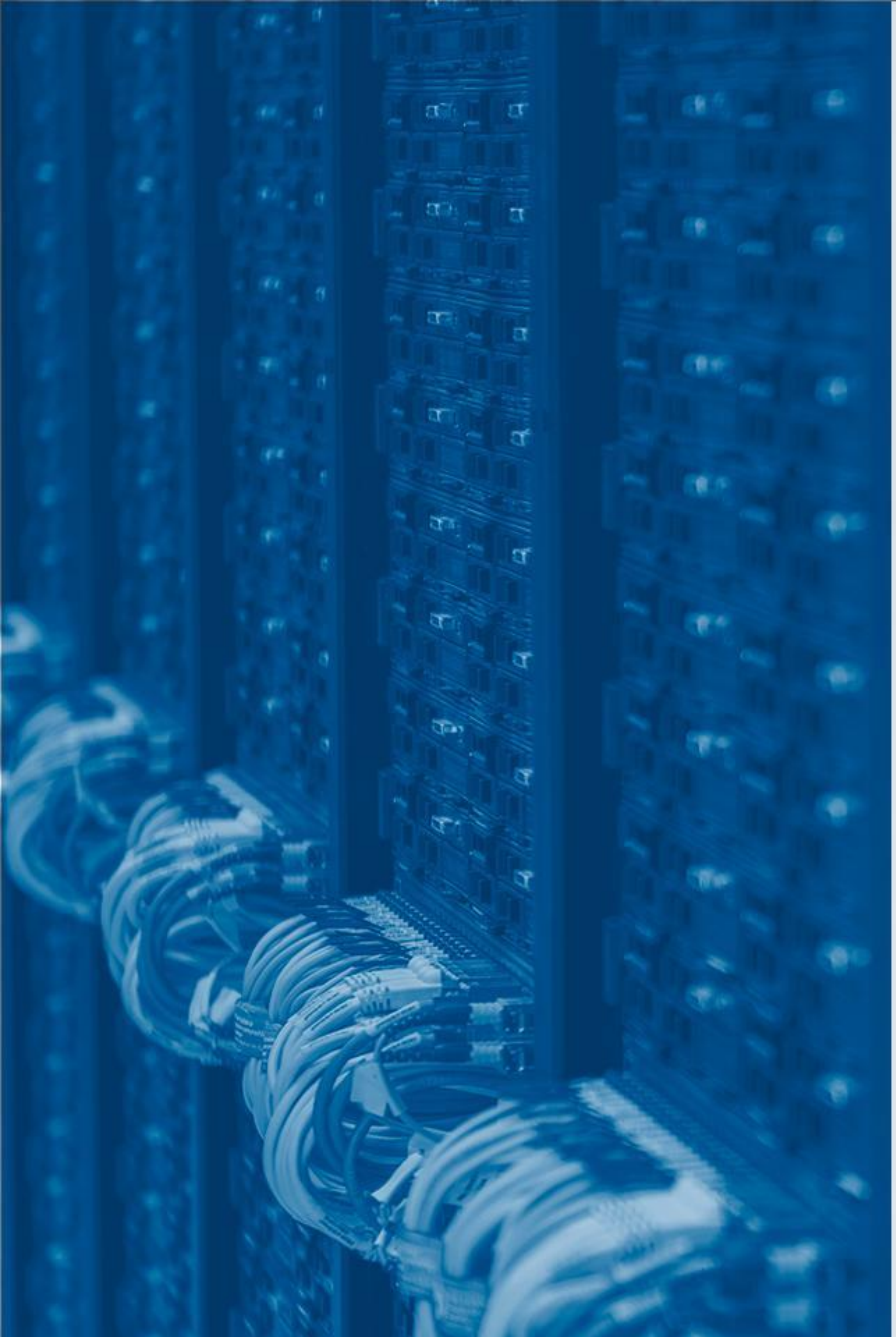


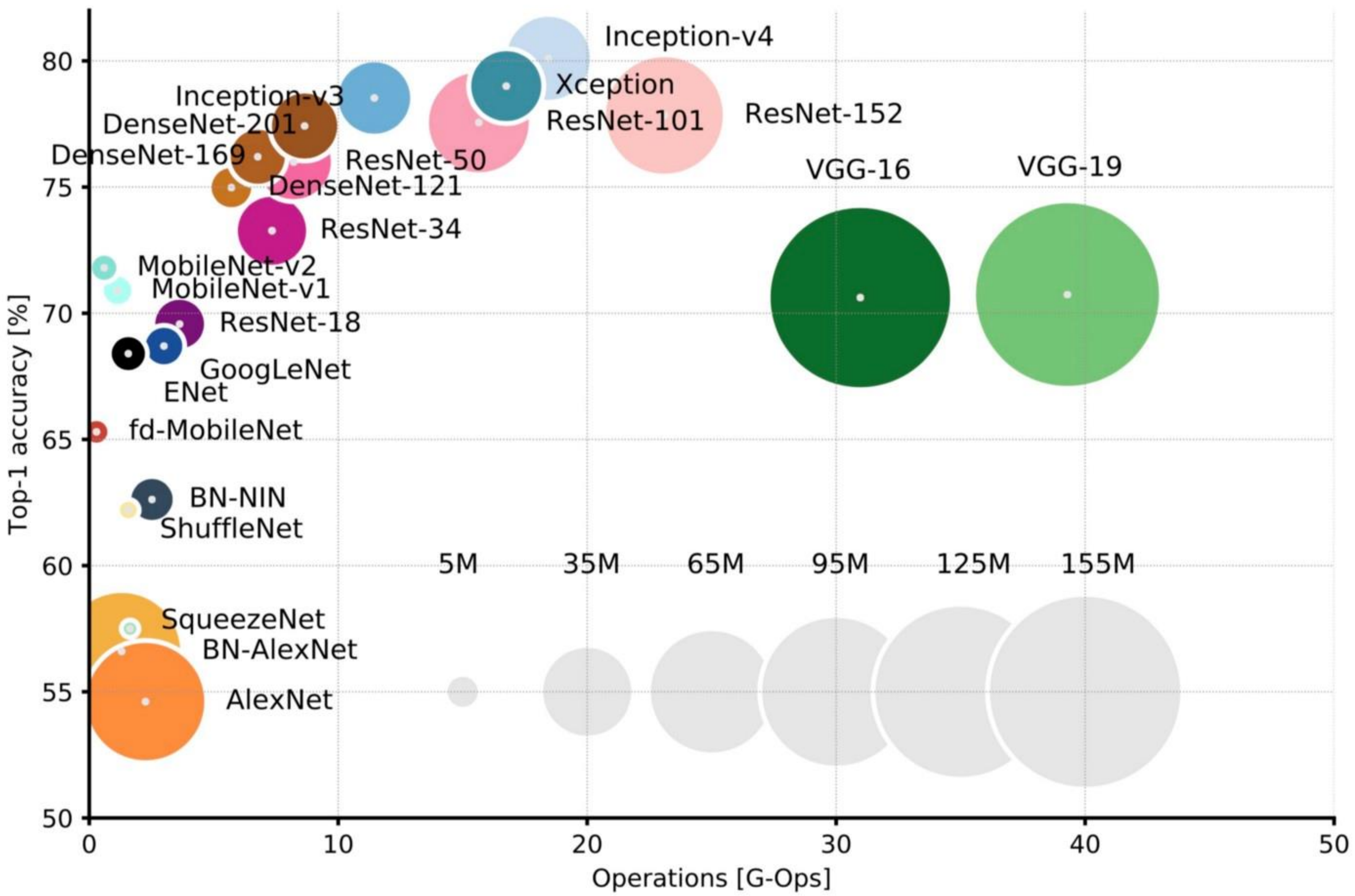
Evaluating Reduced Numerical Datatypes to Train Deep Neural Networks using PIN

John Osorio Ríos[*†§], Adrià Armejach[*†], Eric Petit[§], Greg Henry[§] and Marc Casas[*†]

* Barcelona Supercomputing Center (BSC)
† Universitat Politècnica de Catalunya (UPC)
§ Intel

This work has been partially supported by Intel under the BSC-Intel collaboration.

