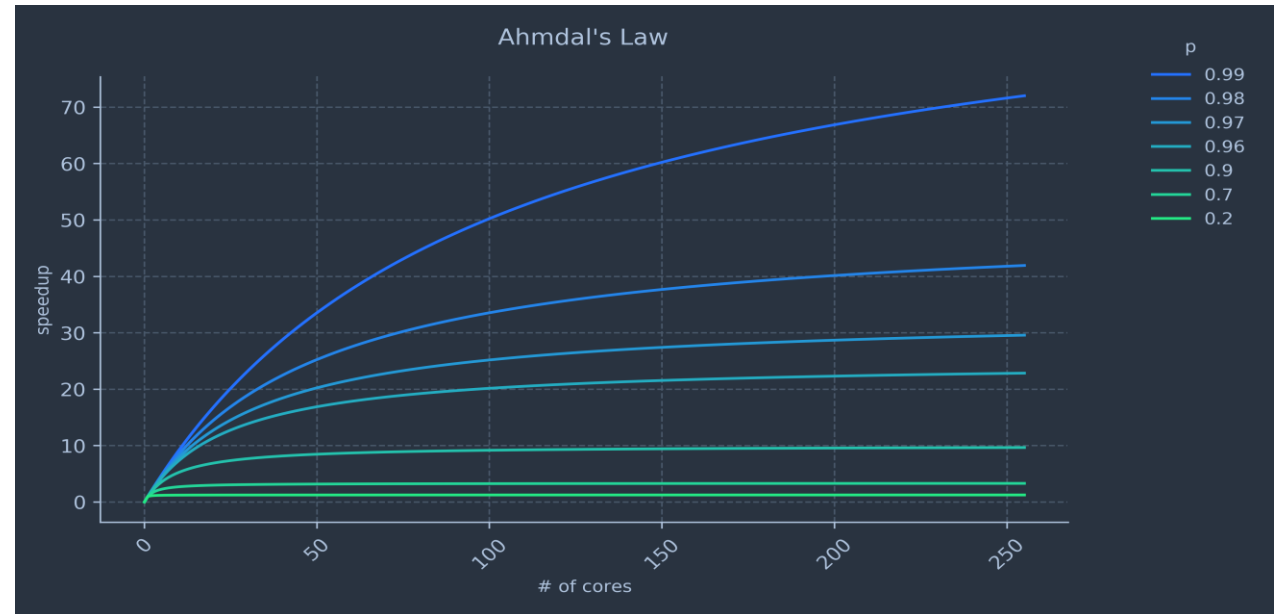


# Analytical Performance Models

## Simple Example: Amdahl's Law [1]

$$S_{\text{speedup}} = \frac{1}{(1 - p) + \frac{p}{s}}$$

$$\lim_{s \rightarrow \infty} S_{\text{speedup}} = \frac{1}{1 - p}$$

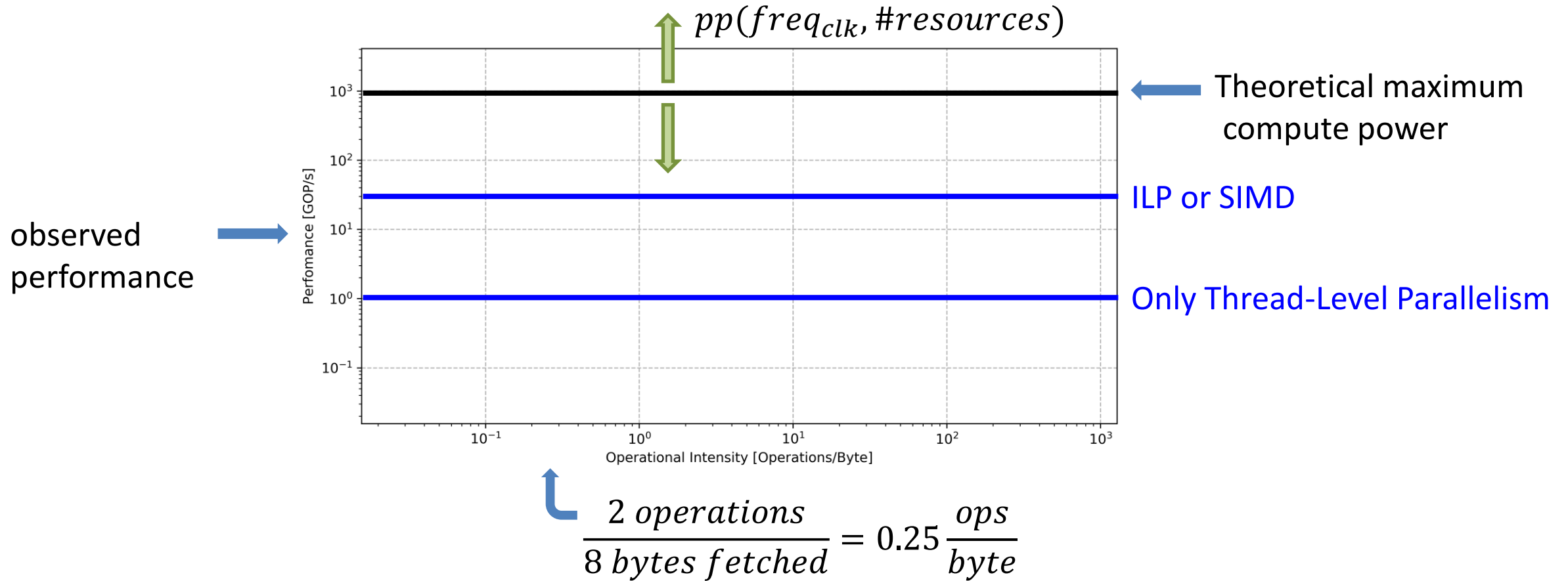


[1] *Validity of the Single Processor Approach to Achieving Large Scale Computing Capabilities* (1967)

- *Simple insightful formula, with restricted applicability, though.*
- *“All models are wrong but some are useful” (George Box, 1978)*



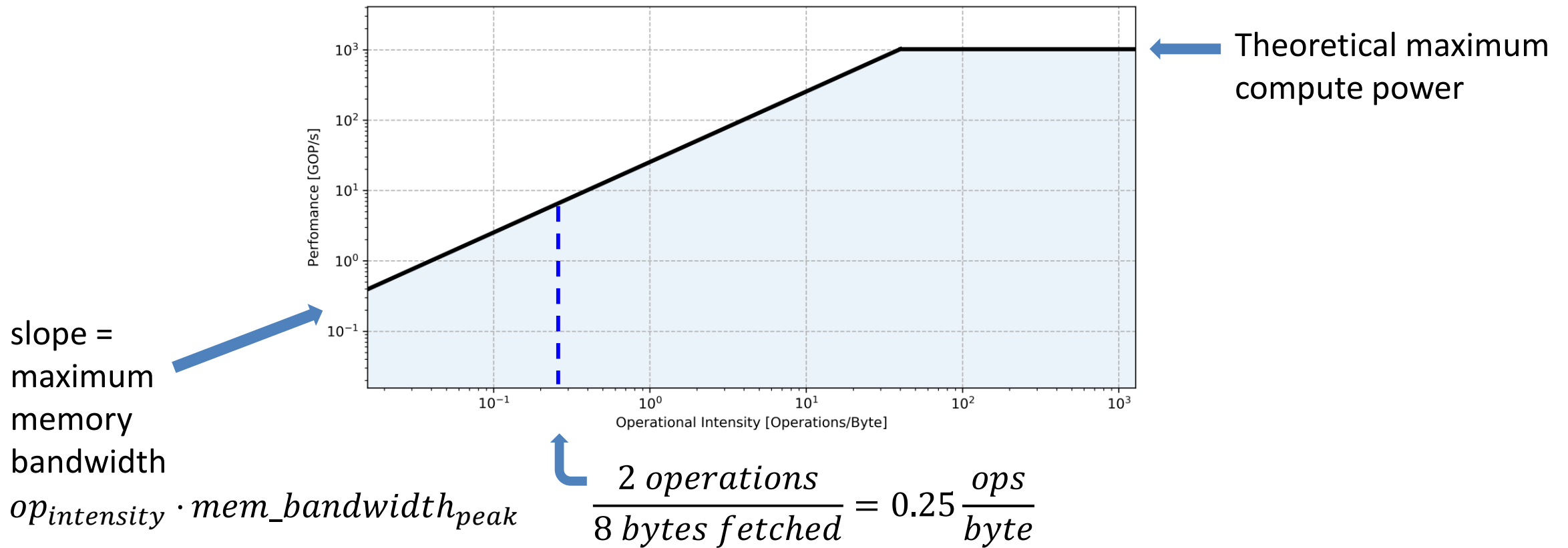
# Analytical Models – Roofline Models (1)



Roofline: an insightful visual performance model for multicore architectures (Williams, Waterman, Patterson, 2009)



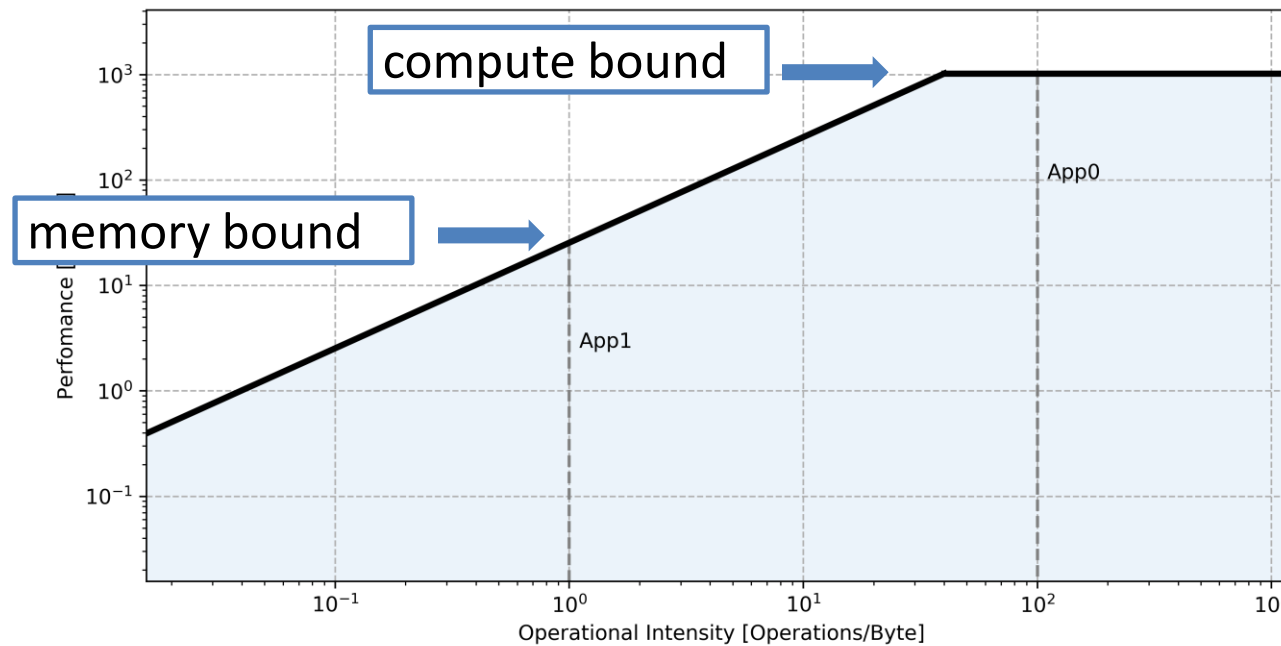
# Analytical Models – Roofline Models (2)



Roofline: an insightful visual performance model for multicore architectures (Williams, Waterman, Patterson, 2009)



# Analytical Models – Roofline Models (3)

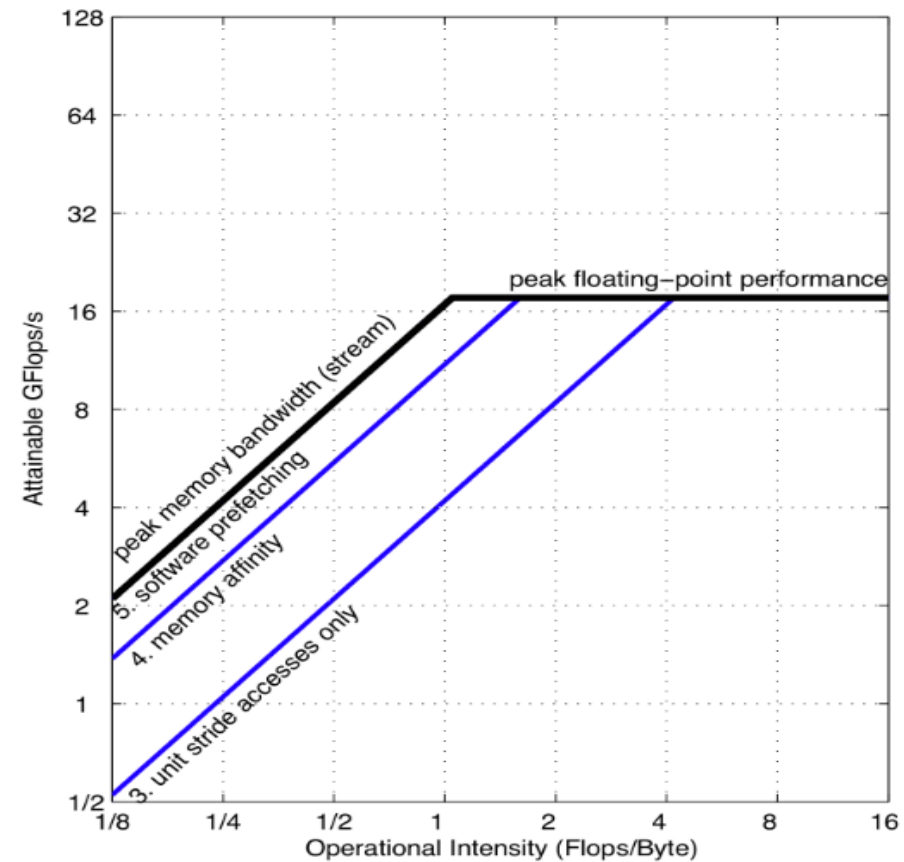
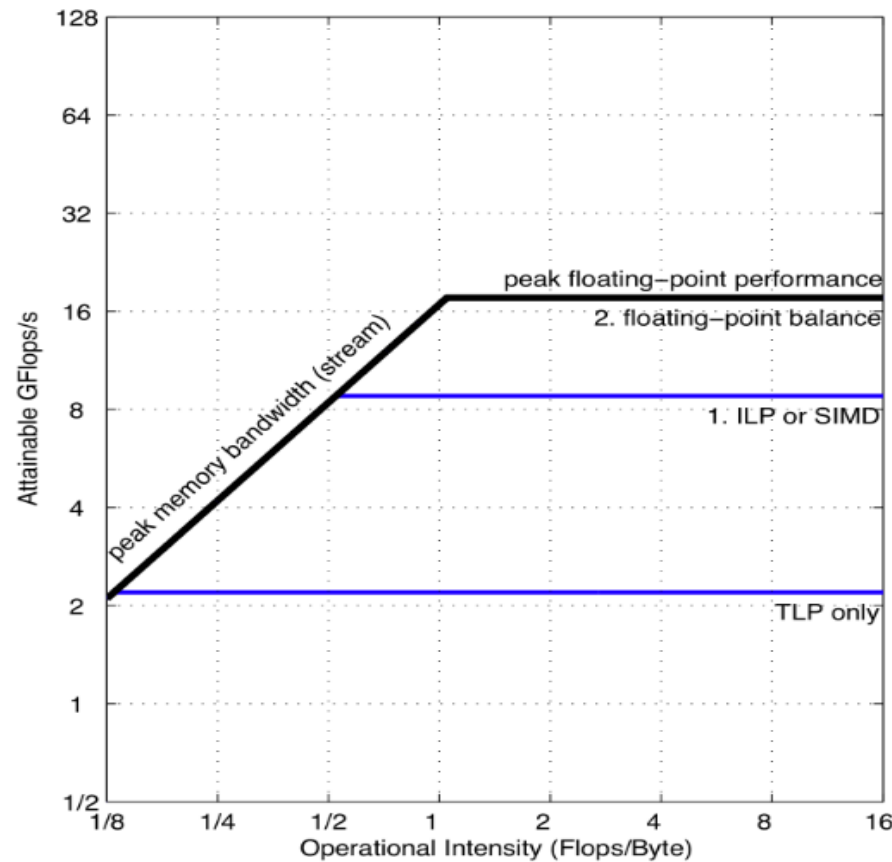


*Roofline: an insightful visual performance model for multicore architectures (Williams, Waterman, Patterson, 2009)*



# Analytical Models – Roofline Models

$$performance = \min( performance_{peak}, op_{intensity} \cdot mem\_bandwidth_{peak} ) \quad op_{intensity} = \frac{operations}{bytes}$$



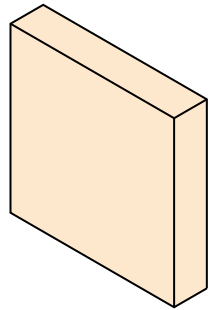
Roofline: an insightful visual performance model for multicore architectures (Williams, Waterman, Patterson, 2009)



