Resource allocation in AI / CV applications

Rami Drucker, Neural Graph Compute Leader, Vision BU, CEVA

September 2020

CEVA®

Executive Summary
Abstract

Efficient edge computing is the foundation underlying AI and computer vision applications in autonomous vehicles, cameras, drones and many other applications

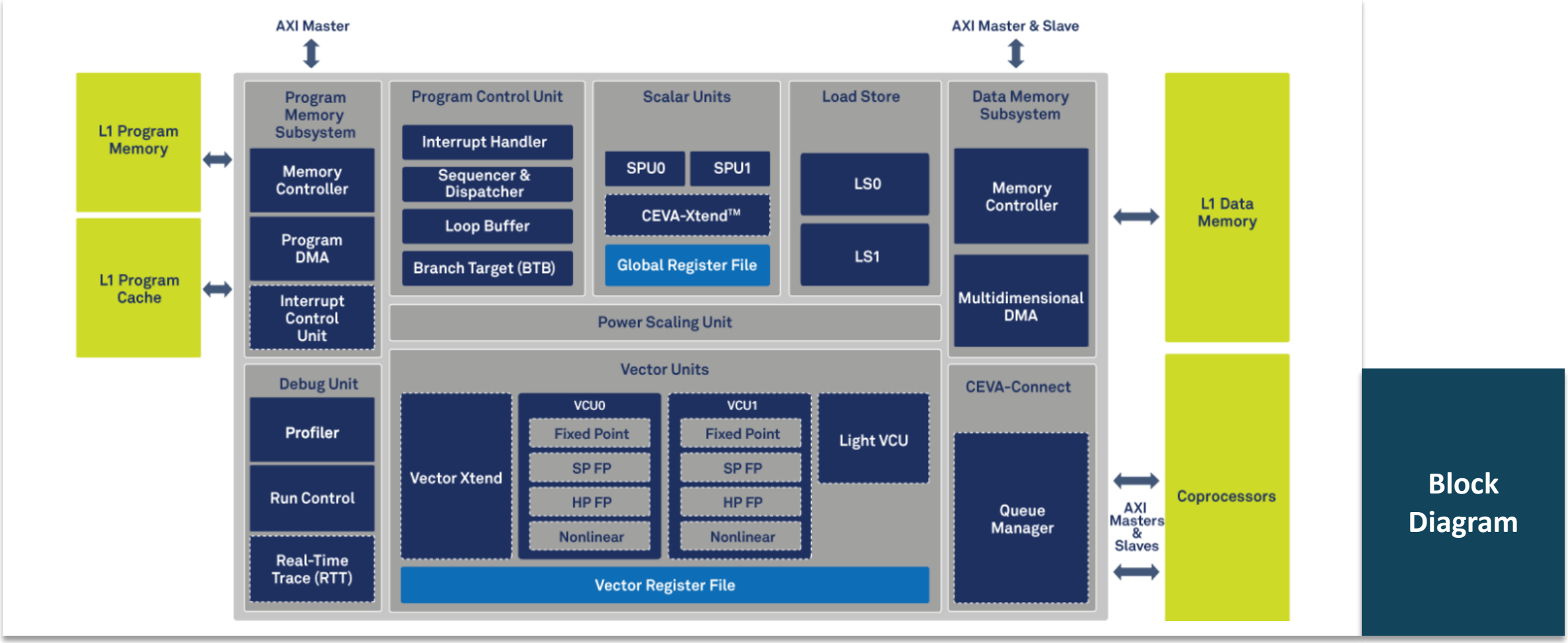
Because deep neural networks (DNNs) are memory and compute intensive, creating efficient implementations of DNNs requires frugal use of memory and memory bandwidth, along with computing power

In this talk, we'll present CEVA's novel approach to enable efficient memory allocation to enable implementing DNNs under strict size and power constraints

Our approach utilizes a unified computational graph and takes into account the differing characteristics of different classes of memory (on-chip L1 and L2 SRAM and external DDR)