DNNs Overview



- The use of Deep Neural Networks is becoming ubiquitous.
- Medicine, sports, chemistry, physics are fields where DNNs are widely used nowadays.
- Models and datasets continue to become deeper and larger. Increasing computational needs.

A. Canziani, E. Culurciello, A. Paszke, « An Analysis of Deep Neural Networks Models for Practical Applications », in *The 2017 IEEE International Symposium on Circuits & Systems*, Baltimore, USA, May 2017.

Motivation

- Training Deep Neural Networks (DNNs) is a costly task in terms of computational resources like execution time or power.
- There are approaches able to reduce training costs without reducing DNNs accuracy. These approaches rely on reduced computer number formats.
- We propose:
  - A method to evaluate several reduced precision datatype approaches (FASE).
  - A technique to dynamically adapt the numerical precision during the training phase.
  - A set of compound datatypes relying on a specific datatype.

Outline

- A Fast, Accurate and Seamless Emulator for Custom Numerical Formats (FASE) (Link)
- Dynamically Adapting Floating-Point Precision to Accelerate Deep Neural Network Training (Link)
- A BF16 FMA is All You Need for DNN Training (Link)

A Fast, Accurate and Seamless Emulator for Custom Numerical Formats (FASE)

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

A Fast, Accurate and Seamless Emulator for Custom Numerical Formats

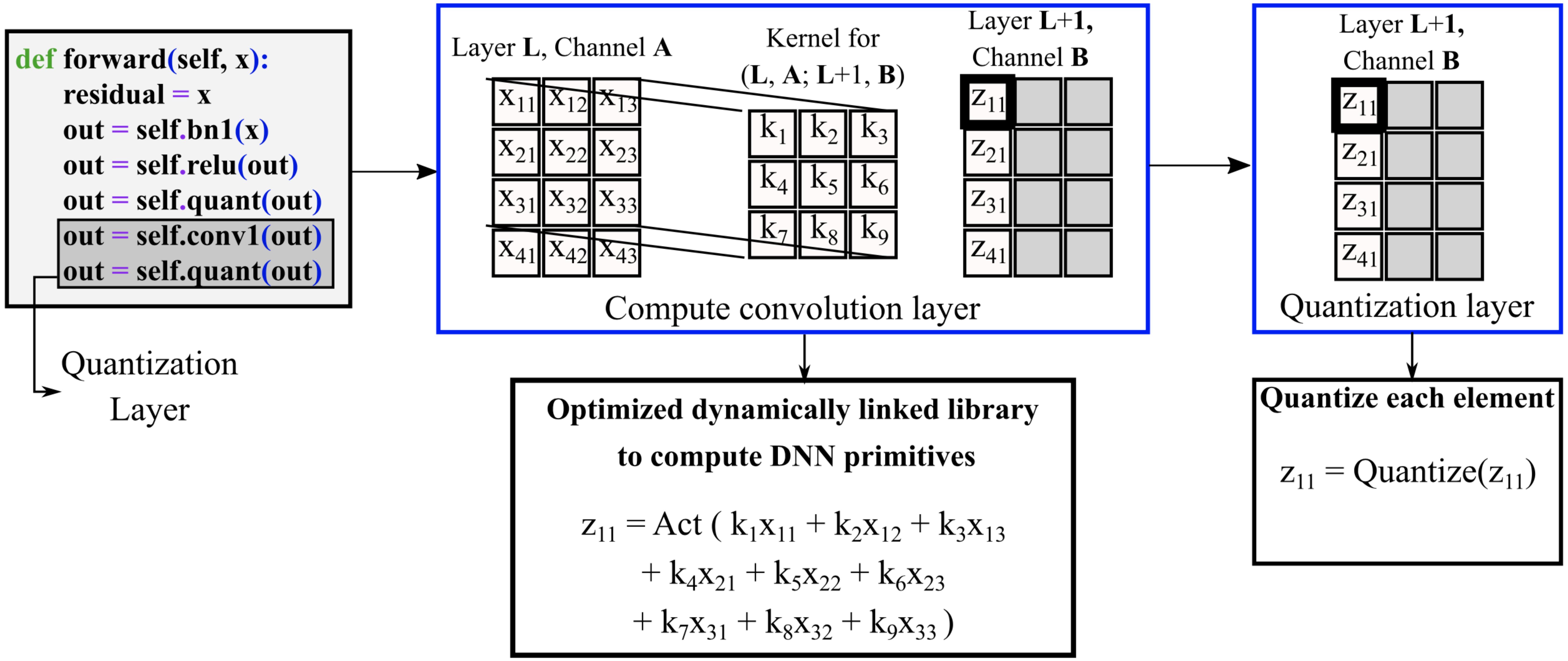
- FASE is a tool that enables the emulation of custom numerical formats on any application.
  - It enables HW architects to understand numerical behavior before committing to costly HW implementations
  - It is based on Intel PIN

Features	**Emulators**				
	RPE [7]	**QPyTorch [39]**	**TensorQuant [26]**	**Verificarlo [4]**	**FASE**
Fast	✗	✓✓	✓	✓✓	✓
Accurate	✓	✗	✓	✓	✓
Seamless	✗	✗	✗	✗(recompilation)	✓
Dynamic Libraries	✗	✗	✗	✗(Lib. recompilation)	✓
Independent	✗	✗	✓	✗(compiler dep.)	✓

A Fast, Accurate and Seamless Emulator for Custom Numerical Formats

- There are various state-of-the-art techniques to emulate reduced precision approaches.
  - Coarse-grain granularity (Function level)
  - Fine-grain granularity (Instruction level)

Steps for coarse-grain emulation on a convolutional layer.