# Example: Analytical Model for CNN Convolutional Layer (1)

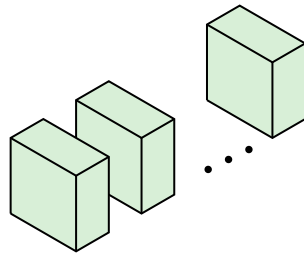
## Conv1 of AlexNet

input cube



227x227x3

filter kernels

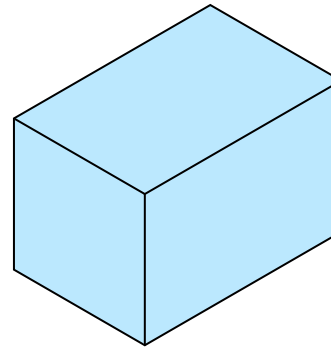


96x11x11x3

\*

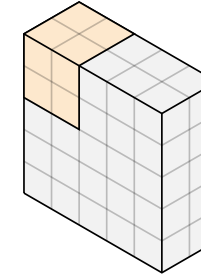
=

output cube

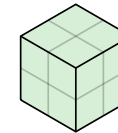


55x55x96

input cube



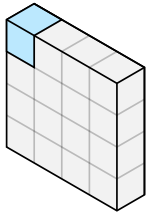
kernel



\*

=

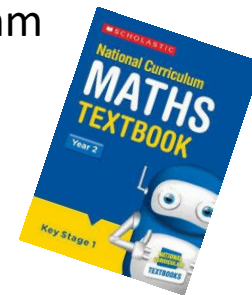
output cube



```
for(row=0; row<oh; row++){
  for(col=0; col<ow; col++){
    for(k=0; k<oc; k++){
      for(ti=0; ti<ic; ti++){
        for(i=0; i<kh; i++){
          for(j=0; j<kw; j++){
            L : outputfm [ k ] [ row ] [ col ] +=
              kernels[ k ][ ti ][ i ][ j ] *
              inputfm[ ti ][ sw*row+i ][ sh*col+j ];
          }
        }
      }
    }
  }
}
```

Maths Textbook

Convolution algorithm



$$\begin{aligned}
 n_{MAC} &= o_h \cdot o_w \cdot o_c \cdot k_w \cdot k_h \cdot i_c \\
 &= 55 \cdot 55 \cdot 96 \cdot 11 \cdot 11 \cdot 3 \\
 &= 105,415,200
 \end{aligned}$$



# Example: Analytical Model for CNN Convolutional Layer (2)

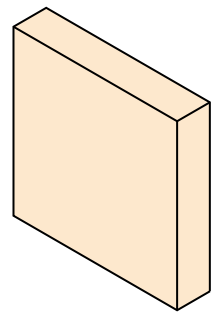
## Conv1 of AlexNet

input cube

filter kernels

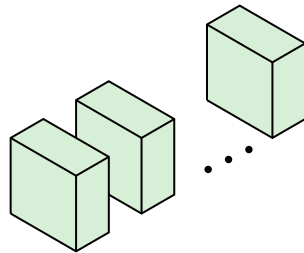
output cube

**But:** here we assume unlimited amount of local memory



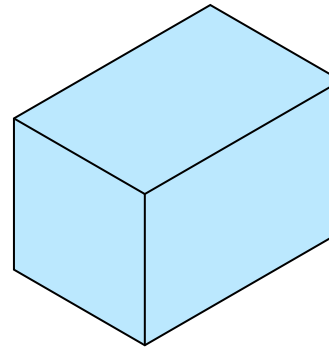
227x227x3

\*



96x11x11x3

=

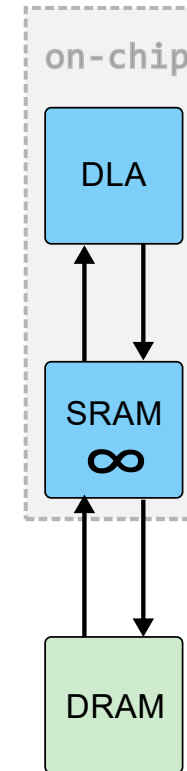
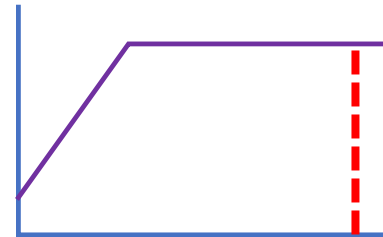


55x55x96

$$n_{MAC} = o_h \cdot o_w \cdot o_c \cdot k_w \cdot k_h \cdot i_c = 55 \cdot 55 \cdot 96 \cdot 11 \cdot 11 \cdot 3 = 105,415,200$$

$$d_{MAC} = d_{ifmap} + d_{kernel} = (i_w \cdot i_h \cdot i_c + k_w \cdot k_h \cdot i_c \cdot k) \cdot B_i \approx 0.38 \text{ MiB}$$

$$\Rightarrow \text{Operational Intensity } I = \frac{n_{MAC}}{d_{MAC}} \approx 278 \text{ ops/B}$$





# Example: Analytical Model for CNN Convolutional Layer (3)

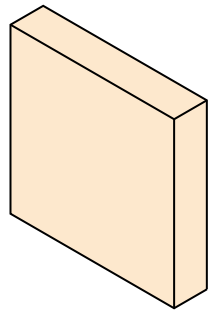
## Conv1 of AlexNet

input cube

filter kernels

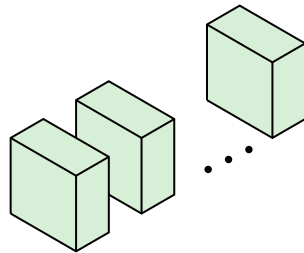
output cube

**Opposite extreme:** we assume no  
local memory



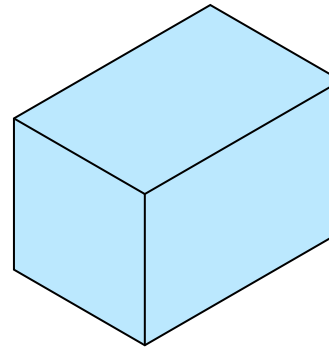
227x227x3

\*



96x11x11x3

=

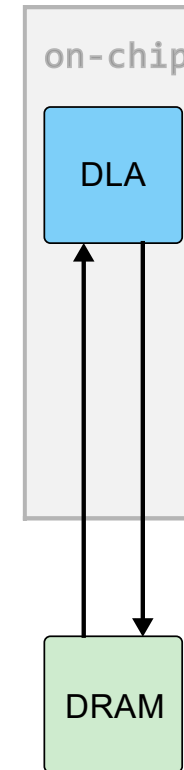
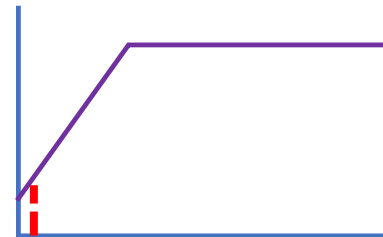


55x55x96

$$n_{MAC} = o_h \cdot o_w \cdot o_c \cdot k_w \cdot k_h \cdot i_c = 55 \cdot 55 \cdot 96 \cdot 11 \cdot 11 \cdot 3 = 105,415,200$$

$$d_{MAC} = 2 \cdot n_{MAC} \cdot B_i \approx 420 \text{ MiB}$$

$$\Rightarrow \text{Operational Intensity } I = \frac{n_{MAC}}{d_{MAC}} \approx \frac{1}{4} \text{ ops/B}$$





# Example: Analytical Model for CNN Convolutional Layer (4)

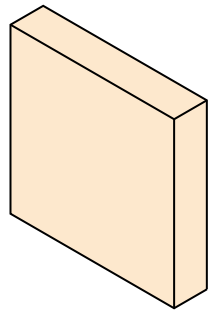
## Conv1 of AlexNet

input cube

filter kernels

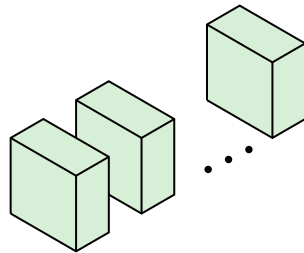
output cube

**Practical setup:** limited amount  
of local memory



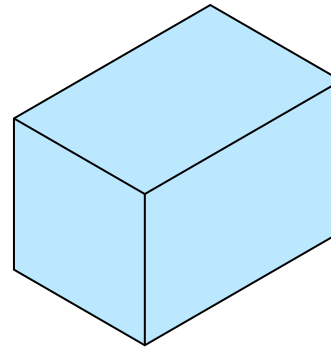
227x227x3

\*



96x11x11x3

=

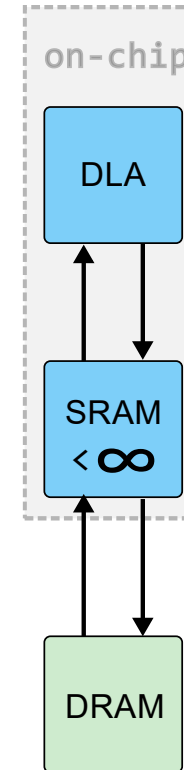
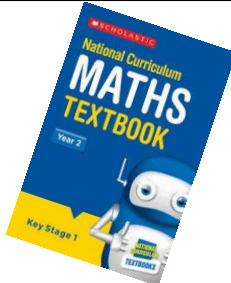


55x55x96

```
for(row=0; row<oh; row++)
  for(col=0; col<ow; col++)
    for(k=0; k<oc; k++){
      for(ti=0; ti<tc; ti++){
        for(i=0; i<sh; i++){
          for(j=0; j<sw; j++){
            L : output_fm[k][row][col][ti] +=
              kernels[k][ti][i][j] * input_fm[ti][sw*row+i][sh*col+j];
          }
        }
      }
    }
  }
```

Maths Textbook

Convolution algorithm

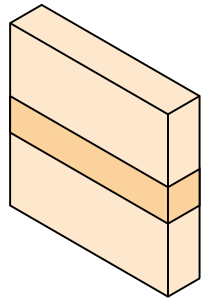




# Example: Analytical Model for CNN Convolutional Layer (5)

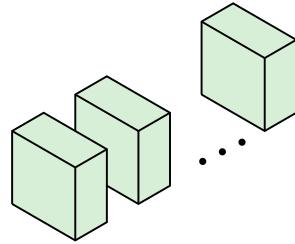
Conv1 of AlexNet – with very simple tiling

input cube



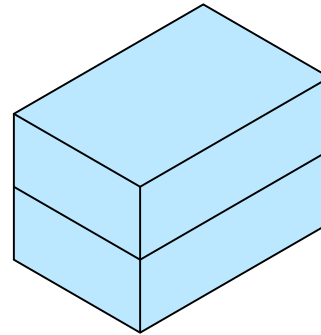
119x227x3  
115x227x3

filter kernels



96x11x11x3

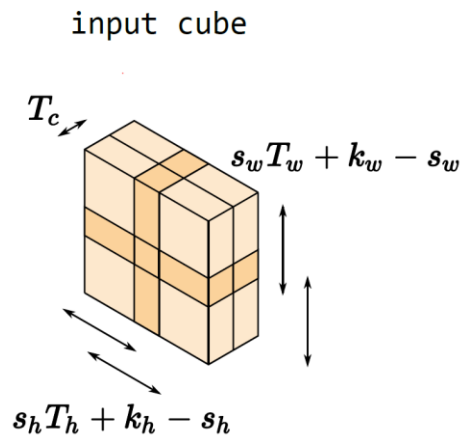
output cube



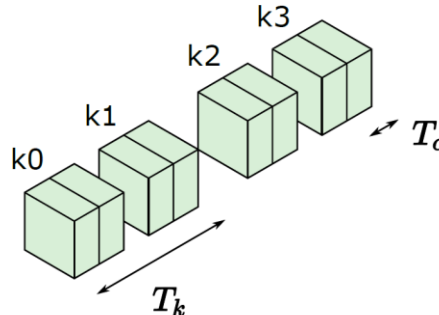
27x55x96  
28x55x96

**Practical setup:** limited amount  
of local memory

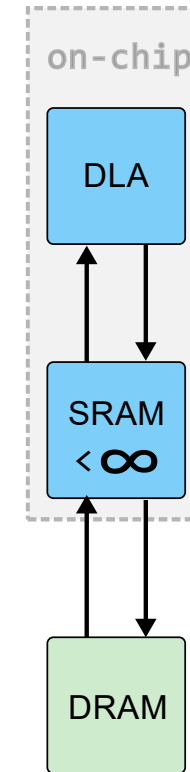
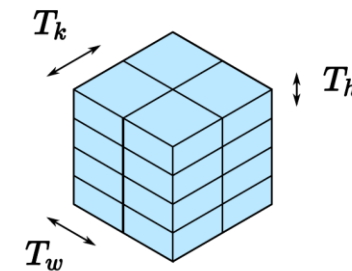
Width + Height  
+ Channel  
+ Kernel Tiling



filter kernels



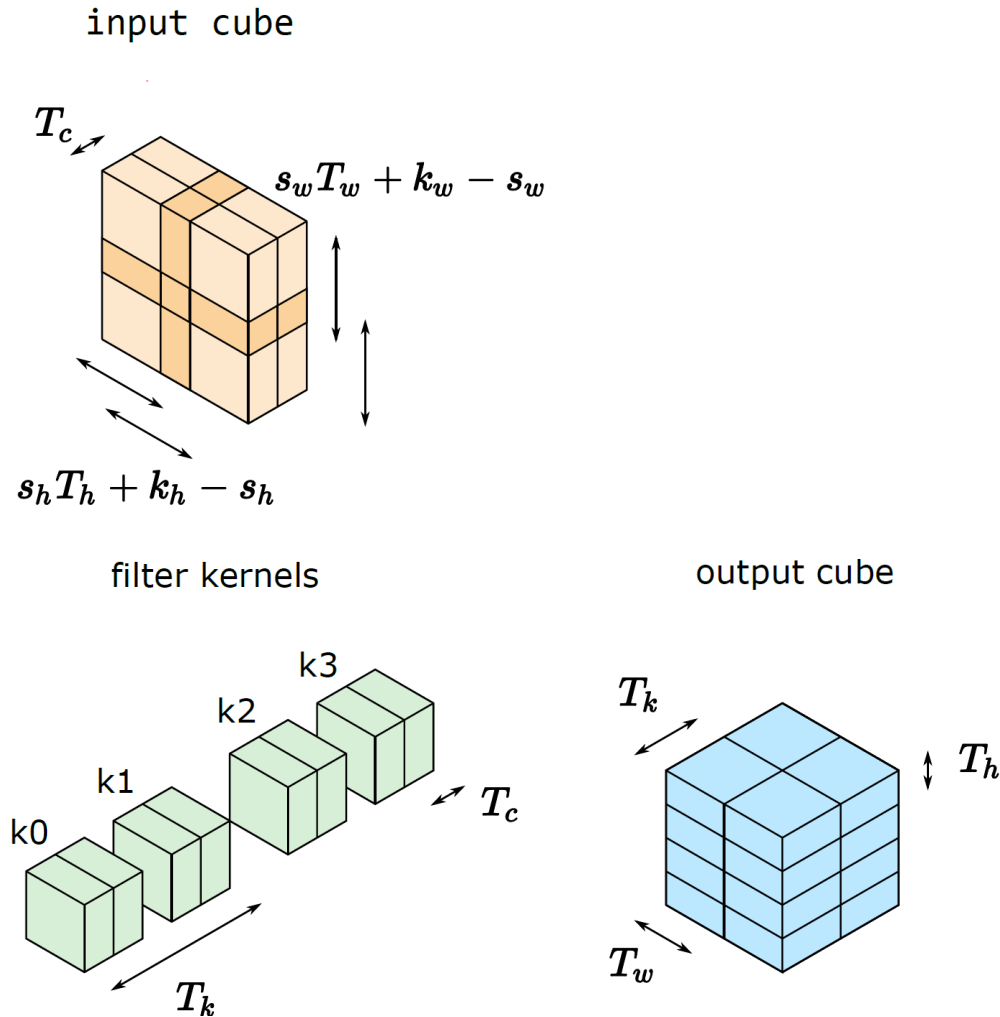
output cube





# Example: Analytical Model for CNN Convolutional Layer (6)

## Conv1 with tiling



```

off-chip
for(row=0; row<oh; row+=Th){
  for(col=0; col<ow; col+=Tw){
    for(to=0; to<k; to+=Tk){
      for(ti=0; ti<ic; ti+=Tc){
        //load output, weight and input

        //on-chip computation
        on-chip
        for(i=0; i<kh; i++){
          for(j=0; j<kw; j++){
            for(trr=row; trr<min(row+Th,oh); trr++){
              for(tcc=col; tcc<min(col+Tw,ow); tcc++){
                #pipeline
                for(too=to; too<min(to+Tk,k); too++){
                  #unroll
                  for(tii=ti; tii<min(ti+Tc, ic); tii++){
                    #unroll
                    output[too][trr][tcc] +=
                      weights[too][tii][i][j]
                      input[tii][sh*trr+i][sw*tcc+j]
                  }
                }
              }
            }
          }
        }
      }
    }
  }
}
//store output
    
```

Source: Optimizing FPGA-based Accelerator Design for Deep Convolutional Neural Networks, Cheng Zhang, 2015



# Example: Analytical Model for CNN Convolutional Layer (6)

## Conv1 with tiling

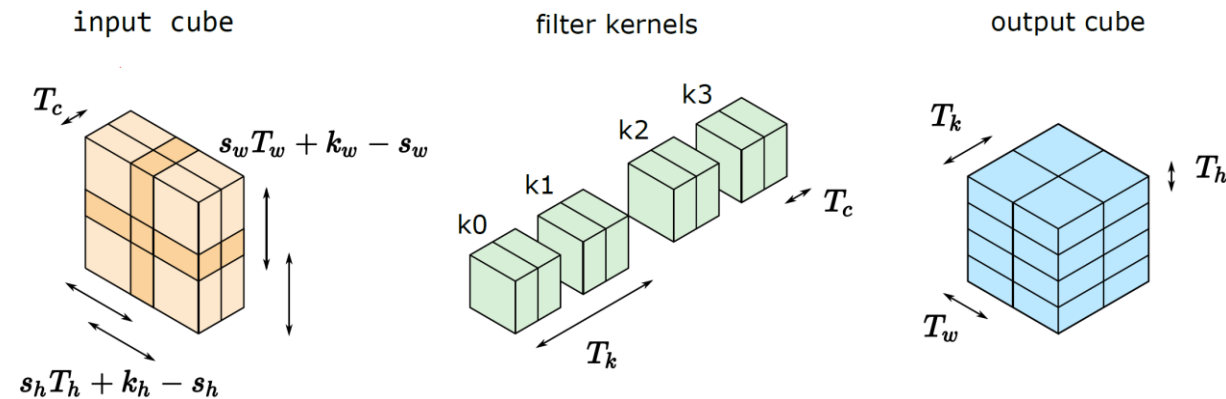
$$d_{input} = \frac{i_c}{T_c} \cdot \frac{k}{T_k} \cdot \frac{o_w}{T_w} \cdot \frac{o_h}{T_h} \cdot (T_c \cdot (S_h T_h + k_h - S_h) \cdot (S_w T_w + k_w - S_w))$$

$$d_{weight} = \frac{i_c}{T_c} \cdot \frac{k}{T_k} \cdot \frac{o_w}{T_w} \cdot \frac{o_h}{T_h} \cdot (T_c T_k \cdot k_w k_h)$$