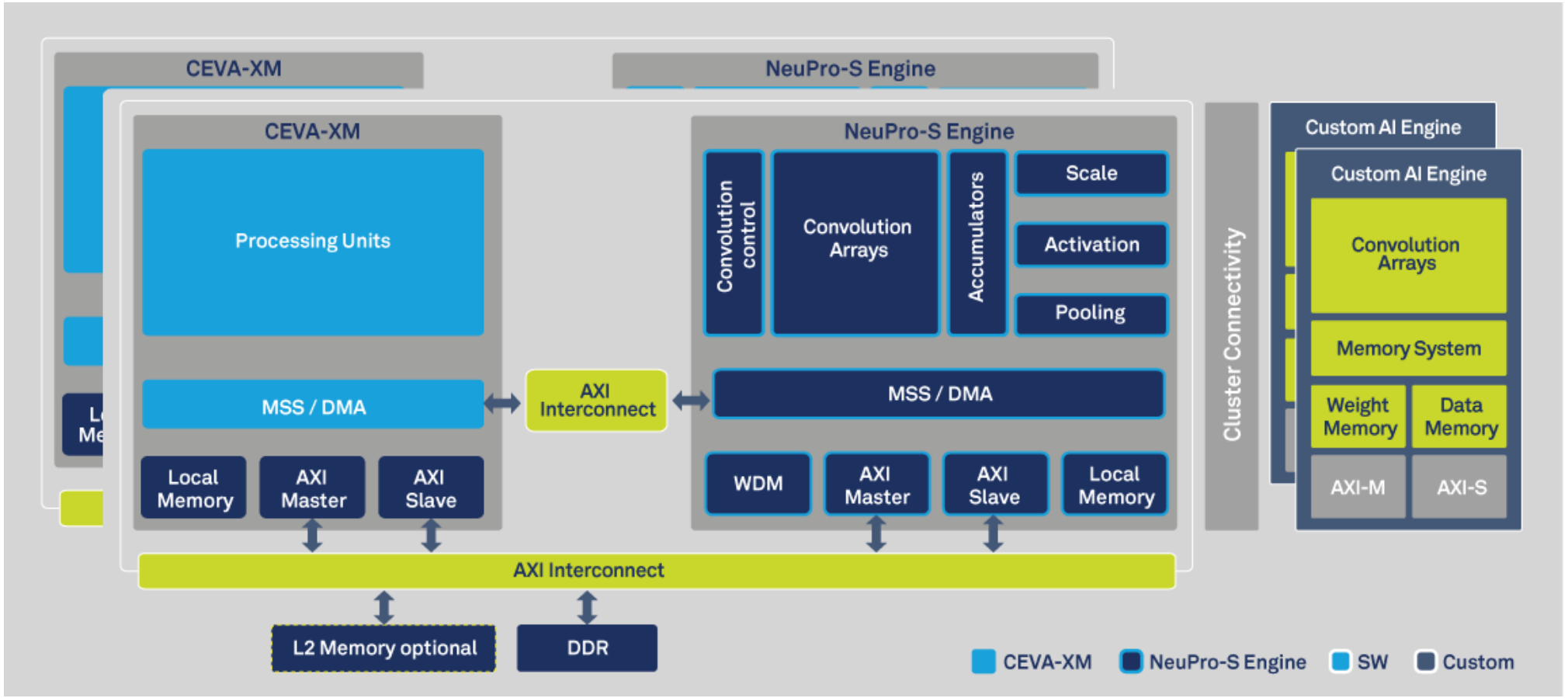
We will also introduce CEVA-XM6 and CEVA SensPro processors for vision and AI