Barcelona Supercomputing Center
Centro Nacional de Supercomputación

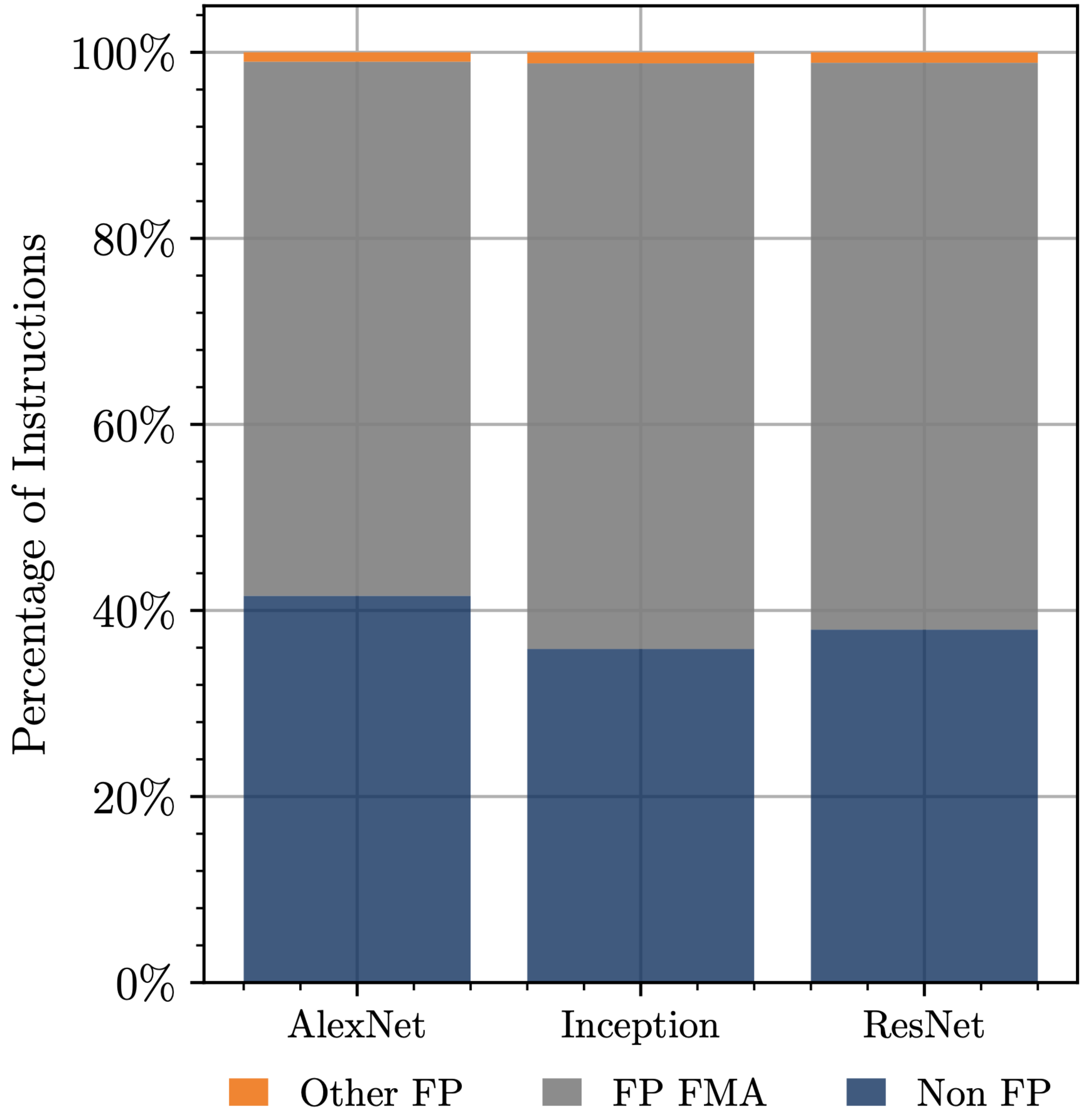
Design Principles

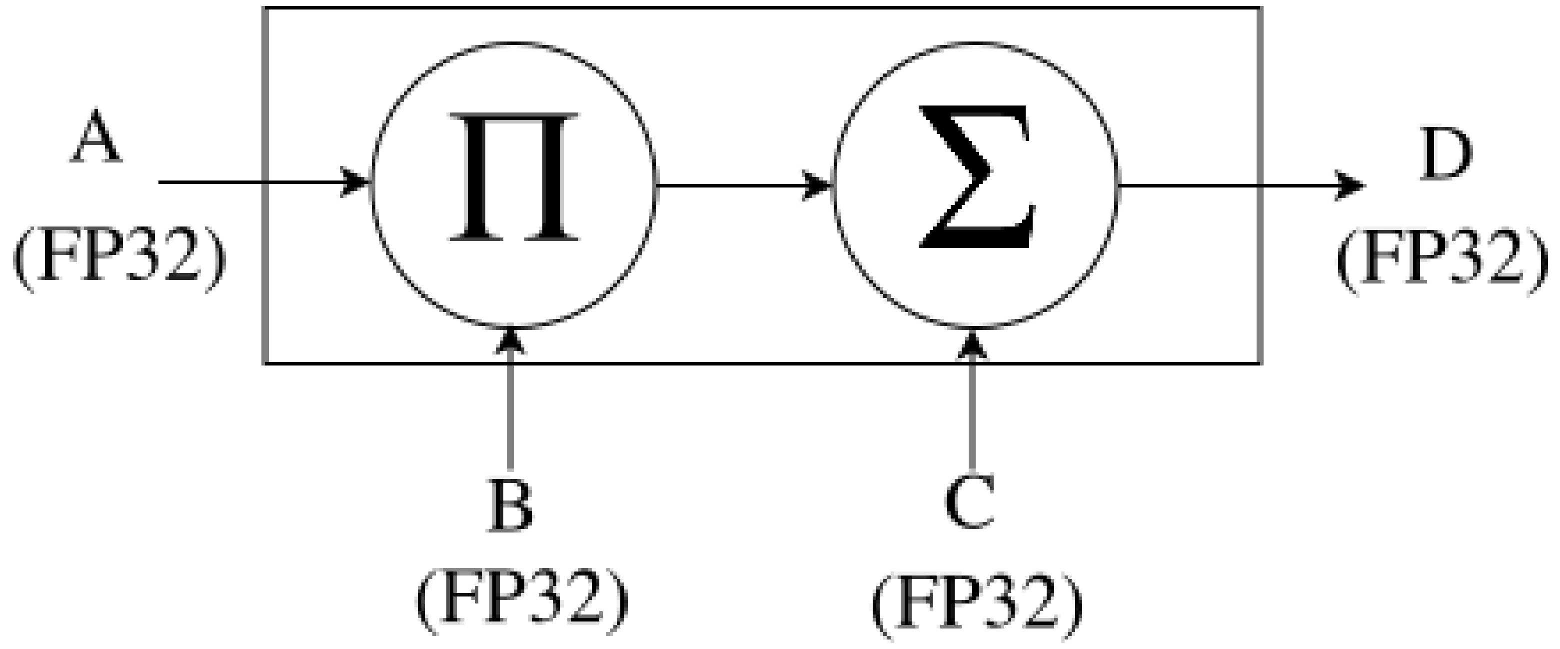
- The simplicity is the most important feature of FASE
  - It enables Fast, Accurate and Seamless emulation of custom numerical formats
  - It emulates code of external dynamically linked libraries

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

Workload Characterization

- We analyze DNN workloads
  - Around 98% of FP instructions are Fused-Multiply-Add (FMA)
  - We focus on instrumenting all FMA instructions in order to emulate the reduced precision approaches



