Challenges and Solutions for Embedding Vision AI

Charles Qi
System Solution Architect
March/25/2018
V 0.4

cadence®

Deep Learning is Broad

Wide spectrum of applications, wider spectrum of technology

Applications

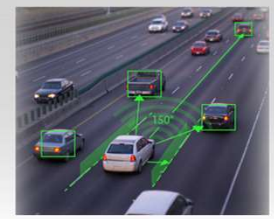
Social media and big-data search: Google, Facebook, Microsoft, Baidu, NEC, IBM, Yahoo, AT&T

Medical: genomics, radiology, screening, protein sequencing

Finance: TradeTrek, M.J. Futures, Alyuda

Speech recognition: Apple, Google, Nuance, Microsoft

Vision/ADAS: Nvidia, Intel



Public Frameworks

Caffe

TensorFlow

Microsoft CNTK

theano

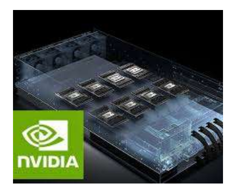
Caffe2

torch

mxnet

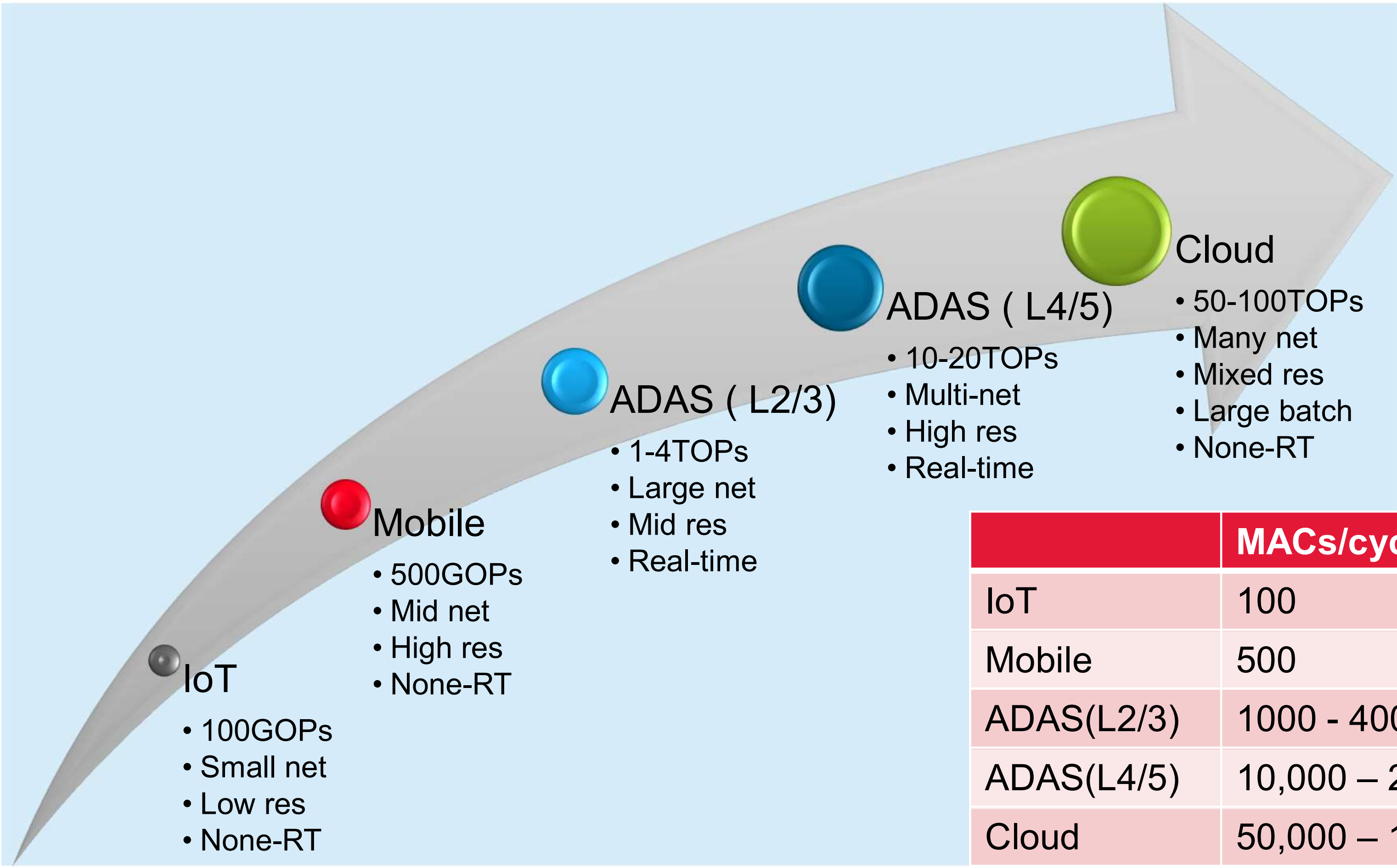
PYTORCH

HW Platforms



cadence

CNN: Convolution Performance is Key – How to Scale?

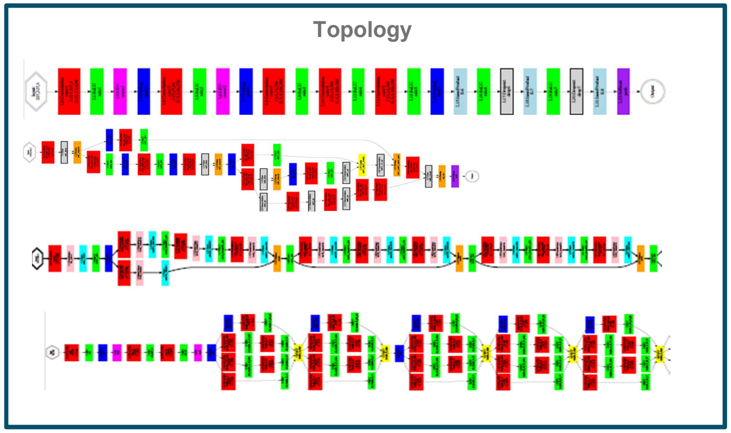


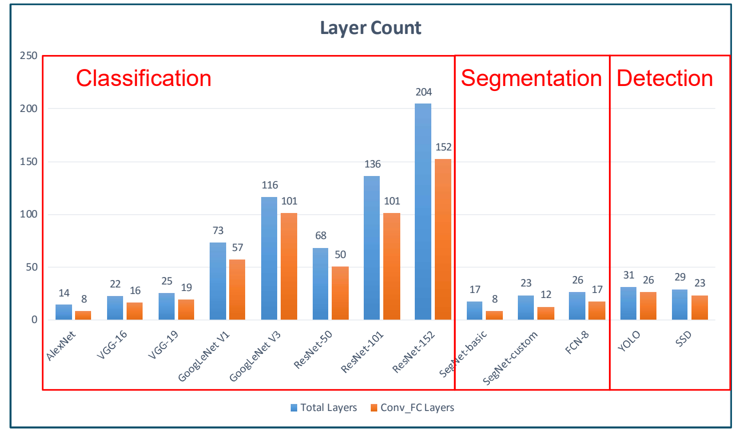
	MACs/cycle @1GHz
IoT	100
Mobile	500
ADAS(L2/3)	1000 - 4000
ADAS(L4/5)	10,000 – 20,000
Cloud	50,000 – 100,000