CEVA SensPro Vision and AI processor



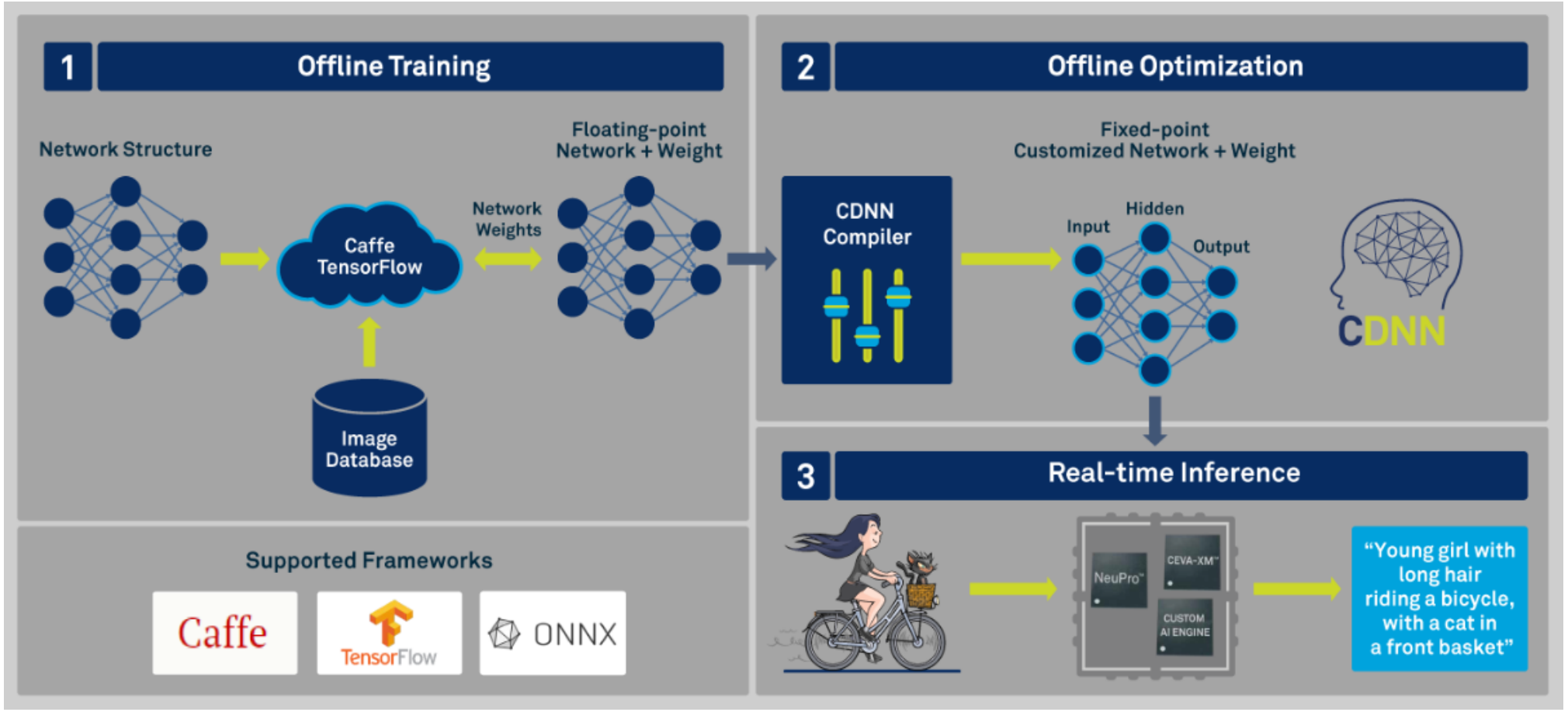
Block Diagram

CEVA NeuPro-S Vision and AI processor



System Block Diagram

CDNN Overview



CDNN

CDNN Usage Flow

Computational graph

- A neural network can be described by a computational graph
- The computational graph is a directed graph that has two types of nodes: data nodes and operation nodes
- A directed edge from a data node to an operation node means that the operation takes the data as its input, while a directed edge from an operation node to a data node means that the operation produces the data as its output

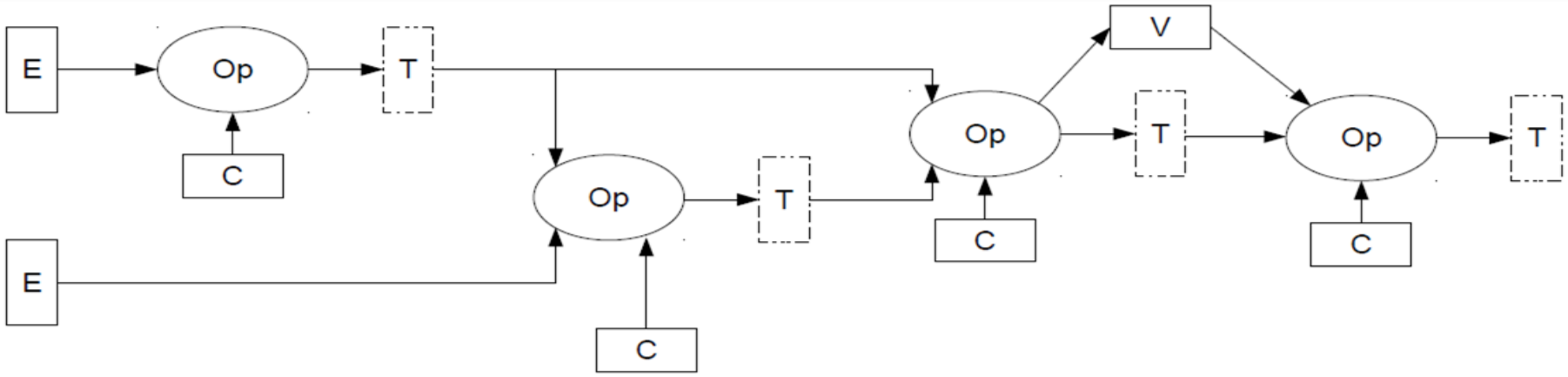
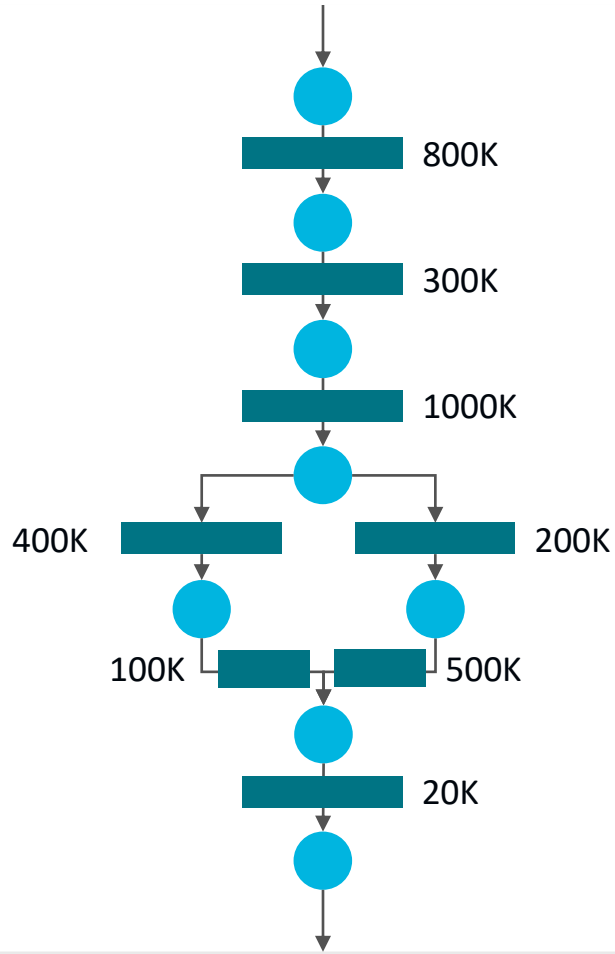