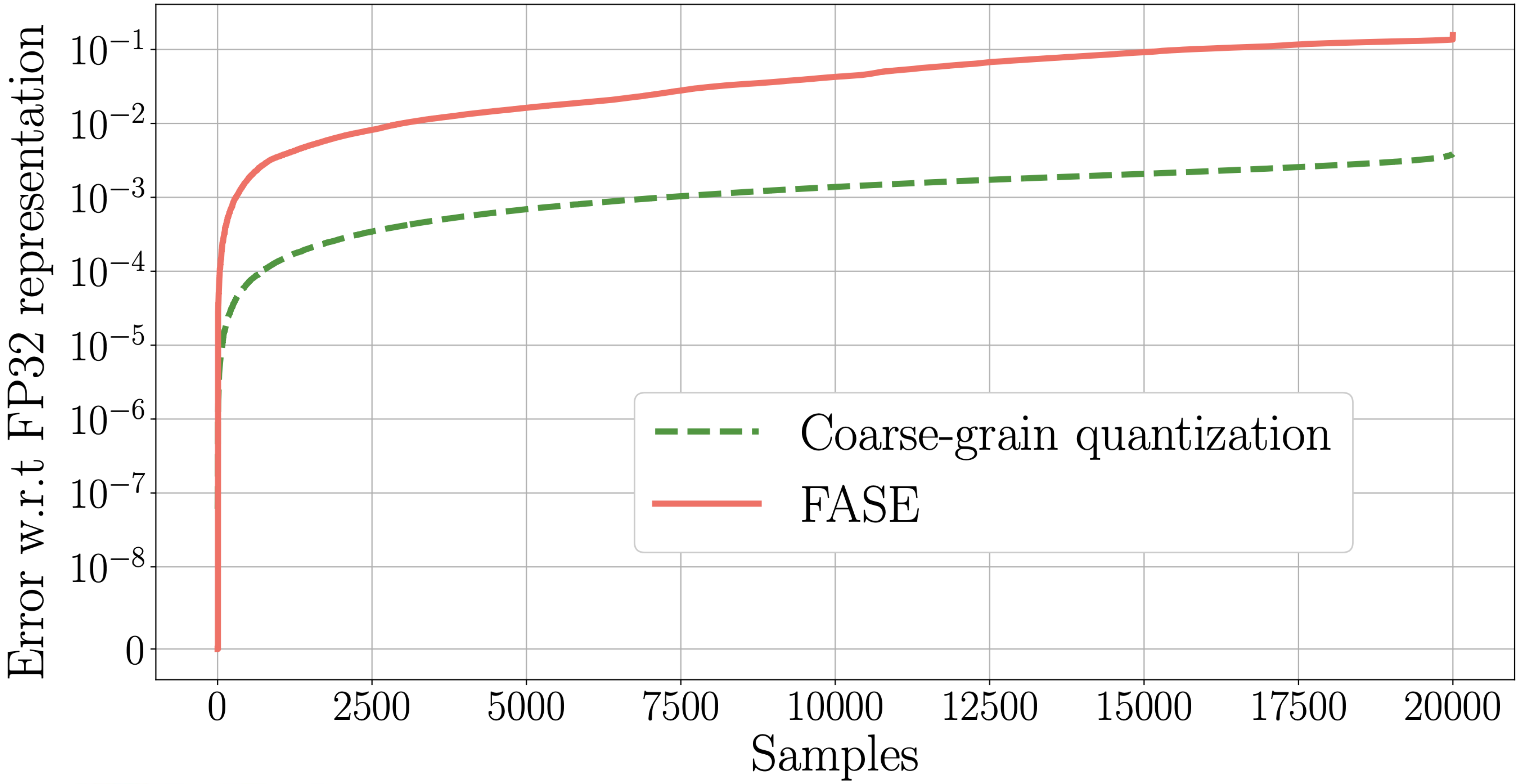


Barcelona Supercomputing Center
Centro Nacional de Supercomputación

Implementation

Emulation Accuracy

- **Methodology:** We use SGEMM to multiply two matrices using the Intel Math Kernel Library.
- **Results:** The figure compares the relative error when employing fine-grain and coarse-grain emulation on the Intel MKL SGEMM kernel.



- **Using FASE (fine-grain):**
  - It is close to what would be observed on real HW
  - Able to track errors that accumulate per instruction
- **Using coarse-grain:**
  - Results more accurate that they should
  - Cannot capture errors that accumulate per instruction

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

Emulation Overhead Measurement

- **Results:** The table shows the emulation latencies introduced by FASE when converting in a fine-grain manner the input and output operands to BF16 with RNE rounding.

Workload (framework)	FASE Instr.	Latency			
		Unopt	Opt1 Basic block	Opt2 Vectorization	Full Opt
SGEMM (MKL)	15×	1809×	880×	82×	39×
ResNet50 (Caffe)	11×	1131×	553×	76×	30×
3DGan (Tensorflow)	7×	714×	340×	66×	28×
LSTM (PyTorch)	18×	1096×	551×	70×	29×
Transformer (PyTorch)	8×	818×	423×	36×	17×

Large Scale Experiments

- **Methodology:** To show FASE supports real workloads we perform a set of large-scale experiments. These tests consider the use of several DNN models, datasets and numerical datatypes.
- **Results:** The table shows the results of using FASE for several full DNN training workloads.

Model	Dataset	Accuracy		
		FP32	BF16	MP
ResNet18	CIFAR100	71.91%	71.46%	71.89%
ResNet34	CIFAR100	73.21%	72.83%	73.86%
ResNet50	CIFAR100	74.78%	69.24%	74.25%
ResNet101	CIFAR100	75.93%	67.10%	75.65%
MobileNetV2	CIFAR100	75.04%	73.92%	75.16%
AlexNet	ImageNet	60.79%	57.80%	60.18%
Inception	ImageNet	74.01%	72.03%	73.73%
LSTMx2 (Perplexity)	PTB	86.86	137.69	87.09
Transformers (BLEU)	IWSLT16	34.53	34.86	34.66

Conclusions

- We propose FASE, an emulation tool for custom numerical formats. FASE is **accurate**, **fast**, and **seamless**.
- Our evaluation demonstrates that FASE is more accurate than other state-of-the-art proposals that employ coarse-grain emulation, uncovering relative errors that appear only in fine-grain emulation.
- We demonstrate that by applying both the basic block and vectorization optimizations, FASE latency overheads are manageable, ranging between 17× to 39× for a wide variety of workloads.