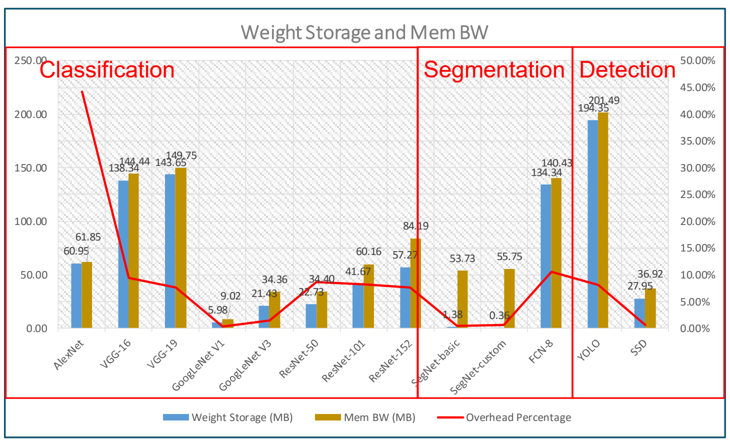
cādence®

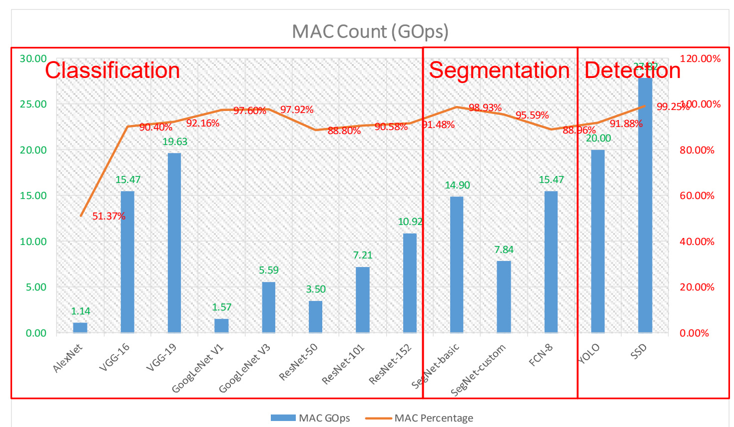
Diversity Drives Neural Network Complexity

Net topology, layer count, feature map and kernel sizes differ significantly









Conv and FC layers account for 90% load

cadence®

Convolution Operation Deep Dive

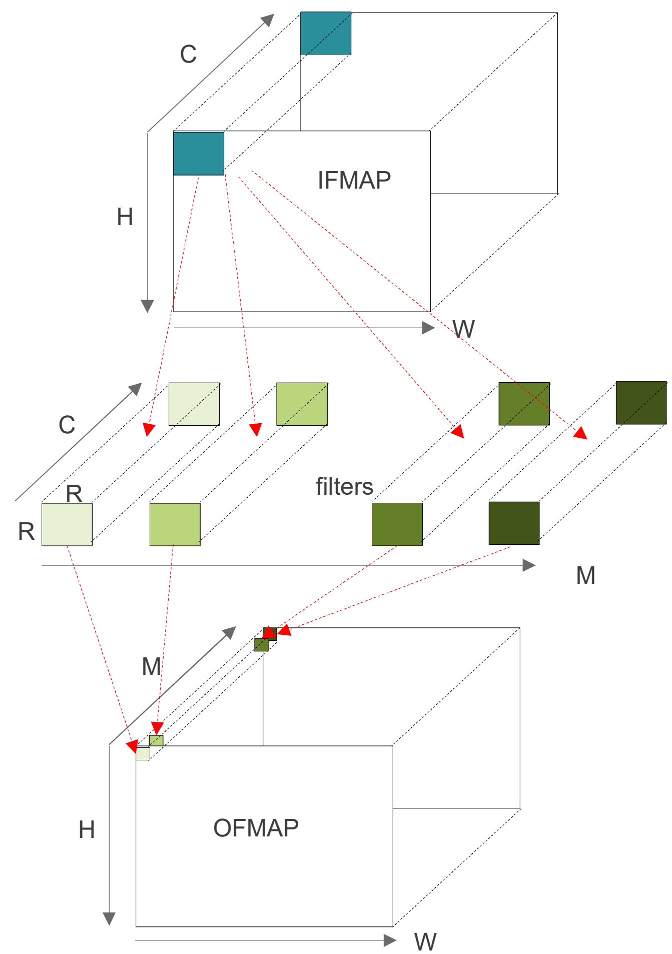
- Convolution layer is multi-dimensional
  - a 3-D input feature tensor convolving with a set of 3-D filter tensors
  - Deep loop stack in software
  - Large variation of input spatial dimension, channel depth and filter count

Operation:

$$F_O(x,y,n) = \sum_{c=0}^{C-1}\sum_{r=-R}^{R}\sum_{r=-R}^{R} W(r,r,c) * F_I(x+r,y+r,c)\ \ \forall x \in X, y \in Y, m \in M$$

Implementation as 8 nested loops:

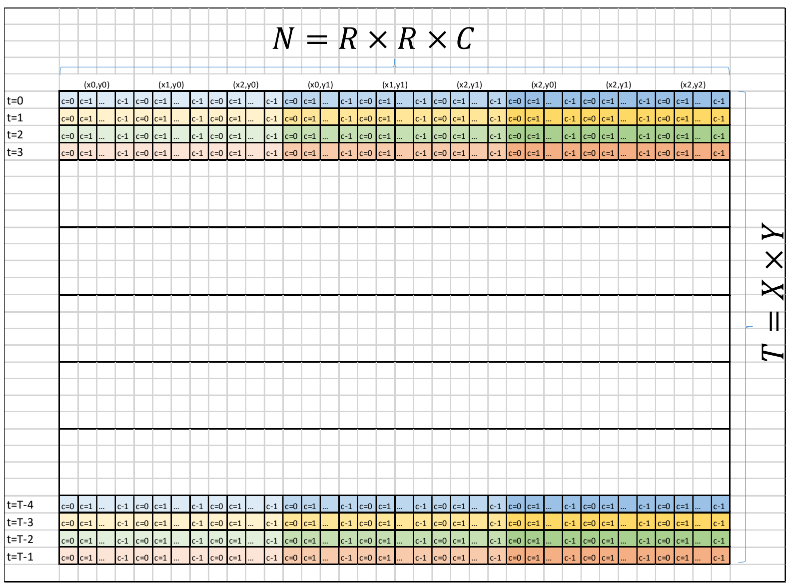
Batch → *for (b=0; b<B; b++)*
Layers → *for (l=0; l<L; l++)*
Filter set → *for(m=0; m<M; m++)*
Ifmap height → *for (y=0; y<H; y++)*
Ifmap width → *for (x=0; x<W; x++)*
$\boldsymbol{F_O(x,y,m)} = 0$
Channels → *for (c=0; c<C; c++)*
Kernel width → *for (ry=0; ry<RY; ry++)*
Kernel height → *for (rx=0; rx<RX; rx++)*
$\boldsymbol{F_O(l,b,x,y,m) \mathrel{+}= Wl_{,b,m}(rx,ry,c) * FI(l,b,x+rx,y+ry,c)}$



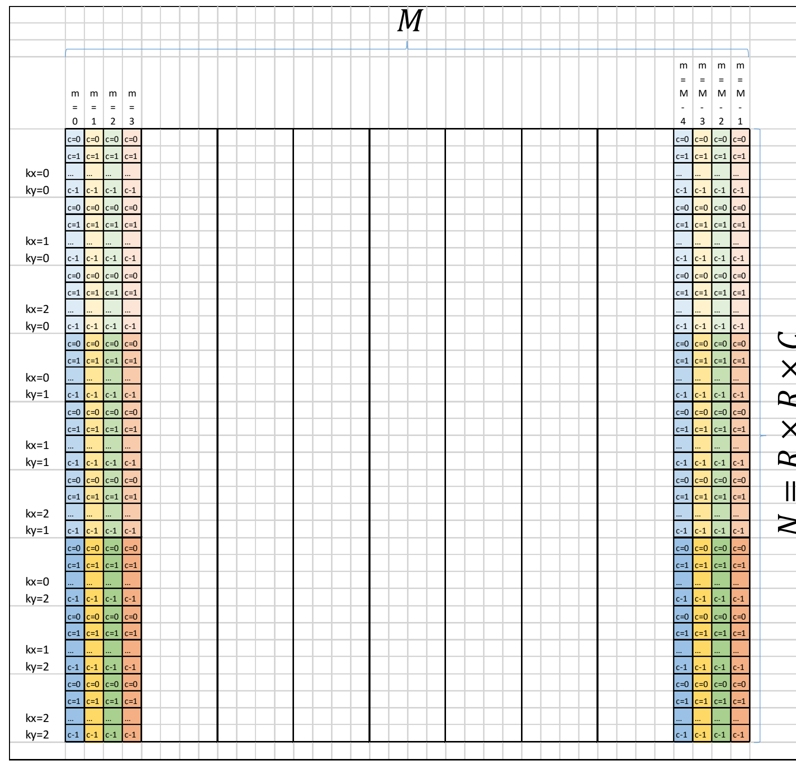
GEMM Approach Non-ideal for Embedded AI

- GEMM converts convolution into 2-D matrix multiply by brute-force
- IFMAP
  - TxN array, linearize spatial dimension X*Y rows
  - Replication of activation data by R*R across C channels
- Filter
  - N*M array, R*R*C channel element rows, M filter columns