Figure 1. An example computational graph: Squares denote tensor data (E: external, C: constant, V: variable, T: regular tensor), ellipses denote operations

Memory allocation algorithm (1/3)

Problem Description	Goal	Constraints
• Given a computational graph G with vertices and edges (V,E) • Input: G=(V,E) • Each node might have several input and output buffers • Each buffer size is determined by the operation size	• Minimize overall network memory consumption • Maximize on-chip L1 and L2 SRAM BW • Minimize external DDR BW	• L1 and L2 memories are fast but small in size (typically 1MB – 4MB)



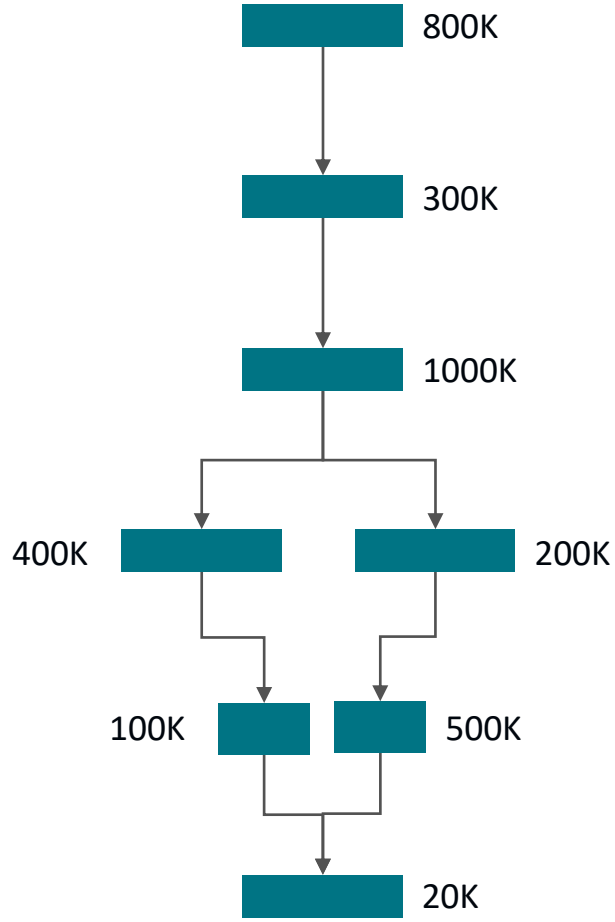
Memory allocation algorithm (2/3)