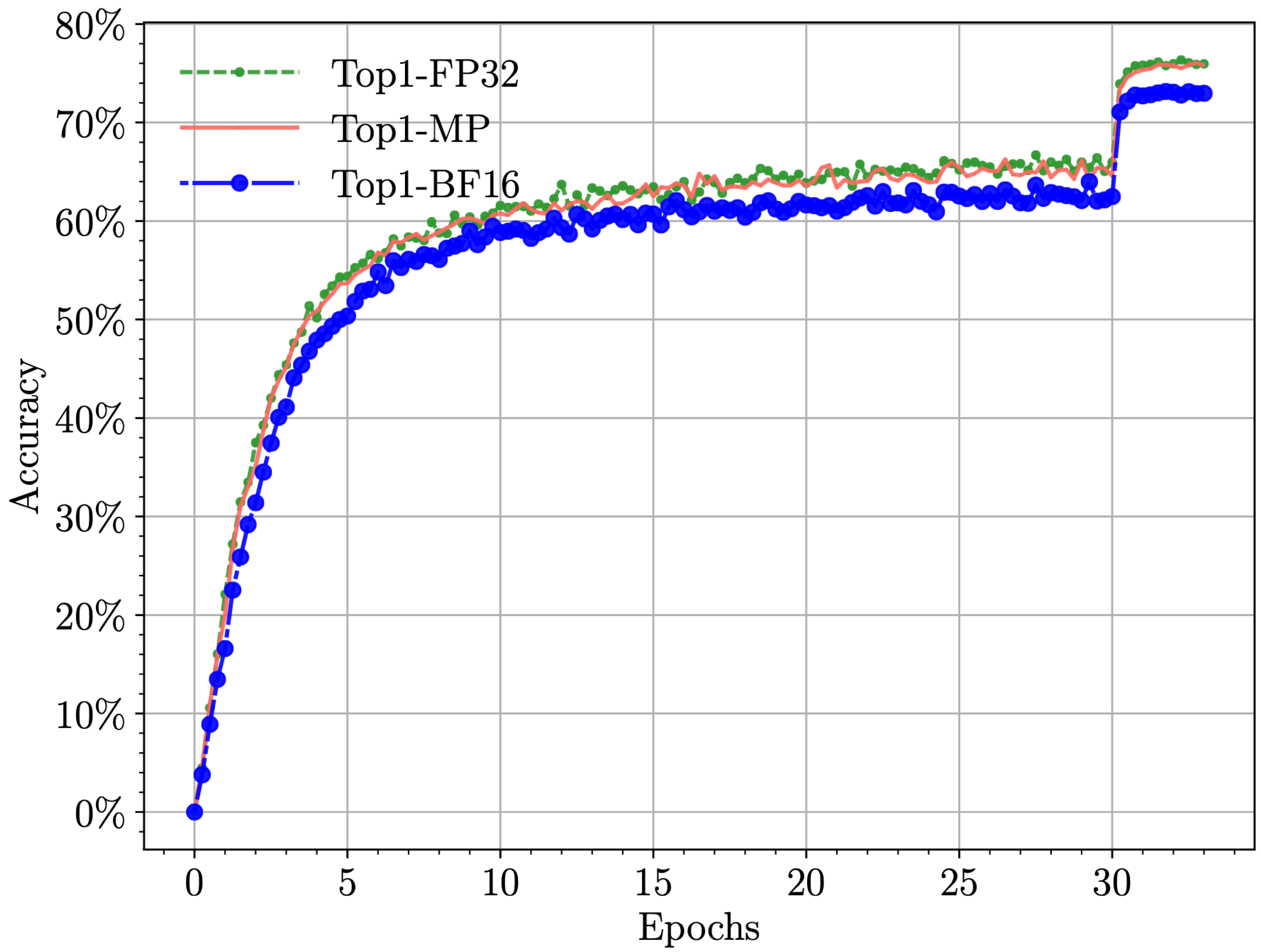
Dynamically Adapting Floating-Point Precision to Accelerate Deep Neural Network Training

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

State-of-the-Art FMAs for Training

Training	Inputs		Output	Multiply	Accum.
	A,B	C	D		
Tensor cores	FP16/BF16	FP32	FP32	FP16/BF16	FP32
Google TPU v3	BF16	FP32	FP32	BF16	FP32
AVX512-BF16	BF16	FP32	FP32	FP32	FP32
Full BF16	BF16	BF16	BF16	BF16	BF16