Input Feature Map Matrix



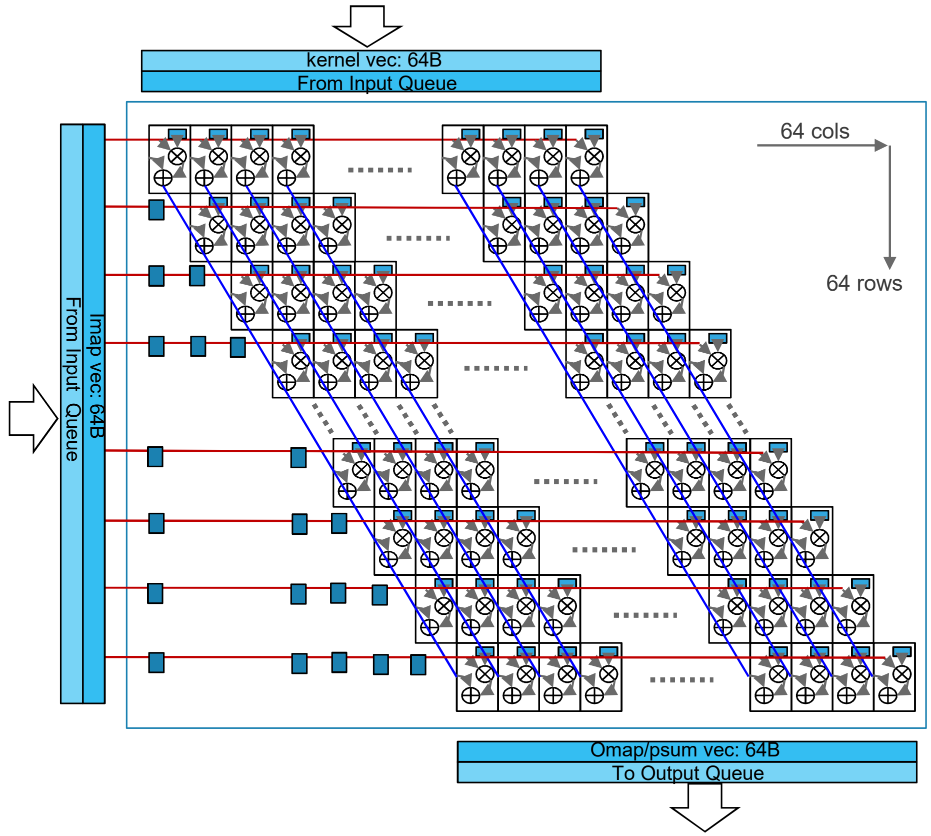
Filter Matrix



cādence®

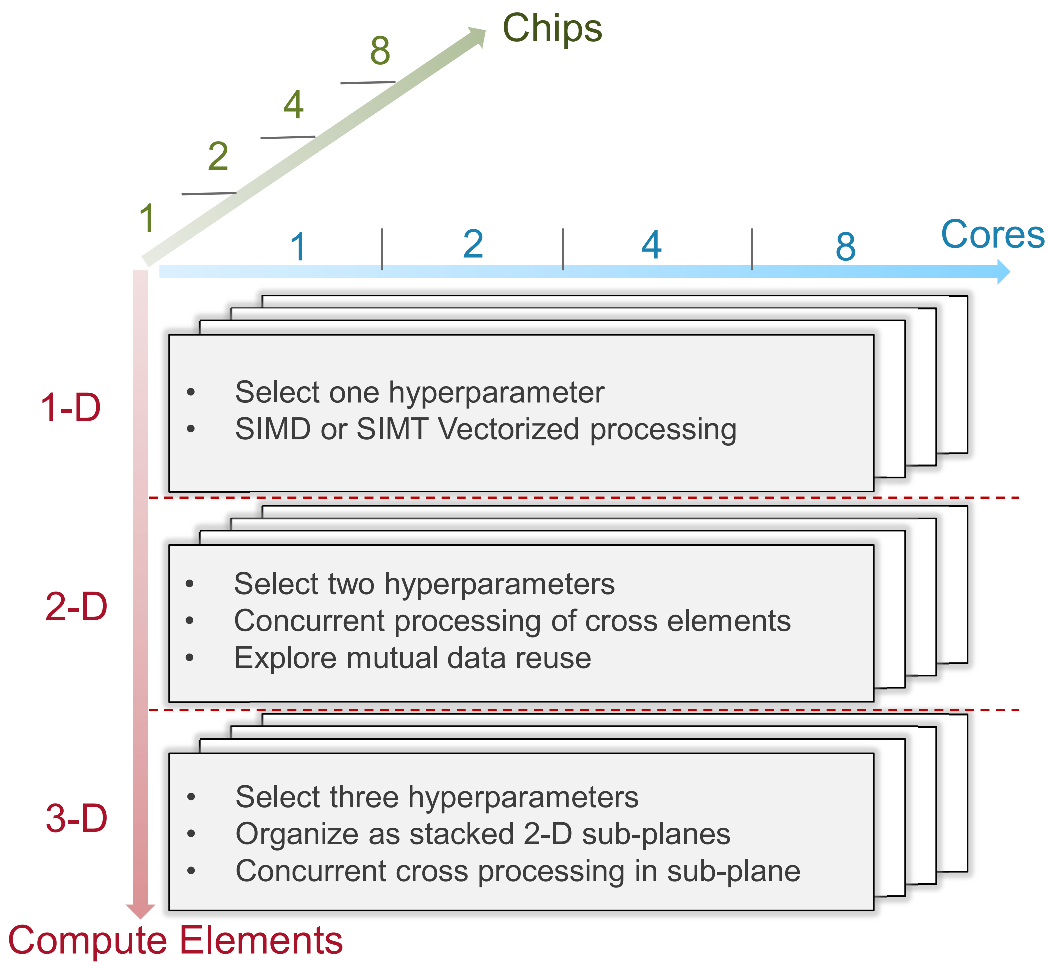
NxN Plain GEMM Accelerator

- NxN, N=64 MAC array
  - 64 independent vertical slices
  - 64 accumulator rows
- Stationary weight array
  - 64 rows of kernel weights preloaded to array cells in 64 cycles
- Dynamic input/output vectors
  - One 64B/cyc imap vector is fed to array
  - One 64B/cyc omap vector is produced
- Psum stationary array also possible
- Limitations
  - Rigid array configuration often under-utilized in various layer dimensions
  - High pipeline overhead
  - im2col implodes memory footprint
  - Inefficient for vector*vector, vector*matrix without large batch
  - Hard to partition for multi-core



cādence®

Scaling the Dimensions to Accelerate



Nominal network size:

- 50 layers
- 500M MACs
- 60 fps

1.5T MACs

1D vector engine (GPU, DSP or HW):