- Switch to a new graph in which the buffers are the nodes
- The 0-1 knapsack problem:
  - Given n items with weights w_i and costs p_i
  - Let the indicator function be: $x_i = \begin{cases} 1, & item\ i\ is\ chosen \\ 0, & otherwise \end{cases}$
  - Need to find a vector $\boldsymbol{x}$ which satisfies

$$(1) \quad \max_{\boldsymbol{x}} \sum_{i=1}^{n} p_i x_i$$

$$(2) \quad \sum_{i=1}^{n} w_i x_i < W$$

Where W is maximum weight capacity



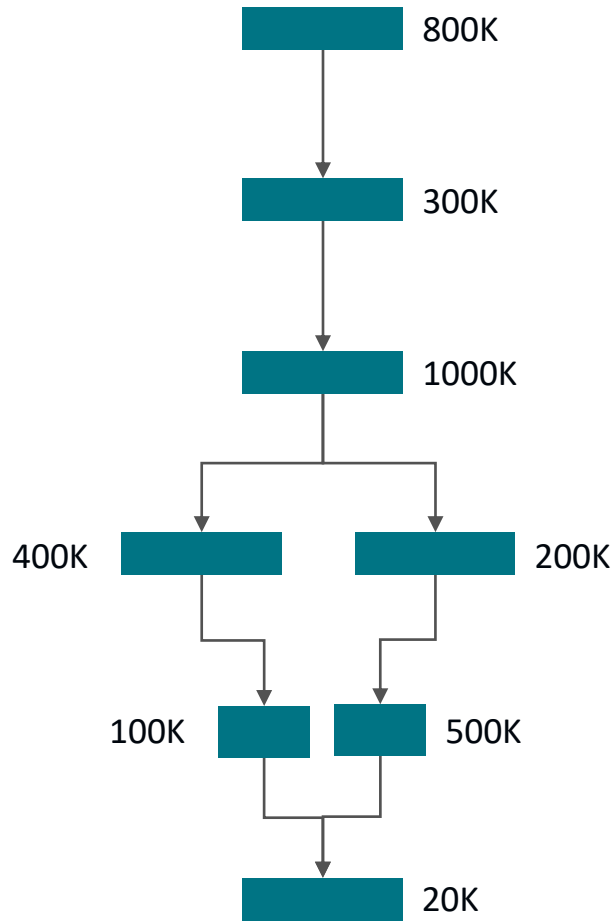
Memory allocation algorithm (3/3)

- Graph G = (V,E) where each node holds a price p_i
- We want find a solution which maximizes L2 BW and minimizes the Total Cost
- Let the indicator function be: $x_i = \begin{cases} 1, & item\ i\ is\ chosen \\ 0, & otherwise \end{cases}$
- Need to find a vector $\boldsymbol{x}$ which satisfies:

$$(1) \qquad \max_{\boldsymbol{x}} \sum_{i=1}^{n} p_i x_i = L2\ BW$$

$$(2) \quad Total\ Cost = \sum_{i=1}^{n} w_i x_i < W$$

  - Can be solved using dynamic programming
    - Time complexity: $O(nW)$
    - Space complexity: $O(nW)$

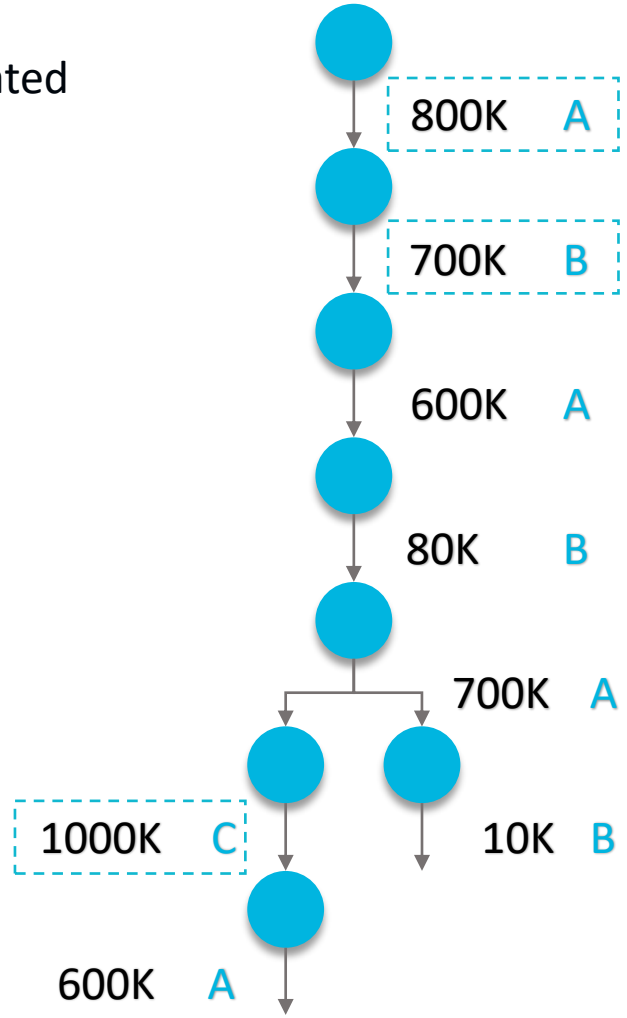


Algorithm flow

- Allocate buffer id for each layer.
  Then take the maximum size for each buffer. The total external memory allocated would be (where n is maximum number of buffers used in parallel)