Analysis for Evaluated DNNs

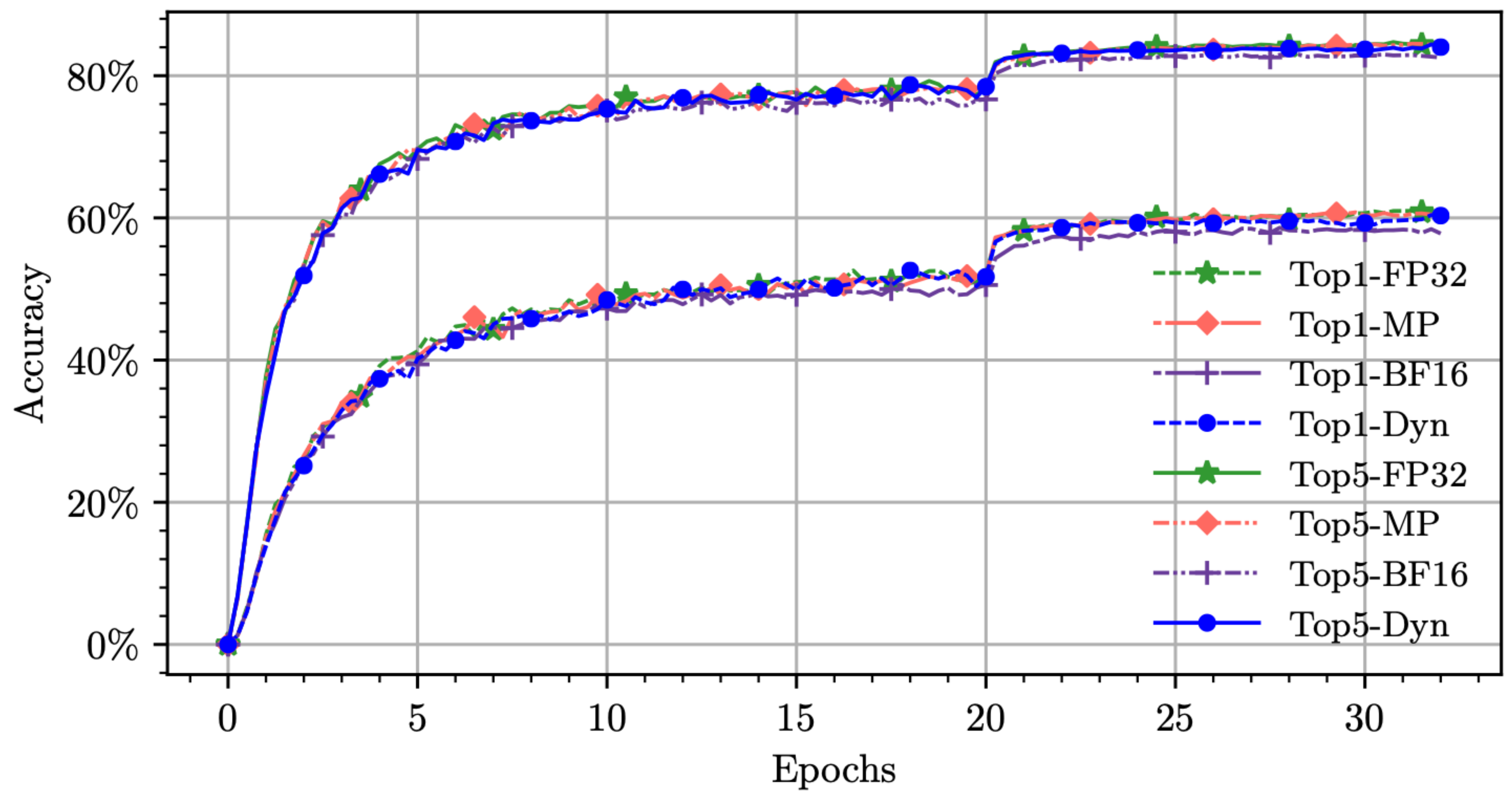


Static Techniques on ResNet-50

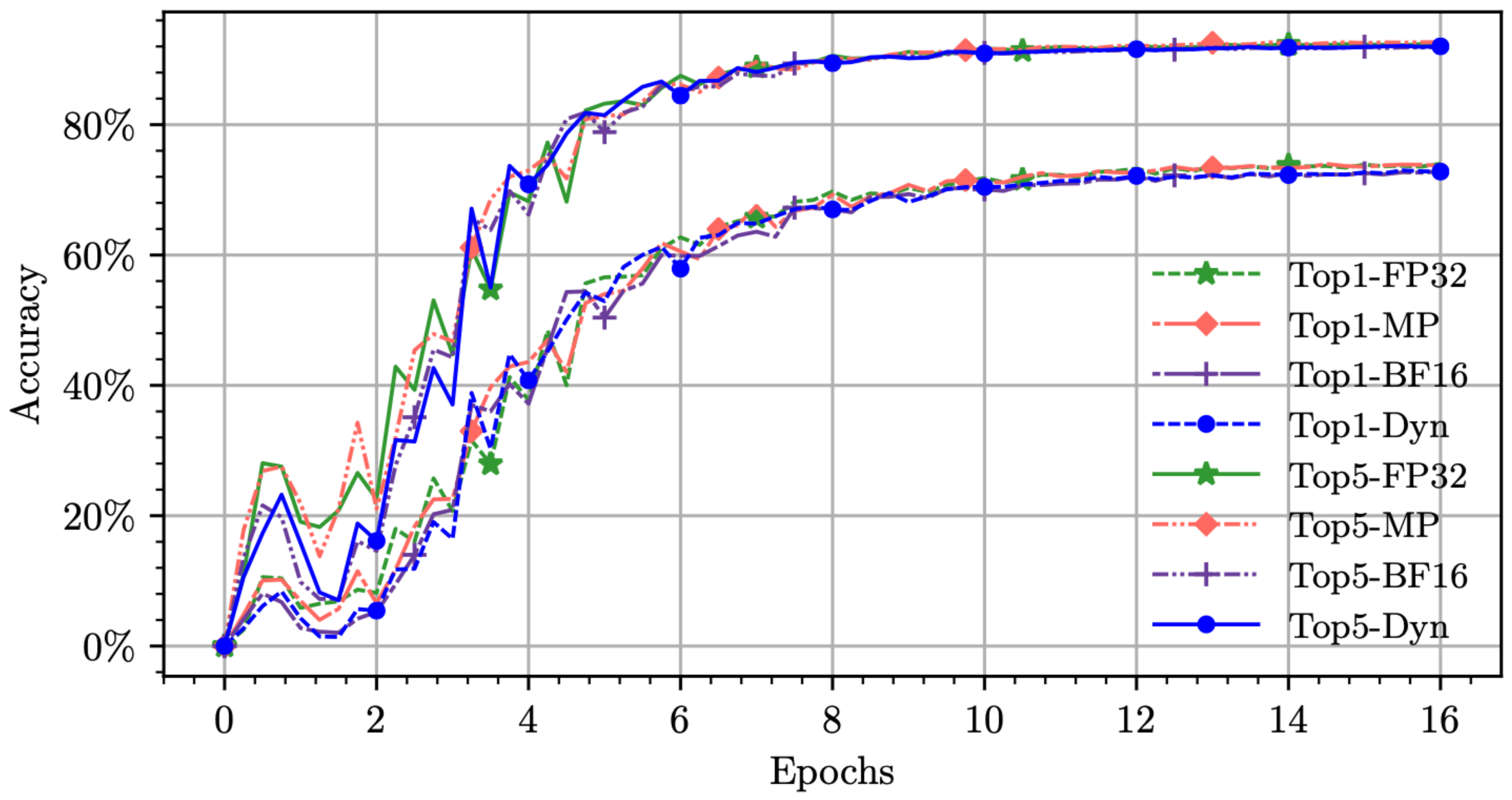
Dynamic Precision Training

```
numBatchesMP ← 10                                   // Number of consecutive MP batches
numBatchesBF16 ← 1000                               // Number of consecutive BF16 batches
emaThreshold ← 0.04                                 // Defines EMA reduction threshold

precisionModeBF16 ← False      // Indicates current precision mode, True means BF16
countBatchesBF16 ← 0           // Counts how many numBatchesBF16 have been executed
numBatchesTrain ← numBatchesMP // Number of batches per training loop iteration

for i = 0 to niter do
    train.step(numBatchesTrain)          // numBatchesTrain batches precisionModeBF16
    trainingLoss[i] ← train.trainingLoss
    if i = 5 then                        // Initial history to calculate EMA
        EMA ← average(trainingLoss)
    if i > 5 then
        EMAprev ← EMA
        EMA ← emaCalculation(trainingLoss, EMAprev)        // Each numBatchesMP
        if (precisionModeBF16! = True) then
            if ((EMAprev − EMA) > emaThreshold) then       // If training loss goes down
                precisionModeBF16 ← True
                changeToBF16()                             // Switch precision to BF16
        else
            countBatchesBF16 ← countBatchesBF16 + numBatchesTrain
            if (countBatchesBF16 = numBatchesBF16) then
                if ((EMAprev − EMA) > emaThreshold) then   // If training loss goes down
                    countBatchesBF16 ← 0                   // Stay in BF16 precision
                else                                       // If training loss stagnates
                    precisionModeBF16 ← False
                    changeToMP()                           // Switch precision to MP
                    countBatchesBF16 ← 0
```