- Frequency: 1GHz
- 1000 – 2000 MACs/cycle
- At the limit of datapath width
- Utilization hard to sustain

2D or 3D array engine:

- Frequency: 0.5-1GHz
- 4K MACs/cycle
- 2D: 64x64
- 3D: 32x32x8
- Easy to achieve high utilization
- Data reuse minimize IO
- More complex scheduling

cādence

Extract Parallelism for Multi-dimensional Scaling

Dimension	Dep.	Single Core Multi-Dimension Vectorization	Multi-core (U) Parallel Load Split
Batch of images (B)	Indep.	• Typically not applied because B << V • May be applicable to narrow SIMD • Requires interleaving data from multiple "images"	• Easy to distribute smaller batch to cores when B >> U • Reduce filter weight reuse (weight replication, reduced W reuse) • Latency to process one image not reduced, bad for RT application
Layer (L)	Data dep	• NA, layers processed sequentially or pipelined in different cores due to layer-to-layer data dependency	• No overlapping weights in cores, most effective weight reuse • Challenge is to balance compute load for different layers • Core-to-core data dependency requires core-to-core pipelining
Number of kernels (M)	Indep.	• Most efficient for wide SIMD (M >> V and M%V == 0) • Allow kernel weight or data reuse • Large local mem footprint due to parallel accumulations Requires data and weight reorganization	• Easy to partition for U cores without weight overlap when M >> U • May limit core-level SIMD vectorization on M when M < V*U • Input data is replicated across cores
Spatial (X,Y)	Indep.	• Typically applied to X only when X >> V • Filter weight can be reused for entire vector • Inefficiency if X%V is non-zero	• Flexible to partition tiles to cores • Inefficiency when tile size becomes smaller than SIMD width • Filter weights replicated in cores processing different tiles
Input depth (C)	Accum.	• Inefficient for shallow layer with C << V • Require vector reduction of the partial accumulator values in SIMD ways	• Ineffective for shallow layer with C << U • Require final accumulation of intermediate results across cores
Kernel (R)	Accum.	• Typically not applied due to odd and small R • Different rows of R in the same vector computes different data, require data re-org, lack of reuse	• Typically not applied due to odd and small R • Require final accumulation for R values from different cores

Accelerating Embedded NN at Edge

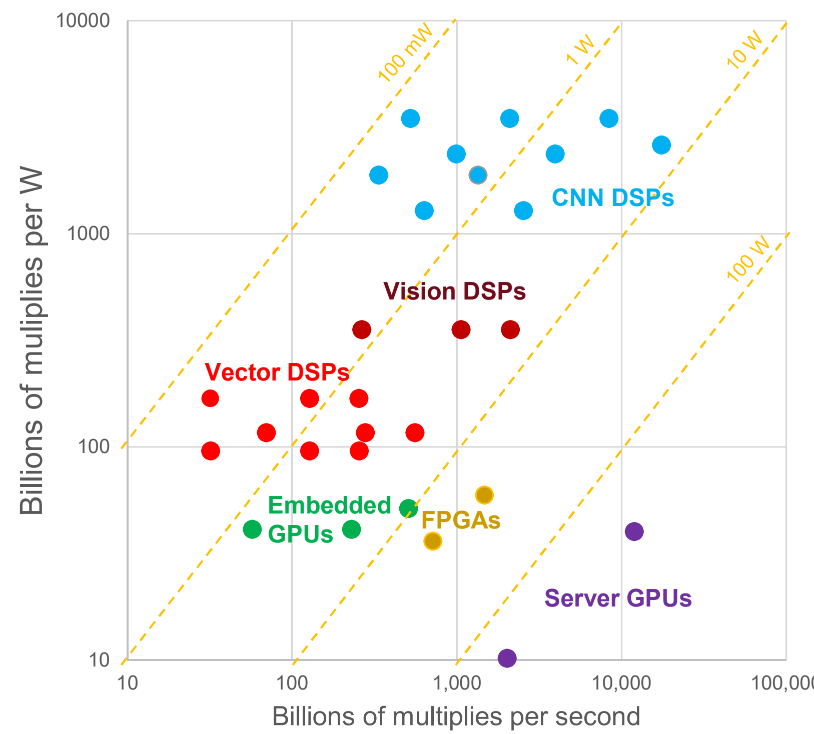
solution need to achieve high performance with architecture flexibility