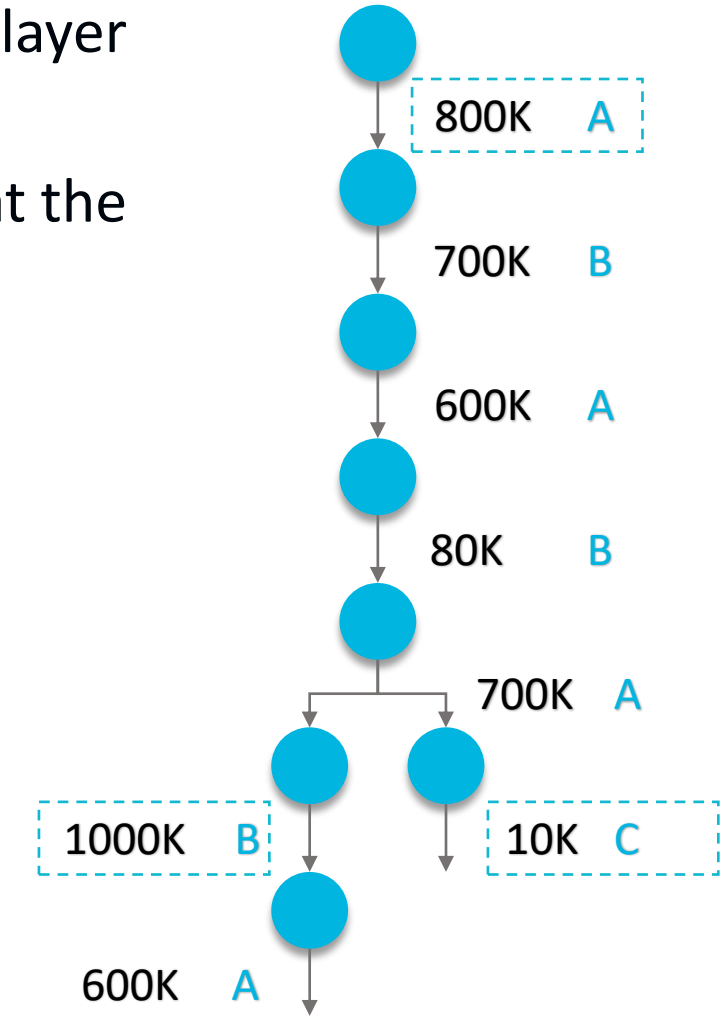
$$\sum_{k=1}^{n} \max(K.size)$$

- In this example, n equals to 3, and the total memory would be:
  1000K + 800K + 700K = 2500K



Optimization methods

- Note that the way the algorithm assigns buffer id to each layer may affect the result
- In this example (highlighted) you'll notice that swapping at the "branch point" between buffers B and C can save around 690K



Old MAX = 2.5MB
{1000K, 800K, **700K**}

New MAX = 1.81MB
{1000K, 800K, **10K**}

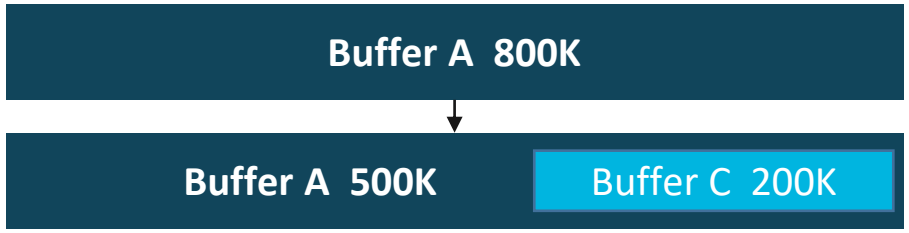
DDR size reduction – Preliminary results