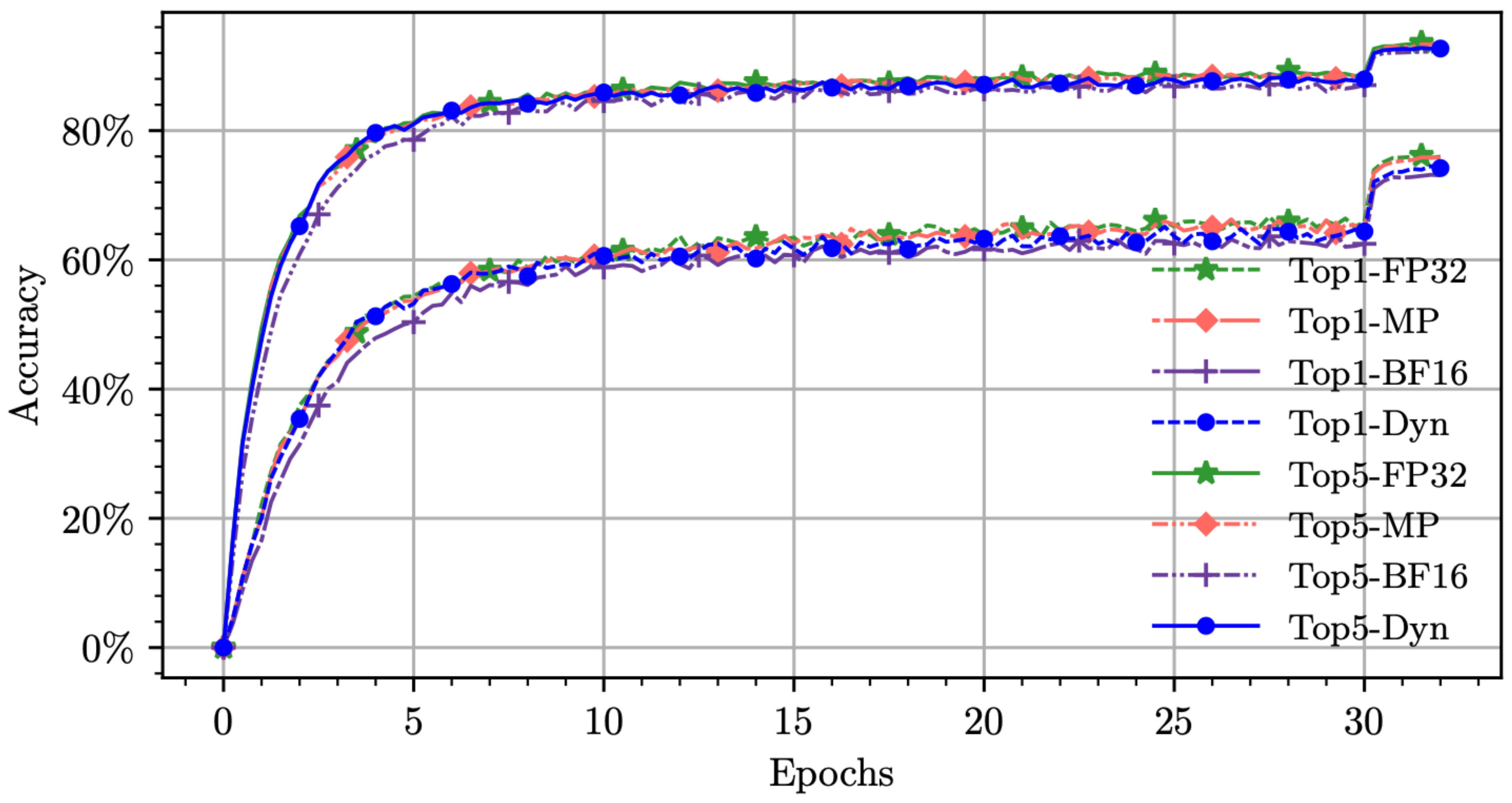
Object Classification DNNs



AlexNet



Inception



ResNet-50

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

Object Classification DNNs

Model	Epoch	FP32		MP		Dynamic			BF16	
		Top-1	Top-5	Top-1	Top-5	Top-1	Top-5	BF16FMA	Top-1	Top-5
AlexNet	32	60.79%	84.50%	60.18%	84.43%	60.32%	84.02%	94.60%	57.80%	82.56%
Inception	16	74.01%	92.36%	73.73%	92.67%	72.80%	92.02%	95.55%	72.03%	92.05%
ResNet-50	32	75.96%	93.37%	75.70%	93.20%	74.20%	92.70%	96.40%	72.97%	92.30%

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

intel

Conclusions

- Full BF16 FMA instructions fail to deliver comparable accuracy levels.
- We proposed a *Dynamic* training technique that performs up-to 94.6% of FMAs using full BF16 ones.
- We used Caffe and PyTorch to show the versatility of FASE to work seamlessly on different DNN frameworks

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

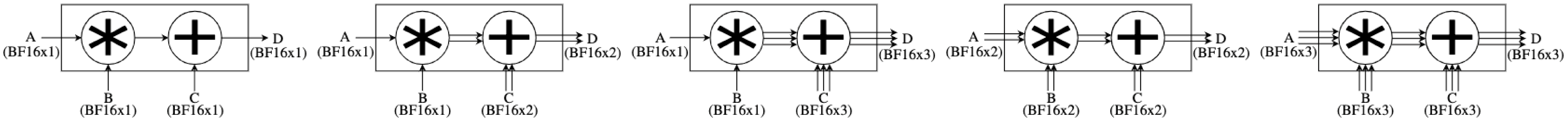
intel

A BF16 FMA is All You Need for DNN Training

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

Introduction

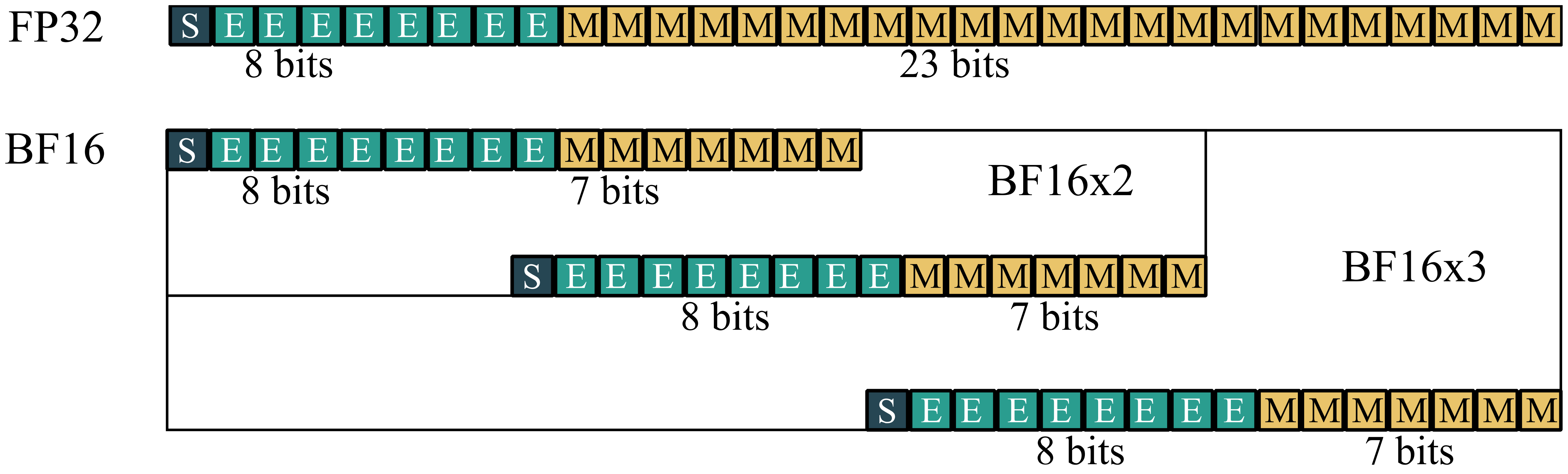
- First approach to train state-of-the-art DNNs entirely using the BF16 format
- We propose a new class of FMA operators, $\mathbf{FMA^{BF16}_{N\_M}}$
- They represent operands A and B using N BF16 literals (**BF16xN**)
- Input C and output D use M BF16 literals (**BF16M**)



(a) $\mathrm{FMA}^{\mathrm{BF16}}_{1\_1}$. (b) $\mathrm{FMA}^{\mathrm{BF16}}_{1\_2}$. (c) $\mathrm{FMA}^{\mathrm{BF16}}_{1\_3}$. (d) $\mathrm{FMA}^{\mathrm{BF16}}_{2\_2}$. (e) $\mathrm{FMA}^{\mathrm{BF16}}_{3\_3}$.