Performance
- 1TMAC Performance @1GHz

Power Efficiency
- Custom ISA to achieve high HW utilization

Flexibility
- Full DSP programming capability for all NN layers

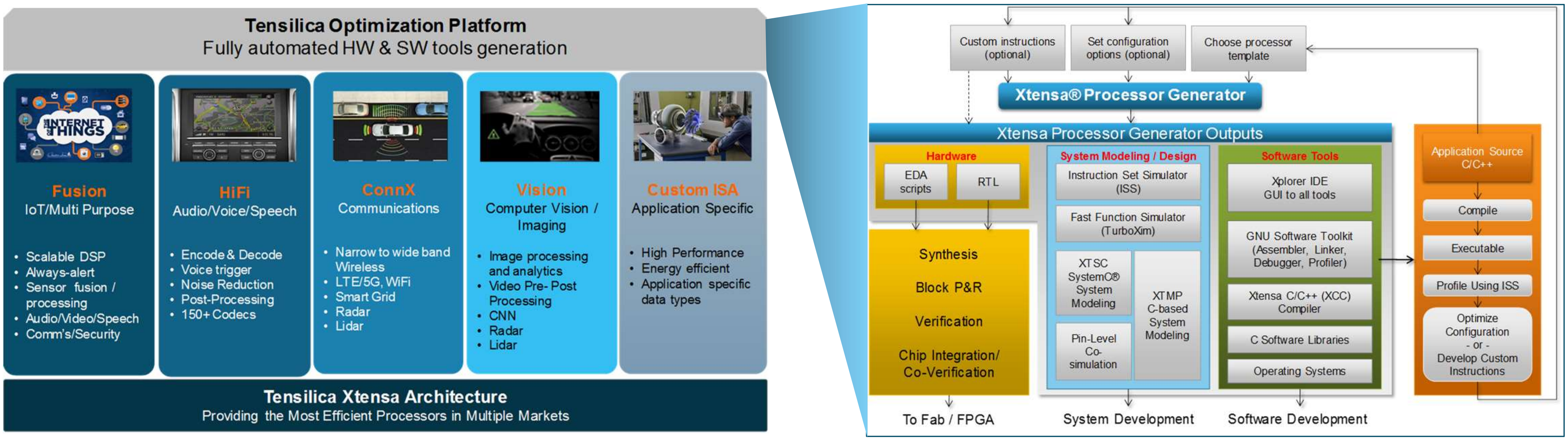
Throughput and Efficiency for CNN



cādence®

Viable Solution: Tensilica Scalable DSP Platform

DSP cores tailored for application specific needs

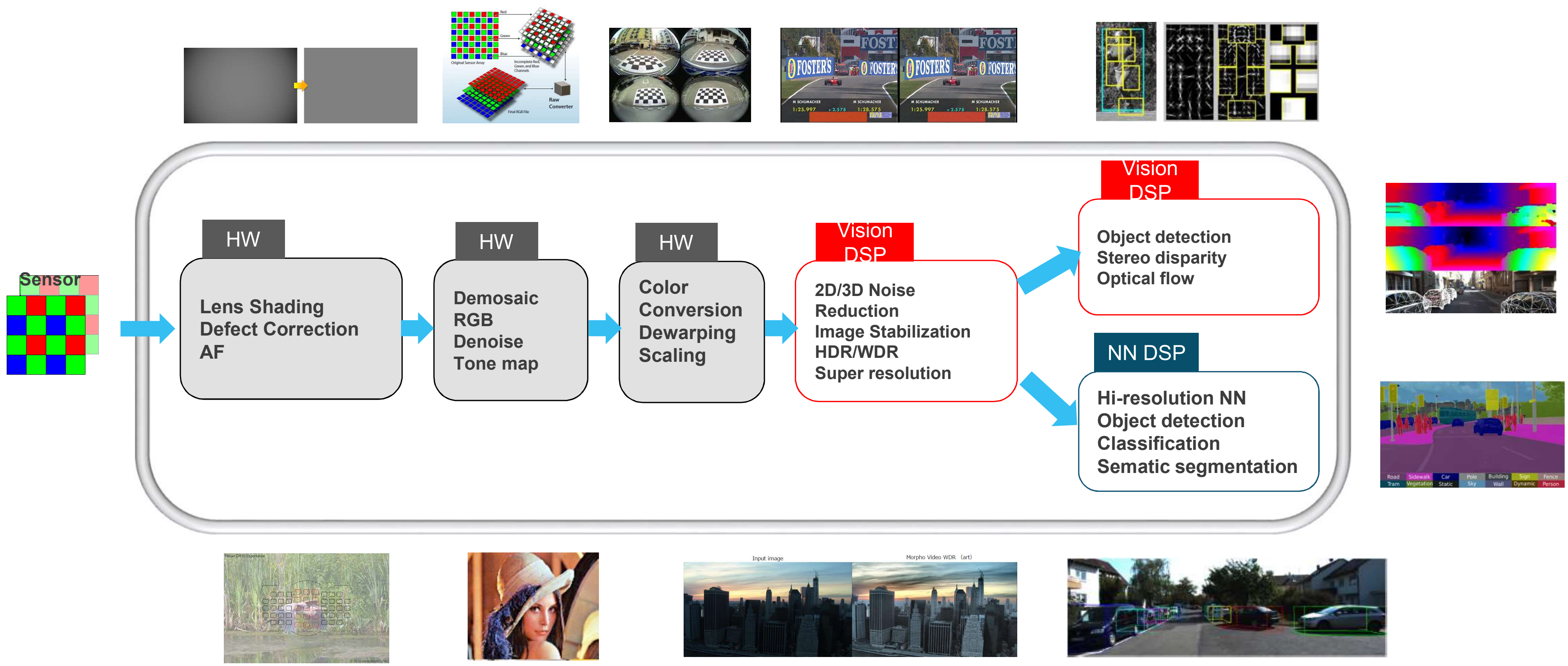


- **Scalable performance, consistent programming model**
- **Common baseline Xtensa architecture + application specific processing optimization**
- **Fully integrated HW/SW tool chain from user-friendly Xplorer IDE**

cadence

The Real-time Embedded Vision AI Pipeline

where intelligence and programmability meet



cadence®

How is DSP Optimized for Embedded Vision/CNN?

Focus on computation efficiency and minimize memory access

Increase computation efficiency

- ✓ ISA level parallelism based on fixed point SIMD
- ✓ Balance memory and compute resource with VLIW
- ✓ Specialized TIE instructions for special tasks

Reduce system memory access overhead

- ✓ Improve access efficiency with local RAM
- ✓ Hide access latency with tiling and DMA
- ✓ Minimize cycle count for non-contiguous access

cādence®

Example of Special Instruction Extension (TIE)

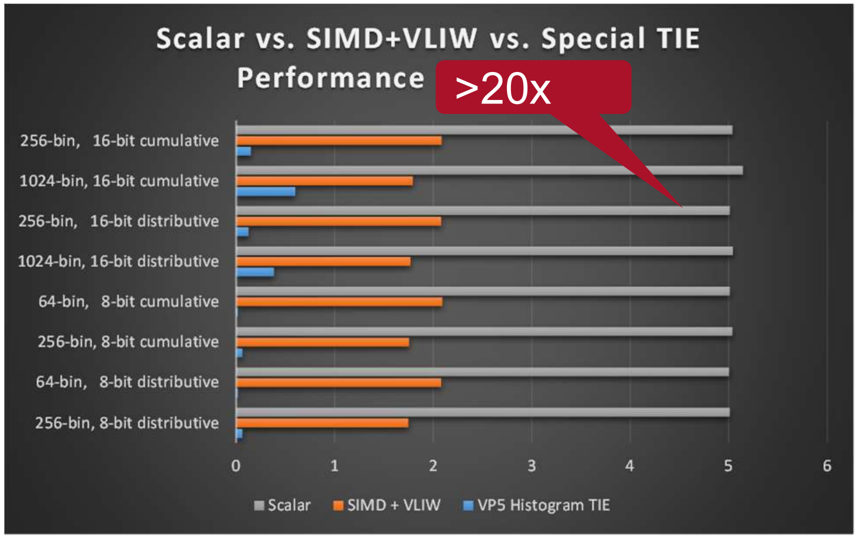
Achieve significant performance while reducing power

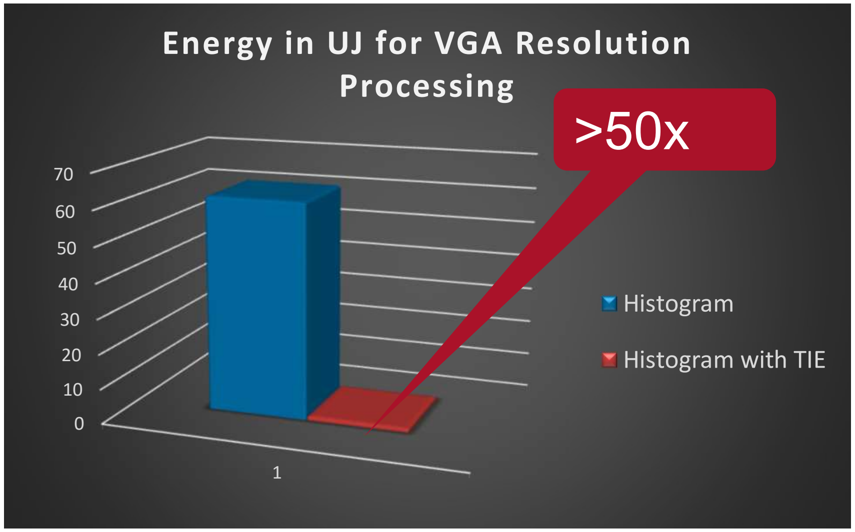
Scalar C

```
//scalar C code
 for (indxData = 0; indxData < len; indxData += 4){
   data = (uint8_t) datain[indxData];
   hist[data >> ShiftRight]++;
   data = (uint8_t) datain[indxData + 1];
   hist[data >> ShiftRight]++;
   data = (uint8_t) datain[indxData + 2];
   hist[data >> ShiftRight]++;
   data = (uint8_t) datain[indxData + 3];
   hist[data >> ShiftRight]++;
 }
```

Specialized TIE for histogram

```
//specialized TIE created for histogram processing
 for (int y = 0; y < height; ++y) {
   IVP_COUNTEQZ4NX8(vdst, sr, vec0, vec1); vhist0 = IVP_ADDNX16(vhist0, vdst); //   0 .. 31
   IVP_COUNTEQ4NX8(vdst, sr, vec0, vec1); vhist1 = IVP_ADDNX16(vhist1, vdst); //  32 .. 63
   IVP_COUNTEQ4NX8(vdst, sr, vec0, vec1); vhist2 = IVP_ADDNX16(vhist2, vdst); //  64 .. 95
   IVP_COUNTEQ4NX8(vdst, sr, vec0, vec1); vhist3 = IVP_ADDNX16(vhist3, vdst); //  96 .. 127
   IVP_COUNTEQ4NX8(vdst, sr, vec0, vec1); vhist4 = IVP_ADDNX16(vhist4, vdst); // 128 .. 159
   IVP_COUNTEQ4NX8(vdst, sr, vec0, vec1); vhist5 = IVP_ADDNX16(vhist5, vdst); // 160 .. 191
   IVP_COUNTEQ4NX8(vdst, sr, vec0, vec1); vhist6 = IVP_ADDNX16(vhist6, vdst); // 192 .. 223
   IVP_COUNTEQ4NX8(vdst, sr, vec0, vec1); vhist7 = IVP_ADDNX16(vhist7, vdst); // 224 .. 255
}
```