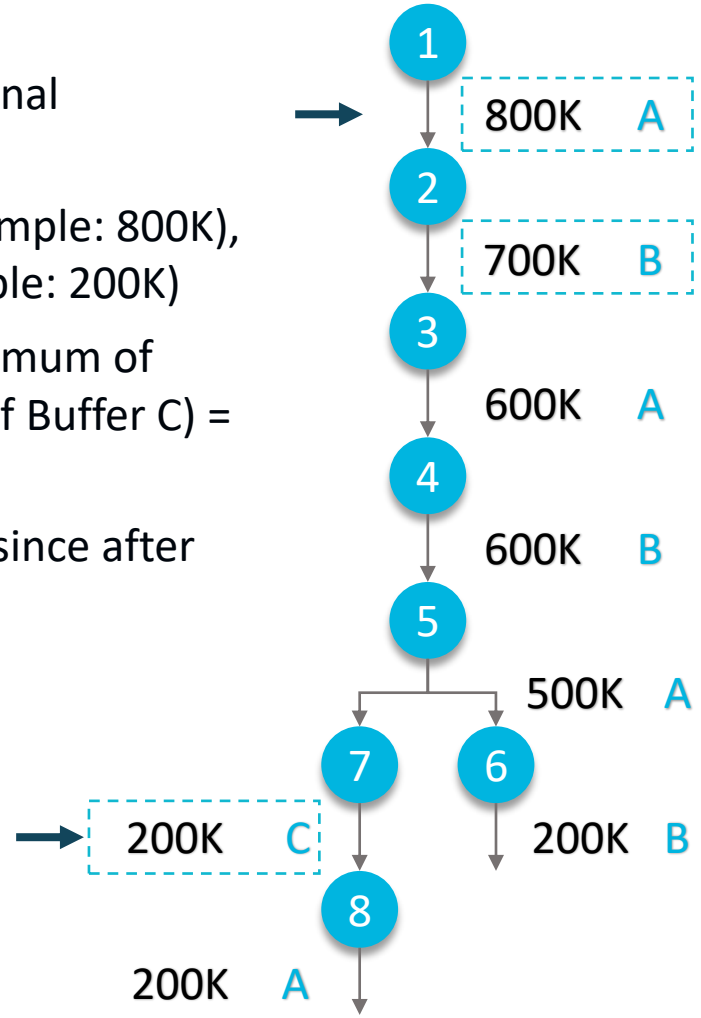
- The numbers shown herein are a result of preliminary optimization effort and there is a need for further work and testing to confirm the results
- This optimization has improved also the L2 utilization
  - A significant number of networks can now fit entirely into L2
    - 66% of tested networks now fit entirely into a 2MB L2 (previously 13%)
    - 83% of tested networks now fit entirely into a 4MB L2 (previously 38%)
  - This Buffer size reduction method is performed on L2 memory as

L2 Size	Amount of networks (in percentages) Before optimization:	Amount of networks (in percentages) After optimization:
2MB	13%	66%
4MB	38%	83%

Merge Buffers Optimization Improvements

- Overview
  - This optimization takes advantage of the fact that in many convolutional networks the size of buffers decreases after the first few layers
  - Thus, a buffer which might need to be large at the beginning (for example: 800K), often becomes very small towards the end of the network (for example: 200K)
  - In this example (see diagram) the algorithm allocates 800K (The Maximum of Buffer A) + 700K (The Maximum of Buffer B) + 200K (The Maximum of Buffer C) = 1.7MB
  - The improved algorithm will allocate buffer C (200K) inside buffer A, since after layer-5 the size of buffer A is 500K