The BF16xN Data Representation

- The BF16xN data representation format is a compound datatype composed of N BF16 literals. The BF16x1 format uses 1-bit and 8-bits storage for sign and exponent, like FP32, and 7 explicit mantissa bits.



$$a_0 = BF(a)$$

$$a_1 = BF(a - a_0)$$

$$a_2 = BF(a - (a_0 - a_1))$$

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

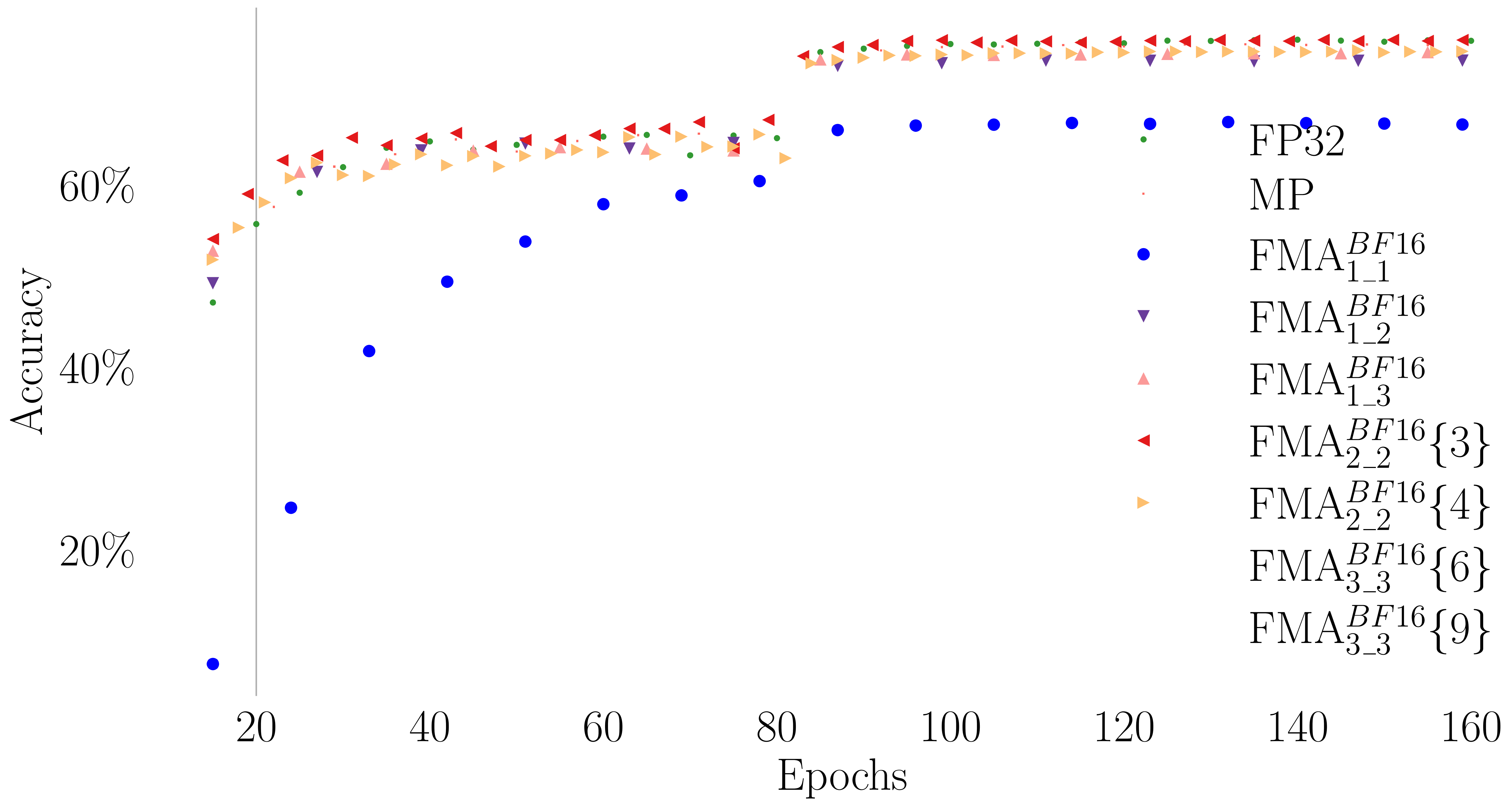
Characterization of $\mathbf{FMA^{BF16}_{N\_M}}$ Units

- To characterize our $\mathbf{FMA^{BF16}_{N\_M}}$ units we use the observation that the area of an FMA is dominated by the multiplier as it grows quadratically with mantissa size. An FP32 FMA requires $24^2 = 576$ area units, while an FMA with BF16 multiplier inputs would require just $8^2 = 64$ units.

$\mathbf{FMA^{BF16}_{N\_M}}$	$\mathbf{FMA^{BF16}_{1\_1}}$	$\mathbf{FMA^{BF16}_{1\_2}}$	$\mathbf{FMA^{BF16}_{1\_3}}$	$\mathbf{FMA^{BF16}_{2\_2}\{3\}}$	$\mathbf{FMA^{BF16}_{2\_2}\{4\}}$	$\mathbf{FMA^{BF16}_{3\_3}\{6\}}$	$\mathbf{FMA^{BF16}_{3\_3}\{9\}}$	**FP32**
Multiplier *mantissa* bits	8	8	8	[15, 16*]	16	[23, 24**]	24	24
Maximum input bitwidth	16	32	48	32	32	48	48	32
# BF16 multiplications	1	1	1	3	4	6	9	N/A
# Area Units	64	64	64	192	256	384	576	576
Speed-up wrt FP32 (equivalent area)	9.0×	9.0×	9.0×	3.0×	2.3×	1.5×	1.0×	1.0×

Evaluation

- The figure shows the results obtained when training ResNet101 using CIFAR100 dataset
  - $\mathbf{FMA_{2\_2}^{BF16}}$\{3\} outperforms the other operators while keep using BF16 during the whole training time



Conclusions

- We propose a new class of FMA operators, $\mathbf{FMA}^{\mathbf{BF16}}_{\mathbf{N\_M}}$, that entirely relies on BF16 FMA hardware instructions but delivers FP32 training accuracy.
- In contrast with previous implementations, we do not employ FP32 routines
- All FMA instructions use BF16 arithmetic for the whole training process
- We evaluate the operators on seven different DNN workloads
  - ResNet18, ResNet34, ResNet50, ResNet101 and MobileNetV2 on CIFAR10/100
  - LSTMx2 on PTB dataset
  - A transformer-based model on the IWSLT16 dataset

Future Work

- Support AMX extensions on FASE
- Evaluate other reduced precision datatypes
  - FP8, INT8, INT4
  - Dynamic compound datatypes
  - Evaluation of possible new numerical datatypes

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

intel

THANKS

John Osorio Ríos (john.osorio.rios@intel.com)
Adria Armejach (adria.armejach@bsc.es)