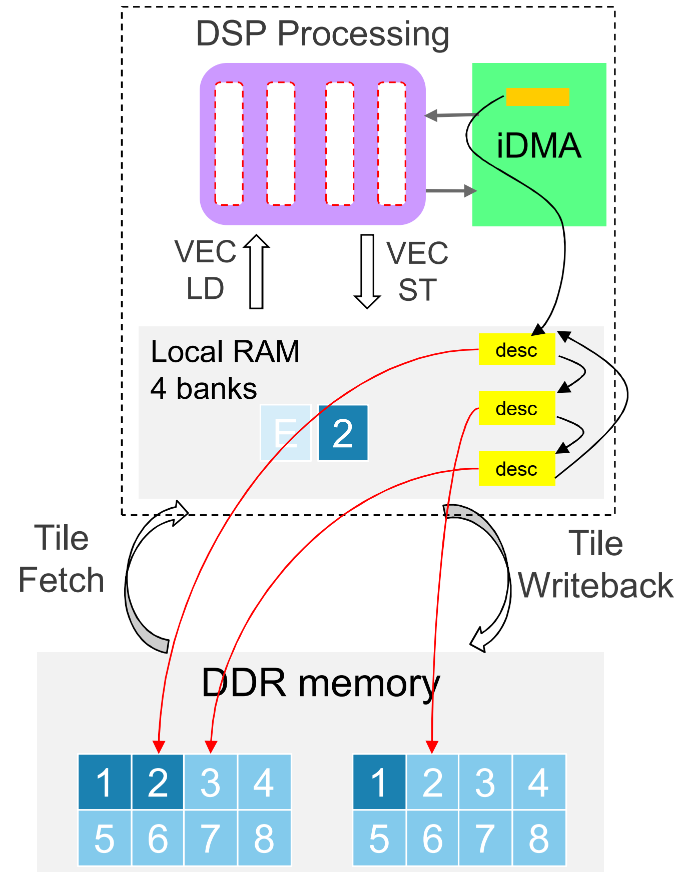


cādence®

Hiding Memory Access Latency via Tiling and DMA

Offload tile fetch and writeback

- Tile fetch and writeback by software is inefficient, blocking data processing
- Build-in DMA performs data moving in parallel
  - Parallel DMA access to local memory banks
  - Dedicated NoC master port for DMA
  - Multiple outstanding requests in pipe to memory
- 2D descriptor ring supports complex memory access patterns
  - DMA is aware of the row-dominant image storage format by skipping row pitch at the end of each row
  - 3D/4D tile stream can be composed using multiple descriptors
  - Flexible data alignment
- DMA register-mapped inside processor to minimize initiation/termination overhead
- Ping-pong tile buffers in local RAM removes dependency



cādence

Representative NN DSP Architecture Considerations

Uniquely design ISA to strive for processing efficiency

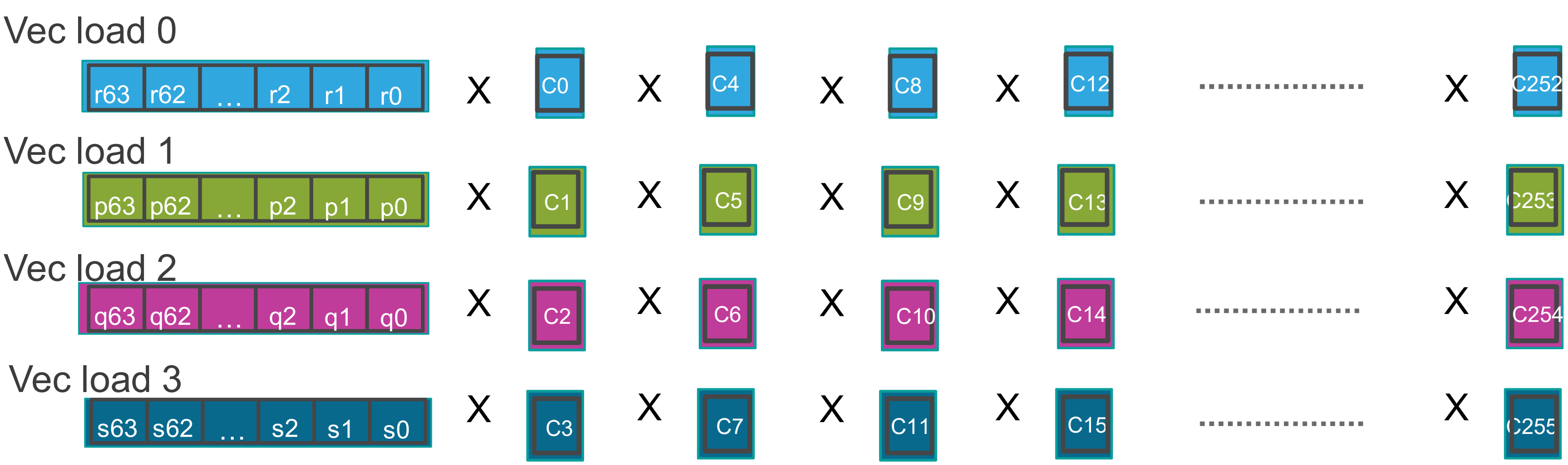
- 256 – 1024 8x8 multiple-accumulate operations per cycle
- Vector-by-vector, vector-by-scalar operations with multiple accumulators
- Flexible vectorization scheme to maximize HW utilization
- Special addressing for load/store 3D tensor data
- Acceleration for pooling, nonlinear activation layers
- On-the-fly Coefficient Compression/Decompression

cādence®

NN ISA Architecture Optimization Concept (1)

Amortize IO cost across multiple computations

- Problem: Vector x Vector operations requires 2 SIMD loads + 1 SIMD store
- Explore Data Reuse prosperity in CNN
  - Multiple filters applied to common IFMAP input data
  - Multiple spatial locations of IFMAP convolve with same filter elements
- Optimization: perform vector x scalar for multiple scalars
  - Amortize vector load BW and keep computation pipeline full

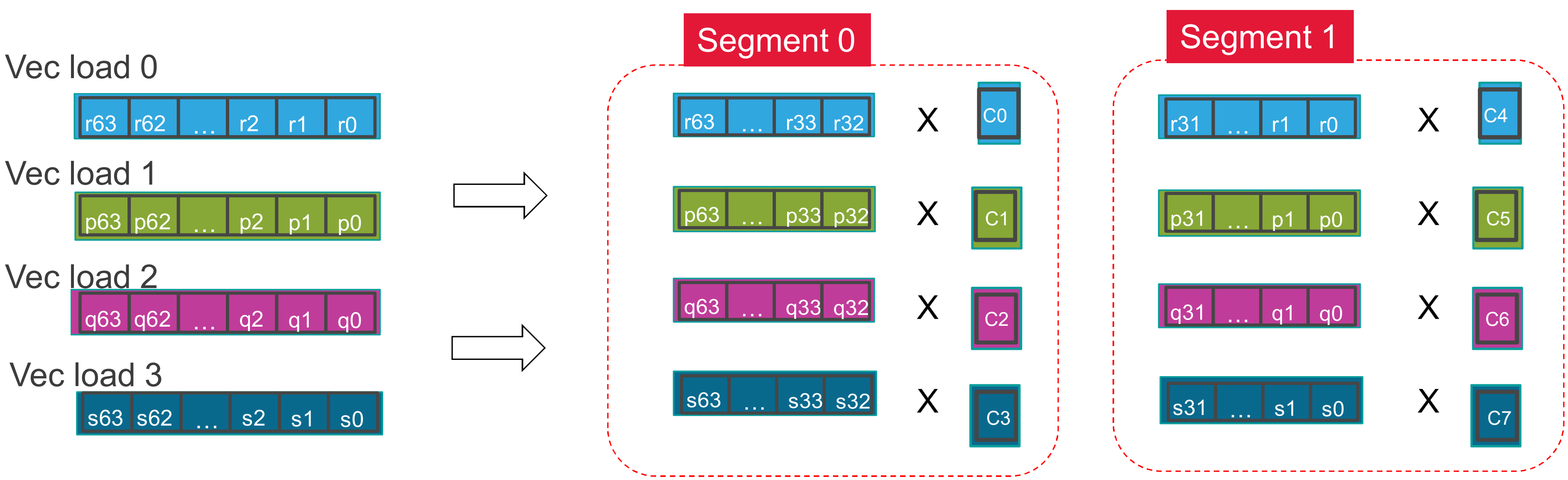


cādence®

NN ISA Architecture Optimization Concept (2)