Now it gets more tricky: Taking into account non-integer relations of tiling parameters and channel dimensions:

$$d_{weight} = \frac{i_c}{T_c} \cdot \frac{k}{T_k} \cdot \left\lceil \frac{o_w}{T_w} \right\rceil \cdot \left\lceil \frac{o_h}{T_h} \right\rceil \cdot (T_c T_k \cdot k_w k_h)$$

$$\begin{aligned} d_{input} = & i_c \cdot \left\lceil \frac{k}{T_k} \right\rceil \cdot \left( \left\lceil \frac{o_w}{T_w} \right\rceil \cdot \left\lceil \frac{o_h}{T_h} \right\rceil \cdot ((S_h T_h + k_h - S_h) \cdot (S_w T_w + k_w - S_w)) \right. \\ & + \left\lceil \frac{(o_w \bmod T_w)}{T_w} \right\rceil \cdot ((S_h T_h + k_h - S_h) \cdot (S_w \cdot (o_w \bmod T_w) + k_w - S_w)) \\ & + \left\lceil \frac{(o_h \bmod T_h)}{T_h} \right\rceil \cdot ((S_h \cdot (o_h \bmod T_h) + k_h - S_h) \cdot (S_w \cdot T_w + k_w - S_w)) \\ & + \left\lceil \frac{(o_h \bmod T_h)}{T_h} \right\rceil \cdot \left\lceil \frac{(o_w \bmod T_w)}{T_w} \right\rceil \cdot ((S_h \cdot (o_h \bmod T_h) + k_h - S_h) \cdot (S_w \cdot (o_w \bmod T_w) + k_w - S_w)) \end{aligned}$$

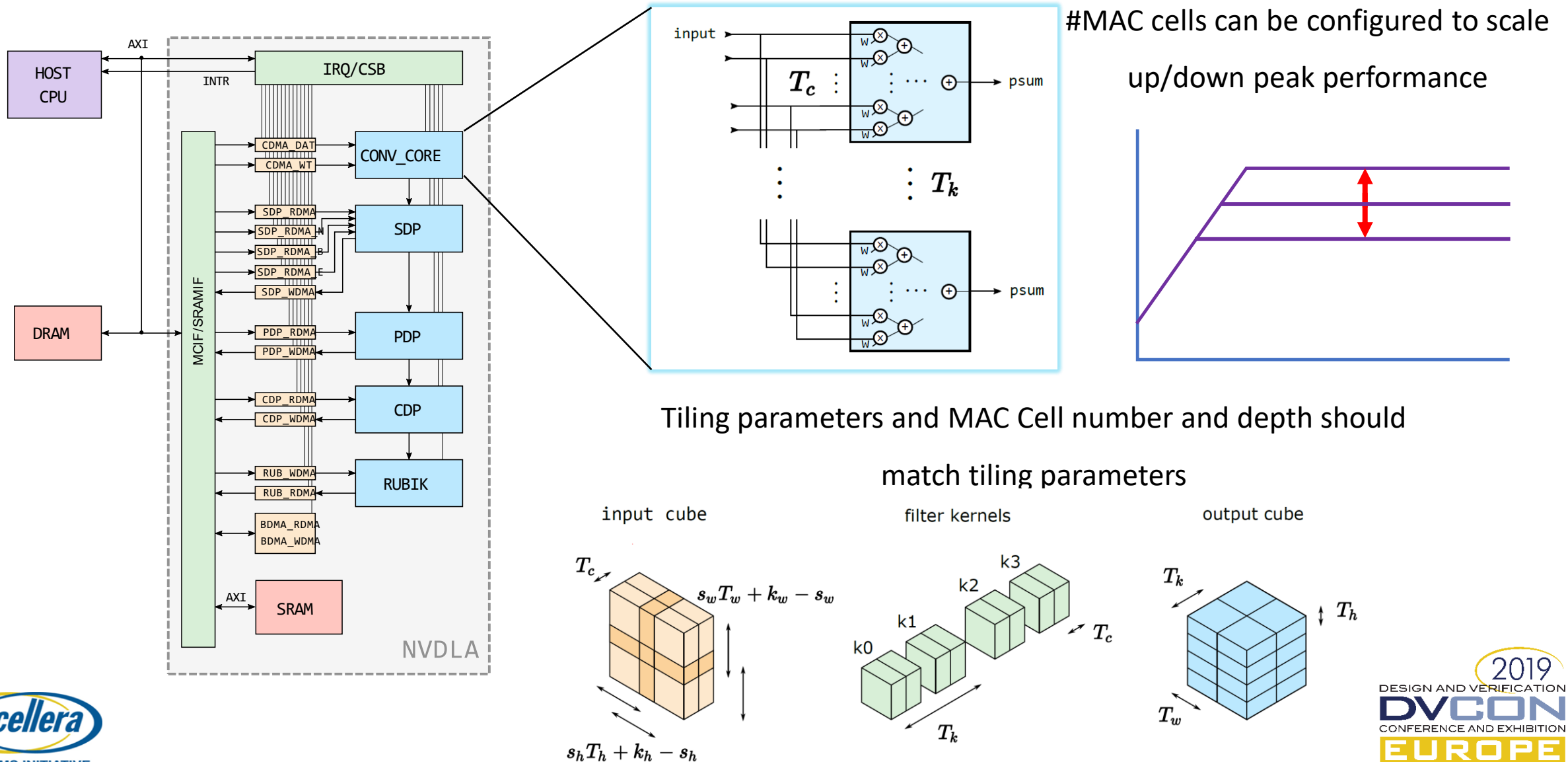


Tiling also brings the operational intensity closer to the optimum HW utilization point





# Example: Analytical Model, Mapping Conv to HW Resources



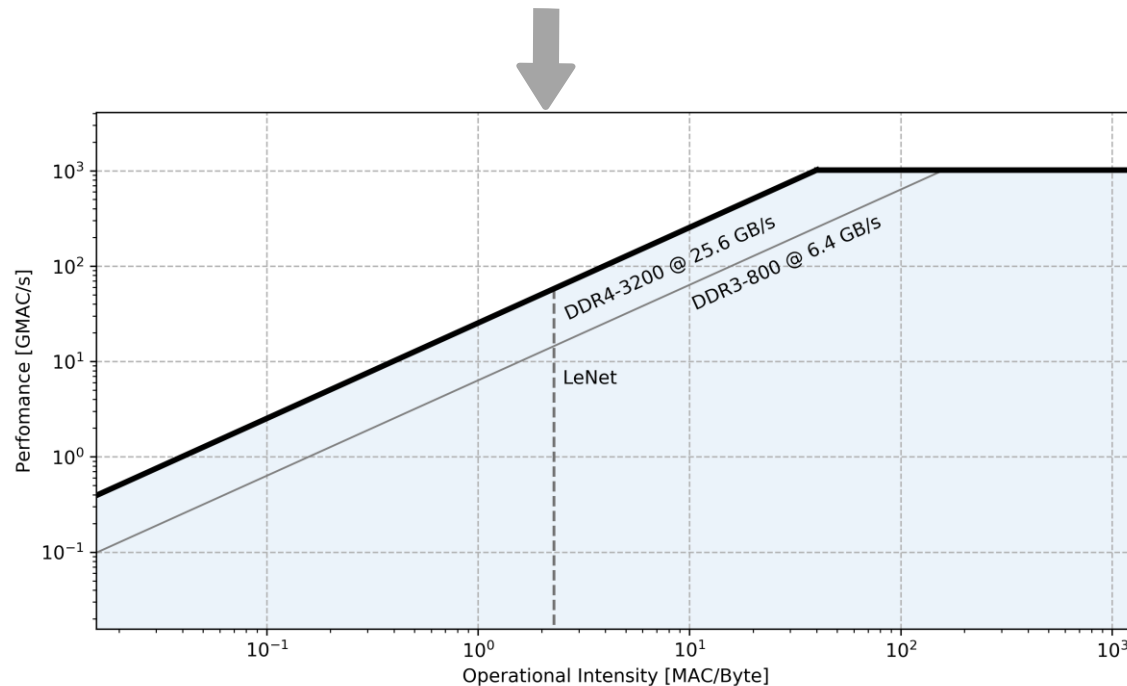


# Roofline model

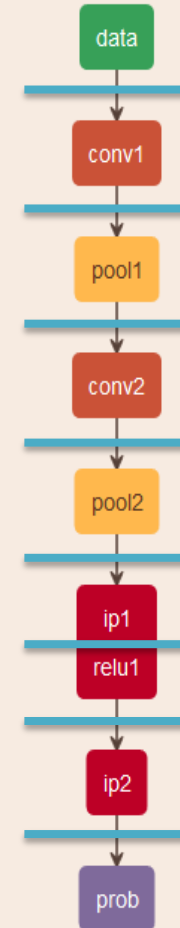
## 4.7.1 Convolution

MAC operations	$o_w \cdot o_h \cdot o_c \cdot k_w \cdot k_h \cdot \frac{i_c}{l_{group}}$
MAC cycles	$l_{group} \left\lceil \frac{i_c}{T_c \cdot l_{group}} \right\rceil \left\lceil \frac{k}{T_k \cdot l_{group}} \right\rceil \cdot o_h \cdot o_w \cdot k_w \cdot l$
$d_{input} = d_{CDMA\_DAT}$	$i_h \cdot i_w \cdot \left\lceil \frac{i_c \cdot b_i}{atom_{DMA} \cdot l_{group}} \right\rceil \cdot atom_{DMA} \cdot l_{group}$

Operational Intensity (Operations/Byte)



## LeNet



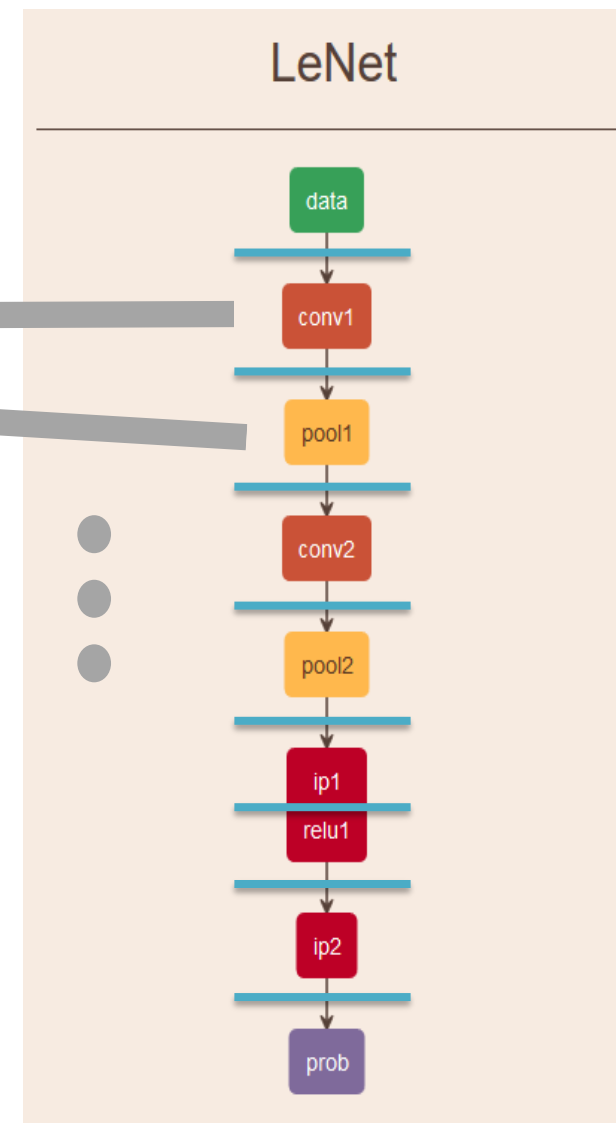
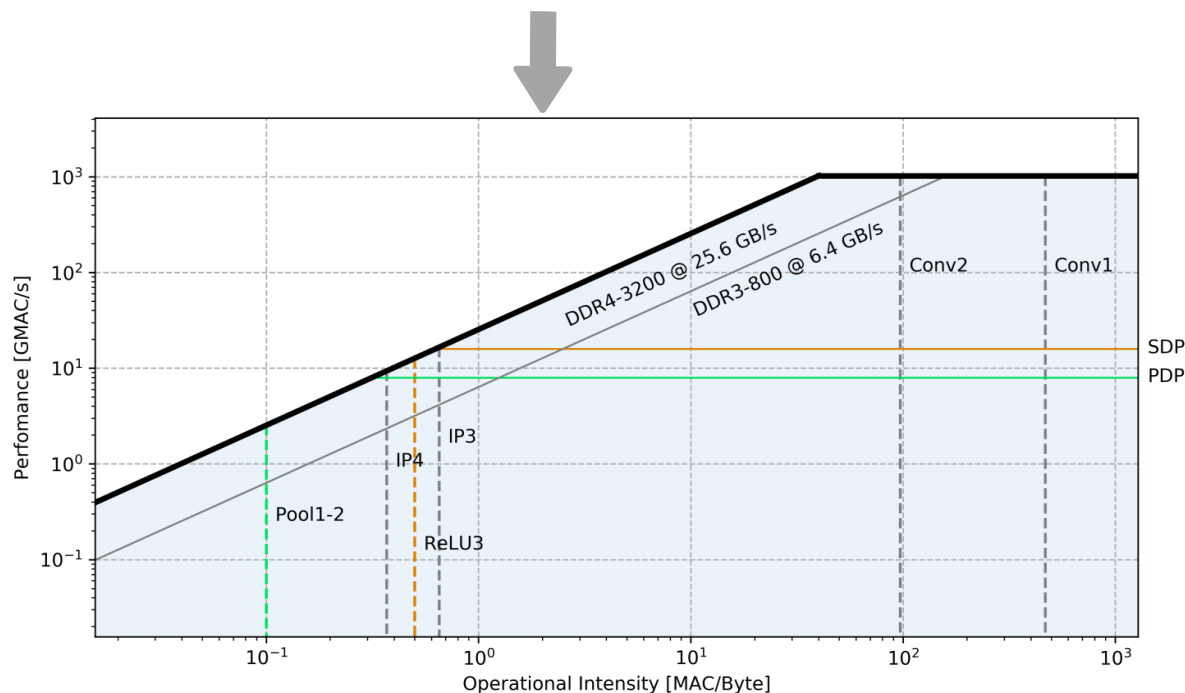


# Roofline model

## 4.7.1 Convolution

MAC operations	$o_w \cdot o_h \cdot o_c \cdot k_w \cdot k_h \cdot \frac{i_c}{l_{group}}$
MAC cycles	$l_{group} \left\lceil \frac{i_c}{T_c \cdot l_{group}} \right\rceil \left\lceil \frac{k}{T_k \cdot l_{group}} \right\rceil \cdot o_h \cdot o_w \cdot k_w \cdot l$
$d_{input} = d_{CDMA\_DAT}$	$i_h \cdot i_w \cdot \left\lceil \frac{i_c \cdot b_i}{atom_{DMA} \cdot l_{group}} \right\rceil \cdot atom_{DMA} \cdot l_{grc}$

Operational Intensity (Operations/Byte)





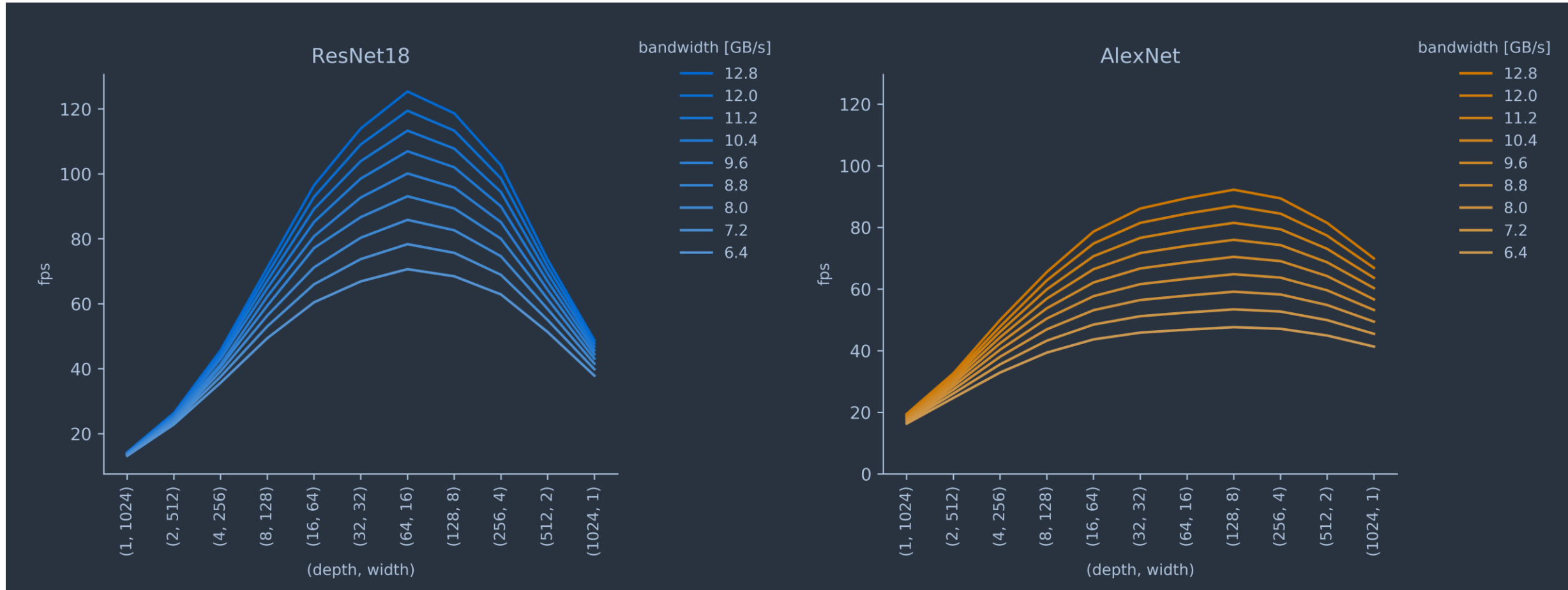
# Analytical Model as Python Generated Spreadsheet