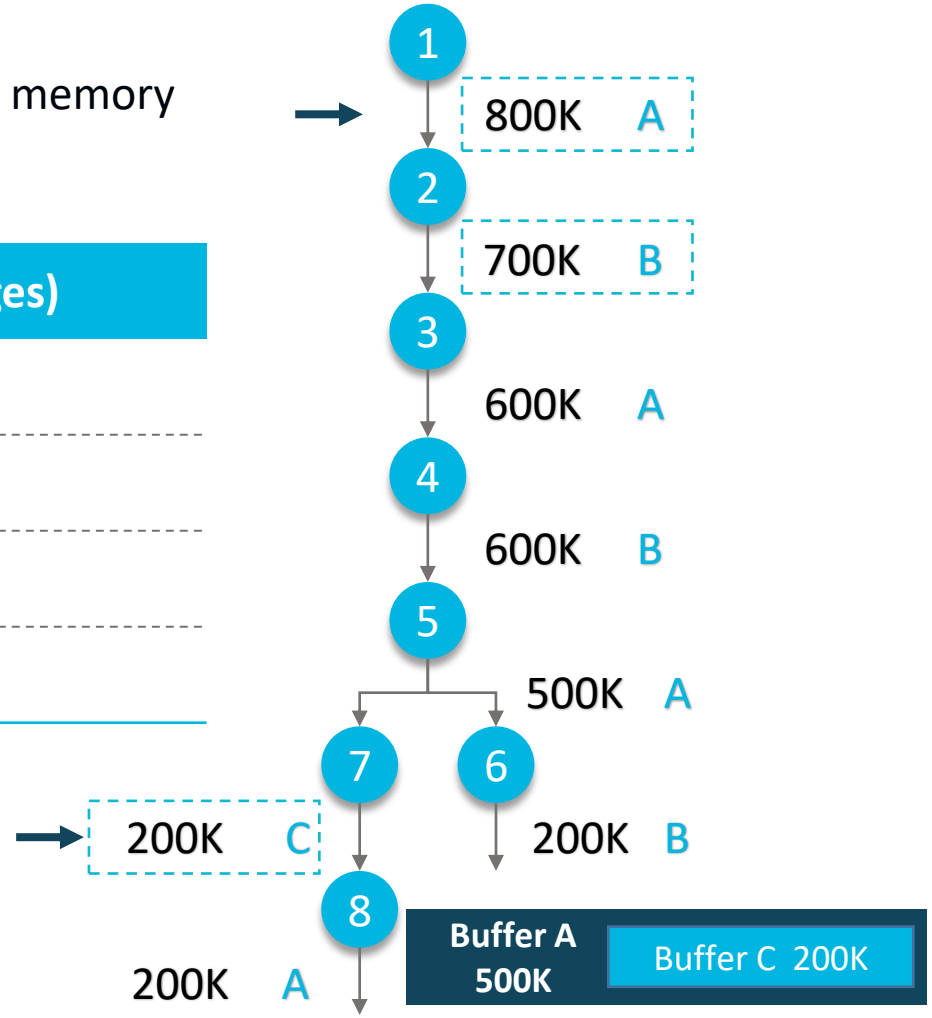




Merge Buffers Optimization Improvements

- Overview
  - This optimization phase reduces even further the amount of external memory allocated in most of our supported networks

Network	Reduced by (in percentages)
Mobilenet_SSD (TensorFlowLite)	35%
Yolo_V3 (Caffe)	28%
Inception_V3 (TensorFlow)	15%
OpenPose_Coco (Caffe)	15%



Summary and results – Classification networks

Network	DDR size reduction in %	L2 – 4MB BW	L2 BW Improvement in %
Alexnet	-100%	All in L2	100%
Inception v3	-100%	All in L2	100%
Inception v4	-66.67%	All in L2	100%
Resnet-50	-28.56%	All in L2	100%
Resnet-18	-68.24%	All in L2	100%
VGG16	-91.66%	All in L2	100%
MobileNetV1	-100%	All in L2	100%
MobileNetV2	-100%	All in L2	100%
MobileNetV3	-90.27%	All in L2	100%
Dense-Net121	-16.15%	-	-
ShuffleNet	-100%	All in L2	100%

Summary and results – Detection networks

Network	DDR size reduction in %	L2 – 4MB BW	L2 BW Improvement in %
MobileNet-ssd	-100%	All in L2	100%
MTCNN onet	-100%	All in L2	100%
MTCNN pnet	-100%	All in L2	100%
MTCNN rnet	-100%	All in L2	100%
FRCNN	-84.75%	Partially in L2	53.95%
Yolov2	-77.87%	All in L2	100%
Yolov3	-62.50%	Partially in L2	101.53%

Summary and results – Segmentation & RNN networks

Network	DDR size reduction in %	L2 – 4MB BW	L2 BW Improvement in %
Enet	-69.12%	Partially in L2	-
SegNet	-71.45%	Partially in L2	109.35%
ERFNet	-89.20%	-	-
Deeplabv3	-56.95%	-	-
UNet	-66.31%	Partially in L2	-
ICNet	-81.89%	Partially in L2	-
FSRCNN	-50.00%	All in L2	100%
OpenPose	-93.75%	All in L2	100%
LSID	-59.36%	-	-
Deepspeech1	-100%	All in L2	100%
Deepspeech2	-	-	-

Summary and conclusions

- We introduced CEVA SensPro and NeuPro-S processors for AI and CV applications.
- These cores are complemented by CDNN, a highly optimized graph compiler and runtime framework.
- We have shown the results of an effort conducted at CEVA to improve network buffer allocation and reduce DDR BW.
- At the same time, the L2 BW has increased significantly, thereby allowing many networks data to reside entirely in L2 memory.
- Further testing has confirmed the initial results.

Thank you

www.ceva-dsp.com