SIMD segmentation

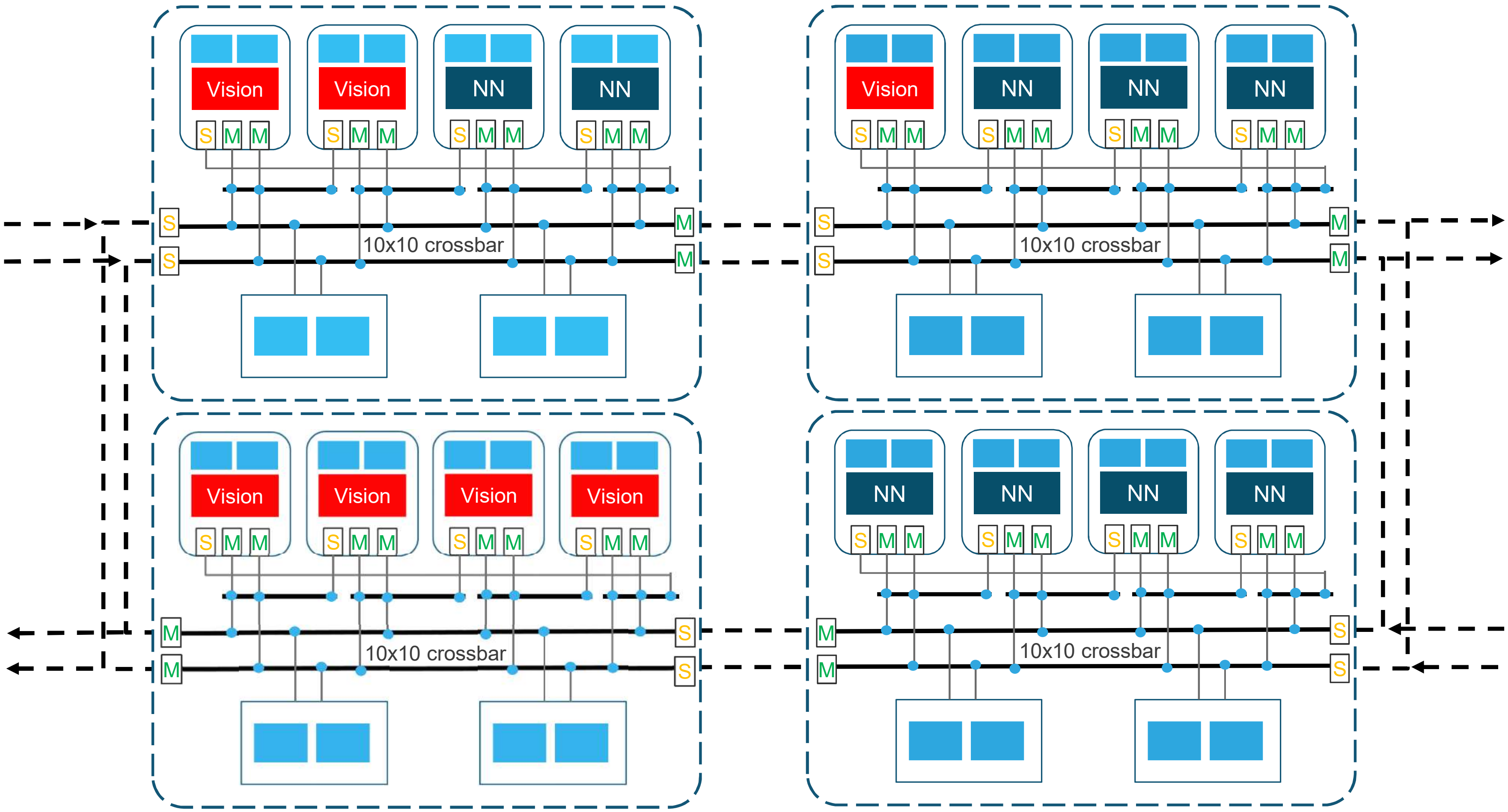
- Problem: layer dimension much smaller than SIMD width
- Optimization: break wide SIMD into smaller segments
  – Apply different scalar data to different segments to fully utilize HW resource



cādence®

Scale Vision Sub-system Heterogeous Multi-core

Flexibility to customize MP cluster under the same programming model



cadence

Multi-core NN Load Partition Example

Split load across layer/batch/kernel

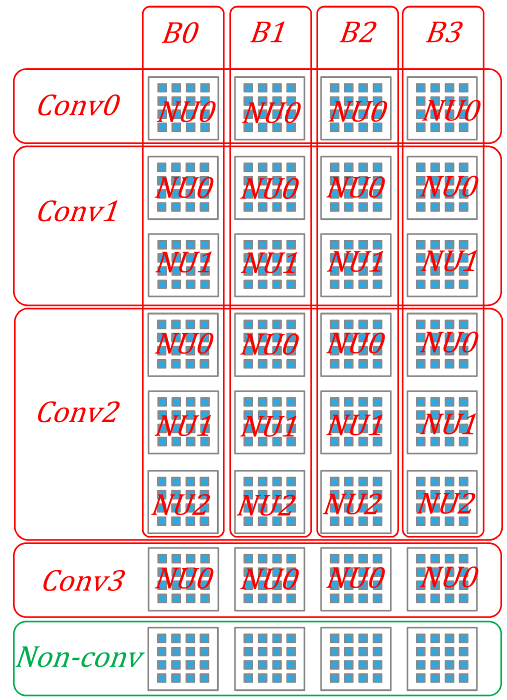
Operation:

$$F_0(x,y,n) = \sum_{z=0}^{Z-1}\sum_{r=-R}^{R}\sum_{r=-R}^{R} W(r,r,z) * FI(x+r,y+r,z)\ \forall x \in X, y \in Y, n \in N$$

Implementation as nested loops:

layers — *~~for (l=0; l<L; l++)~~*
if(layers[l] == CONV) {
Ifmap batch — *~~for (b=0; b<B; b++)~~*
kernels — *~~for (nu=0; nu<NU; nu++)~~*
kernels — *for (nv=0; nv<NV; nv++)*
Ifmap height — *for (y=0; y<Y; y+=S)*
Ifmap width — *for (x=0; x<X; x+=S)*
Ifmap channels — *for (z=0; z<Z; z++)*
Kernel width — *for (ky=0; ky<KY; ky++)*
Kernel height — *for (kx=0; kx<KX; kx++)*
$\boldsymbol{F_0(x,y,n) \mathrel{+}= Wl_{,b,n}(kx,ky,z) * FI(x+kx,y+ky,z)}$
} #end if