Expressions represent both Algorithmic and HW -> calculate attainable performance

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	AA	AB
1																												
2																												
3																												
4																												
5																												
6																												
7																												
8																												
9																												
10																												
11																												
12																												
13																												
14																												
15																												
16																												
17																												
18																												
19																												
20																												
21																												



# Exploring different numbers of MAC cells and their depth





# Analytical Model Summary

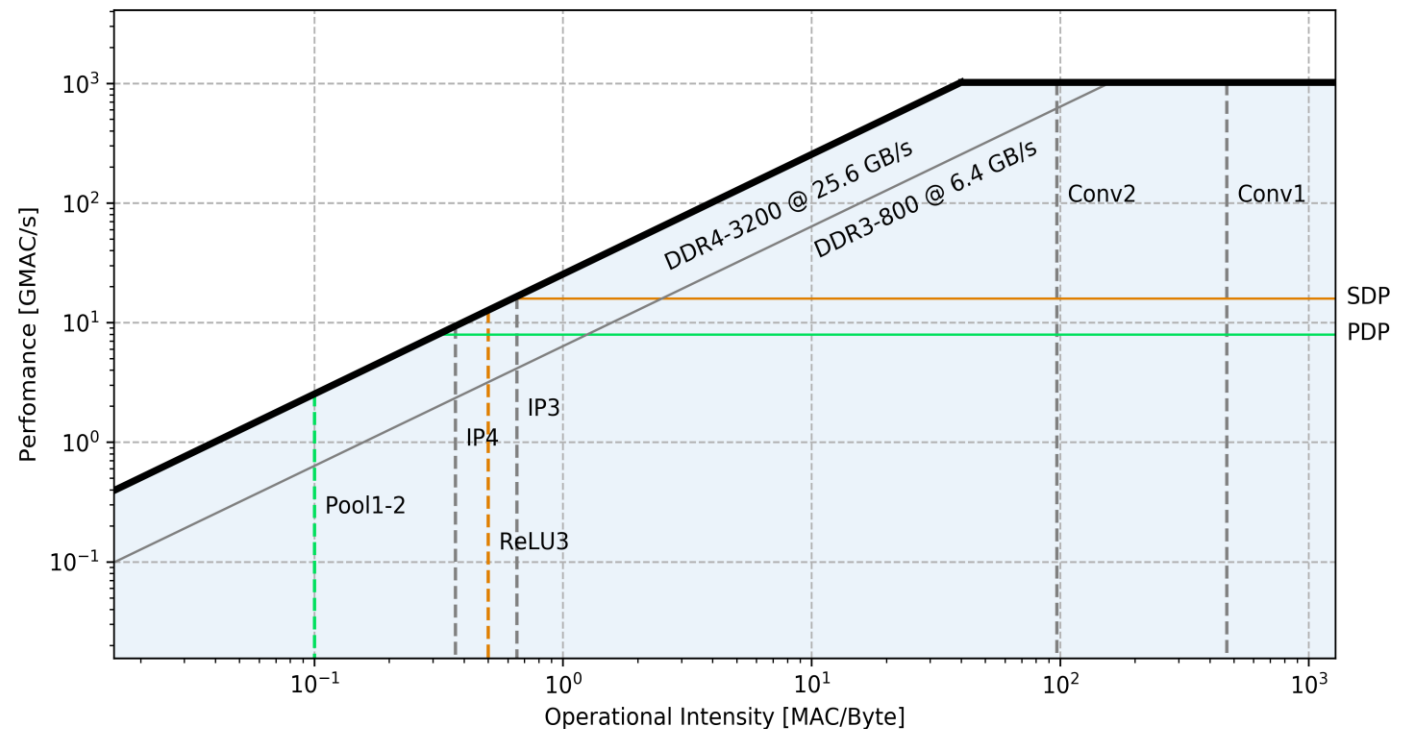
What is achieved and what comes next?

What we have seen:

- + Good first order analysis of static effects
- + Results within minutes
- ~ Requires deep understanding of both algorithm and architecture

What is not covered

- Implementation overhead is hard to predict and not 'priced in' in first round
- Omits dynamic effects
- Joint performance and power is difficult

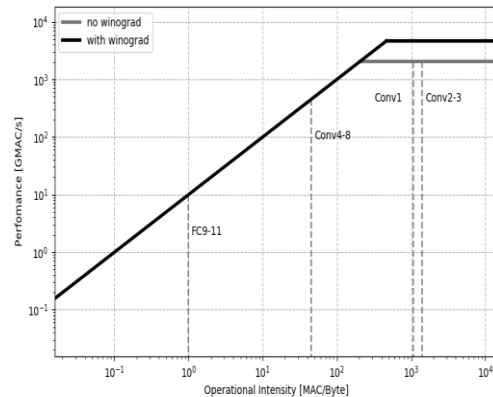




# How to design a DLA?

## Analytical Models

- + Good first order
- + Results within minutes
- Omits dynamic effects



validate  
back-annotate

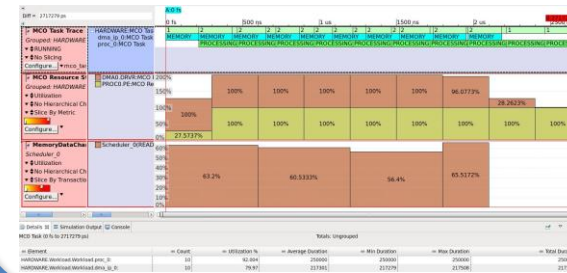
Refine

Refine

validate  
back-annotate

## High-Level Architecture

- + Good for hardware exploration
- + Simulations in minutes/hours
- ~ Varying Accuracy



Refine

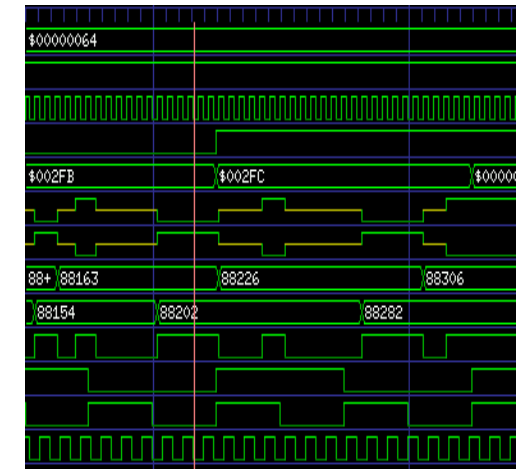
## Functional LT Model (VDK)

- + Good for SW development
- + Simulations in minutes/hours
- + Trace Ops, Memory accesses
- Low Timing Accuracy

validate  
back-annotate

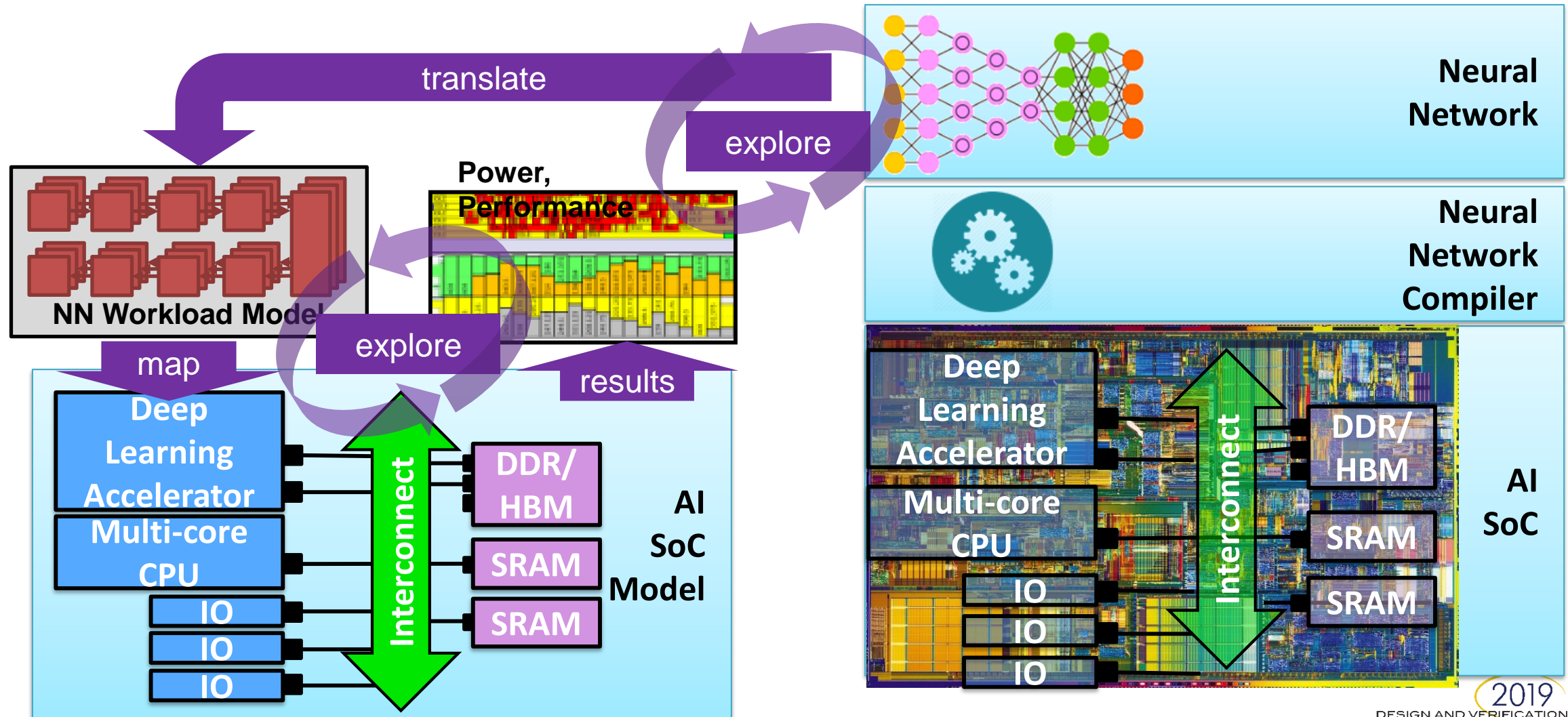
## RTL Simulation

- + Perfect accuracy
- High computational needs
- High turn-around costs





# Shift Left Architecture Analysis and Optimization

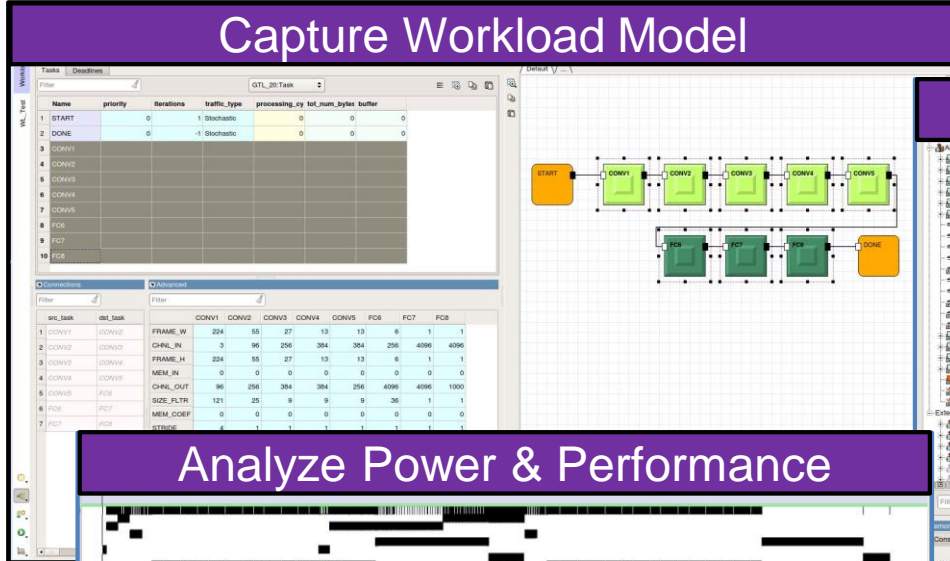




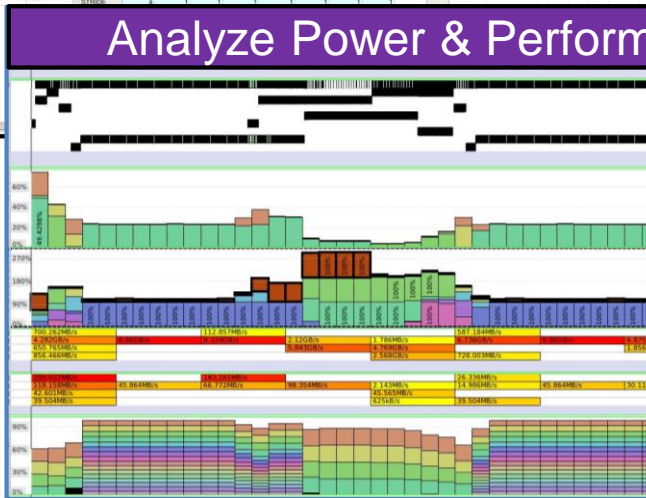
# Platform Architect Ultra

Providing a Comprehensive Library of Generic and Vendor Specific Models

## Capture Workload Model



## Analyze Power & Performance

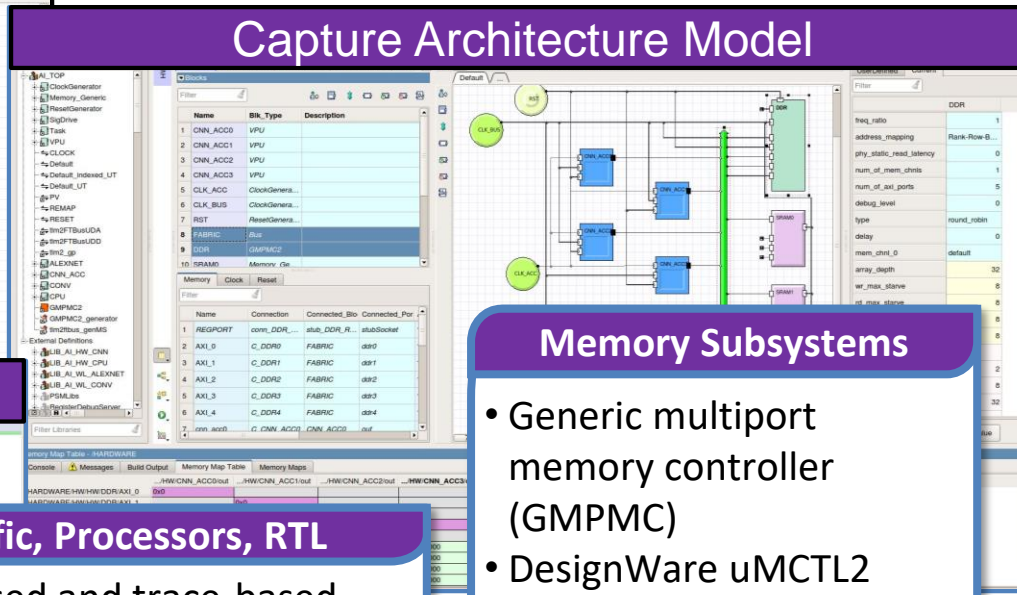


## Traffic, Processors, RTL

- Task-based and trace-based workload models
- Cycle accurate processor for ARM, ARC, Tensilica, CEVA
- RTL Co-simulation/emulation



## Capture Architecture Model



## Memory Subsystems

- Generic multiport memory controller (GMPMC)
- DesignWare uMCTL2 memory controller
- DesignWare LPDDR5 memory controller
- Co-simulate with RTL



## Interconnect Models

Generic:

- SBL-TLM2-FT (AXI)
- SBL-GCCI (ACE, CHI)

IP Specific:

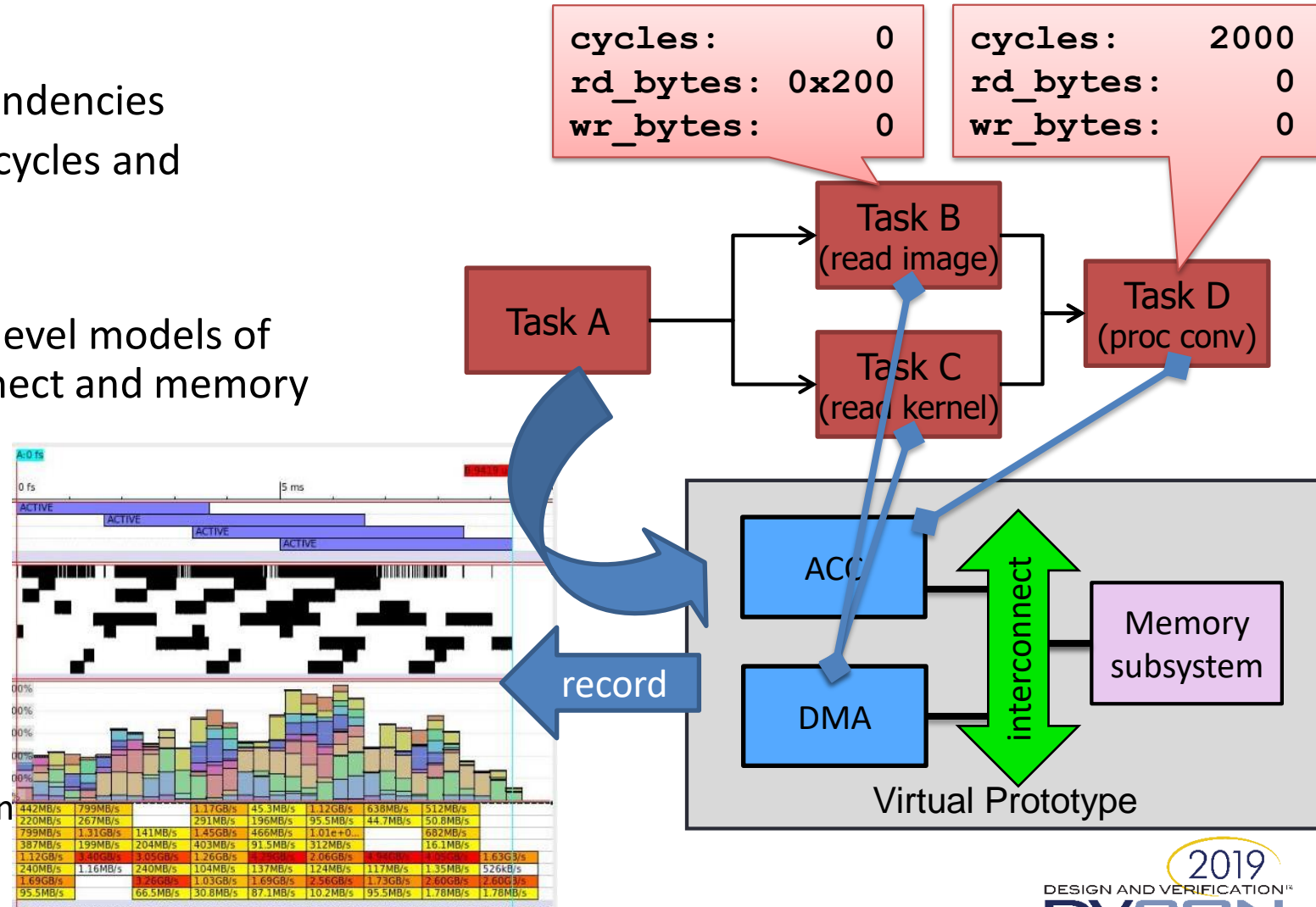
- Arteris FlexNoC & Ncore
- Arm AHB/APB
- Arm PL300
- Arm SBL-301
- Arm SBL-400
- Synopsys DW AXI





# Workload Modeling and Mapping

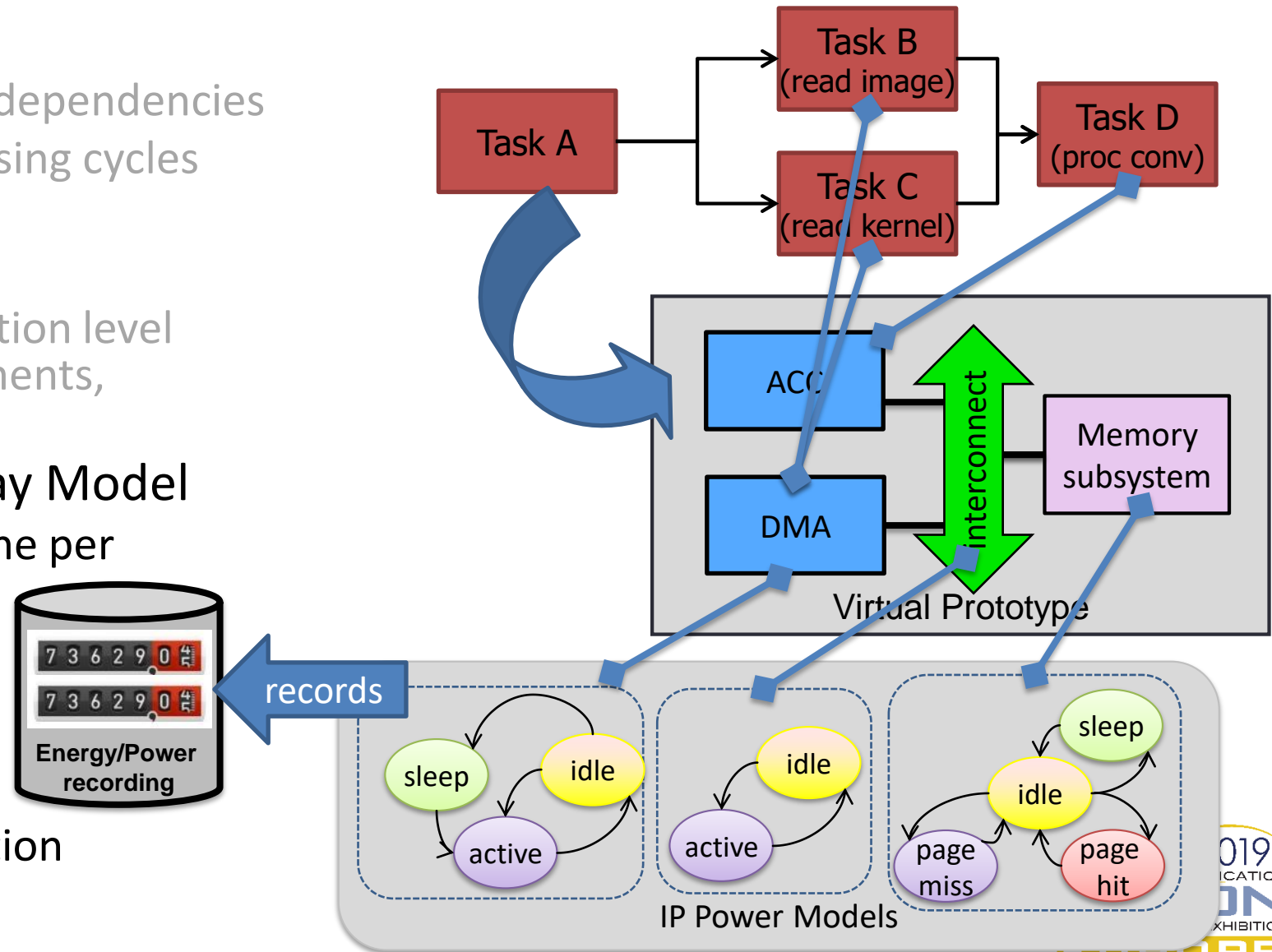
- Workload Model
  - Task level parallelism and dependencies
  - Characterized with processing cycles and memory accesses
- SoC Platform Model
  - Accurate SystemC Transaction level models of processing elements, interconnect and memory
- Map workload to platform
- Analyze performance metrics
  - End-to-end constraints
  - Workload activity
  - Utilization of resources
  - Interconnect metrics
    - Latency, Throughput, Contention
    - Outstanding transactions
    - ...





# System Level Power Modeling

- Workload Model
  - Task level parallelism and dependencies
  - Characterized with processing cycles and memory accesses
- SoC Platform Model
  - Accurate SystemC Transaction level models of processing elements, interconnect and memory
- System-level Power Overlay Model
  - Define power state machine per component
  - Bind power models to Virtual Prototype
  - Measure power and performance based on real activity and utilization

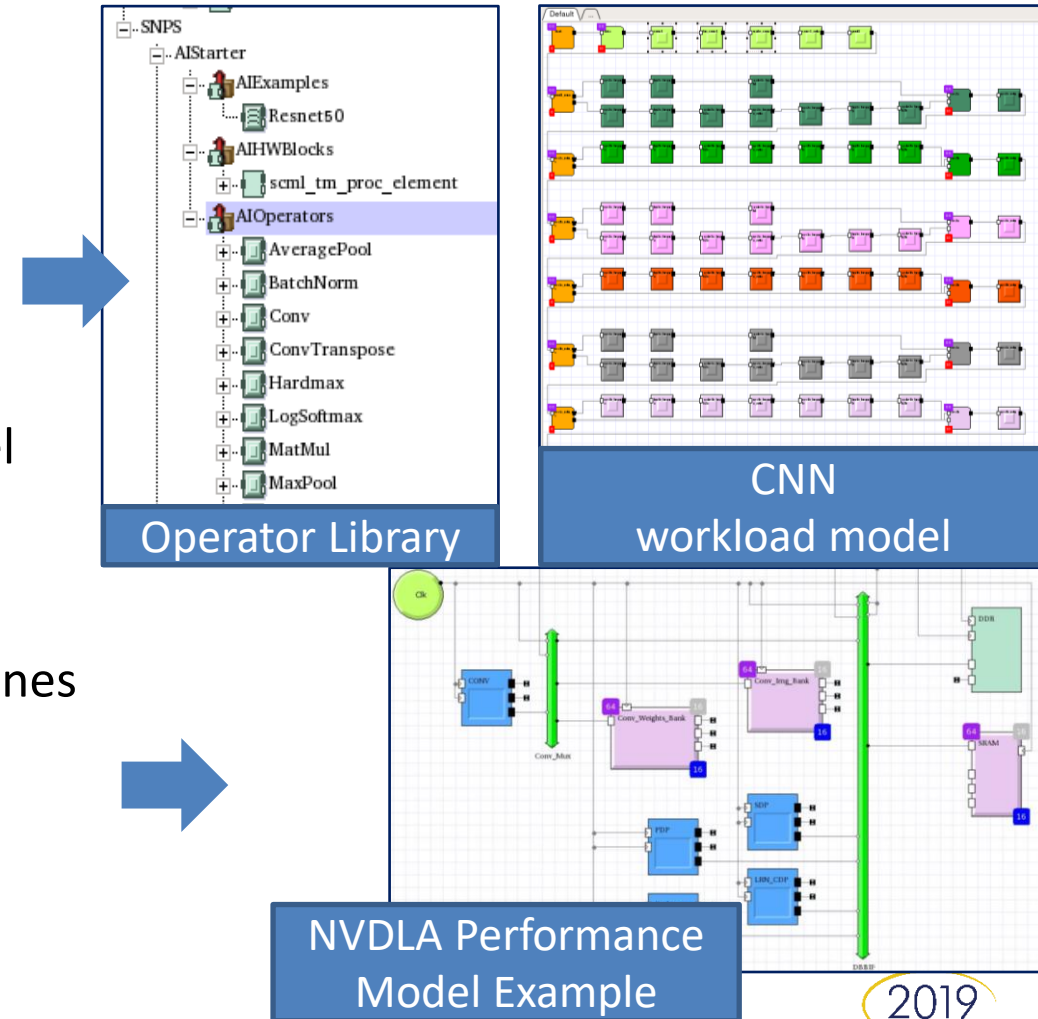




# Platform Architect Ultra AI Exploration Pack (XP)

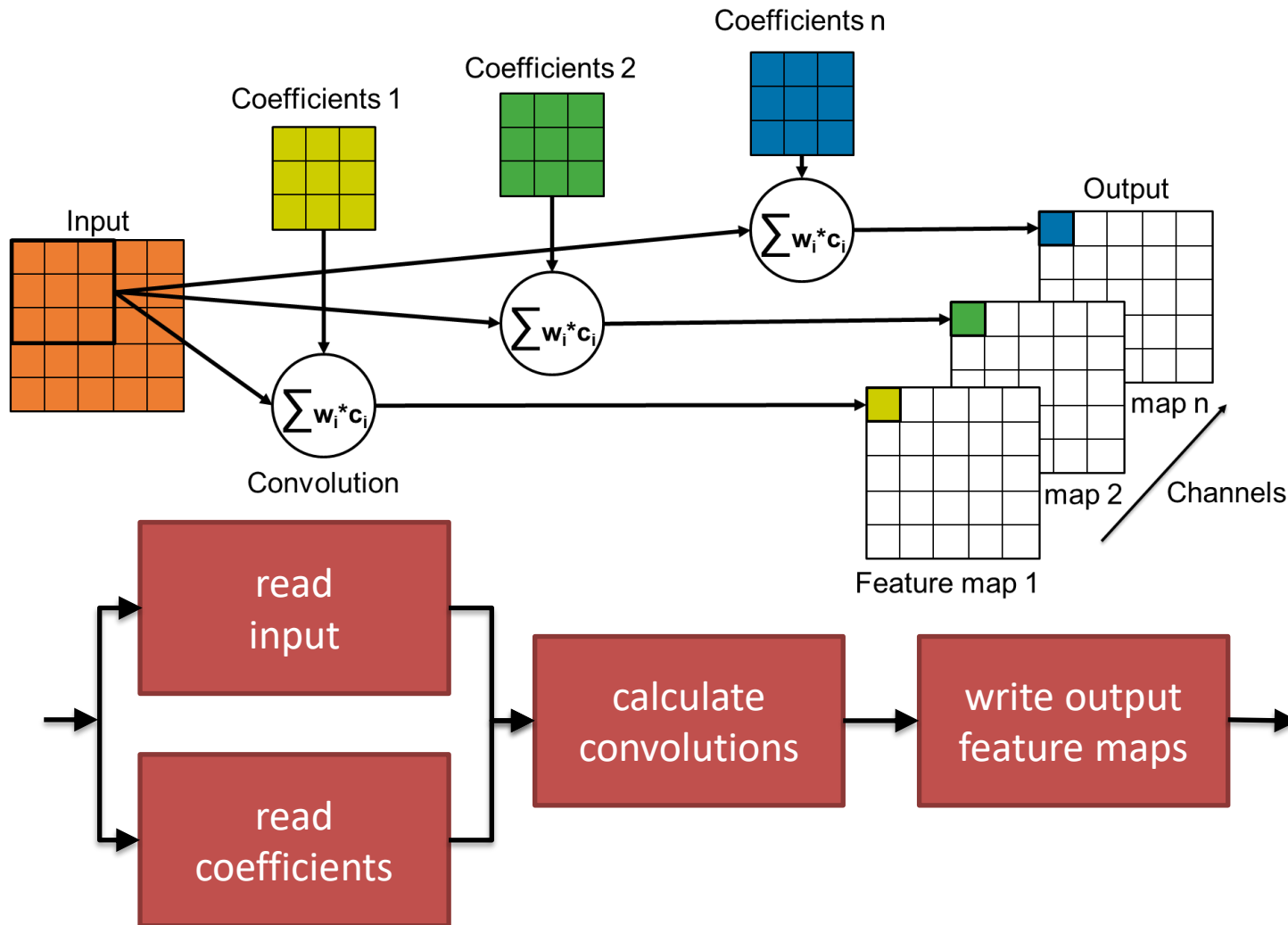
Exploration & optimization of AI designs

- Automated generation of workloads from AI frameworks
  - AI Operator Library for Neural Network modeling
    - E.g. Convolution, Matmul, MaxPool, BatchNorm etc.
  - Example workload model of ResNet50 Neural Network
  - Utility to convert prototxt description to workload model using AI operator library
- AI centric HW architecture model library
  - VPUUs configured to represent AI compute and DMA engines
  - Interconnect and memory subsystem models
  - Example performance model of NVIDIA Deep Learning Accelerator (NVDLA)
- AI centric analysis views: memory + processing utilization





# Workload Model of One Convolution Layer



AI algorithm params

in_img_h	32
in_img_w	32
num_in_chnls	3
in_kernel_h	7
in_kernel_w	7
num_filters	64
padding_h	3
padding_w	3
stride_h	2
stride_w	2

Mapping params

in_img_buff_base	0
in_img_buff_size	4096
in_kernel_buff_base	4096
in_kernel_buff_size	1024
out_feature_buff_base	0
out_feature_buff_size	4096
in_img_fetch_scaling	1
in_kernel_fetch_scaling	1
out_feature_scaling	1
num_ops_scaling	1

in_img_fetch_bytes	3072
in_kernel_fetch_bytes	9408
num_ops	4800512
out_feature_bytes	16384

Scaling parameters reflect the DLA architecture – can be taken from analytical model.

Workload params



# Agenda

- Deep Learning Market and Technology Trends
- How to Design a Deep Learning Accelerator (DLA)
  - Analytical Performance Modeling
  - Shift Left Architecture Analysis and Optimization with Virtual Prototyping

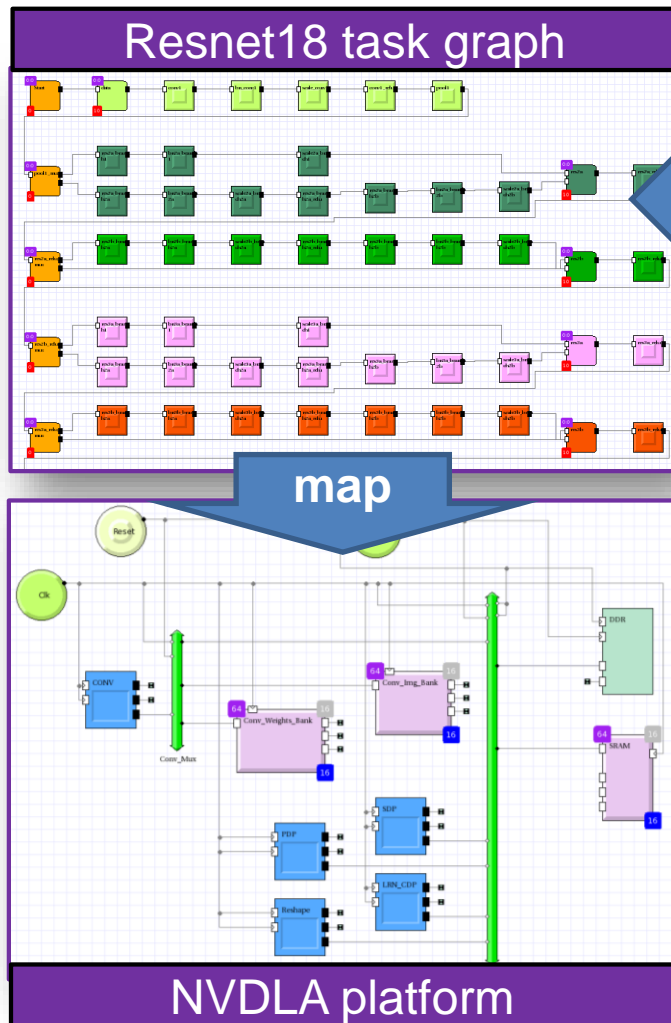


## Example

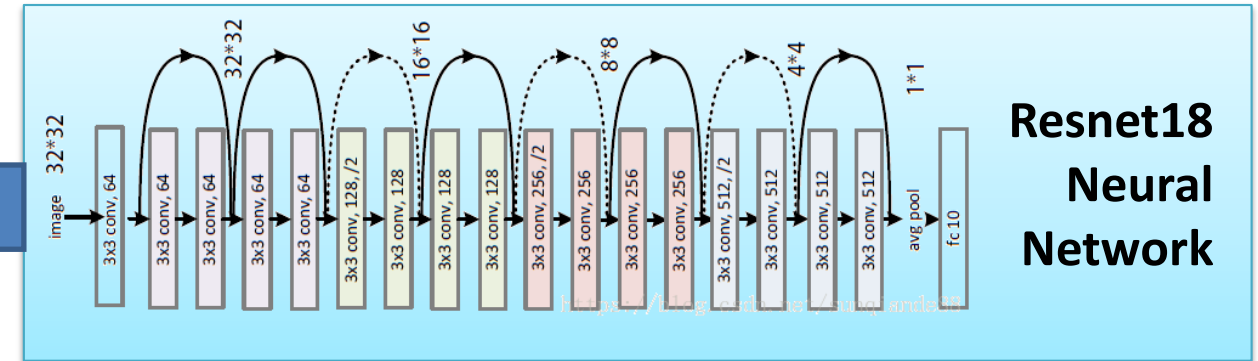
- Importing Network Algorithms as prototxt + generate analytical model spreadsheet
- Find suited configuration and scaling parameters in analytical model
- Validate first results, and explore architecture for dynamic and power aspects using Virtual Platforms
- Summary



## Example: Resnet-18 (Inference) with NV-DLA



## Import prototxt



## Goals:

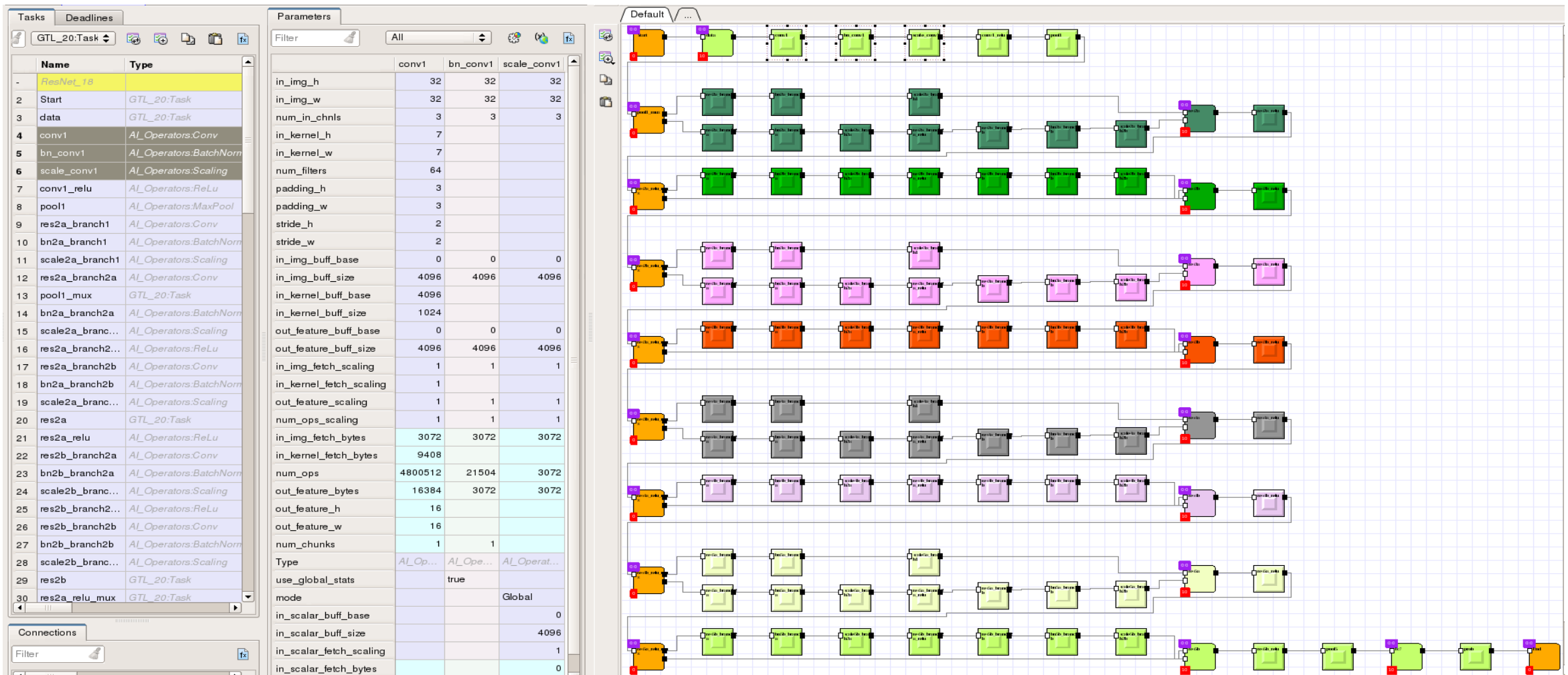
- ① 100 ms latency, ② minimize power, ③ minimize energy

## Optimize Hardware configuration:

- SIMD width
- Burst size, outstanding transactions
- speed of DDR memory and of data path

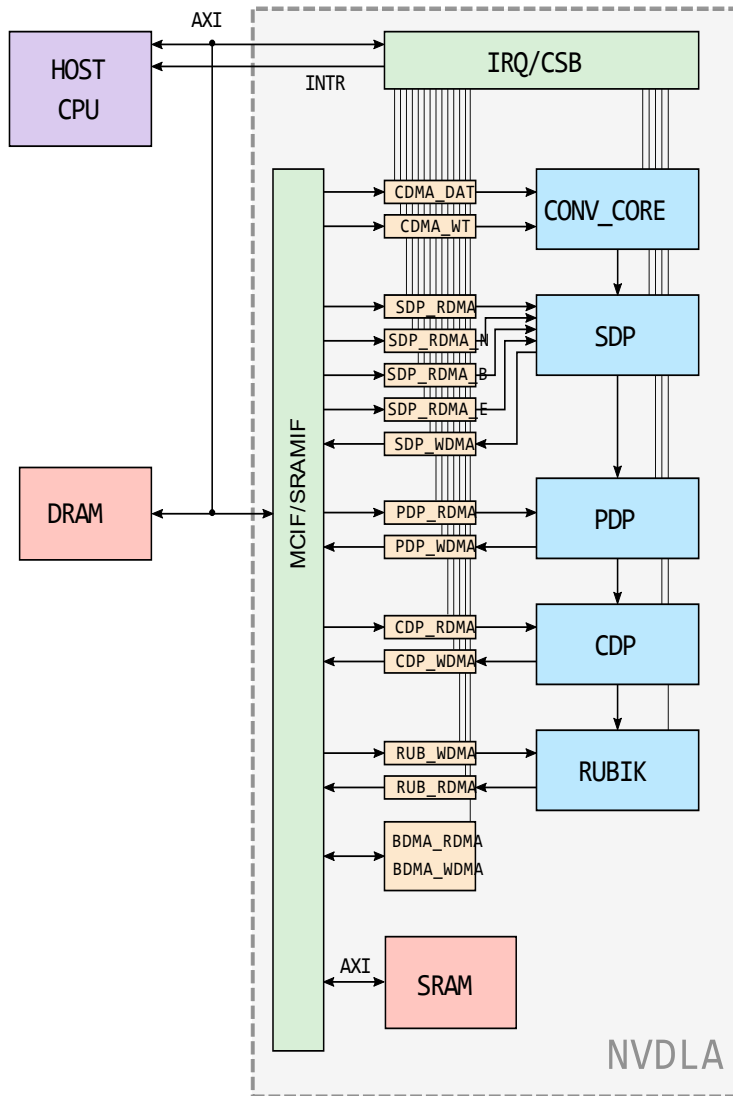


# ResNet-18 Workload model generated with AI-XP





# Example: Brief Overview of NVDLA



## Convolution Engine (CONV\_CORE)

- Works on two sets of data: offline-trained *kernels* (weights) and input *features* (images)
- configurable MAC units and convolutional buffer (RAM)
- Executes operations such as `tf.nn.conv2d`

## Single Data Point Processor (SDP)

- Applies linear and non-linear (activation) functions onto individual data points.
- Executes e.g. `tf.nn.batch_normalization`, `tf.nn.bias_add`, `tf.nn.elu`, `tf.nn.relu`, `tf.sigmoid`, `tf.tanh`, and more.

## Planar Data Processor (PDP)

- Applies common CNN spatial operations such as min/max/avg pooling
- Executes e.g. `tf.nn.avg_pool`, `tf.nn.max_pool`, `tf.nn.pool`.

## Cross-channel Data Processor (CDP)

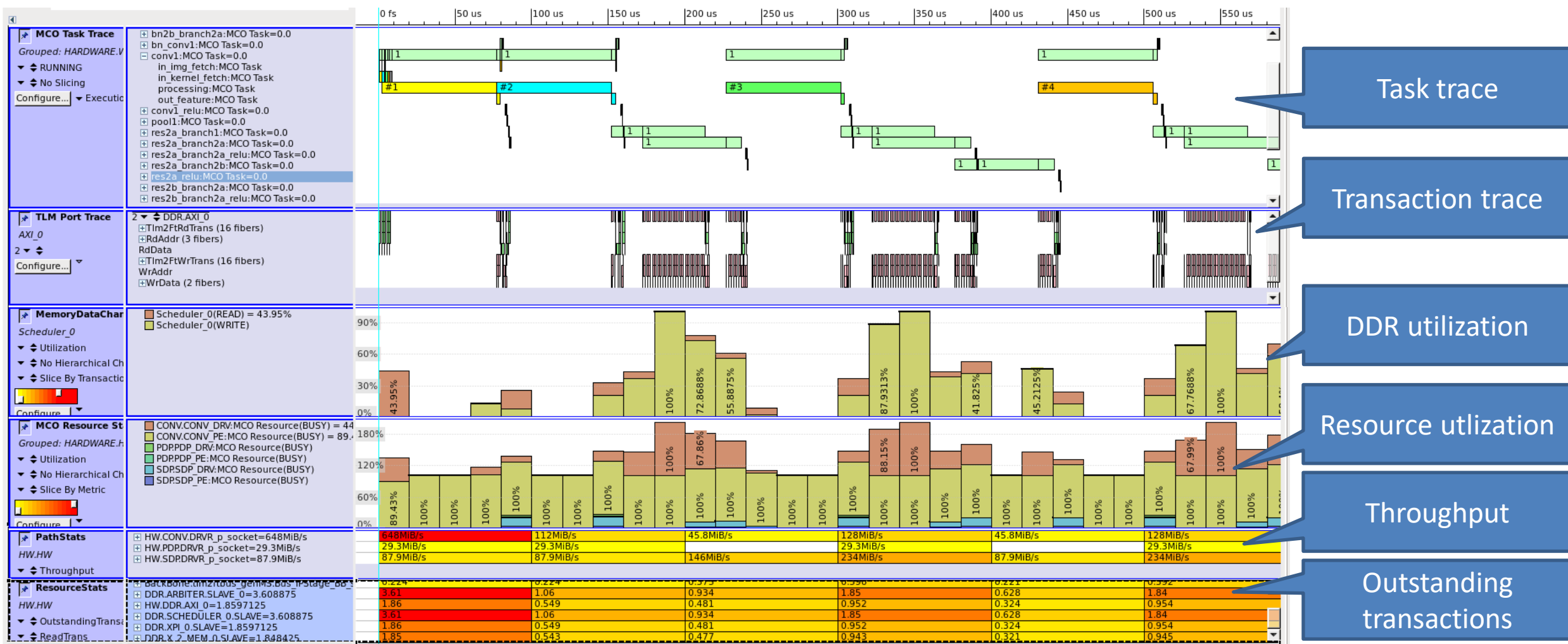
- Processes data from different channels/features, e.g. local response normalization (LRN) function
- Executes e.g. `tf.nn.local_response_normalization`

## Data Reshape Engine (RUBIK)

- Performs data format transformations (splitting, slicing, merging, ...)
- Executes e.g. `tf.nn.conv2d_transpose`, `tf.concat`, `tf.slice`, etc.



# VP Simulation Results of Initial Configuration



## Performance limited by processing, use wider SIMD data path



# Simulation Reveals Implementation Effects... (1)

Differences between calculated and measured data read/write amount

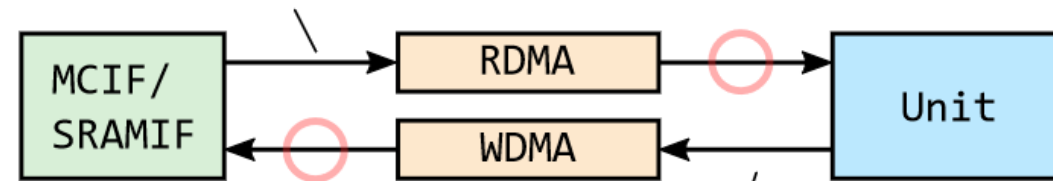
AlexNet (Norm1):

Expected: 580,800 Bytes

Measured: 654,720 Bytes

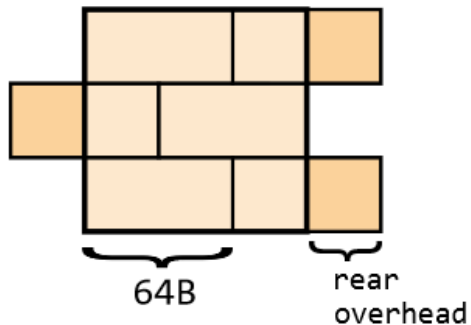
Inflation by ~12.72%

AXI\_TRANSACTION\_ATOM\_SIZE=64

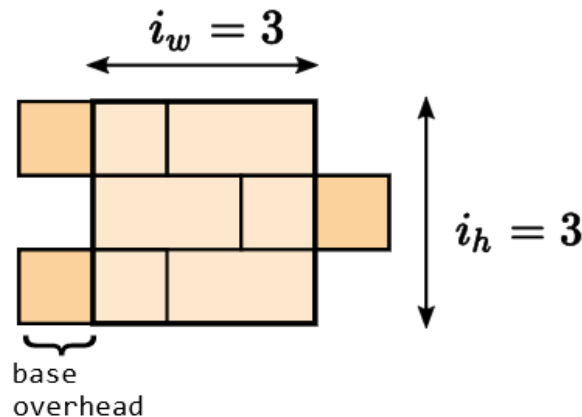


DMA\_TRANSACTION\_ATOM\_SIZE=32

Surface 0  
Addr: 0x00



Surface 1  
Addr: 0x120

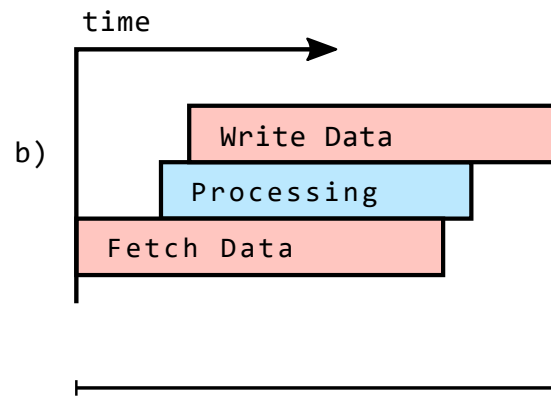
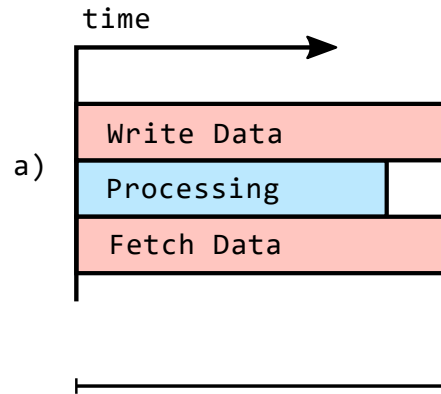


⇒ "Dark Bandwidth"