SIMD vectorization in NV or X

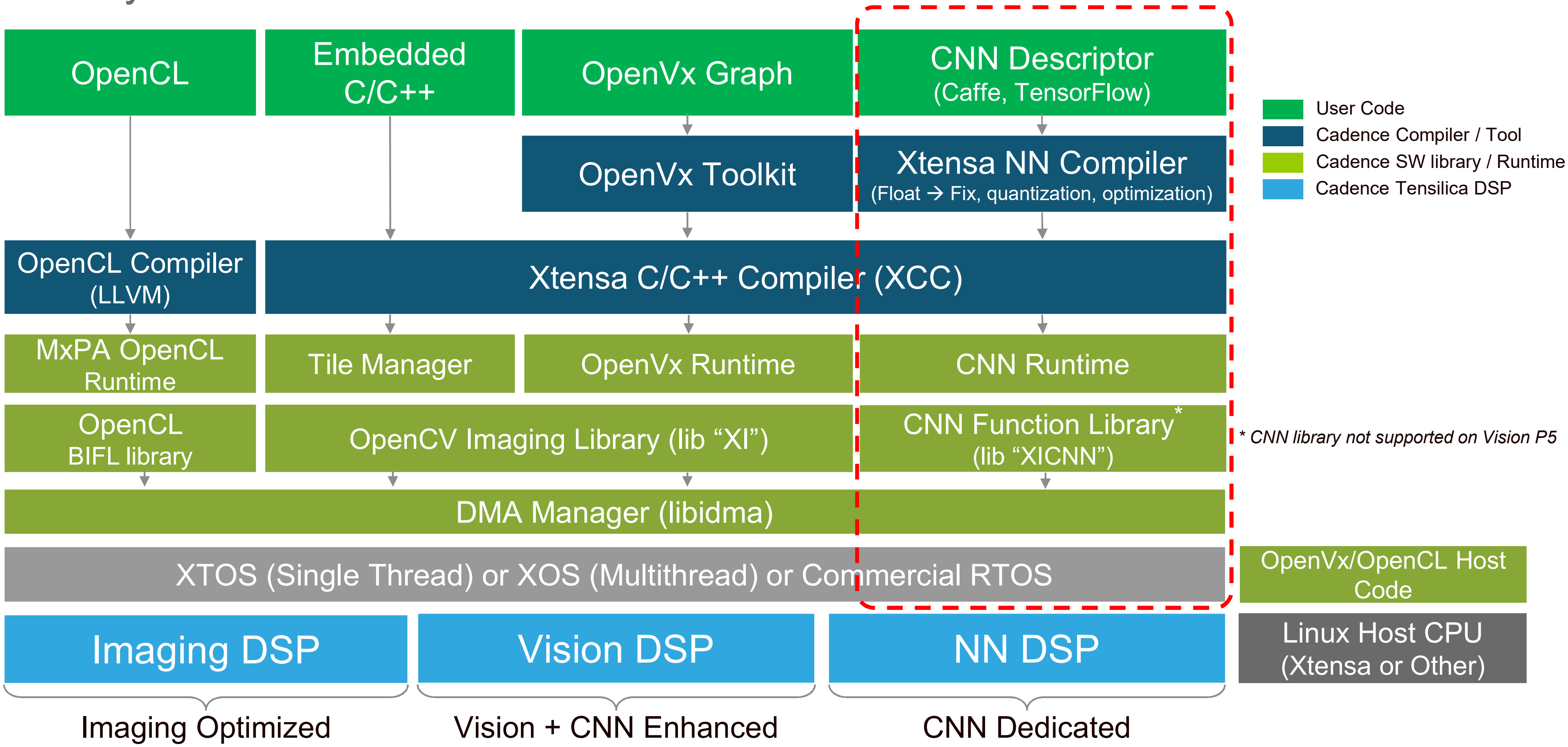


- L and B loops are distributed across MP cores
- N is split into two loops, NU, NV and N = NU*NV
  - NU is distributed across cores
  - NV is handled by vectored SIMD

Cores are designated to certain layers and batches. Most efficient if layers can be load-balanced, no overlap in weight coefficients across cores.

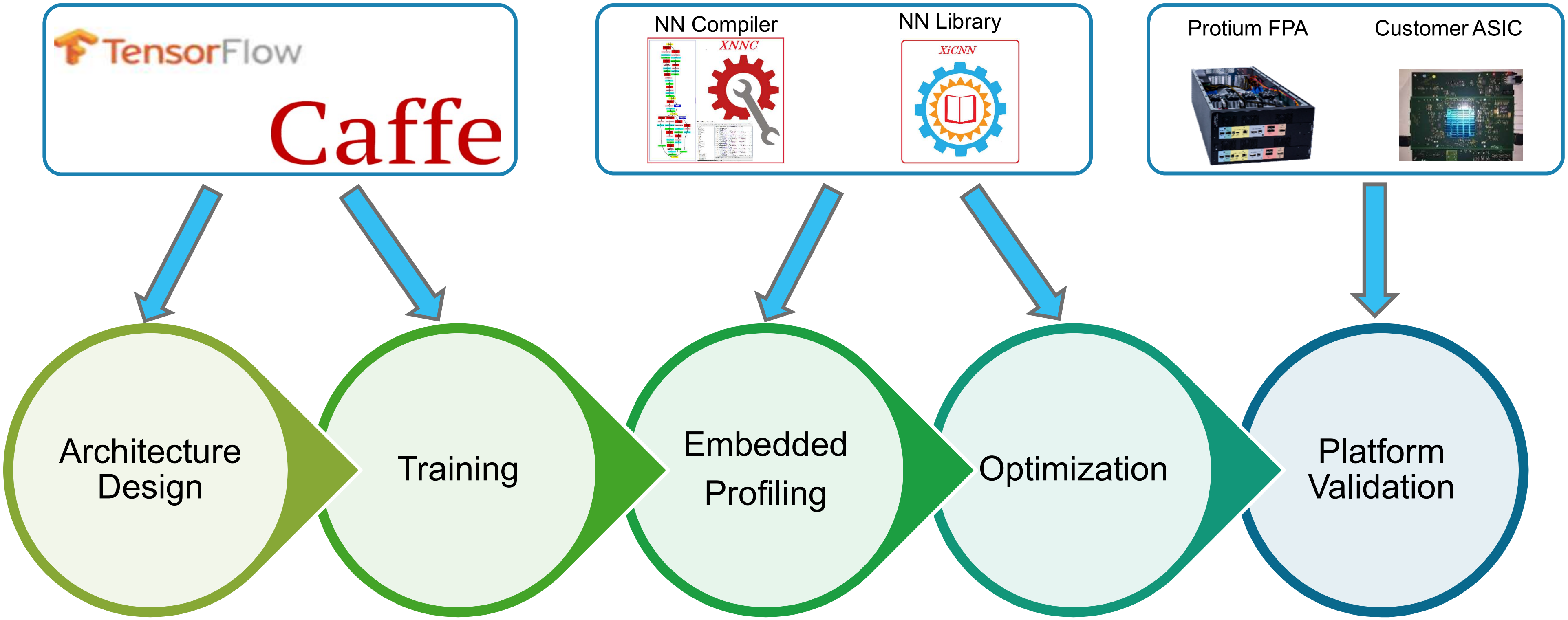
Consistent and Comprehensive Vision Software Solution

Full ecosystem of software frameworks and toolchain across common architecture



cādence

Concept to Hardware - Embed AI in Five Streamlined Steps



Architecture Design
- Network Type
- Network Topology
- Network Depth

Training
- Performance Enhancement
- Layer Pruning

Embedded Profiling
- Quantization
- Layer Merging
- Layer Organization

Optimization
- Tiling
- Sequencing
- DMA
- Vectorization

Platform Validation
- E2E Application
- Performance Validation
- In-field Validation

cadence®

Conclusion

Wide demand for NN in embedded applications

- Diversified application needs drive various NN architectures
- Demand for higher and higher computation performance
- Embedded power profile and implementation cost drive optimization

Optimization of embedded vision/AI remains challenging

- Too many architecture variants makes it hard to optimize across
- Brute-force approach leads to higher power/memory BW, lower utilization
- Need multi-dimensional scaling without losing flexibility

Specialized NN DSP provides efficiency and flexibility

- Highly optimized ISA improves NN computation efficiency
- End-to-end full network implementation in image/vision pipeline
- SW framework and library automates NN implementation