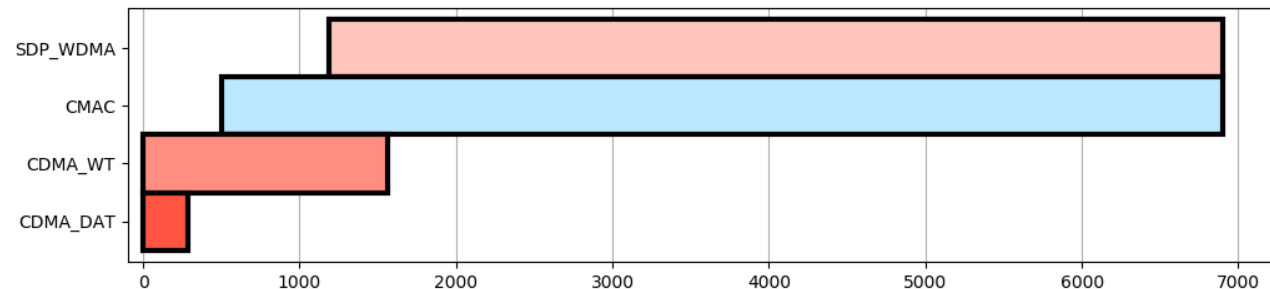
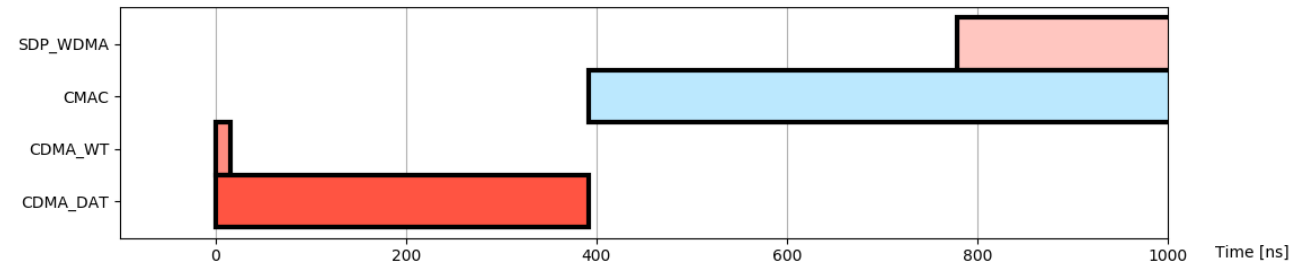


# Simulation Reveals Implementation Effects... (2)

Differences between calculated and measured execution time



Convolutional Layers 1&2 of LeNet on NVDLA



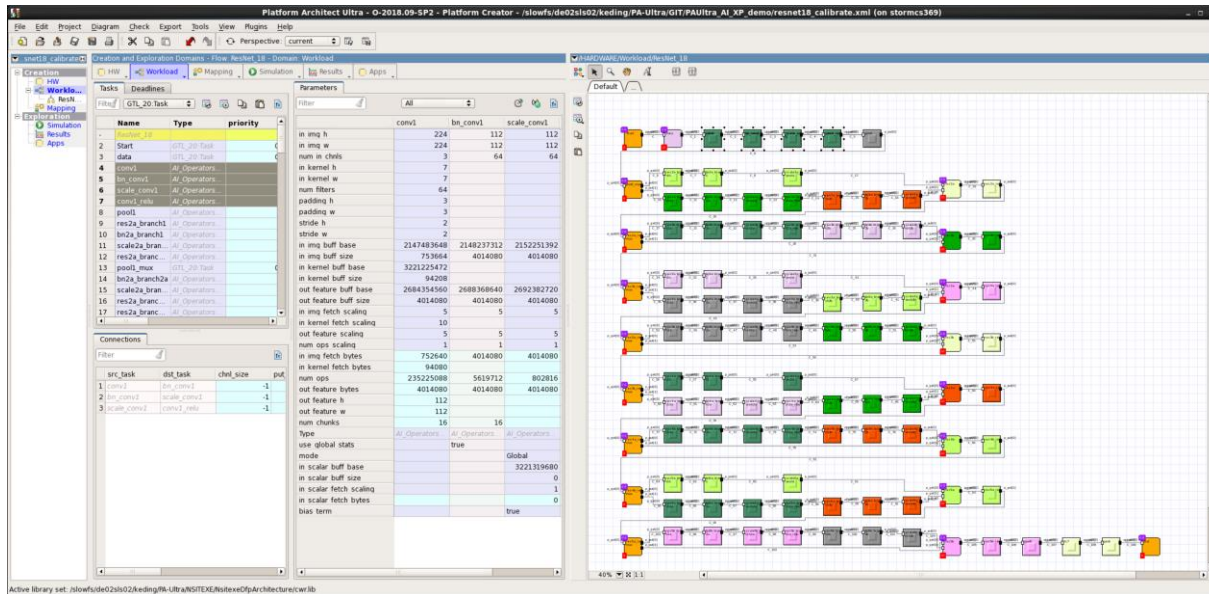


# Back-Annotate Simulation Findings To Analytical Model

Caffe .prototxt



Platform Architect / Simulation Model



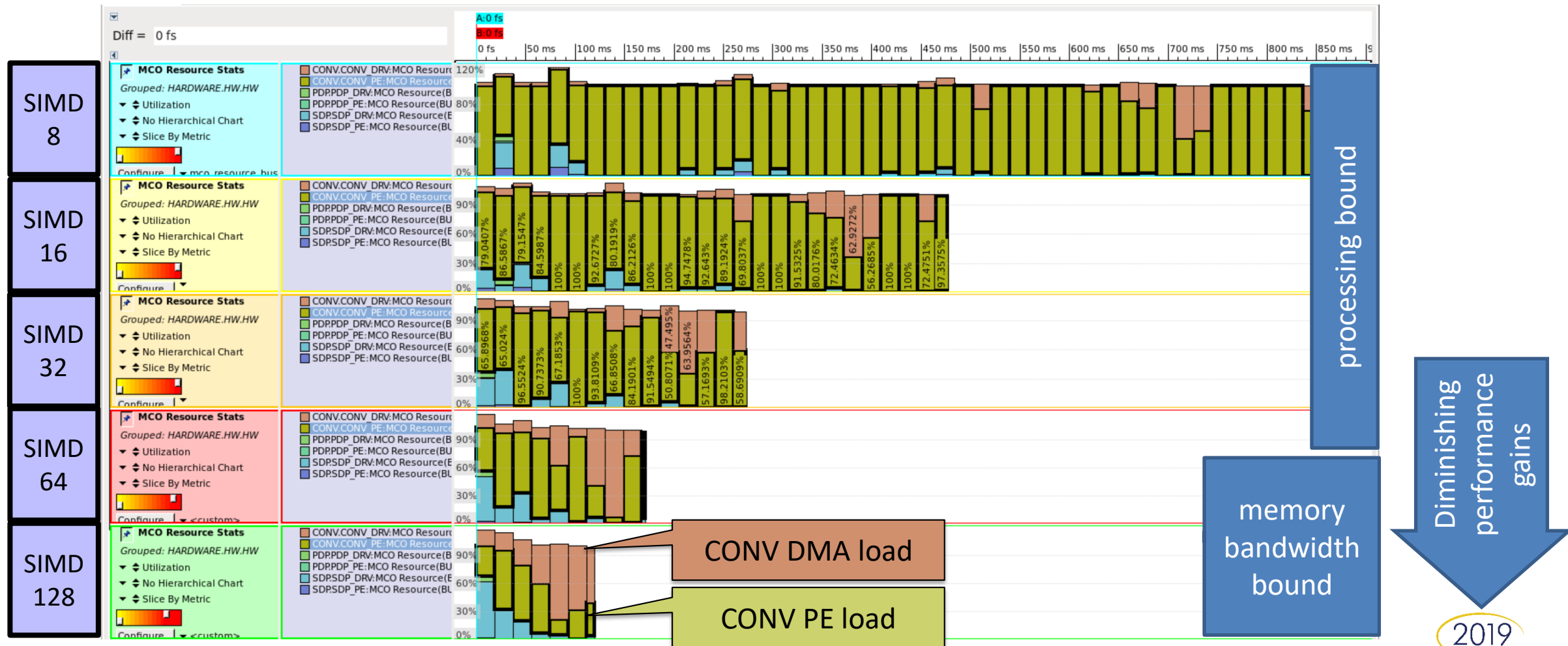
Spreadsheet / Analytical Model

num	depth	MAC	data_type	data_type	AXI4	DMA	Clock	Memory	Bandwidth	SOP	element	output	output	output	output	Needs	MAC	MAC			
cell	cell	Units	per	weight	CM	TCM	speed	bandwidth	utilization	element	size	height	width	channel	surrounding	tiling	operations	model			
15	64	1024	2	2	64	32	1000	12.8	1	16	512										
Layer	Type	image_w	image_h	image_c	kernels	kernel_w	kernel_h	stride_w	stride_h	zero_pad	pad_h	pad_w	group	bias	output_height	output_width	output_channel	surrounding	Needs tiling	MAC operations	MAC model
Conv1	CONV	227	58	3	96	11	11	4	4	FALSE	0	0	1	TRUE	12	55	96	1			
Conv1	CONV	227	58	3	96	11	11	4	4	FALSE	0	0	1	TRUE	12	55	96	1			
Conv1	CONV	227	58	3	96	11	11	4	4	FALSE	0	0	1	TRUE	12	55	96	1			
Conv1	CONV	227	58	3	96	11	11	4	4	FALSE	0	0	1	TRUE	12	55	96	1			
Conv1	CONV	227	35	3	96	11	11	4	4	FALSE	0	0	1	TRUE	7	55	96	1			
Conv1	CONV	227	227	3	96	11	11	4	4	FALSE	0	0	1	TRUE	55	55	96	1	TRUE	105415200	2196150
Relu1	RELU	55	55	96						FALSE	0	0	1		55	55	96	6	FALSE	0	0
Norm1	NORM	55	55	96							0	0			55	55	96	6	FALSE	0	0
Pool1	POOL	55	55	96	1	3	3	2	2	FALSE	0	0			27	27	96	6	FALSE	0	0
Conv2	CONV	27	27	96	256	5	5	1	1	TRUE	2	2	2	TRUE	27	27	256	6	FALSE	223948800	291600
Relu2	RELU	27	27	256							0	0			27	27	256	16	FALSE	0	0
Norm2	NORM	27	27	256							0	0			27	27	256	16	FALSE	0	0
Pool2	POOL	27	27	256	1	3	3	2	2	FALSE	0	0			13	13	256	16	FALSE	0	0
Conv3	CONV	13	13	256	384	3	3	1	1	TRUE	1	1	1	TRUE	13	13	384	16	FALSE	149520384	146016
Relu3	RELU	13	13	384							0	0			13	13	384	24	FALSE	0	0
Conv4	CONV	13	13	384	384	3	3	1	1	TRUE	1	1	2	TRUE	13	13	384	24	FALSE	112140288	109512
Relu4	RELU	13	13	384							0	0			13	13	384	24	FALSE	0	0
Conv5	CONV	13	13	384	256	3	3	1	1	TRUE	1	1	2	TRUE	13	13	256	24	FALSE	74760192	73008
Relu5	RELU	13	13	256							0	0			13	13	256	16	FALSE	0	0
Pool5	POOL	13	13	256	1	3	3	2	2	FALSE	0	0			6	6	256	16	FALSE	0	0
IP6	IP	6	6	256	4096	6	6	1	1	FALSE	0	0	1	TRUE	1	1	4096	16	FALSE	0	36864
Relu6	RELU	1	1	4096							0	0			1	1	4096	256	FALSE	0	0
IP7	IP	1	1	4096	4096	1	1	1	1	FALSE	0	0	1	TRUE	1	1	4096	256	FALSE	0	16384
Relu7	RELU	1	1	4096							0	0			1	1	4096	256	FALSE	0	0
IP8	IP	1	1	4096	1000	1	1	1	1	FALSE	0	0	1	TRUE	1	1	1000	256	FALSE	0	4032
Total																					2873566



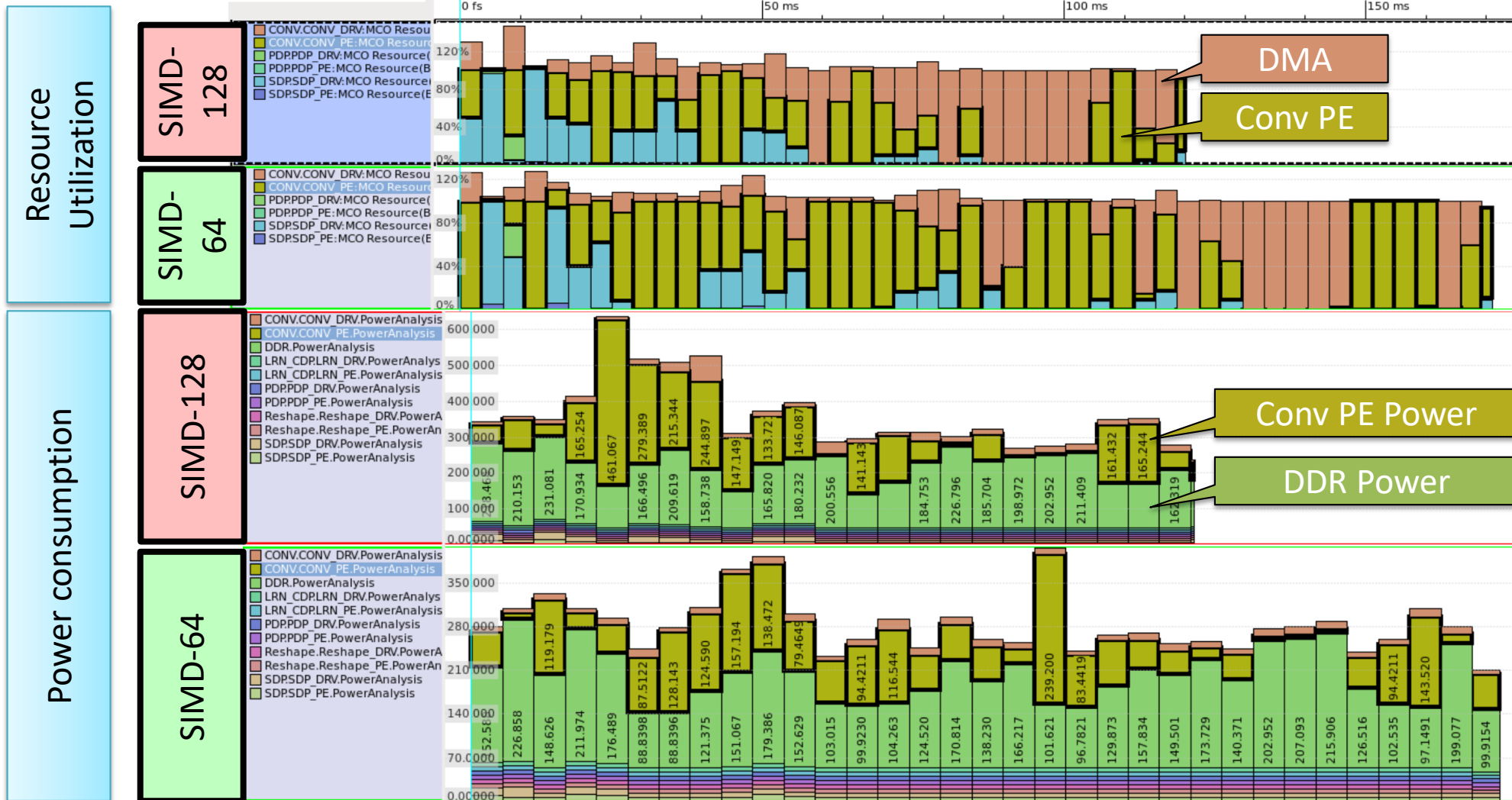
# Impact of SIMD Width on Performance

Resource Utilization of CONV Datapath (yellow), CONV DMA (red) and other components





# DDR Memory Bandwidth and Power Improvement



25% faster

10% lower total energy

20% lower average power



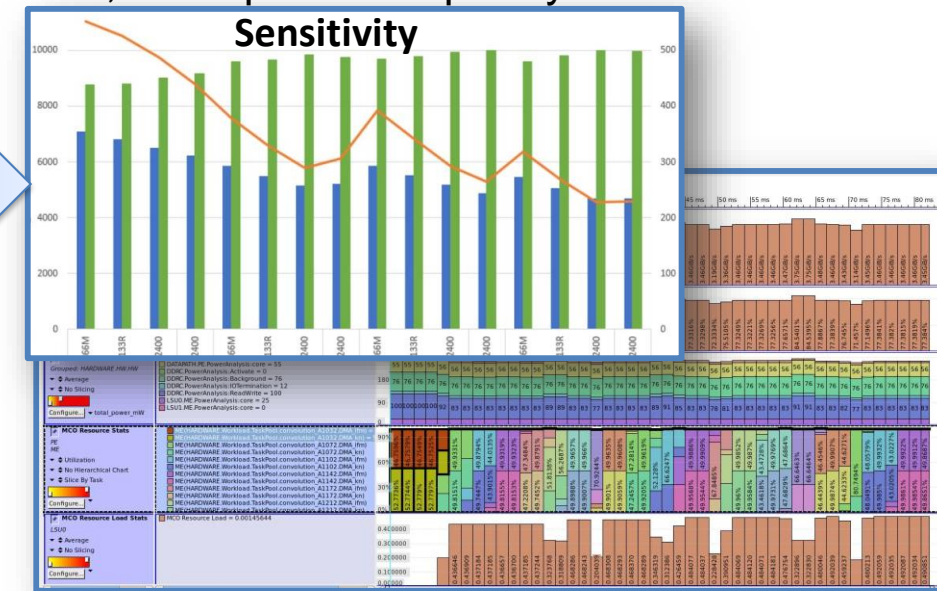
# Resnet 18 Example Sweep

Goal: 100 ms latency, minimize power & energy

	Name	simtime_us	...outstanding	...sz_in_bytes	...speed_bin	...Clk/period	...opc
23	run_b32_opc64_clk05_DDR4_1866M_os4	4416.966	4	32	DDR4-18...	0.5	64
24	run_b32_opc64_clk05_DDR4_1866M_os8	4235.459	8	32	DDR4-18...	0.5	64
25	run_b16_opc128_clk10_DDR4_2400_os4	5990.249	4	16	DDR4-2400	1	128
26	run_b16_opc128_clk10_DDR4_2400_os8	5657.129	8	16	DDR4-2400	1	128
27	run_b16_opc128_clk10_DDR4_1866M_os4	6994.128	4	16	DDR4-18...	1	128
28	run_b16_opc128_clk10_DDR4_1866M_os8	6676.136	8	16	DDR4-18...	1	128
29	run_b16_opc128_clk075_DDR4_2400_os4	5509.487	4	16	DDR4-2400	0.75	128
30	run_b16_opc128_clk075_DDR4_2400_os8	5189.458	8	16	DDR4-2400	0.75	128
31	run_b16_opc128_clk075_DDR4_1866M_os4	6485.257	4	16	DDR4-18...	0.75	128
32	run_b16_opc128_clk075_DDR4_1866M_os8	6207.377	8	16	DDR4-18...	0.75	128
33	run_b16_opc128_clk05_DDR4_2400_os4	5141.269	4	16	DDR4-2400	0.5	128
34	run_b16_opc128_clk05_DDR4_2400_os8	4797.354	8	16	DDR4-2400	0.5	128
35	run_b16_opc128_clk05_DDR4_1866M_os4	6402.813	4	16	DDR4-18...	0.5	128
36	run_b16_opc128_clk05_DDR4_1866M_os8	6080.660	8	16	DDR4-18...	0.5	128
37	run_b32_opc128_clk10_DDR4_2400_os4	4228.984	4	32	DDR4-2400	1	128
38	run_b32_opc128_clk10_DDR4_2400_os8	4066.654	8	32	DDR4-2400	1	128
39	run_b32_opc128_clk10_DDR4_1866M_os4	4444.511	4	32	DDR4-18...	1	128
40	run_b32_opc128_clk10_DDR4_1866M_os8	4236.462	8	32	DDR4-18...	1	128
41	run_b32_opc128_clk075_DDR4_2400_os4	3464.214	4	32	DDR4-2400	0.75	128
42	run_b32_opc128_clk075_DDR4_2400_os8	3261.505	8	32	DDR4-2400	0.75	128
43	run_b32_opc128_clk075_DDR4_1866M_os4	3963.731	4	32	DDR4-18...	0.75	128
44	run_b32_opc128_clk075_DDR4_1866M_os8	3769.144	8	32	DDR4-18...	0.75	128
45	run_b32_opc128_clk05_DDR4_2400_os4	2981.385	4	32	DDR4-2400	0.5	128
46	run_b32_opc128_clk05_DDR4_2400_os8	2795.655	8	32	DDR4-2400	0.5	128
47	run_b32_opc128_clk05_DDR4_1866M_os4	3483.204	4	32	DDR4-18...	0.5	128
48	run_b32_opc128_clk05_DDR4_1866M_os8	3301.767	8	32	DDR4-18...	0.5	128

## Sweep parameters

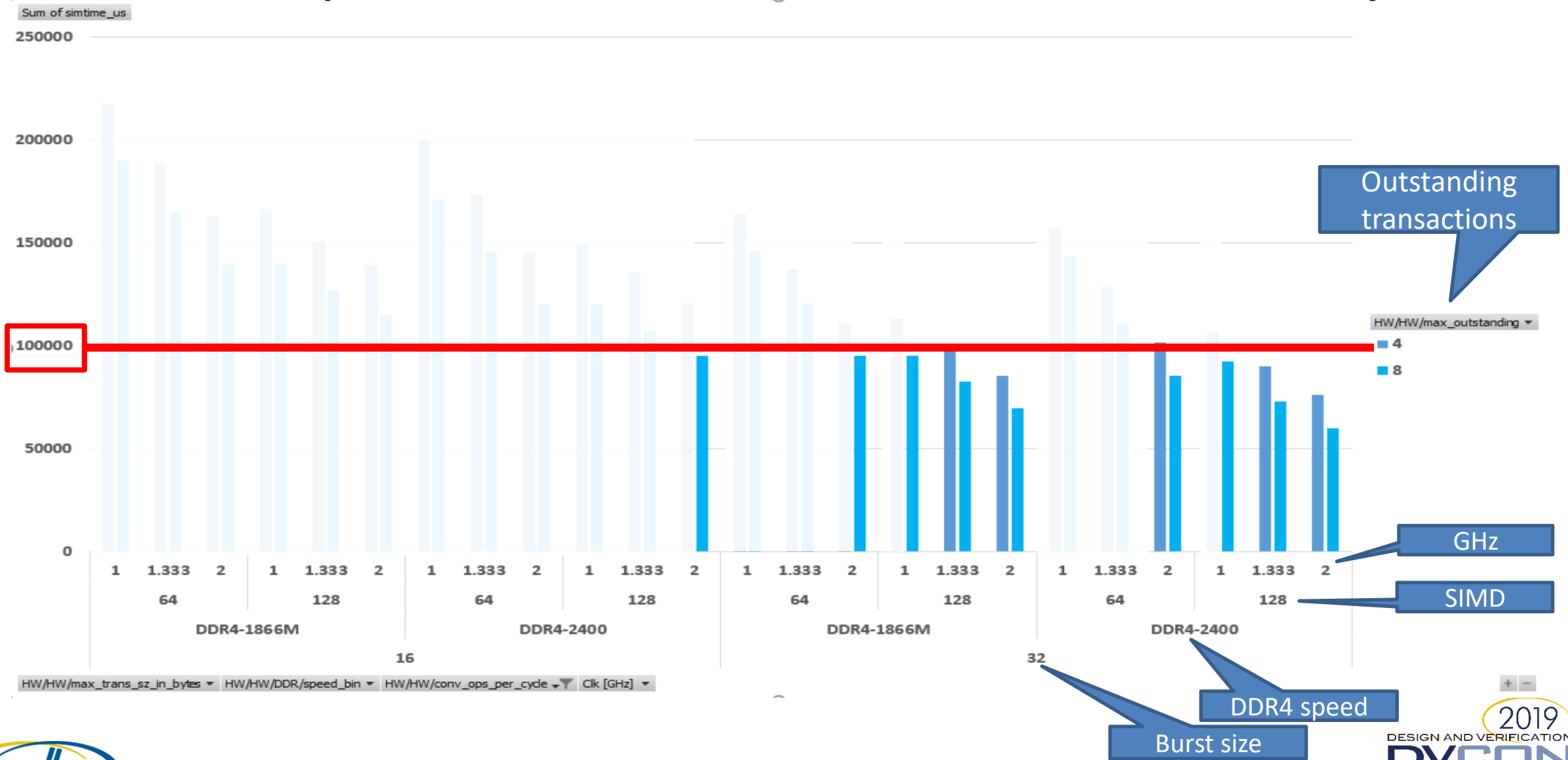
- Burst size: 16, 32
- Outstanding transactions: 4, 8
- DDR memory speed: DDR4-1866, DDR4-2400
- Clock frequency of data path: 1, 1.33, 2GHz
- SIMD width: 64, 128 operations per cycle



Root-Cause  
Analysis

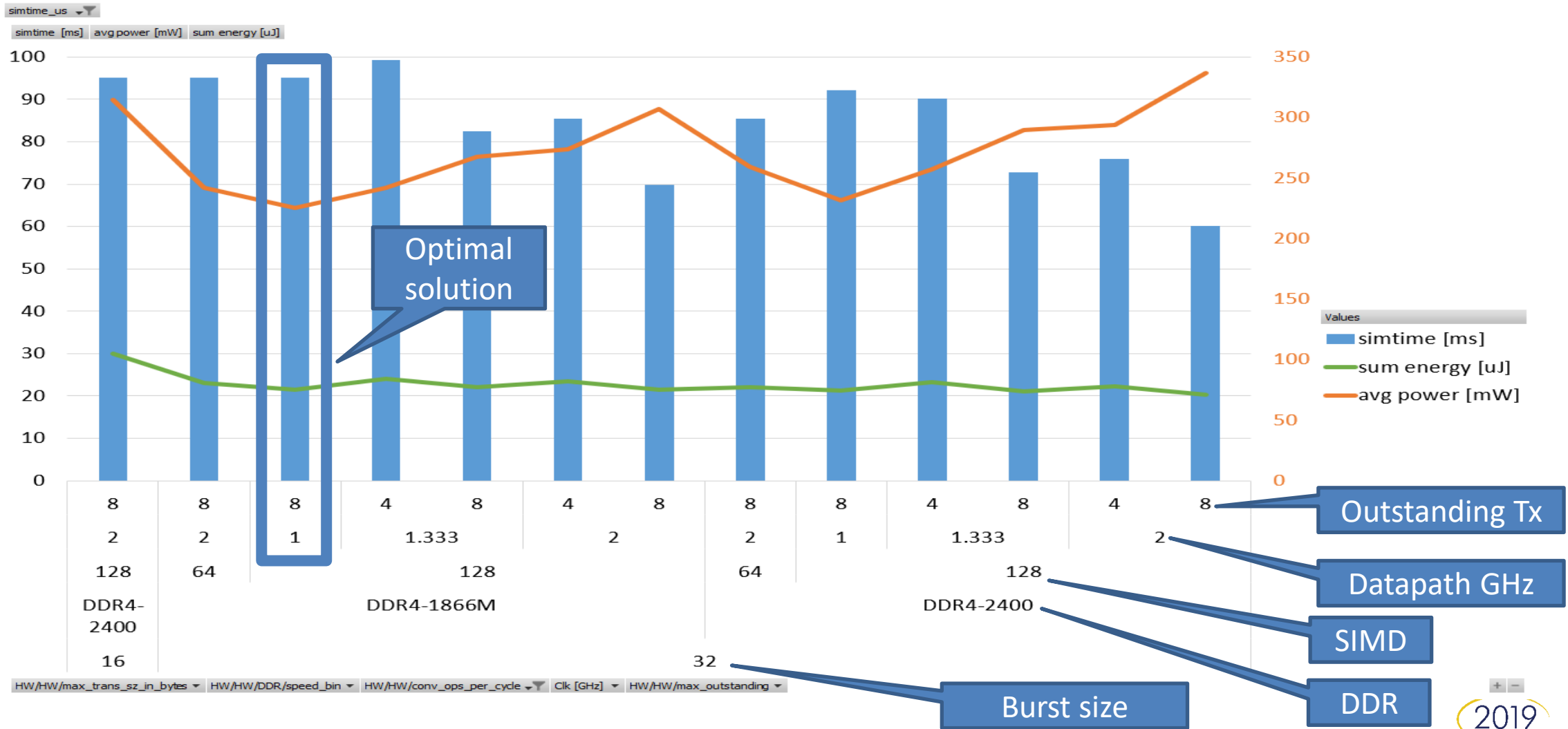


# Sweep Over Hardware Parameters, Latency



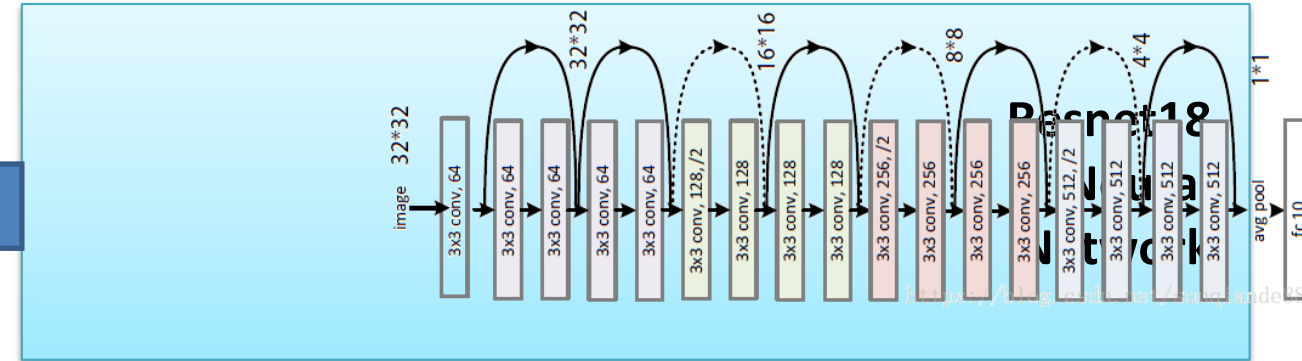
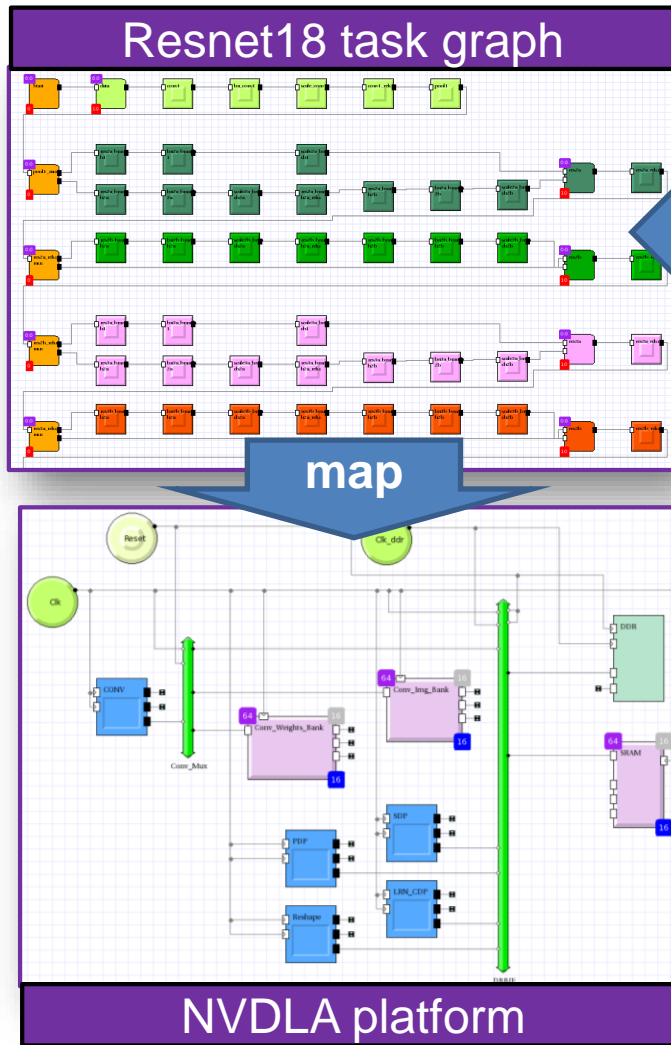


# Power/Performance/Energy Trade-off Analysis





# Example: Resnet-18 with NV-DLA



## Goal:

- 100 ms latency, minimize energy

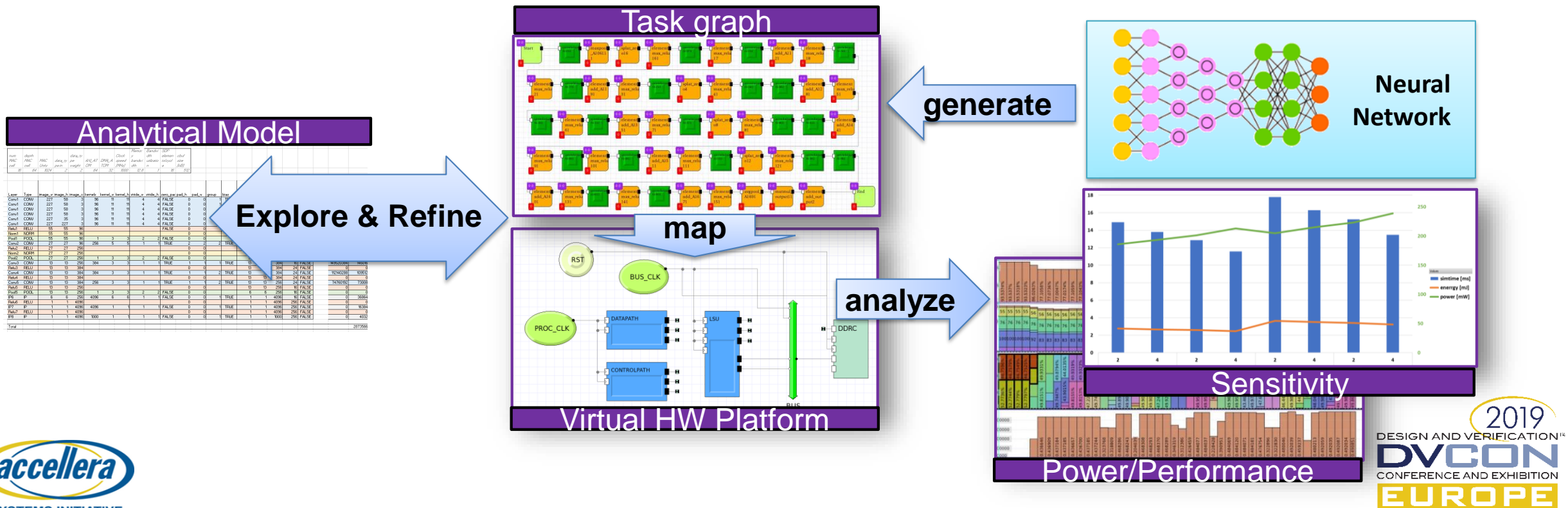
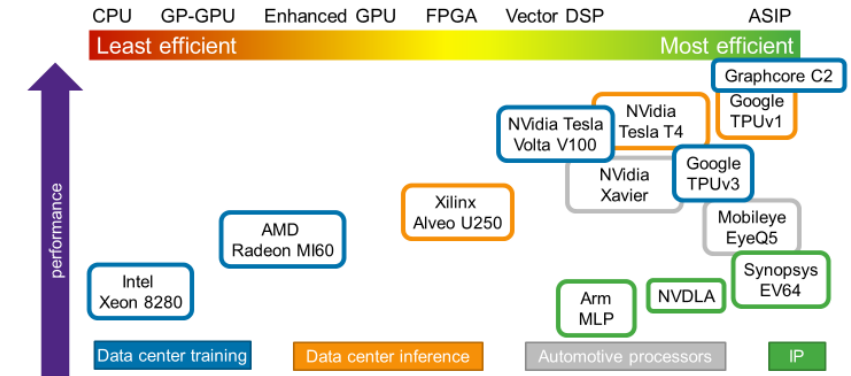
## Optimize Hardware configuration:

- SIMD width: 128 operations per cycle
- Burst size: 32 bytes
- outstanding transactions: 8
- speed of DDR memory: DDR4-1866
- speed of data path: 1GHz



# Summary

- Be fast and get it right!
- Shift Left with Virtual Prototyping
- Joint Optimization of Algorithm, Architecture, and Compiler





Thank You

Questions